Generative adversarial imitation learning in agent’s imagination

Ekaterina Amozova

Master’s thesis

UNIVERSITY OF EASTERN FINLAND

School of Computing
Computer Science
June 2024

UNIVERSITY OF EASTERN FINLAND, Faculty of Science, Forestry and Technology
Joensuu
School of Computing
Computer Science

Amozova, Ekaterina: Generative adversarial imitation learning in agent's imagination
Master's thesis, 35 p.
Supervisor: Ville Hautamäki
June 2024

Abstract: The goal of this work is to combine a world model based agent called Dreamer with adversarial imitation learning, addressing the situation where the environment does not provide rewards. Dreamer learns behaviors purely on the basis of predictions made by its world model in a compact latent space. This work adds a discriminator module that is trained on expert demonstrations and policy rollouts to provide an implicit reward for Dreamer. The model is tested on the Crafter environment using three tasks of different nature. The results are evaluated both numerically and visually. The experiments show that this approach appears to work in open-world environments with complex dynamics and that the design of the discriminator seems to have great importance for the algorithm's performance.

Keywords: machine learning; reinforcement learning; world models; imitation learning; adversarial imitation learning; latent space

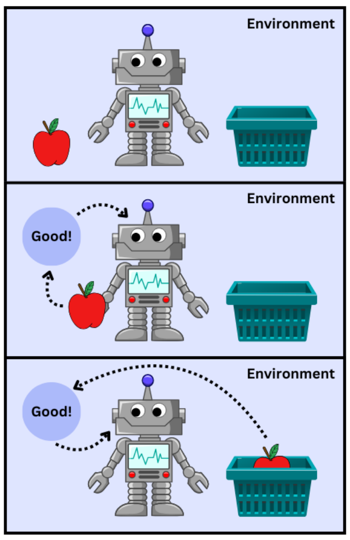
Contents

1. Introduction

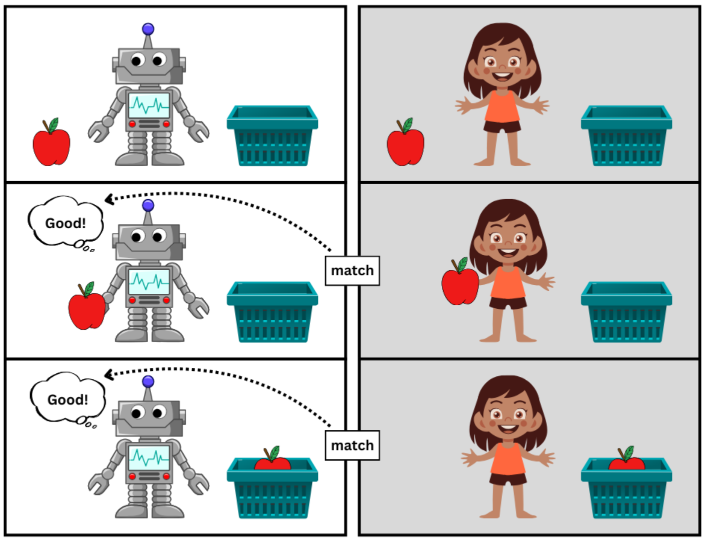
Out of the three main machine learning paradigms, *reinforcement learning* (RL), or learning from interacting with the environment, is the one that most closely resembles how humans and other animals learn (Sutton & Barto, 2014-2015). Reinforcement learning involves sequential decision-making, where each action influences future states of the environment. One distinct feature of RL is that it is goal-directed, making it especially suitable for settings where a certain goal is defined, such as tasks in robotics or games with a win condition.

Consequently, robotics simulation environments (Plappert et al., 2018) and 2D games (Bellemare, Naddaf, Veness, & Bowling, 2012) have remained the main testing grounds in RL research, bridging the gap between simulation and reality. However, the need for more challenging environments grows along with the need for more novel and successful algorithms. Zhang, Wu, and Pineau (2018) criticise the widely used RL benchmarks for their limited complexity and deterministic settings. The real world is, on the contrary, complex and non-deterministic. To develop algorithms that can solve tasks in a constantly changing environment, researchers require benchmarks that both demand and encourage generalization abilities and adaptability. Open-world games as simulation environments (Fan et al., 2022; Johnson, Hofmann, Hutton, & Bignell, 2016) can be considered a step towards the complexity of the real world, compared to standard benchmarks.

Some RL algorithms build an internal model that captures their knowledge about the world, allowing them to plan ahead. Some of these *model-based* algorithms (Sutton & Barto, 2014-2015) can even learn by "imagining" situations within the model, reducing the amount of environment interactions needed. *Model-free* algorithms (Sutton & Barto, 2014-2015), on the other hand, learn simply by trial and error. However, whether model-based or model-free, a classic RL algorithm would still depend on rewards that come from the environment to learn (Fig. 1.1a). For complex tasks, defining a reward function can be challenging (Milani et al., 2023): for example, a task defined as "find a cave" would first prompt a definition of what can be considered "a cave" in terms of



(a) Reinforcement learning



(b) Imitation learning

Figure 1.1: Main differences between RL and IL. In RL, the agent receives a reward from the environment for taking an action on the basis of how that action affects the environment. In IL, the agent receives a reward for how well its behavior matches the expert's behavior.

visual perception. Defining a suitable reward function is often an iterative process, and human designers cannot account for all possible cases in an uncertain world, which may lead to undesirable behavior and the reward failing to represent the goal adequately (He & Dragan, 2021). *Imitation learning* (IL) is different from reinforcement learning in the sense that it does not require a reward function for a specific task. In imitation learning, an agent learns to copy an expert's behavior, the only thing defining its success being how good it is at mimicking the expert (Fig. 1.1b). This could be interpreted as learning the internal reward that drives the expert to complete a given task.

Despite the differences, these approaches can be combined. By applying an imitation learning approach to a model-based RL agent, this work aims to answer the following research questions:

1. Is it possible to bring an agent's imagination closer to human behavior using imitation learning in complex environments?
2. How do design choices affect learning in such cases?

The following chapters can be divided into four topics: the main formal concepts involved, world models, imitation learning, and the core of this work - adversarial imitation learning applied to a model-based RL agent Dreamer. Description of the idea and implementation constitutes its own chapter, followed by experimental setup, results of the experiments, conclusions and discussion of the results.

2. Background theory

2.1 Reinforcement learning

To discuss more advanced concepts in the following chapters, it is first necessary to provide foundation for them by describing the reinforcement learning framework in more detail. The setting of reinforcement learning amounts to the *agent* - a decision making entity - learning to achieve a goal from interaction with the *environment* - everything outside the agent (Sutton & Barto, 2014-2015). From the agent's perspective, the goal is to maximize the total amount of *reward*, or the expected *return* it gets from the environment. At each discrete time step t, the agent receives a representation of the environment's state S_t and uses it to select an action A_t; at the following time step $t + 1$, the agent receives a reward R_{t+1}, and the environment transitions to state S_{t+1} (Figure 2.1). The other relevant concepts are *policy* π, which is a mapping from states to probabilities of selecting actions, and *value function*, which estimates either how good it is for an agent to be in a given state (*state-value function*) or how good it is to perform a given action in a given state (*action-value function*) (Sutton & Barto, 2014-2015). In other words, value function evaluates the current policy.

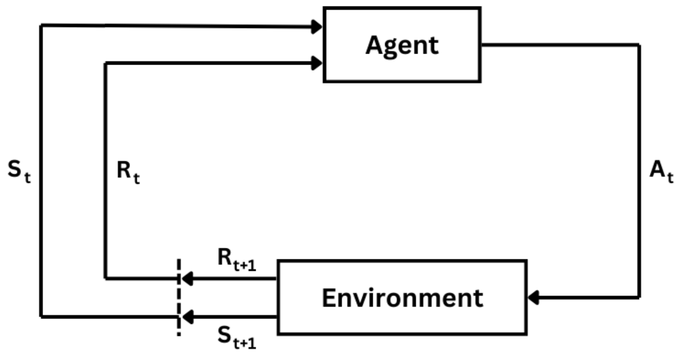


Figure 2.1: The interaction between agent and environment in reinforcement learning.

The expected return the agent aims to maximize can be expressed as $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$; however, this formulation assigns equal importance to every expected reward, however distant it might be from the present moment. For the purpose of distinguishing between immediate and distant future rewards, the *discount rate* γ is introduced into the formulation:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.1}$$

where $0 \leq \gamma \leq 1$. The closer the discount rate is to 0, the more interested the agent is in the immediate rewards and the less farsighted it becomes. Using the formulation in Equation 2.1, the state-value function v_π can be defined formally as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right], \tag{2.2}$$

and the action-value function q_π can be defined as:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]. \tag{2.3}$$

Often, it does not make sense to consider rewards for all future steps, especially if T is not fixed. Then, expected returns are only considered up to some horizon h, resulting in *partial returns* G_t^h (Sutton & Barto, 2014-2015).

A reinforcement learning task can be modeled as a *Markov decision process* (MDP) (Sutton & Barto, 2014-2015), if its environment satisfies the *Markov property*. An environment having the Markov property means that its one-step dynamics $p(s', r | s, a) = Pr\{R_{t+1} = r, S_{t+1} = s' | S_t, A_t\}$ allows to predict the next state and expected reward given the current state and action without needing the history of previous actions and states. More generally, it means that the state signal retains all the information necessary to predict the dynamics. An MDP is described by a tuple $\langle S, A, T, R \rangle$, where S is a set of states, A is a set of actions, R is the reward function $R(s, a)$, and T is the state-transition function such that $T(s, a, s')$ is the probability of ending in state s' given starting state s and action a.

In reality, the Markov property is not always satisfied, but even in such a case it is still appropriate to treat a reinforcement learning task as an approximation to an MDP. For example, an improved state representation can be constructed from a non-Markov one provided by the environment. A *partially observable Markov decision process* (POMDP) (Sutton & Barto, 2014-2015) is a generalization of an MDP that uses this approach. In POMDP, the environment's state is not directly observable, but the agent receives an observation signal that is stochastically related to the state. It is often the case in games. Take an enemy's position, for example: in terms of the environment's state, it would be defined with coordinates, while the visual position on the screen would be the observation, indicative of the true state and available to the agent. In case of POMDP, the definition of policy is changed to a mapping from *belief states* that summarize the previous experience to probabilities of selecting actions (Kaelbling, Littman, & Cassandra, 1998). A POMDP can be described as a tuple $\langle S, A, T, R, \Omega, O \rangle$, where Ω is a set of observations the agent can experience and O is the observation function which gives a probability distribution over Ω such that $O(s', a, o)$ is the probability of making observation o given action a and resulting state s'. A belief state b is a probability distribution over states S such that $b'(s') = Pr(s'|o, a, b)$, where a new belief b' for a given state s' is computed given an old belief state b, an action a and an observation o.

The *history of observations*, or more simply, a *trajectory* is usually a sequence of state-action pairs $\tau = (s_0, a_1, s_1, \ldots, s_T)$. An *episode* is a subsequence of agent-environment interaction with finite number of time steps T (Sutton & Barto, 2014-2015). A trajectory can cover a whole episode, only a part of it, or possibly combine different episodes, depending on what it is used for.

Based on whether the same policy is used for selecting actions and being improved or evaluated, reinforcement learning methods can be divided into two groups. *On-policy* methods improve the same policy that is used to select actions, while *off-policy* methods select actions based on a different policy from the one that is learned (Sutton & Barto, 2014-2015). From all the variety of RL methods, an on-policy method called *actor-critic* is the most relevant to this work. Actor-critic methods owe their name to their dual structure, where the policy exists separately and independently from the value function. The policy structure, or the actor, selects actions, while the estimated value function, or the critic, judges its actions. After an action has been selected and the environment has transitioned into a new state, the critic determines whether the action selection has led to a better or worse outcome than expected.

2.2 Variational inference

Another concept that can prove beneficial for understanding a large part of the following work is *variational inference*. Variational inference (Jordan, Ghahramani, Jaakkola, & Saul, 1998) methods are machine learning methods for approximating probability densities, related to Bayesian statistics. In Bayesian models, the random process generating the data is assumed to involve unobserved *latent variables* (Blei, Kucukelbir, & McAuliffe, 2017; Kingma & Welling, 2022). The goal of variational inference is to approximate a density of these latent variables given observed variables, which can then be used to estimate latent variables or form predictions of new data (Blei et al., 2017).

The setting for the inference problem includes a joint density $p(z, x) = p(z)p(x|z)$, where the latent variables z are drawn from a *prior* density $p(z)$ and then related to the observations x through the likelihood $p(x|z)$. The inference problem is to compute the *posterior* density $p(z|x) = \frac{p(z,x)}{p(x)}$, where $p(x)$ is the *evidence*, or marginal density of the observations. In many cases, the evidence is either unavailable or hard to compute, making the inference difficult.

Variational inference uses optimization to approximate the posterior instead of computing it. For that, it defines a family of approximate densities Q (a set of densities over the latent variables) and tries to find a candidate $q(z)$ that minimizes the Kullback-Leibler (KL) divergence to the posterior. The KL divergence is the average extra amount of information required to encode the data using the candidate probability distribution instead of the actual distribution (Ganguly & Earp, 2021), and for two probability distributions Q(S) and P(S) it is defined as $\mathrm{KL}(Q\|P) = \sum_{\{S\}} Q(S) \log \frac{Q(S)}{P(S)}$ (Jordan et al., 1998). The objective in question can be expressed as:

$$\mathrm{KL}(q(z)\|p(z|x)) = \mathbb{E}[\log q(z)] - \mathbb{E}[\log p(z|x)] \tag{2.4}$$

This objective, however, is not computable because it requires computing the evidence. An alternative objective, the evidence lower bound (ELBO), can be derived from the one in Equation 2.4 (Blei et al., 2017):

$$\mathrm{ELBO}(q) = \mathbb{E}[\log p(z, x)] - \mathbb{E}[\log q(z)] \tag{2.5}$$

ELBO lower-bounds the (log) evidence, and maximizing it is equivalent to minimizing the KL divergence in Equation 2.4.

2.3 Variational autoencoders

Variational autoencoder (VAE) (Kingma & Welling, 2022) is a neural network architecture that makes use of variational inference methods. The formulation of the inference problem remains largely the same as explained in Section 2.2: a generative model $p_\theta(z)p_\theta(x|z)$, where the true parameters θ and values of the latent variables z are unknown. Additionally, Kingma and Welling (2022) introduce a *recognition model* $q_\varphi(z|x)$, which is an approximation to the intractable true posterior $p_\theta(z|x)$. The resulting model is summarized in Figure 2.2. The recognition model can be called a probabilistic *encoder*, because given x it produces a distribution over the possible values of the latent variables, or the code z from which x could have been generated. $p_\theta(x|z)$ is its counterpart, a probabilistic *decoder*, because it produces a distribution over the possible values of x that could correspond to a given code z. The encoder aims to learn patterns from the input data, while the decoder uses latent space representation to regenerate the data which is similar to the input. VAE can be used both for learning the representation of data and mimicking the generative process to generate new data.

For the purpose of learning the recognition model parameters φ jointly with the generative model parameters θ, Kingma and Welling (2022) introduce a Stochastic Gradient Variational Bayes (SGVB) estimator, which is based on the reparameterization trick. The random variable $\tilde{z} \sim g_\varphi(z|x)$ is reparameterized using a differentiable transformation of an auxiliary noise variable $\epsilon \sim p(\epsilon)$: $\tilde{z} = g_\varphi(\epsilon, x)$. This way, latent variable is represented as a deterministic function of the stochastic noise variable and observations, which allows to bypass the stochastic variable when backpropagating the gradient. In short, *backpropagation* involves updating the weights of the neural network so that the cost or loss function is minimized, and the gradient descent is the algorithm that finds this local minimum (Kelley, 1960).

First, a generic SGVB estimator A for a datapoint $x^{(i)}$ is defined, which is similar to ELBO (Eq. 2.5):

$$\tilde{\mathcal{L}}^A(\theta, \varphi; x^{(i)}) = \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x^{(i)}, z^{(i,l)}) - \log q_\varphi(z^{(i,l)}|x^{(i)}) \tag{2.6}$$

Then, SGVB estimator B can be defined for the cases where the KL divergence can be

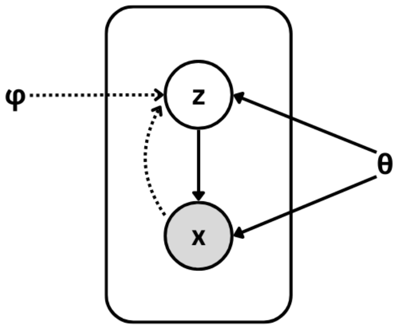


Figure 2.2: A directed model used in VAE. Parameters θ along with solid lines denote the generative model $p_\theta(z)p_\theta(x|z)$. Parameters φ along with dashed lines denote the variational approximation $q_\varphi(z|x)$.

integrated analytically:

$$\tilde{\mathcal{L}}^B(\theta, \varphi; x^{(i)}) = -D_{KL}(q_\varphi(z|x^{(i)})\|p_\theta(z)) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(x^{(i)}, z^{(i,l)}), \tag{2.7}$$

for example, when both distributions (prior and approximate posterior) are Gaussian, which is the typical case for the VAE.

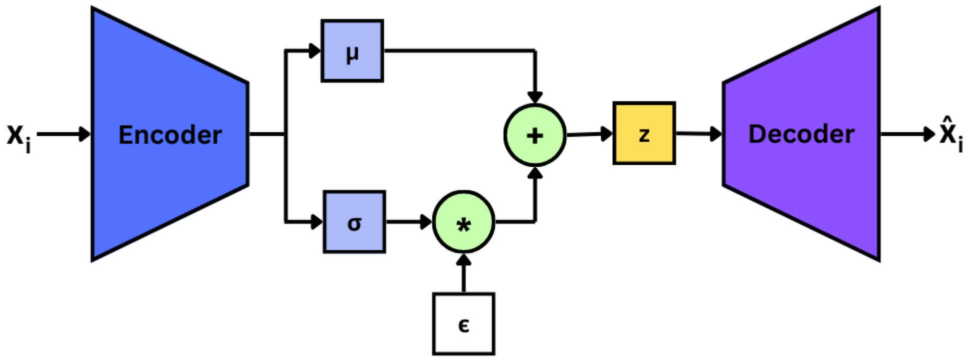


Figure 2.3: VAE architecture. Encoder $q_\varphi(z|x)$ produces mean (μ) and variance (σ) of the posterior distribution, and the posterior is sampled using $\mu + \sigma * \epsilon$. The sample z is fed into the decoder $p_\theta(x|z)$ to reconstruct the input.

Kingma and Welling (2022) propose the Auto-Encoding VB (AEVB) algorithm for the case with an i.i.d. dataset and continuous latent variables per datapoint. Typically, both the prior over the latent variables and the variational approximate posterior are modeled as Gaussian distributions ($p_\theta(z) = \mathcal{N}(z; 0, I)$ and $\log q_\varphi(z|x^{(i)}) = \log \mathcal{N}(z; \mu^{(i)}, \sigma^{2(i)} I)$, respectively). The output of the encoder consists of two parameters: mean (μ) and variance (σ) of the posterior distribution. The latent space (posterior) distribution is then sampled randomly, and the sample is assumed to generate the input data. The decoder $\log p_\theta(x|z)$ is either a Gaussian or Bernoulli distribution, depending on the type of the data. This process is depicted and summarized in Figure 2.3. The loss function for the autoencoder corresponding to Eq. 2.7 can be written as follows:

$$L(x) = -D_{KL}(q_\varphi(z|x)\|p_\theta(z)) + \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] \tag{2.8}$$

2.4 Generative adversarial networks

The goal of *deep learning* is to automatically discover hierarchical models that represent probability distributions over data (Bengio, 2009; Goodfellow et al., 2014). Deep architectures, such as neural nets with many hidden layers, can discover abstractions of the data of varying levels: from the lowest level features, like edges in the image, to the

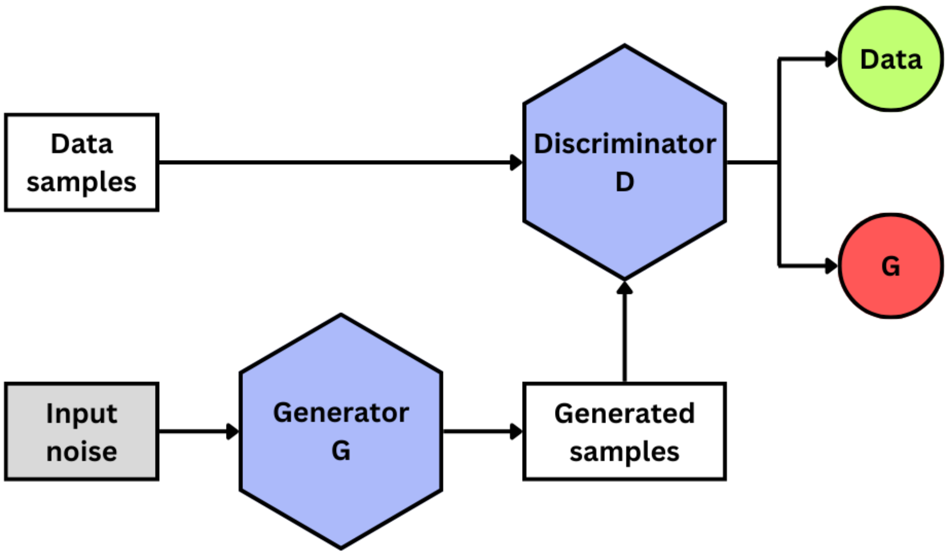


Figure 2.4: GAN architecture. G generates samples from input noise, D tries to distinguish whether they come from the actual data distribution or are generated by G.

highest level concepts, like the object depicted (Bengio, 2009). Deep discriminative models combine deep neural networks with *discriminative models*, which model the conditional probability $P(Y|X)$, mapping inputs X to class labels Y (A. Ng & Jordan, 2001), essentially classifying the data. Deep generative models combine deep neural networks with *generative models*. Generative models model the joint probability $P(X, Y)$ (A. Ng & Jordan, 2001), from which the conditional probability $P(X|Y)$ can be computed, allowing to generate random samples X conditioned on labels Y (Mitchell, 2015).

In a *generative adversarial network* (GAN), a generative model G that generates samples from random noise competes with a discriminative model D, both models being multilayer perceptrons. D is trained to maximize the probability of assigning a correct label to the input, in other words, determining which distribution a given sample comes from. Figure 2.4 illustrates the architecture in simple terms. G is trained to minimize the probability of the generated samples being labeled correctly. As a result, D and G play the following minimax game:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))], \qquad (2.9)$$

where $G(z)$ is a mapping of input noise to data space, $p_z(z)$ is the prior on input noise variables, and $D(x)$ outputs a single scalar that represents the probability that x came from data rather than from the generator's distribution.

Rather than training G to minimize $\log(1 - D(G(z)))$, it can be trained to maximize $\log D(G(z))$ instead to avoid the saturation problem, when the gradients become so

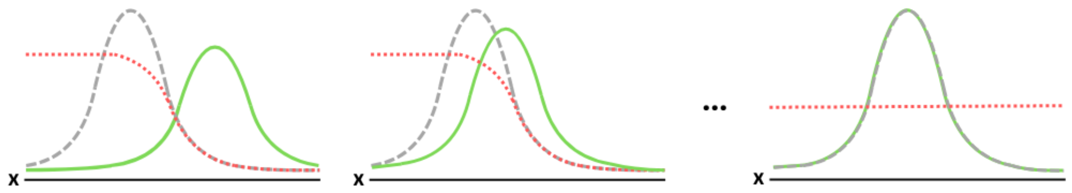


Figure 2.5: GAN convergence process. The discriminative distribution (red dotted line) is updated so that it distinguishes between samples from the actual data distribution p_x (grey dashed line) and samples from the generative distribution p_g (green solid line). The gradient of D guides $G(z)$ to flow closer to the actual data distribution. If G and D have enough capacity, they will eventually arrive at the point where $p_g = p_{\text{data}}$ and D is unable to differentiate between them.

small that it stagnates learning. D must also be synchronized well with G during training in order for G to not collapse, which would result in p_g not being diverse enough to model p_{data}. Usually, D is optimized for several steps, which are then followed by a single step of optimizing G. This is done in order to maintain D near its optimal solution, while G learns slowly enough to remain stable. The process of convergence is shown in Figure 2.5. An undeniable advantage of the architecture is that no inference is needed during learning.

3. World models

Many concepts in machine learning are inspired by processes that occur in living organisms. *World models*, in particular, correspond to mental models that humans build of the world around them and base their decisions on (Forrester, 1971). Model-based reinforcement learning uses a world model to represent the behavior of the environment. Dyna, presented by Sutton (1991), is an example of model-based RL architecture. Dyna uses an *action model* that takes in a state and an action and outputs a prediction for the resulting state and reward. The action model is learned through interacting with the environment and accumulating examples of desired behavior. According to Lin (1992), the action model can output either the most likely outcome or a list of outcomes and probabilities associated with them. In Dyna, planning and selecting an action are not strongly coupled: these processes happen in parallel. The action model is trained in the background, gradually improving the agent's reactive policy. Using world models for planning can improve data efficiency because of a richer training signal, and the learned dynamics can potentially be transferred to other tasks (Hafner et al., 2018). Being a general architecture, Dyna does not explicitly address the way the RL task is modeled; however, in its most straightforward form it is modeled as an MDP.

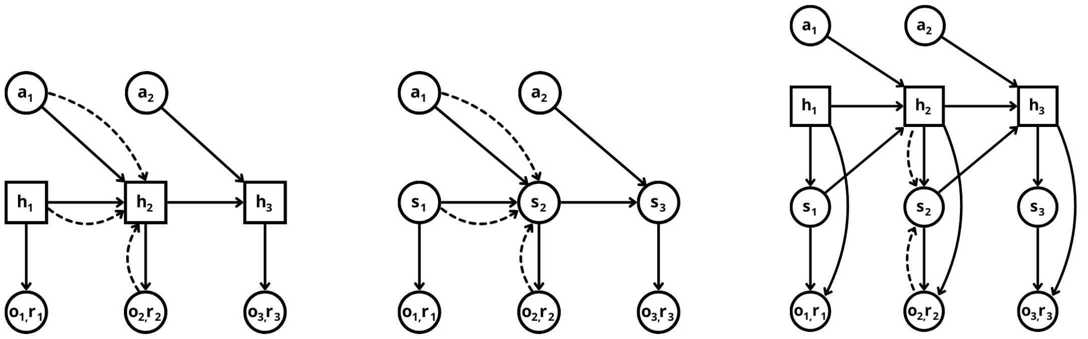
3.1 Latent space

While a model can be learned directly from pixels, making predictions and planning in the same high-dimensional space is computationally expensive. Watter, Springenberg, Boedecker, and Riedmiller (2015) emphasize the need for algorithms that can overcome this limitation and propose their own system that uses a variational autoencoder to learn a mapping of high-dimensional inputs to a low-dimensional latent space. Ha and Schmidhuber (2018) describe the two main components of a world model that uses representation learning: 1) visual component that compresses inputs into a latent representation and 2) memory component that makes predictions based on history.

The memory component remains an integral part of a world model, while the visual component serves to enable faster planning in the latent space.

A model that operates in a latent space, like described above, can be called a *latent dynamics model* or a *state-space model* (SSM). Buesing et al. (2018) describe two types of SSMs: *deterministic* (dSSM) and *stochastic* (sSSM). The latent transition of a deterministic SSM $s_{t+1} = g(s_t, a_t)$ is a deterministic function of the past state s_t and past action a_t. A stochastic SSM instead explicitly models uncertainty over the state s_{t+1} using transition distributions $p(s_{t+1}|s_t, a_t)$. For every t, a latent variable $z_{t+1} \sim p(z_{t+1}|s_t, a_t)$ exists; the state function then becomes $s_{t+1} = g(s_t, a_t, z_{t+1})$. These two types of SSMs are shown in Figure 3.1, though the notation is somewhat different: in the figure, treat h_t as s_t in the dSSM. The main disadvantage of sSSM is that stochastic transitions make it difficult for the model to remember information over multiple time steps, while dSSM cannot capture multiple futures due to determinism and is prone to inaccuracies in predictions (Hafner et al., 2018).

3.2 RSSM: PlaNet



(a) Deterministic SSM **(b)** Stochastic SSM **(c)** RSSM

Figure 3.1: Different SSM types, figure based on Hafner et al. (2018). Square nodes denote deterministic variables, while round nodes denote stochastic ones. h_t denotes deterministic states, s_t - stochastic states, a_t - actions, o_t - observations and r_t - rewards. Straight lines denote the generative process, dashed lines denote the inference process.

Deep Planning Network (PlaNet) (Hafner et al., 2018) is an example of an agent that is based on a state-space model. The task is modeled as a POMDP, because individual image observations do not reveal the full state of the environment. The model Hafner et al. (2018) propose is called recurrent state-space model (RSSM). RSSM splits the state into a stochastic part and a deterministic part to ensure the model learns robustly to predict multiple futures; the difference in structure between dSSM, sSSM and RSSM is illustrated in Figure 3.1. The RSSM consists of the following parts:

$$\begin{array}{ll} \text{Deterministic state model:} & h_t = f(h_{t-1}, s_{t-1}, a_{t-1}), \\ \text{Stochastic state model:} & s_t \sim p(s_t|h_t), \\ \text{Observation model:} & o_t \sim p(o_t|h_t, s_t), \\ \text{Reward model:} & r_t \sim p(r_t|h_t, s_t), \end{array}$$

where t is a time step, s_t denotes hidden states, o_t denotes image observations, a_t denotes continuous action vectors, and r_t denotes scalar rewards. The policy is defined as $a_t \sim \mathrm{p}(a_t|o_{\le t}, a_{\le t})$, where $\le t$ denotes all time steps before and including t, and implemented as a planning algorithm that searches for the best sequence of future actions. An approximate belief over the current hidden state is inferred using the encoder $q(s_t|o_{\le t}, a_{\le t})$.

The training process of PlaNet consists of two parts: 1) model fitting and 2) data collection. Initial dataset used for model fitting consists from a small number of episodes collected under random policy. During model fitting, this experience dataset is sampled and the dynamics model is updated based on the samples. During data collection, a belief over current state is inferred, an action is selected using policy, exploration noise $\epsilon \sim p(\epsilon)$ is added to the action, and the action is repeated several times to provide a clearer learning signal. The episode is then added to the experience dataset to be used in the model fitting.

PlaNet uses *latent overshooting*, which can be interpreted as a regularizer in latent space meant to maintain consistency between one-step and multi-step predictions. To do that, Hafner et al. (2018) generalize the variational bound (analogous to one in Equation 2.8) on one-step predictions to the variational bound on the multi-step predictive distribution p_d. Multi-step predictions of a fixed distance d are calculated by repeatedly applying the transition model and integrating out the intermediate states. Latent overshooting objective that trains the model on multi-step predictions of all distances $1 \le d \le D$ is defined as follows:

$$\begin{aligned} \frac{1}{D}\sum_{d=1}^{D} \ln p_d(o_{1:T}) \ge \sum_{t=1}^{T} \Bigg(& \mathbb{E}_{q(s_t|o_{\le t})[\ln p(o_t|s_t)]} \\ & - \frac{1}{D}\sum_{d=1}^{D} \mathbb{E}_{p(s_{t-1}|s_{t-d})q(s_{t-d}|o_{\le t-d})}\Big[\mathrm{KL}[q(s_t|o_{\le t})\|p(s_t|st-1)]\Big]\Bigg) \end{aligned} \tag{3.1}$$

Latent overshooting, along with the RSSM's dual structure, ensures the robustness of the predictions and enables longer planning horizons by predicting all multi-step priors.

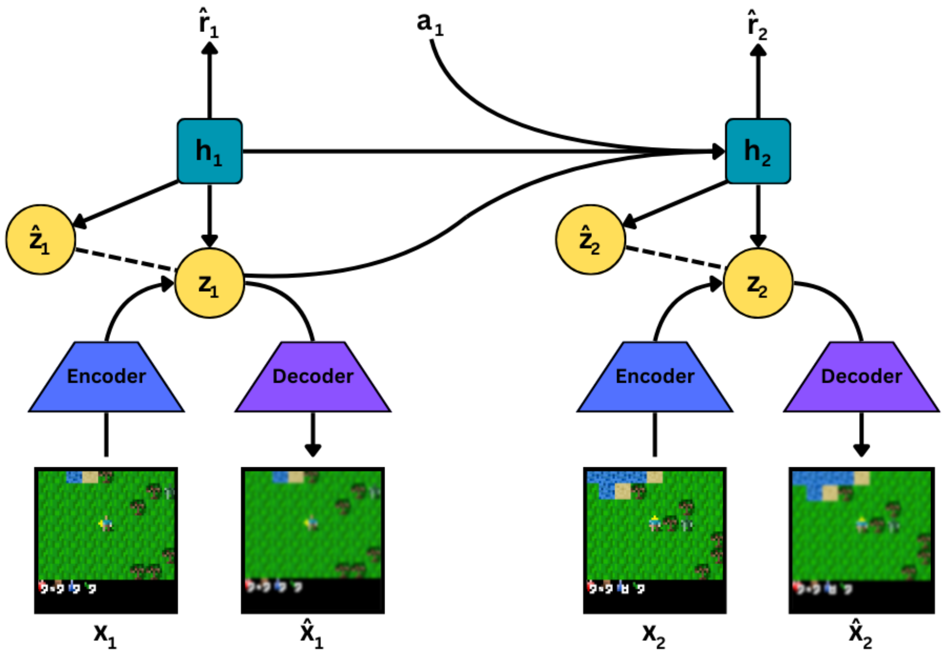


Figure 3.2: World Model learning. At each step, RSSM computes a posterior state z_t and a prior state $\hat{z}_t$ that tries to predict the posterior without access to the current image x_t. The learned prior is used for imagination. $\hat{x}_t$ is the reconstructed input.

Efficient planning in latent space also allows to scale this algorithm to larger state and action spaces.

3.3 Dreamer

Dreamer is a model-based reinforcement learning agent that utilizes the RSSM model described in previous section as a transition model. The defining characteristics of DreamerV1 (Hafner, Lillicrap, Ba, & Norouzi, 2020) include predicting both actions and state values to allow for predicting beyond the imagination horizon, as well as learning behaviors purely by imagination. DreamerV2 (Hafner, Lillicrap, Norouzi, & Ba, 2020) builds upon the previous version of the algorithm, changing Gaussian latents used by the world model to categorical ones. It is hypothesized that categorical latents outperform Gaussian ones because they are better at capturing small details and make it easier to predict changes between subsequent images. DreamerV2 achieves human-level performance on the Atari benchmark (Bellemare et al., 2012). The most recent version of the algorithm, DreamerV3 (Hafner, Pasukonis, Ba, & Lillicrap, 2024), can master diverse domains without the need for hyperparameter tuning.

Dreamer consists of three neural networks: world model, actor and critic. The world model learns from a dataset of past experience (Fig. 3.2), while actor and critic learn from abstract sequences predicted by the world model (Fig. 3.3). The actor interacts with the environment to grow the experience dataset, from which sequences are sampled uniformly regardless of episode boundaries.

The world model of DreamerV3 consists of the following parts:

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

The encoder, sequence model and dynamics predictor form the RSSM. Distributions that generate samples in the real environment are denoted with p, and the approximations of these distributions that enable latent imagination are denoted with q. The encoder maps sensory inputs x_t to discrete representations z_t. The sequence model with recurrent state h_t predicts the sequence of these representations given past actions a_{t-1}. The model state $s_t \doteq \{h_t, z_t\}$ is used to predict rewards r_t and episode continuation flags c_t as well as to reconstruct the inputs, yielding $\hat{x}_t$. A sequence batch of inputs, actions, rewards and continuation flags is used to optimize the world model's parameters ϕ to minimize the total loss:

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi}\Big[\sum_{t=1}^{T}(\beta_{\text{pred}}\mathcal{L}_{\text{pred}}(\phi) + \beta_{\text{dyn}}\mathcal{L}_{\text{dyn}}(\phi) + \beta_{\text{rep}}\mathcal{L}_{\text{rep}}(\phi))\Big], \tag{3.2}$$

where $\mathcal{L}_{\text{pred}}$ is the prediction loss with loss weight $\beta_{\text{pred}} = 1$, $\mathcal{L}_{\text{dyn}}$ is the dynamics loss with loss weight $\beta_{\text{dyn}} = 0.5$, and $\mathcal{L}_{\text{rep}}$ is the representation loss with loss weight $\beta_{\text{rep}} = 0.1$. The prediction loss $\mathcal{L}_{\text{pred}} = -\ln p_\phi(x_t|z_t, h_t) - \ln p_\phi(r_t|z_t, h_t) - \ln p_\phi(c_t|z_t, h_t)$ trains the decoder, reward and continue predictors. The dynamics loss $\mathcal{L}_{\text{dyn}} = \max(1, \text{KL}[\text{sg}(q_\phi(z_t|h_t, x_t))\|p_\phi(z_t|h_t)])$ trains the sequence model. The representation loss $\mathcal{L}_{\text{dyn}} = \max(1, \text{KL}[q_\phi(z_t|h_t, x_t)\|\text{sg}(p_\phi(z_t|h_t))])$ trains the representations to become more predictable. The $\text{sg}(\cdot)$ in the losses is the stop-gradient operator.

The critic learns to predict the return of each state under current policy, in other words, it predicts the expected value of the return distribution: $v(s_t) \approx E[R_t]$. The actor $a_t \sim \pi(a_t|s_t)$ aims to maximize the expected return for each model state. The actor uses an entropy regularizer to ensure sufficient exploration as well as return scaling to accelerate exploration in the case of sparse rewards.

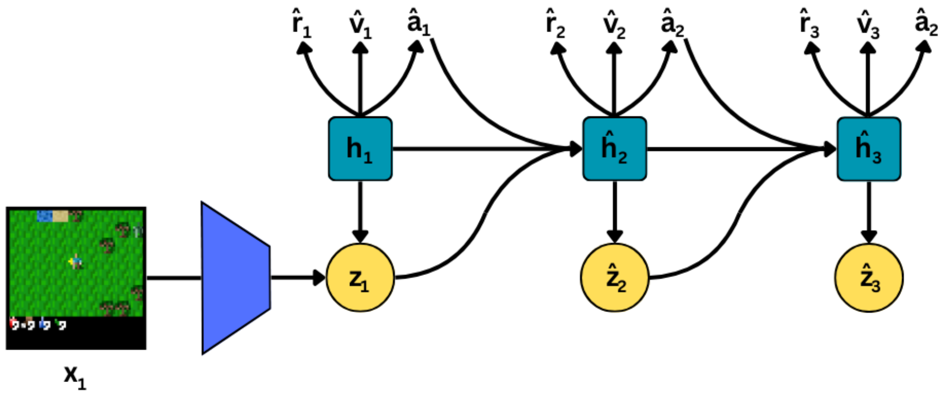


Figure 3.3: Actor-Critic learning. A trajectory of abstract representations is predicted from an actual posterior state, training both actor and critic within the current world model.

Similar to PlaNet, Dreamer's training process can be divided into two main parts: 1) dynamics and behavior learning, and 2) environment interaction. During the dynamics learning, data sequences (x_t, a_t, o_t, c_t) are drawn from the experience dataset and the world model is updated. Then, during the behavior learning, the dynamics predictor and the actor together produce imagined trajectories of form $(s_{1:T}, a_{1:T}, r_{1:T}, c_{1:T})$, where T is the imagination horizon, starting from each representation of the replayed inputs. The actor aims to predict actions that maximize the critic's estimate, while the critic aims to accurately estimate the expected returns. During the environment interaction for collecting the data, the actions are sampled without lookahead planning, which is different from PlaNet.

4. Imitation learning

Imitation learning (Schaal, 1999) (IL) paradigm involves an agent learning by first observing an expert completing a task and then mimicking it. For complex tasks, IL is generally preferred to standard RL for two main reasons. First, IL is in general more sample efficient (Hussein, Gaber, Elyan, & Jayne, 2017). Second, by not requiring a reward signal, IL tackles a harder problem than standard RL (Popov et al., 2017). While the agent can have access to the environment in IL, the environment would only provide the information about the state, without outputting the reward. Instead, the agent learns the policy based on expert trajectories of form $\tau = (s_0, a_1, - \dots s_T)$, where T is the final time step of the episode.

Two classic imitation learning approaches can be used to learn a policy. *Behavioral cloning* (BC) (Bain & Sammut, 1995; Sammut, Hurst, Kedzier, & Michie, 1970) is a method in which an agent is trained using *supervised learning* (SL) to directly map states to actions based on expert demonstrations (Bratko, Urbančič, & Sammut, 1995). Well-known flaws of this approach include distributional shift (Ross & Bagnell, 2010) and poor generalization, resulting in the need to use large amounts of data for training. *Inverse reinforcement learning* (IRL) (A. Y. Ng & Russell, 2000) can be used to infer a reward function from expert data, from which a policy is then extracted via reinforcement learning. The reward function learned by IRL explains expert behavior but does not necessarily tell the agent how to act (Ho & Ermon, 2016). Despite some success in control tasks (A. Y. Ng & Russell, 2000), due to computational costs associated with the IRL approach it has not been successfully applied to larger problems.

4.1 GAIL

Generative adversarial imitation learning (GAIL) (Ho & Ermon, 2016) is an imitation learning approach that draws inspiration from generative adversarial networks, or GANs (Goodfellow et al., 2014). GANs train a generative model G and a discriminative

1 Generative adversarial imitation learning

```
Require: Expert trajectories $\mathcal{T}_E \sim \pi_E$
1: for step i do
2:     Update the discriminator parameters by sampling trajectory $\mathcal{T}_i$ from a fixed policy $\pi$
3:     Optimize policy such that it decreases the expected cost function $c(s, a) = log(D(s, a))$
4: end for
```

classifier D. D learns to distinguish between the distribution of data generated by G from the distribution of true data, while G learns to generate samples that are closer to the true data. GAIL uses a concept of *occupancy measure* instead, which can be interpreted as the distribution of state-action pairs the agent (or the expert) encounters. The goal of GAIL is then to find such policy for which its occupancy measure is closer to the occupancy measure of the expert. GAIL is sample-efficient in terms of expert data, unlike BC, and is a direct approach, unlike IRL. However, the training can be slow and unstable, as is the case with normal GANs (Arjovsky & Bottou, 2017).

Algorithm 1 describes the process of GAIL as explained by Ho and Ermon (2016), where $D(s, a)$ denotes the output of the discriminator given a state-action pair. It can be said that the discriminator aims to solve the binary classification problem of distinguishing between policy and expert occupancy measures (ρ_π and ρ_{π_E}, respectively). The optimal objective for the policy to minimize (and for the discriminator to maximize) is then the Jensen-Shannon divergence between the two distributions

$$JS(\rho_\pi, \rho_{\pi_E}) = KL(\rho_\pi \| (\rho_\pi + \rho_{\pi_E})/2) + KL(\rho_{\pi_E} \| (\rho_\pi + \rho_{\pi_E})/2), \tag{4.1}$$

where KL is the Kullback-Leibler divergence (Goodfellow et al., 2014; Ho & Ermon, 2016). The discriminator loss corresponding to the objective in Equation 4.1 and used in line 2 can be expressed as

$$\mathbb{E}_\pi[-\log(1 - D(s, a))] + \mathbb{E}_{\pi_E}[-\log(D(s, a))], \tag{4.2}$$

which effectively punishes the discriminator for misclassifying policy samples as expert ones and expert samples as generated by the policy.

The reward function corresponding to the cost function on line 3 can be written as $r(s, a) = -\log(1 - D(s, a))$, but it is not the only alternative. Orsini et al. (2021) examine the effect of different design choices, including implicit reward function choice, on the performance of GAIL. The following reward functions were compared: GAIL reward $r(s, a) = -\log(1 - D(s, a))$, AIRL reward $r(s, a) = \log(D(s, a)) - \log(1 - D(s, a))$ (Fu,

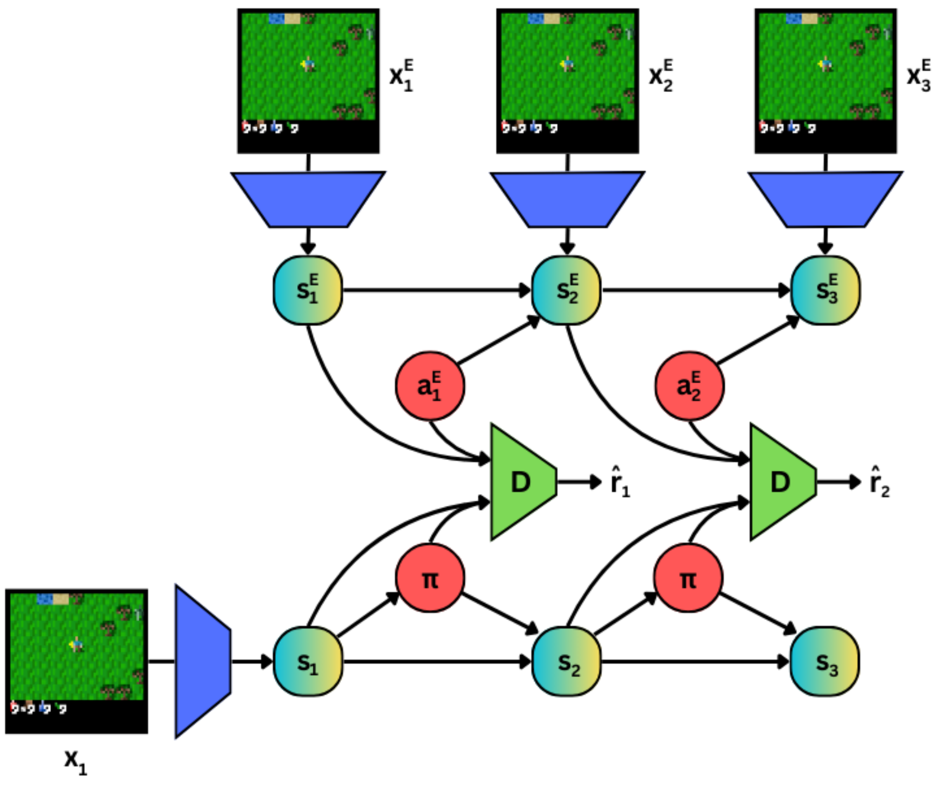


Figure 4.1: Adversarial policy learning in latent space. x_t^E in the upper row are the inputs belonging to an expert trajectory, s_t are model states (concatenation of h_t and z_t), the lower rows depict the policy rollout. D is the discriminator.

Luo, & Levine, 2017), and $r(s, a) = \log(D(s, a))$. The experiments have shown that AIRL reward performs best for synthetic demonstrations while GAIL reward is best suited for human demonstrations. GAIL reward optimizes the symmetric Jensen-Shannon divergence, which means this reward provides an equal penalty for doing things the expert never does and for not doing some of the things the expert does. AIRL function optimizes the Kullback–Leibler divergence that is not symmetric, and as a result, it penalizes the policy much more heavily for doing things the expert never does. Additionally, it was hypothesized that the performance of adversarial imitation learning heavily depends on whether the demonstrations were provided by a human or generated by an RL algorithm.

4.2 V-MAIL

Variational model-based adversarial imitation learning, or V-MAIL (Rafailov, Yu, Rajeswaran, & Finn, 2021), is an attempt at uniting the benefits of model-based RL and adversarial imitation learning. The authors claim that model-based approach improves the stability of adversarial training because it enables on-policy learning. However, V-MAIL was only applied to simple control tasks. Hence, its effectiveness on more complex domains, like open-world games, has yet to be investigated.

V-MAIL trains a variational latent-space dynamics model, like Dreamer, and a discriminator. The discriminator is trained alongside actor and critic/value, separately from the world model, generally following the process illustrated in Figure 4.1. The discriminator parameters are updated with the gradient used by Ho and Ermon (2016) in line 2. The discriminator output $D(s, a)$ for imagined trajectories is then used as a learning signal to the policy. Contrary to the original GAIL, which does not use any discriminator regularizer, V-MAIL uses *gradient penalty* (Gulrajani, Ahmed, Arjovsky, Dumoulin, & Courville, 2017). The resulting discriminator loss is as follows:

$$\mathbb{E}_{\pi}[-\log(1 - D(s, a))] + \mathbb{E}_{\pi_E}[-\log(D(s, a))] + \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\| - 1)^2], \quad (4.3)$$

where the last term is the gradient penalty, λ is the penalty coefficient, and $\mathbb{P}_{\hat{x}}$ samples uniformly along straight lines between pairs of points sampled from π and π_E, in this case.

5. Dreamer with Adversarial Imitation Learning

This work follows V-MAIL's approach for integrating adversarial policy learning with Dreamer's architecture. Implementation details aside, V-MAIL's non-adversarial part consists of roughly the same networks as Dreamer: world model including RSSM, actor, and critic. World model learning is not affected by adding the adversarial part, and all loss definitions remain unchanged from the normal Dreamer (Chapter 3.3). In the implementation, the discriminator is considered a part of actor network for simplicity and trained alongside actor and critic, following the process in Figure 4.1. The discriminator is updated using the loss formulation from V-MAIL (Equation 4.3), along with the gradient penalty regularizer. The pseudocode for the resulting learning process is shown in Algorithm 2.

The agent's reward function is changed to the one that uses the discriminator output. Out of the three reward functions investigated by Orsini et al. (2021), the AIRL reward function (Fu et al., 2017), defined as $r(s, a) = \ln(D(s, a)) - \ln(1 - D(s, a))$ was chosen. In the preliminary experiments it was found that $r(s, a) = \ln(D(s, a))$ slowed down the convergence of AIL, while $r(s, a) = -\ln(1 - D(s, a))$ led to a successful but random policy (Figures 7.1 and 7.2). The AIRL reward was accepted as a valid compromise between them.

A distinct aspect of Dreamer that has to be considered when integrating the use of expert data into the algorithm is that the first step of normal agent's trajectories is explicitly marked as the first step of the corresponding trajectory. While this detail might seem insignificant, the "first step" flag is considered in the observe function of the RSSM, which the encoded expert trajectories have to go through in order to be used for training. This aspect was not explored in detail, but the experiments have shown that setting this flag to false seems to lead to a more relaxed training and better scores. However, for consistency purposes the agents were trained on the trajectories with this flag set to true, following the Dreamer's approach to its own trajectories.

2 Dreamer with AIL

```
Initialize dataset $B_\pi$ with random seed episodes
Randomly initialize neural network parameters for Dreamer components and discriminator $D$
for number of iterations do
    for update step do
        // Dynamics learning
        Sample $B$ trajectories of length $L$ from $B_\pi$
        Map inputs to stochastic representations and predict the sequence of representations given past actions
        From model states $s_t$, predict rewards $r_t$ and continuation flags $c_t$
        Reconstruct the inputs
        Update the world model
        // Adversarial policy learning
        Sample $B$ trajectories of length $L$ from expert dataset $B_\mathrm{E}$
        Infer expert latent states $z^\mathrm{E}$ and form expert model states $s^\mathrm{E}$
        Generate latent rollouts (imagine trajectories) $z^\pi$ and form policy model states $s^\pi$
        Update the discriminator $D$ using $s^\mathrm{E}$ and $s^\pi$
        Update the policy $\pi$ using $r_t = \ln(D(s^\pi)) - \ln(1 - D(s^\pi))$
    end for
    // Environment interaction
    Reset environment
    for time step $t = 1..T$ do
        Compute model states from history
        Sample action $a_t$ from the actor model
        Environment step
    end for
    Add experience to $B_\pi$
end for
```

6. Experimental setup

Dreamer provides several pre-defined model sizes: small, medium, large and extra large. The model size is directly proportional to both performance and data-efficiency. A medium size Dreamer model was chosen for the experiments as a compromise between lower computational costs and better performance. Dreamer's hyperparameters were left unaltered and matching those in Hafner et al. (2024).

In the experiments, AIL+Dreamer is compared with two baselines: a random policy and a vanilla BC agent. For each method, three agents are trained on three different seeds of the environment. The BC baselines were trained for a maximum of 30 epochs, stopping the training prematurely to avoid overfitting if convergence was reached early. The AIL+Dreamer models were trained on either 500k or 1M timesteps, depending on the complexity of the task. The agents were evaluated over 5 runs of 100 episodes each.

6.1 Environment

Crafter (Hafner, 2021), a 2D open-world survival game provided as one of the default available environments for Dreamer, was chosen to be the main environment for the experiments. The environment features 22 unique achievements, related to collecting resources or crafting tools (a complete list in Table 8.1). The environment features a hierarchical crafting system: some resources, like stone, can be collected only upon crafting a specific tool; conversely, some tools can be crafted only after collecting a specific resource. The wide variety of achievements, combined with the hierarchical crafting system, creates a complex and generally hard environment. Additionally, Crafter features enemies like zombies and skeletons that can harm the player. Because of this fast-paced game world, surviving in Crafter requires strong generalization abilities.

6.2 Tasks and datasets

The approach was evaluated on both custom and pre-defined tasks. More specifically, two custom tasks were defined to assess basic navigation and interaction skills of an agent: Circle and Wood. The aim of the Circle task is learning the behavior of walking in circles. While deceivingly simple, this task might be challenging for an agent because 1) it does not include actions that directly affect the environment, 2) it includes a certain looped sequence of actions, and 3) the environment introduces natural moving disruptions to the walking pattern such as cows and zombies. Conversely, the aim of the Wood task is learning to consistently chop trees. Contrary to the Circle task, it focuses on the actions that directly affect the environment and are reflected visually (i.e. when chopped, a tree disappears). Intuitively, it would be easier for an agent to learn to succeed in the Wood task.

16 human demonstrations of Crafter gameplay were manually recorded for these two tasks. For the Circle dataset, the demonstrations feature the behavior of running around in 3x3 circles in a world with the same seed; the circle patterns are sometimes interrupted by cows and zombies but mostly preserved. For the Wood dataset, the demonstrations feature walking over to the observable trees and chopping them. The datasets feature no other behavior besides mentioned.

In addition to these two tasks that mostly serve to assess the basic learning and generalization capabilities of the model, another goal was to see whether the model can learn meaningful behavior based on a more "chaotic" dataset of natural human gameplay demonstrations. This goal is reflected in the broader Survival task. The author of the Crafter environment provides a human gameplay dataset of 100 episodes total. However, for more efficient training, 16 episodes that feature a wide range of achievements (see Table 8.1) and are long enough to be considered successful (200+ steps) were selected to be used for the Survival task. Conversely, the BC baselines were trained on the full dataset.

6.3 Evaluation

Crafter natively defines an explicit numerical reward signal, even though AIL+Dreamer does not use it. Additionally, the environment provides a measure called Crafter score, defined as the geometric mean of achievement success rates. Following the evaluation guidelines of Hafner (2021), Crafter score is reported for each trained agent. For the

Survival task, the success rates of achievements can provide meaningful insights into an agent's behavior as well.

In addition to numerical evaluation, the agent's behavior is also evaluated visually. While it is not an objective method of evaluation, in certain cases it is beneficial: even though an agent might perform poorly in numerical terms, its behavior can visually resemble that of a human player, which would be considered a meaningful result. Moreover, the performance on certain tasks (such as Circle) might not be related to a numerical score.

7. Results

The AIRL reward was chosen based on the experiments results in Figures 7.1 and 7.2. Since the purpose of any learning is to learn behaviors, which are not chaotic and random but rather directed and purposeful instead, random policy is not what we want to achieve in an agent. The measure of randomness depends on the measure of entropy (Jaynes, 1968), which in the case of machine learning measures the unpredictability of the actions agent takes under given policy. In the experiments, agents that used GAIL reward function tended to fare not so differently from others in the very beginning of their training. However, this reward function soon led to a random policy and was deemed unsuitable.

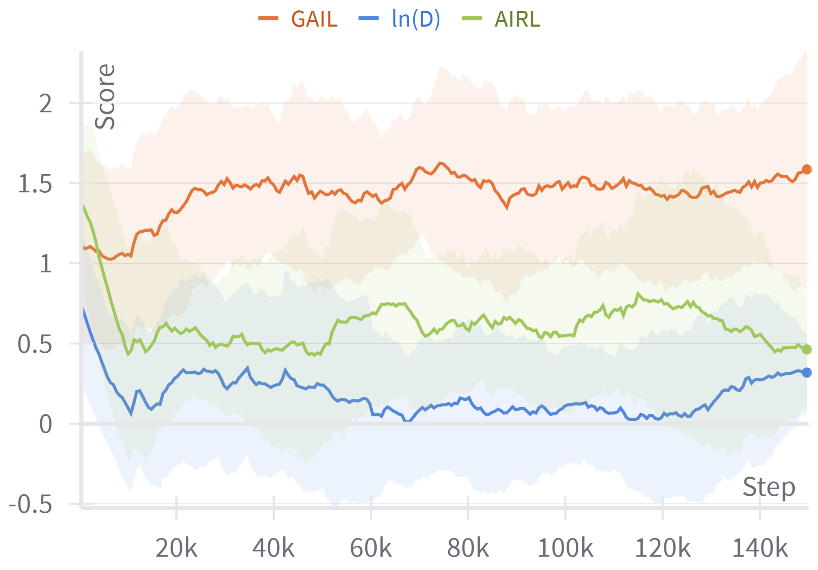


Figure 7.1: Episode scores comparison for the reward functions.

The three BC agents trained on the Circle dataset mostly display the behavior of walking towards the bottom of the screen, with two of them also occasionally turning and going in the opposite direction. The cause for the agents' inability to learn the behavior might be the temporally-extended nature of the task: the vanilla BC baseline does not retain any past information, making it hard for the agent to complete tasks that require memory.

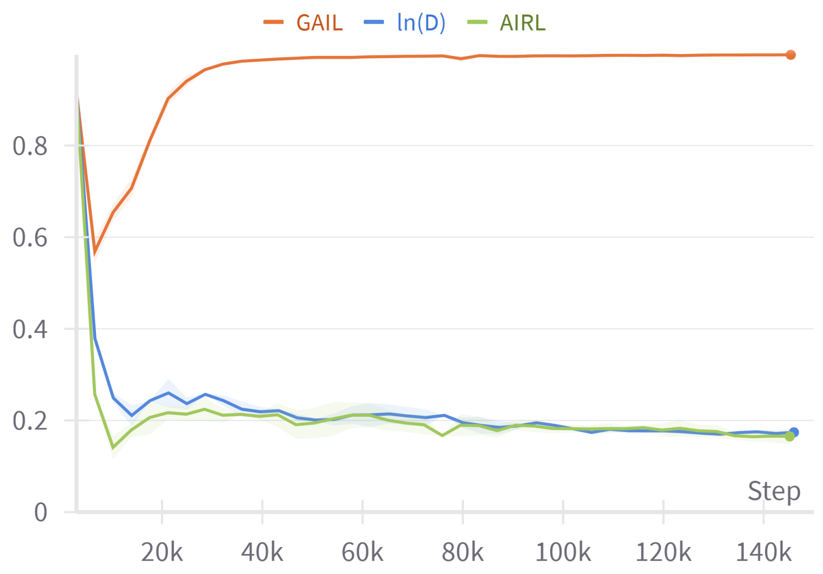


Figure 7.2: Policy randomness measure comparison for the reward functions.

All three Dreamer+AIL agents trained on the Circle dataset rarely exhibit complete patterns of walking in circles. An example of what can be considered a complete circle pattern is shown in Figure 7.3. However, the agents tend to do considerably fewer actions aside from walking around than a random agent would, which is reflected in the overall score (a mean of 0.56).

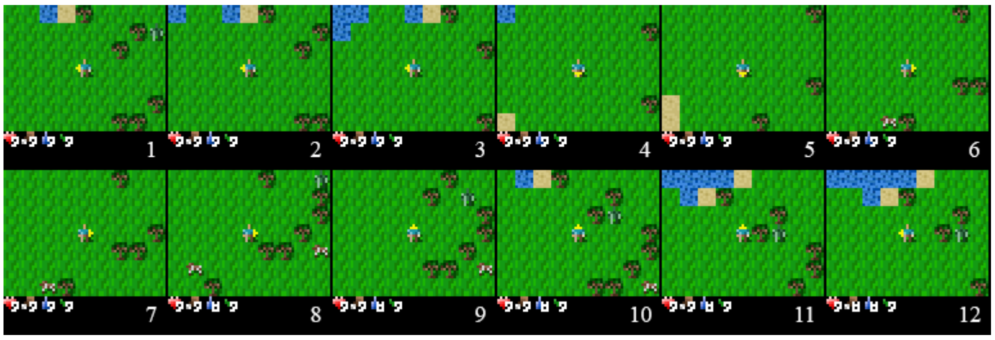


Figure 7.3: An example of circle pattern exhibited by one of the models trained on the Circle dataset.

The three BC agents trained on the Wood dataset were not able to learn the wood collecting behavior. Surprisingly, the Dreamer agents trained on the Wood dataset achieve the best score among the trained models with a mean of 1.11. The success rate of the Collect Wood achievement is 35.72%, compared to 25% in random agent. However, it is considerably less than 52.64% achieved by the Survival models. Figures 7.4 and 7.5 contain scores and success rates for Wood and Circle models compared to the BC survival baseline and the random policy.

Out of three BC agents trained on the full Survival dataset, two were able to achieve

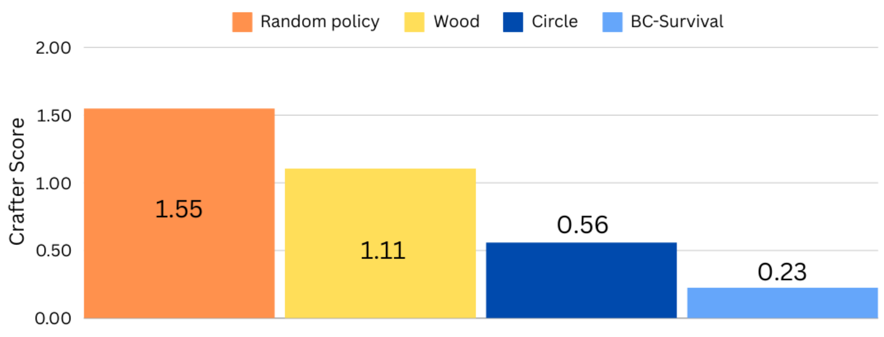


Figure 7.4: Crafter scores for Wood and Circle models.

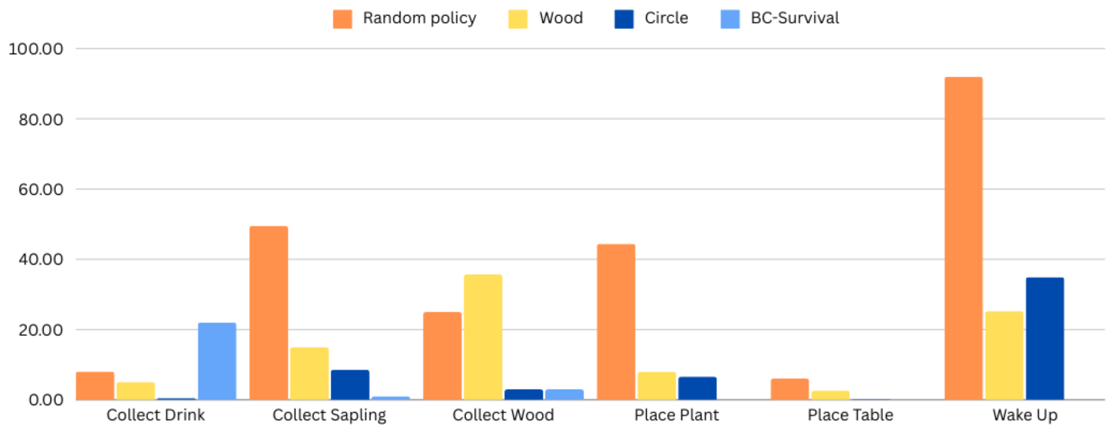


Figure 7.5: Achievement success rates for Wood and Circle models.

some basic goals. In particular, agents were able to drink water and, occasionally, chop wood. On the contrary, the third agent was able to complete some basic exploration, but did not complete any achievements. The Dreamer agents successfully learn to collect wood, place table and occasionally make wood pickaxes and swords. They tend to do fewer actions associated with plants (collecting saplings or placing plants) as well as sleep less than a random agent; this result is more or less consistent with expert gameplay, as humans rarely choose to do these actions. On the other hand, the agents seem to have trouble learning to drink water compared to random and BC agents, which is different from human behavior. Figure 7.6 contains Crafter scores comparison for Survival models. Figures 7.7 and 7.8 contain success rates for the more prominent achievements of Survival models.

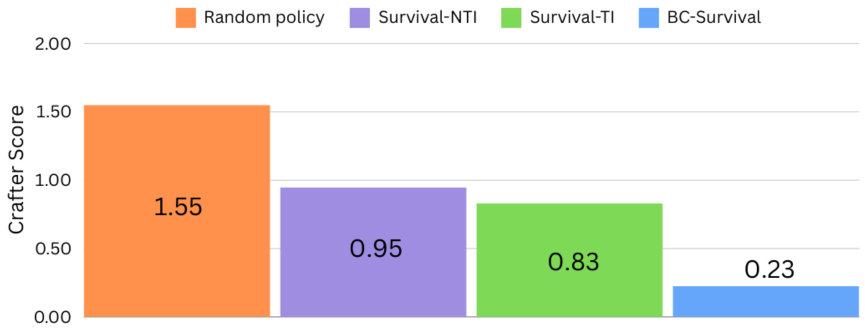


Figure 7.6: Crafter scores for Survival models. Survival-TI has the first step of an expert trajectory marked as the first step, Survival-NTI makes no distinction.

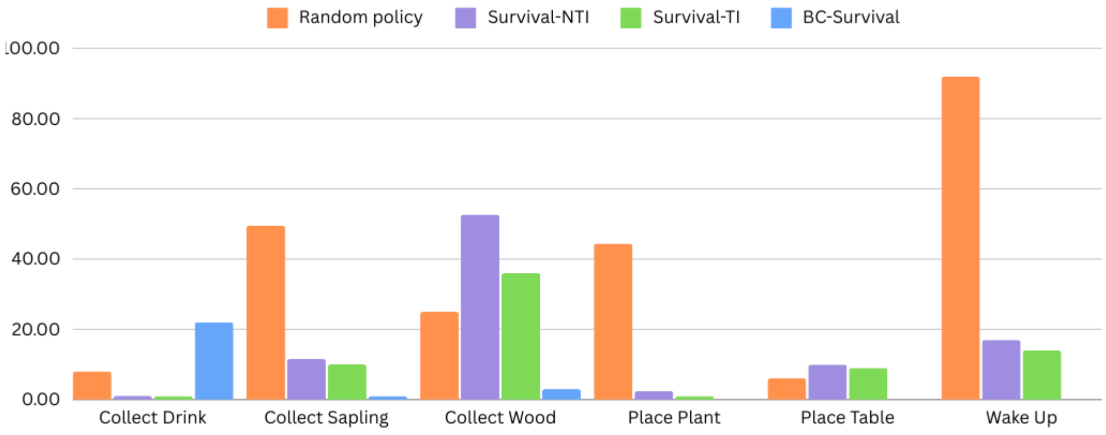


Figure 7.7: Achievement success rates for Survival models (Part 1).

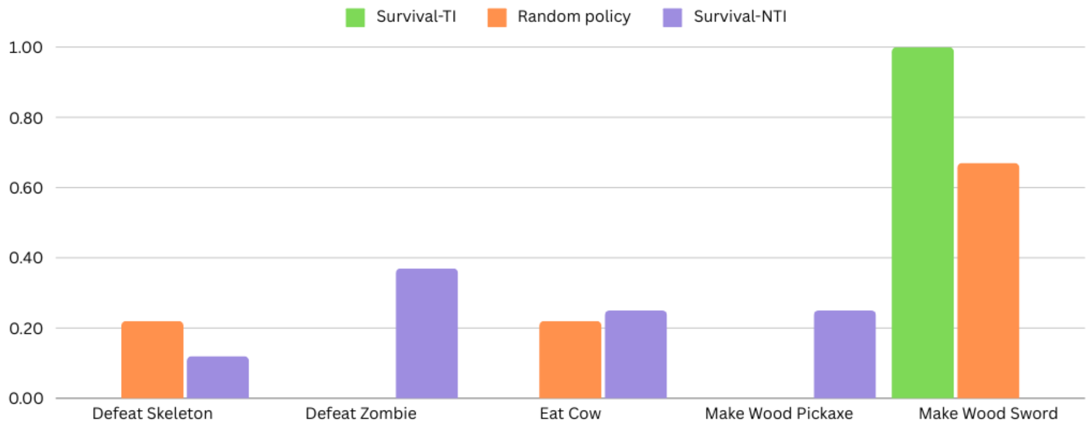


Figure 7.8: Achievement success rates for Survival models (Part 2).

8. Conclusions

Generally, the experiments have shown the agents to be able to learn expected behaviors. The agents' performance, compared to a random agent's one, is consistent with human gameplay, aside from not being able to learn to drink water. The answer to the first research question is therefore positive: it is possible to bring an agent's behavior closer to human behavior using imitation learning in its imagination. However, survival abilities of the agents leave something to be desired in terms of scores. There are multiple possible causes for this. Agents might have difficulty predicting the movements of nondeterministic elements like moving enemies (Lin, 1992) and learning to consistently defend themselves as a result. Some achievements are only unlocked once in an expert episode, and with the random trajectory sampling there is no guarantee the agent will get a certain part of an episode often enough to affect its policy. There is a multitude of possible solutions to this problem, ranging from more careful manual selection of the trajectories/episodes to implementing a search over expert dataset to obtain the most relevant actions (Malato & Hautamaki, 2024).

Additionally, Dreamer ensuring sufficient exploration might have both positive and negative consequences. It allows the agent to try out more actions even if they are not featured in the expert trajectories that often, but it also makes it more difficult to discern the *learned* behavior from the exploratory one. On a more technical note, there is a need for conducting more experiments and training the agents for the full 1M steps budget. It would also be beneficial to use larger models, as they provide performance gains, and Hafner et al. (2024) use an extra large Dreamer model on Crafter.

The answer to the second research question is less certain and warrants further research in this area. The design choices seem to be interconnected and it is difficult to predict how the different combinations and changes will affect the training. The environments' specifics certainly play their own part: a solution that worked excellently for robotics simulation environments might not work as well for open-world games.

One aspect that requires further exploration is the choice of discriminator regularizer, which might be more important when dealing with complex tasks (Orsini et al., 2021).

A regularizer that performed best for the discriminator training was found by Orsini et al. (2021) to be spectral norm (Miyato, Kataoka, Koyama, & Yoshida, 2018). Another regularization technique worth mentioning is variational discriminator bottleneck (Peng, Kanazawa, Toyer, Abbeel, & Levine, 2018), designed specifically for adversarial learning methods and based on the idea of constraining information flow. A third approach that could be tried is converting the discriminator into a SAN-type discriminator instead of the one used in GANs. Takida et al. (2024) question whether the standard minimax objective used in GANs actually brings the generated distribution closer to the target one, and propose a different training scheme for GANs, called slicing adversarial network. The results shown in the paper are promising, and converting a GAN into a SAN only requires small modifications.

References

Arjovsky, M., & Bottou, L. (2017). *Towards principled methods for training generative adversarial networks.* Retrieved from `https://arxiv.org/abs/1701.04862`

Bain, M., & Sammut, C. (1995). A framework for behavioural cloning. In *Machine intelligence 15.* Retrieved from `https://api.semanticscholar.org/CorpusID:10738655`

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2012). The arcade learning environment: An evaluation platform for general agents. *CoRR*, *abs/1207.4708*. Retrieved from `http://arxiv.org/abs/1207.4708`

Bengio, Y. (2009).
doi: 10.1561/2200000006

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017, April). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, *112*(518), 859–877. Retrieved from `http://dx.doi.org/10.1080/01621459.2017.1285773` doi: 10.1080/01621459.2017.1285773

Bratko, I., Urbančič, T., & Sammut, C. (1995). Behavioural cloning: Phenomena, results and problems. *IFAC Proceedings Volumes*, *28*(21), 143-149. Retrieved from `https://www.sciencedirect.com/science/article/pii/S1474667017467164` (5th IFAC Symposium on Automated Systems Based on Human Skill (Joint Design of Technology and Organisation), Berlin, Germany, 26-28 September)

Buesing, L., Weber, T., Racanière, S., Eslami, S. M. A., Rezende, D. J., Reichert, D. P., . . . Wierstra, D. (2018). Learning and querying fast generative models for reinforcement learning. *CoRR*, *abs/1802.03006*. Retrieved from `http://arxiv.org/abs/1802.03006`

Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., . . . Anandkumar, A. (2022). *Minedojo: Building open-ended embodied agents with internet-scale knowledge.* Retrieved from `https://arxiv.org/abs/2206.08853`

Forrester, J. W. (1971). Counterintuitive behavior of social systems. *Technological Forecasting and Social Change*, *3*, 1-22. Retrieved from `https://www`

.sciencedirect.com/science/article/pii/S004016257180001X doi: https://doi.org/10.1016/S0040-1625(71)80001-X

Fu, J., Luo, K., & Levine, S. (2017). Learning robust rewards with adversarial inverse reinforcement learning. *CoRR*, *abs/1710.11248*. Retrieved from http://arxiv.org/abs/1710.11248

Ganguly, A., & Earp, S. W. F. (2021). An introduction to variational inference. *CoRR*, *abs/2108.13083*. Retrieved from https://arxiv.org/abs/2108.13083

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). *Generative adversarial networks.*

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. *CoRR*, *abs/1704.00028*. Retrieved from http://arxiv.org/abs/1704.00028

Ha, D., & Schmidhuber, J. (2018). World models. *CoRR*, *abs/1803.10122*. Retrieved from http://arxiv.org/abs/1803.10122

Hafner, D. (2021). Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*.

Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=S1lOTC4tDS

Hafner, D., Lillicrap, T. P., Fischer, I., Villegas, R., Ha, D., Lee, H., & Davidson, J. (2018). Learning latent dynamics for planning from pixels. *CoRR*, *abs/1811.04551*. Retrieved from http://arxiv.org/abs/1811.04551

Hafner, D., Lillicrap, T. P., Norouzi, M., & Ba, J. (2020). Mastering atari with discrete world models. *CoRR*, *abs/2010.02193*. Retrieved from https://arxiv.org/abs/2010.02193

Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2024). *Mastering diverse domains through world models.*

He, J. Z.-Y., & Dragan, A. D. (2021). *Assisted robust reward design.*

Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. *CoRR*, *abs/1606.03476*. Retrieved from http://arxiv.org/abs/1606.03476

Hussein, A., Gaber, M. M., Elyan, E., & Jayne, C. (2017, apr). Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, *50*(2). Retrieved from https://doi.org/10.1145/3054912

Jaynes, E. T. (1968). Prior probabilities. *IEEE Transactions on Systems Science and Cybernetics*, *4*(3), 227-241. doi: 10.1109/TSSC.1968.300117

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The malmo platform for artificial intelligence experimentation. In S. Kambhampati (Ed.), *Proc. 25th international joint conference on artificial intelligence* (p. 4246). Palo Alto, California, USA: AAAI Press.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1998). An introduction to variational methods for graphical models. In M. I. Jordan (Ed.), *Learning in graphical models* (pp. 105–161). Dordrecht: Springer Netherlands. Retrieved from `https://doi.org/10.1007/978-94-011-5014-9_5` doi: 10.1007/978-94-011-5014-9_5

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*(1), 99-134. Retrieved from `https://www.sciencedirect.com/science/article/pii/S000437029800023X` doi: https://doi.org/10.1016/S0004-3702(98)00023-X

Kelley, H. J. (1960). Gradient theory of optimal flight paths. *Ars Journal*, *30*(10), 947–954.

Kingma, D. P., & Welling, M. (2022). *Auto-encoding variational bayes.*

Lin, L.-J. (1992, May). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*(3–4), 293–321. Retrieved from `http://dx.doi.org/10.1007/BF00992699` doi: 10.1007/bf00992699

Malato, F., & Hautamaki, V. (2024). *Online adaptation for enhancing imitation learning policies.* Retrieved from `https://arxiv.org/abs/2406.04913`

Milani, S., Kanervisto, A., Ramanauskas, K., Schulhoff, S., Houghton, B., Mohanty, S., . . . Shah, R. (2023). *Towards solving fuzzy tasks with human feedback: A retrospective of the minerl basalt 2022 competition.*

Mitchell, T. (2015). Chapter 3 generative and discriminative classifiers : Naive bayes and logistic regression machine learning.. Retrieved from `https://api.semanticscholar.org/CorpusID:267911086`

Miyato, T., Kataoka, T., Koyama, M., & Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *CoRR*, *abs/1802.05957*. Retrieved from `http://arxiv.org/abs/1802.05957`

Ng, A., & Jordan, M. (2001). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in neural information processing systems* (Vol. 14). MIT Press. Retrieved from `https://proceedings.neurips.cc/paper_files/paper/2001/file/7b7a53e239400a13bd6be6c91c4f6c4e-Paper.pdf`

Ng, A. Y., & Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the seventeenth international conference on machine learning* (p. 663–670). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Orsini, M., Raichuk, A., Hussenot, L., Vincent, D., Dadashi, R., Girgin, S., . . . Andrychowicz, M. (2021). What matters for adversarial imitation learning? *CoRR*, *abs/2106.00672*. Retrieved from `https://arxiv.org/abs/2106.00672`

Peng, X. B., Kanazawa, A., Toyer, S., Abbeel, P., & Levine, S. (2018). Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow. *CoRR*, *abs/1810.00821*. Retrieved from `http://arxiv.org/abs/1810.00821`

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., ... Zaremba, W. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *CoRR*, *abs/1802.09464*. Retrieved from `http://arxiv.org/abs/1802.09464`

Popov, I., Heess, N., Lillicrap, T. P., Hafner, R., Barth-Maron, G., Vecerík, M., ... Riedmiller, M. A. (2017). Data-efficient deep reinforcement learning for dexterous manipulation. *CoRR*, *abs/1704.03073*. Retrieved from `http://arxiv.org/abs/1704.03073`

Rafailov, R., Yu, T., Rajeswaran, A., & Finn, C. (2021). Visual adversarial imitation learning using variational models. *Neural Information Processing Systems*.

Ross, S., & Bagnell, D. (2010, 13–15 May). Efficient reductions for imitation learning. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Vol. 9, pp. 661–668). Chia Laguna Resort, Sardinia, Italy: PMLR. Retrieved from `https://proceedings.mlr.press/v9/ross10a.html`

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1970, 02). Learning to fly.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, *3*(6), 233-242. Retrieved from `https://www.sciencedirect.com/science/article/pii/S1364661399013273` doi: https://doi.org/10.1016/S1364-6613(99)01327-3

Sutton, R. S. (1991, jul). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, *2*(4), 160–163. Retrieved from `https://doi.org/10.1145/122344.122377` doi: 10.1145/122344.122377

Sutton, R. S., & Barto, A. G. (2014-2015). *Reinforcement learning: An introduction*. Cambridge, Massachusetts: The MIT Press.

Takida, Y., Imaizumi, M., Shibuya, T., Lai, C.-H., Uesaka, T., Murata, N., & Mitsufuji, Y. (2024). *San: Inducing metrizability of gan with discriminative normalized linear layer.* Retrieved from `https://arxiv.org/abs/2301.12811`

Watter, M., Springenberg, J. T., Boedecker, J., & Riedmiller, M. A. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. *CoRR*, *abs/1506.07365*. Retrieved from `http://arxiv.org/abs/1506.07365`

Zhang, A., Wu, Y., & Pineau, J. (2018). Natural environment benchmarks for reinforcement learning. *CoRR*, *abs/1811.06032*. Retrieved from `http://arxiv.org/abs/1811.06032`

	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11	e12	e13	e14	e15	e16
Collect Coal	4	2	4	7	2	2	4	4	9	5	4	9	2	5	5	2
Collect Diamond	0	0	1	1	1	0	2	0	1	1	1	0	1	1	2	0
Drink	30	34	33	26	0	18	141	14	38	41	121	119	83	130	115	7
Collect Iron	2	4	3	2	1	2	5	5	4	2	9	4	5	2	4	2
Collect Sapling	5	2	6	1	0	3	10	0	1	2	8	7	5	8	8	2
Collect Stone	19	44	18	11	25	29	69	18	35	41	72	84	48	72	41	11
Collect Wood	8	14	17	13	15	12	17	10	12	11	18	26	29	20	11	9
Defeat Skeleton	0	3	0	0	0	0	4	0	2	1	3	4	4	2	4	1
Defeat Zombie	1	4	9	0	1	1	7	2	4	1	14	12	11	6	8	2
Eat Cow	2	4	4	2	1	5	11	2	5	5	9	13	10	6	6	0
Eat Plant	0	0	0	0	0	1	6	0	0	0	12	5	3	17	7	0
Iron Pickaxe	1	1	1	1	1	1	2	1	1	1	2	1	1	1	2	1
Iron Sword	1	1	1	1	0	1	2	1	1	1	1	1	1	1	2	1
Stone Pickaxe	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Stone Sword	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Wood Pickaxe	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Wood Sword	0	0	1	1	0	0	1	1	1	0	1	1	1	1	1	0
Place Furnace	1	1	1	1	1	1	2	1	1	1	2	1	1	1	1	1
Place Plant	4	0	6	1	0	3	10	0	0	2	6	6	4	5	6	0
Place Stone	12	9	3	1	1	9	31	1	10	9	21	20	17	22	22	1
Place Table	1	2	1	1	1	2	3	1	2	2	3	1	3	3	1	1
Wake Up	1	2	2	2	0	2	8	1	4	3	10	7	5	10	6	0

Table 8.1: Number of times an achievement has been unlocked in an expert episode.