Reinforcement Learning

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2026-01 Dozent: Karsten Flügge

Themen des Tages

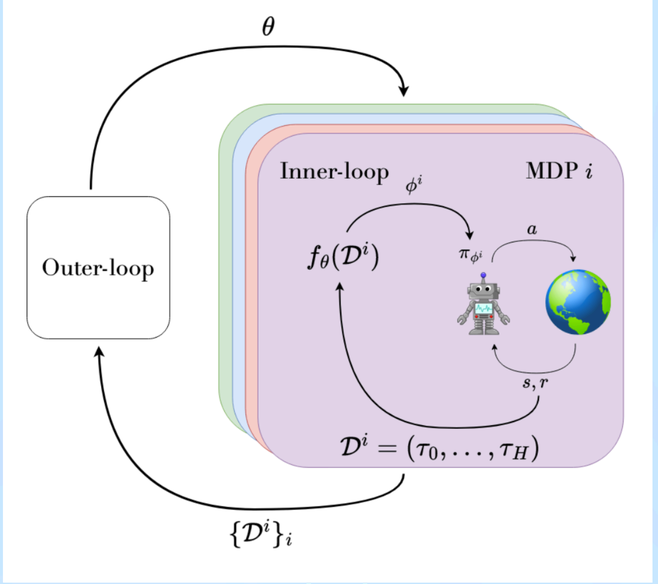
Tag 12

Meta Reinforcement

Hierarchic RL <> HR

Meta Reinforcement

"learn to reinforcement learn"
`Vortrainieren / Transfer Learning` in RL:



Meta Reinforcement Algorithmen lernen, sich schnell an **neue Aufgaben anzupassen**, indem man **frühere Erfahrungen** nutzt. Der Fokus liegt darauf, die **Effizienz** von Lernalgorithmen zu **verbessern**, sodass sie mit **minimalen Daten** über verschiedene Aufgaben hinweg **generalisieren** können.

Während wir in RL eine Policy **π** lernen, lernt ein Meta RL "**controller**"
eine Funktion **f** welche eine **policy generiert,** zu gegebenen **D**aten $\mathbf{D_i}$
(gesampelte episodische Trajektorien τ).

$f(D_i)=\pi_i$ controller liefert Vorwissen "Startpolicy" an nächsten Agenten

Few-shot learning

Der Begriff **few-shot learning** bezeichnet ein Lernparadigma, bei dem ein Modell mit nur **wenigen Beispielen pro Klasse** oder Aufgabe trainiert wird – im Gegensatz zu klassischen Deep-Learning-Verfahren, die oft tausende Trainingsbeispiele benötigen.

Heiliger Gral des Maschinelearnings … "so wie Menschen".

Typische Methoden

- Matching Networks
- Prototypical Networks
- MAML (Model-Agnostic **Meta-Learning**)
- **GPT**s!

Beispiele und Anwendungen

- **Klassifikation** seltener Objekte
- Roboter lernen neue Bewegungen mit wenigen Demonstrationen
- Seltene Krankheiten (mit Statistik + Intuition)

One/**Zero-Shot Learning**: Spezialfall mit **gar keinem** Beispiel, nur mit Beschreibung oder semantischer Information (z.B. Wortvektor)

Few-shot learning

Der Begriff **few-shot learning** bezeichnet ein Lernparadigma, bei dem ein Modell mit nur **wenigen Beispielen pro Klasse** oder Aufgabe trainiert wird – im Gegensatz zu klassischen Deep-Learning-Verfahren, die oft tausende Trainingsbeispiele benötigen.

Aber:

Few-shot learning Fähigkeit ergibt sich oft nachdem man schon tausende **andere** Erfahrungen gesammelt hat!

"Nachdem wir 1000 Aufgaben mühselig gelernt haben,
können wir die 100**1**^{te} Aufgabe ganz schnell lösen."

"Nachdem wir 1000 Objekte erkannt haben,
können wir die 100**1**^{te} Objekt ganz schnell erkennen."

Meta Reinforcement

A Survey of Meta-Reinforcement Learning

https://arxiv.org/abs/2301.08028

2017 and 2022

Where RL learns a policy, meta-RL learns the RL algorithm f that outputs the policy.
This **does not remove all of the human effort** from the process, but shifts it from directly designing and implementing the RL algorithms into **developing** the **training environments** and **parameterizations** required for learning parts of them in a **data-driven** way.

motivation:

- often poor data efficiency
- limited generality of the produced policies

cost:

meta-learning requires **significantly more data** than standard learning

improved sample efficiency at test time, at the expense of sample efficiency during training and generality at test time

Ziel ist es, eine **Strategie** zu erlernen, die in der Lage ist, sich mit möglichst wenig Daten an jede **neue Aufgabe** aus der Aufgabenverteilung anzupassen. single/**few-shot-adaptation**

Meta Reinforcement

Alternative: *Generalisierung*

Einfach ein Agent / Netz nacheinander auf verschiedene Aufgaben loslassen.

Beispiel: Kochroboter

Unterschied

- Oberbegriff:

- **Generalisieren** / Transfer Learning
  - bei Supervised Learning
- **Immitation** Learning (mountain_car_trainer_policy.py) <<
- **Model** Basiert (ChatGpt) (latentes Weltmodell)
- Inverse RL
- **Meta** RL (prinzipiell natürlich besser als blind ein Netz auf verschiedene Tasks loszulassen, *aber* Weltmodell noch besser!)

Beispiel: Kochroboter

Frage: wie kann da Meta RL zum Einsatz kommen?

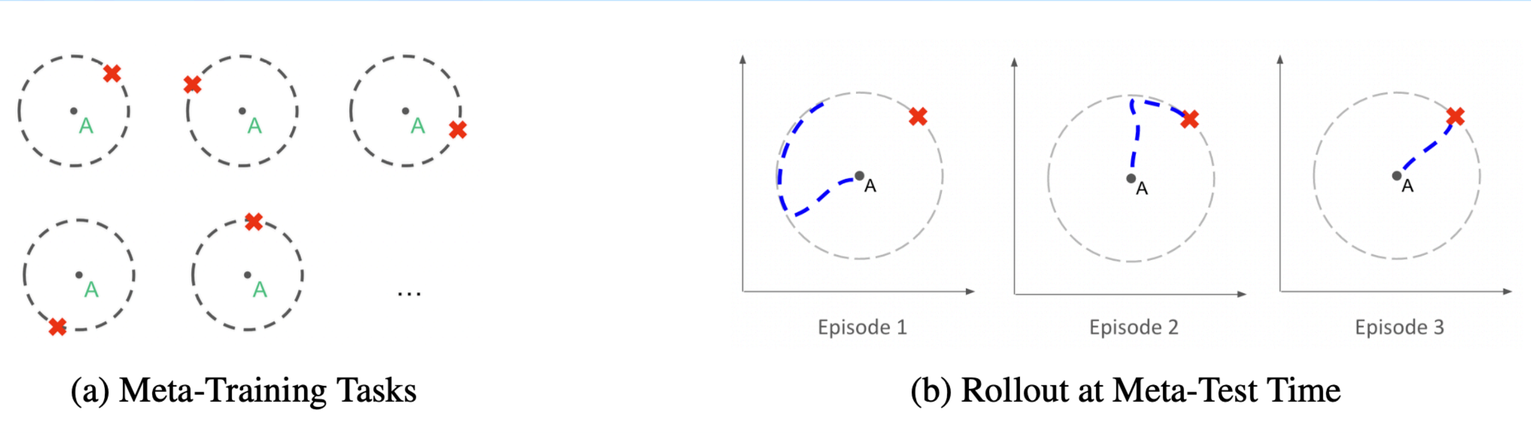
"Zwei Milliarden Rezepte"

Circus Group
https://www.circus-group.com/

Sollte in **verschiedenen Küchen** funktionieren.
Adaption an anderes Schrank / Kochplatten layout …
Versteckte Koch-**Utensilien** finden!
Abwasch nach Roulade ≈ Abwasch nach Pizza.

Alternative: **Fine-Tuning** für jede Küche/Utensil.
Unterschied: Generalisierung?

Beispiel: Goal Posts



(a) Meta-Training Tasks

(b) Rollout at Meta-Test Time

Ein Agent A lernt ein Ziel X auf einem Kreis zu erreichen.
Zur Testzeit befindet sich das Ziel am verschiedenen Flecken des Kreises.

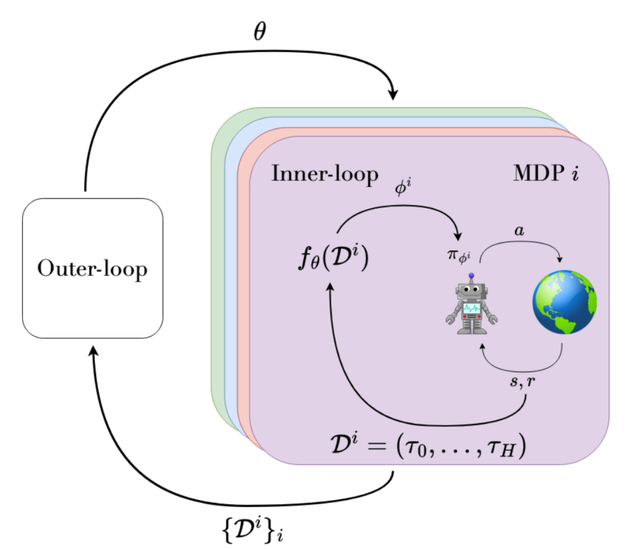
Normale (nicht meta) Algorithmen könnten sehr lange brauchen um das Ziel zu finden, lernen meist spezifische Lösung für eine feste Zielposition.

Beim Meta Lernen wird erkannt, dass das Ziel sich immer auf dem Kreis befindet, für eine konkrete Aufgabe wird dann einfach der Kreis abgesucht.

Inner & Outer Loop

Während wir in RL eine Policy π lernen, lernt Meta RL eine Funktion **f** welche eine **policy generiert**:
f erzeugt die Agenten-Gewichte ϕ von π^{ϕ}.

In der **inneren Schleife** werden neue **Daten** also episodische **Trajektorien** gesammelt: $D^{\iota}=(\tau_0, \ldots, \tau_n)$
Solche Datensätze der inneren Schleife nennt sich auch **Meta-Trajektorien** $\{D^{\iota}\}_j$ (**D**emonstrationen) welche den **Trainingsdatensatz** für die **äußere Schleife** bilden.



$$f(\mathcal{D}) : ((\mathcal{S} \times \mathcal{A} \times \mathbb{R})^T)^{\bar{H}} \rightarrow \Phi$$

Die **äußere Schleife** berechnet upgedatete **Meta-parameter** für unsere **Policy erzeugende Funktion f^{θ}.**
Also wenn unsere **Funktion f ein Netz** ist parametrisiert mit **Gewichten θ**, dann wird θ per Gradient verbessert.

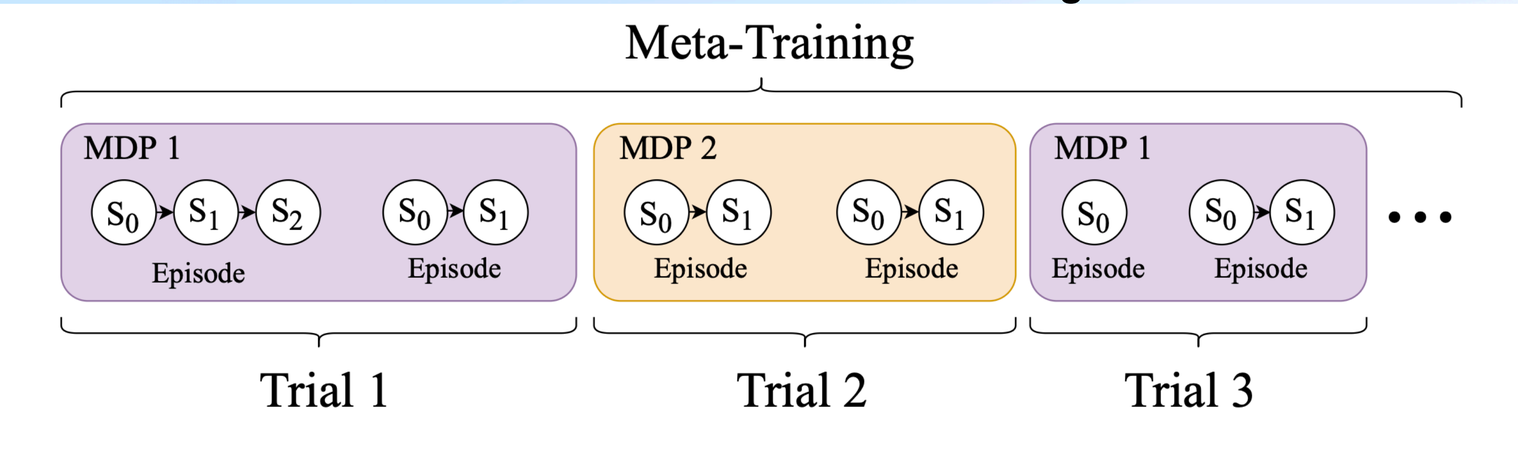
Inner & Outer Loop

In der **inneren Schleife** werden nicht nur neue **Daten** also episodische **Trajektorien** gesammelt
$D^{\iota}=(\tau_0, \ldots, \tau_n)$ $\{D^{\iota}\}$

Die policy **π** für alle training "**tasks**" (**M**DPs) soll auch **optimiert** werden wie bisher, dass heisst, wir wollen alle Probleme 'lösen'.
Die **M**enge aller Tasks/**M**DPs beim Training bezeichnet man auch als $\mathcal{M}$, manchmal zufällig ausgewählt gemäß p($\mathcal{M}$).

Prinzipiell können die **Aufgaben** von **unterschiedlicher Natur** sein in der Praxis sind es aber **meistens Parametrisierungen** von *ähnlichen* Aufgaben mit gemeinsamen Zustands und Aktionsraum.

Ein Durchlauf der **inneren Schleife** in welcher ein bestimmtes MDP Problem optimiert wird nennt sich auch **trial** (or **lifetime/rollout**).
Ein Durchlauf der **äusseren Schleife** nennt sich **Meta-training.**



Updates

Das **Update** der Agenten-policy Parameter **φ** von π^{Φ} anhand von Daten **D** durch unsere Meta-funktion **f** erfolgt je nach Algorithmus mehr oder weniger **häufig.**

Die Anzahl der Episoden in einem Versuchsdurchführung denotiert man manchmal mit **H** für task-**Horizont**

Je nach Setting kann man mehr oder weniger lange **explorieren,** den Anteil der Durchläufe bei welchen die **Ausgabe ignoriert** wird bezeichnet man als '**shots**' **K** (**exploration** Horizont)

$D_{K:H}$

K = 0 keine explorations Episoden, no_grad, noch kein Lernen

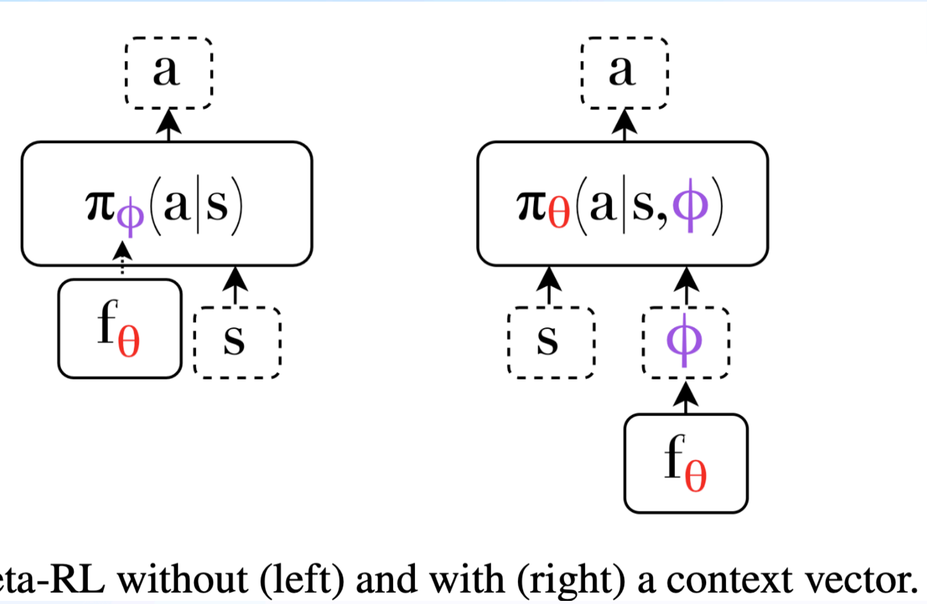
out-of-distribution wenn p(M) beim beim Trainieren anders ist als beim Testen

Formal versuchen wir den **erwarteten Gewinn über alle Tasks** zu maximieren

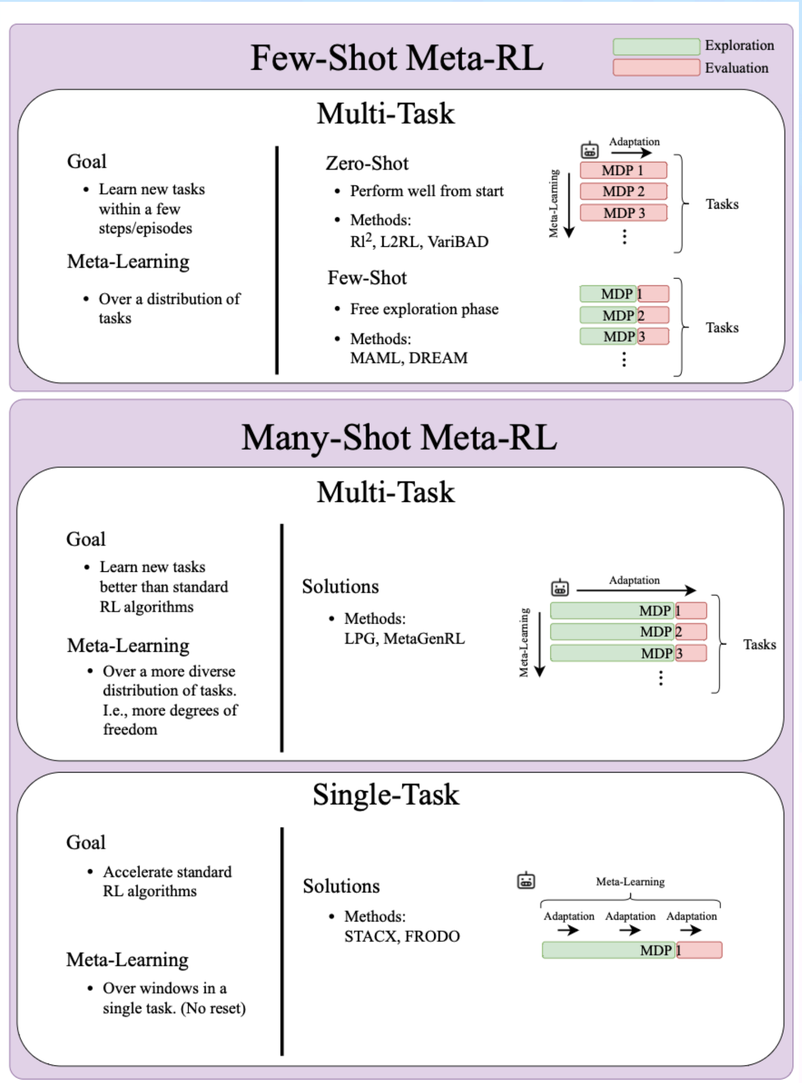
$$\mathcal{J}(\theta) = \mathbb{E}_{\mathcal{M}^i \sim p(\mathcal{M})}\left[\mathbb{E}_{\mathcal{D}}\left[\sum_{\tau \in \mathcal{D}_{K:H}} G(\tau) \middle| f_\theta, \mathcal{M}^i\right]\right]$$

Aspekte / Kategorien

task-**horizon H** is **short** (a few episodes) **or long** (hundreds of episodes or more)
task distribution **p(M)** contains **multiple** tasks **or** just **one** task
meta-exploration **K<H** few / many shot
Outer-loop **on**/off **policy** wie oft aktualisieren wir **f**
(un)**supervised** (vorher behavioral cloning)
(meta)**imitation** (trainer-policy, model based)
with(out) **context vector**
phasic PPG / black box (?)
gradient vs **skill** (gradient-free evolutional)
"learn to pick or combine what you already know"

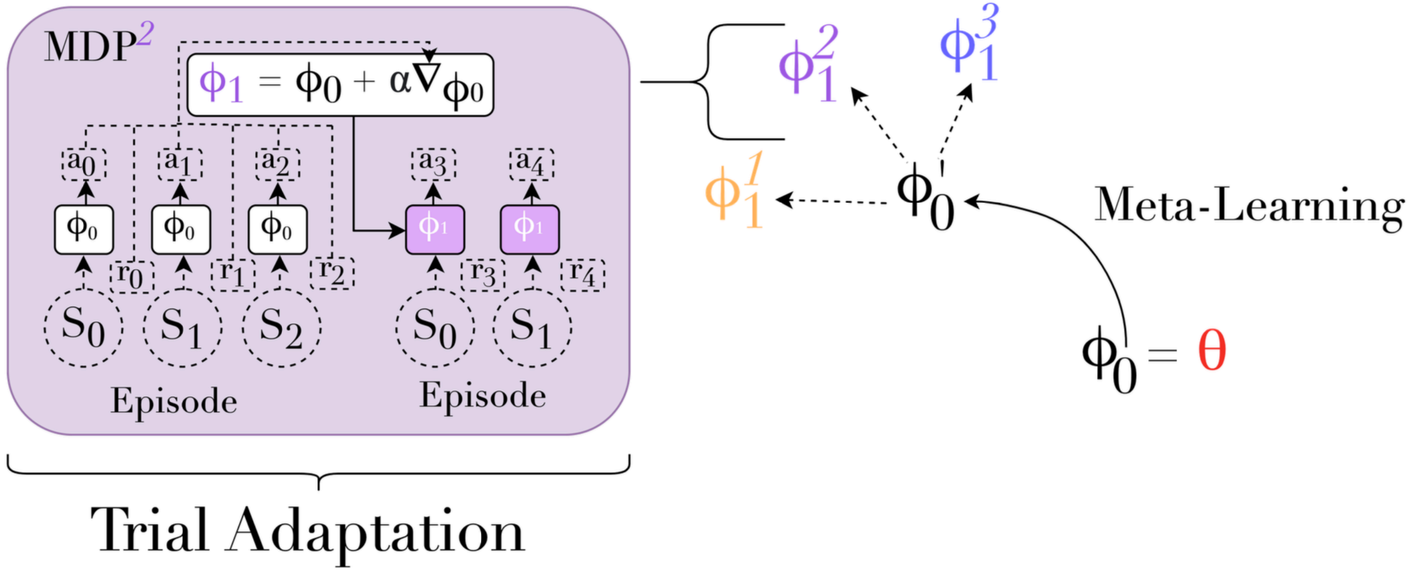


meta-RL without (left) and with (right) a context vector.



Model-Agnostic Meta-Learning (MAML)

Unsere Metafunktion f verwendet dessen
Controller Netz Parameter **θ** zum **initialisieren** der **policy** Parameter φ_0 von π^{Φ}:

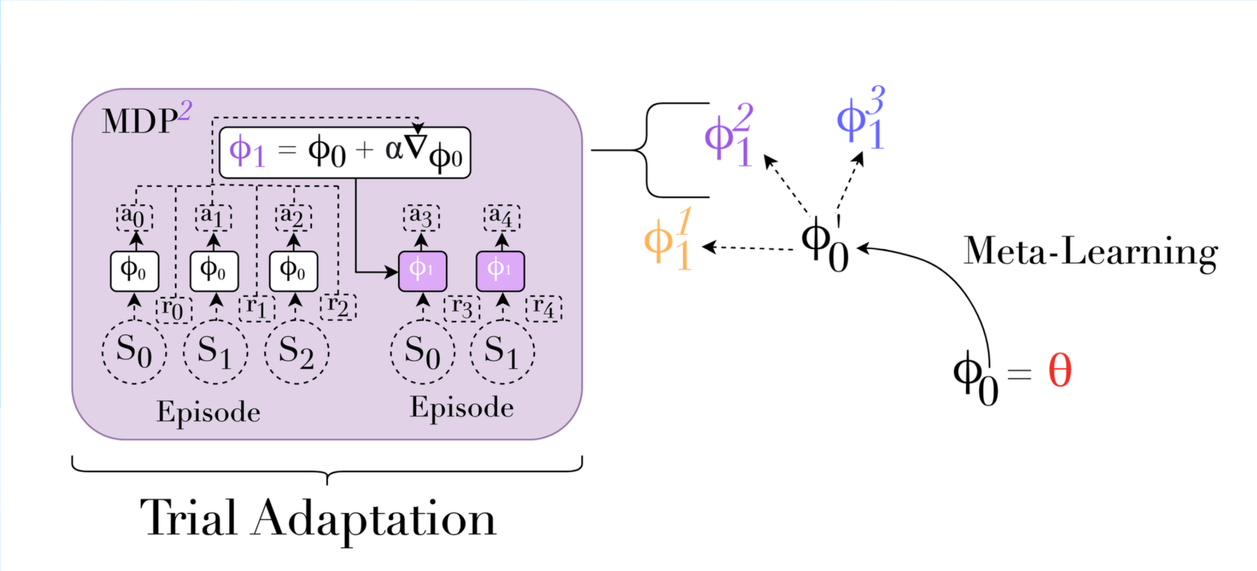


Der zentrale Punkt ist, dass eine solche innere Schleife eine **differenzierbare Funktion der Anfangsparameter** ist, sodass die **Initialisierung mittels Gradientenabstieg optimiert** werden kann, um einen **guten Ausgangspunkt** für das Lernen auf Aufgaben aus der Aufgabenverteilung zu bieten.

https://arxiv.org/abs/1703.03400

Model-Agnostic Meta-Learning (MAML)

Unsere Metafunktion f verwendet dessen
Controller Netz Parameter **θ** zum **initialisieren** der **policy** Parameter φ_0 von π^{Φ}:



Was ist der Unterschied zu **Target-Netz** Ansatz? Target-Netze sind passive Kopien von Gewichten.

https://arxiv.org/abs/1703.03400

Model-Agnostic Meta-Learning (MAML)

Unsere Metafunktion f verwendet dessen
Controller Netz Parameter θ zum **initialisieren** der **policy** Parameter φ_0 von π^{Φ}:

```
policy_net = controller_net.copy()
# controller.load(policy_net.weights) :
policy_net.weights = controller.weigths
```

controller hat Gewinn : ∑gain der Agenten.
damit können wir controller mit klassischem RL trainieren.

reward shaping:

- frage ist immer: was liefert Umgebung schon? was hat Agent schon für custom reward
- wie lange haben die Sub-Task Agenten für Aufgabe gebraucht?
  (custom reward für controller)
- Gesamt Gewinn des Agenten
- wie **präzise / effizient**
- wie **wichtig** war skill Sub-Task (Beispiel: Einparken < erfolgreiches Überholen)

Model-Agnostic Meta-Learning (MAML)

Unsere Metafunktion **f** verwendet dessen
Parameter **θ** zum **initialisieren** der policy Parameter φ_0 von π^{Φ}:

Der zentrale Punkt ist, dass eine solche innere Schleife eine **differenzierbare Funktion der Anfangsparameter** ist, sodass die **Initialisierung mittels Gradientenabstieg optimiert** werden kann, um einen **guten Ausgangspunkt** für das Lernen auf Aufgaben aus der Aufgabenverteilung zu bieten.

$$\phi_1^i = f(\mathcal{D}_0^i, \phi_0) = \phi_0 + \alpha \nabla_{\phi_0} \hat{J}(\mathcal{D}_0^i, \pi_{\phi_0})$$

J(D,π) berechnete Schätzung der Gewinne von π während Aufgabe M^i (MDP task i) auf Daten D^i (?)

Um die initialen Parameter **θ = ϕ_0** in der Outer-Loop zu **aktualisieren**,
sammelt MAML eine zweite Datenmenge D_1 unter Verwendung der angepassten Politik **ϕ_1**
und berechnet den Gradienten der daraus resultierenden Erträge:

$$\phi_0' = \phi_0 + \beta \nabla_{\phi_0} \sum_{\mathcal{M}^i \sim p(\mathcal{M})} \hat{J}(\mathcal{D}_1^i, \pi_{\phi_1}^i)$$

Model-Agnostic Meta-Learning (MAML)

Unsere Metafunktion **f** verwendet dessen
Parameter **θ** zum **initialisieren** der policy Parameter φ_0 von π^{Φ}:

Algorithm 1 MAML for Reinforcement Learning

1: Initialize meta-parameters $\phi_0 = \theta$
2: **while** not done **do**
3: Sample tasks $\mathcal{M}^i \sim p(\mathcal{M})$
4: **for** each task $\mathcal{M}^i$ **do**
5: Collect data $\mathcal{D}_0^i$ using the initial policy π_{ϕ_0}
6: Adapt policy parameters using a policy gradient step: $\phi_1^i = \phi_0 + \alpha \nabla_{\phi_0} \hat{J}(\mathcal{D}_0^i, \pi_{\phi_0})$
7: **end for**
8: Update ϕ_0 using the meta-gradient: $\phi_0' = \phi_0 + \beta \nabla_{\phi_0} \sum_{\mathcal{M}^i \sim p(\mathcal{M})} \hat{J}(\mathcal{D}_1^i, \pi_{\phi_1}^i)$.
9: **end while**

β ist unsere meta-lernrate

Der zentrale Punkt ist, dass eine solche innere Schleife eine **differenzierbare Funktion der Anfangsparameter** ist, sodass die **Initialisierung mittels Gradientenabstieg optimiert** werden kann, um einen **guten Ausgangspunkt** für das Lernen auf Aufgaben aus der Aufgabenverteilung zu bieten.

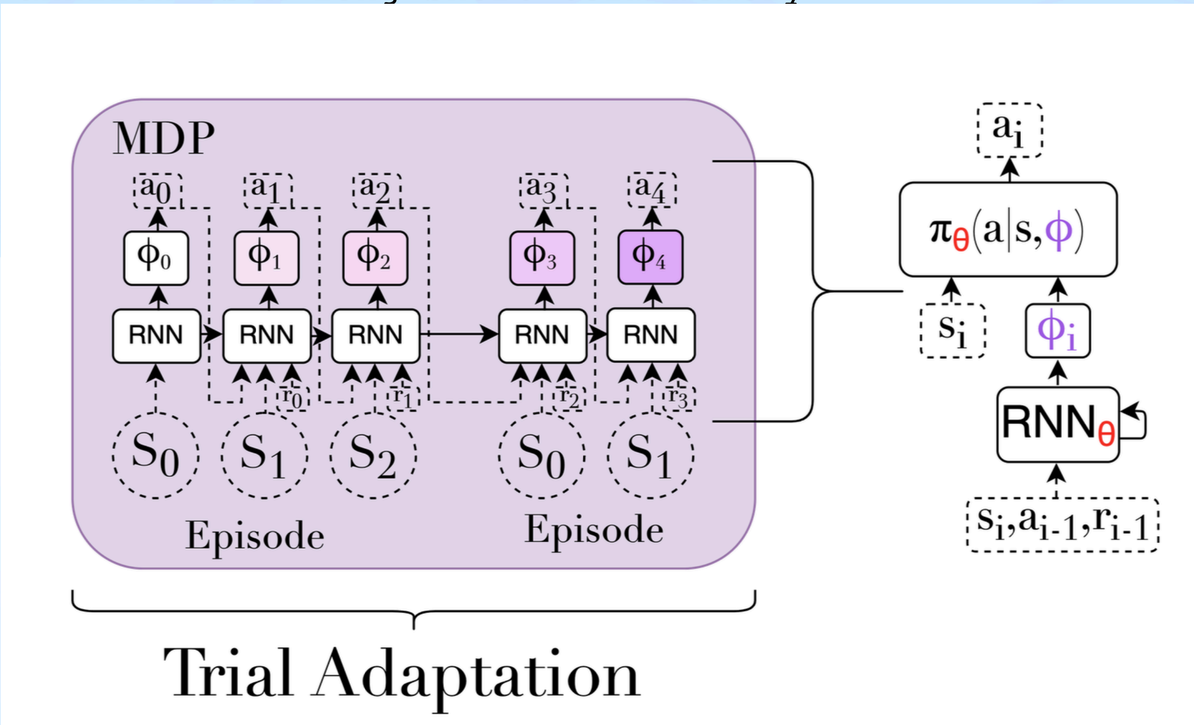
https://arxiv.org/abs/1703.03400

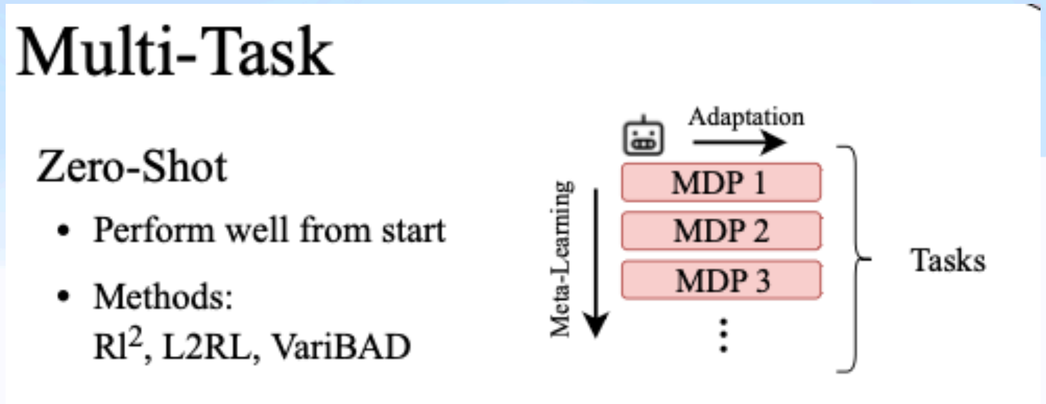
RL² "Fast RL via Slow RL"

Idee: Speichere Policy **Historie** in **RNN f**

Der **verborgene Zustand** wird während eines Versuchs nicht zurückgesetzt und stattdessen über Episodengrenzen hinweg beibehalten.

ähnlich LSTM long short term memory





RL² "Fast RL via Slow RL"

Der verborgene **Zustand** wird während eines Versuchs nicht zurückgesetzt und stattdessen über Episodengrenzen hinweg beibehalten.

Algorithm 2 RL^2 for Meta-Reinforcement Learning

Initialize meta-parameters θ (RNN and other neural network parameters)
while not done **do**
  Sample tasks $\mathcal{M}^i \sim p(\mathcal{M})$
  for each task $\mathcal{M}^i$ **do**
    Initialize RNN hidden state ϕ_0
    Run a continuous trial consisting of multiple episodes
    for each timestep t in the trial **do**
      Observe current state s_t, previous action a_{t-1}, and previous reward r_{t-1}
      Update RNN hidden state with input $[s_t, a_{t-1}, r_{t-1}]$ and parameters θ:

$$\phi_t = f_\theta(s_t, a_{t-1}, r_{t-1}, \phi_{t-1})$$

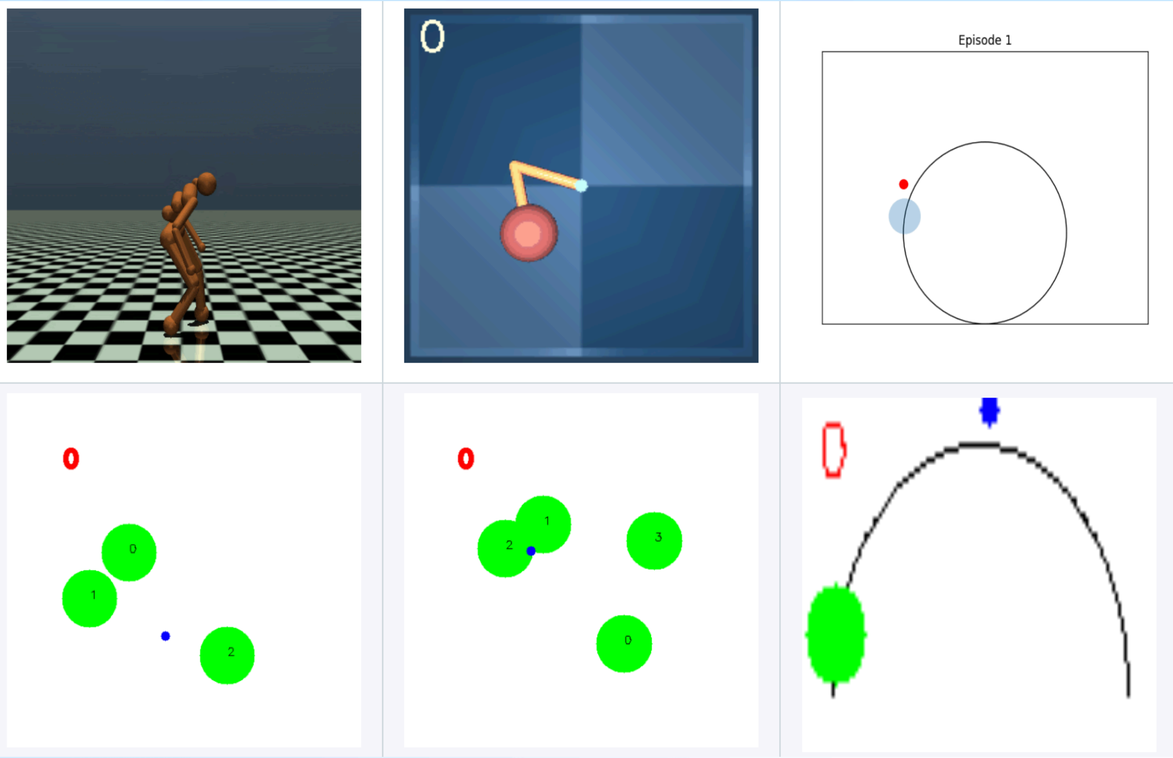
      Sample action $a_t \sim \pi_\theta(\cdot|s_t, \phi_t)$
      Execute action a_t and receive reward r_t
    end for
  end for
  Update meta-parameters θ by optimizing the expected discounted sum of rewards across the entire trial (4).
end while

MAMBA

MAMBA: an Effective World Model Approach for Meta-Reinforcement Learning

model-based RL methods have been successful in solving partially observable MDPs, of which meta-RL is a special case.
better sample efficiency (up to) while requiring very little hyperparameter tuning

https://openreview.net/forum?id=1RE0H6mU7M
https://github.com/zoharri/mamba



⚠️ ≠ https://github.com/kyegomez/MambaTransformer

MAMBA

MAMBA: an Effective World Model Approach for Meta-Reinforcement Learning

https://github.com/zoharri/mamba

⚠️ **Nur in virtueller Umgebung! nutzt mujoco! geht 99%ig nicht :(**

`python3 -m venv venv`
`source venv/bin/activate` nicht vergessen bei requirements.txt virtueller umgebung

PEARL

Algorithm 3 PEARL Inner-Loop

Sample task $\mathcal{M}^i \sim p(\mathcal{M})$
Initialize empty meta-trajectory $\mathcal{D}^i_{0:H}$
for each shot $k = 0, ..., H-1$ **do**
    Sample $\phi \sim q_\theta(z|\mathcal{D}^i_{0:k})$
    Roll out $\pi_\theta(a|s, \phi)$ to collect trajectory τ_k
    Set $\mathcal{D}^i_k = \tau_k$
end for

Optimal Exploration

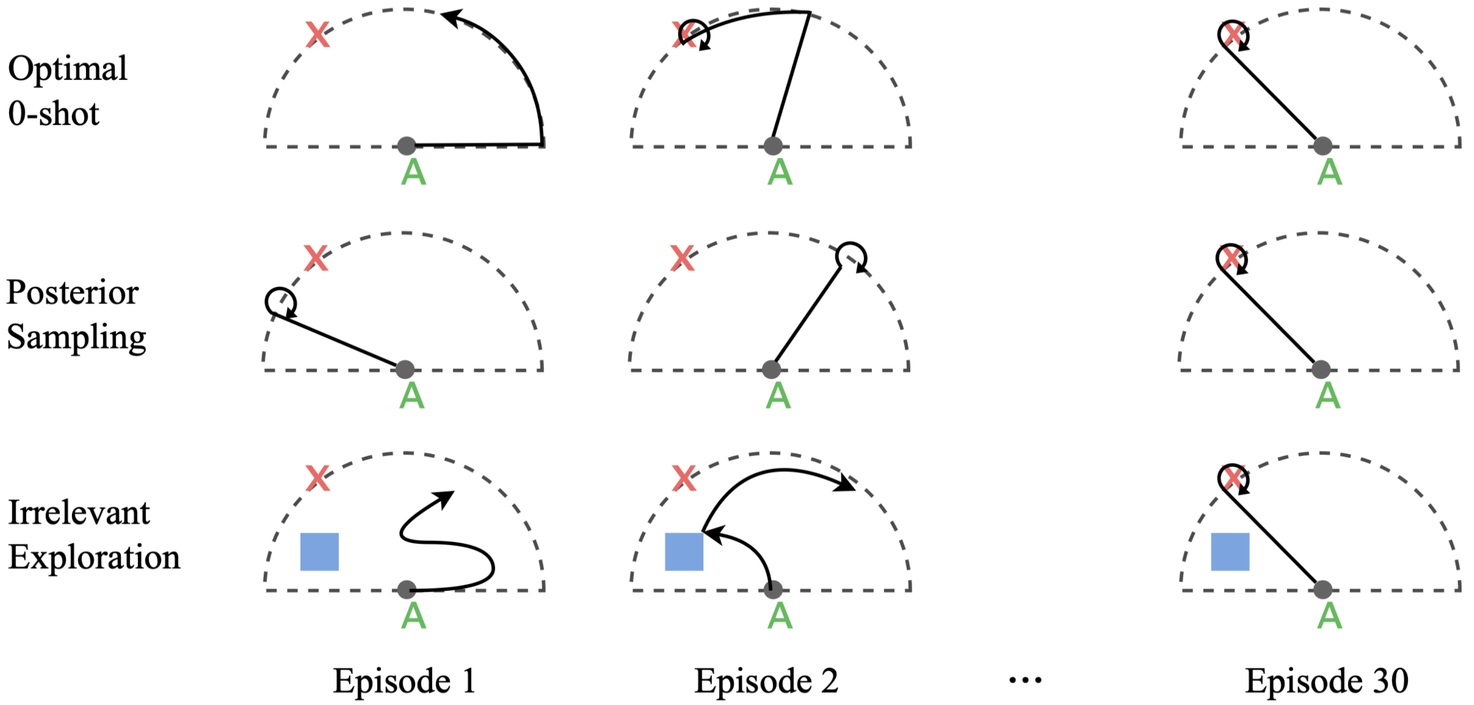


Figure 14: Exploration for a robot chef (green) finding a stove (red) along a circular counter. The first two rows compare optimal exploration and posterior sampling. The third row shows excessive exploration (blue) when irrelevant information is available in the sampled MDP.

DREAM

Decoupling **R**eward-free **E**xploration **a**nd Execution in **M**eta-Reinforcement Learning

Algorithm 4 DREAM Meta-Testing

Sample task $\mathcal{M}^i \sim p(\mathcal{M})$
Initialize empty meta-trajectory $\mathcal{D}^i_{0:K}$
for each exploration $k = 0, ..., K$ **do**
    Roll out exploration policy $\pi_\theta(a|s, \mathcal{D}^i_k)$ to collect trajectory τ_k
    Set $\mathcal{D}^i_k = \tau_k$
end for
Infer $\hat{\phi} \sim q_\theta(\hat{\phi}|\mathcal{D}^i_{0:K})$
for each exploitation episode $h = K, ..., H-1$ **do**
    Roll out exploitation policy $\pi^{\text{multi}}_\theta(a|s, \hat{\phi})$
end for

DREAM

Algorithm 5 DREAM Meta-Training

while not done meta-training **do**
  Sample tasks $\mathcal{M}^i \sim p(\mathcal{M})$
  for each task $\mathcal{M}^i$ **do**
    Initialize meta-parameters, θ, and empty meta-trajectory $\mathcal{D}^i_{0:K}$
    Sample $\phi \sim g_\theta(z|c_\mathcal{M})$, a task embedding with IB
    for each exploitation episode $h = K, ..., H-1$ **do**
      Every other episode, replace ϕ with $\hat{\phi} \sim q_\theta$, to make the meta-training task distribution closer to the meta-test distribution for stability
      Roll out exploitation policy $\pi^{\text{multi}}_\theta(a|s, \phi)$
      Update $\pi^{\text{multi}}_\theta$ and g_θ using task reward and prior
    end for
    for each exploration episode $k = 0, ..., K-1$ **do**
      Roll out exploration policy $\pi_\theta(a|s, \mathcal{D}^i_k)$ to collect trajectory τ_k
      Set $\mathcal{D}^i_k = \tau_k$
      Update $q_\theta(\hat{\phi}|\mathcal{D}^i_{0:k})$ to infer ϕ using variational inference
      Compute $r^{\text{explore}} = -||\phi - \hat{\phi}_t||^2_2 + ||\phi - \hat{\phi}_{t-1}||^2_2 - c$ for constant c, and all timesteps t
      Update π_θ using exploration reward, r^{explore}, i.e. the information gain in q for each transition
    end for
  end for
end while

Related Work / Related Fields

AutoML
meta-supervised learning (meta-SL)
multi-task RL (≠ multi agent)

Ausblick

RL-GPT shows great efficiency on solving diverse Minecraft tasks, obtaining **Diamond at 8% success**

Meta-World+: An Improved, Standardized, RL **Benchmark**

Transformer-Based Model with Tree Search 2022

Andere

PEARL

Graph Structured Surrogate Models (GSSM)

Meta-Model-Based Meta-Policy Optimization (M3PO)

Model-Based Adversarial Meta-Reinforcement Learning (AdMRL)

Model-Based Meta-Policy Optimization (MB-MPO)

Skill based Meta-RL https://arxiv.org/abs/2204.11828

https://pypi.org/project/metagpt/ ?

Code

Leider oft schon vier bis **zehn Jahre alt**

The field of few-shot learning has recently seen substantial advancements.

https://arxiv.org/abs/2301.08028 A Tutorial on Meta-Reinforcement Learning
https://arxiv.org/abs/1810.09502 MAML
https://github.com/AntreasAntoniou/HowToTrainYourMAMLPytorch
https://github.com/tristandeleu/pytorch-**maml**
https://github.com/tristandeleu/pytorch-meta **4 years ago**

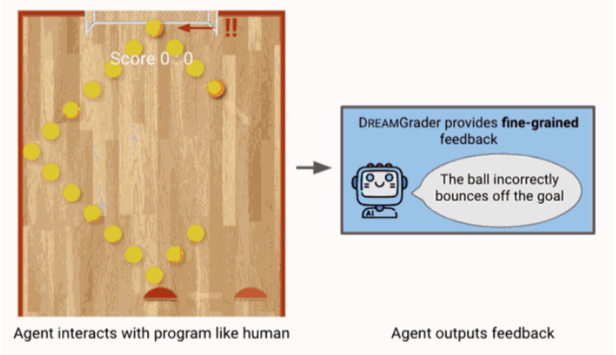
https://github.com/RobvanGastel/meta-rl-algorithms **10 months ago**

https://github.com/danijar/dreamerv3 **≠ dream**

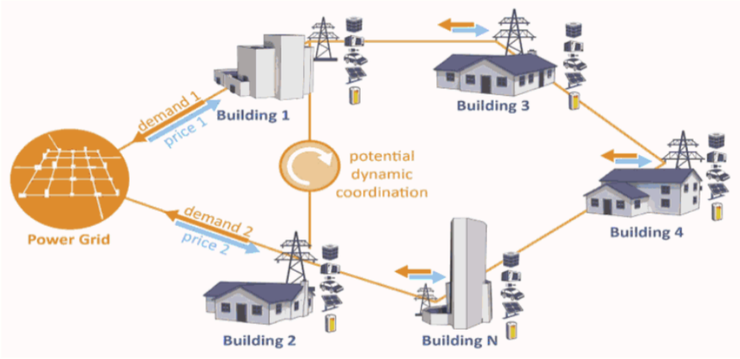
https://github.com/zoharri/mamba **3 weeks ago, basierend auf Dreamer!?!** 2 April 2025

Pause

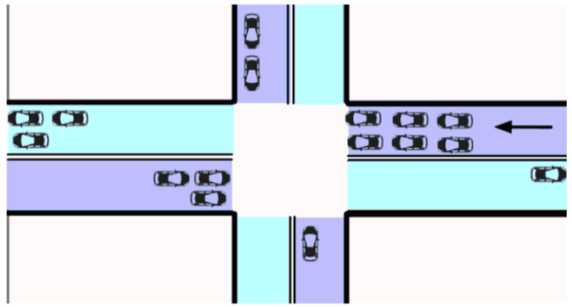
Code grading



Building energy control



Traffic control



Meta-RL

Robot locomotion

Multi-agent RL

Robot manipulation

Pause

Ausblick

1. LLM-basierte Agenten als Meta-Lerner

- **LLMs** (GPT, PaLM, Gemini) werden zunehmend als *few-shot **policy** generators* genutzt.
- Prompting ≈ Meta-Lernen: Policy-Anpassung erfolgt *nicht durch Gradienten*, sondern durch Kontext.
- **Beispiele**:
  - **Voyager** (Minecraft Agent): Lernt Aufgabenstrukturen durch Self-prompting.
  - **ReAct**, **AutoGPT**, **MetaGPT**: Planen, Adaptieren ohne klassische RL-Schleifen.
  - **Open-ended RL** via Tool-use: LLM + Environment Memory (Reinforcement als Re-ranking statt Backprop).

Latent space : Nicht mit Rohdaten arbeiten sondern mit **komprimierten / abstrahierten Daten**

=> pokemon!

Ausblick

2. World Model + Planning statt Meta-RL

- World models (z. B. **Dreamer**, PlaNet, MuZero) bauen generative Umweltmodelle, auf denen Agenten planen.
- Adaption erfolgt durch Planungsdiversifikation, nicht durch Meta-Optimierung.
- **Lernen ≠ Policy Learning**, sondern **Lernen = Modelllernen**.

3. Unsupervised RL / Skill Discovery

- Paradigmawechsel: Statt für viele Aufgaben Meta-Lernen, lernt man **verallgemeinerbare Fähigkeiten**.
- **Beispiele**: DIAYN, DADS, SMM, APS – alle ohne Aufgabenbezug.
- Kombiniert mit **goal conditioning**, kann das als *task-agnostische Meta-RL* gesehen werden.

Open Problems

Meta-RL is an active area of research with an increasing number of applications, and in this nascent field there are **many opportunities for future work.**

6.1 Few-Shot Meta-RL: Generalization

- **Problem**: Poor generalization beyond narrow meta-training task distributions.
- **Challenges**:
  - Narrow benchmarks lead to trivial task inference, limiting learned inductive biases.
  - Broader task distributions introduce **hard exploration** and **distributional shift**.
- **Unsolved Issues**:
  - Design of **benchmarks** with both task diversity and structured inductive bias.
  - Reliable generalization to out-of-distribution (OOD) tasks, especially under significant shift.
  - Graceful failure and selective use of inductive biases for OOD tasks.

Open Problems

Meta-RL is an active area of research with an increasing number of applications, and in this nascent field there are **many opportunities for future work.**

6.2 Many-Shot Meta-RL: Optimization & Benchmarking

- **Optimization Problems**:
  - Outer-loop gradients biased due to truncated inner-loop (e.g., few updates used as surrogate).
  - Non-stationarity in single-task settings prevents consistent outer-loop updates.
  - Trade-off between **bias and variance** in meta-gradient estimation remains unsolved.
- **Benchmarking Issues**:
  - No standardized benchmarks or task-splitting criteria.
  - General-purpose RL algorithm evaluation lacks clarity on what constitutes a “reasonable” task distribution.

Open Problems

Meta-RL is an active area of research with an increasing number of applications, and in this nascent field there are **many opportunities for future work.**

6.3 Offline Meta-RL

- **Settings**:
  - Offline/Offline: Adaptation and training both from static logs—suffers from reward overestimation and poor task identification.
  - Offline/Online: Offline training of exploration policies for online adaptation—plagued by severe distribution shift and ambiguity.
  - Online/Offline: Offline adaptation guided by online meta-training—promising but underexplored.
- **Open Questions**:
  - How to meta-learn adaptation strategies from offline data in complex task distributions.
  - How to systematically relax assumptions about data collection.

6.4 Critical Limitations

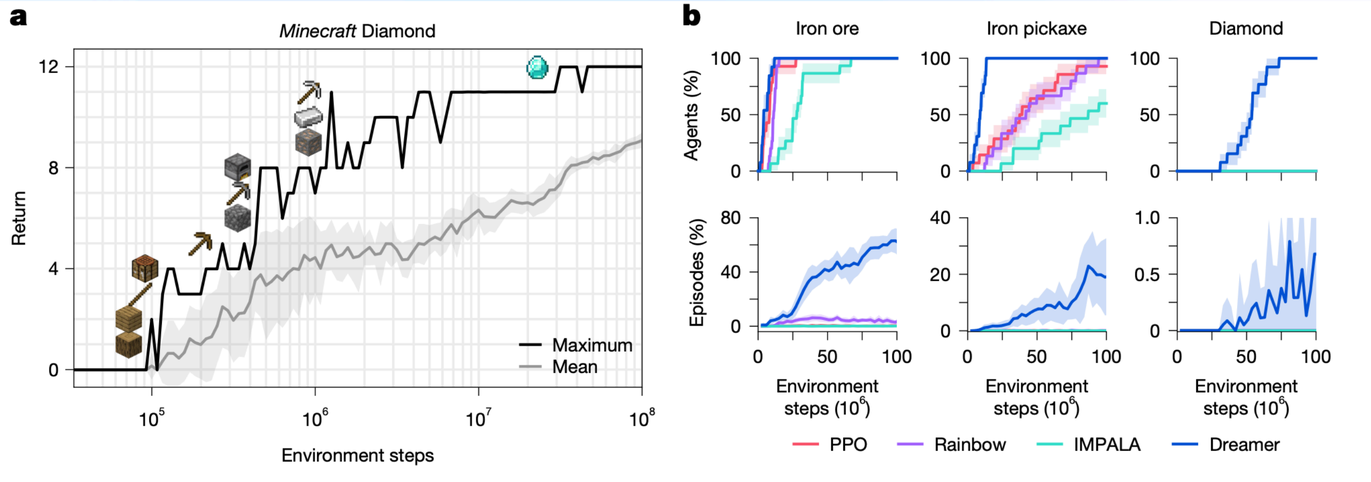
- **Sample Complexity**: Gains in test-time sample efficiency come at the cost of high training-time complexity.
- **Interpretability vs Performance**: Black-box meta-learners are hard to interpret or transfer.
- **Usefulness Depends on Task Distribution**:
  - Meta-RL may perform worse than pretraining or fine-tuning in some distributions.
  - Adaptation may not be necessary if state distributions are non-overlapping.
- **Necessity of Specialized Methods**:
  - Meta-RL reduces to a POMDP; general POMDP methods (e.g., transformers, curriculum learning) may suffice.
  - Nevertheless, specialized methods may yield more efficient learning.

Dreamer v3

general algorithm that learns to **solve tasks** across a **wide range** of applications
outperforms specialized methods across over **150 diverse tasks,**
with a **single configuration.**

model of the environment and improves its behaviour by **imagining** future scenario

Robustness techniques based on **normalization, balancing and transformations:**
- **EMA, percentile clip,**
- **Return normalization with a denominator limit**
- **symlog squared error**
- **two-hot encoding** to approximate continuous scalars



Latent Space

In tiefen Neuronalen Netzen gibt es in den mittleren Schichten oft einen '**Bottleneck**' in dem die komplizierten Eingangsdaten komprimiert als ein Vektor abgebildet.

Autoencoder!

Latent Space = mittlere Schicht Vektor(raum)

Entscheidungen finden nicht mehr auf Pixelbasis sondern auf "abstrakter Situation"
z.B. Lunar Lander per Bilderkennung => Latent Space = (Position, Winkel, Geschwindigkeit)
400*300 pixel => Vektor mit 6 Einträgen extreme Komprimierung!

Abstrakte Entscheidungen: "flieg nach oben rechts" => Aktion: gib gas 30% Situationsabhängig
=> Skills

ChatGPT = 4096 "Großmutterneuron" => abstrakt verteilt

20 Questions 2^**20**
lebend? ja
pflanze? ja
europa? ja
wohnzimmer? ja
tannenbaum ja
akinator

Dreamer v4

dreamerv3 pdf tag 14!
https://github.com/danijar/dreamerv3

Dreamer v4 = Model on steroids!
Alles per Video! (multi frame prediction in minecraft)
"angeblich komplett über Simulation"

https://arxiv.org/abs/2509.24527
Training Agents Inside of Scalable World Models

A human player counterfactually interacts with the world model in real time via mouse and keyboard to perform the same task from the same initial

single H100 GPU!

Diffusion Forcing Transformer

Shortcut Forcing

1000 60 minute episodes

Dreamer v3

Traum: Ein großes Netz was alle Probleme löst

- GPT Moment für RL!
- Noch nicht da!
- Noch kein "foundation model" zum runterladen
- RL Probleme sind sehr divers
    - (verschiedene observation state Dimensionen)

Verschmelzung mit LLM!

Reflexbildung
Erdachte Handlungen als eigener Trainer

Anwendung

Alles als Gymnasium problem? JA aber muss nicht
Hauptsache `next_state_and_reward(action)`

Schnittstelle bleibt gleich auch für Industrie.

Schnittstelle austauschen von Simulation zu realem Gerät.
Wie bei SQL: Eine Adresse ändern.

Custom Environment
Prozess Optimierung
Eigene Belohnung

Anwendung

Eigener Gymnasium wrapper? einfach *aber muss nicht*

erben von gym.Env

```
class UnsereEnvironment(gym.Env) :

 def __init__(self):
     eigener_start_zustand = []
        self.action_space = gym.spaces.Discrete(2)  # wie viele Aktionen
        self.observation_space = gym.spaces.Tuple(( # wie viele Zustände
           gym.spaces.Discrete(32),  # player score 0-31
           gym.spaces.Discrete(13)  # dealer card 1-10 (index 0-12)
        ))

  def step(self, action):
    done = False
    reward = 0
  ... < eigene Logik !
    result=(next_state, reward, done, truncated, info)
    return result
```

Vokabular

Daten D_i (gesampelte episodische Trajektorien τ).

`inner-loop` `"adaptation"`

outer-loop "meta-training"

training MDPs "**task**s/**skill**s"

Menge aller **M**DPs **M**

p(M) Auswahl probability von M "nimm einfach eine Aufgabe;)"

trial / lifetime

adaptation ≈ fine tuning einer Task

H Horizont (#episodes)

K '**shot**s' inner_steps exploitation horizon phase

meta-testing of **f** on new settings

f meta **f**unction f(D)=π spuckt vortrainierte policy aus

out-of-distribution "Ausserhalb der Verteilung" wenn p(M) beim beim Trainieren anders ist als beim Testen

POMDP partially observable Markov decision process