Learning Generative Models with Goal-conditioned Reinforcement Learning

Mariana Vargas Vieyra　　　　　　　　　　　　　　　　　mariana.vargas-vieyra@inria.fr
Inria, France

Pierre Ménard　　　　　　　　　　　　　　　　　　　　　　pierre.menard@ens-lyon.fr
ENS Lyon, France

Abstract

We present a novel, alternative framework for learning generative models with goal-conditioned reinforcement learning. We define two agents, a *goal conditioned agent* (GC-agent) and a *supervised agent* (S-agent). Given a user-input initial state, the GC-agent learns to reconstruct the training set. In this context, elements in the training set are the *goals*. During training, the S-agent learns to imitate the GC-agent while remaining agnostic of the goals. At inference we generate new samples with the S-agent. Following a similar route as in variational auto-encoders, we derive an upper bound on the negative log-likelihood that consists of a reconstruction term and a divergence between the GC-agent policy and the (goal-agnostic) S-agent policy. We empirically demonstrate that our method is able to generate diverse and high quality samples in the task of image synthesis.

Keywords: Generative Models, Goal-conditioned Reinforcement Learning.

1. Introduction

We consider the problem of learning a *generative model*. In recent years the study of generative models became a vast and prolific research field in the machine learning community. Because of their ability to capture important information about the distribution of the available data they have the capacity to generate new samples, thus emulating the nature of the training set. Generative models have shown outstanding results in applications such as speech generation (van den Oord et al., 2016) and high-quality image generation (Brock et al., 2019; Razavi et al., 2019). Recent examples of these models include Generative Adversarial Networks (GAN) (Goodfellow et al., 2014; Karras et al., 2018; Brock et al., 2019), Variational Auto-Encoders (VAE) (Rezende et al., 2014; Kingma and Welling, 2014; van den Oord et al., 2017), Flow-based Models (Dinh et al., 2017; Kingma and Dhariwal, 2018), Auto-regressive Models (Germain et al., 2015; Oord et al., 2016) and Diffusion Models (DM) (Sohl-Dickstein et al., 2015; Ho et al., 2020; Nichol and Dhariwal, 2021; Song et al., 2021a).

Given a sample from the training set, Diffusion Models work by adding a small amount of Gaussian noise to the sample in some pre-defined amount of steps, denoted by H. This *diffusion process* yields a sequence of H intermediate representations, each of which have a larger amount of noise than the previous one. Note that by following this process, the last representation we obtain is a sample from a Gaussian distribution. The generative model is then obtained by learning how to reverse this diffusion process. Once trained, the model is able to generate new samples by applying the reverse diffusion process on gaussian noise. Recently, Sohl-Dickstein et al. (2015); Ho et al. (2020); Nichol and Dhariwal (2021); Dhariwal and Nichol (2021) showed that DMs are competitive with GANs in the task of high-quality image synthesis. Interestingly, a dual view of DM (Ho et al., 2020) is the score-based generative model (Song and Ermon, 2019), where the generative model is the result of several steps of a Langevin dynamic (Parisi, 1981) following a score function. Such score function is learned beforehand in order to approximate the gradient of the log-likelihood. Although DMs are simple to train, they are slow to sample from (Ho et al., 2020) as the model in general requires a large number of steps H ($H \approx 1000$) to be able to reconstruct the inputs. Several techniques have been proposed to overcome this issue, but they speed-up the sampling process in detriment of the output quality (Nichol and Dhariwal, 2021; Song et al., 2021a). This calls for approaches that naturally require less steps to generate samples.

In this paper we adopt a reverse point of view: we learn how to transform an arbitrary fixed initial state into any sample of the training set using the Goal-Conditioned Reinforcement Learning (GCRL) framework (Kaelbling, 1993; Pong et al., 2018; Andrychowicz et al., 2017; Schaul et al., 2015). In GCRL the agent aims at a particular state called the *goal*. At each step, the environment yields a loss that accounts for "how far" the agent is from the goal it is targeting.

For example, the loss can be defined as the Euclidean distance between the current state and the goal. In the context of learning a generative model we consider the training set to be a *set of goals*. Intuitively, our learning procedure consists of training a family of *goal-conditioned agents* (GC-policy) that learn to reach the different elements in the training set by producing a *trajectory* of intermediate representations, departing from the fixed initial state. At the same time, we obtain the generative model by learning a *mixture* policy of these goal-conditioned policies where the *goal is sampled uniformly at random from the training set*. Concretely, the generative model approximates a policy that randomly picks a trajectory that departs from the fixed initial state and leads to an element of the training set. This can be cast as a supervised learning procedure. Note that the goal-conditioned agents are used for training only. At inference time, we generate trajectories with the mixture policy and collect the states reached at the final step. Such states are the new generated samples. Similar to variational inference (Blei et al., 2017), we derive a lower bound on the log-likelihood that consists of two terms. The first term measures how good the GC-policy is able to reconstruct the samples. The second term is a divergence term between the mixture policy and the GC-policy. We also provide empirical evidence that our method is able to effectively reconstruct the observed data and generate a rich variety of new samples in a small number of steps ($H \approx 16$).

To summarize we highlight our main contributions:

- We bridge the gap between the fields of RL and generative models by introducing a novel framework that learns a generative model using GCRL. Although GCRL has been studied in similar contexts (Rudner et al., 2021; Attias, 2003), it is the first time to the best of our knowledge that RL is used as a building block for learning generative models.
- We derive a lower bound on the log-likelihood that accounts for the reconstruction quality of the model.
- We provide an empirical evaluation that compares our method to the Variational Auto-Encoder model. Comparisons to other generative models are left as future work.

2. Inference as Goal-Conditioned Reinforcement Learning

We assume that we have access to a training set $\{\widetilde{x}^1, \ldots, \widetilde{x}^N\} =: \mathcal{D} \subseteq \mathcal{X}$ of samples from some unknown distribution. Our goal is to perform inference on this training set where the candidates probability distributions are the final state distribution of a policy in a certain episodic MDP.

Markov Decision Problem (MDP)   We consider a loss-free episodic MDP $\mathcal{M} = (\mathcal{X}, \mathcal{Y}, H, p)$ where $\mathcal{X}$ is the set of states (that also contains the training set $\mathcal{D}$), $\mathcal{Y}$ the action set, H the number of steps, $p_h(x'|x, y)$ is the transition probability from state x to state x' by taking the action y at step $h \in [H]$, where $[H] := \{1, \ldots, H\}$.

Policy, reach probability and value function   A policy π is a collection of functions $\pi_h : \mathcal{X} \to \Delta(\mathcal{Y})$ for all $h \in [H]$, where every π_h maps each state to a probability distribution over actions.

Under policy π a trajectory is generated as follows: an initial state $x_1 \sim p_0$ is sampled. Then for $h \in [H]$, given the current sate x_h, the agent samples an action $y_h \sim \pi_h(\cdot|x_h)$ and the next state $x_{h+1} \sim p_h(\cdot|x_h, y_h)$ is generated according to the transition probability. We denote by $p_h^\pi(x)$ the probability to reach the state x in the MDP $\mathcal{M}$ at step h under the policy π. Similarly, we denote by $p^\pi(\tau)$ the probability distribution of a trajectory $\tau = x_1, a_1, \ldots, x_H, a_H, x_{H+1}$ under the policy π. Given two policies π and π' the Kullback-Leibler divergence between p^π and $p^{\pi'}$ is, by the chain rule,

$$\mathrm{KL}(p^\pi, p^{\pi'}) = \mathbb{E}^\pi \left[\sum_{h=1}^H \mathrm{KL}\big(\pi(x_h), \pi'(x_h)\big) \right],$$

where $\mathbb{E}_\pi$ is the expectation under p^π.

Upper bound the negative log-likelihood   We assume as probabilistic model for the training set the reach probability of the last step $p_{H+1}^\pi(\cdot)$ *parameterized by the policy* π. We then want to find the policy that minimizes the negative log-likelihood of the training set $L(\pi)$ where

$$L(\pi) := \frac{1}{N} \sum_{\widetilde{x} \in \mathcal{D}} \log \frac{1}{p_{H+1}^\pi(\widetilde{x})} .$$

Solving this optimization problem is difficult because the probability $p_{H+1}^{\pi}(\widetilde{x})$ is typically intractable. Similarly to variational inference (Blei et al., 2017), we will instead minimize *an upper bound* on the negative log-likelihood. For a fixed element $\widetilde{x} \in \mathcal{D}$, conditioned on the state-action pair (x_H, y_H) it holds that

$$
\log \frac{1}{p_{H+1}^{\pi}(\widetilde{x})} = -\log \left(\mathbb{E}_{\pi} \left[p_H(\widetilde{x}|x_H, y_H) \right] \right)
$$
$$
\leq \min_{\pi'} \mathbb{E}_{\pi'} \left[\log \frac{1}{p_H(\widetilde{x}|x_H, y_H)} \right] + \mathrm{KL}(p^{\pi'}, p^{\pi}). \tag{1}
$$

Equation (1) follows from the variational formula for the moment generating function (see Lemma 1 in Appendix B) and it defines an upper-bound on the negative log-likelihood of $\widetilde{x}$.

We can then define a surrogate loss parameterized by a policy π and a family of goal-conditioned policies $(\pi^{\widetilde{x}})_{\widetilde{x} \in \mathcal{D}}$, indexed by an element of the training set as:

$$
L_{\mathrm{ub}}(\pi, (\pi^{\widetilde{x}})_{\widetilde{x} \in \mathcal{D}}) := \frac{1}{N} \sum_{\widetilde{x} \in \mathcal{D}} \mathbb{E}_{\pi^{\widetilde{x}}} \left[\log \frac{1}{p_H(\widetilde{x}|x_H, y_H)} \right] + \mathrm{KL}(p^{\pi^{\widetilde{x}}}, p^{\pi}). \tag{2}
$$

Using the fact that L_{ub} is an upper-bound on the loss L our problem becomes:

$$
\min_{\pi} L(\pi) \leq \min_{\pi} \min_{(\pi^{\widetilde{x}})_{\widetilde{x} \in \mathcal{D}}} L_{\mathrm{ub}}(\pi, (\pi^x)_{x \in \mathcal{X}})
$$
$$
= \min_{\pi} \frac{1}{N} \sum_{\widetilde{x} \in \mathcal{D}} \min_{\pi^{\widetilde{x}}} \mathbb{E}_{\pi^{\widetilde{x}}} \left[\log \frac{1}{p_H(\widetilde{x}|x_H, y_H)} \right] + \mathrm{KL}(p^{\pi^{\widetilde{x}}}, p^{\pi}).
$$

At a high level the minimization of L_{ub} goes as follows. For each point $\widetilde{x} \in \mathcal{D}$ in the training set with goal-conditioned RL we learn a policy $\pi^{\widetilde{x}}$ not too far from π that leads to $\widetilde{x}$ with high probability. Simultaneously we train the policy π to reproduce trajectories from the policies $\pi^{\widetilde{x}}$.

Inner minimization: goal-conditioned RL If we fix the policy π and a state $\widetilde{x} \in \mathcal{D}$ in the training set then solving (1) is equivalent to solving a regularized *goal-conditioned RL problem*. That is, we want to find a policy $\pi^{\widetilde{x}}$ close to π that lead to the goal $\widetilde{x}$ with high probability. Let

$$
\ell_h^{\widetilde{x}}(x, y) := \begin{cases} \log \dfrac{1}{p_H(\widetilde{x}|x, y)} & \text{if } h = H \\ 0 & \text{otherwise} \end{cases}, \tag{3}
$$

be a loss function parameterized by the goal $\widetilde{x}$, and let $\mathcal{M}^{\widetilde{x}} = (\mathcal{X}, \mathcal{Y}, H, p, \ell^{\widetilde{x}})$ be an MDP. Then the minimization of (1) is equivalent to solving the MDP $\mathcal{M}^{\widetilde{x}}$ regularized by the policy π. Precisely, we have

$$
\min_{\pi^{\widetilde{x}}} \mathbb{E}_{\pi^{\widetilde{x}}} \left[\log \frac{1}{p_H(\widetilde{x}|x_H, y_H)} \right] + \mathrm{KL}(\pi^{\widetilde{x}}, p^{\pi}) = \min_{\pi^{\widetilde{x}}} V^{\pi^{\widetilde{x}}, \widetilde{x}} + \mathrm{KL}(\pi^{\widetilde{x}}, p^{\pi}),
$$

where $V^{\pi^{\widetilde{x}}, \widetilde{x}}$ is the value of the goal-conditioned policy $\pi^{\widetilde{x}}$ in the MDP $\mathcal{M}^{\widetilde{x}}$. Although this goal-conditioned RL problem has already been studied by Rudner et al. (2021) and Attias (2003), it is the first time to the best of our knowledge that this formulation is used as an intermediate task to build a generative model. The link with variational inference is clear: we seek a policy π' that approximates well the posterior distribution of a trajectory generated by π conditioned on the fact that this trajectory reaches the goal $\widetilde{x}$.

Outer minimization: supervised learning Now, if we fix the family of goal-conditioned policies, minimizing the surrogate loss $L_{\mathrm{ub}}(\cdot, (\pi^{\widetilde{x}})_{\widetilde{x} \in \mathcal{D}})$ over the policy π amounts to minimizing a convex combination of Kullback-Leibler divergences

$$
\arg \min_{\pi} L_{\mathrm{ub}}(\pi, (\pi^{\widetilde{x}})_{\widetilde{x} \in \mathcal{D}}) = \arg \min_{\pi} \frac{1}{N} \sum_{\widetilde{x} \in \mathcal{D}} \mathrm{KL}(p^{\pi^{\widetilde{x}}}, p^{\pi}) \tag{4}
$$

This optimization problem can be efficiently solved with supervised learning. First, we sample goals according the the empirical distribution of the training set. Then, for each goal, we generate a trajectory with the goal-conditioned policy, and collect state-action pairs. We then use these state-action pairs to supervise the policy π.

Variational agent In the sequel we call the pair $(\pi, (\pi^{\widetilde{x}})_{\widetilde{x} \in \mathcal{D}})$ a variational agent (V-agent) made of π a supervised agent (S-agent) and $(\pi^{\widetilde{x}})_{\widetilde{x} \in \mathcal{D}}$ a goal-conditioned agent (GC-agent).

3. Algorithm

In this section we describe a particular instanciaton of MDP and V-agent and introduce an algorithm to learn the V-agent. In what follows we assume the space of states and actions are both the real d-space $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}^d$, for some dimension d.

3.1 Instanciation

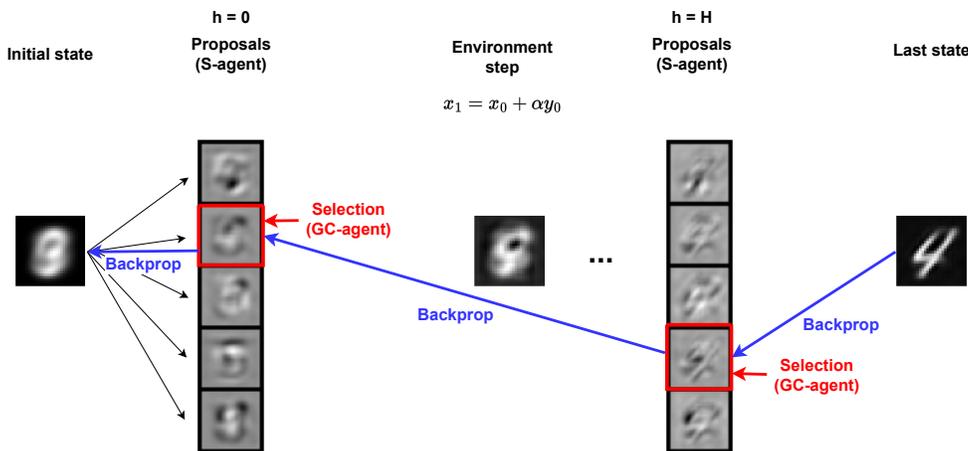We first describe a particular MDP and family of variational agents.

Figure 1: Generation of a trajectory with the GC-agent for a sample of MNIST: At a step h, given a state the GC-agent select one of the proposals provided by the S-agent. The action, i.e. the selected proposal, is scaled by some rate α_h and added to the previous state. For a fixed sequence of actions the proposal network is differentiable. Therefore we can backpropagate the loss from the last state to the initial state once the sequence of actions is determined.

A particular MDP We present a specific MPD $\mathcal{M} = \{\mathcal{X}, \mathcal{Y}, H, p\}$ parameterized by a sequence of rates $(\alpha_h)_{h \in [H]}$ and a variance σ^2. The initial state is fixed, e.g. x_1 is the mean of the elements in $\mathcal{D}$. The first $H - 1$ transition are deterministic. Precisely for $h \in [H - 1]$, given the state x_h and action y_h the next state x_{h+1} is

$$x_{h+1} = x_h + \alpha_h y_h. \tag{5}$$

The last transition probability distribution is a Gaussian distribution of variance σ^2, that is $p_H(\cdot | x_H, y_H) = \mathcal{N}(x_H + \alpha_H y_H, \sigma^2)$. Note that in this case the goal-conditioned loss is proportional to the mean square error between the last state and the goal $\ell_h^{\widetilde{x}}(x, y) = \mathbb{1}\{h = H\}\|x + \alpha_h y - \widetilde{x}\|_2^2/(2\sigma^2)$.

A family of variational agents As mentioned earlier a V-agent is made of a GC-agent and a S-agent. We only consider agents with policy given by a *mixture of A Dirac probability distributions* for some fixed $A \in \mathbb{N}$. That is, given a step and a state (and a goal for the GC-agent) the agent samples an index $a_h \in [A]$ that we call *selection*. The agent also samples a vector of actions $(z_{h,a})_{a \in [A]} \in \mathcal{Y}^A$, that we call *proposals*, from the collection of Dirac distributions. Then, the action is given by the proposal that corresponds to the selected index $y_h = z_{h,a_h}$. Note that, since the Kullback-Leibler divergence between two Dirac distribution with different supports diverges, in order to maintain the loss L_{ub} defined in (2) finite *the GC-agent and the S-agent should share the same proposals*. In particular, the proposal function of the GC-agent that maps the step, state and goal to the collection of proposal *is independent*

of the goal since it is the case for the proposal function of the GC-agent. We use GCRL to learn to *select* the best proposal among the A proposals. In other words, we represent the action to be taken by its index $a_h \in [A]$. This allows us to operate on a discrete space of actions. Intuitively, the agent learns to map samples from the underlying distribution of the data to sequences of indexes representing the selected actions at each step. The proposal function is fit to provide proposals that approach the agent to the goals, see below. Therefore, the MDP is non-stationary as the proposals change during the learning. By sharing the proposal function between both agents and representing actions by their indexes, we traded a GCRL task with continuous action space for a (non-stationary) GCRL task with discrete action space.

Remark 1 *We use this particular class of policies for the CG and S-agent: a mixture of Dirac distributions that allows us to represent a highly multi-modal distribution in few steps (provided that the rates α_h are large enough). Note that if one were to use the class of Gaussian probability distributions instead to model the policy, one would need much more steps to model a multi-modal distribution. Crucially, it would be harder to "branch out" to several different specific states from the current state by only adding Gaussian noise, as the model would be prone to mode collapse.*

Parametrization We parameterize the S-agent by two neural networks: The proposal network `S-prop-net`$_\theta$ with weights θ that takes as input a state, step and selection (x, h, a) and outputs an action $y = $ `S-prop-net`$_\theta(x, h, a)$. The selection network `S-select-net`$_\phi$ with weights ϕ that takes as input a state, step (x, h) and outputs a distributions over selections. It is important to note that the S-agent selection network and the proposal network are not conditioned on the goal. As stated above the GC-agent only needs to pick indexes $a_h \in [A]$. It is parameterized by a GC-selection Q-values network `GC-select-Qnet`$_\psi$ with weights ψ that takes as input a state, step and goal $(x, h, \widetilde{x})$ and outputs Q-values $\left(\text{GC-select-Qnet}_{\psi,a}(x, h, \widetilde{x})\right)_{a \in [A]}$. Typically for the `S-prop-net`$_\theta$ we choose a `U-Net` architecture (Ronneberger et al., 2015) whereas the `S-select-net`$_\phi$ and `GC-select-Qnet`$_\psi$ are simple convolutional neural networks. See Appendix B for more details.

Unfolding the proposal network Note that once a sequence of selections $(a_1, \ldots, a_H)$ is fixed, the last state (or more precisely, the state's mean) of a trajectory generated with actions $y_h = $ `S-prop-net`$_\theta(x_h, h, a_h)$ from the proposal network and the aforementioned selections is a differential function of the weights θ. To see this one just needs to unfold (5) trough the steps $h \in [H]$, see Figure 1. We denote by `S-prop-net`$_{\theta,U}(a_1, \ldots, a_H) = x_H + \alpha_H y_H$ the unfolded network that maps a sequence of selections to the mean of the state at step $H + 1$ when the actions are obtained with the S-agent proposal network.

3.2 Training

The learning procedure of the V-agent is split into two parts: trajectories are sampled with the GC-agent (and the proposal from the S-agent) to reconstruct a goal sampled at random from the training set. Then these trajectories are used to learn the different networks that parameterized the GC-agent and the S-agent. In particular the GC-agent is learned with the `DQN` algorithm[1] (Mnih et al., 2013).

Memory We use three replay buffers, one for each network. The replay buffer $\mathcal{R}_{\text{GC-select}}$ associated with the GC Q-values network `GC-select-Qnet`, that will be fed with transitions. The replay buffer $\mathcal{R}_{\text{S-select}}$ associated with the S-agent selection network `S-select-net`. And the replay buffer $\mathcal{R}_{\text{S-prop}}$ of the S-agent proposal network `S-prop-net`.

Sample trajectory We first sample a goal $\widetilde{x}$ uniformly at random from the training set $\mathcal{D}$. Then a trajectory is generated with the GC-agent as follows. At step h, given the current state x_h, we pick a selection index $a_h \in$ $\arg\min \text{GC-select-Qnet}_{\psi,a}(x_h, h, \widetilde{x})$. The action $y_h = $ `S-prop-net`$_\theta(x_h, h, a_h)$ is the output of S-agent proposal network that corresponds to the index a_h. The next state x_{h+1} is given by (5) and the GC-agent receives a loss $\ell_h = \ell_h^{\widetilde{x}}(x_h, y_h)$. The observed transition is stored in the replay-buffer $\mathcal{R}_{\text{S-prop}}$ of the GC-agent, as well as the selection (x_h, h, a_h) in the selection replay-buffer $\mathcal{R}_{\text{S-select}}$ of the S-agent. Once the horizon is reached we record the

---

1. Precisely, in the experiments we use the `Double-DQN` algorithm (Hasselt et al., 2016) with Dueling Q-Network (Wang et al., 2016).

selections and the goal $(\{a_1, \ldots, a_h\}, \widetilde{x})$ in the proposal replay-buffer $\mathcal{R}_{\texttt{S-prop}}$ of the S-agent. See Algorithm 1 for a detailed description.

Algorithm 1 `Sample-trajectory`

 1: Sample uniformly at random goal $\widetilde{x}$ from the train set $\mathcal{D}$.
 2: Get initial state x_1.
 3: **for** $h \in [H]$ **do**   ▷ Generate trajectory with the GC-agent.
 4:     Get selection $a_h \in \arg\min_{a \in [A]} \texttt{GC-select-Qnet}_{\psi,a}(x_h, h, \widetilde{x})$ ▷ We can add an exploration malus.
 5:     Get action $y_h = \texttt{S-prop-net}_\theta(x_h, h, a_h)$.
 6:     Observe next state $x_{h+1} = x_h + \alpha_h y_h$.
 7:     Get loss $\ell_h = \ell_h^{\widetilde{x}}(x_h, y_h)$.
 8:     Record transition $(x_h, h, a_h, x_{h+1}, \ell_h, \widetilde{x})$ in $\mathcal{R}_{\texttt{GC-select}}$ and selection (x_h, h, a_h) in $\mathcal{R}_{\texttt{S-select}}$.
 9: **end for**
10: Record trajectory selections $(\{a_1, \ldots, a_H\}, \widetilde{x})$ in $\mathcal{R}_{\texttt{S-prop}}$.

Update   The GC-agent is learned with the `DQN` algorithm (Mnih et al., 2013). Thus we need to define a target Q-values network parameterized by the weights $\psi^{\texttt{target}}$. At a high level we use the target Q-values network `GC-select-Qnet`$_{\psi^{\texttt{target}}}$ and the transitions stored in the replay buffer $\mathcal{R}_{\texttt{GC-select}}$ to compute new targets via the optimal Bellman equations. Then the GC-agent Q-values network `GC-select-Qnet`$_\psi$ is trained to fit these targets by gradient descent on the mean squared error. Note that the targets weights are updated as an exponential moving average of the weights $\psi^{\texttt{target}} \leftarrow \rho\psi + (1-\rho)\psi^{\texttt{target}}$ for some parameter $\rho \in (0,1)$. The update of the S-agent is simpler. The S-agent selection network is trained to reproduce the selections picked by the GC-agent by minimizing the cross-entropy loss between the selections from the replay-buffer $\mathcal{R}_{\texttt{S-select}}$ and its prediction. The S-agent unfolded proposal network `S-prop-net`$_{\theta,U}$ is trained to map the sequence of selections stored in the replay-buffer $\mathcal{R}_{\texttt{S-prop}}$ to the associated goals. Thus the weights θ are updated by gradient descent on the mean square error between the output of `S-prop-net`$_{\theta,U}$ evaluated on the selections and the goal, see Figure 1. A complete description is provided in Algorithm 2.

Exploration   For the exploration we do not rely on the by default ε-greedy mechanism. As a matter of fact, it is not clear how this exploration technique would interact with the learning of the proposal network which is based on the trajectories generated by the GC-agent. Instead we propose to add a penalty to the Q-values. This penalty is defined as the logarithm of the probability of the S-agent policy. That is, for some parameter $\kappa > 0$,

$$a_h \in \arg\min_{a \in [A]} \texttt{GC-select-Qnet}_{\psi,a}(x_h, h, \widetilde{x}) + \kappa \log\big(\texttt{S-select-net}_{\phi,a}(x_h, h)\big).$$

This will encourage the agent to take the selections that have not been used. Note that this penalty does not interfere with the policy of the GC agent systematically picking the same indexes for a specific goal as long as the other indexes are used for other goals.

3.3   Inference

To obtain as sample from the learned model one just needs to generate a trajectory using the S-agent. Note that at inference time we do not use the GC-agent, and that the selections are entirely made by the S-agent selection network `S-select-net`$_\phi$. It is important to point out that once the S-agent proposal network is fixed the number of possible samples generated by the model is finite and upper-bounded by A^H. For a reasonably small choice of A and H this upper-bound is very large and not restrictive. Observe that this model thus produces a discrete representation of the input. Discrete latent representations have been explored by Van Den Oord et al. (2017). The only source of randomness in the model comes from sampling a selection from the S-agent selection network `S-select-net`$_\phi$. A detailed sampling procedure is provided in Algorithm 3

Algorithm 2 `Update`

1: **if** time to update `S-prop-net`$_\theta$ **then**
2:      Sample a batch $\mathcal{B}_{\texttt{S-prop}}$ in $\mathcal{R}_{\texttt{S-prop}}$
3:      Update `S-prop-net`$_\theta$ with one step of gradient,

$$\nabla_\theta \frac{1}{|\mathcal{B}_{\texttt{S-prop}}|} \sum_{(\{a_1,\ldots,a_H\},\widetilde{x}) \in \mathcal{B}_{\texttt{S-prop}}} \|\texttt{S-prop-net}_{\theta,U}(a_1,\ldots a_H) - \widetilde{x}\|_2^2 \,.$$

4: **end if**
5: **if** time to update `S-select-net`$_\phi$ **then**
6:      Sample a batch $\mathcal{B}_{\texttt{S-select}}$ in $\mathcal{R}_{\texttt{S-select}}$
7:      Update `S-select-net`$_\phi$ with one step of gradient,

$$\nabla_\phi \frac{1}{|\mathcal{B}_{\texttt{S-select}}|} \sum_{(x,h,a) \in \mathcal{B}_{\texttt{S-select}}} -\log\left(\texttt{S-select-net}_{\phi,a}(x,h)\right) \,.$$

8: **end if**
9: **if** time to update `GC-select-Qnet`$_\psi$ **then**
10:      Sample a batch $\mathcal{B}_{\texttt{GC-select}}$ in $\mathcal{R}_{\texttt{GC-select}}$
11:      **for** $(x,h,a,x',\ell,\widetilde{x}) \in \mathcal{B}_{\texttt{GC-select}}$ **do**
12:          Compute target
$$q^{\texttt{target}}(h,x',\ell,\widetilde{x}) = \ell + \min_b \texttt{GC-select-Qnet}_{\psi^{\texttt{target}},b}(x',h+1,\widetilde{x})$$

13:      **end for**
14:      Update `GC-select-Qnet`$_\psi$ with one step of gradient,

$$\nabla_\psi \frac{1}{|\mathcal{B}_{\texttt{GC-select}}|} \sum_{(x,h,a,x',\ell,\widetilde{x}) \in \mathcal{B}_{\texttt{GC-select}}} \big\|\texttt{GC-select-Qnet}_{\psi,a}(x,h,\widetilde{x})$$
$$- q^{\texttt{target}}(h,x',\ell,\widetilde{x})\big\|_2^2 \,.$$

15:      **if** time to update `GC-select-Qnet`$_{\psi^{\texttt{target}}}$ **then**
16:          Update target Q-values weights $\psi^{\texttt{target}} \leftarrow \rho\psi + (1-\rho)\psi^{\texttt{target}}$.
17:      **end if**
18: **end if**

Algorithm 3 `Sample`

1: Get initial state x_1.
2: **for** $h \in [H]$ **do**   ▷ Generate trajectory with the S-agent.
3:      Get selection $a_h \sim \texttt{S-select-net}_\psi(x_h,h)$.
4:      Get action $y_h = \texttt{S-prop-net}_\theta(x_h,h,a_h)$.
5:      Observe next state $x_{h+1} = x_h + \alpha_h y_h$.
6: **end for**
7: Return the last state x_{H+1}.

---

4. Experiments

We empirically assess the quality of our method on the task of image reconstruction and generation. Note that our goal here is to validate experimentally our new approach rather than providing a new state of the art method. To that end we conducted experiments on two publicly available datasets, namely Fashion-MNIST (Xiao et al., 2017) and MNIST (Deng, 2012). In what follows we describe the experimental setup and present the results.

4.1 Setup

We compare our method with two types of generative models, Variational Auto-Encoders and Diffusion Models. For the VAE baselines we use a convolutional VAE architecture with the state-of-the-art hyperparameters as Subramanian

(2020). For the DM baselines we use the publicly available implementation by Song et al. (2021b)[2] to run experiments. In particular, we fix the number of time-steps to 1000 and the forward diffusion process hyperparameters as in Ho et al. (2020). For the sake of comparison we parameterize the reverse process with the same architecture as in our method, that is, a four-layers U-Net (Ronneberger et al., 2015) where each layer consists of convolution layers followed by group normalization and a ReLU activation function. To model time-steps we do as follows: At each time-step $t \in [0, \dots, 1000]$ we use an embedding layer to compute an embedding the size of the image. We then reshape and concatenate this embedding as a set of extra channels to the input image.

For training our method we generated goal-conditioned trajectories as described in Section 3. We fixed the horizon $H = 16$ and the number of proposed actions per step $A = 16$. The sequence or rates $(\alpha_h)_{h \in [H]}$ is defined such that $\alpha_h = 1/h$ for $h = 1, \dots, H$. Similar to DMs, we parameterize the agents' proposal network with a four-layers U-Net, where the dependence on the current step h is also modeled with an embedding layer. Both the GC-agent and S-agent selection networks are parameterized as Convolutional Neural Networks with three blocks each. Further details on the architecture of our model and the baselines can be found in Appendix B. The rest of hyperparameters are chosen as follows: The exploration parameter is defined as $\kappa = 0.05$, and $\rho = 0.06$. We fixed the initial state to zero. The models were optimized using Adam optimizer (Kingma and Ba, 2015) with a learning rate fixed to 0.0001. All models were trained until convergence.

4.2 Results

To compare our method with the baselines we collect the following quantities: Mean Squared Error (MSE) loss between the test set and the test set reconstruction (Rec.), Fréchet Inception Distance (FID) (Heusel et al., 2017) between the test set and its reconstruction (Rec.), and FID of randomly generated samples with respect to the training set (Samples). For the VAE, a random sample can be generated by feeding a sample from an isotropic Gaussian distribution (the prior of the VAE latent space) to the decoder. To sample from the trained DMs we used the sampling method described by Song et al. (2021b). For our method, random samples are generated as described in Section 3.3.

Dataset	Method	MSE	FID	
		Rec.	Rec.	Samples
Fashion-MNIST	VAE	.13	45.09	60.06
	DM	–	–	**9.63**
	Us	**.10**	**39.10**	47.74
MNIST	VAE	.12	16.49	27.96
	DM	–	–	**1.94**
	Us	**.06**	**7.77**	19.46

Table 1: Evaluation of VAE, DM, and our model by MSE between the test set and test sample reconstruction, FID (lower is better) between test set and test sample reconstruction, and FID of randomly generated samples with respect to the training set.

(a) MNIST

(b) Fashion-MNIST

Figure 2: Four trajectories generated by the S-agent for MNIST (left) and Fashion-MNIST (right). Note that all trajectories depart from the same initial state. The source or randomness comes from the selection function of the agents.

---

2. https://github.com/yang-song/score_sde

(a) Batch of goals from MNIST test set.   (b) Reconstruction of batch.   (c) Random samples.

Figure 3: Reconstruction and generation of images. We sampled a batch of 100 images from the MNIST test set (left). We then generated trajectories where the selected actions were given by the GC-agent, in order to reconstruct the items in the batch (center). Finally, we randomly generated 100 images with the S-agent (right). A qualitative analysis of the generated samples (right) shows that our generative model is able to produce a broad variety of images.

We report our results in Table 1. We first note that our method outperforms the VAE baseline in terms of MSE and FID by a large margin in both, test sample reconstruction and randomly generated images. Our model lags behind DMs in terms of FID in both settings, reconstruction and randomly generated samples. This difference might be explained by the fact that we chose a very small horizon ($H = 16$). Exploring other values for H is left as future work.

Our results confirm that the proposed model is not only able to accurately reconstruct the input but it is also able to randomly generate a rich variety of images.In Figure 2 we show two randomly generated trajectories, that is, trajectories produced by the S-agent, for the trained models in MNIST and Fashion-MNIST datasets. A qualitative inspection of the reconstructed images and generated samples in Figure 3 shows that our model effectively captures the underlying distribution of the data.

5. Conclusion and Discussion

We introduced a novel framework to learn generative models based on goal-conditioned reinforcement learning. The main idea of this work is to consider the elements of the training set as being generated by an agent that reaches those states after a fixed number of steps. We then learn a mixture policy that approximates a family of goal-conditioned agents that are trained to generate trajectories that lead to the training points. Following this line of reasoning we derived an upper-bound for the negative log-likelihood consisting of two terms. A reconstruction error term that measures the reconstruction quality of the goal-conditioned policy, and a divergence term that encourages the mixture policy to remain close to the family of goal-conditioned agents. Our experiments demonstrate that our method is able to effectively reconstruct the training set and generate a rich variety of outputs in the task of image synthesis.

There are some aspects of the algorithm we presented that are yet to be explored. Regarding the use of GCRL to parameterize a generative model an interesting research direction could be to explore whether Hindsight Experience Replay (Andrychowicz et al., 2017) or a variant of it could contribute to obtaining a more effective GC-agent in terms of reconstruction loss. We empirically showed that our method needs significantly less steps to reach the goals than DMs without compromising the sample quality (in terms of FID score). A more thorough comparative study of the required number of steps by DMs and our method would be of interest. Also, note that we have not use the state-of-the-art architectures for DMs nor for our algorithm due to time and resource constraints. Studying and comparing DMs and our method with more sophisticated models, and on bigger datasets, is left as future work.

We believe that the framework we introduced could be useful for a broad scope of tasks. In particular, there are applications where the space of actions is inherently discrete. For instance, in molecule design one could consider adding atoms and bonds to be the space of actions. In this case one can dispense with a proposal function, and construct a model that only needs to learn to select the right actions. Addressing this task is an interesting research direction.

Acknowledgments

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 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

Hagai Attias. Planning by probabilistic inference. In Christopher M. Bishop and Brendan J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, volume R4 of *Proceedings of Machine Learning Research*, pages 9–16. PMLR, 03–06 Jan 2003. URL https://proceedings.mlr.press/r4/attias03a.html. Reissued by PMLR on 01 April 2021.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859877, Apr 2017. ISSN 1537-274X. doi: 10.1080/01621459.2017.1285773. URL http://dx.doi.org/10.1080/01621459.2017.1285773.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=B1xsqj09Fm.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=HkpbnH9lx.

M. D. Donsker and S. R. S. Varadhan. Asymptotic evaluation of certain markov process expectations for large time. iv. *Communications on Pure and Applied Mathematics*, 36(2):183–212, 1983. doi: https://doi.org/10.1002/cpa.3160360204. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160360204.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/germain15.html.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 20942100. AAAI Press, 2016.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1094–8, Chambery, France, 1993. Morgan Kaufmann. URL http://people.csail.mit.edu/lpk/papers/ijcai93.ps.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Hk99zCeAb.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/d139db6a236200b21cc7f752979132d0-Paper.pdf.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013. URL http://arxiv.org/abs/1312.5602. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/nichol21a.html.

Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1747–1756, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/oord16.html.

G. Parisi. Correlation Functions and Computer Simulations. *Nucl. Phys. B*, 180:378, 1981. doi: 10.1016/0550-3213(81)90056-0.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep RL for model-based control. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Skw0n-W0Z.

Emmanuel Quemener and Marianne Corvellec. Sidusthe solution for extreme deduplication of an operating system. *Linux J.*, 2013(235), nov 2013. ISSN 1075-3583.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/5f8e2fa1718d1bbcadf1cd9c7a54fb8c-Paper.pdf.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/rezende14.html.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

Tim GJ Rudner, Vitchyr H Pong, Rowan McAllister, Yarin Gal, and Sergey Levine. Outcome-driven reinforcement learning via variational inference. *arXiv preprint arXiv:2104.10190*, 2021.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/sohl-dickstein15.html.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021a. URL https://openreview.net/forum?id=St1giarCHLP.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/3001ef257407d5a371a96dcd947c7d93-Paper.pdf.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=PxTIG12RRHS.

A.K Subramanian. Pytorch-vae. https://github.com/AntixK/PyTorch-VAE, 2020.

Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Aron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Arxiv*, 2016. URL https://arxiv.org/abs/1609.03499.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/wangf16.html.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Appendix A. Derivations of Section 2

In this appendix we detail the missing derivations of Section 2.

Lemma 2 *For any policy π, state x,*

$$-\log\left(\mathbb{E}_\pi\left[p_H(x|x_H, y_H)\right]\right) \le \min_{\pi' \text{ policy}} \mathbb{E}_{\pi'}\left[\log\frac{1}{p_H(x|x_H, y_H)}\right] + \text{KL}(p^{\pi'}, p^\pi).$$

Proof Thanks to the Donsker-Varadhan's formula (Donsker and Varadhan, 1983) it holds

$$-\log\left(\mathbb{E}_\pi\left[p_H(x|x_H, y_H)\right]\right) = \min_{q\in\Delta(\text{Traj})} \mathbb{E}_q\left[\log\frac{1}{p_H(x|x_H, y_H)}\right] + \text{KL}(q, p^\pi),$$

where $\Delta(\text{Traj})$ is the set of probability distributions supported on the trajectory $\tau = (x_1, y_1, \ldots, x_H, y_H)$ up to step H with $x_h \in \mathcal{X}$ and $y_h \in \mathcal{Y}$. Restricting the infimum to probability distributions $q = p^{\pi'}$ induced by some policy π' allows us to conclude. ∎

Appendix B. Network architectures

In this appendix we describe the architecture of the proposal, Q-value and selection networks.

B.1 Proposal network

The proposal network is parameterized as a four-layer `U-Net`[3]. Each layer has two blocks consisting of a convolution layer followed by group normalization and a ReLU activation function.

The different networks used in our model also depends on discrete time-step $h \in [H]$. We incorporate this information in the model as follows. At each time-step h the proposal network receives an input of the form (x_h, h, a_h). The current state x_h is of shape $w \times t \times c$, where w is the width, t is the height, and c is the number of channels (for example, $c = 3$ for RGB images). The selected index a_h indicates which of the A available proposals is to be chosen at the current time-step h. We first calculate an embedding $\mathbf{h}_{a,h}$ representing h and a through an Embedding layer to obtain a tensor the same size as x_h. We then concatenate x_h and $\mathbf{h}_{a,h}$ on the channel axis to obtain a tensor of shape $w \times t \times 2c$ that is fed to the `U-Net`. Finally, the `U-Net` outputs the tensor x_{h+1} of shape $w \times t \times c$. This process is described as follows

$$
\begin{aligned}
h'_{a,h} &= Ah + a \\
\mathbf{h}_{a,h} &= \texttt{Emb}(h'_{a,h}) \\
x_{h+1} &= \texttt{U-Net}([x_h, \mathbf{h}_{a,h}]),
\end{aligned}
\tag{6}
$$

where $[\cdot, \cdot]$ represents concatenation of 3D tensors on the channel axis. At each pass the network generates the proposal that corresponds to the specified index. The output of each pass will then have the same shape as the inputs. We provide a detailed architecture of the network in Figure 4.

Note that, with this embedding of the time-step and selection the proposal network can memorize at most $H \times A$ distinct images. Thus combining proposals by choosing the selections in a trajectory is essential to be able to model a rich distribution.

B.2 Selection network

The selection network architecture is the same for both agents. It consists of three blocks of convolution layers followed by a group normalization and a ReLU activation function. The network outputs a tensor of size `batch_size` $\times$ A corresponding to the selected actions for each element in the batch. The embedding of the time-step follows the same method as in the proposal network. Figure 5 presents a sketch of the architecture we used for our experiments. The batch size was set to 128, proposal size $A = 16$ and number of steps $H = 16$.

B.3 Q-value selection network

The architecture of the Q-value network is the same as the one of the selection network except for the last layer where the output is not normalized to obtain a probability distribution. For the last layer we use a Dueling Network architecture (Wang et al., 2016).

---

3. The `U-Net` implementation is publicly available at https://github.com/milesial/Pytorch-UNet.
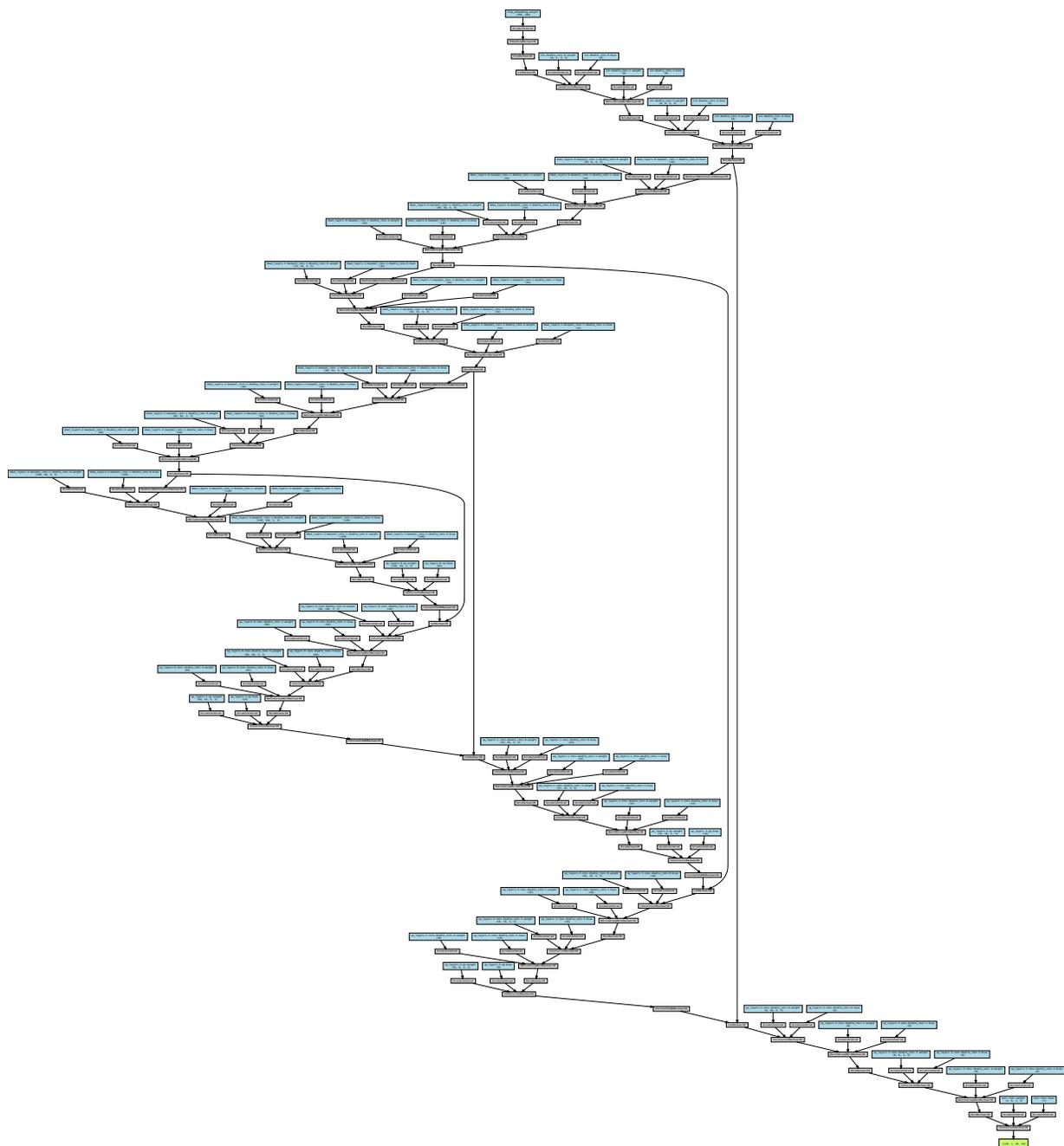
Figure 4: Proposal network architecture (shared between both agents) with batch size 128 and number of steps $H = 16$.

Figure 5: Selection network architecture for the GC-agent and S-agent with batch size 128 and selection size $A = 16$.