Summary of the paper:

Mastering Diverse Domains trough World Models

Ignacio Monardes

January 9, 2026

Reference

Authors: Danijar Hafner, et al.
Title: *Mastering Diverse Domains trough World Models*
Conference/Journal: Nature
Link: `https://arxiv.org/pdf/2301.04104`

1 Motivation

Problem to solve

Dreamer v3 is an algorithm that learns the world model of the environemnts and can learn how to act in over 150 tasks including minecraft with a single configuration of hyperparameters.

Relevance of the problematic

The relevance of the problem is that with this algorithm we can use it in many environements without the necessity of tunning the hyperparameters and with a performance comparable of specialized agents. The robustness in many different scenarios is a result of using normalization, balancing and transformations during trainning.

Hole in the literature / previous limitations

The previous limitations was the inhability of the agents to just be runned with an agent without a simple configuration. This led to running many configurations to set the better one.

2 Main idea

Concise explanation

There are three neural networks:

- The World Model predicts the outcomes of potential actions.
- The critic judges the value of each outcome.

- The actor chooses actions to reach the most valuable outcomes.

All these components need to accomodate different signal magnitudes and robustly balance terms in their objectives.

Innovation

The innovation of this particular papers is the use of many regularizers. The innovation of dreamer in particular is to learn a policy in the imaginated world which is much cheaper than to train on the real image that may contain unnecesary or noisy information.

3 Methodology

Modeling

- World Model: The world model consist of an autoencoder that enables planning by predicting the future.

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

1. The encoder maps sensory data (observation) x_t to an stochastic representation z_t.
2. A sequence model (RNN) predicts the representation of $\hat{z}_t$ using the past actions (a_{t-1}). The state is defined as $s_t := (h_t, z_t)$ from which we predict rewards r_t and episode continuation flags $c_t \in \{0, 1\}$ and reconstructs the inputs to ensure informative representations
3. The representations z_t are sampled from a vector of softmax distributions.
4. Given a sequence batch of inputs $x_{1:T}$, actions $a_{1:T}$, rewards: $r_{1:T}$ and continuations $r_{1:T}$ the world model parameters ϕ are optimized end-to-end to minimize the prediction loss $\mathcal{L}_{pred}$, the dynamics loss $\mathcal{L}_{dyn}$ and the representation loss $\mathcal{L}_{rep}$ with corresponding weights: $\beta_{pred} = 1$, $\beta_{dyn} = 1$, $\beta_{rep} = 0.1$:

$$\begin{aligned} \mathcal{L}_\phi &:= \mathbb{E}_{q_\phi}\left[\sum_{t=1}^{T} \left(\beta_{pred}\, \mathcal{L}_{pred}(\phi) + \beta_{dyn}\, \mathcal{L}_{dyn}(\phi) + \beta_{rep}\, \mathcal{L}_{rep}(\phi)\right)\right] \\ \mathcal{L}_{pred} &:= -\ln p_\phi(x_t \mid z_t, h_t) - \ln p_\phi(r_t \mid z_t, h_t) - \ln p_\phi(c_t \mid , z_t, h_t) \\ \mathcal{L}_{dyn} &:= \max\left(1, KL\left[sg(q_\phi(z_t \mid h_t, x_t)) \parallel p_\phi(z_t \mid h_t)\right]\right) \\ \mathcal{L}_{rep} &:= \max\left(1, KL\left[q_\phi(z_t \mid h_t, x_t) \parallel sg(p_\phi(z_t \mid h_t))\right]\right) \end{aligned}$$

$$KL(q \parallel p) := \mathbb{E}_q[\log(q) - \log(p)]$$

5. The prediction loss trains the decoder and reward via the symlog squared loss. The continue predictor via logistic regression.
6. The dynamics loss trains the sequence model to predict the next representation by minimizing the KL divergence between the predictor and the next stochastic representation.
7. The representation loss trains the representations to become more predictable.

- Critic: Both actor and critic are only trainned on imagined trajectories. The actor aims to maximize the return with a discount factor of $\gamma = 0.997$ for each model state. For rewards beyond the horizon $T = 16$, the critic learns to approximate the distribution of the returns.

$$\text{Actor:} \quad a_t \sim \pi_\theta(a_t \mid s_t) \qquad \text{Critic:} \quad v_\psi(R_t \mid s_t).$$

From the replayed inputs the WM and actor generates a trajectory $\{(s_t,\, a_t,\, r_t,\, c_t)\}_{t=1:T}$. The critic must learn this distribution:

$$v_t := \mathbb{E}[v_\psi(\,\cdot \mid s_t)]$$

The loss of the critic is:

$$\mathcal{L}(\psi) := -\sum_{t=1}^{T} \ln p_\psi(R_t^\lambda \mid s_t)$$

The critic is parametrized as a categorical distribution with exponentially spaced bins.

- Actor: the actor learns to choose actions that maximize return while exploring trough an entropy regularizer. The scale of the regularizer depends on the sparsity and scale of the rewards. Explore more if the reward is sparse. To manage the scale we could just normalize.

$$\mathcal{L}(\theta) := -\sum_{t=1}^{T} \operatorname{sg}\Big((R_t^\lambda - v_\psi(s_t))/\max(1, S)\Big) \log \pi_\theta(a_t \mid s_t) + \eta H\left[\pi_\theta(a_t \mid s_t)\right]$$

$$S := \operatorname{EMA}(\operatorname{Per}(R_t^\lambda, 96) - \operatorname{Per}(R_t^\lambda, 5), 0.99)$$

Substracting an offset doesn't change the actor gradient (sparsity) and dividing by S is sufficient for the normalization.

- Robust prediction: predict large targets using squarred loss can lead to divergence. Normalizing targets based on running statistics introduces non-stationarity into the optimization. The paper proposes the symlog squarred error. A neural network $f(x, \theta)$ with inputs x and params θ learns to predict a transformed version of its targets y. To get the real prediction we just use $\hat{y}$ which is the inverse transformation.

$$\mathcal{L}(\theta) := \frac{1}{2}(f(x, \theta) - \operatorname{symlog}(y))^2 \quad \hat{y} := \operatorname{symexp}(f(x, \theta))$$

$$\text{symlog}(x) := \text{sign}(x) \ln(|x| + 1) \quad \text{symexp}(x) := \text{sign}(x) \left(\exp(|x|) - 1\right)$$

The symlog allows negative and positive values. The big absolute values are compressed, while the small absolute values are almost the same.

For stochastic targets (rewards, returns, etc) there is the symexp twohot loss. The network outputs the logits for a softmax distribution over exponentially spaced bins $b_i \in B$. The network can achieve any value by a linear combination of the twohot values.

The twohot is just a generalization of onehot but for continue values to use linear combination. Every value has 0 except the two closes which sums up to 1 by the linear combination. The network is trained to minimize the categorical cross entropy loss for clasification.

$$\hat{y} := \text{softmax}(f(x))^\top B \quad B := \text{sympex}([-20, \dots, 20])$$

$$\mathcal{L}(\theta) := -\text{twohot}(y)^\top \log \text{softmax}(f(x, \theta))$$

Assumptions

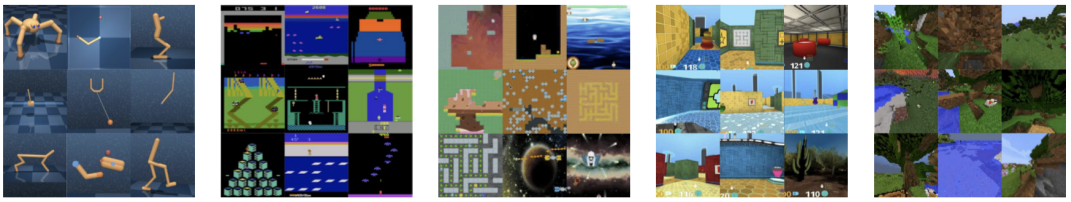
Arquitechture

- The encoder and decoder consists of CNNs for image inputs and MLPs for vector inputs.
- The dynamics, reward and continue predictor are MLPs.
- The representations z_t are sampled from a softmax distributions and we take straight-trough gradients trough sampling step.

4 Experiments and Results

Environments or datasets

Dreamer was evaluated in 8 domains with over 150 tasks with fixed parameters. The authors tunned PPO with fixed hyperparameters to use acrros all the tasks as a baseline comparison. Finally, the authors disabled and tested the relevance of each component individually.

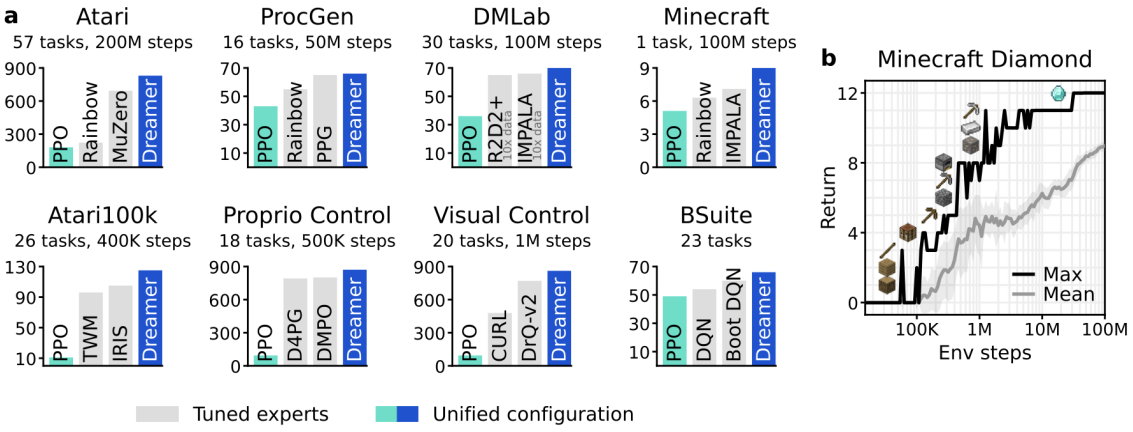


(a) Control Suite **(b)** Atari **(c)** ProcGen **(d)** DMLab **(e)** Minecraft

Figure 2: Diverse visual domains used in the experiments. Dreamer succeeds across these domains, ranging from robot locomotion and manipulation tasks over Atari games, procedurally generated ProcGen levels, and DMLab tasks, that require spatial and temporal reasoning, to the complex and infinite world of Minecraft. We also evaluate Dreamer on non-visual domains.

- **Atari**: 57 atari 2600 games with a budget of 200M frames. Dreamer outperforms MuZero with only a fraction of computational resources. It also outperformed Rainbow and IQN.
- **ProcGen**: 16 games with randomized levels and visual distractions. Dreamer matches PPG expert and outperforms Rainbow.
- **DMLab**: 30 3D tasks with spatial and temporal reasoning. Outperforms IMPALA and R2D2+ with an efficiency of 1000%.
- **Atari100k**: 26 atari games with 400k frames. EfficientZero holds the state of the art with many techniques and with an early reset. Dreamer outperforms every other model.
- **Proprio Control**: 18 control tasks with continuous actions. Control and robotic arms. Dense and sparse rewards. Dreamer sets new state of the art.
- **Visual Control**: 20 continuous control tasks with images. New state of the art.
- **BSuite**: 23 environments and 468 configs. State of the art and much better in the robustness category.
- **Minecraft**: Collecting diamonds in minecraft. Very difficult task which consists in getting 12 items by crafting or foragning from resources with sparse rewards.

5 Results



In which escenarios it works better or worse?

6 Critical Discussion

Strengths

Weakness

Cuestionable assumptions

Not very clear

7 Conclusions

Personal comment

Is it a good contribution?

Would I use it in my research?