GENERALIZED SKILL LEARNING, SAFETY, AND EXPLORATION WITH FLOW-BASED MODELS

Alvin Zhang
Electrical Engineering and Computer Science
UC Berkeley
alvinz@berkeley.edu

ABSTRACT

In order to train fully autonomous, unsupervised agents, reinforcement learning algorithms must be able to operate without human-specified domain knowledge of the task, agent, or environment. To this end, we propose a framework which utilizes a flow-based model to generate goals directly in an agent's state space, and trains a policy to achieve these goals. This allows us to encourage exploration by sampling goals from less-visited regions of the state space or generating novel goals. Further, we introduce a safety-motivated formulation which assumes pessimal dynamics while training our agents. Empirically, we demonstrate that goal generation in the full state space allows unsupervised agents to learn pose- and velocity-based behaviors in two simulated robotic environments.

1 INTRODUCTION

Recent advances in reinforcement learning have made great strides in training self-sufficient, general-purpose agents which can explore their environment and learn to perform a variety of tasks in an unsupervised fashion. In this work, we consider the problem of training an unsupervised agent capable of achieving arbitrary goals in its state space. Without prior knowledge of the environment, this requires that the agent explores and sets novel goals for itself during training.

However, previous work in exploration with goal-conditioned policies has relied on a human-specified goal space and reward function. In particular, the goal space in Hindsight Experience Replay (HER) (Andrychowicz et al. (2017)) and GoalGAN (Florensa et al. (2018)) are 2- or 3-dimensional position vectors of an object in the environment. As such, these agents are incapable of learning skills requiring pose and/or velocity information, such as balancing or spinning. We instead specify goals directly in the agent's full state space.

Further, in order to encourage our agent to explore novel regions of its environment and develop diverse behaviors, we fit a flow-based generative model to the set of states our agent has experienced. During training, we then automatically generate goals from regions of the state space which the model estimates to have lower density.

Lastly, we introduce a safety-motivated modification to the Stochastic Value Gradient (SVG) framework (Heess et al. (2015)). If we have access to a model of the environment's dynamics, we can optimize our policy while assuming adversarial dynamics or sensor noise, thus teaching the agent to avoid potentially catastrophic outcomes.

In summary, our main contributions are 1) automatic goal generation for exploration in the full state space, and 2) a method for training an agent under a pessimal dynamics model.

2 BACKGROUND

2.1 REINFORCEMENT LEARNING

We consider a discrete-time, goal-conditioned Markov Decision Process (MDP) defined by states and goals $s, g \in \mathcal{S}$, actions $a \in \mathcal{A}$, a transition probability distribution $d(s'|s, a)$, and a reward

function $r(s, a, s', g) \in \mathbb{R}$. Our goal is to train a policy $\pi(a|s, g)$ which maximizes the expected sum of rewards.

2.2 Stochastic Value Gradients

Value gradient methods optimize a policy by propagating gradients through a learned dynamics model and value function. Stochastic value gradients (Heess et al. (2015)) utilize the reparameterization trick to enable gradient propagation in the case of a stochastic policy and/or dynamics model.

2.3 Hindsight Experience Replay

Hindsight experience replay (Andrychowicz et al. (2017)) is a technique for increasing the sample efficiency and learning signal for goal-conditioned reinforcement learning. Since the environment's dynamics are fixed, we can relabel the states achieved along a trajectory as goals for training purposes. Combined with an off-policy reinforcement learning algorithm, this has been shown to increase sample efficiency and allow learning even if rewards are extremely sparse.

2.4 Flow-Based Models

Flow-based models, introduced in Dinh et al. (2016), learn an invertible transformation f between the data $x \in \mathbb{R}^n$ and a latent variable $z \sim Z = \mathcal{N}(0, I_n)$. If $Z = f_{\text{enc}}(X)$ and $f^{-1} = f_{\text{dec}}$, $p_X(x)$ is given by the change-of-variables formula

$$\log(p_X(x)) = \log(p_Z(z)) + \log(\Big| \det \frac{\partial f_{\text{enc}}(x)}{\partial x} \Big|). \tag{1}$$

In this work, we utilize flow-based networks to parameterize models of the state distribution $f(s)$, the dynamics distribution $d(s'|s, a)$, and the policy $\pi(a|s, g)$. We construct our neural networks by stacking invertible matrices (as introduced in Kingma & Dhariwal (2018)) with smooth parameterized ReLU nonlinearities (refer to Appendix B for details).

3 Related Work

Our work is closely related to prior approaches in intrinsic motivation (Deci & Ryan (2010)), where agents are incentivized to explore or learn skills in the absence of a provided reward function.

Curiosity-driven Exploration by Self-supervised Prediction (Pathak et al. (2017)) and DIAYN (Eysenbach et al. (2018)) have demonstrated the feasibility of unsupervised exploration and skill aquisition in an environment. However, these works do not follow the goal-conditioned reinforcement learning framework. Their agents therefore require post-algorithmic retraining or fine-tuning in order to accomplish specific tasks.

On the other hand, GoalGAN (Florensa et al. (2018)) utilizes the goal-conditioned framework, but requires a human-specified goal subspace of the state subspace, which limits the range of behaviors learnable by the policy. A recent work, Skew-Fit (Pong et al. (2019)), utilizes a similar approach to ours and operates in the high-dimensional observation space of images. However, it simplifies the control problem for the agent by making the action space (x, y, z) positions or velocities of an end-effector. We demonstrate that this approach can succeed in high-dimensional action spaces with complex, nonlinear, contact dynamics.

4 Methods

We use the SVG framework with HER (Andrychowicz et al. (2017)) to train our agent. Additionally, we train invertible, flow-based generative models to maximize the log-likelihood of the state distribution $f(s)$, the dynamics distribution $d(s'|s, a)$, and an estimate of a safe policy $\pi^{\text{safe}}(a|s, g)$.

4.1 Goal Generation for Exploration

In order to encourage the agent to explore previously less-visited areas of the state space, we wish to sample goals for the agent in those less-visited areas. Therefore, during each episode, we sample m potential goals $g = f_{\text{dec}}(z)$, $z \sim Z = \mathcal{N}(0, I)$. We then select the goal with the least likelihood $p_f(g)$ under f for the agent for that episode.

4.2 Choice of Reward Function

Following Florensa et al. (2019), we wish to use the minimum number of timesteps needed to reach a goal as the learning signal for our RL agent. This corresponds to the optimal (undiscounted) value function being equal to the negative distance (in number of actions) between a state and a goal. Combined with a dynamics model, this allows an agent to perform planning towards a goal directly in the action space.

However, rather than using the binary reward $r(s, a, s', g) = \mathbb{1}\{s' == g\}$, we wish to award "partial credit" for achieving states similar to the goal state. This is necessary when working with the full state space (including velocities), as the sampling rate of the system may otherwise lead to improperly rewarded actions. For example, consider a one-dimensional agent operating at 1 Hz attempting to reach $g = (x, \dot{x}) = (0.5, 1)$. If the agent starts at $s = (0, 1)$ and the environment is frictionless, the agent will achieve the goal if it applies no force. However, under the binary reward specification, the agent would receive no reward, as $s' = (1, 1) \neq g = (0.5, 1)$.

We therefore choose the reward function to be $r(s, a, s', g) = -1 + \exp(-k||f_{\text{enc}}(s') - f_{\text{enc}}(g)||^2)$ for some constant k. Increasing k has the effect of removing reward shaping and making the signal more sparse; as k approaches infinity, the original binary reward signal is recovered (since f is an invertible function).

4.3 Learning a Safety-Based Policy

We adopt the undiscounted value function formulation from SVG:

$$V^\pi(s, s) = 0, V^\pi(s, g \neq s) = \mathbb{E}_{s' \sim d(s,a), a \sim \pi(s,g)}[r(s, a, s', g) + V^\pi(s', g)]. \tag{2}$$

However, rather than training our policy to maximize $V^\pi(s, g)$, we train

$$\pi^{\text{safe}}(s, g) = \arg\max_{a \in \mathcal{A}} \min_{s' \in \text{supp}(d(s,a))} [r(s, a, s', g) + V^{\pi^{\text{safe}}}(s', g)], \tag{3}$$

where $\text{supp}(d(s, a))$ denotes the support of $d(s, a)$. If $d(s, a)$ has infinite support, this formulation can be modified to select s' from the n^{th} percentile of $[r(s, a, s', g) + V^{\pi^{\text{safe}}}(s', g)]$.

Compared to optimizing for V^π, learning $V^{\pi^{\text{safe}}}$ has the advantage of being risk-averse with respect to the dynamics model. This formulation is enabled by our use of the shaped reward function: in a stochastic environment, an agent would never be able to receive the binary reward assuming adversarial dynamics or sensor noise. However, the shaped reward is more robust to these effects.

In practice, we do not solve for the optimal a and s' in the formulation of π^{safe}. Instead, we evaluate a set of generated candidate actions $\mathcal{A}_{cand}$ under N rollouts of our learned dynamics model, and update the policy to maximize the log-likelihood $\pi^{\text{safe}}(a_{\text{best}}|s, g)$ of the best action.

Our full algorithm is presented in Appendix A.

5 Experiments

We test our algorithm in two standard robotic environments, Cartpole and Ant, using the MuJoCo simulator (Todorov et al. (2012)). We use the XML files from the DeepMind Control Suite (Tassa et al. (2018)), with one modification. In order to further our long-term objective of developing agents capable of life-long learning in the real world, we do not reset the environment. This in turn requires that the agent not have any "dead" states from which it is unable to recover (such as the Ant flipping over). We therefore extend the control range for the Ant's joints to be symmetric about the xy-plane.

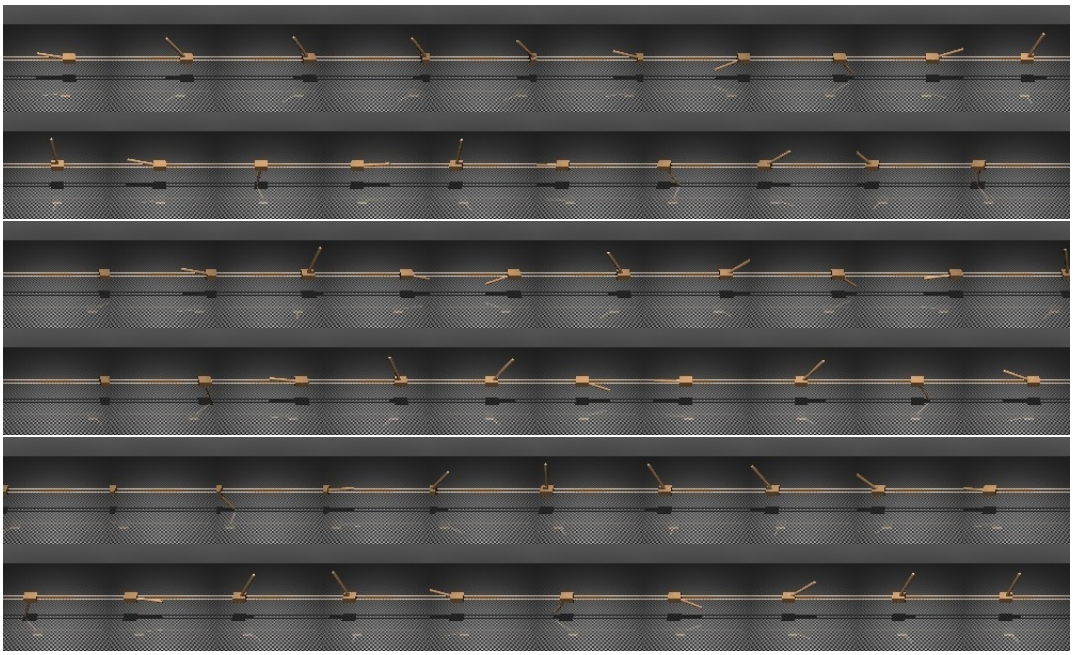

Figure 1: The Cartpole agent is able to keep the pole swinging to the left (top) or the right (middle), or to briefly balance it (bottom).

We find that we are able to learn general agents capable of achieving a variety of desired behaviors. Although our Cartpole agent (Figure 1) is unable to perfectly balance the pole, it takes an active approach towards balancing and can continuously spin the pole in either direction, demonstrating a more general, human-like distribution of skills. Our Ant agent (Figure 2) exhibits the ability to perform a diverse set of actions, including flips, jumping, spinning, handstands, and locomotion. Videos of our experiments are available at this site.

5.1 Ablation Studies

We conduct ablation studies on a toy environment (Figure 3) to determine the effects of exploration using our generative model and our safety-based policy. In this Ramp Pointmass environment, a ramp placed on the right hand side of the environment affords access to a ledge at the top. Noise (sampled from a truncated normal distribution) is added to the agent's action at every timestep; however, the environment is bounded by walls, so that the agent cannot fall off of the ramp on the right or the ledge on the top. We show that exploration reduces the time required for the agent to first reach the ledge. We also find that a safety-based policy learns more quickly than an expectation-based policy to hug the right wall when climbing the ramp.

6 Conclusions and Future Work

We have demonstrated that completely unsupervised agents can explore their environment and learn a diverse set of behaviors while prioritizing safety. In particular, we have shown that a goal-driven agent benefits from exploration through state density estimation in the state space, and introduced a safety-based modification to the SVG framework which results in more efficient learning under noisy conditions. Future work includes investigating the application of these techniques for image-based domains and representation learning. Lastly, we have developed several techniques for reducing the training time for the agent; these are detailed in Appendix A and we hope to provide a more thorough analysis of these in the future.

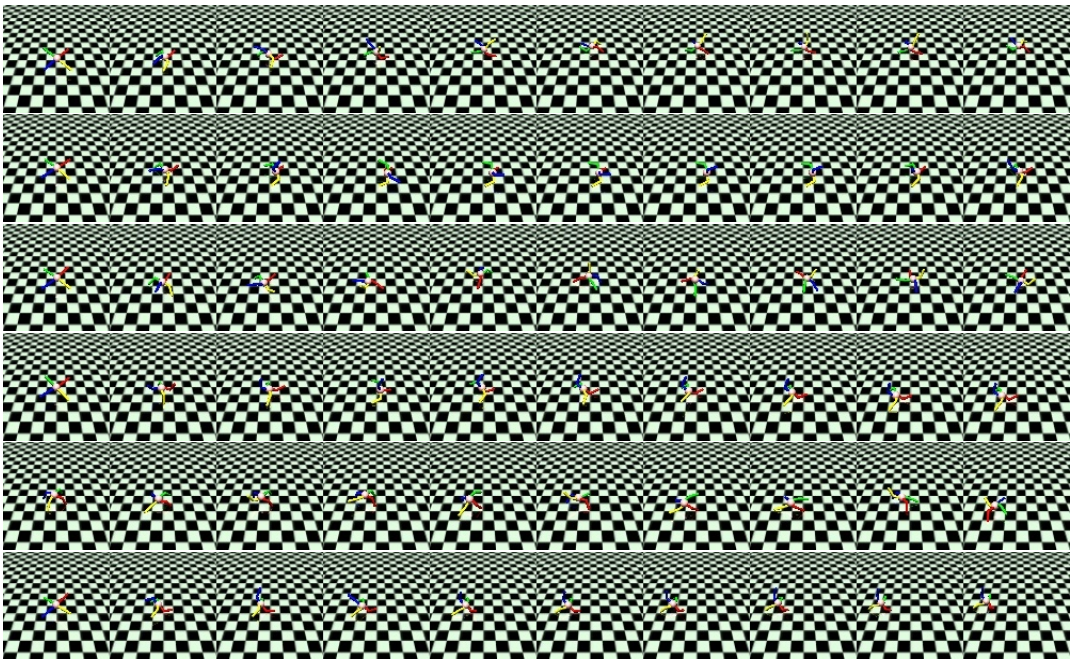

Figure 2: The Ant agent is capable of a variety of skills, including (top-to-bottom) flipping, handstands, spinning, jumping, and two modes of locomotion. We encourage readers to view videos of the agent here.

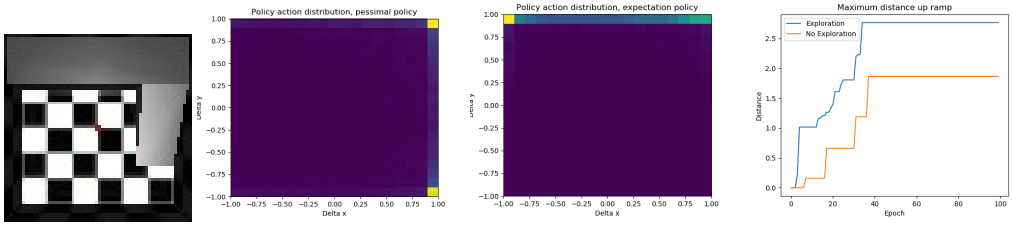

Figure 3: Far left: The Ramp Pointmass environment. A good agent should explore the environment and learn to reach the top ledge while operating under noisy conditions. Middle left: The policy action distribution after 100 epochs for an agent moving up the ramp under the pessimal policy. Note that it strongly avoids moving to the left, as doing so could lose much more progress than just moving down the ramp. Middle right: The policy action distribution after 100 epochs for an agent moving up the ramp under the non-pessimal policy. In contrast to the pessimal policy, this agent does not avoid moving to the left. Far right: The maximum distance up the ramp obtained by agents sampling goals from an exploratory distribution or the state distribution.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Edward L Deci and Richard M Ryan. Intrinsic motivation. *The corsini encyclopedia of psychology*, pp. 1–2, 2010.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, pp. 1514–1523, 2018.

Carlos Florensa, Jonas Degrave, Nicolas Heess, Jost Tobias Springenberg, and Martin Riedmiller. Self-supervised learning of image embedding for continuous control. *arXiv preprint arXiv:1901.00943*, 2019.

Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10236–10245, 2018.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

A TRAINING ALGORITHM

Since the publication of SVG (Heess et al. (2015)), a number of improvements to Q-learning have been proposed. We find that we can apply the intuition for many of these improvements to the SVG framework. Double DQN (Van Hasselt et al. (2016)) noted that Q-learning suffered from overestimation of Q-values and proposed to ameliorate the problem by training two Q-functions. This reduced the optimism bias of Q-learning agents; our safety-based policy extends this idea by assuming pessimal dynamics, similarly to TD3 (Fujimoto et al. (2018)).

TD3 also notes that the variance of the value function estimator can cause instability during training. They recommend using a slowly-updating target network and multiple value updates per policy update; however, we are able to avoid these issues by training a curriculum network (inspired by GoalGAN (Florensa et al. (2018))) to propose samples for training the value function. We introduce a generative curriculum model $c(s|g)$ for the distribution $p_c(s|g) \propto l(s,g)\exp(V(s,g))$, where $V(s,g)$ is the value function estimate for (s,g) and $l(s,g)$ is the squared error for $V(s,g)$. Since our value function is approximately the negative number of timesteps needed to reach the goal, drawing samples from $c(s|g)$ prioritizes minimizing the error for states near the goal. This helps to avoid propagating errors from $V(s_{t+1},g)$ to $V(s_t,g)$ when training the value function.

Additionally, we use replay buffers with weighted entries to speed up agents' exploration and training. Although we would like to explore goals in low-density regions of the state space ("rare" goals), they are unlikely to be sampled from a traditional replay buffer. If an agent seldom updates its policy to reach rare goals, it is then even more unlikely to encounter those rare goals, further hindering exploration. A weighted goal replay buffer can address this problem by upweighting the likelihood of sampling these rare goals when training the dynamics model, policy, or value function.

Lastly, we make the assumption that the agent will generally pass through high-density regions of the state space when going from one low-density region to another. If this assumption holds, using an unweighted state replay buffer could cause high-density regions of the state space to become increasingly likely throughout training. However, we can downweight the states encountered in earlier episodes to make these high-density states less likely. Notably, under the assumption, this downweighting does not affect the relative ordering of states by their likelihood. Empirically, we find this downweighting beneficial when training the state distribution model f. A consideration for future work is avoiding making this assumption by sampling goals that are both exploratory and near the agent's current state.

Our full algorithm is detailed in Algorithm 1.

B HYPERPARAMETERS

For all of our experiments, we used exploration factor $m = 20$, probability of sampling from curriculum $p_{\text{curric}} = 0.8$, size of action candidate set $j = 1$, number of dynamics rollouts to approximate value function $N = 20$, minibatch size $mb\_size = 1000$, number of updates per episode $n\_updates = 100$, and ran 50 environments in parallel to decrease wall-clock time. The only hyperparameters we varied were the state memory half-life h and the reward shaping factor k. We set $h = 20$ for the Ramp Pointmass and Cartpole environments, and $h = 50$ for the Ant; we set $k = 100$ for the Ramp Pointmass environment and $k = 10$ for Cartpole and Ant.

We used multilayer perceptrons (MLPs) with 4 hidden layers of size 200 and parameterized ReLU (PReLU) activations (He et al. (2015)) to parameterize our models, and use AdamOptimizer (Kingma & Ba (2014)) with learning rate 0.001 for training.

B.1 FLOW-BASED MODEL ARCHITECTURE

Following Kingma & Dhariwal (2018), we parameterize our flow-based transformations as $f_{\text{enc}}(x) = W_1 h_1(...W_{n-1}h_{n-1}^{-1}(W_n x)...)$, where the W_is are invertible matrices of the form $W = PLDU$ and $D = \text{diag}(d_{\text{sgn}}\exp(d))$ for $d_{\text{sgn}} \in \{-1, 1\}$. When initializing our model, we sample a random matrix W, decompose it into

Algorithm 1: Training an unsupervised, generalist agent in an arbitrary environment

Initialize generative, likelihood-based models for the state distribution $f(s)$, dynamics $d(s'|s,a)$, policy $\pi(a|s,g)$, and curriculum $c(s|g)$. Initialize a value function approximator $V(s,g)$.
Initialize weighted replay buffers for states s, goals g, and dynamics transition tuples $t=(s,a,s')$.
Get initial state $s=$ agent.get_obs().
Hyperparameters: exploration factor m, probability of sampling from curriculum p_{curric}, size of action candidate set j, number of dynamics rollouts to approximate value function N, state memory half-life h, reward shaping factor k, minibatch size mb_size, number of updates per episode $n\_updates$.

while *true* **do**
  // Update target value function.
  $V_{\text{targ}} \leftarrow V$
  // Collect data for the episode.
  Take samples $\{z_1,\ldots,z_m\}$ from $z\sim\mathcal{N}(0,I)$ and decode goals $g_i=f_{\text{dec}}(z_i)$. Choose $g=\arg\min_{\{g_1,\ldots,g_m\}} p_f(g_i)$.
  Estimate $v_{\text{cur}}=v_{\text{prev}}=V(s,g)$.
  while $v_{cur}\geq v_{prev}$ **do**
    $v_{\text{prev}}\leftarrow v_{\text{cur}}$
    Sample $a\sim\pi(s,g)$.
    Execute agent.act(a) and receive new state s'.
    Add s' to the state and goal replay buffers and (s,a,s') to the dynamics replay buffer. Set the weight of s' to 1 in the state replay buffer and to $\frac{1}{p_f(s')}$ in the goal replay buffer. Set the weight of (s,a,s') in the dynamics replay buffer to $\frac{1}{p_f(s_i^d)p_f(s_i'^d)}$.
    $s\leftarrow s'$
    Estimate $v_{\text{cur}}=V(s,g)$.
  end
  // Update networks and replay buffers.
  for *update* $\in$ *range*$(n\_updates)$ **do**
    Sample a minibatch of mb_size samples from each of the replay buffers.
    Update state distribution model, dynamics model, and policy to maximize the expected log-likelihood of $f(s)$, $d(s'|s,a)$, and $\pi(a|s,g)$, respectively.
    For each transition $t_i=(s_i^d,a_i^d,s_i'^d)$ in this minibatch, update the weight of t_i in the dynamics replay buffer to $\frac{1}{p_f(s_i^d)p_f(s_i'^d)}$.
    For each goal g_i in this minibatch, update the weight of g_i in the goal replay buffer to $\frac{1}{p_f(g_i)}$.
    For each goal g_i, sample $s_{c_i}=c(g_i;z), z\sim\mathcal{N}(0,I)$.
    For each state s_i, update $s_i\leftarrow s_{c_i}$ with probability $\frac{p_{\text{curric}}p_f(s_{c_i})}{p_{\text{curric}}p_f(s_{c_i})+(1-p_{\text{curric}})p_f(s_i)}$.
    For each (s_i,g_i) tuple, do $(l_i,v_i)=$ `Update_Value_Function_and_Policy` (s_i,g_i).
    For each (s_i,g_i,l_i,v_i) tuple, update $c(s_i|g_i)$ to maximize the probability of s_i, weighted by $l_i\exp(v_i)$.
    For each g_i, update the value function approximator to minimize $V(g_i,g_i)^2$.
  end
  Downweight all states in the state replay buffer by a factor of $w_{\text{state\_decay}}=0.5^{\frac{1}{h}}$.
end
Function `Update_Value_Function_and_Policy` (s,g)**:**
  Sample $\mathcal{A}_{\text{cand}}=\{a_i^{\pi}\sim\pi(s,g)\}\cup\{a_i^{\text{rand}}\sim U([-1,1]^{\text{action\_dim}})\}, i\in\{1,\ldots,j\}$.
  For each $a\in\mathcal{A}_{\text{cand}}$, add $a'=a+\nabla_a(r(s,a,s',g)+V_{\text{targ}}(s',g))$ for $s'\sim d(s,a)$, $r(s,a,s',g)=-1+\exp(-k||f_{\text{enc}}(s')-f_{\text{enc}}(g)||^2)$ to $\mathcal{A}_{\text{cand}}$.
  Sample $\epsilon_i\sim\mathcal{N}(0,I)$ for $i\in\{1,\ldots,N\}$.
  Select $a_{\text{best}}=\arg\max_{a\in\mathcal{A}_{\text{cand}}}\min_{\epsilon_i\in\{\epsilon_1,\ldots,\epsilon_N\}}[r(s,a,s_i',g)+V_{\text{targ}}(s_i',g)]$, where $s_i'=d(s,a;\epsilon_i)$.
  Compute $v_{\text{targ}}=\frac{1}{N}\sum_{i=1}^{N}[r(s,a_{\text{best}},s_i',g)+V_{\text{targ}}(s_i',g)]$, where $s_i'=d(s,a;\epsilon_i)$.
  Update V to minimize $l=(V(s,g)-v_{\text{targ}})^2$.
  Update π to maximize $\pi(a_{\text{best}}|s,g)$.
  return (l,v_{targ})

$PLDU$, fix P and d_{sgn}, and learn L, d, and U. We introduce $h(x; \alpha, \beta) = \begin{cases} \alpha x & x \leq -1 \\ a(x^2+1)+b & -1 < x < 1 \\ \beta x & x \geq 1 \end{cases}$, where $a = \frac{\beta-\alpha}{4}$ and $b = \frac{\beta+\alpha}{2}$, as smooth PReLU, which is invertible and everywhere continuous and differentiable for $\beta > 0$ and $\alpha > 0$. We alternate between h and h^{-1} nonlinearities in constructing f, as we find that this stabilizes training.

When we are learning a conditional model $f_{\text{enc}}(x; c)$, we modify this formulation only sightly, by learning $(\log d_i, \log a_i, \log b_i) = \text{MLP}(c)$ for $i \in \{1, \ldots, n\}$.

For $f(s)$ and $c(s|g)$ we use $n = 20$; for $d(s'|s, a)$ and $\pi(a|s, g)$ we use $n = 2$.