# Reinforcement Learning

## Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2025-05 Dozent: Karsten Flügge

# Themen des Tages

## Tag 14

**Anwendungen**

LLM + HRL

**Wiederholung**en: Monte Carlo

# Anwendung

## Anwendung von RL auf echte Probleme:

## • Selbst Umgebungsklasse erstellen!

– Ein **Gym-Wrapper** ist im Prinzip nur ein Adapter: er bringt die API in die Form **reset(), step(action) → (state, reward, done, info)**

– Ein **Roboter** oder ein Simulator hat ohnehin eine **Schnittstelle**, die Zustände liefert (**Sensorwerte, Kamera-Frames, Joint-Positionen**) und Aktionen entgegennimmt (**Motorbefehle**).

– Damit kann man **dieselben RL-Algorithmen** nutzen wie in den Übungen, **nur dass jetzt statt Pixeln oder Pendeln eben reale Motoren** angesteuert werden.

## • Trainieren wie immer!

Beispiele:

– **ROS-Umgebungen** (Robot Operating System) lassen sich mit Gym kompatibel machen; es gibt Projekte wie gym-gazebo

– **Arduino** kann man entweder einfachen algo nehmen Q-Learning https://planetachatbot.com/q-learning-con-arduino-crawling-robot-espanol/

– **Arduino** kann man großes Netz auf PC trainieren und dann drauf **kopieren**

– **Drohnen** oder manipulatorspezifische **SDKs (DJI, UR5**) können mit **wenigen Zeilen Code** in ein Gym-Interface gegossen werden.

– Auch nicht-robotische APIs lassen sich so fassen: **Börsendaten**, **Gebäudesensoren**, sogar **Web-APIs oder Computer-APIs** (Mouse!)

# Dumb exploration

**Bisher**ige Exploration war **zufällig** und **ungerichtet**:

**ε-greedy** zufällig per Definition (aber ok wegen Gesetz der großen Zahlen)

Besser:

**softmax**, Aktion je nach W-keit (nie null!)

# Pause

# Monte Carlo

- **NN für Q-Value**
- **Speicher Episode in Trajectory** als Gegensatz zu Schrittweise TD (temporal difference)
- **measured Q :  G = ∑gamma^i*r   G = r + gamma*G'    reverse vom Ziel**
- **predicted - measured Q-Q'**
- **standard Gradient descend**

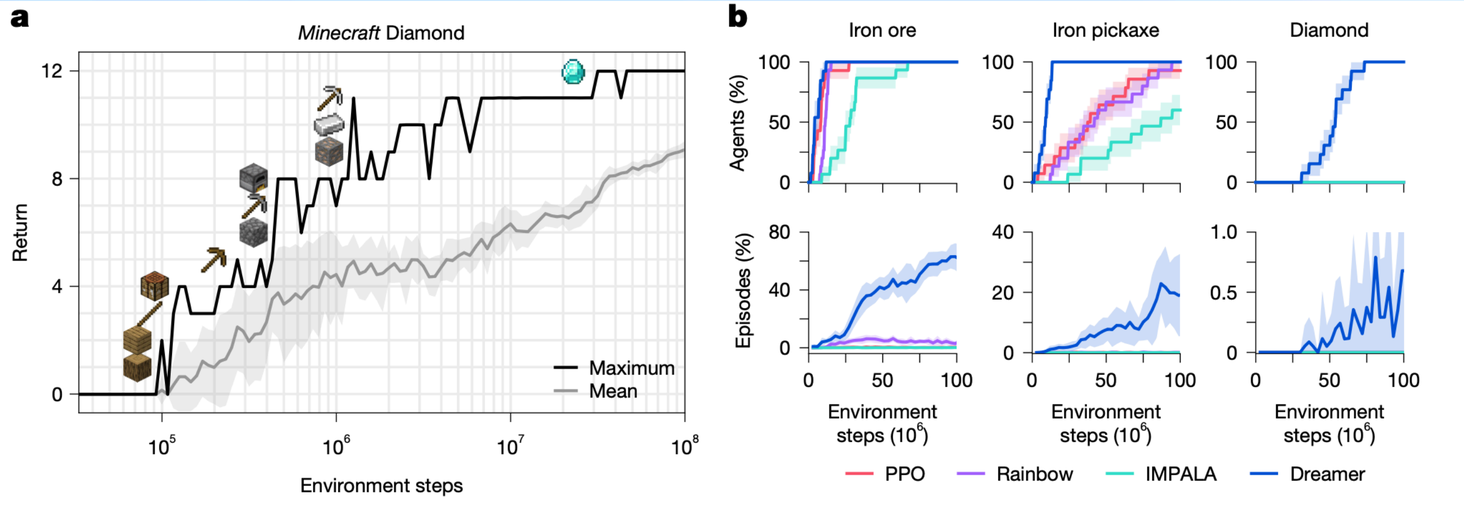
**<u>Einfacher sehr guter Algorithmus!</u>**

```
# Monte Carlo update learning
  Gain = 0
  for state, action, reward in reversed(trajectory):
    Gain = reward + gamma * Gain
    prediction = q_values(state)[action] # q-value for the action
    # Train the model
    loss = (prediction - Gain)**2 # Mean Squared Error loss
```

# 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:** no hyperparameter changes!

**First algo to get Diamonds in Minecraft**
(without any cheats / pre-training / custom-rewards ...)



# Dreamer v3

## Key components / innovations:

- **model** of the environment improves its behavior by **imagining** future scenario
- **latent space** abstrakter state vector

**Robustness** techniques based on **normalization, balancing and transformations:**

- **EMA Exponential Moving Average …**
- **max/percentile *clip* (KL / free bits?)** *todo*
- **return(gain) normalization** with a denominator limit `r=(r-mean)/std(r)`
- **symlog squared error**
- **two-hot encoding** to approximate continuous scalars

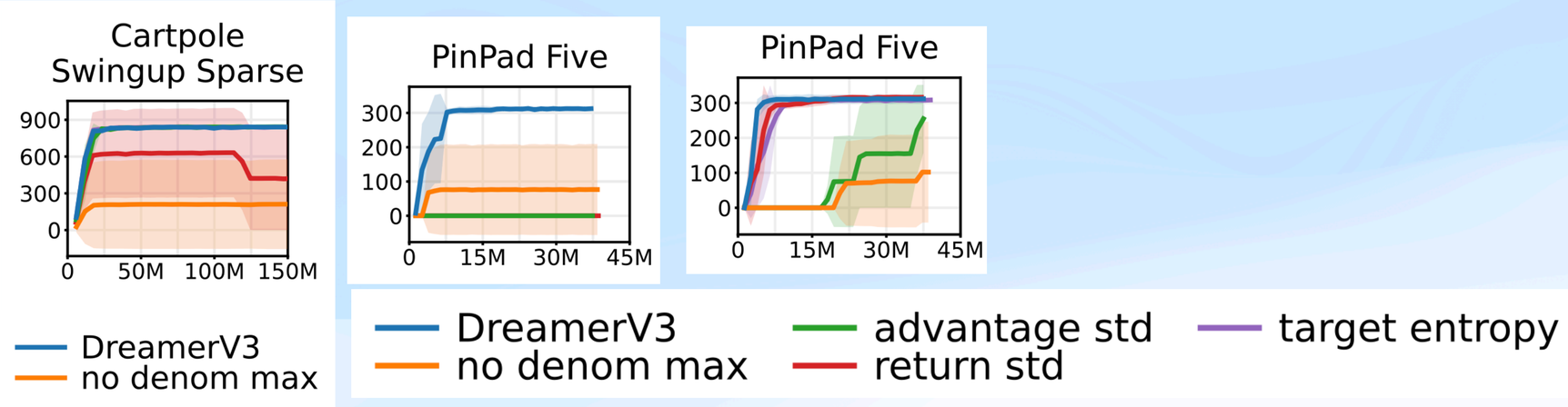
Alle dieser Regularisierungs-Techniken sind *notwendig*:
Bei Tests bei welchen eine weggelassen wurde, trainierte das Netz nicht mehr (so gut).

# latent space

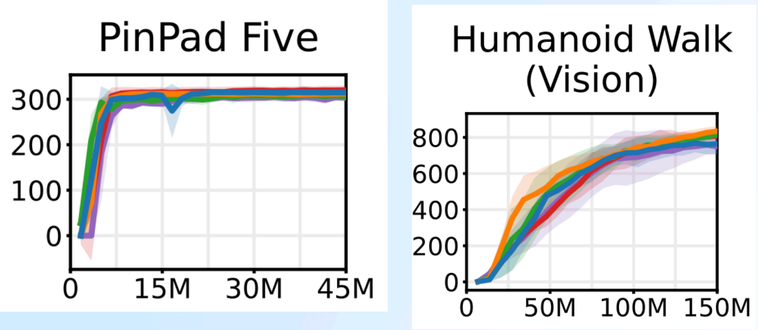
- latent space innere Schicht von tiefen Netzen
- Dimensionsreduktion 'bottleneck' Kompression der Information!
- abstrakte Representation
- ähnliche Daten werden 'in der Nähe' gespeichert
- generalisieren: neue ähnliche Objekte nah/zwischen anderen.

# Dreamer v3 ablation study

***Alle* diese Regularisierungs-Techniken sind *notwendig*:**
Bei Tests bei welchen eine weggelassen wurde (*ablation*),
trainierte das Netz nicht immer / nicht mehr (so gut).




**Es gibt aber viele Probleme, bei welchen das Weglassen von *einer* der Robustness techniques keinen großen Nachteil bringt:**



# Dreamer Komponenten

| | |
|---|---|
| Sequence model: | $h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1})$ |
| Encoder: | $z_t \sim q_\phi(z_t \mid h_t, x_t)$ |
| Dynamics predictor: | $\hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t)$ |
| Reward predictor: | $\hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t)$ |
| Continue predictor: | $\hat{c}_t \sim p_\phi(\hat{c}_t \mid h_t, z_t)$ |
| Decoder: | $\hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, z_t)$ |

**h** Hidden state **h** (**rnn** ähnlich wie meta rnn) ≈ **action a** + (nicht markovsche) **H**istorie
**f** (wie meta rnn?)
**z** latent **s**tate : encoded input state **s**  x (context dependent like in Attention)
**p** world *model* "*dynamics*"  p(z|h)=**z,r**  **p**redict **state+reward+'done'** wie wir es kennen
**c** continue/**done** flag
**x** state back from latent z  ($f^{-1}$=p?)  für autoencoder
**v** value Netz für actor critic setzt auf Weltmodell encoding auf v(z) 'head' Netz

# Dreamer

| | |
|---|---|
| Sequence model: | $h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1})$ |
| Encoder: | $z_t \sim q_\phi(z_t \mid h_t, x_t)$ |
| Dynamics predictor: | $\hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t)$ |
| Reward predictor: | $\hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t)$ |
| Continue predictor: | $\hat{c}_t \sim p_\phi(\hat{c}_t \mid h_t, z_t)$ |
| Decoder: | $\hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, z_t)$ |

$$\mathcal{L}_{\text{pred}}(\phi) \doteq -\log p_\phi(x_t \mid z_t, h_t) - \log p_\phi(r_t \mid z_t, h_t) - \log p_\phi(c_t \mid z_t, h_t)$$

$$\mathcal{L}_{\text{dyn}}(\phi) \doteq \max(1, \mathrm{KL}[\mathrm{sg}(q_\phi(z_t \mid h_t, x_t)) \| p_\phi(z_t \mid h_t)])$$

$$\mathcal{L}_{\text{rep}}(\phi) \doteq \max(1, \mathrm{KL}[q_\phi(z_t \mid h_t, x_t) \| \mathrm{sg}(p_\phi(z_t \mid h_t))])$$

$$\mathcal{L}(\phi) \doteq E_{q_\phi}\left[\sum_{t=1}^{T} (\beta_{\text{pred}}\mathcal{L}_{\text{pred}}(\phi) + \beta_{\text{dyn}}\mathcal{L}_{\text{dyn}}(\phi) + \beta_{\text{rep}}\mathcal{L}_{\text{rep}}(\phi))\right],$$

# Dreamer

$$\mathcal{L}_{\text{pred}}(\phi) \doteq -\log p_\phi(x_t|z_t,h_t) - \log p_\phi(r_t|z_t,h_t) - \log p_\phi(c_t|z_t,h_t)$$
$$\mathcal{L}_{\text{dyn}}(\phi) \doteq \max(1, \mathrm{KL}[\mathrm{sg}(q_\phi(z_t|h_t,x_t)) \| p_\phi(z_t|h_t)])$$
$$\mathcal{L}_{\text{rep}}(\phi) \doteq \max(1, \mathrm{KL}[q_\phi(z_t|h_t,x_t) \| \mathrm{sg}(p_\phi(z_t|h_t))])$$

**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:

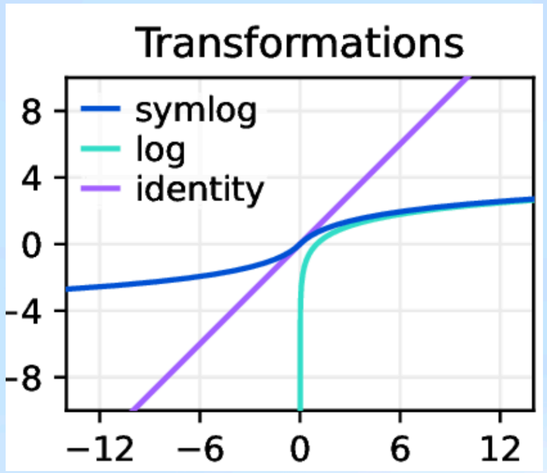
$$D_{\mathrm{KL}}(\pi_{\text{alt}} \parallel \pi_{\text{neu}}) = \sum_a \pi_{\text{alt}}(a \mid s) \log \left( \frac{\pi_{\text{alt}}(a \mid s)}{\pi_{\text{neu}}(a \mid s)} \right)$$

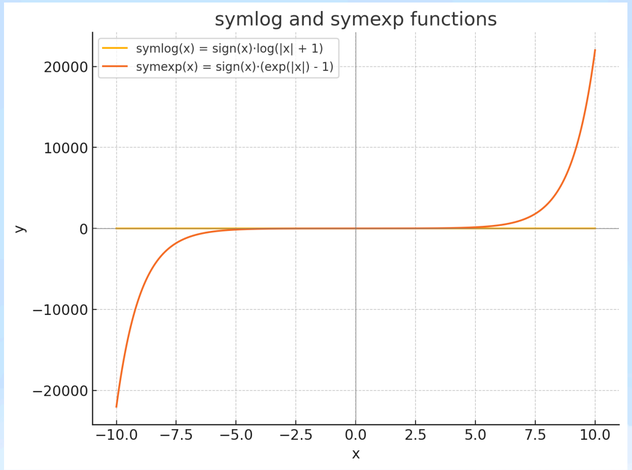
This disables them while they are already minimized well to focus learning on the prediction loss

# Dreamer

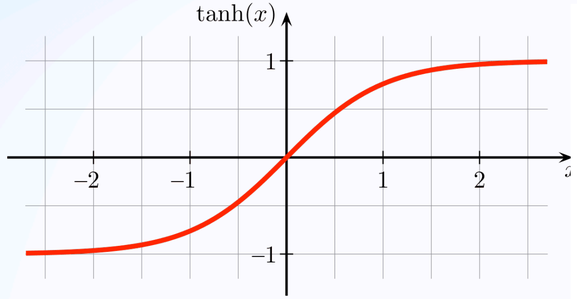
$$\operatorname{symlog}(x) \doteq \operatorname{sign}(x)\log(|x|+1)$$
$$\operatorname{symexp}(x) \doteq \operatorname{sign}(x)(\exp(|x|)-1)$$

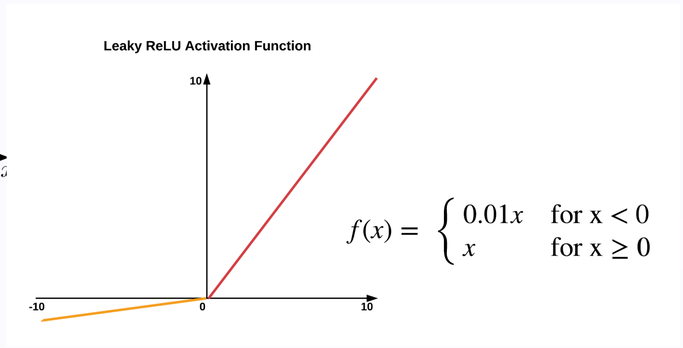
"leaky **tanh**"




**symlog**(x) = sign(x) · log(|x| + 1) **wächst langsam** für große |x| (ähnlich wie log) **symmetrisch**
**symexp**(x) = sign(x) · (exp(|x|) – 1) **wächst sehr schnell** für große |x| (ähnlich wie exp) **symmetrisch**



# Twohot encodings

Zwei **Twohot encodings:**
**1.) Zwei Einsen erlaubt**
**2.) Zwei Werte summieren sich zu 1!**
**Twohot** erlaubt darstellen von **kontinuierlichen Aktionen,** indem man zwischen zwei outputs interpoliert
Wahrscheinlichkeiten nicht-binär (aber **summierend zu 1**)

Die zwei Ausschlag gebenden Aktivierungen werden behalten

$$\text{twohot}(x)_i \doteq \begin{cases} |b_{k+1} - x| / |b_{k+1} - b_k| & \text{if } i = k \\ |b_k - x| / |b_{k+1} - b_k| & \text{if } i = k+1 \\ 0 & \text{else} \end{cases} \qquad k \doteq \sum_{j=1}^{|B|} \delta(b_j < x)$$

Aktivierung = [ forward, backward, **left**, right ]  # diskret : 1

Aktivierung = [ forward, backward, left, right ]  # continuous **2 hot extra/interpolieren**

Aktivierung = [ forward, sideward ]  # continuous classic

[f b l r ] **Aktivierung** = **Mittelung** aus zwei größten Softmax Werten

[f b l r ] = [ 0 0 **.8 .2**]  = fahre **überwiegend** nach **links**  (.8+.2=1)

v = binominial(f,b)

f=.5 b=.5  v = 0

f=1 b=0    v = 1

f=.8 b=.2    v = .6

# Dreamer

sg "stop gradient"

In PyTorch: `x.detach()`

In TensorFlow: `tf.stop_gradient(x)`

In JAX: `jax.lax.stop_gradient(x)`

Kontrollierter Gradienten Flow.

$$\mathcal{L}_{\text{pred}}(\phi) \doteq -\log p_\phi(x_t|z_t,h_t) - \log p_\phi(r_t|z_t,h_t) - \log p_\phi(c_t|z_t,h_t)$$
$$\mathcal{L}_{\text{dyn}}(\phi) \doteq \max(1, \text{KL}[\text{sg}(q_\phi(z_t|h_t,x_t)) \| p_\phi(z_t|h_t)])$$
$$\mathcal{L}_{\text{rep}}(\phi) \doteq \max(1, \text{KL}[q_\phi(z_t|h_t,x_t) \| \text{sg}(p_\phi(z_t|h_t))])$$

pytorch: with **torch.no_grad**():

Heisst: Dieser teil soll nicht in Gradienten mitoptimiert werden.

# Dreamer

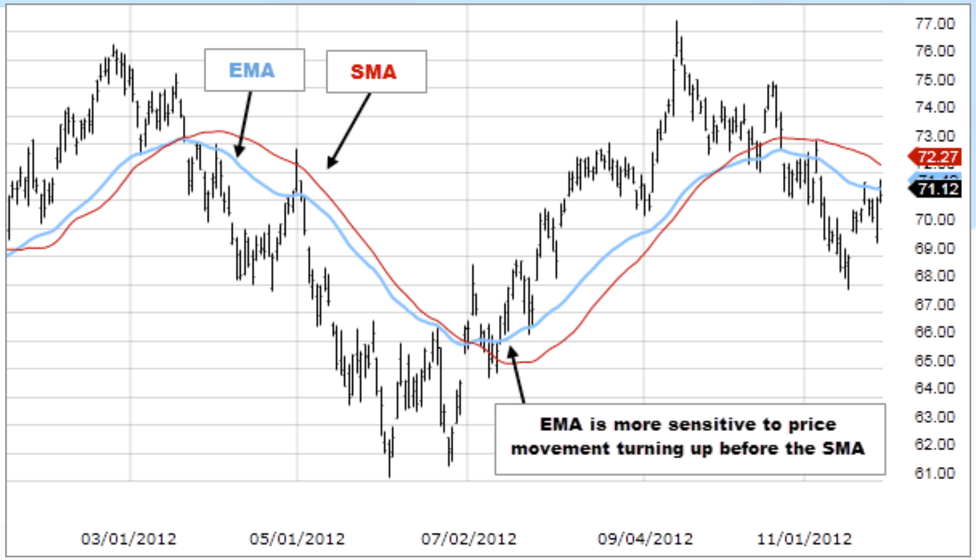
**Regularisierung:**

**EMA Exponential Moving Average**

- Maintain a **target actor** with EMA updates during training

$$\theta_{\text{EMA}} \leftarrow \tau \cdot \theta_{\text{EMA}} + (1 - \tau) \cdot \theta$$

with $\tau \in [0.99, 0.999]$




**Allgemeine Technik**, nicht nur bei RL / ML

<> Elastic Net?

# Dreamer v3

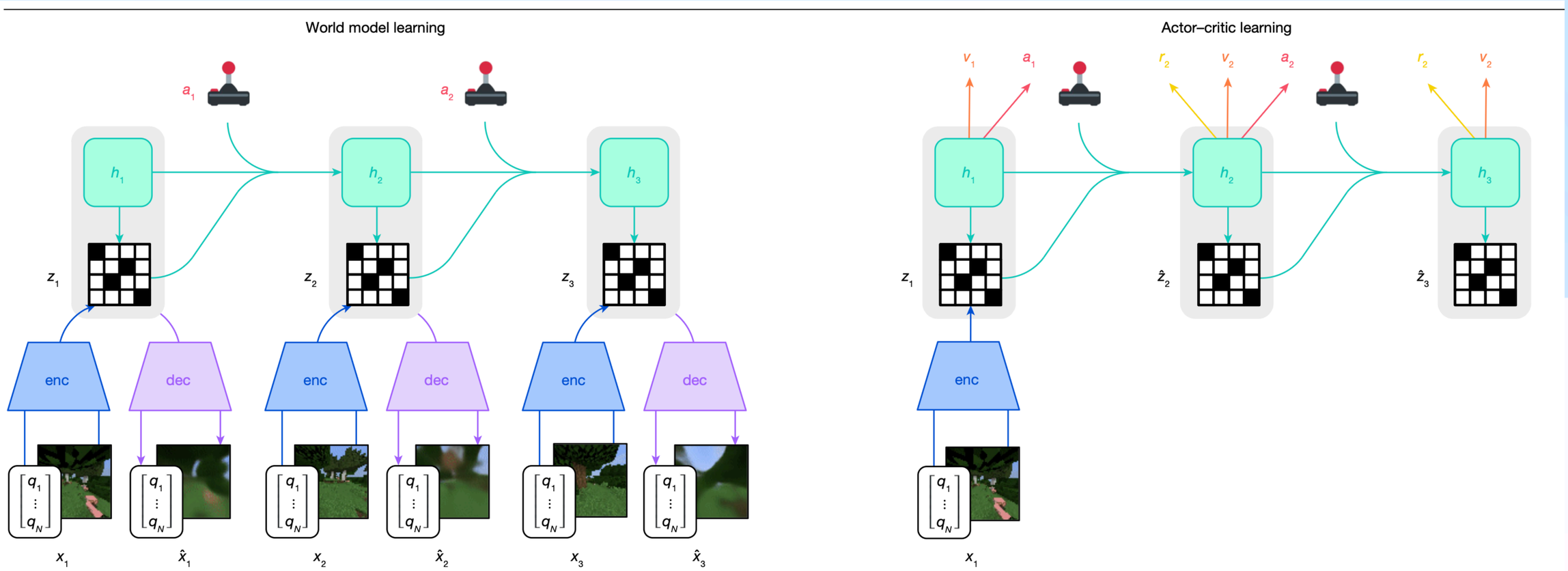



**Fig. 1 | Training process of Dreamer.** The world model encodes sensory inputs $x_t$ using the encoder (enc) into discrete representations $z_t$ that are predicted by a sequence model with recurrent state $h_t$ given actions $a_t$. The inputs are reconstructed as $\hat{x}_t$ using the decoder (dec) to shape the representations. The actor and critic predict actions $a_t$ and values $v_t$ and learn from trajectories of abstract representations $\hat{z}_t$ and rewards $r_t$ predicted by the world model.

# Baselines

**PPO**
**IMPALA** and Rainbow for Minecraft MineRL v0.4.4 abstract crafting actions (cheat?)

To investigate whether Dreamer can scale robustly, we train **6 model sizes** ranging from **12 million to 400 million** parameters, as well as different replay ratios.

Trainable on **one GPU** in **10 days.**

# Dreamer V3

https://github.com/danijar/dreamerv3

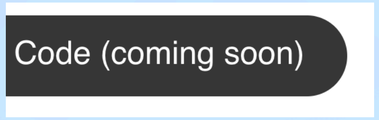
https://arxiv.org/pdf/2301.04104v1.pdf

# Pause

**Reinforcement Learning with Foundation Priors:**
**Let the Embodied Agent Efficiently Learn on Its Own**

**https://arxiv.org/abs/2310.02635**

**https://yewr.github.io/rlfp/**



# Dumb exploration

**Bisher**ige Exploration war **zufällig** und **ungerichtet**:

**ε-greedy** zufällig per Definition (aber ok wegen Gesetz der großen Zahlen)

Besser:

**softmax**, Aktion je nach W-keit (nie null!)

# Profi exploration

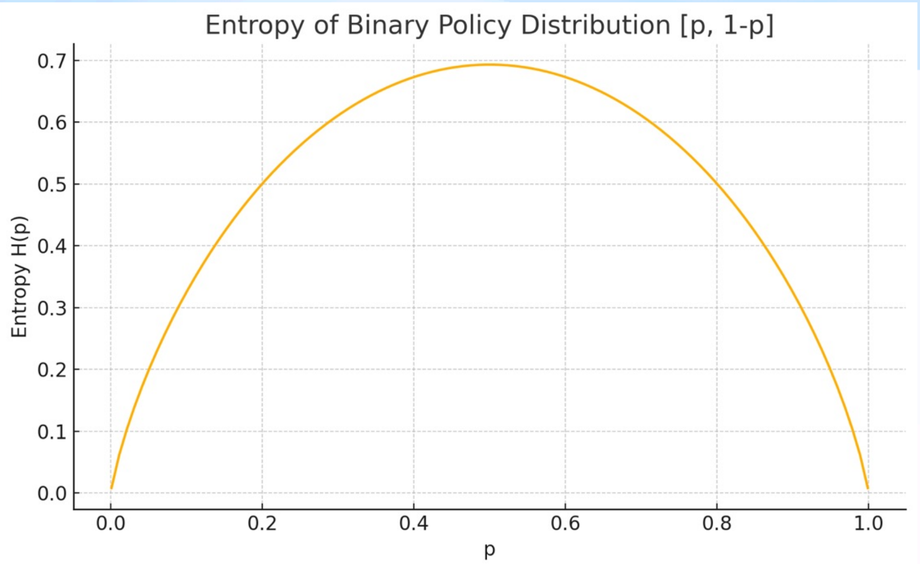
## Entropie:

$$\mathcal{L}_{\text{actor}} = \mathbb{E}[A(s,a) + \alpha \cdot \mathcal{H}(\pi(\cdot|s))]$$

- $\mathcal{H}(\pi)$: Entropy of the policy, i.e. $-\sum \pi(a|s) \log \pi(a|s)$
- Promotes **action diversity** (randomness), especially early in training

Does **not modify rewards**, but affects the gradient

- **No environment signal** involved
- **No model** involved
- **No reward** shaping




**Entropie: Maß für Chaos / Unsicherheit / Diversität**

**Kreuzentropie: -∑p*log(q) Maß für Treffsicherheit / Informationsverlust** (bei Softmax)

# 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:

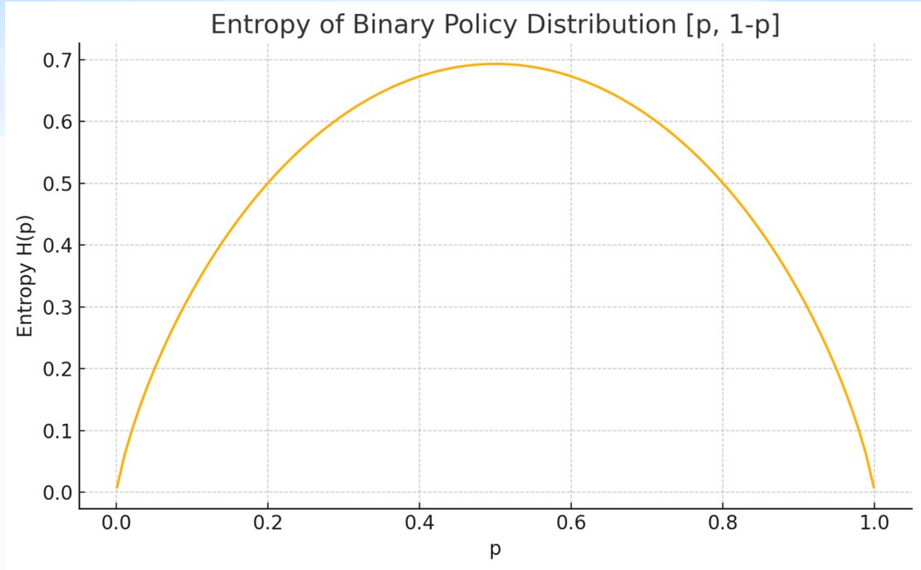
**Entropie (mittlere Shannon-Information):**

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**

# Profi exploration

$$\mathcal{L}_{\text{actor}} = \mathbb{E}[A(s,a) + \alpha \cdot \mathcal{H}(\pi(\cdot|s))]$$

- $\mathcal{H}(\pi)$: Entropy of the policy, i.e. $-\sum \pi(a|s) \log \pi(a|s)$
- Promotes **action diversity** (randomness), especially early in training

**Automatische Entropie:**

Adaptives α: Viele moderne Verfahren (z. B. Soft Actor-Critic) lernen α automatisch durch eine zusätzliche Optimierung:

Dadurch wird α **erhöht**, wenn die Policy zu **deterministisch** wird, und **gesenkt**, wenn **zu viel Entropie** da ist.

# Novelty, surprise: curiosity exploration

- **Novelty** (e.g., **count-based** or hash-based methods)
- Curiosity via **surprise** (e.g., **prediction error** of *models*) <>
- Curiosity via **uncertainty** (e.g., Bayesian uncertainty or **ensemble disagreement**)

Bekannte Loss-Terme, die systematischer Exploration induzieren:

• **Exploration Bonus (Count-based / Pseudo-count):**
Zusätzlicher Reward $r' = r + \beta/\sqrt{N(s,a)}$. Das geht direkt in den Policy-Loss ein, ohne an Entropie gebunden zu sein. Implementiert als "intrinsic reward", nicht als Entropie.

• **KL-Regularisierung gegen eine Prior-Policy:**
Statt Entropie $-\sum \pi \log \pi$, minimiert man $\mathrm{KL}(\pi(\cdot|s)\|\pi_0(\cdot))$ gegen eine gewünschte Explorationsprior (z.B. uniforme Policy).
Loss-Term:
$L_{\mathrm{KL}} = \alpha \mathrm{KL}(\pi_\theta(\cdot|s) \,\|\, U(\cdot))$

→ zwingt systematisch nahe an Uniformität, bis genügend Daten vorliegen.

• **Information Gain (VIME, variational intrinsic motivation):**
Reward proportional zur Veränderung der Belief-Verteilung über das Modell durch eine Aktion. Loss-Terme enthalten dann KL zwischen posterior und prior Model-Parameterverteilungen.

• **Divergence Measures statt Entropie:**
Tsallis- oder Rényi-Entropie-Regularizer. Höhere Sensitivität gegen zu kleine Wahrscheinlichkeiten, kann systematischer unentdeckte Actions hochziehen.

# Novelty, surprise: curiosity exploration

KL  hält ja Abstände klein, wie kann es gleichzeitig Diversität erhöhen?

Das scheinbare Paradox löst sich, wenn man genau hinsieht, *welche* Referenzverteilung in der KL steht:

- **Trust Region (PPO/TRPO)**

$D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{old}})$

→ KL klein halten ⇒ Policy-Update bleibt *nah* an der alten Policy. Das sichert Stabilität, bremst aber Diversität.

- **Exploration-Regularisierung**

$D_{\mathrm{KL}}(\pi_\theta(\cdot|s) \,\|\, U(\cdot))$

wobei $U$ die uniforme Verteilung ist.

→ KL klein halten ⇒ Policy *muss* nah an Uniformität sein ⇒ jede Aktion erhält positive Wahrscheinlichkeit ⇒ Diversität ↑.

- **Information Gain (VIME u.ä.)**

$\mathbb{E}\left[D_{\mathrm{KL}}(p(\theta|D, s') \,\|\, p(\theta|D))\right]$

→ Maximierung der KL-Differenz zwischen Posterior und Prior = Aktionen suchen, die das Modell maximal verändern ⇒ führt zu Explorationsverhalten.

**Stabilität versus Diversität!**

# Novelty, surprise: curiosity exploration

## KL-Divergenz

**KL(p,p') = Kreuzentropie(p,p') - Entropie(p)**

**KL(p,p') = -∑p·log(p') + ∑p·log(p)  = ∑p· log(p)/log(p') =  ∑p · log_ratio(p,p')**

$D_{\mathrm{KL}}(p \parallel q) = H(p,q) - H(p)$ *Extra-Unsicherheit*, die nur entsteht, weil Modell **p'** nicht zur Wirklichkeit **p** passt.

**KL-Divergenz** misst, wie „**anders**“ die neue Policy **π′** im Vergleich zur alten Policy **π** ist.

KL Target ist ein Sollwert:

Wenn die gemessene KL kleiner als der Zielwert ist → **die Lernrate kann erhöht werden**

(größere Updates → schnellere Anpassung).

Wenn die gemessene KL größer ist → **die Lernrate wird reduziert**

(kleinere Updates → stabiler, weniger riskant).

Das Ziel: nicht zu große Policy-Sprünge (verhindert „Policy Collapse“),

aber auch nicht so kleine Updates, dass das Training ewig dauert.

# Novelty, surprise: curiosity exploration

- **Novelty** (e.g., **count-based** or hash-based methods)

  **zähle** wie oft ein State erreicht wurde!

  diskret oder über continuous neighborhood **estimate, distance**

- Curiosity via **surprise** (e.g., **prediction error** of forward models) <>
- Curiosity via **uncertainty** (e.g., Bayesian uncertainty or **ensemble disagreement**)

sigmoid peeks vs near uniform

Steuerung zum Beispiel in **policy direkt** oder über **reward**

$$r_t^{\text{aug}} = r_t^{\text{env}} + \beta \cdot r_t^{\text{explore}}$$

<> **Advantage** orthogonal objectives:

*advantage* optimizes **known rewards**,

*exploration* bonuses **bias** toward acquiring new knowledge

# Novelty, surprise: curiosity exploration

- Curiosity via **surprise** (e.g., **prediction error** of forward models) <>

Steuerung zum Beispiel in **policy direkt** oder über **reward** $r_t^{\text{aug}} = r_t^{\text{env}} + \beta \cdot r_t^{\text{explore}}$

Führt so eine Belohnung dazu dass absichtlich falsche Vorhersagen getätigt werden??

# Novelty, surprise: curiosity exploration

• Curiosity via **surprise** (e.g., **prediction error** of forward models) <>

Steuerung zum Beispiel in **policy direkt** oder über **reward**

Führt so eine Belohnung dazu dass absichtlich falsche Vorhersagen getätigt werden?

**JA! Problematisch:**

• **Exploration wird Selbstzweck**

→ Agent sucht maximales Unwissen statt sinnvolle Information

• **Loophole bei reward shaping**

→ Agent trickst das Lernsystem aus, um hohe intrinsische Belohnung zu erhalten

• **Non-stationäres Ziel**

→ Belohnung basiert auf einem sich ständig verändernden Modell

**Gegenmaßnahmen:**

• **Im Latent space arbeiten**

• **Bonus nur temporär** oder nur in **frühen Phasen**

• **Decay des Intrinsic-Reward-Gewichts**

• **Maskierung chaotischer Zustände** (z. B. durch epistemische Unterscheidung)

• **Use surprise only when learning is possible** (→ predictability prior)

Belohne nur Zustände mit **hoher epistemischer Unsicherheit** (nicht bloß hoher Fehler)

**Bild 8.5** Das Problem des verrauschten Fernsehers ist ein theoretisches und praktisches Problem, bei dem ein Reinforcement-Learning-Agent mit einem einfachen Sinn für Neugierde von einem verrauschten Fernseher in den Bann gezogen wird und ihn ewig anstarrt. Das liegt daran, dass er von Natur aus durch Unvorhersehbarkeit belohnt wird, und weißes Rauschen sehr unberechenbar ist.

# Novelty, curiosity, surprise: Exploration

- **Novelty** (e.g., **count-based** or hash-based methods)
- **Curiosity** (e.g., **prediction error** of forward models) <>
- **Surprise** (e.g., Bayesian uncertainty or **ensemble disagreement**)

NICHT **explizit** in Dreamer! Aber **implizit:**

Dreamer relies purely on:

1. **Stochastic world models** (e.g., RSSM)

→ Intrinsic variability in latent imagination leads to **structured exploration**.

2. **Entropy regularization** in the actor loss:

$$\mathbb{E}_{a\sim\pi}[\log \pi(a|s)]$$

→ Encourages **policy entropy**, indirectly promoting exploration.

Dreamer avoids these by **not adding any intrinsic reward**

- Leverages **learned latent dynamics** for **long-horizon planning**

# Novelty, curiosity, surprise: Exploration

**Manuell alle Aktionen ausprobieren**

```
# funktioniert bestens, wenn Zustandsraum klein und diskret ist
def policy_try_new(x,y):
  min = 2**31
  best_action = 0
  tries = 0
  for action in ACTION_SYMBOLS.keys():
    tries = actions_tried[x, y, action]
    if tries < min:
      min = tries
      best_action = action
  return best_action

# Entropie macht so etwas ähnliches
```

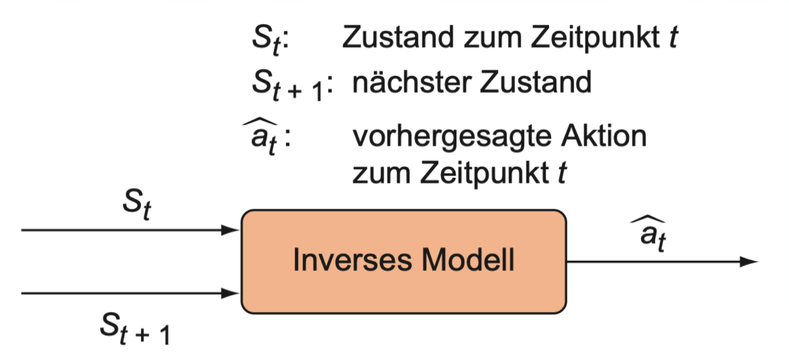
# „Curiosity-driven Exploration by Self-supervised Prediction“ (2017)

einer der am einfachsten zu implementierenden Algorithmen

Zusätzlich zur Dynamik, also zum Modell

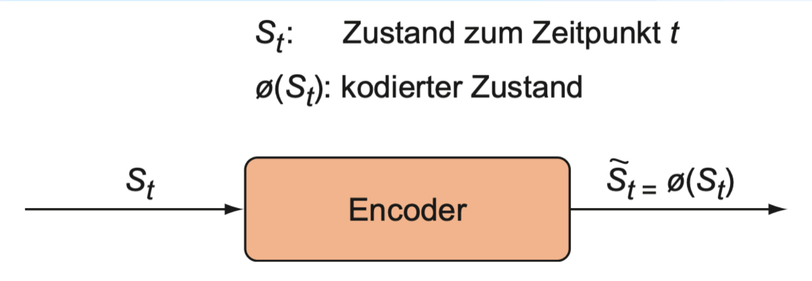
**p(s,a)=>s'** bauen wir die Rückrichtung

**f(s,s')=>a** welche versucht vorherzusagen mithilfe von **welcher Aktion zum nächsten Zustand** gefunden wurde.




Wir wollen uns nur auf '**wichtige Aspekte**' des Zustandes konzentrieren, daher

Encoder s=>**z**



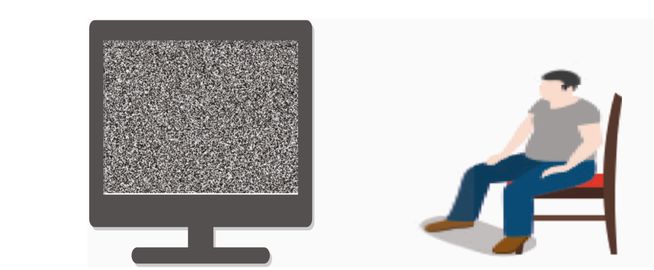



**Bild 8.5** Das Problem des verrauschten Fernsehers ist ein theoretisches und praktisches Problem, bei dem ein Reinforcement-Learning-Agent mit einem einfachen Sinn für Neugierde von einem verrauschten Fernseher in den Bann gezogen wird und ihn ewig anstarrt. Das liegt daran, dass er von Natur aus durch Unvorhersehbarkeit belohnt wird, und weißes Rauschen sehr unberechenbar ist.

# „Curiosity-driven Exploration by Self-supervised Prediction“ (2017)

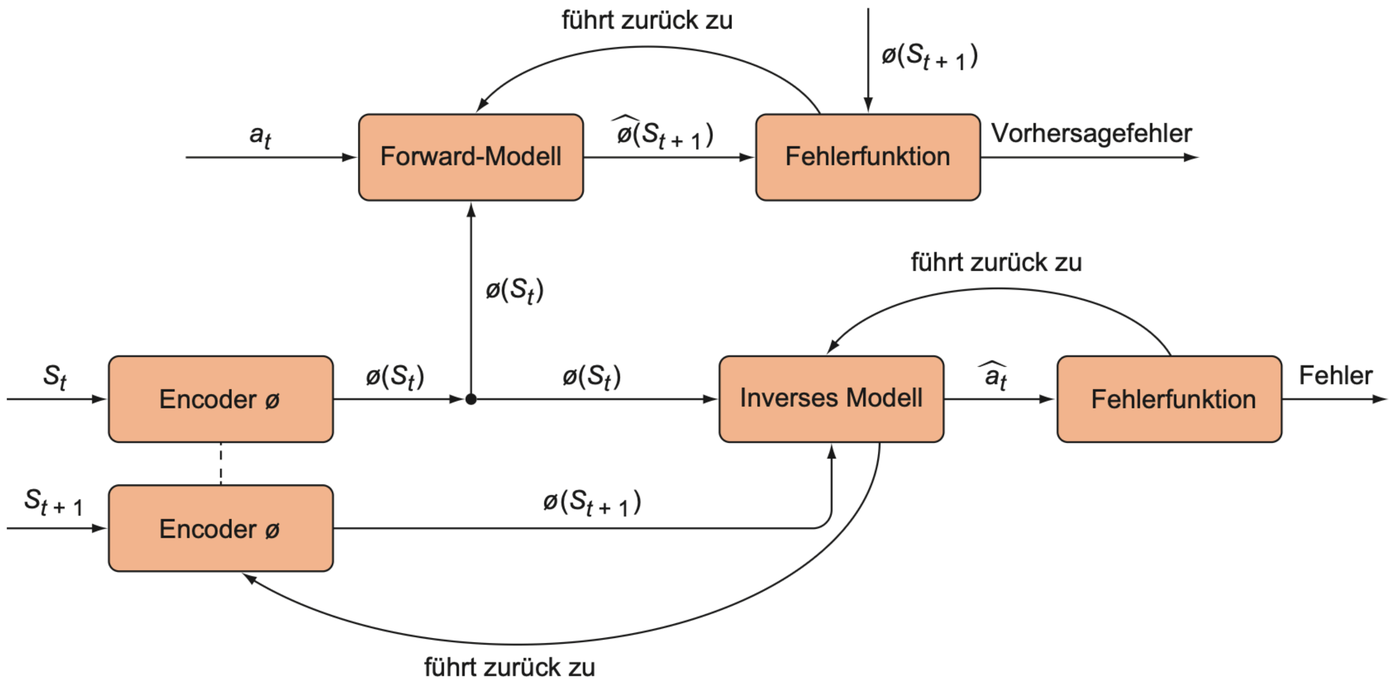



**Bild 8.10** Das Neugierde-Modul. Zuerst kodiert der Encoder die Zustände $S_t$ und $S_{t+1}$ in niedrigdimensionale Vektoren, $\phi(S_t)$ bzw. $\phi(S_{t+1})$. Diese kodierten Zustände werden an das Forward- und das inverse Modell weitergegeben. Beachten Sie, dass das inverse Modell auf das Encoder-Modell zurückgeführt wird und es somit durch seinen eigenen Fehler trainiert. Das Forward-Modell wird durch Backpropagieren aus seiner eigenen Fehlerfunktion trainiert, aber es wird nicht wie das inverse Modell zum Encoder zurückgeführt. Dadurch wird sichergestellt, dass der Encoder lernt, Zustandsdarstellungen zu erzeugen, die nur für die Vorhersage, welche Aktion durchgeführt wurde, nützlich sind. Der schwarze Kreis zeigt einen Kopiervorgang an, bei dem die Ausgabe vom Encoder kopiert und die Kopien an das Forward- und das inverse Modell weitergegeben werden.

# Pause

Later work (e.g., **Plan2Explore**, **DreamerV3 with pretraining**) does explore **intrinsic reward extensions** — but not Dreamer V1/V2 in their core forms.

**Contrasted with methods that do use explicit signals:**

| Method | Exploration Signal |
|---|---|
| ICM (Curiosity, 2017) | Prediction error of forward model |
| RND (Burda et al., 2018) | Disagreement with fixed random net |
| Count-based (MBIE-EB) | State visitation counts |
| Pseudo-counts | Density models (PixelCNN, etc.) |
| Disagreement (e.g. PETS) | Variance in model predictions |

Dreamer avoids these by **not adding any intrinsic reward**:

$$r_t^{\text{total}} = r_t^{\text{env}} \quad \text{(no exploration bonus)}$$

# Pause

# LLM+RL

**LLM+RL**

**RLHF RL with human feedback OpenAI finetuning**

**LLM liest sich Internet durch => Nazi**

**Supervised Fine-Tuning (SFT)**

**DeepSeek R1 'reasoning Modelle' nach GPT-4**

**LLM** liest sich Internet durch =>

Mathe Aufgaben: <think>…</think><solution>X</solution> extended chain-of-thought

**reward** bei richtiger Lösung, Lösungsweg frei!

Group Relative Policy Optimization (GRPO) Vereinfachung von PPO

https://www.datacamp.com/blog/what-is-grpo-group-relative-policy-optimization

https://verl.readthedocs.io/en/latest/algo/grpo.html

GRPO is defined as follows:

$$\max_{\pi} \mathbb{E}_{y\sim\pi_k(\cdot|x)} \min\left(\frac{\pi(y|x)}{\pi_k(y|x)} A_{\pi_k}(x,y),\ \text{clip}\left(\frac{\pi(y|x)}{\pi_k(y|x)}, 1-\epsilon, 1+\epsilon\right) A_{\pi_k}(x,y)\right) - \beta \text{KL}(\pi||\pi_{\text{ref}}),$$

**Liefert das RL hier wirklich Mehrwert? Wird aktuell diskutiert/debattiert/gestritten.**

**Hierarchical Reasoning Model**

https://arxiv.org/pdf/2506.21734 mit LLM kombinierbar? Für RL Probleme einsetzbar? Sudoku ≈ Pole?? Maze-Hard (30x30) ≈ Frozen lake?

**Attention Blocks**! mehr RL? momentan nur zum Wechsel zwischen fast/slow net.

**self reflecting**

# Übersicht

**Unsere Algorithmen und Formeln und Loss**

**SARSA** basic loop

**TD Temporal Difference** (one step)

**Deep Q-Learning  DQL / DQN**

**Policy Gradient-Methoden (PG)**  REINFORCE

**Actor-Critic-Methoden (AC)**

**Monte Carlo (MC)** (whole episode)

**PPO**

# Übersicht

**Unsere Algorithmen und Formeln und Loss**

| Methode | Typ | Update-Regel / Loss | Bemerkung |
|---|---|---|---|
| SARSA | On-Policy **TD** | **Q(s,a)** ← Q(s,a) + α [r + γ Q(s′,a′) – Q(s,a)] | Focus auf **Loop** |
| Temporal Difference TD(0) | On-Policy **TD** | V(s) ← V(s) + α [r + **γ V(s′) – V(s)**] | ein **Step**, V / Q |
| Monte Carlo (MC) | episodisch | V(s) ← V(s) + α [$\mathbf{G}_t$ – V(s)]  $G_t = \sum_k \gamma^k r_{t+k}$ | **G**anze **Episode**, V / Q |
| DQN / DQL | Off-Policy TD + NN | Loss L = [r + **γ max$_a$′ $Q_{t-}$(s′,a′) – $Q_t$(s,a)**]² | **PG** Loss via Adam |
| REINFORCE | On-Policy PG | Loss L = ∑ log π(a \| s) · $\mathbf{G}_t$ | $G_t$ kann durch A / δ ersetzt werden: |
| Actor-Critic (AC) | Hybrid PG + T**D** | Actor Loss L = log π(a \| s) · **δ**;  δ = r + γ V(s′) – V(s)  Critic L= δ | auch mit MC! |
| PPO | On-Policy PG | L = E[min($r_t\hat{A}_t$, **clip**($r_t$, 1–ε, 1+ε)$\hat{A}_t$)]  $r_t$ = π(a\|s)/$\pi_{\text{ı}}$(a\|s)  ratio | |