Title

Self-Organizing Generative Models for Diverse Imitation Learning

Author

Vahabpour, Arash

Publication Date

2022

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Self-Organizing Generative Models for Diverse Imitation Learning

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical and Computer Engineering

by

Arash Vahabpour

2022

ABSTRACT OF THE DISSERTATION

Self-Organizing Generative Models for
Diverse Imitation Learning

by

Arash Vahabpour
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2022
Professor Vwani P. Roychowdhury, Chair

The goal of AI is to allow computers to analyze and understand the world through algorithms. Generative models are a class of algorithms that can produce data with the same distribution as observed data are one of the most versatile tools towards creating such understanding. Amongst all possible models that could approximate any given data distribution, models that are constrained and hence forced to learn the essence of the way natural data is created are to be preferred. In this thesis, we propose a generative model that works based on the principle of sampling and searching over a low-dimensional latent representation for the data. We name it Self-Organizing Generative Model (SOG) due to its natural clustering of similar data points in their latent representation space, and its connection to the classical Self-Organizing Maps. We provide a theoretical analysis showing that this model adopts an Expectation-Maximization (EM) approach for maximizing the data likelihood. We demonstrate through various experiments that, unlike several existing models, the SOG model successfully learns all modes of data distribution. This property enables us to address imitation of diverse behaviors performed by a robot/agent in a Markov Division Process domain. More specifically, we focus on the task of imitation learning which aims at replicating

expert policy from demonstrations, without access to any reward function that might have motivated the expert. This task becomes particularly challenging when the expert exhibits a mixture of behaviors. To model variations in the expert policy, prior work has considered adding latent variables to existing imitation learning frameworks. Our experiments show that the existing works do not exhibit appropriate imitation of all modes. To tackle this problem, we incorporate our generative model, SOG, into behavior cloning (BC). Behavior cloning is a supervised method to imitate expert behavior. The generative process of SOG adds diversity to BC. The resulting model, SOG-BC, shows promise to accurately distinguish and imitate different modes of behavior. Another model for imitation learning is known as GAIL, which adopts an adversarial approach to imitation. GAIL considers long-term dynamics in its optimization process. Therefore, it alleviates the problem of compounding errors inherent in BC and is more robust to noise. To make the imitated policy of SOG-BC robust towards compounding errors at unseen states, we integrate it with GAIL. We show that our method significantly outperforms the state of the art across multiple experiments.

The dissertation of Arash Vahabpour is approved.

Jonathan Chau-Yan Kao

Arash Ali Amini

Lieven Vandenberghe

Vwani P. Roychowdhury, Committee Chair

University of California, Los Angeles

2022

To my parents . . .

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincerest thanks to my advisor, Professor Vwani Roychowdhury, for his immense support and mentorship. I will always cherish our countless hours of meetings, where I had the invaluable opportunity to enjoy his passionate discussions and keen insight. I would also like to express my gratitude to the members of my doctoral committee, Professor Jonathon Kao, Professor Arash Amini, and Professor Lieven Vandenberghe, for taking the time to serve on this committee and providing feedback on this work.

I want to also thank my very talented collaborators and academic friends for inspiring me with their knowledge and various forms of support through the years. I would like to name Tianyi Wang, Qiujing Lu, Hossein Razavi, Omid Poursaeed, Shadi Shahsavari, Mojtaba Sahraee Ardakan, Mohammad Kachuee, Tonmoy Monsoor, Pavan Holur, Manoj Reddy, Misagh Falahi, Ehsan Ebrahimzadeh, Behnam Shahbazi, and Omead Pooladzandi, among many others.

Last but not least, I would like to thank my parents and siblings for their immeasurable love and support. Words cannot express my gratitude for their unconditional love.

VITA

2010-2015 B.S. (Electrical Engineering) and Minor B.S. Degree (Math), Sharif University of Technology.

2015-2016 M.S. (Electrical Engineering), UCLA.

CHAPTER 1

Introduction

1.1 Overview

In the last two decades, applications of Artificial Intelligence (AI) have revolutionized people's daily lives. Many people around the world nowadays find free or low-cost access to various human-level AI products. Examples include but not limit to: text to speech, speech to text, image recognition, machine translation, chat bots, visual filters, augmented reality, and many more. Most of these successes owe to the regained popularity of neural networks, and the applications built around them. Neural networks endow researchers with much flexibility in enabling machines to replicate human intelligence in different domains of application. Such problems are often modelled as "regression", which involves learning the mapping between data point pairs. One can model such mappings with neural networks and optimize their parameters in standard ways. Another class of useful models for making inference about data are generative models. They are typically black-box tools for learning the data at a distribution level.

While many regression and generative modelling techniques have been widely developed and adopted, much of the existing body of work misses a context-based understanding of data. For instance, the traditional methods for object detection involve feeding the pixel-level information to a neural network and receive class labels at the output. However, such a naive paradigm lacks considering contextual clues that can potentially help in disambiguation of possible guesses. Let us consider the example of a low-resolution object crop. In the lack of context, even human eyes might not have a clear perception of the ground-truth reality

of the object. Nevertheless, contextual clues can massively help humans to overcome this type of ambiguity. Contextual clues are so helpful that they can be informative on their own, without the actual data. For instance, humans can often guess the class of an object by merely seeing the context pixels around it, even if the main object is masked.

Context recovery has been proven to be a main factor to improve the quality of many AI applications. For instance, in machine translation, the meaning of a "polysemous" word needs to be recovered from the words around it. In speech to text, the speakers accent and the topic of the conversation play a major role in converting the raw audio waves into language. Examples in computer vision are abundant, e.g. object detection which was discussed earlier. Such context recovery and adoption of the recovered context is typically pursued using generative models in the literature. In this thesis, we study the existing approaches and propose our own novel approach that overcomes the challenges existing in the literature. In the next section we view the problem of context-based data-driven prediction through the lens of multimodal synthesis.

1.2 Multimodal Regression

One of the main problems in statistical machine learning is regression. Regression aims to find the mapping between the value of an input variable $\boldsymbol{x} \in \mathbb{R}^m$ and a target variable $\boldsymbol{y} \in \mathbb{R}^n$. There exist a variety of well-known approaches for addressing this problem. For example, in the broad class of linear regression models, the input variables, or non-linear combinations of their fixed, are linearly transformed to produce the output variables. Other models include support vector regression, neural networks, random forests, and beyond. [3].

Training regression algorithms requires a training set $\mathcal{D}$, comprising of N input data points $\{\boldsymbol{x}_n\}$, where $n = 1, \ldots, N$ together with corresponding output variables $\{\boldsymbol{y}_n\}$. In order to formulate the problem of regression, we assume that the target variable $\boldsymbol{y}$ is related

to the input variable $\boldsymbol{x}$ through a fixed function f plus an additive noise ϵ

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon. \tag{1.1}$$

Under this model, $\boldsymbol{y}$ takes a Gaussian distribution with mean $f(\boldsymbol{x})$ and covariance $\sigma^2\boldsymbol{I}$. Regression algorithms assume that a parameterized function f_θ can approximate the function f. These algorithms are often characterized by finding parameters θ that make the data samples most probable under the assumed model. Such a framework is known as "maximum likelihood estimation" and is studied extensively in statistical machine learning [3]. Maximizing the likelihood of observed data under the model of Equation (1.1) can be written in the following form:

$$\max_\theta \mathbb{E}_{\mathcal{D}}[\log p(\boldsymbol{y}|\boldsymbol{x})]. \tag{1.2}$$

Since we assumed an additive Gaussian noise in Equation (1.1), we can rewrite Equation (1.2) as

$$\max_\theta \mathbb{E}_{\mathcal{D}}[-\frac{1}{\sigma^2}||f(\boldsymbol{x}) - \boldsymbol{y}||^2 + C]. \tag{1.3}$$

where C denotes the constant terms w.r.t parameters θ. Further simplification and setting the gradient w.r.t θ as zero we obtain

$$\nabla_\theta E_{\mathcal{D}}[||f_\theta(\boldsymbol{x}) - \boldsymbol{y}||^2] = 0. \tag{1.4}$$

Moving the gradient operation inside the expectation, we obtain

$$E_{\mathcal{D}}[\nabla_\theta f_\theta(\boldsymbol{x})^T(f_\theta(\boldsymbol{x}) - \boldsymbol{y})] = 0. \tag{1.5}$$

If we seperate the expctation w.r.t $\boldsymbol{x}$ and $\boldsymbol{y}$, we obtain the following

$$E_{\boldsymbol{x}}[E_{\boldsymbol{y}}[\nabla_\theta f_\theta(\boldsymbol{x})^T(f_\theta(\boldsymbol{x}) - \boldsymbol{y})]] = 0, \tag{1.6}$$

which yields

$$E_{\boldsymbol{x}}[\nabla_\theta f_\theta(\boldsymbol{x})^T E_{\boldsymbol{y}}[f_\theta(\boldsymbol{x}) - \boldsymbol{y}|\boldsymbol{x}]] = 0. \tag{1.7}$$

In the optimization problem of Equation (1.7), we can see that parameters θ, which make $f_\theta(\boldsymbol{x}) = E_{\boldsymbol{y}}[\boldsymbol{y}|\boldsymbol{x}]$, constitute an optimum. In other words, conventional regression formalism

can result in predictions that average across the distribution of the output variable $\boldsymbol{y}$ given the input variable $\boldsymbol{x}$. This observation gives rise to one of the core motivations behind this thesis. In particular, we are interested in the following two questions: (1) in what situations can a vanilla maximum likelihood framework be problematic? (2) How can one avoid such problems by modeling the variations in the target variable and learning all variations simultaneously? In the next subsection, we examine a case study where the averaging phenomenon of Equation (1.7) is evident.

The averaging phenomenon explained above can manifest itself in various application domains. For instance, in natural language processing, different words in the vocabulary are often modeled by "word vectors" [32, 34]. This traditional approach for modeling words incurs the drawback of averaging the meaning of "homonymous" or "polysemous" words. Therefore, recent language models, e.g. [10], adopt context-aware methods for modeling words.

"Image Super Resolution" is another domain where the mode mixing phenomenon poses problems. In this problem, the goal is to resize an image to a higher resolution and at the same time create more fine-grained details. This problem is inherently challenging and somewhat ill posed, because various valid corresponding images might be valid. Here, the "big data" paradigm no more avails because a vanilla regression pipeline can at best learn to produce an average of all possible higher, regardless of whether it is trained on a large dataset. This phenomenon was pointed out and addressed in [28]. In Figure 1.1 we observe that super-resolution of a sample patch through regression can result in blurry results due to averaging many possible outputs.

An additional motivating example is image colorization. Assume that we train a neural network to map grayscale copies or silhouettes of natural images, to their colorful counterpart. In this scenario, certain types of natural images might face ambiguities in their mapping. For instance, a grayscale image of a bird/flower/car can be mapped to various colors. Therefore, the trained model will converge to output an average of possible values.

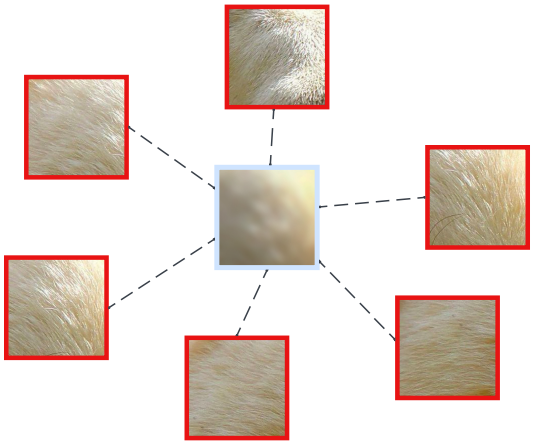


Figure 1.1: Mode mixing in image super-resolution. The patches with red boundaries represent ideally upsampled patches for a hypothetical lower resolution patch. A naive regression model can learn an average of all possible outcomes: the blue patch (blurry and undesirable).

However, in this case, the pixel-level average of different colors becomes gray or close to gray. This is not a desirable outcome and the problem of image colorization remains unsolved. In Chapter 2, we propose a network that enables multimodal regression of different colors. Training this architecture on a dataset of handbags and their corresponding edges, we obtain results illustrated in Figure 1.2. We repeated the same experiment on grayscale cars and observed separation in darkness level, as depicted in Figure 1.3.

The last example we consider for the mode mixing phenomenon is imitation of sequential behaviors in a reinforcement learning context. The pathology in this setting and our approach to solving this problem are presented in Chapter 5.

In a nutshell, learning to map input data to multimodal output values, if tackled with the traditional regression paradigm, can potentially suffer mixing among possible outputs. Through this thesis, we introduce a generative model that can reliably recover different modes of data. Then we showcase the adaptation of this model to different domains of application. In particular, in Chapter 5, we demonstrate how our model enables imitation of diverse behaviors in sequential decision-making.

Figure 1.2: Result of our multimodal image-to-image translation framework on a dataset of handbags. We notice a clear separation in colors

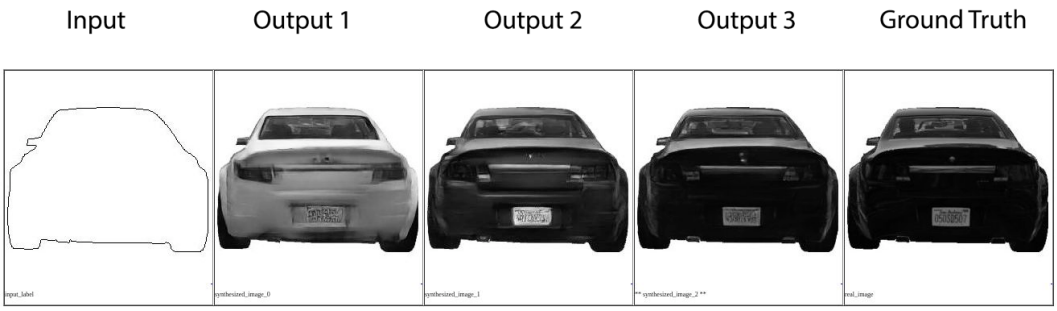


Figure 1.3: Result of our multimodal image-to-image translation framework on a dataset of cars. The branches get separated based on the darkness level

1.3 Problem Statement and Contributions

In this thesis, we take a close look at the important role of contextual features in making predictions. Inspired by humans' adoption of context in their cognitive tasks, we propose that in the presence of multiple modes, machines need to create an understanding of the conditional context based on which they can make their predictions. Typically such a contextual analysis is pursued via generative models in AI. We first study the theory and applications of the existing approaches in several application domains. Then, we propose a novel model which successfully overcomes some of the challenges inherent in the existing methods. Our proposed model is named Self-Organizing Generative Model (SOG), in recognition of the traditional Self-Organizing Maps, and the fact that similar data points cluster together in the latent space. We spend several sections studying the theory of this model. For example, we demonstrate that our model is closely connected to Expectation-Maximization (EM) for maximizing the likelihood of data. Therefore, our model can provably learn the distribution of any arbitrary dataset given mild assumptions. Furthermore, we show that Laplace Approximation for Integrals can also prove the connection of our model with Maximum Likelihood Estimation. We demonstrate the self-organization phenomenon discussed earlier theoretically under the assumption of "Lipchitzness" of the predicting function. Lastly, we propose that in a high-dimensional data setting, a greedy coordinate-wise search over different axes helps in breaking the time complexity, and reduces the cost of sampling down to linear. Another part of our contribution involves application of the SOG model in imitation learning, where an experts' sequential decision making has to be imitated. We discuss that when there exist multiple modes of behaviors, naive methods cannot learn all the behaviors at once, in fact none might be mimicked appropriately. We propose a method for applying the SOG model in distinguishing and learning every individual mode of behavior simultaneously. We benchmark this model with various simulated experiments and demonstrate that our model significantly outperforms the state of the art. In the next section, we summarize

the organization of the rest of this thesis.

1.4 Outline of the Thesis

This thesis is organized as follows. In Chapter 2 we discuss a simple experiment for conditional image synthesis, where the synthesized images need to get disentangled into several modalities. In Chapter 3, we review the existing literature in generative models, and also the main concepts in reinforcement learning and imitation learning. In Chapter 4, we propose the Self-Organizing Generative Models (SOG) and derive theory for it. In Chapter 5, we show how this model enables imitation of diverse behaviors. Lastly, in Chapter 6 we we summarize our contributions and discuss some of the avenues to extend the work of this thesis.

CHAPTER 2

Multimodality in Image Synthesis

In this section, we study a motivating example for multimodal regression. In particular, we look into a method for synthesizing high-resolution photo-realistic images from semantic label maps, using convolutional neural networks and adversarial training. The goal of the study in this section has been to improve the visual quality of individual objects in complex synthesized images. We show that in order to synthesize high-quality images with diverse colors and shapes, multimodal regression paradigm should be taken into account. We then explain how we can address this problem by generating the geometry of objects ahead of time. We perform simultaneous edge and image generation of objects to enhance the crispness of synthesis. Once the individual objects are generated properly, we feed a guiding map of the synthesized objects as a geometry prior. This additional input results in more realistic results in complex scenes. We also resolve the issue of ambiguity in the correspondence of inputs to outputs via a novel multimodal regression module. Finally, by bringing in more data of objects from similar datasets, we overcome the scarcity of samples in certain modes of data. A qualitative assessment of our results suggests that our proposed method outperforms state-of-the-art baselines in terms of crispness and realism.

Convolutional networks are an important ingredient of modern neural networks. At each layer, a set of learned matched filters are applied to the input channels according to the following equation.

$$I_{\text{out}}\big(i,j\big) = \sum_{k_1=1}^{k}\sum_{k_2=1}^{k}\sum_{d=1}^{\delta} I_{\text{in}}\big(i-k_1, j-k_2, d\big)\cdot K\big(k_1,k_2,d\big) \tag{2.1}$$

The resulting response maps for each filter are referred to as output channels. After applying

a non-linearity, these channels are concatenated and fed to the next layer. The same process can be repeated for several layers and form a "convolutional neural network".

Convolutional networks can be used as a differentiable classification or regression module. They can also be deployed to map latent codes to higher dimensional synthetic images. This can be achieved by transposing the matrix representation associated with the linear operation of convolution. In particular, if a linear mapping W maps an n-dimensional vector to an m-dimensional one, one can consider a reverse transformation from m dimensions to n dimensions by transposing the transformation matrix. In particular:

$$x^{1\times m} \xrightarrow{W^{m\times n}} y^{1\times n} \quad \Rightarrow \quad y^{1\times n} \xrightarrow{\left(W^{m\times n}\right)^T} x^{1\times m}$$

This operation is often known as "deconvolution" or "transposed convolution". Deconvolution layers can be stacked to produce images from lower dimensional data [37].

A number of problems in computer graphics and computer vision can be viewed as "translating" an input image to a corresponding output image. Some examples of image-to-image translation include: synthesizing photos from semantic label maps, synthesizing object images from their edges, image colorization, and semantic segmentation, among other tasks. U-Net [40] is an effective neural network architecture for a variety of such image translation tasks. The structure of U-Net comprises a typical convolutional encoder/decoder architecture that additionally adopts skip connections bypassing geometrical information to the decoding stage. In the absence of such bypass connections, the geometrical information would be lost in the encoding bottleneck of the encoder/decoder network. In most image translation tasks, geometry needs to be preserved between the input and the output; thus the U-Net architecture is a popular choice. A schematic of the architecture of U-Net is illustrated in Figure 2.1.

Generative adversarial networks (GANs) are models that learn a mapping from a random input vector z to an output y, such that random samples y resemble actual data in a dataset. We study them in more detail in Section 3.1.1. Conditional GANs are a variant of GANs

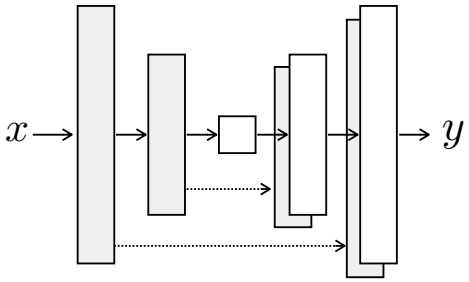


Figure 2.1: Schematic of U-Net

that can learn a mapping from an input x, and a random noise vector z, to an output y. The generator G is adversarially trained to produce outputs that not only correspond to the input but also look realistic. The objective of a conditional GAN be expressed as

$$\mathcal{L}_{cGAN} = \log D(x, y) + \log D(x, G(x, z)). \tag{2.2}$$

Conditional GANs can be used to enhance the synthesis quality in image translation tasks. One example of this application is Pix2Pix [20], where the authors deploy a U-Net together with an adversarial loss to perform photorealistic image synthesis. Pix2Pix adopts the following loss:

$$\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{2.3}$$

where $\mathcal{L}_{L1}$ measures the pixel-level difference between the generated image and the ground truth image.

A higher-resolution extension of Pix2Pix is proposed in Pix2PixHD [50]. In this work, multiple discriminators at different depths of decoding are used to add more fine-grained loss terms. Moreover, the pixel-level L_1 loss is replaced by differences in higher-level features. These features are extracted by the discriminators (feature matching loss), as well as by a pretrained object detection network known as VGG-Net (perceptual loss)[22]. VGG loss is defined as:

$$\mathcal{L}_{VGG} = \sum_{i=1}^{N_F} \frac{1}{M_{F,i}} ||F^{(i)}(y) - F^{(i)}(\hat{y})||_1 \tag{2.4}$$

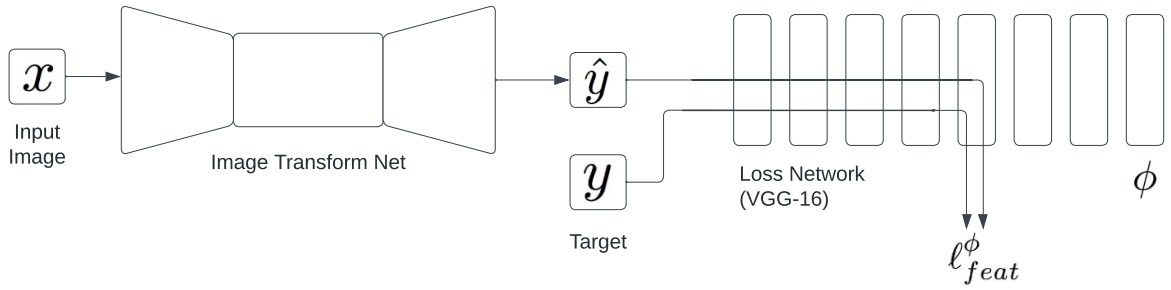


Figure 2.2: Schematic of VGG loss calculation

where N_F denotes the number of layers of the VGG network, M_F, i denotes the number of activations in its i^{th} layer, and $F^{(i)}$ denotes the feature map at its i^{th} layer. A schematic of the VGG loss function is illustrated in Figure 2.2. The feature matching loss is defined also defined in a similar way, except that the features are extracted by the problem-specific discriminator:

$$\mathcal{L}_{FM} = \sum_{i=1}^{N_D} \frac{1}{M_{D,i}} ||D^{(i)}(y) - D^{(i)}(\hat{y})||_1 \tag{2.5}$$

The overall loss function of Pix2PixHD is the following:

$$\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda_1 \mathcal{L}_{VGG}(G) + \lambda_2 \mathcal{L}_{FM}(G, D) \tag{2.6}$$

2.1 Missing Information in Image Translation

In this section, we study what can be improved in the existing image-to-image translation baselines, in particular Pix2PixHD. In Sections 2.3, 2.4, we address these issues and provide solutions, which will be the main tasks of this experiment.

Geometrical Priors

Literature shows that image translation can be improved by auxiliary geometrical information in the form of edges. For example, [50] improved the synthesis quality of city images

from semantic label map by providing the boundaries between objects of the same class. In [33], blurry artifacts of image inpainting are improved by predicting the inner edges before image completion. In [6], the human pose is translated by providing the target's skeleton to the U-Net architecture.

As a preliminary experiment, we applied a modified Pix2PixHD pipeline to the Cityscapes dataset [9] to translate label maps to images. In particular, we fed an extra input channel comprising of Canny edges [5] of the ground truth images. As illustrated in Figure 2.3, we observe that the output quality improves significantly, especially for objects that have complex part-based semantics (e.g., cars, buildings, etc.). However, we should point out that in this method, through the edges, part of the ground truth information is leaked into the input. Therefore, it is not a correct comparison to contrast the results of this experiment with those of the original Pix2PixHD. Nevertheless, our observation unveils the potential for quality improvement by supplying geometrical information.

(a) Ground truth edges

(b) Result of Pix2PixHD after augmenting the input with ground-truth edges

Figure 2.3: An experiment in which the ground-truth edges are fed as an additional input to the Pix2PixHD pipeline

Observing the results of the original Pix2PixHD network and loss function during training, we notice that the pipeline is able to memorize the translation of training data accurately. We attribute this to a large number of weights and use of perceptual loss [22]. At test time, however, we identify that synthesis quality for object classes that have rich part-based semantics [7, 14] is relatively poor compared to simpler objects such as trees and skies that solely comprise textures.

Scarcity of Data

Another challenge that Pix2PixHD faces is scarcity of data under particular conditions. For instance, we might have very few examples of cars pointing at particular angles. We resolve this issue by augmenting data from [39]. We will elaborate on this further in Section 2.3.

2.2 Modality Mixing

We identify the mode mixing phenomenon described in Chapter 1 a major drawback of the existing approaches. In image translation, this phenomenon results in averaging and blurring. In [28], it was pointed out that image super-resolution with mean squared error loss function results in blurring unless more sophisticated machinery is exploited. This argument applies to many other loss functions as well, where the optimizer tries to learn a hybrid of possible outputs, such that less amount of loss is incurred with different possible outputs. For instance, in our image translation task, we observe that for complex object classes with multiple modes, e.g., cars and buildings, the results look like a hybrid of different modes. In particular, in Figure 2.4b the vanilla model has synthesized cars whose appearance resembles views from multiple angles. Also, most cars and buildings look gray and colorless, mainly because gray is the mean pixel value over colors and incurs less VGG loss when both dark and light predictions are plausible.

(a) Groundtruth

(b) Result of the original Pix2PixHD

Figure 2.4: A challenging example for Pix2PixHD. The resulting image is relatively colorless, and the synthesized car has a hybrid of two angles.

In a nutshell, whenever in a regression task the input does not provide sufficient information about the output, the predictions average across all plausible outputs. Therefore, the synthesized images look like a hybrid of all modes and not similar to any particular mode.

In [52], the authors avoid such issues by providing sufficient statistics. In our experiment, we account for the insufficiency of the input information in a novel way which will be explained in Section 2.4

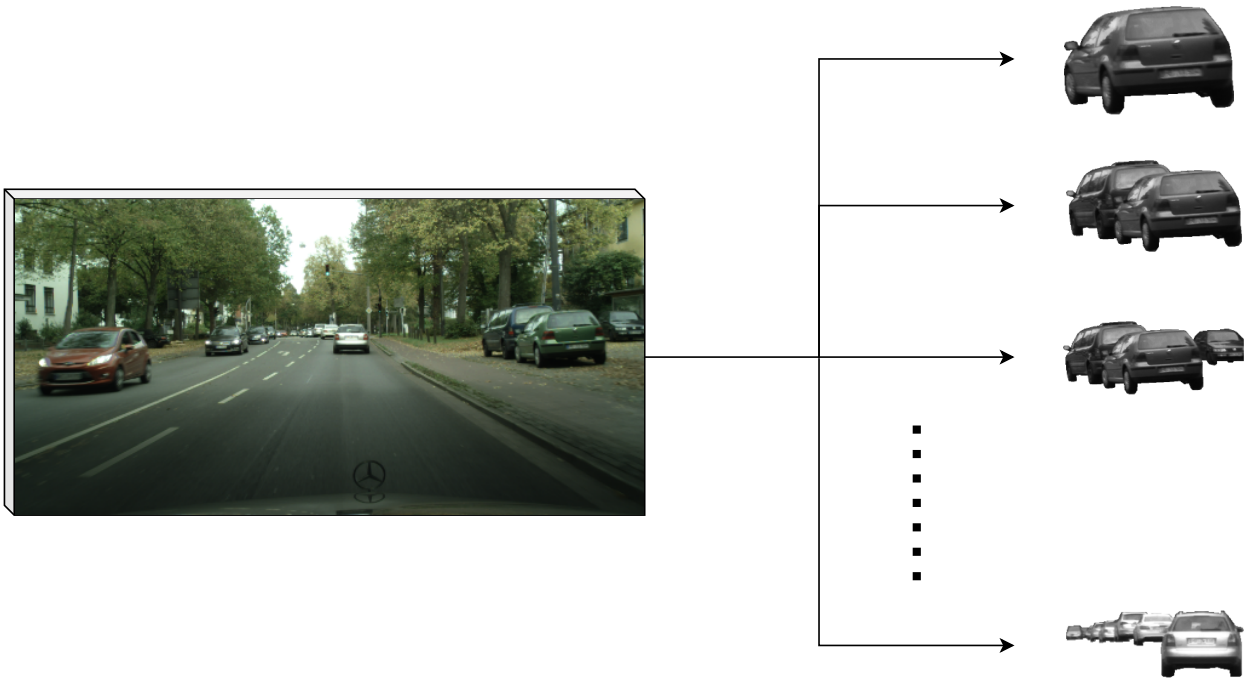


Figure 2.5: Stage 1 of the proposed refinement network

2.3 Multi-Stage Refinement for High-Quality Image Synthesis

The main goal of this experiment is to map semantic label maps of city images [9] by augmenting data from [39] and processing individual objects ahead of time. In this subsection, we discuss different stages of our pipeline.

Stage 1: Creating Dataset for Different Object Classes

There are 35 object classes in the Cityscapes dataset. For each object class, we create a dataset. Let us take the example of cars. We take out cars from all images and resize them to a fixed size. To augment with more data, we take the stacks and individual examples as separate data points. We also augment data by flipping data samples. We discard very small cars to avoid noise. This stage is illustrated in Figure 2.5.

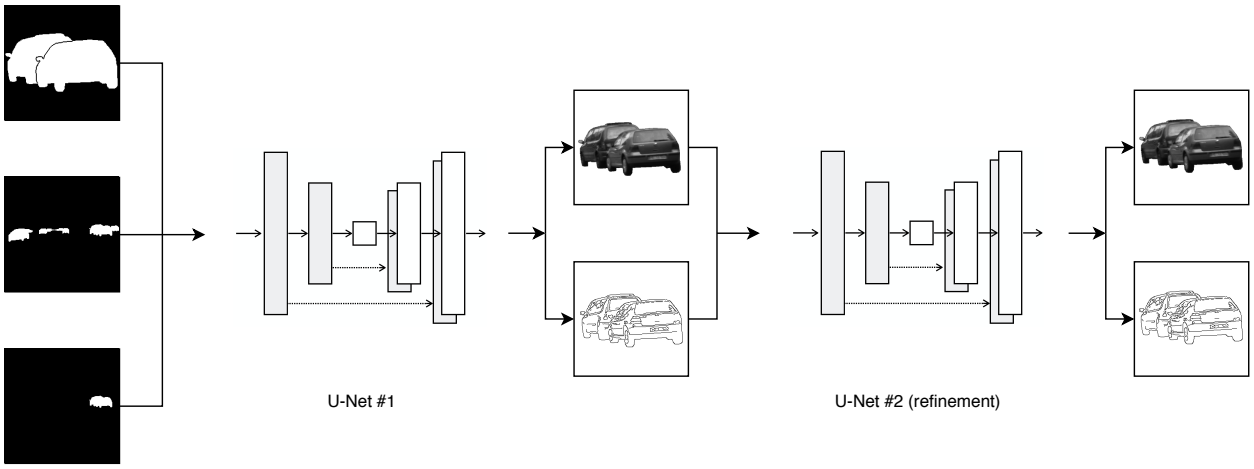


Figure 2.6: Stage 2 of the proposed refinement network

Stage 2: Generating Objects and Refinement

Once the dataset for an object class is created, we train a model to generate the pixel-level appearance of the objects given their input masks. As we discussed in Section 2.1, generating salient edges improves the quality of the overall image synthesis significantly. Without the inner edges, the geometry of the synthesized object becomes mixed or blurred. However, one should not feed ground truth inner edges as input because they are not available at test time. We propose an architecture that enforces distinctiveness and sharpness of object parts, while not requiring ground-truth edges as an input. Our proposed network architecture is illustrated in Figure 2.6. We consider two separate discriminator networks for the edges and the images.

The idea is that the optimization objectives for output edges and output images are different. However, since all prior activations are shared between the output edges and images, a correspondence holds. As a result, the perceptual loss of the output images helps output edges to make spatial sense. Moreover, generated edges guide the network to correspondingly synthesize abrupt color changes in the output images. In other words, the network learns to account for the inherent correspondence that exists between real images and their edges.

Figure 2.7: Pre-made map of cars provided to the PixPixHD pipeline

Therefore, the seemingly independent optimizations of the output edges and images boost one another. Lastly, the second part of this architecture is aimed to refine the results of the first part and generate even sharper and more realistic objects.

Step 3: Feeding Synthesized Edge Maps to Pix2PixHD

By the means discussed in the previous stage, we can synthesize objects without mode mixing described in Chapter 1. We then resize and stitch the synthesized objects together back to their original size and location on a white canvas. Finally, we feed the resulting object map as an additional input channel, as shown in Figure 2.7.

2.4 Multimodal Image-to-Image Translation

We pointed out the issue of modality mixing in Section 2.2. To address that, we propose a multimodal image-to-image translation architecture as illustrated in Figure 2.8. This model generalizes the architecture of U-Net to multiple branches. In particular, it encodes the input in a shared way among different modes. However, it decodes the encoded image along separate pathways, which we call branches, each yielding one mode of the output. We define

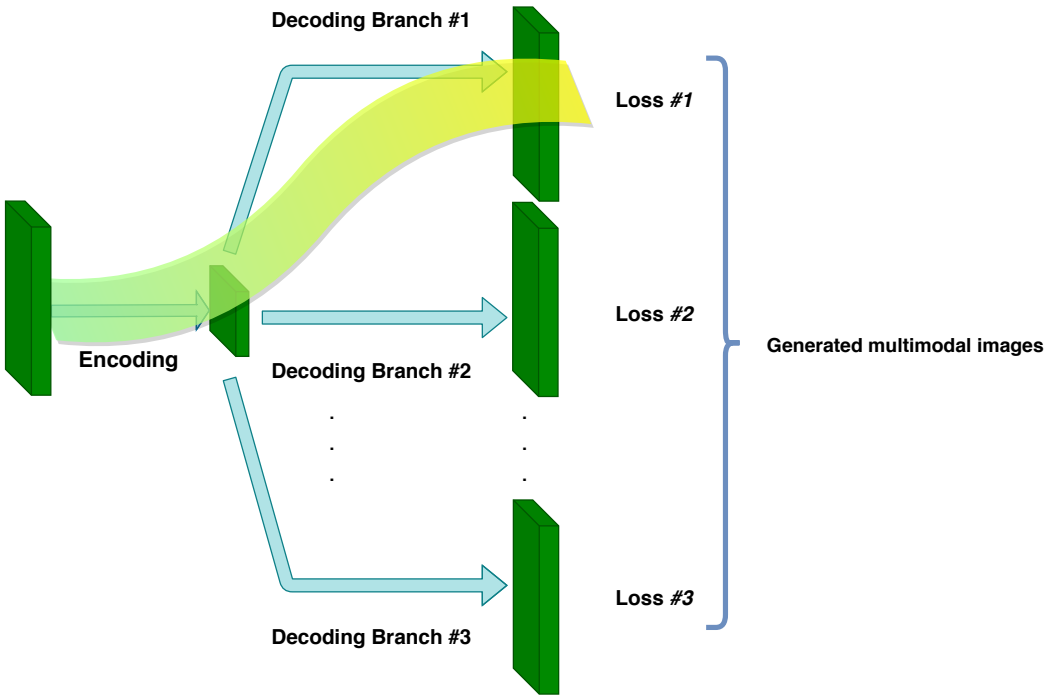


Figure 2.8: Multimodal image to image translation

the objective function as

$$f(x) = \min\{f_1(x), \ldots, f_N(x)\} \tag{2.7}$$

where $f_i(.)$ denotes the i^{th} branch. The gradient is thus derived as follows

$$\frac{\partial}{\partial W} f(x) = \begin{cases} \frac{\partial}{\partial W} f_1(x) & \text{if } f_1(x) = \min\{f_1(x), \ldots, f_N(x)\} \\ \quad \vdots & \\ \frac{\partial}{\partial W} f_N(x) & \text{if } f_N(x) = \min\{f_1(x), \ldots, f_N(x)\} \end{cases}. \tag{2.8}$$

Optimization of this objective creates dynamics by which each branch produces a specific modes of output. Initially, the weights of different branches are randomly initialized. Each branch by randomness has slight initial advantages towards predicting the output in a certain mode. Branches winning the quality competitions get trained and reinforced for their specialized modality. Through time, different branches get separated out, each producing data in a particular mode.

The criterion according to which the branches specialize directly depends on the mechanism by which the winning branch gets selected. For example, if the winning branches are selected to minimize the L2 loss between the generated image and ground truth image, then the branches specialize based on pixel values, i.e., colors. We conjecture that if we select winning branches according to a loss involving higher-level semantics, e.g., VGG loss, then the branches separate according to different "geometries".

In our experiments, the goal is to discover multimodality aspects beyond color and pixel-level features. We aim to generate different geometries of objects given an object boundary as input. We trained our proposed refinement network on cars of the Cityscapes dataset [9]. We fed the resulting car maps as an additional channel to the Pix2PixHD pipeline and received a significant boost in the synthesis quality. The overall results of Cityscapes scenes are illustrated in Section 2.4.

2.5 Further Thoughts

The framework in this experiment can be further improved by adding all object classes into the refinement network pipeline. Another opportunity to improve our existing results is to step beyond pixel-level separation and generate objects of different inner geometry given an input contour. For example, one can generate cars of multiple brands given a contour of a sedan car. Towards generating multiple geometries, one can use richer loss functions for branch specialization. One such candidate loss function is the VGG loss described earlier. The visual fidelity of the synthesized images can be quantified either through human surveys or automated methods.

(a) Sample image synthesized with the refinement network pipeline

(b) Sample image synthesized with the original Pix2PixHD

Figure 2.9: Comparison of (a) vanilla Pix2PixHD framework vs (b) our framework. The cars have become much clearer and have less artifacts.

CHAPTER 3

Background

3.1 Generative Models

Generative models are a class of machine learning models that are trained to represent probability distributions over a particular variable. Some of such models explain the distribution explicitly, whereas some others only form operations to implicitly get a sense of the distribution through sampling from the learned model. In the past few years, the community has become particularly interested in such models. Some classical models include Gaussian Mixture Models (GMM), Hidden Markov Models (HMM), and Boltzmann Machines. More recent algorithms include Normalizing Flows (NF), Variational Autoencoders (VAE), and Generative Adversarial Networks (GAN), which will be covered in this chapter. These models have close connections with the work proposed in this thesis. Therefore, we devote the rest of this section to explaining these models.

3.1.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are composed of two components, a generator network, and a discriminator network. The generator creates samples that are intended to come from the same distribution as the training data. The discriminator examines samples to determine whether they are real or fake using supervised learning techniques.

In the first stage of the generator network, a random vector is fed to a decoding module to produce a fake signal. An example of decoding modules is deconvolution layers, which

map low dimensional inputs to high dimensional tensors.

The signals generated by the generator, along with real data, are then passed to the discriminator network. The discriminator is a classifier, which can be a stack of convolutional and fully connected layers, mapping the data to real/fake labels.

The cost function of the discriminator is designed to distinguish samples x of real data from samples generated by feeding random vectors z to the generator:

$$J^{(D)}(\theta(D), \theta(G)) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log\left(1 - D(G(\mathbf{z}))\right) \tag{3.1}$$

The generator aims to increase the error of the discriminator, so its loss function is:

$$J^{(G)}(\theta(D), \theta(G)) = \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log\left(1 - D(G(\mathbf{z}))\right)$$

The training process consists of simultaneous stochastic gradient steps. On each step, two minibatches are sampled: a minibatch of x values from the dataset and a minibatch of z values drawn from the model's prior over latent variables. Then two gradient steps are made simultaneously: one updating $\theta(D)$ to reduce $J(D)$ and one updating $\theta(G)$ to reduce $J(G)$.

Global Optimality of $p_g = p_{\text{data}}$

The promise of GANs is that in the equilibrium, $p_g = p_{\text{data}}$, where p_g and p_{data} denote the generated and the data distributions, respectively. That is, the learned distribution matches the data distribution. For the proof, we refer the readers to the original paper.

3.1.1.1 Mode Collapse

One common caveat with this model is that in its vanilla form, the optimization procedure tends to converge to local optima where only parts of the data distribution are generated, whereas others get missed. This phenomenon is known as mode collapse and has been addressed by community in specific domains of application. An example of this phenomenon on MNIST digits is illustrated in Figure 3.1.

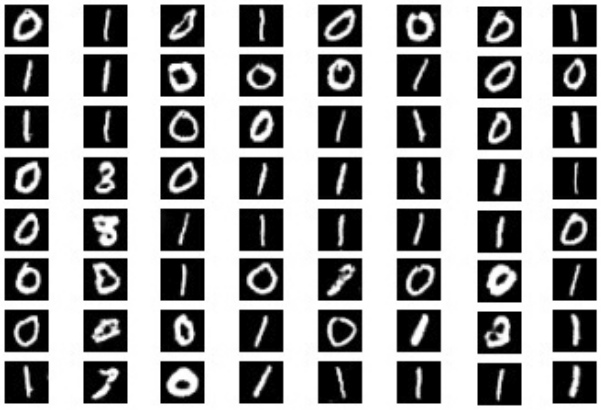


Figure 3.1: Mode collapse in vanilla GAN upon generating handwritten digits. Only certain types of digits are learned to be reproduced

3.1.2 Normalizing Flows

Flow models constitute another class of popular models that exploit maximum likelihood optimization. In order to introduce them, we first spend a few paragraphs reviewing the necessary background on linear algebra.

Jacobian Matrix

Given a function mapping from an n-dimensional input $\boldsymbol{x}$ to an m-dimensional output vector, $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, the matrix of partial derivatives $\frac{\partial f_i}{\partial \boldsymbol{x}_j}$ is called the "Jacobian matrix" and is defined as follows:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{3.2}$$

The "Leibniz formula" for the determinant of a matrix A can be defined as follows

$$\det(A) = \sum_{\sigma \in S_n} \left(\operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i} \right), \tag{3.3}$$

where S_n is the set of all $n!$ permutations σ over the rows of matrix A, and σ_i denotes the location of i'th row under permutation σ. Also, sgn denotes the sign of a permutation, which is uniquely defined as how many times the rows of the original matrix have to swap to get to the permuted form.

Change of Variables

Given a random variable sampled from a prior distribution $\boldsymbol{z} \sim \pi(\boldsymbol{z})$, the distribution of a variable $\boldsymbol{x} = f(\boldsymbol{z})$ is derived as follows

$$p(\boldsymbol{x}) = \pi(\boldsymbol{z}) \left| \det \frac{d\boldsymbol{z}}{d\boldsymbol{x}} \right| = \pi(f^{-1}(\boldsymbol{x})) \left| \det \frac{df^{-1}}{d\boldsymbol{x}} \right| \quad . \tag{3.4}$$

Transformations in Normalizing Flows

Normalizing flows models process latent variables by feeding a latent variable to an invertible function, hoping that the space of feasible functions is versatile enough to produce any desired distribution. The model is fit using a maximum likelihood objective. A latent variable $\boldsymbol{z}_0$ is processed through a cascade of invertible functions f_i:

$$\begin{aligned} \boldsymbol{z}_{i-1} &\sim p_{i-1}(\boldsymbol{z}_{i-1}) \\ \boldsymbol{z}_i = f_i(\boldsymbol{z}_{i-1});\ \boldsymbol{z}_{i-1} &= f_i^{-1}(\boldsymbol{z}_i) \\ p_i(\boldsymbol{z}_i) = p_{i-1}(f_i^{-1}(\boldsymbol{z}_i)) &\left| \det \frac{df_i^{-1}}{d\boldsymbol{z}_i} \right| . \end{aligned} \tag{3.5}$$

Given such a chain of density functions and recalling the derivative of an inverse function, we obtain the following expression for the log-likelihood of the data

$$\begin{aligned} \boldsymbol{x} = \boldsymbol{z}_K &= f_K \circ f_{K-1} \circ \cdots \circ f_1(\boldsymbol{z}_0) \\ \log p(\boldsymbol{x}) = \log \pi_K(\boldsymbol{z}_K) &= \log \pi_{K-1}(\boldsymbol{z}_{K-1}) - \log \left| \det \frac{df_K}{d\boldsymbol{z}_{K-1}} \right| \\ &= \log \pi_{K-2}(\boldsymbol{z}_{K-2}) - \log \left| \det \frac{df_{K-1}}{d\boldsymbol{z}_{K-2}} \right| - \log \left| \det \frac{df_K}{d\boldsymbol{z}_{K-1}} \right| \\ &= \dots \\ &= \log \pi_0(\boldsymbol{z}_0) - \sum_{i=1}^{K} \log \left| \det \frac{df_i}{d\boldsymbol{z}_{i-1}} \right| \end{aligned} \tag{3.6}$$

RealNVP

The RealNVP [11] model stacks a set of bijections known as affine coupling layers. Assume the input has two parts

- The first d dimensions remain unchanged.
- The second part, dimensions $d+1$ to D, are transformed by an affine (scaling and shifting) transformation whose weights are a function of the first part.

Concretely

$$\begin{aligned} \boldsymbol{y}_{1:d} &= \boldsymbol{x}_{1:d} \\ \boldsymbol{y}_{d+1:D} &= \boldsymbol{x}_{d+1:D} \odot \exp(s(\boldsymbol{x}_{1:d})) + t(\boldsymbol{x}_{1:d}), \end{aligned} \tag{3.7}$$

where $s(\cdot)$ and $t(\cdot)$ are mappings $\mathbb{R}^d \mapsto \mathbb{R}^{D-d}$. We realize that such a transformation is invertible

$$\begin{aligned} \boldsymbol{x}_{1:d} &= \boldsymbol{y}_{1:d} \\ \boldsymbol{x}_{d+1:D} &= (\boldsymbol{y}_{d+1:D} - t(\boldsymbol{y}_{1:d})) \odot \exp(-s(\boldsymbol{y}_{1:d})) \end{aligned} \tag{3.8}$$

and that its Jacobian has a closed form:

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & \mathbf{0}_{d \times (D-d)} \\ \frac{\partial \boldsymbol{y}_{d+1:D}}{\partial \boldsymbol{x}_{1:d}} & \operatorname{diag}(\exp(s(\boldsymbol{x}_{1:d}))) \end{bmatrix} \tag{3.9}$$

Since this matrix is triangular, its determinant is the product of the diagonal terms

$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp(s(\boldsymbol{x}_{1:d}))_j = \exp(\sum_{j=1}^{D-d} s(\boldsymbol{x}_{1:d})_j) \tag{3.10}$$

We notice that each layer of this transformation leaves some of the dimensions unchanged. To ensure that all dimensions are transformed, in RealNVP different dimensions are permuted from one layer to the next.

NICE

In NICE the transformation does not have the scaling term

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} + m(\mathbf{x}_{1:d}) \end{aligned} \tag{3.11}$$

Glow

The Glow model [25] simplifies the reverse permutation operation with an invertible 1×1 convolution. An illustration of this model is presented in Figure 3.2.

3.1.3 Variational Autoencoders

Variational Autoencoders (VAE) were proposed and studied in [24, 12]. The model assumes a prior distribution $p(\boldsymbol{z})$ over a latent variable $\boldsymbol{z}$. Then it transforms this prior distribution through a parameterized function $f(\boldsymbol{z}; \theta)$. In order to match a learned distribution with the actual data distribution, one can minimize the KL divergence between the two. Derivation below shows that this can be simplified to maximize the likelihood of the observed data

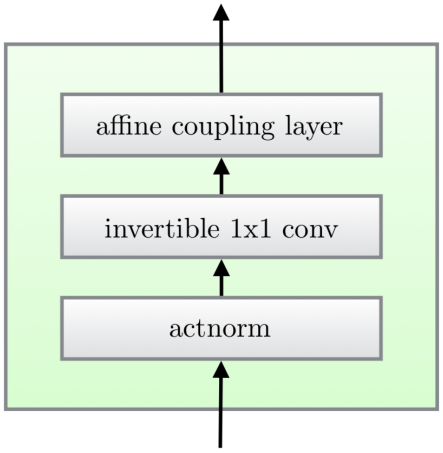


Figure 3.2: One step of the Glow model

under the model.

$$D_{KL}[p_{\mathcal{D}}(\boldsymbol{x}) \,\|\, p_\theta(\boldsymbol{x})] = \mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{D}}(\boldsymbol{x})}\left[\log \frac{p_{\mathcal{D}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right] \tag{3.12a}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{D}}(x)}\left[\log\, p_{\mathcal{D}}(\boldsymbol{x}) - \log\, p_\theta(\boldsymbol{x})\right] \tag{3.12b}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{D}}(\boldsymbol{x})}\left[\log\, p_{\mathcal{D}}(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{D}}(\boldsymbol{x})}\left[\log\, p_\theta(\boldsymbol{x})\right] \tag{3.12c}$$

where $p_D(\cdot)$ and $p_\theta(\cdot)$ respectively denote the data distribution and the model distrbition. The maximum likelihood objective to be optimized is the following

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z};\theta)p(\boldsymbol{z})d\boldsymbol{z}. \tag{3.13}$$

This integral, however, is intractable to calculate. Therefore VAE's implement a variational lower bound to this objective and optimize that.

In VAEs, the choice of the output distribution is often a diagonal Gaussian, i.e.

$$p(\boldsymbol{x}|\boldsymbol{z};\theta) = \mathcal{N}(\boldsymbol{x}|f(\boldsymbol{z};\theta),\, \sigma^2 I). \tag{3.14}$$

That is, the covariance is equal to the identity matrix times a scaler σ. The variational approximation is carried out using an auxiliary distribution function $q(\boldsymbol{z}|\boldsymbol{x})$. The relationship

between $p(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{x})$ is as follows

$$\mathcal{D}\left[q(\boldsymbol{z})|p(\boldsymbol{z}|\boldsymbol{x})\right] = \mathbb{E}_{\boldsymbol{z}\sim q}\left[\log q(\boldsymbol{z}) - \log p(\boldsymbol{z}|\boldsymbol{x})\right]. \tag{3.15}$$

Applying Bayes' rule to this equation we get:

$$\mathcal{D}\left[q(\boldsymbol{z})|p(\boldsymbol{z}|\boldsymbol{x})\right] = \mathbb{E}_{\boldsymbol{z}\sim q}\left[\log q(\boldsymbol{z}) - \log p(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z})\right] + \log p(\boldsymbol{x}), \tag{3.16}$$

where $\log p(\boldsymbol{x})$ comes out of the expectation due to its non-dependency on the latent variable. After rearrangement we obtain

$$\log p(\boldsymbol{x}) - \mathcal{D}\left[q(\boldsymbol{z})|p(\boldsymbol{z}|\boldsymbol{x})\right] = \mathbb{E}_{\boldsymbol{z}\sim q}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] + \mathcal{D}[q(\boldsymbol{z})|p(\boldsymbol{z})], \tag{3.17}$$

We are interested in such a function $q(z)$ which makes $\mathcal{D}\left[q(\boldsymbol{z})|p(\boldsymbol{z}|\boldsymbol{x})\right]$ small. In particular, it could be designed as a function of $\boldsymbol{x}$.

$$\log p(\boldsymbol{x}) - \mathcal{D}\left[q(\boldsymbol{z}|\boldsymbol{x})|p(\boldsymbol{z}|\boldsymbol{x})\right] = \mathbb{E}_{\boldsymbol{z}\sim q}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] + \mathcal{D}[q(\boldsymbol{z}|\boldsymbol{x})|p(\boldsymbol{z})], \tag{3.18}$$

The next assumption is the $q(\boldsymbol{z}|\boldsymbol{x})$ takes a Gaussian form with a trainable mean and covariance $\mathcal{N}(\boldsymbol{z}|\mu(\boldsymbol{x};\theta), \Sigma(\boldsymbol{x};\theta))$. We note that the KL divergence between two multivariate Gaussian distributions is calculated as follows:

$$\begin{aligned}&\mathcal{D}[\mathcal{N}(\mu_0, \Sigma_0)|\mathcal{N}(\mu_1, \Sigma_1)] \\ &\qquad = \frac{1}{2}\left(\mathrm{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T\Sigma_1^{-1}(\mu_1 - \mu_0) - k + \log\frac{\det\Sigma_1}{\det\Sigma_0}\right),\end{aligned} \tag{3.19}$$

where k is the dimensionality of the distribution. With VAE's assumption of a Gaussian posterior, this KL divergence simplifies to

$$\mathcal{D}[\mathcal{N}(\mu(\boldsymbol{x}), \Sigma(\boldsymbol{x}))|\mathcal{N}(0, I)] = \frac{1}{2}\left(\mathrm{tr}(\Sigma(\boldsymbol{x})) + (\mu(\boldsymbol{x}))^T(\mu(\boldsymbol{x})) - k - \log\det\Sigma(\boldsymbol{x})\right), \tag{3.20}$$

Here, we have assumed a unit Gaussian prior over the latent variable $\boldsymbol{z}$.

Next, we notice that still the integral of $\mathbb{E}_{\boldsymbol{z}\sim Q}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right]$ is intractable. However, this can be estimated through sampling. Good estimates require many samples, which would be expensive. However, in a stochastic gradient descent setting, one can use just one sample of

$\boldsymbol{z}$ and treat $\log p(\boldsymbol{x}|\boldsymbol{z})$ as an approximation to its expected value. We notice that there is already another source of stochastic gradient descent from the fact that we do not use all the entire dataset at once for training.

We can sample a single value of $\boldsymbol{x}$ and a single value of $\boldsymbol{z}$ from the distribution $q(\boldsymbol{z}|\boldsymbol{x})$, and compute the gradient of

$$\log p(\boldsymbol{x}|\boldsymbol{z}) - \mathcal{D}[q(\boldsymbol{z}|\boldsymbol{x})|p(\boldsymbol{z})]. \tag{3.21}$$

Then an average of the gradient of this function over many samples of $\boldsymbol{x}$ and $\boldsymbol{z}$ converges to the entire dataset objective

$$\begin{aligned} &\mathbb{E}_{\boldsymbol{x}\sim D}[\mathcal{D}[\mathcal{N}(\mu(\boldsymbol{x}), \Sigma(\boldsymbol{x}))|\mathcal{N}(0, I)]] \\ &\quad = \mathbb{E}_{\boldsymbol{x}\sim D}\left[\frac{1}{2}\left(\operatorname{tr}(\Sigma(\boldsymbol{x})) + (\mu(\boldsymbol{x}))^T(\mu(\boldsymbol{x})) - k - \log\det\Sigma(\boldsymbol{x})\right)\right]. \end{aligned} \tag{3.22}$$

The next step arises in realizing that while the forward pass of sampling from $\boldsymbol{z}$ is straightforward, for back-propagation over a sample we we do not have a direct form. Therefore, the authors introduce the idea of reparameterization trick. It essentially samples a unit Gaussian variable ϵ, and then introduces $\boldsymbol{z}$ as

$$\boldsymbol{z} = \epsilon * \Sigma^{1/2}(\boldsymbol{x}) + \mu(\boldsymbol{x}). \tag{3.23}$$

Thus, VAE calculates the gradient of the following

$$\begin{aligned} &\mathbb{E}_{\boldsymbol{x}\sim D}[\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[\log p(\boldsymbol{x}|\boldsymbol{z} = \mu(\boldsymbol{x}) + \Sigma^{\frac{1}{2}}(\boldsymbol{x}) * \epsilon)] \\ &\qquad -D(q(\boldsymbol{z}|\boldsymbol{x})|p(\boldsymbol{z})]. \end{aligned} \tag{3.24}$$

The derivations above can further be simplified since, with a Gaussian assumption, the logarithm of the probability $p(\boldsymbol{x}|\boldsymbol{z})$ takes the form of an L_2 loss. One can consider variational autoencoders, as autoencoders regularized with the KL divergence term, thus how it is named. A schematic of how VAE works is demonstrated in Figure 3.3.

At inference time, VAE's sample from the prior decodes the latent variable directly. Thus, the encoder will be removed altogether.

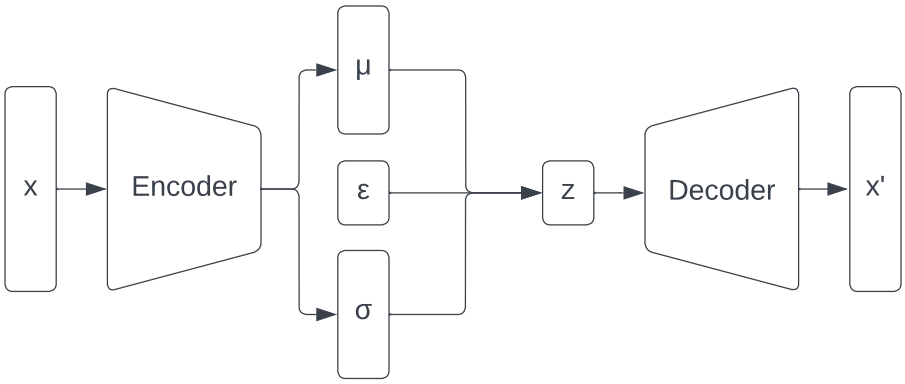


Figure 3.3: Schematic of Variational Autoencoders

3.1.4 Concluding Remarks

In this chapter, we introduced several deep learning architectures capable of learning arbitrary distributions. Each model may face inherent challenges in particular domains of application. For instance, we discussed the mode collapse phenomenon in GAN. In the following chapters, we introduce a new generative model that naturally fits better in certain application domains.

3.2 Reinforcement Learning and Imitation Learning

Reinforcement learning is the study of how agents can learn by trials and errors in sequential decision-making. The learning is carried out by rewarding/punishing the agent for repeating/forgoing the same behavior in the future. In this chapter, we review some of the main concepts in reinforcement learning.

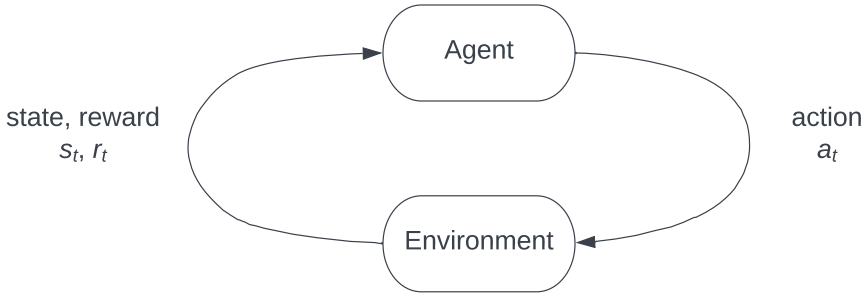


Figure 3.4: Interaction loop of the agent with the environment in an RL setup

3.2.1 Review of Reinforcement Learning Concepts

In a reinforcement learning scenario, there are two primary entities: an environment and an agent. The environment is the world where the agent lives and within which it can perform actions. At every step of interaction, the agent is provided with a (potentially partially observed) state of the environment. Then, it takes an action. As a result of the action, the (full) state of the environment changes. The state of the environment might also change due to extraneous factors. Figure 3.4 demonstrates this paradigm.

Upon taking actions, the agent also receives reward signals from the environment, a scalar number indicating how good or bad the taken action is. The goal of RL is to maximize the expected cumulative reward, known as the return. Below we make more formal definitions.

States and Observations

A *state* $\boldsymbol{s}$ is the complete information about the current status of the environment. However, this information might be available only partially to the agent, known as *observation*.

Examples of observations are raw pixel values in Atari games; or location, speed, torque, etc., of the joints of a robot. These examples show that the states can be tensors of different sizes and dimensions.

Actions

Different environment allow for different types of actions. Examples are discrete action spaces, consisting of only a finite number of actions, or continuous, consisting of real-valued vectors. Each class of action space leads to a different set of algorithms.

Policies

A policy is the governing rule for taking actions. The policies can be deterministic or stochastic. In the deterministic case, the action is sampled from a policy distribution:

$$\boldsymbol{a}_t \sim \mu(\cdot|\boldsymbol{s}_t) \tag{3.25}$$

Typically we consider the policy parameterized by a neural network and optimize its weights. Depending on the state, the network may be fully connected or convolutional.

Stochastic policies are modeled differently for categorical vs continuous policies. For categorical policies one typically samples from the discrete (softmax) distribution over actions, whereas in the continuous case, a diagonal Gaussian policy can be used.

Trajectories

A trajectory (also known as a rollout, or an episode) is a sequence of states and actions taken by the agent

$$\tau = (s_0, a_0, s_1, a_1, \ldots). \tag{3.26}$$

Initially, a state is sampled from a prior start-state distribution ρ_0.

$$\boldsymbol{s}_0 \sim \rho_0(\cdot). \tag{3.27}$$

After taking an action $\boldsymbol{a}$ at time t, the state of the environment changes. This change can be stochastic or deterministic. For the stochastic case,

$$\boldsymbol{s}_{t+1} \sim P(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t), \tag{3.28}$$

where P is a transition probability function.

Reward and Return

The reward signal is the essence of training a reinforcement learning agent. Given a state $\boldsymbol{s}$ and actions $\boldsymbol{a}$, the reward is defined as

$$r_t = R(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{3.29}$$

The goal of reinforcement learning is to maximize the expected cumulative reward received over a trajectory. The "infinite-horizon" discounted return is the summation of all rewards ever seen into the future. Since this sum can be infinite, the rewards are discounted by a factor $\gamma \in (0, 1)$

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t. \tag{3.30}$$

Often, this discount is interpreted as "having some cash now is better than having it later", similar to the notion of "interest" in economics.

Optimization in RL

The goal of RL is to maximize the expected return of the agent w.r.t to the policy function. Each trajectory is going to be produced stochastically, as a result of the stochasticity from the environment and the policy function.

$$P(\tau|\pi) = \rho_0(\boldsymbol{s}_0)\Pi_{t=0}^{T-1} P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)\pi(\boldsymbol{a}_t|\boldsymbol{s}_t). \tag{3.31}$$

Then the expected return can be defined as the integral over returns of different trajectories weighted by their probability, that is, the expectation of the expected return.

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}R(\tau). \tag{3.32}$$

We define the maximizer of J as the optimal policy and denote it with π^*.

Value Function

The value of a state is defined the expected return starting from a particular state. There exist two definitions of the value function. If the initial action is not provided, it is defined as

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)|\boldsymbol{s}_0 = \boldsymbol{s}\right], \tag{3.33}$$

and with the initial action $\boldsymbol{a}$, it is defined as

$$Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\,|\boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}\right]. \tag{3.34}$$

Bellman Equations

Bellman equations for both value functions express a recursive relationship for the value of a particular state or state-action pair

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a}\sim\pi,\,\boldsymbol{s}'\sim P}[r(\boldsymbol{s},\boldsymbol{a}) + \gamma V^{\pi}(\boldsymbol{s}')]. \tag{3.35}$$

A similar relationship holds for the value of a state-action pair,

$$Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) = \mathbb{E}_{\boldsymbol{s}'\sim P}[r(\boldsymbol{s},\boldsymbol{a}) + \gamma\mathbb{E}_{\boldsymbol{a}'\sim\pi}[Q^{\pi}(\boldsymbol{s}',\boldsymbol{a}')]], \tag{3.36}$$

where $\boldsymbol{s}' \sim P$ indicates the next state $\boldsymbol{s}'$ is sampled from the state transition probability function P. Also $\boldsymbol{a}' \sim \pi$ indicates sampling of the action $\boldsymbol{a}'$ from the stochastic policy function π.

Advantage Function

For better training of the policy function π, it might be of interest to optimize not for the absolute value function, but instead its relative advantage. This approach gives rise to the notion of advantage function. The advantage function $A^{\pi}(\boldsymbol{s},\boldsymbol{a})$ represents the amount by which taking action $\boldsymbol{a}$ at state $\boldsymbol{s}$ is better, comparing the average value when the action is

randomly taken with policy function $\pi(\cdot|\boldsymbol{s})$. Concretely,

$$A^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) - V^{\pi}(\boldsymbol{s}). \tag{3.37}$$

Policy Gradient

The key idea of policy gradient [46] is to encourage actions with higher return and discourage actions with low returns. The core equation of policy gradient is the gradient of the expected finite-horizon un-discounted return of the policy.

$$\mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) A^{\pi_\theta}(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \tag{3.38}$$

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: policy parameters θ_0, value function parameters ϕ_0

2: **for** $k = 0, 1, 2, ...$ **do**

3: Gather trajectories $\mathcal{D}_k = \{\tau_i\}$ by taking actions according to the policy π parameterized by θ_k.

4: Use the current value function V_{ϕ_k} to estimate the advantage, $\hat{A}_t$.

5: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

6: Use gradient ascent, or Adam optimizer, to update the policy

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k.$$

7: Apply gradient descent on the mean-squared error of the value function:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

where $\hat{R}_t$ denotes "rewards-to-go".

8: **end for**

In step, the reward-to-go is defined as the sum of rewards after a point in a trajectory

$$\hat{R}_t \doteq \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) \tag{3.39}$$

This form is not directly apt for back-propagation as it involves an expected value. Towards making the loss function, one can use a lemma that is straightforward to be shown

$$\mathbb{E}_{\boldsymbol{x}\sim P_\theta} \left[\nabla_\theta \log P_\theta(\boldsymbol{x})\right] = 0. \tag{3.40}$$

Trust Region Policy Optimization (TRPO)

Trust region policy optimization is an algorithm that has better convergence properties than the vanilla gradient algorithm. It takes the largest possible step for performance improvement, while satisfying a constraint on the KL-Divergence, that is, the difference between the previous and the current policy distributions. In other words, it tries to keep the new and old policies close in parameter space. This helps the policy not to undergo radical changes, and thus stabilizes training. The reason is that small changes in parameter space (of the policy function) can dramatically affect the performance because single bad steps may totally deteriorate a whole episode's performance. Therefore, very large steps in optimization are not so desired.

In principle, TRPO aims to take the following update step:

$$\begin{aligned} \theta_{k+1} &= \arg\max_\theta \ \mathcal{L}(\theta_k, \theta) \\ &\text{s.t. } \bar{D}_{KL}(\theta||\theta_k) \leq \delta \end{aligned} \tag{3.41}$$

Here, $\mathcal{L}$ is the *surrogate advantage*, which is a measure of how much the new policy π_θ outperforms the old policy π_{θ_k}. It is defined as

$$\mathcal{L}(\theta_k, \theta) = \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\pi_{\theta_k}} \left[\frac{\pi_\theta(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\theta_k}(\boldsymbol{a}|\boldsymbol{s})} A^{\pi_{\theta_k}}(\boldsymbol{s}, \boldsymbol{a})\right], \tag{3.42}$$

Also, $D_{KL}(\theta||\theta_k)$ is a shorthand for the KL-divergence between the distribution of actions with the old policy vs. the updated policy:

$$\bar{D}_{KL}(\theta||\theta_k) = \mathbb{E}_{\boldsymbol{s}\sim\pi_{\theta_k}} \left[D_{KL}\left(\pi_\theta(\cdot|\boldsymbol{s})||\pi_{\theta_k}(\cdot|s)\right)\right]. \tag{3.43}$$

Monotonically improve performance.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is another algorithm that aims to take the biggest possible policy improvement step without stepping so far that performance accidentally deteriorates. It achieves that by optimizing the following "surrogate" objective at i^{th} iteration:

$$\begin{aligned} &\mathcal{L}_{\text{PPO}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}_i, \boldsymbol{\theta}) = \\ &\min\left(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}_i}(\boldsymbol{a}|\boldsymbol{s})} A^{\pi_{\boldsymbol{\theta}_i}}(\boldsymbol{s}, \boldsymbol{a}),\ \ g(\epsilon, A^{\pi_{\boldsymbol{\theta}_i}}(\boldsymbol{s}, \boldsymbol{a}))\right), \end{aligned} \tag{3.44}$$

where A is the advantage function [43], and g is a "clipping" function for some small parameter $\epsilon > 0$:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases} \tag{3.45}$$

Inverse Reinforcement Learning

There are certain scenarios in real world where the reward signal is available, such as in predicting stock prices, health care, etc. In such cases, RL is an apt approach to solve the problem. However, there might be other cases where no well-defined reward signal is provided. For example, the desired objective might be too complex and difficult for humans to write in an explicit form. In such cases, it is desired for the learning agent to imitate the behavior of an expert directly. This type of learning is often known as imitation learning.

Historically, there have been several mainstream approaches to address this problem: behavioral cloning (BC) and Inverse Reinforcement Learning (IRL). Besides, a more modern approach, known as Generative Adversarial Imitation Learning (GAIL), optimizes an adversarial objective. This latter approach has close connections with the IRL approach. In the rest of this section, we explain each approach.

In Inverse Reinforcement Learning (IRL), the assumption is to think of the expert as

trying to maximize a reward function of features. With this paradigm [1] tried to recover this reward function assuming that is expressible as a linear combination of known. The resulting algorithm involved solving a linear program

$$\max_{c \in \mathcal{C}} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi} \left[c(s,a) \right] \right) - \mathbb{E}_{\pi_E} \left[c(s,a) \right] \tag{3.46}$$

where $H(\pi) = \mathbb{E}_{\pi} \left[-\log \pi(a|s) \right]$ is the γ-discounted causal entropy of the policy π. Also, π_E is accessible through a set of trajectories (sequences of state-action pairs) sampled by running the policy π_E in the environment. This objective finds a cost function c that assigns lower costs to the expert policy comparing any other policy. Typically an IRL step is followed by a reinforcement learning step which minimizes the expected cumulative cost, regularized by an entropy term:

$$\mathrm{RL}(c) = \arg\min_{\pi \in \Pi} - H(\pi) + \mathbb{E}_{\pi} \left[c(s,a) \right]. \tag{3.47}$$

where $H(\pi)$ is the γ-discounted causal entropy of the policy π_θ defined as $\mathbb{E}_{\pi} \left[-\log \pi(a|s) \right]$.

Behavior Cloning

Behavior cloning involves a "supervised learning" view of the imitation learning problem. This algorithm maximizes the likelihood of the expert actions given the states in the expert trajectories.

$$\max_{\theta} \pi_\theta(\boldsymbol{a}|\boldsymbol{s}) \tag{3.48}$$

A Gaussian assumption on the policy output simplifies maximizing the likelihood to minimizing the L_2 distance between the expert actions and the predicted actions by the policy π_θ. This approach was first adopted by [36], where the authors taught a car to self-drive given the images of the road. An illustration of this method is shown in Figure 3.5.

A major drawback of the BC approach is known to be its lack of any mechanism to take into account the long-term consequences of taking particular actions. As the objective implies, BC only considers the immediate next actions. Since BC does not build on recovering the underlying reward function, it will not optimize for the long-term return. As a result,

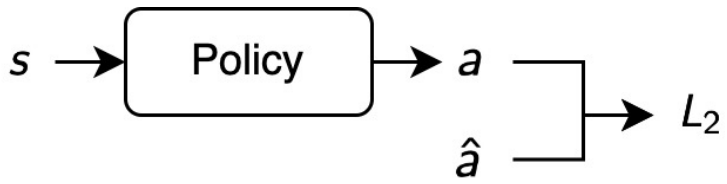


Figure 3.5: Behavior cloning pipeline

it might favor slightly suboptimal actions in the long term sense. Suboptimal actions might lead the agent to visit states that are slightly off the distribution of state and actions provided via the expert trajectories. As a result, the actions taken by the policy function might be even more suboptimal. Such a dynamic will diminish the quality of actions in an increasing fashion. Eventually, the agent might fall into catastrophic states. This problem is often referred to as compounding errors.

3.2.2 Generative Adversarial Imitation Learning

Many existing IRL algorithms are computationally expensive, because their inner loop requires one pass of RL. Therefore, they are not directly scalable to large environments. On the other hand, though the learned reward is used for performing expert actions, it is not directly useful for any other purpose. Therefore, Generative Adversarial Imitation Learning (GAIL) [18] was proposed to address such challenges.

GAIL optimizes the following objective

$$\begin{aligned}\min_{\pi_{\boldsymbol{\theta}}} \max_{D} \{&\mathbb{E}_{\pi_{\boldsymbol{\theta}}} \left[\log D(\boldsymbol{s}, \boldsymbol{a})\right] \\ &+\mathbb{E}_{\pi_E} \left[\log \left(1 - D(\boldsymbol{s}, \boldsymbol{a})\right)\right] \\ &-\lambda H(\pi_{\boldsymbol{\theta}})\}.\end{aligned} \tag{3.49}$$

The optimization involves 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) [45] or Proximal Policy Optimization (PPO) [44] step to decrease it with respect to the policy parameters. This process is demonstrated in Figure 3.6.

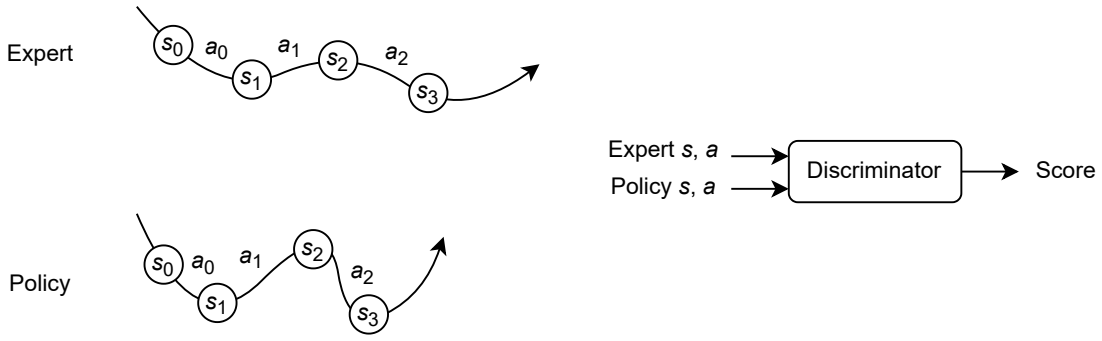


Figure 3.6: Generative adversarial imitation learning pipeline

GAIL uses a novel paradigm to avoid computational complexity. In this work, the authors define a notion called occupancy measure, which is very similar to probability distributions. They show that directly optimize for matching the occupancy measure of imitating state-action pairs with that of the expert. It is shown that the GAIL optimization under certain assumptions, GAIL objective is equivalent to performing RL following IRL. Therefore, GAIL provides an imitation learning pipeline while bypassing the expensive intermediate IRL step.

In contrast to BC, GAIL deploys an RL framework and optimizes for trajectory sequences that, along the way, produce states and actions similar to the expert. Thus it has been shown successful to learn a range of tasks that has been too difficult for BC to learn. However, as with any adversarial objective, GAIL is very unstable to train. Various efforts have tried to improve the stability of training. The most successful variant of GAIL adopts the Wasserstein GAN framework.

GAIL is also prone to mode collapse (see Section 3.1.1.1). If a range of actions is possible at a particular state, GAIL may not learn all of them properly. Mode collapse is the greatest challenge with GAIL. The work of this thesis alleviates such problems.

3.2.2.1 VAE-GAIL

The main idea behind VAE-GAIL is that while VAEs can model diverse behaviors without dropping modes, they fail to learn robust policies. On the other hand, GAIL can learn robust

policies. However, the learned behaviors do not tend to be sufficiently diverse. Therefore, VAE-GAIL proposes a way to combine the effect of the two methods.

The first step towards solving the multimodality problem in VAE-GAIL is using a variational autoencoder. This framework adopts the following loss functions for learning an embedding for each trajectory.

$$\mathcal{L}(\alpha, w, \phi; \tau_i) = -\mathbb{E}_{q_\psi(z|x^i_{1:T_i})}\big[\sum \log \pi_\alpha(a^i_t|x^i_t, z) + \log p_w(x^i_{t+1}|x^i_t, z)\big] + D_{KL}(q_\phi(z|x^i_{1:T_i})||p(z))\big] \tag{3.50}$$

Here, q denotes an encoder module that uses a bi-directional LSTM. To produce the final embedding, it calculates the average of all the outputs of the second layer of this LSTM before applying a final linear transformation to generate the mean and standard deviation of a Gaussian. Then, a sample from this Gaussian is drawn as the demonstration encoding. The action decoder is an MLP that maps both the state and the embedding to the parameters of a Gaussian distribution. The state decoder is similar to a conditional WaveNet model [48].

To enable GAIL to produce diverse solutions, VAE-GAIL uses the embeddings generated by the VAE encoder and conditions the discriminator in the GAIL objective on that embedding, then integrates it with respect to the variational posterior $q_\phi(z|x_{1:T})$. Specifically, the training of VAE-GAIL involves optimizing the following objective for the discriminator.

3.2.2.2 InfoGAIL

The idea of InfoGAIL is to extend InfoGAN (see Section 4.5.3) to imitation learning settings. InfoGAIL considers an auxiliary posterior network to predict the latent variable given a state-action pair. The objective of the posterior network ψ is to maximize the likelihood of the state-action pair

$$\Delta_{\psi_i} = \hat{\mathbb{E}}_{\mathcal{X}_i}\left[\nabla_{\psi_i} \log Q_{\psi_i}(c|s, a)\right], \tag{3.51}$$

where $\mathcal{X}_i$ denotes expert trajectories.

InfoGAIL promotes mutual information between the expert trajectories and the latent

codes via a variational lower bound (since the calculation of actual mutual information requires access to the posterior $P(c|\tau)$). This lower bound takes the following form

$$\begin{aligned} L_I(\pi, Q) &= \mathbb{E}_{c\sim p(c), a\sim\pi(\cdot|s,c)} \left[\log Q(c|\tau)\right] + H(c) \\ &\leq I(c;\tau) \end{aligned} \tag{3.52}$$

where $Q(c|\tau)$ approximates the true posterior $P(c|\tau)$. In order to discover the semantical meanings in the latent codes, InfoGAIL seeks to maximize the mutual information between latent codes and trajectories.

Overall, the training pipeline of InfoGAIL can be re-written as the following modification to GAIL objective

$$\min_{\pi,Q} \max_{D} \mathbb{E}_{\pi} \left[\log D(s,a)\right] + \mathbb{E}_{\pi_E} \left[\log(1 - D(s,a))\right] - \lambda_1 L_I(\pi, Q) - \lambda_2 H(\pi) \tag{3.53}$$

To simplify the computation, the posterior is simplified to approximation $Q(c|s,a)$, since directly working with entire trajectories τ would be too expensive, especially when the dimension of the observations is very high (such as images). Therefore, the network Q can be chosen as a fully connected or convolutional network that takes states and actions to softmax probabilities (for discrete latent variables) or parameters of a Gaussian distribution (for continuous latent variables). We shall emphasize that for the continuous case, the log probability of the Gaussian posterior simplifies to an L_2 distance between the true latent variable and the recovered latent variable

$$\hat{\mathbb{E}}_{\mathcal{X}_i}[\log D_{w_{i+1}}(s,a)] - \lambda_1 L_I(\pi_{\theta_i}, Q_{\psi_{i+1}}) - \lambda_2 H(\pi_{\theta_i})]. \tag{3.54}$$

Lastly, the reward of GAIL is augmented with the L_I term to encourage mutual information. Hence, the procedure of training InfoGAIL involves alternating optimization of GAIL objective and posterior network, where GAIL uses the reward of Equation (3.54).

3.2.2.3 TripleGAIL

TripleGAIL adopts a loss consisting of three terms

$$\begin{aligned}\min_{\alpha,\theta}\max_{\psi}\ &\mathbb{E}_{\pi_E}[\log(1-D_\psi(s,a,c)] \\ &+\omega\mathbb{E}_{\pi_\theta}\left[\log D_\psi(s,a,c)\right] \\ &+(1-\omega)\mathbb{E}_C[\log D_\psi(s,a,c)] - \lambda_H H(\pi_\theta)\end{aligned} \quad . \tag{3.55}$$

Here, the first term encourages the discriminator to detect expert state-action pairs as expert data. The second term encourages generated trajectories with the learned policy π_θ to be detected as fake. Lastly, the third term ensures recovering correct latent codes for expert state-action pairs, where the expectation over C denotes the selection of latent codes given the state-action pairs. Hyper-parameter $\omega \in (0, 1)$ balances the weights of policy generation and skill selection. In a sense, the idea of TripleGAIL is similar to InfoGAIL, in that it aims to recover codes and generate imitated trajectories simultaneously. The only difference is that TripleGAIL adopts a combined three-term loss function for that purpose.

CHAPTER 4

Self-Organizing Generative Models

In Section 3.1 various types of generative models and their objectives were discussed. In general, each model has certain pros and cons inherited from its method of optimization. For instance, Generative Adversarial Networks often fail to reproduce parts of the data distribution. This is known as "mode collapse" and is discussed in Section 3.1.1. This drawback is addressed and mitigated in a variety of application domains, such as image synthesis. However, it remains a fundamental challenge with this model in its generic forms, which requires fixes for each specific application domain. Variational Autoencoders (see Section 3.1.3) also incur some amount of mode collapse due to imperfect matching of the posterior distribution with a desired Gaussian distribution. This is because the KL divergence term in the objective only acts as a regularizer. Lastly, Flow models do not provide any means to compress data. As a result, they are not directly applicable to problems where a low dimensional representation of data is desired.

One particular class of generative models address the challenges above by removing the recurrent encoder module, and instead directly search over samples of the latent variable [49, 4, 19]. In this thesis, we examine the theory and properties of these models in a much greater depth. Furthermore, prior work on encoder-free models has mainly focused on non-sequential data. In chapter 5, we propose considering encoder-free models for sequential decision making, and in particular, multimodal imitation learning.

The rest of this chapter is organized as follows. First, we explain the methodology of our encoder-free algorithm in its general form. Second, we derive the theoretical logic behind

this model. Third, we discuss the self-organization phenomenon where similar data points cluster together in the latent space. And lastly, we provide several experimental results verifying that our model is capable of learning arbitrary distributions.

4.1 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:

$$p(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x}; f) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{z}, \boldsymbol{x}), \sigma^2 \boldsymbol{I}), \tag{4.1}$$

   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 $\boldsymbol{y}$.

This process is illustrated in Figure 4.1.

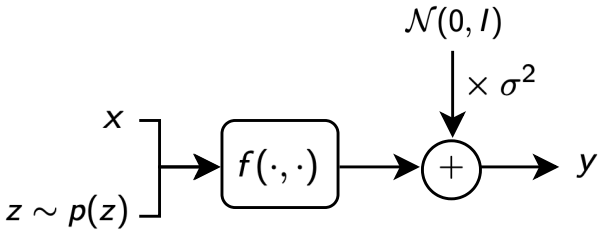


Figure 4.1: The generative process of data distribution.

4.1.1 The SOG Algorithm in General

Given 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, we design an algorithm to estimate $\boldsymbol{\theta}$, such that the samples generated by the inferred model match the given data in distribution. First, 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 $\boldsymbol{\theta}$. Next, starting with a random initialization of $\boldsymbol{\theta}$, an iterative algorithm is adopted: (1) Given current $\boldsymbol{\theta}$, one computes estimates of the latent variables $\boldsymbol{z}_i$, $i = 1, \cdots, N$, corresponding to each data point. In a marginal likelihood framework, a posterior distribution for $\boldsymbol{z}_i$ is estimated. In our SOG algorithm, we instead obtain point estimates $\boldsymbol{z}_i = \boldsymbol{z}_i^*$, where $\boldsymbol{z}_i^* = \arg\max_{\boldsymbol{z}_i} p(\boldsymbol{y}_i|\boldsymbol{z}_i, \boldsymbol{x}_i; \boldsymbol{\theta})$. If the latent variable is discrete, $\boldsymbol{z}_i^*$ is found by an exhaustive search over $\boldsymbol{z} \in \{1, \ldots, K\}$. In the continuous case, the optimal $\boldsymbol{z}$ is searched over samples of the prior distribution $p(\boldsymbol{z})$. (2) Given the best latent codes, one takes a gradient step on $\boldsymbol{\theta}$ to increase the likelihood $p(\boldsymbol{y}_i|\boldsymbol{z}_i^*, \boldsymbol{x}_i; \boldsymbol{\theta})$. Both steps decrease the same loss function, guaranteeing convergence.

Algorithm 2 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 $\boldsymbol{z}_i \coloneqq \arg\min_{\boldsymbol{z}_j} \mathcal{L}\left(f_{\boldsymbol{\theta}}(\boldsymbol{z}_j, \boldsymbol{x}_i), \boldsymbol{y}_i\right)$.
    end for
    Calculate $\mathcal{L}_{\text{SOG}} = \sum_{i=1}^{N_{\text{data}}} \mathcal{L}\left(f_{\boldsymbol{\theta}}(\boldsymbol{z}_i, \boldsymbol{x}_i), \boldsymbol{y}_i\right)$ 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

Formal relationships and equivalences between the SOG algorithm – that iteratively computes a Maximum Likelihood Estimate (MLE) of $\boldsymbol{z}$ for given $\boldsymbol{\theta}$ and then increases the likelihood of data computed at these MLE estimates of $\boldsymbol{z}$ by updating the model parameters $\boldsymbol{\theta}$ – and algorithms based on maximizing the marginalized likelihood – that iteratively compute a posterior distribution of $\boldsymbol{z}$ for given $\boldsymbol{\theta}$, and then increase the likelihood of data marginalized over the distribution of $\boldsymbol{z}$ by updating the model parameters $\boldsymbol{\theta}$ – are included in Sections 4.2 and 4.3.

4.1.2 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. Due

to space limits, the corresponding algorithms are presented in Section 5.2.

Multimodal Behavior Cloning with SOG

We adopt Algorithm 2 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 4.

Multimodal Combination of BC and GAIL using SOG

Next, we introduce Algorithm 5, where we combine SOG, as a means for BC, with GAIL to ensure robustness towards unseen states. This algorithm is inspired by [21], which in a unimodal setting optimizes a weighted sum of BC loss and the GAIL "surrogate" loss (see Section 3.2.2).

4.1.3 Further Properties of SOG

Sample Complexity

Training procedure of Algorithm 2, given a high-dimensional continuous latent variable, requires a prohibitively large number of latent code samples (line 7 of the algorithm). In Section 4.5, we propose a computationally efficient extension of this algorithm suited for high dimensions of the latent space.

Self-Organization in the Latent Space

In Section 4.4, we show that latent codes corresponding to nearby data points get organized close to each other. In addition, visual results in Sections 4.5 and 5.3.4 show that different regions of the latent space organize towards generating different modes of data. Similar properties have been studied in "self-organizing maps" [3], without a deep network. Hence,

we chose the phrase "self-organization" in naming of Algorithm 2.

4.2 Discrete Latent Variables: Maximum Marginal Likelihood Approach vs SOG

Expectation-maximization (EM) is a well-known method for optimizing the maximum likelihood objective –marginalized over the latent variables – in models involving latent variable. In this section, we demonstrate how the iterative procedure of Algorithm 2 can be derived as a special case of the EM algorithm when the latent variable has a discrete distribution.

4.2.1 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 4.1, 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 any given arbitrary distribution $q(\boldsymbol{Z})$ as follows:

$$\begin{aligned}
&\mathcal{L}(q(\boldsymbol{Z}); \boldsymbol{\theta}, \boldsymbol{\Pi}) \\
&= \mathbb{E}_{\boldsymbol{Z}\sim q(\boldsymbol{Z})} \log \{p(\boldsymbol{Y}, \boldsymbol{Z}|\boldsymbol{X};\, \boldsymbol{\theta}, \boldsymbol{\Pi})\} + H(q(\boldsymbol{Z})) && \text{(4.2a)} \\
&= \mathbb{E}_{\boldsymbol{Z}\sim q(\boldsymbol{Z})} \log \{p(\boldsymbol{Y}|\boldsymbol{X};\, \boldsymbol{\theta}, \boldsymbol{\Pi})\} \\
&\quad + \mathbb{E}_{\boldsymbol{Z}\sim q(\boldsymbol{Z})} \log \left\{ \frac{p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{Y};\, \boldsymbol{\theta}, \boldsymbol{\Pi})}{q(\boldsymbol{Z})} \right\} && \text{(4.2b)} \\
&\equiv \log p(\texttt{data}|\boldsymbol{\theta}, \boldsymbol{\Pi}) \\
&\quad - D_{\mathrm{KL}}\big(q(\boldsymbol{Z}) \,||\, p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{Y};\, \boldsymbol{\theta}, \boldsymbol{\Pi})\big) && \text{(4.2c)}
\end{aligned}$$

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

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

$$q^{t+1}(\boldsymbol{Z}) \leftarrow \arg\max_{q} \mathcal{L}(q;\, \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) \tag{4.3a}$$

$$= p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{Y};\, \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) \tag{4.3b}$$

$$\equiv \prod_{i=1}^{N} r^t_{i,\boldsymbol{z}_i} \tag{4.3c}$$

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

$$\begin{aligned} &\boldsymbol{\theta}^{t+1}, \boldsymbol{\Pi}^{t+1} \\ &\leftarrow \arg\max_{\boldsymbol{\theta}, \boldsymbol{\Pi}} \mathcal{L}(q^{t+1};\, \boldsymbol{\theta}, \boldsymbol{\Pi}) \end{aligned} \tag{4.4a}$$

$$= \arg\max_{\boldsymbol{\theta}, \boldsymbol{\Pi}} \mathbb{E}_{\boldsymbol{Z} \sim q^{t+1}}[\log P(\boldsymbol{Y}, \boldsymbol{Z}|\boldsymbol{X};\, \boldsymbol{\theta}, \boldsymbol{\Pi})] \tag{4.4b}$$

$$\begin{aligned} =\ &\arg\max_{\boldsymbol{\theta}, \boldsymbol{\Pi}} \sum_{i=1}^{N} \sum_{k=1}^{K} r^t_{ik} (\log \pi_k \\ &+ \log p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{z}_i = k;\, \boldsymbol{\theta})) \end{aligned} \tag{4.4c}$$

where

$$\begin{aligned} r^t_{ik} &= p(\boldsymbol{z}_i = k \mid \boldsymbol{x}_i, \boldsymbol{y}_i;\, \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) \\ &= \frac{p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{z}_i = k;\, \boldsymbol{\theta}^t)\, \pi^t_k}{\sum_{l=1}^{K} p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{z}_i = l;\, \boldsymbol{\theta}^t)\, \pi^t_l}. \end{aligned} \tag{4.5}$$

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 [3]. Also, it is straightforward to derive the following update rule for $\boldsymbol{\Pi}$ by maximizing the objective of Equation (4.4c) subject to the constraint $\sum_{k=1}^{K} \pi_k = 1$:

$$\pi^t_k \leftarrow \frac{\sum_i r^t_{ik}}{\sum_{il} r^t_{il}}. \tag{4.6}$$

The E step can be carried out in two ways:

1. **Soft.** using all the posterior probabilities in Equation (4.5), 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. In the following subsection, we derive when the regular EM reduces to the Hard EM method.

4.2.2 Hard Assignment EM and SOG (Algorithm 2)

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 [3]. As in SOG, one replaces the probabilities r_{ik} in Equation (4.5) with binary values denoted as γ_{ik}. This hard assignment is performed by allocating all the probability mass to the categorical variable for which the posterior probability is the maximum.

Let $k_i^* = \arg\max_k p(\boldsymbol{z}_i = k \mid \boldsymbol{x}_i, \boldsymbol{y}_i;\ \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t)$. Then in the Hard EM case, the E step becomes

$$\gamma_{ik}^t = \begin{cases} 1, & k = k_i^* \\ 0, & k \neq k_i^* \end{cases}, \tag{4.7}$$

and the M step of Equation (4.4c) (w.r.t. $\boldsymbol{\theta}$) simplifies to:

$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} ||f(\boldsymbol{z}_i = k_i^*, \boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{y}_i||^2. \tag{4.8}$$

It follows from Equation (4.5) that if $\pi_k = 1/K;\ k = 1, \ldots, K$, then

$$\begin{aligned} &\arg\max_{k \in \{1,\ldots,K\}} p(\boldsymbol{z}_i = k \mid \boldsymbol{x}_i, \boldsymbol{y}_i;\ \boldsymbol{\theta}^t) \\ &\qquad = \arg\max_{k \in \{1,\ldots,K\}} p(\boldsymbol{y}_i \mid \boldsymbol{z}_i = k, \boldsymbol{x}_i, ;\ \boldsymbol{\theta}^t). \end{aligned}$$

Hence, the Hard EM is equivalent to the update step in Algorithm 2 in this special case.

Next we provide a sufficient condition under which both the Hard EM and SOG (for any values of π_k's) become a special case of the general EM. First, recall that

$$\begin{aligned} & p(\boldsymbol{y}_i|\boldsymbol{z}_i = k, \boldsymbol{x}_i; \boldsymbol{\theta}) \\ & \quad \propto \exp[-||f(\boldsymbol{z}_i = k, \boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{y}_i||^2/(2\sigma^2)], \end{aligned} \tag{4.9}$$

where σ determines the spread around the manifold of data. Let

$$k_i^\dagger = \underset{k_i \in \{1,\ldots,K\}}{\arg\max}\ p(\boldsymbol{y}_i|\boldsymbol{z}_i = k, \boldsymbol{x}_i; \boldsymbol{\theta}) \tag{4.10a}$$

$$= \underset{k_i \in \{1,\ldots,K\}}{\arg\min}\ ||f(\boldsymbol{z}_i = k, \boldsymbol{x}_i; \boldsymbol{\theta}) - \boldsymbol{y}_i||^2. \tag{4.10b}$$

Assuming a very small spread σ, we get for all $k \neq k_i^\dagger$

$$\lim_{\sigma \to 0} \frac{p(\boldsymbol{y}_i|\boldsymbol{z}_i = k, \boldsymbol{x}_i; \boldsymbol{\theta})}{p(\boldsymbol{y}_i|\boldsymbol{z}_i = k_i^\dagger, \boldsymbol{x}_i; \boldsymbol{\theta})} = 0\ , \tag{4.11}$$

and substituting these probabilities in the equation for r_{ik}^t in Equation (4.5), we obtain that the posteriors follow the same pattern:

$$\lim_{\sigma \to 0} p(\boldsymbol{z}_i = k|\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}) = \begin{cases} 1, & k = k_i^\dagger \\ 0, & k \neq k_i^\dagger \end{cases}\ , \tag{4.12}$$

Thus, the SOG algorithm that computes an MLE estimate of $\boldsymbol{z}$ at every iteration (i.e. the value of the latent variable for which data likelihood is maximized given $\boldsymbol{\theta}$) is equivalent to the standard EM when $\sigma \approx 0$.

4.3 Continuous Latent Variables: Maximum Marginal Likelihood Approach vs SOG

In this section, we consider the case of continuous latent variables, and demonstrate that in the asymptotic limit where σ is sufficiently small, the same relationship derived in the preceding section for the discrete case still holds: The marginalized data log-likelihood $p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta})$

is well approximated by the maximum of the data log-likelihood computed over the latent variables as done in Algorithm 2. In this limit, we can use Laplace approximation for integrals. Laplace approximation provides a general way to approach marginalization problems. The basic setting for Laplace approximation is a multivariate integral of the form:

$$I = \int e^{-h(\boldsymbol{z})/t}\, d\boldsymbol{z}, \tag{4.13}$$

where $\boldsymbol{z}$ is a d-dimensional variable, and $h(\boldsymbol{z})$ is a scalar function with a unique global minimum at $\boldsymbol{z} = \boldsymbol{z}^*$. Laplace approximation considers a Taylor series expansion of $h(\boldsymbol{z})$ around $\boldsymbol{z}^*$. In this way, as $t \to 0$, the integral of Equation (4.13) is approximated as:

$$I = e^{-h(\boldsymbol{z}^*)/t}\, (2\pi)^{d/2}\, |\boldsymbol{A}|^{1/2}\, t^{d/2} \; + O(t), \tag{4.14}$$

where $\boldsymbol{A} = (\boldsymbol{D}^2 h(\boldsymbol{z}^*))^{-1}$ is the inverse of the Hessian of $h(\boldsymbol{z})$ evaluated at $\boldsymbol{z}^*$.

Now, consider the expression for the data likelihood of the model in Section 4.1

$$\begin{aligned} I &= p(\boldsymbol{y}|\boldsymbol{x};\, \boldsymbol{\theta}) && \text{(4.15a)} \\ &= \int p(\boldsymbol{y}|\boldsymbol{z}, \boldsymbol{x};\, \boldsymbol{\theta}) p(\boldsymbol{z}) d\boldsymbol{z} && \text{(4.15b)} \\ &\propto \int \exp\left[-||f(\boldsymbol{z}, \boldsymbol{x}) - \boldsymbol{y}||^2/(2\sigma^2)\right] \\ &\qquad \times \exp\left[-||\boldsymbol{z}||^2/2\right] d\boldsymbol{z}. && \text{(4.15c)} \end{aligned}$$

For σ sufficiently small, we can use Laplace approximation for this integral. Specifically, comparing with Equation (4.13) we identify:

$$\begin{aligned} h(\boldsymbol{z}) &\coloneqq ||f(\boldsymbol{z}, \boldsymbol{x}) - \boldsymbol{y}||^2 + \sigma^2 ||\boldsymbol{z}||^2, && \text{(4.16a)} \\ t &\coloneqq 2\sigma^2, && \text{(4.16b)} \end{aligned}$$

thus deriving:

$$\begin{aligned} \boldsymbol{z}^* &= \arg\min h(\boldsymbol{z}) && \text{(4.17a)} \\ &= \arg\min ||f(\boldsymbol{z}, \boldsymbol{x}) - \boldsymbol{y}||^2 + \sigma^2 ||\boldsymbol{z}||^2 && \text{(4.17b)} \\ &\approx \arg\min ||f(\boldsymbol{z}, \boldsymbol{x}) - \boldsymbol{y}||^2, && \text{(4.17c)} \end{aligned}$$

which is equivalent to line 8 of Algorithm 2. The only nuance is that Algorithm 2 finds an approximation to $\boldsymbol{z}^*$ by searching among a batch of sampled candidate $\boldsymbol{z}$'s. However, given large batches of $\boldsymbol{z}$'s, the best candidate stays close to $\boldsymbol{z}^*$ with a high probability. Sampling of $\boldsymbol{z}$ from a Gaussian prior ensures that $||\boldsymbol{z}||$ remains small, making the approximation of Equation (4.17c) valid.

Plugging the terms in Equation (4.16) into Equation (4.14), we obtain the following approximation to the data log-likelihood, which is later maximized w.r.t. model parameters $\boldsymbol{\theta}$,

$$\begin{aligned}\log p(\boldsymbol{y}|\boldsymbol{x};\,\boldsymbol{\theta}) \approx &-\frac{1}{2\sigma^2}\left(||f(\boldsymbol{z}^*,\boldsymbol{x})-\boldsymbol{y}||^2+\sigma^2||\boldsymbol{z}^*||\right)\\ &+\frac{1}{2}\log|\boldsymbol{A}|\\ &+d\log\sigma\\ &+\text{constant}.\end{aligned} \tag{4.18}$$

We notice that in the asymptote of $\sigma \to 0$, the term $||f(\boldsymbol{z}^*,\boldsymbol{x})-\boldsymbol{y}||^2/(2\sigma^2)$ dominates the other terms. Particularly, $||\boldsymbol{z}^*||$ and $\log|\boldsymbol{A}|$ do not depend on σ, and the growth of $\log\sigma$ is dominated by that of $1/\sigma^2$. Therefore, in the limit we can approximate Equation (4.18) with only the first summand. That is,

$$\log p(\boldsymbol{y}|\boldsymbol{x};\,\boldsymbol{\theta}) \approx -\frac{1}{2\sigma^2}\,||f(\boldsymbol{z}^*,\boldsymbol{x})-\boldsymbol{y}||^2, \tag{4.19}$$

which yields line 10 of Algorithm 2 since σ is constant w.r.t. $\boldsymbol{\theta}$. We assume Lipchitz continuity for f. Thus, the approximate choice of $\boldsymbol{z}^*$, given large batches over the latent codes, introduces an insignificant error to Equation (4.19).

It is worth mentioning that similar approximation results are well studied in the framework of Bayesian model selection. In that context, the data likelihood is marginalized instead over the model parameters, and also the asymptotic variable is rather the number of observed data samples becoming large [3].

4.4 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 2. 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 2 over all points in the neighborhood is small.

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 $B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y) = \{(\boldsymbol{x}, \boldsymbol{y}) \mid ||\boldsymbol{x} - \boldsymbol{x}_1|| < \delta_x/M,\ ||\boldsymbol{y} - \boldsymbol{y}_1|| < \delta_y\}$. 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 4.4.1. *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:

$$
\begin{aligned}
&|||f(\boldsymbol{x}_1, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y}_1|| - ||f(\boldsymbol{x}_2, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y}_2||| && \text{(4.20a)}\\
&\leq ||f(\boldsymbol{x}_1, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y}_1 - (f(\boldsymbol{x}_2, \boldsymbol{z}; \boldsymbol{\theta}) - \boldsymbol{y}_2)|| && \text{(4.20b)}\\
&\leq ||f(\boldsymbol{x}_1, \boldsymbol{z}; \boldsymbol{\theta}) - f(\boldsymbol{x}_2, \boldsymbol{z}; \boldsymbol{\theta})|| + ||\boldsymbol{y}_1 - \boldsymbol{y}_2|| && \text{(4.20c)}\\
&\leq M||\boldsymbol{x}_1 - \boldsymbol{x}_2|| + ||\boldsymbol{y}_1 - \boldsymbol{y}_2|| && \text{(4.20d)}\\
&< \delta_x + \delta_y = \delta. && \text{(4.20e)}
\end{aligned}
$$

where Equations (4.20b) and (4.20c) follow reverse triangle inequality and triangle inequality, respectively. Besides, Equation (4.20d) follows from the Lipschitz continuity assumption. □

Proposition 4.4.2. *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 4.4.1,

the following sequence of inequalities holds

$$
\begin{aligned}
d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_i) &\geq d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_i) - \delta && \text{(4.21a)} \\
&> Cd(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) - \delta && \text{(4.21b)} \\
&> Cd(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) - \eta d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) && \text{(4.21c)} \\
&= (C - \eta) d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}_j) && \text{(4.21d)} \\
&> (C - \eta)(d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_j) - \delta) && \text{(4.21e)} \\
&> (C - \eta) \left(1 - \tfrac{\eta}{1-\eta}\right) d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}_j), && \text{(4.21f)}
\end{aligned}
$$

where the step from (4.21e) to (4.21f) 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 4.4.3. *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 exist 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 $\forall (\boldsymbol{x}_2, \boldsymbol{y}_2) \in B(\boldsymbol{x}_1, \boldsymbol{y}_1; \delta_x, \delta_y)$

$$
\begin{aligned}
d(\boldsymbol{x}_2, \boldsymbol{y}_2, \boldsymbol{z}; \boldsymbol{\theta}_2) &\leq d^{\max}_{(\delta_x, \delta_y)}(\boldsymbol{\theta}_2) && \text{(4.22a)} \\
&\leq d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \boldsymbol{\theta}_2) + \delta && \text{(4.22b)} \\
&\leq d(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{z}; \boldsymbol{\theta}_1) - a + \delta && \text{(4.22c)} \\
&\leq d^{\max}_{(\delta_x, \delta_y)}(\boldsymbol{\theta}_1) - a + \delta && \text{(4.22d)}
\end{aligned}
$$

where Equation (4.22b) follows from Proposition 4.4.1.

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

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

□

4.5 Extension of SOG to Large Latent Spaces

Algorithm 2 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 3. This modification reduces the computational cost from exponential to linear.

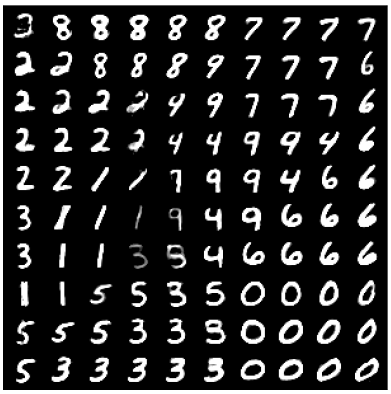
In the FetchReach experiment, the latent dimension is 3 and we adopted Algorithm 3 that resulted in superior results (see Figure 5.5). Besides, in Figures 4.2 to 4.4, we present visual results of Algorithm 3 demonstrating the capability of our method in fitting complex distributions. The axes in Figures 4.2 and 4.3 are the CDFs of the Gaussian latent codes for better visualization. We used the network architecture of the generator in [37].

In this section we argue that Algorithm 2 can be applied to the case where the latent code is a continuous variable with a unit Gaussian prior:

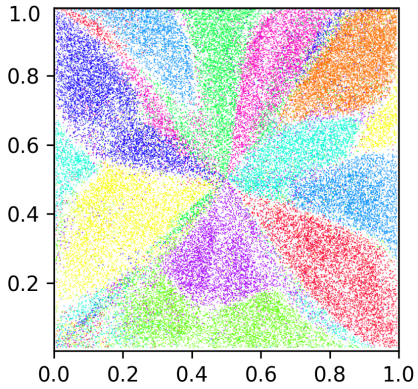
$$
p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I}) \tag{4.23}
$$

Algorithm 3 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



(a) Synthesized MNIST digits



(b) Embedding of MNIST test data in 2 dimensions

Figure 4.2: MNIST dataset. Output results of Algorithm 3 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.

We first note that the following identity holds for all latent code $\boldsymbol{z}$ due to the Bayes' rule.

$$\underbrace{\log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})}_{\text{data log-likelihood}} = \underbrace{\log p(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{x};\boldsymbol{\theta})}_{\text{SOG objective}} - \underbrace{\log \frac{p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y};\boldsymbol{\theta})}{p(\boldsymbol{z})}}_{\text{gap}};\ \forall \boldsymbol{z}. \tag{4.24}$$

The left-hand side of Equation (4.24) is the log-likelihood of the data, and the first term on the right-hand side is the SOG objective function. Therefore, between the data log-likelihood, and the SOG term that is being optimized, there is a gap equal to the second term on the right-hand side.

To analyse this gap, we first realize that the log-likelihood is not a function of $\boldsymbol{z}$, the optimal $\boldsymbol{z}$ in Algorithm 2 also maximizes the gap:

$$\boldsymbol{z}^* = \arg\max_{\boldsymbol{z}} \log p(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{x};\boldsymbol{\theta}) \tag{4.25a}$$

$$= \arg\max_{\boldsymbol{z}} \log \frac{p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y};\boldsymbol{\theta})}{p(\boldsymbol{z})} \tag{4.25b}$$

We further assume that the function f is implemented by a high capacity neural network,

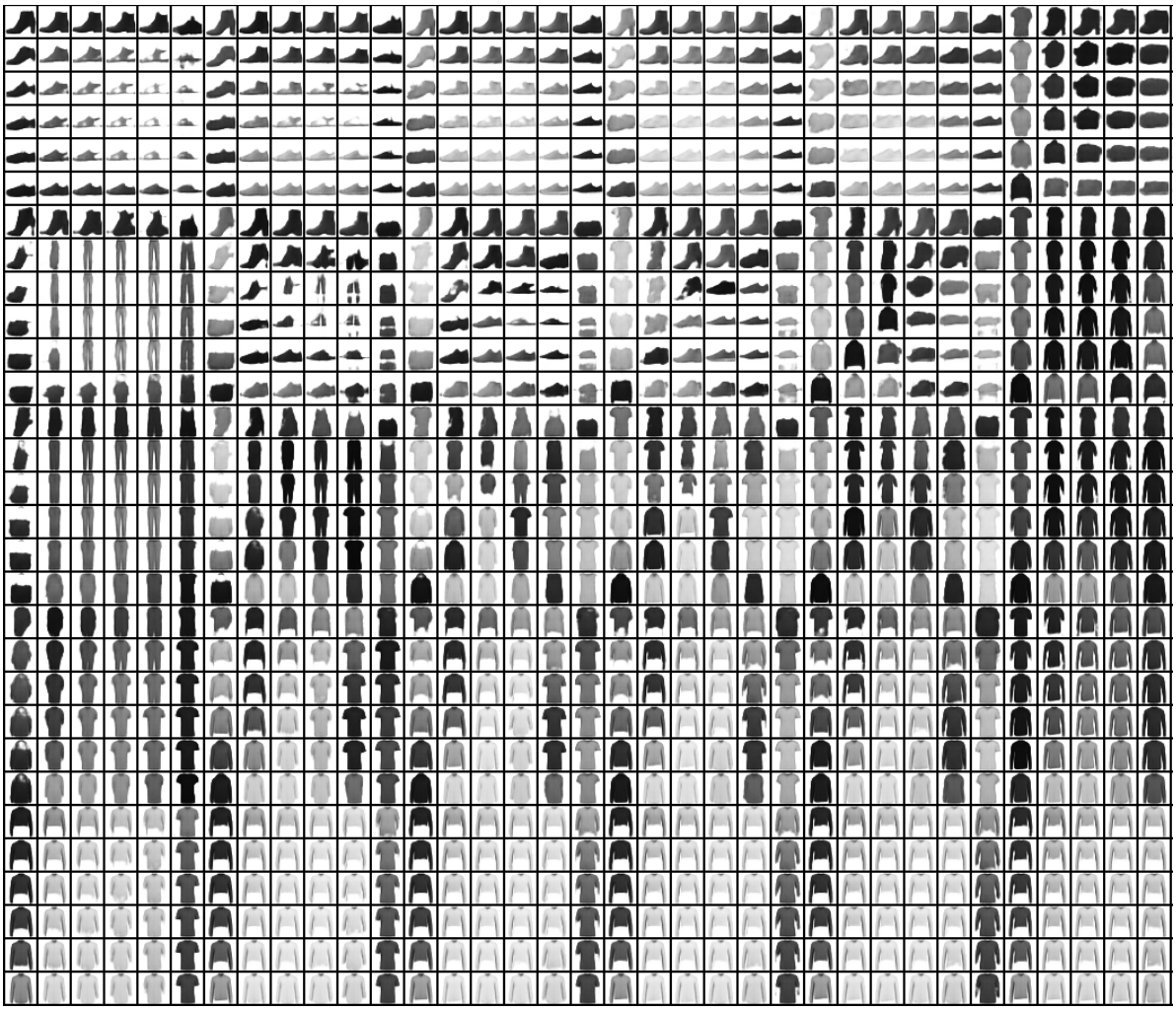


Figure 4.3: Fashion MNIST. Output results of Algorithm 3 for the latent dimension of 8. Each 6×6 block (or block of blocks, etc.) sweeps over one dimension of the latent space. Due to limited space, only part of the entire latent space is plotted. Different latent dimensions have smoothly captured semantically 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 4.4: CelebA. Output results of Algorithm 3 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.

(a) Olivietti ground truth images

(b) Olivietti synthesized images

Figure 4.5: Olivietti faces. Output results of Algorithm 3 for the latent dimension of 3. (a) Ground truth samples. (b) Synthesized samples by SOG.

and as a result, the posterior can take an arbitrary form [24], e.g. a Gaussian:

$$p(\boldsymbol{z}|\boldsymbol{y},\boldsymbol{x},\theta) = \mathcal{N}(\boldsymbol{z};\boldsymbol{\mu}_\theta(\boldsymbol{x},\boldsymbol{y}),\boldsymbol{\Sigma}_\theta(\boldsymbol{x},\boldsymbol{y})), \tag{4.26}$$

Denoting $\boldsymbol{\mu} = \boldsymbol{\mu}_\theta(\boldsymbol{x},\boldsymbol{y})$ and $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_\theta(\boldsymbol{x},\boldsymbol{y})$ to avoid clutter, the optimal solution for Equation (4.25b) can be written as:

$$\begin{aligned}\boldsymbol{z}^* &= \arg\max_{\boldsymbol{z}} -\frac{1}{2}\left[(\boldsymbol{z}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{z}-\boldsymbol{\mu})^T\right] - \frac{1}{2}\log\det\boldsymbol{\Sigma} + \frac{1}{2}\boldsymbol{z}^T\boldsymbol{z} \\ &= \begin{cases} (\boldsymbol{I}-\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}, & \text{if } \boldsymbol{I}-\boldsymbol{\Sigma} \succ 0 \\ \text{unbounded}, & \text{otherwise} \end{cases}\end{aligned} \tag{4.27}$$

Note that since the network is trained to create mutual information between the data points and the unit variance latent code Normal distribution, we expect that the uncertainty in the posterior distribution should be less than in the prior distribution in all directions. Thus the condition $(\boldsymbol{I}-\boldsymbol{\Sigma}) \succ 0$ should hold. Plugging $\boldsymbol{z}^*$ into (4.25b), and rewriting the covariance matrix in the diagonalized form, $\boldsymbol{\Sigma} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ and $\boldsymbol{\Sigma}^{-1} = \boldsymbol{Q}\boldsymbol{\Lambda}^{-1}\boldsymbol{Q}^T$, we obtain:

$$\max_{\boldsymbol{z}} \log\frac{p(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y};\boldsymbol{\theta})}{p(\boldsymbol{z})} \tag{4.28a}$$

$$= -\frac{1}{2}\left[(\boldsymbol{z}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{z}-\boldsymbol{\mu})^T\right]\Big|_{\boldsymbol{z}=\boldsymbol{z}^*} - \frac{1}{2}\log\det\boldsymbol{\Sigma} \tag{4.28b}$$

$$= \frac{1}{2}\left[\boldsymbol{z}^T(\boldsymbol{I}-\boldsymbol{\Sigma}^{-1})\boldsymbol{z} + 2\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{z} - \boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right]\Big|_{\boldsymbol{z}=\boldsymbol{z}^*} - \frac{1}{2}\log\det\boldsymbol{\Sigma} \tag{4.28c}$$

$$= \frac{1}{2}(\boldsymbol{Q}^T\boldsymbol{\mu})^T(\boldsymbol{I}-\boldsymbol{\Lambda})^{-1}(\boldsymbol{Q}^T\boldsymbol{\mu}) - \frac{1}{2}\log\det\boldsymbol{\Sigma} \tag{4.28d}$$

$$= \frac{1}{2}\boldsymbol{\mu}^T\left(\boldsymbol{Q}(\boldsymbol{I}-\boldsymbol{\Lambda})\boldsymbol{Q}^T\right)^{-1}\boldsymbol{\mu} - \frac{1}{2}\log\det\boldsymbol{\Sigma} \tag{4.28e}$$

$$= \frac{1}{2}\boldsymbol{\mu}^T(\boldsymbol{I}-\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu} - \frac{1}{2}\log\det\boldsymbol{\Sigma} \tag{4.28f}$$

$$= g(\boldsymbol{\mu},\boldsymbol{\Sigma}) \tag{4.28g}$$

We now argue that when the optimization in Algorithm 2 reaches the optimum, one can make the following assumptions:

1. Magnitude of $\boldsymbol{\mu}_\theta$ is bounded within a constant range; because the sampling scheme in Algorithm 2 implies optimization of the loss function for values of $\boldsymbol{z}$ that are samples of a unit Gaussian prior.

2. One can easily show that $\boldsymbol{\mu}^T(\boldsymbol{I}-\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu} = \mathrm{Tr}(\boldsymbol{\mu}\boldsymbol{\mu}^T(\boldsymbol{I}-\boldsymbol{\Sigma})^{-1}) \leq \|\boldsymbol{\mu}\|^2\sqrt{\mathrm{Tr}((\boldsymbol{I}-\boldsymbol{\Sigma})^{-2})}$. This uses the inequality that if A and B are positive semidefinite matrices, then $\mathrm{Tr}(AB) \leq \sqrt{\mathrm{Tr}(A^2)\,\mathrm{Tr}(B^2)}$. Now using the spectral decomposition of the covariance matrix,

$$\boldsymbol{\Sigma} = Q\boldsymbol{\Lambda}Q^T, \quad \text{where } Q \text{ is orthogonal,}$$

we get $g(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \leq \frac{1}{2}(c\sqrt{\mathrm{Tr}((I-\boldsymbol{\Lambda})^{-2})} + \log\det\boldsymbol{\Lambda}) = h(\boldsymbol{\Lambda})$. Recall that $\boldsymbol{\Lambda} \leq \boldsymbol{I}$ and with training the elements of $\boldsymbol{\Lambda}$ decrease as the posterior gets sharper. Then one can show that $h(\boldsymbol{\Lambda})$ decreases initially with training when the the elements of $\boldsymbol{\Lambda}$ are not too small. Thus, going back to Equation (4.24) we get that (data log-likelihood) $\geq$ -(SOG-loss + $h(\boldsymbol{\Lambda})$) and with iterations both terms in this lower bound increases initially until the eigenvalues become too small. Thus the log-likelihood of the data is increased as training progreses.

3. Next, assuming no overfitting of the data by the network f (this is possible even with a network with high expressive power), one is guaranteed that the entries of $\boldsymbol{\Sigma}$ are not too small even at convergence, so that $\log\det(\boldsymbol{\Sigma})$ is bounded and $g(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ stays bounded. In particular, at convergence the parameters, θ are not too far from the point that maximizes the data log-likelihood.

Given the assumptions above of Gaussian posterior, we have shown that the Algorithm 2 maximizes the data log-likelihood and by avoiding overfitting it stays close to the maximum point.

4.5.1 Soft Assignment EM

In this section we compare and connect the two variants above, using a toy example to demonstrate that both variants are able to recover the structure (three modes) in the dataset and illustrate the effectiveness and efficiency of SOG (see Figure 4.6).

We first explain the toy example setup. We use a multi-modal linear generative model to generate the dataset of tuples $(\boldsymbol{x}_i, \boldsymbol{y}_i, z_i)$, $i = 1, \ldots, N$. 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.

4.5.1.1 Toy Experiment Results

The experiments on the toy example showed that both variants are able to produce satisfactory fitting results. As we can see in Figure 4.6, 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 4.7 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 4.6, 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 4.8a, 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 4.7). 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 never attain purely one-hot posterior, which is shown as follows.

First we elaborate the intuition behind the observation that the posterior distribution in the EM approach always tends to one-hot. For simplicity, we assume that the three clusters in the toy example are balanced and have the same number of data points. This leads to uniform cluster prior distribution at convergence: $\pi_k^t \to 1/3, k = 1, 2, 3$. The posterior assignment distribution of a data point over the three clusters (Equation (4.5)) is then simply proportional to the conditional likelihood for the three 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.

$$p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = \kappa; \theta^t) > p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = l; \theta^t), \quad \text{if } l \neq \kappa, \tag{4.29}$$

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

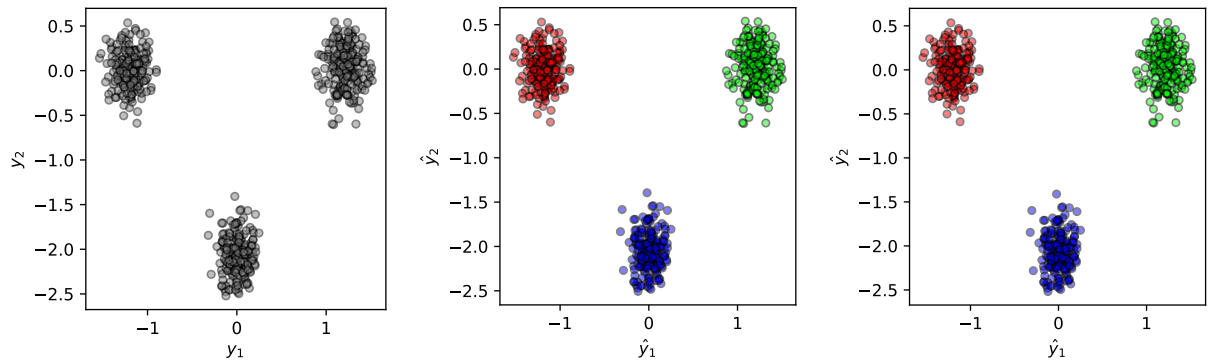
$$r_{i\kappa}^t > r_{il}^t, \quad \text{if } l \neq \kappa,$$

Therefore, the "winning" cluster gains more weight in the loss function, and tends to be improved more than the alternatives, so that Equation (4.29) 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 $r_{i\kappa}^t \approx 1$, while $r_{il}^t \approx 0$ for $l \neq \kappa$.

Now we show that this EM posterior can at most approach one-hot with a finite gap when the likelihood function is assumed to be Gaussian, i.e. $p(\boldsymbol{y}_i \,|\, \boldsymbol{x}_i, z) = \mathcal{N}(\boldsymbol{y}; f_\theta(z, \boldsymbol{x}), \sigma^2 \boldsymbol{I})$, 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. $r_{il}^t > 0$ 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



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

Figure 4.6: The three-mode 2D clustering example: ground truth and model predictions at convergence. Points are colored-coded with the latent code assignments from the 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

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

Now that the r_{il}^t 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 $r_{il}^t = 0$ for $l \neq \kappa$. This is confirmed by the plot Figure 4.8b 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 $p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, z = \kappa \ ; \ \theta)$.

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 4.8c.

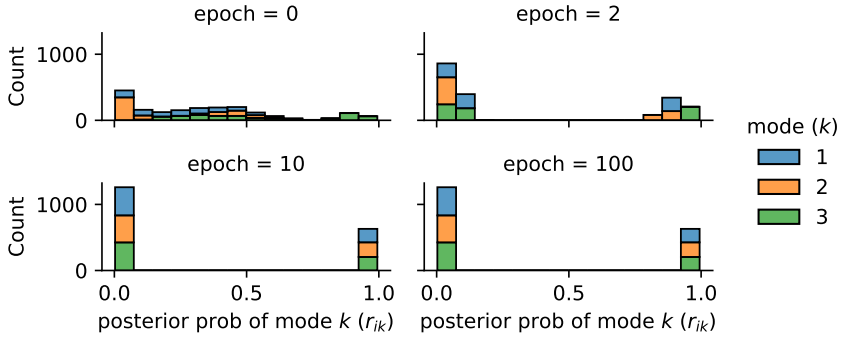
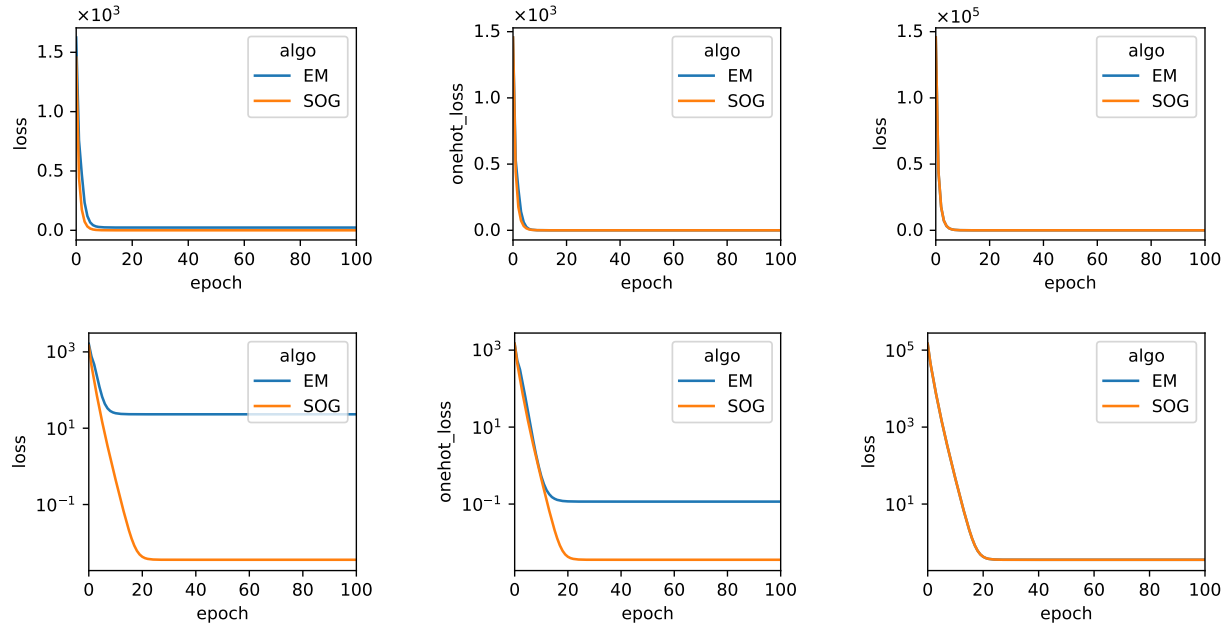


Figure 4.7: The posterior distribution from the EM algorithm along its training



(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 4.8: The training curve of the EM algorithm vs that of SOG.

4.5.2 Hard Assignment EM (the SOG Algorithm)

Motivated by the observation in 4.5.1, one can conjecture that the optimization can be performed directly for one-hot latent codes. We shall make an assumption about the generative model that $\pi_k = 1/K;\ k = 1, \ldots, K$, and that we can replace the probabilities r_{ik} in Equation (4.5) with binary values. These assumptions are closely connected to those made in the K-means algorithm [3]. Therefore, the E step becomes

$$\gamma_{ik}^t = p(\boldsymbol{z} = k \mid \boldsymbol{x}_i, \boldsymbol{y}_i;\ \boldsymbol{\theta}^t, \boldsymbol{\Pi}^t) = \begin{cases} 1, & k = \arg\min_k ||f(\boldsymbol{z}_k, \boldsymbol{x}_i;\ \boldsymbol{\theta}) - \boldsymbol{y}_i|| \\ 0, & \text{otherwise} \end{cases}, \tag{4.30}$$

and the M step of Equation (4.4c) (w.r.t $\boldsymbol{\theta}$) simplifies to:

$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \min_k ||f(\boldsymbol{z} = k, \boldsymbol{x}; \boldsymbol{\theta}) - \boldsymbol{y}||^2; \tag{4.31}$$

which is equivalent to the update step in Algorithm 2.

4.5.3 Disentangled Representations

In deep learning, a major challenge is to understand high dimensional data and extract useful information in an unsupervised manner. A line of work has been dedicated to recover these rich representations in a disentangled way. This means that the learned model should be able to capture the independent features of a datapoint, i.e. an image, in such a way that if one feature changes, the others remain unaffected. Below we briefly review some of the existing ideas in this domain. The main challenge that these works tried to solve has been to learn generating shapes, such that different coordinates of the latent space change different aspects of the images such as type, rotation, etc. in a non-covariant way.

Beta VAE

The objective of a variational autoencoder can be seen as an autoencoding L_2 objective plus a regularization which matches the posterior distribution, in the KL divergence sense, to a

Gaussian distribution (see Section 3.1.3). Beta-VAE [17] increases the amount of the penalty of the regularization term through adjusting the factor β.

$$\frac{1}{N}\sum_{i=1}^{N}\left[\mathbb{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}|z)\right]-\beta\,\mathrm{KL}\left(q\left(z|x^{(i)}\right)||p(z)\right)\right] \tag{4.32}$$

Since the prior Gaussian has a diagonal covariant matrix, increasing β encourages the posterior to have a factorized distribution. Parameter β trades off between disentanglement and reconstruction.

Factor VAE

Factor VAE [23] adopts a more direct method for encouraging disentanglement of the distribution of the latent variable. The objective of Factor VAE is

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}[\mathbb{E}_{q(z|x^{(i)})}\left[\log p(x^{(i)}|z)\right]-\ &\mathrm{KL}(q(z|x^{(i)})||p(z))]\\ &-\gamma\mathrm{KL}(q(z)||\bar{q}(z)),\end{aligned} \tag{4.33}$$

where $\bar{q}(z) \coloneqq \prod_{j=1}^{d} q(z_j)$, known as total correlation. To calculate the total correlation efficiently, Factor VAE trains a discriminator network (see Section 3.1.1) to predict whether a sample of z is drawn from the distribution $q(z)$ or $\bar{q}(z)$. In this way, the objective of FactorVAE can be approximated as

$$\begin{aligned}\mathrm{KL}(q(z)||\bar{q}(z)) &= \mathbb{E}_{q(z)}\left[\log\frac{q(z)}{\bar{q}(z)}\right]\\ &\approx \mathbb{E}_{q(z)}\left[\log\frac{D(z)}{1-D(z)}\right]\end{aligned} \tag{4.34}$$

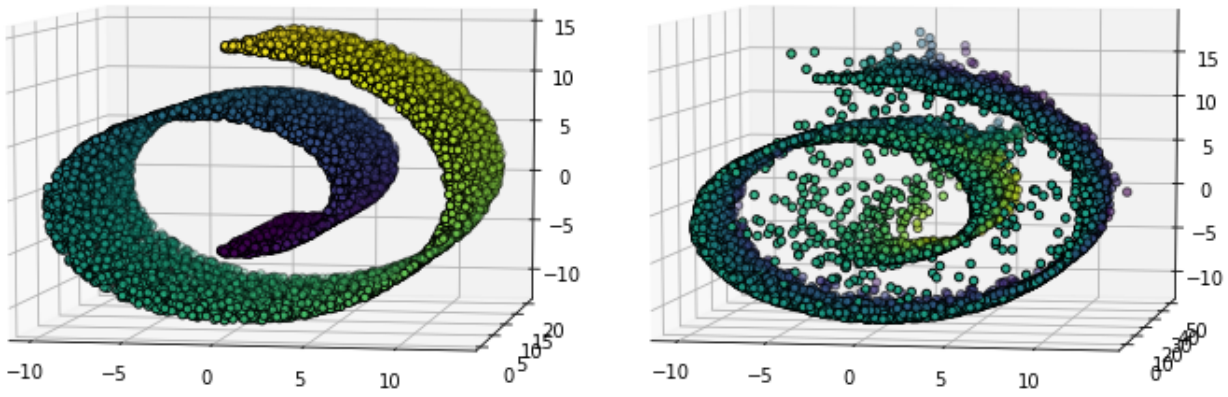
InfoGAN

InfoGAN [8] aims to encourage the mutual information between the latent variable and the synthesized data. InfoGAN obtains a lower bound on the mutual information by defining an

auxilliary distribution $Q(c|x)$ that approximates $P(c|x)$,

$$\begin{aligned} I(c; G(z,c)) &= H(c) - H(c|G(z,c)) \\ &= \mathbb{E}_{x\sim G(z,c)} \left[\mathbb{E}_{c'\sim P(c|x)} \left[\log P(c'|x) \right] \right] + H(c) \\ &= \mathbb{E}_{x\sim G(z,c)} \left[\mathbb{E}_{c'\sim P(c|x)} \left[D_{\mathrm{KL}}(P(\cdot|x) \,||\, Q(\cdot|x)) \right] + \mathbb{E}_{c'\sim P(c|x)} \left[\log Q(c'|x) \right] \right] + H(c) \\ &\geq \mathbb{E}_{x\sim G(z,c)} \left[\mathbb{E}_{c'\sim P(c|x)} \left[\log Q(c'|x) \right] \right] + H(c) \end{aligned} \tag{4.35}$$

where the last inequality holds due to the fact that KL divergence is non-negative. This loss function is added to the adversarial optimization for better correlation of the latent variable, which is a factorial unit Gaussian, with the synthesized data. This paradigm is also adopted in the context of adversarial imitation learning. We will mention those applications in more details in Chapter 5.



(a) Original dataset

(b) Reconstruction by SOG

Figure 4.9: Manifold of the "swiss roll" dataset learned by SOG. The latent dimension used has been two. This figure indicates that SOG is capable of appropriate mapping of the information from a low dimensional manifold in a higher dimensional space, into a more compact latent variable.

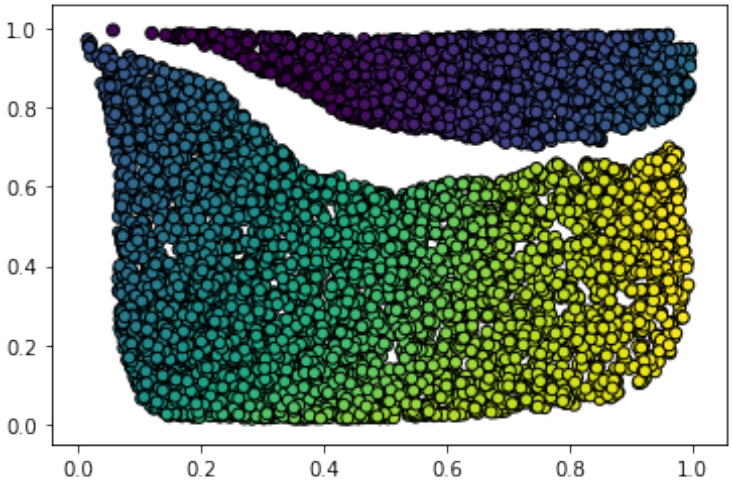


Figure 4.10: Embedding of the points in the latent space for the swiss roll dataset. Coloring of the points match those of Section 4.5.3. The two-dimensional manifold has been mapped from three dimensions to two dimensions. We observe that the manifold is split into two parts which shows that the algorithm may not always perform the mapping in the most intuitive way.

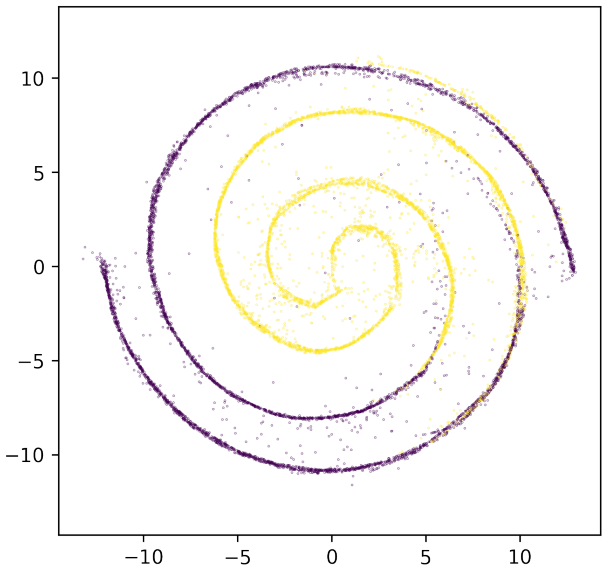


Figure 4.11: Embedding of the points in the latent space for the "two moons" dataset. A consinuous latent variable along with one discrete binary variable. Color coding is according to the recovered binary variable. The one-dimensional manifold has been mapped from two dimensions to one dimension (plus a binary variable). As humans, we expect the binary variable to be assigned to each of the spiral halves. However, in the loss function nothing encourags that behavior. Therefore, we observe that the binary variable inner spiral vs the outer spiral. Again, this shows that the algorithm may not always perform the mapping in the most intuitive way.

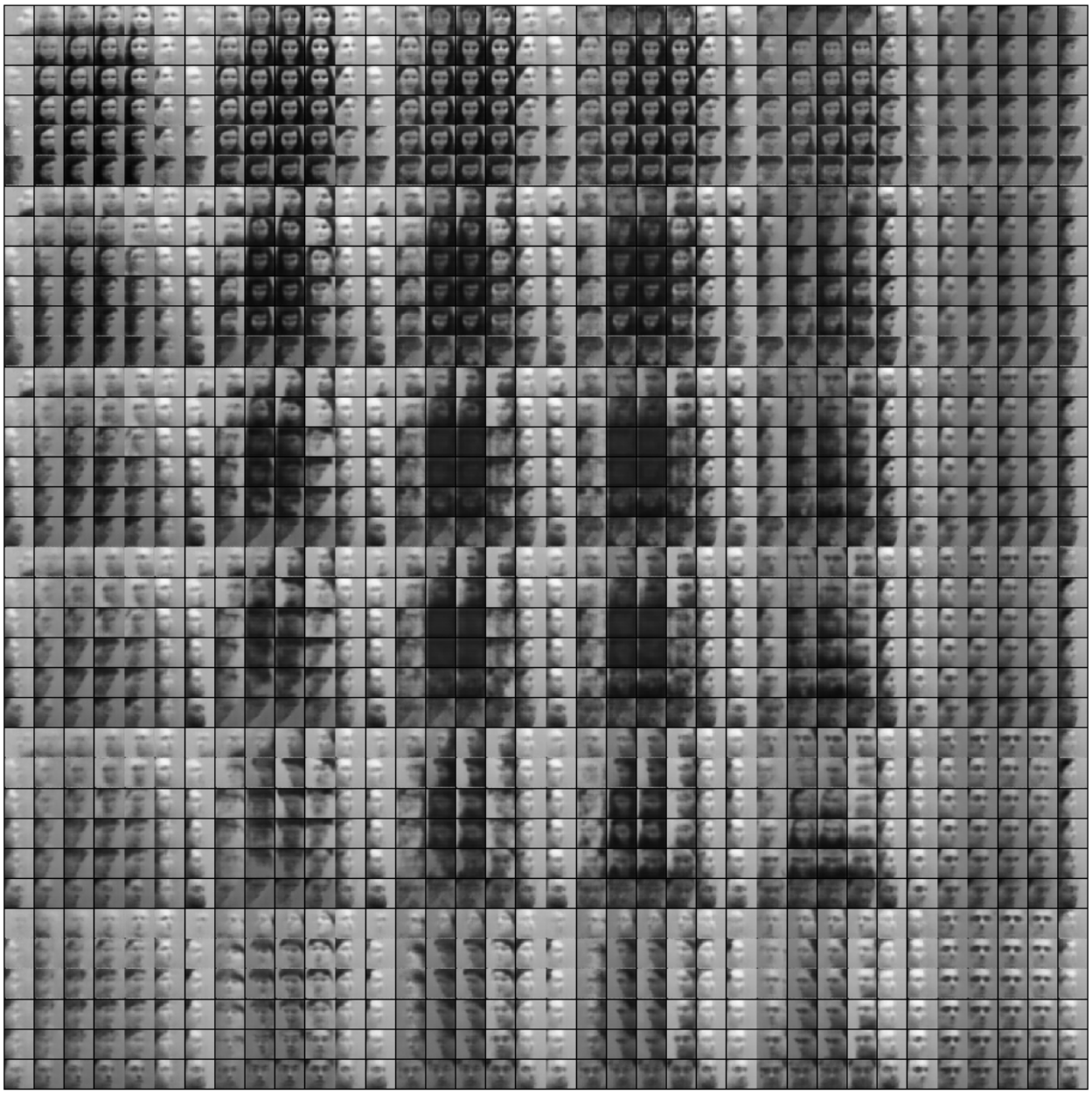


Figure 4.12: Sample images from QMUL faces dataset. The latent dimension used is four. One can notice a six-by-six grid of six-by-six grids, covering different corners of the four-dimensional latent space. This example demonstrate a clear disentanglement in the representation. As each latent variable is sweeped, different aspects of the image, e.g. face type, and yaw and pitch of the head are variaed. Also lighter images and darker images (more hair) have been separated.

CHAPTER 5

Diverse Imitation Learning via SOG

We discussed in Section 3.2 that 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) [36], 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 [41, 42], 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) [54], recovers a reward function that makes the expert policy optimal. Such an approach often requires iterative calculation of the optimal policy with reinforcement learning (RL) in an inner loop, making it computationally expensive. The computational cost can be reduced by jointly optimizing a policy—via RL, and a cost function—via maximum causal entropy IRL [54, 53]. The resulting algorithm is called generative adversarial imitation learning (GAIL) [18], and is closely connected to generative adversarial networks [15]. 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 [29, 51].

Imitation of diverse behaviors is addressed in several prior works. [13, 31] use labels of expert modes in training. These works differ from ours in that we assume expert modes are unlabeled. InfoGAIL [29] and Intention-GAN [16] 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. [51] 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.

Multimodal imitation learning requires finding the correspondence between expert trajectories and a latent code. [51] suggest learning this correspondence using the encoder in a VAE. 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.

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

1. We adopt SOG as an encoder-free generative model that can be used for multimodal BC. We show that SOG procedure is closely connected to maximizing the marginal likelihood of data. Besides, we introduce an extension of 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.

5.1 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: $\boldsymbol{z} \sim p(\boldsymbol{z}), \boldsymbol{s}_0 \sim \rho_0, \boldsymbol{a}_t \sim \pi_E(\boldsymbol{a}_t|\boldsymbol{s}_t, \boldsymbol{z}), \boldsymbol{s}_{t+1} \sim P(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t, \boldsymbol{s}_t)$.

5.2 Imitation Learning Algorithms

In this section, we present the imitation learning algorithms described in Section 4.1.2.

Algorithm 4 SOG-BC: Multimodal Behavior Cloning with SOG

Define: Loss function $\mathcal{L}(\hat{\boldsymbol{a}}, \boldsymbol{a}) \coloneqq ||\hat{\boldsymbol{a}} - \boldsymbol{a}||^2$

Input: Initial parameters of policy network $\boldsymbol{\theta}_0$; expert trajectories $\tau_E \sim \pi_E$

for *iteration* $= 0, 1, 2, \ldots$ **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 $\boldsymbol{z}_E \coloneqq \arg\min_{\boldsymbol{z}_i} \hat{\mathbb{E}}_{\tau_E} [\mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{z}_i, \boldsymbol{s}), \boldsymbol{a})]$.

  Calculate $\mathcal{L}_{\text{SOG}} = \hat{\mathbb{E}}_{\tau_E} [\mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{z}_E, \boldsymbol{s}), \boldsymbol{a})]$ 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

Algorithm 5 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 $i = 0, 1, 2, \ldots$ **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 $\mathcal{L}_{\text{PPO}}$ 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 $\mathcal{L}_{\text{SOG}}$ per Algorithm 4.

  Take a policy step from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_{i+1}$ w.r.t. the objective $\mathcal{L}_{\text{PPO}} + \lambda_S \mathcal{L}_{\text{SOG}}$.

end for

5.3 Experiments

In our experiments, we demonstrate that our method can recover and distinguish all modes of expert 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 [29] and VAE-GAIL [51].

5.3.1 Experiment Setup

We adopt an experiment from [29] 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 [47]. These experiments, borrowed from [38], 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. 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 [35]. 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.

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 [38], a few-shot reinforcement learning algorithm. For the FetchReach experiment, we considered a pretrained DDPG+HER model as expert [30, 2].

In Section 5.3, 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 [38]. 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 5.1. As a reference, HalfCheetah has a torso length of around 1 m.

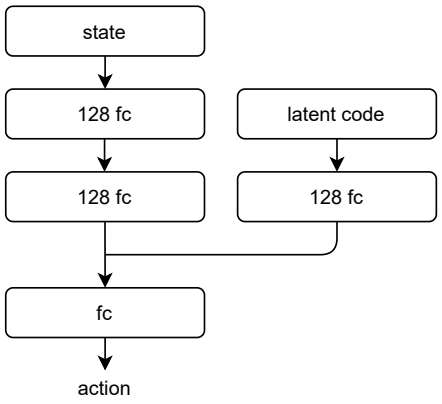
Table 5.1: Set of possible expert velocities in different environments.

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 $[1.5, 3]$

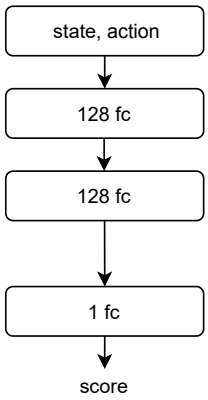
Episode lengths in different environments are listed in Table 5.2.

Table 5.2: Episode lengths in different environments.

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



(a) Policy network



(b) The discriminator

Figure 5.1: Neural network architectures. (a) the policy network, (b) the discriminator in GAIL.

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 final softmax layer in the discrete case. For continuous latent codes, InfoGAIL adopts a Gaussian posterior. The architectures are illustrated in Figure 5.1.

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 4. 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 5), at values of $\lambda = i{*}10^j;\ i = 1, 2, 5;\ 0.01 \leq \lambda \leq 1$.

5.3.2 Evaluation

Table 5.3: 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	1333.9 ± 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	$\mathbf{2222.3 \pm 431.1}$	2015.3 ± 547.3	1216.8 ± 70.8	1065.8 ± 30.1	2229.7 ± 438.6

Visualization

In Figure 5.2, we visualize the generated trajectories of four locomotion tasks with discrete modes. In this figure, we observe planar displacements of locomotion agents moving towards multiple directions. We have color coded these plots according to the different latent codes at inference time. The number of modes in the experiments are therefore as many as the number of colors. To better show the full behavior of the agent, for some experiments we have shown multiple trajectories per latent code. 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 5.4, 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.

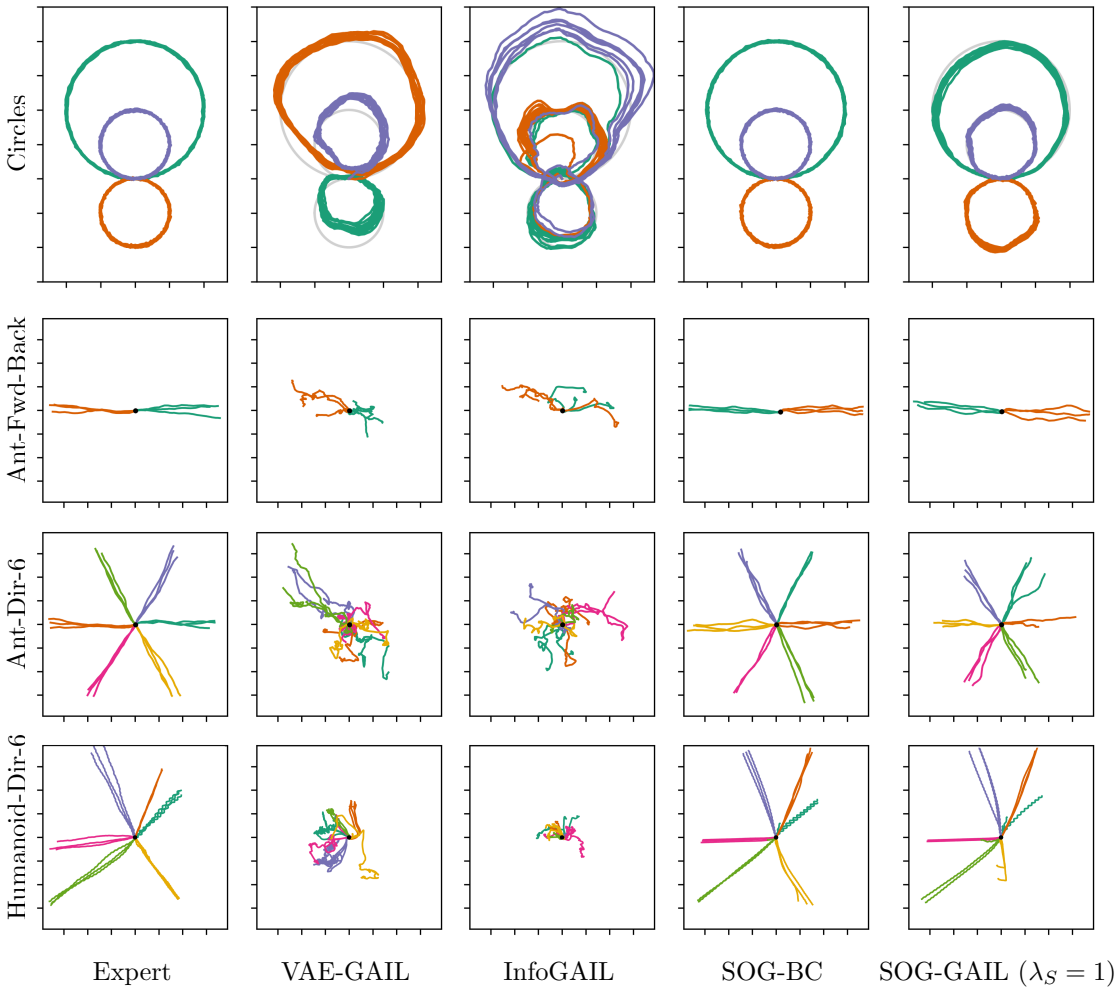


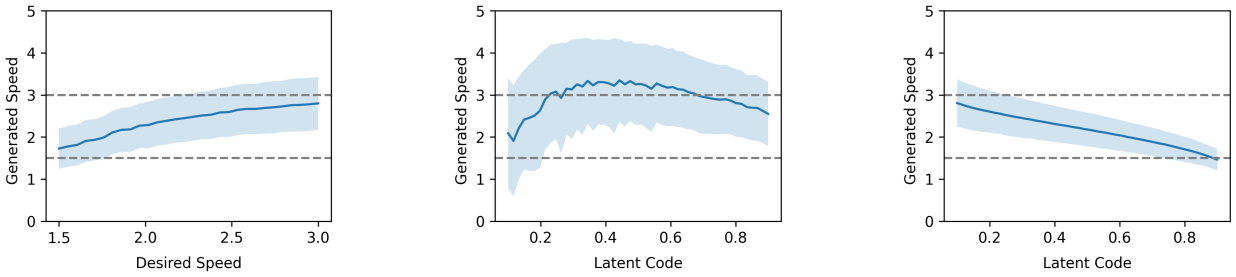
Figure 5.2: Discrete latent variable: locomotion towards multiple directions. Trajectories of the imitated policies for four tasks. Different latent codes at inference time are color coded. From top to bottom: number of modes are 3, 2, 6, 6; number of trajectories per latent code are 1, 3, 3, 3. Both 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.

Table 5.4: 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	InfoGAIL	VAE-GAIL	Expert
Entropy (nats)	**2.05**	1.89	0.27	0.85	2.18
Hit Rate	**100**%	97.0%	N/A	18.6%	100%

Table 5.5: 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	InfoGAIL	VAE-GAIL
1.584	1.431	0.145	0.750



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

Figure 5.3: 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.

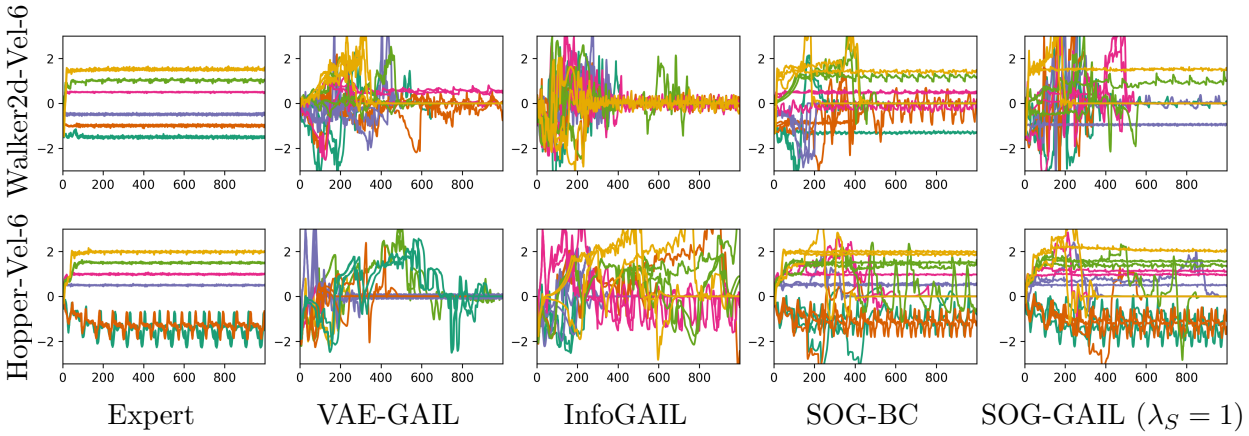
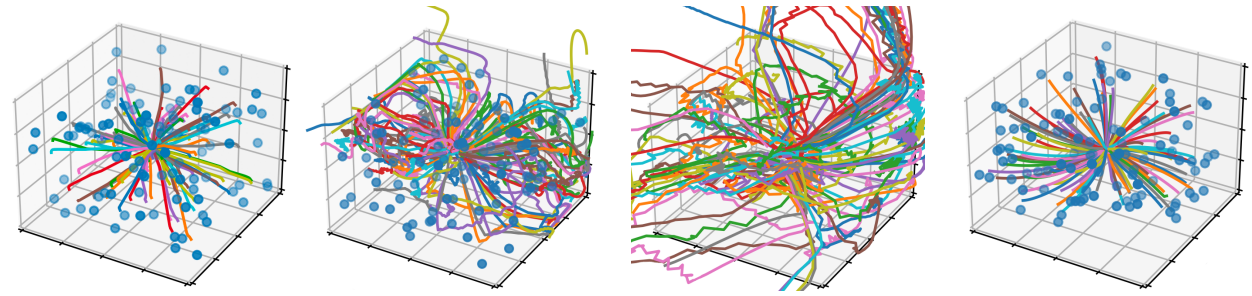


Figure 5.4: Discrete latent variable: locomotion at six velocities. Velocities generated by the policies vs rollout time steps. Three rollouts per mode are visualized. Each mode is marked with a distinct color. Both SOG-BC and SOG-GAIL separate and imitate most of the modes, producing the desired velocities. VAE-GAIL and InfoGAIL exhibit an inferior performance.



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

Figure 5.5: 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.

In Figures 5.3, 5.5 and 5.9, 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 5.5, 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 5.9, we plot the reached targets for different choices of the latent code, and observe a semantically meaningful interpolation in the latent space. Figure 5.6 demonstrates how the robotic arm has been able to reach different sides of the space in action.

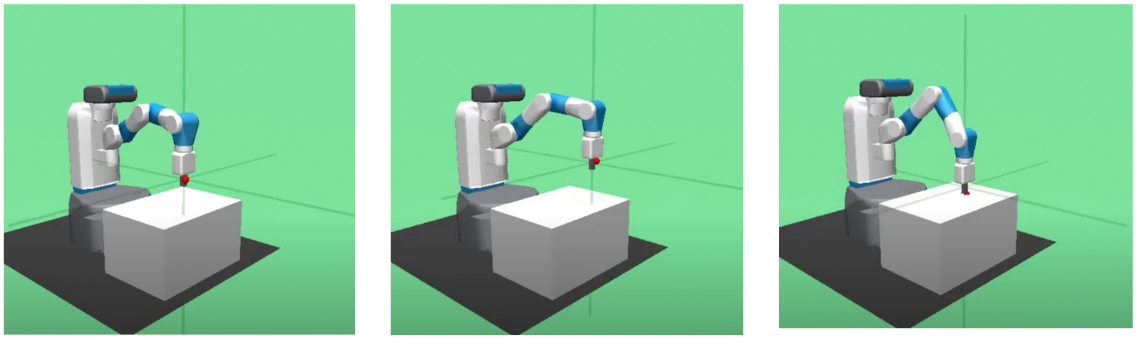


Figure 5.6: The fetch agent trained with our model capable of reaching different targets

In Figure 5.3, 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.

Metrics

In Table 5.3, we use ground-truth reward functions to evaluate the collected rewards under different policies. 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 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. In particular, in locomotion tasks, a main part of the reward functions involves moving in a particular direction. If modes are not properly assigned, the agent may receive low or negative rewards if it moves towards wrong directions. 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 5.4, 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 [26]. 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 [27]. The results are presented in Table 5.5.

Robustness

We design additional experiments to show that integration of SOG with GAIL in Algorithm 5 makes the learned policies robust towards unseen states. Observing space limits, we present the results in Section 5.3.3.

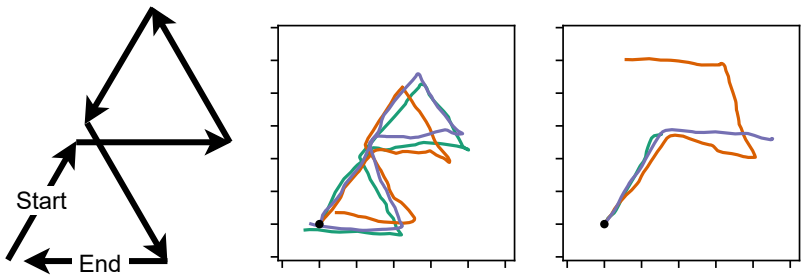
5.3.3 Robustness Analysis

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 agent 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 in several experiments. High values of λ_S in limit lead to SOG-BC. We observe that increasing λ_S improves scores in Table 5.3 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 5.7 and 5.8.

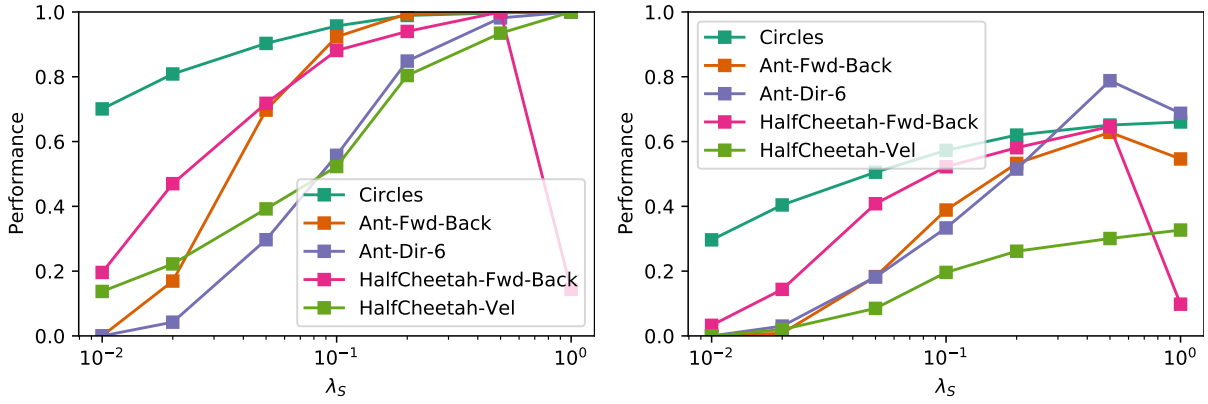
5.3.4 Latent Space Interpretability

In Figure 5.9 we demonstrate that in the Fetch experiment, different areas of the latent space smoothly correspond to various target points.



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

Figure 5.7: 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 5.8: Optimal choice of $\boldsymbol{\lambda_S}$. Parameter λ_S is the coefficient of the SOG loss in Algorithm 5. Performance of Algorithm 5 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 .

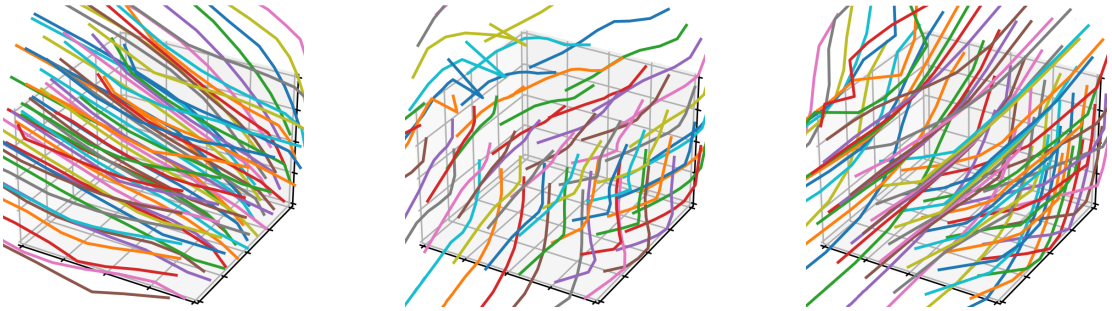


Figure 5.9: Self-organization in the latent space: FetchReach. In each plot, two dimensions among the three dimensions of the latent space are fixed, whereas the third is varied. Each line corresponds to the location of the robotic arm at the terminal state of different trajectories. In each line, two latent dimensions take a fixed value, while the third varies. Lines are randomly colored for better visual distinction.

CHAPTER 6

Concluding Remarks and Future Work

In this dissertation, we focus on improving the ability of machine learning algorithms to learn multiple patterns in a single training process. We recognize that one of the main ways humans can learn multiple modalities of a concept is to get help from conditioning the learning on some form of "context". Making inferences from real-world data in many domains of applications requires such a context-based treatment. This thesis proposes a robust, efficient, automated, and general-purpose method for contextual inference. Our method is based on searching for the best context and conditioning the learning on the best existing context. In this fashion, different data modes start to group naturally and learn concurrently. This approach is inspired by the well-known Expectation-Maximization (EM) paradigm. EM typically involves two stages finding the best mode and updating parameters for the best recovered modes. Therefore, EM constitutes the essence of our proposed algorithm. We demonstrated that our proposed method called "Self-Organizing Generative Models" (SOG), connects closely to EM, thus performing Maximum Likelihood Estimation. The naming of the algorithm pays tribute to the classical work of Self-Organizing Maps, which can be thought of as a simplified form of our Deep Learning framework. We also showed that our method organizes embedding of similar data points close to each other under reasonable assumptions about the learned neural network. Lastly, another contribution of this thesis about the base SOG model is a greedy method for high-dimensional search, which makes it possible to learn high-dimensional data. We showed that this model could successfully learn high-dimensional image data through this trick.

Our method was proved as a powerful means to learn multiple modes of data in various

domains of machine learning research. One such domain is multimodal imitation learning, which involves mimicking multiple sequential behaviors without a knowledge of the underlying reward function. This task has been particularly challenging for the existing literature. However, our method was able to achieve a far superior performance in numerous simulated environments. We verified the efficacy of applying SOG to multimodal imitation learning through various plots, videos, and multimodal performance metrics.

We believe that the SOG model can serve as a general-purpose method for multimodal inference, therefore having the potential to influence multiple directions of research. For instance, in natural language processing, the SOG model can help in the disambiguation of semantics. Besides, computer vision can facilitate applications where multiple modes of synthesis can be considered. Similar situations where contextual clues help in learning to map from input data to output data suggest future follow-up research on the contribution of this thesis.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. "Apprenticeship learning via inverse reinforcement learning". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.

[2] Marcin Andrychowicz et al. "Hindsight experience replay". In: *arXiv preprint arXiv:1707.01495* (2017).

[3] Christopher M Bishop. *Pattern Recognition and Machine Learning*. springer, 2006. Chap. 9.

[4] Piotr Bojanowski et al. "Optimizing the Latent Space of Generative Networks". In: *arXiv preprint arXiv:1707.05776* (2017).

[5] John Canny. "A computational approach to edge detection". In: *Readings in computer vision*. Elsevier, 1987, pp. 184–203.

[6] Caroline Chan et al. "Everybody dance now". In: *arXiv preprint arXiv:1808.07371* (2018).

[7] Lichao Chen et al. "Brain-inspired automated visual object discovery and detection". In: *Proceedings of the National Academy of Sciences* 116.1 (2019), pp. 96–105.

[8] Xi Chen et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets". In: *Advances in neural information processing systems* 29 (2016).

[9] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[10] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[11] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using real nvp". In: *arXiv preprint arXiv:1605.08803* (2016).

[12] Carl Doersch. "Tutorial on variational autoencoders". In: *arXiv preprint arXiv:1606.05908* (2016).

[13] Cong Fei et al. "Triple-GAIL: a Multi-Modal Imitation Learning Framework with Generative Adversarial Nets". In: *arXiv preprint arXiv:2005.10622* (2020).

[14] Pedro F Felzenszwalb et al. "Object detection with discriminatively trained part-based models". In: *IEEE transactions on pattern analysis and machine intelligence* 32.9 (2009), pp. 1627–1645.

[15] Ian J Goodfellow et al. "Generative Adversarial Networks". In: *arXiv preprint arXiv:1406.2661* (2014).

[16] Karol Hausman et al. "Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets". In: *arXiv preprint arXiv:1705.10479* (2017).

[17] Irina Higgins et al. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: (2016).

[18] Jonathan Ho and Stefano Ermon. "Generative Adversarial Imitation Learning". In: *arXiv preprint arXiv:1606.03476* (2016).

[19] 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*. 2019, pp. 5811–5819.

[20] Phillip Isola et al. "Image-to-image translation with conditional adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[21] Rohit Jena, Changliu Liu, and Katia Sycara. "Augmenting GAIL with BC for sample efficient imitation learning". In: *arXiv preprint arXiv:2001.07798* (2020).

[22] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution". In: *European conference on computer vision*. Springer. 2016, pp. 694–711.

[23] Hyunjik Kim and Andriy Mnih. "Disentangling by factorising". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2649–2658.

[24] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[25] Durk P Kingma and Prafulla Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions". In: *Advances in neural information processing systems* 31 (2018).

[26] LF Kozachenko and Nikolai N Leonenko. "Sample estimate of the entropy of a random vector". In: *Problemy Peredachi Informatsii* 23.2 (1987), pp. 9–16.

[27] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. "Estimating Mutual Information". In: *Physical review E* 69.6 (2004), p. 066138.

[28] Christian Ledig et al. "Photo-realistic single image super-resolution using a generative adversarial network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4681–4690.

[29] Yunzhu Li, Jiaming Song, and Stefano Ermon. "InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations". In: *arXiv preprint arXiv:1703.08840* (2017).

[30] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[31] Josh Merel et al. "Learning human behaviors from motion capture by adversarial imitation". In: *arXiv preprint arXiv:1707.02201* (2017).

[32] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[33] Kamyar Nazeri et al. “Edgeconnect: Generative image inpainting with adversarial edge learning”. In: *arXiv preprint arXiv:1901.00212* (2019).

[34] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[35] Matthias Plappert et al. “Multi-goal reinforcement learning: Challenging robotics environments and request for research”. In: *arXiv preprint arXiv:1802.09464* (2018).

[36] Dean A Pomerleau. “Efficient Training of Artificial Neural Networks for Autonomous Navigation”. In: *Neural computation* 3.1 (1991), pp. 88–97.

[37] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[38] Kate Rakelly et al. “Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables”. In: *International conference on machine learning*. PMLR. 2019, pp. 5331–5340.

[39] Stephan R Richter et al. “Playing for data: Ground truth from computer games”. In: *European conference on computer vision*. Springer. 2016, pp. 102–118.

[40] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

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

[42] 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*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[43] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[44] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[45] John Schulman et al. "Trust Region Policy Optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[46] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12 (1999).

[47] 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*. IEEE. 2012, pp. 5026–5033.

[48] A Van den Oord et al. "Wavenet: a generative model for raw audio. arXiv". In: *arXiv preprint arXiv:1609.03499* (2016).

[49] 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*. 2016, pp. 98–106.

[50] Ting-Chun Wang et al. "High-resolution image synthesis and semantic manipulation with conditional gans". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8798–8807.

[51] Ziyu Wang et al. "Robust Imitation of Diverse Behaviors". In: *arXiv preprint arXiv:1707.02747* (2017).

[52] Jun-Yan Zhu et al. “Toward multimodal image-to-image translation”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 465–476.

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

[54] Brian D Ziebart et al. “Maximum Entropy Inverse Reinforcement Learning.” In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.