THE UNIVERSITY OF CHICAGO

DEEP GENERATIVE MODELS: DESIGN, IMPROVEMENTS AND APPLICATIONS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF STATISTICS

BY

QING YAN

CHICAGO, ILLINOIS

JUNE 2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to acknowledge several people who helped me create this work.

First, I would like to express my deepest gratitude to my advisor, prof. Yali Amit, for accepting me as his Ph.D. student at the University of Chicago and guiding me through my graduate study and research. Yali allowed me to study, explore, and develop new projects in my areas of interest and supported me in many aspects. Our frequent presentations and discussions were extremely important for me to learn new knowledge and build new ideas.

The colleague I interacted with most during my time in UChicago was Dr. Zhisheng Xiao. I would like to thank Zhisheng for his help in my study, research, and life. I really enjoyed the time we discussed new ideas, worked on new papers, and the hard-core travels these years.

I would like to thank other professors in the department of statistics, CAM, and TTIC for their helpful courses and seminars. Also, I would like to thank other staff in my department for the administrative help, and the computation resources support. Besides, I'd like to thank my other graduate classmates and my former and current roommates for their help in study and life.

Most importantly, I would like to thank my family for the most essential support to my life and study.

ABSTRACT

Deep generative models (DGM) combine the deep neural networks with generative models to learn the underlying generation mechanism of the data of interest. This has emerged as a important approach to extracting knowledge from data in machine learning and artificial intelligence. However, there are many challenges to learning and applying DGMs in different domains despite their promising potential. Therefore, this thesis focuses on understanding, improving, and applying different deep generative models.

First, we introduce the underlying principles under different DGMs, including the Variational Auto-Encoder (VAE), the Flow-Based Model, the Generative Adversarial Network (GAN), and the Energy-Based Model (EBM). We also propose a novel counterpart of VAEs: Variational Latent Optimization (VLO), which does not require an encoder structure. Besides, we provide a new angle to understand the generation process of EBMs, build a connection between EBMs and GANs, and design a new approach to improve the sample quality of EBMs from that.

Next, we propose two hybrids of DGMs to improve the generation quality of current models. First, we combine Flow-based models and Variational Auto-Encoders to improve the generation quality of Auto-Encoder-based generative models. Second, we borrow the idea of exponential tilting and combine the energy-based model with other likelihood-based generative models to obtain better samples.

In the end, we conduct various applications related to modern deep generative models, including using generative models as likelihood-based methods for out-of-distribution (OOD) detection and designing controllable generative models over human faces. We propose a novel OOD-detect score called likelihood regret to help detect OOD samples with VAEs. Also, we propose to add a new structure to current key-points-based face reenactment models and combine them with the 3D morphable model to improve their generation quality and generalization ability.

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Machine Learning and AI

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and data [Mitchell et al., 1997]. ML algorithms build a model based on sample data, known as training data, to make predictions or decisions without being explicitly programmed to do so. It is seen as part of the domain of artificial intelligence (AI), which aims to develop intelligence demonstrated by machines to mimic natural intelligence displayed by animals, including humans. Although the discussion of these terms can be traced back to the 1960s, they have become more popular and essential for applications nowadays. We rely on ML algorithms and models in various domains, such as computer vision, natural language processing, economics, financial analysis, Bioinformatics, Climate science, and game strategies. Therefore, exploring new algorithms and models in different areas and branches is significantly meaningful. Particularly, generative models create a promising direction to guide the machine in building knowledge from the treasure of big data, which also is the main focus of this thesis.

1.2 Generative v.s. Discriminative Models

The idea behind the generative model is that if vast amounts of data are collected from a specific domain, we may want to build a model that can generate similar (but not the same) data. This intuition is also expressed in a famous quote from Richard Feynman.

> "What I cannot create, I do not understand."
>
> —Richard Feynman

Unlike its counterpart, the **discriminative model**, which has a clear target and aims to make predictions, a generative model is designed for more general purposes. Even for the ultimate goal of "generating similar data", it is unclear how such similarity is defined. This gives us more freedom to design and evaluate generative models. As suggested, generative models are usually learned in an unsupervised fashion without any information other than the data itself, whereas discriminative models often require additional labels to compute the learning objective. It is straightforward to train a discriminative model if the matched dataset is provided, and a well-trained discriminative model has good performance in its corresponding area but usually cannot directly generalize to other problems. In contrast, as it learns the underlying generation mechanism, a generative model is generally harder to train, but with the reward that it is straightforward to re-use for other downstream tasks, including image classification [He et al., 2021], language modeling [Devlin et al., 2018] and time series anomaly detection[Li et al., 2018]. The general idea is to learn an unsupervised representation of the data using the generative model and then train the downstream task based on the learned representation. These works revealed many advantages when using the learned features on the downstream tasks against the raw data, including training efficiency, final accuracy, and generalization ability. Therefore, generative models constitute one of the foundations of teaching a machine how to extract knowledge from data.

1.3 Probabilistic Models and Deep Neural Networks

Viewing our world in a probabilistic way provides a natural approach to defining the generation process. More specifically, given a collection of data $\{x_i \in \mathcal{X}, i = 1, 2 \cdots, n\}$, we assume they are independent and identically distributed (i.i.d) from the underlying distribution $p(x)$. If the model captures this distribution, we can generate new data through sampling.

Following this probabilistic intuition, **deep generative models (DGMs)** apply deep neural networks to learn and mimic the underlying distribution. Deep learning models have

shown tremendous potential in many tasks, as they possess an enormous capacity to fit arbitrarily complicated functions and generalize well. It is promising to borrow the strength of deep neural networks to fit the unknown distribution.

Note that our goal is to generate new samples from an unknown distribution. Thus, there are two different kinds of generative models:

1. *Explicit models* provide explicit parametric specifications to the data distribution and have tractable likelihood functions.

2. *Implicit models* do not specify the distribution of the data itself but instead define a stochastic process that, after training, aims to draw samples from the underlying data distribution.

Examples of *implicit models* include: variational auto-encoders (VAEs) [Kingma and Welling, 2013], generative adversarial networks (GANs) [Goodfellow et al., 2014]. While energy-based models (EBMs) [Du and Mordatch, 2019] are typical examples of explicit models. Normalizing flows (NFs) [Dinh et al., 2016, Kingma and Dhariwal, 2018, Dinh et al., 2014] are special since they not only define a stochastic process for the generation but also provide an explicit form for the likelihood. The training of an explicit model is often related to the *maximum likelihood principle*, which applies carefully designed optimization methods to reach the optimal configuration θ of the likelihood $p_\theta(x)$ defined by the model. The intuition behind maximizing the likelihood is its equivalence to minimizing the Kullback-Leibler (KL) divergence between the empirical distribution of the raw data p_{data} and p_θ. On the other hand, for implicit models, different measures of similarity in $\mathcal{X}$ space are defined, and the model is trained to maximize the similarity between the samples generated by the model and samples of the raw data.

1.4 Research Topics and Contributions

The thesis mainly focuses on understanding, improving and applying modern deep generative models. Overall, in chapter 2, we introduce the basic mechanisms of different DGMs and provide our new understandings for some of them; in Chapter 3, we propose several directions to improve the performance of these DGMs by model combination; in chapter 4, we select several domains as downstream tasks and apply DGMs. Our contributions can be summarized as follows:

1. We propose the optimization trick of VAEs (section 2.1.6) and the Variational Latent Optimization (VLO) (section 2.1.7). The optimization trick helps the VAE to infer the approximate posterior distribution of the latent variables without the information of the encoder, which also forms the key idea of Likelihood Regret (section 4.2). At the same time, the VLO replaces the encoder of the VAE with the optimization trick during the training stage. Experimental results are presented to show that the Variational Latent Optimization obtains similar performance compared to the original VAE in general, but it possesses advantages over the VAE when training data is limited.

2. We propose a new understanding of the sampling dynamics of Energy-based models (EBMs) (section 2.4). Maximum likelihood estimation is widely used in training Energy-based models (EBMs). Training requires samples from an unnormalized distribution, which is usually intractable, and in practice, these are obtained by MCMC algorithms such as Langevin dynamics. However, since MCMC in high-dimensional space converges extremely slowly, the current understanding of maximum likelihood training, which assumes approximate samples from the model can be drawn, is problematic [Nijkamp et al., 2019]. We try to understand this training procedure by replacing Langevin dynamics with the deterministic solution of the associated gradient descent ODE. Doing so allows us to study the density induced by the dynamics (if the

dynamics are invertible) and make a connection to GANs by treating the dynamic as a generator model, the initial values as latent variables, and the loss as optimizing a critic defined by the very same energy that determines the generator through its gradient. We show that reintroducing the noise in the dynamics merely reduces the quality of the generator and the EBM training is effectively a self-adversarial procedure rather than maximum likelihood estimation.

3. We propose the Generative Latent Flow (GLF) (section 3.2), an algorithm for generative modeling of data distributions. GLF uses an auto-encoder to learn latent representations of the data and a normalizing Flow to map the distribution of the latent variables to that of a standard Gaussian. GLF can be seen as a variational auto-encoder, with normalizing Flow prior and a vanishing limit of the pixel-wise variance of the data. We carefully study this relationship and the pros and cons of using an auto-encoder vs. a variational auto-encoder. In contrast to many other auto-encoder(AE)-based generative models, which use various regularizers to encourage the encoded latent distribution to match the prior distribution, our model explicitly constructs a mapping between these two distributions, leading to better density matching while avoiding over regularizing the latent variables. We compare our model with several related techniques and show that our method achieves state-of-the-art sample quality and diversity among AE-based models on commonly used datasets and is competitive with GANs' benchmarks under standard quantitative evaluations.

4. We propose to improve the generation quality of likelihood-based generative models by exponential tilting (section 3.3). Our method constructs an energy-based model on the latent variable space that yields an energy function on samples produced by the pre-trained generative model. The energy-based model is efficiently trained by maximizing the data likelihood, and after training, new samples in the latent space are generated from the energy-based model and passed through the generator to produce samples

in the data space. We show that our proposed method greatly improves the sample quality of popular likelihood-based generative models, such as normalizing flows and VAEs, with very little computational overhead.

5. We propose Likelihood Regret to improve the Out-of-distribution detection results of the VAE (section 4.2). Deep probabilistic generative models enable modeling the likelihoods of very high-dimensional data. Therefore, an important application of generative modeling should be detecting out-of-distribution (OOD) samples by setting a threshold on the likelihood. However, some recent studies show that probabilistic generative models can, in some cases, assign higher likelihoods to certain types of OOD samples, making the OOD detection rules based on likelihood threshold problematic. Several OOD detection methods have been proposed for deep generative models to address this issue. However, we observe that many of these methods fail when applied to generative models based on Variational Auto-encoders (VAE). As an alternative, we propose Likelihood Regret, an efficient OOD score for VAEs. We benchmark our proposed method over existing approaches, and empirical results suggest that our method obtains the best overall OOD detection performances when applied to VAEs.

6. We propose to improve the key-points-based face reenactment model by adding a delta net and using the 3D Morphable Model (section 4.3). The face reenactment task transfers the motion and expression pattern from a driving facial video to another source image. The current key-points-based method [Siarohin et al., 2019] suffers from low-quality generation and insufficient ability to disentangle the motion information and the identity information from the facial inputs. We propose using a pre-trained 3D facial landmark detector with a delta net to improve the training speed and stability of the previous key-points-based method. This also ensures that the model captures detailed movement, for example, eyeball movements, which the other methods fail to learn. Besides, by combining the key-point-based method with 3DMM, we suc-

cessfully improve the key-point-based method's generalization ability. Experimental results confirm that our new method generates excellent samples and can be used for the controllable generation of human faces.

CHAPTER 2

DEEP GENERATIVE MODELS

2.1 Variational Auto-Encoders

2.1.1 Introduction

The **Variational Auto-Encoder (VAE)** is the probabilistic graphical model that combines the Auto-Encoder (AE) and variational inference. It is a basic but powerful generative model introduced by Diederik P. Kingma and Max Welling. As an AE-based model, the VAE contains an encoder that is designed as a recognition model and a decoder that plays the role of a generative model. The encoder maps the input to an approximation of the posterior distribution over the corresponding latent random variable to obtain a reduced representation of the original input in the latent space. At the same time, the decoder takes a sample generated from the approximated posterior distribution as input and aims to reconstruct the original input of the encoder. Equipped with deep neural networks, both the encoder and the decoder possess sufficient ability to capture strange and complicated distributions that we are interested in.

Although it is not a complicated model, the VAE is one of the most useful and popular generative models, as we can see its successful applications in image generation [Razavi et al., 2019], text generation [Wang et al., 2019], domain translation [Liu et al., 2017] and video synthesis [Walker et al., 2021]. Besides, it is fruitful to study the simplified and low-dimensional latent space derived from the VAE to explore the properties of the original high-dimensional complex domain[Sundar et al., 2020]. Obviously, studying the VAE in detail is worthwhile to understand and improve modern generative models.

2.1.2 Variational Inference and Distribution Learning

The ultimate goal of a generative model is to approximate the unknown distribution $p(\mathbf{x})$ of the training dataset $\{x_i \in \mathcal{X}, i = 1, 2 \cdots, n\}$ and provide a sampling method to generate new samples. The VAE uses a latent variable $\mathbf{z}$ with prior $p(\mathbf{z})$, and a conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$, to model the observed variable $\mathbf{x}$. The model distribution, denoted by $p_\theta(\mathbf{x})$, can be formulated as $p_\theta(\mathbf{x}) = \int_{\mathcal{Z}} p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}$. Following the maximum likelihood principle, we need to maximize the likelihood to minimize the distance between the empirical distribution $p_{\text{data}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{\mathbf{x}=x_i\}}$ and $p_\theta(\mathbf{x})$:

$$D_{\text{KL}}\left[p_{\text{data}} \| p_\theta\right] = -\sum_{i=1}^{n} \frac{1}{n} \log p_\theta(x_i) + const.$$

However, direct computation of this likelihood is intractable in high dimensions. Thus variational inference is used to derive a lower bound on the log-likelihood of $\mathbf{x}$. This leads to the famous evidence lower bound (ELBO):

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log \int_{\mathcal{Z}} \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} q_\phi(\mathbf{z}|\mathbf{x})\mathrm{d}\mathbf{z} \\
&\geq \int_{\mathcal{Z}} \log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} q_\phi(\mathbf{z}|\mathbf{x})\mathrm{d}\mathbf{z} \quad \text{(Jensen's inequality)} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\text{KL}}\left[q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\right] \\
&\triangleq \mathcal{L}(\mathbf{x}; \theta, \phi),
\end{aligned} \tag{2.1}$$

where $q_\phi(\mathbf{z}|\mathbf{x})$ is the variational approximation to the true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$. The inequality becomes equality (ELBO equals the log likelihood) when $q_\phi(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{z}|\mathbf{x})$. Both $q_\phi(\mathbf{z}|\mathbf{x})$ and $p_\theta(\mathbf{x}|\mathbf{z})$ are parameterized by neural networks with parameters ϕ (encoder) and θ (decoder), respectively. The VAE is trained by maximizing $\mathcal{L}(\mathbf{x}; \theta, \phi)$ over the training data.

Unlike generative models using exact inference so that the likelihood can be directly

computed, VAE only outputs a lower bound of the log-likelihood, and the exact log-likelihood needs to be estimated, usually by an importance weighted lower bound [Burda et al., 2015]:

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}^1,\ldots,\mathbf{z}^K \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta\left(\mathbf{x}|\mathbf{z}^k\right) p(\mathbf{z}^k)}{q_\phi\left(\mathbf{z}^k|\mathbf{x}\right)} \right] \triangleq \mathcal{L}_K(\mathbf{x};\theta,\phi), \tag{2.2}$$

where each $\mathbf{z}^k$ is a sample from the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$. Using Jensen's inequality, it is easy to prove that:

$$\mathcal{L}_1(\mathbf{x};\theta,\phi) \leq \mathcal{L}_2(\mathbf{x};\theta,\phi) \leq \cdots \leq \mathcal{L}_K(\mathbf{x};\theta,\phi) \leq \cdots \to \log p_\theta(\mathbf{x}).$$

Therefore, for large K, $\mathcal{L}_K(\mathbf{x};\theta,\phi)$ is a good estimator for the log-likelihood.

2.1.3 Analysis of ELBO

While the prior $p(\mathbf{z})$ and the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ are often chosen to be Gaussian distributions, there are multiple choices for the decoding distribution $p_\theta(\mathbf{x}|\mathbf{z})$ depending on the type of data. In the Gaussian case,

$$\log p_\theta(\mathbf{x}|\mathbf{z}) = -\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2}{2\sigma^2} + const.$$

When σ is fixed, the log Gaussian density is equivalent to the (negative) reconstruction loss between the predicted mean $\hat{\mathbf{x}}$ and the original input $\mathbf{x}$. Thus, the first part of the ELBO, $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$, is also known as the **reconstruction term**, which constrains the output reconstruction from the decoder $\hat{\mathbf{x}}$ to be close to the original input $\mathbf{x}$ and ensures the latent variable $\mathbf{z}$ captures the necessary information from $\mathbf{x}$.

The second term of the ELBO, $D_{\mathrm{KL}}\left[q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right]$, is called the **KL-term**, which constraints the approximate posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to be close to the prior distribution $p(\mathbf{z})$.

2.1.4 The Reparameterization Trick

To train the encoder and the decoder, we need to estimate the gradient of the loss function $\nabla_\theta \mathcal{L}(\mathbf{x};\theta,\phi)$ and $\nabla_\phi \mathcal{L}(\mathbf{x};\theta,\phi)$. Under the Gaussian assumption, $q_\phi(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}\left(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x})\right)$, $p(\mathbf{z}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x})$ is the mean vector and the covariance matrix of the approximated posterior distribution returned by the encoder. To simplify the computation, the VAE assumes $\Sigma_\phi(\mathbf{x})$ to be a diagonal matrix, which means the approximate posterior distribution is a factorized Gaussian distribution. Then the KL-term in the ELBO becomes the KL-divergence between two factorized Gaussian distributions, which has an analytical form, and it is straightforward to compute the gradient. However, this is not the case for the reconstruction term.

To easily and accurately estimate the gradient of the reconstruction term, [Kingma and Welling, 2013] proposes the **Reparameterization Trick**. Assume

$$f_\phi(\epsilon) = \mu_\phi(\mathbf{x}) + \sqrt{\Sigma_\phi(\mathbf{x})}\epsilon,$$

then

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] = \mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\log p_\theta(\mathbf{x}|f_\phi(\epsilon))\right].$$

We can approximate the gradient by:

$$\bigtriangledown \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] \approx \frac{1}{N}\sum_{i=1}^{N} \nabla \log p_\theta(\mathbf{x}|f_\phi(\epsilon_i)), \text{ for } \epsilon_i \sim^{i.i.d} \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

It has been shown in experiments that the gradient approximation using the Reparameterization trick has lower variance than other approximation methods, for example, the Reinforce method (even with control variates)[Kucukelbir et al., 2017], and it is favored in the implementations of the VAE.

2.1.5 Reparameterization v.s. Reinforce

Although it is well known that in many experiment settings, including VAEs, the Reparameterization produces estimators with lower variances than the reinforce estimator, there is no theoretical guarantee that one dominates another, indicating that under certain circumstances, reinforce might be better, yet such a situation does not occur often. They are both Monte Carlo estimators to estimate $\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z})}[f(\mathbf{z})]$. The reinforce method uses the fact that

$$\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z})}[f(\mathbf{z})] = \mathbb{E}_{q_\phi(\mathbf{z})}\left[f(\mathbf{z})\nabla_\phi \log q_\phi(\mathbf{z})\right],$$

and builds an estimator over a set of i.i.d samples $\{\mathbf{z}_i \sim q_\phi | i = 1, ..., n\}$:

$$\text{RF} = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{z}_i)\nabla_\phi \log q_\phi(\mathbf{z}_i). \tag{2.3}$$

The formula of the Reparameterization can be derived from the idea of section 2.1.4 as

$$\text{RP} = \frac{1}{n}\sum_{i=1}^{n} \left[\nabla_\phi \mathbf{z}(\phi, \epsilon_i)\right]^T \nabla_{\mathbf{z}} f(\mathbf{z}_i),\ \epsilon_i \sim^{i.i.d} p(\epsilon), \tag{2.4}$$

if sampling $\mathbf{z}_i \sim q_\phi(\mathbf{z})$ is equivalent to setting $\mathbf{z}_i = \mathbf{z}(\phi, \epsilon_i), \epsilon_i \sim p(\epsilon)$. To show some intuition why the Reparameterization is preferred in training VAEs, we analyze these two methods in estimating the gradient of the ELBO under a linear assumption on the decoder G_θ. In that case, $q_\phi(\mathbf{z}) = q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mu(\phi, \mathbf{x}), \Sigma(\phi, \mathbf{x}) = \text{diag}\left(\sigma_1^2, ..., \sigma_k^2\right)\right)$ is returned by the encoder (assume the dimension of $\mathbf{z}$ is k), $\mathbf{z}(\phi, \epsilon) = \mu + \sqrt{\Sigma}\epsilon$, $p(\epsilon) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $f(\mathbf{z}) = \log p_\theta(\mathbf{x}|\mathbf{z}) =$

$-\frac{\|\mathbf{x}-G_\theta(\mathbf{z})\|_2^2}{2\sigma^2} + const$. Suppose $\mathbf{z} = (z_1, ..., z_k), \mu = \mu(\phi, \mathbf{x}) = (\mu_1, ..., \mu_k)$, then we have

$$\log q_\phi(\mathbf{z}|\mathbf{x}) = \sum_{i=1}^{k} \left[-\frac{1}{2}\frac{(z_i - \mu_i)^2}{\sigma_i^2} - \frac{1}{2}\log 2\pi - \frac{1}{2}\log \sigma_i^2 \right]$$

$$f(\mathbf{z}) \nabla_\phi \log q_\phi(\mathbf{z}|\mathbf{x}) = \sum_{i=1}^{k} f(\mathbf{z}) \frac{\partial \log q_\phi}{\partial \mu_i}\frac{\partial \mu_i}{\partial \phi} + \sum_{i=1}^{k} f(\mathbf{z}) \frac{\partial \log q_\phi}{\partial \sigma_i^2}\frac{\partial \sigma_i^2}{\partial \phi}$$

$$\left[\nabla_\phi \mathbf{z}(\phi, \epsilon)\right]^T \nabla_\mathbf{z} f(\mathbf{z}) = \sum_{i=1}^{k} \frac{\partial f(\mathbf{z})}{\partial z_i} \left[\frac{\partial z_i}{\partial \mu_i}\frac{\partial \mu_i}{\partial \phi} + \frac{\partial z_i}{\partial \sigma_i^2}\frac{\partial \sigma_i^2}{\partial \phi} \right]$$

Comparing the coefficients: both estimators are linear combinations of $\frac{\partial \mu_i}{\partial \phi}, \frac{\partial \sigma_i^2}{\partial \phi}$, which depend on the structure of the encoder, thus we do not have an analytical form for them. Luckily they do not involve randomness and we can compare the variance of the coefficients of $\frac{\partial \mu_i}{\partial \phi}, \frac{\partial \sigma_i^2}{\partial \phi}$. Define $T_{\mu_i}(\mathbf{z}) = f(\mathbf{z})\frac{\partial \log q_\phi}{\partial \mu_i}$, $T_{\sigma_i}(\mathbf{z}) = f(\mathbf{z})\frac{\partial \log q_\phi}{\partial \sigma_i^2}$, $S_{\mu_i}(\mathbf{z}) = \frac{\partial f(\mathbf{z})}{\partial z_i}\frac{\partial z_i}{\partial \mu_i}$, $S_{\sigma_i}(\mathbf{z}) = \frac{\partial f(\mathbf{z})}{\partial z_i}\frac{\partial z_i}{\partial \sigma_i^2}$, and let $\mathbb{P}_d(g) = \frac{1}{n}\sum_{i=1}^{n} g(\mathbf{z}_i)$ denotes the empirical average of a function g, then

$$\text{RF} = \sum_{i=1}^{k} \mathbb{P}_d(T_{\mu_i})\frac{\partial \mu_i}{\partial \phi} + \sum_{i=1}^{k} \mathbb{P}_d(T_{\sigma_i})\frac{\partial \sigma_i^2}{\partial \phi};$$

$$\text{RP} = \sum_{i=1}^{k} \mathbb{P}_d(S_{\mu_i})\frac{\partial \mu_i}{\partial \phi} + \sum_{i=1}^{k} \mathbb{P}_d(S_{\sigma_i})\frac{\partial \sigma_i^2}{\partial \phi}.$$

Our conclusion is: under such conditions, if G_θ is a linear mapping, then $\forall i = 1, ..., k$,

$$\text{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[T_{\mu_i}(\mathbf{z})\right] \geq \frac{3}{2}\text{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[S_{\mu_i}(\mathbf{z})\right]$$

$$\text{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[T_{\sigma_i}(\mathbf{z})\right] \geq 3\text{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[S_{\sigma_i}(\mathbf{z})\right]$$

Although this is not equivalent to Var[RF] $\geq$ Var[RP], it provides evidence to show that the **RF** algorithm is much more variable than the **RP** algorithm under Gaussian and linear assumptions. The proof is given below.

Assume G_θ is a linear mapping, then $f(\mathbf{z})$ becomes a quadratic function of $\mathbf{z}$:

$$f(\mathbf{z}) = C + W^T(\mathbf{z} - \mu) + 1/2(\mathbf{z} - \mu)^T H(\mathbf{z} - \mu),$$

for $C \in \mathbb{R}, W \in \mathbb{R}^k, H \in \mathbb{R}^{k \times k}$ that are independent of $\mathbf{z}$.

Comparison between the coefficients of $\frac{\partial \mu_i}{\partial \phi}$

For **RP**,

$$\frac{\partial f(\mathbf{z})}{\partial z_i}\frac{\partial z_i}{\partial \mu_i} = W_i + \sum_{j=1}^{k} H_{ij}(z_j - \mu_j)$$
$$\mathrm{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{\partial f(\mathbf{z})}{\partial z_i}\frac{\partial z_i}{\partial \mu_i}\right] = \sum_{j=1}^{k} H_{ij}^2 \sigma_j^2$$

For **RF**,

$$\frac{\partial \log q_\phi(\mathbf{z}|\mathbf{x})}{\partial \mu_i} = \frac{z_i - \mu_i}{\sigma_i^2}$$
$$f(\mathbf{z})\frac{\partial \log q_\phi}{\partial \mu_i} = \left(C + W^T(\mathbf{z} - \mu) + \frac{1}{2}(\mathbf{z} - \mu)^T H(\mathbf{z} - \mu)\right)\frac{z_i - \mu_i}{\sigma_i^2}$$

Notice that $(z_i - \mu_i)(z_j - \mu_j), (z_i - \mu_i)(z_k - \mu_k)$ are uncorrelated unless $j = k$; $(z_i - \mu_i)(z_j - \mu_j), (z_i - \mu_i)(z_k - \mu_k)(z_s - \mu_s)$ are uncorrelated; $(z_i - \mu_i)(z_j - \mu_j)(z_r - \mu_r), (z_i - \mu_i)(z_k - \mu_k)(z_s - \mu_s)$ are uncorrelated unless $\{j, r\} = \{k, s\}$.

Write

$$f(\mathbf{z}) = \underbrace{\left(W_i(z_i - \mu_i) + \sum_{j \neq i} (z_i - \mu_i) H_{ij} (z_j - \mu_j) \right)}_{l_{z1}(i)} + \underbrace{\left(C + \frac{1}{2} \sum_{j=1}^{k} H_{jj} (z_j - \mu_j)^2 \right)}_{l_{z2}} + \underbrace{\left(\sum_{j \neq i} W_j (z_j - \mu_j) + \frac{1}{2} \sum_{j \neq i, m \neq i, j \neq m} (z_j - \mu_j) H_{jm} (z_m - \mu_m) \right)}_{l_z(-i)},$$

then it is easy to verify that $l_{z1}(i)\frac{z_i - \mu_i}{\sigma_i^2}, l_{z2}\frac{z_i - \mu_i}{\sigma_i^2}, l_z(-i)\frac{z_i - \mu_i}{\sigma_i^2}$ are uncorrelated, so

$$\begin{aligned} \mathrm{Var}\left(f(\mathbf{z}) \frac{\partial \log q_\phi}{\partial \mu_i} \right) &\geq \mathrm{Var}\left(l_{z1}(i) \frac{\partial \log q_\phi}{\partial \mu_i} \right) + \mathrm{Var}\left[l_{z2} \frac{\partial \log q}{\partial \mu_i} \right] \\ &= \mathrm{Var}\left[\left(W_i + \sum_{j \neq i} H_{ij} (z_j - \mu_j) \right) \frac{(z_i - \mu_i)^2}{\sigma_i^2} \right] + \mathrm{Var}\left[l_{z2} \frac{\partial \log q_\phi}{\partial \mu_i} \right] \end{aligned}$$

If A, B are independent, then

$$\begin{aligned} \mathrm{Var}(AB) = \mathbb{E}A^2B^2 - (\mathbb{E}AB)^2 &= \mathbb{E}A^2 \mathbb{E}B^2 - (\mathbb{E}A)^2 (\mathbb{E}B)^2 \\ &= [\mathbb{E}A^2 - (\mathbb{E}A)^2] \mathbb{E}B^2 + [\mathbb{E}B^2 - (\mathbb{E}B)^2] (\mathbb{E}A)^2 \\ &= \mathrm{Var}A \cdot \mathbb{E}B^2 + \mathrm{Var}B \cdot (\mathbb{E}A)^2. \end{aligned}$$

Let $A = W_i + \sum_{j\neq i} H_{ij}(z_j - \mu_j), B = \frac{(z_i-\mu_i)^2}{\sigma_i^2}$, so

$$\mathrm{Var}(AB) = 3\sum_{j\neq i} H(\mu)_{ij}^2\sigma_j^2 + 2W_i^2.$$

Also, let $U = \frac{z_i-\mu_i}{\sigma_i}, V = C + \frac{1}{2}\sum_{j\neq i} H_{jj}(z_j-\mu_j)^2$, so U, V are independent. Assume $\rho = \frac{\sigma_i^2 H_{ii}}{2}$, then

$$\begin{aligned}
\mathrm{Var}\left[l_{z2}\frac{\partial \log q_\phi}{\partial \mu_i}\right] &= \frac{1}{\sigma_i^2}\mathrm{Var}[(\frac{\sigma_i^2 H_{ii}}{2}U^2 + V)U] = \frac{1}{\sigma_i^2}\mathbb{E}\left[\rho U^3 + UV\right]^2 \\
&= \frac{1}{\sigma_i^2}\left[\rho^2\mathbb{E}U^6 + 2\rho\mathbb{E}U^4\mathbb{E}V + \mathbb{E}U^2\mathbb{E}V^2\right] \\
&= \frac{1}{\sigma_i^2}\left[15\rho^2 + 6\rho\mathbb{E}V + \mathbb{E}V^2\right] \\
&= \frac{6\rho^2}{\sigma_i^2} + \frac{1}{\sigma_i^2}\mathbb{E}\left[3\rho + V\right]^2 \geq \frac{6\rho^2}{\sigma_i^2} \\
&= \frac{3}{2}H_{ii}^2\sigma_i^2
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathrm{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left(f(\mathbf{z})\frac{\partial \log q_\phi}{\partial \mu_i}\right) &\geq 3\sum_{j\neq i} H_{ij}^2\sigma_j^2 + 2W_i^2 + \frac{3}{2}H_{ii}^2\sigma_i^2 \\
&\geq \frac{3}{2}\sum_{j=1}^{k} H_{ij}^2\sigma_j^2 \\
&= \frac{3}{2}\mathrm{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{\partial f(\mathbf{z})}{\partial z_i}\frac{\partial z_i}{\partial \mu_i}\right]
\end{aligned}$$

Comparison between the coefficients of $\frac{\partial \sigma_i^2}{\partial \phi}$

For **RP**,

$$\frac{\partial f(\mathbf{z})}{\partial z_i}\frac{\partial z_i}{\partial \sigma_i^2} = \left[W_i + \sum_{j=1}^{k} H_{ij}(z_j - \mu_j)\right] \cdot \frac{1}{2\sigma_i}\epsilon_i$$

$$\mathrm{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{\partial f(\mathbf{z})}{\partial z_i}\frac{\partial z_i}{\partial \sigma_i^2}\right] = \frac{1}{4\sigma_i^2}\left[W_i^2 + \sum_{j\neq i} H_{ij}^2\sigma_j^2 + 2H_{ii}^2\sigma_i^2\right]$$

For **RF**

$$\frac{\partial \log q_\phi(\mathbf{z}|\mathbf{x})}{\partial \sigma_i^2} = \frac{(z_i-\mu_i)^2}{2\sigma_i^4} - \frac{1}{2\sigma_i^2}$$

$$f(\mathbf{z})\frac{\partial \log q_\phi}{\partial \sigma_i^2} = \left(C + W^T(z-\mu) + \frac{1}{2}(z-\mu)^T H(z-\mu)\right)\cdot\left(\frac{(z_i-\mu_i)^2}{2\sigma_i^4} - \frac{1}{2\sigma_i^2}\right)$$

$$f(\mathbf{z}) = l_{z1}(i) + l_{z2} + l_z(-i)$$

It is true that $l_{z1}(i)[\frac{(z_i-\mu_i)^2}{2\sigma_i^4} - \frac{1}{2\sigma_i^2}], l_{z2}[\frac{(z_i-\mu_i)^2}{2\sigma_i^4} - \frac{1}{2\sigma_i^2}], l_z(-i)[\frac{(z_i-\mu_i)^2}{2\sigma_i^4} - \frac{1}{2\sigma_i^2}]$ are uncorrelated, so

$$\mathrm{Var}\left(f(\mathbf{z})\frac{\partial \log q_\phi}{\partial \sigma_i^2}\right) \geq \mathrm{Var}\left(l_{z1}(i)\frac{\partial \log q_\phi}{\partial \sigma_i^2}\right) + \mathrm{Var}\left[l_{z2}\frac{\partial \log q_\phi}{\partial \sigma_i^2}\right]$$

$$= \mathrm{Var}\left[\frac{W_i + \sum_{j\neq i} H_{ij}(z_j-\mu_j)}{2\sigma_i^2}\left(\frac{(z_i-\mu_i)^3}{\sigma_i^2} - (z_i-\mu_i)\right)\right] + \mathrm{Var}\left[l_{z2}\frac{\partial \log q}{\partial \sigma_i^2}\right].$$

Let $A = \frac{W_i + \sum_{j\neq i} H_{ij}(z_j-\mu_j)}{2\sigma_i^2}, B = \frac{(z_i-\mu_i)^3}{\sigma_i^2} - (z_i-\mu_i)$,

$$\mathrm{Var}\left[AB\right] = \frac{5}{2\sigma_i^2}\left[\sum_{j\neq i} H_{ij}^2\sigma_j^2\right] + \frac{5}{2\sigma_i^2}W_i^2.$$

Also, let $U = \frac{(z_i-\mu_i)^2}{\sigma_i^2}, V = C + \frac{1}{2}\sum_{j\neq i} H_{jj}(z_j-\mu_j)^2$, then U, V are independent.

Assume $\rho = \frac{\sigma_i^2 H_{ii}}{2}$, then

$$\begin{aligned}\mathrm{Var}\left[l_{z2}\frac{\partial \log q_\phi}{\partial \sigma_i^2}\right] &= \mathrm{Var}\left[\left(\frac{\sigma_i^2 H_{ii}}{2}U + V\right)\frac{U-1}{2\sigma_i^2}\right] \\ &= \frac{1}{4\sigma_i^4}\mathrm{Var}\left[(\rho(U-1)+(V+\rho))(U-1)\right].\end{aligned}$$

Let $U' = U - 1, V' = V + \rho$, U', V' are independent. $\mathbb{E}(U')^2 = 2, \mathbb{E}(U')^3 = 8, \mathbb{E}(U')^4 = 60$. Then

$$\begin{aligned}\mathrm{Var}\left[l_{z2}\frac{\partial \log q_\phi}{\partial \sigma_i^2}\right] &= \frac{1}{4\sigma_i^4}\mathrm{Var}\left[(\rho U' + V')U'\right] \\ &= \frac{1}{4\sigma_i^4}\left[60\rho^2 + 16\rho\mathbb{E}V' + 2\mathbb{E}(V')^2 - 4\rho^2\right] \\ &= \frac{1}{4\sigma_i^4}\left[24\rho^2 + 2\mathbb{E}(4\rho + V')^2\right] \\ &\geq \frac{3H_{ii}^2}{2}.\end{aligned}$$

As a result,

$$\begin{aligned}\mathrm{Var}_z\left(f(\mathbf{z})\frac{\partial \log q_\phi}{\partial \sigma_i^2}\right) &\geq \frac{5}{2\sigma_i^2}\left[\sum_{j\neq i} H_{ij}^2\sigma_j^2\right] + \frac{5}{2\sigma_i^2}W_i^2 + \frac{3H_{ii}^2}{2} \\ &\geq \frac{3}{4\sigma_i^2}\left[W_i^2 + \sum_{j\neq i} H_{ij}^2\sigma_j^2 + 2H_{ii}^2\sigma_i^2\right] \\ &= 3\mathrm{Var}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{\partial f(\mathbf{z})}{\partial z_i}\frac{\partial z_i}{\partial \sigma_i^2}\right]\end{aligned}$$

2.1.6 The Optimization Trick

Suppose you have trained the VAE and obtained the fine-tuned parameters (θ^*, ϕ^*) for the encoder and decoder, but suddenly you cannot get access to the encoder, how could you infer $q_{\phi^*}(\mathbf{z}|\mathbf{x})$? Here we can leverage the **optimization trick** to compute the approximated

posterior distribution without using the encoder. Note that

$$\mathcal{L}(\mathbf{x};\theta,\phi) = \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}\left[q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})\right].$$

Since $q_\phi(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}\left(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x})\right)$, we can compute $q_{\phi^*}(\mathbf{z}|\mathbf{x})$ through re-optimization over the ELBO:

$$\left(\hat{\mu}_\phi(\mathbf{x}), \hat{\Sigma}_\phi(\mathbf{x})\right) = \underset{q(\mathbf{z})=\mathcal{N}(\mu,\Sigma)}{\operatorname{argmax}} \mathbb{E}_{q(\mathbf{z})} \log p_{\theta^*}(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}\left[q(\mathbf{z})||p(\mathbf{z})\right]. \tag{2.5}$$

Then $\hat{q}_\phi(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}\left(\hat{\mu}_\phi(\mathbf{x}), \hat{\Sigma}_\phi(\mathbf{x})\right)$ is close to $p_{\theta^*}(\mathbf{x}|\mathbf{z})$ and we can use it as the approximate posterior distribution, which can be interpreted as the optimal posterior distribution (within the mean field Gaussian space) that maximizes ELBO at $\mathbf{x}$. In general $\hat{q}_\phi(\mathbf{z}|\mathbf{x}) \neq q_{\phi^*}(\mathbf{z}|\mathbf{x})$, and this gap is crucial for our **likelihood regret** method as we will see in chapter 4.2.

2.1.7 Variational Latent Optimization

From the discussion in section 2.1.6, we know that the encoder is not necessary during the inference stage, but is it also possible to remove the encoder during the training stage? The answer is **Yes**. We propose the **Variational Latent Optimization (VLO)** as the counterpart of the VAE without the encoder. In the VLO setting, it only has a decoder parameterized by a neural network with parameters θ, and the encoder is replaced by (2.5), both in training and inference. The entire training process is explained in Algorithm 1. The algorithm updates and stores the approximate posterior means and variances for each training sample in matrices M, Σ. $\mathcal{L}(x, \hat{x}, \mu, \sigma, \theta)$ is the negative ELBO to emphasize that it depends on a given μ, σ instead of the output of an encoder. Since the optimization is done for each example, in principle, this should provide a better variational approximation than the encoder, as we will see in the experimental section 2.1.8.

Algorithm 1 Variational Latent Optimization

Require: latent variable dimension d, size of dataset N, number of inner loop steps s, learning rates γ_1, γ_2.
Initialization: matrix $M, \Sigma \in \mathbb{R}^{N \times d}$, decoder network G_θ.
Define $\mathcal{L}$ to be negative ELBO.
for mini-batches k ($k = 0, 1, 2, \dots$) **do**
    x: data sampled in the k^{th} minibatch.
    i: index of x in the dataset.
    $\mu, \sigma \leftarrow M[i, :], \Sigma[i, :]$
    for inner loops j ($j = 0, \dots, s$) **do**
        $z = \text{reparametrize}(\mu, \sigma)$
        $\hat{x} = G_\theta(z)$
        $\mu \leftarrow \mu - \gamma_1 \nabla_\mu \mathcal{L}(x, \hat{x}, \mu, \sigma, \theta)$
        $\sigma \leftarrow \sigma - \gamma_1 \nabla_\sigma \mathcal{L}(x, \hat{x}, \mu, \sigma, \theta)$
    $\theta \leftarrow \theta - \gamma_2 \nabla_\theta \mathcal{L}(x, \hat{x}, \mu, \sigma, \theta)$
    $M[i, :], \Sigma[i, :] \leftarrow \mu, \sigma$

2.1.8 Experiments

We empirically evaluate our methods through experiments on the MNIST dataset [LeCun et al., 2010]. We compare our method with the VAE using commonly used metrics: ELBO and Negative Log-likelihood (NLL) on the test set. We adopt a relatively simple decoder structure consisting of two hidden layers, each with 512 hidden units. The VAE has the same decoder and a symmetrically-defined encoder. We use batch size 200 for training both models. For the VLO, we initialize the mean matrix M as the PCA projection of the training set, where the PCA is computed on 10000 training images. We initialize the matrix Σ that stores log-variance as 0. We train both models for 200 epochs using the Adam optimizer. For the VAE, the initial learning rate is 10^{-3}, and it is halved after 100 epochs. For the VLO, the initial learning rates are 10^{-3} for the decoder parameters, 10^{-1} for μ and $\log \sigma^2$. Both learning rates are halved after 50 epochs. We choose the number of inner loop iterations $s = 5$.

When testing VLO, for each image in the test set, we fix the decoder parameter and update μ and $\log \sigma^2$ for 200 epochs. Both μ and $\log \sigma^2$ are initialized as 0. The NLL is given

by (2.2) with $k = 1000$. Results are presented in Table 2.1.

Table 2.1: Comparing VAE and VLO

latent dimension	Reconstruction Loss		-ELBO		NLL	
	VAE	VLO	VAE	VLO	VAE	VLO
10	78.68	77.44	98.47	97.41	94.02	94.40
20	75.23	71.14	96.72	96.35	91.91	91.61
40	76.35	71.10	97.11	96.19	92.26	91.29
64	76.08	71.11	96.89	96.20	92.30	91.38

Comment on the results: In general, we observe that both methods obtain similar performances in test ELBO and NLL. The VLO has significantly lower reconstruction loss than the VAE, while the similar ELBOs indicate that the KL divergence term of the VLO is larger than that of the VAE. Interestingly, when the dimension of z is large (such as 40 or 64), the VAEs seem to perform worse than the case when z is 20-dimensional. The network capacity may be insufficient for VAEs with large latent dimensions. The last two rows of our table show that the VLO performs slightly better than VAEs.

Next, we explore the relative performances of the VLO and the VAE by varying the available amount of training data. We train our models with different size training data and report their test ELBO and NLL loss. We use the same network structures and the latent dimension of 40. We adjust the number of training epochs to ensure each model is trained in approximately the same number of iterations. We restrict the PCA initialization over the samples in the training data and reduce it to 1k images if the training size is smaller than 10k. Results are in Table 2.2.

Table 2.2: Comparing VAE and VLO with different training sizes

training size	Reconstruction Loss		-ELBO		NLL	
	VAE	VLO	VAE	VLO	VAE	VLO
5k	102.1	78.65	123.1	104.4	112.8	99.17
10k	89.75	75.96	110.5	101.7	102.1	96.40
30k	80.72	72.51	101.0	97.85	95.22	92.63
60k	76.35	71.10	97.11	96.19	92.26	91.29

When the number of training samples is small, the VLO performs much better than the VAE on the test set, which is reasonable as the optimization step helps infer the mean and variance of the posterior distribution of the hidden variables. At the same time, the encoder cannot obtain good regression results on the approximated posterior distribution with smaller training sets.

In summary, the VLO obtains similar performance compared to the original VAE, and the VLO possesses advantages over the VAE when training data is limited. Therefore, we can treat the VLO as an alternative to the VAE.

2.2 Flow-Based Models

2.2.1 Introduction

Normalizing flows (NFs) are carefully-designed invertible networks that map the training data to a simple distribution. Thanks to the invertible structure, it is straightforward to compute the likelihood through the **change-of-variable** formula and generate new samples from the inverse process. In contrast, other generative models, for example, VAEs and GANs, can not easily do both. Normalizing flows have been applied on a variety of tasks, including: image generation[Kingma and Dhariwal, 2018], audio generation[Ping et al., 2020], Point-cloud modeling[Yang et al., 2019] and video generation[Kumar et al., 2019].

2.2.2 Invertible Networks and Distribution Learning

It is easy to track the distribution change when a random variable goes through a differentiable invertible mapping. Suppose $\mathbf{z} \sim p_0(\mathbf{z})$ and f is a differentiable invertible mapping, then $\mathbf{x} = f(\mathbf{z})$ follows the distribution with the density:

$$p(\mathbf{x}) = p_0(\mathbf{z}) \left| \det \frac{df(\mathbf{z})}{d\mathbf{z}} \right|^{-1}. \tag{2.6}$$

This change-of-variable formula directly provides a way for distribution learning by setting $\mathbf{x}$ as the random variable from the target distribution and $\mathbf{z}$ as the random variable with an analytical density and an efficient sampling scheme. It is common to select the standard Gaussian distribution as p_0. After building an invertible neural network F_θ to model f^{-1}, the normalizing flow can be trained by maximizing the log-likelihood of $\mathbf{x}$:

$$\mathcal{L}(\mathbf{x}, \theta) = \log p_0 \left(F_\theta(\mathbf{x})\right) + \log \left| \det \frac{dF_\theta(\mathbf{x})}{d\mathbf{x}} \right|.$$

After training, it is easy to sample a new $\mathbf{x}$ from the normalizing flow through:

$$\mathbf{x} = F_\theta^{-1}(\mathbf{z}), \mathbf{z} \sim p_0.$$

2.2.3 Coupling Layers

The key aspect of normalizing flows is to build an invertible neural network for which it is easy to compute the forward evaluation, the inverse evaluation and the determinant of the Jacobian. In general, the invertible neural network is a combination of multiple invertible blocks: $F_\theta = F_k \circ F_{k-1} \circ \cdots \circ F_0$. One typical example of this neural network block is the **affine coupling layer**.

Given D dimensional input data $\mathbf{z} = (z_1, ..., z_D)$ and $d < D$, we partition the input into two vectors $\mathbf{z}_1 = z_{1:d}$ and $\mathbf{z}_2 = z_{d+1:D}$. The output of one affine coupling layer is given by $\mathbf{y}_1 = \mathbf{z}_1$, $\mathbf{y}_2 = \mathbf{z}_2 \odot \exp(s(\mathbf{z}_1)) + t(\mathbf{z}_1)$ where s and t are functions from $\mathbb{R}^d \to \mathbb{R}^{D-d}$ and $\odot$ is the element-wise product. The inverse of the transformation is explicitly given by $\mathbf{z}_1 = \mathbf{y}_1$, $\mathbf{z}_2 = (\mathbf{y}_2 - t(\mathbf{y}_1)) \odot \exp(-s(\mathbf{y}_1))$. The determinant of the Jacobian matrix of this transformation is $\det \frac{\partial \mathbf{y}}{\partial \mathbf{z}} = \prod_{\mathrm{j}=1}^{\mathrm{d}} (\exp[\mathrm{s}(\mathbf{z}_1)_\mathrm{j}])$. Since computing both the inverse and the Jacobian of an affine coupling layer does not require computing the inverse and Jacobian of s and t, both functions can be arbitrarily complex. Affine coupling layers leave some

components of the input data unchanged. In order to transform all the components, two coupling layers are combined in an alternating pattern to form a coupling block so that the unchanged components in the first layer can be transformed in the second layer. It is common to add a random permutation of the input data coordinates at the end of each coupling block.

While being mathematically clear and well defined, normalizing flows keep the dimensionality of the original data in order to maintain bijectivity. Consequently, they cannot provide low-dimensional data representations, and the training is computationally expensive. Considering the prohibitively long training time and advanced hardware requirements in training large-scale flow models such as [Kingma and Dhariwal, 2018], we believe that it is worth exploring the application of flows in the low dimensional representation spaces rather than over the original data. Furthermore, this becomes one of the motivations of our Generative Latent Flow model (chapter 3.2).

2.3 Generative Adversarial Networks

2.3.1 Introduction

The **generative adversarial network (GAN)** is a generative framework designed by Ian Goodfellow and his colleagues in June 2014[Goodfellow et al., 2014], which is one of the most successful generative models that produce state-of-the-art results in many areas[Karras et al., 2020]. The key character of the GAN is the adversarial training, where a generator is competing with a discriminator. The GAN learns the underlying distribution of the dataset and the ability to generate new samples through the adversarial game.

2.3.2 Adversarial Training and Distribution Learning

The GAN is an implicit model in that it directly defines a generation process $\mathbf{z} \sim p(\mathbf{z}), x = G_\psi(\mathbf{z})$ (denote this distribution of x as $p_\psi(\mathbf{x})$). To measure the quality of samples, it deploys a discriminator D_ϕ to distinguish between generated samples and the raw data. Both G_ψ and D_ϕ are modeled as deep neural networks parameterized by $\theta = (\phi, \psi)$. The generator aims to generate samples that can fool the discriminator such that the discriminator cannot determine whether they are generated by the generator or are from the original data, while the discriminator tries not to be fooled. Such adversarial competition is the main feature of GANs.

The training process is defined by alternating between the following steps:

$$\min_{\psi} \quad \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} \log \left[1 - D_\phi(G_\psi(\mathbf{z}))\right] \tag{2.7}$$

$$\max_{\phi} \quad \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D_\phi(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} \log \left[1 - D_\phi(G_\psi(\mathbf{z}))\right], \tag{2.8}$$

where $D_\phi(\mathbf{x})$ outputs a scalar in $[0, 1]$ to measure how confident the discriminator is that $\mathbf{x}$ belongs to the true distribution. Step (2.7) updates the generator to generate samples with high confidence returned by the discriminator, while step (2.8) updates the discriminator to assign high confidence to the training data and low confidence to the fake data generated by the generator. These steps are nearly minimizing the Kullback–Leibler divergence $D_{\text{KL}}\left[p_{\text{data}} || \left(p_{\text{data}} + p_\psi\right)/2\right] + D_{\text{KL}}\left[p_\psi || \left(p_{\text{data}} + p_\psi\right)/2\right]$ [Goodfellow et al., 2014], which provides an intuitive explanation for the training of GANs.

2.3.3 Wasserstein GAN

The GAN in the previous section minimizes the *Jensen-Shannon* (JS) divergence between two distributions. Besides this divergence, many other distances build different types of GANs. It is well known that this adversarial training is closely related to minimizing the Wasserstein

distance between the generated distribution and the original distribution[Arjovsky et al., 2017], which has the form:

$$W(P_1, P_2) = \sup_{||f||_L \leq 1} \mathbb{E}_{\mathbf{x} \sim P_1} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim P_2} [f(\mathbf{x})], \tag{2.9}$$

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \mapsto \mathbb{R}$. Since the Wasserstein distance is weaker than KL or JS distances, it is more sensible when learning distributions supported by low dimensional manifolds [Arjovsky et al., 2017]. Therefore, the Wasserstein-GAN [Arjovsky et al., 2017] defines the training process in the following steps:

$$\max_{\psi} \quad \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} D_\phi(G_\psi(\mathbf{z})) \tag{2.10}$$

$$\max_{\phi} \quad \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} D_\phi(\mathbf{x}) - \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} D_\phi(G_\psi(\mathbf{z})). \tag{2.11}$$

D_ϕ is parameterized by a deep neural network with Lipschitz-constraint to play the role of f in (2.9). Common approaches to controlling the Lipschitz-constraint include gradient clipping, gradient penalty and Spectral Normalization [Miyato et al., 2018]. When D_ϕ is optimal, G_ψ is about to minimize the Wasserstein distance between p_{data} and p_ψ. In practice, people usually run multiple steps of (2.11) before one step of (2.10) to approximately minimize the Wasserstein distance.

2.4 Energy-Based Models

2.4.1 Introduction

Energy-based models (EBMs) are likelihood-based generative models defined in terms of an unnormalized data density, also called the energy, which assigns low energy to high-probability regions in the data space. New samples are generated using the Markov chain Monte Carlo (MCMC) implicit sampling method. Combining implicit sampling with energy-

based models for generative modeling has many conceptual advantages compared to methods such as VAEs and GANs, which use explicit functions to generate samples [Du and Mordatch, 2019]. For example, since EBMs only need a model with scalar output, they require fewer model parts and parameters (no AE-structure or extra discriminator); besides, it is easier to compare low-likelihood areas with high-likelihood areas through this explicit density function. Recently, by using a neural network as the energy function, deep EBMs [Xie et al., 2016, Du and Mordatch, 2019] are able to model complex data such as natural images [Xiao et al., 2021, Gao et al., 2021], 3D shapes [Xie et al., 2020] and texts [Deng et al., 2020].

2.4.2 Maximum Likelihood Training

An Energy-based Model assumes a Gibbs distribution

$$p_\theta(\mathbf{x}) = \frac{\exp\left(-E_\theta(\mathbf{x})\right)}{Z_\theta} \tag{2.12}$$

over data $\mathbf{x} \in \mathcal{X}$. $E_\theta(\mathbf{x})$ is the energy function with parameter θ, and $Z_\theta = \int_{\mathbf{x}} \exp\left(-E_\theta(\mathbf{x})\right)$ is the normalizing constant. When $\mathbf{x}$ is an image, $E_\theta(\mathbf{x})$ is usually chosen to be a convolutional neural network with scalar output.

There are a variety of ways to train EBMs, including minimizing the KL-divergence [Du and Mordatch, 2019] or general F-divergence [Yu et al., 2020], score matching [Li et al., 2019] and contrastive estimation [Gutmann and Hyvärinen, 2010, Gao et al., 2020]. Among them, the KL divergence minimization (equivalent to maximum likelihood estimation) is most widely used, where the training is designed to minimize the negative log likelihood $L(\theta) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[-\log p_\theta(\mathbf{x})\right]$. This objective is known to have derivative

$$\frac{\partial L(\theta)}{\partial \theta} = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\frac{\partial E_\theta\left(\mathbf{x}\right)}{\partial \theta}\right] - \mathbb{E}_{p_\theta(\mathbf{x}')}\left[\frac{\partial E_\theta\left(\mathbf{x}'\right)}{\partial \theta}\right] \tag{2.13}$$

where $\mathbf{x}'$ represents a sample drawn from the EBM. However, it is difficult to draw samples

from such complex unnormalized distributions, and one typically needs to employ MCMC algorithms. One famous MCMC algorithm in high dimensional continuous state spaces is Stochastic Gradient Langevin dynamics popularized in the statistics literature in the early 90's in [Amit et al., 1991] and introduced in the deep learning literature in [Welling and Teh, 2011]. Given an initial sample $\mathbf{x}_0$, **Langevin dynamics (LD)** solves the Stochastic differential equation (SDE)

$$d\mathbf{x}_t = -\frac{1}{2}\nabla_{\mathbf{x}} E_\theta(\mathbf{x}_t)dt + d\mathbf{w}_t, \tag{2.14}$$

where $\mathbf{w}_t$ is Brownian motion. The discretized version, using the simplest Euler approximation yields:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{2}\nabla_{\mathbf{x}} E_\theta(\mathbf{x}_t) + \sqrt{\eta}\omega_t, \tag{2.15}$$

where $\omega_t \sim \mathcal{N}(0, \mathbf{I})$ and η is the step-size. Theoretically, we need to run the discretized LD with infinitely many steps and diminishing step sizes to obtain true samples. However, we usually run LD for a finite number of steps with a fixed step size in practice. After training, samples are obtained by running the same Langevin dynamics, typically with the same number of steps.

2.4.3 Understanding the MCMC Steps

Although the maximum likelihood training scheme is simple and intuitively appealing, we might not fully understand its mechanism. Since the convergence of MCMC is extremely difficult when the energy function is complicated, we cannot easily overlook the gap between running the LD in practice (usually called short-run LD) and truly obtaining samples from $p_\theta(\mathbf{x})$. Indeed, some interesting observations are made from training the EBMs through maximum likelihood. Firstly, in practice, the noise scale of LD is usually much smaller

than the correct one in (2.15), which makes the LD similar to gradient descent [Du and Mordatch, 2019]. Secondly, unless the shape of the energy function is carefully modified by introducing a base distribution as done in [Nijkamp et al., 2020a, Xiao et al., 2020], LD usually does not mix, i.e., samples obtained by running longer LD get trapped in different local modes instead of traversing between modes. Probably as a consequence, the initial points $\mathbf{x}_0$ contain information about the outcome, and therefore short-run LD is observed to be capable of reconstructing the data and interpolating different samples [Nijkamp et al., 2019]. In addition, sometimes, while we can obtain good samples by running short-run LD, the density of the EBMs can be drastically different from the true data densities (e.g., Figure 2 of [Gao et al., 2021]). These observations suggest that running short-run LD may be fundamentally different from obtaining samples from the EBMs, and the maximum likelihood explanation for the training procedure may be invalid.

[Nijkamp et al., 2019] first study the intriguing properties of short-run LD. They conjecture that the short-run LD behaves more like a generator model and try to explain the maximum likelihood training by introducing q_θ, the marginal distribution of the sample after running K steps of LD starting from a fixed initial distribution. However, they do not study q_θ with an explicit formulation. To further explore this direction, we follow their work to provide an alternative understanding of the maximum likelihood training of EBMs. In particular, we replace the LD sampling with noise-free dynamics to produce the output samples by a deterministic transformation of the initial points. In this case, we regard the dynamics as a generator model and the initial points as latent variables. Furthermore, by ensuring that the generator is invertible, we can explicitly study the density of the distribution induced by the sampling dynamics (where the randomness is entirely determined by the initial points).

2.4.4 Noise-free Sampling Dynamics as a Flow Model

To explicitly obtain the density induced by the noise-free sampling dynamics, We start by replacing the Langevin dynamics in (2.14) with the noise-free gradient descent ODE:

$$\mathbf{x}'(t) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0, \quad t \in [0, T], \tag{2.16}$$

which is guaranteed to produce an invertible map under very mild conditions on E, and we can write the continuous flow [Chen et al., 2018, Grathwohl et al., 2018]:

$$\mathbf{x}_T = G_\theta^T(\mathbf{x}_0) = \text{ODESolve}(-\nabla_{\mathbf{x}} E_\theta(\mathbf{x}(t)), \mathbf{x}_0, [0, T]). \tag{2.17}$$

Since there is no noise term, given $\mathbf{x}_0$, the process can be represented as a deterministic generator model with latent variable $\mathbf{x}_0$. We want to emphasize that T is an important component of the generator model, and we should use roughly the same T as we used in training when generating new samples. Moreover, as $G_\theta^T(\mathbf{x}_0)$ is invertible, the likelihood along the path can be obtained by the instantaneous change of variables formula, which corresponds to solving another ODE [Chen et al., 2018]:

$$\frac{\partial \log p(\mathbf{x}(t))}{\partial t} = \text{tr}\left[\nabla_{\mathbf{xx}} E_\theta(\mathbf{x}(t))\right]. \tag{2.18}$$

Then, the log likelihood of data $\mathbf{x}$ under the flow model can be computed by

$$\log p(\mathbf{x}) = \log p(\mathbf{x}_0) + \int_0^T \text{tr}\left[\nabla_{\mathbf{xx}} E_\theta(\mathbf{x}(t))\right] dt. \tag{2.19}$$

As a special case, the forward Euler solver for this equation yields:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{2} \nabla_{\mathbf{x}} E_\theta(\mathbf{x}_t), \quad t = 0, 1, \cdots, K-1, \tag{2.20}$$

with initialization $\mathbf{x}_0$ from some fixed simple distribution p_0 in $\mathbb{R}^d$, such as the standard Gaussian. In particular, $G_\theta(\mathbf{x}_0) : \mathbb{R}^d \to \mathbb{R}^d$ is a residual flow [Behrmann et al., 2019]:

$$\mathbf{x}_K = G_\theta(\mathbf{x}_0) = (I - \frac{\eta}{2}\nabla_{\mathbf{x}} E_\theta)^K(\mathbf{x}_0). \tag{2.21}$$

$G_\theta(\mathbf{x}_0)$ is guaranteed to be invertible if $\text{lip}\left(\frac{\eta}{2}\nabla_{\mathbf{x}} E_\theta\right) < 1$ [Behrmann et al., 2019]. This holds as long as $\nabla_{\mathbf{x}} E_\theta$ has bounded Lipschitz constant and the step size η is sufficiently small. However, it is still difficult to choose the step size that ensures invertibility, and therefore, we generalize $G_\theta(\mathbf{x}_0)$ to be any numerical solution to the initial value ODE problem (2.16).

To summarize, we can train the energy network E_θ by doing the gradient update (2.13) with EBM samples obtained from (2.17). After training, we can obtain new samples by running (2.17) with an initial point sampled from p_0, and compute the likelihood of a data point $\mathbf{x}$ by solving the ODE (2.16) in the *reverse* direction to find the corresponding initial point $\mathbf{x}_0$ and then apply (2.19).

2.4.5 Connection to GAN

It is well known that the maximum likelihood training of EBMs is closely related to the training of Wasserstein-GANs [Arjovsky et al., 2017], where the objective for the WGAN critic f (assuming f is 1-Lipschitz) is

$$\max_f \mathop{\mathbb{E}}_{\mathbf{x}\sim p_{\text{data}}}[f(\mathbf{x})] - \mathop{\mathbb{E}}_{\tilde{\mathbf{x}}\sim p_G}[f(\tilde{\mathbf{x}})]\,. \tag{2.22}$$

The gradient of (2.22) is (up to a sign) very similar to (2.13) except that here the negative samples are drawn from the generator G, while in (2.13), the negative samples are drawn from the EBM itself. Note that the sign does not matter as we can model the negative energy instead. W-GANs use the critic f to contrast true data and samples generated by the generator G, while EBMs use the energy function E to contrast true data and samples generated

by E itself implicitly through the gradient flow. Therefore, the maximum likelihood training of EBMs can be described as a *self-adversarial game*.

In W-GAN, after the critic is updated by (2.22), the generator G is then updated by

$$\max_{G} \mathop{\mathbb{E}}_{\tilde{\mathbf{x}} \sim p_G} [f(\tilde{\mathbf{x}})] . \tag{2.23}$$

In other words, it maximizes the critic's output for fake samples generated by G. Strictly speaking, there is no corresponding loss term in the training of EBMs, as the sampling is done by MCMC rather than deterministic mapping. However, as discussed in section 2.4.4, in practice, the sampling process can be seen as a generator model with initial points as latent variables. In this case, we actually have an *explicit* generator G_θ defined in (2.17), and therefore we can update the parameter of G_θ (which is just θ) by the following objective:

$$\min_{\theta} E_{\mathrm{sg}(\theta)}(G_\theta(\mathbf{x}_0)), \tag{2.24}$$

where $\mathbf{x}_0$ is the latent variables sampled from p_0, and $\mathrm{sg}(\cdot)$ is the stop gradient operation. Here we stop the gradient of E_θ because we only want to differentiate through the generation process.

Hence, we propose to add the extra update step for G_θ at each iteration so that we are essentially training a W-GAN whose critic and generator share the same set of parameters and conjecture that the adversarial training will improve the sample quality.

One modification is made for the implementation. Typically, when training GANs, we alternate the update of the parameters of the critic and the generator; hence, two batches of samples are generated. However, this can be slow in our case, as drawing samples requires iterative updates. Therefore, we use the same batch of samples to update E_θ and G_θ, and since we only have one set of parameters θ, it is equivalent to optimizing the following

objective:

$$\min_{\theta} E_\theta(\mathbf{x}) - E_\theta(G_{\mathrm{sg}(\theta)}(\mathbf{x}_0)) + E_{\mathrm{sg}(\theta)}(G_\theta(\mathbf{x}_0)), \quad \mathbf{x} \sim p_{\mathrm{data}}, \mathbf{x}_0 \sim p_0. \tag{2.25}$$

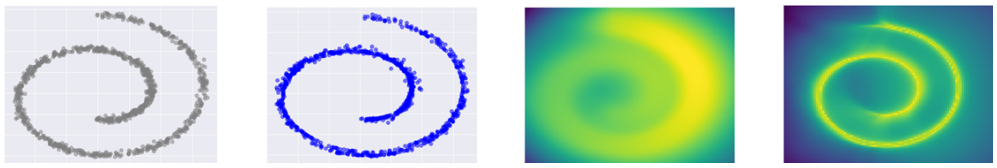
We will empirically study the effectiveness of this training objective in section 2.4.6.

2.4.6 Experiments

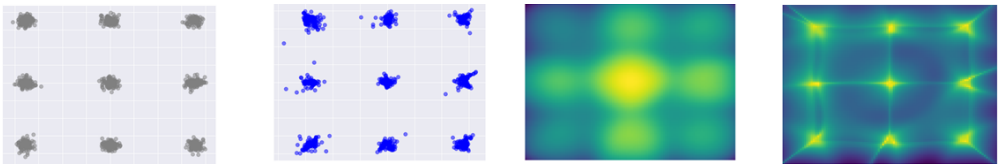
In this section, we conduct experiments to verify our proposed methods and arguments in section 2.4.4 and 2.4.5. Specifically, we train energy functions on 2d-toy data and image data by replacing the MCMC sampling with deterministic dynamics. We initialize the dynamics with noise sampled from the standard Gaussian distribution throughout the experiments. We do not use persistent sampling, which initializes the sampling dynamics at the samples generated from the model in previous training stages[Tieleman, 2008], as we want to interpret the model as a generator with fixed prior. The deterministic dynamics can be defined by (2.20), or more generally, the path to solving the ODE as in (2.17). In particular, we need to use the latter method to compute the density induced by the dynamics. More experimental detail can be found in appendix 2.4.7.

2D toy data

We use the Swiss roll and 9 Gaussian mixture grid as the true distributions, and our energy function $E_\theta : \mathbb{R}^2 \to \mathbb{R}$ is a simple neural network with several fully connected layers. We use the neural ODE formulation and solve the ODE in (2.16) with the default Dormand–Prince solver as in [Chen et al., 2018]. This process of solving the EBM-related ODE is the flow of our model. In Figure 2.1, we plot the samples obtained from solving the ODE (2.17). In addition, we plot the log density of the ODE flow and the value of the negative energy function (the unnormalized log density of the corresponding EBM) in the same figure. We



((a)) Swiss roll



((b)) Gaussian grid

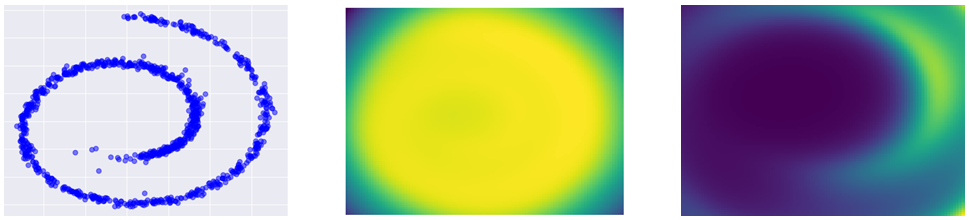
Figure 2.1: For each toy dataset, **column 1:** samples from the true data distribution; **column 2:** samples from the ODE flow; **column 3:** (unnormalized) log density of the EBM by plotting the value of $-E_\theta(\mathbf{x})$; **column 4:** log density of the ODE flow computed by (2.19). The spurious connections between components will visually disappear if we take exponential (see figure 2.3). We plot log density because the sampling dynamics directly use it.

observe that we can obtain good samples, even though the densities of the EBMs are not close to the ground truth densities. In contrast, the density function induced by the ODE flow captures the densities of the true data distributions very well.

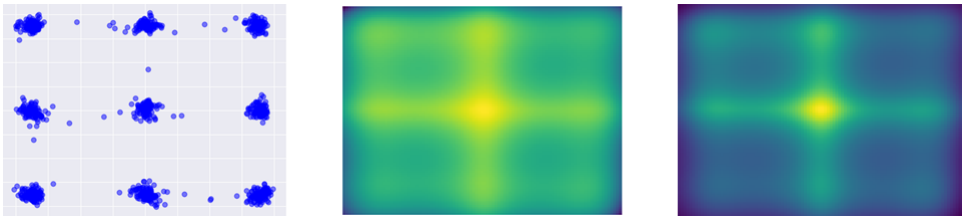
In Figure 2.2, we plot the samples and (unnormalized) density of EBMs trained with noisy sampling dynamics (finite steps of Langevin dynamics). The shown samples are generated by running the same Langevin dynamics. Note that we cannot plot the density induced by the dynamic itself as the noise term makes it not invertible. However, we have similar observations as in Figure 2.1. While the sample quality is good, the density of EBMs fails to match the true data distribution.

This suggests that even in the usual case where EBMs are trained and sampled with LD, samples are actually generated from a (noise injected) generator model instead of the EBM. Therefore, the mechanism behind maximum likelihood training of EBMs is actually training generator models implicitly defined by the sampling dynamics.

We also plot the normalized density of the EBMs and gradient flows in Figure 2.3, where we observe that the spurious high density region shown in the log density plot in Figure 2.1



((a)) Swiss roll



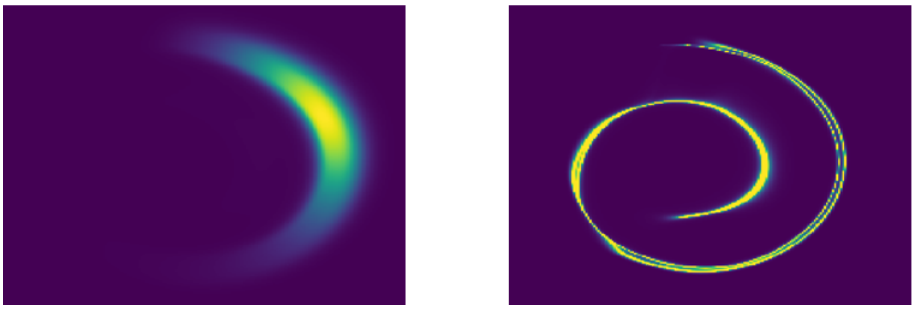
((b)) Gaussian grid

Figure 2.2: Results of EBMs trained and sampled from using noisy dynamics on toy data. For each sub-figure, we plot the **left:** samples obtained from running Langevin dynamics, **middle:** (unnormalized) log density of the EBM , and **right:** normalized density of the EBM, where the normalization constant is estimated by numerical integration.

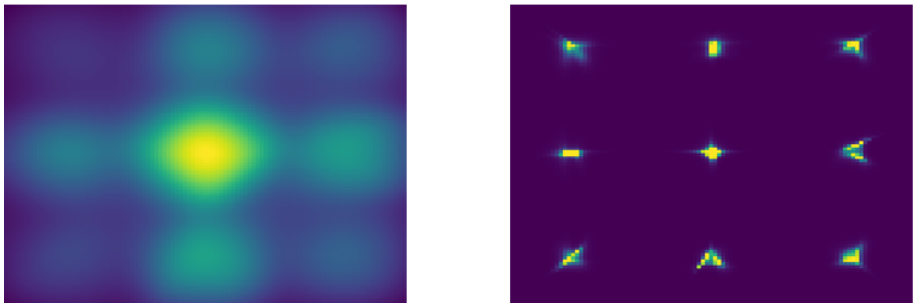
disappears, and we still find that the density of the gradient flows captures the true density much better than that of the EBMs.

These observations provide evidence that maximum likelihood training of EBMs is actually training a gradient flow model. Since the density defined by the final energy function completely fails to capture the true data density, arguments that running the sampling dynamics draws samples from the EBM is certainly incorrect; instead, we show that the dynamics *itself* is the generative model to sample from, as its density matches the shape of the true density.

In addition, we also train the ODE flows with the same formulation and structure using the maximum likelihood objective (where the likelihood is defined in (2.19)) and compare the obtained data likelihood with that of the flows trained by the EBM objective. For the ODE flows trained by maximum likelihood, the test data log-likelihood (averaged over 10000 test samples) is **-0.69** nats on Swiss roll and **-1.47** nats on Gaussian grid. On the other hand, the test data likelihood of the ODE flows trained by the EBM objective is **-0.86** nats



((a)) Swiss roll



((b)) Gaussian grid

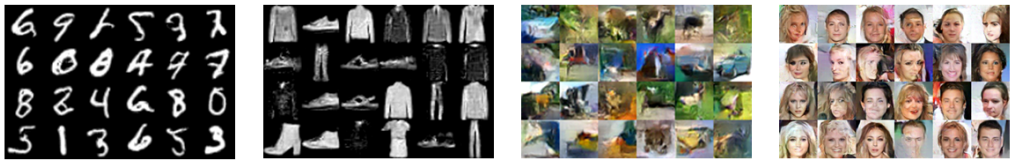
Figure 2.3: For each sub-figure, **left:** normalized density of the EBM, and **right:** density of the gradient flow.

and **-1.95** nats on these two datasets, respectively. As expected, the flows directly trained by maximizing data likelihood have higher test likelihood, but the flows trained by the EBM objective still perform reasonably well.

Image data

Experiments on toy data reveal that the maximum likelihood training of EBMs may lead to training generators or flow models. If this is true, then the noise term used in Langevin dynamics may be unnecessary or even harmful. We confirm this by studying the sample quality on standard image datasets, including MNIST [LeCun et al., 2010], Fashion MNIST [Xiao et al., 2017], CIFAR-10 [Krizhevsky et al., 2009] and CelebA [Liu et al., 2015]. For simplicity, our energy functions are simple convolutional nets instead of more complex residual networks used in [Du and Mordatch, 2019, Xiao et al., 2021], and therefore we only compare

((a)) EBMs w/ noisy dynamics



((b)) EBMs w/ noise-free dynamics

((c)) EBMs w/ noise-free dynamics + generator loss

Figure 2.4: Qualitative samples of models with noisy or noise-free sampling dynamics, and models with extra loss to update the generator defined by the dynamics.

relative performances. We use the Fréchet Inception Distance (FID) [Heusel et al., 2017] as a metric for image generation quality. FID is computed by first extracting features of a set of real images x and a set of generated images g from an intermediate layer of the Inception network [Szegedy et al., 2015]. Each set of features is fitted with a Gaussian distribution, yielding means μ_x, μ_g and co-variances matrices Σ_x, Σ_g. The FID score is defined to be the Fréchet distance between these two Gaussians:

$$\text{FID}(x, g) = \left\|\mu_x - \mu_g\right\|_2^2 + \text{Tr}\left(\Sigma_x + \Sigma_g - 2\left(\Sigma_x\Sigma_g\right)^{\frac{1}{2}}\right)$$

It is claimed that the FID score is sensitive to mode collapse and correlates well with human perception of generator quality [Lucic et al., 2018].

We train energy functions using the gradient update in (2.13), where either noisy or noise-free sampling dynamics generate the samples. For noise-free dynamics, we use the gradient descent formulation in (2.20) instead of the formulation induced by other ODE

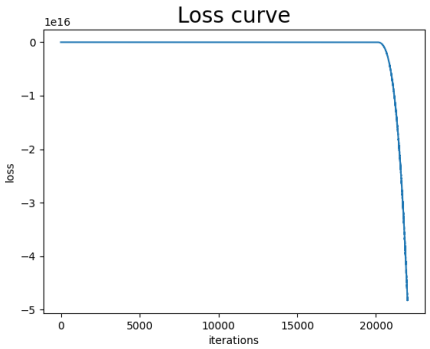
Table 2.3: FID scores on image datasets for different models

	MNIST	Fashion-MNIST	CIFAR-10	CelebA
EBM w/ noisy dynamic	15.4	61.7	70.4	69.6
EBM w/ noise-free dynamic	11.7	50.1	61.6	56.6
+ Generator loss	7.7	40.6	47.9	34.8

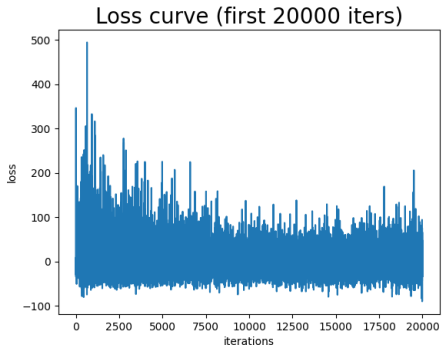
solvers because we only want to study the effect of noise while keeping all other factors the same. We reduce the noise scale for noisy sampling dynamics as it is done in almost all other works. Otherwise, the training diverges quickly. We report the FID scores [Heusel et al., 2017] in Table 2.3, and qualitative samples in Figure 2.4. We observe that EBMs trained with noise-free dynamics indeed obtain better sample quality on all datasets.

Treating the noise-free dynamics as generator models, we further apply the additional adversarial loss term for the W-GAN generator update discussed in section 2.4.5. In particular, we train the model with the loss in (2.25). We report the FID in the last line of Table 2.3, where we find the generator update significantly improves the sample quality. Qualitative samples are shown in Figure 2.4 and additional samples in the appendix 2.4.7. This experiment shows that the noise-free dynamics is indeed a generator, and we can use it to train GANs.

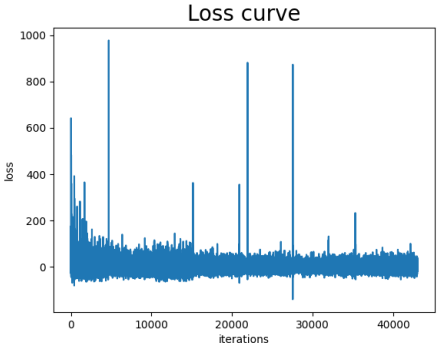
In Figure 2.5, we plot the loss curve along with the training of models with noisy or noise-free dynamics on CIFAR-10. We observe that for both models, the losses oscillate around zero, as observed in [Nijkamp et al., 2020b]. However, the model trained with noisy dynamics diverges after 20000 iterations, while training the model with noise-free dynamics is much more stable. These results suggest that the noise term in sampling dynamics may have negative effects, which further supports the argument that we should treat the model as a generator defined by the gradient of the energy instead of an EBM. In addition, we observe that adding the extra generator loss as discussed in section 2.4.5 does not affect the training stability.



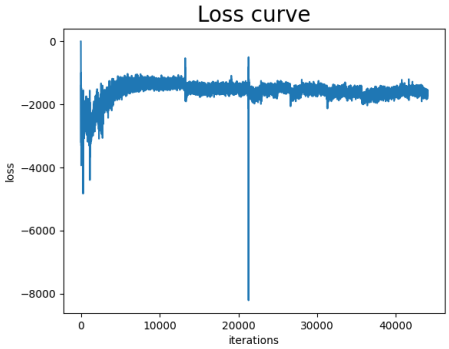
((a)) EBMs w/ noisy dynamics



((b)) EBMs w/ noisy dynamics, first 20000 iterations



((c)) EBMs w/ noise-free dynamics



((d)) EBMs w/ noise-free dynamics + generator loss

Figure 2.5: Plots of loss curves on CIFAR-10 dataset. **(a):** When sampling using the noisy MCMC, the training diverges after 20000 iterations. **(b):** For better visualization, we plot the loss curve for the first 20000 iterations. **(c):** When using noise-free dynamics, the training is more stable. **(d):** With the additional generator loss, although we see some jumps on the loss curve, the training is overall stable.

Conclusion and Discussion

We provide new insights to understand the maximum likelihood training of EBMs, and we believe that instead of training EBMs, the maximum likelihood objective actually trains generator models through a self-adversarial mechanism. The generator model is defined implicitly by the gradient of the energy network, and we study the property of the generator model by removing the noise in the MCMC sampling dynamics. We conduct experiments to justify our thoughts and make the following observations:

- On toy data, the density function induced by the invertible noise-free dynamics is close to the shape of the true data density, while the density of the EBM with the corresponding energy function fails to capture the true density.

- On image datasets, we observe that removing the noise in the LD improves sample quality and training stability.

- The sample quality can be further improved by introducing the generator update discussed in section 2.4.5, i.e., making the self-adversarial game into an adversarial game.

These observations together suggest that the mechanism behind the ML training of EBMs is to train a generator or a gradient flow model, and we can benefit from removing the noise in the sampling dynamics. Given the difficulty of running MCMC in high dimensions, we should study the convergence of MCMC sampling in high dimensions more carefully and probably focus more on training techniques without sampling if our goal is to train valid energy-based models.

2.4.7 Appendix

This section introduces detailed settings of our experiments and additional results on image data.

2D toy data

On 2D toy data, we use a 5-layer fully connected network with 256 hidden units and a swish activation function. We train our models with Adam optimizer, with a constant learning rate $1e-3$. The models are trained for 3000 iterations with batch size 800.

We draw negative samples by solving the ODE in (2.17). To do so, we use the solver implemented by [Chen et al., 2018]. We set the initial value to random samples from 2-d

MNIST	CIFAR-10	CelebA
3×3 Conv_{nf} Stride 1	3×3 Conv_{nf} Stride 1	3×3 Conv_{nf} Stride 1
4×4 $\text{Conv}_{2\times\text{nf}}$ Stride 2	4×4 $\text{Conv}_{2\times\text{nf}}$ Stride 2	4×4 $\text{Conv}_{2\times\text{nf}}$ Stride 2
4×4 $\text{Conv}_{2\times\text{nf}}$ Stride 2	4×4 $\text{Conv}_{4\times\text{nf}}$ Stride 2	4×4 $\text{Conv}_{4\times\text{nf}}$ Stride 2
4×4 $\text{Conv}_{2\times\text{nf}}$ Stride 2	4×4 $\text{Conv}_{8\times\text{nf}}$ Stride 2	4×4 $\text{Conv}_{8\times\text{nf}}$ Stride 2
Faltten, FC layer to scalar	Faltten, FC layer to scalar	4×4 $\text{Conv}_{16\times\text{nf}}$ Stride 2
		Faltten, FC layer to scalar

Table 2.4: Network structures for different datasets. nf means number of filters. For MNIST, Fashion MNIST and CelebA, nf = 32; for CIFAR-10, nf = 64. Swish activation is applied after each convolutional layer.

standard Gaussian distribution. We use the default dopri5 solver, $T \in [0, 0.2]$, and numerical error tolerance tolerance $1e-5$. After training, samples are drawn by solving the same ODE.

Image data

We resize MNIST and Fashion-MNIST to 32×32. The network structures are presented in Table 2.4. We train all models with Adam optimizer with a learning rate $5e-4$ and batch size 64. As we mention in the main text, the training of EBMs with noisy dynamics is unstable, and it will diverge after a certain number of iterations. This is also observed in [Du and Mordatch, 2019] and [Xiao et al., 2021]. Therefore, we follow their setting to train the EBMs until divergence. For EBMs trained with noise-free dynamics, we found the training to be more stable. We set the number of training iterations similar to EBMs with noisy dynamics. In particular, we train 8000 iterations for MNIST and Fashion-MNIST, 40000 iterations for CIFAR-10, and 30000 iterations for CelebA. To draw negative samples, we set the step size to be 0.1 and the number of steps to be 40 for MNIST/Fashion-MNIST and 60 for CIFAR-10 and CelebA. For the noisy sampling dynamics, we set the noise scale to be 0.1.

For the extra GAN loss, we need to store the gradient while running the gradient descent steps (2.20). This can be done by setting the create graph option when computing the gradient in PyTorch's auto differential package [Paszke et al., 2019].

Additional results on image data

In Figure 2.6, 2.7 and 2.8, we present additional qualitative samples corresponding to Figure 2.4 in the main text.

Figure 2.6: Additional samples from EBMs w/ noisy dynamics

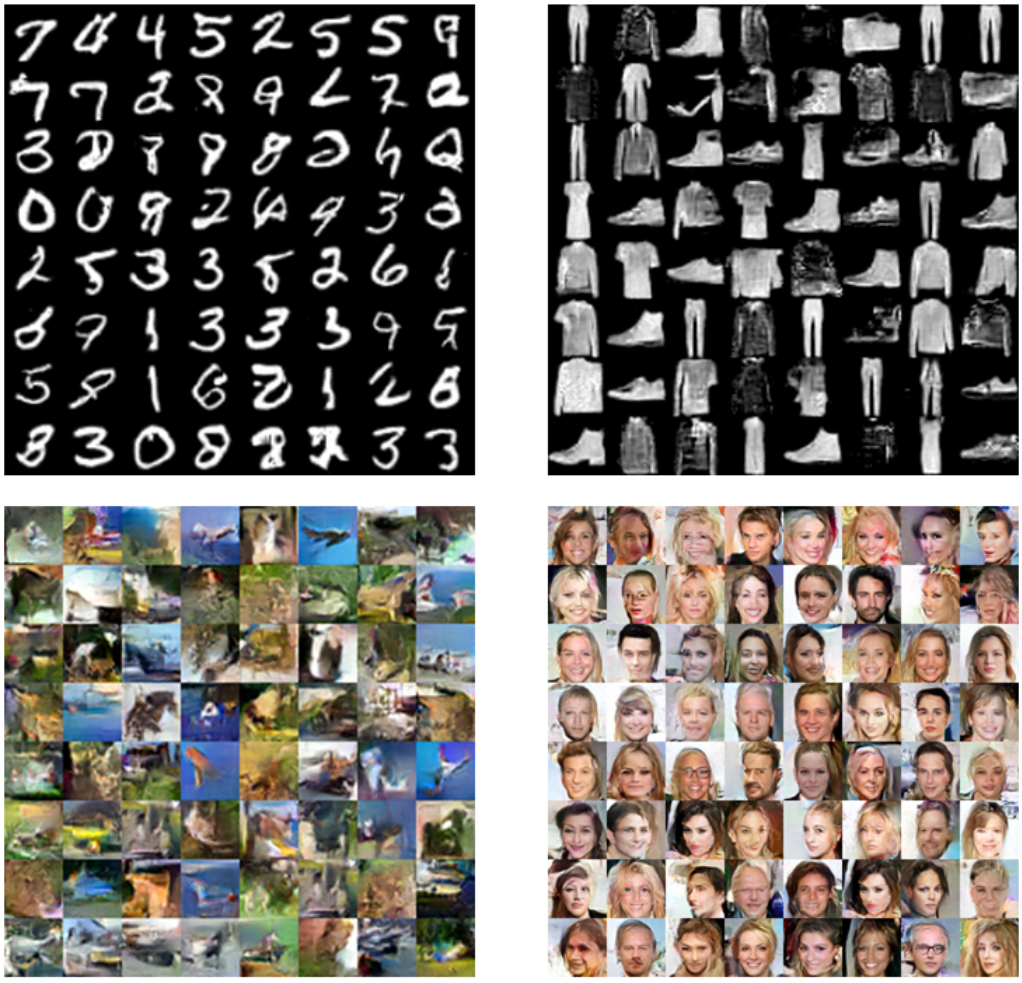


Figure 2.7: Additional samples from EBMs w/ noise-free dynamics

Figure 2.8: Additional samples from EBMs w/ noise-free dynamics plus extra generator loss

CHAPTER 3

HYBRIDS OF DEEP GENERATIVE MODELS

3.1 Overview

Although deep generative models exhibit good results in various tasks, it is necessary to think about new ways to fix drawbacks and improve the generation quality. As generative models tend to show different advantages and disadvantages within their variations, one way to improve is to consider various hybrids between them. In this chapter, we propose two hybrids:

1. we combine Flow-based models and Variational Auto-Encoders to improve the generation quality of AE-based generative models (chapter 3.2);

2. we combine energy-based models with other likelihood-based generative models to obtain better samples (chapter 3.3).

We present detailed experimental results to show the improvement of our methods compared to the original models. We hope these will provide helpful guidance in future research about new deep generative models.

3.2 Generative Latent Flow

3.2.1 Introduction

It has been shown that although VAEs are easy to train, their generation quality still lies far below that of GANs, as they tend to generate blurry images [Dosovitskiy and Brox, 2016]. Therefore, many methods are proposed to improve the VAE's generation quality [Tolstikhin et al., 2017, Makhzani et al., 2015]. In general, in order for an AE based model with the

encoder-decoder structure to generate samples resembling the training data distribution, two criteria need to be ensured:

1. the decoder is able to produce a good reconstruction of a training image given its encoded latent variable $\mathbf{z}$;

2. the empirical latent distribution $q(\mathbf{z})$ is close to the prior $p(\mathbf{z})$.

In VAEs, the empirical latent distribution is often called the aggregated or the marginal posterior: $q(\mathbf{z}) = \mathbb{E}_{\mathbf{x}\sim p_{data}}[q(\mathbf{z}|\mathbf{x})]$. While the reconstruction loss mainly drives criterion 1, satisfying criterion 2 is more complicated. Intuitively, criterion 2 can be achieved by designing mechanisms that modify the empirical latent distribution $q(\mathbf{z})$, or modify the prior $p(\mathbf{z})$. In this chapter we focus on modifying the prior.

Whereas the original VAE uses a standard Gaussian prior, it can be extended by introducing a learnable parameterized prior distribution. Several studies have pursued this direction [Tomczak and Welling, 2017, Klushyn et al., 2019, Bauer and Mnih, 2018], some of which use a normalizing flow parameterization, where the prior is modeled as a trainable continuous bijective transformation of the standard Gaussian distribution. We carefully study this method and make the surprising novel observation that it is necessary to increase the reconstruction loss weight to produce high-quality samples. This corresponds to decreasing the variance of the observational noise of the generative model at each pixel, where we are assuming the data distribution is factorial Gaussian conditioned on the output of the decoder, which yields the MSE as the reconstruction loss, as in section 2.1.3. It is important to note that increasing this weight alone without access to a trainable prior does not consistently improve generation quality.

We show that we approach a vanishing noise limit corresponding to a deterministic auto-encoder as this weight increases. This leads to a new algorithm we call Generative Latent Flow (GLF), which combines a deterministic auto-encoder that learns a mapping to and from a latent space and a normalizing flow that matches the standard Gaussian to the distribution

of latent variables of the training data produced by the encoder. Our contributions are summarized as follows: i) we carefully study the effects of equipping VAEs with a normalizing flow prior to image generation quality as the weight of reconstruction loss increases. ii) Based on this finding, we introduce Generative Latent Flow, which uses auto-encoders instead of VAEs. iii) Through standard evaluations, we show that our proposed model achieves state-of-the-art sample quality among competing AE-based models and has the additional advantage of faster convergence.

3.2.2 VAEs with a Normalizing Flow Prior

If we introduce a normalizing flow f_η for the prior distribution, then the prior p_η becomes $p_\eta(\mathbf{z}) = p_0(f_\eta(\mathbf{z}))\left|\det\left(\frac{\partial f_\eta(\mathbf{z})}{\partial \mathbf{z}}\right)\right|$, where p_0 is the standard Gaussian density. Substituting this prior into the training objective of VAEs (2.1), we obtain the modified ELBO for VAEs with flow prior:

$$\mathbf{ELBO}_{\text{Flow}} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_0\left(f_\eta(\mathbf{z})\right) + \log\left|\det\left(\frac{\partial f_\eta(\mathbf{z})}{\partial \mathbf{z}}\right)\right| - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \tag{3.1}$$

The second and third terms together are the log-likelihood of z under the prior distribution modeled by the flow. The last term corresponds to the entropy of the posterior distribution returned by the encoder. Both the VAE and the normalizing flow are trained by minimizing $-\text{ELBO}_{\text{Flow}}$.

Previous work on VAEs with a flow prior did not consider tuning the variance parameter σ in $p_\theta(\mathbf{x}|\mathbf{z})$ (which means the reconstruction loss and the KL loss are weighted equally) as they focused on comparing the obtained log-likelihoods with those from plain VAEs. However, we observe that when $\sigma = 1$, VAEs with a flow prior do not significantly improve the generation quality (see section 3.2.4 and Table 3.1). The reason might be that although $p(\mathbf{z})$ is matched with $q(\mathbf{z})$ due to the flow transformation, the decoder is not good enough to reconstruct

sharp images (i.e., criterion 1 is not ensured). In contrast, we find that decreasing σ in the objective produces samples with significantly higher quality. Intuitively, a larger weight on the reconstruction loss forces the decoder to produce sharper reconstructed images, while the normalizing flow prior is flexible enough to match the latent distribution.

To the best of our knowledge, we are the first to observe such a relation between the weight of the reconstruction loss and the generation quality of VAEs with flow prior. As σ decreases, two things occur, as demonstrated empirically in Section 3.2.4. First, the estimated variances from the encoder decrease, and second the generation quality consistently improves. In the limit, as the posterior variance goes to zero, we obtain a deterministic auto-encoder and a normalizing flow used to match the distribution of the latent variables obtained from the data. This is described in detail in the next section.

3.2.3 Generative Latent Flow (GLF)

In an auto-encoder, $\mathbf{z} = E_\phi(\mathbf{x})$ is deterministic so that $q_\phi(\mathbf{z}|\mathbf{x})$ becomes a delta distribution and the entropy term in (3.1) can be removed. The overall training loss is then given by

$$\mathcal{L}_\beta^{\text{reg}}(\eta, \phi, \theta) = \frac{1}{N} \sum_{i=1}^{N} \Big(\beta \mathcal{L}_{\text{recon}}\big(\mathbf{x}_i, G_\theta(E_\phi(\mathbf{x}_i))\big) + \mathcal{L}_{\text{NLL}}\big(f_\eta(E_\phi(\mathbf{x}_i))\big) \Big), \tag{3.2}$$

where $\mathcal{L}_{\text{recon}}\big(\mathbf{x}_i, G_\theta(E_\phi(\mathbf{x}_i))\big)$ corresponds to $-\log p_\theta(\mathbf{x}_i|z_i)$ with $\sigma = 1$ and $\mathcal{L}_{\text{NLL}}\big(f_\eta(E_\phi(\mathbf{x}_i))$ to $-\log p_\eta(z)$ in the negative of (3.1). In particular, We use β to control the weights between the reconstruction-loss and the NLL-loss, which plays the role of $\frac{1}{\sigma^2}$ in $\log p_\theta(\mathbf{x}_i|z_i)$. (please check section 2.1.3)

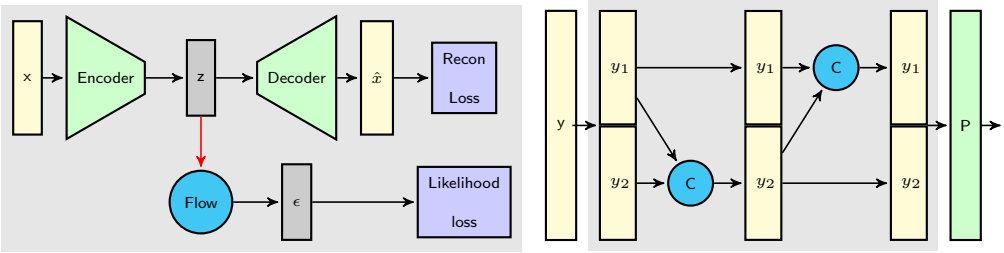
As noted in section 3.2.2, larger β's yield better results, in which case the parameters of the auto-encoder are affected almost exclusively by $\mathcal{L}_{\text{recon}}$, while $\mathcal{L}_{\text{NLL}}$ only affects the parameters η of the normalizing flow. Therefore, optimizing (3.2) with extremely large β is

approximately equivalent to optimizing

$$\mathcal{L}(\eta, \phi, \theta) = \frac{1}{N} \sum_{i=1}^{N} \Big(\mathcal{L}_{\text{recon}}\big(\mathbf{x}_i, G_\theta(E_\phi(\mathbf{x}_i))\big) + \mathcal{L}_{\text{NLL}}\big(f_\eta(\text{sg}\left[E_\phi(\mathbf{x}_i)\right])\big) \Big), \tag{3.3}$$

where sg[·] is the stop gradient operation. The weight parameter β is no longer needed because the two loss terms affect independent sets of parameters. We name the model trained by (3.3) as **Generative Latent Flow** (GLF), to highlight that our model applies normalizing flows on latent variables. See Figure 3.1(a) for an illustration of the GLF model. We call the model trained by (3.2), without stopped gradient, **regularized GLF**, since the flow acts as a regularizer on the encoder.

Note that when stopping the gradients, GLF can also be trained in two stages, namely an auto-encoder is trained first, and then the flow is trained to map the distribution of estimated latent variables to the standard Gaussian. Empirically, we find that the two-stage training strategy leads to similar performance, so we only focus on one-stage training as it follows our derivation more naturally.



((a)) ((b))

Figure 3.1: (a) Illustration of the GLF model. The red arrow contains a stop gradient operation (see section 3.2.3). (b) Structure of one flow block. The input is split into two parts $y = (y_1, y_2)$, go through two coupling layers C (see section 2.2.3). Finally, a random permutation P is applied.

Necessity of Stopping The Gradients

The stop gradient operation is necessary when using deterministic auto-encoders. In VAEs with flow prior, the entropy term, which encourages the posterior to have large variance, prevents the degeneracy of the $\mathbf{z}$'s. However, when using a deterministic encoder, if we let gradients of $\mathcal{L}_{\text{NLL}}$ backpropagate into the latent variables, training can lead to degenerate z's produced by the encoder E_ϕ. This is because f_η has to transform the $\mathbf{z}$'s to unit Gaussian noise, so the smaller the scale of the $\mathbf{z}$'s, the larger the magnitude of the log-determinant of the Jacobian. Since there is no constraint on the output scale of E_ϕ, the Jacobian term can dominate the entire objective. While the latent variables cannot become exactly 0 because of the reconstruction loss, the tiny scale of $\mathbf{z}$ may cause numerical issues that lead to severe fluctuations. In summary, we stop the gradient of $\mathcal{L}_{\text{NLL}}$ at the latent variables, preventing it from modifying the values of $\mathbf{z}$ and affecting the parameters of the encoder.

Currently we have 3 losses:

1. loss (3.1) for VAEs with flow prior model;
2. loss (3.2) for regularized-GLF;
3. loss (3.3) for GLF.

We conduct experiments to compare them in Section 3.2.4, and the results show that regularized GLF is unstable, while the performance of VAEs with flow prior converges to GLF, which has the best performance among all of them, as $\sigma \to 0$.

3.2.4 Experiments

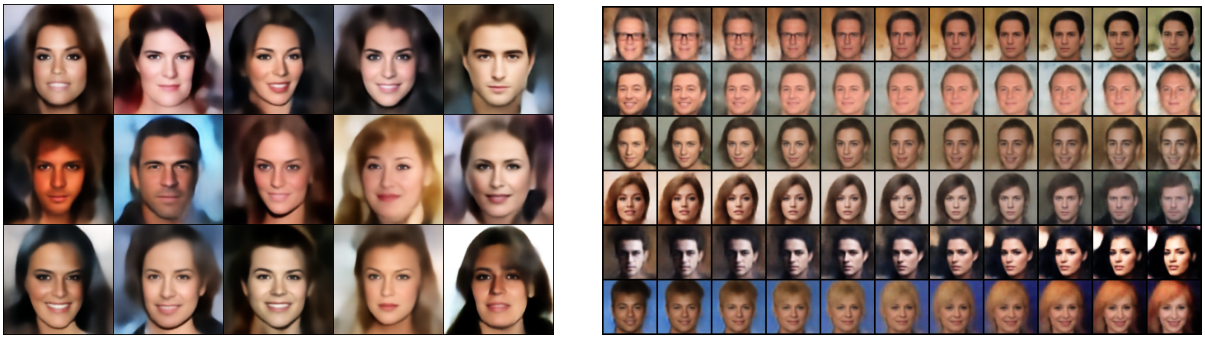
To demonstrate the performance of our method, we present both quantitative and qualitative evaluations on four commonly used datasets for generative models: MNIST [LeCun et al., 2010], Fashion MNIST [Xiao et al., 2017], CIFAR-10 [Krizhevsky et al., 2009] and CelebA

[Liu et al., 2015]. We use 20-dimensional latent variables for MNIST and Fashion MNIST throughout the experiments and 64-dimensional latent variables for CIFAR-10 and CelebA. [Lucic et al., 2018] adopted a common network architecture based on InfoGAN [Chen et al., 2016] to evaluate GANs. In order to make fair comparisons without designing arbitrarily large networks to achieve better performance, we use the generator architecture of InfoGAN as our decoder's architecture, and the encoder is set symmetric to the decoder. For details of the AE network structures, see Appendix 3.2.6. For the flow applied to the latent variables, we use four affine coupling blocks as shown in Figure 3.1(b), where each block contains three fully connected layers, each with k hidden units. For MNIST and Fashion MNIST, $k = 64$, while for CIFAR-10 and CelebA, $k = 256$. Note that the flow only adds a small parameter overhead on the auto-encoder (less than 3%).

Metrics

Estimated test data log-likelihood is a popular metric to evaluate models based on VAEs. It is not trivial to estimate the log-likelihood obtained from GLF, as it uses a deterministic auto-encoder. More importantly, as shown in [Grover et al., 2018, Theis et al., 2015], likelihood is not well correlated with sample quality. We use the Fréchet Inception Distance (FID) [Heusel et al., 2017] as a metric for image generation quality. Recently, [Sajjadi et al., 2018] proposed using Precision and Recall for Distributions (PRD) which can assess both the quality and diversity of generated samples. Therefore, we also include PRD in our studies.

((a)) MNIST ((b)) Fashion MNIST ((c)) CIFAR-10 ((d)) CelebA



((e)) CelebA-HQ ((f)) Noise interpolation

Figure 3.2: (a)-(e): Randomly generated samples from our method trained on different datasets. (f): Random noise interpolation on CelebA.

Table 3.1: FID scores obtained from different models. We executed 10 independent trials for our reported results and reported the mean and standard deviation of the FID scores. Each trail computes the FID between 10k generated images and 10k real images.

	MNIST	Fashion	CIFAR-10	CelebA
VAE	28.2 ± 0.3	57.5 ± 0.4	142.5 ± 0.6	71.0 ± 0.5
WAE-GAN	12.4 ± 0.2	31.5 ± 0.4	93.1 ± 0.5	66.5 ± 0.7
Two-Stage VAE	10.9 ± 0.7	26.1 ± 0.9	96.1 ± 0.9	65.2 ± 0.8
RAE + GMM	10.8 ± 0.1	25.1 ± 0.2	91.6 ± 0.6	57.8 ± 0.4
VAE+flow prior	28.3 ± 0.2	51.8 ± 0.3	110.4 ± 0.5	54.3 ± 0.3
VAE+flow posterior	26.7 ± 0.3	55.1 ± 0.3	143.6 ± 0.8	67.9 ± 0.3
GLF (ours)	**8.2** $\pm$ 0.1	**21.3** $\pm$ 0.2	**88.3** $\pm$ 0.4	**53.2** $\pm$ 0.2
GLANN with perceptual loss	8.6 ± 0.1	13.0 ± 0.1	46.5 ± 0.2	46.3 ± 0.1
GLF+perceptual loss (ours)	**5.8** $\pm$ 0.1	**10.3** $\pm$ 0.1	**44.6** $\pm$ 0.3	**41.8** $\pm$ 0.2

Results

Table 3.1 summarizes the main results of this work. We compare the FID scores obtained by GLF with the scores of the VAE baseline and several existing AE-based models that are claimed to produce high-quality samples. Instead of directly citing their reported results, we re-ran the experiments because we want to evaluate them under standardized settings so that all models adopt the same AE architectures, latent dimensions, and image pre-processing. We report the results of VAE+flow prior/posterior with $\beta = 1$. For other methods, we largely follow their proposed experimental settings. Details of each experiment are presented in Appendix 3.2.6.

Note that WAE [Tolstikhin et al., 2017] authors propose two variants, namely WAE-GAN and WAE-MMD. We only report WAE-GAN results, as we found it consistently outperforms WAE-MMD. Note also that GLANN [Hoshen et al., 2019] obtains impressive FID scores, but it uses perceptual loss [Johnson et al., 2016] as the reconstruction loss, while other models use MSE loss. The perceptual loss is obtained by feeding both training images and reconstructed images into a pre-trained network such as VGG [Simonyan and Zisserman, 2014] and computing the L_1 distance between some of the intermediate layers' activation. We also train our method with perceptual loss and compare it with GLANN in the last two rows of Table 3.1.

As shown in Table 3.1, our method obtains significantly lower FID scores than competing AE-based models across all four datasets. In particular, GLF significantly outperforms VAE+flow prior with the default setting of $\beta = 1$. More detailed analysis and comparison between the two methods will be made in Section 3.2.4. We also confirm that VAE+flow posterior cannot improve generation quality. Perhaps the competing model with the closest performance to ours is RAE+GMM [Ghosh et al., 2019], which trains a regularized auto-encoder [Alain and Bengio, 2014] and fits a mixture of Gaussian distribution on the latent space. It shares some similarities with GLF in that both methods fit the density of the latent

variables of an AE explicitly. In Table 3.2, we combine our reported results of AE-based models, and the FID scores of GANs cited from [Lucic et al., 2018]. In [Lucic et al., 2018], the authors conduct standardized and comprehensive evaluations of representative GAN models with large-scale hyper-parameter searches, and therefore, their results can serve as a strong baseline. The results indicate that our method's generation quality is competitive with carefully tuned GANs.

Table 3.2: FID score comparisons of GANs and various AE based models

	MNIST	Fashion	CIFAR-10	CelebA
MM GAN	9.8 ± 0.9	29.6 ± 1.6	72.7 ± 3.6	65.6 ± 4.2
NS GAN	6.8 ± 0.5	26.5 ± 1.6	58.5 ± 1.9	55.0 ± 3.3
LSGAN	7.8 ± 0.6	30.7 ± 2.2	87.1 ± 47.5	53.9 ± 2.8
WGAN	6.7 ± 0.4	21.5 ± 1.6	55.2 ± 2.3	41.3 ± 2.0
WGAN GP	20.3 ± 5.0	24.5 ± 2.1	55.8 ± 0.9	30.3 ± 1.0
DRAGAN	7.6 ± 0.4	27.7 ± 1.2	69.8 ± 2.0	42.3 ± 3.0
BEGAN	13.1 ± 1.0	22.9 ± 0.9	71.4 ± 1.6	38.9 ± 0.9
VAE	28.2 ± 0.3	57.5 ± 0.4	142.5 ± 0.6	71.0 ± 0.5
WAE-GAN	12.4 ± 0.2	31.5 ± 0.4	93.1 ± 0.5	66.5 ± 0.7
Two-Stage VAE	10.9 ± 0.7	26.1 ± 0.9	96.1 ± 0.9	65.2 ± 0.8
RAE + GMM	10.8 ± 0.1	25.1 ± 0.2	91.6 ± 0.6	57.8 ± 0.4
GLANN (with perceptual loss)	8.6 ± 0.1	13.0 ± 0.1	46.5 ± 0.2	46.3 ± 0.1
VAE+flow prior	28.3 ± 0.2	51.8 ± 0.3	110.4 ± 0.5	54.3 ± 0.3
VAE+flow posterior	26.7 ± 0.3	55.1 ± 0.3	143.6 ± 0.8	67.9 ± 0.3
GLF (ours)	8.2 ± 0.1	21.3 ± 0.2	88.3 ± 0.4	53.2 ± 0.2
GLF+perceptual loss (ours)	5.8 ± 0.1	10.3 ± 0.1	44.6 ± 0.3	41.8 ± 0.2

Next, we report the precision and recall (PRD) evaluation of samples on each dataset in Table 3.3. We compare WAE-GAN, Two-stage VAE, RAE+GMM, and GLANN (with perceptual loss). In the case of FID scores, for GLANN, we directly cite their reported results, and we compute the results of other models. The training settings of different models are introduced in Appendix 3.2.6. The two numbers in each entry are $F_8, F_{\frac{1}{8}}$ that capture recall and precision, respectively. See [Sajjadi et al., 2018] for more details. Higher numbers are preferred.

Some qualitative results are shown in Figure 3.2. Besides samples of the datasets used

Table 3.3: Evaluation of sample quality by precision/recall.

	MNIST	Fashion	CIFAR-10	CelebA
WAE-GAN	(0.978, 0.956)	(0.901, 0.837)	(0.414, 0.723)	(0.501, 0.512)
Two-stage VAE	(0.982, 0.977)	(**0.937**, 0.845)	(0.382, 0.669)	(0.452, 0.558)
RAE+GMM	(**0.988**, 0.971)	(0.922, 0.924)	(0.370, 0.733)	(0.333, 0.445)
GLF (ours)	(0.982, **0.985**)	(0.932, **0.926**)	(**0.485**, **0.767**)	(**0.542**, **0.618**)
GLANN+perceptual loss	(0.971, 0.979)	(0.985, 0.963)	(**0.860**, 0.825)	(0.574, 0.681)
GLF+perceptual loss (ours)	(**0.990**, **0.992**)	(**0.987**, **0.980**)	(0.765, **0.845**)	(**0.760**, **0.778**)

for quantitative evaluation, samples of CelebA-HQ [Karras et al., 2017] with the larger size of 256×256 are also included in Figure 3.2(e) to show our method's ability to scale up to images with higher resolution. Qualitative results show that our model can generate sharp and diverse samples in each dataset. Figure 3.2(f) shows CelebA images generated by linearly interpolating two sampled random noise vectors. The smooth and natural transition shows that our model can generate samples not seen during training.

Nearest Neighbors of Generated Samples in the Training Set

Quantitative measurements of sample quality, such as FID score and Precision/Recall, can be minimized by letting the generative model memorize the training set. Therefore, we show the smooth transition of noise interpolation in Figure 3.2 to provide evidence that our model can generalize, i.e., it generates samples that have not been seen during training.

We randomly generated samples from the models trained on MNIST and CelebA datasets. Then we present the 5 nearest neighbors of each generated sample in the training set. The nearest neighbor is defined in terms of L_2 distance. Results are shown in Figure 3.3. By inspecting the figure, we can easily differentiate each generated sample from the closest training data, which indicates that our model generalizes well.

((a)) MNIST

((b)) CelebA

Figure 3.3: Some randomly generated samples are presented in the **leftmost** column in each picture. The other 5 columns of each picture show the top 5 nearest neighbors of the corresponding sample in the training set.

More qualitative results

In Figure 3.4, we show more samples of each dataset generated by GLF, using either MSE or perceptual loss as reconstruction loss. Next, in Figure 3.5, we show samples of CelebA-HQ datasets from GLF trained with perceptual loss. Finally, in Figure 3.6, we show examples of interpolations between two randomly sampled noises on CelebA from GLF trained with perceptual loss.

Comparisons: GLF vs. Regularized GLF and VAE+flow Prior.

As discussed in section 3.2.1 and section 3.2.2, we underline the novel finding regarding the relationship between the weight on the reconstruction loss and the sample quality of VAEs

((a)) MNIST ((b)) Fashion MNIST ((c)) CIFAR-10 ((d)) CelebA

((e)) MNIST ((f)) Fashion MNIST ((g)) CIFAR-10 ((h)) CelebA

Figure 3.4: (a)-(d) Randomly generated samples from our method with MSE loss. (e)-(h) Randomly generated samples from our method with perceptual loss.

Figure 3.5: Randomly generated samples from our method with perceptual loss on CelebA-HQ dataset

Figure 3.6: Noise interpolation on CelebA

with flow prior. In this section, we present detailed experiments on this relation. We train VAEs+flow prior on CIFAR-10 for different choices of β, plus one with a learnable β [Dai and Wipf, 2019]. We record the progression of FID scores of these models in Figure 3.7a. In Figure 3.7b, we plot the entropy term, which is the last term in (3.1), the objective of VAE+flow prior. The entropy is expressed as $-\sum_{j=1}^{d} \log(\sigma_j)/2$, where σ_j is the standard deviation of the approximate posterior on the j^{th} latent variable. Higher entropy means that the latent variables have lower variances. In Figure 3.7c, we plot the NLL loss. We omit the results for $\beta = 1$ because the obtained FID scores are too high to fit the scale of the plot. We use batch size 256 and an initial learning rate of 10^{-3} for both AE and flow. We train all models for 500 epochs with learning rates decaying by a factor of 2 every 150 epochs.

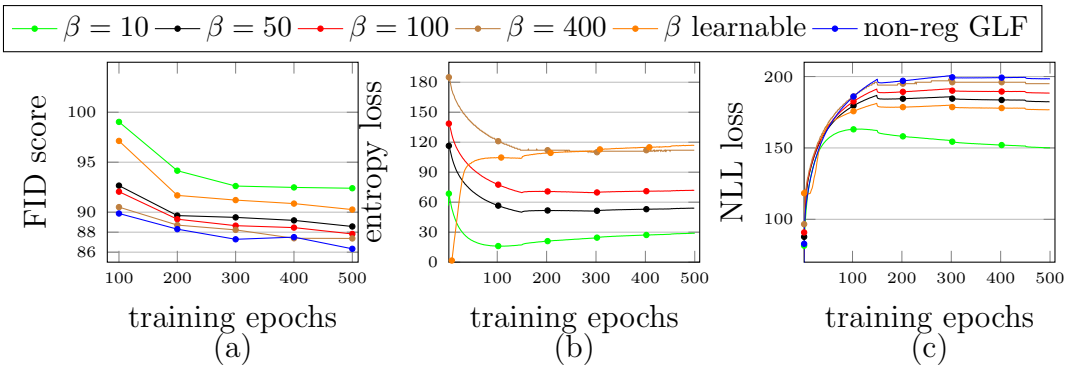


Figure 3.7: (a) Record of FID scores on CIFAR-10 for VAEs+flow prior with different values of β and GLF. (b) Record of entropy losses for corresponding models. (c) Record of NLL losses for corresponding models.

From Figure 3.7a, we observe the trend that the generation quality measured by FID scores improves as β increases. We also observe that as β increases, the performance gap between VAE+flow prior and GLF closes, indicating that GLF captures the limiting behavior of VAE+flow prior. We also find that learnable β is not effective, probably due to the relatively small values of β at the early stages of training. When β is large, as indicated by Figure 3.7b, the posterior variances of VAEs become very small, so that effectively, we are training an AE. For example, as shown in Figure 3.7b, when $\beta = 400$, the corresponding

average posterior variance is around 10^{-4}. This motivates us to use a deterministic auto-encoder in GLF, which, as we have said above, can be seen as the vanishing observational variance limit of VAE+flow prior. It is important to note that the relation between β and generation quality only exists for VAEs with a trainable prior (such as Normalizing Flows), as we empirically find that increasing β on plain VAEs leads to worse FID scores.

As discussed in section 3.2.3, training regularized GLF is unstable because of the degeneracy of the latent variables driven by the NLL loss. We empirically study the effect of latent regularization as a function of β and present results in Figure 3.8. The NLL loss completely dominates the learning signal for low values of $\beta = 1$ and 10, and the reconstruction loss quickly diverges. Therefore, we omit them in the plot. For larger values of $\beta = 50, 100, 400$ we observe that the NLL loss decreases to a negative value of a huge magnitude, and although overall performance is reasonable, it oscillates quite strongly as training proceeds. In contrast, for GLF, where the Flow does not modify z, the NLL loss does not degenerate, resulting in stable FID scores improvements as training progresses.

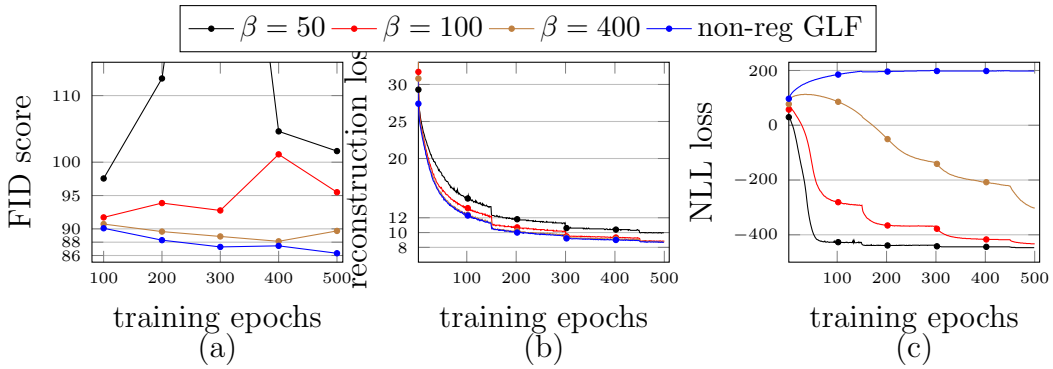


Figure 3.8: (a) Record of FID scores on CIFAR-10 for regularized GLF with different values of β and GLF. $\beta = 1$ and 10 are omitted because they lead to divergence in the reconstruction loss. (b) Record of reconstruction loss for the corresponding models. (c) Record of NLL loss for the corresponding models.

In contrast to regularized GLF, which uses a deterministic encoder, no degeneracy in the latent variables is observed for VAE+flow prior, thanks to the noise introduced in the stochastic encoder and the corresponding entropy term. Indeed, Figure 3.7c shows that the

training of VAE+flow prior does not over-fit the NLL loss, as opposed to regularized GLF, where severe over-fitting to NLL loss occurs as shown in Figure 3.8c. Comparing Figure 3.7a and 3.8a, we observe that unlike regularized GLF, VAE+flow prior does not suffer from divergence or fluctuations in FID scores, even with relatively small β. In summary, FID scores show that regularized GLF is unstable, while as β increases, the performance of VAE+flow prior converges to that of GLF. Note that although GLF only slightly outperforms VAE+flow prior, even when β is very large, it has the advantage that there is no need to tune β.

Training time

Besides better performance, our method also has the advantage of faster convergence among competing methods such as GLANN and Two-stage VAE. In Table 3.4, we compare the number of training epochs to obtain the FID scores in Table 3.1. We also compare the per epoch training clock time in Table 3.5. Note that the per epoch training time is longer for methods using perceptual loss because VGG activations need to be computed. These two tables show that GLF needs a much shorter training time than the two competing methods. In GLF, training the flow does not add much computational time due to the low dimensionality.

Table 3.4: Number of training epochs for Two-stage VAE, GLANN, and GLF

	MNIST/Fashion	CIFAR-10	CelebA
Two-stage VAE First/Second	400/800	1000/2000	120/300
GLANN First/Second	500/50	500/50	500/50
GLF	100	200	40

3.2.5 Conclusion

In this chapter, we introduce Generative Latent Flow, a novel generative model which uses an auto-encoder to learn a latent space from training data and a normalizing flow to match the

Table 3.5: Per-epoch training time in seconds

	MNIST/Fashion	CIFAR-10	CelebA
2-stage VAE 1st/2nd	5/2	6/2	60/28
GLF	10	13	108
GLANN with perceptual loss	14	16	292
GLF with perceptual loss	16	19	343

distribution of the latent variables with the prior. Under standardized evaluations, our model achieves state-of-the-art results in image generation quality and diversity among several recently proposed auto-encoder-based models. While we are not claiming that our GLF model is superior to GANs, we believe that it opens the door to realizing the potential of AE-based models to produce high-quality samples just as GANs do. Furthermore, our proposed model is motivated by our novel finding on the relation between large reconstruction weight and generation quality of VAEs with normalizing flow prior. The finding itself is crucial, as it can potentially motivate future work to study the trade-off between reconstruction and density matching in the objective of VAEs with learnable priors.

3.2.6 Appendix

Network Architectures

In this section, we provide Table 3.6 that summarizes the auto-encoder network structure. The network structure is adopted from InfoGAN[Chen et al., 2016], and the difference between the networks we used for each dataset is the size of the fully connected layers, which depends on the size of the image. All convolution and deconvolution layers have stride = 2 and padding = 1 to ensure the spatial dimension decreases/increases by a factor of 2. M is simply the size of an input image divided by 4. Specifically, for MNIST and Fashion MNIST, $M = 7$; for CIFAR-10, $M = 8$; for CelebA, $M = 16$. BN stands for batch normalization.

For VAEs, the final FC layer of the encoder will have doubled output size to return both the mean and standard deviation of latent variables.

Table 3.6: Network structure for auto-encoder based on InfoGAN

Encoder	Decoder
Input x	Input z
4×4 Conv_{64}, ReLU	FC nz $\rightarrow$ 1024, BN, ReLU
4×4 Conv_{128}, BN, ReLU	FC 1024 $\rightarrow 128 \times M \times M$, BN, ReLU
Flatten, FC $128 \times M \times M \rightarrow 1024$, BN, ReLU	4×4 Deconv_{64}, BN, ReLU
FC 1024 $\rightarrow$ nz	4×4 Deconv_{128}, Sigmoid

Experimental Settings

In this section, we present the details of our experimental settings for results in Table 3.1. Since the settings for MNIST and Fashion MNIST are the same, we only mention MNIST for simplicity. For GLANN, we directly cite the results from [Hoshen et al., 2019], as their experimental settings are very similar to ours.

We use the original images in the training sets for MNIST, Fashion MNIST, and CIFAR-10. For CelebA, we follow the same pre-processing as in [Lucic et al., 2018]: center crop to 160×160 and then resize to 64×64. We normalize the pixel values to $[0, 1]$ without adding noise to pixels (i.e., no de-quantization).

Settings for training GLF

For all datasets (except CelebA-HQ), we use batch size 256 and Adam [Kingma and Ba, 2014] optimizer with an initial learning rate of 10^{-3} for the parameters of both the AE and the flow. We add a weight decay 2×10^{-5} to the optimizer for the flow. For MNIST, we train our model for 100 epochs, with the learning rate decaying by a factor of 2 after 50 epochs. For CIFAR-10, we train our model for 200 epochs, with the learning rate decaying by a factor of 2 every 50 epochs. For CelebA, we train our model for 40 epochs with no learning rate decay.

For GLF with perceptual loss, we compute the perceptual loss as suggested in [Hoshen and Wolf, 2018]. See `https://github.com/facebookresearch/NAM/blob/master/code/`

`perceptual_loss.py` for their implementation. Other settings are the same.

For the CelebA-HQ dataset, we adopt our AE network structure based on DCGAN [Radford et al., 2015]. Note that this is a relatively simple network for high-resolution images. We use batch size 64, with an initial learning rate of 10^{-3} for both the AE and the flow. We train our model for 60 epochs, with the learning rate decaying by a factor of 2 after 40 epochs.

Settings for training VAEs and VAE variants

We adopt standard settings for our reported results of VAE, VAE+flow prior, and VAE+flow posterior. We use $\beta = 1$ for all three VAE variants. We still use batch size 256 and Adam optimizer with an initial learning rate of 10^{-3} for both the VAE and the flow, if applicable. We find VAEs need a longer time to converge, so we double the training epochs. We train MNIST for 200 epochs, with learning rate decaying by a factor of 2 after 100 epochs. We train CIFAR-10 for 400 epochs, with the learning rate decaying by a factor of 2 every 100 epochs. We train CelebA for 80 epochs with learning rate decaying by a factor of 2 after 40 epochs.

Settings for training WAE-GAN

We follow the settings introduced in the original WAE paper [Tolstikhin et al., 2017]. The adversary architecture in WAE-GAN has the following architecture:

$$\begin{aligned} z \in \mathcal{R}^d &\to \mathrm{FC}_{512} \to \mathrm{ReLU} \\ &\to \mathrm{FC}_{512} \to \mathrm{ReLU} \\ &\to \mathrm{FC}_{512} \to \mathrm{ReLU} \\ &\to \mathrm{FC}_{512} \to \mathrm{ReLU} \to \mathrm{FC}_1 \end{aligned}$$

where d is the dimension of the latent variables.

WAE has two major hyper-parameters: λ that controls the weight coefficient of the adversarial regularizer, and σ^2, which is the variance of the prior. The batch size is 100 for all datasets. For MNIST, $\lambda = 10$ and $\sigma^2 = 1$, and the model is trained for 100 epochs. The initial learning rate is 10^{-3} for the AE and 5×10^{-4} for the adversary. After 30 epochs, both learning rates decreased by 2, and after the first 50 epochs, further by 5. For CIFAR, $\lambda = 10$ and $\sigma^2 = 1$ and the model is trained for 200 epochs. The initial learning rates are the same as training MNIST, and the learning rate decays by a factor of 2 after the first 60 epochs and further by a factor of 5 after 120 epochs. For CelebA, $\lambda = 1$ and $\sigma^2 = 2$. The model is trained for 55 epochs. The initial learning rate is 3×10^{-4} for the AE and 10^{-3} for the adversary. Both learning rates decay by 2 after 30 epochs and 5 after 50 first epochs.

Settings for training Two stage VAE

We adopt the settings in the original paper [Dai and Wipf, 2019]. For all datasets, the batch size is set to be 64, and the initial learning rate for both the first and the second is 10^{-4}. For MNIST, the first VAE is trained for 400 epochs, with a learning rate halved every 150 epochs; the second VAE is trained for 800 epochs with a learning rate halved every 300 epochs. For CIFAR-10, 1000 and 2000 epochs are trained for the two VAEs respectively, and the learning rates are halved every 300 and 600 epochs for the two stages. For CelebA, 120 and 300 epochs are trained for the two VAEs, respectively, and the learning rates are halved every 48 and 120 epochs for the two stages.

Explaining the discrepancy between our reported results and the results in the original paper: The original Two-stage VAE paper adopts similar settings to our experiments, but we observe large discrepancies in the results of CIFAR-10 and CelebA. After carefully reviewing their published codes, we find an issue in their FID score computation, particularly for the CIFAR-10 dataset. Specifically, the true images used for computing the

FID on CIFAR-10 are obtained from saving the original data file in .jpg format and reading them back, and the saving will cause some errors in pixel values. After fixing this issue, we re-ran their published codes and obtained similar results as we reported. We also run through their original FID computation protocol using samples from our models, and we obtain scores around 65. For CelebA, one particular detail worth noting is that, [Dai and Wipf, 2019] applies 128×128 center-crop before re-sizing on CelebA, while 160×160 center-crop is used in our evaluations. With smaller center crops, the human faces occupy a larger portion of the image with less background, making the generative modeling easier.

Settings for training RAE+GMM

The batch size, learning rate schedule, and the number of epochs for training RAE are the same as GLF. The objective of the RAE is reconstruction loss plus a penalty on the norm of the latent variable. Since the author does not report their choices for the penalty coefficient γ, we search over $\gamma \in 0.1, 0.5, 1, 2$, and we find that $\beta = 0.5$ leads to the best overall performances, and therefore we let $\gamma = 0.5$. After training the RAE, we fit a 10-component Gaussian mixture distribution on the latent variables.

3.3 EBM as Booster

3.3.1 Introduction

This section aims to improve the sample quality of an existing generative model through a better sampling procedure. Note that in deep generative models, samples are usually obtained by sending latent variables through a deterministic transformation, where the latent variables are sampled from a pre-defined prior distribution. Controlling the temperature of sampling from the prior may produce better samples, but this comes at the cost of less diversity [Kingma and Dhariwal, 2018]. We notice that there is a line of research that improves the sample quality of pre-trained GANs [Tanaka, 2019, Che et al., 2020]. They utilize the information contained in the discriminator of GANs to obtain better samples of the latent variable. In particular, [Che et al., 2020] samples latent variables from an energy-based model (EBM) defined jointly by the generator and discriminator.

The extension of these ideas to likelihood-based models is nontrivial because the discriminator of GANs is critical to guiding the movement of the latent variables. We extend these methods to generative models without adversarial training by constructing a latent variable EBM that consists of the pre-trained generative model and an energy function. The EBM can be trained efficiently by maximizing the data likelihood, and we observe that training the EBM only adds a slight computational overhead, as the convergence is very fast in the low dimensional space of the latent variables. After convergence, new samples are produced by latent variables sampled from the EBM. We show that our method can effectively improve the sample quality of various pre-trained generative models, including normalizing flows, VAEs, and a combination of the two.

3.3.2 Exponential Tilting of Generative Models

Suppose we have a pre-trained probabilistic generative model $p_{\phi^*}(\mathbf{x})$ over data space $\mathcal{X}$, we can define a new model by "exponential tilting" with energy function $E_\theta(\mathbf{x})$:

$$p_{\phi^*,\theta}(\mathbf{x}) = \frac{p_{\phi^*}(\mathbf{x}) \exp\left(-E_\theta(\mathbf{x})\right)}{Z_{\phi^*,\theta}}, \tag{3.4}$$

where $Z_{\phi^*,\theta} = \int p_{\phi^*}(\mathbf{x}) \exp\left(-E_\theta(\mathbf{x})\right) d\mathbf{x}$ is the corresponding normalizing constant. Since the plain EBM (2.12) is a special case with $p_{\phi^*} = \text{const}$, we can apply the maximum likelihood training strategies of EBM in section 2.4.2 to $p_{\phi^*,\theta}(\mathbf{x})$. The objective $L(\theta) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[-\log p_{\phi^*,\theta}(\mathbf{x})\right]$ has gradient

$$\frac{\partial L(\theta)}{\partial \theta} = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\frac{\partial E_\theta\left(\mathbf{x}\right)}{\partial \theta}\right] - \mathbb{E}_{p_{\phi^*,\theta}(\mathbf{x}')}\left[\frac{\partial E_\theta\left(\mathbf{x}'\right)}{\partial \theta}\right]. \tag{3.5}$$

In particular, the Langevin dynamics to generate samples from $p_{\phi^*,\theta}(\mathbf{x})$ is

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \frac{\epsilon}{2}\nabla_{\mathbf{x}}\left(E_\theta(\mathbf{x}) - \log p_{\phi^*}(\mathbf{x})\right) + \sqrt{\epsilon}\omega.$$

Note that the derivative of the log-likelihood of the original generative model $p_{\phi^*}(\mathbf{x})$ appears in the update, driving the Langevin dynamics to samples $\mathbf{x}$ with a large likelihood and low energy simultaneously. However, this only works for generative models with tractable likelihood. More importantly, operating in the pixel space may be inefficient as it may require moving between different modes with barriers that the Langevin dynamics cannot easily overcome.

3.3.3 EBM in Latent Space

Many types of probabilistic generative models, including normalizing flows and VAEs, adopt a decoder structure in their generation process. Namely, there is a pre-defined prior distri-

bution $p(\mathbf{z})$, and samples are generated by

$$\mathbf{z} \sim p(\mathbf{z}), \quad \mathbf{x} = G_{\phi^*}(\mathbf{z}).$$

We can therefore re-parametrize the EBM $p_{\phi^*,\theta}(\mathbf{x})$ in (3.4) with the latent variable $\mathbf{z}$ and obtain the corresponding density:

$$p_{\phi^*,\theta}(\mathbf{z}) = \frac{p(\mathbf{z}) \exp\left(-E_\theta(G_{\phi^*}(\mathbf{z}))\right)}{Z_{\phi^*,\theta}}. \tag{3.6}$$

When $p_{\phi^*,\theta}(\mathbf{z})$ is trained by maximizing the likelihood, the second term of the gradient (3.5) can also be re-parametrized to z space:

$$\mathbb{E}_{p_{\phi^*,\theta}(\mathbf{x})}\left[\frac{\partial E_\theta(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{p_{\phi^*,\theta}(z)}\left[\frac{\partial E_\theta(G_{\phi^*}(z))}{\partial \theta}\right] = -\frac{\partial \log Z_{\phi^*,\theta}}{\partial \theta} \tag{3.7}$$

Here is a simple derivation. Since p_{ϕ^*} is generated through a deterministic mapping G_{ϕ^*} from the latent space $\mathcal{Z}$ to the observation space $\mathcal{X}$, for any function f on $\mathcal{X}$ we have:

$$\begin{aligned}\mathbb{E}_{p_{\phi^*}}[f(\mathbf{x})] &= \int_{\mathcal{X}} f(\mathbf{x}) p_{\phi^*}(\mathbf{x}) d\mathbf{x} \\ &= \int_{\mathcal{Z}} f(G_{\phi^*}(\mathbf{z})) p(\mathbf{z}) d\mathbf{z} = \mathbb{E}_{p(\mathbf{z})}[f(G_{\phi^*}(\mathbf{z})].\end{aligned}$$

Therefore

$$Z_{\phi^*,\theta} = \mathbb{E}_{p_{\phi^*}(\mathbf{x})}\left[e^{-E_\theta(\mathbf{x})}\right] = \mathbb{E}_{p(\mathbf{z})}\left[e^{-E_\theta(G_{\phi^*}(\mathbf{z}))}\right]$$

Taking the derivative w.r.t θ we can get

$$\begin{aligned}\frac{\partial \log Z_{\phi^*,\theta}}{\partial \theta} &= -\mathbb{E}_{p_{\phi^*}(\mathbf{x})}\left[\frac{e^{-E_\theta(\mathbf{x})}}{Z_{\phi^*,\theta}}\frac{\partial E_\theta(\mathbf{x})}{\partial \theta}\right] \\ &= -\mathbb{E}_{p(\mathbf{z})}\left[\frac{e^{-E_\theta(G_{\phi^*}(\mathbf{z}))}}{Z_{\phi^*,\theta}}\frac{\partial E_\theta(G_{\phi^*}(\mathbf{z}))}{\partial \theta}\right] \\ &= -\mathbb{E}_{p_{\phi^*,\theta}(\mathbf{z})}\left[\frac{\partial E_\theta\left(G_{\phi^*}(\mathbf{z})\right)}{\partial \theta}\right],\end{aligned}$$

which is exactly (3.7).

Similarly, samples from $p_{\phi^*,\theta}(\mathbf{z})$ can be obtained through running the Langevin dynamics

$$\mathbf{z}_{i+1} = \mathbf{z}_i - \frac{\epsilon}{2}\nabla_{\mathbf{z}}\left(E_\theta(G_{\phi^*}(\mathbf{z})) - \log p(\mathbf{z})\right) + \sqrt{\epsilon}\omega. \tag{3.8}$$

Sometimes $\mathbf{z}$ has lower dimensionality than $\mathbf{x}$ and $p(\mathbf{z})$ can often be computed easily, therefore training and sampling from $p_{\phi^*,\theta}(\mathbf{z})$ is more efficient.

Instead of applying short-run LD with a tiny noise level or without noise to generate new samples from prior initialization as in section 2.4.3 and section 2.4.4, here we use LD with balanced noise to train and test the exponential tilting EBM in latent space. There are several reasons for that choice: 1. Since we use the log-likelihood $p(\mathbf{z})$ of a pre-trained generative model, the EBM here is better motivated in terms of the probability model rather than the gradient flow model; 2. Here, the EBM is helping to sample in the latent space, which is often a lower-dimensional Gaussian noise space. Therefore it is easier for the original LD to produce the correct samples in the latent space than in the high-dimensional data space; 3. Empirically we find that it generates better samples when we run LD with balanced noise in the latent space. Furthermore, running a longer LD than during training leads to better performance during testing.

3.3.4 Experiments

Toy Dataset

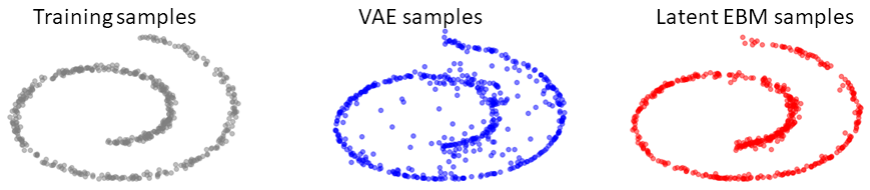
To give a quick proof-of-concept, we apply our method on toy datasets (25-Gaussians and Swiss Roll) following the setting of [Tanaka, 2019]. We first train a VAE on the training data, and then we fix the VAE and train a latent EBM as described in Section 3.3.3. The decoder and the energy function (which corresponds to the discriminator in GANs) have a simple fully connected structure:

1. the decoder maps a 2d input to a 2d output and has 4 fully connected layers with 256 hidden dimensions and leaky ReLU activation function;

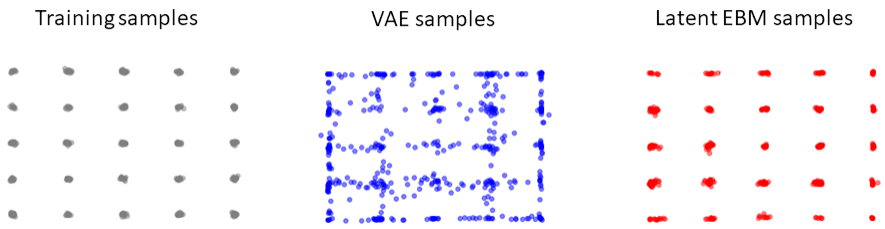
2. the energy function maps a 2d input to a scalar output and has 4 fully connected layers with 512 hidden dimensions and leaky ReLU activation function.

Note that we do not use normalizing flows on toy datasets because the vanilla flow is heavily constrained by the manifold structure of the prior distribution, making it very hard to model distributions like the 25-Gaussians.

We show qualitative results in Figure 3.9. Although samples from VAEs can basically cover the shape of the true distribution, many samples still appear in low-density regions. In contrast, by sampling and decoding latent variables obtained from the post-trained latent EBM, we can accurately preserve all modes in the target distribution while eliminating spurious modes in the 25-Gaussians case. In the Swiss Roll case, it is also clear that the EBM better captures the underlying data distribution.



((a)) Swiss Roll



((b)) 25-Gaussians

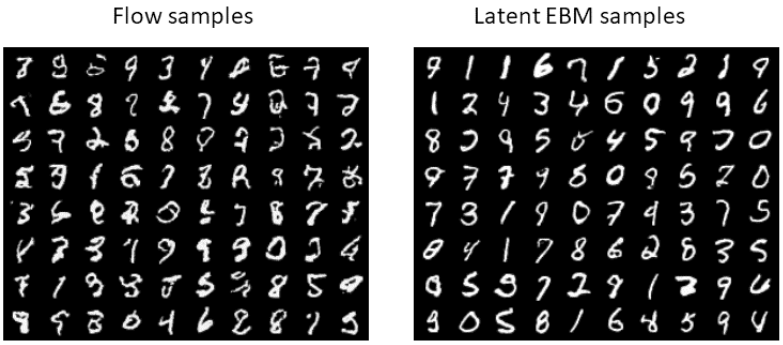
Figure 3.9: Applying latent EBM to VAEs trained on Swiss Roll and 25-Gaussians datset.

Image dataset

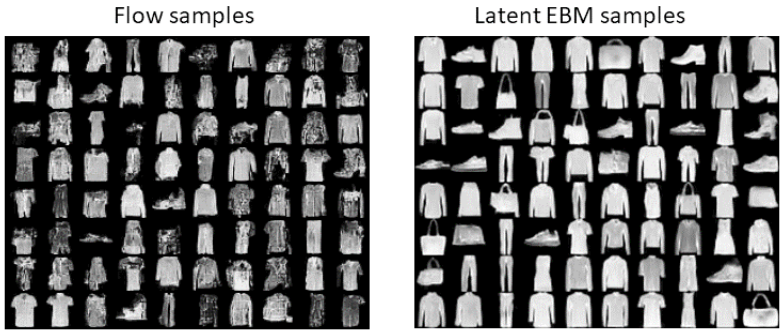
In this section, we evaluate the performance of the proposed latent EBM on MNIST, Fashion MNIST, and CIFAR-10 datasets. We use different decoder based generative models $p_{\phi^*}(\mathbf{x})$, including normalizing flow, VAE and GLF (section 3.2). Note that our main focus is on the relative improvements of samples from the EBMs over samples from base generative models, and therefore the performances of the base generative models may not be state-of-the-art. We adopt relatively simple network structures for convenience.

As in [Du and Mordatch, 2019], we use a convolutional network with scalar outputs as E_θ. We adopt short-run non-persistent MCMC, so the Langevin dynamics (3.8) is run with $\mathbf{z}_0$ initialized from $p(\mathbf{z})$ for a small number of steps. We fix $G_{\phi^*}(\mathbf{z})$ and train E_θ through maximum likelihood principle. For details on the settings of our experiments, see Appendix 3.3.6. It should be noted that the energy function only introduces a small parameter overhead. For example, its number of parameters is less than 5% of GLOW.

We show some qualitative results of training our proposed latent EBMs on top of a



((a)) MNIST



((b)) Fashion MNIST

((c)) CIFAR-10

Figure 3.10: Applying latent EBM to GLOW trained on MNIST, Fashion and CIFAR-10. **Left**: samples generated by $\mathbf{z}$'s from the prior. **Right**: samples generated by $\mathbf{z}$'s from $p_{\phi^*,\theta}(\mathbf{z})$.

GLOW [Kingma and Dhariwal, 2018] model in Figure 3.10. From Figure 3.10, we clearly observe that samples generated by latent variables obtained from the latent EBMs have higher quality than samples from the base generative model (i.e., decoding latent variables from prior distribution). On MNIST and Fashion MNIST, samples obtained through the latent EBM have smoother shapes than samples from the GLOW. On CIFAR-10, the latent EBM effectively corrects the noisy backgrounds of the samples generated by the GLOW. We illustrate the process of Langevin dynamics sampling from the latent EBM in Figure 3.11, where we generate samples for every 10 iterations. The Langevin dynamics move towards latent variables that produce more semantically meaningful and sharp samples.

Figure 3.11: MNIST Langevin dynamics visualization, initialized at samples from prior (the leftmost column).

More qualitative examples are shown below. In Figure 3.12, we show samples from VAE and VAE + latent EBM, from which we observe that the VAE+latent EBM generates sharper samples than the VAE alone. However, it should be noted that, since our EBM operates on the latent space, the overall sample quality is constrained by the capacity of the base generative models. In Figure 3.13, we present more examples of samples from GLOW and GLOW + latent EBM. In Figure 3.14, we show samples from GLF and GLF + latent EBM.

We observe that latent EBMs improve the sample quality of base generative models in all of these experiments. Finally, in Figure 3.15, we show samples from EBMs trained on pixel space.

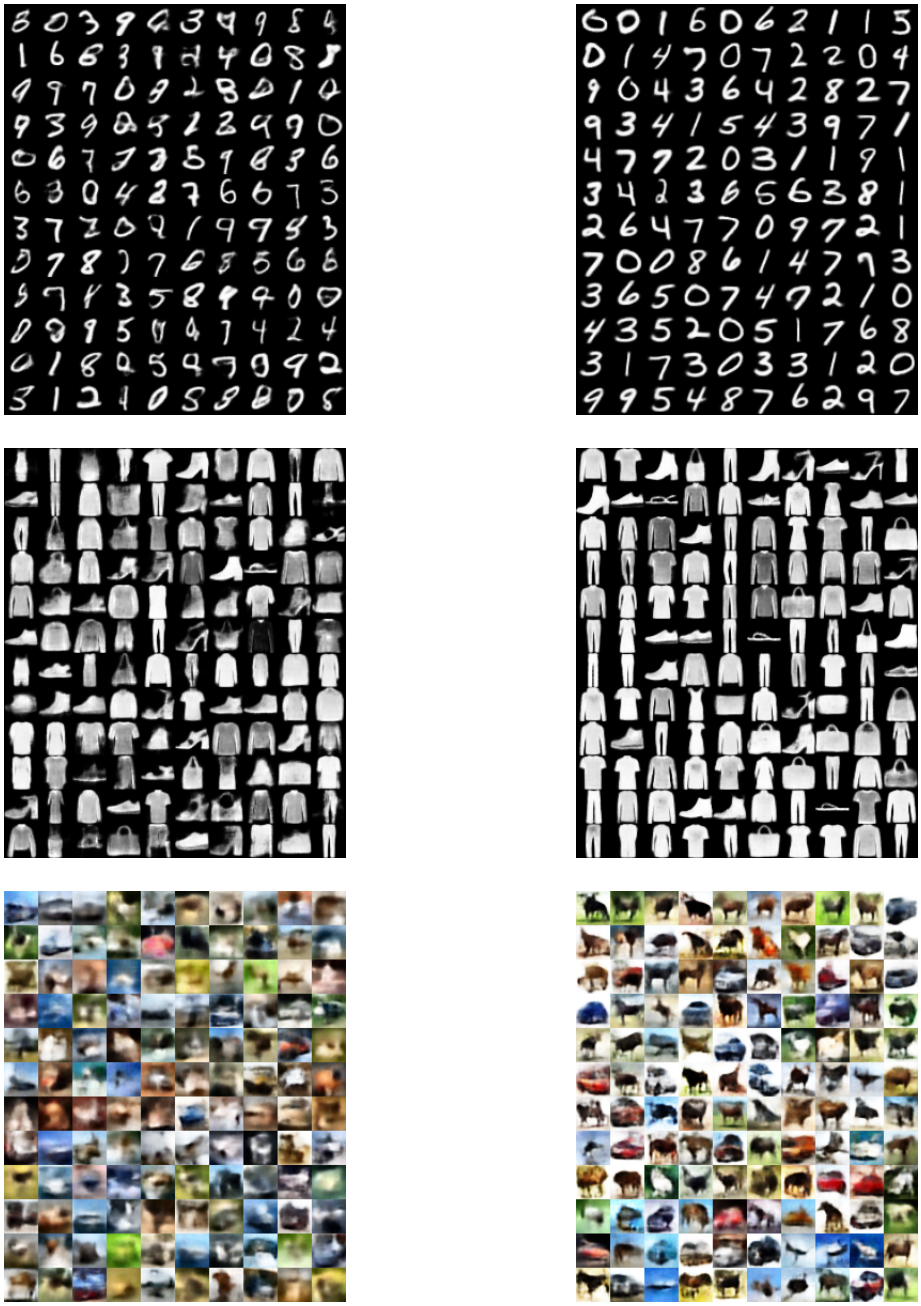


Figure 3.12: Qualitative results of VAE + latent EBM on MNIST, Fashion and CIFAR-10. **Left**: samples generated by $\mathbf{z}$'s from the prior. **Right**: samples generated by $\mathbf{z}$'s from $p_{\phi^*,\theta}(\mathbf{z})$.

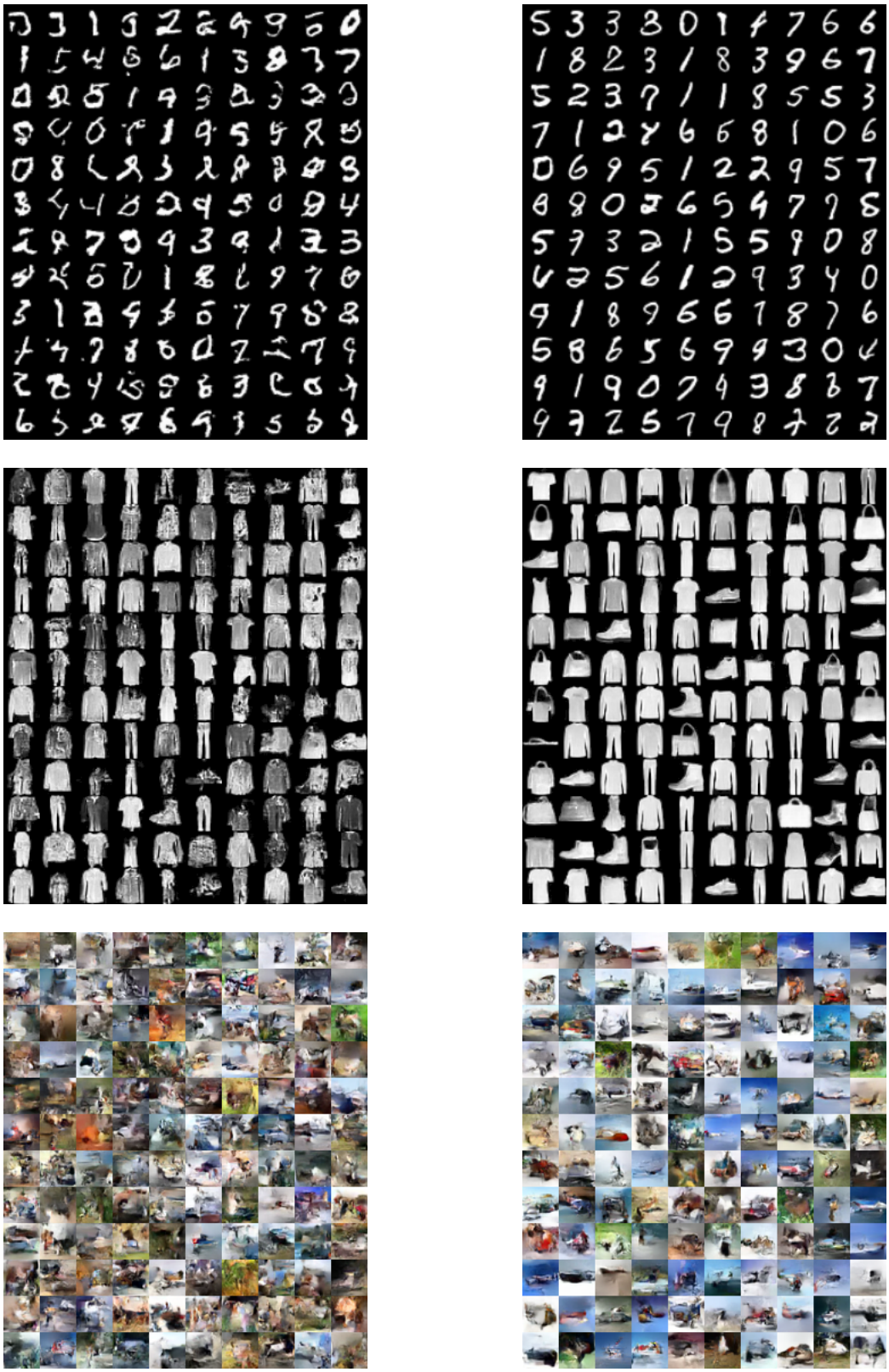


Figure 3.13: Additional qualitative results of GLOW + atent EBM on MNIST, Fashion and CIFAR-10. **Left**: samples generated by $\mathbf{z}$'s from the prior. **Right**: samples generated by $\mathbf{z}$'s from $p_{\phi^*,\theta}(\mathbf{z})$.

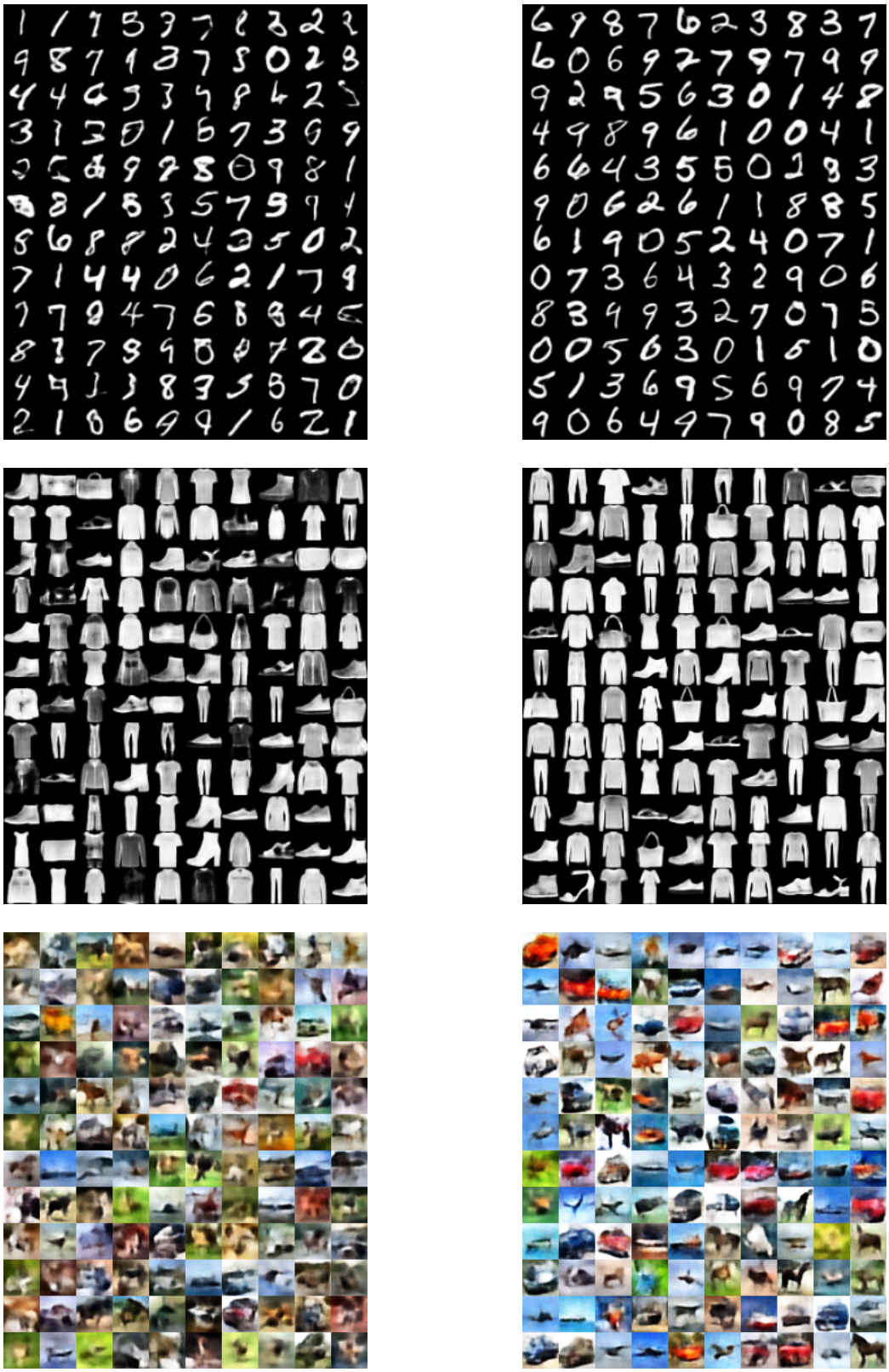
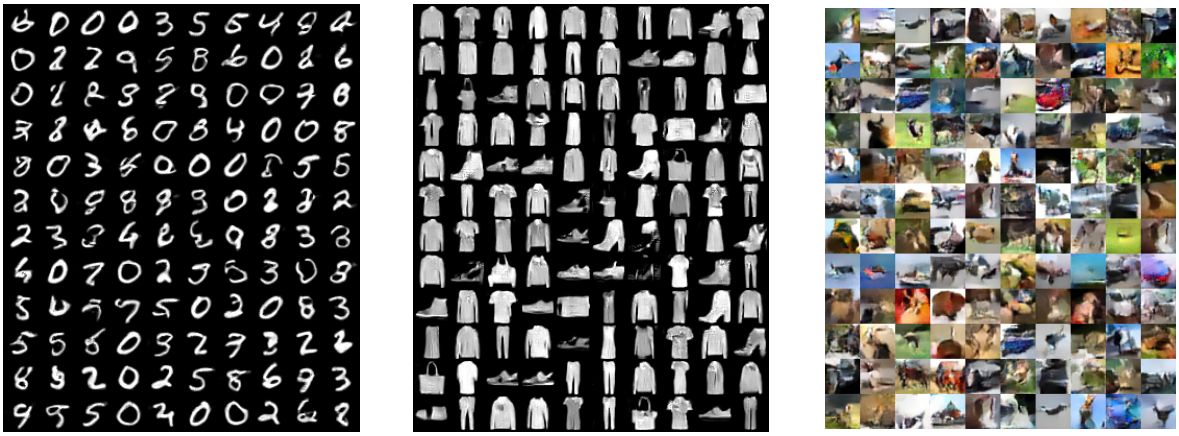


Figure 3.14: Qualitative results of GLF + latent EBM on MNIST, Fashion and CIFAR-10. **Left**: samples generated by $\mathbf{z}$'s from the prior. **Right**: samples generated by $\mathbf{z}$'s from $p_{\phi^*,\theta}(\mathbf{z})$.



((a)) MNIST ((b)) Fashion MNIST ((c)) CIFAR-10

Figure 3.15: Qualitative samples from EBMs trained on pixel space.

Our observation of the improvements in sample quality can be confirmed by quantitative results in Table 3.7, where we compare the FID scores [Heusel et al., 2017] of different models. We see that sampling latent variables from the latent EBM significantly improves the quality of generated samples over directly sampling from $p_\theta(\mathbf{x})$. In addition, we see that other methods, for example, Two-stage VAE[Dai and Wipf, 2019], do not improve the sample quality of VAEs *without posing a large weight on reconstruction term*. In contrast, our method can generate better samples without changing the VAE's objective, leading to good sample quality *and* structured latent representation. For completeness, we also present results of training EBMs directly on pixel space using the same model structures. The results are in the same range as the latent EBM models, but we observe that training EBMs on data is more sensitive to hyper-parameter settings and more computationally expensive.

Overfitting issue of training latent EBMs

As pointed out in [Grathwohl et al., 2019, Nijkamp et al., 2020b], instability and overfitting are frequently observed when training Energy-based Models. Overfitting happens when the energy of training samples is much lower than samples drawn from the EBM, which causes the model to produce worse samples. We find heuristic approaches such as energy

	MNIST	Fashion	CIFAR-10
GLOW	29.4	58.7	76.2
GLOW + EBM	12.3	41.6	67.8
VAE	18.9	57.1	139.6
Two-stage VAE	19.3	55.7	134.6
VAE + flow prior	18.6	52.3	128.2
VAE + EBM	16.0	38.1	108.4
GLF	14.2	32.5	96.6
GLF + EBM	12.1	25.3	85.1
EBM on data	25.5	39.3	80.6

Table 3.7: Comparing the FID scores of base generative models and generative models + exponential tilting with latent EBMs. Scores are computed using 10000 generated samples and real samples from the test set.

regularization and gradient clipping not helpful in preventing overfitting, so we simply stop the training of the latent EBM when sample quality deteriorates. We believe that future studies in improving the stability of EBM training can further boost the performance of our method.

Training time

One training step of EBM requires obtaining a sample by running multiple steps of MCMC, and it is much slower than one training step of the base generative model. However, we find that very few training iterations are needed to achieve the results in Table 3.7. Specifically, we only train latent EBMs for 200 steps when $G_{\phi^*}(\mathbf{z})$ is a GLOW or GLF, and 1000 steps when $G_{\phi^*}(\mathbf{z})$ is a VAE. Here a step refers to **one** batch, not an entire epoch. These numbers are several orders of magnitude smaller than the training steps needed for the base generative models. Therefore, our method does not add much computational overhead. In comparison, training EBMs on pixel space typically requires more than 100k steps.

3.3.5 Conclusion

This sub-chapter proposes to train an Energy-based model on the latent space of pre-trained generative models. We show that with little computational overhead, we can improve the sample quality of a variety of generative models, including normalizing flow and VAE, by sampling latent variables from the EBM. Our method also provides a general framework that connects Energy-based models and other likelihood-based generative models. We believe this connection is an interesting direction for future research.

3.3.6 Appendix

Experimental Settings

Base generative models:

We first introduce training settings of the base generative models we used in our experiments. We train GLOW following the settings provided in [Nalisnick et al., 2018]. For MNIST and Fashion MNIST, we use a GLOW architecture of 2 blocks of 16 affine coupling layers, squeezing the spatial dimension between the two blocks. We use a 3-layer Highway network with 64 hidden channels for the coupling function. For CIFAR-10, we use 3 blocks of 32 affine coupling blocks, applying the multi-scale architecture between each block. The coupling function is a 3-layer Highway network with 256 hidden channels. We modify the model size to fit in a single GPU for training. For MNIST and Fashion MNIST, we train the GLOW for 128 epochs with batch size 64 and Adam optimizer with fixed learning rate 5×10^{-4}. For CIFAR-10, we train the GLOW for 256 epochs with batch size 64 and Adam optimizer with fixed learning rate 5×10^{-4}.

We use the DCGAN [Radford et al., 2015] structure on the decoders of our VAEs, and the encoders are designed to be symmetric to the decoder. We use latent dimension 100 for all experiments. We use binary cross-entropy for MNIST and Fashion datasets as reconstruction

loss, while for CIFAR-10, we use MSE loss. All VAEs are trained for 256 epochs with batch size 128 and Adam optimizer with fixed learning rate 1×10^{-3}.

The GLF adopts the same encoder-decoder structure as in our settings for training VAEs. We use latent dimension 64 for all experiments. The normalizing flow for matching the latent distribution is a simple GLOW network with 4 affine coupling layers, and each consists of one fully connected layer with 256 units. The AE and the flow are jointly trained for 256 epochs with batch size 128 and Adam optimizer with fixed learning rate 1×10^{-3}.

Energy based models:

We used a simplified version of the network structure described in [Du and Mordatch, 2019] to define our E_θ. In particular, our method consists of 3 resnet blocks with 64 hidden channels, and 3 resent blocks with 128 hidden channels, followed by Global Sum Pooling and a FC layer. We also find the network structure in [Nalisnick et al., 2018], which has much fewer parameters, leads to only slightly worse performances. Therefore, their energy function can be used for parameter efficiency.

Unlike [Du and Mordatch, 2019, Nijkamp et al., 2020b] where the Langevin dynamics is dominated by the gradient, we find our latent EBMs work well with balanced noise and gradient in (3.8). For Langevin dynamics, we use $\epsilon = 0.01$ and run the chain for 60 steps. We find that adding a small amount (with coefficient 0.1) of energy regularization helps avoid over-fitting early in training. After training, we find sampling latent variables with a longer chain leads to better performances. We generate samples from $p_{\phi^*,\theta}(\mathbf{z})$ by running the chain for 100 steps.

For EBMs on the pixel space, we find short-run non-persistent training as described in [Nijkamp et al., 2020b] hard to converge on MNIST and Fashion MNIST, so we follow the setting in [Du and Mordatch, 2019], where they use persistent initialization for the Langevin dynamics. They maintain a sample replay buffer during the training, and samples from the buffer are used to initialize the chain. We follow the hyper-parameter settings in [Du and

Mordatch, 2019], and we train EBMs on MNIST and Fashion MNIST for 20k steps and CIFAR-10 for 100k steps. Note that we train less number of steps on CIFAR-10 than the open-source implementation of [Du and Mordatch, 2019] because it takes a prohibitively long time due to the hardware constraint. We find our samples qualitatively comparable to those of [Du and Mordatch, 2019] (see Figure 3.15), but we are unable to match their reported FID scores on CIFAR-10, possibly due to not training long enough. After training, new samples are generated from chains initialized from the replay buffer.

CHAPTER 4

FURTHER APPLICATIONS OF DEEP GENERATIVE MODEL

4.1 Overview

It is well known that deep generative models can generate high-quality samples from the target domain if they possess the ideal structure, sufficient amount of training data, and an appropriate training schedule [Karras et al., 2021, Ho et al., 2020]. Nevertheless, beyond their success in different generation settings, the generative principle behind them implies a potential for other applications. For example, a generative model must capture specific properties of the target domain before it can produce samples, which means a generative model can be used to understand a dataset or extract valuable features for downstream tasks. To see that, most of the deep generative models learn the distribution of the target dataset and compute the likelihood (a.k.a probability density) for each data point, with which we can conduct statistical analysis over the original dataset or new incoming data [Li et al., 2018]. Besides, if a generative model has a latent space, it captures the fundamental properties of the target dataset with a low-dimensional representation. Thus, it is convenient and efficient to deploy it on downstream tasks [Ding et al., 2018]. However, deep generative models need to be applied in downstream tasks with great care, as many experiments have shown [Nalisnick et al., 2018, Choi et al., 2018].

Another important direction of generative models is controllable generation, generating accurately targeted samples. This plays an essential role in real-life applications and requires the generative models to capture the conditional distributions of sub-areas within the domain that are defined by specific conditions. Designing a controllable generative model in areas like human face generation [Zhou et al., 2021], human body generation[Alexanderson et al., 2020] or text generation[Keskar et al., 2019] has huge commercial value, but also exposes plenty of challenges.

This chapter will focus on various applications related to modern deep generative models, including using generative models as likelihood-based methods for out-of-distribution detection and designing controllable generative models over human faces.

4.2 Out-of-Distribution Detection

4.2.1 Motivation and Background

In order to make reliable and safe decisions, deep learning models that are deployed for real life applications need to be able to identify whether the input data is anomalous or significantly different from the training data. Such data are called out-of-distribution (OOD) data. However, it is known that neural network classifiers can over-confidently classify OOD data into one of the training categories [Nguyen et al., 2015]. This observation poses a great challenge to the reliability and safety of AI [Amodei et al., 2016], making OOD detection a problem of primary importance. Several approaches have been proposed to detect OOD data based on deep classifiers [Hendrycks and Gimpel, 2016, Lakshminarayanan et al., 2017, DeVries and Taylor, 2018, Hendrycks et al., 2018]. Unfortunately, these methods cannot be applied to OOD detection for models trained without supervision, such as many generative models. An appealing OOD detection approach that may work for probabilistic generative models is to use their likelihood estimates. Such models can evaluate or estimate the likelihood of input data, and if a generative model fits the training data distribution well enough, it should assign high likelihood to samples from the training distribution and low likelihood to OOD samples.

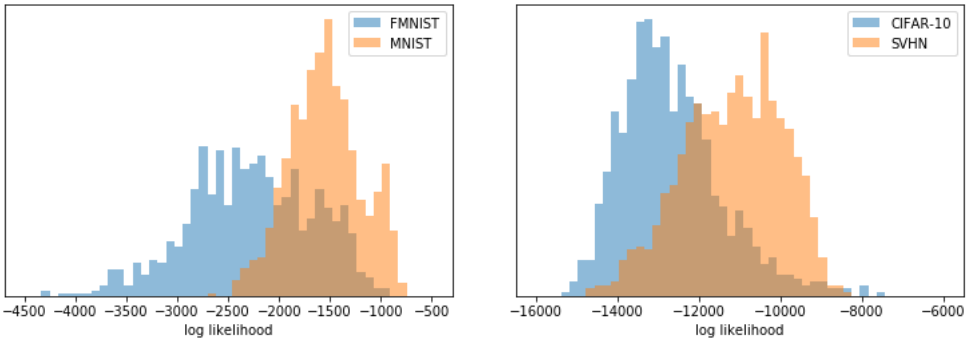
Recent advances in deep probabilistic generative models [Kingma and Welling, 2013, Van den Oord et al., 2016, Salimans et al., 2017, Kingma and Dhariwal, 2018] make possible generative modeling of very high dimensional and complicated data such as natural images, sequences [Oord et al., 2016] and graphs [Kipf and Welling, 2016]. These models can

evaluate the likelihood of input data easily and generate realistic samples, indicating that they succesfully approximate the distribution of training data. Therefore, it would appear promising to use deep generative models to detect OOD data [Li et al., 2018]. However, some recent studies [Nalisnick et al., 2018, Choi et al., 2018] reveal a counter intuitive phenomenon that challenges the validity of unsupervised OOD detection using generative models. They observe that likelihoods obtained from current state-of-the-art deep probabilistic generative models fail to distinguish between training data and some obvious OOD input types that are easily recognizable by humans. For example, [Nalisnick et al., 2018] shows that generative models trained on CIFAR-10 output higher likelihood on SVHN than on CIFAR-10 itself, despite the fact that images in CIFAR-10 (contains dogs, trucks, horses, etc.) and SVHN (containing license plates numbers) have very different semantic content.

At this point, no effective method has been discovered to ensure these generative models make the correct likelihood assignment on OOD data. Alternatively, some new scores based on likelihood are proposed to alleviate this issue [Ren et al., 2019, Nalisnick et al., 2019, Serrà et al., 2019]. The OOD detection is performed by setting thresholds on the new scores rather than on likelihood. Some of these methods obtain impressive OOD detection performance on invertible flow-based models [Kingma and Dhariwal, 2018] and auto-regressive models [Salimans et al., 2017]. Interestingly, we observe that these scores can be much less effective for Variational Auto-encoders (VAE), an important type of probabilistic generative models. The failure of current OOD scores on VAE suggests that a new score is necessary. Therefore, we propose a simple yet effective metric called Likelihood Regret (LR) to detect OOD samples with VAEs. The Likelihood Regret of a single input can be interpreted as the log ratio between its likelihood obtained by the posterior distribution optimized individually for that input and the likelihood approximated by the VAE. We conduct comprehensive experiments to evaluate our proposed score on a variety of image OOD detection tasks, and we show that it obtains the best overall performance.

4.2.2 Existing Problems of Current Generative Models

Suppose we have a set of N training samples $\{\mathbf{x}_i\}_{i=1}^N$ drawn from some underlying data distribution $\mathbf{x}_i \sim p(\mathbf{x})$. Our goal is to decide whether a test sample $\mathbf{x}$ is OOD, which, by definition in [Liang et al., 2018b], means that $\mathbf{x}$ has low density under $p(\mathbf{x})$. Probabilistic generative models $p_\theta(\mathbf{x})$ are trained on the set of training samples by maximizing the likelihood (or lower bound of likelihood). It is well known that maximizing the likelihood $p_\theta(\mathbf{x})$ is approximately equivalent to minimizing $D_{\mathrm{KL}}\left[p(\mathbf{x})\|p_\theta(\mathbf{x})\right]$, and thus a well trained generative model provides a good approximation to the true data distribution $p(\mathbf{x})$. Therefore, OOD data should have low likelihood under $p_\theta(\mathbf{x})$, since they stay in low probability regions of $p(\mathbf{x})$.



((a)) ((b))

Figure 4.1: Histogram that compares the log likelihood of test samples from **(a)**: Fashion MNIST and MNIST on a VAE trained on Fashion MNIST, and **(b)**: CIFAR-10 and SVHN on a VAE trained on CIFAR-10. Both experiments show that VAEs may assign high likelihoods to OOD samples.

However, the above argument fails in practice, as noted in [Nalisnick et al., 2018, Choi et al., 2018]. In particular, [Nalisnick et al., 2018] observe that almost all major types of probabilistic generative models, including VAE, flow-based model and auto-regressive model, can assign spuriously high likelihood to OOD samples. In Figure 4.1, we confirm that such a likelihood misalignment does exist on VAE, which is the model we focus on in this paper. We further observe that VAEs obtain surprisingly good reconstruction quality on OOD data

(Figure 4.2), indicating that they model OOD samples very well. These observations suggest that VAEs do not really regard some samples from completely different distributions as OOD, and it is extremely unreliable to use likelihood as an OOD detector.

((a)) ((b)) ((c)) ((d))

Figure 4.2: For each subfigure, the top row contains original images and the bottom row contains the reconstructed images. **(a), (b):** reconstruction of Fashion MNIST and MNIST images by a VAE trained on Fashion MNIST. **(c), (d):** reconstruction of CIFAR-10 and SVHN images by a VAE trained on CIFAR-10.

4.2.3 Re-use of the Optimization Trick

Our proposed OOD detection score, called Likelihood Regret (LR), measures the log likelihood improvement of the model configuration that maximizes the likelihood of an individual sample over the configuration that maximizes population likelihood. Intuitively, if a generative model is well trained on the training data distribution, given an in-distribution test sample, the improvement of likelihood by replacing the current model configuration with the optimal one for the single sample should be relatively small, hence resulting in low LR. In contrast, for an OOD test sample, since the model has not seen similar samples during training, the current model configuration is much less likely to be close to the optimal one, hence the LR could be large. Therefore, LR can serve as a good OOD detection score. However, in practice, a generative model can easily overfit when being optimized on a single sample and obtain very high likelihood, thus we have to seek some form of regularization on the model configuration that is being optimized. Luckily, the bottleneck structure of VAE provides a

natural regularization, by restricting the optimization to the parameters of the variational posterior distribution.

Formally, suppose we have a VAE with encoder E_ϕ and decoder D_θ. As commonly used in VAE [Kingma and Welling, 2013], $q_\phi(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\mu_\mathbf{x}, \sigma_\mathbf{x}^2\mathbf{I})$, so the encoder outputs the mean and variance: $E_\phi(\mathbf{x}) = (\mu_\mathbf{x}, \sigma_\mathbf{x})$. For clarity, we use $\tau(\mathbf{x}, \phi)$ to denote the sufficient statistics $(\mu_\mathbf{x}, \sigma_\mathbf{x})$ of $q_\phi(\mathbf{z}|\mathbf{x})$. Recall that the training loss of VAE (ELBO) has the form

$$\mathcal{L}(\mathbf{x}; \theta, \phi) \triangleq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}} \left[q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\right], \tag{4.1}$$

we express ELBO as $\mathcal{L}(\mathbf{x}; \theta, \tau(\mathbf{x}, \phi))$ to emphasize its direct dependency on $\tau(\mathbf{x}, \phi)$. During training, based on empirical risk minimization (ERM) criterion, our objective is to obtain network parameters $\Theta^* = (\phi^*, \theta^*)$ that maximize the population log likelihood $\log p_\theta(\mathbf{x})$. In other words, Θ^* are the parameters that achieve best *average* log likelihood over the finite training set. In practice, we train VAE by maximizing ELBO instead of log likelihood. Since ELBO is a lower bound of log likelihood, we can regard maximizing EBLO as a good surrogate for maximizing log likelihood, so

$$(\phi^*, \theta^*) \approx \arg\max_{(\phi,\theta)} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\mathbf{x}_i; \theta, \tau(\mathbf{x}_i, \phi)). \tag{4.2}$$

Since $q_\phi(\mathbf{z}|\mathbf{x})$ is fully determined by $\tau(\mathbf{x}, \phi)$, we also denote $\Theta = (\tau, \theta)$ or $\Theta = (\tau(\cdot, \phi), \theta)$ as an abuse of notation.

For a specific test input $\mathbf{x}$, we can fix the decoder parameters θ^*, and find the optimal configuration of the variational posterior distribution parameter $\hat{\tau}(\mathbf{x}) = (\hat{\mu}_\mathbf{x}, \hat{\sigma}_\mathbf{x})$ that maximizes its *individual* ELBO:

$$\hat{\tau}(\mathbf{x}) = \operatorname*{argmax}_{\tau} \mathcal{L}(\mathbf{x}; \theta^*, \tau). \tag{4.3}$$

In other words, $\hat{\tau}(\mathbf{x})$ is the optimal approximate posterior distribution of the latent variable $\mathbf{z}$ given the particular input $\mathbf{x}$ and the optimal decoder θ^* obtained from the training set. Now we define the Likelihood Regret (LR) of the input data $\mathbf{x}$ as

$$\text{LR}(\mathbf{x}) = \mathcal{L}_K(\mathbf{x}; \theta^*, \hat{\tau}(\mathbf{x})) - \mathcal{L}_K(\mathbf{x}; \theta^*, \phi^*), \tag{4.4}$$

where $\mathcal{L}_K$ is the estimated log-likelihood of the VAE model from (2.2).

LR can also be interpreted as the log ratio of the likelihood obtained from the generative model (VAE) with the optimal configuration of the variational posterior distribution for an individual input, to its likelihood obtained from the VAE trained on training set. This interpretation connects our method to [Ren et al., 2019, Serrà et al., 2019], which also have a likelihood ratio interpretation. The naming of our method is motivated by the concept of regret in online learning, which measures how 'sorry' the learner is, in retrospect, not to have followed the predictions of the optimal hypothesis [Shalev-Shwartz and Singer, 2007].

Implementation: The major component of evaluating LR is computing $\hat{\tau}(\mathbf{x})$ defined in (4.3), namely the configuration $\tau(\mathbf{x})$ that maximizes the likelihood (ELBO) for a single sample. To do this, we fix the decoder θ^*, and apply iterative optimization algorithms on τ with initialization $\tau^*(\mathbf{x}) = E_{\phi^*}(\mathbf{x})$ using $\mathcal{L}(\mathbf{x}; \theta^*, \tau)$ as the objective function until convergence. As an alternative approach, instead of optimizing τ directly, we can also optimize the parameters ϕ (initialized as ϕ^*) of the encoder given $\mathbf{x}$ and $\theta = \theta^*$. We summarize the computation of Likelihood Regret in Algorithm 2.

4.2.4 Experiments

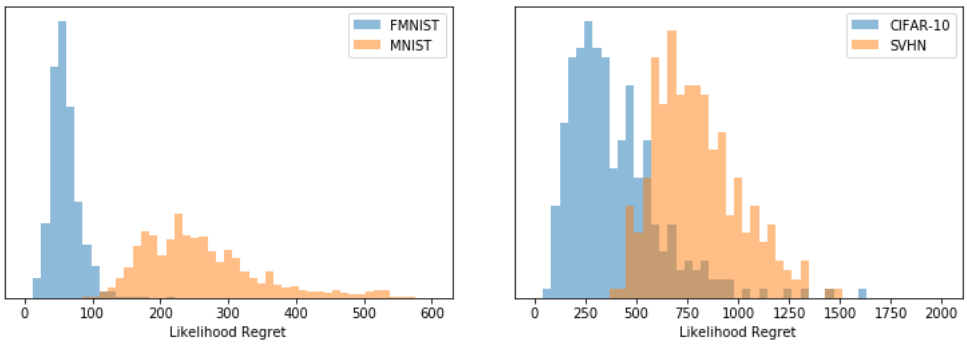
In this section, we conduct experiments to benchmark the performance of Likelihood Regret on OOD detection tasks on image datasets. For most experiments, we train VAEs with samples only from the training set of in-distribution data (Fashion-MNIST and CIFAR-10),

Algorithm 2 Computing Likelihood Regret (LR)

Input: Test sample $\mathbf{x}$, trained VAE $(E_{\phi^*}, D_{\theta^*})$, number of posterior samples for likelihood estimation K, number of optimization step S, learning rate γ.

1: $L_{\text{VAE}} = \mathcal{L}_K(\mathbf{x}; \theta^*, \phi^*)$ ▷ Estimate the log likelihood of $\mathbf{x}$ under the VAE model by (2.2)
2: Set $\phi = \phi^*$
3: **for** S iterations **do**
4: $\quad \phi \leftarrow \text{Adam}(\phi, \nabla_\phi(-\mathcal{L}(\mathbf{x}; \theta^*, \phi)), \gamma)$ ▷ Optimize ϕ by maximizing the ELBO objective
5: $L_{\text{OPT}} = \mathcal{L}_K(\mathbf{x}; \theta^*, \phi)$ ▷ Estimate the log likelihood of $\mathbf{x}$ with optimized encoder
6: $\text{LR} = L_{\text{OPT}} - L_{\text{VAE}}$

and use test samples from different datasets to measure the OOD performances. Details regarding datasets and experimental settings can be found in Appendix 4.2.6.



((a)) ((b))

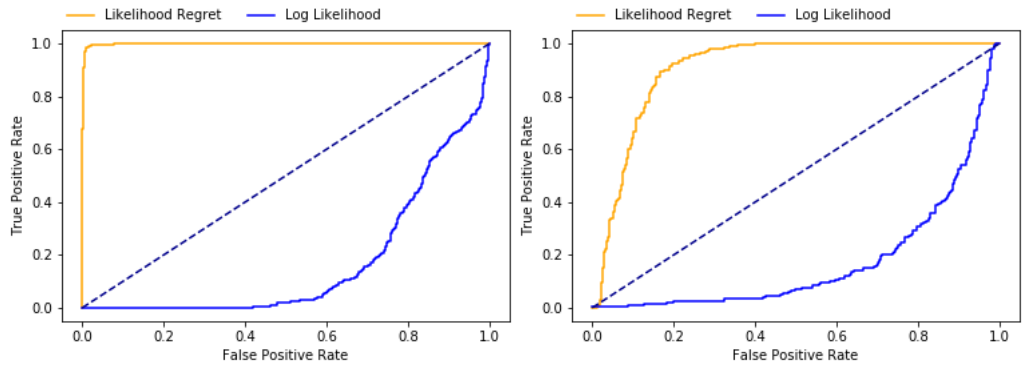
Figure 4.3: Histogram that compares the Likelihood Regret of test samples from **(a)**: Fsahion MNIST and MNIST on a VAE trained on Fashion MNIST, and **(b)**: CIFAR-10 and SVHN on a VAE trained on CIFAR-10. Both experiments show that OOD samples tend to have higher LR, as expected.

We first follow the setting of [Nalisnick et al., 2018] and conduct the following two experiments as a proof-of-concept: (a) Fashion-MNIST as in-distribution and MNIST as OOD, and (b) CIFAR-10 as in-distribution and SVHN as OOD. We train VAEs on the training set of Fashion MNIST and CIFAR-10, and compute the Likelihood Regret for 1000 random samples from the test set of in-distribution data and corresponding OOD data. We plot the histograms of LR in Figure 4.3. Comparing Figure 4.1 with Figure 4.3, we observe that the VAE will assign higher likelihoods to OOD samples, while Likelihood Regret can

largely correct such likelihood misalignment. OOD samples typically have larger LR than in-distribution samples, which confirms the effectiveness of our OOD score.

Metrics

Like other OOD scores, Likelihood Regret needs a threshold when applied to the OOD detection tasks. The choice of a threshold depends on the particular application. To quantitatively evaluate our method, we mainly use Area Under the Curve-Receiver Operating Characteristics (AUCROC↑) as a metric. AUCROC is commonly used in OOD detection tasks [Hendrycks and Gimpel, 2016], and it provides an effective performance summary across different score thresholds [Fawcett, 2006, Metz, 1978]. In Figure 4.4, we plot the ROC curves for the above two experiments, and results show that log-likelihood is an extremely bad OOD detector as it obtains AUC-ROC much lower than 0.5 (random guessing), while LR achieves good OOD performances. In addition to AUCROC, we also include Area Under the Curve-Precision Recall Curve (AUCPRC↑) and the False Positive Rate at 80% True Positive Rate (FPR80↓) as quantitative measurements.



((a)) Fashion MNIST v.s. MNIST ((b)) CIFAR-10 v.s. SVHN

Figure 4.4: Comparing the ROC curves of using Likelihood Regret and Log Likelihood for OOD detection. On Fashion MNIST v.s. MNIST experiment, Likelihood Regret improves the AUC-ROC of OOD detection from 0.165 to 0.999. On CIFAR-10 v.s. SVHN experiment, Likelihood Regret improves the AUC-ROC of OOD detection from 0.161 to 0.876.

	LR_E	LR_Z	Likelihood	IC (png)	IC (jp2)	Likelihood Ratio	LMD
MNIST	**0.988**	0.967	0.201	0.946	0.553	0.924	0.877
CIFAR-10	0.997	0.998	**1**	0.907	0.999	0.968	0.995
SVHN	**1**	**1**	0.999	0.992	**1**	0.785	0.995
KMNIST	**0.994**	0.983	0.731	0.708	0.599	0.983	0.955
NotMNIST	0.999	**1**	0.943	0.923	0.966	0.996	0.998
Noise	**1**	0.963	**1**	0.453	**1**	**1**	**1**
Constant	**1**	**1**	0.928	**1**	**1**	0.775	0.981

((a)) VAE trained on Fashion MNIST

	LR_E	LR_Z	Likelihood	IC (png)	IC (jp2)	Likelihood Ratio	LMD
MNIST	**0.998**	0.976	0.008	0.994	0.988	0.792	0.027
FMNIST	0.991	0.963	0.074	**0.992**	0.990	0.807	0.183
SVHN	0.875	0.843	0.193	**0.912**	0.908	0.265	0.279
LSUN	**0.691**	0.640	0.494	0.624	0.315	0.632	0.527
CelebA	**0.714**	0.690	0.465	0.641	0.564	0.447	0.576
Noise	0.994	0.922	**1**	0.032	0.054	**1**	0.983
Constant	0.974	0.924	0.258	**1**	**1**	0.470	0.431

((b)) VAE trained on CIFAR-10

Table 4.1: AUCROC of Likelihood Regret (LR) and other OOD detection scores on different datasets. Each row contains the results of an OOD dataset.

Quantitative Comparison with Other OOD Scores

After a simple verification of the effectiveness of our proposed OOD score, we carefully study its performance and compare it with competing methods. In our comparison we use likelihood as a simple baseline, and include OOD scores designed for other generative models, including two variants of input complexity adjusted score (**IC (png)** and **IC (jp2)**) [Serrà et al., 2019], likelihood ratio with background model (**Likelihood Ratio**) [Ren et al., 2019] as well as latent Mahalanobis distance (**LMD**) [Bulusu et al., 2020]. Details of these methods and their implementations can be found in Appendix 4.2.6. We present two implementations of Likelihood Regret: one optimizes the whole encoder ($\mathbf{LR_E}$), and the other only optimizes $(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}})$ ($\mathbf{LR_Z}$).

The main results of this work are presented in Table 4.1, which shows the AUCROC scores of different methods on different datasets. We also present results of AUPRC and FPR80 in Table 4.7 and Table 4.8 in Appendix 4.2.6. We see that OOD detection scores exhibit

largely consistent behavior across these metrics. We make several important observations from these results:

1. We confirm that likelihood is problematic in OOD detection. For example, VAE trained on CIFAR-10 not only assigns significantly higher test likelihood on SVHN, but also on MNIST, Fashion MNIST and random constant images. The severe likelihood misalignment is reflected in an AUCROC value close to 0.

2. Likelihood Regret successfully corrects the misalignment of likelihood and obtains good OOD detection performances across all tasks, as the AUCROC values are close to the optimal value 1 on all experiments except for CIFAR-10 v.s CelebA and CIFAR-10 v.s LSUN. Note that these are the hard cases as all methods do not perform well. Samples from these datasets have similar texture as samples from CIFAR-10, and therefore it is hard for generative models trained without class labels to distinguish between them. Further, we observe that optimizing the encoder leads to slightly better performance than optimizing $(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}})$ only. One possible explanation is that, for in distribution samples, optimizing the encoder is more constrained than directly optimizing z, which prevents the latent variables from moving too much.

3. While competing methods also obtain good OOD detection performance on some tasks, all of them exhibit severe issues on certain specific tasks. For example, likelihood ratio with background model fails on the classic task of detecting SVHN from in-distribution CIFAR-10 (AUCROC only 0.28); both variants of input complexity adjusted likelihood fail almost completely on distinguishing between random uniform noise and CIFAR-10 (AUCROC close to 0); latent Mahalanobis distance does not perform well on almost all OOD datasets when CIFAR-10 is the in-distribution dataset. These failure cases suggest that none of these OOD scores can be safely applied, at least for VAE models. In contrast, likelihood regret is shown to be effective on all the tasks.

	LR_{ED}	LR_D
MNIST	0.124	0.189
KMNIST	0.495	0.609
NotMNIST	0.969	0.947
Noise	0.001	0.004
Constant	0.989	1

((a)) VAE trained on Fashion MNIST

	LR_{ED}	LR_D
SVHN	0.195	0.212
LSUN	0.468	0.526
CelebA	0.488	0.655
Noise	0.002	0
Constant	0.322	0.435

((b)) VAE trained on CIFAR-10

Table 4.2: AUCROC of Likelihood Regret (LR) obtained by optimizing different components of the VAE. LR_{ED} corresponds to optimizing both the encoder and decoder, and LR_D corresponds to optimizing the decoder only.

Overall, based on results from Table 4.1, 4.7 and 4.8, we claim that LR achieves the best OOD detection performance. On those tasks where the performance of LR is not the best it is still very close to the best results. More importantly, it achieves good performance without any failure case, while all competing methods are shown to be ineffective on some experiments.

In summary, the experiments provide strong evidence that current state-of-the-art OOD scores for generative models may not be applicable to VAE, while our proposed score achieves good OOD detection results on a variety of tasks. To the best of our knowledge, Likelihood Regret is the only effective OOD score that can correct the likelihood misalignment of VAE.

Optimizing other components of the VAE: One key reason for the effectiveness of LR on VAE is that we only optimize the configuration of latent variables for a single test input, which is done by optimizing $\mathbf{z}$ directly or optimizing the encoder parameters. The bottleneck structure at the latent variable provides natural regularization that avoids overfitting on the single example. We argue that optimizing other components of the VAE will be less effective, as the optimization can easily overfit any test sample regardless if it is OOD or not. To show this, we perform ablation studies that optimize either the decoder only or the whole VAE (both encoder and decoder), and results are shown in Table 4.2. We observe that these alternative approaches are significantly worse than optimizing the latent variables or encoder parameters.

	$\beta = 0.1$		$\beta = 0.5$		$\beta = 5$		$\beta = 10$	
	Likelihood	LR_E	Likelihood	LR_E	Likelihood	LR_E	Likelihood	LR_E
MNIST	0.191	0.988	0.154	0.948	0.231	0.997	0.225	0.966
KMNIST	0.704	0.997	0.679	0.985	0.696	0.996	0.698	0.952
NotMNIST	0.986	1	0.983	1	0.985	1	0.99	0.999
Noise	1	0.989	1	0.992	1	0.996	1	0.991
Constant	0.982	0.995	0.964	0.998	0.928	0.998	0.968	0.986

((a)) $\beta-$VAEs trained on Fashion MNIST

	$\beta = 0.1$		$\beta = 0.5$		$\beta = 5$		$\beta = 10$	
	Likelihood	LR_E	Likelihood	LR_E	Likelihood	LR_E	Likelihood	LR_E
SVHN	0.189	0.847	0.171	0.835	0.194	0.866	0.153	0.873
LSUN	0.474	0.645	0.465	0.655	0.469	0.654	0.442	0.657
CelebA	0.499	0.709	0.477	0.714	0.528	0.718	0.462	0.706
Noise	1	1	1	0.999	1	0.994	1	0.9735
Constant	0.319	0.964	0.279	0.947	0.268	0.962	0.284	0.954

((b)) β-VAEs trained on CIFAR-10

Table 4.3: AUCROC of Likelihood Regret (LR) and Likelihoods for $\beta-$VAEs on different datasets.

Results on β-VAE: VAEs are typically trained with the ELBO objective in (2.1), however, in practice, sometimes we train β-VAEs, where there is a coefficient $\beta \neq 1$ on the KL divergence term. Usually we use $\beta < 1$ for better sample quality [Dai and Wipf, 2019, Xiao et al., 2019], and $\beta > 1$ for disentanglement in latent space [Higgins et al., 2016, Burgess et al., 1804]. We evaluate the OOD detection performances of LR on $\beta-$VAE variants in Table 4.3. From Table 4.3, we observe that LR behaves consistently across different values of β.

On the capacity of the VAEs: One concern regarding the effectiveness of Likelihood Regret is that it may depend on the capacity of VAEs. For example, a VAE with large capacity is easier to optimize for the encoder configuration on a single sample, leading to larger Likelihood Regret score. we evaluate LR on VAEs with different capacities, and results are presented in Table 4.4. We conclude that the performance of LR is robust to the capacity of VAEs. The AUCROC slightly drops for VAEs with large capacity trained on CIFAR-10. By inspection, we observe that large VAEs overfit the training data, as the test NLL on

	$C = \frac{1}{4}\times$		$C = \frac{1}{2}\times$		$C = 2\times$		$C = 4\times$	
	Likelihood	LR_E	Likelihood	LR_E	Likelihood	LR_E	Likelihood	LR_E
MNIST	0.110	0.966	0.148	0.986	0.125	0.975	0.237	0.984
KMNIST	0.608	0.991	0.65	0.996	0.711	0.992	0.812	0.994
NotMNIST	0.977	1	0.978	1	0.99	0.999	1	0.997
Noise	1	0.987	1	0.999	1	0.997	1	1
Constant	0.966	0.998	0.947	0.998	0.957	0.994	0.979	0.997

((a)) VAEs trained on Fashion MNIST

	$C = \frac{1}{4}\times$		$C = \frac{1}{2}\times$		$C = 2\times$		$C = 4\times$	
	Likelihood	LR_E	Likelihood	LR_E	Likelihood	LR_E	Likelihood	LR_E
SVHN	0.198	0.868	0.176	0.875	0.203	0.814	0.198	0.79
LSUN	0.462	0.641	0.453	0.652	0.445	0.635	0.467	0.611
CelebA	0.455	0.706	0.445	0.717	0.521	0.793	0.475	0.753
Noise	1	1	1	1	1	0.998	1	0.999
Constant	0.218	0.999	0.276	0.998	0.292	0.989	0.246	0.962

((b)) VAEs trained on CIFAR-10

Table 4.4: AUCROC of Likelihood Regret (LR) and Likelihoods for VAEs with different capacity, where we proportionally increase/decrease the number of channels of convolution layers. For example, $C = \frac{1}{4}\times$ means that the VAE has $\frac{1}{4}$ channels compared to the baseline VAE.

CIFAR-10 is even larger than that of the baseline VAE. In the case of overfitting, the LR for in-distribution test data will be larger.

Runtime: Our method requires 2 likelihood estimations and several optimization iterations for each testing image, which will make it slower than its competing methods. As a comparison, input complexity adjusted likelihood does not have computationally overhead, and likelihood ratio with background model needs to train 2 models and take 2 likelihood estimations. However, we observe that in our experiments, on average the computation of LR takes less than 0.3s, which is comparable to the IWAE likelihood estimation (also around 0.3s), so the computational overhead is acceptable.

Reverse Direction: In Table 4.5, we include results of a set of simple experiments for VAE trained on SVHN. Note that in this case, likelihood itself works well on most tasks. We do not show results of VAE trained on MNIST because every method works nearly perfectly. We observe that LR achieves good performances on all the tasks, while input

complexity still has trouble with distinguishing noise from in-distribution data. In addition, its performance on SVHN v.s. CIFAR-10 lies far behind LR. As in Table 4.1, likelihood ratio has trouble on CIFAR-10 v.s. Constant. These failures suggest that competing OOD scores have systematical issues on VAE.

	LR_E	LR_Z	Likelihood	IC (png)	IC (jp2)	Likelihood Ratio	LMD
MNIST	**1**	**1**	**1**	**1**	**1**	**1**	**1**
FMNIST	**1**	0.999	**1**	**1**	**1**	0.928	0.998
CIFAR-10	0.924	0.842	**0.982**	0.524	0.608	0.936	0.955
Noise	**1**	**1**	**1**	0.235	0.105	**1**	**1**
Constant	**1**	**1**	0.213	**1**	**1**	0.136	0.818

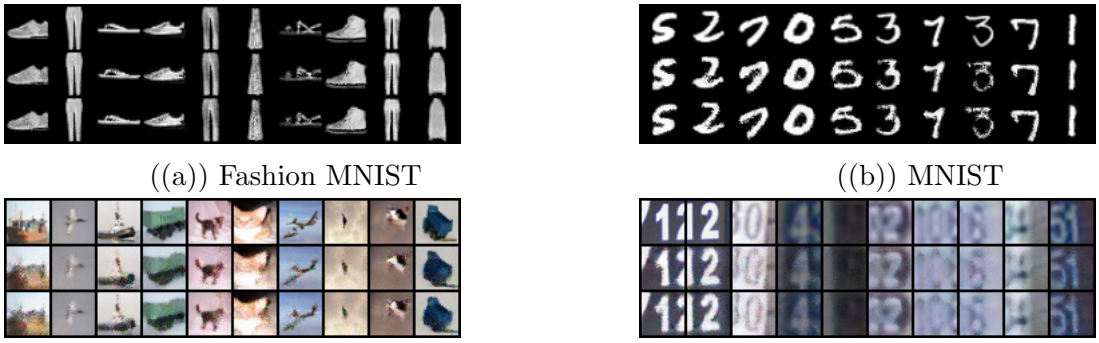
Table 4.5: AUCROC for model trained on SVHN

Illustration of Optimization: To better illustrate Likelihood Regret, we compare the reconstruction of some test samples with trained VAE and optimized posterior distribution in Figure 4.5. Since as we show in Figure 4.2, the VAE can reconstruct both the in-distribution and out of distribution data very well, visually we cannot see obvious differences between reconstructed images from VAE and from optimized $q_\phi(\mathbf{z}|\mathbf{x})$. However, we can still observe the improvements of reconstruction by the optimal $q_\phi(\mathbf{z}|\mathbf{x})$ in MNIST examples.

Additional Results: In Appendix 4.2.6, we show more qualitative examples of reconstruction on different test datasets. In Appendix 4.2.6, we also display some randomly generated samples from the VAEs.

4.2.5 Conclusion

In summary, we carefully study the task of unsupervised out-of-distribution detection for VAEs. We evaluate some current state-of-the-art OOD detection scores on a set of experiments, and we conclude that their success on OOD detection for other probabilistic models cannot be easily transferred to VAEs. To overcome the difficulty, we propose Likelihood



((a)) Fashion MNIST ((b)) MNIST

((c)) CIFAR-10 ((d)) SVHN

Figure 4.5: For each subfigure, the top row contains original images, the middle row contains the reconstruction from VAE, and the bottom row contains the reconstruction images with optimized encoder. **(a), (b)** are obtained from VAE trained on Fsahion MNIST. **(c), (d)** are obtained from VAE trained on CIFAR-10.

Regret, an OOD score for VAEs that is effective on all the tasks we evaluated. We believe that Likelihood Regret can be extended to other generative models if a good optimizable model configuration is defined. We hope this work can lead to further progress on OOD detection for probabilistic generative models.

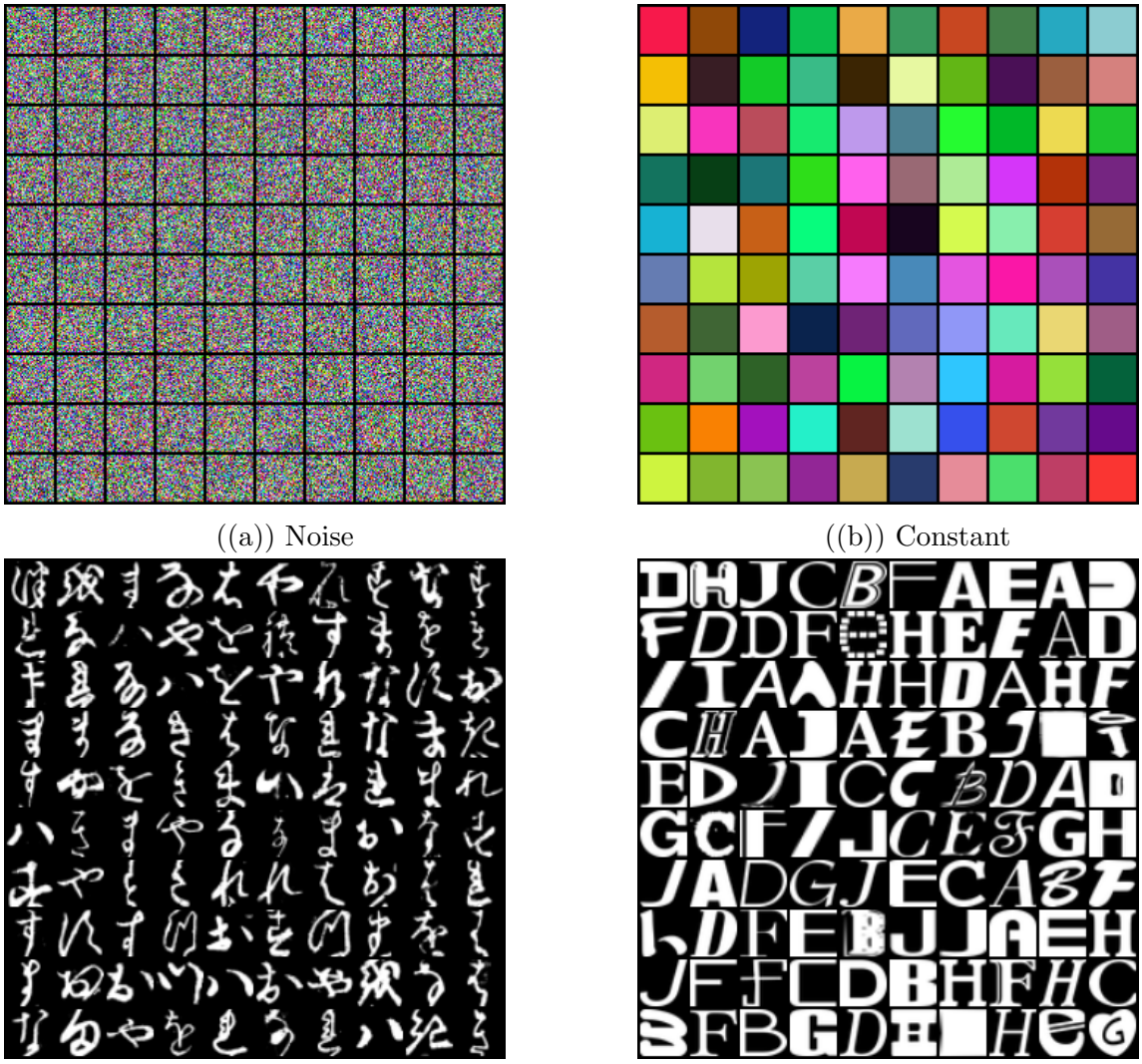
4.2.6 Appendix

In this section, we introduce detailed settings of our experiments.

Datasets

We use several publicly available datasets in our experiments. These datasets include MNIST [LeCun et al., 2010], Fashion-MNIST[Xiao et al., 2017], KMNIST[Clanuwat et al., 2018], notMNIST, CIFAR-10 and CIFAR-100[Krizhevsky et al., 2009], SVHN [Netzer et al., 2011], CelebA [Liu et al., 2015] and LSUN[Yu et al., 2015]. We also create two types of synthetic images: Noise and Constant. Noise images are random samples from Unif$\{0, ..., 255\}$ distribution for each pixel, and Constant images are images with constant value sampled from Unif$\{0, ..., 255\}$ for each channel. Some examples of our created images are shown in Figure

4.6. We also present some examples of KMNIST and NotMNIST, as they are less familiar to the public. They are used as OOD datasets for models trained on Fashion MNIST.



((a)) Noise ((b)) Constant

((c)) KMNIST ((d)) NotMNIST

Figure 4.6: Some examples of our created images used in experiments.

For most of our experiments, the VAE is trained on Fashion-MNIST and CIFAR-10, where we use the training partition of the datasets. For other datasets used for testing, we use a test partition if it is available, and use randomly sampled data if no predefined partition is available.

We resize images to spatial dimension 32×32 for all datasets. We trained VAE on

Encoder	Decoder
Input x	Input z, reshape to $\text{nz} \times 1 \times 1$
4×4 Conv_{nf} Stride 2, BN, ReLU	4×4 $\text{Deconv}_{4\times\text{nf}}$ Stride 1, BN, ReLU
4×4 $\text{Conv}_{2\times\text{nf}}$ Stride 2, BN, ReLU	4×4 $\text{Deconv}_{2\times\text{nf}}$ Stride 2, BN, ReLU
4×4 $\text{Conv}_{4\times\text{nf}}$ Stride 2, BN, ReLU	4×4 $\text{Deconv}_{\text{nf}}$ Stride 2, BN, ReLU
4×4 $\text{Conv}_{2\times\text{nz}}$ Stride 1	4×4 $\text{Conv}_{256\times\text{nc}}$ Stride 2

Table 4.6: Network structure for VAE based on DCGAN. $\text{nz} = 100$ for all models. For VAE trained on Fashion MNIST, $\text{nf} = 32, \text{nc} = 1$; for VAE trained on CIFAR-10, $\text{nf} = 64, \text{nc} = 3$.

Fashion MNIST using $32 \times 32 \times 1$ images, and when we test it on color images, we use only the first channel of the color image. When we use Fashion MNIST or MNIST images to test the VAE trained on CIFAR, we copy the channel three times to make them $32 \times 32 \times 3$.

When computing quantitative metrics, we use 5000 randomly chosen images from in-distribution and OOD datasets, respectively.

Implementation Detail

The training and testing of our models largely follow the setting of [Nalisnick et al., 2018]. In particular, we train VAEs with the DCGAN [Radford et al., 2015] structure. We present the network structure in Table 4.6.

On Fashion MNIST we train the VAE for 100 epochs with constant learning rate $5e-4$ using Adam optimizer and batch size 64. On CIFAR-10 we train the VAE for 200 epochs with constant learning rate $5e-4$ using Adam optimizer and batch size 64. When computing Likelihood Regret, we have the choice of optimizing the whole encoder or only optimizing the mean and variance of posterior. For the former, we start with the trained encoder and optimize its parameters for 100 steps using the Adam optimizer with learning rate $1e-4$. For the later, we start with the encoding mean and variance, and run optimization for 300 steps using Adam optimizer with learning rate $1e-4$.

Implementing Competing Methods

Input complexity adjusted likelihood [Serrà et al., 2019] is computed by subtracting a measure of the input's complexity from the negative log likelihood with. The input complexity can be obtained from the length of the binary string returned by some lossless compression algorithms. We simply follow their work and use PNG compression and JPEG2000 compression implemented in OpenCV.

Likelihood ratio with background model [Ren et al., 2019] is computed by subtracting the log likelihood of the background model from the log likelihood of the main model. The background model is trained by perturbing a proportion of randomly chosen pixels, where the perturbation is done by replacing the pixel value by a uniformly sampled random value between 0 and 255. One of the key hyper-parameters μ, is the percentage of pixels to be perturbed. The authors suggest to choose μ between 0.1 and 0.3. We do a simple grid search on $\{0.1, 0.2, 0.3\}$, and use the best one (0.3 for Fashion MNIST, and 0.2 for CIFAR-10). We trained the background VAE with the same setting of the main VAE, except that we apply $\lambda = 10$ L_2 weight decay as suggested by the authors.

Latent Mahalanobis distance [Bulusu et al., 2020] combines the reconstruction loss and Mahalanobis distance in the latent space as an OOD score. In particular, their score is defined by

$$\text{novelty } (\mathbf{x}) = \alpha \cdot D_M(E(\mathbf{x})) + \beta \cdot \ell(x, D(E(\mathbf{x}))),$$

where D and E are the decoder and encoder, respectively. The second term is the reconstruction loss, and the first term is the Mahalanobis distance (D_M) between the encoded sample and the mean vector of the encoded training set. Their method is designed for auto-encoders, so we take $E(x)$ to be the mean of the variational posterior output by the encoder. As suggested by the author, to balance the two terms, α was set to the reciprocal

of the standard deviation of the Mahalanobis distance between the encoded validation data and the mean latent train vector, and β to the reciprocal of the standard deviation of the reconstruction error on the validation set.

Additional Quantitative results: AUPRC and FPR80

	LR_E	LR_Z	Likelihood	IC (png)	IC (jp2)	Likelihood Ratio	LMD
MNIST	**0.980**	0.938	0.344	0.923	0.563	0.917	0.866
CIFAR-10	0.995	0.998	**1**	0.904	**1**	0.912	0.998
SVHN	0.998	**1**	0.996	0.993	1	0.621	0.999
KMNIST	**0.993**	0.985	0.734	0.642	0.568	0.984	0.962
NotMNIST	0.999	**1**	0.935	0.943	0.953	0.996	0.999
Noise	0.983	0.954	**1**	0.4342	**1**	**1**	**1**
Constant	0.999	**1**	0.915	**1**	**1**	0.628	0.985

((a)) VAE trained on Fashion MNIST

	LR_E	LR_Z	Likelihood	IC (png)	IC (jp2)	Likelihood Ratio	LMD
MNIST	0.993	0.979	0.307	**0.997**	0.988	0.665	0.308
FMNIST	0.980	0.956	0.314	0.982	**0.987**	0.656	0.337
SVHN	0.841	0.799	0.343	**0.922**	0.914	0.337	0.389
LSUN	**0.681**	0.610	0.508	0.611	0.408	0.611	0.520
CelebA	**0.715**	0.634	0.474	0.572	0.509	0.405	0.524
Noise	0.940	0.814	**1**	0.313	0.318	**1**	0.964
Constant	0.965	0.924	0.445	**1**	**1**	0.410	0.593

((b)) VAE trained on CIFAR-10

Table 4.7: AUCPRC of Likelihood Regret (LR) and other OOD detection scores on different datasets.

	LR_E	LR_Z	Likelihood	IC (png)	IC (jp2)	Likelihood Ratio	LMD
MNIST	0.02	0.09	0.97	0.1	0.68	0.15	0.2
CIFAR-10	0	0	0	0.16	0	0.02	0
SVHN	0	0	0	0	0	0.35	0
KMNIST	0.01	0.01	0.51	0.44	0.57	0.02	0.06
NotMNIST	0	0	0.01	0.11	0.01	0.01	0
Noise	0	0.06	0	0.57	0	0	0
Constant	0	0	0.06	0	0	0.27	0.01

((a)) VAE trained on Fashion MNIST

	LR_E	LR_Z	Likelihood	IC (png)	IC (jp2)	Likelihood Ratio	LMD
MNIST	0.01	0.02	1	0.01	0	0.27	1
FMNIST	0.02	0.06	0.99	0.01	0.01	0.32	0.98
SVHN	0.18	0.27	0.97	0.05	0.1	0.91	0.97
LSUN	0.46	0.55	0.77	0.5	0.86	0.62	0.77
CelebA	0.37	0.48	0.68	0.55	0.67	0.82	0.67
Noise	0.02	0.08	0	0.99	0.93	0	0
Constant	0.03	0.09	0.98	0	0	0.65	0.96

((b)) VAE trained on CIFAR-10

Table 4.8: FPR80 of Likelihood Regret (LR) and other OOD detection scores on different datasets.

More Reconstruction Examples

In this section, we present examples of reconstructed images in different datasets. See Figure 4.7 and Figure 4.8. We observe that VAE trained on CIFAR-10 can reconstruct images from multiple datasets very well.

((a)) Fashion MNIST

((b)) MNIST

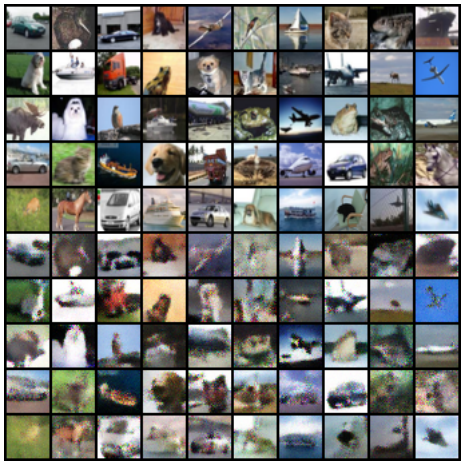
((c)) CIFAR-10

((d)) SVHN

Figure 4.7: Some examples of reconstructed images using VAE trained on Fashion MNIST. For each subfigure, the first 5 rows are original images and the last 5 rows are their corresponding reconstructions.

((a)) Fashion MNIST

((b)) MNIST



((c)) CIFAR-10

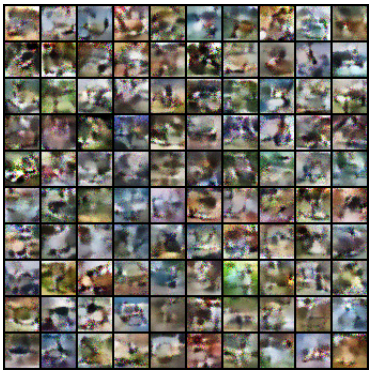
((d)) SVHN

Figure 4.8: Some examples of reconstructed images using VAE trained on CIFAR-10. For each subfigure, the first 5 rows are original images and the last 5 rows are their corresponding reconstructions.

Randomly Generated Samples

We show some randomly generated samples from VAEs in Figure 4.9. Although the samples are blurry and a bit noisy (due to the cross entropy loss), the semantics of the samples suggest that our VAEs model the in-distribution data well.

((a)) Fashion MNIST



((b)) CIFAR

Figure 4.9: Some examples of randomly generated samples.

4.3 Controllable Generation

4.3.1 Motivation and Background

Generative models learn the distribution of the target dataset and generate new samples through a particular sampling schedule. An essential aspect of the generation is whether we can manually control specific properties of the generated samples. For example, customers may require us to create an image from MNIST with particular features: the image should contain the digit seven on the top-right with a given thickness and skewness. Therefore, an ideal model should give options to produce accurate target images during generation.

One direction to explore controllable generation is by building a disentangled generative model. As we know, a generator G maps a latent vector $z \in \mathcal{Z}$ to a sample in the target domain $x \in \mathcal{X}$. Disentangled generative models ensure that each latent factor controls a distinct and independent aspect of the generated sample. For example, for a disentangled generator of MNIST, z_1 might only control the position of the digit in the image while z_2 and z_3 only control the skewness and thickness. When users want to set the position of the generated number, they only need to traverse over the space of z_1 and select the desired one. Researchers have explored the disentanglement property of different generative models and have developed plenty of unsupervised methods to enhance the disentanglement on VAEs, Flows and GANs. Most of them add additional regularization terms on the loss function to limit the information occupied by each latent factor and force them to control separate dimensions [Higgins et al., 2016, Lin et al., 2020, Ren et al., 2021].

Because of their unsupervised nature, previous disentangled generative models suffer from low accuracy and low-quality generation. Another direction for constructing a controllable generative model is conditional generation, which generates higher precision and quality samples. A conditional generative model requires a latent space $\mathcal{Z}$ and a condition space $\mathcal{C}$. The generator $G : \mathcal{Z} \times \mathcal{C} \to \mathcal{X}$ maps the concatenation of a latent vector and a condition

vector to a valid sample that satisfies the input condition. Since this is a supervised method, it requires a dataset from the target domain with condition labels $(x, c) \in \mathcal{X} \times \mathcal{C}$ so that the model can learn to generate samples from specific sub-areas of the domain that corresponds to the given condition c. A classification model learns to predict the label c when x is given, while this conditional generator learns the inverse process of how to rebuild x when the label c is provided. A labeled dataset provides more knowledge to the model, helping it to distinguish and generate samples with different conditions. The limitation of this method appears when multiple condition spaces are mixed. It is complicated to prepare a dataset with a clear label for multiple conditions $\prod_{i=1}^{k} \mathcal{C}_i$. Besides, the dataset will become intolerable large to ensure enough data points under different combinations of conditions.

We will focus on a specific domain of controllable generation: controlling the pose and expression when generating human faces. In section 4.3.2 we introduce details of the facial generation problem. Section 4.3.3 discusses key-points-based methods, which include 2D key-points and 3D key-points methods. In section 4.3.4 we show the details of 3D morphable model (3DMM), which provides an easy way to disentangle the identity information and motion & expression information from a facial image. Finally, our contributions are shown in section 4.3.5, including a combination of supervised and unsupervised methods and a combination of 3DMM and key-points based methods, which not only fixes the drawbacks of previous key-points-based methods but also improves the generation accuracy and quality.

4.3.2 Controling the Pose of the Human Face

We are interested in manipulating different properties of human face images. The domain dataset is restricted to images containing only one person's head and neck parts. Each image contains several features that are crucial to that problem: the identity information about whom this image belongs to, the pose information about the position of the head and the direction the face looks towards, and information about the facial expression present

in the image. The target is to build a model that can freely generate facial images with given pose and expression information. It is closely related to the task of face reenactment and motion transfer, which aims at automatically synthesizing videos by combining the appearance extracted from a source image with motion patterns derived from a driving video [Siarohin et al., 2019]. Both tasks are looking for a model to generate new facial images under specific pose and expression information. However, controllable generation does not restrict the source of that information, while face reenactment needs to extract that information from the driving video. Despite this, controllable generation is not more straightforward than face reenactment since we do not have a satisfying facial dataset containing every image's pose and expression information. Therefore, extracting the pose and expression information in face reenactment models is also crucial. We will focus on the face reenactment task and explore the corresponding models.

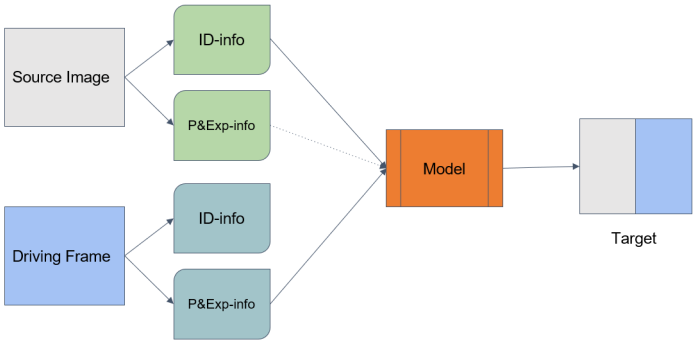


Figure 4.10: Basic framework for reenactment models

Figure 4.10 shows that basic framework for a reenactment model, including a step of separating information from facial images and a conditional generator to combine identity information from the source image and pose & expression information from the driving image. The final output is a face image of the person from the source image but mimicking the motion and expression extracted from the driving image.

The dataset we use is VoxCeleba2, which contains over 1 million video segments for 6,112

celebrities extracted from videos uploaded to YouTube.

4.3.3 Key-points Based Methods

Extracting a set of key points is thought of as a precise yet easy way to represent pose and expression information from a facial image. Those key points depict the contour and skeleton of a face only using a few points, which significantly compresses the representation but still keeps the essential information to model motion and expression. An unsupervised method learns a key points extractor and a conditional generator together to achieve face reenactment without external knowledge. Nevertheless, with the help of human-labeled data, a supervised method generates more accurate and higher-quality samples.

Unsupervised Methods

First order motion model (FOMM) [Siarohin et al., 2019] is an unsupervised model consisting of a key-points extractor L, a dense motion model M, and a generator G. Given a source image I_s and a driving video $V = (V_1, ..., V_k)$, the target is to generate a new video $Y = (Y_1, ..., Y_k)$ which keeps the identity information from I_s while mimicking the motion and expression information from the driving video. In general, if I_s and V are arbitrary, then it is hard to obtain the ground truth video Y for training. The trick is to extract I_s and V from the same video to let them have the same identity information. Then, we have valid ground truth $Y = V$, and the training can be conducted with a reconstruction loss.

Taking a facial image I as input, the key-points extractor L outputs a set of key-points $\{p_1, ..., p_k\}$ and a set of 2×2 matrices $\{A_1, ..., A_k\}$. It is designed to represent each face by a combination of k parts, and every key point is a local center point for a part. Each pair (p_j, A_j) forms an affine mapping $T_j(p) = A_j(p - p_j)$ that is thought of as a first-order approximation of the motion mapping between the j-th part of I and the center part of a reference frame R, which maps a point of I to a point of R.

Assume

$$L(I_s) = \{p_{sj}, A_{sj}, j = 1, ..., k\},\ L(V_i) = \{p_{dj}, A_{dj}, j = 1, ..., k\},$$

then

$$T_{sdj}(p) = T_{sj}^{-1} \circ T_{dj}(p) = A_{sj}^{-1} A_{dj}(p - p_{dj}) + p_{sj}, (j = 1, ..., k)$$

builds a group of local mappings between the source image I_s and the driving frame V_i.

Next the dense motion model M takes the source image I_s and a set of local linear mappings $\{T_{sdj}, j = 1, ..., k\}$ as input and aims to output a global motion mapping T_G that is assumed to capture the motion transfer between I_s and V_i. The trainable parameters are contained in a convolution neural network M_θ. The model process can be summarized by the following steps:

1. compute k warped images (or features): $l_j = \text{warp}(I_s, T_{sdj}), j = 1, ..., k$;
2. compute the heatmap representation for each $T_{sdj} : p \mapsto A_{sj}^{-1} A_{dj}(p - p_{dj}) + p_{sj}$.

$$h_j(z) = exp\left(\frac{(p_{dj} - z)^2}{\sigma}\right) - exp\left(\frac{(p_{sj} - z)^2}{\sigma}\right);$$

3. compute the weights: $W = M_\theta(\{l_j\}, \{h_j\})$;
4. obtain the final output: $T_G(p) = \sum_{j=1}^{k} W(p) T_{sdj}(p)$.

In the end the generator takes I_s and T_G as input and generates the target image $\hat{V}_i = G(I_s, T_G)$. The 2D mapping T is conducted over an image or an image feature map F through the warping operation:

$$\text{warp}(F, T)[i, j] = F[T(i, j)_x, T(i, j)_y], \forall \text{2D location: } (i, j) \tag{4.5}$$

where warp(F, T) is the output feature map whose (i, j)-th entry takes the value from an entry of input F that is determined by the 2-d mapping T. If $T(i, j)$ returns non-integers, then interpolation is applied. After the warping layer, warped features are fed into several convolution layers to generate the final output. These can be summarized as the following steps:

1. compute the warped image (or feature): $l = \text{warp}(I_s, T_G)$;
2. obtain the final output: $\hat{V}_i = G_\phi(l)$.

The trainable parameters are contained in G_ϕ. These are the steps for a 2D model, and if it is a 3D model, then the input image should be replaced by a 3D feature map returned from an encoder. Also, the coordinate mapping and warping operation are lifted to 3D.

Overall, this method uses purely end-to-end training without any external information other than video itself. The reconstruction loss function

$$L(\hat{V}_i, V_i) = ||\hat{V}_i - V_i||$$

simultaneously guides the key points extraction and also the facial images conditional generation.

From 2D to 3D

The previous method uses 2D key points, which are natural and straightforward for 2D images and 2D feature maps related to our setting. Nevertheless, using 2D points to capture 3D pose and motion loses part of the information. Thus, a better choice is to lift the settings to 3D space. Note that a feature map derived from a 2D-convolution layer is a 3D tensor (ignoring the batch dimension) where the first two dimensions correspond to the height and width while the last dimension contains different channels. Therefore, applying 2D warping over such feature maps is straightforward, but it is not the case for 3D warping.

In the 3D setting, the key points extractor L outputs a set of 3D key points and a set of 3×3 matrices to build linear mappings in 3D space. The feature map lifts to a 4D tensor with three spatial dimensions: height, width, and depth, along with a channel dimension. To be adapted to 4D-feature maps, the 2D convolution layer changes to the 3D alternative. In addition, the 3D warping operates over the 4D feature maps derived from an encoder instead of directly over the original images.

In theory, extending from 2D key points to 3D key points strengthens the model's ability to capture 3D changes between different frames in a video, yet it also makes the model harder to train, especially in an unsupervised way. Therefore, adding particular supervision during training is necessary to stabilize the training process [Wang et al., 2021].

Supervised Method

The most efficient supervision method provides information on the correct locations of the key points. There exist works to predict 2D/3D facial landmarks given a single face image. Figure 4.11 shows some examples of 2D landmarks and figure 4.12 contains some 3D examples. As we can see, this landmark predictor has already captured the contour of the input face, generating key points that explain the 3D pose and motion information in the image.

Figure 4.11: 2D facial landmarks.

Instead of training the key points extractor L, dense motion model M, and generator G together, we can use a pre-trained landmark detector to predict the landmarks, set them as the key points and only train M, G through the reconstruction loss. This step simplifies

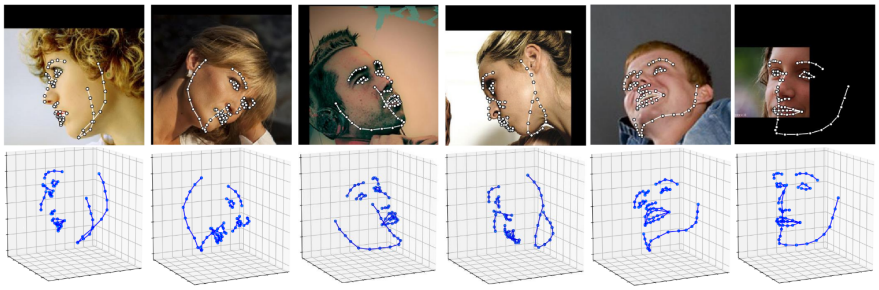


Figure 4.12: 3D facial landmarks.

the training process and improves the sample quality and convergence speed. However, eliminating the training of the key point predictor also prevents the model from learning a better representation to transfer 3D motion, limiting the model's ability to explore other directions in key point space.

4.3.4 The 3D Morphable Model

With an unsupervised model or a supervised model of face reenactment, we can extract pose and expression information from a face image and transfer the pose and motion from image to image. Nevertheless, we cannot construct new poses and expression information from scratch. We can always train a generative model on the space of pose and expression, but applying the 3D morphable model(3DMM) to model pose and expression significantly simplifies this step.

3DMM models a face with a 3D mesh grid $V = \{v_i = (x_i, y_i, z_i)\}_{i=1}^{N}$. As a result, each face has a representation $V \in \mathbb{R}^{N\times 3}$. Usually $N \sim 60000$ is large, to speed up and simplify the usage of 3DMM, it provides basis matrices derived from PCA to lower the dimension. The 3DMM formulas are:

$$V_c = \bar{x} + U^{id}P^{id} + U^{exp}P^{exp}, \tag{4.6}$$

$$V = s \cdot R \cdot \text{reshape}(V_c) + b, \tag{4.7}$$

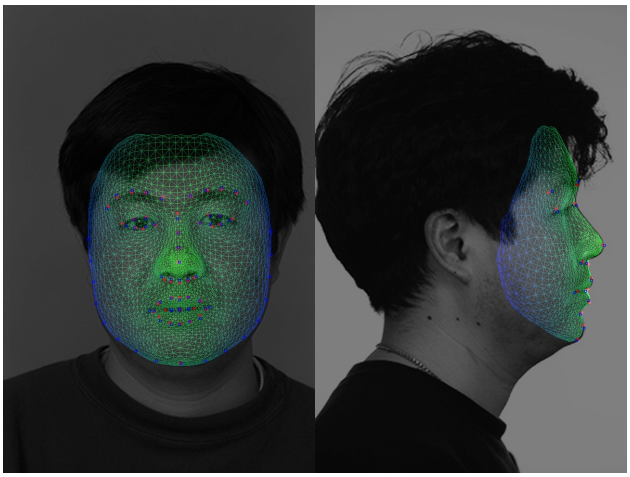


Figure 4.13: Mesh grid of 3DMM.

where $U^{id} \in \mathbb{R}^{3N\times\alpha}$, $U^{exp} \in \mathbb{R}^{3N\times\beta}$ are two basis matrices; $\bar{x} \in \mathbb{R}^{3N}$ is the average 3D mesh of all the faces in the space; $P^{id} \in \mathbb{R}^{\alpha}$ is the coefficient vector to control the identity information; $P^{exp} \in \mathbb{R}^{\beta}$ is the coefficient vector to model tiny expression shift; $V_c \in \mathbb{R}^{3N}$ is the canonical mesh grid; reshape$(V_c) \in \mathbb{R}^{N\times 3}$ is a reshaped version of V_c so that we can apply 3D geometric transformation on each 3D point; $(s \in \mathbb{R}^{+}, R \in \mathbb{R}^{3\times 3}, b \in \mathbb{R}^{3})$ are the parameters of 3D geometric transformation: s is scaling parameter, R is rotation matrix, b is translation vector.

Each mesh is represented by $(P^{id}, P^{exp}, s, R, b) \in \mathbb{R}^{\alpha+\beta+1+3+3}$. In this thesis we use U^{id} from Basel Face Model [Paysan et al., 2009] and U^{exp} from FaceWarehouse [Cao et al., 2013]. The Basel Face Model(BFM) builds the basis matrix U^{id} from registered 3D scans of 100 male and 100 female faces. The geometry of the BFM consists of 53490 3D vertices connected by 160470 triangles. Faces of different identities can be composed as linear combinations of 199 principal components. FaceWarehouse is a database of 3D facial expressions for visual computing applications, which captures different expressions from 150 individuals aged 7-80 from various ethnic backgrounds. For each person, the authors captured the RGBD data of his/her different expressions, including the neutral expression and 19 other expressions such as mouth-opening, smile, kiss, etc. For every RGBD raw data record, a set of facial

feature points on the color image such as eye corners, mouth contour, and the nose tip are automatically localized and manually adjusted if better accuracy is required. They then deform a template facial mesh to fit the depth data as closely as possible while matching the feature points on the color image to their corresponding points on the mesh. These fitted face meshes construct a set of individual-specific expression blendshapes for each person. We use their PCA basis matrix U^{exp} to model the high dimensional expression space. Since $\alpha, \beta \sim 100$, this representation is much lower-dimensional and easier to compute. These two models use the same 3D mesh to model faces, although the two basis matrices U^{id} and U^{exp} are not orthogonal.

As we can see in figure 4.13, the facial landmarks are already a subset of the 3D mesh grid. Therefore, on the one hand, 3DMM coefficients are sufficient to compute facial landmarks. On the other hand, we can obtain 3DMM coefficients through optimization with the knowledge of facial landmarks. To do that, we define a function H that maps the 3DMM coefficients to landmarks:

$$H(P^{id}, P^{exp}, s, R, b) = V[\text{IDX}, :], \tag{4.8}$$

where $V = V(P^{id}, P^{exp}, s, R, b)$ is the 3D-mesh defined in (4.7), and IDX is the corresponding index of each landmark point in the large mesh matrix V. Suppose we have the facial landmarks K_I of a face, we can compute the 3DMM coefficients through optimization:

$$(P^{id}, P^{exp}, s, R, b)_I = \text{argmin}_{(P^{id}, P^{exp}, s, R, b)} \, ||H(P^{id}, P^{exp}, s, R, b) - K_I||_2^2.$$

We can obtain the coefficients through vanilla gradient descent, but usually we initialize (P^{id}, P^{exp}) with $\mathbf{0}$ vectors and first run an optimization to obtain a proper set of rigid-transformation parameters (s, R, b), and then we run an optimization on the full set of coefficients $(P^{id}, P^{exp}, s, R, b)$ to the end.

With the help of 3DMM, it is easy to manipulate 3D facial landmarks: (s, R, b) control the pose; P^{id} controls the identity information while P^{exp} controls the expression information. To generate a new set of 3D facial landmarks, we can sample the 3DMM coefficients. Since the PCA step already normalizes the space of P^{id}, P^{exp}, we can either directly sample from standard Gaussian distribution or fit a multivariate Gaussian (or a VAE model) to obtain new samples.

4.3.5 Combination and Improvement

We improve the key-points-based methods in two ways:

1. we combine the unsupervised and supervised methods to build a model that shares the advantages from both areas and captures more delicate motions than old methods;

2. we leverage 3DMM over key-points-based methods to enhance the model's generalization ability without changing the training process.

Adding the Shift Predictor

Our new key-points-based model contains a Delta Net D, a 3D encoder E, a dense motion net M, and a Generator G. In addition, a pre-trained 3D landmark detector F is required to supply initialization for key points. Figure 4.14 provides the details of our model.

Given a source image I_s and a driving image V, the model generates the target image with the following steps:

1. Compute 3D landmarks: $q_s = F(I_s) \in \mathbb{R}^{k\times 3}, q_d = F(V)$;

2. Compute key points corrections for both the source image and the driving image: $\delta_s = D(I_s) \in \mathbb{R}^{k\times 3}, \delta_d = D(V)$;
   Obtain key points: $p_s = q_s + \delta_s, p_d = q_d + \delta_d$;

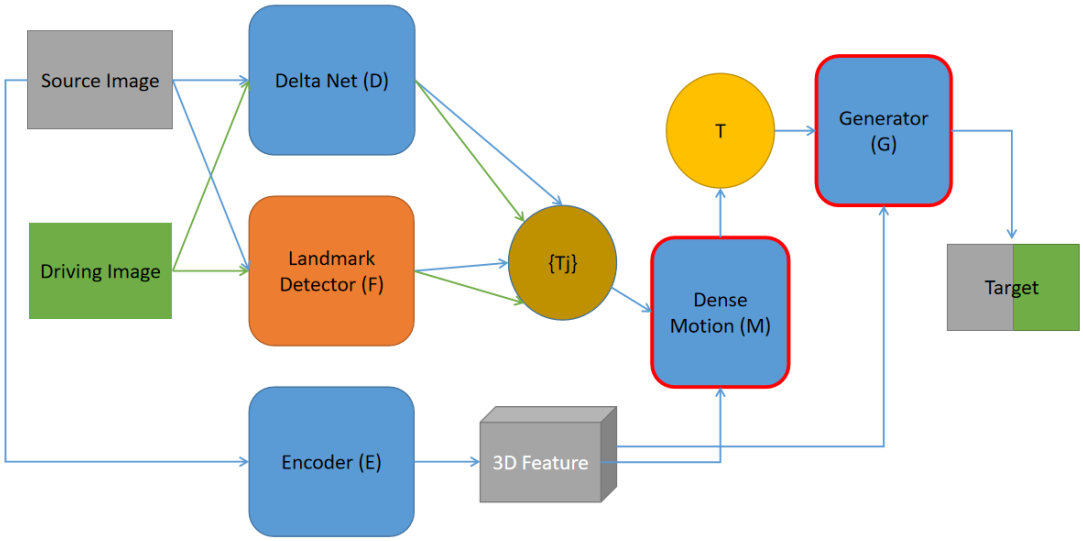


Figure 4.14: Details of our model with delta net. Blue parts contain trainable parameters; red frames contain warping operation.

3. Build linear shifts based on key points: $T_j(p) = (p - p_{dj}) + p_{sj}, j = 1, ..., k$;

4. Compute 3D feature map of the source image: $f_s = E(I_s)$;

5. Obtain global motion mapping: $T = M(f_s, \{T_j\})$;

6. Obtain target: $\hat{V} = G(f_s, T)$.

Compared to the original unsupervised key-points-based method, we change the key point predictor into a delta net and add a pre-trained landmark detector. Also, we remove the matrix part $\{A_j\}$ to simplify the process of building linear mappings without harming the model capacity. We deploy a landmark predictor to get the initialization of key points. Then, a delta net is trained to fine-tune the key points, providing enough freedom to search and discover the best direction of key points for motion transfer.

The loss function is the weighted sum of a reconstruction loss and a sparse penalty over

δ:

$$L(V, \hat{V}) = ||V - \hat{V}|| + \lambda \cdot (||\delta_s||_1 + ||\delta_d||_1). \tag{4.9}$$

The L_1 penalty constrains the scale of δ, which ensures the key points are not far from the initial landmarks. The FOMM leverages the perceptual loss [Johnson et al., 2016] as the reconstruction loss. Besides, it also has an equivariance constraint to regularize the training of the keypoint extractor, which forces the model to predict consistent keypoints with respect to known geometric transformations [Siarohin et al., 2019]. In our model, we use the LPIPS [Zhang et al., 2018] loss (a more delicate perceptual loss) as the reconstruction loss, and we do not need the equivariance constraint since we have prior knowledge of facial landmarks.

With the help of the pre-trained face landmark predictor, the model receives the facial landmarks information at the early training stage, which speeds up and stabilizes the training process. In the meantime, the delta net ensures the model builds its knowledge about the key points instead of keeping the initial predicted landmarks fixed, as we know the facial landmarks cannot always be the best key points to represent motion. We will show that in the experiments section 4.3.6, an unexpected advantage of delta net is that it can capture other tiny shifts, for example, eyeball movement, which the other models always ignore.

Combining 3DMM and the Key Points Based Method

A drawback of the key-points-based method is that the generalization ability is limited. Note that both the source and driving images are from the same video. Thus, in training we can only transfer motion within the same video. In applications, we either need the source image and the driving video to contain the same figure, or we need the source image and the first frame of the driving video contain a face with the same pose. Otherwise, it will generate weird animation.

To overcome this constraint, we propose to use 3DMM to enhance the generalization ability during the inference stage. Given a source image I_s and a driving video $V = \{V_i\}_{i=1}^N$, 3DMM fits a mesh grid for each of the facial images:

$$\text{3DMM: } I_s \to (P_0^{id}, P_0^{exp}, s_0, R_0, b_0) \in \mathbb{R}^\alpha \times \mathbb{R}^\beta \times \mathbb{R}^1 \times \mathbb{R}^3 \times \mathbb{R}^3$$
$$\text{3DMM: } V \to (Q^{id}, Q^{exp}, s, R, b) \in \mathbb{R}^{1\times\alpha} \times \mathbb{R}^{N\times\beta} \times \mathbb{R}^{N\times1} \times \mathbb{R}^{N\times3} \times \mathbb{R}^{N\times3}.$$

Here $Q^{id} \in \mathbb{R}^{1\times\alpha}$ because all the frames share the same identity information within the driving video, while for other coefficients, each frame has a unique version.

Next we compute the transferred 3DMM mesh and facial landmarks:

$$\text{3DMM: } (P_0^{id}, Q^{exp}, s, R, b) \to \{\text{mesh}_i\}_{i=1}^N \to \{\text{ldmk}_i\}_{i=1}^N.$$

This set of landmarks is computed using the identity information from I_s and the pose and expression information from the driving video. Finally, we input these landmarks into the dense motion net and generator to obtain the target animation.

With the help of 3DMM, we overcome the limitations of the original model. In addition, we automatically obtain a way to disentangle identity information and pose&expression information from a facial image.

4.3.6 Experiments

First, we show an example of face reenactment. Figure 4.15 contains the source image, figure 4.16 presents the comparison results between several models: a supervised 2D model; our model without 3DMM; headGAN [Doukas et al., 2020]; and our model with 3DMM. HeadGAN is another face reenactment model that does not use key points but Projected Normalized Coordinate Code (PNCC) from 3DMM to capture 3D pose and motion.

As we can see, our model can capture the motion of eyeballs while other methods cannot.

Figure 4.15: Example of source image.

Figure 4.16: Comparison between different models. 2D means an supervised 2D key points model; 3D means our model; 3DMM means headGAN; 3DMM+3D means our model with 3DMM.

Also, with the help of 3DMM, the model becomes precise about the 3D pose of the face.

Figure 4.17 shows the problem of the generalization ability for key-points-based methods. Since the key points still contain identity information (for example, the size of the face), it is easy to transfer this part to the generated image. As shown in the example, the model generates an enlarged face when the driving image contains a larger face than the source image. This issue is resolved with the help of 3DMM.

Next, we show some examples of pose control and face frontalization. In figure 4.18, we generate a frontal face that looks towards the camera in each case. Given a face image I_s, we first compute the 3DMM coefficients $(P^{id}, P^{exp}, s, R, b)$ and the corresponding facial landmarks K_I. Next, we replace the rotation matrix R with the one obtained from zero angles R_0 and re-compute the facial landmarks from (4.8): $K_{I0} = H((P^{id}, P^{exp}, s, R_0, b))$. In the end, our model generates the frontal face from the steps in 4.3.5. Since we do not have a driving image, we directly set $p_s = K_I, p_d = K_{I0}$ without running the delta net. Unfortunately, ignoring the delta step harms the model's power, and the generated faces stare in strange directions, which is reasonable as we do not provide eyeball information in

Figure 4.17: Comparison w/w.o. 3DMM. From left to right: source image; driving image; generated sample without 3DMM; generated sample with 3DMM.

the driving facial landmarks. To generate the frontal face that looks towards the camera, we need to add a shift on the eyeball points of the driving landmarks: $K'_{I0} = K_{I0} + \triangle_{eye}$. $\triangle_{eye}$ only depends on (s, R), so it is easy to compute.

Figure 4.18: Example of face frontalization. In each case, left is the original image, right is the transferred image.

Finally, we show an example of controlling the expression. As shown in figure 4.19, with the help of 3DMM, we can quickly generate new expressions or manually change the expression to the desired one. Given a face image I_s, we first obtain the 3DMM coefficients: $(P^{id}, P^{exp}, s, R, b)$ and the facial landmarks K_I. Next we randomly sample $q^{exp} \sim N(0, I_\beta)$ and re-compute the facial landmarks with a random expression: $K_{Ir} = H(P^{id}, q^{exp}, s, R, b)$. In the end, we follow the steps in 4.3.5 to generate a new face. Similar to the pose control experiments, as we do not have a driving image, we directly set $p_s = K_I, p_d = K_{Ir}$. The difference is that the generator always outputs a good face even if we ignore the delta step. The landmarks endure smaller changes in expression control experiments than pose control experiments. The output images look reasonable even if the model does not modify the position of the eyeballs. Again we can always add an eyeball shift to the driving landmarks:

$K'_{I0} = K_{I0} + \triangle_{eye}$ to generate more variable faces.

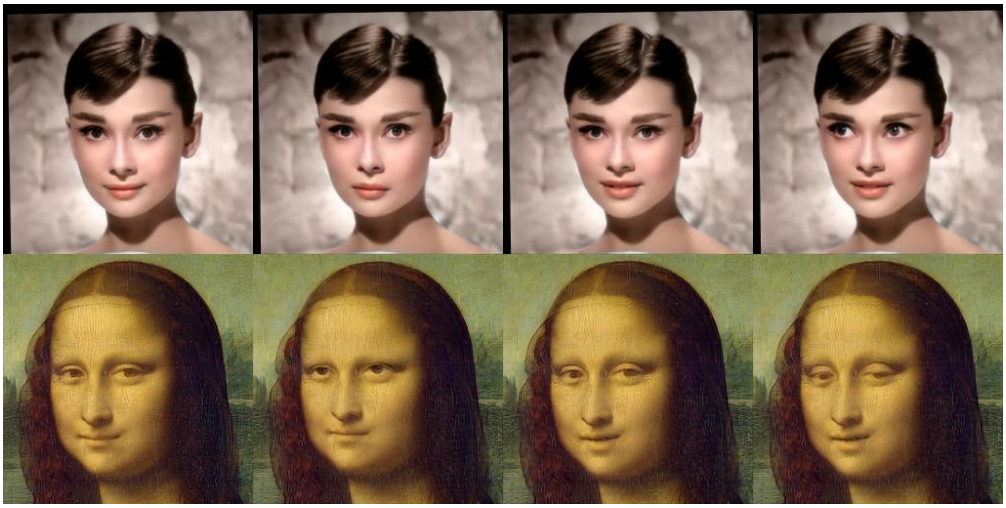


Figure 4.19: Random samples with different expressions. From left to right: original image; generated image with random expression; generated image with a different random expression; generated image after manually changing the place of eyeballs.

4.3.7 Conclusion

We propose to add a delta net and use 3DMM in key-points-based methods of face reenactment. We show that our method resolves the drawbacks of the original key-points-based method and captures finer motions like eyeball movements. In addition, our model is easier to train and generates higher-quality samples. We can freely control the pose and expression when generating human faces by running our model.

4.3.8 Appendix

Network Architecture

We follow the architecture settings in Figure 12 and Figure 13 of [Wang et al., 2021]. The encoder follows the structure in Figure 12 (a), the dense motion net follows the structure in Figure 12 (d), the generator following the structure in Figure 12 (e), the delta net follows

the structure of delta branch of Figure 12 (c). We also provides the details of architectures here in table 4.9 and table 4.10. For the details of each block, please refer to Figure 13 of [Wang et al., 2021].

Table 4.9: Network structure for Encoder and Generator

Encoder	Generator
Input x	Input: Warped feature $w(f_s, T)$
7×7 Conv_{64}, BN, ReLU	Reshape $D16 \times C32 \rightarrow C512$
DownBlock2D-128	3×3-Conv-256, BN, LReLU
DownBlock2D-256	1×1 Conv_{256}
Reshape $C512 \rightarrow D16 \times C32$	occlusion $\otimes$
ResBlock3D-32 $\times 6$	ResBlock2D-256 $\times 6$
Output feature map f_s	UpBlock2D-128
	UpBlock2D-64
	7×7 Conv_3-sigmoid
	Output Image

Table 4.10: Network structure for Dense Motion Net and Delta Net

Dense Motion Net	Delta Net
Input $w_k(f_s)$	Input: x
DownBlock3D-64	7×7-$\downarrow 2$ Conv_{64}, BN, ReLU,$\downarrow 2$
DownBlock3D-128	ResBottleneck-256 $\times 3$
DownBlock3D-256	ResBottleneck-512-$\downarrow 2$
DownBlock3D-512	ResBottleneck-512 $\times 3$
DownBlock3D-1024	ResBottleneck-1024-$\downarrow 2$
UpBlock3D-512	ResBottleneck-1024 $\times 5$
UpBlock3D-256	ResBottleneck-2048-$\downarrow 2$
UpBlock3D-128	ResBottleneck-2048 $\times 2$
UpBlock3D-64	7×7-AvgPool
UpBlock3D-32	fc60
$7 \times 7 \times 7$ Conv_{21}-softmax $\rightarrow$ mask	Output Delta δ
Reshape $\rightarrow 7 \times 7$ $\text{Conv}_1 \rightarrow$ occlusion	

Source of Figures

Figure 4.11, figure 4.12 are from https://github.com/1adrianb/face-alignment.

Figure 4.13 is from https://github.com/Yinghao-Li/3DMM-fitting.

CHAPTER 5

CONCLUSION

The deep generative model deserves systematic and comprehensive exploration as it is a promising and essential machine learning method. As a result, this thesis aims to provide more thoughts about understanding, improving, and applying modern DGMs, including the variational Auto-Encoder, the Flow-Based model, the Generative adversarial network, and the energy-based model.

The variational auto-encoder (VAE) combines the variational inference and the auto-encoder. Motivated by its bottleneck structure and the EM algorithm in variational inference, we propose an optimization trick to obtain the approximated posterior distribution of the latent variable that only requires the optimized decoder. Then we design a new model named Variational Latent Optimization (VLO) by applying this optimization trick during training. Roughly speaking, the VLO is a VAE without the encoder. We find that the VLO has similar performance under the standard case compared to the VAE, and it has advantages against the VAE when the training data size is limited. Besides, this optimization trick motivates us to build an Out-of-distribution (OOD) detection score called the Likelihood Regret upon the VAE model. Our experiments reveal that the Likelihood Regret is an effective OOD score that can correct the likelihood misalignment of VAE that happens when we directly detect OOD samples using the likelihood of VAE models.

We then study the approaches to improve the generation quality of VAEs as they usually generate blurred samples. First, we explore the VAE+learnable prior direction and make a novel discovery that it is necessary to increase the reconstruction loss weight to produce high-quality samples. Then we propose the Generative Latent Flow (GLF), which uses an auto-encoder to learn latent representations of the data and a normalizing flow to map the distribution of the latent variables to standard Gaussian. GLF can be seen as a variational auto-encoder, with normalizing flow prior and a vanishing limit of the pixel-wise variance of

the data. We compare our model with several related techniques and show that our method achieves state-of-the-art sample quality and diversity among AE-based models on commonly used datasets and is competitive with GANs' benchmarks under standard quantitative evaluations.

The energy-based model (EBM) builds an unnormalized data density called the energy function to capture the data distribution. We find that the maximum likelihood principle does not explain the training process of EBMs with short-run Langevin dynamics (LD). To develop new angles to rationalize the training process of EBMs, we propose noise-free sampling dynamics to replace the short-run LD. Doing so allows us to study the density induced by the dynamics (if the dynamics are invertible), and connect with Wasserstein-GANs by treating the dynamic as a generator model, the initial values as latent variables, and the loss as optimizing a critic defined by the very same energy that determines the generator through its gradient. We show that reintroducing the noise in the dynamics merely reduces the quality of the generator and the EBM training is effectively a self-adversarial procedure rather than maximum likelihood estimation. We further improve the generation quality of EBMs by introducing the generator update of WGAN.

Although the maximum likelihood principle does not reveal the truth of training EBMs with short-run LD in the pixel space, it can be used to explain the EBMs in the latent space, as the latent space is usually simpler and lower-dimensional. Based on that, we propose to train an EBM on the latent space of a pre-trained likelihood-based deep generative model to improve the sample quality of the latter. The energy-based model is efficiently trained by maximizing the data likelihood, and after training, new samples in the latent space are generated from the energy-based model and passed through the pre-trained generative model to produce samples in the data space. We show that our proposed method greatly improves the sample quality of popular likelihood-based generative models, such as normalizing flows and VAEs, with very little computational overhead.

Finally, we explore the controllable generative models in the human face area. We propose to improve the key-points-based face reenactment model by adding a delta net and using the 3D Morphable Model. The face reenactment task transfers the motion and expression pattern from a driving facial video to another source image. The current key-points-based method suffers from low-quality generation and insufficient ability to disentangle the facial inputs' motion information and identity information. We propose using a pre-trained 3D facial landmark detector with a delta net to improve the training speed and stability of the previous key-points-based method, which also ensures the model captures detailed movement, for example, eyeballs movements, those other methods fail to learn. Besides, by combining the key-point-based method with 3DMM, we successfully improve the key-point-based method's generalization ability. Experiments results confirm that our new method generates excellent samples and can be used for the controllable generation of human faces.

REFERENCES

Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. Ganomaly: Semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*, pages 622–637. Springer, 2018.

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593, 2014.

Simon Alexanderson, Gustav Eje Henter, Taras Kucherenko, and Jonas Beskow. Style-controllable speech-driven gesture synthesis using normalising flows. In *Computer Graphics Forum*, volume 39, pages 487–496. Wiley Online Library, 2020.

Y. Amit, U. Grenander, and M. Piccioni. Structural image restoration through deformable template. *Journal of the American Statistical Association*, 86(414):376–387, 1991.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. 2015.

Lynton Ardizzone, Radek Mackowiak, Ullrich Köthe, and Carsten Rother. Exact information bottleneck with invertible neural networks: Getting the best of discriminative and generative modeling. *arXiv preprint arXiv:2001.06448*, 2020.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

Matthias Bauer and Andriy Mnih. Resampled priors for variational autoencoders. *arXiv preprint arXiv:1810.11428*, 2018.

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. Invertible residual networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 573–582. PMLR, 09–15 Jun 2019. URL `http://proceedings.mlr.press/v97/behrmann19a.html`.

Saikiran Bulusu, Bhavya Kailkhura, Bo Li, Pramod K Varshney, and Dawn Song. Anomalous instance detection in deep learning: A survey. *arXiv preprint arXiv:2003.06979*, 2020.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. arxiv 2018. *arXiv preprint arXiv:1804.03599*, 1804.

Chen Cao, Yanlin Weng, Shun Zhou, Yiying Tong, and Kun Zhou. Facewarehouse: A 3d facial expression database for visual computing. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):413–425, 2013.

Tong Che, Ruixiang Zhang, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. Your gan is secretly an energy-based model and you should use discriminator driven latent sampling. *arXiv preprint arXiv:2003.06060*, 2020.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

Hyunsun Choi, Eric Jang, and Alexander A Alemi. Waic, but why? generative ensembles for robust anomaly detection. *arXiv preprint arXiv:1810.01392*, 2018.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

Adrián Csiszárik, Beatrix Benkő, and Dániel Varga. Negative sampling in variational autoencoders. *arXiv preprint arXiv:1910.02760*, 2019.

Bin Dai and David Wipf. Diagnosing and enhancing vae models. *arXiv preprint arXiv:1903.05789*, 2019.

David Dehaene, Oriel Frigo, Sébastien Combrexelle, and Pierre Eline. Iterative energy-based projection on a normal data manifold for anomaly localization. *arXiv preprint arXiv:2002.03734*, 2020.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Yuntian Deng, Anton Bakhtin, Myle Ott, Arthur Szlam, and Marc'Aurelio Ranzato. Residual energy-based models for text generation. *arXiv preprint arXiv:2004.11714*, 2020.

Taylor Denouden, Rick Salay, Krzysztof Czarnecki, Vahdat Abdelzad, Buu Phan, and Sachin Vernekar. Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance. *arXiv preprint arXiv:1812.02765*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

Jiarui Ding, Anne Condon, and Sohrab P Shah. Interpretable dimensionality reduction of single cell transcriptome data with deep generative models. *Nature communications*, 9(1): 1–13, 2018.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

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

Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *Advances in neural information processing systems*, pages 658–666, 2016.

Michail Christos Doukas, Stefanos Zafeiriou, and Viktoriia Sharmanska. Headgan: One-shot neural head synthesis and editing. *arXiv preprint arXiv:2012.08261*, 2020.

Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models. *arXiv preprint arXiv:1903.08689*, 2019.

Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

Ruiqi Gao, Erik Nijkamp, Diederik P Kingma, Zhen Xu, Andrew M Dai, and Ying Nian Wu. Flow contrastive estimation of energy-based models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7518–7528, 2020.

Ruiqi Gao, Yang Song, Ben Poole, Ying Nian Wu, and Diederik P Kingma. Learning energy-based models by diffusion recovery likelihood. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=v_1Soh8QUNc`.

Partha Ghosh, Mehdi SM Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From variational to deterministic autoencoders. *arXiv preprint arXiv:1903.12436*, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*, 2019.

Karol Gregor, George Papamakarios, Frederic Besse, Lars Buesing, and Theophane Weber. Temporal difference variational auto-encoder. *arXiv preprint arXiv:186.03107*, 2018.

Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*, 2018.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Yedid Hoshen and Lior Wolf. Nam: Non-adversarial unsupervised domain mapping. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 436–451, 2018.

Yedid Hoshen, Ke Li, and Jitendra Malik. Non-adversarial image synthesis with generative latent nearest neighbors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5811–5819, 2019.

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

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.

Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

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

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

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Alexej Klushyn, Nutan Chen, Richard Kurle, Botond Cseke, and Patrick van der Smagt. Learning hierarchical priors in vaes. *arXiv preprint arXiv:1905.04982*, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *Journal of machine learning research*, 2017.

Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A conditional flow-based model for stochastic video generation. *arXiv preprint arXiv:1903.01434*, 2019.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. 2010.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 7167–7177, 2018.

Dan Li, Dacheng Chen, Jonathan Goh, and See-Kiong Ng. Anomaly detection with generative adversarial networks for multivariate time series. *arXiv preprint arXiv:1809.04758*, 2018.

Zengyi Li, Yubei Chen, and Friedrich T Sommer. Learning energy-based models in high-dimensional spaces with multi-scale denoising score matching. *arXiv preprint arXiv:1910.07762*, 2019.

Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference*, pages 689–698, 2018a.

Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*, 2018b.

Zinan Lin, Kiran Thekumparampil, Giulia Fanti, and Sewoong Oh. Infogan-cr and modelcentrality: Self-supervised model training and selection for disentangling gans. In *International Conference on Machine Learning*, pages 6127–6139. PMLR, 2020.

Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *Advances in neural information processing systems*, pages 700–708, 2017.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *Advances in Neural Information Processing Systems 31*, pages 700–709, 2018.

Lars Maaløe, Marco Fraccaro, Valentin Liévin, and Ole Winther. Biva: A very deep hierarchy of latent variables for generative modeling. In *Advances in neural information processing systems*, pages 6548–6558, 2019.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Charles E Metz. Basic principles of roc analysis. In *Seminars in nuclear medicine*, pages 283–298. WB Saunders, 1978.

Tom M Mitchell et al. Machine learning, 1997.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? *arXiv preprint arXiv:1810.09136*, 2018.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using a test for typicality. *arXiv preprint arXiv:1906.02994*, 2019.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011. URL `http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf`.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

Erik Nijkamp, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. Learning non-convergent non-persistent short-run mcmc toward energy-based model. *arXiv preprint arXiv:1904.09770*, 2019.

Erik Nijkamp, Ruiqi Gao, Pavel Sountsov, Srinivas Vasudevan, Bo Pang, Song-Chun Zhu, and Ying Nian Wu. Learning energy-based model with flow-based backbone by neural transport mcmc. *arXiv preprint arXiv:2006.06897*, 2020a.

Erik Nijkamp, Mitch Hill, Tian Han, Song-Chun Zhu, and Ying Nian Wu. On the anatomy of mcmc-based maximum likelihood learning of energy-based models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5272–5280, 2020b.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3d face model for pose and illumination invariant face recognition. In *2009 sixth IEEE international conference on advanced video and signal based surveillance*, pages 296–301. Ieee, 2009.

Wei Ping, Kainan Peng, Kexin Zhao, and Zhao Song. Waveflow: A compact flow-based model for raw audio. In *International Conference on Machine Learning*, pages 7706–7716. PMLR, 2020.

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

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. In *Advances in Neural Information Processing Systems*, pages 14680–14691, 2019.

Xuanchi Ren, Tao Yang, Yuwang Wang, and Wenjun Zeng. Do generative models know disentanglement? contrastive learning is all you need. *arXiv preprint arXiv:2102.10543*, 2021.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. In *Advances in Neural Information Processing Systems 31*, pages 5228–5237, 2018.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Joan Serrà, David Álvarez, Vicenç Gómez, Olga Slizovskaia, José F Núñez, and Jordi Luque. Input complexity and out-of-distribution detection with likelihood-based generative models. *arXiv preprint arXiv:1909.11480*, 2019.

Shai Shalev-Shwartz and Yoram Singer. Online learning: Theory, algorithms, and applications. 2007.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. First order motion model for image animation. *Advances in Neural Information Processing Systems*, 32:7137–7147, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jiaming Song, Yang Song, and Stefano Ermon. Unsupervised out-of-distribution detection with batch normalization. *arXiv preprint arXiv:1910.09115*, 2019.

Vijaya Kumar Sundar, Shreyas Ramakrishna, Zahra Rahiminasab, Arvind Easwaran, and Abhishek Dubey. Out-of-distribution detection in multi-label datasets using latent space of β-vae. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 250–255. IEEE, 2020.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Akinori Tanaka. Discriminator optimal transport. In *Advances in Neural Information Processing Systems*, pages 6813–6823, 2019.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071, 2008.

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

Jakub M Tomczak and Max Welling. Vae with a vampprior. *arXiv preprint arXiv:1705.07120*, 2017.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

Jacob Walker, Ali Razavi, and Aäron van den Oord. Predicting video with vqvae. *arXiv preprint arXiv:2103.01950*, 2021.

Ting-Chun Wang, Arun Mallya, and Ming-Yu Liu. One-shot free-view neural talking-head synthesis for video conferencing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10039–10049, 2021.

Wenlin Wang, Zhe Gan, Hongteng Xu, Ruiyi Zhang, Guoyin Wang, Dinghan Shen, Changyou Chen, and Lawrence Carin. Topic-guided variational autoencoders for text generation. *arXiv preprint arXiv:1903.07137*, 2019.

Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 681–688, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Zhisheng Xiao, Qing Yan, and Yali Amit. Generative latent flow. *arXiv preprint arXiv:1905.10485*, 2019.

Zhisheng Xiao, Qing Yan, and Yali Amit. Exponential tilting of generative models: Improving sample quality by training and sampling from latent energy. *arXiv preprint arXiv:2006.08100*, 2020.

Zhisheng Xiao, Karsten Kreis, Jan Kautz, and Arash Vahdat. Vaebm: A symbiosis between variational autoencoders and energy-based models. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=5m3SEczOV8L`.

J. Xie, Z. Zheng, R. Gao, W. Wang, S. C. Zhu, and Y. N. Wu. Generative voxelnet: Learning energy-based models for 3d shape synthesis and analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020. doi: 10.1109/TPAMI.2020.3045010.

Jianwen Xie, Yang Lu, Song-Chun Zhu, and Yingnian Wu. A theory of generative convnet. In *International Conference on Machine Learning*, pages 2635–2644, 2016.

Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, pages 187–196, 2018.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4541–4550, 2019.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Lantao Yu, Yang Song, Jiaming Song, and Stefano Ermon. Training deep energy-based models with f-divergence minimization. In *International Conference on Machine Learning*, pages 10957–10967. PMLR, 2020.

Chunkai Zhang and Yingyang Chen. Time series anomaly detection with variational autoencoders. *arXiv preprint arXiv:1907.01702*, 2019.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

Hang Zhou, Yasheng Sun, Wayne Wu, Chen Change Loy, Xiaogang Wang, and Ziwei Liu. Pose-controllable talking face generation by implicitly modularized audio-visual representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4176–4186, 2021.

Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJJLHbb0-`.