Reinforcement Learning

Grundlagen Grundbegriffe Grundübungen

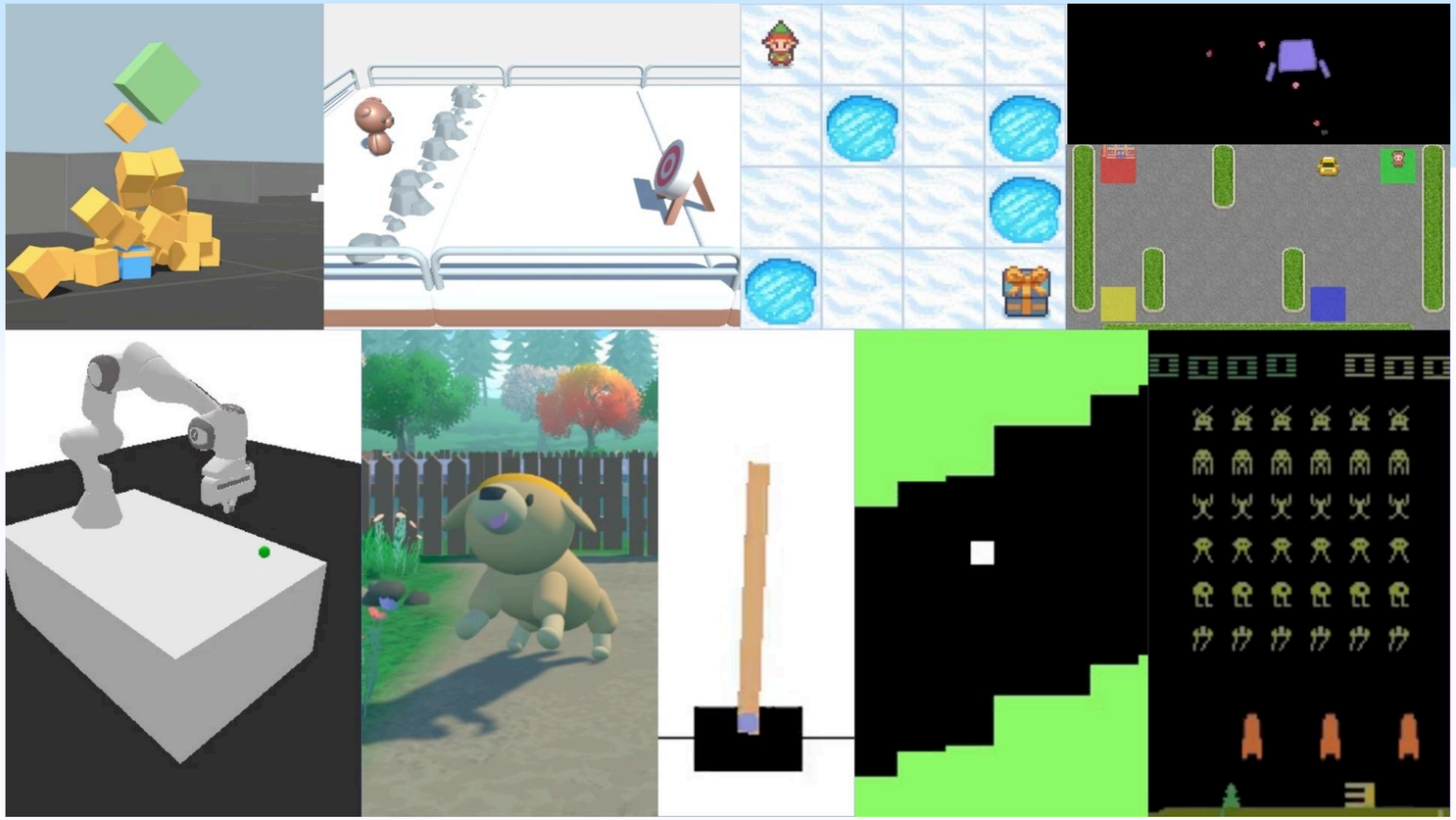
Alfatraining Kurs 2026-01 Dozent: Karsten Flügge

Themen des Tages

Tag 10

Multi-Agent RL

S.259-299



Multi-Agent RL

Multi-**A**gent **R**einforcement **L**earning (**MARL**)

Szenarien mit mehreren **Agent**en, die in einer gemeinsamen Umgebung agieren.

Diese Agenten können **kooperieren, konkurrieren** oder eine **Mischung** aus beidem zeigen.

Jeder Agent optimiert seine **eigene Policy** unter Berücksichtigung anderer Agenten.

Die **Umgebung** ist **nicht mehr stationär**, da sich Policies anderer Agenten kontinuierlich ändern.

MARL ist besonders relevant in komplexen, dynamischen Systemen wie autonomen **Fahrzeugflotten**, Mehrspieler-**Spielen** oder verteilten **Robotik**systemen.

Es geht also nicht nur um das mathematisch/algorithmische Einfügen von mehreren Agenten sondern teilweise tatsächlich um **diskrete physikalische** Agenten.

Wie kann eine Policy Rückwirkung auf eine Umgebung haben?

Multi-Agent RL

- **Vorteile:**
- Modellierung komplexer, realitätsnaher Systeme.
- Ermöglicht **emergentes Verhalten** (z.B. Schwarmintelligenz). "Quantität erzeugt Qualität"

- **Herausforderungen:**
- **Instabilität** durch sich verändernde Umgebung.
- **Skalierbarkeit**
- **Koordination**
- Teilweise Beobachtbarkeit und **Kommunik**ationsrestriktionen.

- **Beispiele & Anwendungen**
- Kooperative Robotik (z.B. Drohnenschwärme)
- Strategische Spiele (z.B. StarCraft, Poker, …) Puzzles
- Verkehrssteuerung (z.B. Ampeloptimierung mit mehreren Agenten?)
- Verteilte *Energiemanagement*systeme

Multi-Agent RL

Bottom Up vs Top Down?

Mixture of Experts MoE (Börse / LLM)

verschiedene Agenten geben verschiedene "*Meinungen*" ab

am Ende wird eine gemeinsame Entscheidung getroffen,

Anhand von verschiedenen Kriterien.

https://github.com/AI4Finance-Foundation/FinRL-Trading/blob/master/rl_model.py

Beispiele & Anwendungen

- Kooperative **Robotik** (z.B. **Drohnenschwärme**)
    - **Industrie**/Reinigungsroboter (Kollisionsvermeidung)
    - **Robocup** Fußball WM (kleine *Hunde*)
- Strategische **Spiele** (z.B. **StarCraft**, Poker)
- Industrieroboter (Assemblieren Gabelstabler Transporter)
- **Verkehrssteuerung** (z.B. **Ampel**optimierung mit mehreren Agenten?)

  Lieferfahrzeuge Flotten Fahrer

  Flugtaxis Lilium, Luftraumsteuerung
- Verteilte *Energiemanagement*systeme
- **Leben** (Mensch und Tier und Pflanze und Maschine) **Soziale Insekten! Gruppenverhalten** (Ameisen, Bienen in Panik…)

  Kolonie als Meta-Organism / 'Agent'

  Pilze / Flechten / Korallen (Territorialkämpfe)
- **Börsen** (Trading bots stocks) Mixture of Experts
- CyberSecurity (Agents = Computers, Penetration Testing ≈ Friendly Hackers + External Evil Agents)

  Anomaly / Thread-Detection-**Net** Mustererkennung Hardware-cheats Autofire Forced Trainer RL

Beispiele & Anwendungen MARL

Synchron Turnen/Schwimmen/Fliegen/Tanzen (Trillerpfeife)
Dronen Flugshow (statt Böller!)
Selbstfahrende Autos Waymo Einparken (konkurierend statt kooperierend;)
Supervised / Fest einprogrammiert / RL
Medizinkapseln
Kapselendoskope(-) Gewebeproben entnehmen oder minimalinvasive Eingriffe
Sport
Boxroboter Wettbewerb Fußball Basketball …
Perplexity? LLM Agenten MoE. MARL finetuning 'denkbar'
Baustelle ? Teuer MARL zu trainieren? Vorinvestition.
Humanoide? Wann? 2030 2040? Problem: Taktilität, Kraft, Preis
Erntehelfer: Sensibel genug? (Zukunft)
künstliche Gliedmaßen optimieren (nicht MARL aber DL)
Reinigungsflotte wer war schon wo? (minimal MARL: wo waren wir schon)
Industrie Lager
E-Gabelstabler
AMRs = **Autonomous Mobile Robots**

Aspekte von Multi-Agent Algorithmen

kooperativ / konkurrierend

kommunikativ / eigenständig

eigene / shared Policy?

eigene Hyperparameter?

eigenes Modell / Kopie von Umgebung?

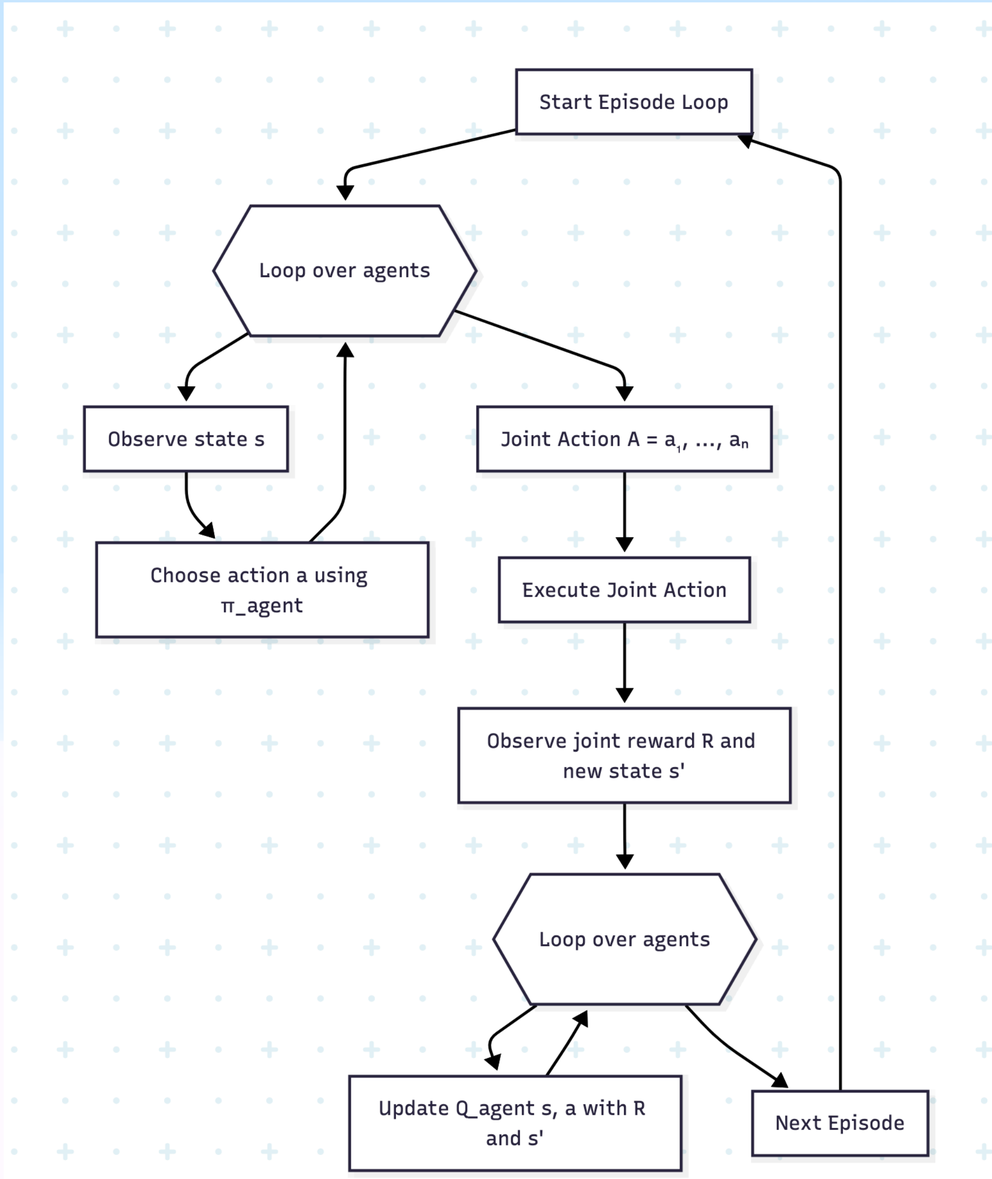
(de)zentralisiert / autonom (top/bottom up)

(a)synchron

Kooperation

Pseudocode für **kooperatives Ensemble:**

```
for episode in episodes:
    for agent in agents:
        observe state s
        choose action a using independent policy π-agents
    execute joint action A = (a1, ..., an)
    observe joint reward R and new state s'
    for agent in agents:
        update Q_agent(s, a) with R and s'
```



Advantage

a) TD Advantage "stepwise PG"

A(s,a) = **Q**(s,a) – V(s)

b) Monte Carlo Advantage "episodisches PG"

$A_t = \mathbf{G_t}$ - $V(s_t)$ "baseline"

$G_t = \sum_i \gamma^{t+i} r_i$

Generalized **A**dvantage **E**stimator (GAE)

$$\widehat{A}t^{(\lambda)} = \sum (\gamma\lambda)^l \delta_{t+l}$$

- Für $\lambda = 1$: volle Monte Carlo Advantage
- Für $\lambda = 0$: reiner TD-Fehler

PS: $\mathbf{E}_{a\sim\pi(s)}[\mathbf{A}_\pi(s,a)] = 0$

GAE

Generalized Advantage Estimate

$$A_t := \sum (\gamma\lambda)^l \delta_{t+l}$$

δ_t ist unser delta von TD

$\delta_t = \underline{V_t}(S_t) - V_t(S_t)$ "Temporal Difference of Target V to real V"
$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)$ "TD error of estimate"

$\lambda = 0 \rightarrow$ 1-step TD error $A_t = \delta_t$

$\lambda = 1 \rightarrow$ **full GAE** (infinite sum, Monte Carlo-like)

Asynchronous Kooperation

A3C **A**synchronous **A**dvantage **A**ctor **C**ritic

A3C Pseudocode für **kooperatives hive-mind Ensemble:**

```
for episode in episodes:
    get joint policy
    for agent in agents:
        observe state s
        choose action a using own policy π-agents
    go execute joint action A = (a1, ..., an)
    observe joint reward R and new state s'
    for agent in agents:
        update Q_agent(s, a) with R and s'
```

Advantage A2C

A2C – Advantage Actor-Critic

Initialize policy π(a | s), value function V(s)

for iteration = 1, 2, ... do:

Collect N trajectories of length T using π_θ

for each **trajectory** τ = (s_0, a_0, r_0, ..., s_T):

for t = 0 to T–1:

A_t = **Q**(s_t, a_t) - **V**(s_t) , $Q(s_t, a_t) \approx r_t + \gamma * V(s_{t+1})$

A_t = $r_t + \gamma * V(s_{t+1}) - V(s_t)$ # Advantage estimate ≈ delta!

Loss_π = – E_t [**log π(a_t | s_t) * A_t**] # policy loss E_t => **.mean() !**

Loss_V = E_t [**(V(s_t) – (r_t + γ * V(s_{t+1}))²**] # critic/value MSE loss

H = E_t [–log π_θ(a_t | s_t)] # Noise Compute entropy bonus (optional)

Update θ using **∇θ** (Loss_π – β * H)

Update ϕ using **∇ϕ** Loss_V

A3C – Asynchronous Advantage Actor-Critic. (Always Absolute Apsilon A8C)

A3C

Anpassen von **A2C** – **Asynchronous Actor-Critic => A3C**

Initialize global policy & value (parameters θ, ϕ)

for each **worker** thread do: # agents

Initialize **local** $\theta' \leftarrow \theta$, $\phi' \leftarrow \phi$

repeat:

Collect **trajectory** $\tau = (s_0, a_0, r_0, ..., s\_T)$ using π_θ'

for t = 0 to T–1:

$A_t = r_t + \gamma * V\phi'(s_{t_{+1}}) - V\phi'(s_t)$

$Loss\_\pi' = - E_t [\log \pi\_\theta'(a_t \mid s_t) * A_t]$

$Loss\_V' = E_t [(V\_\phi'(s_t) - (r_t + \gamma * V\_\phi'(s_{t_{+1}})))^2]$

$H' = E_t [-\log \pi\_\theta'(a_t \mid s_t)]$

$g\_\theta' \leftarrow \nabla\theta' (Loss\_\pi' - \beta * H')$

$g\_\phi' \leftarrow \nabla\phi' Loss\_V'$

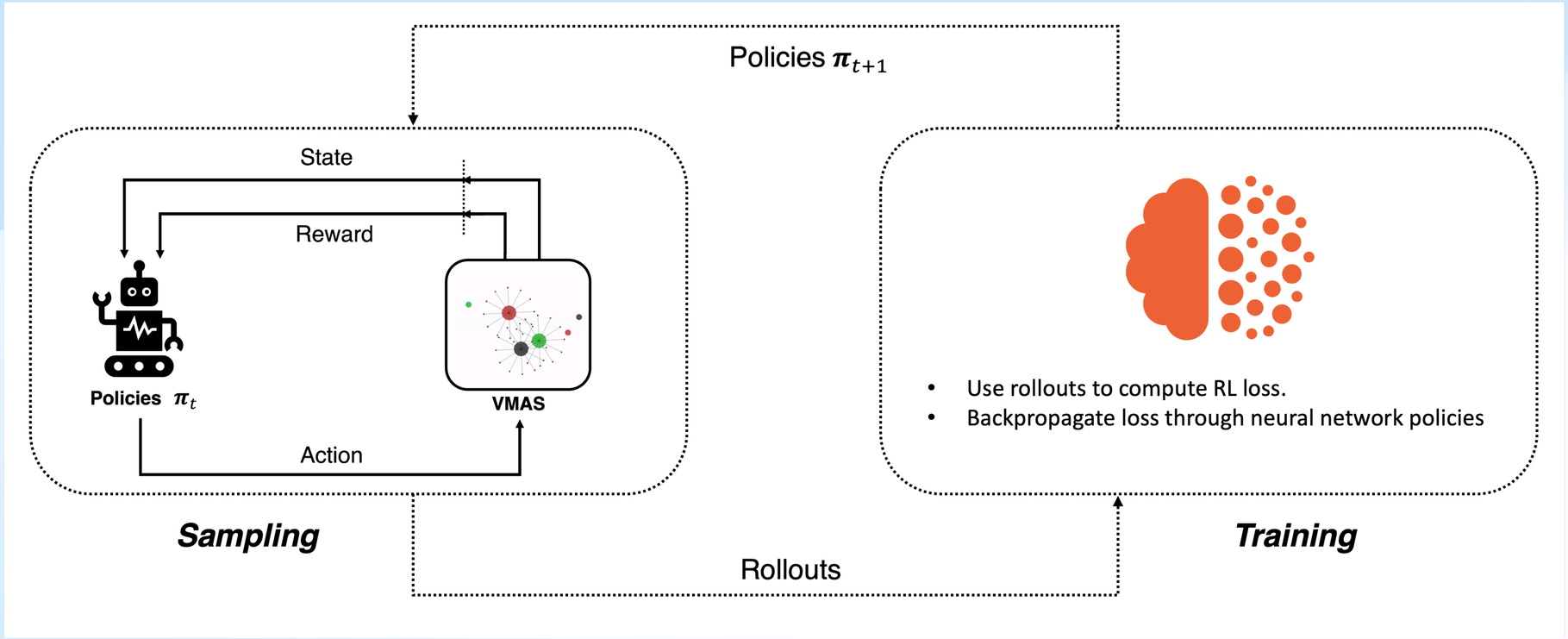
Asynchronously apply g_θ', g_ϕ' to global θ, ϕ

Sync local $\theta' \leftarrow \theta$, $\phi' \leftarrow \phi$

Aufgaben

Multi Agent **Library** / Example

Paper: Mastering diverse control tasks through world models



Libraries

https://github.com/Replicable-MARL/MARLlib **2023 :(**

https://docs.pytorch.org/rl/stable/tutorials/multiagent_ppo.html **TorchRL**

https://docs.ray.io/en/latest/ **Ray /** RL**Lib**

https://tianshou.org/en/stable/

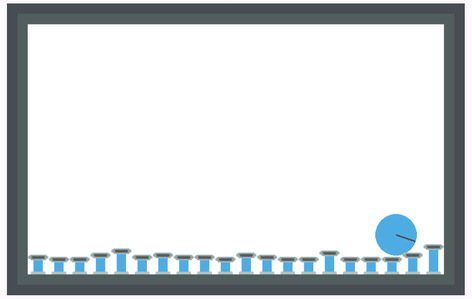
https://github.com/Farama-Foundation/**PettingZoo**

```
from pettingzoo.butterfly import pistonball_v6

env = pistonball_v6.parallel_env(render_mode="human")
observations, infos = env.reset()

while env.agents:
    # this is where you would insert your policy
    actions = {agent: env.action_space(agent).sample() for agent in env.agents}

    observations, rewards, terminations, truncations, infos = env.step(actions)
env.close()
```



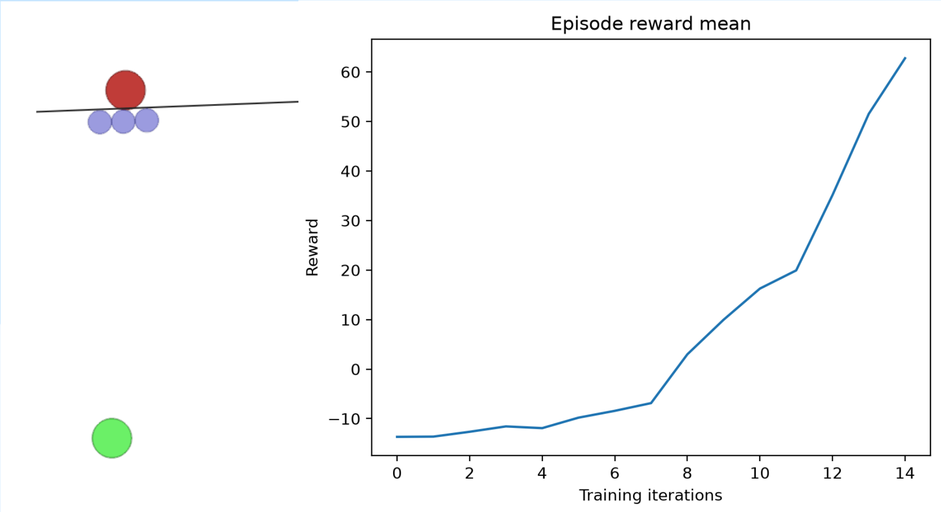
TorchRL 'gym'

https://docs.pytorch.org/rl/stable/tutorials/multiagent_ppo.html

TorchRL 'gym'

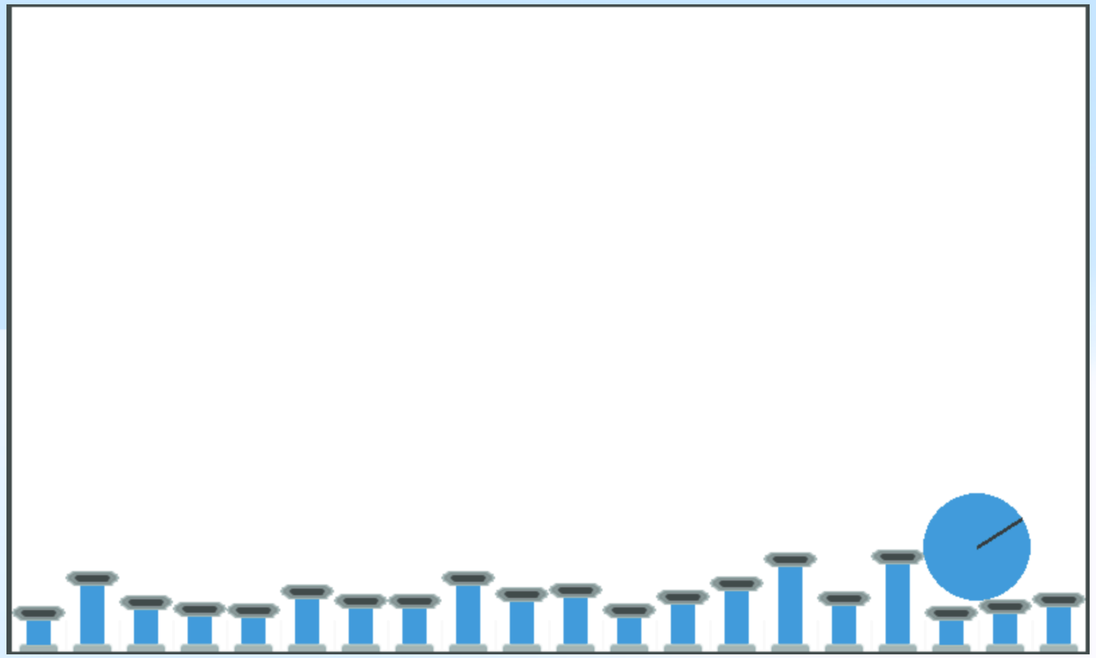
https://docs.pytorch.org/rl/stable/tutorials/multiagent_ppo.html

Dadurch dass die drei Agenten **live trainieren** und sich **gegenseitig austauschen** wird der Trainingserfolg schon sehr schnell nach 14 Episoden erreicht. (fast, übers Ziel hinaus;)



pettingzoo

- 3 Tasks: pistonball, knights, pong, tag (Ticken)
- pistonball: Ziel ist es durch geschickte Koordinierung der Kolben den
- Ball von rechts bis zur linken Wand bewegen zu lassen.



Aufgabe

Welche Aspekte von Multi-Agent Algorithmen sind in *multiagent_ppo.py* umgesetzt?

a / b

kooperativ / konkurrierend ? a (*emergent!*) / b (ignorierend bei give way)

kommunikativ / eigenständig? Kommunikation über collector? wissen wir nicht. eher nicht / indirekt b

eigene / shared Policy? **shared** (wenn eingeschaltet!)

eigene Hyperparameter? **nein**

je eigenes Modell / Kopie von Umgebung? **nein**

autonom / zentralisiert? autonom (weil centralised=False, controller nur zum Organisieren)

(**a**)synchron? **synchrone Schleife**

MARL Fazit

- Umfangreiches Gebiet mit vielen Ansätzen und Szenarien
- MARL ist weniger ein Algorithmusfeld als ein **Problem-Setting**
- Gib keinen Standardalgorithmus, eher ad hoc
- eher ENGINEERING problem, als theoretisches
- nutze alte Algorithmen / baseline in Schleife über alle Agenten
- Bibliotheken sind kurzlebig (zu viel Wandel)
- Spannend: **Self-play** gegen eigene Policy spielen (z.b. AlphaZero / Chess / tic-tac-toe)
- Ist etwas aus der Mode gekommen, kann aber wieder akut werden!

Familien: independent learning, centralized training/decentralized execution, value decomposition, population/self-play, opponent modelling, communication learning.

MARL Fazit

Nash Equilibrium (kompetitiv):

Kein Agent kann sich allein verbessern, wenn die anderen ihre Policy festhalten.

PPO

Proximal Policy Optimization

Vielleicht der wichtigste **RL Regularisierung**sansatz der Dekade.

Stelle sicher dass die updated **policy** *nahe* (***proximal***) zur alten policy ist, also ohne katastrophale Sprünge: Kontinuität beim lernen.

Bei RL ist der "Pfad zum Erfolg" oft ein **schmaler Grat**.
Zu kleine Schritte führen zum **fast Stillstand**,
Zu große Schritte zu **katastrophalen Fehlschritten**.

PPO

Proximal Policy Optimization

proxima ist lateinisch für "nahe"
proximal kennen wir von …
approximiert "angenähert"
Proxima Centauri 3-Sterne **4.25 light-years 'nahe'** 3 body problem!
NICHT Der proxy < procuratio „Besorgung, Verwaltung“

vibe proofing. coding
Terence Tao best mathematician

PPO

Proximal Policy Optimization

Wie stellen wir sicher, dass die updated **policy** nahe (***proximal***) zur alten policy ist?

π≈π' <> π-π' ≈ 0 oder besser …

PPO

Proximal Policy Optimization

Wie stellen wir sicher, dass die updated **policy** nahe (***proximal***) zur alten policy ist?

π≈π' <> **π/π' ≈ 1**

this quotient **ratio** is often called $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$

Zusatzbedingung kann man als Loss mitgeben.

PPO

Proximal Policy Optimization

Wie stellen wir sicher, dass die updated **policy** nahe (***proximal***) zur alten policy ist?

π≈π' <> **π/π' ≈ 1**

this quotient **ratio** is often called $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$

Sota : **GRPO** group relative policy optimization, **DeepSeek** LLM

PPO

Proximal Policy Optimization

Wie stellen wir sicher, dass die updated **policy** nahe (***proximal***) zur alten policy ist?

π≈π' <> **π/π' ≈ 1**

this quotient **ratio** is often called $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$

Naiv: **loss += MSE(π/π' , 1) = (π/π' - 1)^2**

PPO

Proximal Policy Optimization

updated **policy** nahe (***proximal***) zur alten policy ist?

Wir stellen sicher dass die **ratio nahe bei 1** ist:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Via **clip**ping, oder andere **loss**es

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta))\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$$

Trusted Regions: erlaube nur r im Bereich **[1-ε, 1+ε]**

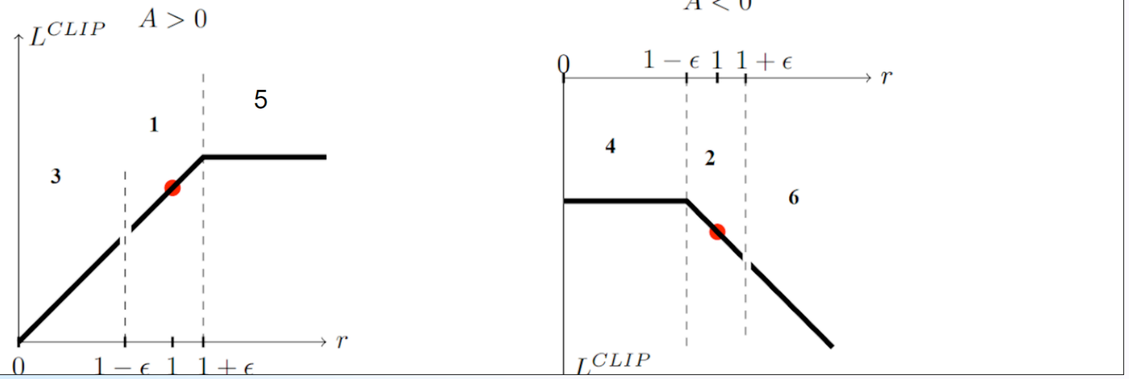
PPO

normale Formel L=-∑log(π)*A wird ge-clip'ed und mit r ersetzt:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

https://huggingface.co/blog/deep-rl-ppo#visualize-the-clipped-surrogate-objective

	$p_t(\theta) > 0$	A_t	Return Value of *min*	Objective is Clipped	Sign of Objective	Gradient
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	+	$p_t(\theta)A_t$	no	+	✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	−	$p_t(\theta)A_t$	no	−	✓
3	$p_t(\theta) < 1-\epsilon$	+	$p_t(\theta)A_t$	no	+	✓
4	$p_t(\theta) < 1-\epsilon$	−	$(1-\epsilon)A_t$	yes	−	**0**
5	$p_t(\theta) > 1+\epsilon$	+	$(1+\epsilon)A_t$	yes	+	**0**
6	$p_t(\theta) > 1+\epsilon$	−	$p_t(\theta)A_t$	no	−	✓



PPO

Proximal Policy Optimization

updated **policy** nahe (***proximal***) zur alten policy ist?

Wir stellen sicher dass die **ratio nahe bei 1** ist:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Via **clip**ping, oder andere **loss**es:

custom_loss = **(1-r)^2** *punish* r>>1
loss += custom_loss **prinzipiell** einfach **bestrafen**
schlechtes toy example, geht nur um das Prinzip in Wirklichkeit **KL**

log(a/b) = log a - log b
softmax = e(x) / ∑(e(xi))

Vorschau: Information und Entropie

Shannon-Information := -log(p) self Information (≠ Fischer Information)

p=1 => -log(1) = 0 keine 'Information' wenn ein sicheres Ereignis eintritt

p≈0 => -log(0) ≈ ∞ maximale 'Information' wenn ein seltenes Ereignis doch eintritt

"punktweise Überraschung"

Entropie:

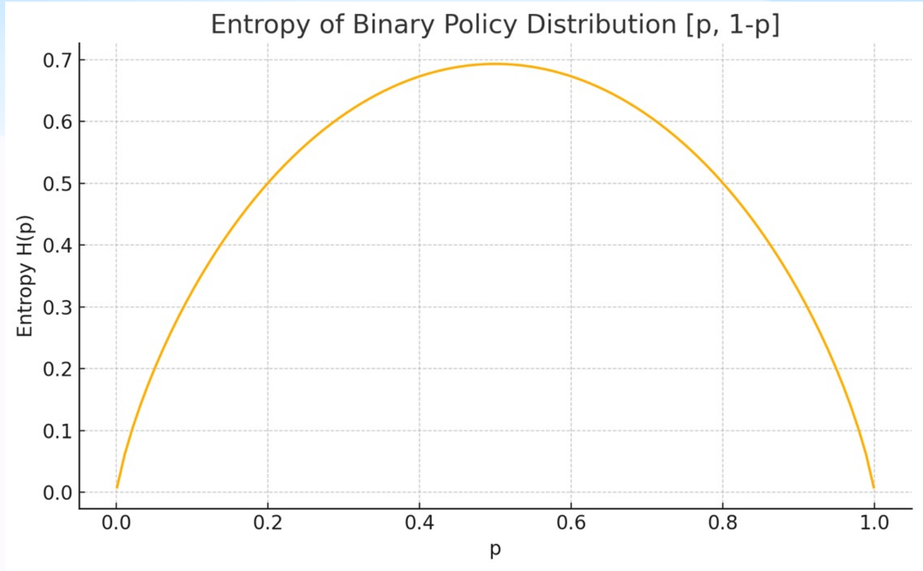
Entropie (mittlere Shannon-Information):

durchschnittliche Überraschung / Unordnung (einer Verteilung)

$$H(p) = -\sum p_i * \log(p_i) \quad \sum p_i = 1$$

Bei Unfairer Münze die immer Kopf/Zahl ergibt, gibt es keine Überraschungen => H=0

Bei Fairer Münze die mal Kopf/Zahl ergibt, gibt ist Ergebnis nicht vorhersehbar "Überraschungsfaktor H=.7"



Wird verwended z.B. in **SAC, PPO, MaxEntropy IRL**

KL Divergenz

KL Kullback-Leibler-Divergenz misst wie **nah** unsere neue policy(?) an der alten ist (so wie **PPO** mit log, mit clipping 1 ≈ 1.44 bits?)

Je kleiner die KL-Divergenz, desto ähnlicher sind die Policies:

DKL=∑π(a)·log(π(a)/π'(a))

$$D_{\mathrm{KL}}(\pi_{\mathrm{alt}} \parallel \pi_{\mathrm{neu}}) = \sum_{a} \pi_{\mathrm{alt}}(a \mid s) \log \left(\frac{\pi_{\mathrm{alt}}(a \mid s)}{\pi_{\mathrm{neu}}(a \mid s)} \right)$$

This disables them while they are already minimized well to focus learning on the prediction loss

Sum is always **positive!**
Some terms **might be negative** but since π is probability π(i) in [0,1] and ∑π(i)=1 => cancel out (Gibbs inequality)

Fantastic KL Divergence and How to (Actually) Compute It
https://youtu.be/tXE23653JrU

KL Divergenz

KL Kullback-Leibler-Divergenz

Ist eng mit dem Konzept von **Cross Entropie** verwandt:

Die **Cross-Entropie** zwischen P und Q ist definiert als:

H(P,Q)= - ∑**P(x)**log(Q(x))

Die Self-**Entropie** von P ist definiert als:

H(P)= - ∑P(x)log(P(x))

Zeige: **KL Divergenz**:
D_{kl} = H(P,Q) - H(P)
KL = **Cross-Entropie - Entropie**

D_{kl}= - ∑**P(x)**log(Q(x)) - - ∑P(x)log(P(x)) = ∑P(x)(log(P(x))-log(Q(x)) = ∑P(x)(log(P(x)/Q(x))

D_{kl}=∑π(a)·log(π(a)/π'(a)) = … 💡 Nutze log(a/b) = log(a) - log(b)! (Q=neue policy P=alte)

$$D_{\mathrm{KL}}(\pi_{\mathrm{alt}} \parallel \pi_{\mathrm{neu}}) = \sum_{a} \pi_{\mathrm{alt}}(a \mid s) \log \left(\frac{\pi_{\mathrm{alt}}(a \mid s)}{\pi_{\mathrm{neu}}(a \mid s)} \right)$$

Für One Hot Encoded Target fallen die beiden Konzepte tatsächlich zusammen weil die Entropie von One Hot and Coding null ist!

KL Divergenz

KL Kullback-Leibler-Divergenz

Zum Messen der 'Nähe' von policies.
Anwendungen:

Maschinelles Lernen / Deep Learning:

– Verlustfunktionen in Klassifikationsaufgaben (z. B. Softmax + cross-entropy ist KL).

– Variational Inference: Approximation von Posteriors (z. B. in Variational Autoencoders).

– Generative Modelle (GAN-Varianten, Diffusion Models).

• **Statistik**:

– Modellselektion (AIC, BIC basieren auf KL).

– Hypothesentests und Likelihood-Ratios.

• **Signalverarbeitung & Informationstheorie**:

– Quantifizierung von Redundanz, Kanaloptimierung.

– Sprach- und Bildverarbeitung (z. B. Speaker Verification, Texture Analysis).

• **Naturwissenschaften**:

– Bioinformatik (Vergleich von Sequenzmotiven, Populationsverteilungen).

– Neurowissenschaften (Vergleich von Feuerratenverteilungen).

• **Ökonomie / Sozialwissenschaften**:

– Messen von Ungleichheiten in Wahrscheinlichkeitsverteilungen (z. B. Marktanteile).

KL Kullback-Leibler-Divergenz

Anwendung:

Extra Loss term kann einfach mit eingebaut werden!

$$D_{\mathrm{KL}}(\pi_{\mathrm{alt}} \parallel \pi_{\mathrm{neu}}) = \sum_{a} \pi_{\mathrm{alt}}(a \mid s) \log \left(\frac{\pi_{\mathrm{alt}}(a \mid s)}{\pi_{\mathrm{neu}}(a \mid s)} \right)$$

KL-Divergenz ein mathematischer Anker für **Stabilität, Exploration und Approximation** in praktisch allen state-of-the-art RL-Algorithmen.

"𝔼rwartungen / **Funktionen"**

Sind unseren Größen Q, V, G stets einheitlich definiert?
Alle Werte/Funktionen/Formeln gibt es in drei Formen:

- **empirisch (konkret)**
- **geschätzt (modellbasiert)**
- **theoretisch (Erwartungswert unter Verteilung)**

G kann ein **konkreter Wert** sein, **berechnet** aus **gesampelt**en **r**
G kann ein **gemittelter Wert** sein, **berechnet** aus **batch**
G kann ein **erwarteter Wert** sein, **geschätzt** durch ein **Q-**Netzwerk
G kann ein **Erwarteter Wert** $\mathbb{E}$ sein, im Sinne im Kontext einer **Verteilung θ unseres Netzwerks**
G kann ein **Erwarteter Wert** $\mathbb{E}$ sein, im Sinne im Kontext einer theoretischen **Verteilung p** unserer Umgebung

$G_t = \sum(i=0, i<T-t)\ \gamma^i\ r_{t+i}$
$G_t = \mathbb{E}\ \sum(i=0, i<T-t)\ \gamma^i\ (r_{t+i} \mid \pi)$

Q intuitiv als Qualität einer Aktion
Q = $\mathbb{E}(G_t)$ abstrakte / theoretische Definition von Quality vs MDP
q(s, a) = $\sum p^* [r + \gamma \sum_{a'} \pi(a' \mid s')\ q^p(s', a')]$ exakte **Schätzung** mit Bellman Gleichung (im limit konvergiert Schätzung gegen "wahren Wert")
Q <= Q + $\alpha (r + \gamma Q(s',a') - Q)$ pragmatischer **Schätz-**Ansatz mit **updates** z.B. bootstrapping, **DQN**, …

Nachtrag: *Return*

G Gain expected cumulative reward ≈ **Return R** function **synonym**? JA! Oder Nicht ganz!?
⚠️ Vermeide *Return* **R** wegen Verwechslungsgefahr mit *reward* **r**!

G = ∑r "**G**ewinn ist Summe der **r**ewards"
G = ∑gamma^i * r G_t **Expected return starting from time *t***

V=E(G | s) *local*!
Q=E(G | s,a)

R^ = J(π) := E(s≈μ,a≈π)[Q] 💡 overall *global* expected return
R^ = ∑μ(s)∑$_a$ π*Q(s,a) # **μ state distribution** 💡

μ(s)=∑$_t$gamma^t*p(s_t|π) # the expected **frequency of visiting state s** under policy 💡

eigentlich E[R] so Q so **b**ellman?

Custom Loss

Loss = … beliebig setzbar …
Loss = **mathematisch motiviert** :
Loss = classical Policy Gradient :
Loss = -log_prob * **Phi 'surrogate'**

letzteres hat immer selben Erwartungswert! nur andere Varianz bei anderem Phi ≈ Streuung

Sei

$$g(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)\, G], \qquad L_\Phi(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)\, [G - \Phi(s)]].$$

Da $\sum_a \pi_\theta(a \mid s) = 1$ gilt

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)\,] = \nabla_\theta \sum_a \pi_\theta(a \mid s) = \nabla_\theta 1 = 0,$$

folgt unmittelbar

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)\, \Phi(s)\,] = 0 \quad \Longrightarrow \quad L_\Phi(\theta) = g(\theta).$$

- Erwartungswert: unverändert (unbiased), unabhängig von der Wahl von Φ, solange Φ nicht von a abhängt.
- Varianz: $Var[\,\nabla_\theta \log \pi_\theta(a \mid s)\, (G - \Phi(s))\,]$ hängt stark von Φ.

Die optimale (minimal-Varianz)-Wahl ist

$$\Phi^\star(s) = \mathbb{E}_{\pi_\theta}[G \mid s] = V^{\pi_\theta}(s),$$

RL bei LLM

a) Finetuning
b) Environment python evaluation reward = 1 bei richtiger Lösung

Man nimmt Foundation Model (GPT-3)
Gib 100000 Mathe Aufgaben und löse sie (geleitet von PPO/GRPO)

<think>
mach was du willst
<think>
<solution>
wert
</solution>

Referate / Extra Wunsch Themen

Inverse Reinforcement
Meta Reinforcement Learning

Novelty Exploration **Surprise**

Hierarchisches Reinforcement Learning (arc!)

RL + **LLM** (DeepSeek, …)

Dreamer

Alpha**Zero**, **µZero?** AbsoluteZero

Praxis: **Eigene Umgebungen**

FactoryIO.com / NVIDIA / Siemens **Digital Twin**

bullet.py robot arm

Referate : Extra Wunsch Themen

Wiederholung
RL in Sprachmodellen **LLM** (RLHF / **reasoning** / pokemon)
Aufmerksamkeitsmodelle (?)
priorisierte Wiederholung (?)
RL in Spielen? (Vitalij)
Algorithmen: **SAC, TD3**
virtual twin / factory.io / Kuka?
Sparmodell? **BudgetEnv**? *Custom Environment*.
mit baseline evaluieren?

reward shaping?
https://gibberblot.github.io/rl-notes/single-agent/reward-shaping.html
https://arxiv.org/abs/2011.02669

einfachster State of the Art Algorithmus **SoTa ±** ?
most **impressive example** a) running locally b) trainable locally c) tuneable locally
RL in Spielen welche Spiele wurden (noch nicht) gelöst.

Vokabular

PPO

MARL

lambda in GAE

Emergent

PPO
KL
(Cross/Self)Entropy
centralized

Extra Themen

KI **continuous** learning (Referat?)

Bilder Generieren? RL? **Flow Matching!** ***RL*** (Karsten)

Zusammenfassung!

Ein Algorithmus für alles!

Prozessoptimierung

Custom Environment einfach env.step() selbst!

State of the Art