THE UNIVERSITY OF CHICAGO

DESIGNING DEEP GENERATIVE MODELS WITH SYMBIOTIC COMPOSITION

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

COMMITTEE ON COMPUTATIONAL AND APPLIED MATHEMATICS

BY
ZHISHENG XIAO

CHICAGO, ILLINOIS
JUNE 2022

To my parents for their unconditional love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I have received a great deal of support and assistance throughout my doctoral study. Foremost, I would like to express my sincere gratitude to my advisor Prof. Yali Amit, for his motivation and immense knowledge, which are invaluable in formulating the research questions and methodology. He opened up the world of deep generative models to me, which became my primary research interest throughout the years. We frequently met on paper reading and research discussions, and his insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Yuehaw Khoo and Prof. Greg Shakhnarovich, for their insightful comments on the thesis.

I would like to thank my internship mentors, Dr. Arash Vahdat and Dr. Karsten Kreis, who are research scientists at Nvidia. Although I had not published even a single paper when I applied for the internship, Arash interviewed me, discussed my research seriously, and offered me the opportunity, which turned into two productive summer internships at Nvidia. During the internships, Arash and Karsten helped me brainstorm research ideas and provide suggestions for writing papers. I appreciate their support.

I want to thank Prof. Mary Silber and Zellencia Harris for their outstanding support in administrative and general matters. There can be many uncertainties when launching a new degree program, but their efforts made my Ph.D. study perfectly smooth.

I would like to thank my primary collaborator Qing Yan, who is a Ph.D. student in the Statistics department. We worked on multiple successful research projects together, and we had many interesting research discussions. Outside research, we are also good friends.

Further, I would like to show my gratitude to my friends during the years, including Ruiyi Yang, Zhan Lin, Ziqi Liu, Pinhan Chen, Dongyue Xie, Wanrong Zhu, Dake Zhang, Yi Wang, Yi Liu, and many others. I spent five wonderful years at the University of Chicago, and this would not be possible without them.

I would like to thank my fiancée, Peijun Xiao. We have been together for ten years, but there is never a dull moment with her. We went through so many things and overcame so many challenges, and she brings so much joy to me. There were good times, there were hard times, but there were never bad times [1]. Finally, we will get married soon. I love her — always have, always will.

Last but not least, I would like to thank my parents for their unconditional love and support throughout my life. They support me financially and mentally to study in the U.S., and they always encourage me to pursue my goals. Words cannot express my gratitude to them.

[1] By Steve Jobs

ABSTRACT

Generative models, especially those parameterized by deep neural networks, are powerful unsupervised learning tools for understanding complex data without labels. Deep generative models have achieved tremendous success in recent years, with applications in various tasks, including sample generation, image editing, visual domain adaptation, data augmentation for discriminative models, and solving inverse problems.

Parallel endeavors have been proceeding along various directions – such as generative adversarial networks (GAN), variational autoencoders (VAE), normalizing flows, energy-based methods, autoregressive models, and diffusion models – and we are now able to generate increasingly photorealistic images using deep neural networks. Although these models have distinct formulations and properties, it is critical to have a clear view of fundamental deep generative models, understand their pros and cons as well as know the reasons behind them. With a good understanding of existing generative learning frameworks, we can design new models that can maintain the advantages while eliminating the limitations of previous models. Figure 1 is an illustration of the main theme of this dissertation: we give a panoramic view of the landscape through deep generative models and design new models based on the landscape.

Following this theme, the dissertation can be divided into two parts. In the first part (Chapter 1 and 2), we give a high-level overview of deep generative models and dive deep into several important models, introducing their formulations and analyzing their pros and cons carefully. Motivated by this analysis, in the second part (Chapter 3, 4, 5 and 6), we introduce four advances in the direction of designing new generative models by combining existing ones. For each new model we propose, we carefully present the formulation and explain the motivation behind the composition. We conduct extensive experiments to show that our proposed models can be seen as symbiotic compositions of two different generative models: the two components in each composition help each other get rid of the limitations

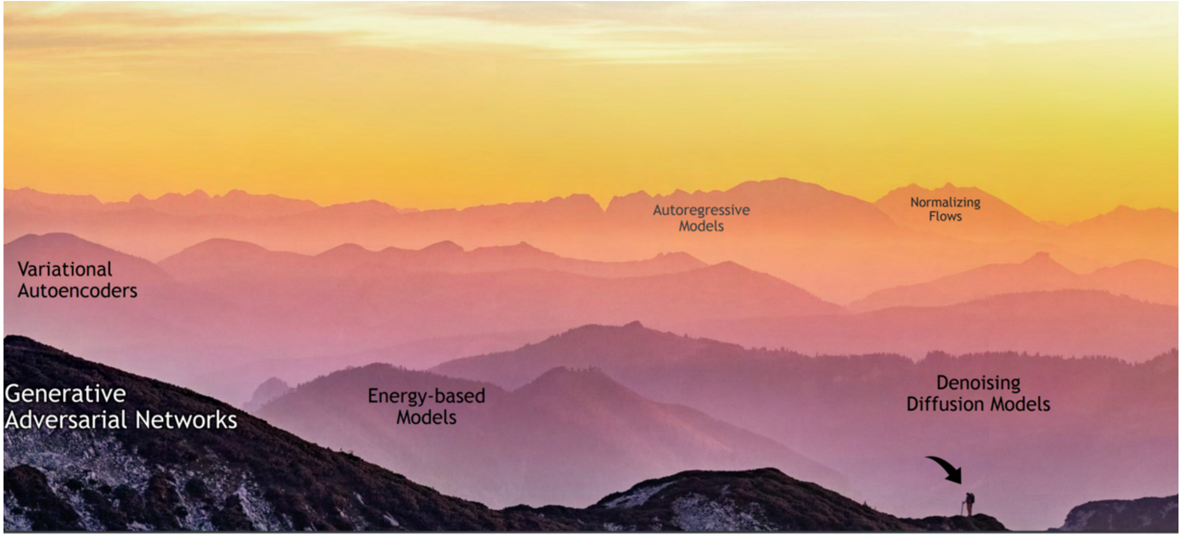


Figure 1: The landscape of deep generative modeling.

while keeping the advantages.

We hope that our findings may serve as a minor contribution to developing deep generative models.

CHAPTER 1

INTRODUCTION

1.1 What are Generative Models?

The objective of this dissertation is to improve deep generative models. As such, a high-level description of generative models is introduced. It begins with motivations for studying generative models, followed by a formal statement of generative modeling and a comparison with discriminative models.

The ability to imagine is one of the most distinctive and powerful aspects of human cognition. Humans are able to synthesize mental objects which are not constrained by what is presented in reality. There are many potential benefits of this capability. For example, it allows humans to do planning by imagining how their actions could affect the outcome, and by imagining an object, humans can learn about its properties without explicit supervision. The sub-field of machine learning, which enables machines with this same essential capacity to imagine and synthesize, is called generative modeling.

1.1.1 Generative Modeling: Motivations

There are many practical motivations for generative models. An obvious one is that they can be used to generate new entities. Generating new samples that maintain characteristics of given training samples is a valuable ability, and therefore generative models have been applied to different domains [Karras et al., 2019, Van Den Oord et al., 2016, Dhariwal et al., 2020, Devlin et al., 2018, Maziarka et al., 2020] as illustrated in Figure 1.1.

Besides the promising capability to generate new samples, there are other motivations for studying generative models. One common argument is that one can use generative models as a way of doing supervised and reinforcement learning with less labeled data. Children can learn things, such as a new language, with relatively rare explicit supervised feedback or

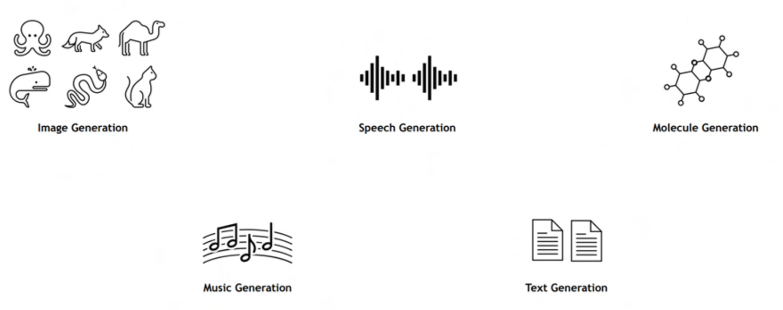


Figure 1.1: Generative models can generate samples in various domain.

reward signals. An appealing hypothesis is that humans use unsupervised generative models to build robust representations of the world and then use those same representations to do supervised and reinforcement learning from small amounts of explicitly labeled data. Since humans are constantly receiving perceptual data (sound, vision, touch), humans should have enormous amounts of unlabeled data which can be used to train generative models, and it is possible that learning generative models requires learning features that are also useful for supervised learning. Indeed, in machine learning, generative models play an essential role in facilitating supervised learning and reinforcement learning [Ha and Schmidhuber, 2018, Kingma et al., 2014, Donahue et al., 2016].

Yet another practical motivation is that generative models can provide a powerful tool to model arbitrary distributions. Directly training generative models on an unlabeled dataset corresponds to modeling the marginal distribution of data (more details in the next section), and the same idea can be extended to modeling other distributions. For example, generative models can be used to facilitate Monte Carlo sampling by learning a proposal distribution for importance sampling [Müller et al., 2019] or learning more efficient transition kernels for Markov Chain Monte Carlo [Song et al., 2017]. Generative models are also used to approximate the complicated posterior distribution of inverse problems [Song et al., 2022, Saharia et al., 2021b, Lugmayr et al., 2022] and simulation-based inference [Cranmer et al., 2020, Papamakarios, 2019]. Indeed, it has been shown that specifically designed generative models can learn arbitrary conditional distributions [Ivanov et al., 2019, Li et al., 2020]. Such flexibility gives generative models an extensive range of applications.

1.1.2 Generative Modeling: Formalizing the Problem

So far, we have discussed generative modeling in qualitative terms. It learns models which can simulate the dynamics of the world, and models that can synthesize realistic-looking data. However, before going further, it is useful to understand the probabilistic interpretation of generative models and provide a a formal framework for their study.

The core idea behind generative models is that observations from the world are samples from an underlying distribution $\mathbf{x} \sim p(\mathbf{x})$. For example, we can think of the distribution over all possible human faces (which can be infinitely many) to be $p(\mathbf{x})$, and each face is a sample. If we have access to a set of faces, we may also choose to treat them as a finite collection of samples from this distribution. Meanwhile, a generative model constructs a distribution $p_\theta(\mathbf{x})$, which is described by a set of parameters θ. Then the task of generative modeling can be framed as trying to ensure that $p_\theta(\mathbf{x})$ becomes as similar as possible to $p(\mathbf{x})$. Statistical divergences give a natural mathematical framework for measuring the similarity between distributions.

In statistics and information geometry, a divergence is a function $D(p\|q) : S \times S \to \mathbb{R}$ taking two distributions p and q over a space of distributions S as inputs, and returning a scalar value. A divergence satisfies the following properties:

- Non-negativity: $D(p\|q) \geq 0$ for all $p, q \in S$
- Identity of indiscernibles: $D(p\|q) = 0$ if and only if $p = q$

Notably, there is no symmetry assumption, meaning that $D(p\|q)$ does not necessarily equal to $D(q\|p)$. This property distinguish divergence from a metric. Formally, the probabilistic approach to generative modeling frames learning as an optimization problem where the loss corresponds to a given divergence:

$$\mathcal{L}(\theta) = \underset{\theta}{\operatorname{argmin}} \quad D\left(p(\mathbf{x}) \| p_{\theta}(\mathbf{x})\right). \tag{1.1}$$

Deep Generative Models: In previous paragraphs, generative modeling is formulated as learning a distribution that has a small divergence with the target distribution, from which the training samples are drawn. However, the target distribution may be extremely complicated (think about the ground truth distribution over all plausible human faces). Unlike standard statistical inference where a mathematical expression for the distribution is sought, typically, the goal of generative modeling on high dimensional space is to obtain a generating function $g_{\theta} : \mathbb{R}^m \rightarrow \mathbb{R}^n$ that maps samples from a tractable distribution $\mathcal{Z}$ (called noise space) supported in $\mathbb{R}^m$ to points in $\mathbb{R}^n$ with the same dimension as data. The distribution constructed by the generative model $p_{\theta}(\mathbf{x})$ relies on the generating function g_{θ}, and when g_{θ} is implemented by neural networks, the resulting generative model is a deep generative model (DGM). In deep generative modeling, the $p_{\theta}(\mathbf{x})$ is not necessarily a distribution with closed-form density, and it can be defined implicitly with the generator g_{θ}.

In deep generative modeling, there is plenty of flexibility in choosing the form of g_{θ}. For example, g_{θ} can either be an explicit mapping (in which case it is called a generator) or an implicit iterative process. In addition, the dimension m of the noise space $\mathcal{Z}$ can generally be different from the dimension of the data space n.

1.1.3 Generative Model vs. Discriminative Model

In traditional statistical learning, the generative model is a concept that usually comes together with the discriminative model. For completeness, a discussion on generative models vs. discriminative models is included, which sheds additional light on explaining generative modeling. In statistical classification, two main approaches are called the generative approach and the discriminative approach. Following Ng and Jordan [2001], Jebara [2012]:

- A generative model is a statistical model of the joint probability distribution $p(\mathbf{x}, \mathbf{y})$ on given observable variable $\mathbf{x}$ and target variable $\mathbf{y}$

- A discriminative model is a model of the conditional probability $p(\mathbf{y}|\mathbf{x})$ of the target $\mathbf{y}$, given an observation $\mathbf{x}$

Note that since the target variable $\mathbf{y}$ is unobserved (except in training), typically it is called a latent variable. In a classification task, the target variable $\mathbf{y}$ is the label associated with the data sample $\mathbf{x}$, but in general, the meaning of latent variables can be more complicated. In the setting of learning classifiers, since the prior distribution $p(\mathbf{y})$ over target variables is relatively simple, the learning of a generative model $p(\mathbf{x}, \mathbf{y})$ is typically done by learning conditional distribution $p(\mathbf{x}|\mathbf{y})$. After that, the marginal distribution $p(\mathbf{x})$ over data can be obtained by marginalizing out $\mathbf{y}$:

$$p(\mathbf{x}) = \int_y p(\mathbf{x}|\mathbf{y})p(\mathbf{y})d\mathbf{y},$$

and the final classifier $p(\mathbf{y}|\mathbf{x})$ can be obtained through Bayes' rule:

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{y})p(\mathbf{y})}{p(\mathbf{x})}.$$

Therefore, generative models provide an alternative approach to inferring the target variable. The generative approach has important advantages over the discriminative approach. One noticeable advantage is that the generative approach is more robust, as it has uncertainty estimates when making the decision. For example, consider a deep neural network that is trained well to classify images into three categories ($\mathbf{y} \in \{\text{cat, dog, horse}\}$). However, as pointed out by Szegedy et al. [2013], adding noise to images could result in a completely

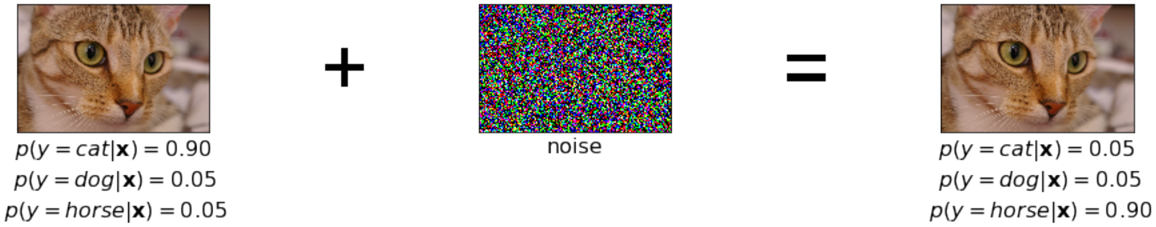

$p(y = cat|\mathbf{x}) = 0.90$
$p(y = dog|\mathbf{x}) = 0.05$
$p(y = horse|\mathbf{x}) = 0.05$
+
noise
=
$p(y = cat|\mathbf{x}) = 0.05$
$p(y = dog|\mathbf{x}) = 0.05$
$p(y = horse|\mathbf{x}) = 0.90$

Figure 1.2: An example of adversarial attack: adding noise to an almost perfectly classified image that results in a shift of predicted label.

false classification. An example of such a situation is presented in Figure 1.2 where adding noise could shift predicted probabilities of labels, however, the visual semantics of the image is barely changed.

This example indicates that discriminative neural networks may lack semantic understanding of images. A discriminative model assign classes merely based on decision boundaries, and therefore as long as $\mathbf{x}$ lies far away from the boundary, the decision is confident. In contrast, generative models can assess uncertainty by incorporating the data probability $p(\mathbf{x})$. For example, assuming there is a well-trained generative model, the marginal likelihood $p(\mathbf{x})$ will be low after adding noise to the image, and hence the joint density $p(\mathbf{x}, \mathbf{y})$, which can be factorized as $p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$ should be low as well, and thus, the decision is uncertain. Therefore, generative models are essential for building reliable models that not only learn how to make decisions but also quantify their beliefs using the language of probability.

1.2 Important Concepts

In the following, some important concepts about generative models will be reviewed, as they will be referred to frequently in the remaining of the dissertation. The review begins by introducing two fundamental approaches for training generative models: the maximum likelihood approach and the density ratio approach. Later, Langevin dynamics, an important mathematical tool for sampling from an unnormalized distribution, will be discussed.

1.2.1 Maximum Likelihood Approach for Training Generative Models

In the previous section, training of generative model was defined as finding a parameterized distribution $p_\theta(\mathbf{x})$ that minimizes a divergence between itself and target data distribution $p(\mathbf{x})$. One important component of training is to select an appropriate statistical divergence for optimization. However, even before selecting the particular divergence, a question to consider is what types of divergence we are able to optimize? Note that in general, the form of the target distribution is unknown, and therefore the density of $p(\mathbf{x})$ cannot be computed. Typically, only a finite set of samples $\mathbf{x} \sim p(\mathbf{x})$ is accessible. Meanwhile, $p_\theta(\mathbf{x})$ is a model that we construct, so it is reasonable to assume that the density can be computed and samples can be drawn from it.

The Kullback–Leibler (KL)-divergence is one candidate that satisfies the requirements. It can be rewritten as an expression in which the only term that depends on the parameters is an expectation over $p(\mathbf{x})$. In other words, only samples from $p(\mathbf{x})$ are needed for optimizing the KL-divergence. Specifically, the KL-divergence is expressed as

$$\begin{aligned} D_{\mathrm{KL}}(p(\mathbf{x}) \| q(\mathbf{x})) &= \int p(\mathbf{x}) \log p(\mathbf{x}) d\mathbf{x} - \int p(\mathbf{x}) \log q(\mathbf{x}) d\mathbf{x} && (1.2) \\ &= \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} && (1.3) \\ &= \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} \left[\log \frac{p(\mathbf{x})}{q(\mathbf{x})} \right] && (1.4) \\ &= \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} [\log p(\mathbf{x}) - \log q(\mathbf{x})]. && (1.5) \end{aligned}$$

In information theory, the first term $\mathbb{E}_{\mathbf{x} \sim p} \log p(\mathbf{x})$ is the negative entropy of $p(\mathbf{x})$:

$$\mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} \log p(\mathbf{x}) = -H(p(\mathbf{x})),$$

and the second term $-\mathbb{E}_{\mathbf{x}\sim p} \log q(\mathbf{x})$ is the cross entropy between p and q:

$$-\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \log q(\mathbf{x}) = CE(p(\mathbf{x}), q(\mathbf{x})).$$

Note that when $q(\mathbf{x})$ is the generative model $p_\theta(\mathbf{x})$ and $p(\mathbf{x})$ is the true data distribution, the KL-divergence can be decomposed into a cross-entropy term and the entropy of data distribution. Since the entropy term does not depend on θ, the KL-divergence can be minimized by minimizing the cross-entropy $-\mathbb{E}_{\mathbf{x}\sim p} \log p_\theta(\mathbf{x})$. Furthermore, note that minimizing the cross-entropy corresponds to maximizing the log-likelihood of the data $\mathbf{x} \sim p(\mathbf{x})$ under p_θ. Therefore, the generative model can be trained by maximizing the likelihood of the training samples. As a by-product, the entropy of the true data distribution can be estimated by a generative model if it maximizes likelihood.

Formally, given a set of i.i.d. training samples $\{\mathbf{x}_i, i = 1, \dots N\}$ from $p(\mathbf{x})$, the ideal parameter θ satisfies

$$\begin{aligned}
\theta* &= \arg\max_\theta \prod_{i=1}^{N} p_\theta\left(\mathbf{x}_i\right) && (1.6)\\
&= \arg\max_\theta \sum_{i=1}^{N} \log p_\theta\left(\mathbf{x}_i\right) && (1.7)\\
&= \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} - \log p_\theta\left(\mathbf{x}_i\right) && (1.8)\\
&\approx \arg\min_\theta \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \left[- \log p_\theta(\mathbf{x})\right] && (1.9)\\
&= \arg\min_\theta D_{\mathrm{KL}}(p(\mathbf{x}) \| p_\theta(\mathbf{x})). && (1.10)
\end{aligned}$$

Therefore, training generative models by minimizing KL-divergence is equivalent to maximizing the likelihood of the training samples. Such a training approach is called maximum likelihood training.

Maximum likelihood training only requires sampling uniformly from the real data and evaluating the log-density of the model $p_\theta(\mathbf{x})$. Some simple properties have to be satisfied by p_θ in order to make a valid distribution. $p_\theta(\mathbf{x})$ needs to be non-negative everywhere and integrate to 1 over the region where its value is defined (called the support of the distribution):

$$\begin{aligned} p_\theta(\mathbf{x}) &\geq 0 \\ \int_{\mathbf{x}\in\mathbb{R}} p_\theta(\mathbf{x}) &= 1. \end{aligned} \tag{1.11}$$

Any parameterized family of functions that satisfies these properties can be used to define $p_\theta(\mathbf{x})$ in maximum likelihood training. The most fundamental one is the Gaussian distribution, which has a density parameterized by $\theta = \{\mu, \sigma^2\}$:

$$p_\theta(\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma}} \exp \frac{-(\mathbf{x}-\mu)^2}{2\sigma^2}.$$

The parameter μ and σ^2 obtained from maximum likelihood training is simply the empirical mean and empirical variance of the training data.

A major limitation of most closed-form densities, is that they are uni-modal, which makes them ill-suited to problems where very distinct points can have high density with regions of low density separating them. One straightforward way to get around these limitations is to replace any density with a mixture over densities with distinct parameters. For example, a Gaussian mixture model with C components has the form

$$p_\theta(\mathbf{x}) = \sum_{k=1}^{C} \pi_k p_{\theta_k}(\mathbf{x}),$$

where each component $p_{\theta_k}(\mathbf{x})$ is a Gaussian distribution and π_k is the corresponding weight. This form is guaranteed to be normalized with the only condition being that the π_k sum to 1. However, the maximum likelihood training of mixture models is more challenging, as it becomes difficult to explicitly find the solution of parameters that maximize likelihood. To get around this, the EM algorithm [Dempster et al., 1977], which is a special case of optimizing the variational bound of the likelihood, needs to be applied.

Besides training simple models such as Gaussian and Gaussian mixture, maximum likelihood can be used to train a variety of advanced deep generative models. More details will be presented in Chapter 2.

1.2.2 Density Ratio Approach for Training Generative Models

An alternative to the maximum likelihood approach involves studying the differences between samples from a generative model model and real samples. In practice this usually takes the form of estimating the density ratio $\frac{p(\mathbf{x})}{p_\theta(\mathbf{x})}$ between the real data distribution $p(\mathbf{x})$ and the model distribution $p_\theta(\mathbf{x})$. The density ratio can be described in terms of the following quantity

$$D_\theta(\mathbf{x}) = \frac{p(\mathbf{x})}{p_\theta(\mathbf{x}) + p(\mathbf{x})}, \tag{1.12}$$

since $\frac{p(\mathbf{x})}{p_\theta(\mathbf{x})} = \frac{D_\theta(\mathbf{x})}{1-D_\theta(\mathbf{x})}$. It can be shown that learning $D_\theta(\mathbf{x})$ is equivalent to training a binary classifier that discriminates between the real data and the model's samples [Sugiyama et al., 2012]. Specifically, a binary classifier $D : \mathbb{R}^n \to (0, 1)$ minimizes the binary cross-entropy loss

$$\min_D -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\log D(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}[\log(1 - D(\mathbf{x}))],$$

which corresponds to training the Bayes classifier under the assumption of equal prior probability on two classes. Methods for successfully training classifiers have been widely studied, and inductive biases that are known to be good for classification could also be good for determining the quality of generations. Another motivation for modeling the ratio between a model and the data distribution is that it allows the model to become sensitive to any clear difference between real samples and generated samples, which may be a much easier task than simply determining the density of a distribution at a given point.

Next, two algorithms for training generative models with the density ratio idea are discussed.

Noise Contrastive Estimation: Gutmann and Hyvärinen [2012] proposed to estimate the density ratio between the data distribution and a fixed model $q(\mathbf{x})$ by the classifier trick introduced above. Typically the fixed model is chosen to be a simple one such as Gaussian noise distribution, so the method is called Noise Contrastive Estimation (NCE). Once the quantity $D_\theta(\mathbf{x})$ is learned, the density ratio serves as a re-weighting factor of the noise distribution. Specifically, assuming $D_\theta(\mathbf{x})$ accurately captures the quantity in Equation 1.12, then

$$\hat{p}(\mathbf{x}) = \frac{D_\theta(\mathbf{x})}{1 - D_\theta(\mathbf{x})} q(\mathbf{x})$$

should be a distribution that is close to the data distribution $p(\mathbf{x})$. Therefore, samples generated by $\hat{p}(\mathbf{x})$ can be approximately seen as samples from $p(\mathbf{x})$, and hence a successful generative model is defined. Sampling from $\hat{p}(\mathbf{x})$ can be done by Sampling-Importance-Resampling (SIR) or Markov chain Monte Carlo methods.

A significant limitation of NCE is that the noise distribution $q(\mathbf{x})$ is required to be close enough to the real data distribution. If $q(\mathbf{x})$ has very small values where $p(\mathbf{x})$ has large values, $D_\theta(\mathbf{x})$ will be close to 1 which leads to very large importance weights and high variance sampling. Intuitively, when the classification between $q(\mathbf{x})$ and $p(\mathbf{x})$ is too easy, not much information about the density ratio can be learned. In a high dimensional space like image space, it would be extremely difficult to find a noise distribution that is close to the data distribution, so the application of NCE is limited [Rhodes et al., 2020].

Generative Adversarial Networks: Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] aim to leverage the strengths of using a classifier for generation, while avoiding the major weaknesses of NCE. Unlike NCE whose model $q(\mathbf{x})$ is a fixed distribution, GANs update the generative model together with the classifier in an alternating fashion. The GAN framework approaches the generative modeling problem from a game theory perspective, as it trains two networks in an adversarial fashion.

To describe the core idea of GANs in the context of NCE, the noise distribution $q(\mathbf{x})$ is replaced by a generator network G_θ that is trained to produce samples which are similar to the training examples by transforming latent variables $\mathbf{z}$ from a fixed noise distribution, and the classifier $D(\mathbf{x})$ plays a similar role as in NCE that is trained to classify between examples from the training set and examples produced by the generator. The generator is optimized to maximize the probability that the discriminator will classify the generated example as "real". This setup is described as adversarial because the loss for the generator is the opposite of a term in the discriminator loss. Note that in GANs, the noise distribution is defined implicitly through the generator network G_θ. In other words, the noise distribution does not have a parameterized density.

For the usual cross-entropy classification objective in NCE, denoting the classifier to be D_ϕ with parameter ϕ, and assuming the latent variable $\mathbf{z}$ comes from a fixed noise distribution $p_0(\mathbf{z})$, the training of GANs can be written as

$$\min_\theta \max_\phi \quad V(G_\theta, D_\phi),$$

where

$$V(G, D) := \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_0(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]. \tag{1.13}$$

The generator cannot directly affect the $\log D(\mathbf{x})$ term, so for the generator, minimizing the loss is equivalent to minimizing $\log(1 - D(G(\mathbf{z})))$.

A practical observation from Goodfellow et al. [2014] is that directly optimizing the generator network to minimize the loss leads to poor performance due to saturated gradients. In practice, a non-saturating objective is often used, where the generator is updated by

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})}[\log D(G_{\theta}(\mathbf{z}))]. \tag{1.14}$$

The above paragraphs provide a high-level overview of the core idea behind GANs. More details about GANs will be discussed in Chapter 2.5.

1.2.3 Langevin Dynamics

The analytically intractable expectations involved in the learning of some generative models may be approximated by various forms of Markov chain Monte Carlo (MCMC) sampling. A simple realization of MCMC is in the form of the Langevin dynamics.

The story of Langevin dynamics began in 1827, when Robert Brown, an English botanist was looking at pollen grains in water and observed them moving around randomly. Many years later, Albert Einstein wrote a paper explaining the pollen's motion which is caused by random impacts of the water molecules on the pollen grain. Such a motion is called Brownian Motion [Einstein, 1905]. Einstein's explanation was later conceptualized by Adriaan Fokker and Max Planck and resulted in the Fokker-Planck equation. Meanwhile, Paul Langevin, a French physicist wrote a different formulation of Brownian motion (see English translation [Lemons and Gythiel, 1997]), which resulted in the Langevin equation. Langevin dynamics was popularized in the statistics literature in the early '90s in [Amit et al., 1991], and was introduced in the deep learning literature in [Welling and Teh, 2011].

Consider the diffusion X_t defined by the (overdamped) Langevin equation

$$dX_t = \nabla f\left(X_t\right) dt + \sqrt{2T} dw_t, \tag{1.15}$$

where f is a given energy function, T is the chosen temperature parameter, and ω_t is a Wiener process with mean 0 and variance dt. Under some mild conditions, particles simulated by Equation 1.15 are samples from the Boltzmann distribution with density

$$\pi(x) \propto \exp(f(x)/T), \tag{1.16}$$

which is the stationary distribution of the continuous time Markov Chain in Equation 1.15.

The simulation of the Langevin equation by digital computers requires discretization. Equation 1.15 can be discretized as

$$X_{t+\Delta t} = X_t + \nabla f\left(X_t\right) \Delta t + \sqrt{2T\Delta t} \epsilon_t, \tag{1.17}$$

where $\epsilon_t \sim \mathcal{N}(0, I)$. The discretized Langevin dynamics in Equation 1.17 can be used to sample from distributions $p(\mathbf{x})$ by iterating with infinitesimally small Δt and infinite many steps, as long as $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ is known. This means that the density does not need to be normalized, as the normalizing constant is independent of $\mathbf{x}$ so its gradient w.r.t $\mathbf{x}$ is 0.

Sampling with Langevin dynamics has a gradient ascent interpretation. Given an initial point $\mathbf{x}_0$, the point of maximum probability can be reached by running gradient ascent following the gradient of the log-density:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_t \nabla_{\mathbf{x}} \log p\left(\mathbf{x}_k\right),$$

where α_t is the step size. By further adding an appropriate amount of noise at each step:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_t \nabla_{\mathbf{x}} \log p\left(\mathbf{x}_k\right) + \sqrt{2\alpha}\epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I), \tag{1.18}$$

a sample from the distribution is generated. The intuition is that by following the gradient, the point reaches high probability regions, but the noise ensures it would not just reach the maximum. Strictly speaking, the convergence of Langevin relies on a Metropolis-Hastings accept/reject step, which depends on the true probability distribution. However, for a sufficiently small step size this is not necessary in practice, because as the step size goes to zero, the probability of acceptance of goes to 1 [Neal, 1993].

1.2.4 Evaluation of Generative Models

It is straightforward to evaluate regressors and classifiers, as typically there is a test dataset on which the same task for training can be used to evaluate the models' performance. However, assessing generative models can be much more difficult. In this section, several common methods for evaluating generative models are introduced, and they will be used to evaluate the models discussed in later chapters.

Test data likelihood: Under the maximum likelihood training approach, a straightforward way of quantifying the performance of the model is to compute the model's average likelihood $p_\theta(\mathbf{x})$ on the test dataset. Test data likelihood can be useful for detecting overfitting and it is easy to compute. However, it has several limitations. For generative models not trained by maximum likelihood, the likelihood may not be easy to compute. More importantly, the implication of test data likelihood is not clear: there's no clue suggesting that higher likelihood on test data leads to better sample quality or better learned representations. Therefore, although test data likelihood is widely used for assessing models trained by maximum likelihood, alternative criteria that are agnostic to the actual form of the generative model and can directly judge the sample quality are needed.

Inception Score: Inception Score (IS) [Salimans et al., 2016] uses a fixed pre-trained classifier as the basis for the scoring metric. It is defined by

$$\exp\left(\mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})} D_{\mathrm{KL}}(p(\mathbf{y}|\mathbf{x})\|p(\mathbf{y}))\right), \tag{1.19}$$

where $\mathbf{x}$ is a sample from the generative model, $p(\mathbf{y}|\mathbf{x})$ is the categorical distribution for labels $\mathbf{y}$ conditional on $\mathbf{x}$ given by a pre-trained classifier, and $p(\mathbf{y})$ is the marginal distribution of the labels in the generated samples according to the classifier. Higher scores imply better generation quality. The intuition behind IS is that a generative model should produce diverse samples from different classes while ensuring that each sample is clearly identifiable as belonging to a single class. For example, if a model can only generate samples from a single class, the IS will be low because $p(\mathbf{y}|\mathbf{x})$ and $p(\mathbf{y})$ are similar (both will concentrate on the single class). Likewise producing blurry samples which do not allow the classifier to make a clear decision will make $p(\mathbf{y}|\mathbf{x})$ uncertain and more similar to $p(\mathbf{y})$.

While IS has been shown to be correlated well to visual quality, there are several limitations. One is that IS could be fooled by a model which produces only a single and clearly identifiable example of each class that the classifier is aware of. This would make $p(\mathbf{y}|\mathbf{x})$ different from $p(\mathbf{y})$ while maintaining high entropy in $p(\mathbf{y})$. Another issue is that it is unclear how accurate the metric is on datasets other than ImageNet, which is the dataset on which the classifier is pre-trained Barratt and Sharma [2018]. In addition, IS cannot be used to judge the sample quality on datasets where there's no clear label, such as the CelebA dataset.

Frechet Inception Distance: Similar to Inception Score, Frechet Inception Distance (FID) [Heusel et al., 2017] assesses generated samples using a pre-trained classifier. However, instead of relying on the classification head of the classifier, FID only uses the intermediate features extracted by the classifier, making it generalize well to different datasets. The key idea is that the score is high when the distribution of the extracted features for generated samples is close to the distribution of features for real data points. It assumes that these features follow a multivariate Gaussian distribution (with a full covariance matrix). Because the features are from the end of a deep classifier, this multivariate Gaussian assumption is much more justified than it would be in the pixel space. To compute the FID score, a Gaussian distribution $\mathcal{N}(\mu_g, \Sigma_g)$ is fitted to features extracted from generated samples, and another Gaussian $\mathcal{N}(\mu_r, \Sigma_r)$ is fitted to features extracted from real samples. From this, the Frechet Distance between these two Gaussians is computed:

$$\left\|\mu_r - \mu_g\right\|_2^2 + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r \Sigma_g\right)^{1/2}\right). \tag{1.20}$$

A smaller FID score implies the set of generated samples is close to the training set, suggesting better sample quality. FID has several advantages over IS. FID can be evaluated on the test data, so it can directly test against overfitting, unlike IS. Moreover, generating a single high-quality example for each class will lead to low FID by giving the features of the generated samples an unnatural distribution. FID is currently the most widely used metric for sample quality, and finding better evaluation criteria for generative models is an ongoing research direction.

1.3 High-level Overview of Proposed Methods

This dissertation includes a sequence of methods for designing deep generative models. In this section, a high-level overview of each proposed method will be given to guide readers through the major content of the dissertation.

1.3.1 Flexible Prior of Auto-encoder Models

Many deep generative models are developed based on the idea of auto-encoding. The idea is that an auto-encoder extracts low dimensional representations on data by enforcing the reconstruction while passing through a bottleneck structure. The resulting representation at the bottleneck can be treated as a latent variable that contains information that can be decoded into a sample. By encouraging the distribution of latent variables to be close to a parametric distribution (which is called the prior distribution), it is possible to generate unseen samples by sampling latent variables from the prior and passing them through the decoder. Earlier methods let the prior be a simple noise distribution such as $\mathcal{N}(0, I)$, however, it is difficult to learn useful representations while enforcing the latent variables to be uninformative. As a result, after training, the gap between the distribution of the latent variables and the prior is large. This leads to poor sample quality when generating samples by drawing from the prior.

One method to overcome the issue is introducing a more flexible prior distribution that can better match the latent distribution. We developed an auto-encoder based generative model called Generative Latent Flow (GLF), which models the prior by normalizing flows. In contrast to some other auto-encoder based generative models, which use various regularizers that encourage the encoded latent distribution to match the prior distribution, our prior normalizing flow explicitly constructs a mapping between these two distributions, leading to better density matching while avoiding over regularizing the latent variables. The model is compared with several related techniques using flexible prior distributions and we show that it has many relative advantages including fast convergence, single-stage training, and minimal reconstruction trade-off. We also study the relationship between the model and its stochastic counterpart and show that our model can be viewed as a vanishing noise limit of VAEs with flow prior.

1.3.2 Exponential Tilting of Generative Models

Generator models such as variational auto-encoders (VAEs) and normalizing flows can generate samples quickly and are equipped with a latent space that enables fast traversal of the data manifold. However, they tend to assign high probability density to regions in data space outside the actual data distribution and often fail at generating sharp images. One possible reason is that maximum likelihood training, which minimizes the forward KL-divergence between the model distribution and the data distribution, will enforce the model density to spread out and cover all the modes of the data distribution. However, since the capacity of the generative model is limited by the constrained structure and network size, spreading over the support of data distribution will result in an unwanted mismatch between the two distributions.

Energy-based models (EBMs) have recently been shown to suffer less from the issue mentioned above, as the maximum likelihood training of EBMs involves explicitly reducing the density of non-data regions. However, training EBMs with maximum likelihood requires sampling from the model, which further requires expensive Markov chain Monte Carlo (MCMC) iterations that mix slowly in high dimensional pixel space. Therefore, the training of EBMs is difficult.

To address the issues of both generator models and EBMs, we propose to design new generative models based on exponential tilting. Specifically, we design generative models whose distribution is composed of the normalized density of a generator model multiplied by the unnormalized density of an EBM. The resulting model can be seen as an exponential tilting of the generator model. Such a formulation has several benefits. The generator model, which is relatively easy to train, captures the overall mode structure of the data distribution and, it relies on its exponential tilting component to explicitly exclude non-data-like regions from the model and hence refine the samples. Moreover, the uninformative latent space of the generator model will provide a smooth geometry for the joint density and speeds up MCMC updates by reparameterization. Based on the properties of resulting generative models, we also propose a novel two-stage training strategy where the generator model is trained first, and the EBM is trained later with the generator fixed. Such a strategy significantly simplifies the training, as each update of the EBM is very expensive, and with a pre-trained generator that roughly captures the target distribution, only a small number of training steps is needed for the EBM.

1.3.3 EBMs with short-run Langevin Dynamics as Generator Models

Among a variety of methods that can be used to train EBMs, maximum likelihood training is the most popular one. However, applying maximum likelihood training to EBMs is less straightforward. Although an EBM has a parameterized explicit density function, the unknown normalizing constant is intractable to estimate. Mathematically, the gradient of the log-likelihood of an EBM can be written in a form that involves an expectation over the model distribution. In other words, Estimating the gradient for maximum likelihood training requires sampling from the EBM. As discussed in Chapter 1.2.3, sampling from an unnormalized EBM requires running the Langevin dynamics. In theory, the Langevin dynamics requires infinite many steps and diminishing step size to ensure convergence, which is certainly not feasible. In practice, the Langevin dynamics are replaced by short-run Langevin dynamics, which only iterate for a finite number of steps and constant step size.

We provide an alternative understanding of training EBMs with short-run Langevin dynamics. We observe that short-run Langevin dynamics behave more like generators that transform the initial noise into a sample. We further try to understand the training procedure by replacing short-run Langevin dynamics with deterministic solutions of the associated gradient descent ODE. Doing so allows us to study the density induced by the dynamics, as now the transformation is noise-free and invertible. In addition, we connect with GANs by treating the dynamics as generator models, 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. With such a connection, we treat EBMs as a special case of Wasserstein GANs [Arjovsky et al., 2017], where the generator is replaced by a deterministic iterative transformation. We also explore the possibility of training the implicit generator with the W-GAN loss by back-propagating through the iterative sampling process.

1.3.4 Denoising Diffusion GANs: Expressive Denoising Distribution in Diffusion Models

A forward discrete-time diffusion process gradually perturbs data samples into white noise step by step by adding a small amount of noise at every step. The resulting process is a Markov process with Gaussian transition kernels. A denoising diffusion model is a generative model that tries to recover the forward diffusion process by iteratively denoising white noises into clean samples. Ideally, the model should be trained to match the ground truth per-step denoising distribution. However, such a distribution is intractable, and instead, the model is trained to match the per-step posterior distribution conditioned on the initial point. Since the posterior distribution is a Gaussian, the denoising model is also parametrized as a Gaussian distribution.

One major drawback of denoising diffusion models is that it requires a large number of denoising steps to generate samples, which makes them difficult to apply in many real-world applications. We investigate the slow sampling issue of denoising diffusion models and argue that it is fundamentally attributed to the Gaussian assumption in the denoising step mentioned above. The Gaussian denoising step assumption is justified only for small step sizes. To enable denoising with large steps, and hence, to reduce the total number of denoising steps, we propose to model the denoising distribution using a complex multi-modal distribution. We introduce denoising diffusion generative adversarial networks (denoising diffusion GANs) that model each denoising step using a multi-modal conditional GAN. Similar to denoising diffusion models, the GAN models the denoising distribution by generating a sample of single-step denoising conditioned on the noisy sample, and the generation is judged by the discriminator. With such a flexible denoising distribution, the number of denoising steps can be reduced to as low as 4, which is a 1000× speed-up compared to traditional denoising diffusion models.

1.4 Organization

The manuscript follows the journey of conducted research toward designing generative models with the core idea of symbiotic composition. Specifically, different generative learning frameworks are combined together to make a stronger generative model. The remainder of the dissertation is organized as follows.

To better understand the motivation behind the composition approach, in Chapter 2, a comparative review of existing deep generative models will be provided. For each type of generative model that is reviewed, its mathematical formulation, history of development, and recent advances will be discussed. Towards the end of Chapter 2, the pros and cons of different generative models will be compared, which leads to the motivation of our compositional approach.

In Chapter 3, 4, 5, and 6, research projects outlined in Chapter 1.3 will be presented in detail. Each chapter will begin with the motivation of the proposed approach, followed by the model specification and derivation. Related work will be discussed to better position our method with contemporary work, and extensive experimental results will be presented to show the effectiveness.

In Chapter 7, a conclusion of the dissertation, as well as a discussion on future work will be provided.

CHAPTER 2

A COMPARATIVE REVIEW OF DEEP GENERATIVE MODELS

This dissertation will introduce several novel deep generative models by combining existing generative learning frameworks. To better understand the motivation and formulation of each proposed model, it is important to review existing deep generative models. In this chapter, several popular deep generative models will be introduced in detail. Towards the end of this chapter, those models will be compared against each other, highlighting the motivation for designing a compositional approach. We roughly categorize deep generative models into two types: explicit models and implicit models. Explicit models refer to those who have an explicit parameterized density, and the training is often done by maximizing the likelihood as introduced in Section 1.2.1. In contrast, implicit models do not have an explicit form of the density, and hence the distribution is defined implicitly through the sample generation process. As a result, alternative training approaches, such as the density ratio trick, introduced in 1.2.2, are needed to train implicit models.

A list of deep generative models that will be covered in this chapter is

- Explicit models
    1. Variational Auto-encoders (in Section 2.1)
    2. Normalizing Flows (in Section 2.2)
    3. Energy-based Models (in Section 2.3)
    4. Denoising Diffusion Models (in Section 2.4)
- Implicit models
    1. Generative Adversarial Networks (in Section 2.5)

2.1 Variational Auto-encoders

The framework of Variational Auto-encoders (VAEs) [Kingma and Welling, 2013, Rezende et al., 2014] provides a principled method for jointly learning deep latent-variable generative models and inference models using gradient-based optimization. A VAE can be viewed as a combination of two coupled models: an encoder or recognition model, and a decoder or generative model. VAEs can be understood from different points of view. From the perspective of deep learning, VAEs can be seen as a generative version of plain Auto-encoders (AEs) whose goal is to reconstruct data. From the perspective of statistics, VAEs are a tool to perform efficient posterior inference on complicated latent variable models. In order to give a comprehensive introduction to VAEs, we first provide a brief review of preliminary knowledge on deep latent variable models, followed by the formulation of VAEs. Later, some limitations of VAEs will be discussed, and hierarchical VAEs, an important extension of VAE models, will be introduced.

2.1.1 Preliminaries

Assuming the observed variable $\mathbf{x}$ is a random sample from an unknown underlying process whose true distribution $p(\mathbf{x})$ is unknown. We attempt to approximate this underlying process with $p_\theta(\mathbf{x})$. Learning corresponds to searching for a value of the parameters θ such that the distribution given by the model $p_\theta(\mathbf{x})$ approximates the true distribution of the data $p(\mathbf{x})$ over a collected dataset $\mathcal{D} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\right\}$. Unlike observable variables $\mathbf{x}$, latent variables are variables that are part of the model, but which we cannot observe, and are therefore not part of the dataset. Typically, latent variables are denoted by $\mathbf{z}$. When we take latent variables $\mathbf{z}$ into consideration, the model represents a joint distribution $p_\theta(\mathbf{x}, \mathbf{z})$ over both the observed variables $\mathbf{x}$ and the latent variables $\mathbf{z}$. The marginal distribution over the observed variables $\mathbf{x}$ can be obtained by marginalizing out $\mathbf{z}$:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}. \tag{2.1}$$

As a simple example, if $\mathbf{z}$ is a discrete categorical random variable, and the conditional distribution $p_\theta(\mathbf{x}|\mathbf{z})$ is a Gaussian, then $p_\theta(\mathbf{x})$ is a Gaussian mixture distribution. For continuous $\mathbf{z}$, $p_\theta(\mathbf{x})$ can be seen as an infinite mixture.

The term Deep Latent Variable Models (DLVM) is used to describe a latent variable model $p_\theta(\mathbf{x})$ that is parameterized by neural networks. One advantage of DLVMs is their flexibility: even if each component in the joint distribution (such as the prior or conditional distribution) has a simple form, the resulting marginal distribution $p_\theta(\mathbf{x})$ can be very complex. The flexibility makes DLVMs attractive for modeling complicated underlying distributions $p(\mathbf{x})$. One simple and common way to factorize a DLVM is given by the following structure:

$$p_\theta(\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}), \tag{2.2}$$

where $p_\theta(\mathbf{x}|\mathbf{z})$ is the conditional distribution and $p_\theta(\mathbf{z})$ is called the prior distribution over $\mathbf{z}$. Since the prior distribution typically does not have trainable parameters, usually it is denoted as $p(\mathbf{z})$.

Since we have an explicit density of DLVMs given in Equation 2.1, it is tempting to train DLVMs by maximum likelihood. However, one major challenge of maximum likelihood learning in DLVMs is that the marginal probability of the data is typically intractable. In high dimensional space, the integral in Equation 2.1 usually does not have an analytic solution or efficient estimator. Due to this intractability, we cannot differentiate it w.r.t. its parameters and optimize it. The intractability also implies an intractable posterior distribution, which is written as

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})}. \tag{2.3}$$

The expression $p_\theta(\mathbf{x}, \mathbf{z})$ is easy to compute (it is assumed to be factorized into a simple form), but the intractable $p_\theta(\mathbf{x})$ makes the posterior also intractable. This makes posterior inference, which aims to infer the latent variable $\mathbf{z}$ given observation $\mathbf{x}$, difficult. Traditional inference methods can be relatively expensive, as they often require a per-datapoint optimization loop, or yield bad posterior approximations.

2.1.2 Formulation of VAEs

In the previous chapter, we introduced deep latent-variable models (DLVMs), and the intractability problem of estimating the marginal and posterior distributions. The framework of Variational Auto-encoders (VAEs) provides a computationally efficient way to overcome the issue of intractability and optimize DLVMs. The core idea that turns the DLVM's intractable posterior inference and learning problems into tractable problems is introducing a parametric inference model $q_\phi(\mathbf{z}|\mathbf{x})$. The inference model is called an encoder. The goal of the inference model is to be as close to the true posterior as possible:

$$q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x}). \tag{2.4}$$

The inference model is parameterized by a neural network with parameter ϕ, and the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is chosen to be a tractable one. One important point is that we use a single encoder neural network to perform posterior inference over all of the samples in the dataset. This is the major difference when compared to more traditional variational inference methods where the variational parameters are not shared, but instead separately optimized for each sample. Such a strategy for sharing parameters for posterior inference is called amortized inference [Gershman and Goodman, 2014].

The introduction of a parameterized inference model $q_\phi(\mathbf{z}|\mathbf{x})$ allows us to obtain a tractable lower bound on the marginal data log-likelihood $\log p_\theta(\mathbf{x})$, and hence we can optimize such a lower bound as a proxy for optimizing the data likelihood. The lower bound is called the variational lower bound or evidence lower bound, abbreviated as ELBO. Below we will provide two different approaches to derive ELBO. In the first approach, the ELBO is derived through Jensen's inequality, saying that given a random variable X and a convex function f, we have

$$f(\mathbb{E}(X)) \leq \mathbb{E}(f(X)). \tag{2.5}$$

The derivation is given as follows. For any distribution on $\mathbf{z}$, and in particular any parameterized inference model $q_\phi(\mathbf{z}|\mathbf{x})$, we have

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \log \int \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \log \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
&\geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log \left[\frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z})\right]. \qquad (2.6)
\end{aligned}$$

Jensen's inequality is used in the fourth line. Note that the second term in the final Equation 2.6 corresponds to a KL-divergence term:

$$\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z})\right] = D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})). \tag{2.7}$$

As it is a lower bound of the marginal data log likelihood $\log p_\theta(\mathbf{x})$, the expression in Equation 2.6 is the ELBO, denoted by $\mathcal{L}_{\theta,\phi}(\mathbf{x})$:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})). \tag{2.8}$$

An alternative derivation of the ELBO is also provided. It will help us to understand the gap between ELBO and the exact marginal log likelihood of data. Again, given any $q_\phi(\mathbf{z}|\mathbf{x})$, we have

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x})] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) - \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})).
\end{aligned} \tag{2.9}$$

Comparing Equation 2.9 with Equation 2.8, we obtain the following relationship:

$$\log p_\theta(\mathbf{x}) = \mathcal{L}_{\theta,\phi}(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})), \tag{2.10}$$

and by the properties of KL-divergence, we know that the ELBO is a lower bound on $\log p_\theta(\mathbf{x})$, and the gap between the ELBO and $\log p_\theta(\mathbf{x})$ (which is called the tightness of the bound) is 0 if and only if $q_\phi(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{z}|\mathbf{x})$, i.e., the parameterized inference model exactly matches the ground truth posterior.

Equation 2.10 tells us that by maximize the ELBO w.r.t. θ and ϕ, we are

1. Approximately maximizing the marginal log likelihood $\log p_\theta(\mathbf{x})$, which means that we are doing maximum likelihood training.

2. Minimizing $D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))$, which closes the gap between true posterior and the parameterized inference model.

2.1.3 Parameterization and Optimization

Parameterization of distributions in the VAE model: We need to find appropriate parameterization for the prior distribution $p(\mathbf{z})$, the encoder distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and the decoder distribution $p_\theta(\mathbf{x}|\mathbf{z})$. While there are multiple ways to define these distributions, and the definition depends on the data type (e.g., discrete or continuous), we will only discuss the most common parameterization assuming both $\mathbf{x}$ and $\mathbf{z}$ are continuous random variables.

- Decoder distribution $p_\theta(\mathbf{x}|\mathbf{z})$. When dealing with continuous data, the decoder distribution is usually chosen to be a factorized Gaussian. For example, if $\mathbf{x}$ represents an image, then each pixel in $\mathbf{x}$ follows an independent Gaussian distribution. Specifically, given the decoder network G_θ, $p_\theta(\mathbf{x}|\mathbf{z})$ is parameterized by

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; G_\theta(\mathbf{z}), \sigma^2 I). \tag{2.11}$$

  Note that typically the variance σ^2 is fixed and shared across all dimensions, although there are exceptions such as Dai and Wipf [2019], Vahdat and Kautz [2020].

- Prior distribution $p(\mathbf{z})$. The prior distribution is usually chosen to be a simple noise distribution that is easy to sample from. One common choice is $\mathcal{N}(\mathbf{x}; 0, I)$. More flexible priors with learnable distribution will be discussed in Chapter 3.

- Encoder distribution $q_\phi(\mathbf{z}|\mathbf{x})$. The most common way to parameterize this is as a Gaussian distribution with diagonal variance. Specifically, given an encoder network E_ϕ,

$$\begin{aligned}(\mu, \log \boldsymbol{\sigma}) &= E_\phi(\mathbf{x}) \\ q_\phi(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\mathbf{z}; \mu, \mathrm{diag}(\boldsymbol{\sigma})).\end{aligned}$$

Training objective: As discussed previously, the training objective of the VAE model is to maximize the ELBO in Equation 2.8, which is equivalent to

$$\min_{\theta,\phi} -\mathcal{L}_{\theta,\phi}(\mathbf{x}) = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})). \tag{2.12}$$

Given the specific parameterization of distributions discussed above, the first term corresponds to the negative Gaussian log likelihood of $\mathbf{x}$:

$$-\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] = \frac{1}{2\sigma^2}\|\mathbf{x} - G_\theta(\mathbf{z})\|^2, \tag{2.13}$$

which corresponds to an L_2 reconstruction error. The second term is the KL-divergence between the encoder distribution and the prior. If the common parameterization of these two distributions is used, then the KL-divergence can be expressed in an analytical form. In the general case, the KL-divergence can be approximated by Monte Carlo samples from $q_\phi(\mathbf{z}|\mathbf{x})$. The training objective of VAEs seeks a balance between

1. Enforcing reconstruction on $\mathbf{x}$ by minimizing the reconstruction loss, and

2. Regularizing $q_\phi(\mathbf{z}|\mathbf{x})$ towards uninformative prior $p(\mathbf{z})$.

Training θ is relatively easy, as the gradient of both terms in Equation 2.12 can be approximated by a batch of samples from $q_\phi(\mathbf{z}|\mathbf{x})$. However, there is a subtle thing when trying to train ϕ: the gradient is hard to compute as it is not trivial to differentiate through the sampling process that is used to approximate the expectation. Fortunately, for continuous $\mathbf{z}$, we can use the reparameterization trick to overcome the challenge.

Reparameterization Trick: The ELBO can be differentiated w.r.t both θ and ϕ through a change of variables, also called the reparameterization trick introduced by Kingma and Welling [2013], Rezende et al. [2014]. The trick is to 'externalize' the randomness in $\mathbf{z}$ by re-parameterizing the variable as a deterministic and differentiable function of an external variable. Specifically, we express the random variable $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ as some differentiable (and invertible) transformation of another random variable $\epsilon \sim p_0(\epsilon)$ that is absolutely independent of $\mathbf{x}$ or ϕ. In other words, denoting the transformation as r, we can draw a sample from $q_\phi(\mathbf{z}|\mathbf{x})$ by $\mathbf{z} = r(\epsilon, \phi, \mathbf{z})$. Given such a change of variable, expectations of a function f over $q_\phi(\mathbf{z}|\mathbf{x})$ can be rewritten in terms of ϵ:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \mathbb{E}_{p(\epsilon)}[f(\mathbf{z})], \tag{2.14}$$

where $\mathbf{z} = r(\epsilon, \phi, \mathbf{z})$. Then, using the fact that the expectation and gradient operators become commutative, we have

$$\begin{aligned} \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] &= \nabla_\phi \mathbb{E}_{p(\epsilon)}[f(\mathbf{z})] \\ &= \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(\mathbf{z})\right] \\ &= \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(r(\epsilon, \phi, \mathbf{z}))\right], \end{aligned} \tag{2.15}$$

which can be approximated by samples from $p_0(\epsilon)$.

When $q_\phi(\mathbf{z}|\mathbf{x})$ has the diagonal Gaussian parameterization, we can define the transformation r by the shift and scale of the Gaussian, and let $\epsilon \sim \mathcal{N}(0, I)$:

$$\mathbf{z} = \mu + \sigma \odot \epsilon,$$

where $(\mu, \log \boldsymbol{\sigma}) = E_\phi(\mathbf{x})$.

2.1.4 Typical issues with VAEs

VAEs are a very powerful class of models, mainly due to their flexibility. However, they also suffer from several issues. Here we would discuss two major issues of VAEs: the posterior collapse and the prior hole problem.

Posterior collapse: Recall that the ELBO can be decomposed into a reconstruction term and a KL regularization term. For a non-trainable prior like the standard Gaussian, the regularization term will be minimized if $q_\phi(\mathbf{z}|\mathbf{x}) = p(\mathbf{z})$ for all $\mathbf{x}$. It is possible with a strong decoder that the model may treat $\mathbf{z}$ as noise and reach an equilibrium state, where $q_\phi(\mathbf{z}|\mathbf{x}) \approx p(\mathbf{z})$ for all $\mathbf{x}$, that is hard to escape. This issue is known as the posterior collapse, which is typical when the VAE is trained on sequence data, due to the powerful auto-regressive decoder. One possible solution is proposed in Bowman et al. [2015], Sønderby et al. [2016], where the weight of the KL-regularization term is slowly annealed from 0 to 1 over the training.

Prior hole: Another issue is caused by the mismatch between the aggregated posterior

$$q_\phi(\mathbf{z}) = \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} \left[q_\phi(\mathbf{z}|\mathbf{x}) \right]$$

and the prior $p(\mathbf{z})$. Note that the aggregated posterior can be seen as the empirical marginal distribution of the latent variables assuming $q(\mathbf{z}|\mathbf{x})$ is the true conditional distribution. Note that new samples from a VAE are generated by sending a sample from the prior distribution to the decoder. Therefore, in order for VAEs to generate realistic samples, the aggregated posterior should match the prior distribution. However, it is observed that there are regions where there prior assigns high probability, but the aggregated posterior assigns low probability, or the other way around. Then, sampling from these holes provides unrealistic latent values and the decoder produces images of very low quality. This problem is referred to as the hole problem. Such a problem can be resolved by introducing a flexible and trainable prior, which is studied comprehensively in Chapter 3.

2.1.5 Hierarchical VAEs

There are many extensions of the VAE model that aim to improve the performance of generative modeling. Among them, the idea of hierarchical VAEs is an important one. Simple VAEs have only one level of latent variables, namely we assume all latent variables are directly dependent on data $\mathbf{x}$. Such a model may not be able to model a hierarchy of abstraction levels. For example, when we infer latent variables from an image of a human face, we may want the latent variables to capture not only low-level information such as skin color, hair color, and size of eyes but also high-level information such as the overall structure of the face. The information corresponds to a different level of abstraction, and it may be difficult to model all latent variables in a single level. As a result, hierarchical VAEs are introduced to increase the expressiveness of both the variational posterior and prior by partitioning the latent variables into disjoint groups $\mathbf{z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_L\}$ and designing conditional dependencies between them.

There are multiple ways to design the factorization. For example, when we assume $\mathbf{z} = \{\mathbf{z}_1, \mathbf{z}_2\}$, the generative model can be factorized as

$$p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta\left(\mathbf{x}|\mathbf{z}_1\right) p_\theta\left(\mathbf{z}_1|\mathbf{z}_2\right) p\left(\mathbf{z}_2\right), \tag{2.16}$$

where we have a unit Gaussian prior on $\mathbf{z}_2$, and $p_\theta\left(\mathbf{z}_1|\mathbf{z}_2\right)$ is a conditional Gaussian. The variational posterior can be factorized in multiple ways. Given the generative model in

Equation 2.16, we want write the variational posterior as either

$$q_\phi(\mathbf{z}|\mathbf{x}) = q_\phi(\mathbf{z}_1|\mathbf{x})q_\phi(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x})$$

or

$$q_\phi(\mathbf{z}|\mathbf{x}) = q_\phi(\mathbf{z}_2|\mathbf{x})q_\phi(\mathbf{z}_1|\mathbf{z}_2, \mathbf{x}).$$

The former is called a bottom-up inference model, and the latter is called a top-down inference model. It has been shown that it is advantageous to follow the top-down approach [Kingma et al., 2016, Salimans, 2016], which lets the generative model and inference model share the topological ordering of latent variables. One advantage of shared ordering is that this allows us to easily share parameters between the inference and generative models, leading to faster learning and better solutions.

Multiple hierarchical VAE models are proposed based on top-down inference. The idea was initially proposed by Kingma et al. [2016], Sønderby et al. [2016], and further developed by Maaløe et al. [2019]. Recently, large VAEs with very deep latent structure [Vahdat and Kautz, 2020, Child, 2021] achieve the best performance on likelihood modeling among VAEs. These approaches differ in their implementations and parameterizations used (i.e., architectures of DNNs), however, they all could be categorized as instantiations of top-down hierarchical VAEs.

2.2 Normalizing Flows

The search for probabilistic models that correctly describe the underlying processes that produce data is one of the enduring objectives of statistics and machine learning. Here we will discuss a straightforward way to address this need: building probability distributions as normalizing flows. Normalizing flows provide a general mechanism for constructing expressive probability distributions, only requiring the specification of a simple base distribution and a series of simple invertible transformations. Normalizing flows work by transforming a simple density through a series of transformations to produce a more complex and multi-modal distribution. The core idea is that repeated application of even simple transformations to a unimodal base density leads to a distribution of high complexity. Such a property makes normalizing flows useful in some key tasks in statistics such as modeling, inference, and simulation.

We will first give a brief historical overview of normalizing flows. The idea of whitening, which means transforming data into white noise through a deterministic transformation, has been discussed in early literature such as Johnson [1966] as a feature pre-processing tool. Chen and Gopinath [2000] use the whitening idea as a density estimation technique, similar to the purpose of modern normalizing flows. Their method is named *Gaussianization*. Tabak and Turner [2013] first introduces the concept of normalizing flows, describing the flow as a composition of simple mappings. Such a composition is essential for ensuring expressivity while preserving computational tractability. Rippel and Adams [2013] connects the idea of normalizing flows to deep learning by parameterizing flows with deep neural networks. Rezende and Mohamed [2015] used the idea and language from Tabak and Turner [2013] to apply normalizing flows in variational inference, making the variational posterior of VAEs more expressive. A series of works explore the parameterization of normalizing flows and introduce a scalable and computationally efficient architecture [Dinh et al., 2014, 2016, Kingma and Dhariwal, 2018], demonstrating further improvements to modeling and inference. They will be reviewed in this chapter.

In the following, we will first introduce the mathematical formulation of normalizing flows, followed by several important flow parameterizations. Applications of normalizing flows and their limitations will be discussed towards the end.

2.2.1 Fundamentals of Normalizing Flows

We begin by outlining basic definitions and properties of normalizing flows. Normalizing flows provide a general way of constructing flexible probability distributions over continuous random variables. Let $\mathbf{x}$ be the random variable we are interested in modeling. The main idea of flow-based modeling is to express $\mathbf{x}$ as an invertible transformation T of a sample $\mathbf{z}$ from a base distribution $p_0(\mathbf{z})$:

$$\mathbf{x} = T(\mathbf{z}), \quad \mathbf{z} \sim p_0(\mathbf{z}). \tag{2.17}$$

Note that in comparing this with the formulation of latent variable models, it is tempting to call $p_0(\mathbf{z})$ the prior distribution and $\mathbf{z}$ the latent variable, but this terminology is not well-suited for normalizing flows, as given observable variable $\mathbf{x}$, $\mathbf{z}$ can be uniquely determined by $\mathbf{z} = T^{-1}(\mathbf{x})$, and hence $\mathbf{z}$ is no longer 'latent'. As a result, we refer to p_0 as the base distribution.

To make a normalizing flow model, the transformation T must be bijective (or invertible), and both T and T^{-1} must be differentiable. Note that these requirements imply that $\mathbf{z}$ must have the same dimension as $\mathbf{x}$. A transformation T that satisfies such properties is called a diffeomorphism. When T is a diffeomorphism, the density of $\mathbf{x}$ is well-defined and can be obtained by change-of-variables. Specifically, we have

$$p(\mathbf{x}) = p_0(\mathbf{z}) \left|\det J_T(\mathbf{z})\right|^{-1}, \tag{2.18}$$

where $\mathbf{z} = T^{-1}(\mathbf{x})$ and the Jacobian $J_T(\mathbf{z})$ is the $D \times D$ matrix (assuming $\mathbf{z}$ and $\mathbf{z}$ are both in $\mathbb{R}^D$) of all partial derivatives of T evaluated at $\mathbf{z}$:

$$J_T(\mathbf{z}) = \begin{bmatrix} \frac{\partial T_1}{\partial \mathbf{z}_1} & \cdots & \frac{\partial T_1}{\partial \mathbf{z}_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_D}{\partial \mathbf{z}_1} & \cdots & \frac{\partial T_D}{\partial \mathbf{z}_D} \end{bmatrix}. \tag{2.19}$$

Equivalently, $p(\mathbf{x})$ can be written in a form that only involves $\mathbf{x}$:

$$p(\mathbf{x}) = p_0(T^{-1}(\mathbf{x})) \left| \det J_{T^{-1}}(\mathbf{x}) \right|, \tag{2.20}$$

due to the inverse function theorem saying that if T^{-1} is continuously differentiable and $\mathbf{z} = T^{-1}(\mathbf{x})$,

$$J_{T^{-1}}(\mathbf{x}) = J_T(\mathbf{z})^{-1}, \tag{2.21}$$

and the property of determinants that $\det A^{-1} = \frac{1}{\det A}$ for any invertible matrix A. In practice, we often construct a flow-based model by implementing T with a neural network and taking p_0 to be a simple distribution such as unit Gaussian. Intuitively, we can think of the transformation T as warping the space in order to push the density $p_0(\mathbf{z})$ into $p(\mathbf{x})$, and the absolute value of the Jacobian determinant term quantifies the relative change of volume caused by applying T.

An important property of invertible and differentiable transformations is that they are composable. Specifically, the composition of two such transformations T_1 and T_2 is also invertible and differentiable. The inverse and determinant of Jacboian of the composition can be expressed as

$$(T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1} \tag{2.22}$$

$$\det J_{T_2 \circ T_1}(\mathbf{z}) = \det J_{T_2}(T_1(\mathbf{z})) \cdot \det J_{T_1}(\mathbf{z}). \tag{2.23}$$

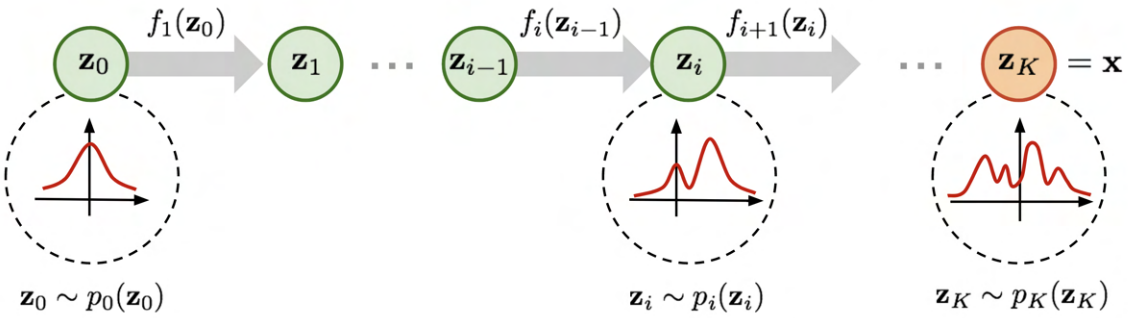


Figure 2.1: Illustration of a normalizing flow model, transforming a simple distribution p_0 to a complex one through composition of transformations.

The second equation is due to the property of the Jacobian of composition of functions and the fact $\det AB = \det A \det B$. In consequence, we can build complex transformations by composing multiple simpler transformations, without compromising the requirements of invertibility and differentiability. The same idea can be extended to composing multiple transformations $T_1, T_2, \ldots, T_K$ to obtain $T = T_K \circ \cdots \circ T_1$, where T_k transforms $\mathbf{z}_{k-1}$ into $\mathbf{z}_k$, assuming $\mathbf{z}_0 = \mathbf{z}$ and $\mathbf{z}_K = \mathbf{x}$. With a chain of transformations, the 'flow' is made of the trajectory that a sample from the base distribution $p_0(\mathbf{z})$ follows as it is gradually transformed by the sequence of transformations. The word 'normalizing' refers to the inverse of the chain of transformations which takes a collection of samples from $p(\mathbf{x})$ and transforms them (hence 'normalizes' them) into a collection of samples from p_0. Figure 2.1 illustrates the idea of normalizing flow [1].

A flow model allows us to

1. Sample from the model by first sampling $\mathbf{z} \sim p_0(\mathbf{z})$ and then apply T, and

2. Evaluate the density of a sample $\mathbf{x}$ by Equation 2.20.

These operations have different computational requirements. Sampling from the model requires the ability to sample from the base distribution and computing the forward trans-

[1] Figure adapted from `https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html`

formation T. Evaluating the model's density requires computing the inverse transformation $T^(-1)$ and compute the determinant of Jacobian. Some parameterizations of flows satisfy all these computational requirements, while some parameterizations only satisfy some of these requirements, making them suitable for specific applications.

Training: Fitting a normalizing flow to a target distribution can be done by maximizing likelihood, as the likelihood of $\mathbf{x}$ under the model can be specified. In particular, denoting the set of parameters of all transformations to be θ, the objective for training a normalizing flow is

$$\begin{aligned}\max_\theta \mathcal{L}(\theta) &= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log p_\theta(\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log p_0(T_\theta^{-1}(\mathbf{x})) + \log\left|\det J_{T_\theta^{-1}}(\mathbf{x})\right|\right]. \end{aligned} \tag{2.24}$$

Maximum likelihood training is well suited for situations in which we have samples from the target distribution but we cannot evaluate the ground truth density $p(\mathbf{x})$. We also need to explicitly compute and differentiate T^{-1} and the determinant of Jacobian.

Alternatively, the model can also be trained by minimizing the reverse KL divergence:

$$\begin{aligned}\min \mathcal{L}_{\text{reverse}}(\theta) &= D_{\text{KL}}\left(p_\theta(\mathbf{x})\|p(\mathbf{x})\right) \\ &= \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\left[\log p_\theta(\mathbf{x}) - \log p(\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{z}\sim p_0(\mathbf{z})}\left[\log p_0(\mathbf{z}) - \log\left|\det J_{T_\theta}(\mathbf{z})\right| - \log p(T_\theta(\mathbf{z}))\right], \end{aligned} \tag{2.25}$$

where in the third line we apply change of variable to express the expectation with respect $\mathbf{z}$. The reverse KL divergence is suitable when we have the ability to evaluate the target density but not necessarily sample from it. For example, this objective is used in variational inference, where a normalizing flow is used to model the variational posterior [Rezende and Mohamed, 2015, Kingma et al., 2016].

2.2.2 Parameterizations of Normalizing Flows

Having described a high-level formulation for normalizing flows, we transition into describing various ways to construct a flow. We have discussed that a flow T consists a chain of transformations T_k that are composed as $T = T_K \circ \cdots \circ T_1$. In the case of maximum likelihood training, we need to compute the inverse and Jacobian determinant, and therefore we need each block of transformation to have tractable inverse and Jacobian determinant. These requirements make the computation of inverse and Jacobian determinant of the whole flow T tractable, since

$$T^{-1} = T_1^{-1} \circ \cdots \circ T_K^{-1}, \tag{2.26}$$

and

$$\log \left|\det J_T\left(\mathbf{z}_0\right)\right| = \log \left|\prod_{k=1}^{K} \det J_{T_k}\left(\mathbf{z}_{k-1}\right)\right| = \sum_{k=1}^{K} \log \left|\det J_{T_k}\left(\mathbf{z}_{k-1}\right)\right|. \tag{2.27}$$

We focus on the density estimation task, which is done by maximum likelihood training. Therefore, we only discuss flow parameterizations that have tractable inverse and Jacobian determinant.

A tractable Jacobian determinant means that we can compute the Jacobian determinant efficiently (in linear time). Note that for a general invertible function with $D-$dimensional input and output, the complexity for computing the Jacobian determinant is D^3, which is infeasible for large D. Hence, we need to design functional forms that allow the Jacobian determinant to be computed in linear time with respect to the input dimensionality.

We introduce two particular parametrizations of normalizing flows: RealNVP [Dinh et al., 2016] and Glow [Kingma and Dhariwal, 2018].

RealNVP: The RealNVP (Real-valued Non-Volume Preserving) model implements a normalizing flow by stacking a sequence of invertible transformation with *affine coupling*. Specifically, for each transformation $f : \mathbf{x} \in \mathbb{R}^D \to \mathbf{y} \in \mathbb{R}^D$, the input dimensions are split into two parts with dimension d and $D - d$ respectively, where

1. The first d dimensions stay the same: $\mathbf{y}_{1:d} = \mathbf{x}_{1:d}$, and
2. The remaining d dimensions undergo an affine transformation, where both the scale and shift parameters are functions of the first d dimensions: $\mathbf{y}_{d+1:D} = \mathbf{x}_{d+1:D} \odot \exp\left(s\left(\mathbf{x}_{1:d}\right)\right) + t\left(\mathbf{x}_{1:d}\right)$, where s and t and translation functions parametrized by neural networks, and $\odot$ denotes elementwise product.

It is easy to verify the invertibility of such a transformation:

$$\begin{aligned}
&\mathbf{x}_{1:d} = \mathbf{y}_{1:d} \\
&\mathbf{x}_{d+1:D} = \left(\mathbf{y}_{d+1:D} - t\left(\mathbf{y}_{1:d}\right)\right) \odot \exp\left(-s\left(\mathbf{y}_{1:d}\right)\right).
\end{aligned}$$

Next, we verify that the Jacobian determinant is easy to compute. The Jacobian has the following form:

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \operatorname{diag}\left(\exp\left(s\left(\mathbf{x}_{1:d}\right)\right)\right), \end{bmatrix} \tag{2.28}$$

which is an upper triangular matrix. We know that the determinant for an upper triangular matrix is simply the product of diagonal entries, so the Jacobian determinant is

$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp\left(s\left(\mathbf{x}_{1:d}\right)\right)_j = \exp\left(\sum_{j=1}^{D-d} s\left(\mathbf{x}_{1:d}\right)_j\right), \tag{2.29}$$

which can be computed in linear time. Further note that in this parameterization, the inverse transformation does not involve the inverse of s or t, and the Jacobian determinant does not involve computing the Jacobian of s or t, so both s and t can be modeled by arbitrarily complex neural networks.

One potential issue of the affine coupling layer is that some dimensions (channels when applied to image data) remain unchanged through the transformation. To make sure all the inputs have a chance to be transformed, the model reverses the ordering in each layer so that different components are left unchanged. Following such an alternating pattern, the set of units that remain identical in one transformation layer is always modified in the next.

Glow: The Glow model makes modifications to the RealNVP model. In particular, it uses affine coupling layers and adds an activation norm module. It simplifies the architecture by replacing the reverse permutation operation on the channel ordering with invertible 1×1 convolutions.

The activation norm is an operation similar to batch normalization, which performs an affine transformation using a scale and bias parameter per channel. However, unlike batch normalization, the activation norm also works when the batch size is 1. Kingma and Dhariwal [2018] found that the additional activation norm module before each transformation improves the performance.

The invertible 1×1 convolution module replaces the permutation operation between two blocks of transformations in the RealNVP model. Recall that between blocks of the RealNVP flow, the ordering of channels is reversed so that all the data dimensions have a chance to be altered. A 1×1 convolution with an equal number of input and output channels is a generalization of any permutation of the channel order. We illustrate that the Jacobian determinant of this module can be computed efficiently. Suppose we want to apply the invertible 1×1 convolution on a feature map $\mathbf{h}$ with size $h \times w \times c$, the weight matrix $\mathbf{W}$ has size $c \times c$. The output f after the convolution is a tensor with size $h \times w \times c$. It can be shown that (see Kingma and Dhariwal [2018])

$$\log \left| \det \frac{\partial f}{\partial \mathbf{h}} \right| = \log \left(| \det \mathbf{W} |^{h \cdot w} | \right) = h \cdot w \cdot \log | \det \mathbf{W} | \tag{2.30}$$

Since the weight matrix is relatively small, the matrix determinant can be computed efficiently.

2.2.3 Applications of Normalizing Flows

Normalizing flows, due to their ability to be expressive while still allowing for exact likelihood calculations, have been widely used in probabilistic modeling. Besides estimating the density of given data, normalizing flows are also powerful generative models that can produce synthetic samples. For example, normalizing flows have been applied to generation tasks on images [Kingma and Dhariwal, 2018], video [Kumar et al., 2019], Audio [Oord et al., 2018, Prenger et al., 2019, Kim et al., 2018], text [Tran et al., 2019, Ziegler and Rush, 2019], graph [Deng et al., 2019, Madhawa et al., 2019] and 3D point cloud [Yang et al., 2019].

Normalizing flows are also applied to model statistical distributions other than the marginal data distribution as in the generation task. For example, in Müller et al. [2019], the proposal distribution for importance sampling is modeled by normalizing flows. In Song et al. [2017], the authors proposed A-NICE-MCMC, an MCMC algorithm similar to Hamiltonian Monto-Carlo but with a volume-preserving normalizing flow as the proposal. Another way of applying flows to MCMC is to use the flow to reparameterize the target distribution [Hoffman et al., 2019]. Normalizing flows can also usefully serve as posterior approximations over latent variables [Rezende and Mohamed, 2015, Kingma et al., 2016, Tomczak and Welling, 2016, Berg et al., 2018]. Flows are also used in likelihood-free inference (also called simulated-based inference). In likelihood-free inference, we assume to have a model parameterized by η, but we do not have the likelihood function $p(\mathbf{x}|\eta)$, rather we have a simulator that takes η and simulates observations $\mathbf{x}$ [Cranmer et al., 2020]. Such simulator-based models are common in scientific fields such as cosmology and high-energy physics. Normalizing flows have shown impressive results on this task [Winkler et al., 2019, Gonçalves et al., 2020].

2.2.4 Limitations of Normalizing Flows

The normalizing flow model has one important limitation: its parameter efficiency. As we discussed before, normalizing flows need to maintain invertibility and tractability of computing the Jacobian determinant. As a result, their functional forms are constrained. Although in theory normalizing flows can represent a broad class of distributions [Kong and Chaudhuri, 2020], the actual expressivity is limited by their constrained parameterizations. Typically the power of each transformation in the flow is very limited. For example, in RealNVP or Glow, the transformation defining an affine coupling layer is just an affine transformation. In residual flows [Chen et al., 2019, Behrmann et al., 2019], the inverse is not defined explicitly but rather relies on the use of the Banach fixed point theorem, which requires the flow to be contractive, i.e. with Lipschitz constant strictly less than unity. The requirement on the Lipschitz constant strongly constrains the expressive power of the model. As a result, usually, a normalizing flow requires a large number of transformation blocks, resulting in a large number of parameters. It is observed that flow models are difficult to train, and their sample quality and test data likelihood typically lag behind competing models [Kingma and Dhariwal, 2018].

One model that alleviates normalizing flows' issue of parameter inefficiency is continuous normalizing flows [Chen et al., 2018a, Grathwohl et al., 2018], which will be discussed in Chapter 5. Another idea is training normalizing flows on a latent space that has relatively low dimensions. This idea will be studied comprehensively in Chapter 3.

2.3 Energy-based Models

In Section 1.2.1, we discussed how to parameterize a distribution for maximum likelihood training. The parameterization needs to satisfy the non-negativity and normalization conditions in Equation 1.11. There are multiple ways to ensure such conditions. For example, VAEs enforce the conditions by designing $p_\theta(\mathbf{x})$ as the marginalization of a tractable joint distribution $p_\theta(\mathbf{x}, \mathbf{z})$, and normalizing flows enforce the conditions by change of variables over a base distribution, where the Jacobian determinant term ensures the normalization. While these model parameterizations satisfy the requirements, specific designs are needed and the model architectures are restricted, and hence the expressivity might be compromised due to the restrictions. In addition, models with a tractable density that satisfies the conditions assume that exact synthesis from the model can be done with a specified, tractable procedure, but such an assumption is not always natural. In this chapter, we give an introduction to Energy-based Models (EBMs), where the non-negativity and normalization conditions are ensured directly by definition. EBMs are much less restrictive in functional form: instead of specifying a normalized probability, they only specify the unnormalized negative log-probability, called the energy function. After briefly reviewing the history of EBMs, we will introduce the formulation of EBMs as well as several training strategies.

EBMs have a long history that dates back to the 80s when models called Boltzmann Machines [Ackley et al., 1985] were proposed. The idea behind Boltzmann Machines is taken from statistical physics and was popularized in the cognitive science community, due to their locality and the Hebbian nature of their training algorithm which connects to neural science. Later, Restricted Boltzmann Machines (RBMs) [Smolensky, 1986, Hinton, 2012] were developed based on Boltzmann machines, with the restriction that their neurons must form a bipartite graph. This restriction allows for more efficient training algorithms than are available for the general class of Boltzmann machines, in particular the gradient-based contrastive divergence algorithm [Carreira-Perpinan and Hinton, 2005], which is still widely used for training deep EBMs today. RBMs can also be used when combined with deep learning. In particular, deep belief networks can be formed by "stacking" RBMs and optionally fine-tuning the resulting deep network with gradient descent and backpropagation [Hinton, 2009]. EBMs were used originally in image analysis in Geman and Geman [1984]. Modern deep EBMs, which use deep neural networks to parameterize the energy function, are developed by [Xie et al., 2016, Du and Mordatch, 2019].

2.3.1 Formulation of EBMs

The core idea behind EBMs is to define a function E_θ which is non-negative by construction, but does not necessarily integrate to 1 over its support. To do so, we introduce the energy function f_θ. One way to define E_θ that satisfies non-negativity is by using an exponential form:

$$E_\theta(\mathbf{x}) = e^{-f_\theta(\mathbf{x})}. \tag{2.31}$$

There are several advantages to this form (over other possible choices that can ensure non-negativity, such as $E_\theta(\mathbf{x}) = f_\theta(\mathbf{x})^2$). The exponential formulation can capture very large variations in density, as log-probability is the natural scale we want to work with, while other formulations may need highly non-smooth f_θ. In addition, this formulation aligns with the intuition that $\mathbf{x}$ with low energy (high $f_\theta(\mathbf{x})$) is more likely.

The normalizing constraint can be satisfied by explicitly normalizing $E_\theta(\mathbf{x})$ by the total integral Z_θ:

$$Z_\theta = \int_{\mathbf{x}} E_\theta(\mathbf{x}) d\mathbf{x}. \tag{2.32}$$

Note that Z_θ does not depend on particular $\mathbf{x}$'s as it is obtained from the integral. Since the constant Z_θ ensures the normalization constraints, it is called the normalizing constant.

With the energy function and the normalizing constant in hand, we can define an EBM as

$$p_\theta(\mathbf{x}) = \frac{1}{Z_\theta} e^{-f_\theta(\mathbf{x})}. \tag{2.33}$$

When Z_θ can be computed explicitly, we obtained a tractable model whose likelihood can be computed. For example, for a Gaussian distribution, we have

$$E_{\mu,\sigma^2}(\mathbf{x}) = e^{-\frac{(\mathbf{x}-\mu)^2}{2\sigma^2}},$$

and the normalizing constant

$$Z_{\mu,\sigma^2} = \int_{\mathbf{x}} e^{-\frac{(\mathbf{x}-\mu)^2}{2\sigma^2}} d\mathbf{x} = \sqrt{2\pi\sigma^2}.$$

Similar derivations can be found for many familiar distributions, such as exponential, Poisson, gamma, etc. Those distributions belong to the so-called exponential family. However, in general, EBMs do not require that Z_θ can be computed analytically. In fact, EBMs allow arbitrary energy functions f_θ whose resulting normalizing constant cannot be computed or even estimated in high-dimensional data space. For example, when $f_\theta(x)$ is a neural network that takes data $\mathbf{x}$ as its input and returns a scalar, certainly there is no easy way to estimate Z_θ. We will discuss how to train EBMs with unknown normalizing constant later.

Next, we discuss several important Energy-based Models.

Product of Experts: A Product of Experts model (PoE) [Hinton, 2002] combines a number of individual component models (the experts) by taking their product and normalizing the result. Each expert is defined as a possibly unnormalized probabilistic model $\tilde{p}_{\theta_i}$:

$$p_\theta(\mathbf{x}) = \frac{1}{Z_\theta} \prod_{i=1}^{M} \tilde{p}_{\theta_i},$$

where θ is the set of all parameters $\{\theta_1, \cdots, \theta_M\}$. Note that PoEs stand in contrast to Mixture Models which combine expert models additively. Variants of PoEs have been applied to compositional visual generation [Du et al., 2020], where each visual concept is modeled by an expert, and sampling from the combined distribution correspond to compositions of concepts.

Restricted Boltzmann Machines: Restricted Boltzmann Machines (RBMs) are EBMs with latent variables. In RBMs, both the observable variable $\mathbf{x}$ and latent variable $\mathbf{z}$ are assumed to be binary. For $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{z} \in \{0,1\}^m$, and RBM model expresses the joint distribution as

$$\begin{aligned} p_{W,b,c}(\mathbf{x}, \mathbf{z}) &= \frac{1}{Z_{W,b,c}} \exp\left(\mathbf{x}^T W \mathbf{z} + b^T \mathbf{x} + c^T \mathbf{z}\right) \\ &= \frac{1}{Z_{W,b,c}} \exp\left(\sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{x}_i \mathbf{z}_j w_{ij} + \sum_{i=1}^{n} \mathbf{x}_i b_i + \sum_{j=1}^{m} \mathbf{z}_j c_j\right). \end{aligned} \tag{2.34}$$

The term 'restricted' refers to the restriction that there are no visible-visible and hidden-hidden connections, i.e., $\mathbf{x}_i \mathbf{x}_j$ or $\mathbf{z}_i \mathbf{z}_j$ terms in the model. Note that RBMs are special cases of PoE models, since marginalizing $\mathbf{z}$ of an RBM model gives

$$P(x) = \frac{1}{\tilde{Z}} \prod_i \exp\left(b_i \mathbf{x}_i\right) \prod_j \left(1 + \exp\left(c_j + \sum_i W_{ij} \mathbf{x}_i\right)\right). \tag{2.35}$$

RBMs have been applied to face recognition [Teh and Hinton, 2000] and collaborative filtering [Salakhutdinov et al., 2007]. in addition, Stacked RBMs are one of the first deep generative models [Hinton, 2009]. In a Stacked RBM, bottom layer variables are pixel values, and layers above represent “higher-level” features (corners, edges, etc). In the early years of deep learning, neural networks for supervised learning had to be pre-trained as Stacked RBMs to make them work.

Deep EBMs: One of the core ideas of deep learning is to replace heuristic designs with end-to-end learning. Deep EBMs follow this approach by directly modeling the energy function f_θ with a deep neural network that takes $\mathbf{x}$ as input and return the scalar energy. When $\mathbf{x}$ is an image, the neural network is typically chosen to have convolutional structures. Examples of deep EBMs include [Xie et al., 2016, Du and Mordatch, 2019].

2.3.2 Maximum Likelihood Training with MCMC

From the formulation of EBMs, we know that EBMs have the advantage that they are extremely flexible, as they allow to plug in any energy function f_θ. Such flexibility allows EBM to model arbitrarily complex data distribution, but it also poses some challenges to the training of EBMs, since many choices of f_θ lead to an intractable normalizing constant. In the following sections, we discuss several methods of training EBMs. The first method we discuss is maximum likelihood training, which is the most widely used.

We cannot directly compute the likelihood of an EBM as in the maximum likelihood approach due to the intractable normalizing constant Z_θ. Fortunately, it turns out that we can still estimate the gradient of the log-likelihood with MCMC methods, allowing for maximum likelihood training with gradient ascent. The derivation for the gradient estimation comes from the contrastive divergence algorithm [Carreira-Perpinan and Hinton, 2005, Woodford, 2006]. We will provide the derivation below.

The gradient of the log-probability of an EBM in Equation 2.33 can be decomposed into two terms:

$$\nabla_\theta \log p_\theta(\mathbf{x}) = -\nabla_\theta f_\theta(\mathbf{x}) - \nabla_\theta \log Z_\theta. \tag{2.36}$$

The first term is straightforward to evaluate with automatic differentiation. However, the second term is difficult to estimate, as the normalizing constant is intractable to compute.

$\nabla_\theta \log Z_\theta$ can be written as

$$\begin{aligned}
\nabla_\theta \log Z_\theta &= \nabla_\theta \log \int \exp\left(-f_\theta(\mathbf{x})\right) d\mathbf{x} \\
&= \frac{1}{\int \exp\left(-f_\theta(\mathbf{x})\right) d\mathbf{x}} \nabla_\theta \int \exp\left(-f_\theta(\mathbf{x})\right) d\mathbf{x} \\
&= \frac{1}{\int \exp\left(-f_\theta(\mathbf{x})\right) d\mathbf{x}} \int \nabla_\theta \exp\left(-f_\theta(\mathbf{x})\right) d\mathbf{x} \\
&= \frac{1}{\int \exp\left(-f_\theta(\mathbf{x})\right) d\mathbf{x}} \int \exp\left(-f_\theta(\mathbf{x})\right) \left(-\nabla_\theta f_\theta(\mathbf{x})\right) d\mathbf{x} \\
&= \int \frac{\exp\left(-f_\theta(\mathbf{x})\right)}{Z_\theta} \left(-\nabla_\theta f_\theta(\mathbf{x})\right) d\mathbf{x} \\
&= \int p_\theta(\mathbf{x}) \left(-\nabla_\theta f_\theta(\mathbf{x})\right) d\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})} \left[-\nabla_\theta f_\theta(\mathbf{x})\right], \qquad (2.37)
\end{aligned}$$

where the second equation follows the gradient of the logarithm, the third equation interchanges the grdient and integral, and the fifth equation recognizes that $Z_\theta = \int \exp\left(-f_\theta(\mathbf{x})\right) d\mathbf{x}$. Therefore, we obtain the gradient of the likelihood as:

$$\nabla_\theta \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \left[\log p_\theta(\mathbf{x})\right] = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \left[-\nabla_\theta f_\theta(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})} \left[\nabla_\theta f_\theta(\mathbf{x})\right]. \qquad (2.38)$$

Both expectations in Equation 2.38 can be approximated by Monte-Carlo samples. For the first expectation, the positive phase, samples are drawn from the data distribution $p(\mathbf{x})$, and for the second expectation, the negative phase, samples are drawn from the model $p_\theta(\mathbf{x})$ itself.

Gradient estimation also gives an interesting interpretation for the maximum likelihood training of EBMs. If we do gradient ascent with the gradient estimated by Equation 2.38, the first term corresponds to minimizing the value of energy function f_θ over real data, which is equivalent to increasing the log-likelihood of real data. This is certainly the goal of maximum likelihood training. Interestingly, the second term corresponds to maximizing the

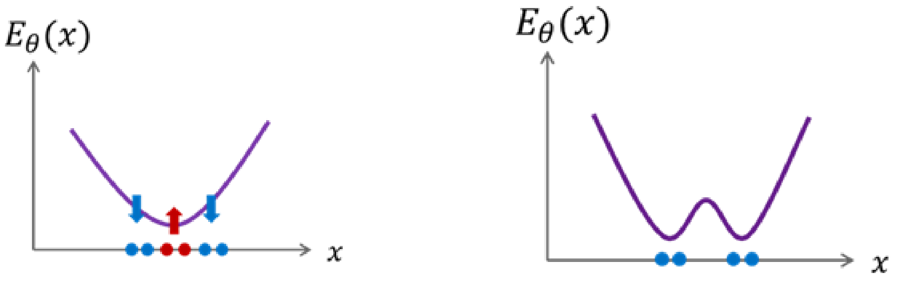


Figure 2.2: Illustration of maximum likelihood training of EBMs. **Left:** shape of the initial energy function. Blue dots are real data, red dots are samples from the model. The training update increases the energy of sampled data, and decreases the energy of real data. **Right:** the energy function after the update. It assigns lower energy value to real data, and higher energy value at sampled data ensure the normalization constraint.

value of energy function f_θ over samples from the model, which is equivalent to decreasing the log-likelihood of sampled data. The purpose of doing so is to ensure the normalizing constraint: when pushing up the density of some regions, we have to push down the density of some other regions to ensure the total integral is 1. This interpretation is illustrated in Figure 2.2.

With gradient estimation, the only remaining thing is to approximate the second term with samples from the model. As long as we can draw random samples from the model, we have access to an unbiased Monte Carlo estimate of the log-likelihood gradient, allowing us to optimize the parameters with stochastic gradient ascent. However, sampling from p_θ itself is non-trivial, as p_θ is unnormalized. MCMC algorithms have to be used, and since we have access to $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ (because $\nabla_{\mathbf{x}} \log Z_\theta = 0$), gradient-based MCMC, such as Langevin dynamics (introduced in Section 1.2.3) and Hamiltonian Monte Carlo [Neal et al., 2011] are natural choices. For example, when using Langevin MCMC to sample from $p_\theta(\mathbf{x})$, we first draw an initial point $\mathbf{x}_0$ from a simple noise distribution and iterate with the Langevin diffusion process for K steps

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha \underbrace{\nabla_{\mathbf{x}} \log p_\theta \left(\mathbf{x}^k\right)}_{=-\nabla_{\mathbf{x}} f_\theta(\mathbf{x})} + \sqrt{2\alpha}\epsilon^k, \quad \epsilon \sim \mathcal{N}(0, I). \tag{2.39}$$

As discussed in Section 1.2.3, $\mathbf{x}^K$ is guaranteed to follow the p_θ distribution if $\alpha \to 0$ and $K \to \infty$ under some regularity conditions. In practice, we have to use a small finite ϵ and a finite number of steps K. There are two practical ways to run the Langevin dynamics for training EBMs with maximum likelihood: persistent LD and short-run LD. In persistent LD, we do not restart the MCMC chain when training on a new data point; rather, we initialize a new MCMC using the state of the previous MCMC chain. This method can be further improved by keeping multiple historical states of the MCMC chain in a replay buffer and initializing new MCMC chains by randomly sampling from it [Du and Mordatch, 2019]. In contrast, short-run LD [Nijkamp et al., 2019] initializes $\mathbf{x}_0$ from noise in every iteration and runs the LD for a small number of steps.

2.3.3 Alternative Methods for Training EBMs

Besides the widely used maximum likelihood training, there are alternative methods for training EBMs. In this section, we discuss two of them: denoising score matching and noise-contrastive estimation.

Denoising Score Matching: We know that if two real-valued functions f and g have the same first-order derivatives everywhere, then they only differ by a constant. When f and g are the log-density of two distributions with with equal first-order derivatives, then by the normalization requirement which requires that both $e^{f(\mathbf{x})}$ and $e^{g(\mathbf{x})}$ integrate to 1, we can conclude that $f(\mathbf{x}) = g(\mathbf{x})$. As a result, one can learn an EBM by matching the derivatives of its log-density to the derivatives of the log-density of the data distribution. The first-order gradient of a log-density is called the score function of the corresponding distribution, so the idea of matching the derivatives is called score matching [Hyvärinen and Dayan, 2005]. The score matching objective is to minimize the Fisher divergence between two distributions:

$$D_F\left(p(\mathbf{x}) \| p_\theta(\mathbf{x})\right)=\mathbb{E}_{p_{\text{data}}\,(\mathbf{x})}\left[\frac{1}{2}\left\|\nabla_{\mathbf{x}} \log p_{\text{data}}\,(\mathbf{x})-\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})\right\|^2\right] \tag{2.40}$$

However, the first term on the right is generally impractical to calculate since it requires knowing $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, the derivative of the density of the data distribution. It can be shown that minimizing the Fisher divergence in Equation 2.40 is equivalent to minimizing the following expression

$$\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\operatorname{tr}\left(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x})\right)+\frac{1}{2}\left\|\mathbf{s}_\theta(\mathbf{x})\right\|_2^2\right] \tag{2.41}$$

where we use $\mathbf{s}_\theta$ to denote $\nabla_{\mathbf{x}} \log p_\theta$. This objective does not need the derivative of the ground truth density, but it requires evaluating higher-order derivatives, which is computationally expensive.

One practical way to do score matching is denoising score matching (DSM) [Vincent, 2011], which involves a kernel density estimate of $p(\mathbf{x})$. It first perturbs the data point $\mathbf{x}$ with a pre-specified noise distribution $q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ and then employs score matching to estimate the score of the perturbed data distribution $q_\sigma(\tilde{\mathbf{x}}) \triangleq \int q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p_{\text{data}}\,(\mathbf{x}) \mathrm{d}\mathbf{x}$, which is the data distribution $p(\mathbf{x})$ convolved with the noise distribution. The objective is

$$\begin{aligned} D_F\left(q(\tilde{\mathbf{x}}) \| p_\theta(\tilde{\mathbf{x}})\right) &= \mathbb{E}_{q(\tilde{\mathbf{x}})}\left[\frac{1}{2}\left\|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}})-\nabla_{\mathbf{x}} \log p_\theta(\tilde{\mathbf{x}})\right\|_2^2\right] \\ &= \mathbb{E}_{q(\tilde{\mathbf{x}}|\mathbf{x}) p(\mathbf{x})}\left[\frac{1}{2}\left\|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}}|\mathbf{x})-\nabla_{\mathbf{x}} \log p_\theta(\tilde{\mathbf{x}})\right\|_2^2\right]+\text{ constant }, \end{aligned} \tag{2.42}$$

where the expectation is approximated by the samples, thus completely avoiding both the unknown term $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ and computationally expensive second order derivatives. Note that DSM is not a consistent objective because the optimal EBM matches the noisy distribution $q_\sigma(\tilde{\mathbf{x}})$, not the data distribution $p(\mathbf{x})$. This inconsistency becomes non-negligible when $q_\sigma(\tilde{\mathbf{x}})$ significantly differs from $p(\mathbf{x})$. One way to alleviate the inconsistency of DSM is to choose $q_\sigma \approx p$, i.e., use a small noise perturbation. However, this often significantly increases the variance of objective values. Variance-reducing techniques for training DSMs are introduced in Wang et al. [2020].

We want to emphasize that DSM is closely connected to score-based generative models [Song and Ermon, 2019], which will be introduced in Section 2.4.

Noise Contrastive Estimation: The high level ideas of Noise Contrastive Estimation (NCE) [Gutmann and Hyvärinen, 2012] were introduced in Section 1.2.2. Here we briefly introduce its application to training EBMs. Contrary to most other EBMs, now we treat Z_θ as a learnable scalar parameter. Suppose we have training examples $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ from $p(\mathbf{x})$ and samples $\{\tilde{\mathbf{x}}_1, \cdots, \tilde{\mathbf{x}}_N\}$ from a noise distribution $q(\mathbf{x})$, the parameter θ (including the learnable estimate of normalizing constant) can be learned by maximizing the following objective [Gutmann and Hyvärinen, 2012]:

$$J(\theta) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x})}{p_\theta(\mathbf{x}) + q(\mathbf{x})}\right] + \mathbb{E}_{\mathbf{x}\sim q(\mathbf{x})}\left[\log \frac{q(\mathbf{x})}{p_\theta(\mathbf{x}) + q(\mathbf{x})}\right]. \tag{2.43}$$

Note that we can compute $p_\theta(\mathbf{x})$ exactly as we now have the normalizing constant. The objective transforms the estimation of EBM into a classification problem.

The choice of the noise distribution $q(\mathbf{x})$ is important. We want $q(\mathbf{x})$ to satisfy the following: (1) analytically tractable density; (2) easy to sample from; (3) close to data distribution. In particular, (3) is important for learning a model over high-dimensional data. If $q(\mathbf{x})$ is not close to the data distribution, the classification problem would be too easy and would not require p_θ to learn much about the data. One choice of $q(\mathbf{x})$ is normalizing flow models [Gao et al., 2020].

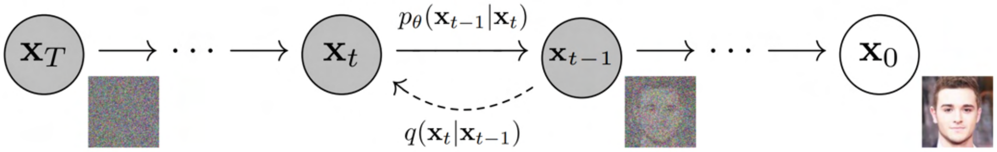

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 2.3: Illustration of denoising diffusion models. The forward process, denoted by q, is a Markov chain of diffusion steps to slowly add random noise to data. The reverse process, denoted by p_θ, is a Markov chain that is learned to reverse the forward diffusion process and recover clean data from noise.

2.4 Denoising Diffusion Models

Generative models that have been reviewed in this chapter share a common feature that their generative process is a black box. VAEs and normalizing flows generate samples by first sampling $\mathbf{z}$ from a noise distribution, and then $\mathbf{z}$ is passed through a neural network to produce samples. EBMs generate samples by iteratively running Langevin dynamics, however, there is no clear interpretation behind each step of LD. This section introduces denoising diffusion models, which has a distinctive property that the generation process corresponds to an explicit inversion of a so-called diffusion process that gradually perturbs data into noise. More specifically, diffusion models define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse the diffusion process to construct desired data samples from the noise. Therefore, the generation process of diffusion models can be clearly interpreted as gradually denoising noisy observations into clean data. An illustration for the denoising diffusion model is presented in Figure 2.3.

Diffusion models are inspired by non-equilibrium thermodynamics [Sohl-Dickstein et al., 2015]. The idea of using a Markov chain to convert one distribution into another gradually was adopted earlier in statistical physics [Jarzynski, 1997] and sequential Monte Carlo [Neal, 2001]. Sohl-Dickstein et al. [2015] first borrowed the idea for generative modeling and designed models with a generative Markov chain which converts a simple known distribution (e.g., a Gaussian) into the distribution using a diffusion process. Later, Ho et al. [2020] improved and extended the model to make it capable of generating high-quality samples.

Ho et al. [2020] also showed that a specific parameterization of diffusion models reveals an equivalence with denoising score matching over multiple noise levels [Song and Ermon, 2019]. In their seminal paper, Song et al. [2021b] extend the discrete time diffusion process used in denoising diffusion models to a continuous time diffusion process modeled by stochastic differential equations and provide a unified framework to analyze denoising diffusion models and score-based models.

Diffusion models have achieved state-of-the-art performance on many downstream tasks and applications. For example, diffusion models beat GANs on the challenging task of conditional generation on ImageNet dataset [Dhariwal and Nichol, 2021, Ho et al., 2022]. They also achieved impressive performance on audio synthesis [Chen et al., 2020a, Kong et al., 2021, Popov et al., 2021], 3D shape generation [Cai et al., 2020] and music generation [Mittal et al., 2021]. Diffusion models can also be combined with other generative models such as VAEs [Vahdat et al., 2021] and EBMs [Gao et al., 2021]. In addition, diffusion models can be a powerful tool to solve a variety of inverse problems including super resolution [Saharia et al., 2021b], image inpainting [Saharia et al., 2021b, Lugmayr et al., 2022] and medical image reconstruction [Song et al., 2022, Jalal et al., 2021].

In what follows, we will provide an overview of denoising diffusion models. We will start with the formulation of the model, followed by the process of training and sampling from the model. Finally, we will discuss the extension of denoising diffusion models to continuous time space.

2.4.1 Formulation of Diffusion Models

As discussed earlier, diffusion models have a forward process that gradually perturbs data into noise. Specifically, given a data point sampled from the real data distribution $\mathbf{x}_0 \sim q(\mathbf{x})$, the forward process adds small amount of Gaussian noise to the sample in each of T steps, producing a sequence of noisy samples $\mathbf{x}_1, \cdots, \mathbf{x}_T$. Each step is assumed to be a conditional

Gaussian distribution

$$q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I\right), \tag{2.44}$$

where the variance schedule $\beta_t, t = 1, \cdots, T$ can be seen as step sizes. The variance schedule is carefully chosen so that $\mathbf{x}_T$ does not contain any information about $\mathbf{x}_0$ and is a sample from white noise distribution. Note that the process is a Markov chain, as $\mathbf{x}_t$ only depends on $\mathbf{x}_{t-1}$ and no previous steps. The joint distribution of $\mathbf{x}_{1:T}$ given the clean data $\mathbf{x}_0$ can be written as

$$q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right) = \prod_{t=1}^{T} q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right). \tag{2.45}$$

One nice property of the forward process is that given initial clean data $\mathbf{x}_0$, we can sample $\mathbf{x}_t$ for arbitrary time step t in closed form. Denote $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$, we have

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-2} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\epsilon}_{t-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon,
\end{aligned}$$

where $\epsilon_{t-1}, \epsilon_{t-2}, \bar{\epsilon}_{t-2}$ and ϵ are all sampled from $\mathcal{N}(0, I)$. The third equation uses that fact that if X and Y are two independent random variables with $X \sim N\left(\mu_X, \sigma_X^2\right)$ and $Y \sim N\left(\mu_Y, \sigma_Y^2\right)$, then $Z = X + Y$ is distributed as $Z \sim N\left(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2\right)$. Therefore, we have

$$q\left(\mathbf{x}_t|\mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \left(1-\bar{\alpha}_t\right) I\right). \tag{2.46}$$

The reverse process (or backward process) is another Markov chain whose goal is to reverse the forward process. Note that if we can sample from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, we can recreate a true sample from a Gaussian noise input $\mathbf{x}_T \sim \mathcal{N}(0, I)$. However, we cannot directly sample from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, as we cannot easily go from higher entropy to lower entropy. The reverse process aims to approximate these conditional probabilities $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ with a parameterized model p_θ. We assume the model also has the form of Gaussian conditional distributions:

$$p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \mu_\theta\left(\mathbf{x}_t, t\right), \Sigma_\theta\left(\mathbf{x}_t, t\right)\right), \tag{2.47}$$

and

$$p_\theta\left(\mathbf{x}_{0:T}\right) = p\left(\mathbf{x}_T\right) \prod_{t=1}^{T} p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right). \tag{2.48}$$

Besides the conditional distribution in the forward and reverse process, there is also an important distribution that will be used later: $q\left(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0\right)$, the posterior distribution given initial point $\mathbf{x}_0$. It is noteworthy that this reverse conditional probability is tractable when conditioned on $\mathbf{x}_0$. To derive the posterior, we first apply the Bayes rule:

$$\begin{aligned} q\left(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0\right) &= \frac{q\left(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0\right) q\left(\mathbf{x}_{t-1}|\mathbf{x}_0\right)}{q\left(\mathbf{x}_t|\mathbf{x}_0\right)} \\ &= \frac{q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right) q\left(\mathbf{x}_{t-1}|\mathbf{x}_0\right)}{q\left(\mathbf{x}_t|\mathbf{x}_0\right)}, \end{aligned} \tag{2.49}$$

where the second equation follows from the Markov property of the forward process. Since all three terms in Equation 2.49 are Gaussians, the posterior $q\left(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0\right)$ is also a Gaussian distribution, and it can be written as

$$q\left(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\mu}\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t I\right). \tag{2.50}$$

By plugging in the expression for $q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)$ in Equation 2.44 and the expression of $q\left(\mathbf{x}_t|\mathbf{x}_0\right)$ and $q\left(\mathbf{x}_{t-1}|\mathbf{x}_0\right)$ in Equation 2.46 into Equation 2.50, together with the fact that $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\mathbf{z}_t\right)$, we can obtain

$$\tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \tag{2.51}$$

2.4.2 Training Denoising Diffusion Models

Given the parameterized reverse process in Equation 2.47, the training objective is to find means $\mu_\theta\left(\mathbf{x}_t, t\right)$ and variances $\Sigma_\theta\left(\mathbf{x}_t, t\right)$ that maximize the data likelihood under the model:

$$p_\theta\left(\mathbf{x}_0\right) = \int p_\theta\left(\mathbf{x}_{0:T}\right) d\mathbf{x}_{1:T},$$

where $p_\theta\left(\mathbf{x}_{0:T}\right)$ is given in Equation 2.48. We can treat $\mathbf{x}_{1:T}$ as latent variables, and obtain the following variational lower bound:

$$\begin{aligned}
\log p_\theta\left(\mathbf{x}_0\right) &\geq \log p_\theta\left(\mathbf{x}_0\right) - D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right)\right) \\
&= \log p_\theta\left(\mathbf{x}_0\right) - \mathbb{E}_{\mathbf{x}_{1:T}\sim q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right)}\left[\log \frac{q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:T}\right)/p_\theta\left(\mathbf{x}_0\right)}\right] \\
&= \log p_\theta\left(\mathbf{x}_0\right) - \mathbb{E}_q\left[\log \frac{q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:T}\right)} + \log p_\theta\left(\mathbf{x}_0\right)\right] \\
&= \mathbb{E}_q\left[-\log \frac{q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{0:T}\right)}\right].
\end{aligned} \tag{2.52}$$

Therefore, we obtain the training objective:

$$\min_\theta \quad L_\theta = \mathbb{E}_q\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right)}\right] \tag{2.53}$$

The objective can be further rewritten to be a combination of several KL-divergence and entropy terms, as discussed in Appendix A in Ho et al. [2020]. One way to rewrite L_θ is

$$\begin{aligned}
L &= \mathbb{E}_q\left[-\log p\left(\mathbf{x}_T\right)-\sum_{t\geq 1}\log\frac{p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}{q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)}\right]\\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}_T\right)-\sum_{t\geq 1}\log\frac{p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}{q\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}\cdot\frac{q\left(\mathbf{x}_{t-1}\right)}{q\left(\mathbf{x}_t\right)}\right]\\
&= \mathbb{E}_q\left[-\log\frac{p\left(\mathbf{x}_T\right)}{q\left(\mathbf{x}_T\right)}-\sum_{t\geq 1}\log\frac{p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}{q\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}-\log q\left(\mathbf{x}_0\right)\right]\\
&= D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T\right)\|p\left(\mathbf{x}_T\right)\right)+\mathbb{E}_q\left[\sum_{t\geq 1}D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)\|p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)\right)\right]+H\left(\mathbf{x}_0\right). \qquad (2.54)
\end{aligned}$$

Note that in the last equation, $D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T\right)\|p\left(\mathbf{x}_T\right)\right)$ is constant during training, as we assume both $q\left(\mathbf{x}_T\right)$ and $p\left(\mathbf{x}_T\right)$ are white noise distributions. In addition, $H\left(\mathbf{x}_0\right)$ is independent of the parameter θ. Therefore, the objective is to minimize

$$\mathbb{E}_q\left[\sum_{t\geq 1}D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)\|p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)\right)\right],$$

which exactly corresponds to the goal of the denoising diffusion model, which is that we want our parameterized reverse process $p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)$ to match the reverse of the forward process $q\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)$. However, since we never know the true reverse process $q\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)$, we cannot directly minimize Equation 2.54 to train the model.

Sohl-Dickstein et al. [2015] show that L_θ can be written in an alternative form with tractable distributions. We include the derivation here.

$$
\begin{aligned}
L &= \mathbb{E}_q\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right)}\right] \\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}_T\right) - \sum_{t\geq 1} \log \frac{p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}{q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)}\right] \\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}_T\right) - \sum_{t\geq 2} \log \frac{p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}{q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)} - \log \frac{p_\theta\left(\mathbf{x}_0|\mathbf{x}_1\right)}{q\left(\mathbf{x}_1|\mathbf{x}_0\right)}\right] \\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}_T\right) - \sum_{t\geq 2} \log \frac{p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}{q\left(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0\right)} \cdot \frac{q\left(\mathbf{x}_{t-1}|\mathbf{x}_0\right)}{q\left(\mathbf{x}_t|\mathbf{x}_0\right)} - \log \frac{p_\theta\left(\mathbf{x}_0|\mathbf{x}_1\right)}{q\left(\mathbf{x}_1|\mathbf{x}_0\right)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta\left(\mathbf{x}_T\right) + \sum_{t=2}^{T} \log \frac{q\left(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)} + \log \frac{q\left(\mathbf{x}_T|\mathbf{x}_0\right)}{q\left(\mathbf{x}_1|\mathbf{x}_0\right)} + \log \frac{q\left(\mathbf{x}_1|\mathbf{x}_0\right)}{p_\theta\left(\mathbf{x}_0|\mathbf{x}_1\right)}\right] \\
&= \mathbb{E}_q\left[-\log \frac{p\left(\mathbf{x}_T\right)}{q\left(\mathbf{x}_T|\mathbf{x}_0\right)} - \sum_{t\geq 2} \log \frac{p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)}{q\left(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0\right)} - \log p_\theta\left(\mathbf{x}_0|\mathbf{x}_1\right)\right] \\
&= \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t,\mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \qquad (2.55) \\
&\quad - \underbrace{\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}] \qquad (2.56)
\end{aligned}
$$

Again, the first term is constant during training. The expression for L_θ in Equation 2.55 rewrites $q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)$ as

$$
q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right) = q\left(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0\right) = \frac{q\left(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0\right) q\left(\mathbf{x}_t|\mathbf{x}_0\right)}{q\left(\mathbf{x}_{t-1}|\mathbf{x}_0\right)},
$$

where the first equality follows from the Markov property. Therefore, we can express the distribution with the posterior distribution $q\left(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0\right)$ in Equation 2.50, which is a tractable Gaussian distribution. Note that each KL-divergence term in Equation 2.55 compares two Gaussian distributions and therefore they can be computed in closed form.

Next, we discuss how to parameterize the means $\mu_\theta(\mathbf{x}_t, t)$ and variance $\Sigma_\theta(\mathbf{x}_t, t)$ of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Ho et al. [2020] set $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 I$, where σ_t^2s are untrained time dependent constants. σ_t^2 is set to be either β_t, the variance in the forward process or $\tilde{\beta}_t$, the variance of the posterior distribution. In contrast, [Nichol and Dhariwal, 2021] propose to learn $\Sigma_\theta(\mathbf{x}_t, t)$ as an interpolation between β_t and $\tilde{\beta}_t$. For simplicity, we follow the setting of Ho et al. [2020].

For the means $\mu_\theta(\mathbf{x}_t, t)$, we want them to approximate the posterior mean $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$. Naively, we can parametrize $\mu_\theta(\mathbf{x}_t, t)$ with a neural network and minimize

$$L_{t-1} = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\left\|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\right\|^2\right] + C \tag{2.57}$$

for each time step. However, Ho et al. [2020] observe that instead of learning $\mu_\theta(\mathbf{x}_t, t)$ to directly predict the posterior means, parameterizing the network with some transformations leads to better results. Firstly, from Equation 2.46, we can express $\mathbf{x}_0$ in terms of $\mathbf{x}_t$ and $\mathbf{z}_t$:

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\mathbf{z}_t\right). \tag{2.58}$$

Plug in the expression of $\mathbf{x}_0$ into the expression of $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$ in Equation 2.51, we have

$$\begin{aligned}\tilde{\mu}_t &= \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\mathbf{z}_t\right) \\ &= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\mathbf{z}_t\right).\end{aligned} \tag{2.59}$$

As a result, given $\mathbf{x}_t$, we only need the noise $\mathbf{z}_t$ to obtain the posterior mean. Since $\mathbf{x}_t$ is available during training, we can use a network $\mathbf{z}_\theta$ to parametrize the noise, and hence we have

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\mathbf{z}_\theta(\mathbf{x}_t, t)\right). \tag{2.60}$$

Plug in the expression of $\tilde{\mu}_t$ in Equation 2.59 and the expression of $\mu_\theta(\mathbf{x}_t, t)$ in Equation 2.60 into the objective in Equation 2.57, we obtain the final training objective

$$\mathbb{E}_{\mathbf{x}_0, \mathbf{z}}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|\mathbf{z}_t - \mathbf{z}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\mathbf{z}_t, t\right)\right\|^2\right]. \tag{2.61}$$

The training can be interpreted as "noise prediction": given noisy observation $\mathbf{x}_t$ obtained from perturbing clean data $\mathbf{x}_0$, the network takes $\mathbf{x}_t$ and t as inputs, and tries to predict the noise $\mathbf{z}_t$ that perturbs $\mathbf{x}_0$ into $\mathbf{x}_t$.

Giving a well-trained noise prediction network $\mathbf{z}_\theta(\mathbf{x}_t, t)$, we can sample $\mathbf{x}_{t-1} \sim p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ by computing

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\mathbf{z}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}, \tag{2.62}$$

where $\mathbf{z} \sim \mathcal{N}(0, I)$. Starting from a white noise $\mathbf{x}_T$ and iteratively running Equation 2.62, we can obtain a clean sample $\mathbf{x}_0$.

2.4.3 Extension of Diffusion Models to Continuous Time

One important hyper-parameter of diffusion models is T, the number of time steps. It is observed that a large number of steps is needed for high sample quality. A larger number of steps means a smaller step size, which in turn makes the modeling more accurate as each time conditioned model only needs to predict a small denoising step. When the step size approaches 0, we obtain the continuous-time diffusion process, which can be described by the dynamics of a stochastic differential equation (SDE).

Song et al. [2021b] extend diffusion models to continuous time using the tool of SDEs, and their models are called score SDE models. In particular, the diffusion process can be modeled as the solution to an Ito SDE

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}, \tag{2.63}$$

where $\mathbf{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is a vector-valued function called the drift coefficient, $g(t) \in \mathbb{R}$ is a real-valued function called the diffusion coefficient, and $\mathrm{d}\mathbf{w}$ can be viewed as infinitesimal white noise. Similar to the discrete case, let $p_0(\mathbf{x}) = p(\mathbf{x})$ as the data distribution, after perturbing $p_0(\mathbf{x})$ with the stochastic process for a sufficiently long time T, $p_T(\mathbf{x})$ becomes a tractable noise distribution.

We want to sample $\mathbf{x}_T$ from the noise distribution $p_T(\mathbf{x})$, and reverse the process to obtain samples from $\mathbf{x}_0$. Remarkably, given the forward diffusion process in Equation 2.63, we have an associated reverse process which is also also a diffusion process, running backward in time and given by the reverse-time SDE [Anderson, 1982]

$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \mathrm{d}t + g(t)\mathrm{d}\mathbf{w}. \tag{2.64}$$

Therefore, if we can estimate the score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, we can simulate Equation 2.64 to generate samples. In order to estimate $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, Song et al. [2021b] adopt score matching and propose to train a time-dependent score-based model $s_\theta(\mathbf{x}, t)$ such that $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$. The training objective is

$$\mathbb{E}_{t \in \mathcal{U}(0,T)} \mathbb{E}_{p_t(\mathbf{x})} \left[\lambda(t) \left\| \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x}, t) \right\|_2^2\right], \tag{2.65}$$

where $\mathcal{U}(0, T)$ is the uniform distribution over the time interval $[0, T]$ and λ is a positive weighting function. Due to the intractability of $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, the actual training objective, as done in Section 2.3.3, uses denoising score matching:

$$\mathbb{E}_{t\in\mathcal{U}(0,T)}\mathbb{E}_{\mathbf{x}(0)\sim p_0(\mathbf{x})}\mathbb{E}_{\mathbf{x}(t)\sim p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))}\left[\lambda(t)\left\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t),t)-\nabla_{\mathbf{x}(t)}\log p_{0t}(\mathbf{x}(t)\mid\mathbf{x}(0))\right\|_2^2\right]. \tag{2.66}$$

After training the time-dependent score matching model, we can simulate the reverse SDE in Equation 2.64 with numerical SDE solvers. For example, the simplest numerical SDE solver is the Euler-Maruyama discretization.

One crucial component of score SDE models is the design of the forward diffusion process. There are multiple ways to design the process. For example, the discrete diffusion process in Equation 2.44 corresponds to the discretization of the following SDE:

$$\mathrm{d}\mathbf{x}=-\frac{1}{2}\beta(t)\mathbf{x}\mathrm{d}t+\sqrt{\beta(t)}\mathrm{d}\mathbf{w}. \tag{2.67}$$

The noise conditional score matching model introduced in Song and Ermon [2019] corresponds to the following SDE:

$$\mathrm{d}\mathbf{x}=\sqrt{\frac{\mathrm{d}\left[\sigma^2(t)\right]}{\mathrm{d}t}}\,\mathrm{d}\mathbf{w}. \tag{2.68}$$

Therefore, Song et al. [2021b] unify denoising diffusion models and denoising score matching models under the framework of score SDE models.

2.4.4 Limitations of Diffusion Models

The most significant limitation of diffusion models is their slow sampling speed. It is very slow to generate samples from a denoising diffusion model by iteratively running Equation 2.62, as T can be up to one or a few thousand steps. Each step involves a network evaluation of $z_\theta(\mathbf{x}_t,t)$, which is expensive. For example, Song et al. [2020a] observe that it takes around

20 hours to sample $50k$ images of size 32×32 from a diffusion model on an Nvidia 2080 Ti GPU, while it takes less than a minute to do so from a GAN. The slow sampling speed makes diffusion models hard to apply to tasks that require real time synthesis, such as interactive image editing. In continuous time space, score SDE models do not have an explicit definition of the time step. However, sampling from them requires numerically solving the reverse SDE by discretization, and each discretization step still requires a network evaluation. To solve the SDE accurately, a large number of discretization steps is needed.

One way to reduce the sampling time is to run a strided sampling schedule, where several steps in the forward process are combined into a single step in the reverse process, with appropriately adjusted mean and variance parameters. In the score SDE model, this corresponds to using a coarse discretization scheme to solve the reverse SDE. However, this approach leads to significantly worse sample quality.

We will discuss other methods that aim to improve the sampling speed of diffusion models in Section 6.

2.5 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] are perhaps the most successful and widely used deep generative models. They have shown great results in many generative tasks that replicate the real-world rich content such as images [Brock et al., 2018, Karras et al., 2017, 2019, 2021], natural language [Subramanian et al., 2017], speech [Kong et al., 2020] and music [Yang et al., 2017]. We postponed the introduction of GANs until the end of this chapter because the majority of this dissertation focuses on likelihood-based generative models. The high-level idea behind GANs has been discussed in Section 1.2.2 as part of the density ratio approach for training generative models. In what follows, we will give a more in-depth introduction to GANs.

A GAN consists of two models:

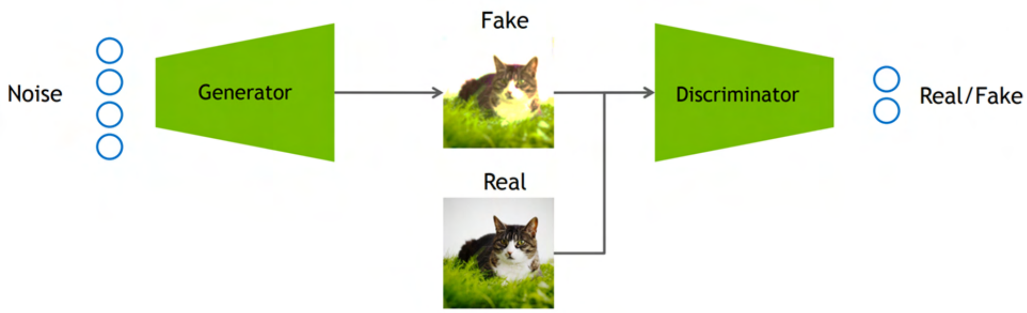


Figure 2.4: Illustration of GANs. The discriminator tries to distinguish real and fake samples, while the generator generates samples that can fool the discriminator.

- A discriminator D estimates the probability of a given sample coming from the real data distribution. It works as a critic and is optimized to distinguish fake samples from the real ones.

- A generator G outputs synthetic samples given a noise variable input $\mathbf{z}$. It is trained to capture the real data distribution so that the samples generated by the G can be as real as possible, or in other words, can fool the discriminator to return a high probability of being real.

These two models compete against each other during the training process: the generator G is trying hard to confuse the discriminator, while the critic model D is trying hard not to be confused. This zero-sum game between the two models motivates both to improve their functionalities. Figure 2.4 provides a illustration of GANs.

At the beginning of Chapter 2, we mentioned that, unlike explicit generative models that are discussed in previous sections, GANs belong to the category of implicit generative models. Here we will provide a detailed explanation of implicit generative models. The distinctive property of implicit generative models is that they get rid of prescribed distributions. Some explicit models, such as VAEs, share some similarities with GANs in that they both have a decoder structure, and the decoder takes latent variables $\mathbf{z}$ as its input. However, a VAE's decoder has a prescribed distribution, as we let $p_\theta(\mathbf{x}|\mathbf{z})$ be a Gaussian. In contrast, a GAN's decoder returns only a single point. In other words, the conditional distribution is a Dirac's delta

$$p_\theta(\mathbf{x} \mid \mathbf{z}) = \delta\left(\mathbf{x} - G_\theta(\mathbf{z})\right), \tag{2.69}$$

where G_θ is the decoder network. This is equivalent to say that instead of a Gaussian (i.e., a mean and a variance), G_θ outputs the mean only. It is still possible to obtain the marginal distribution $p_\theta(\mathbf{x})$, as

$$p_\theta(\mathbf{x}) = \int \delta\left(\mathbf{x} - G_\theta(\mathbf{z})\right) p(\mathbf{z}) \mathrm{d}\mathbf{z}. \tag{2.70}$$

The marginal distribution is an infinite mixture of delta peaks, and a single $\mathbf{z}$ in the latent space corresponds to a single $\mathbf{x}$ in the data space. Imagine that for every $\mathbf{z}$, we plot $G_\theta(\mathbf{z})$ in the data space, then the data space will be covered by infinitely many points, and some regions will be denser than the others. The resulting density still makes $p_\theta(\mathbf{x})$ a valid distribution, but we do not know the exact form of $p_\theta(\mathbf{x})$. This kind of distribution modeling is known as implicit modeling.

Explicit models with a prescribed distribution can be trained by maximum likelihood (minimizing the KL divergence), however, the Dirac's delta distribution in implicit models is ill-defined and cannot be used in many probability measures, including the KL divergence. Luckily, we do not need to stick to the KL divergence. Instead, we can use other metrics that look at a set of points (i.e., distributions represented by a set of points), including kernel-based Minimum Mean Discrepancy (MMD) [Gretton et al., 2006] and other divergences [Van Erven and Harremos, 2014]. GANs use adversarial training as a surrogate to minimize the distance between (implicit) $p_\theta(\mathbf{x})$ and true data distribution $p(\mathbf{x})$, and the exact type of distance depends on the particular choice of GAN loss.

In what follows, we will introduce the training objective of GANs and try to formulate

GAN training in the context of divergence minimization. We will also discuss the challenges of GAN training, as well as some important GAN variants.

2.5.1 Understanding the Training of GANs

The discriminator $D_\phi : \mathcal{X} \rightarrow [0,1]$ takes an object x in the data space and returns a probability whether it is real. The generator $G_\theta : \mathcal{Z} \rightarrow \mathcal{X}$ takes a noise $\mathbf{z}$ and turns it into an object $\mathbf{x}$. Since the discriminator can be seen as a classifier, we can optimize the binary cross-entropy loss function in the following form

$$\begin{aligned} \mathcal{L}_{\phi,\theta} &= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\log D_\phi(\mathbf{x})\right] + \mathbb{E}_{\hat{\mathbf{x}}\sim p_\theta(\mathbf{x})}\left[\log(1 - D_\phi(\hat{\mathbf{x}}))\right] &(2.71)\\ &= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log D_\phi(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - D_\phi(G_\theta(\mathbf{z})))\right]. &(2.72)\end{aligned}$$

On one hand, we want to make sure the discriminator D_ϕ assigns high probability of being real over samples from the data distribution by maximizing $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\log D_\phi(\mathbf{x})\right]$ w.r.t ϕ. Meanwhile, given a fake sample $G_\theta(\mathbf{z})$ where $\mathbf{z} \sim p(\mathbf{z})$ the discriminator is expected to output a probability $D_\phi\left(G_\theta(\mathbf{z})\right)$ close to zero by maximizing $\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - D_\phi(G_\theta(\mathbf{z})))\right]$ w.r.t ϕ. On the other hand, the generator is trained to increase the chances of D_ϕ producing a fake example with a high probability of looking real, thus to minimize $\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - D_\phi(G_\theta(\mathbf{z})))\right]$ w.r.t. θ. When combining both aspects together, D_ϕ and G_θ are playing a minimax game in which we should optimize the following loss function

$$\min_\theta \max_\phi \mathcal{L}_{\phi,\theta} = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log D_\phi(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - D_\phi(G_\theta(\mathbf{z})))\right]. \qquad (2.73)$$

Optimal discriminator: To better understand the training of GANs, we derive the optimal solution of the discriminator when the generator is fixed. The training objective of the discriminator corresponds to finding the optimal Bayesian classifier that distinguishes samples from $p_{\text{data}}(\mathbf{x})$ and $p_\theta(\mathbf{x})$, given that the prior is $\frac{1}{2}$ for each source. In this case, when the generator is fixed, the optimal classifier value is

$$D_{\phi}^{*}(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\theta}(\mathbf{x})} \in [0, 1].$$

Furthermore, we can derive the optimal value of the objective function when the generator and discriminator are both optimal. We first rewrite the objective in Equation 2.71 in the integral form (assuming we have the expression of $p_{\text{data}}(\mathbf{x})$ and $p_{\theta}(\mathbf{x})$, although we do not):

$$D_{\phi}^{*}(\mathbf{x}) = \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) \log(D_{\phi}(\mathbf{x})) + p_{\theta}(\mathbf{x}) \log(1 - D_{\phi}(\mathbf{x})) d\mathbf{x}. \tag{2.74}$$

When the generator is trained to its optimal, $p_{\text{data}}(\mathbf{x})$ is very close to $p_{\theta}(\mathbf{x})$, and in that case, $D_{\phi}^{*}(\mathbf{x}) \approx \frac{1}{2}$, i.e., the discriminator cannot distinguish between real and fake samples. In this case, given the optimal value of $D_{\phi}^{*}(\mathbf{x})$, the resulting optimal value of the objective is

$$\begin{aligned} &\int_{\mathbf{x}} \left(p_{\text{data}}(\mathbf{x}) \log\left(D_{\phi}^{*}(\mathbf{x})\right) + p_{\theta}(\mathbf{x}) \log\left(1 - D_{\phi}^{*}(\mathbf{x})\right)\right) d\mathbf{x} \\ &= \log \frac{1}{2} \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) d\mathbf{x} + \log \frac{1}{2} \int_{\mathbf{x}} p_{\theta}(\mathbf{x}) d\mathbf{x} \\ &= -2 \log 2. \end{aligned} \tag{2.75}$$

For convenience, we denote the objective in Equation 2.74 with the optimal discriminator value in Equation 2.73 as $V(G, D^{*})$.

Connection to Jensen-Shannon divergence minimization: We also show that when the discriminator is optimal, the loss function can be expressed in terms of a Jensen-Shannon divergence term. First we will define Jensen-Shannon (JS) divergence. Recall the definition of KL divergence:

$$D_{\text{KL}}(p(\mathbf{x}) \| q(\mathbf{x})) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x}, \tag{2.76}$$

and note that KL divergence is not symmetric. JS divergence is a symmetric measure of similarity between two probability distributions, and it is defined by

$$D_{JS}(p\|q) = \frac{1}{2}D_{\mathrm{KL}}\left(p\|\frac{p+q}{2}\right) + \frac{1}{2}D_{\mathrm{KL}}\left(q\|\frac{p+q}{2}\right). \tag{2.77}$$

Similar to KL divergence, $D_{JS}(p\|q)$ if and only if $p = q$. With this definition in hand, the JS divergence between $p_{\mathrm{data}}(\mathbf{x})$ and $p_\theta(\mathbf{x})$ can be computed as

$$\begin{aligned} D_{JS}\left(p_{\mathrm{data}}\|p_\theta\right) =& \frac{1}{2}D_{KL}\left(p_{\mathrm{data}}\|\frac{p_{\mathrm{data}}+p_\theta}{2}\right) + \frac{1}{2}D_{KL}\left(p_\theta\|\frac{p_{\mathrm{data}}+p_\theta}{2}\right) \\ =& \frac{1}{2}\left(\log 2 + \int_x p_{\mathrm{data}}(x)\log\frac{p_{\mathrm{data}}(x)}{p_{\mathrm{data}}+p_\theta(x)}dx\right) + \\ & \frac{1}{2}\left(\log 2 + \int_x p_\theta(x)\log\frac{p_\theta(x)}{p_{\mathrm{data}}+p_\theta(x)}dx\right) \\ =& \frac{1}{2}\left(\log 4 + V\left(G, D^*\right)\right), \end{aligned} \tag{2.78}$$

where the last line follows from the definition of $V\left(G, D^*\right)$. Therefore,

$$V\left(G, D^*\right) = 2D_{JS}\left(p_r\|p_g\right) - 2\log 2. \tag{2.79}$$

Essentially the training objective of GANs quantifies the similarity between the $p_\theta(\mathbf{x})$ and the data distribution by JS divergence when the discriminator is optimal. In addition, the best generator will minimize the JS divergence to 0, leading to the optimal loss value $-2\log 2$ in the equilibrium.

2.5.2 Challenges in Training GANs

Although GANs have had great success in many domains, there are many challenges in training GANs. The training process is unstable, and there are multiple possible reasons. We will discuss several of these, in particular, the biggest issue with GANs: the mode collapse.

Challenges of min-max optimization: In GAN training, two models are trained simultaneously to find a Nash equilibrium of a two-player non-cooperative game. Unlike most machine learning problems whose objective is a simple minimization or maximization, the objective of GANs is a min-max problem. Each model updates independently with no respect to the other player in the game with the gradient descent-ascent (GDA) technique [Lin et al., 2020]. Updating the gradient of both models independently concurrently cannot guarantee convergence to a Nash equilibrium, and there are many counter-examples with simple forms of the objective. The non-convergence may lead to instability during training, as discussed by [Salimans et al., 2016]. Min-max optimization (and in particular with non convex-concave objective) is an active research direction in optimization theory [Daskalakis and Panageas, 2018, Farnia and Ozdaglar, 2020, Wang et al., 2019]. Still, so far, there is no principled way to solve the problem.

Imbalance between discriminator and generator: It is important to keep a balance between the discriminator and the generator, and it would be problematic if the discriminator becomes too strong. Imagine a perfect discriminator which can always classify real and fake samples. Then we have $D(\mathbf{x}) = 1$ for all $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$ and $D(\mathbf{x}) = 0$ for all $\mathbf{x} \sim p_{\theta}(\mathbf{x})$, the loss function falls to zero and we end up with no gradient to update the loss during learning iterations. Therefore, we have to keep a subtle balance: if the discriminator is too weak, the generator does not have accurate feedback, and the loss function cannot reflect the real or fake probability; If the discriminator does a great job, the gradient of the loss function drops down to close to zero, and the learning gets stuck. Note that it is always easier to do binary classification than generate new samples, so oftentimes, the discriminator is too strong. As a result, a variety of regularization techniques that restrict the power of the discriminator are proposed [Mescheder et al., 2018, Zhang et al., 2019, Kurach et al., 2019]. However, it is still difficult to keep the balance between two losses.

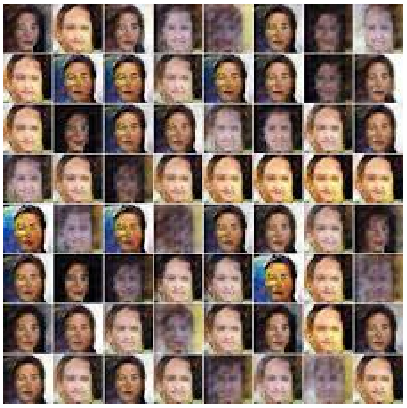


Figure 2.5: An example of mode collapse. The generator produce repeated patterns.

Mode collapse: During training, the generator may collapse to a setting where it always produces the same outputs. This is a common failure case for GANs, commonly referred to as mode collapse. Even though the generator learns to fool the discriminator, it fails to represent the complex data distribution and gets stuck in a small space with extremely low variety. See Figure 2.5 for an example.

The cause of mode collapse is not completely clear. One explanation is that the objective function of training GANs has a reverse KL divergence component (part of the JS divergence). Minimizing the reverse KL divergence $D_{\mathrm{KL}}\left(p_\theta \| p_{\mathrm{data}}\right)$ will encourage the model to cover some of the modes in data distribution since there is no penalty for not covering p_{data} when there is no probability mass under p_θ. In contrast, maximum likelihood training, which minimizes the forward KL divergence $D_{\mathrm{KL}}\left(p_{\mathrm{data}} \| p_\theta\right)$, will penalize any mismatch where $p_{\mathrm{data}}(\mathbf{x})$ is not zero, and hence the model is encouraged to cover all the modes of $p_{\mathrm{data}}(\mathbf{x})$. As a result, models trained by maximum likelihood tend to have better mode coverage than GANs. Several methods are proposed to alleviate the mode collapse issue of GANs [Srivastava et al., 2017, Dieng et al., 2019], but mode collapse is still one of the biggest challenges of GAN training.

2.5.3 Important GAN variants

Because of the popularity of GANs, there are multiple attempts to further improve the GAN model. In this section, we introduce two important variants of GANs. The first one, Wasserstein GAN [Arjovsky et al., 2017], proposes a new training objective based on the Wasserstein distance between two distributions. The second one, Style GAN [Karras et al., 2019], focuses on architectural design and proposes several important modifications to the implementation of GANs, which lead to state-of-the-art sample quality.

Wasserstein GAN: The optimization of the original GAN objective is difficult, due to generator's vanishing gradient problem caused by a powerful discriminator, as described in the previous section. [Goodfellow et al., 2014] propose a slightly different generator loss:

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[\log D(G_{\theta}(\mathbf{z}))],$$

which is called Non-Saturating GAN loss. While this loss function can address the vanishing gradient problem, Arjovsky and Bottou [2017] illustrates that it has a large variance of gradients that makes the training unstable.

To obtain a GAN objective that is easier to optimize, Arjovsky et al. [2017] propose Wasserstein GAN (W-GAN), which is based on the concept of Wasserstein distance (also called the Earth Mover distance). Wasserstein distance between two distributions p and q is the minimum cost of transporting mass in converting distribution q to distribution p. The exact form of Wasserstein distance depends on the choice of cost. For example, the commonly used Wasserstein-1 distance is defined by

$$W(p, q) = \inf_{\gamma\in\Pi(p,q)} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}[||x - y||], \tag{2.80}$$

where $\Pi(p, q)$ denotes the set of all joint distributions $\gamma(\mathbf{x}, \mathbf{y})$ whose marginals are p and q, respectively. By the Kantorovich-Rubinstein duality [Villani, 2009], the Wasserstein-1 distance can be equivalently written in the following form:

$$W(p, q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim q(\mathbf{x})}[f(\mathbf{x})], \tag{2.81}$$

where the constraint on f says that f must be a 1-Lipschitz function.

In W-GAN, the Wasserstein distance is computed between $p_{\text{data}}(\mathbf{x})$ and $p_\theta(\mathbf{x})$. The core idea behind W-GAN is to train a generator that minimizes the Wasserstein distance between p_{data} and p_θ. Since computing the distance itself can be formulated as a maximization problem, the whole objective is a min-max problem, and hence the model is a GAN. The Lipschitz function f in Equation 2.81 can be seen as a discriminator, although it is not a direct critic of telling the fake samples apart from the real ones, and it does not need to output a probability (which means that the output does not need to be in $[0, 1]$). For a better connection with previous GAN models, we denote the function by D_ϕ. The objective for training D_ϕ is

$$\max_\phi \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[D_\phi(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[D_\phi(G_\theta(\mathbf{z}))], \tag{2.82}$$

while keeping the constraint that $\|D_\phi\|_L \leq 1$. Optimizing overall Lipschitz functions is impossible, and the solution is to parameterize them through a deep neural network, constrained to have a Lipschitz constant less than 1. Optimizing this objective corresponds to computing the Wasserstein distance. The objective for training the generator is to minimize the Wasserstein distance, which is done by minimizing the same objective w.r.t θ:

$$\min_\theta -\mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[D_\phi(G_\theta(\mathbf{z}))]. \tag{2.83}$$

One subtle point in training W-GAN is to maintain the Lipschitz constant of D_ϕ. In the original paper of Arjovsky et al. [2017], this is implemented by gradient clipping. Alternative methods such as gradient penalty [Gulrajani et al., 2017] and spectral normalization [Miyato et al., 2018] are proposed later. It is observed that the training of W-GAN is more stable, and the mode collapse issue is greatly alleviated.

It is also noteworthy that in W-GAN, the gradient of the objective for training D_ϕ is very similar to Equation 2.38, the gradient estimate of the maximum likelihood objective for training EBMs. Such a connection will be utilized in Section 5.

Style GAN: While most of the GAN research focuses on optimization, such as new regularization terms [Zhang et al., 2019, Miyato et al., 2018] and new training losses [Arjovsky and Bottou, 2017, Mao et al., 2017], Style GAN [Karras et al., 2019] focuses on the design of the network. Previous GANs had trouble generating high-quality large images (e.g., 1024×1024), and in particular, they may fail to capture the details of high-resolution images. Style GAN re-designs the generator architecture and proposes novel ways to control the image synthesis process. Here we will introduce some essential features of Style GAN.

The first feature is the mapping network, which is a multi-layer, fully connected network that transforms the input noise vector $\mathbf{z}$ into the so-called style vector $\mathbf{w}$ before sending it to the main generator. Specifically, in the original Style GAN, the mapping network consists of 8 fully connected layers, and its output $\mathbf{w}$ is of the same size as the noise input layer (512×1). The purpose of the mapping network is to encode the input vector into an intermediate vector whose different elements control different visual features.

The second feature is the adaptive instance normalization module. In short, the vector $\mathbf{w}$ obtained from the mapping network is injected into the generator by adaptive instance normalization, which is motivated by the literature on style transfer [Huang and Belongie, 2017]. The module is added to each resolution level and aims to define the visual expression of the features in that level. The adaptive instance normalization module works as follows. First, an instance normalization operation is performed, where each channel of the convolution layer output is first normalized to mean 0 and variance 1. Then, the intermediate

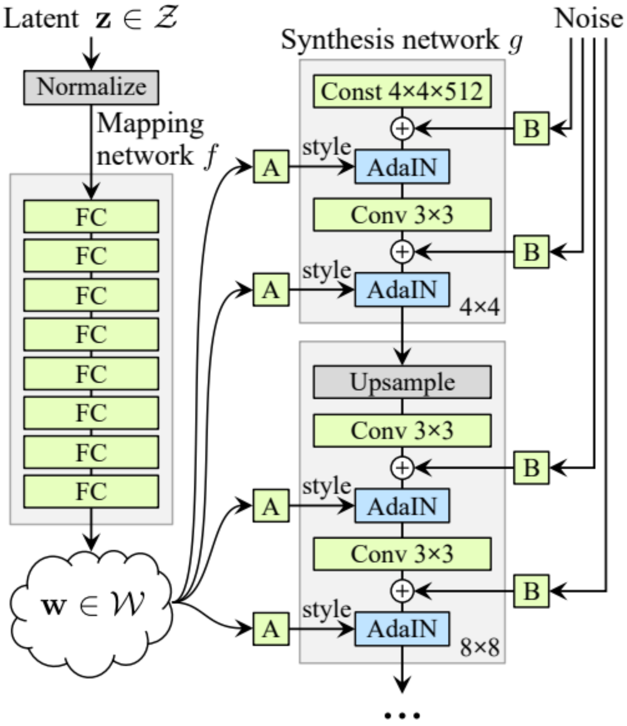


Figure 2.6: Illustration of Style GAN. Figure taken from Karras et al. [2019]. Latent vector $\mathbf{z}$ is first mapped into an intermediate latent space $\mathcal{W}$, which then controls the generator through adaptive instance normalization (AdaIN) at each convolution layer.

vector $\mathbf{w}$ is transformed using a fully-connected layer to obtain the $\mathbf{w}$-dependent shift and scale parameters $\mu(\mathbf{w})$ and $\sigma(\mathbf{w})$. Finally, the normalized convolutional output is affinely transformed by $\mu(\mathbf{w})$ and $\sigma(\mathbf{w})$.

The third feature is that Style GAN removes the standard input of noise vector $\mathbf{z}$ to the generator. Most GANs use the random latent vector $\mathbf{z}$ as the input to the generator. In contrast, Style GAN injects latent variables into the generator with adaptive instance normalization. Therefore, the input to the generator (where the tower of convolutional layers starts) is replaced by a shared but learnable constant. Karras et al. [2019] argue that passing latent information to the network as normalization controllers instead of as inputs improves the performance of modeling fine-grained details of images.

The structure of Style GAN's generator is demonstrated in Figure 2.6. The ideas behind the design of Style GAN are widely used, and in particular, we borrow many elements of Style GAN to the model presented in Section 6.

2.6 Summary

In this chapter, we reviewed 5 popular deep generative models. In summary, we will list the pros and cons of each model in this section. Note the main theme of this dissertation is designing new generative models by combining different existing models, and therefore it is critical to know the advantages and disadvantages of each type of model. At the end of this section, we will highlight the motivation for composing different generative models.

- Variational Auto-encoders
  - Pros: tractable estimate of likelihood; stable training; fast sampling (1 network evaluation); provide low dimensional latent representations.
  - Cons: low sample quality due to the prior hole issue.
- Normalizing Flows:
  - Pros: tractable and exact likelihood computation; fast sampling (1 network evaluation).
  - Cons: low parameter efficiency due to restricted structure; difficult to train; low sample quality.
- Energy-based Models:
  - Pros: no constraint on the functional form; less affected by the over-smoothing effect of maximum likelihood training (because EBMs will explicitly decrease the density outside the support of the data distribution).

  - Cons: expensive update due to MCMC sampling; slow sampling speed; cannot be easily scaled up to high dimensional data.

- Denoising Diffusion Models:
  - Pros: excellent sample quality; good likelihood estimate; stable training (the loss is essentially a L_2 regression).
  - Cons: extremely slow sampling speed.

- Generative Adversarial Nets:
  - Pros: excellent sample quality; fast sampling (1 network evaluation).
  - Cons: training instability; suffered from mode collapse; no likelihood estimation.

The goal of designing deep generative models by combining two existing models is to enjoy the best of both worlds: we hope to obtain a generative model that keeps the advantages while removing the disadvantages of both of its components. We call such a composition *symbiotic composition*. In ecology, symbiosis means the biological interaction between two different biological organisms, and the interaction must be beneficial to both species. We name our core idea with this term to emphasize that our proposed compositions can benefit both sides. The resulting models improve the generative performance and, more importantly, overcome some of the fundamental limitations of deep generative models.

In the following chapters, we will comprehensively introduce several of our proposed deep generative models designed with symbiotic composition.

CHAPTER 3

GENERATIVE LATENT FLOW: TOWARDS FLEXIBLE PRIOR DISTRIBUTIONS IN LATENT SPACE

In this chapter, we discuss Generative Latent Flow, a model we proposed that aims to improve the sample quality of auto-encoder based models by modeling the latent space with a flow-based prior distribution. We will begin with motivating our approach and introducing related work. Then we will give a detailed description of our method and present experimental results.

The material of this chapter is based on Xiao et al. [2019].

3.1 Motivation and Introduction

Auto-encoder is one of the earliest deep learning models for unsupervised learning. An auto-encoder consists of a pair of neural networks: an encoder that learns how to compress the input data into a low-dimensional representation and a decoder that learns how to reconstruct the data from the encoded representation to be as close to the original input as possible. Originally, the compressed expression obtained from the encoder was applied to downstream tasks such as classification. In their original form, auto-encoders were not generative models, as they only compress and reconstruct input data, while it is not clear how they can generate new samples. The idea of generative auto-encoders is to train an encoder-decoder pair with reconstruction loss and utilize the obtained decoder to generate new samples. Such an idea requires that we can sample from the (empirical) marginal distributions of the compressed representations from the encoder. This is because the representations associated with real data may only occupy a small portion of the high-dimensional latent space, and they may be distributed with a specific shape, and only drawing samples accordingly can produce realistic decoded samples.

One method to design generative auto-encoders is to force the distribution of the compressed representations of the training data to follow a simple distribution that can be easily sampled from. Doing so requires regularizing the encoder. For example, Variational Auto-encoders introduced in Section 2.1 follow this approach with a probabilistic encoder so that the compressed representation is a sample from the posterior distribution. VAEs encourage the posterior distribution to be close to a noise distribution (called prior) by minimizing a KL divergence term between them, and it can be shown that such a loss term will equivalently enforce the marginal distributions of the representations to be close to the prior. Some other models with this approach will be discussed in Section 3.2. Forcing the latent representations to follow a noise distribution has several disadvantages. Firstly, doing so will introduce an undesirable trade-off in the reconstruction quality. If the constraint on the latent distribution is stronger, the reconstruction quality (and subsequently the generation quality) will be worse. Secondly, it is observed that even if the latent distribution is constrained to be close to some prior, typically, there is still a significant mismatch between them. In VAEs, there are regions in the latent space that have a high density under the prior but have a low density under the marginal distribution of latent variables (also called the aggregated posterior).

Due to the limitations of regularizing the encoded latent distribution to match the prior distribution, an alternative approach to design generative auto-encoders is proposed to fit the latent distribution with a parameterized generative model. In this approach, data samples can be generated by drawing latent samples from the parameterized latent distribution and decoded with the decoder. The latent generative model can be learned jointly with the reconstruction objective or separately after the auto-encoder is trained. Some models that follow this approach will be discussed in Section 3.2. The parameterized latent distribution approach has several advantages. Firstly, it allows arbitrarily complex distribution of the latent variables, which enables the auto-encoder to reconstruct data faithfully without being constrained. Secondly, due to the low dimension of latent variables, the generative model on the latent space can be easy to train.

We study the approach of a parameterized latent distribution, and in particular, we parameterize the latent distribution with normalizing flows. As a starting point, we replace the simple prior distribution of VAEs with normalizing flows and train the auto-encoder and the normalizing flow prior jointly with the VAEs' objective. We carefully study this method and make the surprising novel observation that in order to produce high-quality samples, it is necessary to increase the weight of the reconstruction loss significantly. 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. 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 as this weight increases, we approach a vanishing noise limit that corresponds to a deterministic auto-encoder. 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 study in this chapter makes several contributions:

- We carefully study the effects of equipping VAEs with a normalizing flow prior on image generation quality as the weight of the reconstruction loss is increased.

- ii) Based on this finding, we introduce Generative Latent Flow, which uses auto-encoders instead of VAEs.

- 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 Related Work

In general, in order for an AE based model with an encoder-decoder structure to generate new samples resembling the training data distribution, two criteria need to be ensured: (a) the decoder is able to produce a good reconstruction of a training image given its encoded latent variable $\mathbf{z}$, and (b) the empirical latent distribution $q(\mathbf{z})$ of $\mathbf{z}$'s returned by the encoder is close to the prior $p(\mathbf{z})$. In VAEs, the empirical latent distribution is often called aggregated or marginal posterior: $q(\mathbf{z}) = \mathbb{E}_{\mathbf{x} \sim p_{data}} [q(\mathbf{z}|\mathbf{x})]$. While (a) is mainly driven by the reconstruction loss, satisfying criterion (b) is more complicated. Intuitively, criterion (b) can possibly be achieved by designing mechanisms that either modify the empirical latent distribution $q(\mathbf{z})$, or conversely modify the prior $p(\mathbf{z})$. There is plenty of previous work in both directions.

Modifying the empirical latent distribution $q(\mathbf{z})$: In the classic VAE model, $D_{\mathrm{KL}}[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ in the ELBO loss can be decomposed as $D_{\mathrm{KL}}[q(\mathbf{z})\|p(\mathbf{z}))$ plus a mutual information term as shown in Hoffman and Johnson [2016]. Therefore, VAEs modify the marginal distribution of latent variables $q(\mathbf{z})$ indirectly through regularizing the variational posterior distribution $q(\mathbf{z}|\mathbf{x})$. Several modifications to VAE's loss [Chen et al., 2018b, Kim and Mnih, 2018], which are designed for the task of unsupervised latent disentanglement, put a stronger penalty specifically on the mismatch between $q(\mathbf{z})$ and $p(\mathbf{z})$, where $p(\mathbf{z})$ is predetermined (e.g., i.i.d Gaussian). There are also attempts to incorporate normalizing flows into the encoder to provide more flexible approximate posteriors [Rezende and Mohamed, 2015, Kingma et al., 2016]. However, empirical evaluation shows that VAEs with flow posteriors do not reduce the mismatch between $q(\mathbf{z})$ and $p(\mathbf{z})$ [Rosca et al., 2018]. Furthermore, as of yet, all these modifications to VAEs have not been shown to improve generation quality.

Adversarial auto-encoders (AAEs) [Makhzani et al., 2015] and Wasserstein auto-encoders (WAEs) [Tolstikhin et al., 2017] use an adversarial regularizer or maximum mean discrepancy (MMD) regularizer to force $q(\mathbf{z})$ to be close to $p(\mathbf{z})$. In AAE, in addition to the auto-encoder which is trained by the reconstruction objective, there is an additional latent discriminator that tries to distinguish between samples from the prior $p(\mathbf{x})$ and samples from the encoder, with an adversarial objective. Note that compared to GANs, AAEs move the adversary from the input (pixel) space to the latent space, where $p(\mathbf{z})$ may have a nice shape with a single mode (for a Gaussian prior), in which case the task should be easier than matching an unknown, complex, and possibly multi-modal distributions as usually done in GANs. WAEs propose to minimize the MMD between samples from $p(\mathbf{x})$ and $q(\mathbf{z})$, which has an unbiased U-statistic estimator that can be optimized with gradient descent. AAEs and WAEs are shown to improve generation quality, as they generate sharper images than VAEs do.

Modifying the prior distribution $p(\mathbf{z})$: An alternative to modifying $q(\mathbf{z})$ is adopting a trainable prior. VampPrior [Tomczak and Welling, 2018] uses a mixture of encoders to represent the prior. However, this requires storing training data or pseudo-data to generate samples at test time. Methods involving the same idea of approximating $q(z)$ using a sampled mixture of posteriors during training include [Bauer and Mnih, 2018, Klushyn et al., 2019]. This is a natural way to let the prior match $q(z)$. However, these methods have not been shown to improve generation quality. Takahashi et al. [2019] use the likelihood ratio estimator to train a prior distribution. However, at test time, the aggregate posterior is used for sampling in the latent space.

A large number of previous or concurrent work can be categorized as two-stage approaches, where an auto-encoder (or VAE) is trained first, and then a generative model on the latent space is trained with the fixed encoder. Two-stage VAE [Dai and Wipf, 2019] introduces another VAE on the latent space defined by the first VAE to learn the distribution of latent variables. VQ-VAE and its follow-up models [Van Den Oord et al., 2017, Razavi et al., 2019, Esser et al., 2021b] first train a vector-quantized auto-encoder with discrete latent variables and then fits an auto-regressive prior to the latent space. GLANN [Hoshen et al., 2019] learns a latent representation by GLO [Bojanowski et al., 2017] and matches the densities of the latent variables with an implicit maximum likelihood estimator [Li and Malik, 2018]. RAE+GMM [Ghosh et al., 2019] trains a regularized auto-encoder (which is an auto-encoder with a constraint on the L_2 norm of latent variables) and fits a mixture of Gaussian distribution on the latent space. NCP-VAE [Aneja et al., 2021] propose an energy-based prior defined by the product of a base prior distribution and a re-weighting factor, designed to bring the base closer to the aggregate posterior. The re-weighting factor is trained by noise contrastive estimation. All these methods have been shown to improve the quality of the generated images.

VAE with normalizing flows: We note that modifications of VAEs with a normalizing flow posterior have been extensively studied. In contrast, VAEs with flow prior have attracted much less attention. [Huang et al., 2017] briefly discusses this model to solve the distribution mismatch in the latent space, and recently [Xu et al., 2019] shows the advantages of learning a flow prior over learning a flow posterior. However, these papers only focus on improvements in the data likelihood.

3.3 Combining Normalizing Flows with AE-based Models

In this section, we discuss the combination of normalizing flow priors with auto-encoder based models in detail. We first introduce VAEs with normalizing flow prior and present some novel observations with respect to this model. Later we propose Generative Latent Flow (GLF) to further simplify the model and improve performance.

3.3.1 VAEs with Normalizing Flow Prior

We can easily replace the simple prior distribution in the original setting of VAEs with a learnable prior parameterized by deep networks. In particular, when we use a normalizing flow as the prior distribution, the training objective (ELBO) can be explicitly obtained due to the fact that we can tractably compute the marginal density of a normalizing flow model.

To derive the ELBO with a normalizing prior, we start by writing the ELBO with a general prior $p_\eta(\mathbf{x})$ with parameter η:

$$\begin{aligned}\mathcal{L}_{\theta,\phi,\eta}(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\eta(\mathbf{z})) \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\phi(\mathbf{x}|\mathbf{z}) + \log p_\eta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right].\end{aligned} \tag{3.1}$$

The first term is related to the reconstruction loss, while the last two terms can be combined as $D_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x})\|p_\eta(\mathbf{z}))$.

We can also introduce an extra hyper-parameter $\beta > 0$ for future reference. β controls the relative weight of the reconstruction loss and the KL divergence loss, and write the objective as

$$\mathcal{L}_{\theta,\phi,\eta,\beta}(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\beta\cdot\log p_\phi(\mathbf{x}|\mathbf{z}) + \log p_\eta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]. \tag{3.2}$$

Note that $\beta = 1$ corresponds to the original ELBO. When $\beta \neq 1$, the expression may not be a valid lower bound for the log likelihood, but it is still frequently used to train VAEs [Higgins et al., 2016]. In the standard formulation of VAEs, the generative model p_θ assumes an independent Gaussian distribution with variance 1 at each pixel. The parameter β allows us to adjust this variance as $1/\beta$.

If we introduce a normalizing flow f_η for the prior distribution, then the prior p_η becomes

$$p_\eta(\mathbf{z}) = p_0(f_\eta(\mathbf{z}))\left|\det\left(\frac{\partial f_\eta(\mathbf{z})}{\partial\mathbf{z}}\right)\right|, \tag{3.3}$$

where p_0 is the standard Gaussian density. Substituting this prior into Equation 3.2, we obtain $\mathcal{L}_{\theta,\phi,\eta}$ for VAEs with flow prior:

$$\mathcal{L}_{\theta,\phi,\eta,\beta}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\beta \cdot \log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_0\left(f_\eta(\mathbf{z})\right) + \log\left|\det\left(\frac{\partial f_\eta(\mathbf{z})}{\partial \mathbf{z}}\right)\right| - \log q_\phi(\mathbf{z}|\mathbf{x})\right]. \tag{3.4}$$

The second and third terms together are the log-likelihood of $\mathbf{z}$ 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 maximizing $\mathcal{L}_{\theta,\phi,\eta,\beta}$ over all training samples.

Previous work on VAEs with a flow prior did not consider tuning β (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 $\beta = 1$, VAEs with a flow prior do not significantly improve the generation quality. The reason might be that although $p_\eta(\mathbf{z})$ is matched with $q_\phi(\mathbf{z})$ due to the flow transformation, the decoder is not good enough to reconstruct sharp images (i.e, criterion (a) in Section 3.2 is not ensured). In contrast, we find that increasing β 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 β increases, two things occur, as demonstrated empirically in Section 3.4. 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 encoder, leading to a deterministic auto-encoder and a normalizing flow that is used to match the distribution of the latent variables obtained from the data.

3.3.2 Generative Latent Flow

We consider the objective of VAEs with a flow prior in Equation 3.2 when VAE is replaced by an auto-encoder. In an auto-encoder, denoting E_ϕ as the encoder, $\mathbf{z} = E_\phi(\mathbf{x})$ is deterministic so that $q_\phi(\mathbf{z}|\mathbf{x})$ in Equation 3.2 becomes a delta distribution and the entropy term can be removed. The overall training objective is then minimizing

$$\mathcal{L}^{\text{reg}}_{\theta,\phi,\eta,\beta}(\mathbf{x}) = \beta \cdot \mathcal{L}_{\text{recon}}\left(\mathbf{x}, G_\theta\left(E_\phi\left(\mathbf{x}\right)\right)\right) + \mathcal{L}_{\text{NLL}}\left(f_\eta\left(E_\phi\left(\mathbf{x}\right)\right)\right), \tag{3.5}$$

where G_θ is the decoder, the reconstruction loss $\mathcal{L}_{\text{recon}}\left(\mathbf{x}, G_\theta\left(E_\phi\left(\mathbf{x}\right)\right)\right)$ corresponds to the negative log likelihood of the generative model $-\log p_\theta(\mathbf{x}|\mathbf{z})$, and the prior negative log likelihood term $\mathcal{L}_{\text{NLL}}\left(f_\eta\left(E_\phi\left(\mathbf{x}\right)\right)\right)$ corresponds to $-\log p_\eta(\mathbf{z})$, which is the negative of the second and the third term of Equation 3.2:

$$-\log p_\eta(\mathbf{z}) = -\log p_0(f_\eta(\mathbf{z})) - \log\left|\det\left(\frac{\partial f_\eta(\mathbf{z})}{\partial \mathbf{z}}\right)\right|. \tag{3.6}$$

As noted before, larger β's yield better reconstruction 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 θ of the normalizing flow. Therefore, optimizing Equation 3.6 with extremely large β is approximately equivalent to optimizing

$$\mathcal{L}^{\text{GLF}}_{\theta,\phi,\eta,\beta}(\mathbf{x}) = \beta \cdot \mathcal{L}_{\text{recon}}\left(\mathbf{x}, G_\theta\left(E_\phi\left(\mathbf{x}\right)\right)\right) + \mathcal{L}_{\text{NLL}}\left(f_\eta\left(\text{sg}\left[E_\phi\left(\mathbf{x}\right)\right]\right)\right), \tag{3.7}$$

where sg$[\cdot]$ 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 Equation 3.7 as **Generative Latent Flow** (GLF), to highlight that our model applies normalizing flows on latent variables. We call the model trained by Equation 3.2, without stopped gradient, **regularized GLF**, since the flow acts as a regularizer on the

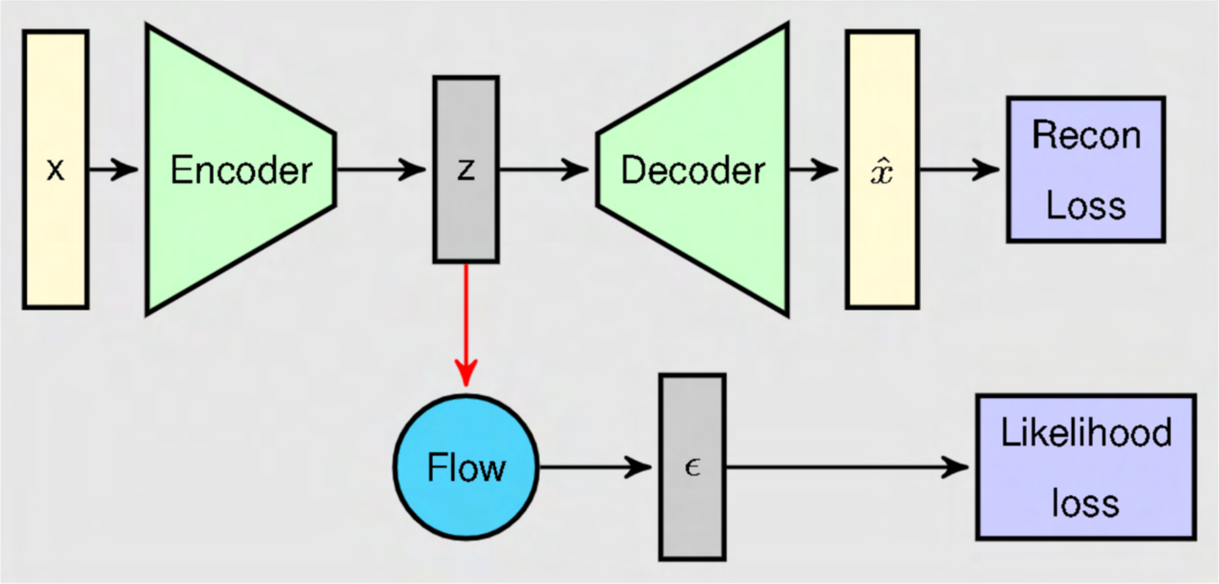


Figure 3.1: Illustration of the GLF model. The red arrow contains a stop gradient operation.

encoder. See Figure 3.1 for an illustration of the GLF model.

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 $\mathbf{z}$'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 scale of the output of E_ϕ, the Jacobian term can dominate the entire objective. While the latent variables cannot become exactly 0 because of the presence of the reconstruction loss, the extremely small 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. We demonstrate the issues with regularized GLF in Section 3.4.

3.4 Experimental Results

To demonstrate the performance of our proposed method, we present both quantitative and qualitative evaluations on four commonly used datasets for generative models: MNIST [Lecun, 2010], Fashion MNIST [Xiao et al., 2017], CIFAR-10 [Krizhevsky et al., 2009] and CelebA [Liu et al., 2015]. Throughout the experiments, we use 20-dimensional latent variables for MNIST and Fashion MNIST, and 64-dimensional latent variables for CIFAR-10 and CelebA.

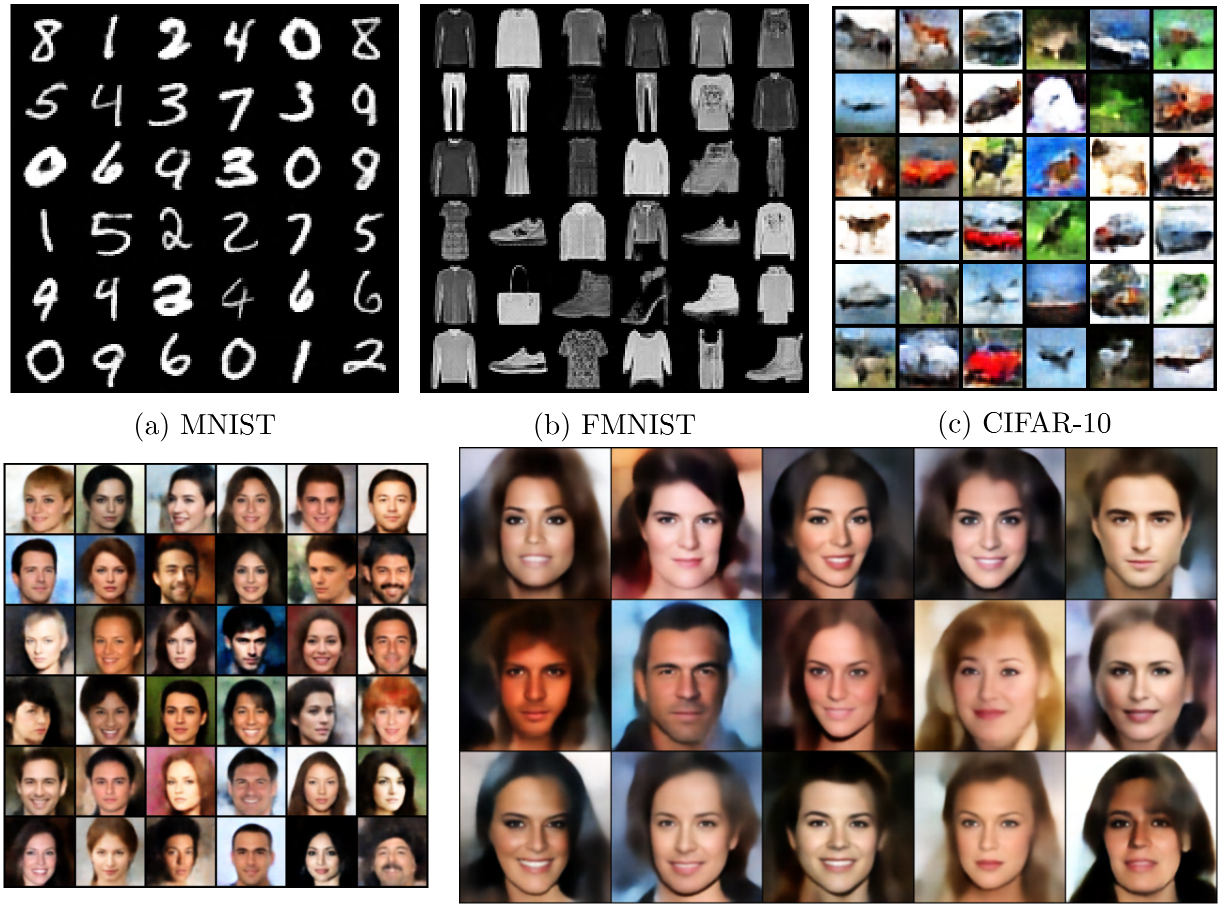
Lucic et al. [2018] adopted a common network architecture based on InfoGAN [Chen et al., 2016] to evaluate GANs. In order to make fair comparisons without designing arbitrarily large networks to achieve better performance, we use the generator architecture of InfoGAN as our decoder's architecture, and the encoder is set symmetric to the decoder. For the flow applied to the latent variables, we use four affine coupling blocks, 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%).

We use the FID score [Heusel et al., 2017] introduced in Section 1.2.4 to evaluate the sample quality. In addition, we use the precision and recall metric [Sajjadi et al., 2018], which can assess both the quality and diversity of generated samples.

We present our main results in Section 3.4.1, where we include both quantitative and qualitative results of our model. In Section 3.4.2, we carefully study several closely correlated methods, including VAEs with flow prior, GLF, and regularized GLF. In Section 3.4.3, we present the experimental settings for reproduction purposes.

3.4.1 Main 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



(a) MNIST

(b) FMNIST

(c) CIFAR-10

(d) CelebA

(e) CelebA-HQ

Figure 3.2: Randomly generated samples from our method trained on different datasets.

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 Section 3.4.3.

Note that the authors of WAE propose two variants, namely WAE-GAN and WAE-MMD. We only report the results of WAE-GAN, 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

Table 3.1: FID scores obtained from different AE-based generative models. For our reported results, we executed 10 independent trials and report the mean and standard deviation of the FID scores. Each trail is computing 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

models use MSE loss. The perceptual loss is obtained by feeding both training images and reconstructed images into a pre-trained network such as the VGG network 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 greatly outperforms VAE+flow prior with the default setting of $\beta = 1$. We also confirm that VAE+flow posterior cannot improve generation quality. Perhaps the competing model with the closest performance to ours is RAE+GMM, which shares some similarities with GLF in that both methods fit the density of the latent variables of an AE explicitly.

To compare our method with GANs, we also include the results from [Lucic et al., 2018] in Table 3.2. 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 that of carefully tuned GANs.

In Table 3.3, we present the Precision and Recall scores of our method and several

Table 3.2: FID score comparisons of GANs and GLF

	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
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

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

	MNIST	Fashion	CIFAR-10	CelebA
WAE-GAN	(0.98, 0.96)	(0.90, 0.83)	(0.41, 0.72)	(0.50, 0.51)
Two-stage VAE	(0.98, 0.97)	(**0.94**, 0.84)	(0.38, 0.66)	(0.45, 0.55)
RAE+GMM	(**0.99**, 0.97)	(0.92, 0.92)	(0.37, 0.73)	(0.33, 0.44)
GLF (ours)	(0.98, **0.98**)	(0.932, **0.92**)	(**0.48**, **0.76**)	(**0.54**, **0.61**)
GLANN+perceptual loss	(0.97, 0.98)	(0.985, 0.963)	(**0.860**, 0.825)	(0.574, 0.681)
GLF+perceptual loss (ours)	(**0.99**, **0.99**)	(**0.99**, **0.98**)	(0.76, **0.85**)	(**0.76**, **0.78**)

competing methods. 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 better. As shown in the table, GLF obtains state-of-the-art Precision and Recall scores across all datasets, indicating that our method outperforms competing methods in terms of both sample quality and diversity.

Some qualitative results are shown in Figure 3.2. Besides samples of the datasets used for quantitative evaluation, samples of CelebA-HQ [Karras et al., 2017] with the larger size of 256×256 are also included 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. In Figure 3.3, we show CelebA images generated by linearly interpolating two sampled random noise vectors. The smooth and natural transition shows that our model

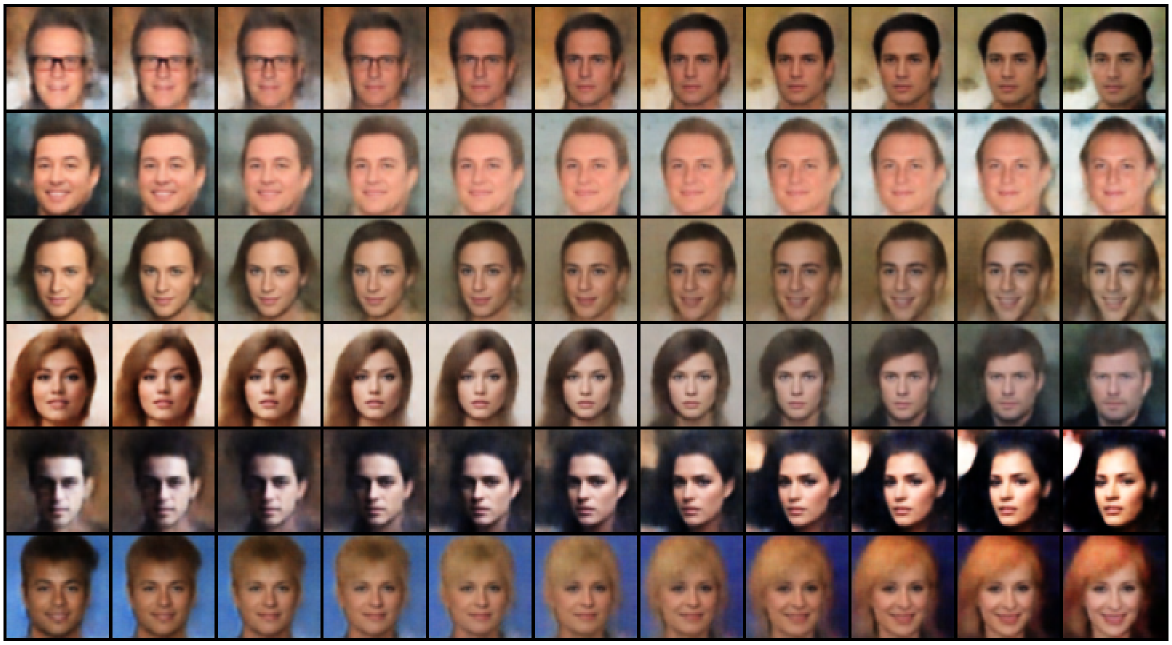
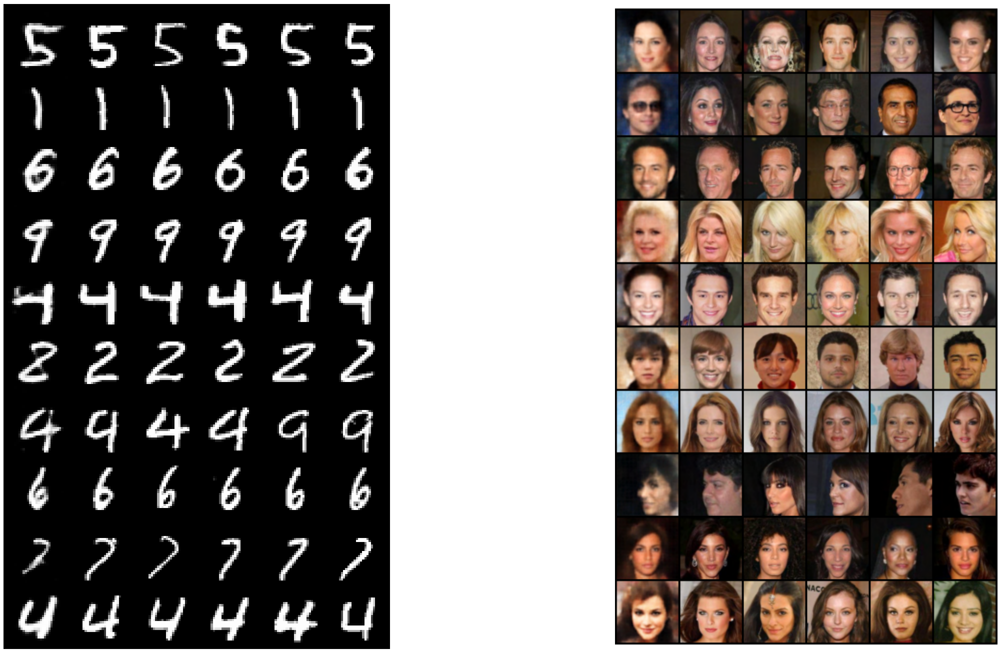


Figure 3.3: Random noise interpolation in the noise space of GLF on CelebA dataset

can generate samples that have not been seen during training. To provide further evidence that our model does not overfit or ‘memorize’ the training set, we show the nearest neighbors in training set for some generated samples in Figure 3.4.

We also observe that samples from models trained with perceptual loss have higher quality. We present samples from models trained with perceptual loss in Figure 3.5.

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 for methods using perceptual loss, the per epoch training time is longer because VGG activations need to be computed. These two tables show that GLF needs 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. The combined results indicate that GLF requires much less training time while generating samples with higher quality.



(a) MNIST

(b) CelebA

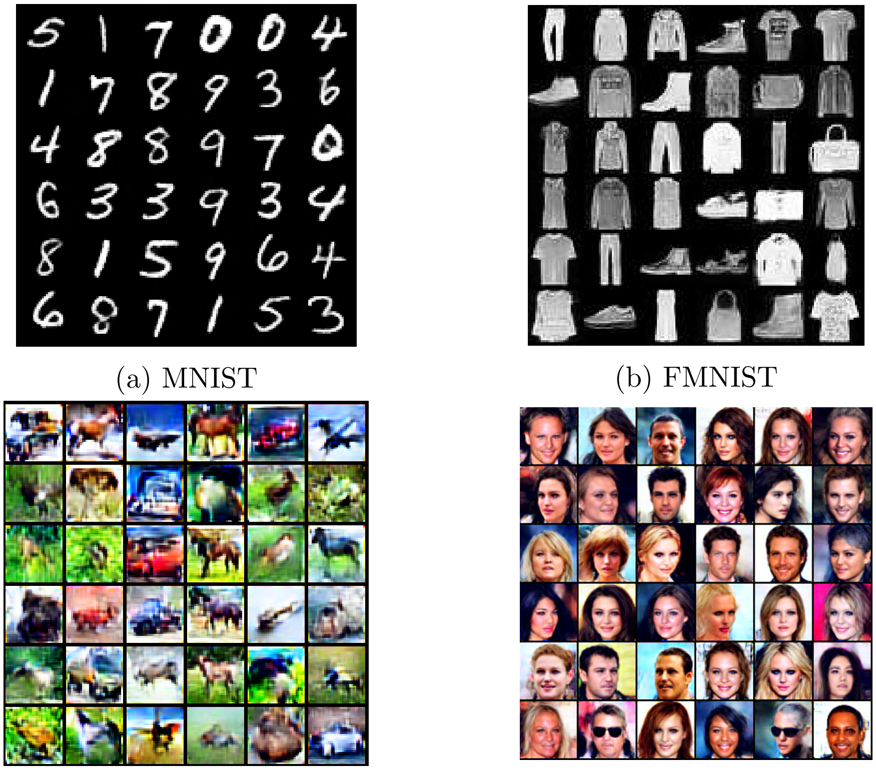
Figure 3.4: Some randomly generated samples are presented in the **leftmost** column in each picture. The other 5 columns of each picture show the top 5 nearest neighbors of the corresponding sample in the training set.

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

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

As discussed in Section 3.3, we underline the novel finding regarding the relationship between the weight on the reconstruction loss and the sample quality of VAEs 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 3.6 (a). In Figure Figure 3.6 (b), we plot the entropy term, which is the last term in Equation 3.4, 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



(a) MNIST

(b) FMNIST

(c) CIFAR-10

(d) CelebA

Figure 3.5: Randomly generated samples from our method with perceptual loss.

deviation of the approximate posterior on the j^{th} latent variable. Higher entropy means that the latent variables have lower variances. In Figure 3.6(c), 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. Settings for the experiments in this subsection can be found in Section 3.4.3.

From Figure 3.6 (a), we clearly 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 3.6 (b), the posterior variances of VAEs become very small, so that effectively we are training an AE. For example, as shown in Figure 3.6 (b), when $\beta = 400$, the corresponding average posterior variance is around 10^{-4}. This motivates us to use a deterministic auto-

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

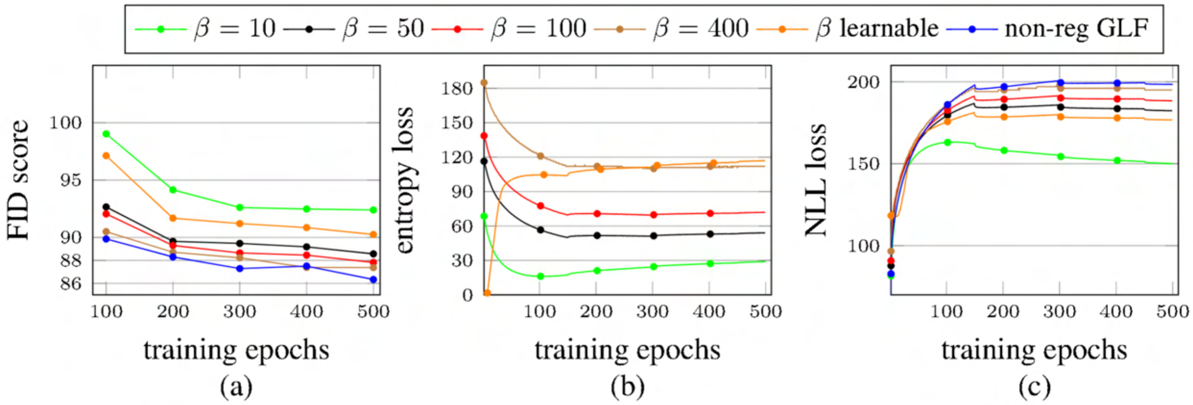


Figure 3.6: (a) Record of FID scores on CIFAR-10 for VAEs+flow prior with different values of β and GLF. (b) Record of entropy losses for corresponding models. (c) Record of NLL losses for corresponding models.

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 flow), as we verify empirically that increasing β on plain VAEs leads to worse FID scores.

As discussed in Section 3.3.2, 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 3.7. For low values of $\beta = 1$ and 10, the NLL loss completely dominates the learning signal, 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 very large 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,

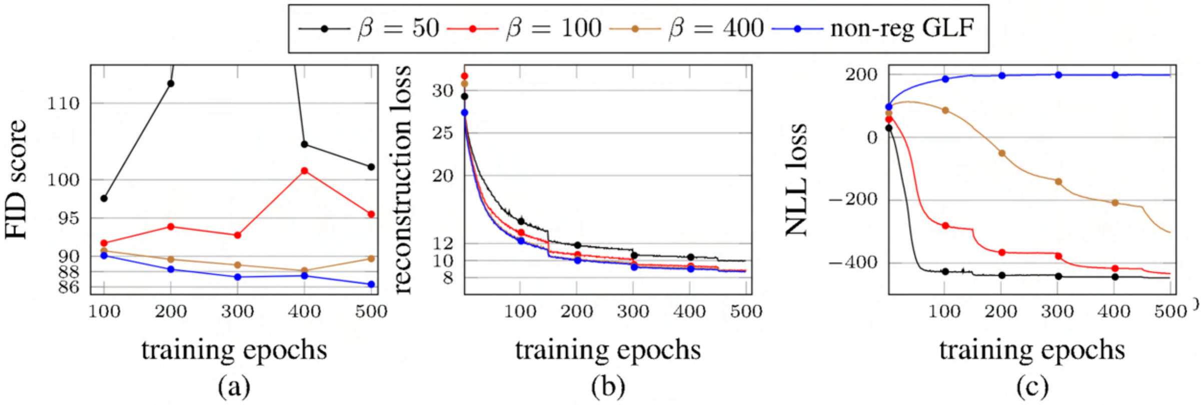


Figure 3.7: (a) Record of FID scores on CIFAR-10 for regularized GLF with different values of β and GLF. $\beta = 1$ and 10 are omitted because they lead to divergence in the reconstruction loss. (b) Record of reconstruction loss for the corresponding models. (c) Record of NLL loss for the corresponding models.

resulting in stable improvements in FID scores as training progress.

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 3.6 (c) 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 3.7 (c). Comparing Figure 3.6 (a) and 3.7 (a), 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, the results of 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.

3.4.3 Experimental Settings

Network Architectures:

In this section we provide Table 3.6 that summarizes the auto-encoder network structure.

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

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.

Details of Experiments:

Here 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 is 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 optimizer with initial learning rate 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 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 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 imgaes. We use batch size 64, with initial learning rate 10^{-3} for both the AE and the flow. We train our model for 60 epochs, with learning rate decaying by a factor of 2 after 40 epochs.

Settings for training VAEs and VAE variants: We adopt common 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 initial learning rate 10^{-3} for both the VAE and the flow, if applicable. We find VAEs need 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 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 learning rate halved every 150 epochs; the second VAE is trained for 800 epochs with 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.

3.4.4 Settings for training RAE+GMM

The settings of batch size, learning rate scheduling and number of epochs for training RAE are the same as those of 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.

Settings for Experiments in Section 3.4.2: For all experiments in Section 3.4.2, we use batch size 256 and initial learning rate 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.

3.5 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 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 do believe that it opens the door to realizing the potential of AE based models to produce high-quality samples just as GANs do. 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.

The GLF model can be seen as a symbiotic composition of auto-encoder and normalizing flows. Specifically, on the one hand, the auto-encoder provides a latent space that is low dimensional and unstructured, which makes the training of normalizing flows significantly easier than training directly on data space. On the other hand, the normalizing flow accurately models the distribution of latent variables, providing a powerful prior that is easy to sample from and allowing the auto-encoder to use its full potential to reconstruct data.

CHAPTER 4

EXPONENTIAL TILTING OF GENERATOR MODELS WITH ENERGY-BASED MODELS

This chapter introduces the idea of exponential tilting of a base generative model with an energy-based model. The EBM can refine the distribution modeled by the base generative model and improve the sample quality. We will begin with motivating our approach and introducing related work. Then we will give a detailed description of our method, including the cases when the base generative model is a normalizing flow or VAE, and present experimental results.

The material of this chapter is based on Xiao et al. [2020a] and Xiao et al. [2021a].

4.1 Motivation and Introduction

In Chapter 2, we mentioned that likelihood-based deep generative models, which are usually trained by maximum likelihood, enjoy the benefits that their training is stable and they cover modes in data more faithfully by construction. The reason is that maximum likelihood corresponds to minimizing the forward KL divergence:

$$\min_{\theta} D_{\mathrm{KL}}(p_{\mathrm{data}}(\mathbf{x}) \| p_{\theta}(\mathbf{x})) = \int p_{\mathrm{data}}(\mathbf{x}) \log \frac{p_{\mathrm{data}}(\mathbf{x})}{p_{\theta}(\mathbf{x})} d\mathbf{x}. \tag{4.1}$$

From this objective, we see that training will be heavily penalized if $p_{\theta}(\mathbf{x}) = 0$ at where $p_{\mathrm{data}}(\mathbf{x}) \neq 0$. In other words, the objective will enforce the model $p_{\theta}(\mathbf{x})$ to spread out over all the support of $p_{\mathrm{data}}(\mathbf{x})$, resulting in good mode coverage. However, the same objective may cause a major disadvantage of likelihood-based generative models: they tend to assign a high probability to regions with low density under the data distribution. The reason is that the real data distribution is very complicated, while parameterized models have limited capacity even when parameterized by a deep neural network, and enforcing a relatively

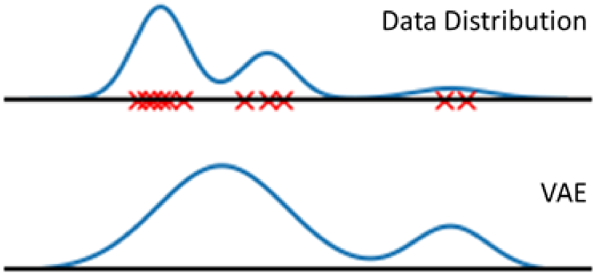


Figure 4.1: illustration of such a density mismatch between true data distribution and a parametrized VAE model. Red crosses are training data.

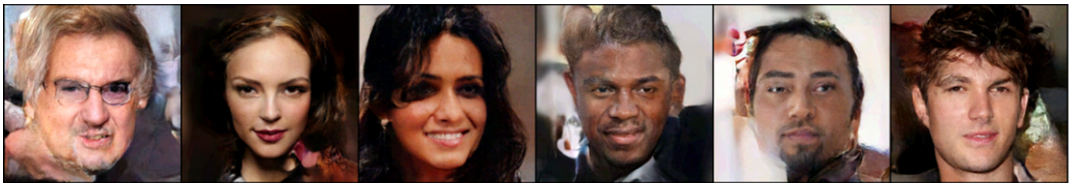


Figure 4.2: Samples from NVAE, one of the strongest VAE models, trained on CelebA-HQ dataset. Although the overall shape of human faces is good, we observe undesirable artifacts especially on the boundary of faces.

simple distribution to cover all the support of a complex distribution will result in severe density mismatch. This mismatch often results in blurry or corrupted samples generated by likelihood-based models. It also explains why likelihood-based generative models often fail at out-of-distribution detection [Nalisnick et al., 2018]. Figure 4.1 gives an illustration of such a density mismatch for a VAE model, where we see that although the VAE roughly captures the shape of the real data distribution, the first mode of the modeled distribution actually belongs to a low-density region of the true data distribution. As a result, even the state-of-the-art VAE models [Vahdat and Kautz, 2020] cannot generate good samples (see Figure 4.2). Similar observations are made on other likelihood-based generative models, such as normalizing flows.

Among likelihood-based models, EBMs model the unnormalized data density by assigning low energy to high-probability regions in the data space [Xie et al., 2016, Du and Mordatch, 2019]. EBMs are appealing because they require almost no restrictions on network architectures (unlike normalizing flows) and are therefore potentially very expressive. They also suffer less from the issue of assigning high likelihood to non-data-like regions compared to other likelihood-based generative models, as they exhibit sharper sample quality and out-of-distribution generalization [Du and Mordatch, 2019]. The reason is that, during the maximum likelihood training of EBMs, areas with high probability under the model but low probability under the data distribution are penalized explicitly, as discussed in Section 2.3. However, training and sampling EBMs usually requires MCMC, which can suffer from slow mode mixing and is computationally expensive when neural networks represent the energy function. In particular, gradient-based MCMC sampling in the data space generally does not mix. The data distribution is typically highly multi-modal, and to approximate such a distribution, the energy function needs to be highly multi-modal as well. When sampling from such a multi-modal density in the data space, gradient-based MCMC tends to get trapped in local modes with little chance to traverse the modes freely. Without being able to generate fair examples from the model, the estimated gradient of the maximum likelihood learning can be very biased. As a result, previous EBM models only obtained limited success, even on small images.

The difficulty of MCMC sampling can possibly be resolved by mapping the data into a latent space and running MCMC in the latent space. For example, Hoffman et al. [2019] observe that it is much more efficient to run an MCMC sampler in the latent space transformed by a normalizing flow. Both VAE and normalizing flow models naturally come with a latent embedding of data that has a single mode and has smooth geometry. The embedding allows fast traversal of the data manifold by moving in the latent space and mapping the movements to the data space.

The above arguments suggest a possible composition of VAEs or normalizing flows with EBMs. The resulting model defines the generative distribution as the product of a base generative model, which is either a VAE or normalizing flow, and an EBM component

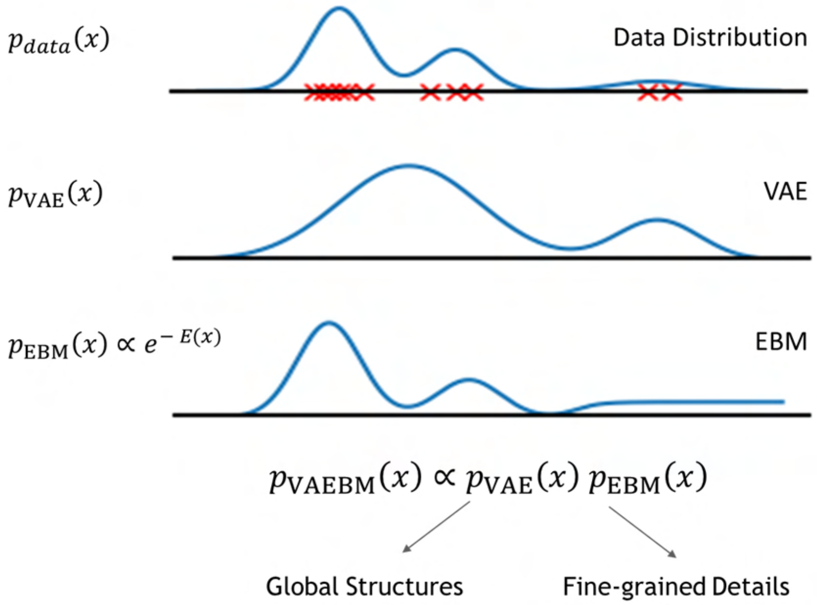

$p_{data}(x)$
Data Distribution
$p_{\mathrm{VAE}}(x)$
VAE
$p_{\mathrm{EBM}}(x) \propto e^{-E(x)}$
EBM
$p_{\mathrm{VAEBM}}(x) \propto p_{\mathrm{VAE}}(x)\, p_{\mathrm{EBM}}(x)$
Global Structures
Fine-grained Details

Figure 4.3: High level illustration of the idea of exponential tilting with a VAE as the base generative model.

defined in pixel space in the form of a correction, or an exponential tilting, of the base model. Intuitively, the base model captures the majority of the mode structure in the data distribution. However, it may still generate samples from low-probability regions in the data space. Thus, the energy function focuses on refining the details and reducing the likelihood of non-data-like regions. This is because in the negative training phase of EBMs, we sample from the model itself and obtain non-data-like samples, whose likelihood is then reduced by the energy function explicitly. The energy function defined in the pixel space also shares similarities with the discriminator in GANs, which can generate crisp and detailed images. Figure 4.3 gives a high level illustration of the idea, when the base generative model is a VAE, which leads to the VAEBM model.

Moreover, we show that training the compositional model by maximizing the data likelihood easily decomposes into training the base model and the EBM component separately. The model is trained using the standard maximum likelihood approach, while the EBM component requires sampling from the joint energy-based model during training. We show that we can sidestep the difficulties of sampling from the joint model, by reparameterizing the MCMC updates using the latent variables. This allows MCMC chains to quickly traverse the model distribution and it speeds up mixing. As a result, we only need to run short chains to obtain approximate samples from the model, accelerating both training and sampling at test time.

Experimental results show that when equipped with a strong base model, our model outperforms previous EBMs and state-of-the-art VAEs on image generation benchmarks, including CIFAR-10, CelebA 64, LSUN Church 64, and CelebA HQ 256 by a large margin, reducing the gap with GANs. We also show that our model faithfully covers the modes in the data distribution while having fewer spurious modes for out-of-distribution data. In particular, with the help of the powerful NVAE [Vahdat and Kautz, 2020], our VAEBM model is the first successful EBM applied to large images.

In summary, this chapter makes the following contributions:

- We propose a new framework of generative models using the product of a VAE or normalizing flow and an EBM defined in the data space.
- We show how training this model can be decomposed into training the base model first, and then training the EBM component.
- We show how MCMC sampling from the model can be pushed to the base model's latent space, significantly accelerating sampling.
- We demonstrate state-of-the-art image synthesis quality among likelihood-based models, confirm complete mode coverage, and show strong out-of-distribution detection performance.

4.2 Related Work

Our proposed framework is based on the exponential tilting of a base generative model by an EBM. It shares similarity with previous work that builds connections between EBMs and other generative models. Zhao et al. [2017], Che et al. [2020], Song et al. [2020b], Arbel et al. [2020] formulate EBMs with GANs, and use the discriminator to assign an energy. In particular, Che et al. [2020] proposed to sample from a EBM of the form

$$p^*(\mathbf{x}) = \frac{1}{Z} p_g(\mathbf{x}) e^{D(\mathbf{x})}, \tag{4.2}$$

where p_g is the implicit distribution defined by the generator G, and D is the discriminator. The model can be seen as a exponential tilting of p_g, and the sampling is done by running MCMC in the noise space by the mapping of $\mathbf{x} = G(\mathbf{z})$. Such a high level idea is closely related to ours.

Our work is partially inspired by neural transport sampling [Hoffman et al., 2019]. For an unnormalized target distribution, the neural transport sampler trains a flow-based model as a variational approximation to the target distribution, and then samples the target distribution in the space of latent variables of the flow-based model via change of variable. In the latent space, the target distribution is close to the prior distribution of the latent variables of the flow-based model, which is usually a unimodal Gaussian white noise distribution. Consequently the target distribution in the latent space is close to be unimodal and is much more conducive to the mixing and fast convergence of MCMC than sampling in the original space. Nijkamp et al. [2022] is a simplified special case of this idea, where they learn the EBM as a correction of a pre-trained flow-based model, so that they do not need to train a separate flow-based approximation to the EBM. The idea of Nijkamp et al. [2022] is exactly the same as Xiao et al. [2020a] and they were developed concurrently. However, we motivate the idea from the perspective of energy-based refinement rather than neural transport MCMC. The idea of neural transport MCMC relies on normalizing flows, because they need to compute the tractable density to ensure valid sampling. However, due to their topology-preserving nature, normalizing flows cannot easily transport complex multimodal data, and their sample quality on images is limited. In Xiao et al. [2021a], we found that the exponential tilting of VAE models leads to much improved sample quality.

A few previous works combine VAEs and EBMs in different ways from ours. Pang et al. [2020] and Vahdat et al. [2018b,a, 2020] use EBMs for the prior distribution, and Han et al. [2020, 2019] jointly learn a VAE and an EBM with independent sets of parameters by an adversarial game.

Finally, as we propose two-stage training where the base model is trained first followed by the training of the EBM, our work is related to post training of latent variable models. Previous work in this direction learns the latent structure of pre-trained VAEs [Dai and Wipf, 2019, Xiao et al., 2019, Ghosh et al., 2019], and sampling from learned latent distributions improves sample quality. These methods cannot easily be extended to VAEs with hierarchical latent variables, as it is difficult to fit the joint distribution of multiple groups of variables. Our purpose in two-stage training is fundamentally different: we post-train an energy function to refine the distribution in data space.

4.3 Formulation of Exponential Tilting with EBMs

In this section, we introduce the formulation of exponential tilting with EBMs. Recall from Chapter 2.3 that we write the density function of an EBM in the following form:

$$p_\theta(\mathbf{x}) = \frac{1}{Z_\theta} e^{-f_\theta(\mathbf{x})}, \tag{4.3}$$

where f_θ is the energy function and Z_θ is the normalizing constant. More generally, EBMs can have a parameterized base distribution $q_\phi(\mathbf{x})$, and we can write the model as

$$p_{\theta,\phi}(\mathbf{x}) = \frac{1}{Z_{\theta,\phi}} q_\phi(\mathbf{x}) e^{-f_\theta(\mathbf{x})}, \tag{4.4}$$

which can be seen as an exponential tilting of $q_\phi(\mathbf{x})$. For example, in Noise Contrastive Estimation introduced in Section 1.2.2, the base distribution $q_\phi(\mathbf{x})$ is the noise distribution, and the energy function is the re-weighting factor obtained from the binary classifier.

In this chapter, we propose to exponentially tilt base generative models, and therefore $q_\phi(\mathbf{x})$ will be another deep generative model such as a normalizing flow or a VAE. The resulting model $p_{\theta,\phi}$ is a correction or refinement over $q_\phi(\mathbf{x})$.

4.3.1 Normalizing Flows as the Base Generative Model

We first discuss the case where normalizing flows are used as the base generative model. Consider the exponential tilting model in Equation 4.4, where $q_\phi(\mathbf{x})$ is a normalizing flow and $f_\theta(\mathbf{x})$ is a free-form neural network. Note that the normalizing flow is defined by an invertible transformation F_ϕ through a change of variables. Specifically, the normalizing flow transforms $\mathbf{z} \sim q_0\mathbf{z}$ to $\mathbf{x} = F_\phi(\mathbf{z})$, and we can express the density in $\mathbf{z}$-space in terms of the density in $\mathbf{x}$-space by the change-of-variable formula:

$$q_0(\mathbf{z}) = q_\phi(\mathbf{x}) \frac{d\mathbf{x}}{d\mathbf{z}}, \tag{4.5}$$

where $d\mathbf{z}$ and $d\mathbf{x}$ are understood as the volumes of the infinitesimal local neighborhoods around $\mathbf{z}$ and $\mathbf{x}$ respectively under the mapping $\mathbf{x} = F_\phi(\mathbf{z})$. We can re-write Equation 4.5 as

$$q_0(\mathbf{z}) d\mathbf{z} = q_\phi(\mathbf{x}) d\mathbf{x}. \tag{4.6}$$

We can apply the same change-of-variable to $p_{\theta,\phi}(\mathbf{x})$, and obtain a corresponding distribution in the $\mathbf{z}$-space. To do this, denote $h_{\theta,\phi}(\mathbf{z})$ to be the distribution of $\mathbf{z}$ under $p_{\theta,\phi}(\mathbf{x})$ in $\mathbf{z}$-space, we have

$$h_{\theta,\phi}(\mathbf{z})d\mathbf{z} = p_{\theta,\phi}(\mathbf{x})d\mathbf{x} = \frac{1}{Z_{\theta,\phi}} q_\phi(\mathbf{x}) e^{-f_\theta(\mathbf{x})} d\mathbf{x}. \tag{4.7}$$

Applying the change-of-variable of $q_\phi(\mathbf{x})$ in Equation 4.6, we obtain

$$h_{\theta,\phi}(\mathbf{z}) = \frac{1}{Z_{\theta,\phi}} q_0(\mathbf{z}) e^{-f_\theta(F_\phi(\mathbf{z}))}. \tag{4.8}$$

Therefore, we obtain an equivalent distribution in $\mathbf{z}$-space, which is an exponential tilting of the prior noise distribution $q_0(\mathbf{z})$. To obtain a sample from $p_{\theta,\phi}(\mathbf{x})$, we can sample from $h_{\theta,\phi}(\mathbf{z})$ and apply the transformation $F_\phi(\mathbf{z})$.

As discussed in Section 4.1, we let the normalizing flow capture the overall shape of data distribution and use the exponential tilting technique to refine the model. Therefore, we first pre-train the normalizing flow with maximum likelihood and then train the energy function f_θ while keeping the flow fixed. We can train $p_{\theta,\phi}(\mathbf{x})$ with fixed ϕ^* for F_ϕ by the following gradient of log-likelihood:

$$\nabla_\theta \mathcal{L}_\theta = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\frac{\partial f_\theta(\mathbf{x})}{\partial \theta}\right] - \mathbb{E}_{\mathbf{x}'\sim p_{\theta,\phi^*}(\mathbf{x}')}\left[\frac{\partial f_\theta\left(\mathbf{x}'\right)}{\partial \theta}\right], \tag{4.9}$$

where samples from p_{θ,ϕ^*} can be drawn by transforming samples from $h_{\theta,\phi}(\mathbf{z})$ with F_{ϕ^*}, as discussed above. We can draw samples from $h_{\theta,\phi^*}(\mathbf{z})$ by running the following Langevin dynamics

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \frac{\epsilon}{2}\nabla_{\mathbf{z}}\left(f_\theta\left(F_{\phi^*}(\mathbf{z})\right) - \log p_0(\mathbf{z})\right) + \sqrt{\epsilon}\omega, \quad \omega \sim \mathcal{N}(0, I). \tag{4.10}$$

It is tempting to apply the above exponential tilting formulation to generator models, such as VAEs, by replacing the transformation F_ϕ with the generator. However, mathematically, exponential tilting of non-invertible generator model is more complicated, as $p_{\theta,\phi}(\mathbf{x})$ is not in closed form, and the distribution $h_{\theta,\phi}(\mathbf{z})$ requires evaluating an integral. Therefore, exponential tilting of flow based model is more convenient. Nevertheless, due to the strong invertible constraint on the structure of normalizing flow models, they cannot easily transport complex multimodal data, and their sample quality on images is limited. It might be a better idea to exponentially tilt generator models, which does better in modeling the complex data distribution. In the next subsection, we will introduce the case where the base distribution is a VAE.

4.3.2 VAEBM: VAEs as the Base Distribution

Here we discuss the case when the base model of the exponential tilting framework is a VAE, and we name this model VAEBM. Note that VAEs are latent variable models without a tractable marginal distribution, and therefore the VAEBM model is also written in the form of the joint distribution over $(\mathbf{x}, \mathbf{z})$. Formally, we define the generative model in VAEBM as

$$p_{\theta,\phi}(\mathbf{x}, \mathbf{z}) = \frac{1}{Z_{\theta,\phi}} p_\phi(\mathbf{x}, \mathbf{z}) e^{-f_\theta(\mathbf{x})}, \tag{4.11}$$

where $p_\phi(\mathbf{x}, \mathbf{z}) = p_\phi(\mathbf{z}) p_\phi(\mathbf{x}|\mathbf{z})$ is a VAE generator and $f_\theta(\mathbf{x})$ is a neural network-based energy function, operating only in the $\mathbf{x}$ space. Marginalizing out the latent variable $\mathbf{z}$ gives

$$p_{\theta,\phi}(\mathbf{x}) = \frac{1}{Z_{\theta,\phi}} \int p_\phi(\mathbf{x}, \mathbf{z}) e^{-f_\theta(\mathbf{x})} d\mathbf{z} = \frac{1}{Z_{\theta,\phi}} p_\phi(\mathbf{x}) e^{-f_\theta(\mathbf{x})}, \tag{4.12}$$

where $p_\phi(\mathbf{x})$ is the (intractable) marginal distribution of the VAE.

Given a training dataset, the parameters of VAEBM, θ, ϕ, can trained by maximizing the marginal log-likelihood on the training data:

$$\log p_{\theta,\phi}(\mathbf{x}) = \log p_\phi(\mathbf{x}) - f_\theta(\mathbf{x}) - \log Z_{\theta,\phi} \tag{4.13}$$

$$\geq \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\psi(\mathbf{z}|\mathbf{x})}[\log p_\phi(\mathbf{x}|\mathbf{z})] - D_{\mathrm{KL}}(q_\psi(\mathbf{z}|\mathbf{x})||p_\phi(\mathbf{z}))}_{\mathcal{L}_{\mathrm{vae}}(\mathbf{x},\psi,\phi)} \underbrace{-f_\theta(\mathbf{x}) - \log Z_{\theta,\phi}}_{\mathcal{L}_{\mathrm{EBM}}(\mathbf{x},\theta,\phi)}, \tag{4.14}$$

where ψ denotes the parameter of the VAE's encoder. Here we replace $\log p_\phi(\mathbf{x})$ with its variational lower bound. Equation 4.14 forms the objective function for training VAEBM. The first term corresponds to the VAE objective and the second term corresponds to training the EBM component. Next, we discuss how we can optimize this objective.

4.3.3 Training of VAEBM

We mentioned that we want to pre-train the base generative model and then train the EBM as a correction. In the case of VAEBM, the two-stage training is not only beneficial but also necessary, as we will explain the difficulty of jointly training the VAE and EBM.

Difficulties of Joint Training:

The whole VAEBM in Equation 4.11 is a special case of EBM, so it can be trained by maximum likelihood. The subtle thing is that the log partition function $\log Z_{\theta,\phi}$ depends on both ϕ and θ. We show that $\log Z_{\theta,\phi}$ has the gradients

$$\partial_\theta \log Z_{\theta,\phi} = \mathbb{E}_{\mathbf{x}\sim p_{\theta,\phi}(\mathbf{x})}\left[-\partial_\theta f_\theta\left(\mathbf{x}\right)\right] \quad \text{and} \quad \partial_\phi \log Z_{\theta,\phi} = \mathbb{E}_{\mathbf{x}\sim p_{\theta,\phi}(\mathbf{x})}\left[\partial_\phi \log p_\phi(\mathbf{x})\right]. \tag{4.15}$$

The derivation is as follows. Recall that $Z_{\theta,\phi} = \int p_\phi(\mathbf{x})e^{-f_\theta(\mathbf{x})}d\mathbf{x}$. For the derivative of

$\log Z_{\theta,\phi}$ w.r.t. ϕ, we have:

$$\begin{aligned}
\frac{\partial}{\partial \phi} \log Z_{\theta,\phi} &= \frac{\partial}{\partial \phi} \log \left(\int p_{\phi}(\mathbf{x}) e^{-f_{\theta}(\mathbf{x})} d\mathbf{x} \right) \\
&= \frac{1}{Z_{\theta,\phi}} \int \frac{\partial p_{\phi}(\mathbf{x})}{\partial \phi} e^{-f_{\theta}(\mathbf{x})} d\mathbf{x} \\
&= \frac{1}{Z_{\theta,\phi}} \int p_{\phi}(\mathbf{x}) e^{-f_{\theta}(\mathbf{x})} \frac{\partial \log p_{\phi}(\mathbf{x})}{\partial \phi} d\mathbf{x} \\
&= \int p_{\theta,\phi}(\mathbf{x}) \frac{\partial \log p_{\phi}(\mathbf{x})}{\partial \phi} d\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x} \sim p_{\theta,\phi}(\mathbf{x})} \left[\frac{\partial \log p_{\phi}(\mathbf{x})}{\partial \phi} \right] \qquad (4.16)
\end{aligned}$$

Similarly, it is easy to show that $\frac{\partial}{\partial \theta} \log Z_{\theta,\phi} = \mathbb{E}_{\mathbf{x} \sim p_{\theta,\phi}(\mathbf{x},\mathbf{z})} \left[-\frac{\partial f_{\theta}(\mathbf{x})}{\partial \theta} \right]$. Intuitively, both gradients encourage reducing the likelihood of the samples generated by the VAEBM model. Since, $p_{\theta,\phi}$ is an EBM, the expectation can be approximated using MCMC samples.

The first gradient in Equation 4.15 can be estimated easily by evaluating the gradient of the energy function at samples drawn from the VAEBM model $p_{\theta,\phi}(\mathbf{x})$ using MCMC. However, the second term involves computing the intractable $\frac{\partial}{\partial \phi} \log p_{\phi}(\mathbf{x})$. Estimating $\frac{\partial}{\partial \phi} \log p_{\phi}(\mathbf{x})$ requires sampling from the VAE's posterior distribution, given model samples $\mathbf{x} \sim p_{\theta,\phi}(\mathbf{x})$. To see this, note that Equation 4.16 can be further expanded to:

$$\frac{\partial}{\partial \phi} \log Z_{\theta,\phi} = \mathbb{E}_{\mathbf{x} \sim p_{\theta,\phi}(\mathbf{x})} \left[\mathbb{E}_{\mathbf{z}' \sim p_{\phi}(\mathbf{z}'|\mathbf{x})} \left[\frac{\partial \log p_{\phi}(\mathbf{x}, \mathbf{z}')}{\partial \phi} \right] \right],$$

which can be approximated by first sampling from VAEBM using MCMC (i.e., $\mathbf{x} \sim p_{\theta,\phi}(\mathbf{x})$) and then sampling from the true posterior of the VAE (i.e., $\mathbf{z}' \sim p_{\phi}(\mathbf{z}'|\mathbf{x})$). We cannot directly draw samples from the true posterior, and approaches can be used to draw approximate samples from $p_{\phi}(\mathbf{z}'|\mathbf{x})$: i) We can replace $p_{\phi}(\mathbf{z}'|\mathbf{x})$ with the approximate posterior $q_{\psi}(\mathbf{z}'|\mathbf{x})$. However, the quality of this estimation depends on how well $q_{\psi}(\mathbf{z}'|\mathbf{x})$ matches the true posterior on samples generated by $p_{\theta,\phi}(\mathbf{x}, \mathbf{z})$, which can be very different from the real data samples; ii) alternatively, we can use MCMC sampling to sample $\mathbf{z}' \sim p_\phi(\mathbf{z}'|\mathbf{x})$. To speed up MCMC, we can initialize the $\mathbf{z}'$ samples in MCMC with the original $\mathbf{z}$ samples that were drawn in the outer expectation (i.e., $\mathbf{x}, \mathbf{z} \sim p_{\theta,\phi}(\mathbf{x}, \mathbf{z})$). However, with this approach, the computational complexity of the gradient estimation for the negative phase is doubled, as we now require running MCMC twice, once for $\mathbf{x}, \mathbf{z} \sim p_{\theta,\phi}(\mathbf{x}, \mathbf{z})$ and again for $\mathbf{z}' \sim p_\phi(\mathbf{z}'|\mathbf{x})$.

Two-stage Training of VAEBM:

To avoid the computational complexity of estimating this term, for example with a second round of MCMC, we propose a two-stage algorithm for training VAEBM. In the first stage, we train the VAE model in our VAEBM by maximizing the $\mathcal{L}_{\text{vae}}(\mathbf{x}, \theta, \phi)$ term in Equation 4.14. This term is identical to the VAE's objective, thus, the parameters ϕ and ψ are trained with the usual ELBO. In the second stage, we keep the VAE model fixed and only train the EBM component. Since ϕ is now fixed, we only require optimizing $\mathcal{L}_{\text{EBM}}(\mathbf{x}, \theta, \phi)$ w.r.t. θ, the parameters of the energy function. The gradient of $L(\theta) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\mathcal{L}_{\text{EBM}}(\mathbf{x}, \theta, \phi)]$ w.r.t. θ is:

$$\partial_\theta L(\theta) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[-\partial_\theta f_\theta\left(\mathbf{x}\right)\right] + \mathbb{E}_{\mathbf{x}\sim p_{\theta,\phi}(\mathbf{x})}\left[\partial_\theta f_\theta\left(\mathbf{x}\right)\right], \tag{4.17}$$

which decomposes into a positive and a negative phase.

We can entirely avoid the additional computational complexity and the complications of estimating $\frac{\partial}{\partial\phi}\log Z_{\theta,\phi}$, if we assume that the VAE is held fixed when training the EBM component of our VAEBM. This way, we require running MCMC only to sample $\mathbf{x} \sim p_{\theta,\phi}(\mathbf{x}, \mathbf{z})$ to compute $\frac{\partial}{\partial\theta}\log Z_{\theta,\phi}$.

Besides avoiding the difficulties of estimating the full gradient of $\log Z_{\theta,\phi}$, two-stage training has additional advantages. As we discussed in Sectionapter 2.3, updating θ is computationally expensive, as each update requires an iterative MCMC procedure to draw samples from the model. The first stage of our training minimizes the distance between the VAE model and the data distribution, and in the second stage, the EBM further reduces the mismatch between the model and the data distribution. As the pre-trained VAE $p_\phi(\mathbf{x})$ provides a good approximation to $p_{\text{data}}(\mathbf{x})$ already, we expect that a relatively small number of expensive updates for training ψ is needed. Moreover, the pre-trained VAE provides a latent space with an effectively lower dimensionality and a smoother distribution than the data distribution, which facilitates a more efficient MCMC. We will discuss this in the following section.

Reparameterized sampling in the negative phase:

For gradient estimation in the negative phase, we can draw samples from the model using MCMC. Naively, we can perform ancestral sampling, first sampling from the prior $p_\phi(\mathbf{z})$, then running MCMC for $p_\phi(\mathbf{x}|\mathbf{z})e^{-f_\theta(\mathbf{x})}$ in $\mathbf{x}$-space. This is problematic, since $p_\phi(\mathbf{x}|\mathbf{z})$ is often sharp and MCMC cannot mix when the conditioning $\mathbf{z}$ is fixed.

To overcome the issue, we instead run the MCMC iterations in the joint space of $\mathbf{z}$ and $\mathbf{x}$. Furthermore, we accelerate the sampling procedure using reparameterization for both $\mathbf{x}$ and the latent variables $\mathbf{z}$. Recall that when sampling from the VAE, we first sample $\mathbf{z} \sim p_\phi(\mathbf{z})$ from the prior and then sample $\mathbf{x} \sim p_\phi(\mathbf{x}|\mathbf{z})$. This sampling scheme can be reparameterized by sampling from a fixed noise distribution (e.g., $(\epsilon_{\mathbf{z}}, \epsilon_{\mathbf{x}}) \sim p_\epsilon = \mathcal{N}(0, \mathbf{I})$) and deterministic transformations T_ϕ such that

$$\mathbf{z} = T_\phi^{\mathbf{z}}(\epsilon_{\mathbf{z}}), \quad \mathbf{x} = T_\phi^{\mathbf{x}}(\mathbf{z}(\epsilon_{\mathbf{z}}), \epsilon_{\mathbf{x}}) = T_\phi^{\mathbf{x}}(T_\phi^{\mathbf{z}}(\epsilon_{\mathbf{z}}), \epsilon_{\mathbf{x}}). \tag{4.18}$$

Here, $T_\phi^{\mathbf{z}}$ denotes the transformation defined by the prior that transforms noise ϵ into prior samples $\mathbf{z}$. When the prior is unit Gaussian, $T_\phi^{\mathbf{z}}$ is just the identity transformation. $T_\phi^{\mathbf{x}}$ represents the decoder that transforms noise $\epsilon_{\mathbf{x}}$ into samples $\mathbf{x}$, given prior samples $\mathbf{z}$.

We can apply the same reparameterization when sampling from $p_{\theta,\phi}(\mathbf{x}, \mathbf{z})$. This corresponds to sampling $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$ from the "base distribution":

$$h_{\theta,\phi}\left(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}\right) \propto e^{-f_\theta\left(T_\theta^{\mathbf{x}}(T_\phi^{\mathbf{z}}(\epsilon_{\mathbf{z}}),\epsilon_{\mathbf{x}})\right)} p_\epsilon\left(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}\right), \tag{4.19}$$

and then transforming them to $\mathbf{x}$ and $\mathbf{z}$ via Eq. 4.18.

Here we will provide a derivation. Suppose we draw the re-parametrization variables $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}) \sim p_\epsilon(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$. For convenience, we denote

$$T_\phi(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}) = \left(T_\phi^{\mathbf{x}}(T_\phi^{\mathbf{z}}(\epsilon_{\mathbf{z}}), \epsilon_{\mathbf{x}}), T_\phi^{\mathbf{z}}(\epsilon_{\mathbf{z}})\right) = (\mathbf{x}, \mathbf{z}). \tag{4.20}$$

Since T_ϕ is a deterministic and invertible transformation that maps $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$ to $(\mathbf{x}, \mathbf{z})$, by the change of variables formula, we can write

$$p_\phi(\mathbf{x}, \mathbf{z}) = p_\epsilon(T_\phi^{-1}(\mathbf{x}, \mathbf{z})) \left|\det\left(J_{T_\phi^{-1}}(\mathbf{x}, \mathbf{z})\right)\right|, \tag{4.21}$$

where $J_{T_\phi^{-1}}$ is the Jacobian of T_ϕ^{-1}. Consider a Gaussian distribution as a simple example: if $\mathbf{z} \sim \mathcal{N}(\mu_{\mathbf{z}}, \sigma_{\mathbf{z}})$ and $\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mu_{\mathbf{x}}(\mathbf{z}), \sigma_{\mathbf{x}}(\mathbf{z}))$, then

$$\mathbf{z} = T_\phi^{\mathbf{z}}(\epsilon_{\mathbf{z}}) = \mu_{\mathbf{z}} + \sigma_{\mathbf{z}} \cdot \epsilon_{\mathbf{z}}, \quad \mathbf{x} = T_\phi^{\mathbf{x}}(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}) = \mu_{\mathbf{x}}(\mathbf{z}) + \sigma_{\mathbf{x}}(\mathbf{z}) \cdot \epsilon_{\mathbf{x}},$$

and

$$J_{T_\phi^{-1}}(\mathbf{x}, \mathbf{z}) = [\sigma_{\mathbf{x}}(\mathbf{z})^{-1}, \sigma_{\mathbf{z}}^{-1}].$$

Recall that the generative model of our EBM is

$$p_{\theta,\phi}(\mathbf{x}, \mathbf{z}) = \frac{e^{-f_\theta(\mathbf{x})} p_\phi(\mathbf{x}, \mathbf{z})}{Z_{\theta,\phi}}. \tag{4.22}$$

We can apply the change of variable to $p_{\theta,\phi}(\mathbf{x}, \mathbf{z})$ to a distribution in $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$ space:

$$h_{\theta,\phi}(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}) = p_{\theta,\phi}(T_{\phi}(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})) \left|\det\left(J_{T_{\phi}}\left(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}\right)\right)\right|, \tag{4.23}$$

where $J_{T_{\phi}}$ is the Jacobian of T_{ϕ}. Since we have the relation

$$J_{\mathbf{f}^{-1}} \circ \mathbf{f} = J_{\mathbf{f}}^{-1} \tag{4.24}$$

for invertible function $\mathbf{f}$, we have that

$$\begin{aligned}
h_{\theta,\phi}(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}) &= p_{\theta,\phi}(T_{\phi}(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})) \left|\det\left(J_{T_{\phi}}\left(\epsilon_{\mathbf{z}}, \epsilon_{\mathbf{x}}\right)\right)\right| & (4.25)\\
&= \frac{1}{Z_{\theta,\phi}} e^{-f_{\theta}(T_{\phi}(\epsilon_{\mathbf{x}},\epsilon_{\mathbf{z}}))} p_{\phi}(T_{\phi}(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})) \left|\det\left(J_{T_{\phi}}\left(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}\right)\right)\right| & (4.26)\\
&= \frac{1}{Z_{\theta,\phi}} e^{-f_{\theta}(T_{\phi}(\epsilon_{\mathbf{x}},\epsilon_{\mathbf{z}}))} p_{\epsilon}(T_{\theta}^{-1}(\mathbf{x}, \mathbf{z})) \left|\det\left(J_{T_{\theta}^{-1}}\left(\mathbf{x}, \mathbf{z}\right)\right)\right| \left|\det\left(J_{T_{\phi}}\left(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}\right)\right)\right| & (4.27)\\
&= \frac{1}{Z_{\theta,\phi}} e^{-f_{\theta}(T_{\phi}(\epsilon_{\mathbf{x}},\epsilon_{\mathbf{z}}))} p_{\epsilon}(T_{\theta}^{-1}(\mathbf{x}, \mathbf{z})) & (4.28)\\
&= \frac{1}{Z_{\theta,\phi}} e^{-f_{\theta}(T_{\phi}(\epsilon_{\mathbf{x}},\epsilon_{\mathbf{z}}))} p_{\epsilon}(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}), & (4.29)
\end{aligned}$$

which is the distribution in Equation 4.19. After we obtained samples $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$ from the distribution in Equation 4.19, we obtain $(\mathbf{x}, \mathbf{z})$ by applying the transformation T_{θ} in Equation 4.18. An illustration of VAEBM with reparametrization is shown in Figure

An alternative, but simpler explanation for the reparamterization with $\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}$ is also presented. Suppose we have $\mathbf{x} = T(\epsilon)$ where ϵ could be anything with known density $p(\epsilon)$. Then $\mathbb{E}[h(\mathbf{x})] = \mathbb{E}[h(T(\epsilon))]$, for any function h. So $\int h(\mathbf{x})p(\mathbf{x})d\mathbf{x} = \int h(T(\epsilon))p(\epsilon)d\epsilon$, and with the tilting: $\int h(\mathbf{x})p(\mathbf{x})e^{f(\mathbf{x})}dx = \int h(T(\epsilon))e^{f(T(\epsilon))}p(\epsilon)d\epsilon$. So if we sample from $e^{f(T(\epsilon))}p(\epsilon)$ and pass it through $T(\epsilon)$, we can get a sample from the tilted distribution $p(\mathrm{v})e^{f(\mathrm{v})}$.

Note that $\epsilon_{\mathbf{z}}$ and $\epsilon_{\mathbf{x}}$ have the same scale, as $p_{\epsilon}\left(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}}\right)$ is a standard Normal distribution,

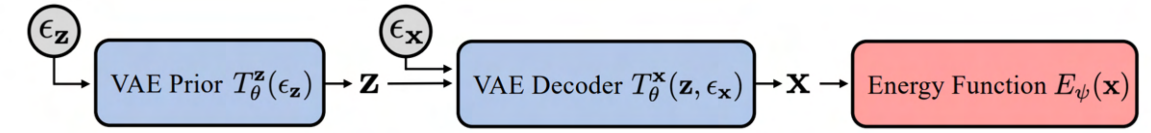

$\epsilon_{\mathbf{z}}$
VAE Prior $T_\theta^{\mathbf{z}}(\epsilon_{\mathbf{z}})$
$\mathbf{z}$
$\epsilon_{\mathbf{x}}$
VAE Decoder $T_\theta^{\mathbf{x}}(\mathbf{z}, \epsilon_{\mathbf{x}})$
$\mathbf{x}$
Energy Function $E_\psi(\mathbf{x})$

Figure 4.4: Our VAEBM is composed of a VAE generator (including the prior and decoder) and an energy function that operates on samples $\mathbf{x}$ generated by the VAE. The VAE component is trained first, using the standard VAE objective; then, the energy function is trained while the generator is fixed. Using the VAE generator, we can express the data variable $\mathbf{x}$ as a deterministic function of white noise samples $\epsilon_{\mathbf{z}}$ and $\epsilon_{\mathbf{x}}$. This allows us to reparameterize sampling from our VAEBM by sampling in the joint space of $\epsilon_{\mathbf{z}}$ and $\epsilon_{\mathbf{x}}$.

while the scales of $\mathbf{x}$ and $\mathbf{z}$ can be very different. Thus, running MCMC sampling with this reparameterization in the $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$-space has the benefit that we do not need to tune the sampling scheme (e.g., step size in LD) for each variable. This is particularly helpful when $\mathbf{z}$ itself has multiple groups, as in our case. We will compare sampling in $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$-space and in $(\mathbf{x}, \mathbf{z})$-space in the following paragraphs.

Comparison of Sampling in $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$-space and in $(\mathbf{x}, \mathbf{z})$-space:

Above we showed that sampling from $h_{\psi,\theta}(\mathbf{x}, \mathbf{z})$ is equivalent to sampling from $h_{\psi,\theta}(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$ and applying the appropriate variable transformation. Here, we further analyze the connections between sampling from these two distributions with Langevin dynamics. Since each component of $\mathbf{x}$ and $\mathbf{z}$ can be re-parametrzied with scaling and translation of standard Gaussian noise, without loss of generality, we assume a variable $\mathbf{c}$ ($\mathbf{c}$ can be a single latent variable in $\mathbf{z}$ or a single pixel in $\mathbf{x}$) and write

$$\mathbf{c} = \mu + \sigma\epsilon.$$

Suppose we sample in the ϵ space with energy function f on $\mathbf{c}$ and step size η. The update for ϵ is

$$\epsilon_{t+1} = \epsilon_t - \frac{\eta}{2}\nabla_\epsilon f + \sqrt{\eta}\omega_t, \quad \omega_t \sim \mathcal{N}(0, \mathbf{I}).$$

Now we plug ϵ_{t+1} into the expression of $\mathbf{c}$ while noting that $\nabla_\epsilon f = \sigma \nabla_{\mathbf{c}} f$. We obtain

$$\begin{aligned}
\mathbf{c}_{t+1} &= \mu + \sigma\epsilon_{t+1} = \mu + \sigma\left(\epsilon_t - \frac{\eta}{2}\nabla_\epsilon f + \sqrt{\eta}\omega_t\right) \\
&= \mu + \sigma\epsilon_t - \frac{\sigma^2\eta}{2}\nabla_{\mathbf{c}} f + \sqrt{\eta\sigma^2}\omega_t \\
&= \mathbf{c}_t - \frac{\sigma^2\eta}{2}\nabla_{\mathbf{c}} f + \sqrt{\eta\sigma^2}\omega_t.
\end{aligned}$$

Therefore, we see that running Langevin dynamics in $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$-space is equivalent to running Langevin dynamics in $(\mathbf{x}, \mathbf{z})$-space with step size for each component of $\mathbf{z}$ and $\mathbf{x}$ adjusted by its variance. However, considering the high dimensionality of $\mathbf{x}$ and $\mathbf{z}$, the step size adjustment is difficult to implement.

The analysis above only considers a variable individually. More importantly, our latent variable $\mathbf{z}$ in the prior follows block-wise auto-regressive Gaussian distributions, so the variance of each component in $\mathbf{z}_i$ depends on the value of $\mathbf{z}_{<i}$. We foresee that because of this dependency, using a fixed step size per component of $\mathbf{z}$ will not be effective, even when it is set differently for each component. In contrast, all the components in $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$-space have a unit variance. Hence, a universal step size for all the variables in this space can be used. We will empirically compare sampling in $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$-space and in $(\mathbf{x}, \mathbf{z})$-space in Section 4.4.

4.3.4 An Extension to the Training Objective of VAEBM

In the first stage of training VAEBM, the VAE model is trained by maximizing the training data log-likelihood which corresponds to minimizing an upper bound on $D_{\mathrm{KL}}(p_{\mathrm{data}}(\mathbf{x})||p_\phi(\mathbf{x}))$

w.r.t. ϕ. In the second stage, when we are training the EBM component, we use the VAE model to sample from the joint VAEBM by running the MCMC updates in the joint space of $\epsilon_{\mathbf{z}}$ and $\epsilon_{\mathbf{x}}$. Ideally, we may want to bring $p_\phi(\mathbf{x})$ closer to $p_{\theta,\phi}(\mathbf{x})$ in the second stage, because when $p_\phi(\mathbf{x}) = p_{\theta,\phi}(\mathbf{x})$, we will not need the expensive updates for θ. We can bring $p_\phi(\mathbf{x})$ closer to $p_{\theta,\phi}(\mathbf{x})$ by minimizing $D_{\mathrm{KL}}(p_\phi(\mathbf{x})||p_{\theta,\phi}(\mathbf{x}))$ with respect to ϕ which was recently discussed in the context of an EBM-interpretation of GANs by Che et al. [2020]. To do so, for one training step of updating ϕ, we assume the target distribution $p_{\theta,\phi}(\mathbf{x})$ is fixed and create a copy of ϕ, named ϕ', and we update ϕ' by the gradient:

$$\nabla_{\phi'} D_{\mathrm{KL}}(p_{\phi'}(\mathbf{x})||p_{\theta,\phi}(\mathbf{x})) = \nabla_{\phi'} \mathbb{E}_{\mathbf{x}\sim p_{\phi'}(\mathbf{x})}\left[f_\theta(\mathbf{x})\right]. \tag{4.30}$$

In other words, one update step for ϕ' that minimizes $D_{\mathrm{KL}}(p'_\phi(\mathbf{x})||p_{\theta,\phi}(\mathbf{x}))$ w.r.t. ϕ' can be easily done by drawing samples from $p'_\phi(\mathbf{x})$ and minimizing the energy-function w.r.t. ϕ'. Note that this approach is similar to the generator update in training Wasserstein GANs [Arjovsky et al., 2017]. Due to the nature of adversarial training, the above KL objective will encourage $p_\phi(\mathbf{x})$ to model dominants modes in $p_{\theta,\phi}(\mathbf{x})$, and it may cause $p_\phi(\mathbf{x})$ to drop modes.

A derivation of Equation 4.30 will be given, where we largely follow [Che et al., 2020]. Note that every time we update ϕ, we are actually taking the gradient w.r.t ϕ', which can be viewed as a copy of ϕ and is initialized as ϕ. In particular, we should note that the ϕ in

$h_{\theta,\phi}(\mathbf{x})$ is fixed. Therefore, we have

$$\begin{aligned}
\nabla_{\phi'} D_{\mathrm{KL}}(p_{\phi'}(\mathbf{x})||p_{\theta,\phi}(\mathbf{x})) &= \nabla_{\phi'} \int p_{\phi'}(\mathbf{x}) \Big[\log p_{\phi'}(\mathbf{x}) - \log p_{\theta,\phi}(\mathbf{x})\Big]\, d\mathbf{x} \\
&= \int \Big[\nabla_{\phi'} p_{\phi'}(\mathbf{x})\Big] \Big[\log p_{\phi'}(\mathbf{x}) - \log p_{\theta,\phi}(\mathbf{x})\Big]\, d\mathbf{x} \\
&\quad + \underbrace{\int p_{\phi'}(\mathbf{x}) \Big[\nabla_{\phi'} \log p_{\phi'}(\mathbf{x}) - \nabla_{\phi'} \log p_{\theta,\phi}(\mathbf{x})\Big]\, d\mathbf{x}}_{=0} \qquad (4.31) \\
&= \int \Big[\nabla_{\phi'} p_{\phi'}(\mathbf{x})\Big] \Big[\log p_{\phi'}(\mathbf{x}) - \log p_{\theta,\phi}(\mathbf{x})\Big]\, d\mathbf{x}, \qquad (4.32)
\end{aligned}$$

where the second term in Equation 4.31 is 0 because the $\log p_{\theta,\phi}(\mathbf{x})$ does not depend on ϕ' and the expectation of the score function is 0:

$$\int p_{\phi'}(\mathbf{x}) \nabla_{\phi'} \log p_{\phi'}(\mathbf{x}) d\mathbf{x} = \mathbb{E}_{\mathbf{x}\sim p_{\phi'}(\mathbf{x})} \Big[\nabla_{\phi'} \log p_{\phi'}(\mathbf{x})\Big] = 0.$$

Recall that θ' has the same value as θ before the update, so

$$\begin{aligned}
\log p_{\phi'}(\mathbf{x}) - \log h_{\psi,\theta}(\mathbf{x}) &= \log \left[\frac{p_{\phi'}(\mathbf{x})}{p_{\theta}(\mathbf{x}) e^{-f_{\theta}(\mathbf{x})}}\right] + \log Z_{\theta,\phi} \\
&= f_{\theta}(\mathbf{x}) + \log Z_{\theta,\phi}. \qquad (4.33)
\end{aligned}$$

Plug Equation 4.33 into Equation 4.32, we have

$$\begin{aligned}
\nabla_{\phi'} D_{\mathrm{KL}}(p_{\phi'}(\mathbf{x})||p_{\theta,\phi}(\mathbf{x})) &= \int \nabla_{\phi'} p_{\phi'}(\mathbf{x}) \left[f_{\theta}(\mathbf{x}) + \log Z_{\psi,\theta}\right] d\mathbf{x} \\
&= \nabla_{\phi'} \mathbb{E}_{\mathbf{x}\sim p_{\phi'}(\mathbf{x})} \left[f_{\theta}(\mathbf{x})\right], \qquad (4.34)
\end{aligned}$$

since

$$\int \nabla_{\phi'} p_{\phi'}(\mathbf{x}) \log Z_{\theta,\phi} d\mathbf{x} = \nabla_{\phi'} \log Z_{\theta,\phi} \int p_{\phi'}(\mathbf{x}) d\mathbf{x} = \nabla_{\phi'} \log Z_{\theta,\phi} = 0.$$

Intuitively, the extension described in this section can be summarized as alternatively updating the energy function and VAE's decoder, where the decoder update corresponds to decrease the energy value of its output. In Section 4.4, we will present empirical results of training with an additional loss term that updates the parameter ϕ to minimize $D_{\mathrm{KL}}(p_\phi(\mathbf{x})||p_{\theta,\phi}(\mathbf{x}))$ as explained above.

4.4 Experimental Results

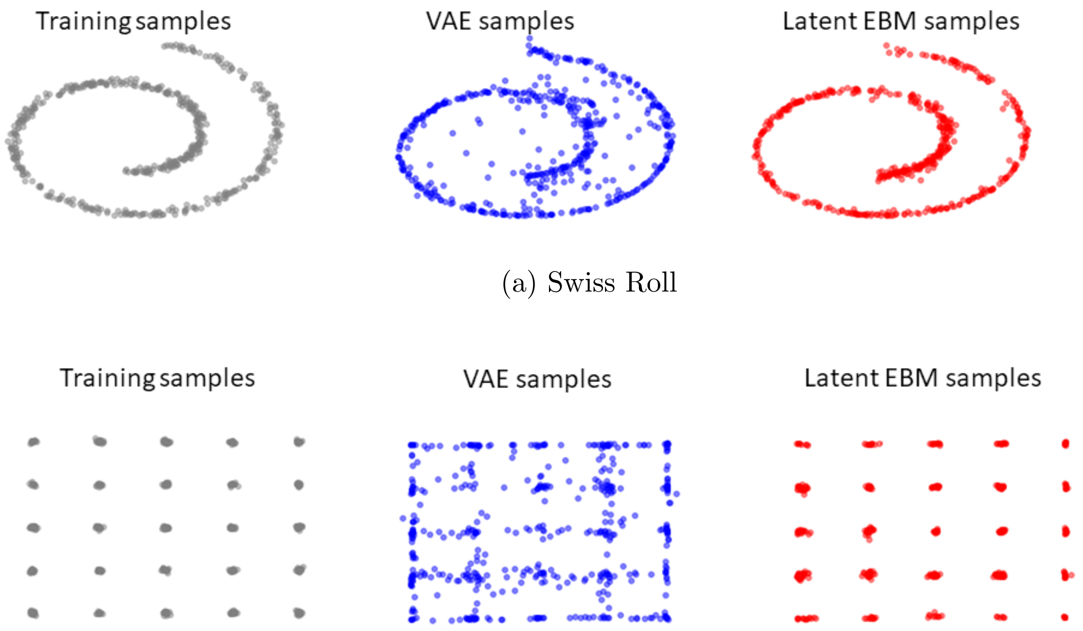
In this section, we evaluate our proposed exponential tilting framework with extensive experiments. We divide our experimental studies into three parts according to different types of base generative models in the exponential tilting framework. Throughout the study, our main focus is on the relative improvements of sampling from the EBMs over sampling from base generative models.

4.4.1 Small VAEs as the Base Model

Toy Datasets

We first present the results of VAEBM when the base generative model is a simple VAE with one layer of stochastic latent variables. 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]. The decoder and the energy function both have simple, fully connected structures as described in Tanaka [2019].

We show qualitative results in Figure 4.5. We observe that 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 from VAEBM, 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 4.5: VAEBMs trained on Swiss Roll and 25-Gaussians datset.

We also compute the test likelihood on 25-Gaussians. Note that VAEBM is an explicit likelihood model with a parameterized density function. However, like other energy-based models, the estimation of the exact likelihood is difficult due to the intractable partition function $\log Z$. One possible way to estimate the partition function is to use Annealed Importance Sampling (AIS) [Neal, 2001]. However, using AIS to estimate $\log Z$ in high-dimensional spaces is challenging. In fact, Du and Mordatch [2019] report that the estimation does not converge in 2 days on CIFAR-10. Furthermore, AIS gives a stochastic lower bound on $\log Z$, and therefore the likelihood computed with this estimated $\log Z$ would be an upper bound for the true likelihood. This makes the estimated likelihood hard to compare with the VAE's likelihood estimate, which is usually a lower bound on the true likelihood [Burda et al., 2015].

In the 2-D domain, the partition function $\log Z$ can be accurately estimated by a numer-

	MNIST	Fashion	CIFAR-10
VAE	18.9	57.1	139.6
VAEBM	16.0	38.1	108.4

Table 4.1: Comparing the FID scores of base VAEs and VAEBMs.

ical integration scheme. For the VAE, we use the IWAE bound [Burda et al., 2015] with 10,000 posterior samples to estimate its likelihood. We use 100,000 test samples from the true distribution to evaluate the likelihood. Our VAEBM obtains the average log-likelihood of **-1.50** nats on test samples, which significantly improves the VAE, whose average test likelihood is **-2.97** nats. As a reference, we also analytically compute the log-likelihood of test samples under the true distribution, and the result is **-1.10** nats.

Image Datasets

We also evaluate the performance of VAEBM on image datasets, including MNIST, Fashion MNIST and CIFAR-10. We show some qualitative results of VAEBMs on top of simple convolutional VAEs in Figure 4.6. From Figure 4.6, we clearly observe that samples generated by VAEBMs have higher quality than samples from base VAEs.

Quantitatively, we compared the FID score of the VAEs and VAEBMs, and results are shown in Table 4.1. We observe that sampling from VAEBMs significantly improves the quality of generated samples over directly sampling from the base VAEs.

Experimental Settings:

For the simple VAE model, 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. For MNIST and Fashion MNIST datasets, we use binary cross-entropy 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

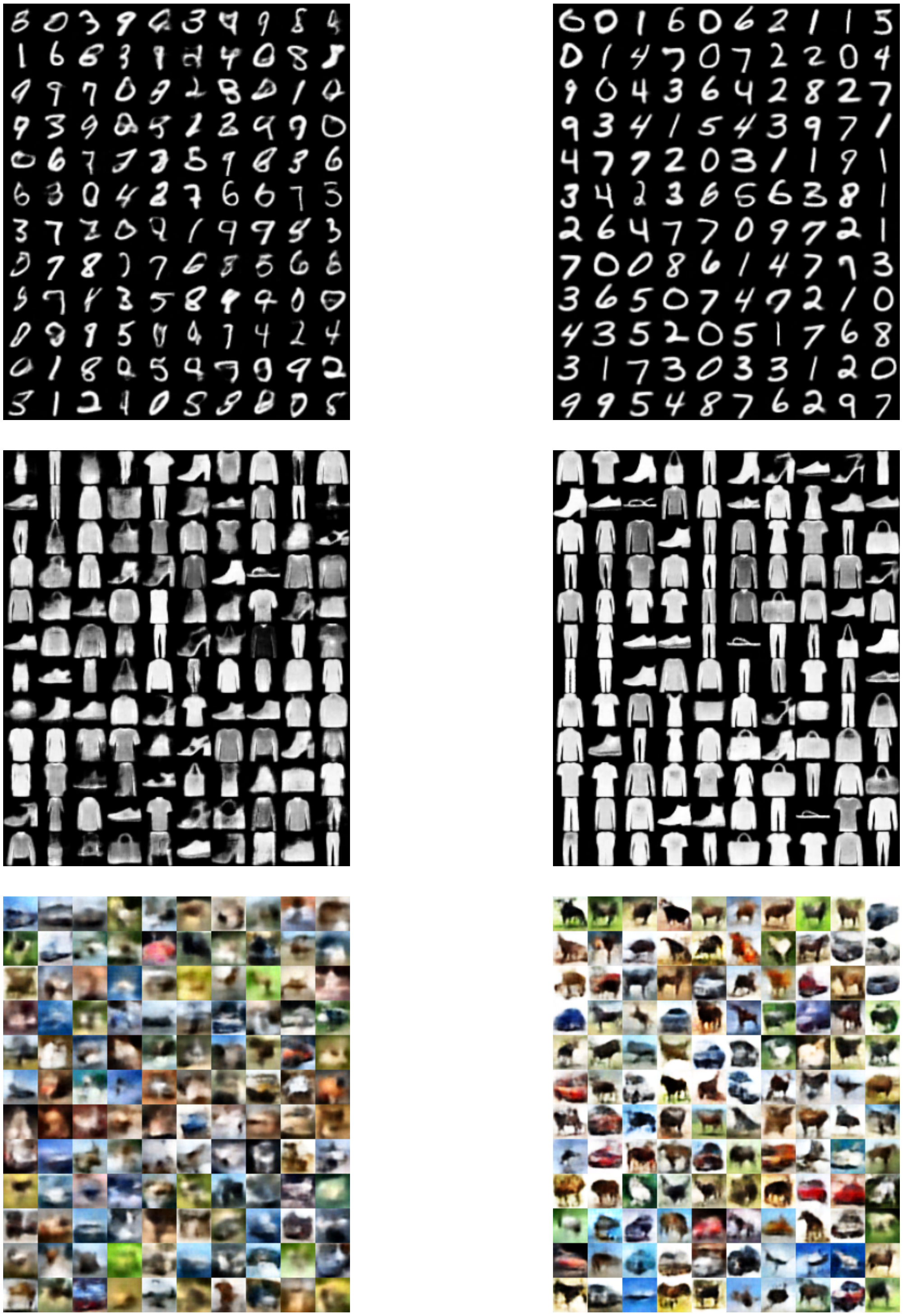


Figure 4.6: Qualitative results of VAEBMs with simple comvolutional VAE as the backbone on MNIST, Fashion MNIST and CIFAR-10. **Left**: samples generated by VAEs. **Right**: samples generated by VAEBMs.

rate 1×10^{-3}.

We use 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 an FC layer. For Langevin dynamics, we use step size 0.01 and run the chain for 60 steps. We find adding a small amount (with a coefficient 0.1) of energy regularization is helpful for avoiding over-fitting early in training. After training, we find sampling latent variables with a longer chain leads to better performances. We generate samples for testing by running the chain for 100 steps.

4.4.2 Normalizing Flows as the Base Model

In this section, we study the exponential tilting with normalizing flows (in particular, GLOW [Kingma and Dhariwal, 2018]) as the base generative model on image datasets. 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 4.7. We clearly observe that samples generated by the EBMs have higher quality than samples from the base GLOW model. 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 4.8, where we generate samples for every ten iterations. Apparently, the Langevin dynamics is going towards latent variables that produce more semantically meaningful and sharp samples.

Quantitatively, we compared the FID score of the GLOWs and GLOWs tilted with EBMs, and results are shown in Table 4.2, where we observe significant improvements made by the

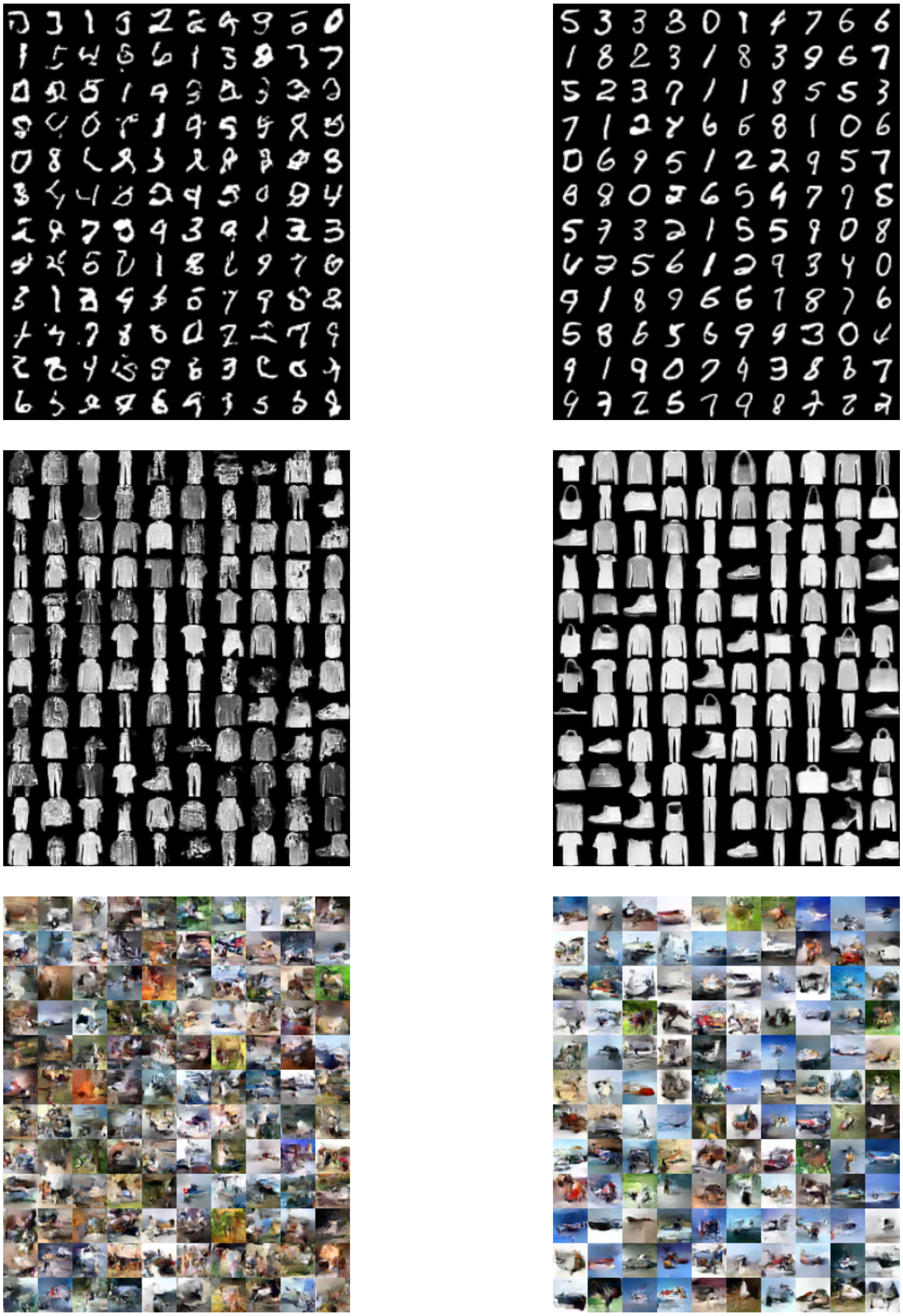


Figure 4.7: Qualitative results of exponential tilting with GLOW backbone on MNIST, Fashion MNIST and CIFAR-10. **Left**: samples generated by from GLOWs. **Right**: samples generated by the EBMs.

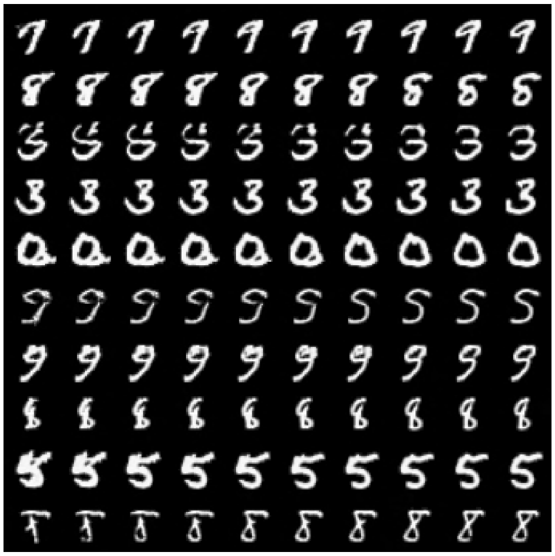


Figure 4.8: MNIST Langevin dynamics visualization, initialized at samples from prior (the leftmost column).

	MNIST	Fashion	CIFAR-10
GLOW	29.4	58.7	76.2
GLOW + EBM	12.3	41.6	67.8

Table 4.2: Comparing the FID scores of base GLOWs and GLOWs tilted with EBMs.

exponential tilting.

Experimental Settings:

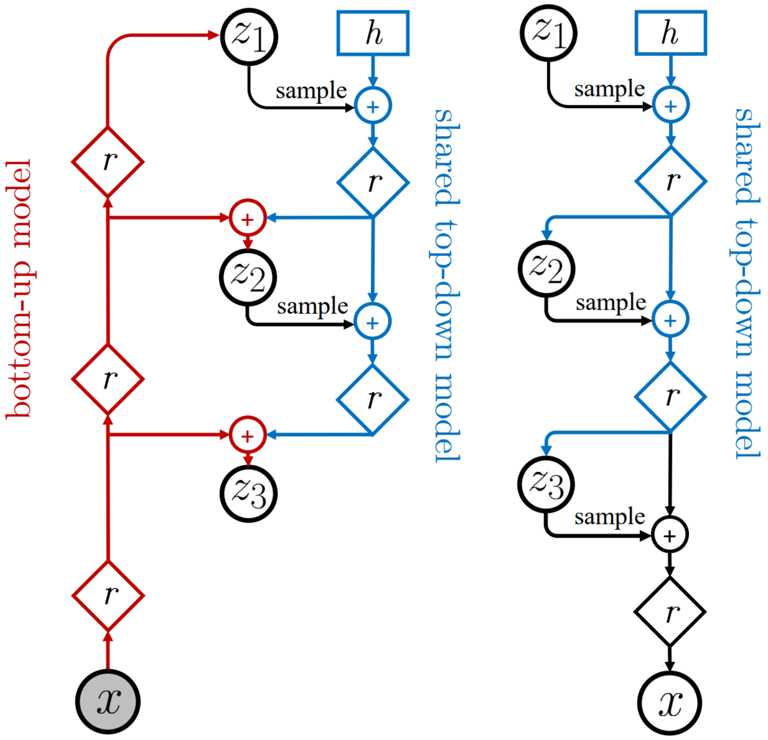
We train GLOW models following the settings provided in Nalisnick et al. [2018]. For MNIST and Fashion MNIST,we use a GLOW architecture of 2 blocks of 16 affine coupling layers, squeezing the spatial dimension in between the 2 blocks. For the coupling function, we use a 3-layer Highway network with 64 hidden channels. 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. Note that 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}.

For the EBM component, we adopt the same setting as in Sectionapter 4.4.1.

4.4.3 Large Hierarchical VAEs as the Base Model

Our exponential tilting framework is constrained by the capacity of the base generative model. In previous sections where simple VAEs with one-layer latent variables or normalizing flows served as the base model, the resulting exponential tilting models cannot obtain sample quality competitive to GANs because of the limitation of base models. For example, a simple VAE even cannot reconstruct data well, which significantly restricts its ability to generate new samples. In this section, we push the limit of exponential tilting by adopting large hierarchical VAEs as the base model. In particular, we try to design VAEBM with NVAE [Vahdat and Kautz, 2020] as the backbone. NVAE is currently the most powerful VAE model. It increases the expressivity of both prior and approximate posterior using hierarchical latent variables [Kingma et al., 2016] where $\mathbf{z}$ is decomposed into a set of disjoint groups, $\mathbf{z} = \{\mathbf{z}_1, \mathbf{z}_1, \ldots, \mathbf{z}_L\}$, and the prior $p_\theta(\mathbf{z}) = \prod_l p_\theta(\mathbf{z}_l|\mathbf{z}_{<l})$ and the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x}) = \prod_l q_\phi(\mathbf{z}_l|\mathbf{z}_{<l}, \mathbf{x})$ are defined using autoregressive distributions over the groups. The conditioning is implemented with the combination of samples and deterministic networks. See Figure 4.9 for an illustration on the implementation of conditioning. NVAE obtains impressive results on likelihood modeling (and hence nearly perfect reconstruction), however, the sample quality of NVAE is still limited. We hope that the exponential tilting framework will significantly improve the sample quality of NVAE and achieve competitive performance with GANs.

In this section, we evaluate our proposed VAEBM with NVAE backbone through comprehensive experiments. Specifically, we benchmark sample quality and provide detailed ablation studies on training techniques. In addition, we study mode coverage of our model



(a) Bidirectional Encoder (b) Generative Model

Figure 4.9: The neural networks implementing an encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and generative model $p_\theta(\mathbf{x}, \mathbf{z})$ for a 3-group hierarchical VAE. Figure taken from Vahdat and Kautz [2020]. Blocks with 'r' denotes residual neural networks. Blocks with '+' denotes feature combination (e.g., concatenation). Blocks with 'h ' denotes trainable parameters.

and test for spurious modes. Note that in NVAE, the prior distribution is a group-wise auto-regressive Gaussian, and the conditional pixel-wise distributions in $\mathbf{x}$ are also Gaussian. Therefore, the reparameterization introduced in Section 4.3.3 corresponds to shift and scale transformations.

Image Generation:

In Table 4.3, we quantitatively compare the sample quality of VAEBM with different generative models on (unconditional) CIFAR-10. We adopt Inception Score (IS) [Salimans et al., 2016] and FID [Heusel et al., 2017] as quantitative metrics. We observe that our VAEBM outperforms previous EBMs and other explicit likelihood-based models by a large margin. Note that introducing persistent chains during training only leads to slight improvement, while Du and Mordatch [2019] rely on persistent chains with a sample replay buffer. This is likely due to the efficiency of sampling in latent space. Our model also produces significantly better samples than NVAE, the VAE component of our VAEBM, implying a significant impact of our proposed energy-based refinement. We also compare our model with state-of-the-art GANs and recently proposed score-based models, and we obtain comparable or better results. Thus, we largely close the gap to GANs and score-models, while maintaining the desirable properties of models trained with maximum likelihood, such as fast sampling and better mode coverage.

Qualitative samples generated by our model are shown in Figure 4.10 and intermediate samples along MCMC chains in Figure 4.11. We find that VAEBM generates good samples by running only a few MCMC steps. Initializing MCMC chains from the pre-trained VAE also helps quick equilibration.

We also train VAEBM on larger images, including CelebA 64, CelebA HQ 256 [Karras et al., 2017] and LSUN Church 64 [Yu et al., 2015]. We report the FID scores for CelebA 64 and CelebA HQ 256 in Tables 4.4 and 4.5. On CelebA 64, our model obtains results

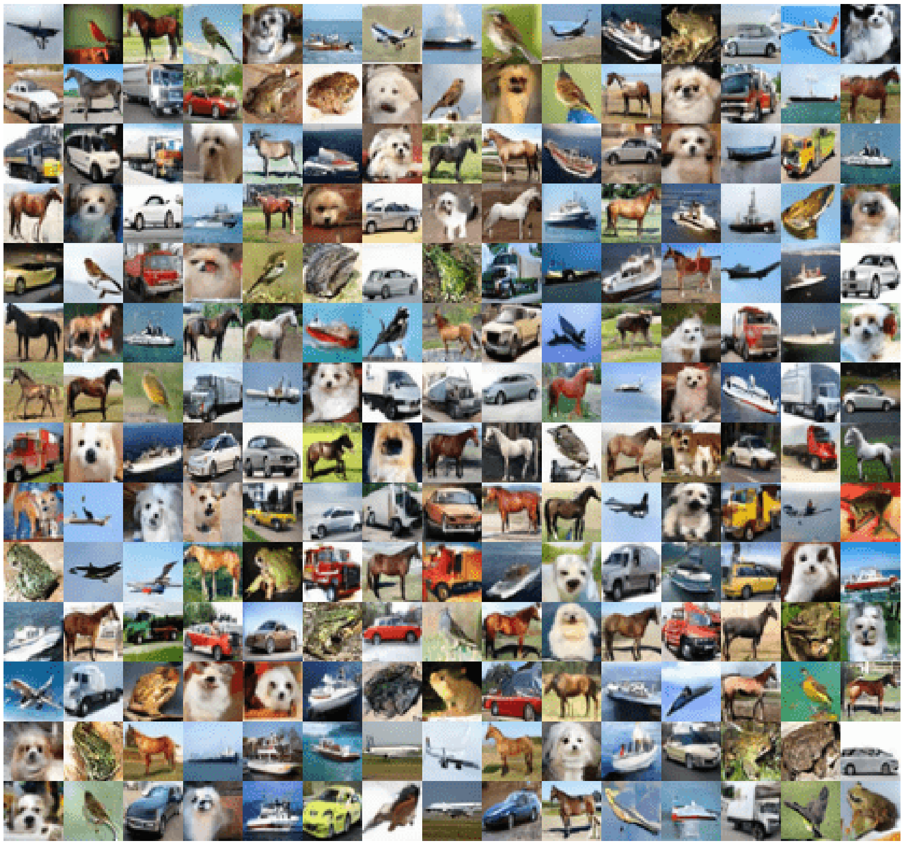


Figure 4.10: CIFAR-10 samples generated by VAEBM with NVAE backbone.

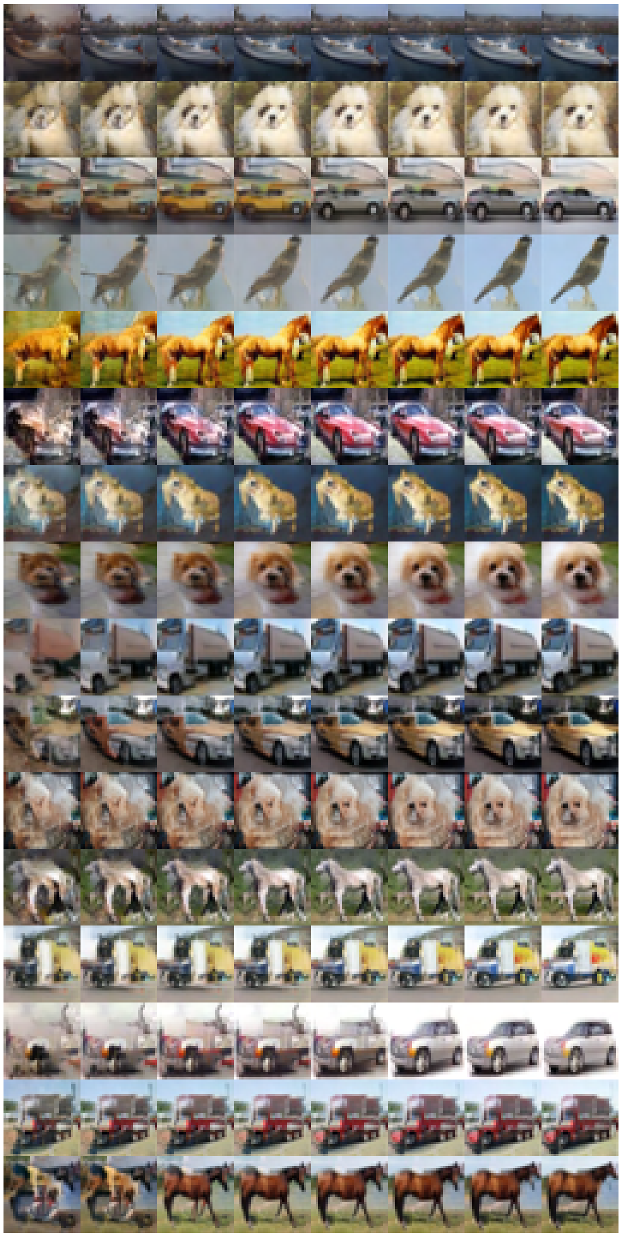


Figure 4.11: Visualizing MCMC sampling chains. Samples are generated by running 16 LD steps. Chains are initialized with pre-trained VAE. We show intermediate samples at every 2 steps.

Table 4.3: Comparing VAEBM and other generative models with IS and FID scores for unconditional generation on CIFAR-10.

	Model	IS↑	FID↓
Ours	VAEBM w/o persistent chain	8.21	12.26
	VAEBM w/ persistent chain	8.43	12.19
EBMs	IGEBM [Du and Mordatch, 2019]	6.02	40.58
	EBM with short-run MCMC [Nijkamp et al., 2019]	6.21	-
	F-div EBM [Yu et al., 2020a]	8.61	30.86
	FlowCE [Gao et al., 2020]	-	37.3
	FlowEBM [Nijkamp et al., 2022]	-	78.12
	GEBM [Arbel et al., 2020]	-	23.02
	Divergence Triangle [Han et al., 2020]	-	30.1
Other Likelihood Models	Glow [Kingma and Dhariwal, 2018]	3.92	48.9
	PixelCNN [Oord et al., 2016]	4.60	65.93
	NVAE [Vahdat and Kautz, 2020]	5.51	51.67
	VAE with EBM prior [Pang et al., 2020]	-	70.15
Score-based Models	NCSN [Song and Ermon, 2019]	8.87	25.32
	NCSN v2 [Song and Ermon, 2020]	-	31.75
	Multi-scale DSM [Li et al., 2019a]	8.31	31.7
	Denoising Diffusion [Ho et al., 2020]	9.46	3.17
GAN-based Models	SNGAN [Miyato et al., 2018]	8.22	21.7
	SNGAN+DDLS [Che et al., 2020]	9.09	15.42
	SNGAN+DCD [Song et al., 2020b]	9.11	16.24
	BigGAN [Brock et al., 2018]	9.22	14.73
	StyleGAN2 w/o ADA [Karras et al., 2020a]	8.99	9.9
Others	PixelIQN [Ostrovski et al., 2018]	5.29	49.46
	MoLM [Ravuri et al., 2018]	7.90	18.9

comparable with the best GANs. Although our model obtains worse results than some advanced GANs on CelebA HQ 256, we significantly reduce the gap between likelihood based models and GANs on this dataset. On LSUN Church 64, we obtain FID 13.51, which significantly improves the NVAE baseline FID 41.3.

We present qualitative samples of CelebA 64, CelebA HQ 256 and LSUN Church 64 in Figure 4.12, 4.13 and 4.14 respectively. We observe that the generated images are realistic and sharp. In Figure 4.15 and 4.16, we visualize the effect of sampling from VAEBM by displaying sample pairs before and after running Lanegvin dynamics, where we clearly see that the EBM significantly refines the base VAE.

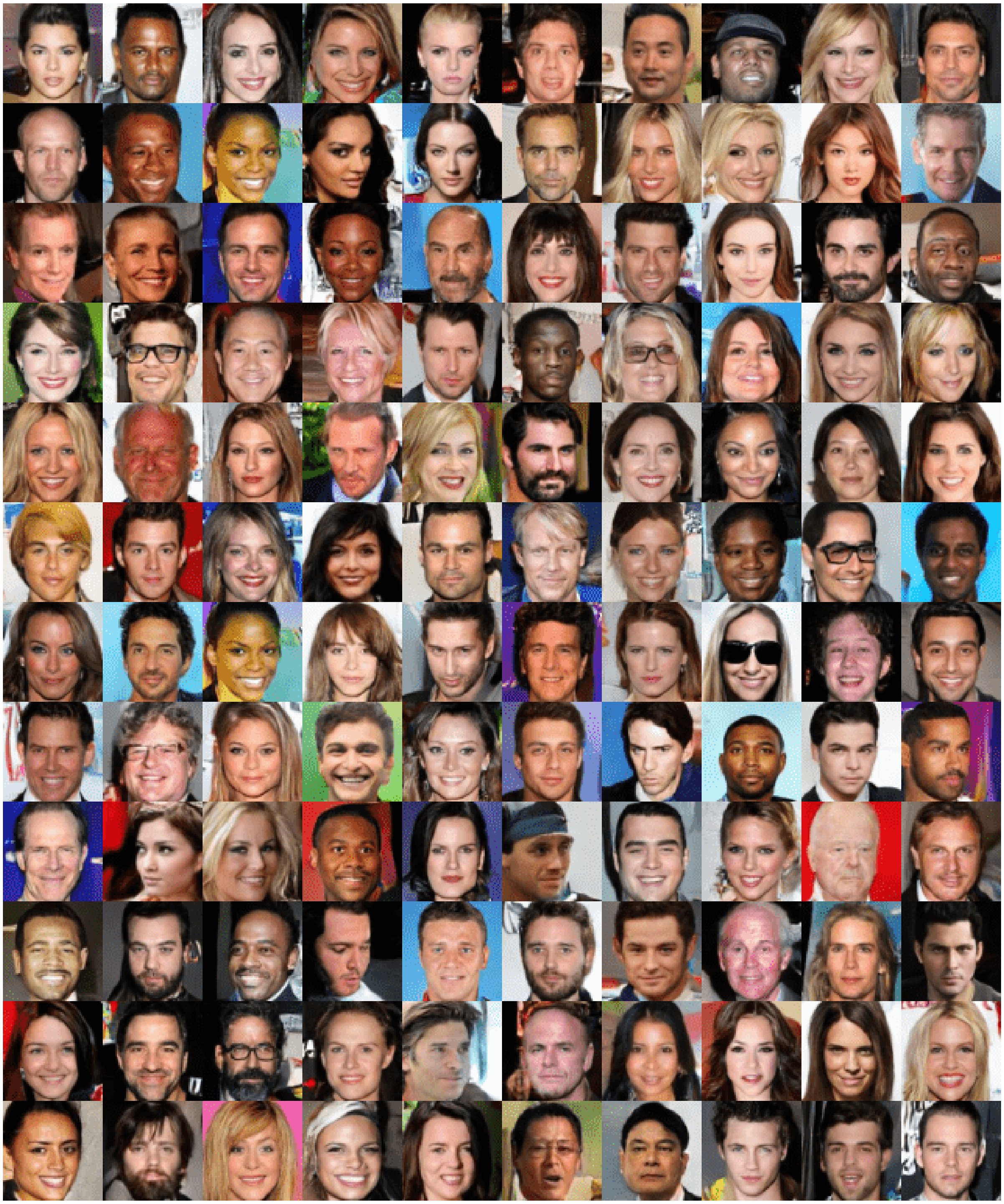


Figure 4.12: CelebA 64 samples generated by VAEBM with NVAE backbone.

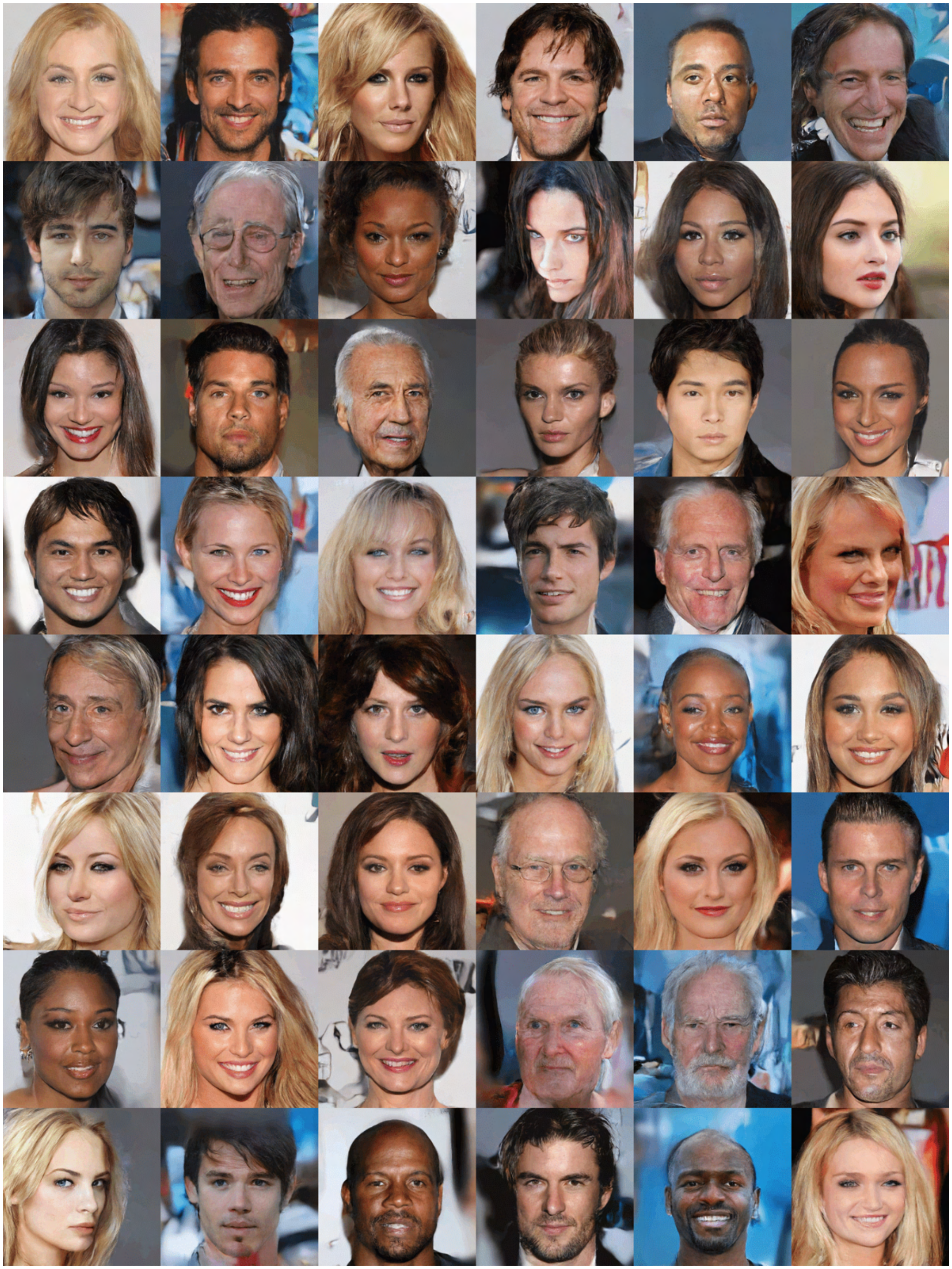


Figure 4.13: CelebA HQ 256 samples generated by VAEBM with NVAE backbone.

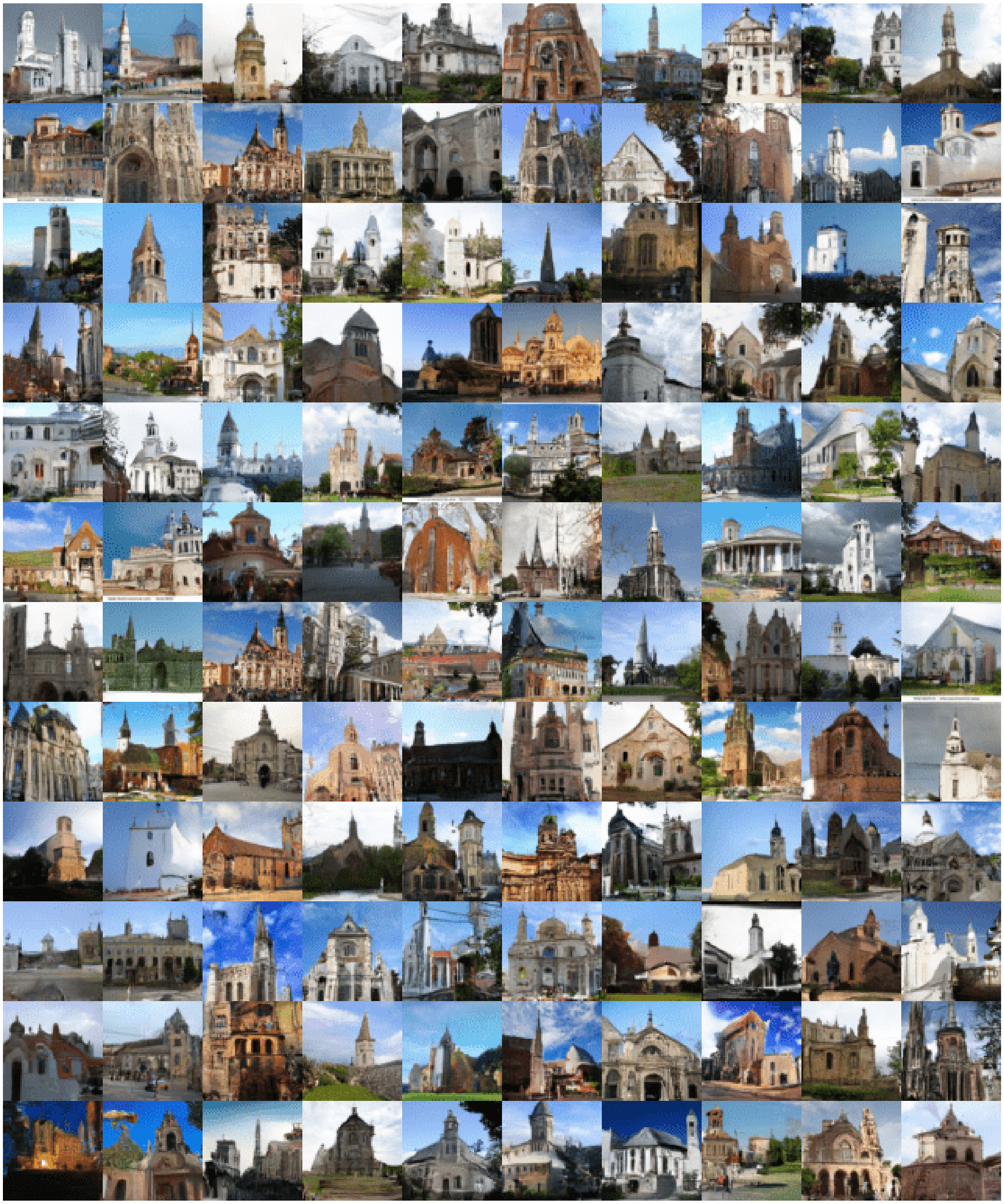


Figure 4.14: LSUN church 64 samples generated by VAEBM with NVAE backbone.

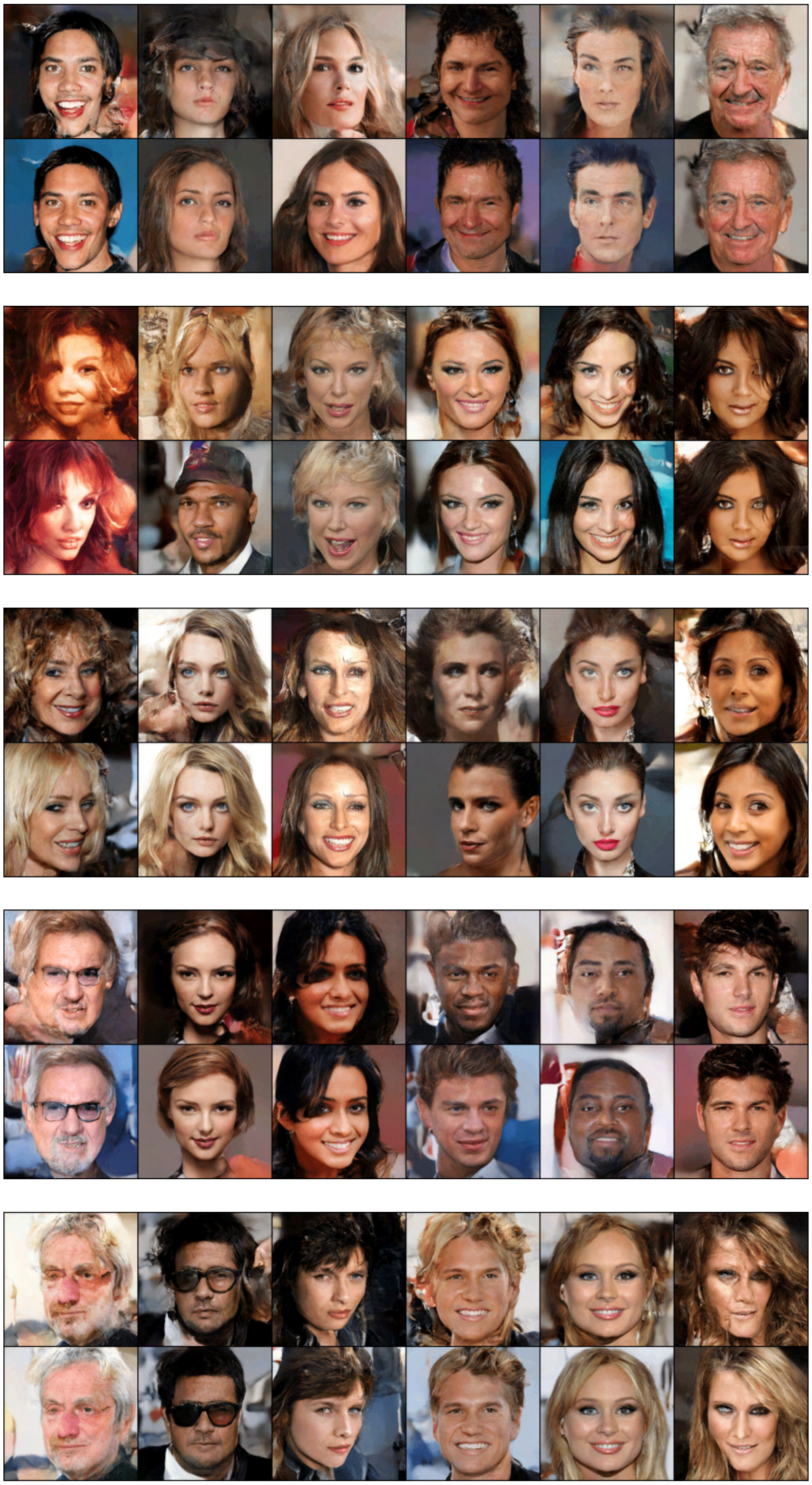


Figure 4.15: Visualizing the effect of MCMC sampling on CelebA HQ 256 dataset. Samples are generated by initializing MCMC with full temperature VAE samples. MCMC sampling fixes the artifacts of VAE samples, especially on hairs.

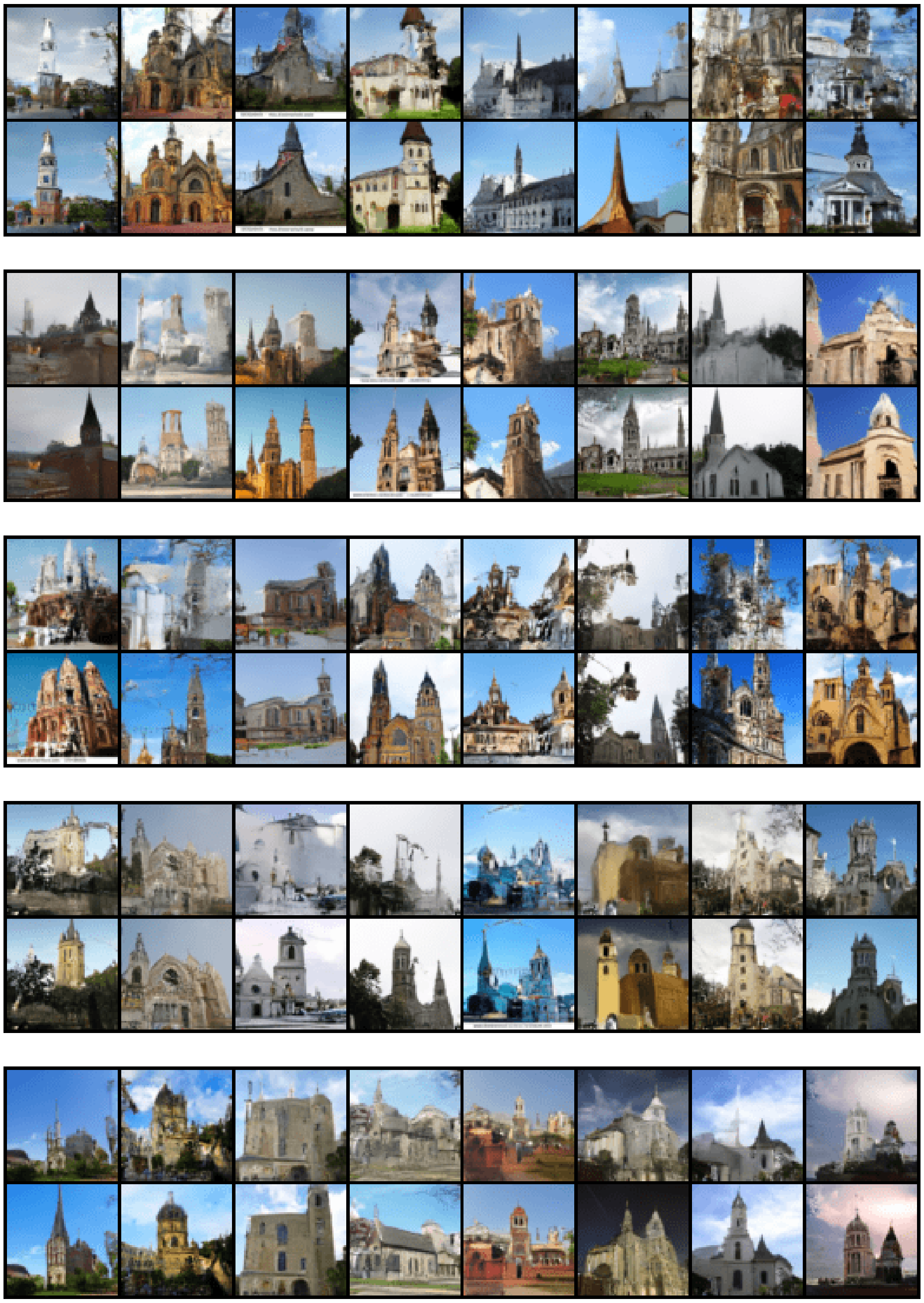


Figure 4.16: Visualizing the effect of MCMC sampling on LSUN Church 64 dataset. For each subfigure, the top row contains initial samples from the VAE, and the bottom row contains corresponding samples after MCMC. We observe that MCMC sampling fixes the corrupted initial samples and refines the details.

Table 4.4: Generative performance of VAEBM on CelebA 64

Model	FID↓
VAEBM (ours)	5.31
NVAE ([Vahdat and Kautz, 2020])	14.74
Flow CE ([Gao et al., 2020])	12.21
Divergence Triangle ([Han et al., 2020])	24.7
NCSNv2 ([Song and Ermon, 2020])	26.86
COCO-GAN ([Lin et al., 2019])	4.0
QA-GAN ([Parimala and Channappayya, 2019])	6.42

Table 4.5: Generative performance of VAEBM on CelebA HQ 256

Model	FID↓
VAEBM (ours)	20.38
NVAE ([Vahdat and Kautz, 2020])	45.11
GLOW ([Kingma and Dhariwal, 2018])	68.93
Advers. LAE ([Pidhorskyi et al., 2020])	19.21
PGGAN ([Karras et al., 2017])	8.03

Ablation Studies:

To better understand the VAEBM model, in Table 4.6, we compare VAEBM to several closely related baselines. All the experiments here are performed on CIFAR-10, and for simplicity, we use smaller models than those used in Table 4.3.

Data space vs. augmented space: One key difference between VAEBM and previous work such as Du and Mordatch [2019] is that our model is defined on the augmented space $(\mathbf{x}, \mathbf{z})$, while their EBM only involves $\mathbf{x}$. Since we pre-train the VAE, one natural question is whether our strong results are due to good initial samples $\mathbf{x}$ from the VAE, which are used to launch the MCMC chains. To address this, we train an EBM purely on $\mathbf{x}$ as done in Du and Mordatch [2019]. We also train another EBM only on $\mathbf{x}$, but we initialize the MCMC chains with samples from the pre-trained NVAE instead of noise. As shown in line 3 of Table 4.6, this initialization helps the EBM which is defined only on $\mathbf{x}$. However, VAEBM in the augmented space outperforms the EBMs on $\mathbf{x}$ only by a large margin.

Adversarial training vs. sampling: When training EBMs, gradient for the energy

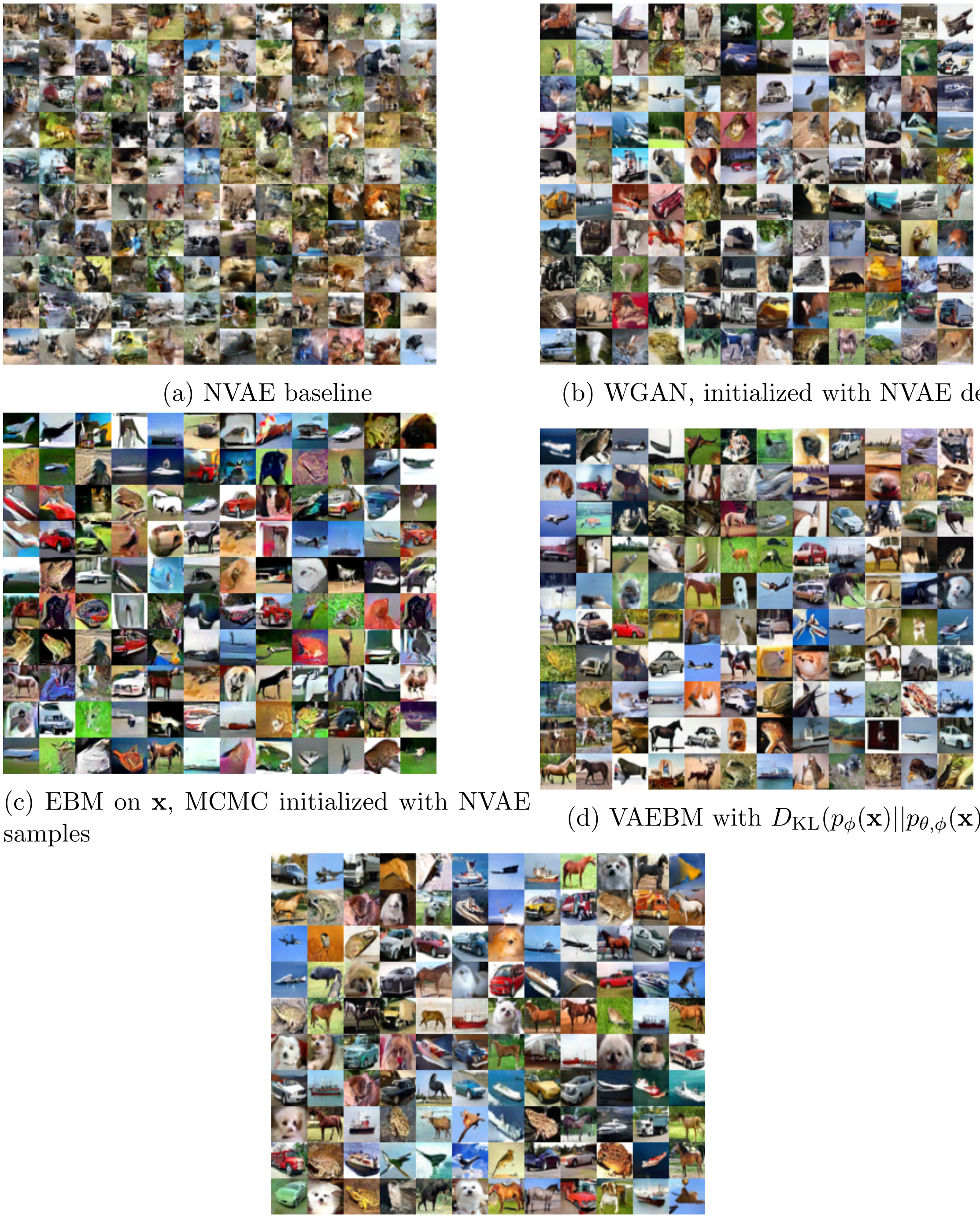
Table 4.6: Generative performance of VAEBM on CelebA HQ 256

Model	IS↑	FID↓
NVAE (Vahdat and Kautz)	5.19	55.97
EBM on $\mathbf{x}$ (Du and Mordatch)	5.85	48.89
EBM on $\mathbf{x}$, MCMC init w/ NVAE	7.28	29.32
WGAN w/ NVAE decoder	7.41	20.39
VAEBM + $D_{\mathrm{KL}}(p_\phi(\mathbf{x})\|\|p_{\theta,\phi}(\mathbf{x}))$	8.05	14.00
VAEBM (ours)	**8.15**	**12.96**

function is similar to the gradient updates of WGAN's discriminator [Arjovsky et al., 2017]. The key difference is that we draw (approximate) samples from the model by MCMC, while WGAN draws negative samples from a generator [Che et al., 2020]. WGAN updates the generator by playing an adversarial game, while we only update the energy function f_θ. We compare these two methods by training the energy function f_θ and a generator with the WGAN objective and initializing the generator with the NVAE decoder. As shown in line 4 of Table 4.6, we significantly outperform the WGAN version of our model, implying the advantage of our method over adversarial training.

Updating VAE generator while training EBM: As discussed in Sectionapter 4.3.4, we can jointly train θ and ϕ, where ϕ is updated with additional loss terms that minimize $D_{\mathrm{KL}}(p_\phi(\mathbf{x})||p_{\theta,\phi}(\mathbf{x}))$. We train VAEBMs with these additional loss and present the results in line 5 in Table 4.6. We observe that updating ϕ with additional losses does not improve the generative performances, and updating the decoder is unnecessary. This is likely because the initial VAE is pulled as closely as possible to the data distribution already, which is also the target for the joint VAEBM $p_{\theta,\phi}(\mathbf{x})$. Therefore, we adopt the simplest training method where we only minimize $D_{\mathrm{KL}}(p_{\mathrm{data}}(\mathbf{x})||p_{\theta,\phi}(\mathbf{x}))$.

In Figure 4.17, we show qualitative samples from models corresponding to each item in Table 4.6.



(a) NVAE baseline

(b) WGAN, initialized with NVAE decoder

(c) EBM on $\mathbf{x}$, MCMC initialized with NVAE samples

(d) VAEBM with $D_{\mathrm{KL}}(p_\phi(\mathbf{x})||p_{\theta,\phi}(\mathbf{x}))$ loss

(e) VAEBM

Figure 4.17: Qualitative results of ablation study

Table 4.7: Generative performance of VAEBM on CelebA HQ 256

Model	Modes↑	KL↓
VEEGAN ([Srivastava et al., 2017])	761.8	2.173
PacGAN ([Lin et al., 2018])	992.0	0.277
PresGAN ([Dieng et al., 2019])	999.6	0.115
InclusiveGAN ([Yu et al., 2020b])	997	0.200
StyleGAN2 ([Karras et al., 2020b])	940	0.424
VAEBM (ours)	**1000**	**0.087**

Test for Spurious or Missing Modes

We evaluate mode coverage on StackedMNIST. This dataset contains images generated by randomly choosing 3 MNIST images and stacking them along the RGB channels. Hence, the data distribution has 1000 modes. After training a generative model on this dataset, we can evaluate the mode coverage by classifying generated samples with a classifier on MNIST.

Following Lin et al. [2018], we report the number of covered modes and the KL divergence from the categorical distribution over 1000 categories from generated samples to true data (Table 4.7). VAEBM covers all modes and achieves the lowest KL divergence even compared to GANs that are specifically designed for this task. Hence, our model covers the modes more equally.

We also plot the histogram of likelihoods for CIFAR-10 train/test images in Figure 4.18. We see that our model assigns similar likelihoods to both train and test set images. This indicates that VAEBM generalizes well to unseen data and covers modes in the training data well.

We evaluate spurious modes in our model by assessing its performance on out-of-distribution (OOD) detection. Nalisnick et al. [2018], Xiao et al. [2020b] observe that some likelihood-based generative models, including VAEs and normalizing flows, assign a higher likelihood to OOD samples. One possible explanation is that likelihood-based models suffer from the mismatch of density discussed in Sectionapter 4.1. Therefore, it is promising to improve the OOD detection by exponential tilting. We use VAEBM trained on CIFAR-10, and es-

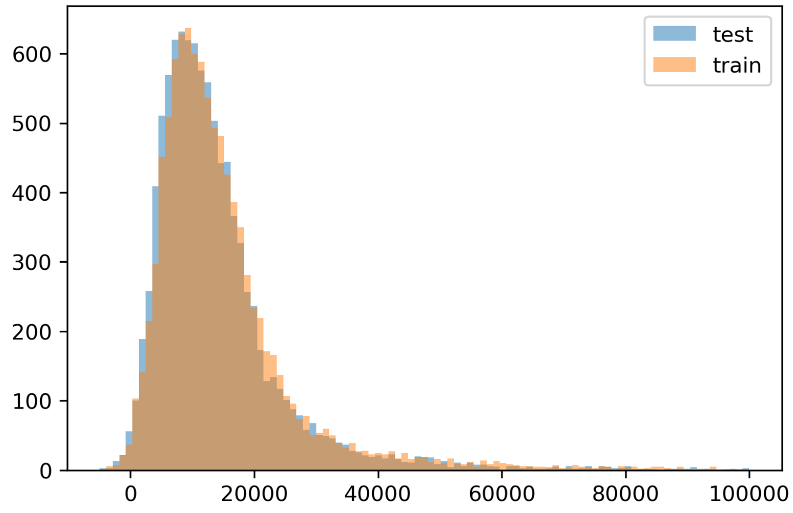


Figure 4.18: Histogram of unnormalized log-likelihoods on 10k CIFAR-10 train and test set images.

timate unnormalized $\log p_{\theta,\phi}(\mathbf{x})$ on in-distribution samples (from CIFAR-10 test set) and OOD samples from several datasets. Following Nalisnick et al. [2018], we use the area under the ROC curve (AUROC) as a quantitative metric, where high AUROC indicates that the model correctly assigns low density to OOD samples. In Table 4.8, we see that VAEBM has significantly higher AUROC than NVAE, justifying our argument that the energy function reduces the likelihood of non-data-like regions. VAEBM also performs better than IGEBM and JEM, while worse than HDGE. However, we note that JEM and HDGE are classifier-based models, known to be better for OOD detection [Liang et al., 2018]. The good performance on OOD detection suggests that our VAEBM successfully refine the density of VAE by excluding non-data-like regions.

Comparison of Sampling in $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$-space and in $(\mathbf{x}, \mathbf{z})$-space

In Section 4.3.3, we highlight the advantage of sampling in the reparametrization space $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$ over sampling in $(\mathbf{x}, \mathbf{z})$-space, as it automatically adjust the per-element step size. To further provide empirical evidence that adjusting the step size for each variable is necessary,

Table 4.8: Table for AUROC↑ of $\log p(\mathbf{x})$ computed on several OOD datasets. In-distribution dataset is CIFAR-10. Interp. corresponds to linear interpolation between CIFAR-10 images.

		SVHN	Interp.	CIFAR100	CelebA
Unsupervised Training	NVAE [Vahdat and Kautz, 2020]	0.42	0.64	0.56	0.68
	Glow [Kingma and Dhariwal, 2018]	0.05	0.51	0.55	0.57
	IGEBM [Du and Mordatch, 2019]	0.63	**0.7**	0.5	0.7
	Divergence Traingle [Han et al., 2020]	0.68	-	-	0.56
	VAEBM (ours)	**0.83**	**0.7**	**0.62**	**0.77**
Supervised Training	JEM [Grathwohl et al., 2020]	0.67	0.65	0.67	0.75
	HDGE [Liu and Abbeel, 2020]	0.96	0.82	0.91	0.8

we try sampling directly in $(\mathbf{x}, \mathbf{z})$-space without adjusting the step size (i.e., use a universal step size for all variables). Qualitative results are presented in Figure 4.19. We examine several choices for the step size and we cannot obtain high-quality samples.

In conclusion, the re-parameterization provides an easy implementation to adjust step size for each variable, and the adjustment is shown to be crucial to obtain good samples.

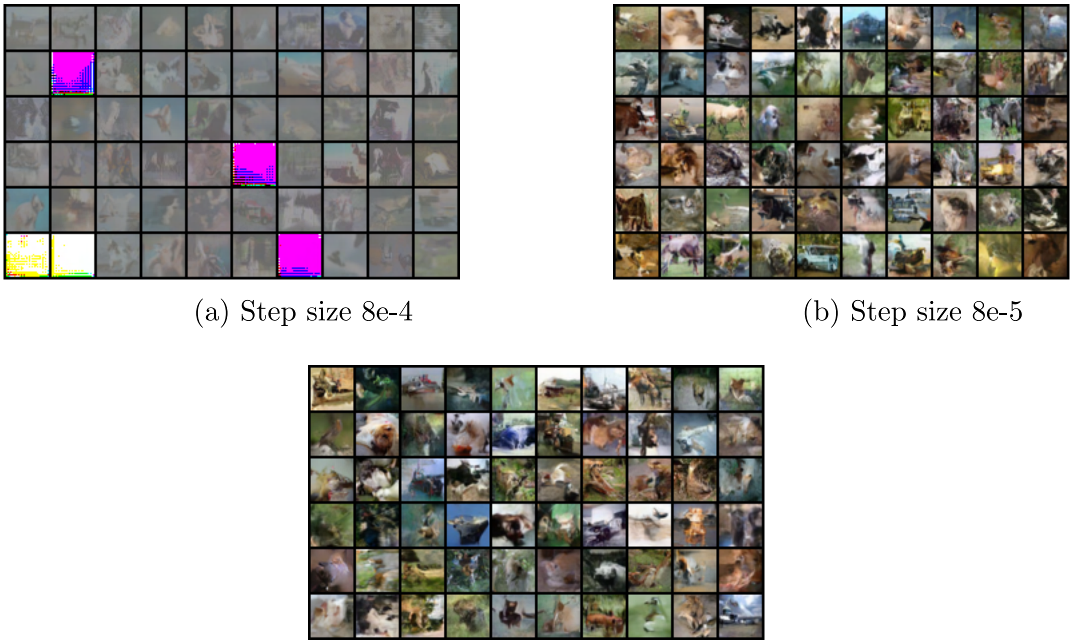
Implementation Details

In this section, we introduce the details of training and sampling from VAEBM. Codes for the VAEBM implementation can be found at `https://github.com/NVlabs/VAEBM`.

NVAE: VAEBM uses NVAE as the $p_\phi(\mathbf{x})$ component in the model. We train the NVAE with its official implementation[1]. We largely follow the default settings, with one major difference that we use a Gaussian decoder instead of a discrete logistic mixture decoder as in Vahdat and Kautz [2020]. The reason for this is that we can run Langevin dynamics only with continuous variables. The number of latent variable groups for CIFAR-10, CelebA 64, LSUN Church 64 and CelebA HQ 256 are 30, 15, 15 and 20, respectively.

Network for energy function: We largely adopt the energy network structure for CIFAR-10 in Du and Mordatch [2019], and we increase the depth of the network for larger images. There are 2 major differences between our energy networks and the ones used in

[1] `https://github.com/NVlabs/NVAE`



(a) Step size 8e-4

(b) Step size 8e-5

(c) Step size 8e-6

Figure 4.19: Qualitative samples obtained from sampling in $(\mathbf{x}, \mathbf{z})$-space with different step sizes.

Du and Mordatch [2019]: **1.** we replace the LeakyReLU activations with Swish activations, as we found it improves training stability, and **2.** we do not use spectral normalization [Miyato et al., 2018]; instead, we use weight normalization with data-dependent initialization [Salimans and Kingma, 2016]. The network structure for each dataset is presented in Table 4.9.

Training of energy function: We train the energy function by minimizing the negative log likelihood and an additional spectral regularization loss which penalizes the spectral norm of each convolutional layer in f_θ. The spectral regularization loss is also used in training NVAE, as we found it helpful to regularize the sharpness of the energy network and better stabilize training. We use a coefficient 0.2 for the spectral regularization loss.

We summarize some key hyper-parameters we used to train VAEBM in Table 4.10. On all datasets, we train VAEBM using the Adam optimizer [Kingma and Ba, 2015] and weight decay 3e−5. We use constant learning rates, shown in Table 4.10. Following Du and Mor-

Table 4.9: Network structures for the energy function $f_\theta(\mathbf{x})$

CIFAR-10
3×3 conv2d, 128
ResBlock down 128
ResBlock 128
ResBlock down 256
ResBlock 256
ResBlock down 256
ResBlock 256
Global Sum Pooling
FC layer → scalar

CelebA 64
3×3 conv2d, 64
ResBlock down 64
ResBlock 64
ResBlock down 128
ResBlock 128
ResBlock down 128
ResBlock 256
ResBlock down 256
ResBlock 256
Global Sum Pooling
FC layer → scalar

LSUN Church 64
3×3 conv2d, 64
ResBlock down 64
ResBlock 64
ResBlock down 128
ResBlock 128
ResBlock 128
ResBlock down 128
ResBlock 256
ResBlock 256
ResBlock down 256
ResBlock 256
Global Sum Pooling
FC layer → scalar

CelebA HQ 256
3×3 conv2d, 64
ResBlock down 64
ResBlock 64
ResBlock down 128
ResBlock 128
ResBlock down 128
ResBlock 128
ResBlock down 256
ResBlock 256
ResBlock down 256
ResBlock 256
ResBlock down 512
ResBlock 512
Global Sum Pooling
FC layer → scalar

datch [2019], we clip training gradients that are more than 3 standard deviations from the 2nd-order Adam parameters. Note that with such a small number of Langevin sampling steps, the discrete Langevin sampling can be better viewed as an implicit generator model rather than an approximation to the Langevin dynamics. We will discuss this issue in detail in Chapter 4.

While persistent sampling using a sample replay buffer has little effect on CIFAR-10, we found it to be useful on large images such as CelebA HQ 256. When we do not use persistent

Table 4.10: Important hyper-parameters for training VAEBM. LR stands for learning rate, BS stands for batch size.

Dataset	LR	BS	Persistent	# steps	Step Size
CIFAR-10 w/o persistent chain	4e−5	32	No	10	8e−5
CIFAR-10 w/ persistent chain	4e−5	32	Yes	6	6e−5
CelebA 64	5e−5	32	No	10	5e−6
LSUN Church 64	4e−5	32	Yes	10	4e−6
CelebA HQ 256	4e−5	16	Yes	6	3e−6

sampling, we always initialize the LD chains with $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$, sampled from a standard Gaussian. When we use persistent sampling in training, we keep a sample replay buffer that only stores samples of $\epsilon_{\mathbf{z}}$, while $\epsilon_{\mathbf{x}}$ is always initialized from a standard Gaussian. The size of the replay buffer is 10,000 for CIFAR-10 and LSUN Church 64, and 8,000 for CelebA HQ 256. At every training iteration, we initialize the MCMC chains on $\epsilon_{\mathbf{z}}$ by drawing $\epsilon_{\mathbf{z}}$ from the replay buffer with probability p and from standard Gaussian with probability $1-p$. For CIFAR-10 and LSUN Church 64, we linearly increase p from 0 to 0.6 in 5,000 training iterations, and for CelebA HQ 256, we linearly increase p from 0 to 0.6 in 3,000 training iterations. The settings of Langevin dynamics are presented in Table 4.10.

We do not explicitly set the number of training iterations. Instead, we follow Du and Mordatch [2019] to train the energy network until we cannot generate realistic samples anymore. This happens when the model overfits the training data and hence energies of negative samples are much larger than energies of training data. Typically, training takes around 25,000 iterations (or 16 epochs) on CIFAR-10, 20,000 iterations (or 3 epochs) on CelebA 64, 20,000 iterations (or 5 epochs) on LSUN Church 64, and 9,000 iterations (or 5 epochs) on CelebA HQ 256.

Test time sampling: After training the model, we generate samples for evaluation by running Langevin dynamics with $(\epsilon_{\mathbf{x}}, \epsilon_{\mathbf{z}})$ initialized from standard Gaussian, regardless of whether persistent sampling is used in training or not. We run slightly longer LD chains than training to obtain the best sample quality. In particular, our reported values are obtained from running 16 steps of LD for CIFAR-10, 20 steps of LD for CelebA64 and LSUN Church 64, and 24 steps for CelebA HQ 256. The step sizes are the same as training step sizes.

Settings for Ablation Study

Here we present the details of ablation experiments. Throughout ablation experiments, we use a smaller NVAE with 20 groups of latent variables trained on CIFAR-10. We use the same network architectures for the energy network as in Table 4.9, with potentially different normalization techniques discussed below. We spent significant efforts on improving each method we compare against, and we report the settings that led to the best results.

WGAN initialized with NVAE decoder: We initialize the generator with the pre-trained NVAE decoder, and the discriminator is initialized by a CIFAR-10 energy network with random weights. We use spectral normalization and batch normalization in the discriminator as we found them necessary for convergence. We update the discriminator using the Adam optimizer with constant learning rate 5e$-$5, and update the generator using the Adam optimizer with initial learning rate 5e$-$6 and cosine decay schedule. We train the generator and discriminator for 40k iterations, and we reach convergence of sample quality towards the end of training.

EBM on x, w/ or w/o initializing MCMC with NVAE samples: We train two EBMs on data space similar to Du and Mordatch [2019], where for one of them, we use the pre-trained NVAE to initialize the MCMC chains that draw samples during training. The setting for training these two EBMs are the same except for the initialization of MCMC. We use spectral normalization in the energy network and energy regularization in the training objective as done in Du and Mordatch [2019] because we found these modifications to improve performance. We train the energy function using the Adam optimizer with constant learning rate 1e$-$4. We train for 100k iterations, and we reach convergence of sample quality towards the end of training. During training, we draw samples from the model following the MCMC settings in Du and Mordatch [2019]. In particular, we use persistent sampling and sample from the sample replay buffer with probability 0.95. We run 60 steps of Langevin dynamics to generate negative samples and we clip gradients to have individual value magnitudes of less than 0.01. We use a step size of 10 for each step of Langevin dynamics. For test time sampling, we generate samples by running 150 steps of LD with the same settings as during training.

VAEBM with $D_{\mathrm{KL}}(p_\phi(\mathbf{x})||p_{\theta,\phi}(\mathbf{x}))$ loss: We use the same network structure for E_ψ as in VAEBM. We find persistent sampling significantly hurts the performance in this case, possibly due to the fact that the decoder is updated and hence the initial samples from the decoder change throughout training. Therefore, we do not use persistent training. We train the energy function using the Adam optimizer with constant learning rate 5e−5. We draw negative samples by running 10 steps of LD with step size 8e−5. We update the decoder with the gradient in Equation 4.34 using the Adam optimizer with initial learning rate 5e−6 and cosine decay schedule. For test time sampling, we run 15 steps of LD with step size 5e−6.

4.5 Conclusion

This chapter introduces the framework of exponential tilting, which trains an energy-based refinement over base generative models. We show that with little computational overhead, we can improve the sample quality of a variety of generative models, including normalizing flows and VAEs, by sampling from exponential tilted models. We show that our model can be trained effectively in two stages with a maximum likelihood objective, and we can efficiently sample it by running short Langevin dynamics chains. Experimental results demonstrate strong generative performance on several image datasets.

In this joint model, the EBM and the base model form a symbiotic relationship:

- The base model learns the overall mode structure, hence saves a lot of time for training the EBM
- The base model provides re-parametrization for MCMC sampling from EBM, so that the MCMC is performed on a distribution with smooth density, which significantly facilitate both training and test-time sampling from the EBM.
- The EBM helps the base model to exclude non-data-like regions and significantly improves the sample quality.

CHAPTER 5
SHORT-RUN LANGEVIN DYNAMICS AS GENERATOR MODELS

In this chapter, we investigate the role of Langevin dynamics in the maximum likelihood training of Energy-based models. We try to understand this training procedure by replacing Langevin dynamics with deterministic solutions of the associated gradient descent ODE. Doing so allows us to study the density induced by the dynamics (if the dynamics are invertible), and connect with GANs by treating the dynamics as generator models, 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 begin with motivating our approach and introducing relative backgrounds. Then we will introduce our modifications to the maximum likelihood training of EBMs and present experimental results.

The material of this chapter is based on Xiao et al. [2021b].

5.1 Motivation and Introduction

As introduced in Section 2.3, Energy-based models (EBMs) are likelihood-based generative models that model the unnormalized data density by assigning low energy to high-probability regions in the data space. Recently, by using a neural network as the energy functions, deep EBMs [Xie et al., 2016, Du and Mordatch, 2019] are able to model complex data. 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., 2020a], score matching [Li et al., 2019b] and contrastive estimation [Gao et al., 2020, Gutmann and Hyvärinen, 2012]. Among them, the KL divergence minimization (equivalent to maximum likelihood estimation) is the most widely used.

The maximum likelihood training of EBMs is introduced in Section 2.3.2. For convenience, we briefly re-state the core idea here. To train an EBM of the form $p_\theta(\mathbf{x}) = \exp\left(-E_\theta(\mathbf{x})\right)/Z_\theta$, where $E_\theta(\mathbf{x})$ is the energy function with parameters θ and Z_θ is the normalizing constant, we can take the derivative of the negative log likelihood function $L(\theta) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[-\log p_\theta(\mathbf{x})\right]$ w.r.t to the model parameter θ [Woodford, 2006]:

$$\partial_\theta L(\theta) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\partial_\theta E_\theta\left(\mathbf{x}\right)\right] - \mathbb{E}_{\mathbf{x}\sim p_\theta(\mathbf{x})}\left[\partial_\theta E_\theta\left(\mathbf{x}\right)\right] \tag{5.1}$$

and minimize $L(\theta)$ by gradient descent. The second expectation in Equation 5.1 can be empirically estimated by samples drawn from the model $p_\theta(\mathbf{x})$ itself. However, sampling from $p_\theta(\mathbf{x})$ is intractable and samples are usually drawn using MCMC. A commonly used MCMC algorithm is the Langevin dynamics (LD) [Neal, 1993]. Given an initial sample $\mathbf{x}_0$, Langevin dynamics solves the SDE

$$d\mathbf{x}_t = -\frac{1}{2}\nabla_{\mathbf{x}} E_\theta(\mathbf{x}_t)dt + d\mathbf{w}_t, \tag{5.2}$$

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

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{2}\nabla_{\mathbf{x}} E_\theta(\mathbf{x}_t) + \sqrt{\eta}\omega_t, \tag{5.3}$$

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, in practice, we usually run LD for finite number of steps with a fixed step size. After training, samples are obtained by running the same Langevin dynamics, typically with the same number of steps.

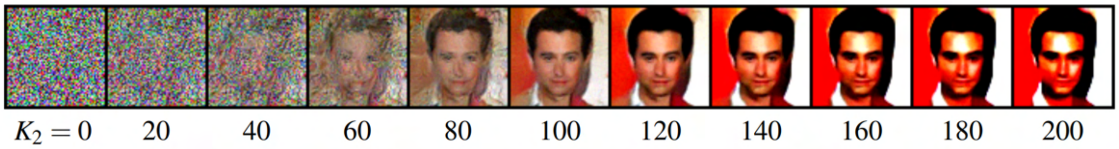


Figure 5.1: Transition with $K_1 = 100$ LD steps for training and varying K_2 LD steps for sampling. Figure taken from Nijkamp et al. [2019].

5.1.1 Alternative understanding of maximum likelihood training

Although the maximum likelihood training scheme is simple and intuitively appealing, we might still 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 Equation 5.3, 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 Xiao et al. [2020a], Nijkamp et al. [2022], LD usually does not mix, i.e., samples obtained by running longer LD get trapped in different local modes instead of traversing between modes. An example taken from Nijkamp et al. [2019] is shown in Figure 5.1, where we observe that increasing the number of LD sampling steps results in over-saturated samples.

Probably as a consequence, the initial points $\mathbf{x}_0$ contain information about the final outcome. Therefore short-run LD is observed to be capable of reconstructing the data and interpolating different samples. For example, Nijkamp et al. [2019] observe that by fixing two noise vectors $\mathbf{z}_1, \mathbf{x}_2 \sim \mathcal{N}(0, I)$, and initialize the LD with interpolations between $\mathbf{z}_1, \mathbf{x}_2$: $\mathbf{z}_\rho = \rho\mathbf{z}_1 + \sqrt{1-\rho^2\mathbf{z}_2}, \rho \in [0,1]$, the resulting samples consist of a meaningful interpolation between the samples generated by initializing LD with $\mathbf{z}_1$ and $\mathbf{z}_2$ (see Figure 5.2). Also, in Nijkamp et al. [2019], the authors observe that any given image can be reconstructed by

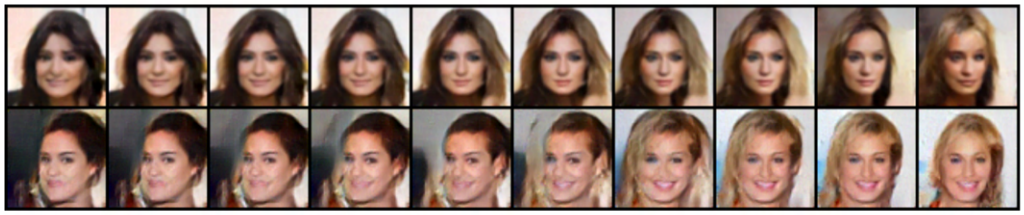


Figure 5.2: Transition of sequence of samples obtained from initializing the LD with interpolated noise $\mathbf{z}_\rho$. The leftmost and rightmost images are samples from initializing with $\mathbf{z}_1$ and $\mathbf{z}_2$, respectively. Figure taken from Nijkamp et al. [2019].

optimizing the initial value $\mathbf{z}_0$ by back-propagating into the LD iterations.

Another interesting observation is that, 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. Therefore 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 or flow model. They consider $p_\theta(\mathbf{x})$ to be a generative model of the following form:

$$\mathbf{z} \sim p_0(\mathbf{z}); \quad \mathbf{x} = M_\theta(\mathbf{z}, u), \tag{5.4}$$

where u denotes all the randomness in the short-run MCMC. For the K-step Langevin dynamics, M_θ can be considered a K-layer noise-injected residual network, $\mathbf{z}$ can be considered latent variables, and p_0 the prior distribution of $\mathbf{z}$. Due to the non-convergence and non-mixing of LD, $\mathbf{x}$ can be highly dependent on $\mathbf{z}$, and $\mathbf{z}$ can be inferred from $\mathbf{x}$. This is completely different from the convergent MCMC, where $\mathbf{x}$ is independent of $\mathbf{z}$. However, they do not study p_θ with an explicit formulation. In this chapter, 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 so that the output samples are produced by a deterministic transformation of the initial points. In this case, we regard the dynamic as a generator model and the initial points as latent variables. By ensuring that the generator is invertible, we can explicitly study the density of the distribution induced by the sampling dynamics (where the initial points entirely determine the randomness). In addition, by treating the sampling dynamics as a generator model, we find that we can improve the sample quality by adding the generator loss term from GANs to the original loss.

5.2 Related Work

The material of this chapter is closely related to earlier studies on the properties of ML training EBMs with short-run non-convergent MCMC [Nijkamp et al., 2019, 2020], where they illustrate through experiments that the short-run LD behaves more like a generator model, and in particular Nijkamp et al. [2019] provide a moment matching framework for explaining the mechanism behind the maximum likelihood training. In addition, Xie et al. [2018, 2020] propose MCMC teaching, where a separate generator is trained to absorb the process of LD sampling. This suggests that their method is based on the assumption that LD used in practice can be represented as a generator model. Additionally, Han et al. [2019] provides a probabilistic way to deal with EBM without MCMC. We take a further step from them to explicitly study the properties of the generator models.

Since our noise-free sampling dynamics can yield an invertible gradient flow , our work is related to the concept of generative gradient flows [Zhang et al., 2018, Huang et al., 2021a]. In addition, Song et al. [2021b] show that the stochastic dynamics of score based generative models [Song and Ermon, 2019, Ho et al., 2020] are equivalent to specific deterministic ODE flows [Chen et al., 2018a, Grathwohl et al., 2018]. However, such equivalence cannot be easily established for Langevin diffusion. Pang et al. [2020] connects EBM and generator model, but what they do is learning an EBM prior for the generator.

Finally, our work is related to previous work that connects GANs with EBMs [Che et al., 2020, Song et al., 2020b, Ansari et al., 2021] or invertible flows [Grover et al., 2018]. In particular, Grover et al. [2018] use invertible structures, such as real-NVP [Dinh et al., 2016], for the generator of GANs, but they focus on hybrid training with adversarial and maximum likelihood objectives.

5.3 Noise-free Sampling Dynamics as Flow Models

In this section, we demonstrate how to explicitly obtain the density induced by the noise-free sampling dynamics by enforcing invertibility. We start by replacing the Langevin dynamics in Equation 5.1 with the noise-free gradient descent ODE:

$$\mathbf{x}'(t) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0, \quad t \in [0, T], \tag{5.5}$$

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

$$\mathbf{x}_T = G_\theta^T(\mathbf{x}_0) = \text{ODESolve}(-\nabla_{\mathbf{x}} E_\theta(\mathbf{x}(t)), \mathbf{x}_0, [0, T]), \tag{5.6}$$

where $\text{ODESolve}(-\nabla_{\mathbf{x}} E_\theta(\mathbf{x}(t)), \mathbf{x}_0, [0, T])$ is a black-box numerical ODE solver that solves tne ODE with function $-\nabla_{\mathbf{x}} E_\theta(\mathbf{x}(t))$ and initial value $\mathbf{x}_0$, from time 0 to T.

Since there is no noise term, given $\mathbf{x}_0$, the process can be represented by a deterministic generator model with latent variable $\mathbf{x}_0$. We denote the model as $G_\theta^T(\mathbf{x}_0)$. We want to emphasize that T is an important component of the generator model, and we should use roughly the same T when sampling. Moreover, as $G_\theta^T(\mathbf{x}_0)$ is invertible, the likelihood along the path can be obtained by instantaneous change of variables formula [Chen et al., 2018a], and the log likelihood of data $\mathbf{x}$ under the flow model can be computed by

$$\log p(\mathbf{x}) = \log p(\mathbf{x}_0) + \int_0^T \mathrm{tr}\left[\nabla_{\mathbf{x}\mathbf{x}} E_\theta(\mathbf{x}(t))\right] dt. \tag{5.7}$$

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

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{2}\nabla_{\mathbf{x}} E_\theta(\mathbf{x}_t), \quad t = 0, 1, \cdots, K-1, \tag{5.8}$$

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

$$\mathbf{x}_K = G_\theta^T(\mathbf{x}_0) = (I - \frac{\eta}{2}\nabla_{\mathbf{x}} E_\theta)^K(\mathbf{x}_0). \tag{5.9}$$

$G_\theta^T(\mathbf{x}_0)$ is guaranteed to be invertible if $\mathrm{Lip}\left(\frac{\eta}{2}\nabla_{\mathbf{x}} E_\theta\right) < 1$ [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 in Equation 5.5.

To summarize, we train the energy network E_θ by doing the gradient update in Equation 5.1 with negative samples obtained from Equation 5.6. After training, we can obtain new samples by running Equation 5.6, and compute the likelihood of data point $\mathbf{x}$ by solving the ODE in the *reverse* direction to find the corresponding initial point $\mathbf{x}_0$ and then apply 5.7.

The method discussed in this section is not computationally efficient. Note that for a general neural ODE, the output $\mathbf{y}$ can be written as

$$\mathbf{y} = \mathrm{ODESolve}(f_\theta(\mathbf{x}(t)), \mathbf{x}, [0, T]), \tag{5.10}$$

where f is a neural network. The goal is to optimize f_θ, and only the first-order derivative is needed to optimize θ. However, in our case, $f_\theta = -\nabla_{\mathbf{x}} E_\theta$, and hence we need to take higher order derivative to optimize θ. This can be prohibitively slow in high dimensions. As a result, we only strictly stick to the gradient flow formulation on 2-D toy data. On image data, we simply remove the noise term in discretized Langevin dynamics in Equation 5.3.

5.4 Connection with W-GAN and the generator loss term

It is well known that the maximum likelihood training of EBMs is closely related to the training of Wasserstein-GANs [Che et al., 2020], where the objective for the discriminator D (assuming D is 1-Lipschitz) is

$$\max_D \quad \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[D(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim p_G}\left[D(\mathbf{x})\right], \tag{5.11}$$

where p_G is the (implicit) distribution defined by the generator. The gradient of Equation 5.11 is (up to a sign) very similar to Equation 5.1 except that here the negative samples are drawn from the generator, while in Equation 5.1, the negative samples are drawn from the EBM itself. Note that the sign does not matter as we can model the negative energy instead. Intuitively, W-GANs use the discriminator D 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 MCMC. Therefore, the maximum likelihood training of EBMs can be described as a *self-adversarial game.*

In W-GANs, after the discriminator is updated by optimizing 5.11, the generator G is then updated by

$$\max_G \mathbb{E}_{\tilde{\mathbf{x}}\sim p_G}\left[D(\tilde{\mathbf{x}})\right]. \tag{5.12}$$

In other words, the generator is trained by maximizing the discriminator's output of 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 5.3, 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 Equation 5.6, and therefore we can update the parameter of G_θ by the following objective:

$$\min_\theta E_{\mathrm{sg}(\theta)}(G_\theta(\mathbf{x}_0)), \tag{5.13}$$

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. Note that the implicit generator is defined by an iterative process, and it is not trivial to take the gradient of such a process. To do so, we unroll the iterative process by storing the derivative of each step and propagating back from the last step to the first step.

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 discriminator 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 discriminator and the generator, and hence two batches of samples are generated. 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 without alternating optimization as in GANs:

$$\min_\theta E_\theta(\mathbf{x}) - E_\theta(G_{\mathrm{sg}(\theta)}(\mathbf{x}_0)) + E_{\mathrm{sg}(\theta)}(G_\theta(\mathbf{x}_0)), \quad \mathbf{x} \sim p_D, \mathbf{x}_0 \sim p_0. \tag{5.14}$$

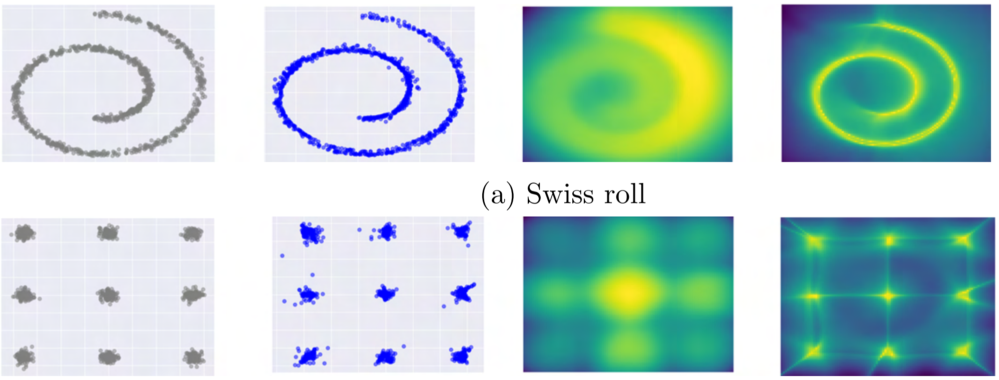
5.5 Experimental Results

In this section, we conduct experiments to verify our proposed methods and arguments in section 5.3 and 5.4. Specifically, we train energy functions on 2d-toy data and image data by replacing the MCMC sampling with deterministic dynamics. Throughout the experiments, we initialize the dynamics with noise sampled from standard Gaussian distribution. We do not use persistent sampling, as we want to interpret the model as a generator with fixed prior. The deterministic dynamics can be simply defined by Equation 5.8, or more generally, the path to solve the ODE as in Equation 5.6. In particular, we need to use the latter method if we want to compute the density induced by the dynamics.

5.5.1 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 Equation 5.5 with the default Dormand–Prince solver as in Chen et al. [2018a]. In Figure 5.3, we plot the samples obtained from solving the ODE using Equation 5.6. As a comparison, we also plot the log density of the ODE flow and the value of the negative energy function (which is the unnormalized log density of the corresponding EBM) in the same figure. We 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 functions induced by the ODE flow capture the densities the true data distributions very well. We also train EBMs with valid MCMC sampling with noise term and plot the density functions and generated samples in Figure 5.4. There we make a similar observation that the densities of EBMs do not match the data distribution.

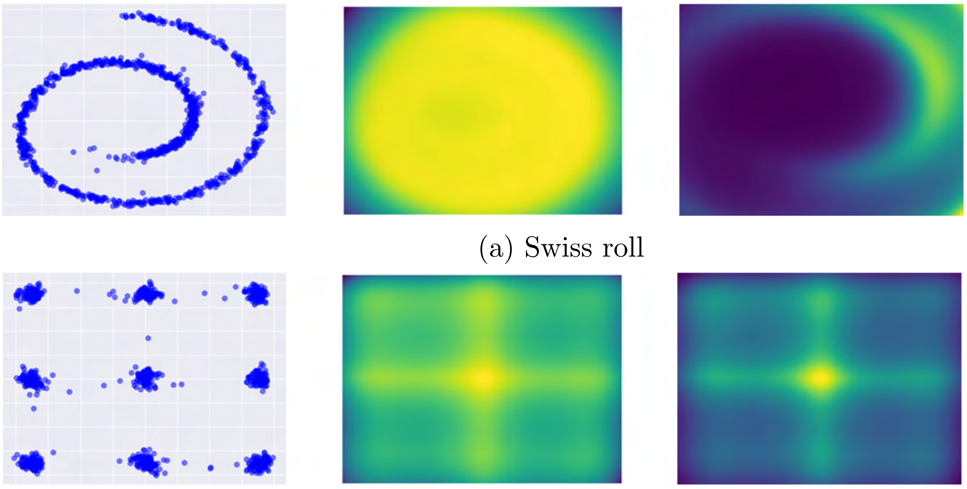
We also plot the normalized density of the EBMs and gradient flows in Figure 5.5, where we observe that the spurious high-density region shown in the log density plot in Figure 5.3 disappears, and we still find that the density of the gradient flows captures the true density



(a) Swiss roll

(b) Gaussian grid

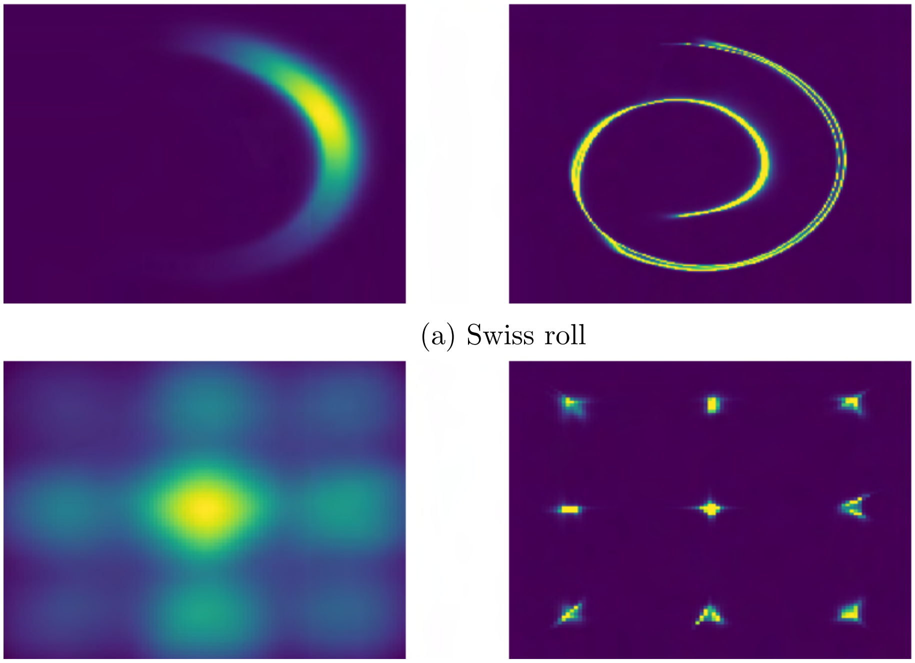
Figure 5.3: For each toy dataset, **column 1:** samples from the true data distribution; **column 2:** samples from the ODE flow; **column 3:** (unnormalized) log density of the EBM by plotting the value of $-E_\theta(\mathbf{x})$; **column 4:** log density of the ODE flow computed by Equation 5.7. The spurious connections between components will visually disappear if we take exponential (see Figure 5.5). We plot log density because the sampling dynamics directly use it.



(a) Swiss roll

(b) Gaussian grid

Figure 5.4: 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.



(a) Swiss roll

(b) Gaussian grid

Figure 5.5: For each sub-figure, **left:** normalized density of the EBM, and **right:** density of the gradient flow.

much better than that of the EBMs.

These observations prove 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 dynamic *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 Equation 5.7 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. The test data likelihood of the ODE flows trained by the EBM objective is **-0.86** nats and **-1.95** nats on these two datasets, respectively. As expected, the flows directly trained by maximizing

Table 5.1: 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

data likelihood have higher test likelihood, but the flows trained by the EBM objective still perform reasonably well.

5.5.2 Image Data

Experiments on toy data reveal that the maximum likelihood training of EBMs may actually lead to training generator 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 common image datasets, including MNIST, Fashion-MNIST, CIFAR-10, and CelebA. For simplicity, our energy functions are simple convolutional nets instead of more complex residual networks used in Du and Mordatch [2019], Xiao et al. [2021a], and therefore we only compare relative performances.

We train energy functions using the gradient update in Equation 5.1, where the samples are generated by either noisy or noise-free sampling dynamics. For noise-free dynamics, we use the gradient descent formulation in Equation 5.8 instead of the neural ODE formulation because we only want to study the effect of noise while keeping all other factors the same. For noisy sampling dynamics, we reduce the noise scale as done in almost all other work, otherwise, the training diverges quickly. We report the FID scores in Table 5.1. In Figure 5.6, 5.7 and 5.8, we present qualitative samples of EBMs with noisy sampling dynamics, EBMs with noise-free dynamics, and EBMs with noise-free dynamics + generator update, respectively. We observe that EBMs trained with noise-free dynamics indeed obtain better sample quality on all datasets.

Besides, we plot the loss curve along with the training of models with noisy or noise-

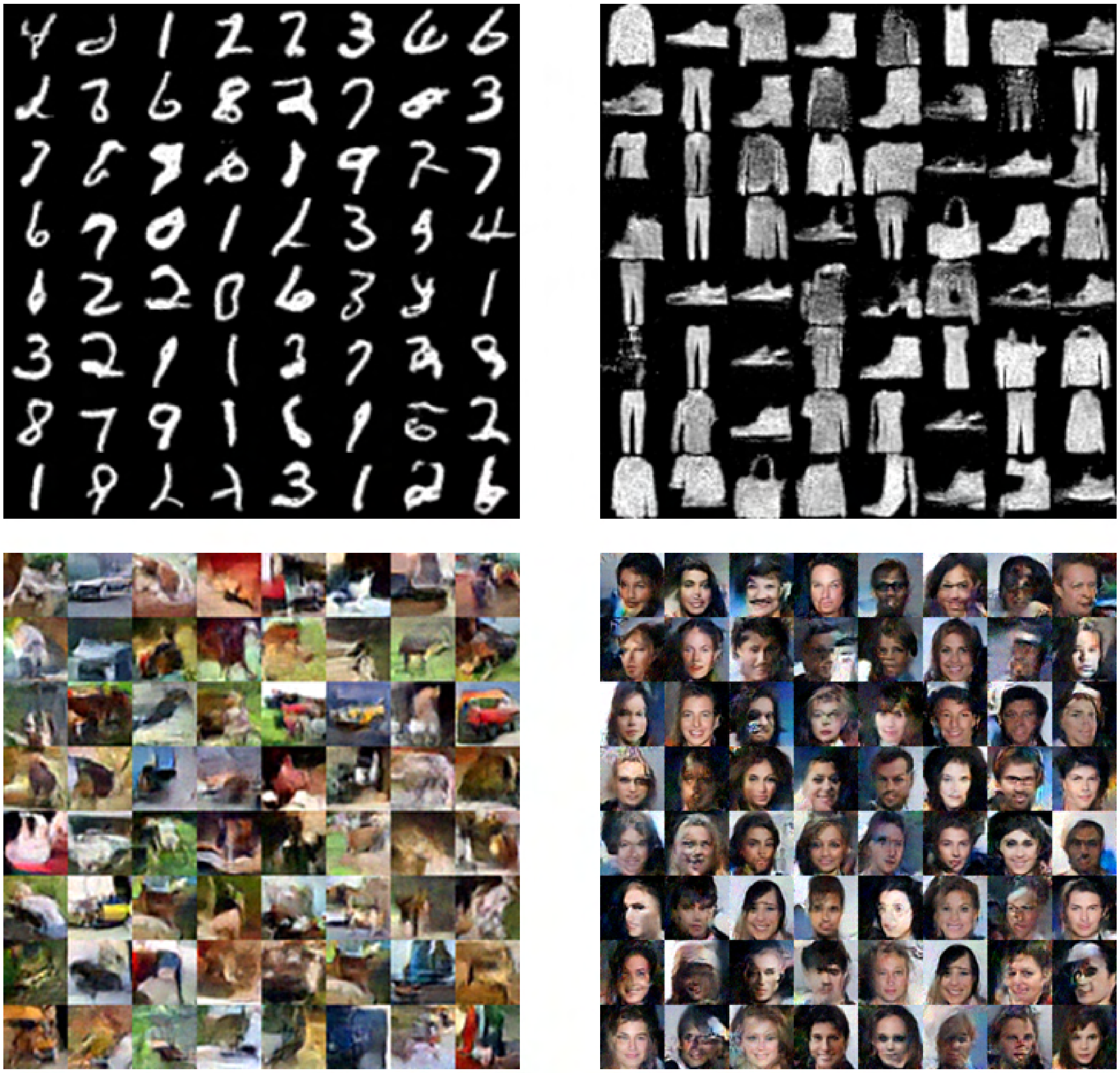


Figure 5.6: Samples from EBMs w/ noisy dynamics

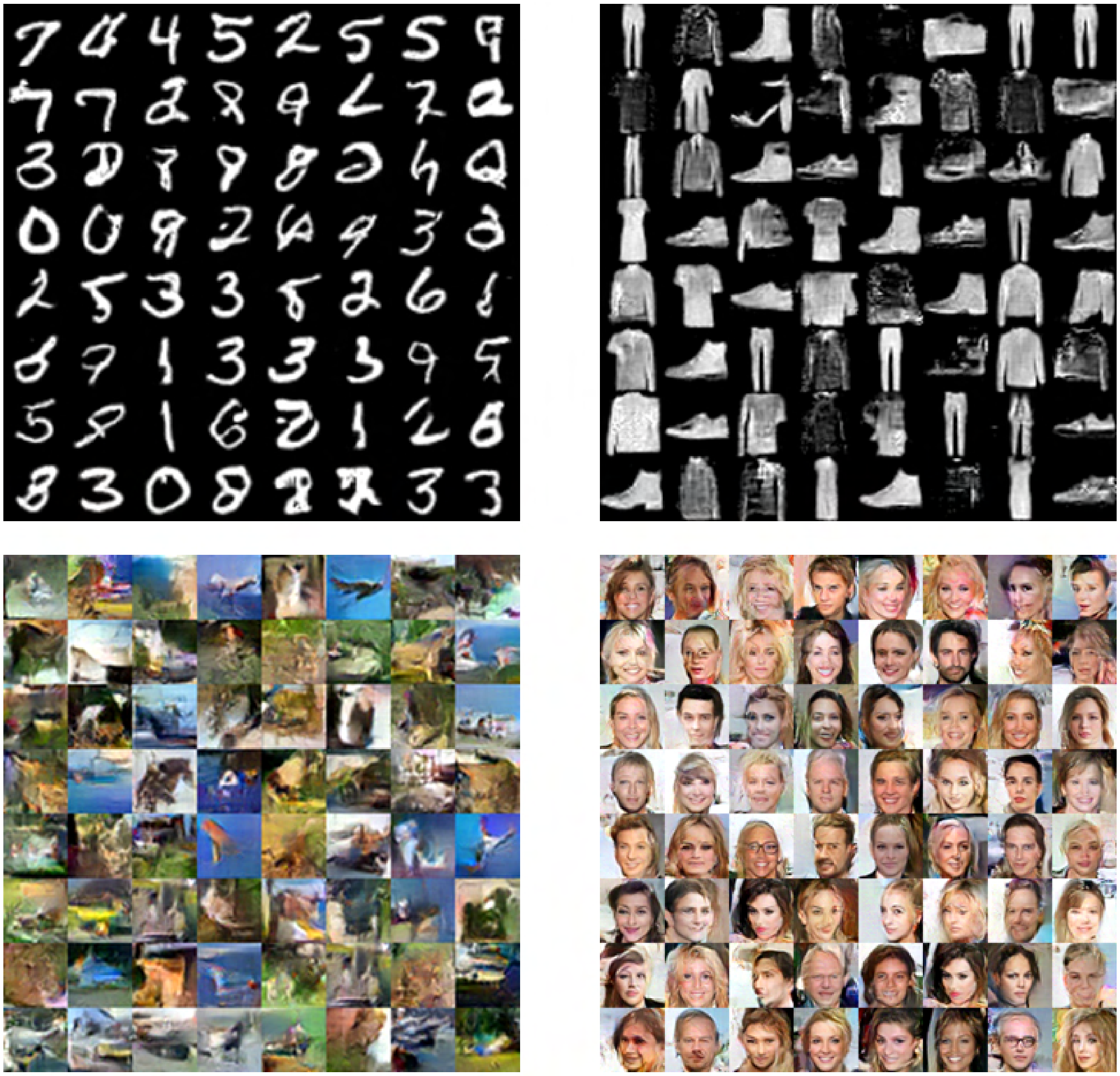


Figure 5.7: Samples from EBMs w/ noise-free dynamics

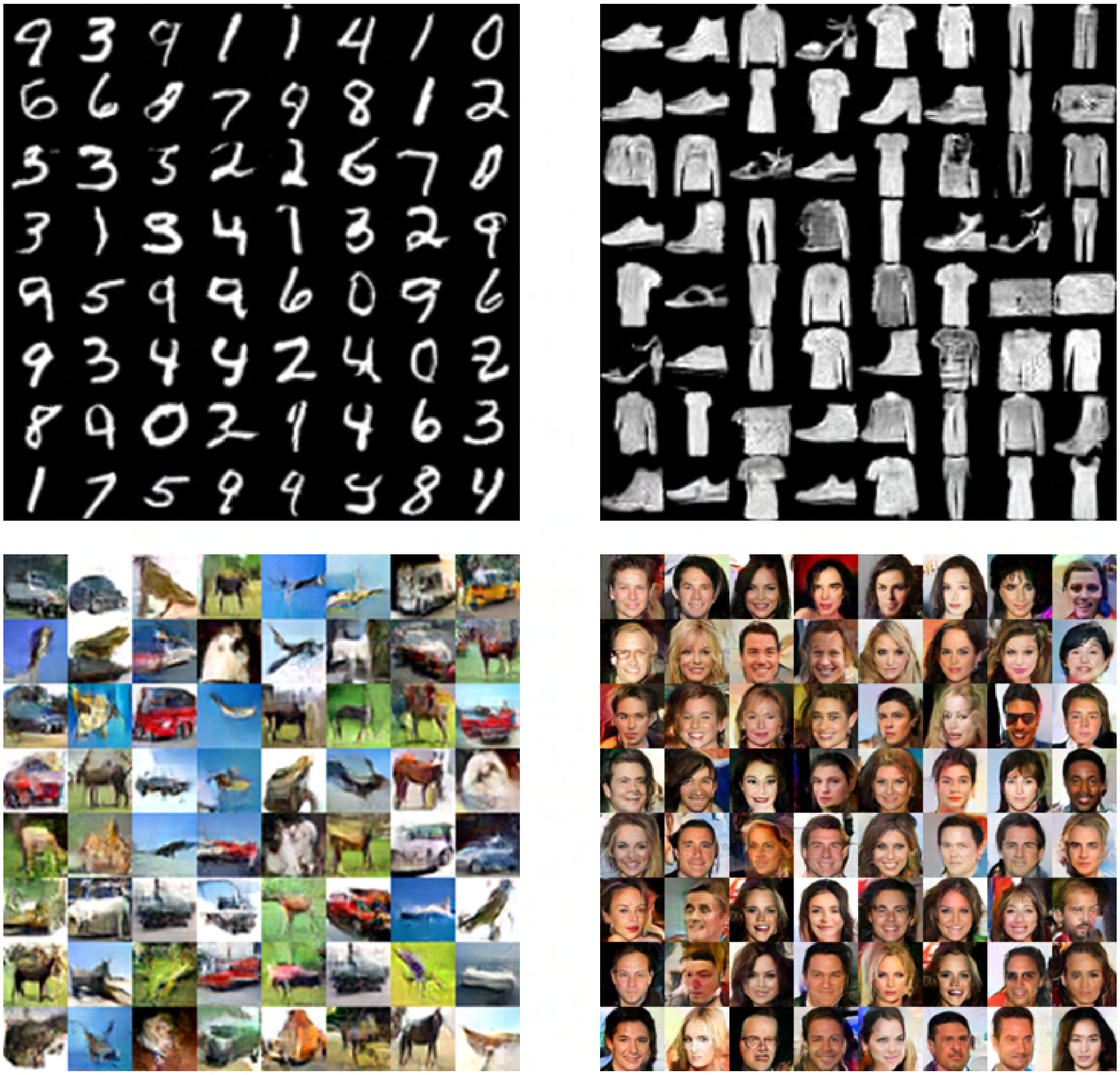


Figure 5.8: Samples from EBMs w/ noise-free dynamics plus extra generator loss

free dynamics on CIFAR-10 in Figure 5.9. We observe that for both models, the losses oscillate around zero, as observed in Nijkamp et al. [2020]. However, the model trained with noisy dynamics diverges after 20000 iterations, while the training of model with noise-free dynamics is much more stable. In addition, we observe that adding the extra generator loss, as discussed in Section 5.4 does not affect the training stability. In contrast, training EBMs with noisy sampling dynamics may still diverge during training. 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.

Treating the noise-free dynamics as generator models, we further apply the additional adversarial loss term for the W-GAN generator update. In particular, we train the model with loss in Equation 5.14. We report the FID in the last line of Table 5.1, where we find the generator update significantly improves the sample quality. This experiment shows that the noise-free dynamics is indeed a generator, and we can use it to train GANs.

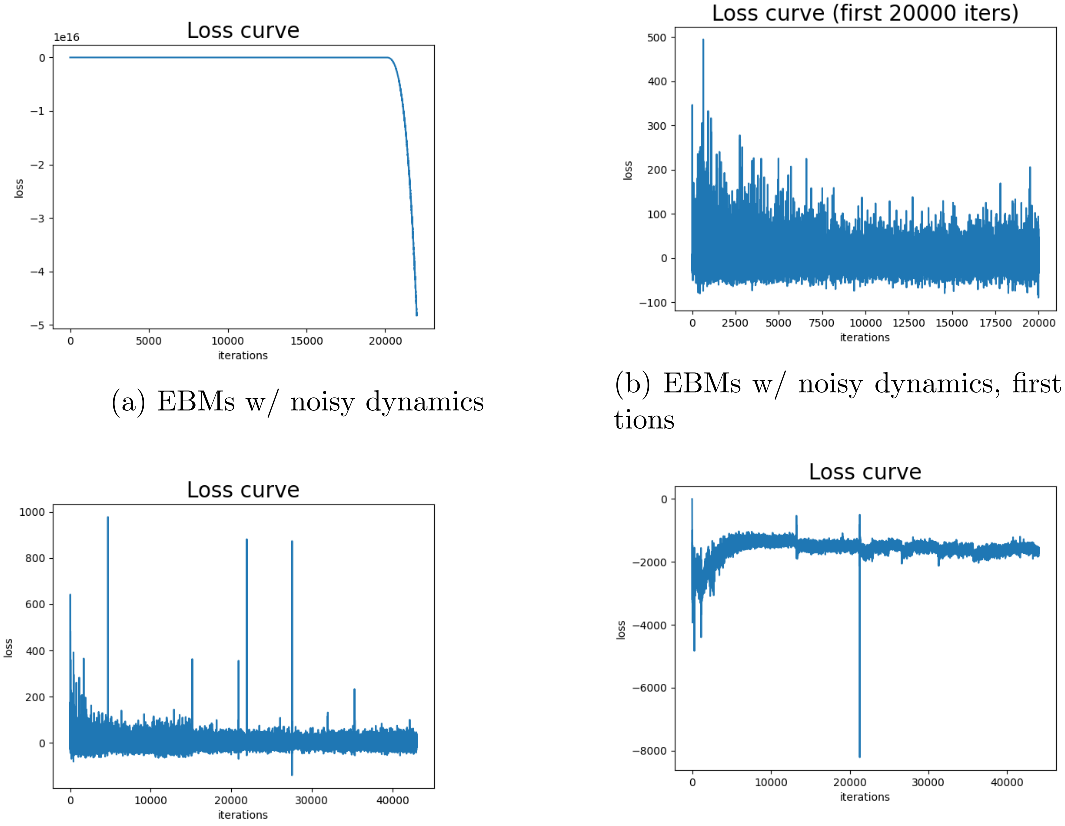
5.5.3 Experimental settings

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

2D toy data

On 2D toy data, we use a 5-layer fully connected networks with 256 hidden units and swish activation function. We train our models with Adam optimizer, with 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 5.6. To do so, we use the solver implemented by Chen et al. [2018a]. We set the initial value to random samples from 2-d 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 neural



(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 5.9: 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.

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 5.2: 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.

ODE.

Image data

We resize MNIST and Fashion-MNIST to 32 × 32. The network structures are presented in Table 5.2. We train all models with Adam optimizer with 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 certain number of iterations. This is also observed in Du and Mordatch [2019] and Xiao et al. [2021a]. 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 that of 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 in Equation 5.8. This can be done by setting the create graph option when computing the gradient in PyTorch's auto differential package [Paszke et al., 2019].

5.6 Conclusion

In this chapter, we provide new insights to understand the maximum likelihood training of EBMs. 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 5.4, 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 gradient flow model, and we can benefit from removing the noise in the sampling dynamics. As a result, 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.

CHAPTER 6
DENOISING DIFFUSION GANS FOR ACCELERATING SAMPLING FROM DENOISING DIFFUSION MODELS

This chapter proposes a model that accelerates the sampling from denoising diffusion models. Specifically, we propose denoising diffusion GAN (DDGAN), which uses conditional GANs to model the denoising distribution in the reverse process of denoising diffusion models. We will begin with motivating our approach and introducing related work. Then we will give a detailed description of our method and present experimental results.

The material of this chapter is based on Xiao et al. [2022].

6.1 Motivation and Introduction

As introduced in Section 2.4, denoising diffusion models [Sohl-Dickstein et al., 2015, Ho et al., 2020] are powerful generative models with many successful applications. Diffusion models define a forward diffusion process that maps data to noise by gradually perturbing the input data. Data generation is achieved using a parameterized reverse process that performs iterative denoising, starting from random noise. They can also be viewed as VAEs with fixed dimensions at all layers [Ho et al., 2020]. Diffusion models demonstrate surprisingly good results in sample quality, beating GANs in image generation [Dhariwal and Nichol, 2021, Ho et al., 2021]. They also demonstrate good mode coverage, indicated by high test likelihood [Song et al., 2021a, Kingma et al., 2021, Huang et al., 2021b]. However, one major drawback of denoising diffusion models is that sampling from them is very slow due to the iterative sampling process. Typically it takes more than 1000 function evaluations to sample from denoising diffusion models, making them difficult to be used in applications that require real-time generation.

Moreover, as discussed in Section 2.6, every deep generative model has its own limitations.

Importantly, we note that there is not a single generative model that can simultaneously excel at the following three criteria:

- Sample quality
- Sample diversity or mode coverage
- Sampling speed

We describe the trade-off between different types of models with the term *generative learning trilemma*, as shown in Figure 6.1. For example, GANs are fast to sample from and have high sample quality, but they are known to suffer from mode collapse. VAEs and normalizing flows are easy to sample from and tend to cover all the modes, but they cannot generate high-quality samples. Denoising diffusion models are excellent in both sample quality and diversity, but they are extremely slow to sample from.

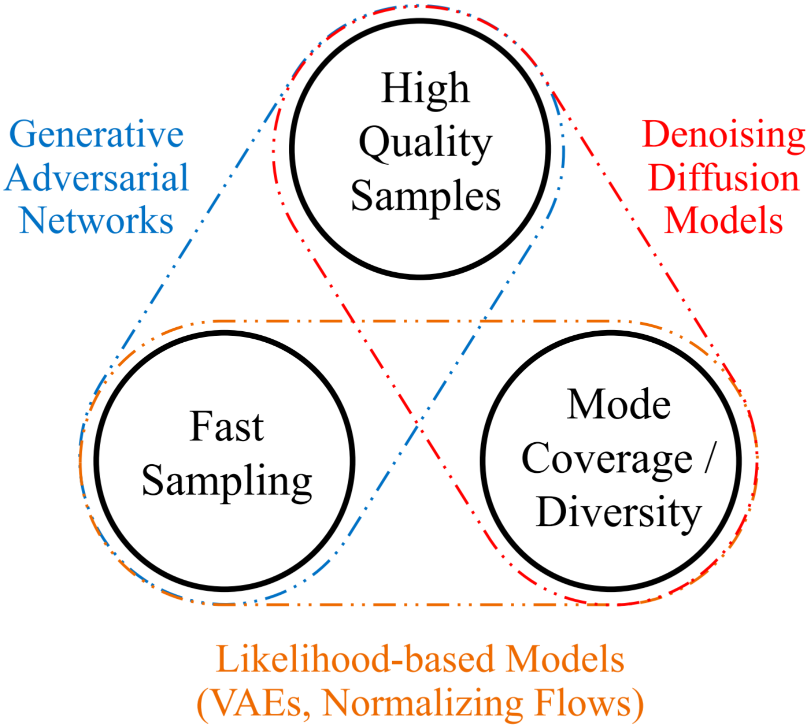

High Quality Samples
Fast Sampling
Mode Coverage / Diversity
Generative Adversarial Networks
Denoising Diffusion Models
Likelihood-based Models (VAEs, Normalizing Flows)

Figure 6.1: Generative learning trilemma.

We want to design a generative model that has a good balance between these three criteria. Previously, there have been many attempts to improve VAEs and normalizing flows [Vahdat and Kautz, 2020, Child, 2021, Kingma and Dhariwal, 2018], but it is observed that even the best VAEs and flows with huge model sizes still cannot generate samples with quality competitive with GANs. Resolving the mode collapse of GANs is also tricky, as this is a fundamental issue of adversarial training. Previous approaches [Srivastava et al., 2017, Dieng et al., 2019] only partially alleviate the issue, and possibly at the cost of reduced sample quality. As a result, we found the direction of accelerating the sampling from denoising diffusion models to be new and promising for tackling the generative learning trilemma.

Some recent papers propose methods that accelerate the sampling from denoising diffusion models, and they will be discussed in 6.2. They still largely follow the same formulations of denoising diffusion models and can reduce the sampling iterations to around 100 steps. Further reducing the number of sampling iterations will result in significant degradation of sample quality. However, a 100-step sampling scheme still takes a lot of time and is impractical for real-time applications. Our goal is to significantly speed-up sampling from denoising diffusion models. In other words, we want to generate high-quality samples in a few steps.

We carefully investigate the slow sampling issue of denoising diffusion models, and we note that diffusion models commonly assume that the denoising distribution can be approximated by Gaussian distributions. As discussed in Section 2.4.1, the denoising distribution is parameterized by

$$p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \mu_\theta\left(\mathbf{x}_t, t\right), \Sigma_\theta\left(\mathbf{x}_t, t\right)\right), \tag{6.1}$$

and the training goal is to let $p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)$ match the true denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. While the true denoising distribution is unknown (otherwise, we can directly use it to denoise samples), it is known that for continuous diffusion (in the limit of small step size), the reversal of the diffusion process has the identical functional form as the forward process

[Feller, 1949]. Since in the forward process, $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is a Gaussian distribution, if the step size is infinitesimally small, the true denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is also a Gaussian. In this case, we can parameterize the denoising distribution with a Gaussian as in Equation 6.1. As a result, previous denoising diffusion models use a very small step size (hence a very large number of steps) to ensure that the true denoising distribution is close to a Gaussian.

If we want to significantly reduce the number of sampling steps, we need to know what the true denoising distribution looks like when the denoising step size is large. Intuitively, given a noisy observation $\mathbf{x}_t$, if we want to denoise for a small step and obtain $\mathbf{x}_{t-1}$, the distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ has a single mode as the condition $\mathbf{x}_t$ contains most of the information about $\mathbf{x}_{t-1}$. However, if we denoise $\mathbf{x}_t$ for a large step, there might be multiple plausible $\mathbf{x}_{t-1}$ and hence $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ will be a multi-modal distribution which cannot be approximated by a Gaussian. The difference between small and large step sizes is illustrated in Figure 6.2.

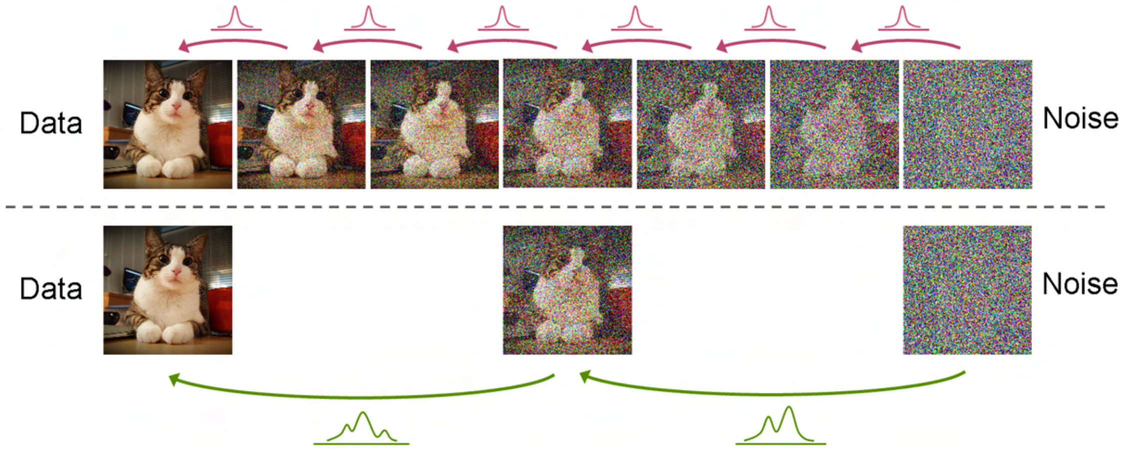


Figure 6.2: Comparison between large and small denoising step sizes. **Top:** when the step size is small, the true denoising distribution is single modal and can be approximated with a Gaussian. **Bottom:** when the step size is large, the true denoising distribution is multi-modal and cannot be approximated by a Gaussian.

We present a concrete example of 1-d toy data in Figure 6.3, where we plot the true denoising distribution with different step sizes. Note that although in general we cannot compute the true denoising distribution in closed form, in 1-d we can compute

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t) = \int_{\mathbf{x}_0} q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)q(\mathbf{x}_0)d\mathbf{x}_0$$

with numerical integration, where $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is the Gaussian posterior in Equation 2.50. We observe that the true denoising distribution becomes more complex and multimodal as the step size increases.

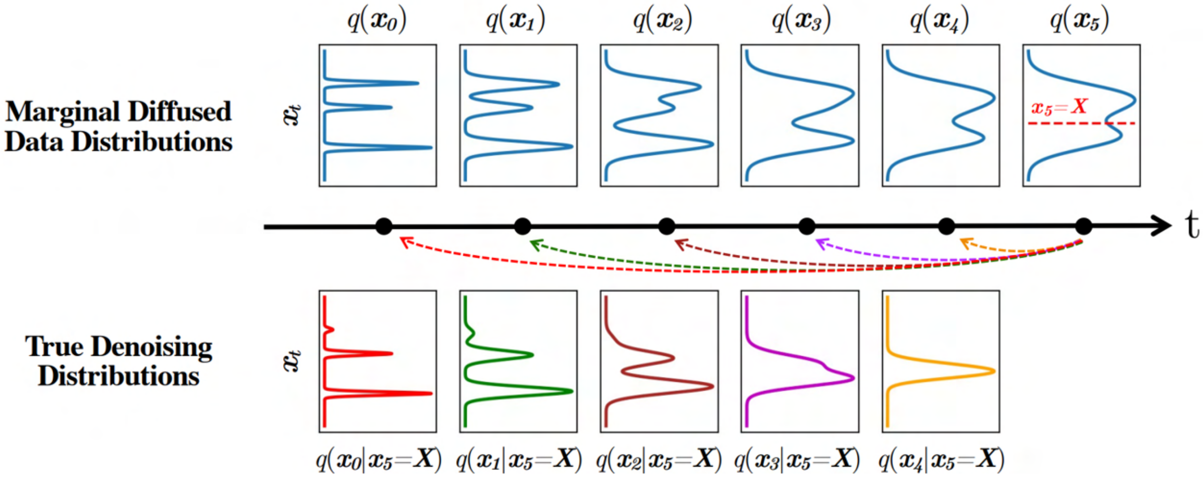


Figure 6.3: **Top**: The evolution of 1D data distribution $q(\mathbf{x}_0)$ through the diffusion process. The distribution of $\mathbf{x}_0$ is a Gaussian mixture. **Bottom**: The visualization of the true denoising distribution for varying step sizes conditioned on a fixed $\mathbf{x}_5$. The true denoising distribution for a small step size (i.e., $q(\mathbf{x}_4|\mathbf{x}_5 = X)$) is close to a Gaussian distribution. However, it becomes more complex and multimodal as the step size increases.

Inspired by this observation, we propose to parameterize the denoising distribution with an expressive multimodal distribution to enable denoising for large steps. We ensure the denoising distribution to be expressive by parameterizing it with a deep generative model instead of a simple Gaussian distribution, and we ensure the denoising distribution to be multimodal by using latent variables, which lead to diverse output given the same condition $\mathbf{x}_t$. In particular, we introduce a novel generative model, termed as *denoising diffusion GAN*, in which the denoising distributions are modeled with conditional GANs. In a denoising diffusion GAN, the generator tries to generate a sample of denoised $\mathbf{x}_{t-1}$ conditioned on $\mathbf{x}_t$, and the discriminator tries to distinguish between generated $\mathbf{x}_{t-1}$ and real $\mathbf{x}_{t-1}$ sampled from the forward diffusion process. In image generation, we observe that our model obtains sample quality and mode coverage competitive with diffusion models while taking only as few as two denoising steps, achieving about 2000× speed-up in sampling compared to the predictor-corrector sampling by Song et al. [2021b] on CIFAR-10.

One potential drawback of our model is that it introduces adversarial training, which may cause mode collapse and training instability issues. However, we observe that our denoising diffusion GAN does not suffer from mode collapse, and the training is stable. Compared to traditional GANs, we show that our model significantly outperforms state-of-the-art GANs in sample diversity while being competitive in sample fidelity. There are two possible explanations. Firstly, our model breaks the generation task of GANs into several easier conditional generation tasks, where each task is to perform a single denoising step. It is known that conditional GANs are easier to train and suffer less from mode collapse than unconditional GANs. For example, all GANs trained on ImageNet dataset require label conditioning [Brock et al., 2018]. Our method provides a natural way to formulate the generation problem with several conditional sub-problems, even when label information is unavailable. Therefore, from the perspective of GAN training, our method can be seen as a self-supervised conditional GAN. Secondly, one important reason of mode collapse and training instability is the overfitting of the discriminator, where the discriminator can distinguish between real and fake samples too easily. In our model, the discriminator needs to distinguish between real and fake $\mathbf{x}_{t-1}$, which is noisy except for the last denoising step. The diffusion process smooths the data distribution, and it is more difficult to judge noisy samples. Therefore our method provides a natural regularization to the discriminator.

In summary, we make the following contributions:

- We attribute the slow sampling of diffusion models to the Gaussian assumption in the denoising distribution and propose to employ complex, multimodal denoising distributions.

- We propose denoising diffusion GANs, a diffusion model whose reverse process is parameterized by conditional GANs.
- Through careful evaluations, we demonstrate that denoising diffusion GANs achieve several orders of magnitude speed-up compared to current diffusion models for both image generation and editing. We show that our model overcomes the deep generative learning trilemma to a large extent, making diffusion models for the first time applicable to interactive, real-world applications at a low computational cost.

6.2 Related Work

Diffusion-based models [Sohl-Dickstein et al., 2015, Ho et al., 2020] learn the finite-time reversal of a diffusion process, sharing the idea of learning transition operators of Markov chains with Goyal et al. [2017], Alain et al. [2016], Bordes et al. [2017]. Since then, there have been several improvements and alternatives to diffusion models. Song et al. [2021b] generalize diffusion processes to continuous time, and provide a unified view of diffusion models and denoising score matching [Vincent, 2011, Song and Ermon, 2019]. Jolicoeur-Martineau et al. [2021b] add an auxiliary adversarial loss to the main objective. This is fundamentally different from ours, as their auxiliary adversarial loss only acts as an image enhancer, and they do not use latent variables; therefore, the denoising distribution is still a unimodal Gaussian.

One major drawback of diffusion or score-based models is the slow sampling speed due to a large number of iterative sampling steps. To alleviate this issue, multiple methods have been proposed. Luhman and Luhman [2021] use knowledge distillation to distill a multi-step denoising process into a single step. After training the denoising diffusion model, they generate a large number of samples that serve as their training set and train a network to predict the generated sample given the initial noise input. However, training their method requires generating samples, which can be time-consuming. Moreover, the iterative sampling process is a random process, and it is unclear whether it can be represented by a deterministic transformation of the initial noise. Their method has a significant degradation in sample quality. [San-Roman et al., 2021] propose a learning scheme that can step-by-step adjust the noise scheduling parameters for any given number of steps. However, the objective for the adjustment is data likelihood, and it turns out that such a method cannot generate high quality samples. Song et al. [2020a] generalizes the forward diffusion process of denoising diffusion models, which is Markovian, to non-Markovian ones. They show that resulting variational training objectives have a shared surrogate objective, which is exactly the objective used to train original denoising diffusion models. Therefore, they can use pre-trained denoising diffusion models and sampling with the corresponding reverse process associated with the non-Markovian diffusion process, and they show that this results in significant speed-ups. Similar ideas are used in [Kong and Ping, 2021]. Jolicoeur-Martineau et al. [2021a] propose to use better SDE solvers that solve the reverse SDE faster than the naive Euler solver for continuous-time diffusion models. LSGM [Vahdat et al., 2021] formulates a diffusion model in the latent space of a VAE, and the VAE and the diffusion model are trained jointly with ELBO objective. LSGM requires fewer sampling steps than denoising diffusion models on the data space. This is because if the data marginal $q(\mathbf{x}_t)$ is Gaussian, the true denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is also a Gaussian distribution. The encoder of the VAE brings the data distribution $q(\mathbf{x}_0)$ and consequently $q(\mathbf{x}_t)$ closer to Gaussian. However, the problem of transforming the data to Gaussian itself is challenging, and VAE encoders cannot solve it perfectly. As a result, LSGM still requires tens to hundreds of steps on complex datasets.

Among variants of diffusion models, Gao et al. [2021] have the closest connection with our method. They propose to model the single-step denoising distribution by a conditional energy-based model (EBM), sharing the high-level idea of using expressive denoising distributions with us. However, they motivate their method from the perspective of facilitating the training of EBMs. More importantly, although only a few denoising steps are needed, expensive MCMC has to be used to sample from each denoising step, making the sampling process slow with ∼180 network evaluations. ImageBART [Esser et al., 2021a] explores modeling the denoising distribution of a diffusion process on discrete latent space with an auto-regressive model per step in a few denoising steps. However, the auto-regressive structure of their denoising distribution still makes sampling slow.

Since our model is trained with adversarial loss, our work is related to recent advances in improving the sample quality and diversity of GANs, including data augmentation [Zhao et al., 2020, Karras et al., 2020a], consistency regularization [Zhang et al., 2019, Zhao et al., 2021] and entropy regularization [Dieng et al., 2019]. In addition, the idea of training generative models with smoothed distributions is also discussed in Meng et al. [2021a] for auto-regressive models.

6.3 Denoising Diffusion GANs

Our goal is to reduce the number of denoising diffusion steps T required in the reverse process of diffusion models. Inspired by the discussion in Section 6.1, we propose to model the denoising distribution with an expressive multimodal distribution. Since conditional GANs have been shown to model complex conditional distributions in the image domain [Mirza and Osindero, 2014, Ledig et al., 2017, Isola et al., 2017], we adopt them to approximate the true denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$.

Specifically, our forward diffusion is set up similarly to the diffusion models in Equation 2.44 with the main assumption that T is assumed to be small ($T \leq 8$) and each diffusion step has larger β_t. Our training is formulated by matching the conditional GAN generator $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ and $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ using an adversarial loss that minimizes a divergence D_{adv} per denoising step:

$$\min_{\theta} \sum_{t\geq 1} \mathbb{E}_{q(\mathbf{x}_t)} \left[D_{\mathrm{adv}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) \right], \tag{6.2}$$

where D_{adv} can be Wasserstein distance, Jenson-Shannon divergence, or f-divergence depending on the adversarial training setup [Arjovsky et al., 2017, Goodfellow et al., 2014, Nowozin et al., 2016]. In our model, we rely on non-saturating GANs [Goodfellow et al., 2014] that are widely used in successful GAN frameworks such as StyleGANs [Karras et al., 2019, 2020b]. In this case, D_{adv} takes a special instance of f-divergence called *softened reverse KL* [Shannon et al., 2020], which is different from the forward KL divergence used in the original denoising diffusion model training in Equation 2.54.

To set up the adversarial training, we denote the time-dependent discriminator network as $D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t) : \mathbb{R}^N \times \mathbb{R}^N \times \mathbb{R} \to [0, 1]$, with parameters ϕ. It takes the N-dimensional $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ as inputs, and decides whether $\mathbf{x}_{t-1}$ is a plausible denoised version of $\mathbf{x}_t$. The discriminator is trained by:

$$\min_{\phi} \sum_{t\geq 1} \mathbb{E}_{q(\mathbf{x}_t)} \Big[\mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_t)} [-\log(D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))] + \mathbb{E}_{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} [-\log(1 - D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))] \Big], \tag{6.3}$$

where fake samples from $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are contrasted against real samples from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. The first expectation requires sampling from $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ which is unknown. However, we use the identity $q(\mathbf{x}_t, \mathbf{x}_{t-1}) = \int d\mathbf{x}_0 q(\mathbf{x}_0) q(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{x}_0) = \int d\mathbf{x}_0 q(\mathbf{x}_0) q(\mathbf{x}_{t-1}|\mathbf{x}_0) q(\mathbf{x}_t|\mathbf{x}_{t-1})$ to rewrite the first expectation in Equation 6.3 as:

$$\mathbb{E}_{q(\mathbf{x}_t) q(\mathbf{x}_{t-1}|\mathbf{x}_t)} \left[-\log \left(D_\phi \left(\mathbf{x}_{t-1}, \mathbf{x}_t, t \right) \right) \right] = \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{x}_{t-1}|\mathbf{x}_0) q(\mathbf{x}_t|\mathbf{x}_{t-1})} \left[-\log \left(D_\phi \left(\mathbf{x}_{t-1}, \mathbf{x}_t, t \right) \right) \right] \tag{6.4}$$

Given the discriminator, we train the generator by

$$\max_{\theta} \sum_{t \geq 1} \mathbb{E}_{q(\mathbf{x}_t)} \mathbb{E}_{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} [\log(D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))], \tag{6.5}$$

which updates the generator with the non-saturating GAN objective.

It is noteworthy that when we have only 1 time step, our model corresponds to training an unconditional GAN, as the conditioning $\mathbf{x}_1$ is a white noise and contains no information about $\mathbf{x}_0$.

6.3.1 Parameterizing the Implicit Denoising Model

Instead of directly predicting $\mathbf{x}_{t-1}$ in the denoising step, diffusion models [Ho et al., 2020] can be interpreted as parameterizing the denoising model by

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0 = f_\theta(\mathbf{x}_t, t)), \tag{6.6}$$

in which first $\mathbf{x}_0$ is predicted using the denoising model $f_\theta(\mathbf{x}_t, t)$, and then, $\mathbf{x}_{t-1}$ is sampled using the posterior distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ given $\mathbf{x}_t$ and the predicted $\mathbf{x}_0$. The distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t)$ is intuitively the distribution over $\mathbf{x}_{t-1}$ when denoising from $\mathbf{x}_t$ towards $\mathbf{x}_0$, and it always has a Gaussian form for the diffusion process in Equation 2.44, independent of the step size and complexity of the data distribution. The form of $q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t)$ is introduced in Equation 2.50.

The interpretation of sampling step as x_0 prediction followed by posterior sampling is not obvious from the original description in Ho et al. [2020], but it is discussed by Song et al. [2020a]. Here we provide arguments to make it clear. We want to show that parameterization of the denoising distribution for current diffusion models such as Ho et al. [2020] can be interpreted as $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0 = f_\theta(\mathbf{x}_t, t))$. Ho et al. [2020] train a noise prediction network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ which predicts the noise that perturbs data $\mathbf{x}_0$ to $\mathbf{x}_t$, and a sample from $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is obtained as (see Algorithm 2 of Ho et al. [2020])

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta\left(\mathbf{x}_t, t\right)\right) + \sigma_t \mathbf{z}, \tag{6.7}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ except for the last denoising step where $\mathbf{z} = 0$, and $\sigma_t = \sqrt{\tilde{\beta}_t}$ is the standard deviation of the Gaussian posterior distribution in Equation 2.51.

Firstly, notice that predicting the perturbation noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is equivalent to predicting $\mathbf{x}_0$. We know that $\mathbf{x}_t$ is generated by adding $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ noise as:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \tag{6.8}$$

Hence, after predicting the noise with $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ we can obtain a prediction of $\mathbf{x}_0$ using:

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right). \tag{6.9}$$

Next, we can plug the expression for $\mathbf{x}_0$ in Equation 6.9 into the mean of the Gaussian posterior distribution in Equation 2.51, and we have

$$\begin{align}
\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \tilde{\mu}_t\left(\mathbf{x}_t, \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(\mathbf{x}_t, t)\right)\right) \tag{6.10}\\
&= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) \tag{6.11}
\end{align}$$

after simplifications. Comparing this with Equation 6.7, we observe that Equation 6.7 simply corresponds to sampling from the Gaussian posterior distribution. Therefore, although Ho et al. [2020] use an alternative re-parameterization, their denoising distribution can still be equivalently interpreted as $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0 = f_\theta(\mathbf{x}_t, t))$, i.e, first predicting $\mathbf{x}_0$ using the time-dependent denoising model $f_\theta(\mathbf{x}_t, t)$, and then sampling $\mathbf{x}_{t-1}$ using the posterior distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ given $\mathbf{x}_t$ and the predicted $\mathbf{x}_0$. Hence we show that the parameterization of Ho et al. [2020] is equivalent to what we describe.

We follow the parameterization of Ho et al. [2020] and define $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ by:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \int p_\theta(\mathbf{x}_0|\mathbf{x}_t) q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) d\mathbf{x}_0 = \int p(\mathbf{z}) q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0 = G_\theta(\mathbf{x}_t, \mathbf{z}, t)) d\mathbf{z}, \tag{6.12}$$

where $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ is the implicit distribution imposed by the GAN generator $G_\theta(\mathbf{x}_t, \mathbf{z}, t) : \mathbb{R}^N \times \mathbb{R}^L \times \mathbb{R} \to \mathbb{R}^N$ that outputs $\mathbf{x}_0$ given $\mathbf{x}_t$ and an L-dimensional latent variable $\mathbf{z} \sim p(\mathbf{z}) := \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$.

Our parameterization has several advantages: Firstly, our $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is formulated similar to DDPM [Ho et al., 2020]. Thus, we can borrow some inductive biases such as the network structure design from DDPM. The main difference is that, in DDPM, $\mathbf{x}_0$ is predicted as a deterministic mapping of $\mathbf{x}_t$, while in our case, $\mathbf{x}_0$ is produced by the generator with random latent variable $\mathbf{z}$. This is the key difference that allows our denoising distribution $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ to become multimodal and complex in contrast to the unimodal denoising model in DDPM. Secondly, note that for different t's, $\mathbf{x}_t$ has different levels of perturbation, and hence using a single network to predict $\mathbf{x}_{t-1}$ directly at different t may be difficult. However, in our case the generator only needs to predict unperturbed $\mathbf{x}_0$ and then add back perturbation using $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$. Figure 6.4 visualizes our training pipeline.

We present pseudo codes of our training pipeline in Algorithm 1 and sampling process in Algorithm 2.

6.3.2 Network Design

Network architecture is a critical component in the design of denoising diffusion GAN. Previous conditional GANs typically use a decoder structure for the generator, where the conditioning information (such as label) is added to the network with embedding. In our case, the condition $\mathbf{x}_t$ is very important as the desired output of the generator is a less noisy

Algorithm 1 A training iteration of denoising diffusion GAN

```
Require: Number of time steps $T$, training sample $\mathbf{x}$, time-dependent discriminator $D_\phi$, time dependent generator $G_\theta$.
Sample $t$ uniformly from $\{0, \cdots, T-1\}$.
Using $\mathbf{x}$ as initial data, sample $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ from the forward diffusion process with Equation 2.46 and Equation 2.44.
Sample $\mathbf{z} \sim \mathcal{N}(0, I)$, and generate $\mathbf{x}_0' = G_\theta(\mathbf{x}_{t+1}, \mathbf{z}, t)$.
Sample $\mathbf{x}_t'$ using $\mathbf{x}_{t+1}$ and $\mathbf{x}_0'$ from the Gaussian posterior distribution in Equation 2.50.
Update the discriminator by minimizing
    $-\log\left(D_\phi\left(\mathbf{x}_t, \mathbf{x}_{t+1}, t\right)\right) - \log\left(1 - D_\phi\left(\mathbf{x}_t', \mathbf{x}_{t+1}, t\right)\right)$
w.r.t $\phi$.
Update the generator by maximizing
    $\log\left(D_\phi\left(\mathbf{x}_t', \mathbf{x}_{t+1}, t\right)\right)$
w.r.t $\theta$.
```

Algorithm 2 Sampling from denoising diffusion GAN

```
Require: Number of time steps $T$, time dependent generator $G_\theta$.
Sample $\mathbf{x}_T \sim \mathcal{N}(0, I)$.
for $t = T-1, \cdots, 0$ do
    Sample $\mathbf{z} \sim \mathcal{N}(0, I)$, and generate $\mathbf{x}_0' = G_\theta(\mathbf{x}_{t+1}, \mathbf{z}, t)$.
    Sample $\mathbf{x}_t'$ using $\mathbf{x}_{t+1}$ and $\mathbf{x}_0'$ from the Gaussian posterior distribution in Equation 2.50.
$\mathbf{x}_0'$ is an sample from the denoising diffusion GAN.
```

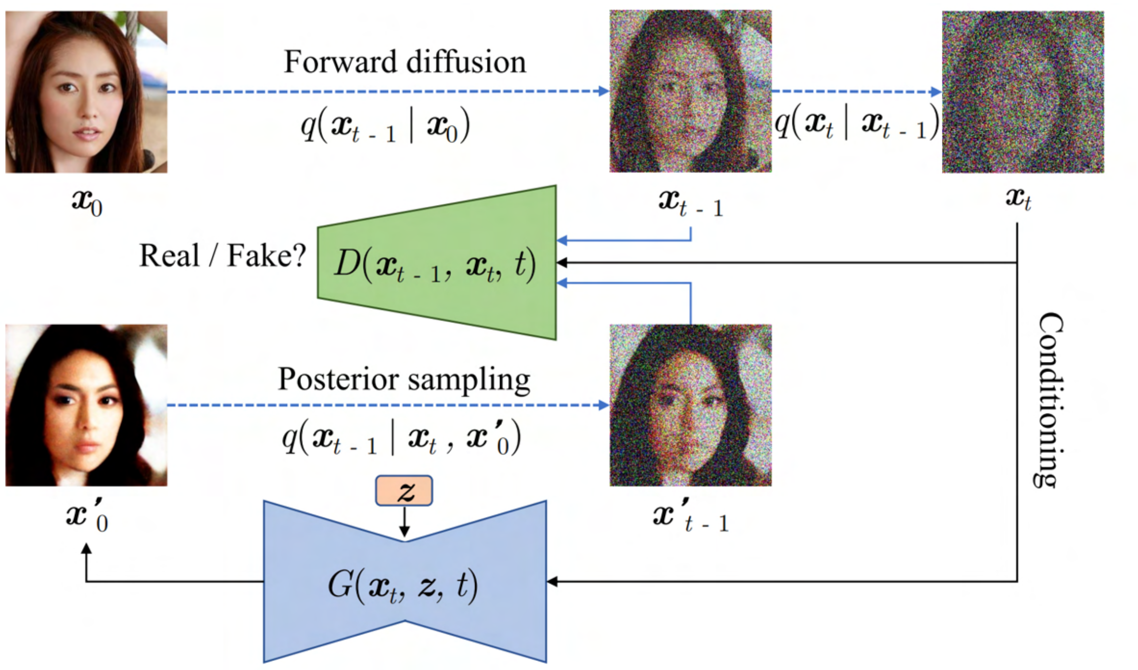

Forward diffusion
$q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0)$
$q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})$
$\boldsymbol{x}_0$
$\boldsymbol{x}_{t-1}$
$\boldsymbol{x}_t$
Real / Fake?
$D(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, t)$
Conditioning
Posterior sampling
$q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}'_0)$
$\boldsymbol{z}$
$\boldsymbol{x}'_0$
$\boldsymbol{x}'_{t-1}$
$G(\boldsymbol{x}_t, \boldsymbol{z}, t)$

Figure 6.4: The training process of denoising diffusion GAN.

version of $\mathbf{x}_t$ (after the posterior sampling). Therefore, we cannot embed $\mathbf{x}_t$ as conditioning, as we need the value of each pixel of $\mathbf{x}_t$ rather than a vector of abstract representation to perform denoising. Therefore, we adopt a U-net structure for the generator, similar to the noise prediction network in Ho et al. [2020]. In our generator, the condition $\mathbf{x}_t$ is the input to the network. With this design, the initial layer input to the generator is no longer the noise vector $\mathbf{z}$, so we borrow the idea of StyleGAN, whose input to the generator is a shared constant tensor, to inject $\mathbf{z}$ to the network. StyleGAN is introduced in Section 2.5.3. Simply speaking, StyleGAN transforms the noise vector $\mathbf{z}$ with fully connected layers and uses the output to control the per-channel shift and scale parameters of normalization layers. We adopt the same idea and design the *adaptive group normalization* module, which controls the shift and scale parameters of the group normalization [Wu and He, 2018] module with transformations of $\mathbf{z}$. Group normalization divides the channels into groups, where each group consists of a fixed number of consecutive channels and computes within each group the mean and variance for normalization. In our case, given an input tensor, $\mathbf{w} \in \mathbb{R}^{M \times N \times C}$, the adaptive group normalization module outputs a normalized tensor $\tilde{\mathbf{w}} \in \mathbb{R}^{M \times N \times C}$ in the following way. Firstly, the latent variable $\mathbf{z}$ is transformed to shift and scale parameters for each channel using fully connected layers $\mu(\mathbf{z}) \in \mathbb{R}^C$ and $\sigma(\mathbf{z}) \in \mathbb{R}^C$, and then the grouped normalized tensor $\bar{\mathbf{w}} = \mathrm{GN}(\mathbf{w})$ is transformed by

$$\tilde{\mathbf{w}} = \mu(\mathbf{z}) + \sigma(\mathbf{z})\bar{\mathbf{w}} \tag{6.13}$$

to obtain $\tilde{\mathbf{w}}$. The time conditioning to the generator is enforced by time embedding techniques similar to Ho et al. [2020].

Our discriminator needs to discriminate between real and fake $\mathbf{x}_{t-1}$ conditioned on $\mathbf{x}_t$. To do so, we concatenate $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ in the channel dimension, and the resulting concatenated tensor serves as the input to the discriminator. The discriminator has a common convolutional structure consisting of multiple ResNet blocks. The time conditioning to the generator is enforced by the same time embedding as the generator.

6.3.3 Diffusion Process

Since we are using a very small number of diffusion steps, it is important to choose the diffusion process and allocate each step in the process. In order to compute β_t per step, we use the discretization of the continuous-time extension of the process described in Equation 2.44, which is called the Variance Preserving (VP) SDE by Song et al. [2021b]. We compute β_t based on the continuous-time diffusion model formulation, as it allows us to ensure that the variance schedule stays the same independent of the number of diffusion steps. Let's define the normalized time variable by $t' := \frac{t}{T}$ which normalizes t to $[0, 1]$. The variance function of VP SDE is given by

$$\sigma^2(t') = 1 - e^{-\beta_{\min} t' - 0.5(\beta_{\max} - \beta_{\min}) t'^2}, \tag{6.14}$$

with the constants $\beta_{\max} = 20$ and $\beta_{\min} = 0.1$. Recall that sampling from t^{th} step in the forward diffusion process can be done with $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$. We compute β_t by solving $1 - \bar{\alpha}_t = \sigma^2(\frac{t}{T})$:

$$\beta_t = 1 - \alpha_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}} = 1 - \frac{1 - \sigma^2(\frac{t}{T})}{1 - \sigma^2(\frac{t-1}{T})} = 1 - e^{-\beta_{\min}(\frac{1}{T}) - 0.5(\beta_{\max} - \beta_{\min})\frac{2t-1}{T^2}} \tag{6.15}$$

6.4 Experimental Results

This section evaluates our proposed denoising diffusion GAN for the image synthesis problem. First, we will present our main results of how our method overcomes the generative learning trilemma on CIFAR-10. Then we will discuss ablation studies to better understand our method, as well as additional results on mode coverage and training stability. Results on high-resolution images and stroke-based generation will also be provided.

6.4.1 Overcoming the Generative Learning Trilemma

One major highlight of our model is that it excels at all three criteria in the generative learning trilemma. Here, we carefully evaluate our model's performances on sample fidelity, sample diversity, and sampling time and benchmark it against a comprehensive list of models on the CIFAR-10 dataset.

Evaluation criteria: We adopt the commonly used Fréchet inception distance (FID) [Heusel et al., 2017] and Inception Score (IS) [Salimans et al., 2016] for evaluating sample fidelity. For sample diversity, we use the improved recall score from Kynkäänniemi et al. [2019], which is an improved version of the original precision and recall metric proposed by Sajjadi et al. [2018]. It is shown that an improved recall score reflects how the variation in the generated samples matches that in the training set [Kynkäänniemi et al., 2019]. For sampling time, we use the number of function evaluations (NFE) and the clock time when generating a batch of 100 images on a V100 GPU.

Results: We present our quantitative results in Table 6.1. We observe that our sample quality is competitive among the best diffusion models and GANs. Although some variants of diffusion models obtain better IS and FID, they require a large number of function evaluations to generate samples (while we use only 4 denoising steps). For example, our sampling time is about 2000× faster than the predictor-corrector sampling by Song et al. [2021b] and ∼20× faster than FastDDPM [Kong and Ping, 2021]. Note that diffusion models can produce samples in fewer steps while trading off the sample quality. To better benchmark our method against existing diffusion models, we plot the FID score versus the sampling time of diffusion models by varying the number of denoising steps (or the error tolerance for continuous-time models) in Figure 6.6. The figure clearly shows the advantage of our model compared to previous diffusion models. In particular, we achieve a nearly 40× speed-up over the fastest denoising diffusion models while maintaining similar sample quality.

When comparing our model to GANs, we observe that only StyleGAN2 with adaptive data augmentation has a slightly better sample quality than ours. However, from Table 6.1, we see that GANs have limited sample diversity, as their recall scores are below 0.5. In contrast, our model obtains a significantly better recall score, even higher than several advanced likelihood-based models, and is competitive among diffusion models. We show qualitative samples of CIFAR-10 in Figure 6.7.

In summary, our model simultaneously excels at sample quality, sample diversity, and sampling speed and tackles the generative learning trilemma to a large extent. We visualize our model's position in the trilemma in Figure 6.5.

6.4.2 Ablation Studies

Here, we provide additional insights into our model by performing ablation studies.

Number of denoising steps: The number of denoising steps (T) is an important hyper-parameter of our model. In the first part of Table 6.2, we study the effect of using a

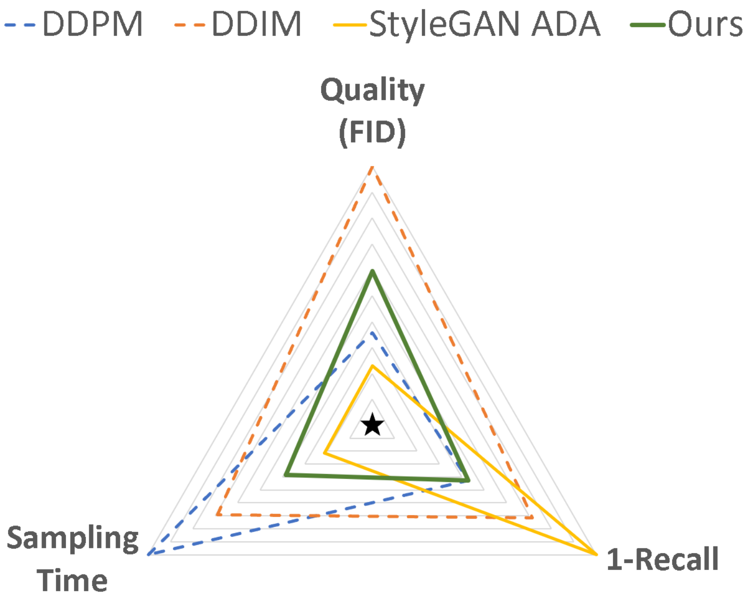


Figure 6.5: Comparing denoising diffusion GAN with other models in the generative learning trilemma.

different number of denoising steps (T). We observe that $T = 1$, which is the unconditional GAN case, leads to significantly worse results with low sample diversity, indicated by the low recall score. This confirms the benefits of breaking generation into several denoising steps, especially for improving the sample diversity. When varying $T > 1$, we observe that $T = 4$ gives the best results, whereas there is a slight degradation in performance for larger T. We hypothesize that we may require a significantly higher capacity to accommodate larger T, as we need a conditional GAN for each denoising step.

Diffusion as data augmentation: Our model shares some similarities with recent work on applying data augmentation to GANs [Karras et al., 2020a, Zhao et al., 2020]. To study the effect of perturbing inputs, we train a one-shot GAN with our network structure following the protocol in [Zhao et al., 2020] with the forward diffusion process as data augmentation. Specifically, in Zhao et al. [2020], a differentiable transformation F is applied to the sample $\mathbf{x}$ (both real and generated) before sending it to the discriminator, and the generator is updated by back-propagating through F. In our study, we choose F to be

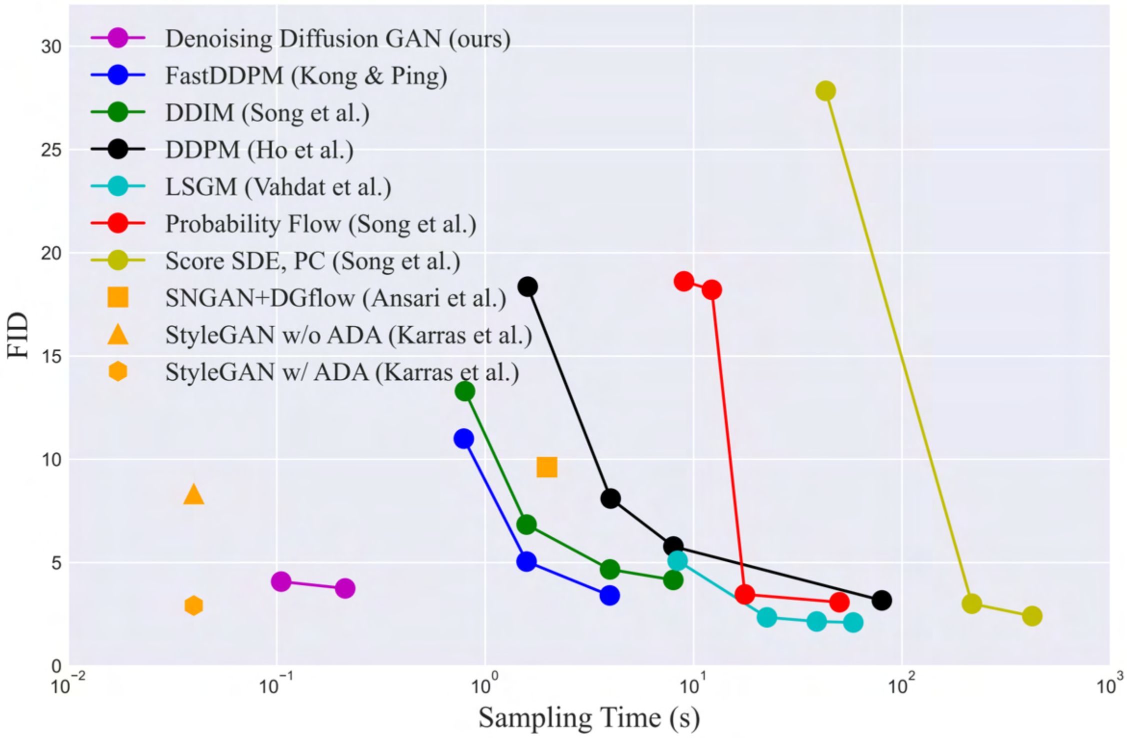


Figure 6.6: Sample quality vs sampling time trade-off.

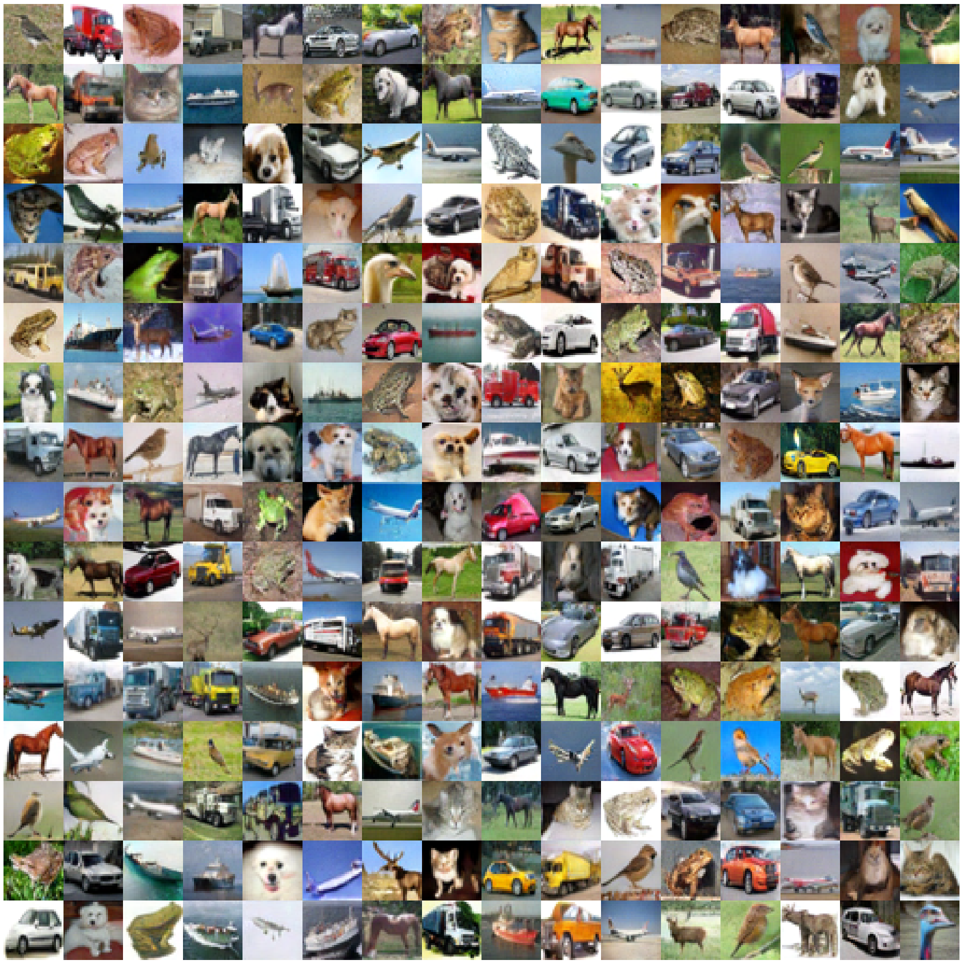


Figure 6.7: CIFAR-10 qualitative samples of denoising diffusion GAN.

Table 6.1: Results for unconditional generation on CIFAR-10.

Model	IS↑	FID↓	Recall↑	NFE ↓	Time (s) ↓
Denoising Diffusion GAN (ours), T=4	9.63	3.75	0.57	4	0.21
DDPM [Ho et al., 2020]	9.46	3.21	0.57	1000	80.5
NCSN [Song and Ermon, 2019]	8.87	25.3	-	1000	107.9
Adversarial DSM [Jolicoeur-Martineau et al., 2021b]	-	6.10	-	1000	-
Likelihood SDE [Song et al., 2021a]	-	2.87	-	-	-
Score SDE (VE) [Song et al., 2021b]	9.89	2.20	0.59	2000	423.2
Score SDE (VP) [Song et al., 2021b]	9.68	2.41	0.59	2000	421.5
Probability Flow (VP) [Song et al., 2021b]	9.83	3.08	0.57	140	50.9
LSGM [Vahdat et al., 2021]	9.87	2.10	0.61	147	44.5
DDIM, T=50 [Song et al., 2020a]	8.78	4.67	0.53	50	4.01
FastDDPM, T=50 [Kong and Ping, 2021]	8.98	3.41	0.56	50	4.01
Recovery EBM [Gao et al., 2021]	8.30	9.58	-	180	-
Improved DDPM [Nichol and Dhariwal, 2021]	-	2.90	-	4000	-
VDM [Kingma et al., 2021]	-	4.00	-	1000	-
UDM [Kim et al., 2021]	10.1	2.33	-	2000	-
D3PMs [Austin et al., 2021]	8.56	7.34	-	1000	-
Gotta Go Fast [Jolicoeur-Martineau et al., 2021a]	-	2.44	-	180	-
DDPM Distillation [Luhman and Luhman, 2021]	8.36	9.36	0.51	1	-
SNGAN [Miyato et al., 2018]	8.22	21.7	0.44	1	-
SNGAN+DGflow [Ansari et al., 2021]	9.35	9.62	0.48	25	1.98
AutoGAN [Gong et al., 2019]	8.60	12.4	0.46	1	-
TransGAN [Jiang et al., 2021]	9.02	9.26	-	1	-
StyleGAN2 w/o ADA [Karras et al., 2020a]	9.18	8.32	0.41	1	0.04
StyleGAN2 w/ ADA [Karras et al., 2020a]	9.83	2.92	0.49	1	0.04
StyleGAN2 w/ Diffaug [Zhao et al., 2020]	9.40	5.79	0.42	1	0.04
Glow [Kingma and Dhariwal, 2018]	3.92	48.9	-	1	-
PixelCNN [Oord et al., 2016]	4.60	65.9	-	1024	-
NVAE [Vahdat and Kautz, 2020]	7.18	23.5	0.51	1	0.36
IGEBM [Du and Mordatch, 2019]	6.02	40.6	-	60	-
VAEBM [Xiao et al., 2021a]	8.43	12.2	0.53	16	8.79

randomly sampled from the diffusion step. The result, presented in the second group of Table 6.2, is significantly worse than our model, indicating that our model is not equivalent to augmenting data before applying the discriminator.

Parametrization for $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$: We study two alternative ways to parametrize the denoising distribution for the same $T = 4$ setting. Instead of letting the generator produce estimated samples of $\mathbf{x}_0$, we set the generator to directly output denoised samples $\mathbf{x}_{t-1}$ without posterior sampling (*direct denoising*), or output the noise ϵ_t that perturbs a clean image to produce $\mathbf{x}_t$ (*noise generation*). Note that the latter case is closely related to most diffusion models where the network deterministically predicts the perturbation noise. In

Table 6.2: Ablation studies on CIFAR-10.

Model Variants	IS↑	FID↓	Recall↑
T = 1	8.93	14.6	0.19
T = 2	**9.80**	4.08	0.54
T = 4	9.63	**3.75**	**0.57**
T = 8	9.43	4.36	0.56
One-shot w/ aug	8.96	13.2	0.25
Direct denoising	9.10	6.03	0.53
Noise generation	8.79	8.04	0.52
No latent variable	8.37	20.6	0.42

Table 6.2, we show that although these alternative parametrizations work reasonably well, our main parametrization outperforms them by a large margin.

Importance of latent variable: Removing latent variables $\mathbf{z}$ converts our denoising model to a unimodal distribution. In the last line of Table 6.2, we study our model's performance without any latent variables $\mathbf{z}$. We see that the sample quality is significantly worse, suggesting the importance of multimodal denoising distributions. In Figure 6.8, we visualize the effect of latent variables by showing samples of $p_\theta(\mathbf{x}_0|\mathbf{x}_1)$, where $\mathbf{x}_1$ is a fixed noisy observation. We see that while the majority of information in the conditioning $\mathbf{x}_1$ is preserved, the samples are diverse due to the latent variables.

6.4.3 Mode Coverage

Besides the recall score in Table 6.1, we also evaluate the mode coverage of our model on the popular 25-Gaussians and StackedMNIST. The 25-Gaussians dataset is a 2-D toy dataset, generated by a mixture of 25 two-dimensional Gaussian distributions, arranged in a grid. We train our denoising diffusion GAN with 4 denoising steps and compare it to other models in Figure 6.9. We observe that the vanilla GAN suffers severely from mode collapse, and while techniques like WGAN-GP [Gulrajani et al., 2017] improve mode coverage, the sample quality is still limited. In contrast, our model covers all the modes while maintaining high

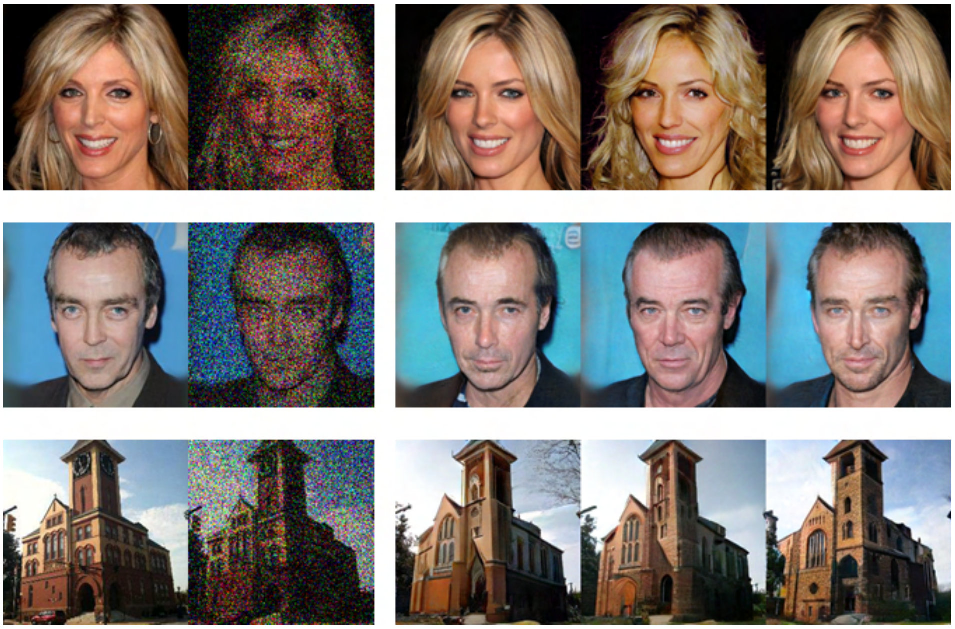


Figure 6.8: Multi-modality of denoising distribution given the same noisy observation. **Left:** clean image $\mathbf{x}_0$ and perturbed image $\mathbf{x}_1$. **Right:** Three samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_1)$.

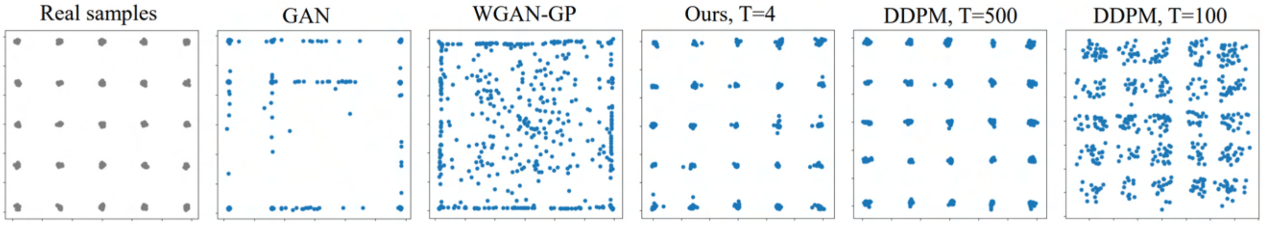


Figure 6.9: Qualitative results on the 25-Gaussians dataset.

sample quality. We also train a diffusion model and plot the samples generated by 100 and 500 denoising steps. We see that diffusion models require a large number of steps to maintain high sample quality.

StackMNIST contains images generated by randomly choosing 3 MNIST images and stacking them along the RGB channels. Hence, the data distribution has 1000 modes. Following the setting of Lin et al. [2018], we report the number of covered modes and the KL divergence from the categorical distribution over 1000 categories of generated samples to true data in Table 6.3. We observe that our model covers all modes faithfully and achieves the lowest KL compared to GANs that are specifically designed for better mode coverage or

Table 6.3: Mode coverage on StackedMNIST.

Model	Modes↑	KL↓
VEEGAN ([Srivastava et al., 2017])	762	2.173
PacGAN ([Lin et al., 2018])	992	0.277
PresGAN ([Dieng et al., 2019])	**1000**	0.115
InclusiveGAN ([Yu et al., 2020b])	997	0.200
StyleGAN2 ([Karras et al., 2020b])	940	0.424
Adv. DSM ([Jolicoeur-Martineau et al., 2021b])	**1000**	1.49
VAEBM ([Xiao et al., 2021a])	**1000**	0.087
Denoising Diffusion GAN (ours)	**1000**	**0.071**

StyleGAN2 which is known to have the best sample quality. Our model even has lower KL divergence than some likelihood models such as VAEBM, suggesting that our model captures the modes of training distribution faithfully.

6.4.4 Training Stability

In Fig. 6.10, we plot the discriminator loss for different time steps in the diffusion process when $T = 4$. We observe that the training of our denoising diffusion GAN is stable, and we do not see any explosion in loss values, as is sometimes reported for other GAN methods such as Brock et al. [2018]. The stability might be attributed to two reasons: First, the conditioning on $\mathbf{x}_t$ for both generator and discriminator provides a strong signal. The generator is required to generate a few plausible samples given $\mathbf{x}_t$, and the discriminator requires classifying them. The $\mathbf{x}_t$ conditioning keeps the discriminator and generator in a balance. Second, we are training the GAN on relatively smooth distributions, as the diffusion process is known as a smoothening process that brings the distributions of fake and real samples closer to each other [Lyu, 2012]. As we can see from Fig. 6.10, the discriminator loss for $t > 0$ is higher than $t = 0$ (the last denoising step). Note that $t > 0$ corresponds to training the discriminator on noisy images, and in this case, the true and generator distributions are closer to each other, making the discrimination harder and hence resulting in higher discriminator loss.

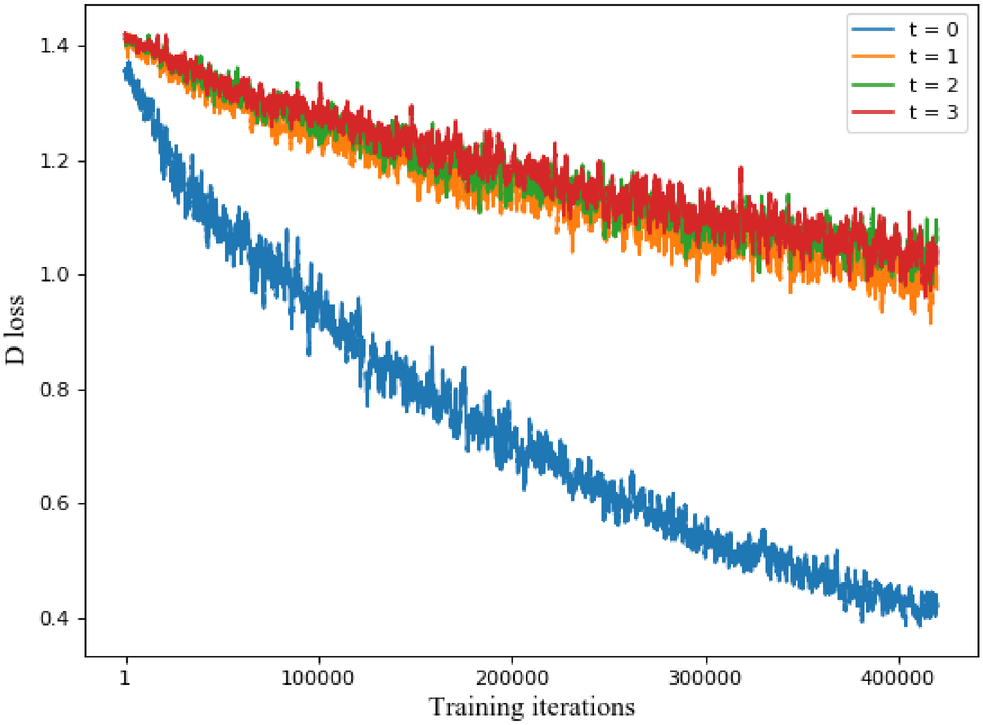


Figure 6.10: The discriminator loss per denoising step during training.

We believe that such a property prevents the discriminator from overfitting, which leads to better training stability.

6.4.5 High Resolution Image

We train our model on datasets with larger images, including CelebA-HQ [Karras et al., 2017] and LSUN Church [Yu et al., 2015] at 256×256px resolution. We report FID on these two datasets in Table 6.4 and 6.5. Similar to CIFAR-10, our model obtains competitive sample quality among the best diffusion models and GANs. In particular, in LSUN Church, our model outperforms DDPM and ImageBART. Although, some GANs perform better on this dataset, their mode coverage is not reflected by the FID score.

Qualitative samples of CelebA-HQ AND LSUN Church are presented in Figure 6.11 and

Table 6.4: Generative results on CelebA-HQ-256

Model	FID↓
Denoising Diffusion GAN (ours)	7.64
Score SDE [Song et al., 2021b]	7.23
LSGM [Vahdat et al., 2021]	7.22
UDM [Kim et al., 2021]	**7.16**
NVAE [Vahdat and Kautz, 2020]	29.7
VAEBM [Xiao et al., 2021a]	20.4
NCP-VAE [Aneja et al., 2021]	24.8
PGGAN [Karras et al., 2017]	8.03
Adv. LAE [Pidhorskyi et al., 2020]	19.2
VQ-GAN [Esser et al., 2021b]	10.2
DC-AE [Parmar et al., 2021]	15.8

Table 6.5: Generative results on LSUN Church 256

Model	FID↓
Denoising Diffusion GAN (ours)	5.25
DDPM [Ho et al., 2020]	7.89
ImageBART [Esser et al., 2021a]	7.32
Gotta Go Fast (Jolicoeur-Martineau et al.)	25.67
PGGAN [Karras et al., 2017])	6.42
StyleGAN [Karras et al., 2019]	4.21
StyleGAN2 [Karras et al., 2020b]	3.86
CIPS [Anokhin et al., 2021]	**2.92**

6.12 respectively.

6.4.6 Additional Results

Stroke-based Image Synthesis

Meng et al. [2021b] propose an interesting application of diffusion models to stroke-based generation. Specifically, they perturb a stroke painting by the forward diffusion process, and denoise it with a diffusion model. The method is particularly promising because it only requires training an unconditional generative model on the target dataset and does not require training images paired with stroke paintings like GAN-based methods [Sangkloy et al., 2017, Park et al., 2019]. We apply our model to stroke-based image synthesis and show qualitative results in Figure 6.13. The generated samples are realistic and diverse, while the conditioning in the stroke paintings is faithfully preserved. Compared to Meng et al. [2021b], our model enjoys a 1100× speedup in generation, as it takes only **0.16**s to generate one image at 256 resolution vs. **181**s for Meng et al. [2021b]. This experiment confirms that our proposed model enables the application of diffusion models to interactive applications such as image editing.

Additional Visualization for $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$

In Figure 6.14 and Figure 6.15, we show visualizations of samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ for different t. Note that except for $p_\theta(\mathbf{x}_0|\mathbf{x}_1)$, the samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ do not need to be sharp, as they are only intermediate outputs of the sampling process. The conditioning is less preserved as the perturbation in $\mathbf{x}_t$ increases, and in particular $\mathbf{x}_T$ ($\mathbf{x}_4$ in our example) contains almost no information of clean data $\mathbf{x}_0$.

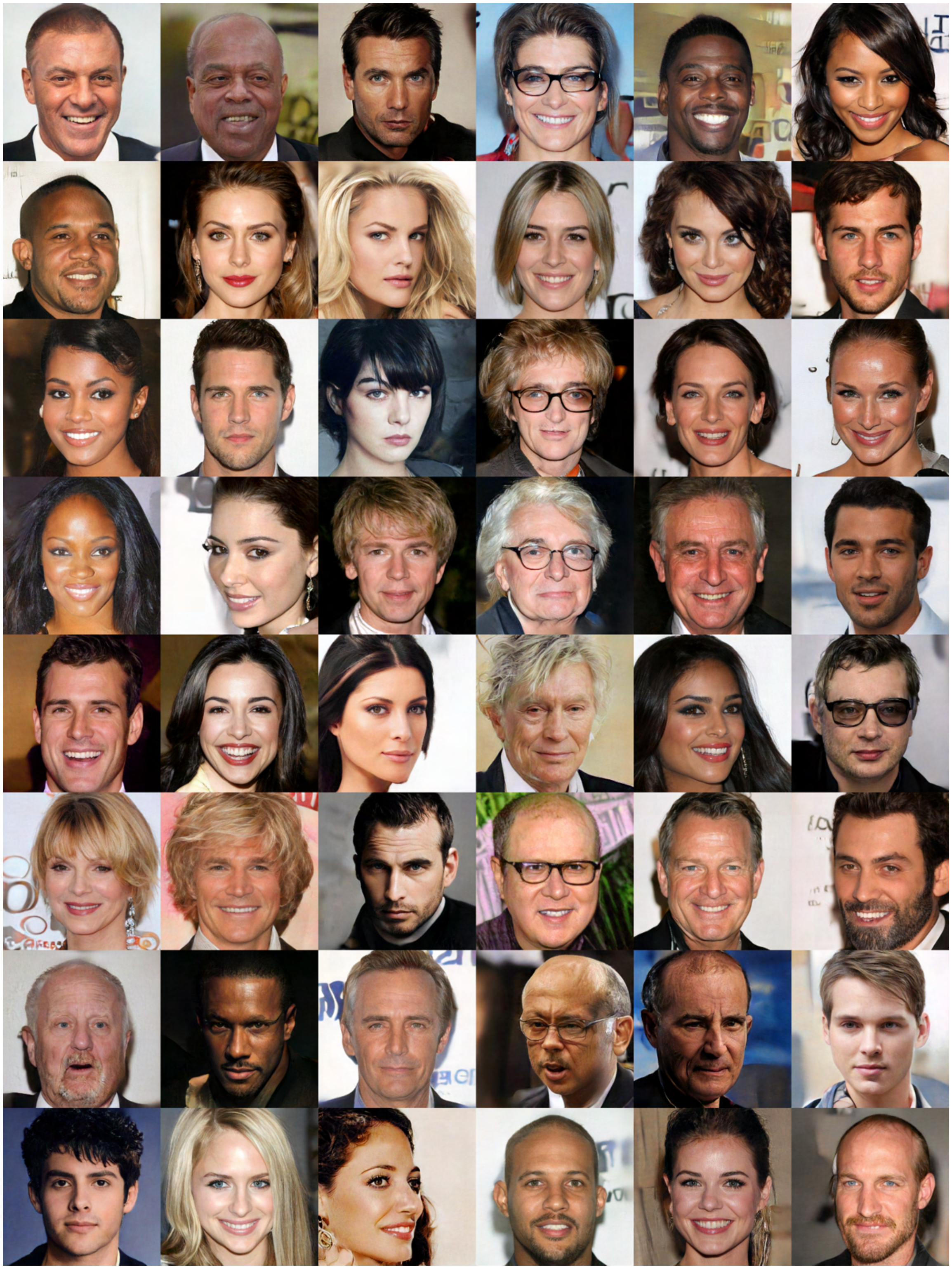


Figure 6.11: Qualitative results on CelebA-HQ of denoising diffusion GAN.

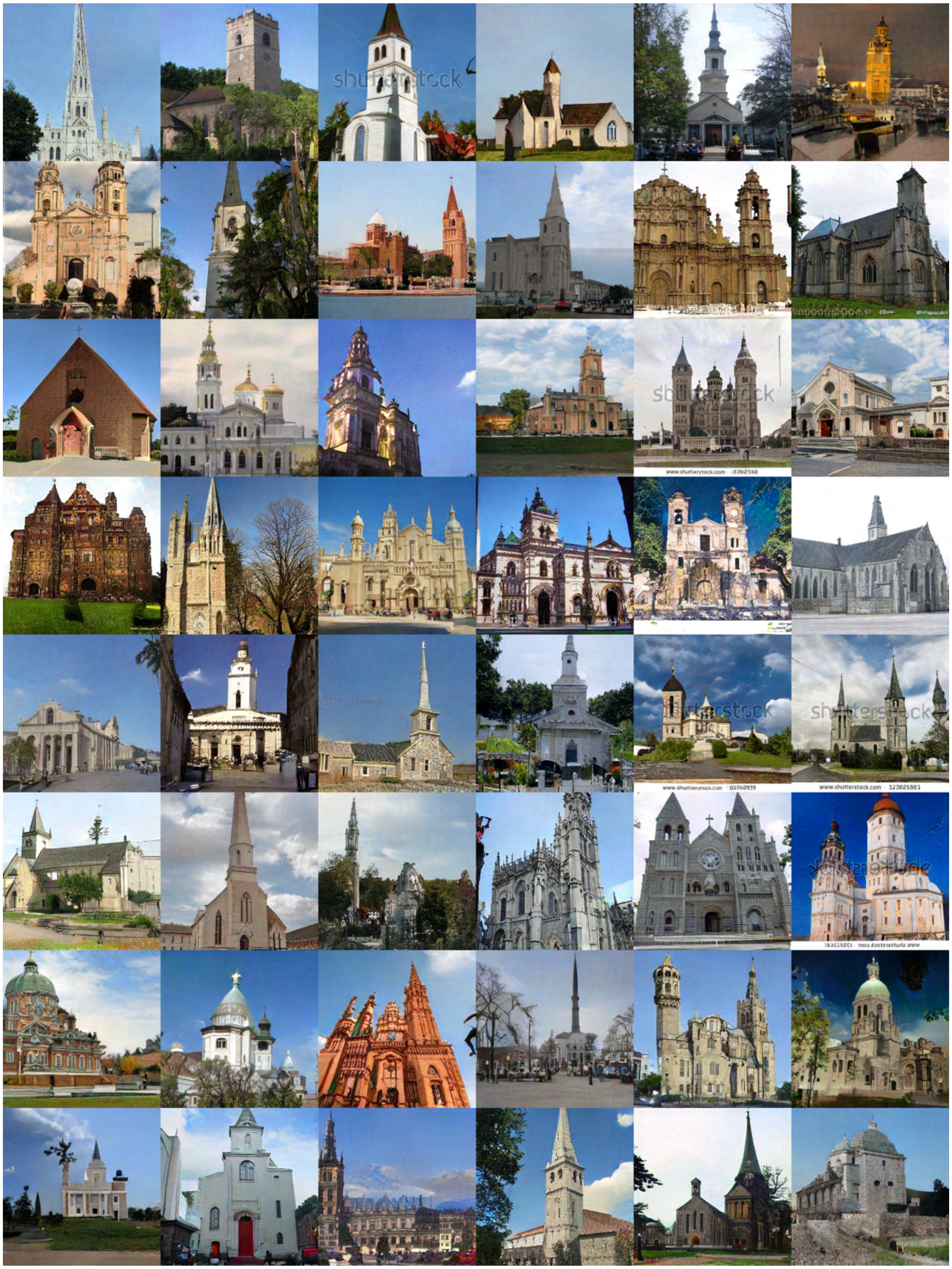


Figure 6.12: Qualitative results on LSUN Church of denoising diffusion GAN.

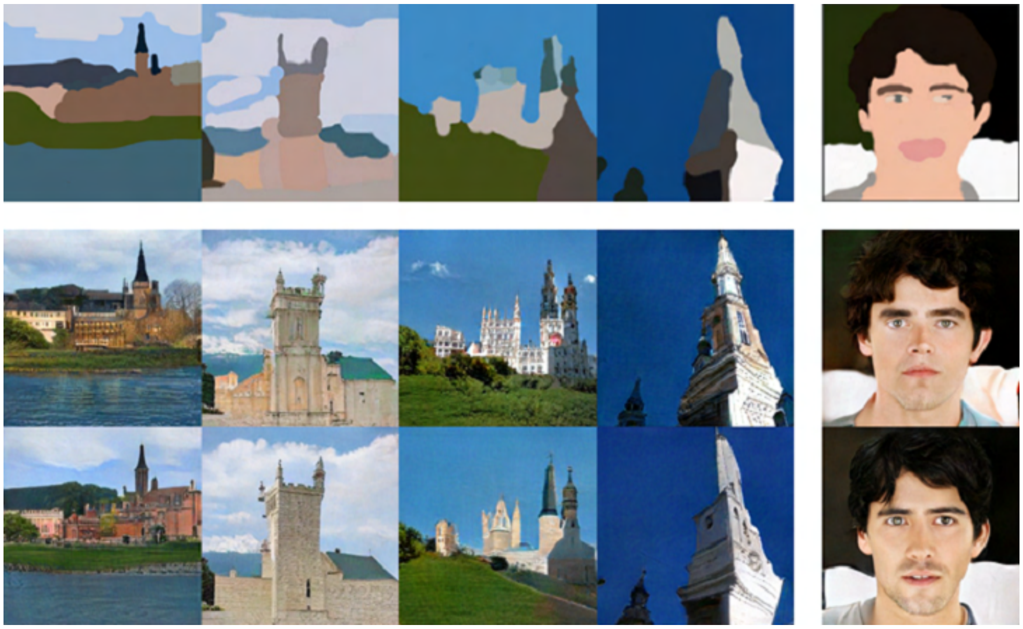


Figure 6.13: Qualitative results on stroke-based synthesis. **Top row:** stroke paintings. **Bottom two rows:** generated samples corresponding to the stroke painting.

6.4.7 Experimental Details

In this section, we present our experimental settings in detail.

Network Structure

Generator: Our generator structure largely follows the U-net structure [Ronneberger et al., 2015] used in NCSN++ [Song et al., 2021b], which consists of multiple ResNet blocks [He et al., 2016] and Attention blocks [Vaswani et al., 2017]. Hyper-parameters for the network design, such as the number of blocks and number of channels, are reported in Table 6.6. We follow the default settings in Song et al. [2021b] for other network configurations not mentioned in the table, including Swish activation function, upsampling and downsampling with anti-aliasing based on Finite Impulse Response (FIR) [Zhang, 2019], re-scaling all skip connections by $\frac{1}{\sqrt{2}}$, using residual block design from BigGAN [Brock et al., 2018] and incorporating progressive growing architectures [Karras et al., 2020b]. See Appendix H of Song et al. [2021b] for more details on these configurations.

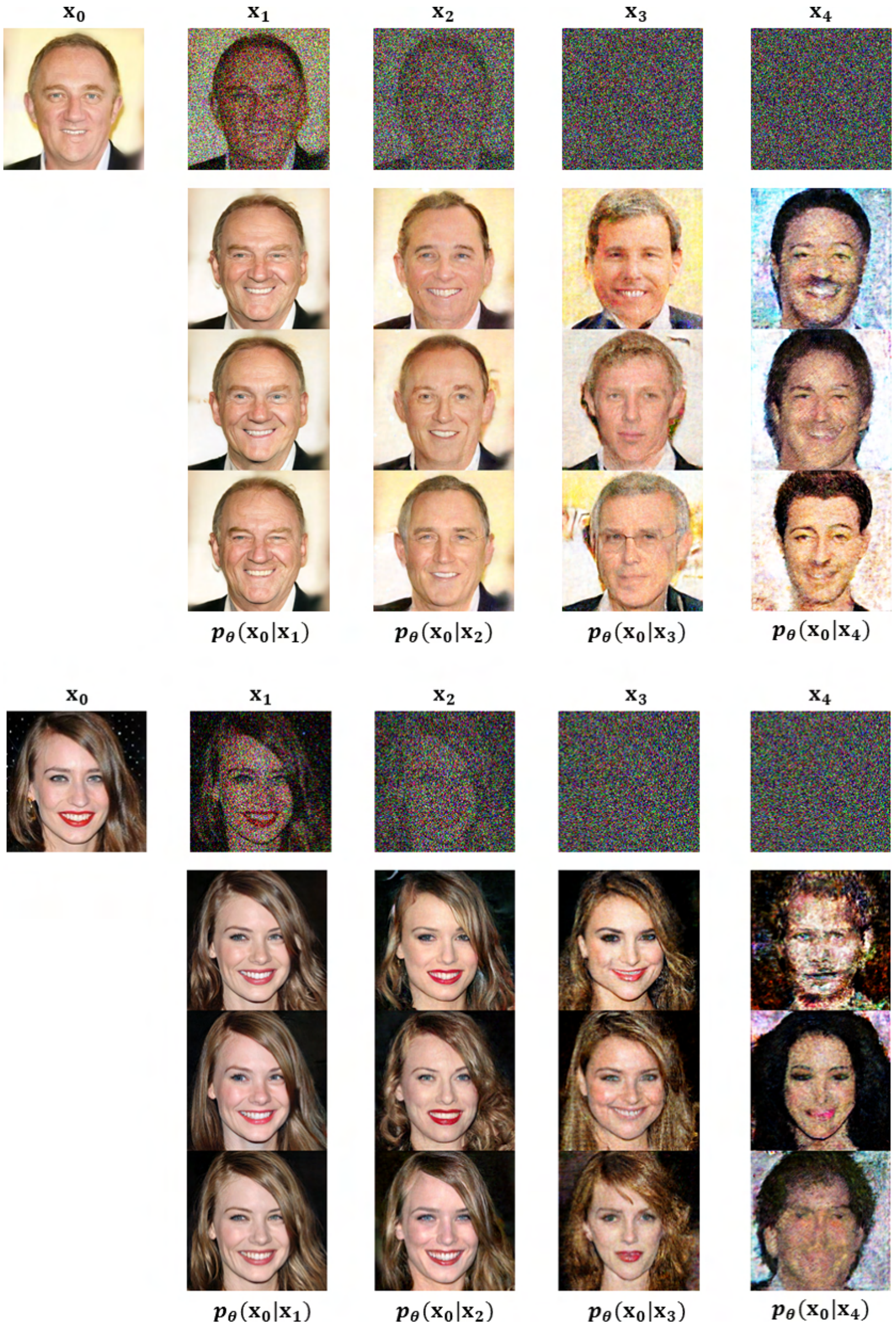

$\mathbf{x_0}$
$\mathbf{x_1}$
$\mathbf{x_2}$
$\mathbf{x_3}$
$\mathbf{x_4}$
$p_\theta(\mathbf{x_0}|\mathbf{x_1})$
$p_\theta(\mathbf{x_0}|\mathbf{x_2})$
$p_\theta(\mathbf{x_0}|\mathbf{x_3})$
$p_\theta(\mathbf{x_0}|\mathbf{x_4})$
$\mathbf{x_0}$
$\mathbf{x_1}$
$\mathbf{x_2}$
$\mathbf{x_3}$
$\mathbf{x_4}$
$p_\theta(\mathbf{x_0}|\mathbf{x_1})$
$p_\theta(\mathbf{x_0}|\mathbf{x_2})$
$p_\theta(\mathbf{x_0}|\mathbf{x_3})$
$p_\theta(\mathbf{x_0}|\mathbf{x_4})$

Figure 6.14: Visualization of samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ for different t on CelebAHQ. For each example, the top row contains $\mathbf{x}_t$ from diffusion process steps, where $\mathbf{x}_0$ is a sample from the dataset. The bottom rows contain 3 samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ for different t's.

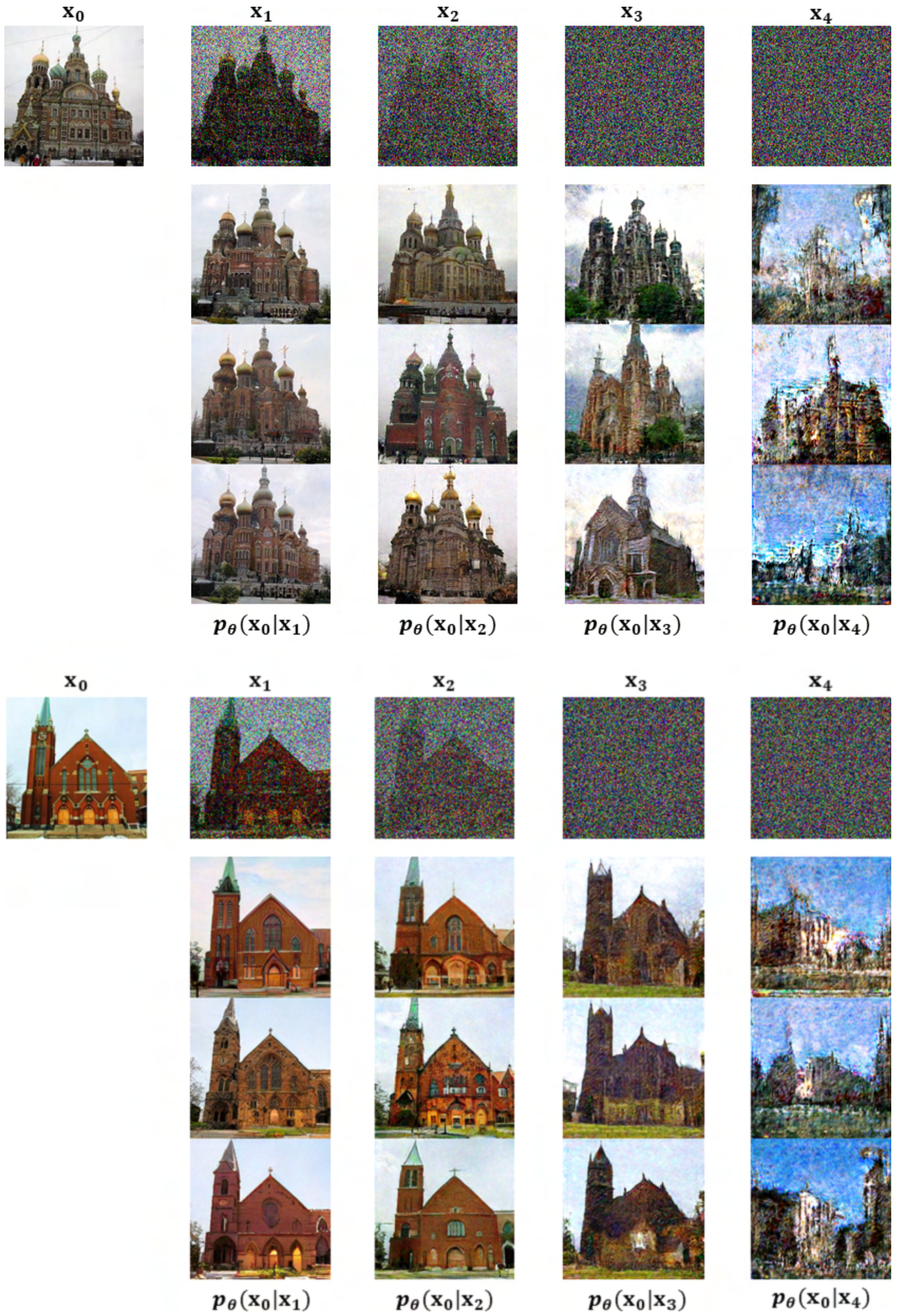

$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$p_\theta(\mathbf{x}_0|\mathbf{x}_1)$
$p_\theta(\mathbf{x}_0|\mathbf{x}_2)$
$p_\theta(\mathbf{x}_0|\mathbf{x}_3)$
$p_\theta(\mathbf{x}_0|\mathbf{x}_4)$
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$p_\theta(\mathbf{x}_0|\mathbf{x}_1)$
$p_\theta(\mathbf{x}_0|\mathbf{x}_2)$
$p_\theta(\mathbf{x}_0|\mathbf{x}_3)$
$p_\theta(\mathbf{x}_0|\mathbf{x}_4)$

Figure 6.15: Visualization of samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ for different t on LSUN Church. For each example, the top row contains $\mathbf{x}_t$ from diffusion process steps, where $\mathbf{x}_0$ is a sample from the dataset. The bottom rows contain 3 samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ for different t's.

Table 6.6: Hyper-parameters for the generator network.

	CIFAR10	CelebaHQ	LSUN Church
# of ResNet blocks per scale	2	2	2
Initial # of channels	128	64	128
Channel multiplier for each scale	$(1, 2, 2, 2)$	$(1, 1, 2, 2, 4, 4)$	$(1, 1, 2, 2, 4, 4)$
Scale of attention block	$(16,)$	$(16,)$	$(16,)$
Latent Dimension	256	100	100
# of latent mapping layers	3	3	3
Latent embedding dimension	512	256	256

We follow Ho et al. [2020] and use sinusoidal positional embeddings for conditioning on integer time steps. The dimension for the time embedding is 4× the number of initial channels presented in Table 6.6.

The fundamental difference between our generator network and the networks of previous diffusion models is that our generator takes an extra latent variable $\mathbf{z}$ as input. We use $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for all experiments. We replace all the group normalization (GN) layers in the network with adaptive group normalization (AdaGN) layers to allow the input of latent variables. The latent variable $\mathbf{z}$ is first transformed by a fully-connected network (called mapping network), and then the resulting embedding vector, denoted by $\mathbf{w}$, is sent to every AdaGN layer. Each AdaGN layer contains one fully-connected layer that takes $\mathbf{w}$ as input, and outputs the per-channel shift and scale parameters for the group normalization. The network's feature maps are then subject to affine transformations using these shift and scale parameters of the AdaGN layers. The mapping network and the fully-connected layer in AdaGN are independent of time steps t, as we found no extra benefit in incorporating time embeddings in these layers. Details about latent variables are also presented in Table 6.6.

Discriminator: We design our time-dependent discriminator with a convolutional network with ResNet blocks, where the design of the ResNet blocks is similar to that of the generator. The discriminator tries to discriminate real and fake $\mathbf{x}_{t-1}$, conditioned on $\mathbf{x}_t$ and t. The time conditioning is enforced by the same sinusoidal positional embedding as in the generator. The $\mathbf{x}_t$ conditioning is enforced by concatenating $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ as the input to the discriminator. We use LeakyReLU activations with a negative slope 0.2 for all layers. Similar to Karras et al. [2020b], we use a minibatch standard deviation layer after all the ResNet blocks. We present the exact architecture of discriminators in Table 6.7.

Table 6.7: Network structures for the discriminator.

CIFAR-10
1 × 1 conv2d, 128
ResBlock, 128
ResBlock down, 256
ResBlock down, 512
ResBlock down, 512
minibatch std layer
Global Sum Pooling
FC layer → scalar

CelebAHQ and LSUN Church
1 × 1 conv2d, 128
ResBlock down, 256
ResBlock down, 512
ResBlock down, 512
ResBlock down, 512
ResBlock down, 512
ResBlock down, 512
minibatch std layer
Global Sum Pooling
FC layer → scalar

Training

Objective: We train our denoising diffusion GAN with the following adversarial objective:

$$\min_{\phi} \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t)} \Big[\mathbb{E}_{q(\mathbf{x}_{t-1}|\mathbf{x}_t)} [-\log(D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t)] + \mathbb{E}_{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} [-\log(1 - D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))] \Big]$$

$$\max_{\theta} \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t)} \mathbb{E}_{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} [\log(D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t))]$$

Similar to Ho et al. [2020], during training we randomly sample an integer time step $t \in [1, 2, 3, 4]$ for each datapoint in a batch. Besides the main objective, we also add an R_1 regularization term [Mescheder et al., 2018] to the objective for the discriminator. The R_1

Table 6.8: Optimization hyper-parameters.

	CIFAR10	CelebaHQ	LSUN Church
Initial learning rate for discriminator	10^{-4}	10^{-4}	10^{-4}
Initial learning rate for generator	1.6×10^{-4}	1.6×10^{-4}	2×10^{-4}
Adam optimizer β_1	0.5	0.5	0.5
Adam optimizer β_2	0.9	0.9	0.9
EMA	0.9999	0.999	0.999
Batch size	128	32	64
# of training iterations	400k	750k	600k
# of GPUs	4	8	8

term is defined as

$$R_1(\phi) = \frac{\gamma}{2}\mathbb{E}_{q(\mathbf{x}_t)q(\mathbf{x}_{t-1}|\mathbf{x}_t)}\left[\left\|\nabla_{\mathbf{x}_{t-1}} D_\phi(\mathbf{x}_{t-1}, \mathbf{x}_t, t)\right\|^2\right], \tag{6.16}$$

where γ is the coefficient for the regularization. We use $\gamma = 0.05$ for CIFAR-10, and $\gamma = 1$ for CelebAHQ and LSUN Church.

Optimization: We train our models using the Adam optimizer [Kingma and Ba, 2015]. We use cosine learning rate decay [Loshchilov and Hutter, 2016] for training both the generator and discriminator. Similar to Ho et al. [2020], Song et al. [2021b], Karras et al. [2020a], we observe that applying an exponential moving average (EMA) on the generator is crucial to achieve high performance. We summarize the optimization hyper-parameters in Table 6.8.

We train our models on CIFAR-10 using 4 V100 GPUs. On CelebAHQ and LSUN Church we use 8 V100 GPUs. The training takes approximately 48 hours on CIFAR-10, and 180 hours on CelebAHQ and LSUN Church.

Evaluation

When evaluating IS, FID and recall score, we use 50k generated samples for CIFAR-10 and LSUN Church, and 30k samples for CelebAHQ (since the CelebA HQ dataset contains only

30k samples).

When evaluating sampling time, we use models trained on CIFAR-10 and generate a batch of 100 samples. We benchmark the sampling time on a machine with a single V100 GPU. We use Pytorch 1.9.0 and CUDA 11.0.

Ablation Studies

Here we introduce the settings for the ablation study in Section 6.4.2. We observe that training requires a larger number of training iterations when T is larger. As a result, we train the model for each T until the FID score does not increase any further. The number of training iteration is 200k for $T = 1$ and $T = 2$, 400k for $T = 4$ and 600k for $T = 8$. We use the same network structures and optimization settings as in the main experiments.

For the data augmentation baseline, we follow the differentiable data augmentation pipeline in Zhao et al. [2020]. In particular, for every (real or fake) image in the batch, we perturbed it by sampling from a random timestep at the diffusion process (except the last diffusion step where the information of data is completely destroyed). We find the results insensitive to the number of possible perturbation levels (i.e, the number of steps in the diffusion process), and we report the result using a diffusion process with 4 steps. Since the perturbation by the diffusion process is differentiable, we can train both the discriminator and generator with the perturbed samples. See Zhao et al. [2020] for a detailed explanation for the training pipeline.

For the experiments on alternative parametrizations, we use $T = 4$ for the diffusion process and keep other settings the same as in the main experiments.

For the experiment on training a model without latent variables, similar to the main experiments, the generator takes the conditioning $\mathbf{x}_t$ as its input, and the time conditioning is still enforced by the time embedding. However, the AdaGN layers are replaced by plain GN layers, such that no latent variable is needed, and the mapping network for $\mathbf{z}$ is removed.

Other settings follow the main experiments.

Toy data and StackedMNIST

For the 25-Gaussian toy dataset, both our generator and discriminator have 3 fully-connected layers each with 512 hidden units and LeakyReLU activations (negative slope of 0.2). We enforce both the conditioning on $\mathbf{x}_t$ and t by concatenation with the input. We use the Adam optimizer with a learning rate of 10^{-4} for both the generator and discriminator. The batch size is 512, and we train the model for 50k iterations.

Our experimental settings for StackedMNIST are the same as those for CIFAR-10, except that we train the model for only 150k iterations.

6.5 Conclusion

Deep generative learning frameworks still struggle with addressing the generative learning trilemma. Diffusion models achieve exceptionally high-quality and diverse sampling. However, their slow sampling and high computational cost do not yet allow them to be widely applied in real-world applications. In this paper, we argued that one of the main sources of slow sampling in diffusion models is the Gaussian assumption in the denoising distribution, which is justified only for very small denoising steps. To remedy this, we proposed denoising diffusion GANs that model each denoising step using a complex multimodal distribution, enabling us to take large denoising steps. In extensive experiments, we showed that denoising diffusion GANs achieve high sample quality and diversity competitive to the original diffusion models while being orders of magnitude faster at sampling. Compared to traditional GANs, our proposed model enjoys better mode coverage and sample diversity. Our denoising diffusion GAN overcomes the generative learning trilemma to a large extent, allowing diffusion models to be applied to real-world problems with low computational cost.

Denoising diffusion GAN successfully tackles the generative learning trilemma. Our model achieves

1. Faster sampling, due to the multimodal complex denoising distribution parametrized by conditional GAN;
2. Better mode coverage, due to simple conditional generation problem at each step
3. High-quality samples, due to adversarial training

Our model is a symbiotic composition of denoising diffusion models and GANs. On one hand, conditional GAN enables the denoising distribution to be multi-modal and expressive, allowing larger denoising steps and significantly faster sampling. On the other hand, the denoising diffusion model breaks the generation into several easier conditional tasks, where each task is defined on smoothened data perturbed by the noise so that the training is stabilized and the mode collapse issue is greatly alleviated.

CHAPTER 7
CONCLUSION

7.1 Summary

The dissertation follows the journey toward pushing the limits of deep generative models. After a high-level introduction of generative learning in Chapter 1, we dive into existing deep generative models that are fundamental in Chapter 2. In particular, we review Variational Auto-encoders, Normalizing Flows, Energy-based Models, Denoising Diffusion Models, and Generative Adversarial Networks. We list and carefully analyze their pros and cons. The analysis of existing generative models provides the motivation for later chapters, where we propose new models with the idea of symbiotic composition, which is to combine two existing models together with the hope that the new model will enjoy the best of both worlds.

In Chapter 3, we propose Generative Latent Flow, which is a combination of auto-encoders and normalizing flows. The auto-encoder learns to reconstruct data with low-dimensional latent variables, and the normalizing flow learns to map the latent variables to noise and vice versa. The resulting model shows superior generative performance over previous auto-encoder based models due to the expressive prior distribution modeled by the normalizing flow, and the auto-encoder makes the training of the normalizing flow easier by mapping the data to a lower-dimensional space.

In Chapter 4, we introduce the idea of exponential tilting with EBMs. We propose to use a base generative model, such as a VAE or a normalizing flow, to capture the shape of the data distribution roughly and later introduce a EBM to refine the obtained distribution. The base generative model makes the training of EBM much more efficient by providing a good starting point as well as a smooth latent space that allows easier MCMC sampling. The EBM refines the density and significantly improves the sample quality by reducing the density mismatch between the base model and the true data distribution.

In Chapter 5, we investigate the role of Langevin dynamics in the maximum likelihood training of EBMs. We treat the Langevin dynamics as an implicit generator model by removing the noise term and further introduce the generator loss of WGAN to optimize the implicit generator. Such a combination of EBMs and GANs improves both the sample quality and training stability of EBMs.

In Chapter 6, we tackle the slow sampling issue of denoising diffusion models. We attribute the slow sampling issue to the Gaussian assumption of the reverse process and propose to model the single-step denoising distribution with conditional GANs. The resulting denoising diffusion GAN model obtains competitive sample quality with denoising diffusion models while enjoying 1000× speed-up in sampling. In addition, the denoising diffusion framework also stabilizes the training of GANs and alleviates the mode collapse issue due to the fact that the GANs are trained with conditional generation tasks on smoothed data. We hope that our findings in this dissertation may serve as a minor contribution to the development of generative learning, and motivate follow up work that further pushes the limits of deep generative models.

7.2 Future Work

In future work, we wish to continue the journey in the field of deep generative models. Specifically, it is our hope to design stronger generative models as well as extend the models to different domains. For example, after the publication of denoising diffusion GAN, the model has been applied to the task of text-to-speech synthesis by Liu et al. [2022]. We believe that all methods proposed in this dissertation have the potential to be used in domains other than image, such as video, sequence, graph and 3D point cloud.

Generative modeling is one sub-topic of the wider concept of unsupervised learning. Recently, we have seen tremendous progress in the field of unsupervised learning that tries to learn useful representations without labels and close the gap to supervised models. In the image domain, the field of unsupervised learning is currently dominated by contrastive pre-training [Chen et al., 2020b, He et al., 2020]. However, recently generative pretraining has also shown promising results [He et al., 2021]. There are other possibilities of extending generative models to the wider context of unsupervised learning and applying them to downstream tasks.

One additional topic of generative models that has recently become popular is multi-modality generation. For example, one interesting task is to generate images conditioned on given text [Ramesh et al., 2021, Nichol et al., 2021]. This is a promising direction with many exciting real-world applications. We would like to explore more possibilities in this direction.

REFERENCES

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

Guillaume Alain, Yoshua Bengio, Li Yao, Jason Yosinski, Eric Thibodeau-Laufer, Saizheng Zhang, and Pascal Vincent. Gsns: generative stochastic networks. *Information and Inference: A Journal of the IMA*, 5(2):210–249, 2016.

Yali Amit, Ulf Grenander, and Mauro Piccioni. Structural image restoration through deformable templates. *Journal of the American Statistical Association*, 86(414):376–387, 1991.

Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

Jyoti Aneja, Alex Schwing, Jan Kautz, and Arash Vahdat. A contrastive learning approach for training variational autoencoder priors. *Advances in Neural Information Processing Systems*, 34, 2021.

Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. Image generators with conditionally-independent pixel synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14278–14287, 2021.

Abdul Fatir Ansari, Ming Liang Ang, and Harold Soh. Refining deep generative models via discriminator gradient flow. In *International Conference on Learning Representations*, 2021.

Michael Arbel, L. Zhou, and A. Gretton. Generalized energy based models. *arXiv preprint arXiv:2003.05033*, 2020.

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

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

Jacob Austin, Daniel Johnson, Jonathan Ho, Danny Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *arXiv preprint arXiv:2107.03006*, 2021.

Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

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

Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019.

Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017.

Florian Bordes, Sina Honari, and Pascal Vincent. Learning to generate samples from noise through infusion training. *arXiv preprint arXiv:1703.06975*, 2017.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

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

Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *European Conference on Computer Vision*, pages 364–381. Springer, 2020.

Miguel A Carreira-Perpinan and Geoffrey Hinton. On contrastive divergence learning. In *International workshop on artificial intelligence and statistics*, pages 33–40. PMLR, 2005.

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.

Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. *arXiv preprint arXiv:2009.00713*, 2020a.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018a.

Ricky TQ Chen, Jens Behrmann, David K Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*, 32, 2019.

Scott Chen and Ramesh Gopinath. Gaussianization. *Advances in neural information processing systems*, 13, 2000.

Tian Qi Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 2610–2620, 2018b.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020b.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016.

Rewon Child. Very deep vaes generalize autoregressive models and can outperform them on images. In *International Conference on Learning Representations*, 2021.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.

Bin Dai and David Wipf. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations*, 2019.

Constantinos Daskalakis and Ioannis Panageas. The limit points of (optimistic) gradient descent in min-max optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Zhiwei Deng, Megha Nawhal, Lili Meng, and Greg Mori. Continuous graph flow. *arXiv preprint arXiv:1908.02436*, 2019.

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

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34, 2021.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Adji B Dieng, Francisco JR Ruiz, David M Blei, and Michalis K Titsias. Prescribed generative adversarial networks. *arXiv preprint arXiv:1910.04302*, 2019.

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.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

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

Yilun Du, Shuang Li, and Igor Mordatch. Compositional visual generation with energy based models. *Advances in Neural Information Processing Systems*, 33:6637–6647, 2020.

Albert Einstein. Über die von der molekularkinetischen theorie der wärme geforderte bewegung von in ruhenden flüssigkeiten suspendierten teilchen. *Annalen der physik*, 4, 1905.

Patrick Esser, Robin Rombach, Andreas Blattmann, and Björn Ommer. Imagebart: Bidirectional context with multinomial diffusion for autoregressive image synthesis. *arXiv preprint arXiv:2108.08827*, 2021a.

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12873–12883, 2021b.

Farzan Farnia and Asuman Ozdaglar. Do gans always have nash equilibria? In *International Conference on Machine Learning*, pages 3029–3039. PMLR, 2020.

William Feller. On the theory of stochastic processes, with particular reference to applications. In *Proceedings of the [First] Berkeley Symposium on Mathematical Statistics and Probability*, pages 403–432. University of California Press, 1949.

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.

Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, pages 721–741, 1984.

Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

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

Pedro J Gonçalves, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. Training deep neural density estimators to identify mechanistic models of neural dynamics. *Elife*, 9:e56261, 2020.

Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3224–3234, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Anirudh Goyal, Nan Rosemary Ke, Surya Ganguli, and Yoshua Bengio. Variational walkback: Learning a transition operator as a stochastic recurrent net. *arXiv preprint arXiv:1711.02282*, 2017.

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, Joern-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *International Conference on Learning Representations*, 2020.

Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19, 2006.

Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

Michael U. Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(11):307–361, 2012.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, 2018.

Tian Han, Erik Nijkamp, Xiaolin Fang, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. Divergence triangle for joint training of generator model, energy-based model, and inferential model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8670–8679, 2019.

Tian Han, Erik Nijkamp, Linqi Zhou, Bo Pang, Song-Chun Zhu, and Ying Nian Wu. Joint training of variational auto-encoder and latent energy-based model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7978–7987, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

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.

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

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. *ICLR*, 2016.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

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

Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *arXiv preprint arXiv:2106.15282*, 2021.

Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23(47):1–33, 2022.

Matthew Hoffman, Pavel Sountsov, Joshua V Dillon, Ian Langmore, Dustin Tran, and Srinivas Vasudevan. Neutra-lizing bad geometry in hamiltonian monte carlo using neural transport. *arXiv preprint arXiv:1903.03704*, 2019.

Matthew D Hoffman and Matthew J Johnson. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, 2016.

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/CVF Conference on Computer Vision and Pattern Recognition*, pages 5811–5819, 2019.

Chin-Wei Huang, Ahmed Touati, Laurent Dinh, Michal Drozdzal, Mohammad Havaei, Laurent Charlin, and Aaron Courville. Learnable explicit density for continuous latent space and variational inference. *arXiv preprint arXiv:1710.02248*, 2017.

Chin-Wei Huang, Ricky T. Q. Chen, Christos Tsirigotis, and Aaron Courville. Convex potential flows: Universal probability distributions with optimal transport and convex optimization. In *International Conference on Learning Representations*, 2021a.

Chin-Wei Huang, Jae Hyun Lim, and Aaron Courville. A variational perspective on diffusion-based generative models and score matching. *arXiv preprint arXiv:2106.02808*, 2021b.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017.

Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. Variational autoencoder with arbitrary conditioning. In *International Conference on Learning Representations*, 2019.

Ajil Jalal, Marius Arvinte, Giannis Daras, Eric Price, Alex Dimakis, and Jonathan Tamir. Robust compressed sensing MRI with deep generative priors. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

Christopher Jarzynski. Equilibrium free-energy differences from nonequilibrium measurements: A master-equation approach. *Physical Review E*, 56(5):5018, 1997.

Tony Jebara. *Machine learning: discriminative and generative*, volume 755. Springer Science & Business Media, 2012.

Yifan Jiang, Shiyu Chang, and Zhangyang Wang. Transgan: Two transformers can make one strong gan. *arXiv preprint arXiv:2102.07074*, 2021.

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.

Richard M Johnson. The minimal transformation to orthonormality. *Psychometrika*, 31(1): 61–66, 1966.

Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. *arXiv preprint arXiv:2105.14080*, 2021a.

Alexia Jolicoeur-Martineau, Rémi Piché-Taillefer, Ioannis Mitliagkas, and Remi Tachet des Combes. Adversarial score matching and improved sampling for image generation. In *International Conference on Learning Representations*, 2021b.

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, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. *arXiv preprint arXiv:2006.06676*, 2020a.

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, 2020b.

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.

Dongjun Kim, Seungjae Shin, Kyungwoo Song, Wanmo Kang, and Il-Chul Moon. Score matching model for unbounded data score. *arXiv preprint arXiv:2106.05527*, 2021.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

Sungwon Kim, Sang-gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. Flowavenet: A generative flow for raw audio. *arXiv preprint arXiv:1811.02155*, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

Diederik P Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *arXiv preprint arXiv:2107.00630*, 2021.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. *Advances in neural information processing systems*, 2014.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 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.

Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems*, 33:17022–17033, 2020.

Zhifeng Kong and Kamalika Chaudhuri. The expressive power of a class of normalizing flow models. In *International Conference on Artificial Intelligence and Statistics*, pages 3599–3609. PMLR, 2020.

Zhifeng Kong and Wei Ping. On fast sampling of diffusion probabilistic models. *arXiv preprint arXiv:2106.00132*, 2021.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations*, 2021.

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

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.

Karol Kurach, Mario Lučić, Xiaohua Zhai, Marcin Michalski, and Sylvain Gelly. A large-scale study on regularization and normalization in gans. In *International Conference on Machine Learning*, pages 3581–3590. PMLR, 2019.

Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *arXiv preprint arXiv:1904.06991*, 2019.

Y. Lecun. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 2010.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

Don S Lemons and Anthony Gythiel. Paul langevin's 1908 paper "on the theory of brownian motion"["sur la théorie du mouvement brownien," cr acad. sci.(paris) 146, 530–533 (1908)]. *American Journal of Physics*, 65(11):1079–1081, 1997.

Ke Li and Jitendra Malik. Implicit maximum likelihood estimation. *arXiv preprint arXiv:1809.09087*, 2018.

Yang Li, Shoaib Akbar, and Junier Oliva. Acflow: Flow models for arbitrary conditional likelihoods. In *International Conference on Machine Learning*, pages 5831–5841. PMLR, 2020.

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

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*, 2019b.

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*, 2018.

Chieh Hubert Lin, Chia-Che Chang, Yu-Sheng Chen, Da-Cheng Juan, Wei Wei, and Hwann-Tzong Chen. Coco-gan: generation by parts via conditional coordinating. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4512–4521, 2019.

Tianyi Lin, Chi Jin, and Michael Jordan. On gradient descent ascent for nonconvex-concave minimax problems. In *International Conference on Machine Learning*, pages 6083–6093. PMLR, 2020.

Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. In *Advances in neural information processing systems*, pages 1498–1507, 2018.

Hao Liu and Pieter Abbeel. Hybrid discriminative-generative training via contrastive learning. *arXiv preprint arXiv:2007.09070*, 2020.

Songxiang Liu, Dan Su, and Dong Yu. Diffgan-tts: High-fidelity and efficient text-to-speech with denoising diffusion gans. *arXiv preprint arXiv:2201.11972*, 2022.

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.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. *Advances in neural information processing systems*, 31, 2018.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. *arXiv preprint arXiv:2201.09865*, 2022.

Eric Luhman and Troy Luhman. Knowledge distillation in iterative generative models for improved sampling speed. *arXiv preprint arXiv:2101.02388*, 2021.

Siwei Lyu. Interpretation and generalization of score matching. *arXiv preprint arXiv:1205.2629*, 2012.

Lars Maaløe, Marco Fraccaro, Valentin Liévin, and Ole Winther. Biva: A very deep hierarchy of latent variables for generative modeling. *Advances in neural information processing systems*, 32, 2019.

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

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

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

Łukasz Maziarka, Agnieszka Pocha, Jan Kaczmarczyk, Krzysztof Rataj, Tomasz Danel, and Michał Warchoł. Mol-cyclegan: a generative model for molecular optimization. *Journal of Cheminformatics*, 12(1):1–18, 2020.

Chenlin Meng, Jiaming Song, Yang Song, Shengjia Zhao, and Stefano Ermon. Improved autoregressive modeling with distribution smoothing. *arXiv preprint arXiv:2103.15089*, 2021a.

Chenlin Meng, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021b.

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Gautam Mittal, Jesse Engel, Curtis Hawthorne, and Ian Simon. Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091*, 2021.

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

Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Transactions on Graphics (TOG)*, 38(5):1–19, 2019.

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.

Radford M Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto Toronto, ON, Canada, 1993.

Radford M Neal. Annealed importance sampling. *Statistics and computing*, 11(2):125–139, 2001.

Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

Andrew Ng and Michael Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14, 2001.

Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

Erik Nijkamp, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. Learning non-convergent non-persistent short-run mcmc toward energy-based model. *Advances in Neural Information Processing Systems*, 32, 2019.

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, 2020.

Erik Nijkamp, Ruiqi Gao, Pavel Sountsov, Srinivas Vasudevan, Bo Pang, Song-Chun Zhu, and Ying Nian Wu. MCMC should mix: Learning energy-based model with flow-based backbone. In *International Conference on Learning Representations*, 2022.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 271–279, 2016.

Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. In *International conference on machine learning*, pages 3918–3926. PMLR, 2018.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *ICML*, 2016.

Georg Ostrovski, Will Dabney, and Rémi Munos. Autoregressive quantile networks for generative modeling. *arXiv preprint arXiv:1806.05575*, 2018.

Bo Pang, Tian Han, Erik Nijkamp, Song-Chun Zhu, and Ying Nian Wu. Learning latent space energy-based prior model. *arXiv preprint arXiv:2006.08205*, 2020.

George Papamakarios. Neural density estimation and likelihood-free inference. *arXiv preprint arXiv:1910.13233*, 2019.

Kancharla Parimala and Sumohana Channappayya. Quality aware generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 2948–2958, 2019.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019.

Gaurav Parmar, Dacheng Li, Kwonjoon Lee, and Zhuowen Tu. Dual contradistinctive generative autoencoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 823–832, 2021.

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.

Stanislav Pidhorskyi, Donald A Adjeroh, and Gianfranco Doretto. Adversarial latent autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14104–14113, 2020.

Vadim Popov, Ivan Vovk, Vladimir Gogoryan, Tasnima Sadekova, and Mikhail Kudinov. Grad-tts: A diffusion probabilistic model for text-to-speech. In *International Conference on Machine Learning*, pages 8599–8608. PMLR, 2021.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019.

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

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

Suman Ravuri, Shakir Mohamed, Mihaela Rosca, and Oriol Vinyals. Learning implicit generative models with the method of learned moments. *arXiv preprint arXiv:1806.11006*, 2018.

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.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

Benjamin Rhodes, Kai Xu, and Michael U. Gutmann. Telescoping density-ratio estimation. In *Advances in Neural Information Processing Systems*, 2020.

Oren Rippel and Ryan Prescott Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

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

Mihaela Rosca, Balaji Lakshminarayanan, and Shakir Mohamed. Distribution matching in variational inference. *arXiv preprint arXiv:1802.06847*, 2018.

Chitwan Saharia, William Chan, Huiwen Chang, Chris A Lee, Jonathan Ho, Tim Salimans, David J Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. *arXiv preprint arXiv:2111.05826*, 2021a.

Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement. *arXiv preprint arXiv:2104.07636*, 2021b.

Mehdi SM Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. *Advances in Neural Information Processing Systems*, 31, 2018.

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798, 2007.

Tim Salimans. A structured variational auto-encoder for learning deep hierarchies of sparse features. *arXiv preprint arXiv:1602.08734*, 2016.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909, 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

Robin San-Roman, Eliya Nachmani, and Lior Wolf. Noise estimation for generative diffusion models. *arXiv preprint arXiv:2104.02600*, 2021.

Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5400–5409, 2017.

Matt Shannon, Ben Poole, Soroosh Mariooryad, Tom Bagby, Eric Battenberg, David Kao, Daisy Stanton, and RJ Skerry-Ryan. Non-saturating gan training as divergence minimization. *arXiv preprint arXiv:2010.08029*, 2020.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. How to train deep variational autoencoders and probabilistic ladder networks. *arXiv preprint arXiv:1602.02282*, 3(2), 2016.

Jiaming Song, Shengjia Zhao, and Stefano Ermon. A-nice-mc: Adversarial training for mcmc. *Advances in Neural Information Processing Systems*, 30, 2017.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020a.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *arXiv preprint arXiv:2006.09011*, 2020.

Yang Song, Conor Durkan, Iain Murray, and S. Ermon. Maximum likelihood training of score-based diffusion models. *arXiv preprint arXiv:2101.09258*, 2021a.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021b.

Yang Song, Liyue Shen, Lei Xing, and Stefano Ermon. Solving inverse problems in medical imaging with score-based generative models. In *International Conference on Learning Representations*, 2022.

Yuxuan Song, Qiwei Ye, Minkai Xu, and Tie-Yan Liu. Discriminator contrastive divergence: Semi-amortized generative modeling by exploring energy of the discriminator. *arXiv preprint arXiv:2004.01704*, 2020b.

Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. *Advances in neural information processing systems*, 30, 2017.

Sandeep Subramanian, Sai Rajeswar, Francis Dutil, Christopher Pal, and Aaron Courville. Adversarial generation of natural language. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 241–251, 2017.

Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori. *Density ratio estimation in machine learning.* Cambridge University Press, 2012.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Hiroshi Takahashi, Tomoharu Iwata, Yuki Yamanaka, Masanori Yamada, and Satoshi Yagi. Variational autoencoder with implicit optimal priors. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5066–5073, 2019.

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

Yee Whye Teh and Geoffrey E Hinton. Rate-coded restricted boltzmann machines for face recognition. *Advances in neural information processing systems*, 13, 2000.

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

Jakub Tomczak and Max Welling. Vae with a vampprior. In *International Conference on Artificial Intelligence and Statistics*, pages 1214–1223. PMLR, 2018.

Jakub M Tomczak and Max Welling. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.

Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. Discrete flows: Invertible generative models of discrete data. *Advances in Neural Information Processing Systems*, 32, 2019.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, 33:19667–19679, 2020.

Arash Vahdat, Evgeny Andriyash, and William G Macready. DVAE#: Discrete variational autoencoders with relaxed Boltzmann priors. In *Neural Information Processing Systems*, 2018a.

Arash Vahdat, William G. Macready, Zhengbing Bian, Amir Khoshaman, and Evgeny Andriyash. DVAE++: Discrete variational autoencoders with overlapping transformations. In *International Conference on Machine Learning (ICML)*, 2018b.

Arash Vahdat, Evgeny Andriyash, and William G Macready. Undirected graphical models as approximate posteriors. In *International Conference on Machine Learning (ICML)*, 2020.

Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125:2, 2016.

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

Tim Van Erven and Peter Harremos. Rényi divergence and kullback-leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797–3820, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2009.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

Yuanhao Wang, Guodong Zhang, and Jimmy Ba. On solving minimax optimization locally: A follow-the-ridge approach. *arXiv preprint arXiv:1910.07512*, 2019.

Ziyu Wang, Shuyu Cheng, Li Yueru, Jun Zhu, and Bo Zhang. A wasserstein minimum velocity approach to learning unnormalized models. In *International Conference on Artificial Intelligence and Statistics*, pages 3728–3738. PMLR, 2020.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019.

Oliver Woodford. Notes on contrastive divergence. *Department of Engineering Science, University of Oxford, Tech. Rep*, 2006.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

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*, 2020a.

Zhisheng Xiao, Qing Yan, and Yali Amit. Likelihood regret: An out-of-distribution detection score for variational auto-encoder. *Advances in neural information processing systems*, 33: 20685–20696, 2020b.

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*, 2021a.

Zhisheng Xiao, Qing Yan, and Yali Amit. EBMs trained with maximum likelihood are generator models trained with a self-adverserial loss. In *Energy Based Models Workshop - ICLR 2021*, 2021b.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion GANs. In *International Conference on Learning Representations*, 2022.

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

Jianwen Xie, Yang Lu, Ruiqi Gao, and Ying Nian Wu. Cooperative learning of energy-based model and latent variable model via mcmc teaching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Jianwen Xie, Zilong Zheng, and Ping Li. Learning energy-based model with variational auto-encoder as amortized sampler. *arXiv preprint arXiv:2012.14936*, 2020.

Haowen Xu, Wenxiao Chen, Jinlin Lai, Zhihan Li, Youjian Zhao, and Dan Pei. On the necessity and effectiveness of learning the prior of variational auto-encoder. *arXiv preprint arXiv:1905.13452*, 2019.

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.

Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

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. *ICML*, 2020a.

Ning Yu, Ke Li, Peng Zhou, Jitendra Malik, Larry Davis, and Mario Fritz. Inclusive gan: Improving data and minority coverage in generative models. *arXiv preprint arXiv:2004.03355*, 2020b.

Han Zhang, Zizhao Zhang, Augustus Odena, and Honglak Lee. Consistency regularization for generative adversarial networks. *arXiv preprint arXiv:1910.12027*, 2019.

Linfeng Zhang, Lei Wang, et al. Monge-amp\ere flow for generative modeling. *arXiv preprint arXiv:1809.10188*, 2018.

Richard Zhang. Making convolutional networks shift-invariant again. In *International conference on machine learning*, pages 7324–7334. PMLR, 2019.

Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *ICLR*, 2017.

Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. *Advances in Neural Information Processing Systems*, 33, 2020.

Zhengli Zhao, Sameer Singh, Honglak Lee, Zizhao Zhang, Augustus Odena, and Han Zhang. Improved consistency regularization for gans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11033–11041, 2021.

Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.

Zachary Ziegler and Alexander Rush. Latent normalizing flows for discrete sequences. In *International Conference on Machine Learning*, pages 7673–7682. PMLR, 2019.