DIVERSE IMITATION LEARNING VIA SELF-ORGANIZING GENERATIVE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Imitation learning is the task of replicating expert policy from demonstrations, without access to a reward function. This task becomes particularly challenging when the expert exhibits a mixture of behaviors. Prior work has introduced latent variables to model the variations of the expert policy. However, our experiments show that the existing works do not exhibit appropriate imitation of individual modes. To tackle this problem, we adopt an encoder-free generative model for behavior cloning (BC) to accurately distinguish and imitate different modes. Then, we integrate it with GAIL to make the learning robust towards compounding errors at unseen states. We show that our method significantly outperforms the state of the art across multiple experiments.

1 INTRODUCTION

The goal of imitation learning is to learn to perform a task from expert trajectories without a reward signal. Towards this end, two approaches are studied in the literature. The first approach is behavior cloning (BC) (Pomerleau, 1991), which learns the mapping between individual expert states-action pairs in a supervised manner. It is well known that BC neglects long-range dynamics within a trajectory. This leads to problem of compounding error (Ross & Bagnell, 2010; Ross et al., 2011), that is, when a trained model visits states that are slightly different from those visited in expert demonstrations, the model takes erroneous actions, and the errors quickly accumulate as the policy unrolls. Therefore, BC can only succeed when training data is abundant.

An alternative approach, known as inverse reinforcement learning (IRL) (Ziebart et al., 2008), recovers a reward function that makes the expert policy optimal. Such approaches often require iterative calculation of the optimal policy with reinforcement learning (RL) in an inner loop, making them computationally expensive. The computational cost can be reduced by jointly optimizing a policy—via RL, and a cost function—via maximum causal entropy IRL (Ziebart et al., 2008; 2010). The resulting algorithm is called generative adversarial imitation learning (GAIL) (Ho & Ermon, 2016), and is closely connected to generative adversarial networks (Goodfellow et al., 2014). GAIL has shown better promise than BC, both in sample efficiency in the number of expert trajectories, and in robustness towards unseen states.

Expert demonstrations of a particular task can display variations, even given the same initial state. This diversity can often stem from multiple human experts performing the same task in different ways. Alternatively, it might arise from pursuing multiple unlabeled objectives, e.g. a robotic arm moving towards different goals. In such scenarios, neither BC nor GAIL enjoys any explicit mechanism to capture different modes of behavior. Particularly, BC averages out actions from different modes. On the other hand, GAIL learns a policy that captures only a subset of control behaviors, because adversarial training is prone to mode collapse (Li et al., 2017; Wang et al., 2017).

Imitation of diverse behaviors is addressed in several prior works. Fei et al. (2020); Merel et al. (2017) use labels of expert modes in training. These works differ from ours in that we assume expert modes are unlabeled. InfoGAIL (Li et al., 2017) and Intention-GAN (Hausman et al., 2017) augment the objective of GAIL with the mutual information between generated trajectories and the corresponding latent codes. This mutual information is evaluated with the help of a posterior network. Wang et al. (2017) use a variational autoencoder (VAE) module to encode expert trajectories into a continuous latent variable. These latent codes are supplied as an additional input to both the policy and the discriminator of GAIL. Nevertheless, as we show in our experiments, both InfoGAIL and VAE-GAIL perform poorly in practice.

Proposing a method for multimodal imitation learning requires finding the correspondence between expert trajectories and a latent variable. Wang et al. (2017) suggest learning this correspondence with VAEs. However, this imposes the difficulty of designing an isolated (as opposed to end-to-end) recurrent module to encode sequences of state-action pairs. Moreover, VAEs are mainly well suited for continuous latent variables. We propose to address these challenges by removing the recurrent encoder module and instead directly searching for an optimal latent variable. Vondrick et al. (2016); Bojanowski et al. (2017); Hoshen et al. (2019) have recently considered similar encoder-free models for non-sequential data.

In this work, we propose an encoder-free model for multimodal BC. We name it self-organizing generative model (SOG), indicating that it can learn semantically smooth latent spaces. We derive SOG as a generative model optimizing the maximum likelihood objective. Moreover, we devise an extension to it enabling training with high dimensional latent spaces. Finally, we combine SOG—as a BC method—with GAIL, to get the best of both worlds. This attempt is inspired by the empirical study of Jena et al. (2020), where in a unimodal setting, a combination of BC with GAIL is shown to yield better performance in fewer training iterations.

In a nutshell, this work mainly focuses on imitation of diverse behaviors while distinguishing different modes. Our contributions are as follows:

1. We introduce SOG as an encoder-free generative model that can be used for multimodal BC. We show that SOG procedure is closely connected to optimization of the maximum likelihood objective. Besides, we introduce an extension to SOG that enables learning with high-dimensional latent spaces.
2. We integrate SOG with GAIL to benefit from accuracy of SOG and robustness of GAIL at the same time. Thus, the resulting model (a) can generate trajectories that are faithful to the expert, (b) is robust to unseen states, (c) learns a latent variable that models variations of the expert policy, and (d) captures all modes successfully without mode collapse.
3. Our empirical results show that our model considerably outperforms the baselines.

2 Background

2.1 Preliminaries

We consider an infinite-horizon discounted Markov decision process (MDP) as a tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$. Namely, $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ represents the state transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_{\geq 0}$ represents the distribution over initial states, and $\gamma \in (0, 1)$ is the reward discount factor.

Let $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ denote a stochastic policy function. Demonstrations of the policy π are sequences of state-action pairs, sampled as follows: $\boldsymbol{s}_0 \sim \rho_0$, $\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$, $\boldsymbol{s}_{t+1} \sim P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$. When states and actions generated by policy π, we define expected return as $\mathbb{E}_\pi\left[r(\boldsymbol{s}, \boldsymbol{a})\right] := \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$.

2.2 Imitation Learning

The goal of imitation learning is to replicate the expert policy from demonstrations without access to the underlying reward function. Two approaches exist for this problem. The first approach is BC, which treats the state-action pairs as inputs and outputs of a policy function, and optimizes for the L_2 loss between the expert actions and the predicted actions. The second approach is GAIL, which optimizes the following objective:

$$\min_{\pi_{\boldsymbol{\theta}}} \max_{D} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\log D(\boldsymbol{s}, \boldsymbol{a})\right] + \mathbb{E}_{\pi_E}\left[\log\left(1 - D(\boldsymbol{s}, \boldsymbol{a})\right)\right] - \lambda H(\pi_{\boldsymbol{\theta}}). \tag{1}$$

In this equation, $\pi_{\boldsymbol{\theta}}$ denotes the policy function realized by a neural network with weights $\boldsymbol{\theta}$, and π_E denotes the expert policy. Moreover, D denotes the discriminative classifier that distinguishes between state-action pairs from the trajectories generated by $\pi_{\boldsymbol{\theta}}$ and π_E. Lastly, $H(\pi_{\boldsymbol{\theta}}) := \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[-\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})]$ represents the discounted causal entropy of the policy $\pi_{\boldsymbol{\theta}}$ (Bloem & Bambos, 2014). We provide additional details about optimization of Equation (1) in Appendix A.

2.3 IMITATION OF DIVERSE EXPERT TRAJECTORIES

We model the variations in the expert policy with a latent variable $\boldsymbol{z}$. In particular, $\boldsymbol{z}$ is sampled from a discrete or continuous prior distribution $p(\boldsymbol{z})$ before each rollout, and specifies the mode of behavior. Therefore, we formalize the generative process of a multimodal expert policy as follows: $\boldsymbol{z} \sim p(\boldsymbol{z}), \boldsymbol{s}_0 \sim \rho_0, \boldsymbol{a}_t \sim \pi_E(\boldsymbol{a}_t|\boldsymbol{s}_t, \boldsymbol{z}), \boldsymbol{s}_{t+1} \sim P(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t, \boldsymbol{s}_t)$.

3 METHOD

In this section, we aim to propose an algorithm for imitation of diverse behaviors. We first introduce a generative model for learning the distribution of arbitrary datasets. Namely, let $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively denote an input data point and a corresponding output data point. We assume that $\boldsymbol{y}$ is generated through a two-stage random process:

1. A latent variable $\boldsymbol{z}$ (independent of $\boldsymbol{x}$) is sampled from a prior distribution $p(\boldsymbol{z})$. This prior can be a given discrete distribution over K categories with probability masses $\pi_1, \ldots, \pi_K$, or an arbitrary continuous distribution, e.g. a multivariate Gaussian $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I})$.
2. A value $\boldsymbol{y}$ is sampled from a conditional distribution $p(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x}; f)$ of the following form:

$$p(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x}; f) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{z}, \boldsymbol{x}), \sigma^2 \boldsymbol{I}), \tag{2}$$

   where f maps $\boldsymbol{z}$ and $\boldsymbol{x}$ to the mean of the distribution, and σ^2 is the isotropic noise variance in the space of $\boldsymbol{y}$.

Now consider a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, consisting of N i.i.d samples generated by the process above. To estimate this model, one needs to restrict the estimation of function f within a family of parameterized functions, e.g. neural networks $f_{\boldsymbol{\theta}}$ with weight parameters $\boldsymbol{\theta}$. One set of models can be derived by maximizing the log-likelihood of the observed data, over both the shared parameters θ, and an assignment $\boldsymbol{z}_i \in \{1, \ldots, K\}$ for each data point $(\boldsymbol{x}_i, \boldsymbol{y}_i)$:

$$\max_{\boldsymbol{\theta}, \boldsymbol{Z}} \sum_{i=1}^{N} \log p(\boldsymbol{y}_i|\boldsymbol{z}_i, \boldsymbol{x}_i; \boldsymbol{\theta}), \quad \text{where } \boldsymbol{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\} \quad \text{and} \quad \boldsymbol{z}_i \in \{1, \ldots, K\}. \tag{3}$$

The rest of this section is organized as follows. In Section 3.1, we introduce an algorithm named Self-Organizing Generative Model (SOG) to efficiently solve the optimization problem of Equation (3). In Section 3.2, we adopt this algorithm for multimodal imitation learning in BC and GAIL settings. In Section 3.3, we briefly mention the relationship between optimization of Equation (3) (as adopted by SOG), and the principle of maximum likelihood estimation, where the data likelihood is marginalized over the latent variable. We expand this relationship in the appendix due to space limits. Finally, in Section 3.4, we elaborate on some additional aspects of the SOG model.

3.1 THE SOG ALGORITHM IN GENERAL

In Algorithm 1, we propose a method to optimize Equation (3) in both cases of discrete and continuous latent variables.

Algorithm 1 Self-Organizing Generative Model (SOG)

Define: Loss function $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \coloneqq ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$
Input: Initial parameters of policy network $\boldsymbol{\theta}_0$
for $epoch = 1, 2, \ldots$ **do**
  for $iteration = 1, 2, \ldots$ **do**
    Sample a minibatch of N_{data} data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ from the dataset.
    for $i = 1, 2, \ldots, N_{\text{data}}$ **do**
      Sample N_z latent codes $\boldsymbol{z}_j \sim p(\boldsymbol{z})$.
      Calculate $\boldsymbol{z}_i \coloneqq \arg\min_{\boldsymbol{z}_j} \mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{z}_j, \boldsymbol{x}_i), \boldsymbol{y}_i)$.
    end for
    Calculate $\mathcal{L}_{\text{SOG}} = \sum_{i=1}^{N_{\text{data}}} \mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{z}_i, \boldsymbol{x}_i), \boldsymbol{y}_i)$ and its gradients w.r.t. $\boldsymbol{\theta}$.
    Update $f_{\boldsymbol{\theta}}$ by stochastic gradient descent on $\boldsymbol{\theta}$ to minimize $\mathcal{L}_{\text{SOG}}$.
  end for
end for

This algorithm performs a two-step optimization over each batch of data: (1) given current $\boldsymbol{\theta}$, find the best latent codes; (2) given the best latent codes, take a gradient step on $\boldsymbol{\theta}$. Both steps decrease the same loss function, which guarantees convergence if gradient steps were taken over the entire data. For mini-batch settings, convergence can be empirically verified through our experiments.

In the case of discrete latent variables, each $\boldsymbol{z}$ correspond to one of the K categories. This effectively results in an exhaustive search over $\boldsymbol{z} \in \{1, \dots, K\}$. In the continuous case, it is impossible to enumerate all candidates of $\boldsymbol{z}$. Therefore, the optimal $\boldsymbol{z}$ is searched over samples of the prior distribution $p(\boldsymbol{z})$.

We justify the name of the algorithm in Section 3.4.

3.2 Adopting SOG for Multimodal Imitation Learning

In the following, we introduce two approaches for multimodal imitation learning: (1) adopting SOG for multimodal BC, and (2) combining multimodal BC with GAIL using SOG.

Multimodal behavior cloning with SOG. We adopt Algorithm 1 for learning the relationship between state-action pairs in a multimodal BC setting. We enforce an additional constraint that the latent code is shared across each trajectory. Hence, we propose Algorithm 2.

Algorithm 2 SOG-BC: Multimodal Behavior Cloning with SOG

Define: Loss function $\mathcal{L}\left(\hat{\boldsymbol{a}}, \boldsymbol{a}\right) \coloneqq ||\hat{\boldsymbol{a}} - \boldsymbol{a}||^2$
Input: Initial parameters of policy network $\boldsymbol{\theta}_0$; expert trajectories $\tau_E \sim \pi_E$
for $iteration = 0, 1, 2, \dots$ **do**
  Sample state-action pairs $\chi_E \sim \tau_E$ with the same batch size.
  Sample N_z latent codes $\boldsymbol{z}_i \sim p(\boldsymbol{z})$ for each trajectory.
  Calculate $\boldsymbol{z}_E \coloneqq \arg\min_{\boldsymbol{z}_i} \hat{\mathbb{E}}_{\tau_E}\left[\mathcal{L}\left(f_{\boldsymbol{\theta}}(\boldsymbol{z}_i, \boldsymbol{s}), \boldsymbol{a}\right)\right]$.
  Calculate $\mathcal{L}_{\mathrm{SOG}} = \hat{\mathbb{E}}_{\tau_E}\left[\mathcal{L}\left(f_{\boldsymbol{\theta}}(\boldsymbol{z}_E, \boldsymbol{s}), \boldsymbol{a}\right)\right]$ and its gradients w.r.t. $\boldsymbol{\theta}$.
  Update $f_{\boldsymbol{\theta}}$ by stochastic gradient descent on $\boldsymbol{\theta}$ to minimize $\mathcal{L}_{\mathrm{SOG}}$.
end for

Multimodal combination of BC and GAIL using SOG. Next, we introduce Algorithm 3, where we combine SOG, as a means for BC, with GAIL to ensure robustness towards unseen states. This algorithm is inspired by Jena et al. (2020), which in a unimodal setting optimizes a weighted sum of BC loss and the GAIL "surrogate" loss (see Appendix A).

Algorithm 3 SOG-GAIL: Multimodal Combination of BC and GAIL

Input: Initial parameters of policy and discriminator networks, $\boldsymbol{\theta}_0, \boldsymbol{w}_0$; expert trajectories $\tau_E \sim \pi_E$
for $i = 0, 1, 2, \dots$ **do**
  Sample a latent code $\boldsymbol{z}_i \sim p(\boldsymbol{z})$, and subsequently a trajectory $\tau_i \sim \pi_{\boldsymbol{\theta}_i}(\cdot|\boldsymbol{z}_i)$.
  Sample state-action pairs $\chi_i \sim \tau_i$ and $\chi_E \sim \tau_E$ with the same batch size.
  Update the discriminator parameters from $\boldsymbol{w}_i$ to $\boldsymbol{w}_{i+1}$ with the gradient

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_{\boldsymbol{w}} \log D_{\boldsymbol{w}}(\boldsymbol{s}, \boldsymbol{a})\right] + \hat{\mathbb{E}}_{\tau_E}\left[\nabla_{\boldsymbol{w}} \log\left(1 - D_{\boldsymbol{w}}(\boldsymbol{s}, \boldsymbol{a})\right)\right].$$

  Calculate the surrogate loss $\mathcal{L}_{\mathrm{PPO}}$ using the PPO rule with the following objective

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_{\boldsymbol{w}} \log D_{\boldsymbol{w}}(\boldsymbol{s}, \boldsymbol{a})\right] - \lambda_H H(\pi_{\boldsymbol{\theta}_i}).$$

  Calculate the SOG loss $\mathcal{L}_{\mathrm{SOG}}$ per Algorithm 2.
  Take a policy step from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_{i+1}$ w.r.t. the objective $\mathcal{L}_{\mathrm{PPO}} + \lambda_S \mathcal{L}_{\mathrm{SOG}}$.
end for

3.3 ANALYSIS OF SOG

Algorithm 1 optimizes the non-marginalized likelihood of Equation (3). However, it can be shown that this algorithm is also related to optimizing the marginalized likelihood of the data, i.e.:

$$\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log \int p(\boldsymbol{y}_i|\boldsymbol{z}, \boldsymbol{x}_i; \boldsymbol{\theta}) p(\boldsymbol{z}) d\boldsymbol{z}. \tag{4}$$

We characterize the relationship between Algorithm 1 and Equation (4) in the appendix due to space limits. We analyse discrete and continuous latent variables separately in Appendices C and D.

3.4 FURTHER PROPERTIES OF SOG

Sample complexity. Training procedure of Algorithm 1, given a high-dimensional continuous latent variable, requires a prohibitively large number of latent code samples (see step 7 of the algorithm). Therefore, in Appendix F, we propose a computationally efficient extension of this algorithm suited for high dimensions of the latent space.

Self-organization in the latent space. In Appendix E, we theoretically derive that latent codes corresponding to nearby data points get organized close to each other. In addition, visual results in Figure 4 and Appendix F show that different regions of the latent space organize towards generating different modes of data. Hence, the phrase "self-organization" in naming of Algorithm 1 is justified.

4 EXPERIMENTS

In our experiments, we demonstrate that our method can recover and distinguish all modes of behavior, and replicate each robustly and with high fidelity. We evaluate our models, SOG-BC and SOG-GAIL, against two multimodal imitation learning baselines: InfoGAIL (Li et al., 2017) and VAE-GAIL (Wang et al., 2017).

4.1 EXPERIMENT SETUP

We adopt an experiment from Li et al. (2017) in which an agent moves freely at limited velocities in a 2D plane. In this experiment, the expert produces three distinct circle-like trajectories. In another series of experiments, we evaluate our model on several complex robotic locomotion tasks, simulated via the MuJoCo simulator (Todorov et al., 2012). These experiments, borrowed from Rakelly et al. (2019), include multimodal tasks with discrete or continuous modes. Discrete tasks include Ant-Fwd-Back, Ant-Dir-6, HalfCheetah-Fwd-Back, Humanoid-Dir-6, Walker2d-Vel-6, and Hopper-Vel-6. Additionally, we conduct two experiments with continuous modes, namely FetchReach and HalfCheetah-Vel. All these experiments are named after standard MuJoCo environments. The suffix "Fwd-Back" indicates that the modes correspond to forward or backward moving directions. Besides, the suffix "Dir-6" implies six moving directions, namely $k \cdot 2\pi/6$ angles. On the other hand, "Vel" and "Vel-6" indicates that the expert selects velocities uniformly at random, resulting in different modes of behavior. The sets of expert velocities are listed in Appendix B. Later, we refer to the environments Ant, HalfCheetah, Humanoid, Walker2d, and Hopper, as "locomotion" tasks.

The FetchReach experiment involves a simulated 7-DoF robotic arm (Plappert et al., 2018). At the beginning of each episode, a target point is uniformly sampled from a cubic region. The expert controls the arm to reach the desired target within a tolerance range.

We explain more details about our experiment setup in Appendix B.

4.2 EVALUATION

Visualization. In Figure 1, we visualize the generated trajectories of four locomotion tasks with discrete modes. Throughout this figure, we can see that both SOG-BC and SOG-GAIL successfully learn the different modes. Moreover, we observe that SOG-BC performs slightly more accurately than SOG-GAIL. We discuss the differences between the two algorithms in a later paragraph, where

we compare them in terms of their robustness. In Figure 2, we visualize the velocities of Walker2d-Vel-6 and Hopper-Vel-6 experiments under different policies. We observe that while SOG-BC successfully distinguishes and reconstructs the desired velocities, the baseline models fail.

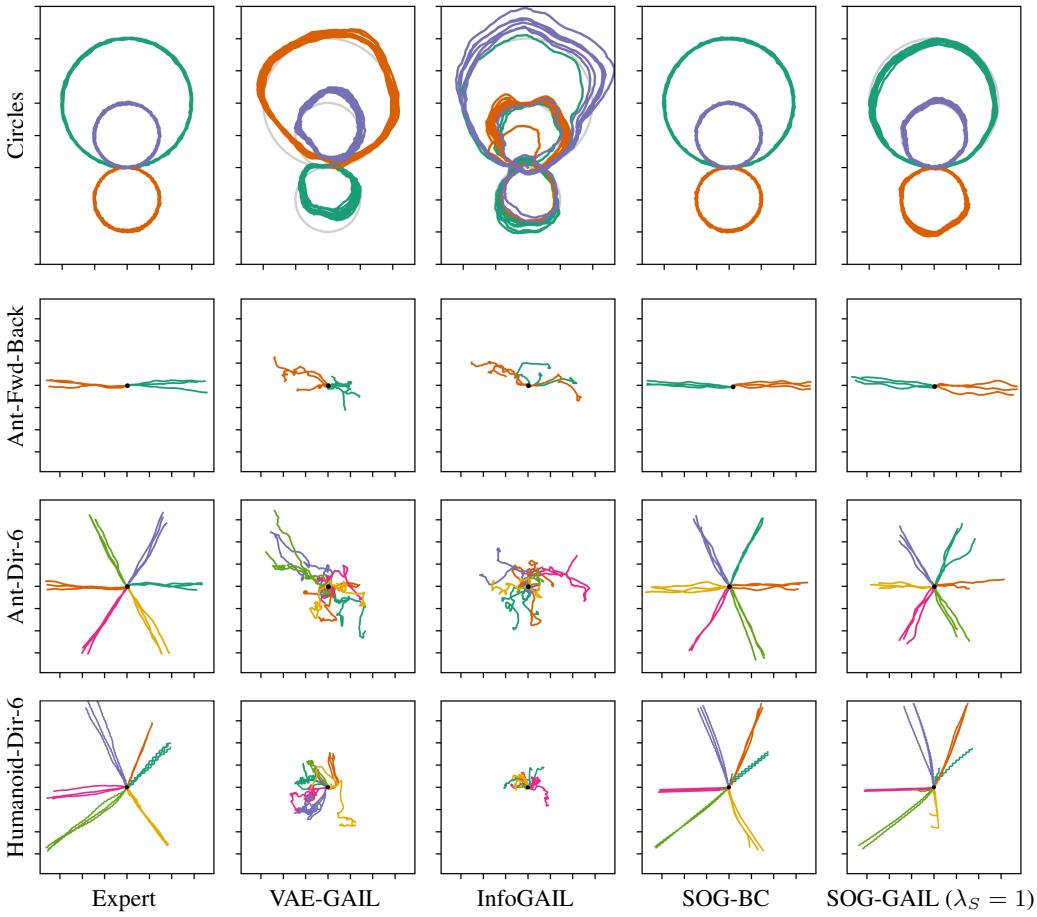


Figure 1: Discrete latent variable: locomotion towards multiple directions. Trajectories of the imitated policies for four tasks. Different latent codes at inference time are color coded. From top to bottom: number of modes are 3, 2, 6, 6; number of trajectories per latent code are 1, 3, 3, 3. Both SOG-BC and SOG-GAIL precisely separate and imitate the modes in all the experiments. However, VAE-GAIL and InfoGAIL fail to separate the modes and to imitate the expert.

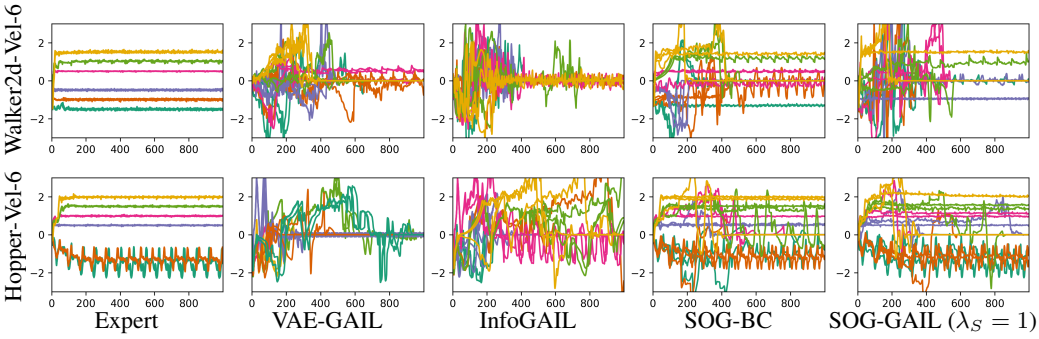
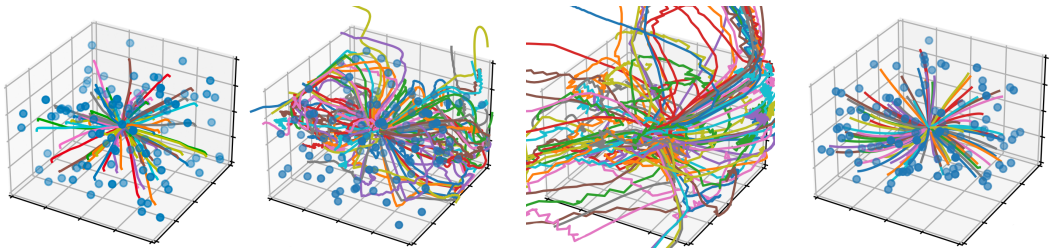


Figure 2: Discrete latent variable: locomotion at six velocities. Velocities generated by the policies vs rollout time steps. Three rollouts per mode are visualized. Each mode is marked with a distinct color. Both SOG-BC and SOG-GAIL separate and imitate most of the modes, producing the desired velocities. VAE-GAIL and InfoGAIL exhibit an inferior performance.

In Figures 3 to 5, we visualize the results of continuous experiments FetchReach and HalfCheetah-Vel. In these figures, SOG-BC and SOG-GAIL produce similar plots, hence we drop the SOG-GAIL plot.

In Figure 3, we plot the trajectories of the Fetch robotic arm for different samples of the latent variable. We observe that SOG-BC is able to control the robotic arm towards any point in the 3D space. In Figure 4, we plot the reached targets for different choices of the latent code, and observe a semantically meaningful interpolation in the latent space. This **video** contains animations of the test rollouts.



(a) Expert **(b)** VAE-GAIL **(c)** InfoGAIL **(d)** SOG-BC

Figure 3: Continuous latent variable: FetchReach. Sample trajectories of the robotic arm. In (b), (d) expert trajectories are embedded and fed to the learned model for reconstruction. In (c), since InfoGAIL lacks any module for encoding expert trajectories, no target point is considered, and random codes are used to generate trajectories. Blue circles mark different targets. Colors of the trajectories are chosen at random for better visual distinction. Compared to the baselines, SOG-BC better reaches different targets.

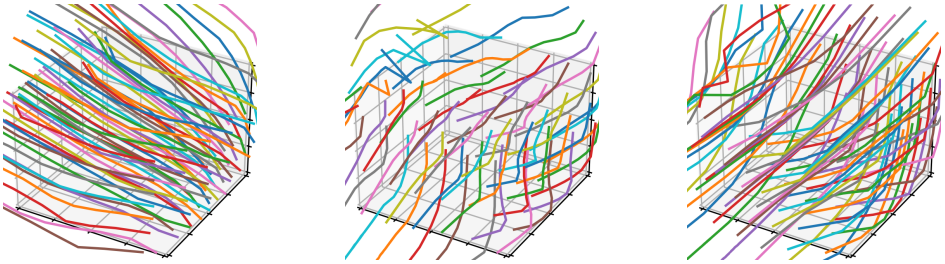
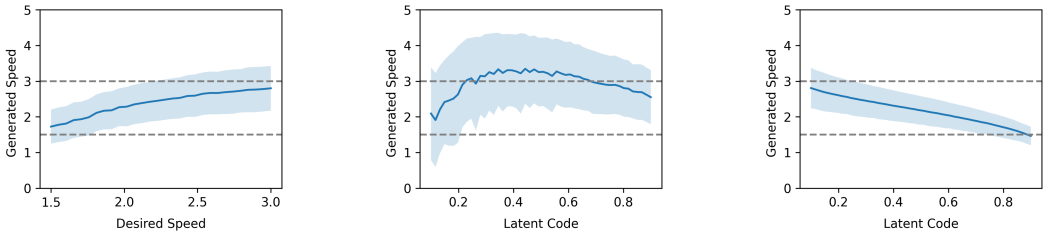


Figure 4: Self-organization in the latent space: FetchReach. In each plot, two dimensions among the three dimensions of the latent space are fixed, whereas the third is varied. Each line corresponds to the location of the robotic arm at the final state of different trajectories. In each line, two latent dimensions take a fixed value, while the third varies. Lines are randomly colored for better visual distinction.

In Figure 5, we plot the spread of instantaneous horizontal velocities of the HalfCheetah agent as the latent variable varies. In SOG-BC and InfoGAIL, the horizontal axis corresponds to the 1-D latent code used to train the model. However, since VAE-GAIL requires a high dimensional latent variable for best performance, the correspondence between latent codes and the model generated horizontal velocities cannot be directly visualized. Therefore, we utilize the following fact: if the latent embedding of an expert trajectories is fed to the learned policy, the generated trajectory produces similar velocities as the expert trajectory. Thus, for VAE-GAIL we plot the variations in the generated velocities as a function of the target velocity of the corresponding expert trajectory. We observe that compared to the baselines, SOG-BC better associates the latent variable to goal velocities in the range $[1.5, 3]$. Also, SOG-BC produces a narrower error band, which indicates better certainty given fixed latent codes. This **video** illustrates the imitated behavior of SOG-BC at different velocities as the latent code varies.

Metrics. In Table 1, we use ground-truth reward functions to evaluate the collected rewards under different policies. We explain these ground-truth rewards in Appendix B. In each entry of the table, we report the best iteration in terms of the mean reward. Particularly, we calculate mean and std of the rewards across 100 rollouts. To find the mode-specific reward for each latent code, we select a



(a) VAE-GAIL **(b)** InfoGAIL **(c)** SOG-BC

Figure 5: Continuous latent variable: HalfCheetah-Vel. In (a), since VAE-GAIL uses a high latent dimension, in the horizontal axis we consider the desired velocity of the expert trajectory corresponding to the latent embedding. However, in (b), (c) the horizontal axis corresponds to (the CDF of) the 1-D Gaussian latent variable. The CDF is applied for better visualization. In SOG-BC, different velocities in range $[1.5, 3]$ are replicated with a higher certainty.

Table 1: Mean and standard deviation of the rewards for locomotion tasks.

Data set	SOG-BC	SOG-GAIL ($\lambda_S = 1$)	InfoGAIL	VAE-GAIL	Expert
Circles	$\mathbf{992.1 \pm 1.0}$	985.9 ± 10.8	766.0 ± 67.9	912.3 ± 8.6	998.3 ± 0.1
Ant-Fwd-Back	$\mathbf{1165.2 \pm 32.1}$	1101.0 ± 61.7	220.6 ± 296.3	-385.3 ± 67.0	1068.7 ± 109.7
Ant-Dir-6	$\mathbf{1073.2 \pm 206.6}$	1023.2 ± 166.1	-14.5 ± 87.6	-572.9 ± 62.9	1031.7 ± 253.7
HalfCheetah-Fwd-Back	221.6 ± 957.3	$\mathbf{1532.6 \pm 148.5}$	484.2 ± 919.4	84.0 ± 152.2	1686.0 ± 135.4
Humanoid-Dir-6	$\mathbf{5996.0 \pm 326.4}$	5457.8 ± 640.9	1333.94 ± 461.4	2285.5 ± 946.1	6206.6 ± 292.6
Walker2d-Vel-6	$\mathbf{1915.3 \pm 402.1}$	1698.7 ± 650.0	947.3 ± 105.3	1183.6 ± 68.0	1964.6 ± 394.6
Hopper-Vel-6	$\mathbf{2222.3 \pm 431.1}$	2015.30 ± 547.3	1216.8 ± 70.8	1065.8 ± 30.1	2229.7 ± 438.6

correspondence of latent codes to actual modes that gives the best expected sum of rewards. This metric measures whether all modes are separated and properly imitated, because each mode has a distinct reward function. Across all experiments, SOG-BC performs significantly better than the baselines. The only exception is HalfCheetah-Dir, where SOG-GAIL performs better. We attribute this latter observation to the difficulty of the task, which is alleviated by the explorations of GAIL.

In Table 2, we evaluate the FetchReach experiment through two metrics. First, we collect the achieved targets under different policies, and clip them within the cubic region from which the targets are sampled. Then, we quantify the desired uniformity of the distribution of the achieved targets over the cube. Since uniform distribution is the maximum entropy distribution over rectangular supports, we alternatively estimate the entropy of the achieved targets via the method of Kozachenko & Leonenko (1987). As a second metric, we embed expert trajectories, and measure the hit rate of the reconstructed trajectories given the same targets as the expert.

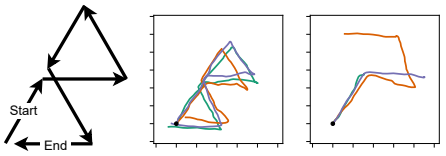
In the HalfCheetah-Vel experiment, to measure the correspondence between the latent variable and generated instantaneous horizontal velocities, we estimate their mutual information through the method of Kraskov et al. (2004). The results are presented in Table 3.

Table 2: Metrics for reached targets in the FetchReach experiment: average hit rate and estimated entropy of achieved targets (higher is better).

Metric	SOG-BC	SOG-GAIL ($\lambda_S = 1$)	InfoGAIL	VAE-GAIL	Expert
Entropy (nats)	$\mathbf{2.05}$	1.89	0.27	0.85	2.18
Hit Rate	$\mathbf{100\%}$	97.0%	N/A	18.6%	100%

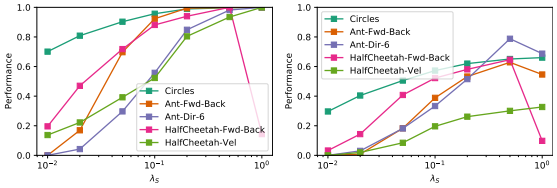
Table 3: Mutual information between the latent variable (target velocity of the embedded trajectory in the case of VAE-GAIL) and generated velocities in HalfCheetah-Vel.

SOG-BC	SOG-GAIL ($\lambda_S = 1$)	InfoGAIL	VAE-GAIL
$\mathbf{1.584}$	1.431	0.145	0.750



(a) Goal **(b)** SOG-GAIL **(c)** SOG-BC

Figure 6: Robustness of SOG-BC vs SOG-GAIL: Ant-Dir-6. To create unseen situations, we switch the latent codes to those corresponding to the directions shown in (a). In (b), (c) we show three trajectories generated in different colors. We observe that the ant agent trained with SOG-GAIL performs the desired task flawlessly, while the one trained with SOG-BC often topples over and fails the task.



(a) Unperturbed policy **(b)** Perturbed policy

Figure 7: Optimal choice of λ_S. Parameter λ_S is the coefficient of the SOG loss in Algorithm 3. Performance of Algorithm 3 is illustrated as λ_S varies. We consider two cases of with/without perturbations. These perturbations are imposed by taking random actions with a chance of 20%. The vertical axis is normalized such that the best performance in unperturbed settings achieves a score of 1, and a random policy achieves 0. In Figure (b), we observe that the policy trained with higher values of λ_S shows less robustness to perturbations in several experiments .

Robustness. We design two experiments to show that SOG-GAIL learns policies that are robust towards unseen states. In the first experiment, upon generating test trajectories for Ant-Dir-6, we switch the movement angles to all six possible directions. Since the motion angles within each expert trajectory are consistent, such changes create unseen circumstances. We observe that maneuvers with acute angles causes the model trained with SOG-BC to topple over and make the episode terminate. In contrast, SOG-GAIL manages to switch between all six directions.

Secondly, we vary the coefficient λ_S in SOG-GAIL. High values of λ_S in limit lead to SOG-BC. We observe that increasing λ_S improves scores in Table 1 across all experiments (except for HalfCheetah-Dir). We now perturb each learned policy by taking random actions with a 20% probability. We observe that under this perturbation, several of the experiments perform worse at higher values of λ_S. This indicates that combination of SOG with GAIL is more robust to perturbations. The results of our two tests are respectively depicted in Figures 6 and 7.

4.3 Comparison

Our empirical results show that SOG-BC and SOG-GAIL distinguish and learn different modes of behavior better than the baseline models, particularly in the discrete case. This advantage can be attributed to the differences between the methods in inducing mode separation in GAIL. Specifically, VAE-GAIL and InfoGAIL directly modify the GAIL optimization to encourage multimodality. However, our method learns multiple modes in GAIL indirectly and through integration with multimodal BC.

5 Concluding Remarks

We introduce a generative model that enables imitation of diverse behaviors when expert demonstrations exhibit a mixture of behaviors. We provide theoretical analysis showing the objective of our model is closely connected to marginalized likelihood of the data. the conditional distribution between input and output variables, when the underlying relationship depends on an unobserved variable. This model shows better performance for both cases of discrete and continuous latent variables. Across multiple complex experiments, we show that our model precisely reconstructs expert trajectories, while simultaneously distinguishing the underlying modes. We also demonstrate that a natural combination of our generative model with adversarial training boosts the robustness of the imitated policies to unseen states.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *arXiv preprint arXiv:1707.01495*, 2017.

Christopher M Bishop. *Pattern Recognition and Machine Learning*. springer, 2006.

Michael Bloem and Nicholas Bambos. Infinite Time Horizon Maximum Causal Entropy Inverse Reinforcement Learning. In *53rd IEEE Conference on Decision and Control*, pp. 4911–4916. IEEE, 2014.

Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the Latent Space of Generative Networks. *arXiv preprint arXiv:1707.05776*, 2017.

Siddhartha Chib. Marginal likelihood from the Gibbs output. *Journal of the american statistical association*, 90(432):1313–1321, 1995.

Cong Fei, Bin Wang, Yuzheng Zhuang, Zongzhang Zhang, Jianye Hao, Hongbo Zhang, Xuewu Ji, and Wulong Liu. Triple-gail: a Multi-Modal Imitation Learning Framework with Generative Adversarial Nets. *arXiv preprint arXiv:2005.10622*, 2020.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv preprint arXiv:1406.2661*, 2014.

Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav Sukhatme, and Joseph Lim. Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets. *arXiv preprint arXiv:1705.10479*, 2017.

Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. *arXiv preprint arXiv:1606.03476*, 2016.

Yedid Hoshen, Ke Li, and Jitendra Malik. Non-Adversarial Image Synthesis with Generative Latent Nearest Neighbors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5811–5819, 2019.

Rohit Jena, Changliu Liu, and Katia Sycara. Augmenting GAIL with BC for sample efficient imitation learning. *arXiv preprint arXiv:2001.07798*, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating Mutual Information. *Physical review E*, 69(6):066138, 2004.

Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable Imitation Learning from Visual Demonstrations. *arXiv preprint arXiv:1703.08840*, 2017.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Dean A Pomerleau. Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural computation*, 3(1):88–97, 1991.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. In *International conference on machine learning*, pp. 5331–5340. PMLR, 2019.

Stéphane Ross and Drew Bagnell. Efficient Reductions for Imitation Learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668. JMLR Workshop and Conference Proceedings, 2010.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating visual representations from unlabeled video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 98–106, 2016.

Ziyu Wang, Josh Merel, Scott Reed, Greg Wayne, Nando de Freitas, and Nicolas Heess. Robust Imitation of Diverse Behaviors. *arXiv preprint arXiv:1707.02747*, 2017.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum Entropy Inverse Reinforcement Learning. In *Aaai*, volume 8, pp. 1433–1438. Chicago, IL, USA, 2008.

Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. Modeling Interaction via the Principle of Maximum Causal Entropy. In *ICML*, 2010.

A Optimization of GAIL

GAIL optimizes the objective of Equation (1) in two alternating steps: (1) a gradient ascent step to increase it with respect to the discriminator parameters, and (2) a Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a) or Proximal Policy Optimization (PPO) (Schulman et al., 2017) step to decrease it with respect to the policy parameters.

TRPO and PPO are policy optimization algorithms that take the biggest possible policy improvement step without stepping so far that performance accidentally deteriorates. In particular, PPO achieves this by optimizing the following "surrogate" objective at i^{th} iteration:

$$\mathcal{L}_{\mathrm{PPO}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}_i, \boldsymbol{\theta}) = \min\left(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}_i}(\boldsymbol{a}|\boldsymbol{s})} A^{\pi_{\boldsymbol{\theta}_i}}(\boldsymbol{s}, \boldsymbol{a}),\ \ g(\epsilon, A^{\pi_{\boldsymbol{\theta}_i}}(\boldsymbol{s}, \boldsymbol{a}))\right), \tag{5}$$

where A is the advantage function (Schulman et al., 2015b), and g is a "clipping" function for some small parameter $\epsilon > 0$:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases} \tag{6}$$

B Details of the Experiment Setup

In the Circles environment, the observed state at time t is a concatenation of positions from time $t-4$ to t. Expert demonstrations appear in three modes, each trying to produce a distinct circle-like trajectory. The expert tries to maintain a $(2\pi/100)$ rad/s angular velocity along the perimeter of a circle. However, its displacement at each step incurs a 10%-magnitude 2D Gaussian noise from the environment.

To train an expert policy for MuJoCo locomotion tasks, we used Pearl (Rakelly et al., 2019), a few-shot reinforcement learning algorithm. For the FetchReach experiment, we considered a pretrained DDPG+HER model as expert (Lillicrap et al., 2015; Andrychowicz et al., 2017).

In Section 4, we measured the rewards of different experiments as follows. For the Circles experiment, we applied a Gaussian kernel to the distance of the agent from the perimeter of each of the three reference trajectories. Hence, each step yields a reward between 0 and 1. For all MuJoCo locomotion environments, i.e. Ant, HalfCheetah, Walker2d, Hopper, and Humanoid, we used the original rewards introduced in Rakelly et al. (2019). We ran all experiments with four random seeds and presented the results for the best training iteration in terms of expected sum of rewards.

In experiment with varied velocities, having a suffix "Vel" in their name, target velocities of the expert are sampled uniformly from the values listed in Table 4. As a reference, HalfCheetah has a torso length of around 1 m.

Table 4: Set of possible expert velocities in different environments.

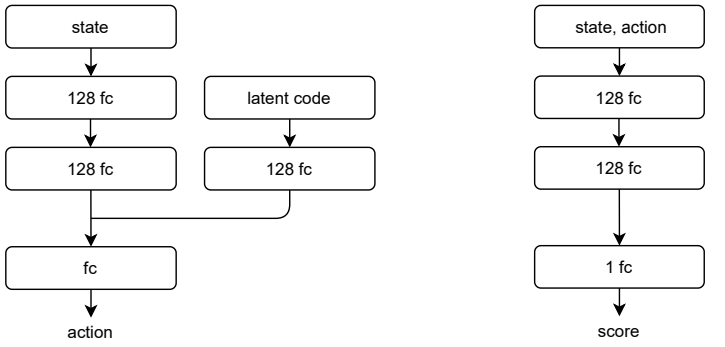
Environment	Walker2d-Vel	Hopper-Vel	HalfCheetah-Vel
Possible Velocities	$\{-1.5, -1, -0.5, 0.5, 1, 1.5\}$	$\{-2, -1.5, 0.5, 1, 1.5, 2\}$	Interval of $[1.5, 3]$

Episode lengths in different environments are listed in Table 5.

Table 5: Episode lengths in different environments.

Environment	Circles	Ant	HalfCheetah	Humanoid	Walker2d	Hopper	FetchReach
Episode Length	1000	200	200	1000	1000	1000	50

We used the same network architectures across all algorithms. The policy network passes the states and latent codes through separate fully connected layers, and then adds them and feeds them to the final layer. The discriminator receives state-action pairs and produces scores for the GAN objective. The posterior network for InfoGAIL has the same architecture as the discriminator, except it has a



(a) Policy network **(b)** The discriminator

Figure 8: Neural network architectures. (a) the policy network, (b) the discriminator in GAIL.

final softmax layer in the discrete case. For continuous latent codes, InfoGAIL adopts a Gaussian posterior. The architectures are illustrated in Figure 8.

For each of the algorithms SOG-GAIL, InfoGAIL, and VAE-GAIL, we pretrain the policy network with behavior cloning for better performance. Since the policy network has an input head for the latent code, in SOG-GAIL we provide the latent codes according to Algorithm 2. Also, because InfoGAIL is not equipped with a module to extract the latent codes prior to training, we feed random samples as the input latent codes during pretraining.

Across all experiments, we varied the coefficients λ_I, corresponding to the mutual information term in InfoGAIL, and λ_S from SOG-GAIL (Algorithm 3), at values of $\lambda = i * 10^j;\ i = 1, 2, 5;\ 0.01 \leq \lambda \leq 1$.

C Discrete Latent Variables

Expectation-maximization (EM) is a well-known method for optimizing the maximum likelihood objective in the models involving latent variable. In this section, we demonstrate how the iterative procedure of Algorithm 1 can be derived as a special case of the EM algorithm for Equation (4).

C.1 The EM Algorithm in General

First, we introduce the EM method in general. Let $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N, \boldsymbol{Y} = \{\boldsymbol{y}_i\}_{i=1}^N, \boldsymbol{Z} = \{\boldsymbol{z}_i\}_{i=1}^N$ constitute a dataset generated by the procedure described in Section 3, and let $\boldsymbol{\Pi} = \{\pi_k\}_{k=1}^K$ denote the set of probability masses for the discrete latent codes. The EM framework defines a total likelihood function for an arbitrary distribution $q(\boldsymbol{Z})$ as follows:

$$\mathcal{L}(q(\boldsymbol{Z}); \boldsymbol{\theta}, \boldsymbol{\Pi}) = \mathbb{E}_{\boldsymbol{Z}\sim q(\boldsymbol{Z})} \log \{p(\boldsymbol{Y}, \boldsymbol{Z}|\boldsymbol{X};\ \boldsymbol{\theta}, \boldsymbol{\Pi})\} + H(q(\boldsymbol{Z})) \tag{7}$$

$$= \mathbb{E}_{\boldsymbol{Z}\sim q(\boldsymbol{Z})} \log \{p(\boldsymbol{Y}|\boldsymbol{X};\ \boldsymbol{\theta}, \boldsymbol{\Pi})\} + \mathbb{E}_{\boldsymbol{Z}\sim q(\boldsymbol{Z})} \log \left\{ \frac{p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{Y};\ \boldsymbol{\theta}, \boldsymbol{\Pi})}{q(\boldsymbol{Z})} \right\} \tag{8}$$

$$\equiv \log p(\texttt{data}|\boldsymbol{\theta}, \boldsymbol{\Pi}) - D_{\mathrm{KL}}\big(q(\boldsymbol{Z}) \,||\, p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{Y};\ \boldsymbol{\theta}, \boldsymbol{\Pi})\big) \tag{9}$$

After initialization of model parameters, the optimization proceeds by alternating between two steps:

1. **Expectation Step (E Step).**

$$q^{t+1}(\boldsymbol{Z}) \leftarrow \arg\max_q \mathcal{L}(q;\ \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) = p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{Y};\ \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) \equiv \prod_{i=1}^N r_{i,\boldsymbol{z}_i}^t \tag{10}$$

2. **Maximization Step (M Step).**

$$\boldsymbol{\theta}^{t+1}, \boldsymbol{\Pi}^{t+1} \leftarrow \arg\max_{\boldsymbol{\theta}, \boldsymbol{\Pi}} \mathcal{L}(q^{t+1};\, \boldsymbol{\theta}, \boldsymbol{\Pi}) \tag{11a}$$

$$= \arg\max_{\boldsymbol{\theta}, \boldsymbol{\Pi}} \mathbb{E}_{\boldsymbol{Z} \sim q^{t+1}}[\log P(\boldsymbol{Y}, \boldsymbol{Z} | \boldsymbol{X};\, \boldsymbol{\theta}, \boldsymbol{\Pi})] \tag{11b}$$

$$= \arg\max_{\boldsymbol{\theta}, \boldsymbol{\Pi}} \sum_{i=1}^{N} \sum_{k=1}^{K} r_{ik}^{t} (\log \pi_k + \log p(\boldsymbol{y}_i | \boldsymbol{x}_i, \boldsymbol{z}_i = k;\, \boldsymbol{\theta})) \tag{11c}$$

where

$$r_{ik}^{t} = p(\boldsymbol{z} = k \mid \boldsymbol{x}_i, \boldsymbol{y}_i;\, \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) = \frac{p(\boldsymbol{y}_i | \boldsymbol{x}_i, \boldsymbol{z} = k;\, \boldsymbol{\theta}^t)\, \pi_k^t}{\sum_{l=1}^{K} p(\boldsymbol{y}_i | \boldsymbol{x}_i, \boldsymbol{z} = l;\, \boldsymbol{\theta}^t)\, \pi_l^t}. \tag{12}$$

Finally, we introduce the update rules in the M step for $\boldsymbol{\theta}, \boldsymbol{\Pi}$. Since the M step does not have a closed-form solution w.r.t. $\boldsymbol{\theta}$, on can instead perform a batch gradient ascent step. This approach for an intractable M step is known as generalized EM (Bishop, 2006). Also, it is straightforward to derive the following update rule for $\boldsymbol{\Pi}$ by maximizing the objective of Equation (11c) subject to the constraint $\sum_{k=1}^{K} \pi_k = 1$:

$$\pi_k^t \leftarrow \frac{\sum_i r_{ik}^t}{\sum_{il} r_{il}^t}. \tag{13}$$

The E step can be carried out in two ways:

1. **Soft.** using all the posterior probabilities in Equation (12), so that each data point has a soft-distribution over the latent codes at each iteration step. This corresponds to the original EM algorithm above.
2. **Hard.** approximating the posterior probabilities by a one-hot distribution, driven by the best latent code for the M step. The method leads to Algorithm 1.

In the following subsections, we elaborate on each method.

C.2 Hard Assignment EM (Algorithm 1)

Hard assignment EM has a long history in the ML community and the assumptions made to derive this algorithm are well known and are used to motivate and justify the K-means algorithm from an EM perspective (Bishop, 2006). The underlying assumptions for hard EM are that $\pi_k = 1/K;\ k = 1, \ldots, K$, in the generative model, and that in the update equations, we replace the probabilities r_{ik} in Equation (12) with binary values. This hard assignment is performed by allocating all the probability mass to the categorical variable for which the posterior probability is the maximum.

Hence, the E step becomes

$$\gamma_{ik}^{t} = p(\boldsymbol{z} = k \mid \boldsymbol{x}_i, \boldsymbol{y}_i;\, \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) = \begin{cases} 1, & k = \arg\min_k ||f(\boldsymbol{z}_k, \boldsymbol{x}_i;\, \boldsymbol{\theta}) - \boldsymbol{y}_i|| \\ 0, & \text{otherwise} \end{cases}, \tag{14}$$

and the M step of Equation (11c) (w.r.t. $\boldsymbol{\theta}$) simplifies to:

$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \min_k ||f(\boldsymbol{z} = k, \boldsymbol{x}; \boldsymbol{\theta}) - \boldsymbol{y}||^2; \tag{15}$$

which is equivalent to the update step in Algorithm 1. For our generative model, the following observations make the hard EM an appropriate choice:

1. *Joint learning of latent codes and the network parameters:* The parameterized function $f(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta})$, embodied by a large enough DNN, has almost-universal approximation capability, and thus any mode in the dataset can be well approximated by setting a fixed one-hot code $\boldsymbol{z}$, and training the network. Equally, if the modes are already known then one could train the network jointly across all modes by assigning the same one-hot code for data points belonging to the same mode. Of course, we do not know the mode of each data point, but for any given $\boldsymbol{\theta}$, we can use the network to assign the most likely mode to a data point, and then update $\boldsymbol{\theta}$ to reinforce that it would most likely come true in the future: that is, the network is updated so as increase the likelihood of the data point for the currently assigned mode. This is where

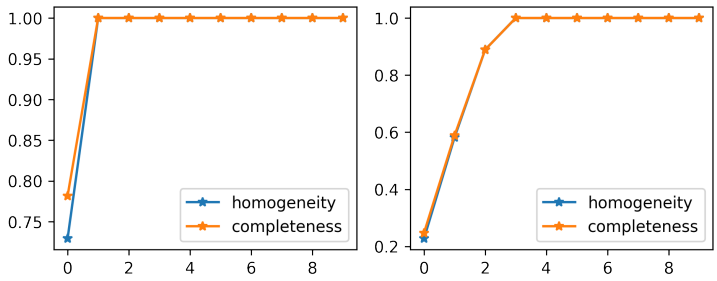


Figure 9: Self-organization and Mode Assignment: In the experiment described in Appendix C, the assignment of the same winning latent code to data points belonging to the same mode in the ground truth data happens very quickly. We create additional one dataset with less separation between ground truth modes compared to the one we used in Appendix C. By the end of epoch 1 for data where the modes are very well separated, and by the end of epoch 3 in data with modes that are closer together, the mode separation is almost complete.

the self-organization principles discussed Appendix E play a crucial role as well. Under certain assumptions about the smoothness of $f(\boldsymbol{z}, \boldsymbol{x}; \boldsymbol{\theta})$, it shows that *two sufficiently close data points will have nearby winning latent codes (or the same winning latent codes in the discrete case* (see Proposition E.2). Thus, the data points belonging to the same data-mode will have the same winning latent code, and thus boosting the specialization of the neural network to that mode. The ground truth mode heterogeneity of data points assigned to any latent code will quickly go to zero.

2. *Log likelihood of data in SOG vs soft EM:* When the modes of the dataset are well separated, as shown in Appendix C.3, soft EM also converges to approximately one-hot posterior codes. Given any data point $(\boldsymbol{x}, \boldsymbol{y})$, the posterior probability of the winning code $\boldsymbol{z}^*$ at convergence, are $p_{EM}(\boldsymbol{z} = \boldsymbol{z}^* | \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = 1 - \epsilon_E$, and let us assume that $p_{SOG}(\boldsymbol{z} = \boldsymbol{z}^* | \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = 1 - \epsilon_S$, and that $\epsilon_S < \epsilon_E$. Next consider the standard data likelihood expression for any $k \in \{1, \ldots, K\}$:

$$\underbrace{\log p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})}_{\substack{\text{marginal data} \\ \text{log-likelihood for a Model}}} = \underbrace{\log p(\boldsymbol{y}|\boldsymbol{z} = k, \boldsymbol{x}; \boldsymbol{\theta})}_{\text{-(Model Loss)}} - \underbrace{\log \frac{p(\boldsymbol{z} = k|\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}{p(\boldsymbol{z})}}_{\text{gap}}. \tag{16}$$

Assuming a balanced data set, we have $p(\boldsymbol{z} = k) = 1/K$, and putting $\boldsymbol{z} = z^*$ we get:

$$\log p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}_S) - \log p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}_E) = \\ [\underbrace{\log p(\boldsymbol{y}|\boldsymbol{z} = z^*, \boldsymbol{x}; \boldsymbol{\theta}_S)}_{\text{-(SOG Loss)}} - \underbrace{\log p(\boldsymbol{y}|\boldsymbol{z} = z^*, \boldsymbol{x}; \boldsymbol{\theta}_E)}_{\text{-(EM Loss)}}] + \log(1 - \epsilon_E) - \log(1 - \epsilon_S). \tag{17}$$

As discussed next, at convergence we expect -(SOG Loss) $\geq$ -(EM-Loss), while $\log(1 - \epsilon_E) - \log(1 - \epsilon_S) \approx \epsilon_S - \epsilon_E < 0$. Thus, the data log likelihood of both the models will be almost the same at convergence.

C.3 Soft Assignment EM

First we observe that if the dataset has distinct modes, then the posterior distribution in the EM approach always tends to one-hot. For simplicity, let us assume that the modes in dataset are balanced and have the same number of data points. This leads to uniform cluster prior distribution at convergence: $\pi_k^t \to 1/K, k = 1, \ldots, K$. The posterior assignment distribution of a data point over the K clusters (Equation (12)) is then simply proportional to the conditional likelihood for the clusters, i.e.

$$r_{ik}^t = p(z = k \mid \boldsymbol{x}_i, \boldsymbol{y}_i;\, \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) = \frac{p(\boldsymbol{y}_i|\boldsymbol{x}_i, z = k;\, \boldsymbol{\theta}^t)\, \pi_k^t}{\sum_{l=1}^K p(\boldsymbol{y}_i|\boldsymbol{x}_i, z = l;\, \boldsymbol{\theta}^t)\, \pi_l^t} \approx \frac{p(\boldsymbol{y}_i|\boldsymbol{x}_i, z = k;\, \boldsymbol{\theta}^t)}{\sum_{l=1}^K p(\boldsymbol{y}_i|\boldsymbol{x}_i, z = l;\, \boldsymbol{\theta}^t)}$$

Consider a specific data point $(\boldsymbol{x}_i, \boldsymbol{y}_i)$. Along the training using EM algorithm, at some step t, one of the cluster, κ, will have higher likelihood compared to the alternatives (because the likelihood conditioned on different clusters remaining perfectly equal is very unlikely), i.e.

$$p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = \kappa; \theta^t) > p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = l; \theta^t), \quad \text{if } l \neq \kappa, \tag{18}$$

This leads to a higher posterior for this "winning" cluster compared to its alternatives:

$$r^t_{i\kappa} > r^t_{il}, \quad \text{if } l \neq \kappa,$$

Therefore, due to the M step, the "winning" cluster gains more weight in the loss function, and tends to be improved more than the alternatives, so that Equation (18) remains to hold, and the "winning margin", $\dfrac{p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = \kappa)}{p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = l)}$, even increases.

As the result of such positive feedback, the posterior distribution tends to one-hot and $r^t_{i\kappa} \approx 1$, while $r^t_{il} \approx 0$ for $l \neq \kappa$.

Now we show that this EM posterior can not be exactly one-hot and will always have a finite gap when the likelihood function is assumed to be Gaussian, i.e. $p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z) = \mathcal{N}(\boldsymbol{y}; f_\theta(z, \boldsymbol{x}), \sigma^2 \boldsymbol{I})$, and the gap is a function of σ. Because the Gaussian distribution is non-zero in the whole domain, and the prediction function f_θ has finite output, the posterior for the non-dominating clusters is always finite, i.e. $r^t_{il} > 0$ is finite, even for $l \neq \kappa$. This has two consequences:

1. The non-dominating clusters will have non-negligible contribution to the loss (unlike in the case of SOG where the search is done over one-hot codes). The loss of EM algorithm (we focus on the M-step, because it has direct correspondence to the loss of SOG) for a data point $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is given by the expected negative log probability

$$\mathcal{L}(\theta) = -\sum_l r^t_{il} \log p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = l \ ; \ \theta).$$

   Now that the r^t_{il} is not one-hot, $\log p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = l \ ; \ \theta)$ will have contribution to the loss for $l \neq \kappa$, making it larger than the loss attained by SOG, where $r^t_{il} = 0$ for $l \neq \kappa$. This is confirmed by the plot Figure 12b in where the loss of EM drops significantly when calculated after converting the posterior to exactly one-hot like in SOG.

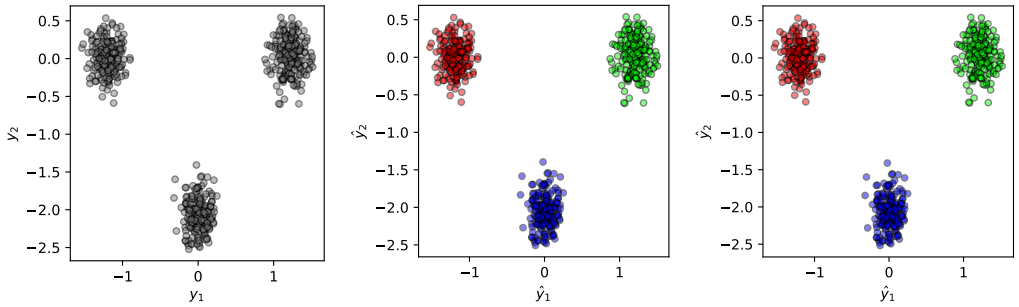
2. As a consequence, the contribution to the loss from the non-dominating clusters will keep the convergence point slightly away from the optimal solution that maximizes the dominating conditional likelihood $p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = \kappa \ ; \ \theta)$.

Experiment setup. In this section, we illustrate some of the points we have made above regarding soft and hard variants of EM, using a toy experiment. This experiment demonstrates that both variants are able to recover the structure (three modes) in the dataset and illustrate the effectiveness of Algorithm 1. We use a multi-modal linear generative model to generate the dataset of tuples $\{(\boldsymbol{x}_i, \boldsymbol{y}_i, z_i)\}_{i=1}^N$. Concretely, $\boldsymbol{x}_i \sim \mathcal{N}(0, \boldsymbol{I}_2)$, $\boldsymbol{y}_i = \boldsymbol{x}_i + \boldsymbol{w}_{z_i} + \boldsymbol{\epsilon_i}$, where $z_i \sim \text{Categorical}(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, $\boldsymbol{\epsilon}_i \sim \mathcal{N}(0, 0.01^2 \boldsymbol{I}_2)$. That is, $\boldsymbol{x}_i \in \mathbb{R}^2$ is drawn from a unit Gaussian distribution, and $\boldsymbol{y}_i \in \mathbb{R}^2$ is computed by offsetting $\boldsymbol{x}_i$ by amount of $\boldsymbol{w}_{z_i} \in \mathbb{R}^2$ according to its corresponding mode $z_i \in \{1, 2, 3\}$.

In both the EM and SOG we train a linear model $f_{\boldsymbol{\theta}}(z, \boldsymbol{x}) = \boldsymbol{\theta} \begin{bmatrix} \texttt{one\_hot}(z) \\ \boldsymbol{x} \end{bmatrix}$, parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{5 \times 2}$ on the data such that the marginal likelihood is maximized.

Results. The experiments on the toy example showed that both variants are able to produce satisfactory fitting results. As we can see in Figure 10, both EM and SOG recover the structure (three modes) existing in the ground truth $\boldsymbol{Y}$ (colors are randomly assigned to distinguish different modes). Furthermore, as expected, the soft variant ultimately leads to one-hot posterior distribution among latent codes in the discrete code case, which is assumed by SOG. Figure 11 shows the posterior distribution from EM algorithm gradually converges from almost uniform distribution in epoch 2 to one-hot in epoch 100 when its loss converges, which justifies SOG's way of approximation to the posterior. Although there are no significant visual distinctions among the synthesized samples in Figure 10, we comment that SOG is more reliable to attain good performance in terms of loss value and convergence speed compared to EM algorithm. Below we provide a detailed analysis.

In Figure 12a, the training curves of the two algorithms are plotted in both linear and logarithm scale, and they show that SOG not only has a faster convergence speed, but also a lower loss at convergence, despite the fact that the posterior distribution of the EM algorithm goes to one-hot just like the hard-assignment used by SOG (shown in Figure 11). This raises the question: now that EM behaves like SOG asymptotically, why does it have inferior performance? The reason is that, limited by the Gaussian likelihood assumption (which is boundless), the EM approach can



(a) The ground truth $\boldsymbol{Y}$ **(b)** The EM predictions **(c)** The SOG predictions

Figure 10: Three-mode clustering toy example. Ground truth and model predictions at convergence. Points are colored-coded with the latent code assignments from the EM algorithm.

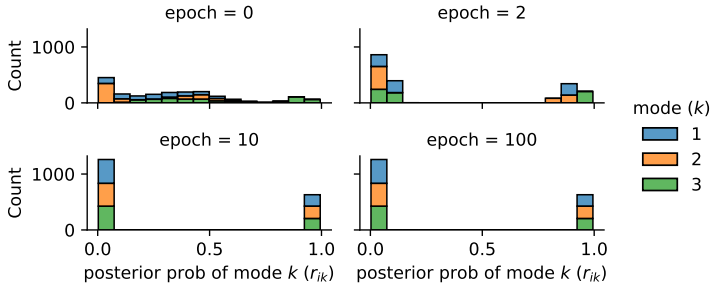


Figure 11: Posterior distribution of soft EM along training

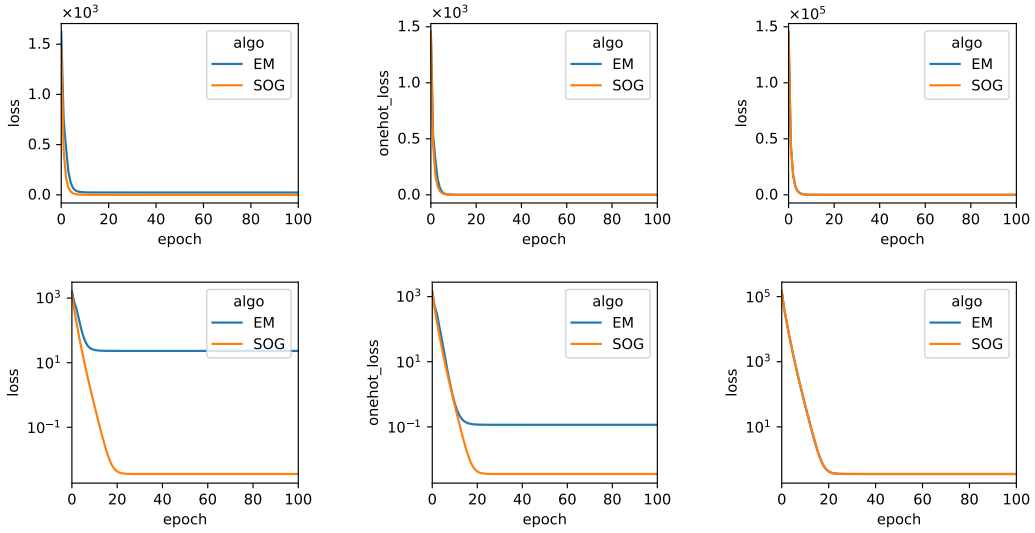
never attain purely one-hot posterior, which is shown as follows. In particular, the EM approach maximizes the expected likelihood of the data (where expectation is taken over all the latent codes), whereas the SOG maximizes the likelihood of the data conditioned at the code where it is maximum. Both of the consequences can be alleviated by decreasing the standard deviation σ assumed in the Gaussian likelihood. Indeed, when we decrease σ by two orders of magnitude from its original value 1 to 0.01, the training curve of EM and SOG match as shown in Figure 12c.

D Continuous Latent Variables

In this section, we analyse the behavior of the data log likelihood when SOG is applied for the continuous latent variables case. Consider the following identity (Chib, 1995), a direct result of the Bayes' rule:

$$\underbrace{\log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})}_{\substack{\text{data}\\ \text{log-likelihood}}} = \underbrace{\log p(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{x};\boldsymbol{\theta})}_{\text{SOG objective}} - \underbrace{\log \frac{p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y};\boldsymbol{\theta})}{p(\boldsymbol{z})}}_{\text{gap}};\ \forall \boldsymbol{z}. \tag{19}$$

This identity describes a gap between the objectives in SOG and the data log-likelihood. The rest of this section aims to show that under certain reasonable assumptions about the posterior distributions of the latent codes, the SOG algorithm increases the overall data log-likelihood function and reaches a maximum, before it starts overfitting, when the posterior distributions become too narrow. We also show that such overfitting can be avoided, and the data log-likelihood be kept high by using a limited number of latent code samples over which the minimum loss is calculated in the first step of SOG.



(a) $\sigma = 1$ for EM. Upper: linear-scale; lower: log-scale

(b) $\sigma = 1$ for EM. Posterior is converted to one-hot when calculating the loss for plotting. Upper: linear-scale; lower: log-scale

(c) $\sigma = 0.01$ for EM. Upper: linear-scale; lower: log-scale

Figure 12: The training curve of the EM algorithm vs that of SOG.

Algorithm 1 searches for a latent code $\boldsymbol{z}^*$ that maximizes the SOG term above. In Equation (19), the marginal data log-likelihood term is not a function of the latent variable. Therefore,

$$\boldsymbol{z}^* = \arg\max_{\boldsymbol{z}} \log p(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x}; \boldsymbol{\theta}) \tag{20a}$$

$$= \arg\max_{\boldsymbol{z}} \log \frac{p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}{p(\boldsymbol{z})}. \tag{20b}$$

We assume the model to be realized with a high-capacity neural network. Hence, we can assume an arbitrary form for the posterior distribution in Equation (20b), e.g. a Gaussian (Kingma & Welling, 2013):

$$p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y})). \tag{21}$$

For a Gaussian form of the prior and the posterior, the gap in Equation (20b) can be written in closed form:

$$\boldsymbol{z}^* = \arg\max_{\boldsymbol{z}} -\frac{1}{2}\left[(\boldsymbol{z}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{z}-\boldsymbol{\mu})^T\right] - \frac{1}{2}\log\det\boldsymbol{\Sigma} + \frac{1}{2}\boldsymbol{z}^T\boldsymbol{z} \tag{22a}$$

$$= \begin{cases} (\boldsymbol{I}-\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}, & \text{if } \boldsymbol{I}-\boldsymbol{\Sigma} \succ \boldsymbol{0} \\ \text{unbounded}, & \text{otherwise} \end{cases} \tag{22b}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ denote a shorthand for $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y})$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y})$. Since the model is trained to create mutual information between the data points and latent codes, the posterior distribution develops more certainty than the prior, that is $\boldsymbol{I} \succ \boldsymbol{\Sigma}$. Hence, Equation (22b) yields $\boldsymbol{z}^* = (\boldsymbol{I}-\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}$.

Next, we use the $\boldsymbol{z}^*$ derived above to obtain the gap term in closed form:

$$
\begin{aligned}
g(\boldsymbol{\mu}, \boldsymbol{\Sigma}) &\equiv \max_{\boldsymbol{z}} \log \frac{p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}{p(\boldsymbol{z})} && \text{(23a)} \\
&= -\frac{1}{2}\left[(\boldsymbol{z}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{z}-\boldsymbol{\mu})^T\right]\Big|_{\boldsymbol{z}=\boldsymbol{z}^*} - \frac{1}{2}\log\det\boldsymbol{\Sigma} && \text{(23b)} \\
&= \frac{1}{2}\left[\boldsymbol{z}^T(\boldsymbol{I}-\boldsymbol{\Sigma}^{-1})\boldsymbol{z} + 2\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{z} - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right]\Big|_{\boldsymbol{z}=\boldsymbol{z}^*} - \frac{1}{2}\log\det\boldsymbol{\Sigma} && \text{(23c)} \\
&= \frac{1}{2}(\boldsymbol{Q}^T\boldsymbol{\mu})^T(\boldsymbol{I}-\boldsymbol{\Lambda})^{-1}(\boldsymbol{Q}^T\boldsymbol{\mu}) - \frac{1}{2}\log\det\boldsymbol{\Sigma} && \text{(23d)} \\
&= \frac{1}{2}\boldsymbol{\mu}^T\left(\boldsymbol{Q}(\boldsymbol{I}-\boldsymbol{\Lambda})\boldsymbol{Q}^T\right)^{-1}\boldsymbol{\mu} - \frac{1}{2}\log\det\boldsymbol{\Sigma} && \text{(23e)} \\
&= \frac{1}{2}\boldsymbol{\mu}^T(\boldsymbol{I}-\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu} - \frac{1}{2}\log\det\boldsymbol{\Sigma} && \text{(23f)} \\
&\leq \|\boldsymbol{\mu}\|^2\sqrt{\operatorname{Tr}((\boldsymbol{I}-\boldsymbol{\Sigma})^{-2})} - \frac{1}{2}\log\det\boldsymbol{\Sigma} && \text{(23g)} \\
&= \|\boldsymbol{\mu}\|^2\sqrt{\operatorname{Tr}((\boldsymbol{I}-\boldsymbol{\Lambda})^{-2})} - \frac{1}{2}\log\det\boldsymbol{\Lambda} && \text{(23h)} \\
&\equiv h(\boldsymbol{\mu}, \boldsymbol{\Lambda}), && \text{(23i)}
\end{aligned}
$$

where in Equations (23d) and (23h), the covariance matrix is diagonalized as $\boldsymbol{\Sigma} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$. Also, Equation (23g) follows from the fact that $\forall \boldsymbol{A}, \boldsymbol{B} \succeq \boldsymbol{0};\ \operatorname{Tr}(\boldsymbol{AB}) \leq \sqrt{\operatorname{Tr}(\boldsymbol{A}^2)\operatorname{Tr}(\boldsymbol{B}^2)}$.

Now we more closely study the properties of the gap upper bound $h(\boldsymbol{\mu}, \boldsymbol{\Lambda})$, and show that the gap is not large under the following assumptions and arguments:

1. Since Algorithm 1 optimizes its objective for latent codes sampled from a Gaussian prior distribution, $\boldsymbol{\mu}$ remains bounded.
2. Along the initial training phase of the model, the SOG objective increases and the gap term in Equation (19) decreases, leading to a increased data log-likelihood.

   Our multi-modal generative model with latent variable has a fundamental assumption: The conditional distribution of the output variable $\boldsymbol{y}$ given the input variable $\boldsymbol{x}$, can be related to the latent variable $\boldsymbol{z}$, such that $\boldsymbol{y}|(\boldsymbol{x}, \boldsymbol{z})$ has less uncertainty compared to $\boldsymbol{y}|\boldsymbol{x}$. Quantitatively, this means $I(\boldsymbol{y}; \boldsymbol{z} \mid \boldsymbol{x}) = H(\boldsymbol{z}|\boldsymbol{x}) - H(\boldsymbol{z}|\boldsymbol{y}, \boldsymbol{x}) = H(\boldsymbol{z}) - H(\boldsymbol{z}|\boldsymbol{y}, \boldsymbol{x}) > 0$, where $I(\boldsymbol{y}; \boldsymbol{z} \mid \boldsymbol{x})$ is the mutual information between $\boldsymbol{y}$ and $\boldsymbol{z}$ conditioned on $\boldsymbol{x}$, $H(\boldsymbol{z})$ is the information entropy of $\boldsymbol{z}$, and $H(\boldsymbol{z}|\boldsymbol{y}, \boldsymbol{x})$ is the conditional entropy of $\boldsymbol{z}$ conditioned on $\boldsymbol{x}$ and $\boldsymbol{y}$.

   At the beginning of the training of the model, the mutual information $I(\boldsymbol{y}; \boldsymbol{z} \mid \boldsymbol{x})$ is zero because the model has no knowledge of the data, and no corresponding structure has been established in the latent space yet. Along the training, the mutual information increases. For Gaussian posterior $p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$, this directly leads to a lower determinant of the covariance matrix, which further leads to a decrease in $h(\boldsymbol{\mu}, \boldsymbol{\Lambda})$, as shown in Figure 13.
3. The Figure 13 shows that the gap becomes large when the posterior variance becomes too small. This happens when the model overfits and a small region in the latent space is over-specialized for certain data, hurting generalization ability.

   However, such over-specialization can hardly happen in SOG, because the randomized search of the latent code limits the "resolution" of the latent code choice and bounds the posterior variance.

E THE SELF-ORGANIZATION EFFECT

In this section, we discuss how the self-organization phenomenon, that is the formation of similar output data in different regions of the latent space, naturally emerges in Algorithm 1. In the following, we present three propositions demonstrating different aspects of the self-organization phenomenon in the latent space, under the assumption that the function $f(\cdot)$ is a sufficiently smooth function of its inputs $(\boldsymbol{x}, \boldsymbol{z})$, e.g. it is Lipschitz continuous:

1. *Continuity in the data space:* We discuss that for every data point, there is a neighborhood in the data space such that for the same latent code $\boldsymbol{z}$, the variations in the L_2-loss in Algorithm 1 over all points in the neighborhood is small.

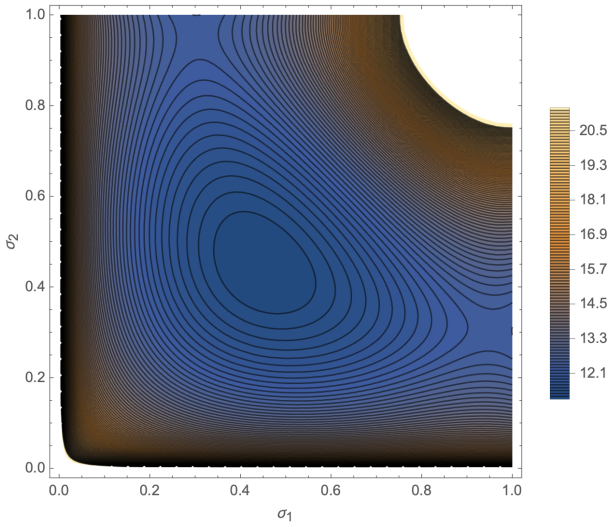


Figure 13: A contour plot demonstration of $h(\boldsymbol{\mu}, \boldsymbol{\Lambda})$ for two-dimensional case. The axes correspond to the diagonal elements of $\boldsymbol{\Lambda} = \text{diag}(\sigma_1^2, \sigma_2^2)$. In this example, $\|\boldsymbol{\mu}\| = 3$.

2. *Two sufficiently close data points will have nearby winning latent codes (or the same winning latent codes in the discrete case):* We show that if for a data point, a particular latent code results in a significantly better L_2-loss than another latent code, then the same will also hold within a neighborhood of that data point. Thus, if $\boldsymbol{z}$ is a winning code for a data point, then the points in its neighborhood will also have winning codes close to $\boldsymbol{z}$ (or the same winning code $\boldsymbol{z}$ in the discrete case).

3. *Network training can be a collaborative process:* We demonstrate that if we update the network parameters $\boldsymbol{\theta}$ to improve the L_2-loss for a particular data point (at a latent code z), then the same update will also collaboratively improve the loss for other data points within the neighborhood of the main data point (w.r.t. the same latent code z).

All three propositions provide insights on why the SOG training algorithm converges fast. In particular, in the mini-batch optimization in any implementation of the SOG algorithm, these propositions imply that the parameter updates for a sample batch of data points, reduce the loss for points neighboring to the sampled data points as well.

Formally, we shall make the assumption that the network f is a Lipschitz continuous function with the constant M. We also define the δ_x-δ_y-neighborhood of a data point $(\boldsymbol{x}, \boldsymbol{y})$ as $B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y) = \{(\boldsymbol{x}, \boldsymbol{y}) | \|\boldsymbol{x} - \boldsymbol{x}_1\| < \delta_x/M, \|\boldsymbol{y} - \boldsymbol{y}_1\| < \delta_y\}$. For later convenience, we define $\delta = \delta_x + \delta_y$. Finally, we denote $d(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}; \boldsymbol{\theta}) \coloneqq \|f(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y})\|$.

Proposition E.1. *For any data point* $(\boldsymbol{x}_1, \boldsymbol{y}_1)$, *if* $(\boldsymbol{x}_2, \boldsymbol{y}_2) \in B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y)$, *then* $|d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \theta) - d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}; \theta)| < \delta_x + \delta_y = \delta$

Proof. From $(\boldsymbol{x}_2, \boldsymbol{y}_2) \in B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y)$, follows that $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\| < \delta_x/M$, $\|\boldsymbol{y}_1 - \boldsymbol{y}_2\| < \delta_y$. Therefore:

$$
\begin{aligned}
| \, \|f(\boldsymbol{x}_1, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y}_1\| - \|f(\boldsymbol{x}_2, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y}_2\| \, | &\leq \|f(\boldsymbol{x}_1, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y}_1 - (f(\boldsymbol{x}_2, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y}_2)\| && \text{(24a)} \\
&\leq \|f(\boldsymbol{x}_1, \boldsymbol{z}; \boldsymbol{\theta}) - f(\boldsymbol{x}_2, \boldsymbol{z}; \boldsymbol{\theta})\| + \|\boldsymbol{y}_1 - \boldsymbol{y}_2\| && \text{(24b)} \\
&\leq M \, \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| + \|\boldsymbol{y}_1 - \boldsymbol{y}_2\| && \text{(24c)} \\
&< \delta_x + \delta_y = \delta. && \text{(24d)}
\end{aligned}
$$

where Equations (24a) and (24b) follow reverse triangle inequality and triangle inequality, respectively. Besides, Equation (24c) follows from the Lipschitz continuity assumption. □

Proposition E.2. *Let $(\boldsymbol{x}_1, \boldsymbol{y}_1)$ be a data point, and $\boldsymbol{z}_i, \boldsymbol{z}_j$ be two choices of the latent code. If $\dfrac{d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_i; \boldsymbol{\theta})}{d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j; \boldsymbol{\theta})} > C > 1$ for a constant C, and $d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j; \boldsymbol{\theta})$ is finite, then $\exists \delta_x, \delta_y$ such that*

$$\forall (\boldsymbol{x}_2, \boldsymbol{y}_2) \in B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y), \quad \frac{d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_i; \boldsymbol{\theta})}{d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_j; \boldsymbol{\theta})} > 1$$

Proof. $\forall \eta > 0$, $\exists \delta_x, \delta_y$ such that $\delta_x + \delta_y = \delta < \eta d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j; \boldsymbol{\theta})$. Then using proposition E.1, the following sequence of inequalities holds

$$\begin{aligned}
d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_i) &\geq d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_i) - \delta && \text{(25a)}\\
&> C d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) - \delta && \text{(25b)}\\
&> C d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) - \eta d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) && \text{(25c)}\\
&= (C - \eta) d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) && \text{(25d)}\\
&> (C - \eta)(d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_j) - \delta) && \text{(25e)}\\
&> (C - \eta)\left(1 - \frac{\eta}{1 - \eta}\right) d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_j) && \text{(25f)}
\end{aligned}$$

where the step from (25e) to (25f) is done by realizing the fact that

$$\eta d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_j) > \eta\big(d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) - \delta\big) > (1 - \eta)\delta$$

and therefore $\delta < \frac{\eta}{1-\eta} d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_j)$.

It can be easily verified that there always exists a small enough value of $\eta \to 0^+$ such that

$$(C - \eta)\left(1 - \frac{\eta}{1 - \eta}\right) > 1$$

and therefore

$$d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_i) > d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_j).$$

□

Proposition E.3. *Let $(\boldsymbol{x}_1, \boldsymbol{y}_1)$ be a data point and let $\boldsymbol{z}$ be a choice of the latent code, respectively. Also assume an improvement update of the network parameters $\boldsymbol{\theta}_1$ to $\boldsymbol{\theta}_2$ that ensures $d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \boldsymbol{\theta}_1) > d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \boldsymbol{\theta}_2)$.*

Then, there exists δ_x and δ_y such that

$$\forall (\boldsymbol{x}_2, \boldsymbol{y}_2) \in B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y), \;\; d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}; \boldsymbol{\theta}_2) < d^{\max}_{(\delta_x, \delta_y)}(\boldsymbol{\theta}_1),$$

where

$$d^{\max}_{(\delta_x, \delta_y)}(\boldsymbol{\theta}) = \max_{(\boldsymbol{x}, \boldsymbol{y}) \in B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y)} d(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}; \boldsymbol{\theta})$$

is the largest reconstruction distance for data points in the neighborhood of $(\boldsymbol{x}_1, \boldsymbol{y}_1)$

Proof. Let $a = d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \boldsymbol{\theta}_1) - d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \boldsymbol{\theta}_2) > 0$, and again $\delta = \delta_x + \delta_y$, then $\forall \delta_x, \delta_y$, and $\forall (\boldsymbol{x}_2, \boldsymbol{y}_2) \in B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y)$

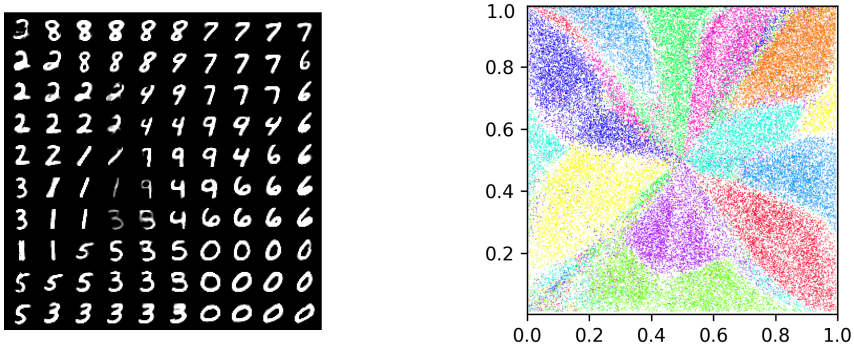
$$\begin{aligned}
d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}; \boldsymbol{\theta}_2) &\leq d^{\max}_{(\delta_x, \delta_y)}(\boldsymbol{\theta}_2) && \text{(26a)}\\
&\leq d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \boldsymbol{\theta}_2) + \delta && \text{(26b)}\\
&\leq d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \boldsymbol{\theta}_1) - a + \delta && \text{(26c)}\\
&\leq d^{\max}_{(\delta_x, \delta_y)}(\boldsymbol{\theta}_1) - a + \delta && \text{(26d)}
\end{aligned}$$

where Equation (26b) follows from Proposition E.1.

Since a is finite, $\exists \delta_x, \delta_y$ such that $\delta < a$, and hence

$$d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}; \boldsymbol{\theta}_2) < d^{\max}_{(\delta_x, \delta_y)}(\boldsymbol{\theta}_1).$$

□



(a) Synthesized MNIST digits **(b)** Embedding of MNIST test data in 2 dimensions

Figure 14: MNIST dataset. Output results of Algorithm 4 for the latent dimension of 2. (a) Synthesized images (b) Embedding of MNIST test data in 2 dimensions by the search mechanism of the SOG algorithm. Color coding represents ground truth digit labels. It is evident that despite the unsupervised training of the SOG algorithm, data samples corresponding to each digits are evenly organized across the latent space.

F Extension of Algorithm 1 to Large Latent Spaces

Algorithm 1 suffers from an exponential growth of the number of samples required, as the latent space dimension increases. To address this, one can consider a coordinate-wise search of the latent code $\boldsymbol{z}$, which results in Algorithm 4. This modification reduces the computational cost from exponential to linear.

Algorithm 4 Coordinate-Wise Search in SOG

Define: Loss function $\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \coloneqq ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2$, block size Δ
for $epoch = 1, 2, \ldots$ **do**
  for $iteration = 1, 2, \ldots$ **do**
    Sample a minibatch of N_{data} data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ from the dataset.
    for $i = 1, 2, \ldots, N_{\text{data}}$ **do**
      Initialize latent code $\boldsymbol{z}_i = \boldsymbol{0} \in \mathbb{R}^d$
      for $b = 1, 2, \ldots, (d/\Delta)$ **do**
        Initialize candidate latent codes $\boldsymbol{z}_j \coloneqq \boldsymbol{z}_i;\ j = 1, \ldots, N_z$.
        Replace the b'th Δ-element block in $\boldsymbol{z}_j$ with a sample from $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I}^{\Delta \times \Delta});\ j = 1, \ldots, N_z$.
        Calculate $\boldsymbol{z}_i \coloneqq \arg\min_{\boldsymbol{z}_j} \mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{z}_j, \boldsymbol{x}_i), \boldsymbol{y}_i)$.
      end for
    end for
    Calculate $\mathcal{L}_{\text{SOG}} = \sum_{i=1}^{N_{\text{data}}} \mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{z}_j, \boldsymbol{x}_i), \boldsymbol{y}_i)$ and its gradient w.r.t. $\boldsymbol{\theta}$.
    Update $f_{\boldsymbol{\theta}}$ by stochastic gradient descent on $\boldsymbol{\theta}$ to minimize $\mathcal{L}_{\text{SOG}}$.
  end for
end for

In Figures 14 to 16, we present visual results of Algorithm 1 demonstrating the capability of our method in fitting complex distributions. The axes in Figures 14 and 15 are the CDFs of the Gaussian latent codes for better visualization. We used the network architecture of the generator in Radford et al. (2015).

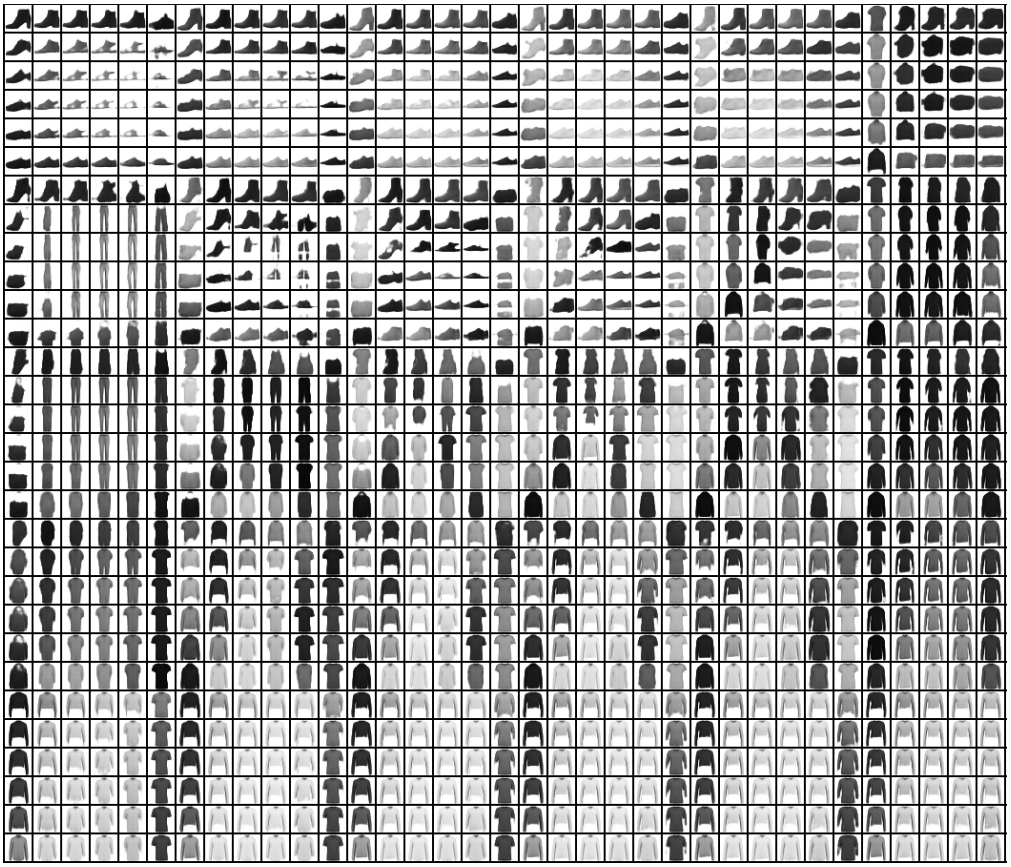


Figure 15: Fashion MNIST. Output results of Algorithm 4 for the latent dimension of 8. Each 6×6 block (or block of blocks, etc.) sweeps over one dimension of the latent space. Due to limited space, only part of the entire latent space is plotted. Full results are provided in the link (see the summplementary material zip file in the review stage). Different latent dimensions have smoothly captured semantically meaningful aspects of the data, i.e. type of clothing, color intensity, thickness, etc.

(a) CelebA ground-truth data **(b)** Reconstruction by SOG algorithm

Figure 16: CelebA. Output results of Algorithm 4 for the latent dimension of 16. (a) Ground truth samples. (b) Reconstructed samples synthesized by SOG. Note that the reconstructed images learned many of the salient features of the original faces, including the pose of the faces, as well as the hairstyle in most cases.