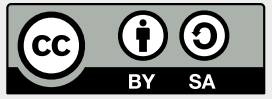
PlaNet, ST and Gumbel-softmax, DreamerV2, DreamerV3

Milan Straka

May 13, 2024

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

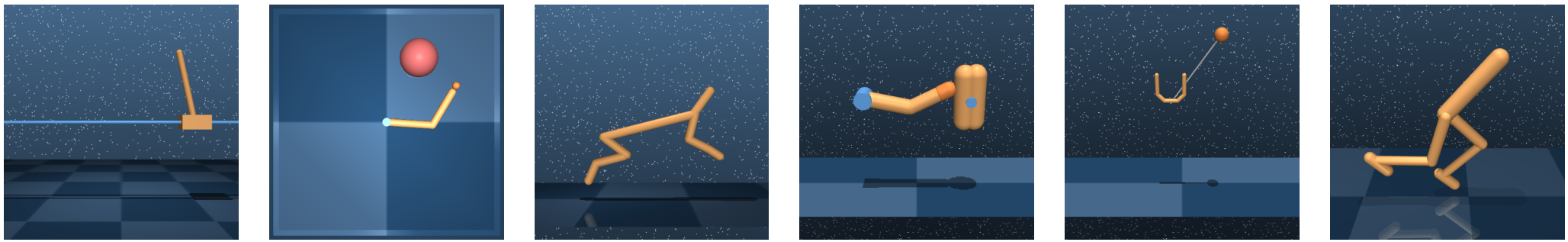
Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



PlaNet

In Nov 2018, an interesting paper from D. Hafner et al. proposed a **Deep Planning Network (PlaNet)**, which is a model-based agent that learns the MDP dynamics from pixels, and then chooses actions using a CEM planner utilizing the learned compact latent space.

The PlaNet is evaluated on selected tasks from the DeepMind control suite



(a) Cartpole (b) Reacher (c) Cheetah (d) Finger (e) Cup (f) Walker

Figure 1 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

In PlaNet, partially observable MDPs following the stochastic dynamics are considered:

$$\begin{aligned}
\text{transition function:} \quad & s_t \sim p(s_t|s_{t-1}, a_{t-1}), \\
\text{observation function:} \quad & o_t \sim p(o_t|s_t), \\
\text{reward function:} \quad & r_t \sim p(r_t|s_t), \\
\text{policy:} \quad & a_t \sim p(a_t|o_{\leq t}, a_{<t}).
\end{aligned}$$

The main goal is to train the first three – the transition function, the observation function, and the reward function.

PlaNet – Data Collection

Algorithm 1: Deep Planning Network (PlaNet)

Input :

R	Action repeat	$p(s_t \mid s_{t-1}, a_{t-1})$	Transition model
S	Seed episodes	$p(o_t \mid s_t)$	Observation model
C	Collect interval	$p(r_t \mid s_t)$	Reward model
B	Batch size	$q(s_t \mid o_{\leq t}, a_{<t})$	Encoder
L	Chunk length	$p(\epsilon)$	Exploration noise
α	Learning rate		

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize model parameters θ randomly.
while *not converged* **do**
  `// Model fitting`
  for *update step* $s = 1..C$ **do**
    Draw sequence chunks $\{(o_t, a_t, r_t)_{t=k}^{L+k}\}_{i=1}^{B} \sim \mathcal{D}$ uniformly at random from the dataset.
    Compute loss $\mathcal{L}(\theta)$ from Equation 3.
    Update model parameters $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$.
  `// Data collection`
  $o_1 \leftarrow$ `env.reset()`
  for *time step* $t = 1..\lceil \frac{T}{R} \rceil$ **do**
    Infer belief over current state $q(s_t \mid o_{\leq t}, a_{<t})$ from the history.
    $a_t \leftarrow$ `planner`$(q(s_t \mid o_{\leq t}, a_{<t}), p)$, see Algorithm 2 in the appendix for details.
    Add exploration noise $\epsilon \sim p(\epsilon)$ to the action.
    for *action repeat* $k = 1..R$ **do**
      $r_t^k, o_{t+1}^k \leftarrow$ `env.step`(a_t)
    $r_t, o_{t+1} \leftarrow \sum_{k=1}^{R} r_t^k, o_{t+1}^R$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$

Algorithm 1 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

Because an untrained agent will most likely not cover all needed environment states, we need to iteratively collect new experience and train the model. The authors propose $S = 5$, $C = 100$, $B = 50$, $L = 50$, R between 2 and 8.

For planning, CEM algorithm (capable of solving all tasks with a true model) is used; $H = 12$, $I = 10$, $J = 1000$, $K = 100$.

Algorithm 2: Latent planning with CEM

Input :

H	Planning horizon distance	$q(s_t \mid o_{\leq t}, a_{<t})$	Current state belief
I	Optimization iterations	$p(s_t \mid s_{t-1}, a_{t-1})$	Transition model
J	Candidates per iteration	$p(r_t \mid s_t)$	Reward model
K	Number of top candidates to fit		

Initialize factorized belief over action sequences $q(a_{t:t+H}) \leftarrow \text{Normal}(0, \mathbb{I})$.
for *optimization iteration* $i = 1..I$ **do**
  `// Evaluate` J `action sequences from the current belief.`
  for *candidate action sequence* $j = 1..J$ **do**
    $a_{t:t+H}^{(j)} \sim q(a_{t:t+H})$
    $s_{t:t+H+1}^{(j)} \sim q(s_t \mid o_{1:t}, a_{1:t-1}) \prod_{\tau=t+1}^{t+H+1} p(s_\tau \mid s_{\tau-1}, a_{\tau-1}^{(j)})$
    $R^{(j)} = \sum_{\tau=t+1}^{t+H+1} \mathrm{E}[p(r_\tau \mid s_\tau^{(j)})]$
  `// Re-fit belief to the` K `best action sequences.`
  $\mathcal{K} \leftarrow \text{argsort}(\{R^{(j)}\}_{j=1}^{J})_{1:K}$
  $\mu_{t:t+H} = \frac{1}{K} \sum_{k \in \mathcal{K}} a_{t:t+H}^{(k)}, \quad \sigma_{t:t+H} = \frac{1}{K-1} \sum_{k \in \mathcal{K}} |a_{t:t+H}^{(k)} - \mu_{t:t+H}|.$
  $q(a_{t:t+H}) \leftarrow \text{Normal}(\mu_{t:t+H}, \sigma_{t:t+H}^2 \mathbb{I})$
return *first action mean* μ_t.

Algorithm 2 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

PlaNet – Latent Dynamics

First let us consider a typical latent-space model, consisting of

$$\begin{aligned}\text{transition function:} \quad & s_t \sim p(s_t|s_{t-1}, a_{t-1}),\\ \text{observation function:} \quad & o_t \sim p(o_t|s_t),\\ \text{reward function:} \quad & r_t \sim p(r_t|s_t).\end{aligned}$$

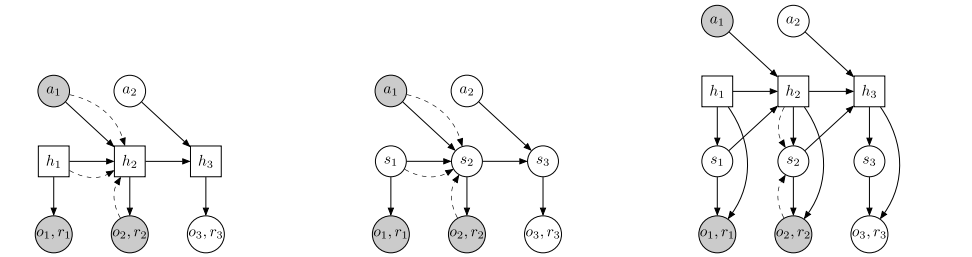


Figure 2 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

The transition model is Gaussian with mean and variance predicted by a network, the observation model is Gaussian with identity covariance and mean predicted by a deconvolutional network, and the reward model is a scalar Gaussian with unit variance and mean predicted by a neural network.

To train such a model, we turn to variational inference, and use an encoder $q(s_{1:T}|o_{1:T}, a_{1:T-1}) = \prod_{t=1}^{T} q(s_t|s_{t-1}, a_{t-1}, o_t)$, which is a Gaussian with mean and variance predicted by a convolutional neural network.

PlaNet – Training Objective

Using the encoder, we obtain the following variational lower bound on the log-likelihood of the observations (for rewards the bound is analogous):

$$\begin{aligned}
&\log p(o_{1:T}|a_{1:T}) \\
&\quad = \log \int \prod_t p(s_t|s_{t-1},a_{t-1}) p(o_t|s_t)\,\mathrm{d}s_{1:T} \\
&\quad \geq \sum_{t=1}^{T} \Big(\underbrace{\mathbb{E}_{q(s_t|o_{\leq t},a_{<t})} \log p(o_t|s_t)}_{\text{reconstruction}} - \underbrace{\mathbb{E}_{q(s_{t-1}|o_{\leq t-1},a_{<t-1})} D_{\mathrm{KL}}\big(q(s_t|o_{\leq t},a_{<t}) \| p(s_t|s_{t-1},a_{t-1})\big)}_{\text{complexity}} \Big).
\end{aligned}$$

We evaluate the expectations using a single sample, and use the reparametrization trick to allow backpropagation through the sampling.

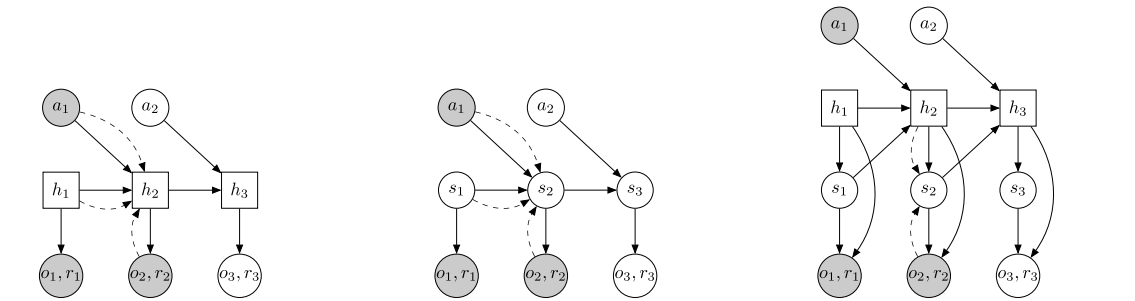
PlaNet – Training Objective Derivation

To derive the training objective, we employ importance sampling and the Jensen's inequality:

$$
\begin{aligned}
&\log p(o_{1:T}|a_{1:T}) \\
&\quad = \log \mathbb{E}_{p(s_{1:T}|a_{1:T})} \prod_{t=1}^{T} p(o_t|s_t) \\
&\quad = \log \mathbb{E}_{q(s_{1:T}|o_{1:T},a_{1:T})} \prod_{t=1}^{T} p(o_t|s_t)p(s_t|s_{t-1},a_{t-1})/q(s_t|o_{\leq t},a_{<t}) \\
&\quad \geq \mathbb{E}_{q(s_{1:T}|o_{1:T},a_{1:T})} \sum_{t=1}^{T} \log p(o_t|s_t) + \log p(s_t|s_{t-1},a_{t-1}) - \log q(s_t|o_{\leq t},a_{<t}) \\
&= \sum_{t=1}^{T} \Big(\underbrace{\mathbb{E}_{q(s_t|o_{\leq t},a_{<t})} \log p(o_t|s_t)}_{\text{reconstruction}} - \underbrace{\mathbb{E}_{q(s_{t-1}|o_{\leq t-1},a_{<t-1})} D_{\mathrm{KL}}\left(q(s_t|o_{\leq t},a_{<t}) \| p(s_t|s_{t-1},a_{t-1})\right)}_{\text{complexity}} \Big).
\end{aligned}
$$

PlaNet – Recurrent State-Space Model

The purely stochastic transitions struggle to store information for multiple timesteps. Therefore, the authors propose to include a deterministic path to the model (providing access to all previous states), obtaining the **recurrent state-space model (RSSM)**:



(a) Deterministic model (RNN) (b) Stochastic model (SSM) (c) Recurrent state-space model (RSSM)

Figure 2 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

$$\begin{aligned}
\text{deterministic state model:} \quad & h_t = f(h_{t-1}, s_{t-1}, a_{t-1}), \\
\text{stochastic state function:} \quad & s_t \sim p(s_t|h_t), \\
\text{observation function:} \quad & o_t \sim p(o_t|h_t, s_t), \\
\text{reward function:} \quad & r_t \sim p(r_t|h_t, s_t), \\
\text{encoder:} \quad & q_t \sim q(s_t|h_t, o_t).
\end{aligned}$$

PlaNet – Results

Table 1: Comparison of PlaNet to the model-free algorithms A3C and D4PG reported by Tassa et al. (2018). The training curves for these are shown as orange lines in Figure 4 and as solid green lines in Figure 6 in their paper. From these, we estimate the number of episodes that D4PG takes to achieve the final performance of PlaNet to estimate the data efficiency gain. We further include CEM planning ($H = 12, I = 10, J = 1000, K = 100$) with the true simulator instead of learned dynamics as an estimated upper bound on performance. Numbers indicate mean final performance over 5 seeds and 10 trajectories.

Method	**Modality**	**Episodes**	**Cartpole Swing Up**	**Reacher Easy**	**Cheetah Run**	**Finger Spin**	**Cup Catch**	**Walker Walk**
A3C	proprioceptive	100,000	558	285	214	129	105	311
D4PG	pixels	100,000	862	967	524	985	980	968
PlaNet (ours)	pixels	1,000	821	832	662	700	930	951
CEM + true simulator	simulator state	0	850	964	656	825	993	994
Data efficiency gain PlaNet over D4PG (factor)			250	40	500+	300	100	90

Table 1 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

PlaNet – Ablations

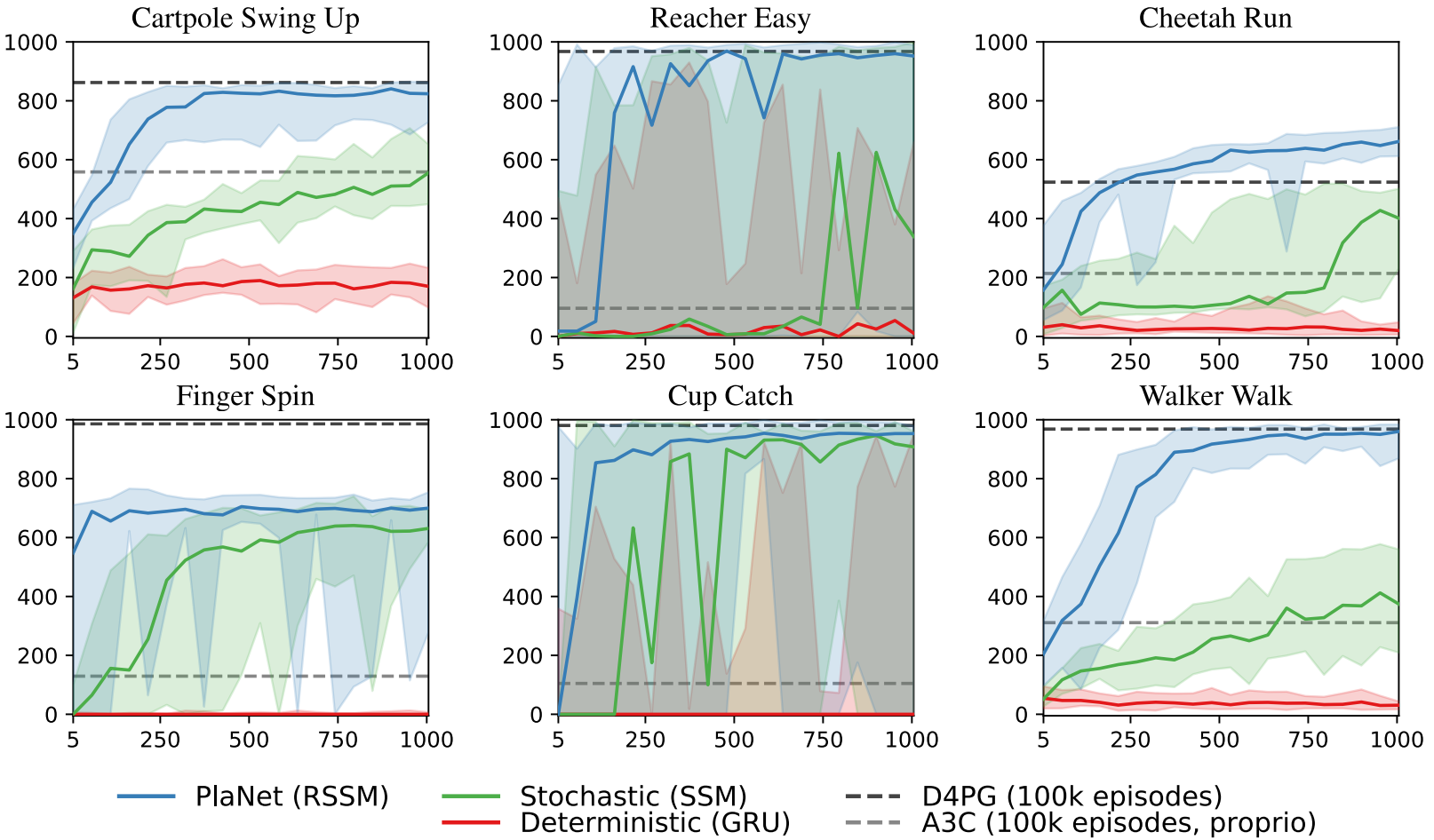


Figure 4 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

PlaNet – Ablations

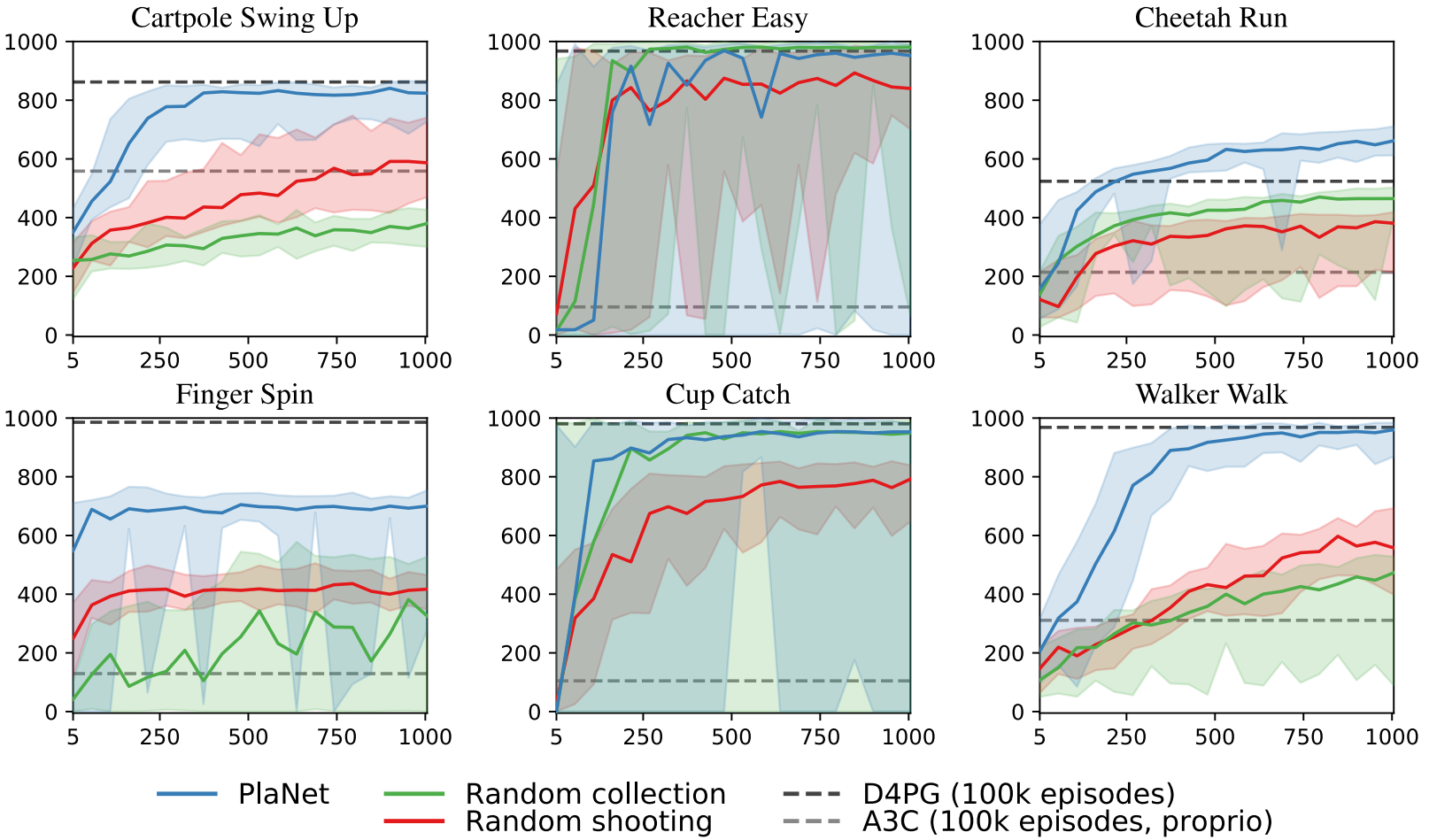


Figure 5 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

Random collection: random actions; random shooting: best action out of 1000 random seqs.

Discrete Latent Variables

Consider that we would like to have discrete neurons on the hidden layer of a neural network.

Note that on the output layer, we relaxed discrete prediction (i.e., an $\arg\max$) with a continuous relaxation – $\mathrm{softmax}$. This way, we can compute the derivatives and also predict the most probable class. (It is possible to derive $\mathrm{softmax}$ as an entropy-regularized $\arg\max$.)

However, on a hidden layer, we also need to *sample* from the predicted categorical distribution, and then backpropagate the gradients.

Stochastic Gradient Estimators

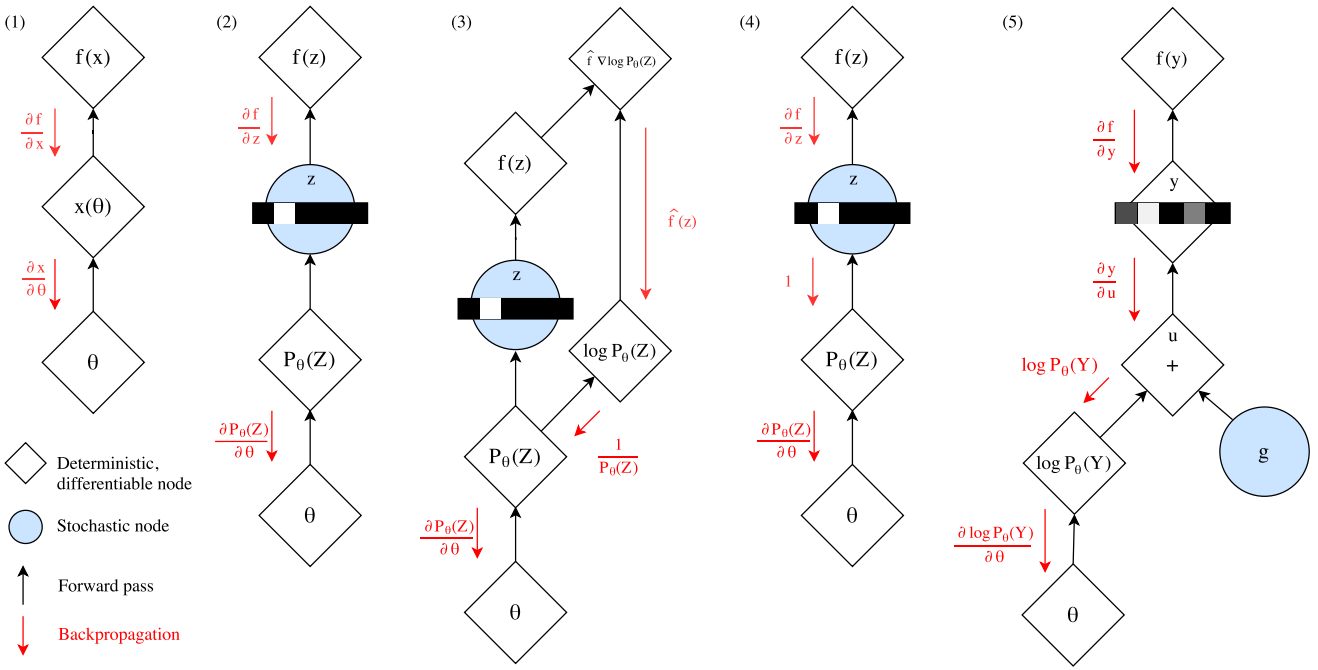


Figure 2: Gradient estimation in stochastic computation graphs. (1) $\nabla_\theta f(x)$ can be computed via backpropagation if $x(\theta)$ is deterministic and differentiable. (2) The presence of stochastic node z precludes backpropagation as the sampler function does not have a well-defined gradient. (3) The score function estimator and its variants (NVIL, DARN, MuProp, VIMCO) obtain an unbiased estimate of $\nabla_\theta f(x)$ by backpropagating along a surrogate loss $\hat{f} \log p_\theta(z)$, where $\hat{f} = f(x) - b$ and b is a baseline for variance reduction. (4) The Straight-Through estimator, developed primarily for Bernoulli variables, approximates $\nabla_\theta z \approx 1$. (5) Gumbel-Softmax is a path derivative estimator for a continuous distribution y that approximates z. Reparameterization allows gradients to flow from $f(y)$ to θ. y can be annealed to one-hot categorical variables over the course of training.

Figure 2 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

Stochastic Gradient Estimators

Consider a model with a discrete categorical latent variable $\boldsymbol{z}$ sampled from $p(\boldsymbol{z};\boldsymbol{\theta})$, with a loss $L(\boldsymbol{z};\boldsymbol{\omega})$. Several gradient estimators have been proposed:

- A REINFORCE-like gradient estimation.

  Using the identity $\nabla_{\boldsymbol{\theta}} p(\boldsymbol{z};\boldsymbol{\theta}) = p(\boldsymbol{z};\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{z};\boldsymbol{\theta})$, we obtain that

  $$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{z}}\left[L(\boldsymbol{z};\boldsymbol{\omega})\right] = \mathbb{E}_{\boldsymbol{z}}\left[L(\boldsymbol{z};\boldsymbol{\omega})\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{z};\boldsymbol{\theta})\right].$$

  Analogously as before, we can also include the baseline for variance reduction, resulting in

  $$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{z}}\left[L(\boldsymbol{z};\boldsymbol{\omega})\right] = \mathbb{E}_{\boldsymbol{z}}\left[(L(\boldsymbol{z};\boldsymbol{\omega}) - b)\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{z};\boldsymbol{\theta})\right].$$

- A **straight-through (ST)** estimator.

  The straight-through estimator has been proposed by Y. Bengio in 2013. It is a biased estimator, which assumes that $\nabla_{\boldsymbol{\theta}} \boldsymbol{z} \approx \nabla_{\boldsymbol{\theta}} p(\boldsymbol{z};\boldsymbol{\theta})$, which implies $\nabla_{p(\boldsymbol{z};\boldsymbol{\theta})} \boldsymbol{z} \approx 1$. Even if the bias can be considerable, it seems to work quite well in practice.

Gumbel-Softmax

The **Gumbel-softmax** distribution was proposed independently in two papers in Nov 2016 (under the name of **Concrete** distribution in the other paper).

It is a continuous distribution over the simplex (over categorical distributions) that can approximate *sampling* from a categorical distribution.

Let z be a categorical variable with class probabilities $\boldsymbol{p} = (p_1, p_2, \ldots, p_K)$.

Recall that the Gumbel-Max trick (based on a 1954 theorem from E. J. Gumbel) states that we can draw samples $z \sim \boldsymbol{p}$ using

$$z = \text{one-hot}\left(\arg\max_i \left(g_i + \log p_i\right)\right),$$

where g_i are independent samples drawn from the $\text{Gumbel}(0, 1)$ distribution.

To sample g from the distribution $\text{Gumbel}(0, 1)$, we can sample $u \sim U(0, 1)$ and then compute $g = -\log(-\log u)$.

Gumbel-Softmax

To obtain a continuous distribution, we relax the $\arg\max$ into a $\mathrm{softmax}$ with temperature T as

$$z_i = \frac{e^{(g_i + \log p_i)/T}}{\sum_j e^{(g_j + \log p_j)/T}}.$$

As the temperature T goes to zero, the generated samples become one-hot, and therefore the Gumbel-softmax distribution converges to the categorical distribution $p(z)$.

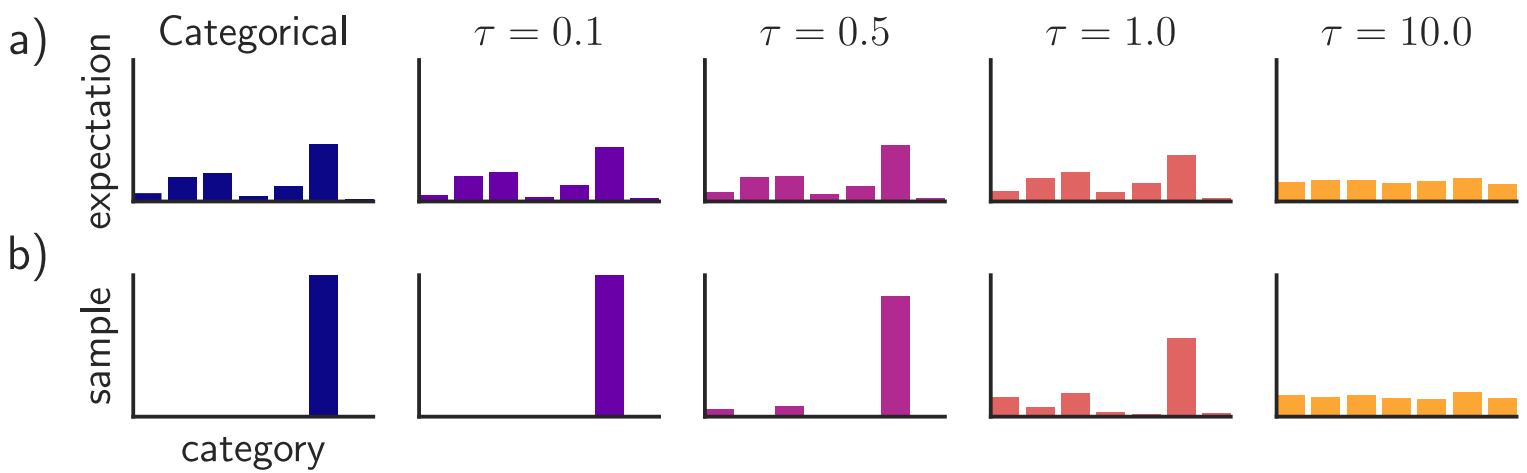


Figure 1 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

Gumbel-Softmax Estimator

The Gumbel-softmax distribution can be used to reparametrize the sampling of the discrete variable using a fully differentiable estimator.

However, the resulting sample is not discrete, it only converges to a discrete sample as the temperature T goes to zero.

If it is a problem, we can combine the Gumbel-softmax with a straight-through estimator, obtaining ST Gumbel-softmax, where we:

- discretize $\boldsymbol{y}$ as $\boldsymbol{z} = \arg\max \boldsymbol{y}$,
- assume $\nabla_{\boldsymbol{\theta}} \boldsymbol{z} \approx \nabla_{\boldsymbol{\theta}} \boldsymbol{y}$, or in other words, $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \approx 1$.

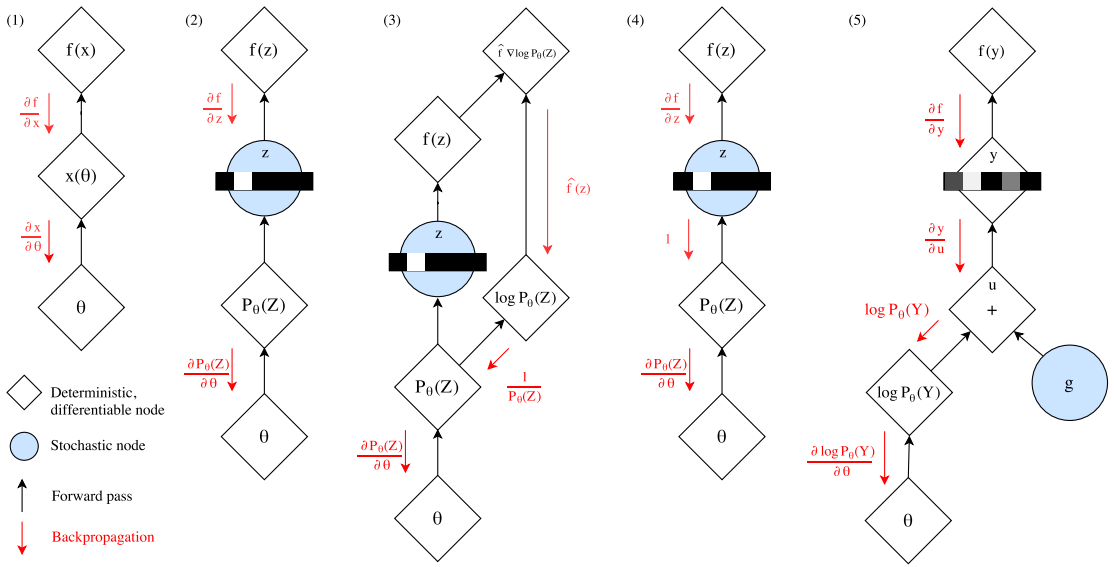


Figure 2: Gradient estimation in stochastic computation graphs. (1) $\nabla_\theta f(x)$ can be computed via backpropagation if $x(\theta)$ is deterministic and differentiable. (2) The presence of stochastic node z precludes backpropagation as the sampler function does not have a well-defined gradient. (3) The score function estimator and its variants (NVIL, DARN, MuProp, VIMCO) obtain an unbiased estimate of $\nabla_\theta f(x)$ by backpropagating along a surrogate loss $\hat{f} \log p_\theta(z)$, where $\hat{f} = f(x) - b$ and b is a baseline for variance reduction. (4) The Straight-Through estimator, developed primarily for Bernoulli variables, approximates $\nabla_\theta z \approx 1$. (5) Gumbel-Softmax is a path derivative estimator for a continuous distribution y that approximates z. Reparameterization allows gradients to flow from $f(y)$ to θ. y can be annealed to one-hot categorical variables over the course of training.

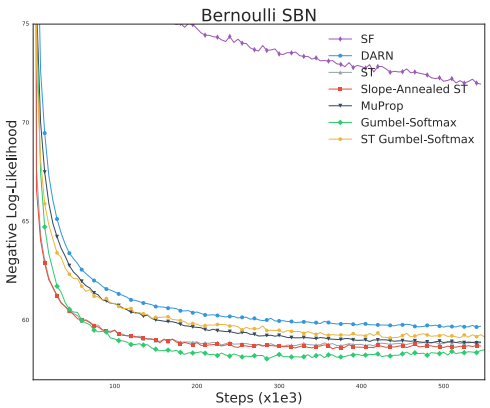
Figure 2 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

Gumbel-Softmax Estimator Results

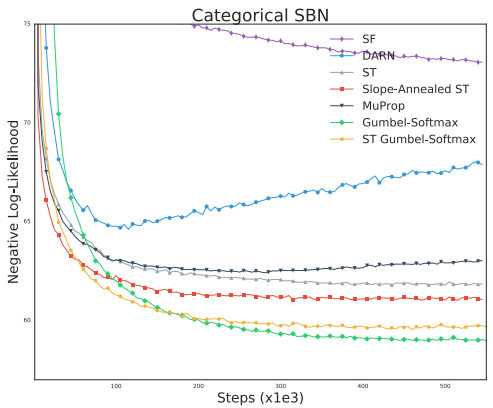
Table 1: The Gumbel-Softmax estimator outperforms other estimators on Bernoulli and Categorical latent variables. For the structured output prediction (SBN) task, numbers correspond to negative log-likelihoods (nats) of input images (lower is better). For the VAE task, numbers correspond to negative variational lower bounds (nats) on the log-likelihood (lower is better).

	SF	DARN	MuProp	ST	Annealed ST	Gumbel-S.	ST Gumbel-S.
SBN (Bern.)	72.0	59.7	58.9	58.9	58.7	**58.5**	59.3
SBN (Cat.)	73.1	67.9	63.0	61.8	61.1	**59.0**	59.7
VAE (Bern.)	112.2	110.9	109.7	116.0	111.5	**105.0**	111.5
VAE (Cat.)	110.6	128.8	107.0	110.9	107.8	**101.5**	107.8

Table 1 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144



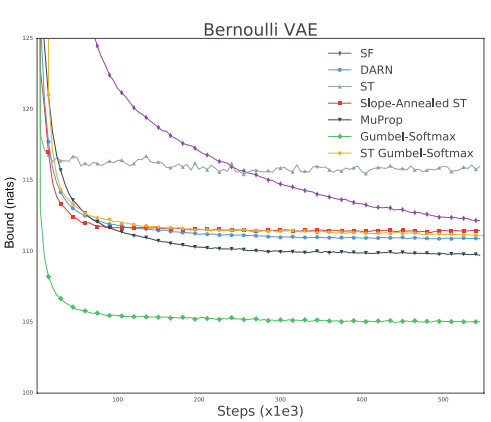
(a)



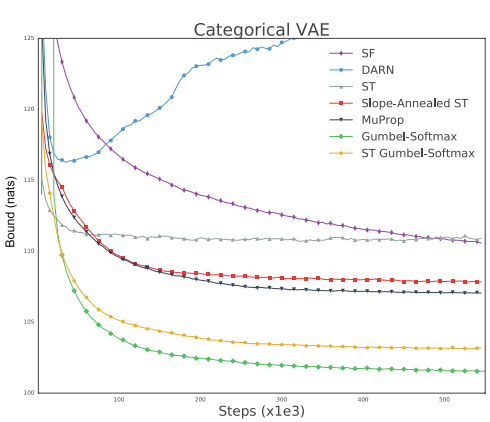
(b)

Figure 3: Test loss (negative log-likelihood) on the structured output prediction task with binarized MNIST using a stochastic binary network with (a) Bernoulli latent variables (392-200-200-392) and (b) categorical latent variables (392-(20×10)-(20×10)-392).

Figure 3 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144



(a)



(b)

Figure 4: Test loss (negative variational lower bound) on binarized MNIST VAE with (a) Bernoulli latent variables $(784 - 200 - 784)$ and (b) categorical latent variables $(784 - (20 \times 10) - 200)$.

Figure 4 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

Applications of Discrete Latent Variables

The discrete latent variables can be used among others to:

- allow the SAC algorithm to be used on **discrete** actions, using either Gumbel-softmax relaxation (if the critic takes the actions as binary indicators, it is possible to pass not just one-hot encoding, but the result of Gumbel-softmax directly), or a straight-through estimator;
- model images using discrete latent variables
  - VQ-VAE, VQ-VAE-2 use "codebook loss" with a straight-through estimator

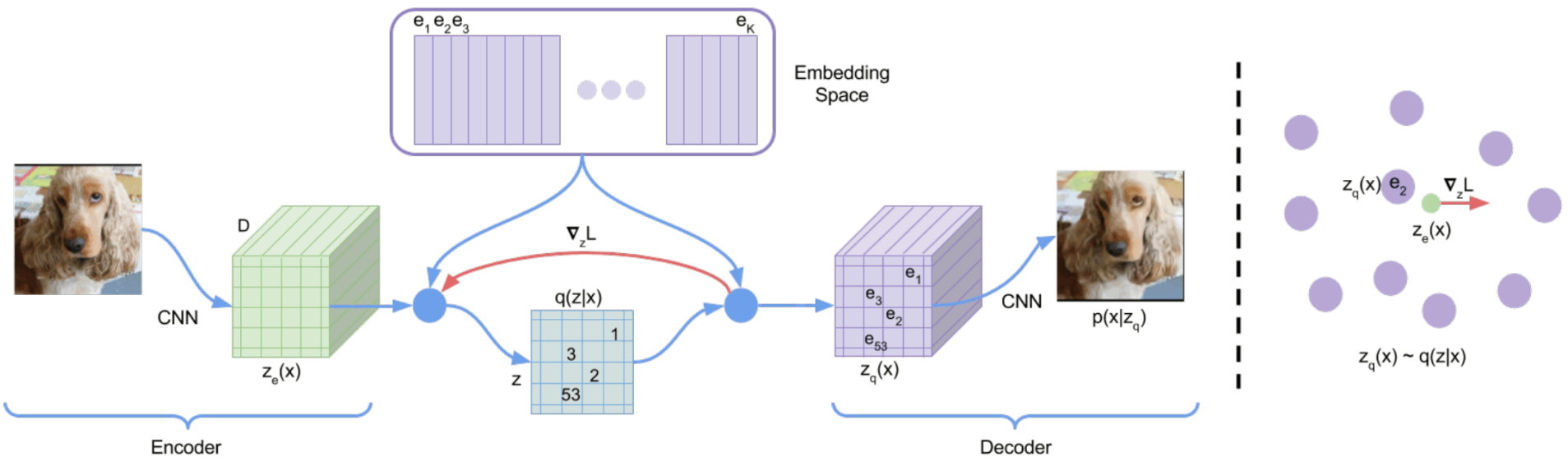


Figure 1 of "Neural Discrete Representation Learning", https://arxiv.org/abs/1711.00937

- VQ-GAN combines the VQ-VAE and Transformers, where the latter is used to generate a sequence of the *discrete* latents.

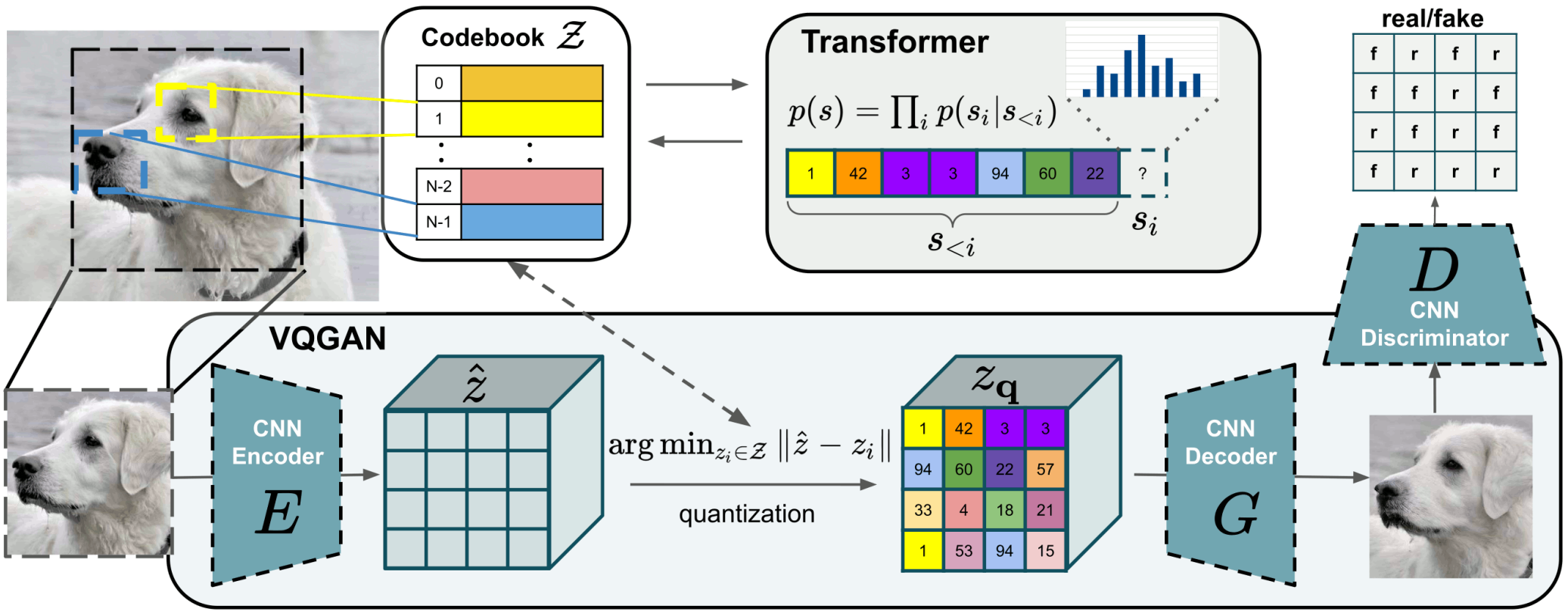


Figure 2 of "Taming Transformers for High-Resolution Image Synthesis", https://arxiv.org/abs/2012.09841

Applications of Discrete Latent Variables – VQ-GAN

Applications of Discrete Latent Variables – DALL-E

- In DALL-E, Transformer is used to model a sequence of words followed by a sequence of the discrete image latent variables.

  The Gumbel-softmax relaxation is used to train the discrete latent states, with temperature annealed with a cosine decay from 1 to 1/16 over the first 150k (out of 3M) updates.

(a) a tapir made of accordion. a tapir with the texture of an accordion.

(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

(c) a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

(d) the exact same cat on the top as a sketch on the bottom

Figure 2 of "Zero-Shot Text-to-Image Generation", https://arxiv.org/abs/2102.12092

DreamerV2

The PlaNet model was followed by Dreamer (Dec 2019) and DreamerV2 (Oct 2020), which train an agent using reinforcement learning using the model alone. After 200M environment steps, it surpasses Rainbow on a collection of 55 Atari games (the authors do not mention why they do not use all 57 games) when training on a single GPU for 10 days per game.

During training, a policy is learned from 486B compact states "dreamed" by the model, which is 10,000 times more than the 50M observations from the real environment (with action repeat 4).

Interestingly, the latent states are represented as a vector of several **categorical** variables – 32 variables with 32 classes each are utilized in the paper.

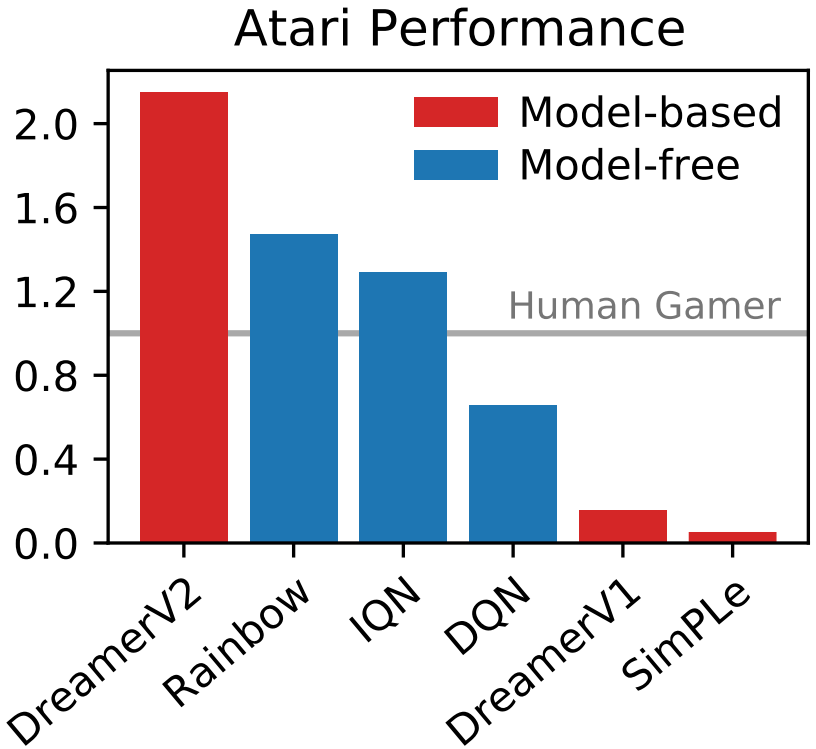


Figure 1 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

DreamerV2 – Model Learning

The model in DreamerV2 is learned using the RSSM, collecting agent experiences of observations, actions, rewards, and discount factors (0.995 within episode and 0 at an episode end). Training is performed on batches of 50 sequences of length at most 50 each.

$$\begin{aligned}
\text{recurrent model:} \quad & h_t = f_\varphi(h_{t-1}, s_{t-1}, a_{t-1}), \\
\text{representation model:} \quad & s_t \sim q_\varphi(s_t | h_t, x_t), \\
\text{transition predictor:} \quad & \bar{s}_t \sim p_\varphi(\bar{s}_t | h_t), \\
\text{image predictor:} \quad & \bar{x}_t \sim p_\varphi(\bar{x}_t | h_t, s_t), \\
\text{reward predictor:} \quad & \bar{r}_t \sim p_\varphi(\bar{r}_t | h_t, s_t), \\
\text{discount predictor:} \quad & \bar{\gamma}_t \sim p_\varphi(\bar{\gamma}_t | h_t, s_t).
\end{aligned}$$

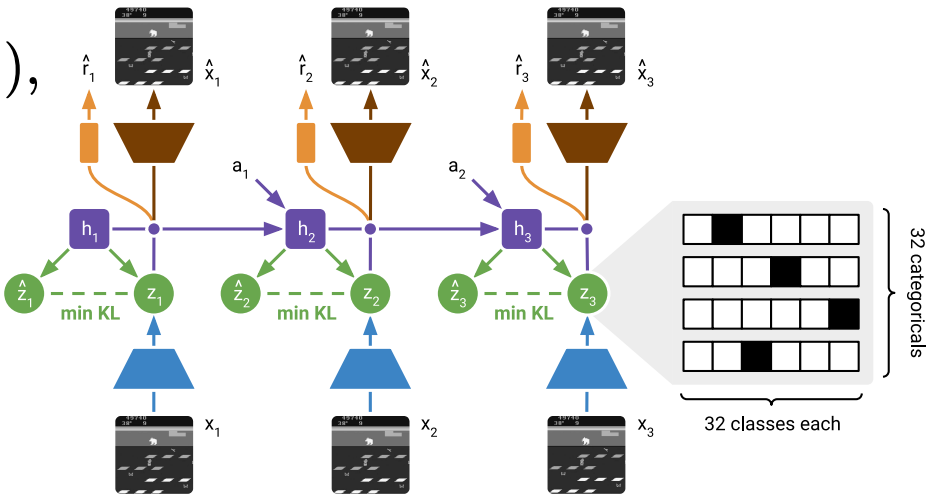


Figure 2 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

Algorithm 1: Straight-Through Gradients with Automatic Differentiation

```
sample = one_hot(draw(logits))              # sample has no gradient
probs  = softmax(logits)                    # want gradient of this
sample = sample + probs - stop_grad(probs)  # has gradient of probs
```

Algorithm 1 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

DreamerV2 – Model Learning

The following loss function is used:

$$\mathcal{L}(\varphi) = \mathbb{E}_{q_\varphi(s_{1:T}|a_{1:T},x_{1:T})}\Bigg[\sum_{t=1}^{T} \underbrace{-\log p_\varphi(x_t|h_t,s_t)}_{\text{image log loss}} \underbrace{-\log p_\varphi(r_t|h_t,s_t)}_{\text{reward log loss}} \underbrace{-\log p_\varphi(\gamma_t|h_t,s_t)}_{\text{discount log loss}} \underbrace{+\beta D_{\mathrm{KL}}\left[q_\varphi(s_t|h_t,x_t)\|p_\varphi(s_t|h_t)\right]}_{\text{KL loss}}\Bigg].$$

In the KL term, we train both the prior and the encoder. However, regularizing the encoder towards the prior makes training harder (especially at the beginning), so the authors propose **KL balancing**, minimizing the KL term faster for the prior ($\alpha = 0.8$) than for the posterior.

Algorithm 2: KL Balancing with Automatic Differentiation

```
kl_loss =       alpha  * compute_kl(stop_grad(approx_posterior), prior)
        + (1 - alpha) * compute_kl(approx_posterior, stop_grad(prior))
```

Algorithm 2 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

DreamerV2 – Policy Learning

The policy is trained solely from the model, starting from the encountered posterior states and then considering $H = 15$ actions simulated in the compact latent state.

We train an actor predicting $\pi_\psi(a_t|s_t)$ and a critic predicting

$$v_\xi(s_t) = \mathbb{E}_{p_\varphi, \pi_\psi}\left[\sum_{r \geq t}\left(\prod_{r'=t+1}^{r} \gamma_{r'}\right) r_t\right].$$

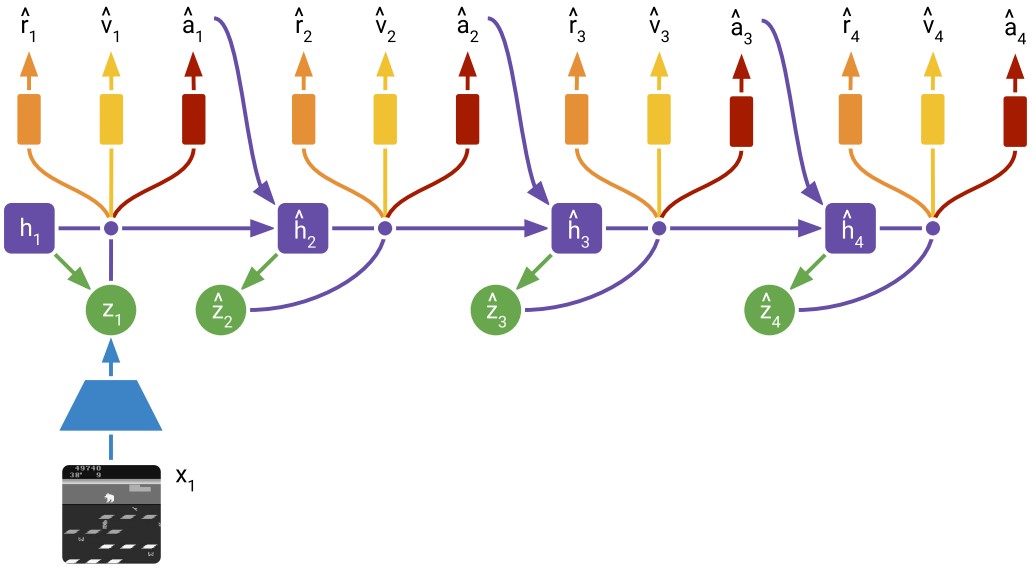


Figure 3 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

The critic is trained by estimating the truncated λ-return as

$$V_t^\lambda = r_t + \gamma_t \begin{cases} (1-\lambda) v_\xi(\hat{z}_{t+1}) + \lambda V_{t+1}^\lambda & \text{if } t < H, \\ v_\xi(\hat{z}_H) & \text{if } t = H. \end{cases}$$

and then minimizing the MSE.

DreamerV2 – Policy Learning

The actor is trained using two approaches:

- the REINFORCE-like loss (with a baseline), which is unbiased, but has a high variance (even with the baseline);
- the reparametrization of discrete actions using a straight-through gradient estimation, which is biased, but has lower variance.

$$\mathcal{L}(\psi) = \mathbb{E}_{p_\varphi, \pi_\psi}\Big[\sum_{t=1}^{H-1}\Big(\underbrace{-\rho \log \pi_\psi(a_t|s_t)\,\mathrm{stop\_gradient}(V_t^\lambda - v_\xi(s_t))}_{\text{reinforce}} \underbrace{-(1-\rho)V_t^\lambda}_{\text{dynamics backprop}} \underbrace{-\eta H(a_t|s_t)}_{\text{entropy regularizer}}\Big)\Big]$$

For Atari domains, authors use $\rho = 1$ and $\eta = 10^{-3}$ (they say it works "substantially better"), while for continuous actions, $\rho = 0$ works "substantially better" (presumably because of the bias in case of discrete actions) and $\eta = 10^{-4}$ is used.

DreamerV2 – Results

The authors evaluate on 55 Atari games. They argue that the commonly used metrics have various flaws:

- **gamer-normalized median** ignores scores on half of the games,
- **gamer-normalized mean** is dominated by several games where the agent achieves super-human performance by several orders.

They therefore propose two additional ones:

- **record-normalized mean** normalizes with respect to any registered human world record for each game; however, in some games the agents still achieve super-human-record performance;
- **clipped record-normalized mean** additionally clips each score to 1; this measure is used as the primary metric in the paper.

DreamerV2 – Results

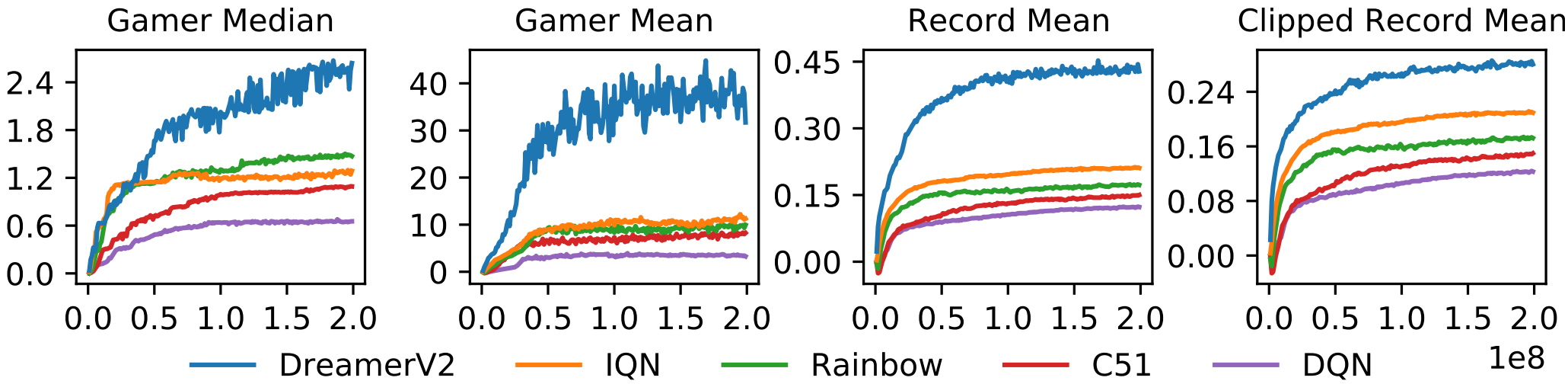


Figure 4 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

Agent	Gamer Median	Gamer Mean	Record Mean	Clipped Record Mean
DreamerV2	2.15	**42.26**	**0.44**	**0.28**
DreamerV2 (schedules)	**2.64**	31.71	0.43	**0.28**
IMPALA	1.92	16.72	0.34	0.23
IQN	1.29	11.27	0.21	0.21
Rainbow	1.47	9.95	0.17	0.17
C51	1.09	8.25	0.15	0.15
DQN	0.65	3.28	0.12	0.12

Table 1 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

Scheduling anneals actor gradient mixing ρ (from 0.1 to 0), entropy loss scale, KL, lr.

DreamerV2 – Ablations

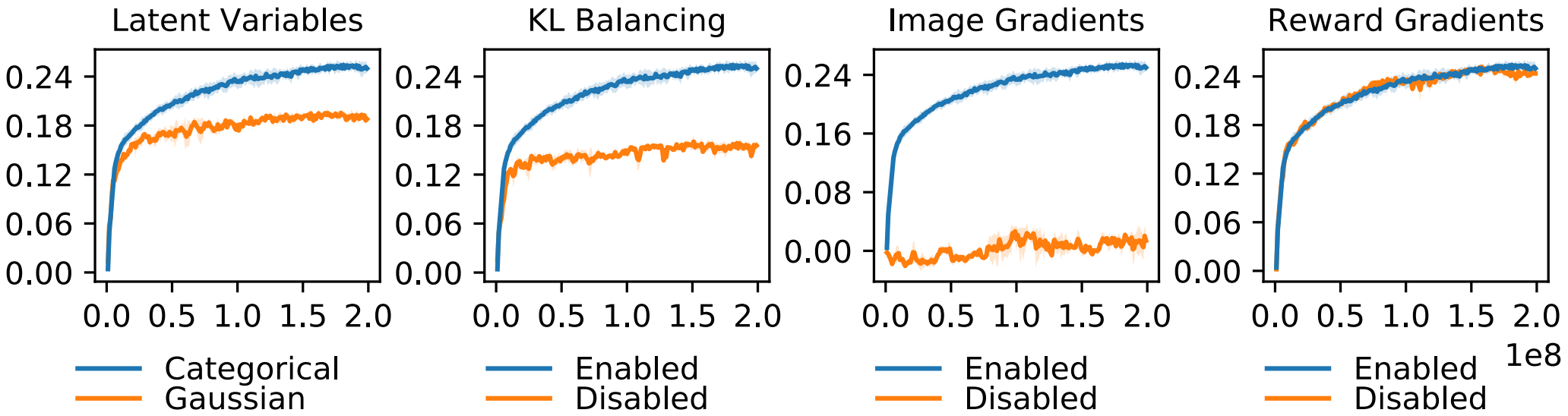


Figure 5 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

Agent	Gamer Median	Gamer Mean	Record Mean	Clipped Record Mean
DreamerV2	1.64	13.39	0.36	0.25
No Layer Norm	1.66	11.29	0.38	0.25
No Reward Gradients	1.68	14.29	0.37	0.24
No Discrete Latents	0.85	3.96	0.24	0.19
No KL Balancing	0.87	4.25	0.19	0.16
No Policy Reinforce	0.72	5.10	0.16	0.15
No Image Gradients	0.05	0.37	0.01	0.01

Table 2 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

DreamerV2 – Discrete Latent Variables

Categorical latent variables outperform Gaussian latent variables on 42 games, tie on 5 games and decrease performance on 8 games (where a tie is defined as being within 5%).

The authors provide several hypotheses why could the categorical latent variables be better:

- Categorical prior can perfectly match aggregated posterior, because mixture of categoricals is categorical, which is not true for Gaussians.
- Sparsity achieved by the 32 categorical variables with 32 classes each could be beneficial for generalization.
- Contrary to intuition, optimizing categorical variables might be easier than optimizing Gaussians, because the straight-through estimator ignores a term which would otherwise scale the gradient, which could reduce exploding/vanishing gradient problem.
- Categorical variables could be a better match for modeling discrete aspect of the Atari games (defeating an enemy, collecting reward, entering a room, ...).

DreamerV2 – Comparison, Hyperparametres

Algorithm	Reward Modeling	Image Modeling	Latent Transitions	Single GPU	Trainable Parameters	Atari Frames	Accelerator Days
DreamerV2	✓	✓	✓	✓	22M	200M	10
SimPLe	✓	✓	✗	✓	74M	4M	40
MuZero	✓	✗	✓	✗	40M	20B	80
MuZero Reanalyze	✓	✗	✓	✗	40M	200M	80

Table 2 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

World Model			Behavior			Common		
Dataset size (FIFO)	—	$2 \cdot 10^6$	Imagination horizon	H	15	Environment steps per update	—	4
Batch size	B	50	Discount	γ	0.995	MPL number of layers	—	4
Sequence length	L	50	λ-target parameter	λ	0.95	MPL number of units	—	400
Discrete latent dimensions	—	32	Actor gradient mixing	ρ	1	Gradient clipping	—	100
Discrete latent classes	—	32	Actor entropy loss scale	η	$1 \cdot 10^{-3}$	Adam epsilon	ϵ	10^{-5}
RSSM number of units	—	600	Actor learning rate	—	$4 \cdot 10^{-5}$	Weight decay (decoupled)	—	10^{-6}
KL loss scale	β	0.1	Critic learning rate	—	$1 \cdot 10^{-4}$			
KL balancing	α	0.8	Slow critic update interval	—	100			
World model learning rate	—	$2 \cdot 10^{-4}$						
Reward transformation	—	tanh						

Table D.1 of "Mastering Atari with Discrete World Models", https://arxiv.org/abs/2010.02193

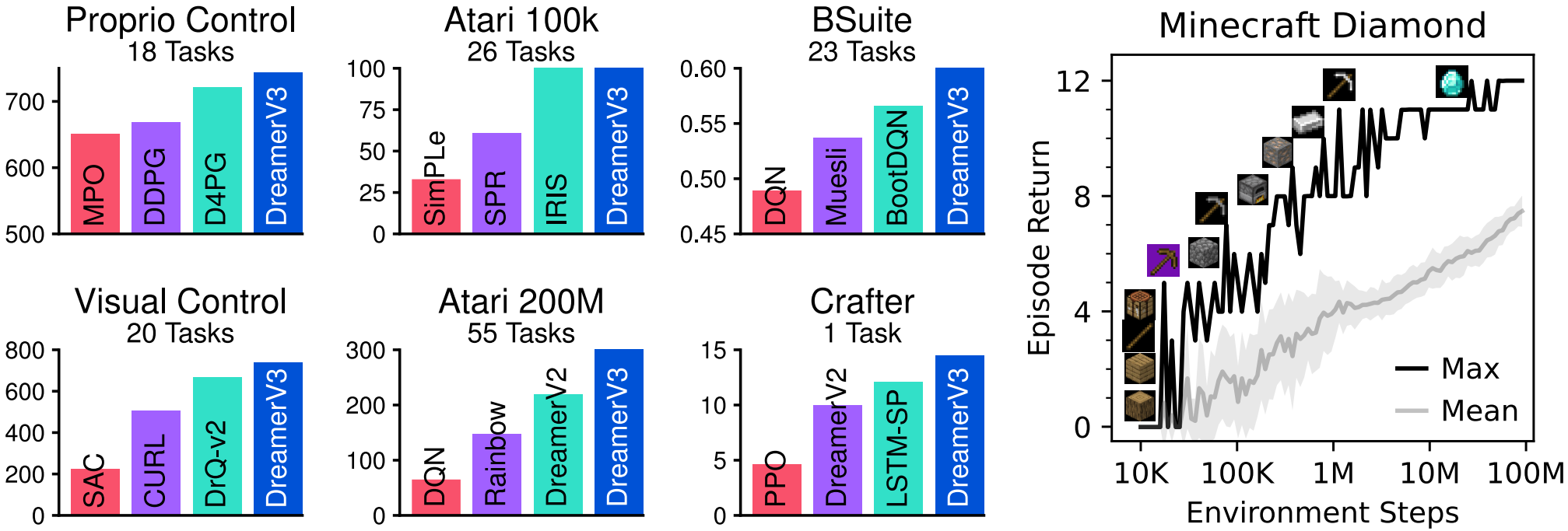
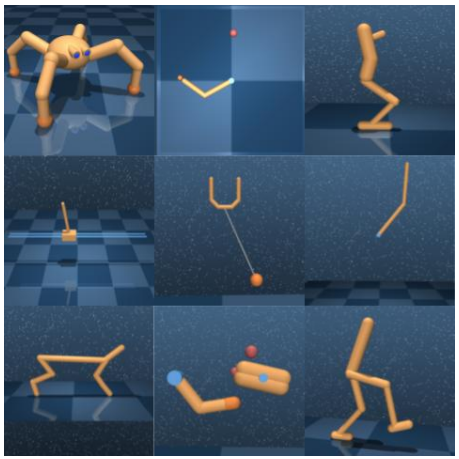


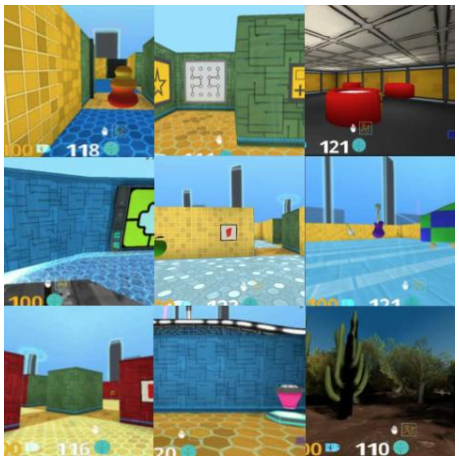
Figure 1: Using the same hyperparameters across all domains, DreamerV3 outperforms specialized model-free and model-based algorithms in a wide range of benchmarks and data-efficiency regimes. Applied out of the box, DreamerV3 also learns to obtain diamonds in the popular video game Minecraft from scratch given sparse rewards, a long-standing challenge in artificial intelligence for which previous approaches required human data or domain-specific heuristics.

Figure 1 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1



(a) Control Suite

(b) Atari

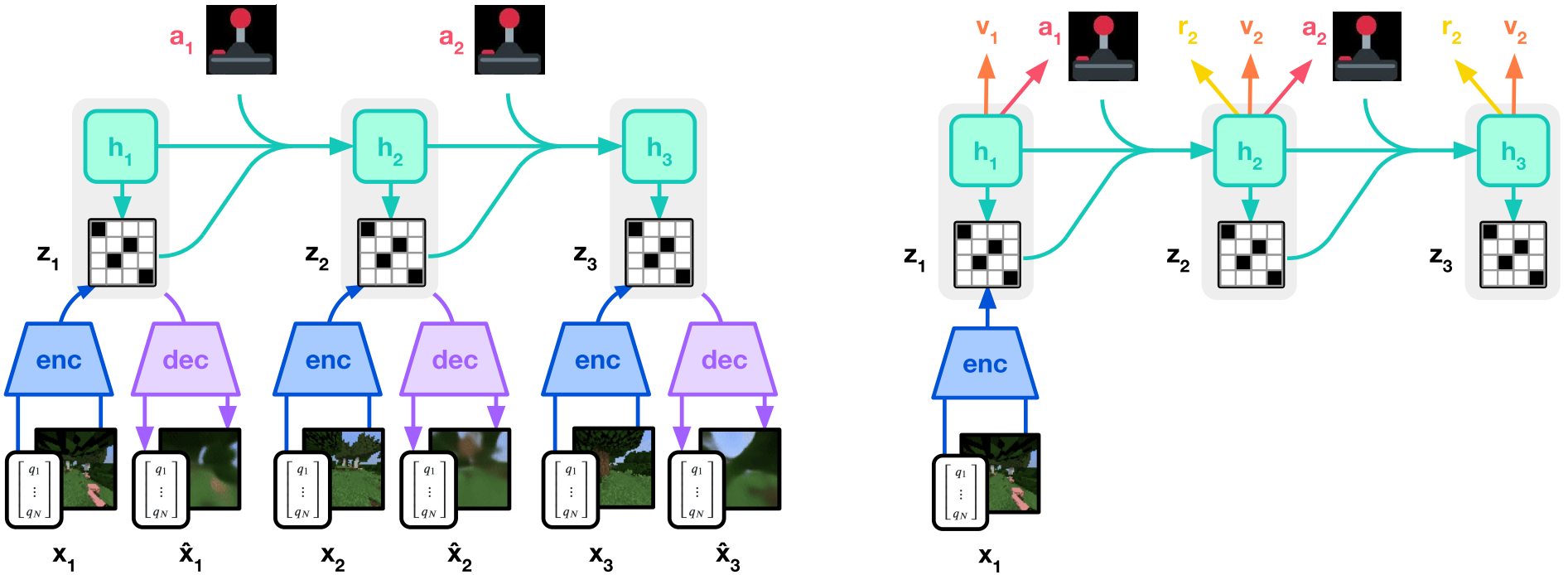


(c) DMLab

(d) Minecraft

Figure 2: Four visual domains considered in this work. DreamerV3 succeeds across these diverse domains, ranging from robot locomotion and manipulation tasks over Atari games with 2D graphics to complex 3D domains such as DMLab and Minecraft that require spatial and temporal reasoning.

Figure 2 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1



(a) World Model Learning

(b) Actor Critic Learning

Figure 3: Training process of DreamerV3. The world model encodes sensory inputs into a discrete representation z_t that is predicted by a sequence model with recurrent state h_t given actions a_t. The inputs are reconstructed as learning signal to shape the representations. The actor and critic learn from trajectories of abstract representations predicted by the world model.

Figure 3 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

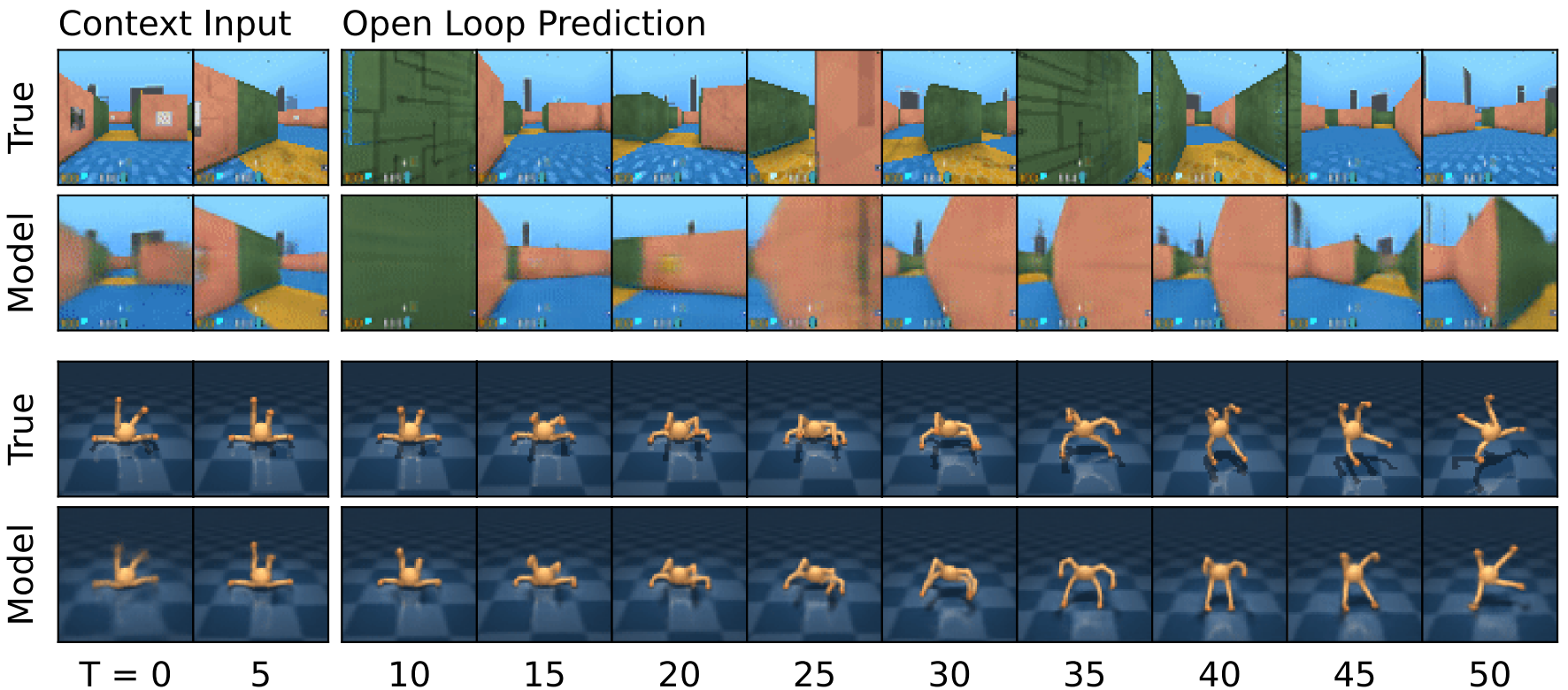


Figure 5: Multi-step video predictions in DMLab (top) and Control Suite (bottom). From 5 frames of context input, the model predicts 45 steps into the future given the action sequence and without access to intermediate images. The world model learns an understanding of the underlying 3D structure of the two environments. Refer to Appendix H for additional video predictions.

Figure 5 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

$$\operatorname{symlog}(x) \stackrel{\text{def}}{=} \operatorname{sign}(x) \log\left(|x| + 1\right),$$
$$\operatorname{symexp}(x) \stackrel{\text{def}}{=} \operatorname{sign}(x)\left(\exp(|x|) - 1\right).$$

$$\mathcal{L}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \tfrac{1}{2}\left(f(\boldsymbol{x};\boldsymbol{\theta}) - \operatorname{symlog}(y)\right)^2$$

$$\hat{y} \stackrel{\text{def}}{=} \operatorname{symexp}\left(f(\boldsymbol{x};\boldsymbol{\theta})\right)$$

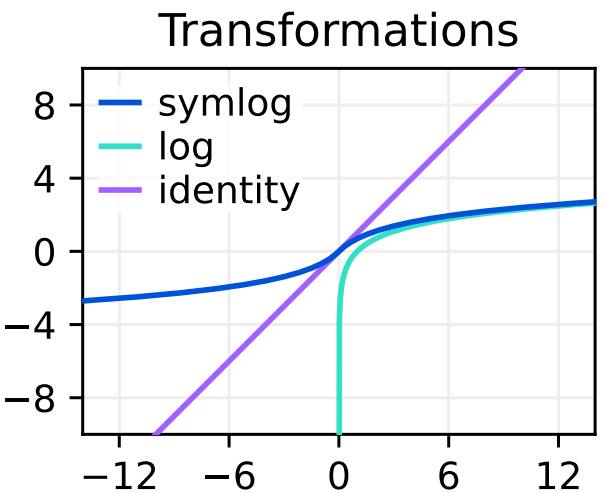


Figure 4: The symlog function compared to logarithm and identity.

Figure 4 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

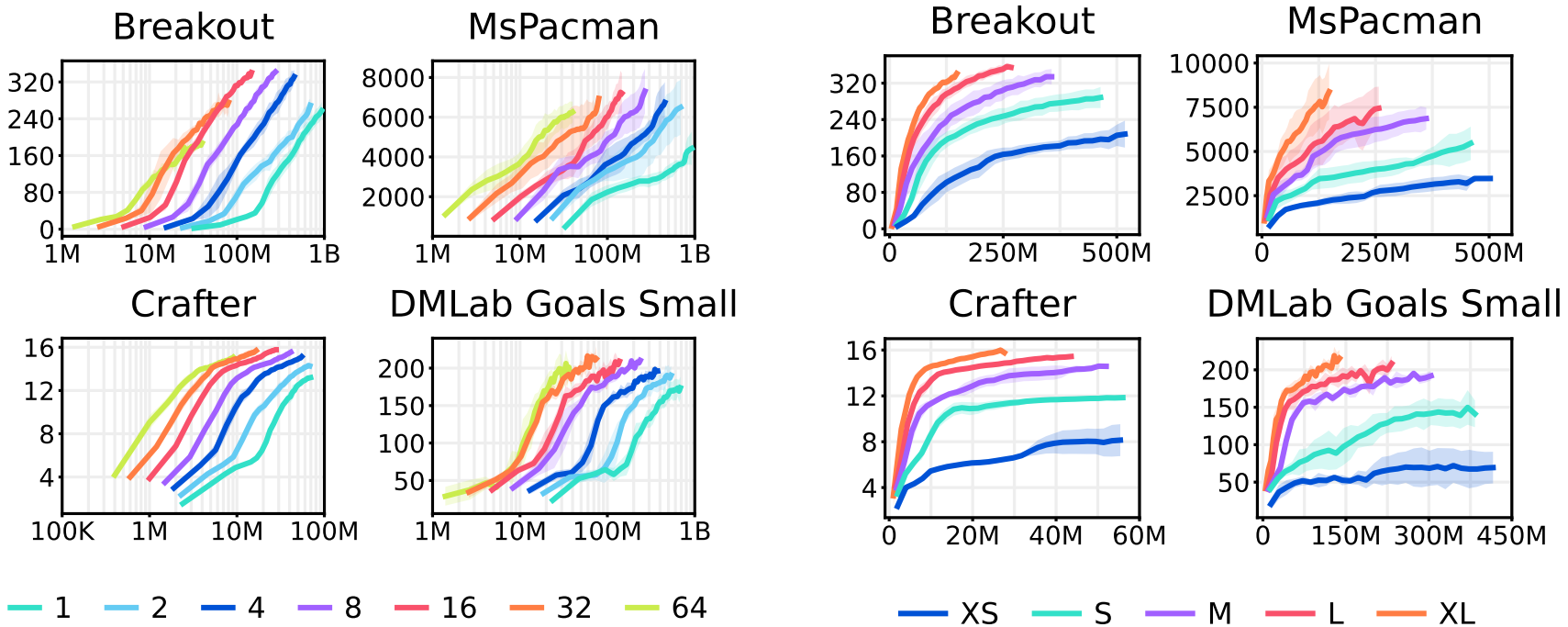
$$
\begin{aligned}
&\text{RSSM sequence model:} && h_t = f_\varphi(h_{t-1}, s_{t-1}, a_{t-1}),\\
&\text{RSSM encoder:} && s_t \sim q_\varphi(s_t|h_t, x_t),\\
&\text{RSSM dynamics predictor:} && \bar{s}_t \sim p_\varphi(\bar{s}_t|h_t),\\
&\text{decoder:} && \bar{x}_t \sim p_\varphi(\bar{x}_t|h_t, s_t),\\
&\text{reward predictor:} && \bar{r}_t \sim p_\varphi(\bar{r}_t|h_t, s_t),\\
&\text{continue predictor:} && \bar{\gamma}_t \sim p_\varphi(\bar{\gamma}_t|h_t, s_t).
\end{aligned}
$$

$$\mathcal{L}(\boldsymbol{\varphi}) \stackrel{\text{def}}{=} \mathbb{E}_{q_{\boldsymbol{\varphi}}}\Big[\sum_{t=1}^{T}\big(\underbrace{\beta_{\text{pred}}}_{1.0}\mathcal{L}_{\text{pred}}(\boldsymbol{\varphi}) + \underbrace{\beta_{\text{dyn}}}_{0.5}\mathcal{L}_{\text{dyn}}(\boldsymbol{\varphi}) + \underbrace{\beta_{\text{rep}}}_{0.1}\mathcal{L}_{\text{rep}}(\boldsymbol{\varphi})\big)\Big],$$

$$\mathcal{L}_{\text{pred}}(\boldsymbol{\varphi}) \stackrel{\text{def}}{=} -\log p_{\boldsymbol{\varphi}}(x_t|h_t,s_t) - \log p_{\boldsymbol{\varphi}}(r_t|h_t,s_t) - \log p_{\boldsymbol{\varphi}}(\gamma_t|h_t,s_t),$$

$$\mathcal{L}_{\text{dyn}}(\boldsymbol{\varphi}) \stackrel{\text{def}}{=} \max\Big(1, D_{\text{KL}}\big(\,\text{sg}(q_{\boldsymbol{\theta}}(s_t|h_t,x_t))\,\big\|\,p_{\boldsymbol{\varphi}}(s_t|h_t)\,\big)\Big),$$

$$\mathcal{L}_{\text{rep}}(\boldsymbol{\varphi}) \stackrel{\text{def}}{=} \max\Big(1, D_{\text{KL}}\big(\,q_{\boldsymbol{\theta}}(s_t|h_t,x_t)\,\big\|\,\text{sg}(p_{\boldsymbol{\varphi}}(s_t|h_t))\big)\Big).$$



(a) Training Ratio

(b) Model Size

Figure 6: Scaling properties of DreamerV3. The graphs show task performance over environment steps for different training ratios and model sizes reaching from 8M to 200M parameters. The training ratio is the ratio of replayed steps to environment steps. The model sizes are detailed in Table B.1. Higher training ratios result in substantially improved data-efficiency. Notably, larger models achieve not only higher final performance but also higher data-efficiency.

Figure 6 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

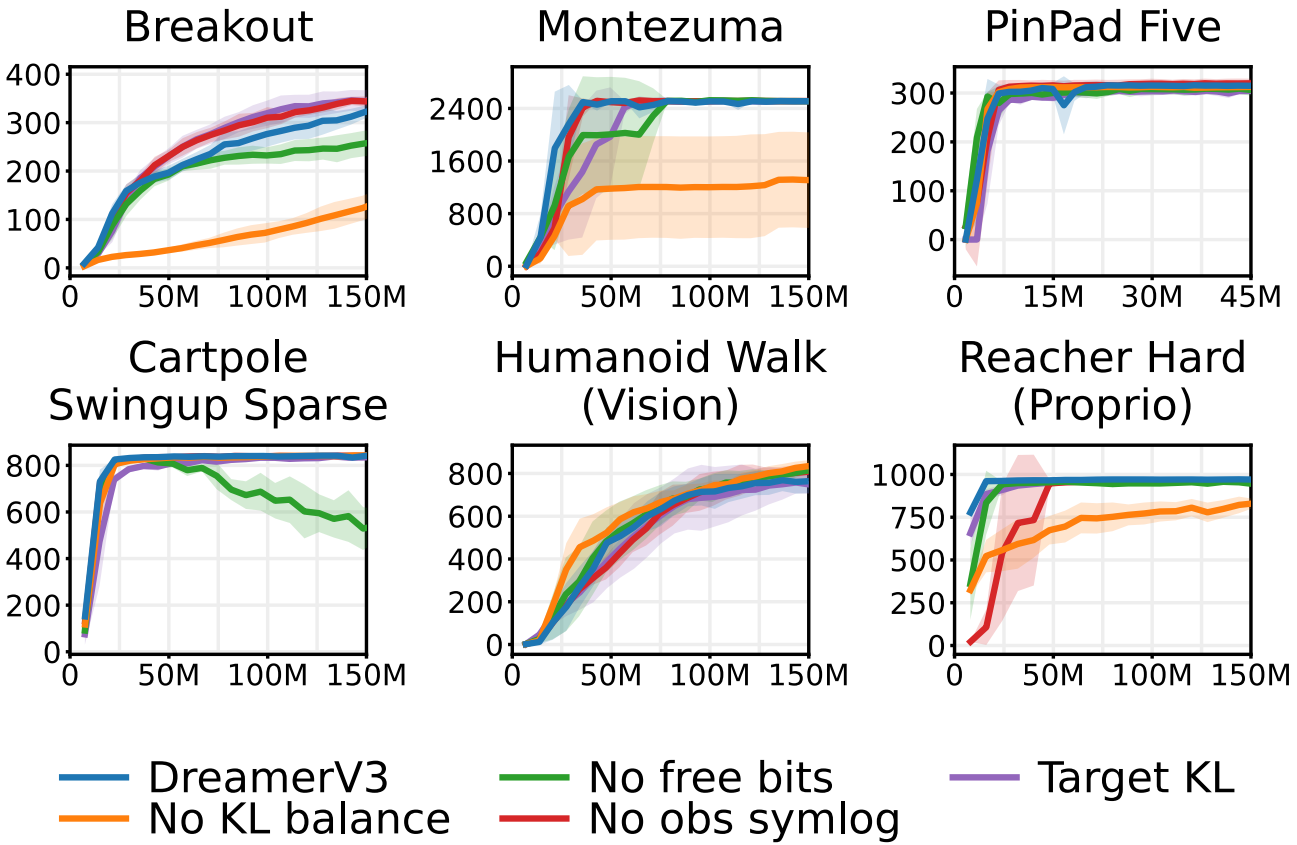


Figure D.1: World Model ablations. KL balancing[27] subsantially accelerates learning. Free bits[28,65] avoids overfitting in simple environments. Symlog encoding and predictions for proprioceptive observations speeds up learning. Adjusting the KL scale over the course of training within a reasonable range to target a fixed KL value is a performant but more complicated alternative to free bits.

Figure D.1 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

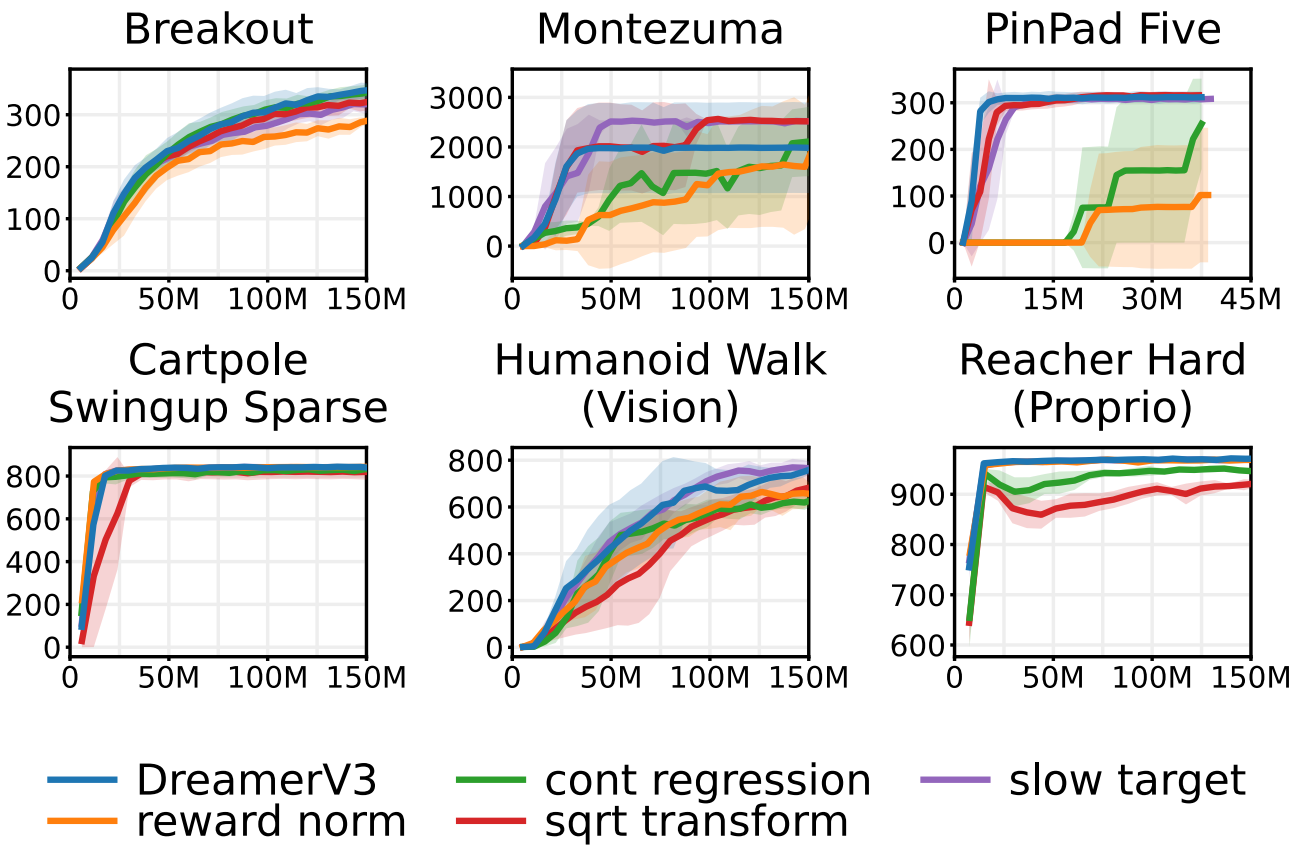


Figure D.2: Critic ablations. The symlog predictions for rewards and values in DreamerV3 outperform non-stationary reward normalization. Discrete regression also contributes significantly[34]. Symlog transformation slightly outperforms the more complex asymmetric square root transformation of R2D2[21]. Using the slow critic for computing targets offers no benefit over using the fast critic and regularizing it towards its own EMA.

Figure D.2 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

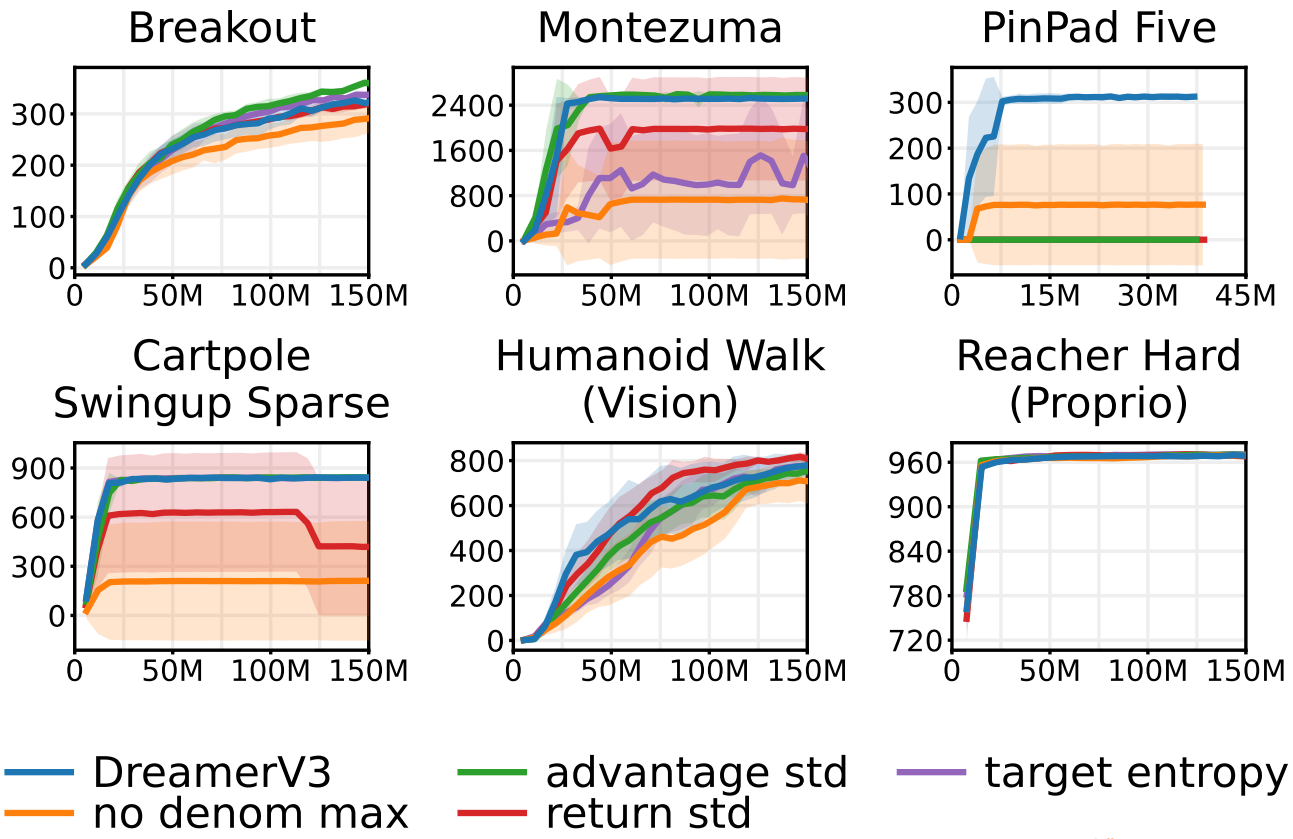


Figure D.3: Actor ablations. The strength of the actor entropy regularizer is important especially under sparse rewards, where the right amount of stochasticity is needed for exploration. The denominator maximum that prevents small returns (often noise) from being amplified is critical. The percentile return normalization of DreamerV3 outperforms normalization based on return stddev or advantage stddev on the sparse reward tasks[72].

Figure D.3 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

Benchmark	Tasks	Env Steps	Action Repeat	Env Instances	Train Ratio	GPU Days	Model Size
DMC Proprio	18	500K	2	4	512	<1	S
DMC Vision	20	1M	2	4	512	<1	S
Crafter	1	1M	1	1	512	2	XL
BSuite	23	—	1	1	1024	<1	XL
Atari 100K	26	400K	4	1	1024	<1	S
Atari 200M	55	200M	4	8	64	16	XL
DMLab	8	50M	4	8	64	4	XL
Minecraft	1	100M	1	16	16	17	XL

Table A.1: Benchmark overview. The train ratio is the number of replayed steps per policy steps rather than environment steps, and thus unaware of the action repeat. BSuite sets the number of episodes rather than env steps, both of which vary across tasks. BSuite requires multiple configurations per environment and one seed per configuration, resulting in 468 runs. For DMC, the proprioceptive benchmark excludes the quadruped tasks that are present in the visual benchmark because of baseline availability in prior work. All agents were trained on 1 Nvidia V100 GPU each.

Table A.1 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

B Model Sizes

Dimension	XS	S	M	L	XL
GRU recurrent units	256	512	1024	2048	4096
CNN multiplier	24	32	48	64	96
Dense hidden units	256	512	640	768	1024
MLP layers	1	2	3	4	5
Parameters	8M	18M	37M	77M	200M

Table B.1: Model sizes. The encoder consists of stride 2 convolutions of doubling depth until resolution 4×4 followed by flattening. The decoder starts with a dense layer, followed by reshaping to $4 \times 4 \times C$ and then inverts the encoder architecture. The dynamics are implemented as RSSM with vectors of categorical representations, consisting of a GRU and dense layers.

Table B.1 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1

Name	Symbol	Value
General		
Replay capacity (FIFO)	—	10^6
Batch size	B	16
Batch length	T	64
Activation	—	$\mathrm{LayerNorm} + \mathrm{SiLU}$
World Model		
Number of latents	—	32
Classes per latent	—	32
Reconstruction loss scale	β_{pred}	1.0
Dynamics loss scale	β_{dyn}	0.5
Representation loss scale	β_{rep}	0.1
Learning rate	—	10^{-4}
Adam epsilon	ϵ	10^{-8}
Gradient clipping	—	1000

Name	Symbol	Value
Actor Critic		
Imagination horizon	H	15
Discount horizon	$1/(1-\gamma)$	333
Return lambda	λ	0.95
Critic EMA decay	—	0.98
Critic EMA regularizer	—	1
Return normalization scale	S	$\mathrm{Per}(R, 95) - \mathrm{Per}(R, 5)$
Return normalization limit	L	1
Return normalization decay	—	0.99
Actor entropy scale	η	$3 \cdot 10^{-4}$
Learning rate	—	$3 \cdot 10^{-5}$
Adam epsilon	ϵ	10^{-5}
Gradient clipping	—	100

Table W.1: DreamerV3 hyper parameters. The same values are used across all benchmark suites, including proprioceptive and visual inputs, continuous and discrete actions, and 2D and 3D domains. We do not use any hyperparameter annealing, weight decay, or dropout.

Table W.1 of "Mastering Diverse Domains through World Models", https://arxiv.org/abs/2301.04104v1