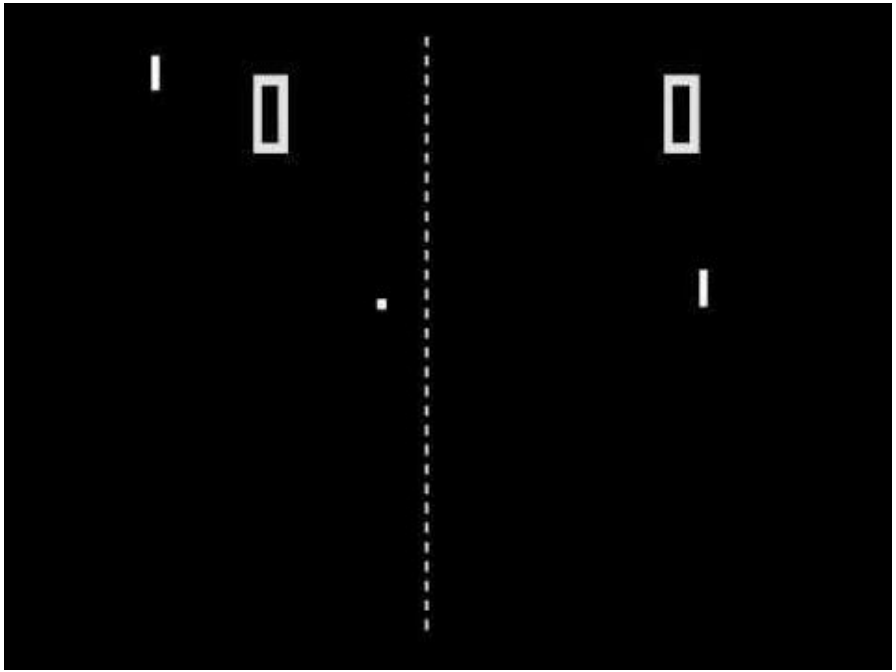
Reinforcement Learning: Advanced Model-Based RL (ESE 3990)

Instructor:
Roberto Ligeralde

Fall 2026
University of Pennsylvania

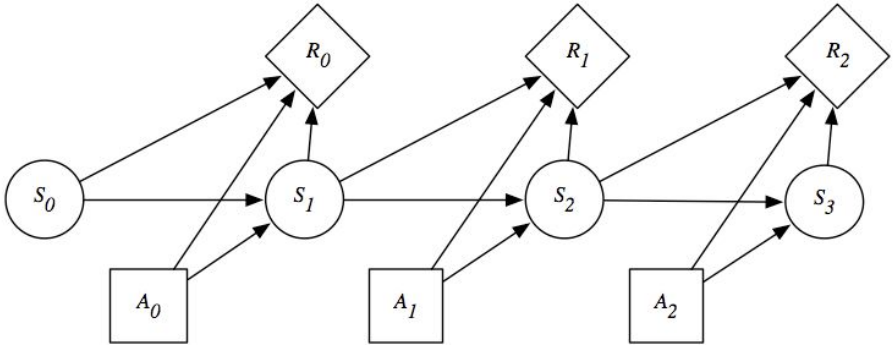
Recap: Observation Space Modeling

- Where should a policy controlling the left paddle move next?
- Single image observation tells us nothing about ball's current velocity
- We must design learning algorithms which can cope with incomplete information

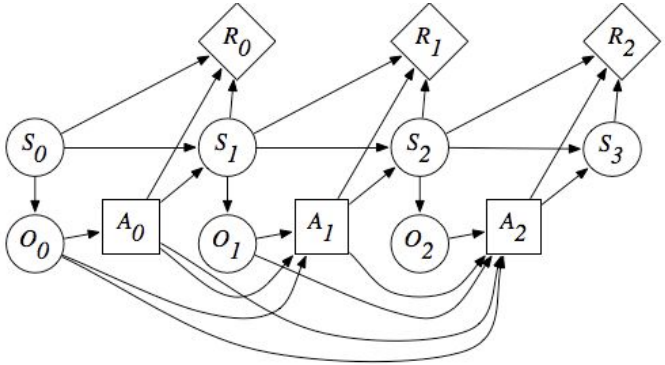


Partial Observability: Definition

Markov Decision Process



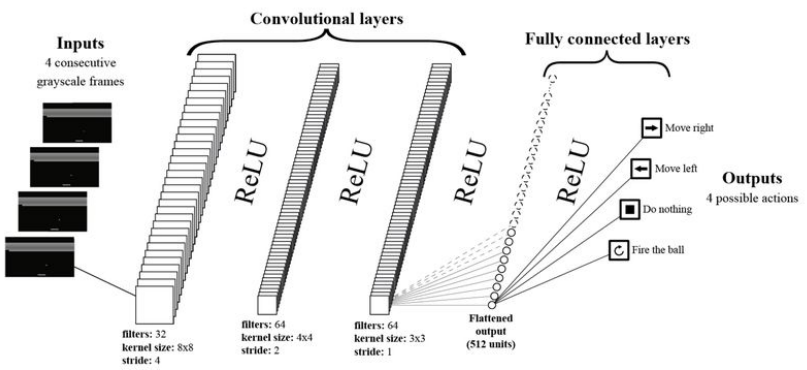
Partially Observable MDP

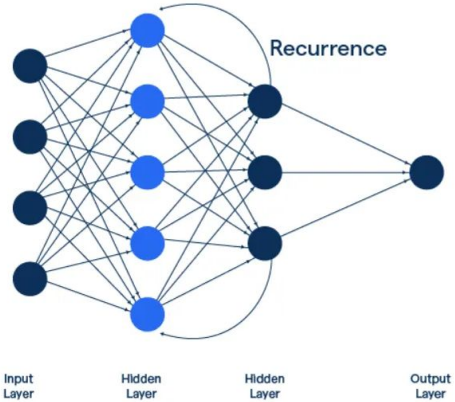


"Artificial Intelligence: Foundations of Computational Agents" (Cambridge University Press, 2023)

- **Partially Observable Markov Decision Process (PO-MDP)**: policy chooses actions from an observation o_t which is a function O of the full state s_t
- Very often, O does not preserve the Markov Property!

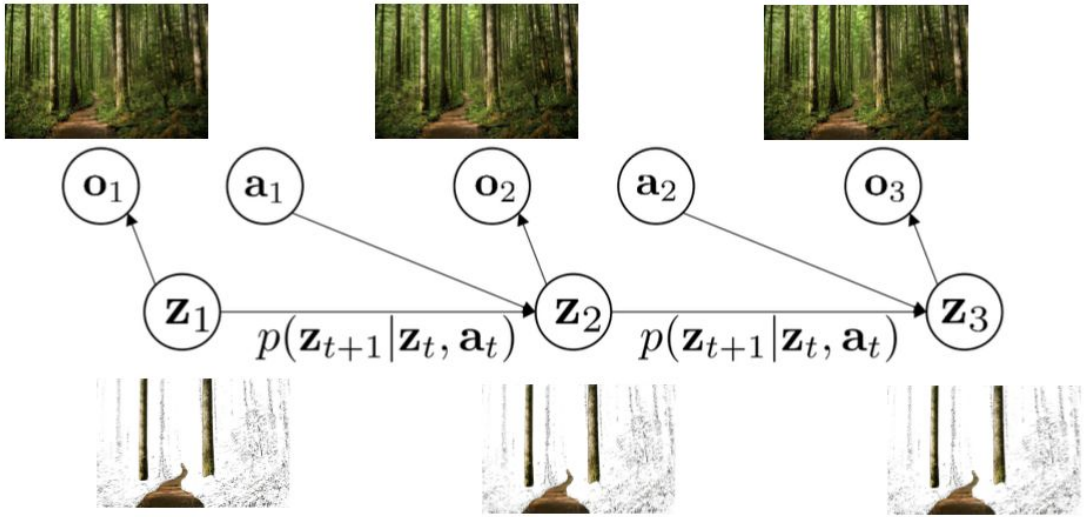
Partial Observability: Partial Solutions





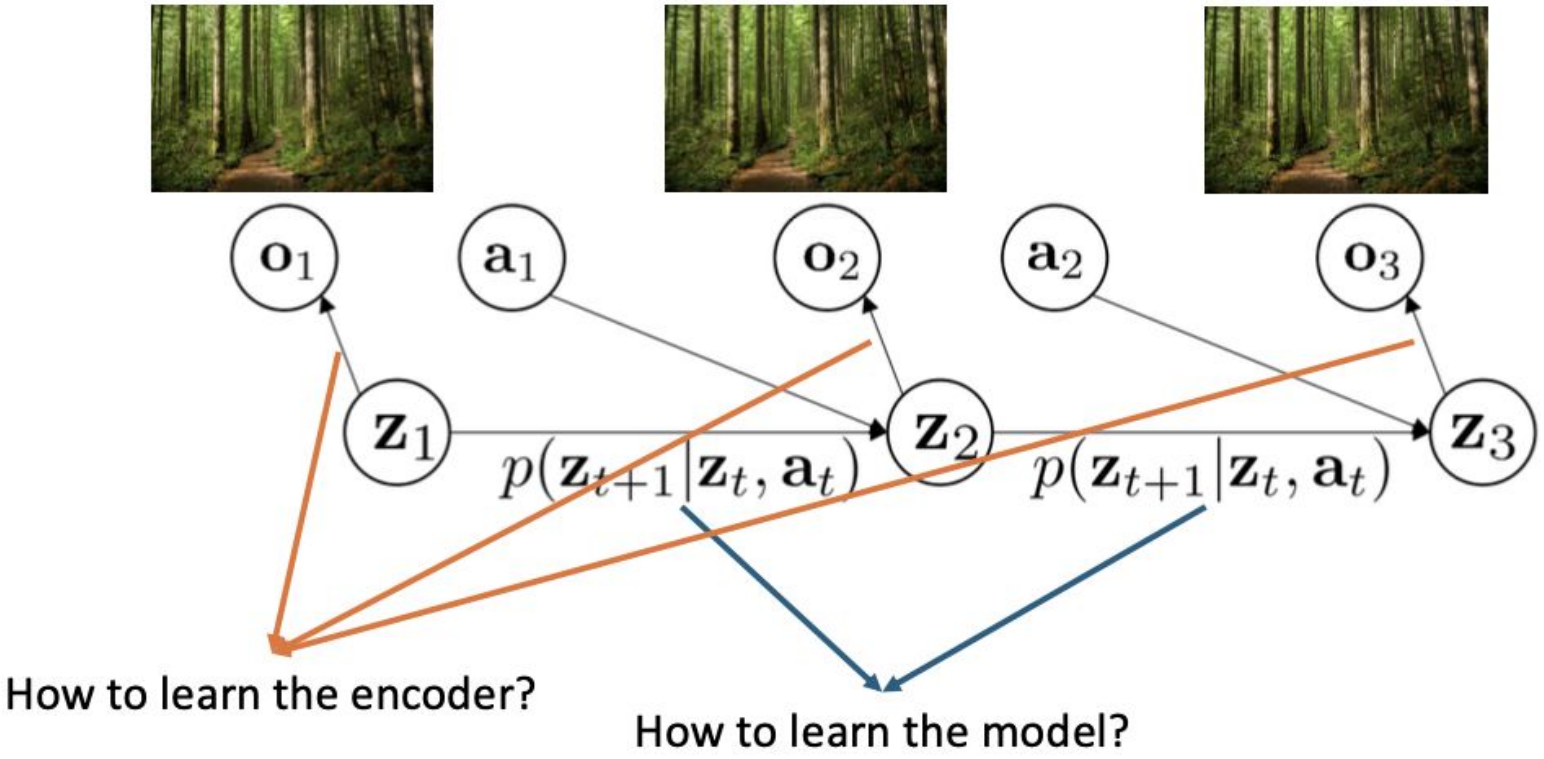
- Option 1: frame stacking (*K* previous observations)
  - Goldfish memory, can't track long-term state
- Option 2: use recurrent networks (GRUs, LSTMs, etc.)
  - Awkward to fit into standard RL training loops, few off-the-shelf implementations
- **Option 3: learn better representations than what the PO-MDP is offering us**

Latent Space Models [1]: Overview



- Idea: learn representations z_t of observations o_t which capture most valuable data
- Sufficiently good z_t can act as a surrogate for s_t in policy learning

Latent Space Models [2]: Components (1)



Latent Space Models [2]: Components (2)

- An encoder with parameters ϕ

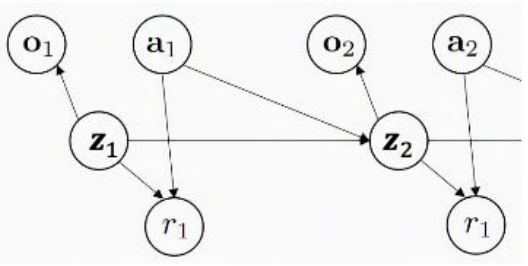
$$z_t = g_\phi(o_t)$$

- (Sometimes) a decoder with parameters δ

$$o_t \approx d_\delta(z_t)$$

- A model with parameters ψ
  - Forward predictor: $z_{t+1} = f_\psi(z_t, a_t)$
  - Reward estimator: $r_t \approx r_\psi(z_t)$

Latent Space Models [3]: Objective Function



- A loss to reconstruct the reward:

$$L_{\mathrm{r}}(\phi,\psi) = \left(r_t - r_\psi\left(g_\phi(o_t)\right)\right)^2$$

- A loss to be consistent between time steps:

$$\mathrm{L_d}(\phi,\psi) = \left(g_\phi(o_{t+1}) - f_\psi\left(g_\phi(o_t), a_t\right)\right)^2$$

- (sometimes) A loss to reconstruct the observation (through a decoder d_δ)

$$\mathrm{L_o}(\phi,\delta) = \left(o_t - d_\delta(g_\phi(o_t))\right)^2$$

- *They depend on each other*, so we generally learn them jointly.
  - Alternate between keeping one fixed and computing the gradients for the other.

Latent Space Models [4]: Training Loop

- Collect transitions (o_t, a_t, r_t, o_{t+1})

- Train single model on (weighted) sum of aforementioned loss terms

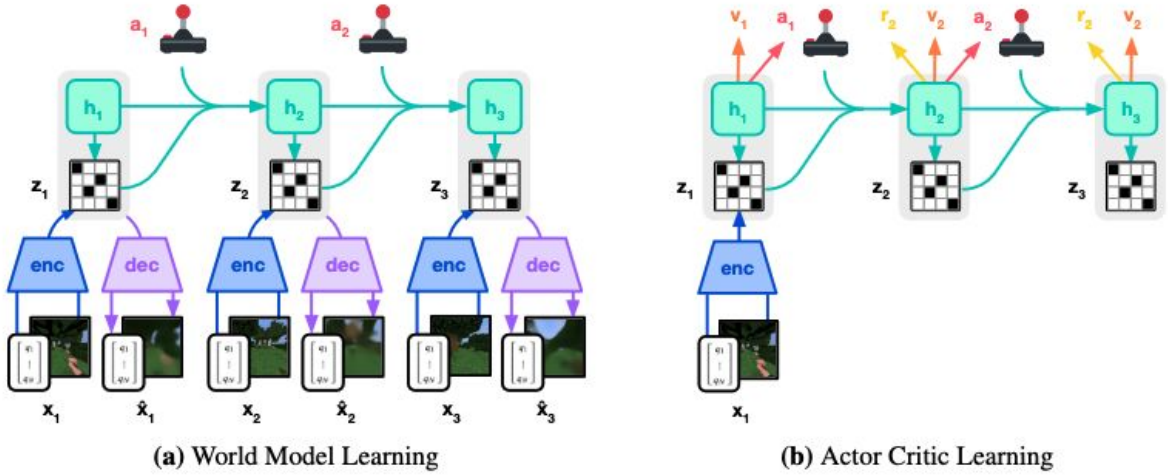
- **Should the dynamics model output a single vector or a distribution? What about the observation encoder?**

Latent Space Models [5]: Imagined Policy Learning

- Initialize policy π_θ, encoder g_ϕ, and model f_ψ, r_ψ
- Until Converged
  - Collect an episode with $\pi_\theta\left(g_\phi(o_t)\right)$ in the real environment. Add it to the replay buffer.
  - Use data from the replay buffer to update the model (slide 53).
  - Plan in the model for n times:
    - Sample an initial state z_0
    - Obtain a latent space trajectory $\left(z_0, a_0 = \pi(z_0), r_\psi(z_0, a_0), \ldots\right)$
    - Compute the policy gradient using the simulated trajectory
    - Update policy

- **What sort of learning algorithm makes sense for policy learning in this setting?**
- **What advantages might this have beyond representation learning?**

Example Latent Space Model: DreamerV3 [1]



- Two-part latent representation from *recurrent state-space model (RSSM)*: deterministic + recurrent h_t , stochastic z_t
- Imagined actor-critic uses A2C to capitalize on cheap samples
- **Why not just a transformer?**

Example Latent Space Model: DreamerV3 [2]

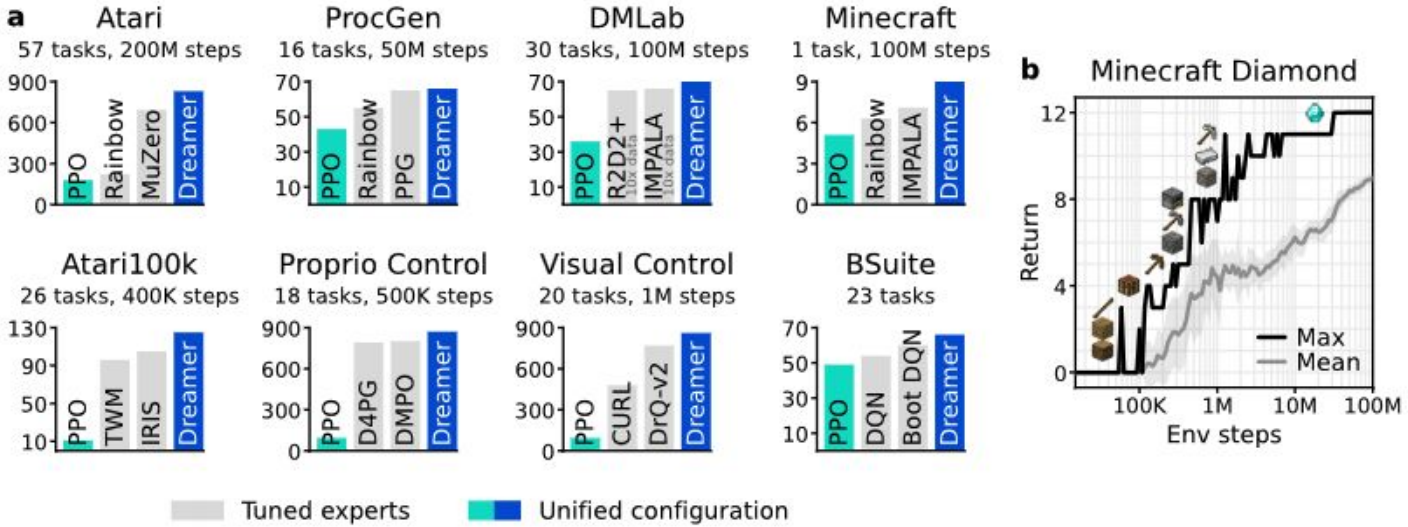
$$\mathcal{L}_{\text{pred}}(\phi) \doteq -\ln p_\phi(x_t \mid z_t, h_t) - \ln p_\phi(r_t \mid z_t, h_t) - \ln p_\phi(c_t \mid z_t, h_t)$$

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

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

- Addresses z_t's two roles by fitting dynamics model to encoder *and vice versa*
- **What failure mode does this approach risk? How might the decoding loss prevent it?**

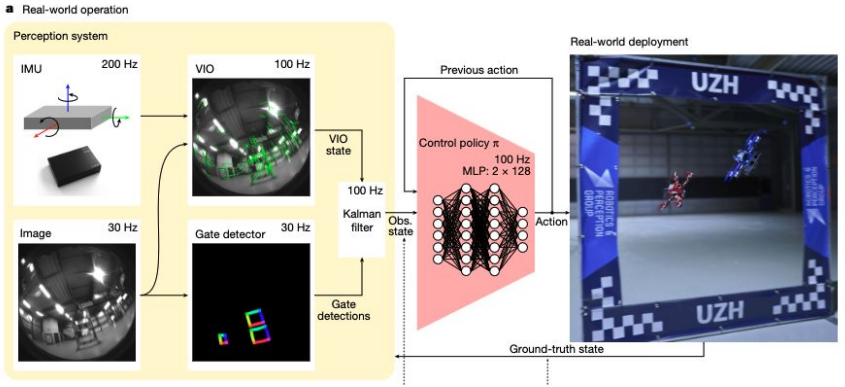
Example Latent Space Model: DreamerV3 [3]



- First model to obtain Minecraft diamonds with no imitation learning **(how?)**
- Substantially outperforms PPO (+ previous WMs) on various transitioned-constrained benchmarks

DreamerV3 Applications: SkyDreamer



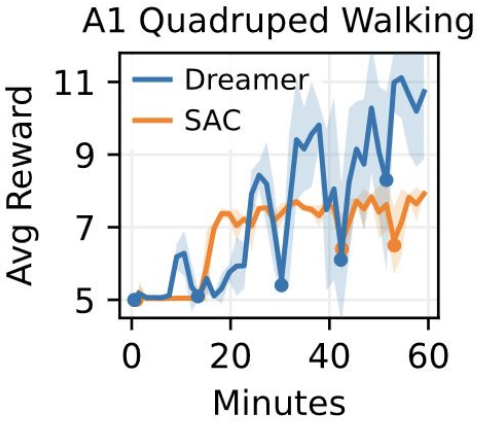


1. “Champion Level Drone Racing using Deep Reinforcement Learning” (Kaufmann et al., Science Robotics 2023)
2. “SkyDreamer: Interpretable End-to-End Vision-Based Drone Racing with Model-Based Reinforcement Learning
(Varraest et al., 2025)

- Very hard for a robot to figure out its own state from sensors (take ESE 6500 for more info)
- Can either use hand-build *state estimators*, or have an LSM learn it for you

DreamerV3 Applications: DayDreamer

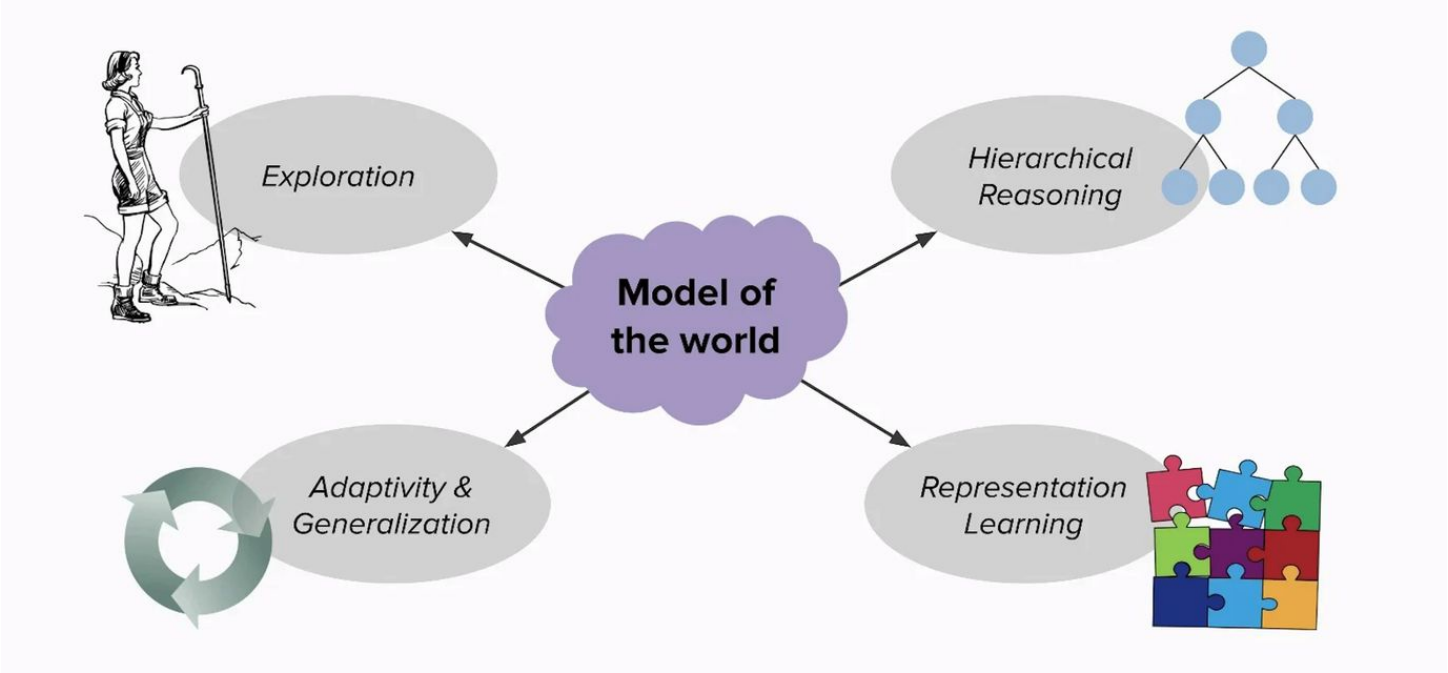




"DayDreamer: World Models for Physical Robot Learning" (Wu et al., CoRL 2022)

- Push Dreamer's sample efficiency to its limit by learning from real-world data
- Learns a better policy than SAC (go-to for sample-efficient on-policy learning), but training much less stable **(why?)**

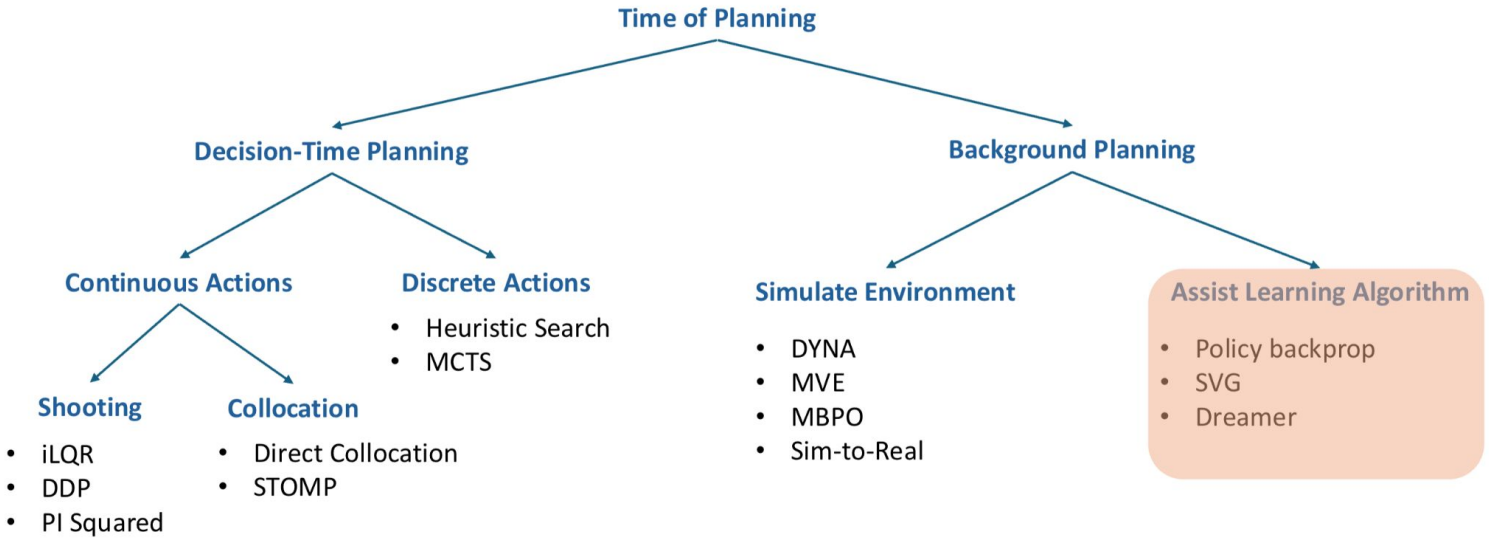
World Modeling: Summary



"Model Based Methods in Reinforcement Learning" (Mordatch & Hamrick, ICML 2020)

- MBRL has benefits beyond representation learning as discussed here
- **Why don't we always use it?**

Policy Backpropagation: Introduction



- We've seen how latent space modeling enables learning with limited environment knowledge
- Now we'll look into algorithms which capitalize on **complete environment knowledge**

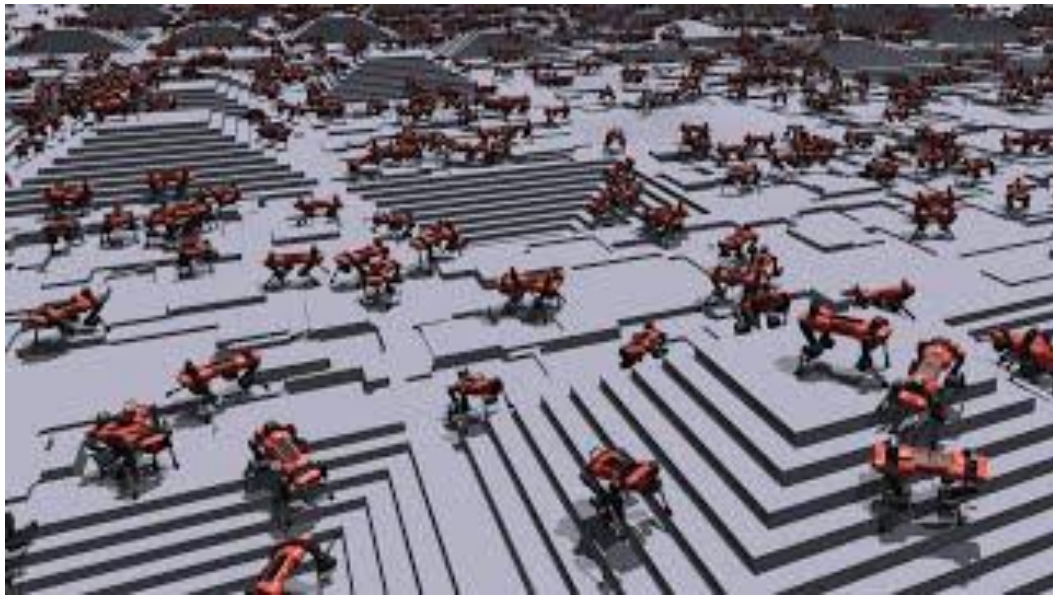
Throwback: Policy Gradient Theorem

$$
\begin{aligned}
& J(\theta) = \mathop{\mathbb{E}}_{\tau \sim p(\theta)} [R(\tau)] \\
\implies & \nabla_\theta J(\theta) = \nabla_\theta \int \Pr(\tau;\theta) R(\tau)\, d\tau && \text{(defn. of expectation)} \\
= & \int \nabla_\theta \Pr(\tau;\theta) R(\tau)\, d\tau && \text{(assuming sane dynamics)} \\
= & \int \Pr(\tau;\theta) \nabla_\theta \log \Pr(\tau;\theta) R(\tau)\, d\tau && \text{(log-derivative trick)} \\
= & \mathop{\mathbb{E}}_{\tau \sim p(\theta)} [\nabla \log p(\tau;\theta) R(\tau)] && \text{(defn. of expectation)}
\end{aligned}
$$

What's nice about the final estimator?

Policy Gradient Methods: The Irony

- Even the best PG methods require copious amounts of data (800 hours in following example)
- Once again the method that boasts freedom from dynamics knowledge needs dynamics
- **How can we design algorithms around having this knowledge?**



"Learning to Walk in Minutes"
(Rudin et al., CoRL 2021)

Differentiable Learning: Setup [1]

- Requirements: environment dynamics *F* are differentiable over state + action space, reward *R* is differentiable in state*

- Idea: differentiate *through* length-*T* trajectories using chain rule to find exact gradients of reward with respect to network weights

* *We will see later on that this is an awkward restriction for many common problem classes*

Differentiable Learning: Setup [2]

- More formally, assume we have partial derivatives of *f* and *r* WRT a_t , s_t
- Goal: optimize the trajectory level reward WRT the policy weights θ
- *Now instead of applying PG theorem, we will use these partials to differentiate through trajectory rewards*

$$f : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S} \qquad \text{(deterministic dynamics)}$$

$$r : \mathcal{S} \mapsto \mathbb{R} \qquad \text{(reward function)}$$

$$s := s_{0:T},\ a := a_{0:T-1},\ r_t := r(s_t) \qquad \text{(macros)}$$

$$\tau = (s_0, a_0, s_1, r_1, a_1, s_2, \ldots, s_T, r_T)$$

$$R(\tau) = \sum_{t=1}^{T} r_t \qquad \text{(trajectory reward)}$$

$$\mathscr{L}(\theta) = \mathop{\mathbb{E}}_{\tau \sim p_\theta} [R(\tau)]$$

Differentiable Learning: Taking Reward Gradients [1]

- From LOE, it suffices to have an expression for the gradient of the per-timestep reward r_t
- Intuitively, gradient of r_t should only depend on *previous* (s_t, r_t): can formalize this using **law of total expectation**

$$\nabla_\theta \mathscr{L}(\theta) = \nabla_\theta \mathop{\mathbb{E}}_{\tau \sim p_\theta} [R(\tau)]$$

$$= \sum_{t=0}^{T} \nabla_\theta \mathop{\mathbb{E}}_{\tau \sim p_\theta} [r_t] \qquad \text{(linearity of expectation)}$$

$$\mathop{\mathbb{E}}_{\tau \sim p_\theta} [r_t] = \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_\theta} [\mathbb{E}(r_t \mid \tau_{0:t})] \qquad \text{(law of total expectation)}$$

$$= \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_\theta} [r_t]$$

$$\therefore \nabla_\theta \mathscr{L}(\theta) = \sum_{t=0}^{T} \nabla_\theta \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_\theta} [r_t]$$

Differentiable Learning: Taking Reward Gradients [2]

- Can’t pull in gradient since the expectation depends on θ
- Apply **reparameterization trick** to remove this dependency by isolating policy randomness

$$\pi_\theta(\cdot \mid s_t) = \mathcal{N}\left(\mu_\theta(s_t), \sigma_\theta^2(s_t)\right) \qquad \text{(Gaussian policy)}$$

$$a_t = \mu_\theta(s_t) + \sigma_\theta(s_t)\varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0,1) \qquad \text{(reparameterization trick)}$$

$$p_\theta(\tau) = \rho(s_0) \prod_{t=0}^{T-1} f(s_{t+1} \mid s_t, a_t)\, \pi_\theta(a_t \mid s_t) \qquad \text{(trajectory density)}$$

Differentiable Learning: Taking Reward Gradients [2] (cont.)

- **What problem would emerge had we allowed stochastic dynamics?**

$$= \rho(s_0) \prod_{t=0}^{T-1} f\left(s_{t+1} \mid s_t, \mu_\theta(s_t) + \sigma_\theta(s_t)\varepsilon_t\right) \mathcal{N}(\varepsilon_t; 0, 1) \quad \text{(reparameterized form)}$$

$$\mathop{\mathbb{E}}_{\tau \sim p_\theta}[r_t] = \mathop{\mathbb{E}}_{s_0 \sim \rho,\ \varepsilon_{0:t-1} \sim \mathcal{N}(0,1)}[r_t] \quad \text{(isolate noise sources)}$$

$$\implies \nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_0, \varepsilon_{0:t-1}}[\nabla_\theta r_t]$$

Differentiable Learning: Taking Reward Gradients [3]

- Now we can start taking reward-network gradients using chain rule
- **What does each derivative represent?**

$$r_{t+1} = r(s_{t+1}) \qquad \text{(definition)}$$
$$s_{t+1} = f(s_t, a_t)$$
$$a_t = \mu_\theta(s_t) + \sigma_\theta(s_t)\varepsilon_t$$

$$\frac{dr_{t+1}}{d\theta} = \frac{\partial r_{t+1}}{\partial a_t}\frac{\partial a_t}{\partial \theta} + \frac{\partial r_{t+1}}{\partial s_t}\frac{ds_t}{d\theta} \qquad \text{(chain rule)}$$
$$\frac{da_t}{d\theta} = \frac{\partial a_t}{\partial \theta} + \frac{\partial a_t}{\partial s_t}\frac{ds_t}{d\theta} \qquad \text{(total derivative)}$$

Differentiable Learning: Taking Reward Gradients [4]

- Using the **chain rule for total derivatives,** we can take state-network gradients
- This leaves a recursive expression in previous state-network terms, which we'll next expand into closed form
- **Why do these suffice for taking reward-action gradients?**

$$\frac{ds_{t+1}}{d\theta} = \frac{\partial s_{t+1}}{\partial s_t}\frac{ds_t}{d\theta} + \frac{\partial s_{t+1}}{\partial a_t}\frac{da_t}{d\theta} \quad \text{(total derivative chain rule)}$$

$$= \left(\frac{\partial s_{t+1}}{\partial s_t} + \frac{\partial s_{t+1}}{\partial s_t}\frac{\partial a_t}{\partial s_t}\right)\frac{ds_t}{d\theta} + \frac{\partial s_{t+1}}{\partial a_t}\frac{\partial a_t}{\partial \theta} \quad \text{(plug in known derivatives)}$$

$$\frac{\partial s_{t+1}}{\partial s_t} + \frac{\partial s_{t+1}}{\partial s_t}\frac{\partial a_t}{\partial s_t} = \frac{ds_{t+1}}{ds_t} \quad \text{(total derivative chain rule)}$$

$$\implies \frac{ds_{t+1}}{d\theta} = \frac{ds_{t+1}}{ds_t}\frac{ds_t}{d\theta} + \frac{\partial s_{t+1}}{\partial a_t}\frac{\partial a_t}{\partial \theta}$$

Differentiable Learning: Taking Reward Gradients [5]

- Repeatedly expanding out the recurrence results in a closed form expression

$$\frac{ds_1}{d\theta} = 0 + \frac{\partial s_1}{\partial a_0}\frac{\partial a_0}{\partial \theta} \qquad \text{(apply one step recurrence)}$$

$$\frac{ds_2}{d\theta} = \frac{ds_2}{ds_1}\frac{\partial s_1}{\partial a_0}\frac{\partial a_0}{\partial \theta} + \frac{\partial s_2}{\partial a_1}\frac{\partial a_1}{\partial \theta} \qquad \text{(plug in above)}$$

Differentiable Learning: Taking Reward Gradients [5] (cont.)

- **What is the significance of each term in the summation?**

$$\frac{ds_3}{d\theta} = \frac{ds_3}{ds_2}\left(\frac{ds_2}{ds_1}\frac{\partial s_1}{\partial a_0}\frac{\partial a_0}{\partial \theta} + \frac{\partial s_2}{\partial a_1}\frac{\partial a_1}{\partial \theta}\right) + \frac{\partial s_3}{\partial a_2}\frac{\partial a_2}{\partial \theta}$$

$$= \frac{ds_3}{ds_1}\frac{\partial s_1}{\partial a_0}\frac{\partial a_0}{\partial \theta} + \frac{ds_3}{ds_2}\frac{\partial s_2}{\partial a_1}\frac{\partial a_1}{\partial \theta} + \frac{\partial s_3}{\partial a_2}\frac{\partial a_2}{\partial \theta} \qquad \text{(distribute + chain rule)}$$

$$\frac{ds_4}{d\theta} = \frac{ds_4}{ds_1}\frac{\partial s_1}{\partial a_0}\frac{\partial a_0}{\partial \theta} + \frac{ds_4}{ds_2}\frac{\partial s_2}{\partial a_1}\frac{\partial a_1}{\partial \theta} + \frac{ds_4}{ds_3}\frac{\partial s_3}{\partial a_2}\frac{\partial a_2}{\partial \theta} + \frac{\partial s_4}{\partial a_3}\frac{\partial a_3}{\partial \theta}$$

$$\dots$$

$$\therefore \frac{ds_t}{d\theta} = \sum_{k=1}^{t} \frac{ds_t}{ds_k}\frac{\partial s_k}{\partial a_{k-1}}\frac{\partial a_{k-1}}{\partial \theta}$$

Differentiable Learning: Taking Reward Gradients [6]

- With one last chain rule application we get a trajectory reward-network gradient
- **What nice properties does this expression have over PG theorem?**
- **Conversely, how might it be more difficult to work with in practice?**

$$\nabla_\theta \mathscr{L}(\theta) = \sum_{t=0}^{T} \mathop{\mathbb{E}}_{s_0, \varepsilon_{0:T-1}} [\nabla_\theta r_t]$$

$$= \sum_{t=0}^{T} \mathop{\mathbb{E}}_{s_0, \varepsilon_{0:T-1}} \left[\frac{dr_t}{ds_t} \frac{ds_t}{d\theta} \right] \qquad \text{(chain rule)}$$

$$= \sum_{t=0}^{T} \mathop{\mathbb{E}}_{s_0, \varepsilon_{0:T-1}} \left[\frac{dr_t}{ds_t} \sum_{k=1}^{t} \frac{ds_t}{ds_k} \frac{\partial s_k}{\partial a_{k-1}} \frac{\partial a_{k-1}}{\partial \theta} \right]$$

Differentiable Learning: Putting it Together [2]

- *Backpropagation Through Time (BPTT)*: on-policy loop similar to that from PPO, with only the update rule changing

- **What sort of benefits might we expect from using analytical gradients over PG estimates?**

- **What sort of tasks might most clearly display said benefits?**

Algorithm 1 Backpropagation Through Time (BPTT)

1: **Input:** initial policy parameters θ_0, differentiable dynamics and rewards f, r, horizon T, batch size N

2: **for** $i = 0, 1, 2, \ldots$ **do**

3: Sample $s_0 \sim \rho$ and $\varepsilon_{0:T-1} \sim \mathcal{N}(0, I)$ for each of N rollouts, and roll out π_{θ_i} through f.

4: Objective

$$J(\theta) = \mathop{\mathbb{E}}_{s_0, \varepsilon_{0:T-1}} \left[\sum_{t=1}^{T} r_t \right]$$

5: Compute state–network gradients for $t = 1, \ldots, T$

$$\frac{ds_t}{d\theta} = \sum_{k=1}^{t} \frac{ds_t}{ds_k} \frac{\partial s_k}{\partial a_{k-1}} \frac{\partial a_{k-1}}{\partial \theta}$$

6: Take Monte Carlo gradient estimate

$$\hat{g}_i = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \frac{dr_t}{ds_t} \frac{ds_t}{d\theta} \bigg|_{\theta_i}$$

7: Update policy parameters

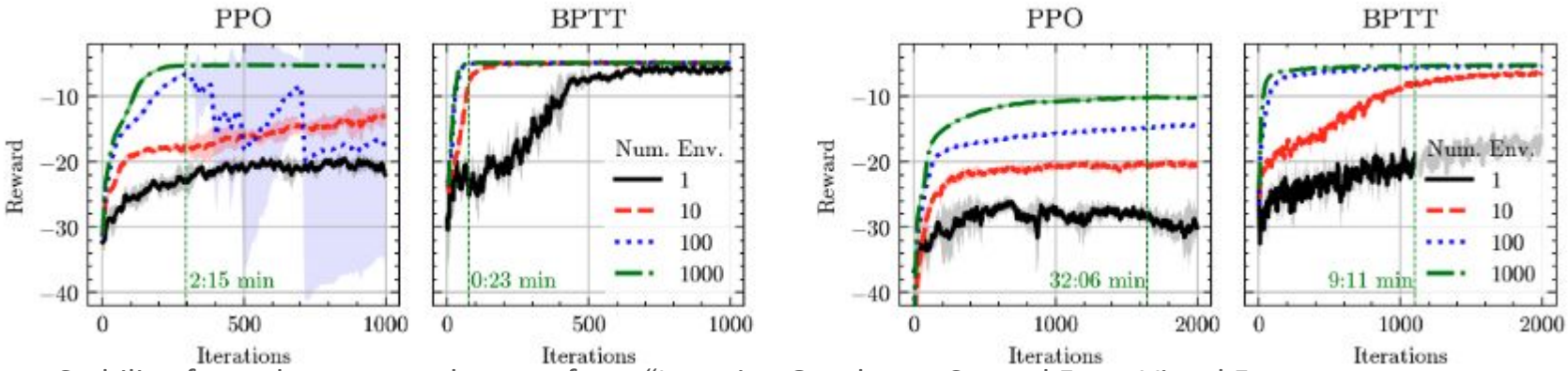
$$\theta_{i+1} = \theta_i + \alpha_i \hat{g}_i$$

8: **end for**

BPTT: The Good

Ground-truth State Observations

FPV Observations



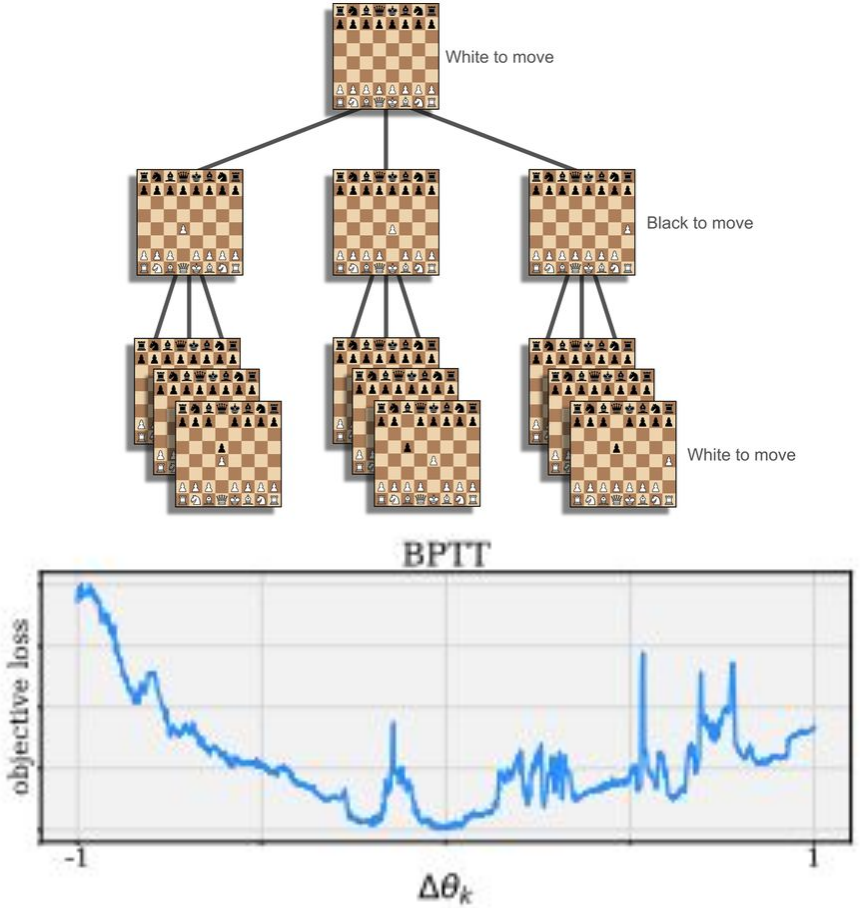
Stabilize-from-throw reward curves from “Learning Quadrotor Control From Visual Features Using Differentiable Simulation” (Heeg et al., ICRA 2025)

- Lower variance gradients mean greatly improved sample efficiency relative to PG baselines
- Differentiating through dynamics greatly aids representation learning in PO-MDPs

BPTT: The Bad

- The theoretically exact gradients are often very difficult to work with in practice!
  - Propagating gradients through long sequences is hard for the same reasons as with big networks
  - Long horizon credit assignment is a double-edged sword

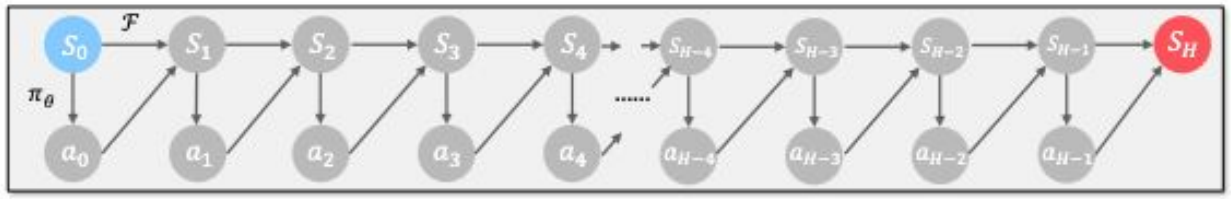
- Bottom chart: gradient values for single parameter with value [-1, 1] for long-horizon (T = 1000) trajectory



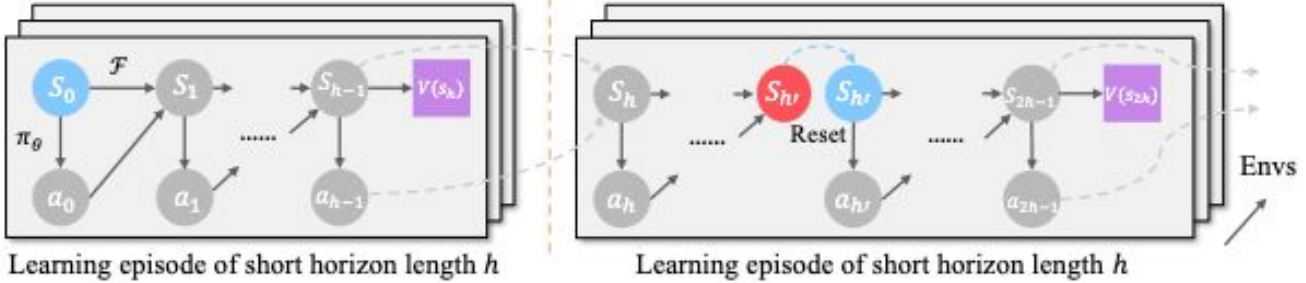
"Accelerated Policy Learning with Parallel Differentiable Simulation" (Xu et al., ICLR 2022)

Improving BPTT: Short Horizon Actor Critic (SHAC) [1]

BPTT Gradient Flow



SHAC Gradient Flow



- Idea: capture the expected long-horizon reward gradient directly using a learned critic
- Split the full length *H* episode into length *h* critic-bootstrapped windows, with BPTT performed in each window
- **How should we change the critic objective to capitalize on dynamics knowledge?**

Improving BPTT: Short Horizon Actor Critic (SHAC) [2]

- Actor-critic VF objective can stay: expected return from state doesn't need dynamics info
- **Why the stop-grad on the critic update?**



Algorithm 2 Short-Horizon Actor-Critic (SHAC)

Input: initial policy parameters θ_0, critic parameters ϕ_0, differentiable dynamics and rewards f, r, short horizon H, discount γ, batch size N

for $i = 0, 1, 2, \ldots$ **do**

Start each rollout from the previous window's final state (or $s_0 \sim \rho$ on reset), treated as constant: $\frac{ds_0}{d\theta} = 0$. Sample $\varepsilon_{0:H-1} \sim \mathcal{N}(0, I)$ and roll out π_{θ_i} for H steps.

Objective

$$J_H(\theta) = \mathop{\mathbb{E}}_{\varepsilon_{0:H-1}}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t + \gamma^H V_{\phi_i}(s_H)\right]$$

Compute state–network gradients for $t = 1, \ldots, H$

$$\frac{ds_t}{d\theta} = \sum_{k=1}^{t} \frac{ds_t}{ds_k} \frac{\partial s_k}{\partial a_{k-1}} \frac{\partial a_{k-1}}{\partial \theta}$$

Take Monte Carlo gradient estimate

$$\hat{g}_i = \frac{1}{NH} \sum_{n=1}^{N} \left[\sum_{t=1}^{H} \gamma^{t-1} \frac{dr_t}{ds_t} \frac{ds_t}{d\theta} + \gamma^H \frac{\partial V_{\phi_i}(s_H)}{\partial s_H} \frac{ds_H}{d\theta}\right]_{\theta_i}$$

Update policy parameters

$$\theta_{i+1} = \theta_i + \alpha_i \hat{g}_i$$

Compute critic targets

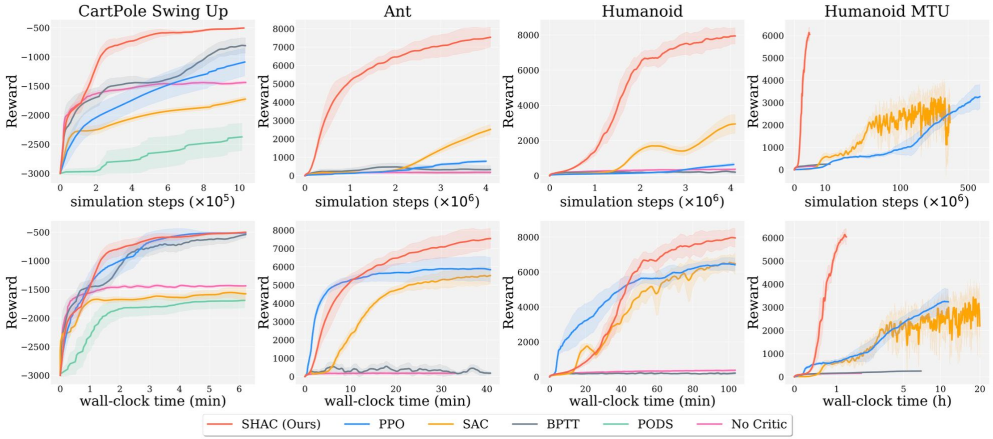
$$\hat{V}_k = \sum_{j=1}^{H-k} \gamma^{j-1} r_{k+j} + \gamma^{H-k} V_{\phi_i}(s_H)$$

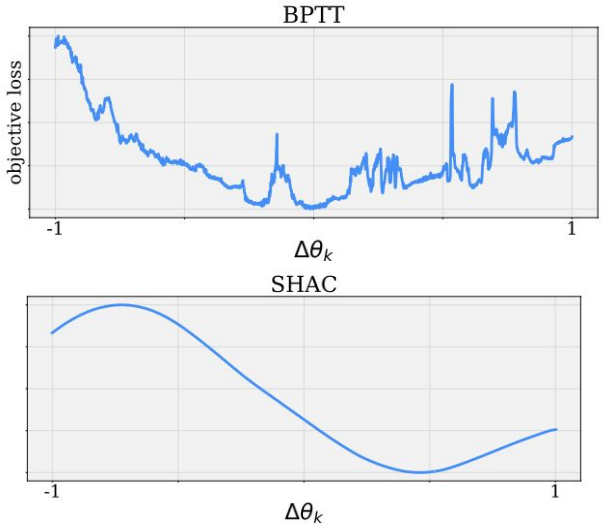
Update critic parameters

$$\phi_{i+1} = \arg\min_{\phi} \frac{1}{NH} \sum_{n=1}^{N} \sum_{k=0}^{H-1} \left(V_\phi(s_k) - \text{stopgrad}(\hat{V}_k)\right)^2$$

end for

Improving BPTT: Short Horizon Actor Critic (SHAC) [3]





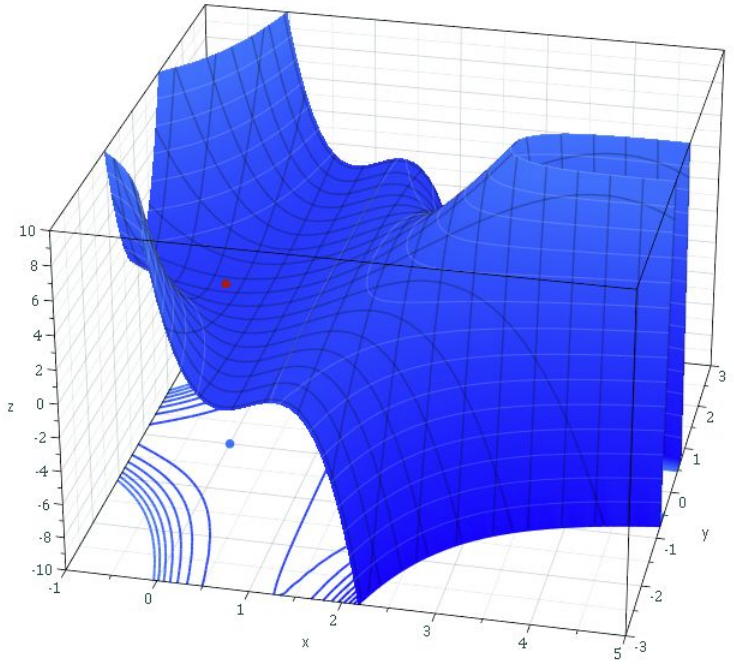
- Improvement most visible in complex, high-dimensional tasks such as motor-level humanoid control
- Attributed to gradients becoming smoother and less prone to exploding

Improving BPTT: Soft Analytic Policy Optimization [1]

- We've seen how to extract max signal from a particular τ_i, but haven't touched how to get good τ_i

- *Much* practical work in RL revolves around setting up hyperparameters + training loops to guide exploration

- Can we also address the issue at the objective level?



Wikipedia: Maximum and Minimum

Improving BPTT: Soft Analytic Policy Optimization (SAPO) [2]

- Assuming a Gaussian policy, we can differentiate the **Maximum Entropy objective** WRT weights as well
- **A common practice for encouraging exploration is to add a separate per-timestep entropy loss. How does MaxEntRL differ?**

$$J_\alpha(\theta) = \mathop{\mathbb{E}}_{\tau \sim p_\theta} \left[\sum_{t=1}^{T} r_t - \alpha \sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t) \right] \quad \text{(objective)}$$

$$h_t := -\alpha \log \pi_\theta(a_t \mid s_t) \quad \text{(entropy bonus)}$$

$$\therefore \nabla_\theta J_\alpha(\theta) = \sum_{t=1}^{T} \mathop{\mathbb{E}}_{s_0, \varepsilon} [\nabla_\theta r_t] + \sum_{t=0}^{T-1} \mathop{\mathbb{E}}_{s_0, \varepsilon} [\nabla_\theta h_t] \quad \text{(linearity)}$$

Improving BPTT: Soft Analytic Policy Optimization (SAPO) [3]

- Since the policy samples each action dimension independently, we can write the entropy as a function of σ
  - Notably, h_t does not depend on a_t, simplifying the total derivative

$$\log \pi_\theta(a_t \mid s_t) = \log \mathcal{N}\left(a_t;\, \mu_\theta(s_t),\, \sigma_\theta^2(s_t)\right)$$

$$= -\frac{1}{2}\sum_{k=1}^{D}\left(\frac{a_{t,k} - \mu_{\theta,k}(s_t)}{\sigma_{\theta,k}(s_t)}\right)^2 - \sum_{k=1}^{D}\log \sigma_{\theta,k}(s_t) - \frac{D}{2}\log 2\pi \qquad \text{(diag. Gaussian)}$$

$$= -\frac{1}{2}\sum_{k=1}^{D}\varepsilon_{t,k}^2 - \sum_{k=1}^{D}\log \sigma_{\theta,k}(s_t) - \frac{D}{2}\log 2\pi \qquad \text{(reparam.)}$$

$$\therefore\ h_t = \alpha \sum_{k=1}^{D}\log \sigma_{\theta,k}(s_t) + \text{const} \qquad \text{(indep. of } a_t\text{)}$$

Improving BPTT: Soft Analytic Policy Optimization (SAPO) [3] (cont.)

- We can then use the chain rule on the total derivative WRT θ as before

$$\frac{dh_t}{d\theta} = \frac{\partial h_t}{\partial \theta} + \frac{\partial h_t}{\partial s_t}\frac{ds_t}{d\theta} \qquad \text{(chain rule)}$$

$$\frac{\partial h_t}{\partial \theta} = \alpha \sum_{k=1}^{D} \frac{1}{\sigma_{\theta,k}(s_t)} \frac{\partial \sigma_{\theta,k}(s_t)}{\partial \theta}$$

$$\frac{\partial h_t}{\partial s_t} = \alpha \sum_{k=1}^{D} \frac{1}{\sigma_{\theta,k}(s_t)} \frac{\partial \sigma_{\theta,k}(s_t)}{\partial s_t}$$

$$\frac{dh_t}{d\theta} = \frac{\partial h_t}{\partial \theta} + \frac{\partial h_t}{\partial s_t} \sum_{k=1}^{t} \frac{ds_t}{ds_k}\frac{\partial s_k}{\partial a_{k-1}}\frac{\partial a_{k-1}}{\partial \theta} \qquad \text{(insert } \tfrac{ds_t}{d\theta}\text{)}$$

Improving BPTT: Soft Analytic Policy Optimization (SAPO) [4]

- Only fundamental change from SHAC is the value function target
  - In practice, “Stabilizing Reinforcement Learning in Differentiable Multiphysics Learning” (Xing et al., ICLR 2025) employs many empirical tricks like critic ensembles
- Automatic entropy tuning from SAC can be used here
- **What sort of improvement do we expect to see over SHAC and BPTT?**

Algorithm 3 Soft Analytic Policy Optimization (SAPO)

Input: initial policy parameters θ_0, critic parameters ϕ_0, entropy temperature α_0, differentiable dynamics and rewards f, r, short horizon H, discount γ, batch size N

for $i = 0, 1, 2, \dots$ **do**

Start each rollout from the previous window's final state (or $s_0 \sim \rho$ on reset), treated as constant: $\frac{ds_0}{d\theta} = 0$. Sample $\varepsilon_{0:H-1} \sim \mathcal{N}(0, I)$ and roll out π_{θ_i} for H steps.

Objective, with $h_t := -\alpha_i \log \pi_\theta(a_t \mid s_t)$

$$J_H(\theta) = \mathop{\mathbb{E}}_{\varepsilon_{0:H-1}}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t + \sum_{t=0}^{H-1} \gamma^t h_t + \gamma^H V_{\phi_i}(s_H)\right]$$

Compute state–network gradients for $t = 1, \dots, H$

$$\frac{ds_t}{d\theta} = \sum_{k=1}^{t} \frac{ds_t}{ds_k} \frac{\partial s_k}{\partial a_{k-1}} \frac{\partial a_{k-1}}{\partial \theta}$$

Estimate soft gradient

$$\hat{g}_i = \frac{1}{N} \sum_{n=1}^{N} \left[\sum_{t=1}^{H} \gamma^{t-1} \frac{dr_t}{d\theta} + \sum_{t=0}^{H-1} \gamma^t \frac{dh_t}{d\theta} + \gamma^H \frac{\partial V_{\phi_i}(s_H)}{\partial s_H} \frac{ds_H}{d\theta}\right]_{\theta_i}$$

Update policy parameters

$$\theta_{i+1} \leftarrow \theta_i + \eta_\theta \hat{g}_i$$

Compute soft value-function targets

$$\hat{V}_k = \sum_{j=1}^{H-k} \gamma^{j-1} \left[r_{k+j} + h_{k+j-1}\right] + \gamma^{H-k} V_{\phi_i}(s_H)$$

Update critic parameters (no derivative through the target)

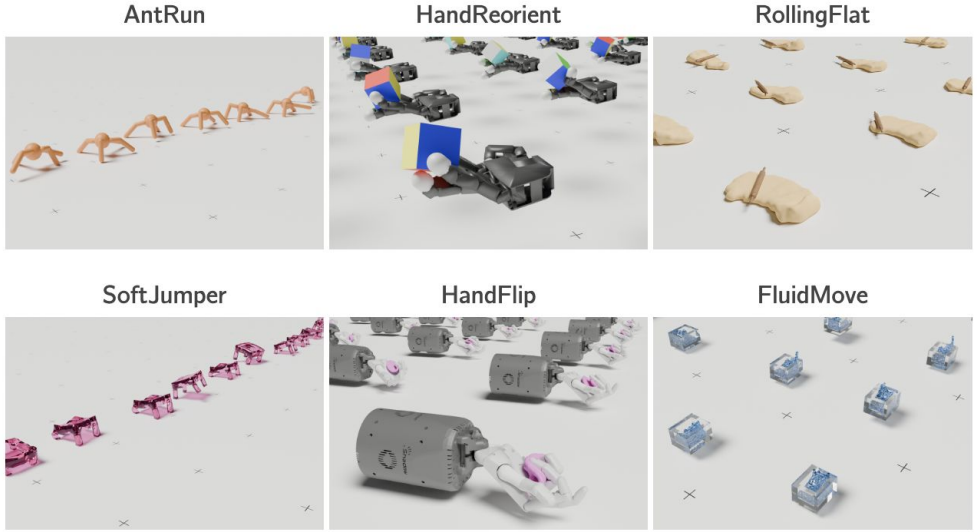
$$\phi_{i+1} \leftarrow \arg\min_{\phi} \frac{1}{N} \sum_{n=1}^{N} \sum_{k=0}^{H-1} \left(V_\phi(s_k) - \text{stopgrad}(\hat{V}_k)\right)^2$$

Update entropy temperature

$$\alpha_{i+1} \leftarrow \arg\min_{\alpha \geq 0} \mathbb{E}_{\varepsilon_k}\left[-\alpha \left(\log \pi_{\theta_{i+1}}(a_k \mid s_k) + \bar{\mathcal{H}}\right)\right]$$

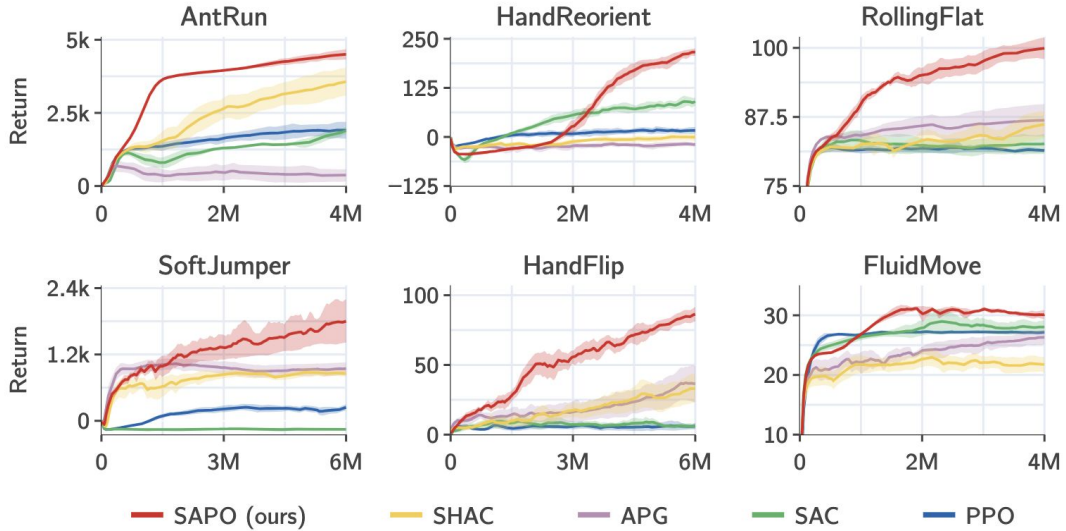
end for

Improving BPTT: Soft Analytic Policy Optimization (SAPO) [4]



- AntRun, HandReorient: rigid bodies, full observability
- Other four: deformable bodies, partial observability through vision

Improving BPTT: Soft Analytic Policy Optimization (SAPO) [5]



- The MaxEnt objective reliably avoids local optima in BPTT (called analytic policy gradient, APG) which short-horizon optimization alone cannot
- In some cases, this is even required in order to beat PPO convincingly

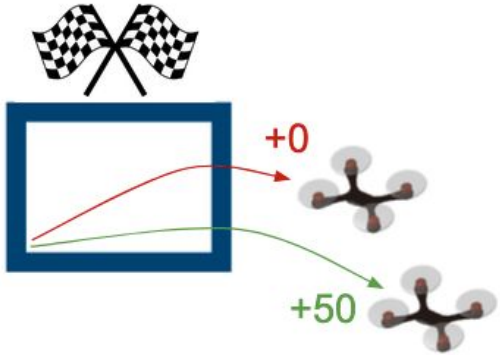
** Of course we must trust that their baseline hyperparameter tuning was legitimate :)*

BPTT: The Ugly

- It’s very common that at least one of your dynamics or reward functions is discontinuous!



- Event-level rewards, contact forces, failure penalties, all break differentiability



- Latter two in particular require differentiable approximations, but even the best are imperfect (Zhong et al., ICML 2022)

BPTT: Summary

- Under ideal conditions, a much more direct approach to policy learning than actor-critic

- However these conditions are rare in practice, and the theoretically promising objective demands its own empirical workarounds

- **One last thought: what if we differentiate through a learned world model? (*Do not!*)**

	SIMULATION			
MODEL	LENGTH	AVG. REWARD	TRANSFERABILITY	REPEATABILITY
LSTM	40CM	0.92 ± 0.37	0%	-
FF-NN	40CM	0.86 ± 0.35	0%	-
NOMINAL	40CM	2.45 ± 1.15	**64%**	-
	PHYSICAL SYSTEM			
	LENGTH	AVG. REWARD	TRANSFERABILITY	REPEATABILITY
	40CM	0.91 ± 0.56	0%	0%
	40CM	1.46 ± 0.78	0%	0%
	40CM	1.41 ± 0.45	0%	0%

"Differentiable Physics Models for Real-world Offline MBRL" (Lutter et al., ICRA 2021)

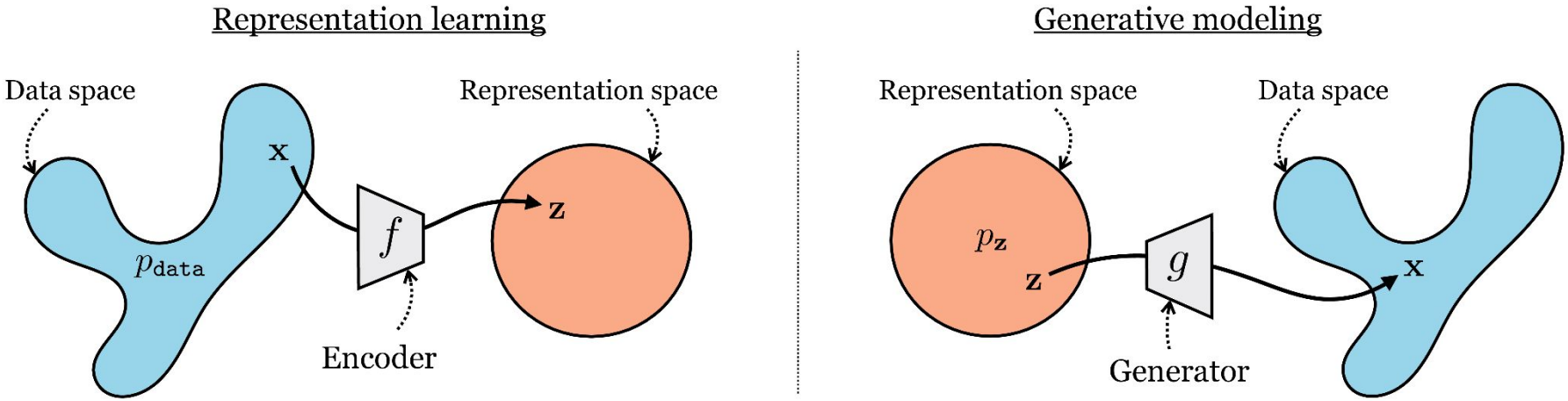
Bonus: Offline World Modeling

- **How does DreamerV3 cope when the policy aims to explore states out of the world model's distribution? Is this an option when working with a fixed offline dataset?**

- **Does the RSSM architecture still make sense for offline data?**

- *Learning world models from offline data is more analogous to generative modeling than to the LSM setting*

Generative Modeling: Problem Statement

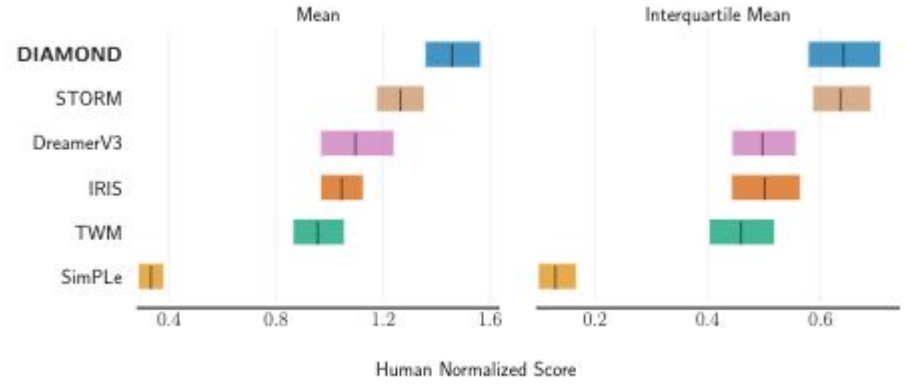


"Foundations of Computer Vision" (Torralba et al., 2024)

- Say we have a fixed set of samples from a distribution p_{data} which we'd like to sample
- Idea: encode samples into representations which follow an easily sampled p_z , then learn a decoder $g : p_z \rightarrow p_{\text{data}}$
- **What's our data distribution in offline world modeling?**

Generative Offline World Modeling: DIAMOND [2]

Mean human-relative performance on Atari100k



Imagined Counter Strike Trajectory



- *Diffusion model*: class of generative model with Gaussian latent space
- Outperforms Dreamer in online sample efficiency + can learn environments offline
- **What advantages might training in DIAMOND imagination have over model-free offline RL (e.g. Q-Learning?)**

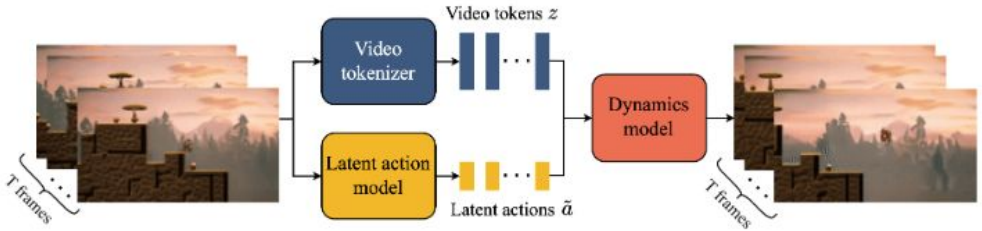
Generative Offline World Modeling: Genie [1]

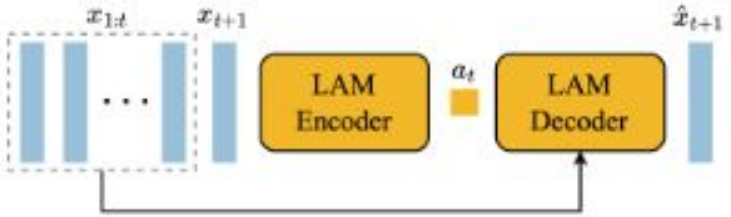
- Let's make the data requirement even more stringent: now try learning a world model without action labels
- *Lots* of video game footage on the internet, i.e. observation trajectories without actions
- **Can we train a world model from observation data alone?**



"Cumulative Minecraft hours watched across selected live-streaming platforms from 3rd quarter 2016 to 2nd quarter 2024 (in millions)" (Statista Research Department)

Generative Offline World Modeling: Genie [2]

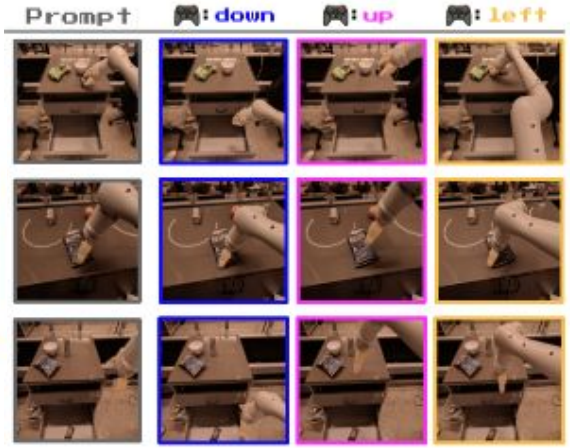




- It turns out that states are not the only thing for which we can learn representations
- *Latent action model* encodes observations $x_{1:t}$, and outputs *discrete* $a_{1:t}$ which the decoder uses to predict x_{t+1}
- **Why might discrete action representations be preferable?**

Generative Offline World Modeling: Genie [3]





- Genie successfully models physical interactions, particularly with deformable objects (bag of chips)
- The LAM outputs have consistent semantics in a variety of settings, reliably corresponding to certain directional movements