DEEP GENERATIVE MODELLING: A COMPARATIVE REVIEW OF NORMALISING FLOWS AND DIFFUSION MODELS

A thesis submitted to the University of Manchester
for the degree of Bachelor of Science
in the Faculty of Science and Engineering

15th May 2023

Yuqi Jing
Department of Mathematics

Contents

Abstract **6**

Declaration **7**

Copyright Statement **8**

Acknowledgements **9**

1 Introduction **11**

2 Background **16**

3 Methods **19**

3.1 Normalising Flows 21

3.1.1 Whitening Transformations and Historical Overview 22

3.1.2 Definition and Expressive Power 22

3.1.3 Maximum Likelihood-based Training 24

3.1.4 Contractive Residual Flow 25

3.2 Diffusion Models 29

3.2.1 Forward Diffusion Process 30

3.2.2 Reverse Generative Process 33

3.2.3 Maximum Likelihood-based Training Objective 36

3.2.4 Sampling Approach 39

3.2.5 Neural Network Architecture 40

4 Results **44**

4.1 Data Sets 44

4.2 Evaluation Metrics 46
4.3 Experimental details 47
4.3.1 Normalising Flow 47
4.3.2 Diffusion Model 48
4.4 Samples and Evaluation 49
4.4.1 MNIST Samples 49
4.4.2 CIFAR10 Samples 53

5 Discussion **59**
5.1 Experimental Results Discussion 59
5.2 Further Improvements and Future Work 63

6 Conclusions **64**

A Computer code **73**

List of Tables

4.1 Datesets in our experiments 45
4.2 Details of the U-Net architecture in experiments 49
4.3 Details of the U-Net training settings in experiments 49
4.4 Inception score and FID score of the MNIST images. 51
4.5 Inception score and FID score of the CIFAR10 images. 55

List of Figures

1.1 An instance of images generated by diffusion models. 14

3.1 Example of noisy images in the forward diffusion process 31
3.2 Illustration of the diffused data distribution. 33
3.3 Example of images in the reverse generative process. 34
3.4 Examples of sinusoidal position embeddings. 42

4.1 MNIST images generated by the normalising flow 50
4.2 MNIST images generated by the diffusion model 50
4.3 MNIST images generated using classifier guidance by diffusion model. 53
4.4 CIFAR10 images generated by the normalising flow. 54
4.5 CIFAR10 images generated by the diffusion model. 54
4.6 CIFAR10 images generated by the Large DDPM. 57
4.7 The loss curve of the MNIST and CIFAR10 training process. 58

The University of Manchester

Yuqi Jing
Bachelor of Science
Deep Generative Modelling: A Comparative Review of Normalising Flows and Diffusion Models
15th May 2023

Deep generative modelling is one of the most successful deep learning applications. Recently, many new deep generative models have been proposed to improve sample quality and diversity. These methods usually share similar structures with distinct learning approaches. A side-to-side comparision can hence gain valuable insights of the model design. In this thesis, we provide a comparative review of such two deep generative models – normalising flows and diffusion models. Specifically, we detail the fundamental principles of these two frameworks and the architectures of two particular models. Extensive experiments are then implemented to show the effectiveness and efficiency of these two models. We demonstrate that normalising flows excel at sample quality, whereas diffusion models learn the class embedding more effectively. Moreover, normalising flows have an edge in sampling speed, although diffusion models are superior in training speed. The code for this thesis is made available from: `https://github.com/yqjing/sem2project`.

Declaration

I declare this thesis is my original work. No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `https://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420`), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `https://www.library.manchester.ac.uk/about/regulations/`) and in The University's Policy on Presentation of Theses.

Acknowledgements

First, I owe deep gratitude to my supervisor Korbinian Strimmer, who motivates me to set high bars, focus on big things and care for details; helps me in improving writing and computer skills; supports me with great patience when I was applying for masters and advises me on the future path; and most importantly inspires my interest in statistics and introduces me to the field of statistical learning (it is true: before taking his course in the second year, I had zero interest in statistics). Being able to conduct the project closely with him is one of the best choices I have ever made. Without this man, I cannot imagine who and what state I will be right now.

Second, I thank my second thesis marker, Theodore Papamarkou, for the time and effort spent on reading and evaluating the thesis.

For some undeserving luck, my Bachelor journey has crossed incredible minds during my living in 12 Denmark Road. Yuqi Bai, being a fourth year chemical engineering Ph.D. candidate, teachs me the invaluable skill of keeping a regular schedule in the volatile day-to-day life; how to think out-of-the-box and always be ready to defend my own ideas, *etc.* I cannot overstate the continued impact of Nan Hu, a third year acoustic Ph.D. candidate, in my study and even in my personal life, and will strive to live-up to his standard. I would also like to thank Boxun Chen - a Taiwanese medical Ph.D. candidate and physician who acts just like a big brother to me, and gives me priceless advice on writing skills, securing funding for study, *etc.*

I also met amazing new classmates and friends after coming to UoM. To name some: Homa Molavi, Niall Starling, Gursel Carkar, Anna Jaeggi, Oscar Kollar, Richard Lin, Shuai Song, William Zhang, Jiayi Zhao, Jiarui Diao, Qiuhua Wang, Jiajun Zhang, Xinyue Xuan. I am especially indebted to: Xuan, who helped me settle down in the first year; William, who polished emails for me and will share an unforgettable trip Vienna; Homa, who jogged with me and made my summer time in the UK more

than enjoyable.

Finally, I would like to dedicate this thesis to my family. Pursuing one's degree some 5000 miles away from the hometown is a big challenge despite modern technologies for connection and communication. Sorry mom, dad and grandpa for missing so many important occasions, and thank you so much for the years of understanding, belief and sometimes lie to make it work.

Chapter 1

Introduction

Deep generative models, a kind of unsupervised probabilistic models, have made tremendous progress in recent years. On the one hand, unsupervised learning (self-supervised) provides a plausible way to supply enough data for training a large learning model with the hundreds of millions of parameters. On the other hand, probabilistic modelling is able to fully utilise the bonus of deep learning revolution, namely the expressiveness and flexibility of the deep neural networks.

Despite the fancy name and various variations, the essential idea of deep generative models is to estimate the probability distribution of data, which categorises it as the probabilistic models. For this, we first collect a large data set from the target distribution which is refered to as *data distribution* in statistical learning literature. In most cases, this data distribution is unknown to us and the sampled data points are *i.i.d* samples. Then a model is created to represent the parameterised probability distribution, which we call the *model distribution*, by learning from these input data points. (In some literature, the model is equivalent with the model distribution. For explicity of notations, we distinguish them in this review.) This stage corresponds to the training stage of the model. The majority of the models are trained based on maximum likelihood.

However, as the name suggests, there is one additional function needed for a generative model, which is the capability to generate new data points, or sample from the model distribution. In practice, this is done by first sampling data points from a standard Gaussian distribution, then transform them using the fitted model distribution. Nevertheless, depending on the approaches, the model distributions

can take various forms. For example, some model distributions are extended to a time series by introducing the time variable. This gives rise to different complexities in terms of sampling from the model distribution. As a result, the sampling stage of the model is the other big concern for generative models.

In order to build powerful generative models that can estimate high-dimentional probability distribution, deep neural networks are leveraged due to their expressiveness and flexibility, which explains the name deep generative models. There are overall two paradigms of neural generative models. The first one is to use a deep neural network to directly represent a probability distribution, but it is often nontrivial to implement. The other approach is to use deep neural networks to model the data distribution in an implicit manner. Specifically, deep neural networks is used to learn representations that contain the important information about the data distribution.

The deep generative models have numerous applications. The most direct and influential applications of the generative model so far are image synthesis and large language model, represented by two famous softwares developed by OpenAI: DALL·E 2 and ChatGPT respectively (Dhariwal and Nichol, 2021; Brown et al., 2020). For the domain of this thesis, we focus on the application of generative models to synthesize images and therefore do not consider the language model, although the fundamental ideas behind them are the same. In the scope of image synthesis, the generative models are able to perform various default tasks, such as image inpainting, super-resolution, text-to-image conversion and image-to-image conversion (Saharia et al., 2022b,a). In addition, Deep generative models are useful for many scientific applications. It has been applied in weather nowcasting to predict the radar map by DeepMind (Ravuri et al., 2021). Some softwares have also used the deep generative models to produce code automatically, such as GitHub Copilot (Chen et al., 2021).

Beyond this, many previously unattainable tasks can be solved efficiently by the generative models, especially in the area of privacy enhancing technology. While privacy enhancing technology classically works on the one-to-one de-identification of the orignal data by adding some perturbation, the generative models can produce independent dataset based on real dataset, therefore the one-to-one correspondence does not exist, so this method will work better. The similar progress in data sharing

and AI audit also demonstrates the usefulness of generative models.

If we watch the landscape of deep generative modelling in computer vision, we will see a lot of frameworks. There are generative adversarial networks (GANs), variational autoencoders, autoregressive models, energy-based models, normalising flows and diffusion models *etc.* (Goodfellow et al., 2014; Kingma and Welling, 2013; Bengio et al., 2000; Hinton, 2002; Rezende and Mohamed, 2015; Sohl-Dickstein et al., 2015). The background of these generative models is described in Chapter 2. All these generative models have their own structures, training and sampling principles. In this review, we mainly focus on two frameworks of generative models that share similar structures, training and sampling approaches. In the meantime, there are key differences in the usage of neural networks and the form of training objective functions between them, resulting in very distinct model behaviours.

The normalising flow model uses the simple idea of invertible functions to achieve both training and sampling. The model uses neural networks directly to represent the data probability distribution. Specifically, the model learns the data distribution by transforming data points sampled from data distribution to the white noise in the standard Gaussian distribution. In the contractive residual flow model (Chen et al., 2020), deep neural network is used to conduct such operation. Due to the invertibilty of the activation functions in the neural network, novel data points can be easily obtained by inversely transforming standard white noise into meaning data points using the fitted model distribution.

Despite the simplicity of the structure, normalising flows are extremely powerful and have found applications in many areas. Apart from basic applications such as probabilistic modelling, volume-preserving flows are also applied in *Monte Carlo Markov Chain* (MCMC) to propose states for the Metropolis–Hastings algorithm (Song et al., 2017). In addition, normalising flows can efficiently disentangle anomalies in the target distribution by enabling MCMC to perform on the less intricate and well-mannered base density (Hoffman et al., 2019).

In the 2020s, a new type of generative model – diffusion model gradually gained momentum, and has become the latest trending model. With its intuitive nature based on the statistical physics, it is able to generate images with amazing quality as

Figure 1.1: An example of images generated by diffusion model, with text-to-image synthesis on LAION dataset (Schuhmann et al., 2021). Prompt: a photograph of an astronaut riding a horse (512×512).

well as a variety of categories. By introducing the time varible, the diffusion models uses neural networks to implicitly represent the data distribution. Analogous to normalising flow, the model learns the data distribution by transforming data points sampled from data distribution to the white noise in the standard Gaussian distribution. Unlike normalising flow, the model adds random noise to the data points at each time step instead of using the neural network to conduct the transformation. The neural network is then deployed to learn the variation in the data points at each step. The sampling is done similarly to that in the normalising flow by transforming the standard white noise back to the meaningful data points using the trained neural networks. However, alike the training, it is done iteratively instead of the one-shot style in the normalising flow.

Intriguingly, the iterative design of the diffusion model allows for a guiding mechanism that can control the image generation process with conditioning. It makes it well-suited for the applicable tasks such as text-to-image synthesis. Figure 1.1, shows an image generated by Stable Diffusion Model, which is the state-of-the-art synthesis model developed by the CompVis group at LMU Munich (Rombach et al.,

2022) .

Given the above description, we propose the following assumptions. ***(i)*** The normalising flow obviously has an edge in sampling speed since the transformation is finished in one-shot, while the diffusion model takes multiple steps and requires input data at each step. ***(ii)*** The diffusion model outperforms in the training speed, since the magnitude of variation in transformation is clearly much smaller for diffusion model than the normalising flow, resulting in much simpler training objective and much faster convergence speed. The sharp contrast in training and sampling efficiency makes these two models a perfect pair for comparison. Given their similar model structure but different learning approach, any big difference in their sample quality is worth examining.

The purpose of this review is twofold. First, we aim to validate the proposed assumptions through experimental implementations. We show that the choice of architectures and learning approaches culminate to trade-offs in training and sampling execution times. Second, by comparing the difference in the sample quality of the two models, we hope to gain insights of the model design and learning approaches in the development of generative models.

The rest of the thesis is organised as follows: in **Chapter** 2, we provide a brief review of the roadmap of development of generative models in the last two decades and introduce the various categories of the deep generative models. The foundational ideas of the normalising flow and diffusion model are introduced in **Chapter** 3. Additionally, we give the detailed description of two particular models – denoising diffusion probabilistic model (DDPM) and contractive residual flow (CRF). The experimental setup and results of DDPM and CRF are presented in **Chapter** 4. We discover that our proposed assumptions are consistent with the experimental results. **Chapter** 5 gives the discussion of the training & sampling execution times, sample quality & class embedding capacity of DDPM and CRF from the experimental results. Besides, the potential next steps for work are outlined. Finally, we provide the concluding remarks about sample quality and class embedding learning ability, training and sampling efficiency of normalising flows and diffusion models in **Chapter** 6.

Chapter 2

Background

We would like to start this chapter by a quote from Geoffrey Hinton who is a Turing Recipient, one of 'Godfathers' of Deep Learning in his 2014 AMA on Reddit (Hinton, 2014) :

> The brain has about 10^{14} synapses and we only live for about 10^9 seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning since the perceptual input (including proprioception) is the only place we can get 10^5 dimensions of constraint per second.

The above description shows that human brain is overparametrized, however a human is able to learn new things and make inference effectively through the huge amount of information inputed into their seneses every second. The same idea also applies to modern machine learning, where the unsupervised (self-supervised) learning may be the only possible way to provide enough data for training a large learning model with the hundreds of millions of parameters. As a result, the future of machine learning, from our perspective, really belongs to the unsupervised learning.

In this review, we focus on deep generative models, a kind of unsupervised probabilistic models, which have made tremendous progress in recent years. For simplicity, we can understand generative models as a machine learning model that learns to generate data. The more detailed description is be provided in Chapter 1.

The history of neural generative models starts long before the prevalence of deep learning (Krizhevsky et al., 2012). The earliest families of deep generative models

include energy-based models (the most typical example is restricted Boltzmann Machine (RBM) (Hinton, 2002)) and autoregressive models (Bengio et al., 2000). Another impactful deep generative model developed at the early stage (2008) is the denoising autoencoder (DAE) (Vincent et al., 2008), which can be regarded as an alternative to RBM and belongs to the energy-based family.

Until 2010, the whole field of deep generative modelling has still been in an exploratory mode. Then time comes to 2012. Motivated by the deep learning revolution (Krizhevsky et al., 2012), the entire data science community was enchanted with the deep neural networks and there has been a cambrian explosion of new deep generative models. Specifically, four main catogeries of model frameworks were developed during this period. Among them, the renowned variational autoencoders (VAE) and generative adversarial networks (GAN) emergied in 2014 (Kingma and Welling, 2013; Goodfellow et al., 2014), while then less well-known normalising flow and denoising diffusion model were proposed in the following year (Rezende and Mohamed, 2015; Sohl-Dickstein et al., 2015). However, the research on the generative models has not been very active because of the overwhelming dominance of supervised learning, to the extent that some of the most faithful researchers in the field of unsupervised learning began to shift focus to supervised learning algorithms(Alain et al., 2016).

It was not until 2018 when language models GTP (autoregressive modelling) and BERT (masked autoencoding) crashed into the field of natural language processing (NLP) that people started to realise the expressive power of unsupervised pre-trained representations (Radford and Narasimhan, 2018; Devlin et al., 2018). Taking advantage of the unsupervised nature of the model, GPT-3 can be trained on unlabeled large dataset such as Wikipedia, Common Crawl to have over 100 trillion parameters (Brown et al., 2020). After fine-tuning on downstream tasks, these highly overparameterized model showed amazing scalability. Likewise, unsupervised visual representation learning also generated masses of excitement in computer vision community. An elementary linear classifier using fixed, unsupervised pre-trained representation on ImageNet is capable of achieving similar accuracy of the supervised end-to-end classifier on the same architecture, the same dataset. In several downstream tasks, it can even surpass their supervised counterparts by a

large amount (He et al., 2020; Chen and He, 2021).

The success in the unsupervised pre-training greatly drew the community's attention back to the deep generative models and its inherent representation learning capability. GAN and VAE were first to demonstrate impressive performance on demanding taskes such as generating photorealistic human faces (Karras et al., 2018; Vahdat and Kautz, 2020). More recently, especially in the 2020s, normalising flow and diffusion model gained popularity after showing competitive performance results. Interestingly, the images generated by diffusion model display similar high-quality as well as diverse catogeries which was not achieved previously by GAN (Ho et al., 2020).

It's worth noting that all generative models mentioned above are distilled (generalised) ideas. They can be imagined as **building blocks** for the model architecture. In practice, a powerful method usually comprises several of these ideas in order to trade off the advantages and disadvantages of different models. For example, normalising flow is usually combined with autoregressive model, and variational bayes is also used in diffusion model (Bond-Taylor et al., 2021). Given the intertwined nature of deep generative model, in this review we delve into the foundations of two paradigms of deep generative models - **normalising flow** and **denoising diffusion model**, with sequential popularity in the last half a decade. We also discuss practical implementation details of these two models and compare the experimental results. We detail the rationale for choosing these two particular models in Chapter 1.

Chapter 3

Methods

In this chapter, we give detailed descriptions of two deep generative models, *i.e.* normalising flows and diffusion models (Rezende and Mohamed, 2015; Ho et al., 2020).

Distinct from many other generative models, the two models mentioned above share the same fundamental training and sampling principles. More importantly, the diffusion model is built on research on the normalising flow with some key differences. As a result, by bringing these two models together, we aim to reveal the underlying ideas behind generative models and inspire further improvements to the design of generative models.

The chapter is organised as follows: in Section 3 we first set up the uniform notation for this chapter and the rest of the thesis. Section 3.1 establishs the formal and conceptual definition of normalising flows as well as the specific details of contractive residual flow model. The denoising diffusion probabilistic model and a general perspective of diffusion models are detailed in Section 3.2.

Notations

In the framework of deep generative models , we aim to acquire a knowledge of the distribution of underlying data. In order to achieve that, we explore the statistical approaches that learn the parameters θ of a model by minimising some objective function $L(\theta)$ (Vincent, 2011). For the sake of uniformity of reading, we introduce the notations of all generative models covered in this review in this section. Specifically,

we express all distributions with their probability density functions (p.d.f.) on $\mathbb{R}^m$. Additionally, the discrete distributions will be represented by Dirac-deltas δ.

$\nabla_\theta f = [\frac{\partial f}{\partial \theta_1}, \ldots, \frac{\partial f}{\partial \theta_m}]$	Gradient operator w.r.t. m-dimensinal parameter θ.
$\mathbf{J}_f(\cdot)$	The Jacobian of a function $f : \mathbb{R}^m \to \mathbb{R}^m$.
$\log p_\theta(\mathbf{x})$	Log *model* density of input data in flow-based models
$q(\mathbf{x})$	Unknown *true* p.d.f. $\mathbf{x} \in \mathbb{R}^m$.
$D_n = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$	Training set: independent and identically distributed (i.i.d) samples from q.
$\mathbf{x}_t,\ t \in \{1, 2, ..., \mathrm{T}\}$	A sequence of noisy images.
$q(\mathbf{x}_t\|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\ \beta_t \mathbf{I}_m)$	Gaussian kernel: adding noise progressively.
$q(\mathbf{x}_0, \mathbf{x}_t) = q(\mathbf{x}_0)\, q(\mathbf{x}_t\|\mathbf{x}_0)$	Joint pdf.
$q(\mathbf{x}_1, ..., \mathbf{x}_T\|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t\|\mathbf{x}_{t-1})$	Joint distribution: Markov chain property.
$q(\mathbf{x}_t\|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_0, (1-\alpha_t)\mathbf{I}_m)$	Diffusion kernel by marginalising $q(\mathbf{x}_1, ..., \mathbf{x}_T\|\mathbf{x}_0)$.
$q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{0}, \boldsymbol{I})$	Pdf of $\mathbf{x}_T$.
$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$	The standard Gaussian white noise $\boldsymbol{\epsilon}$.
$q(\mathbf{x}_{t-1}\|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1}) q(\mathbf{x}_t\|\mathbf{x}_{t-1})$	Bayes theorem.
$q(\mathbf{x}_{t-1}\|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \boldsymbol{I})$	Inverse (posterior) distribution conditioned on $\mathbf{x}_t$, $\mathbf{x}_0$.
$p_\theta(\mathbf{x})$	Density *model* with parameter θ.
$\mu_\theta(\mathbf{x}_t, t)$	Denoising autoencoder with input $(\mathbf{x}_t, t)$.
$-\log p_\theta(\mathbf{x}_0)$	Negative log likelihood of data $\mathbf{x}_0$ in diffusion models.
$\mathbb{E}_{q(\mathbf{x})}\big[p(\mathbf{x})\big] = \int_{\mathbf{x}} q(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$	Expectation with respect to distribution q.
$\mathbb{E}_{q(\mathbf{x}_0)}\big[-\log p_\theta(\mathbf{x}_0)\big]$	Expectation of $-\log p_\theta(\mathbf{x}_0)$ with respect to $q(\mathbf{x}_0)$.
$D_{KL}(q(\mathbf{x}_{t-1}\|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}\|\mathbf{x}_t))$	KL divergence between $q(\mathbf{x}_{t-1}\|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}\|\mathbf{x}_t)$.
$\|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2$	L^2 norm between $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$ and $\mu_\theta(\mathbf{x}_t, t)$.

3.1 Normalising Flows

The curse of dimensionality is the central research area in high-dimensional statistics. The traditional approaches use the parametric functional form to estimate the density of the data set. However, it often struggles to capture the dependence of data in the high-dimensional information set. In recent years, nonparametric modelling has become popularised among statistics community. But curse of dimensionality in nonparametric modelling has been a large obstacle for many years until the development of algorithmic solutions such as neural network.

The compositional property of neural network is the main force to fight the curse of dimensionality in high-dimensional data set. In neural nets, we are getting compositionality in two ways. The first way is through the distributed representations, namely for any given input we can have a number of hidden units that are being composed with each other in one layer, which is the width of the neural nets. Additionally, by stacking one layer on top of another layer it is essentially the natural mathematical compositional functions, which is termed as the exponential advantage of depth (Montufar et al., 2014).

The similar idea of composition also saw its emergence in the machine learning model "normalising flow" (Tabak and Turner, 2013), where compositionality is built into the model through a series of simple nonlinear invertible transformations called "flows". The key idea of normalising flow resembles that of the deep neural network, which is the representation power of compositionality. By parameterising flows with deep neural networks, the flow-based method is able to take full advantage of the compositional power of computation. This results in a quite universal and expressive probabilitic model.

Most importantly, as the flow-based models start to show their colour in carefully designed models such as GLOW (Kingma and Dhariwal, 2018), they demonstrate that a generative model optimized towards the plain log-likelihood objective is capable of performing efficient generation of large and photo-realistic images, which was previously monopolised by non-maximum-likelihood models. This greatly injects confidence to the maximum likelihood-based probabilistic modelling, and lays the foundation for the later diffusion-based model.

The rest of the section is organised as follows: in Section 3.1.1, we give a brief historical overview of the normalising transformation. We present the definition of normalising flow and some justification of its expressive power in Section 3.1.2. The maximum likelihood-based training method is introduced in Section 3.1.3. Finally, the contractive residual flow is presented in Section 3.1.4 as well as the model architecture.

3.1.1 Whitening Transformations and Historical Overview

Whitening transformations are long-established class of invertible location-scale transformations (Strimmer, 2022). They utilise the decorrelation property to convert high-dimensional models into uncorrelated univariate models. Specifically, whitening means transforming the covariance matrix of the model into a spherical, isotropic covariance matrix $\mathbf{I}_d$ (such as standard multivariate normal distribution). There are several well-known strategies for linear whitening procedures $\mathbf{h}(\mathbf{x}) = \mathbf{a} + \mathbf{B}\mathbf{x}$, such as ZCA, PCA and Cholesky. In addition, whitening transformations have been applied in density estimation through the invertible transformations (Chen and Gopinath, 2000).

However, linear transformations have limited expressive capacity such that they struggle to represent potentially more multi-modal distributions. This is where nonlinear transformations such as general invertible tranformation $\mathbf{h}(\mathbf{x}) = \mathbf{y}(\mathbf{x}) \rightarrow \mathbf{h}^{-1}(\mathbf{y}) = \mathbf{x}(\mathbf{y})$ come to rescue. Further, repeated simple nonlinear transformations to a unimodel distribution can result in distributions of extremely complex structure, which is the key idea of normalising flow-based models. Specifically, in normalising flow models, instead of going from the true data distributions to standard white noise directly like in the linear model, the data distributions are gradually transformed to a standard Gaussian distribution through a lot of intermediate representations.

3.1.2 Definition and Expressive Power

Let $\mathbf{x}$ be a m-dimensional vector sampled from true data distribution $q_x(\mathbf{x})$, and suppose $\boldsymbol{u}$ is sampled from the *base distribution* $q_z(\mathbf{z})$. In this essay we set the base

distribution to be fixed as $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. Nevertheless it's worth noting that $p(\mathbf{z})$ can have parameters of its own in some cases.

A normalising flow model $f : \mathbb{R}^m \to \mathbb{R}^m$ is constructed as an invertible transformation that maps the observed data $\mathbf{x}$ to a standard Gaussian latent variable (Ho et al., 2019):

$$\mathbf{z} = f(\mathbf{x}). \tag{3.1}$$

The central idea in the construction of a normalising flow is to configurate f by stacking individual simple invertible transformations (Ho et al., 2019). Specifically, f is assembled by composing a sequence of invertible flows as $f(\mathbf{x}) = f_1 \circ \cdots \circ f_T(\mathbf{x})$, where each $f_i : \mathbb{R}^m \to \mathbb{R}^m$ is invertible, differentiable and has a tractable determinant of the Jacobian. Under these conditions, the sampling from the model is simple and fast, as $\mathbf{x}$ can be obtained by calculating

$$\mathbf{x} = f^{-1}(\mathbf{z}) = f_T^{-1} \circ \cdots \circ f_1^{-1}(\mathbf{z}), \tag{3.2}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_m)$, and density of $\mathbf{x}$ can be represented with $\boldsymbol{u}$ by the change of variables (Papamakarios et al., 2021):

$$q_x(\mathbf{x}) = q_z(\mathbf{z}) \left|\det \mathbf{J}_{f^{-1}}(\mathbf{z})\right|^{-1}, \tag{3.3}$$

where $\mathbf{z} = f(\mathbf{x})$. In practice, the transformation f is often designed to be a neural network with parameter θ. In order to train the transformation f_θ, we rewrite the Equation 3.3 in the form of the estimated density model:

$$\begin{aligned} p_\theta(\mathbf{x}) &= q_z(f(\mathbf{x})) \left|\det \mathbf{J}_f(\mathbf{x})\right| \\ &= \mathcal{N}(f(\mathbf{x}); \mathbf{0}, \mathbf{I}_m) \prod_{i=1}^{T} \left|\det \frac{\partial f_i}{\partial f_{i-1}}\right|, \end{aligned} \tag{3.4}$$

where $p_\theta(\mathbf{x})$ is the model density, $q_z(\cdot)$ is of the distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I}_m)$, the Jacobian $\partial f_i / \partial f_{i-1}$ is the $m \times m$ matrix of the form:

$$\frac{\partial f_i}{\partial f_{i-1}} = \begin{bmatrix} \frac{\partial f_{i,1}}{\partial f_{i-1,1}} & \cdots & \frac{\partial f_{i,1}}{\partial f_{i-1,m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{i,m}}{\partial f_{i-1,1}} & \cdots & \frac{\partial f_{i,m}}{\partial f_{i-1,m}} \end{bmatrix}. \tag{3.5}$$

For the purpose of training by maximum likelihood, the model density is often represented in log form:

$$\log p_\theta(\mathbf{x}) = \log \mathcal{N}(f(\mathbf{x}); \mathbf{0}, \mathbf{I}_m) \; + \; \sum_{i=1}^{T} \log \left|\det \frac{\partial f_i}{\partial f_{i-1}}\right|. \tag{3.6}$$

To sum up, there are two directions in a flow-based model:

1. Training is defined as the forward direction, where we go from the input data $\mathbf{x}$ to the *base data* $\mathbf{z}$ through the trained transformation f_θ. Explicitly, we train the transformation f_θ by evaluating the model's density in Equation 3.6. Therefore, we are required to compute the transformation f_θ and its Jacobian determinant for each training data point $\mathbf{x}$.

2. Sampling is defined as the backward direction, which goes from base data $\mathbf{z}$ to generated data $\mathbf{x}$ through the inverted transformation f^{-1}. Hence, we need to sample from $q_z(\mathbf{z})$ and compute the inverted transformation f^{-1} for each generated data $\mathbf{x}$.

As a result, it is worth reemphasising that we must ensure that the transformation f is invertible and has a tractable Jabobian determinant.

3.1.3 Maximum Likelihood-based Training

In this section, we introduce two objective functions for training the normalising flow based on the maximum likelihood metrics. Explicitly, we present the forward KL divergence and reverse KL divergence for training under different situations (Papamakarios et al., 2021).

Given the log density of the flow-based model $p_\theta(\mathbf{x})$, the forward KL divergence between the true data distribution $q_x(\mathbf{x})$ and the normalising flow model $p_\theta(\mathbf{x})$ can be represented as:

$$\begin{aligned} L_{\text{forward}} &= D_{KL}\Big(q_x(\mathbf{x}) \| p_\theta(\mathbf{x})\Big) \\ &= -\mathbb{E}_{q_x(\mathbf{x})}\Big(\log q_z\big(f_\theta(\mathbf{x})\big) + \log\Big|\det \mathbf{J}_{f_\theta}(\mathbf{x})\Big|\Big) \ + \ \text{const.} \end{aligned} \tag{3.7}$$

The forward KL divergence is applicable in scenarios when we have access to the samples $D_n = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$ from the true data distribution $q_x(\mathbf{x})$, while in the meantime we cannot calculate the true data density $q_x(\mathbf{x})$ (Papamakarios et al., 2021). Given the set of samples $D_n = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$, we further derive an empirical estimate

for expectation over $q_x(\mathbf{x})$ using Monte Carlo:

$$L_{\text{forward}} \approx -\frac{1}{N}\sum_{i=1}^{N}\Big(\log q_z\big(f_\theta(\mathbf{x}^{(i)})\big) + \log\Big|\det \mathbf{J}_{f_\theta}(\mathbf{x}^{(i)})\Big|\Big) \;+\; \text{const.} \tag{3.8}$$

Minimising the KL divergence in Equation 3.8 is equal to maximising the maximum likelihood for the normalising flow f_θ with the data $D_n = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$.

In terms of optimisation of parameter θ, we apply the stochastic gradient descent methods. We give the estimate of parameter gradient of the objective function Equation 3.8 below:

$$\nabla_\theta L_{\text{forward}} \approx -\frac{1}{N}\sum_{i=1}^{N}\Big(\nabla_\theta \log q_z\big(f_\theta(\mathbf{x}^{(i)})\big) + \nabla_\theta \log\Big|\det \mathbf{J}_{f_\theta}(\mathbf{x}^{(i)})\Big|\Big). \tag{3.9}$$

In order to cover broader training situations, we present the another objective function – reverse KL divergence for training the flow-based model. The reverse KL divergence is formulated as:

$$\begin{aligned} L_{\text{reverse}} &= D_{KL}\Big(p_\theta(\mathbf{x})\|q_x(\mathbf{x})\Big) \\ &= \mathbb{E}_{q_z(\mathbf{z})}\Big(\log q_z(\mathbf{z}) \;-\; \log\Big|\det \mathbf{J}_{f_\theta{}^{-1}}(\mathbf{z}) \;-\; \log q_x\big(f_\theta^{-1}(\mathbf{z})\big)\Big|\Big) \;+\; \text{const.} \end{aligned} \tag{3.10}$$

In the case of reverse KL divergence, we are able to calculate the true data density $q_x(\mathbf{x})$, but we do not have access to samples from it. Instead we sample a set of $\{\mathbf{z}\}_{i=1}^{n}$ from the base distribution $q_z(\mathbf{z})$. Then we optimise the parameter θ by computing the inverse transformation f_θ^{-1} and its Jacobian determinant with respect to the base data.

The reverse KL divergence objective function is often applied in the situations of *variational inference*. In this respect, given a prior density of the true data, we aim to derive the posterior distribution of the true data $p_\theta(\cdot)$ by multiplying the prior with a likelihood function (Papamakarios et al., 2021).

3.1.4 Contractive Residual Flow

In the literature of the flow-based model, the majority of the work concentrates on creation of maximally expressive transformation f_θ while retaining the invertibility of the flow and the tractable Jacobian determinant calculation (Kingma and Dhariwal,

2018). In this section, we introduce the *contractive residual flow*, a novel flow-based model that achieves the above goal, in the meantime bringing about the memory-efficient training (Chen et al., 2019). The memory saving property of the contractive residual flow largely results in us choosing it as the demonstrating model in the experiments. As illustrated in Section 4, it can achieve efficient training and competitive performance in the generation tasks compared to the GLOW (Kingma and Dhariwal, 2018), while maintaining low GPU memory consumption.

The general residual flow follows the simple transformation (Chen et al., 2019):

$$\mathbf{z} = f(\mathbf{x}) = \mathbf{x} + g(\mathbf{x}),$$

where g is a function that resembles the skip connection in the residual network (He et al., 2016). The residual transformation is not invertible inherently. However, it is invertible if g is contractive by the Banach fixed point theorem (Chen et al., 2019). In general, the function g is defined to be contractive with respect to a distance measure δ if there exists a constant $L < 1$ such that for any two inputs $\mathbf{u}_\alpha$ and $\mathbf{u}_\beta$ there is:

$$\delta\big(g(\mathbf{u}_\alpha), g(\mathbf{u}_\beta)\big) \leq L\delta(\mathbf{u}_\alpha, \mathbf{u}_\beta), \tag{3.11}$$

where L is the Lipschitz constant. The inverse transformation f^{-1} is often derived iteratively. It can be shown that as the Lipschitz constant L gets smaller, fewer iterations are required to invert the flow, but the flow gets less expressive. Therefore, it is important to choose an optimal L in order to trade off the computational cost and performance when designing the contractive residual flow model.

One of the major born edges of the contractive flow is that most of the activation layers (nonlinear functions) in deep neutral network are contractive with $L < 1$, such as logistic sigmoid, tanh, ReLu etc. In addition, linear layers, including fully-connected layers and convolutional layers, can be easily made contractive by dividing with a constant strictly larger than their operator norm (Papamakarios et al., 2021). Finally, the composition of T Lipschitz contractive functions can be proved to be contractive as well. As a consequence, a carefully designed deep neural network can serve as a perfect fit for the transformation f in contractive residual flow. In other words, the contractive residual flow is in some way equivalent to a deep neural network (DNN).

Positioning a DNN in the place of the flow transformation also solves the difficulty to calculate the Jacobian determinant, where automatic differentiation techniques such as backpropogation can be leveraged. However, it is often infeasible to compute the Jacobian determinant with such techniques for high-dimentional data such as images.

3.1.4.1 Training

To tackle the feasibility of computing Jacobian determinant, there has been two stages for developing the robust estimate for the Jacobian determinant in the contractive residual flow model in order to train the model. Firstly, Behrmann et al. (2019) proposed to use a truncation estimator to estimate the Jacobian determinant. However, it causes a large bias by cropping the infinite series, which inevitably damages the model's performance. In the second stage of the development, Chen et al. (2019) presents an unbiased estimate of the log absolute Jacobian determinant. To replicate their work, we first present the log absolute Jacobian determinant as a power series:

$$\log\left|\det \mathbf{J}_{f_\theta}(\mathbf{x})\right| = \log\left|\det\left(\mathbf{I}_m + \mathbf{J}^k_{g_\theta}(\mathbf{x})\right)\right| = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \mathrm{Tr}\left\{ \mathbf{J}^k_{g_\theta}(\mathbf{x}) \right\}, \tag{3.12}$$

where $\mathbf{J}^k_{g_\theta}(\mathbf{x})$ is the k-th power of the Maclaurin series at $\mathbf{x}$. The infinite series converges if and only if $\left\|\mathbf{J}^k_{g_\theta}(\mathbf{x})\right\| < 1$ for some matrix norm $\left\| \cdot \right\|$, which is equivalent to g_θ being contractive.

The trace of $\mathbf{J}^k_{g_\theta}(\mathbf{x})$ can be estimated efficiently with the operation:

$$\mathrm{Tr}\left\{ \mathbf{J}^k_{g_\theta}(\mathbf{x}) \right\} \approx \mathbf{v}^T \mathbf{J}^k_{g_\theta}(\mathbf{x})\, \mathbf{v}, \tag{3.13}$$

where $\mathbf{v}$ is any m-dimensional vector with mean zero and covariance identity matrix. This estimate is termed as *Hutchinson trace estimator* (Hutchinson, 1990) proposed in 1990. The product $\mathbf{v}^T \mathbf{J}^k_{g_\theta}(\mathbf{x})\, \mathbf{v}$ can be obtained with k backpropagation passes. Eventually, in Chen et al. (2019), the infinite sum of Maclaurin series is efficiently approximated by a finite sum of reweighted terms using the *Russian-roulette estimator* (Chen et al., 2019).

3.1.4.2 Sampling and Generative Modelling

For the generating tasks in the contractive residual flow, the sampling is done iteratively. Specifically, the sample $\mathbf{z}$ drawn from the base distribution is transformed by the f_θ iteratively until convergence, as the inverse transformation f_θ^{-1} cannot be obtained analytically. This procedure is different from the one-pass sampling provided in the traditional flow-based models. However, as shown later in Section 4, the speed of sampling in the contractive residual flow is still extremely fast even though the compromise of computation of iterations (Chen et al., 2019).

3.1.4.3 Neural Network Architecture

The neural network architecture applied in the contractive residual flow model is essentially a standard ResNet architectures made invertible. This enpowers the model to perform classification, density estimation and generation, with the later two capacity unattainable in the standard architectures (Behrmann et al., 2019). As mentioned above, since the activation layers (ReLu) used in the standard ResNet is naturally contractive, we only need to enforce a normalisation step to the convolutional layers and fully-connected layers in the standard ResNet, in order to make it invertible. Specifically, these layers are made contractive by dividing with a constant strictly larger than their operator norm (Papamakarios et al., 2021). More detailed description of architectures is in Section 4.

It's worth noting that when constructing the f_θ transformation function, the capacity to inverting the transformation f_θ is of significant importance. The inverse transformation f^{-1} is used either in sampling or training purpose. If the inverse of f is not efficient, either density evaluation or sampling will be inefficient or even intractable.

3.2 Diffusion Models

The theory of the diffusion models (DMs) was first proposed in the 2015 paper Sohl-Dickstein et al. (2015) based on ideas from statistical thermodynamics. The model was further formalised and refined in papers (Song and Ermon, 2019; Ho et al., 2020). **It is built on research on flow-based generative models**.

Since diffusion models took over the title of the state-of-the-art generative model from GAN in 2020, numerous variations of the original model have been proposed to broaden the framework. We list three of the most paradigmatic models below:

- The denoising diffusion probabilistic models (DDPMs) (Ho et al., 2020), which was one of the earlies diffusion models that beat GAN in terms of image synthesis, boosted directly on the methodology of the original papaer (Sohl-Dickstein et al., 2015) – the theory of non-equilibrium thermodynamics.

- Song and Ermon (2019) put forward the noise conditioned score matching network (NCSMNs), where a neural network is trained to estimate the score function of the data distribution (rather than the log density) using the technique of score matching. After that, Annealed Langevin dynamics is applied to generate new images from noisy inputs. The authors further suggested to use stochastic differential equations (SDEs) to represent the forward and reverse process of the diffusion model, which gives rise to a more generalised formulation of the framework.

- To date, one of the most powerful and flexible diffusion models in image synthesis is the latent diffusion models (LDMs) (Rombach et al., 2022). It starts by first compressing the image from pixel space to a latent space with smaller dimensionality, therefore grasping a deeper semantic understanding of the images. By projecting the images to the latent space of pretrained autoencoders, the method achieves competitive performance on unconditional image generation compared to pixel-based diffusion models, in the meantime significantly reducing the computational cost. Additionally, high-resolution conditional synthesis is made possible by training the LDM on pairs of latent images data and conditioning texts.

In this section, we present the formulation of the denoising diffusion probabilistic models (DDPMs). We consider it to be the most representative and theoretically-founded model in the diffusion model family. Our review is mainly based on the works: Ho et al. (2020) and CVPR 2022 Tutorial on denoising diffusion model (Kreis et al., 2022). We focus on the model from the probabilistic machine learning perspective.

Denoising diffusion model is comprised of two processes. **(*i*)** A fixed *forward diffusion process* progressively adds noise to the input data, and **(*ii*)** a *reverse generative process* learns to recover the data by denoising. Specifically, this is done by training a U-Net that denoises the output from forward diffusion process to obtain a less noisy version of the image (Ronneberger et al., 2015).

The rest of the section is organised as follows: in Section 3.2.1, we present the forward diffusion process of DDPMs. The reverse generative process is introduced in Section 3.2.2. We give the maximum likelihood-based training objective in Section 3.2.3. Finally, we provide a brief introduction to the configurations of the neural network *i.e.* the U-Net used in the DDPMs (Ho et al., 2020) as well as some further improvements to the architecture in Section 3.2.5.

3.2.1 Forward Diffusion Process

Suppose our input data type is image. The forward process starts from the clean image and generates a sequence of noisy images by injecting noise at every time step. As illustrated in Figure 3.1, there are T time steps in the forward process, which gives rise to a sequence of noisy images $\mathbf{x}_t$, $t \in \{1, 2, ..., \mathrm{T}\}$. Under our assumption the forward process uses a Gaussian distribution to generate the noisy image at current time step t conditioned on the image at the previous time step $t-1$. The noise model has the form of a Gaussian kernel:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t;\, \sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\, \beta_t \mathbf{I}_m), \tag{3.14}$$

where $q(\mathbf{x}_t)$ represents the unknown *true* density of the image $\mathbf{x} \in \mathbb{R}^m$ at time step t (*e.g.* $q(\mathbf{x}_0)$ is the unknown true density of the clean image $\mathbf{x}_0 \in \mathbb{R}^m$). $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ takes the image $\mathbf{x}_{t-1}$ at the previous time step and generates the image $\mathbf{x}_t$ at the current time step. The noise distribution is a Gaussian distribution with respect to the image

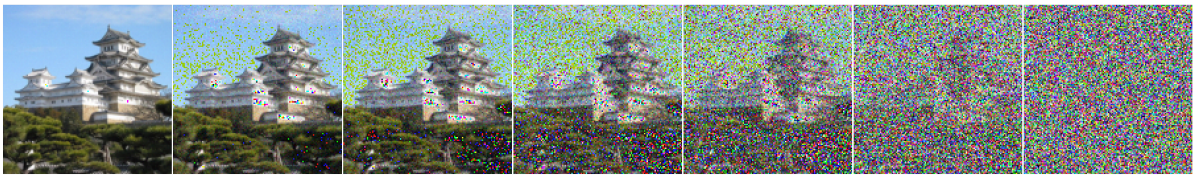

Figure 3.1: Example of intermediate noisy versions of a smaple image. The figure shows seven noisy images (Wikipedia, the free encyclopedia, 2002) sampled from image forward diffusion process with total time steps $T = 200$. The images are extracted from time steps $t \in \{0, 10, 20, 40, 60, 100, 199\}$.

$\mathbf{x}_t$ at the current time step. The mean of the distribution is $\sqrt{1-\beta_t}\,\mathbf{x}_{t-1}$. The variance of the distribution is $\beta_t \mathbf{I}_m$. $\mathbf{I}_m$ is $m \times m$ identity matrix.

In the original paper Sohl-Dickstein et al. (2015), the parameter β_t is termed as *noise schedule*. It is a positive scalar with very small magnitude. We can understand that the β_t performs two functions. First, the coefficient $\sqrt{1-\beta_t}$ in the mean scales down the image step by step. In other words, it reduces the magnitude of value of each pixel in the image at each time step. As T gets very large, the value of pixels in the image $\mathbf{x}_T$ will converge to zero. In the meantime, the β_t in the variance adds tiny bit of noise to the image at each time step. This two simultaneous effects on the image resembles the diffusion process of the image. Therefore, the process is also referred to as forward diffusion process.

We also define the joint distribution for all the noisy images generated in the forward diffusion process conditioned on the clean image $\mathbf{x}_0$:

$$q(\mathbf{x}_1, ..., \mathbf{x}_T|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \tag{3.15}$$

where the diffusion forward process is regarded as a Markov process that generates one noisy image at one time step given the previous time step. Hence, the equation 3.15 follows the property of a discrete-time markov chain.

Given the above formulation and the advantageous properties of Gaussian distribution, we are able to sample any intermediate noisy images directly from the forward process instead of starting from $\mathbf{x}_0$ and adding noise repetitively. First, we

define a new parameter α_t:

$$\alpha_t = \prod_{s=1}^{t}(1-\beta_s), \tag{3.16}$$

where we assume β_s is constant across all time steps. Then the marginal density of $\mathbf{x}_t$ given $\mathbf{x}_0$ can be derived analytically by manipulating the joint density 3.15 and leveraging the properties of Gaussian distribution:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t;\ \sqrt{\alpha_t}\mathbf{x}_0, (1-\alpha_t)\mathbf{I}_m). \tag{3.17}$$

Therefore the density of $\mathbf{x}_t$ given $\mathbf{x}_0$ is also a simple normal distribution with mean $\sqrt{\alpha_t}\mathbf{x}_0$ and variance $(1-\alpha_t)\mathbf{I}_m$. We term this Gaussian distribution as the *diffusion kernel* in contrast to the noise model. Once again, by using the property of a normal distribution, we apply the reparameterisation trick to sample the intermediate noisy image $\mathbf{x}_t$ from the conditional distribution:

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{(1-\alpha_t)}\,\boldsymbol{\epsilon}, \tag{3.18}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. As a result, given the input image $\mathbf{x}_0$ at time step zero we are able to generate any sample $\mathbf{x}_t$ at time step t.

As discussed above, if the noise schedule β_t is designed properly, the parameter α_T will converge to zero as T gets large. Equivalently, the density of $\mathbf{x}_T$ given $\mathbf{x}_0$ can be approximated by a standard normal distribution:

$$q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \boldsymbol{I}). \tag{3.19}$$

It's worth noting that the diffusion kernel $q(\mathbf{x}_t|\mathbf{x}_0)$ is different from the diffused data distribution $q(\mathbf{x}_t)$. Their relationship is as follows:

$$q(\mathbf{x}_t) = \int q(\mathbf{x}_0, \mathbf{x}_t)\, d\mathbf{x}_0 = \int q(\mathbf{x}_0)\, q(\mathbf{x}_t|\mathbf{x}_0)\, d\mathbf{x}_0 \tag{3.20}$$

where $q(\mathbf{x}_t)$ is the diffused data distribution, $q(\mathbf{x}_0)$ is the clean image density and $q(\mathbf{x}_t|\mathbf{x}_0)$ is the diffusion kernel. The diffusion kernel makes the distribution of $q(\mathbf{x}_t)$ become smoother and smoother as the forward process proceeds, as illustrated in figure 3.2. More importantly, the above formulation shows that as T gets very large, the final diffused data distribution $q(\mathbf{x}_T)$ converges to the diffusion kernel $q(\mathbf{x}_T|\mathbf{x}_0)$, which implies that we can use standard normal distribution to represent the diffused data distribution as well:

$$q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{3.21}$$

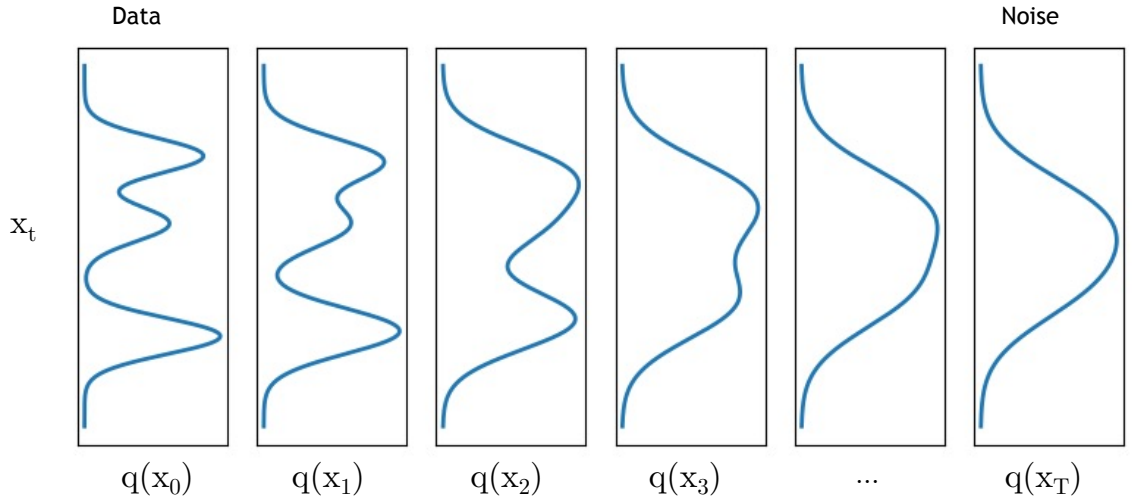

Figure 3.2: 1-D illustration of the diffused data distribution of $\mathbf{x}_t$ (Kreis et al., 2022). The diffusion kernel makes the distribution $q(\mathbf{x}_t)$ smoother and smoother along the Markov Chain. Further, $q(\mathbf{x}_t)$ converges to standard Gaussian distribution $q(\mathbf{x}_T)$ at time step T.

In some sense, we can understand this diffusion kernel as Gaussian convolution in the signal process, where it makes the input data distribution smoother at each time step.

To summarise, in the forward process, we don't have access to the clean image distribution $q(\mathbf{x}_0)$ nor the intermediate diffused data distribution $q(\mathbf{x}_t)$. However, by leveraging the simple form of the Gaussian distribution of the diffusion kernel $q(\mathbf{x}_t|\mathbf{x}_0)$ and known samples $\mathbf{x}_0$ from the distribution $q(\mathbf{x}_0)$, we are able to effectively draw samples $\mathbf{x}_t$ from the diffused data distribution $q(\mathbf{x}_t)$.

3.2.2 Reverse Generative Process

So far, we have introduced the forward diffusion process and how we can sample intermediate noisy images from the forward process. Built upon that, in this section, we present the reverse generative process. Specifically, we introduce how to generate images from this process.

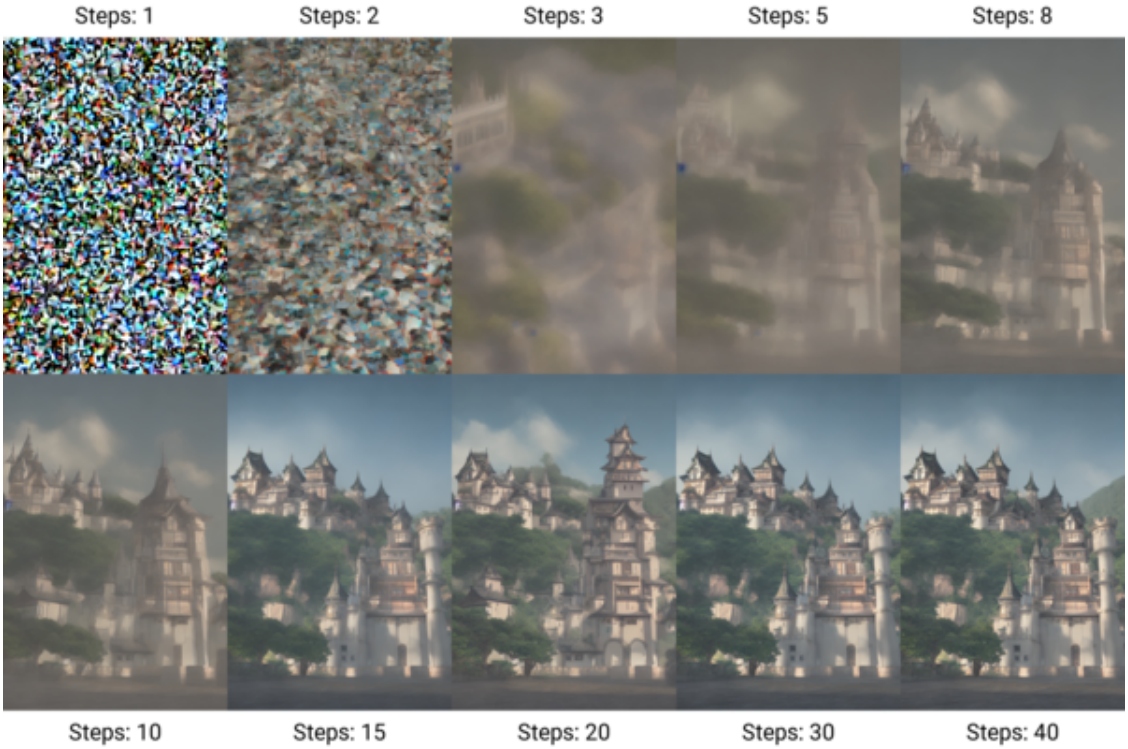

Figure 3.3: Reverse generative process with total time steps $T = 40$. The image at Steps: 1 (referred to as $\mathbf{x}_T$ in this essay) is simply the standard Gaussion white noise following distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The Figure is taken from the source (Wikipedia, the free encyclopedia, 2022), created using Stable Diffusion v1-5 diffusion model (Rombach et al., 2022).

In the reverse process, as shown in the Figure 3.3, we start from the final diffused data distribution $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \boldsymbol{I})$ and then go bardwards in the diffusion process. We call this denoising process. In other words, we first sample from the final time step $q(\mathbf{x}_T)$. Then, by following the same technique in the forward process, we can effectively generate samples from intermediate diffused data distribution $q(\mathbf{x}_t)$ once the *true denoising distribution* $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is known.

However, generally the denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ (Bayes theorem) is intractable since the $q(\mathbf{x}_{t-1})$ has been unknown to us. Thus, we have to approximate the denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Fortunately, experiments show that if the noise schedule β_t in the Gaussian kernel 3.14 is small, the denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ can be effectively approximated by a Gaussian distribution:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1};\, \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \boldsymbol{I}), \tag{3.22}$$

where $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is our density *denoising model* with parameter θ, which is an

approximate of the true denoising distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. The mean of the Gaussian distribution $\mu_\theta(\mathbf{x}_t, t)$ is **the core of our denoising model**. Specifically, it is a parametric model which takes in $\mathbf{x}_t$ and learns to predict the mean of the less noisy image $\mathbf{x}_{t-1}$ at the previous time step conditioned on $\mathbf{x}_t$. There are many ways to define this parametric model. In our approach, we let the mean $\mu_\theta(\mathbf{x}_t, t)$ be a deep neural network in order to fully leverage the deep learning revolution. For the parameter σ_t^2 in the variance, we simply set it to be a scalar. Moreover, similar to the equation 3.15 in the forward process, through the property of a Markov process, we can show that the joint distribution of our denoising model follows that:

$$p_\theta(\mathbf{x}_0, ..., \mathbf{x}_T) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t). \tag{3.23}$$

In the paper Ho et al. (2020), the authors proposed a novel approach to train a deep neural network to compute $\mu_\theta(\mathbf{x}_t, t)$ using reparameterisation. Instead of training a model to approximate the mean of $\mathbf{x}_{t-1}$ given $\mathbf{x}_t$ directly, they choose to train a model that approximates the noise $\boldsymbol{\epsilon}$ used in the forward process to generate $\mathbf{x}_t$. Recall that $\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{(1-\alpha_t)}\,\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. As a result, the mean of $\mathbf{x}_{t-1}$ given $\mathbf{x}_t$ can be reparameterised as:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right), \tag{3.24}$$

where parameter $\boldsymbol{\epsilon}_{\theta(\mathbf{x}_t,t)}$ is a *noise-prediction* network that learns to approximate the truth noise $\boldsymbol{\epsilon}$ used to generate $\mathbf{x}_t$. The coefficient $\beta_t/\sqrt{1-\alpha_t}$ rescales the $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to be the tiny bit of noise added to $\mathbf{x}_{t-1}$ in order to generate $\mathbf{x}_t$. Thus, we can simply subtract $(\beta_t/\sqrt{1-\alpha_t}) \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ from $\mathbf{x}_t$ to get back to $\mathbf{x}_{t-1}$.

The idea of learning the difference between two time steps or two layers achieves significant gains in performance not only in the diffusion model but in many other areas of deep learning as well. In fact, one of the most famous applications of the idea is the 2015 ImageNet winning architecture ResNet (He et al., 2016), where the network is trained to learn the residual links between layers instead of the direct mapping. This approach successfully tackled the optimisation difficulty in the deep neural networks and allowed the construction of very deep neural networks.

Assume our learnt noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is in place, we can easily derive the density denoising model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Therefore, given the samples drawn from the final

diffused data distribution $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \boldsymbol{I})$, we can iteratively go backwards in the diffusion process from the final time step T to the starting time step 0, and generate samples from the clean data distribution $q(\mathbf{x}_0)$. The detailed sampling procedure will be presented in section 3.2.4. Here we first give a brief formulation of the approach. For $t = T, ..., 1$:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \boldsymbol{z}, \tag{3.25}$$

where $\boldsymbol{z} \approx \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. This is similar to the reparameterisation trick we've seen in the forward diffusion process. First, we sample $\mathbf{x}_T$ from final diffused data distribution $\mathcal{N}(\mathbf{x}_T; \mathbf{0}, \boldsymbol{I})$. Then at each time step t, we add noise to the $\mu_\theta(\mathbf{x}_t, t)$ rescaled by the standard deviation σ_t. By repeating this operation T times, we can generate samples from the clean image distribution $q(\mathbf{x}_0)$.

In order to generate samples close to the clean image distribution $q(\mathbf{x}_0)$, one natural intuition is to approximate the true noise $\boldsymbol{\epsilon}$ used in the diffusion process as accurately as possible, which gives us the first loss function for training (objective function):

$$\text{Minimise } \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2. \tag{3.26}$$

Therefore, we aim to minimise the L^2 distance between the true noise $\boldsymbol{\epsilon}$ and our trained parameterised noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$.

3.2.3 Maximum Likelihood-based Training Objective

In this section, we look at the training objective from the perspective of maximum likelihood. We show that the training objective with regard to noise presented in section 3.2.2 is equivalent to maximise a weighted log likelihood of data under our marginal model density with respect to the clean data distribution.

In general, in the context of maximum likelihood-based training, we want to minimise the objective function with such form:

$$\mathbb{E}_{q(\mathbf{x}_0)}\big[-\log p_\theta(\mathbf{x}_0)\big], \tag{3.27}$$

where the $-\log p_\theta(\mathbf{x}_0)$ is the negative log likelihood of our input image $\mathbf{x}_0$ under the marginal density of our parameterised denoising model $p_\theta(\cdot)$. Unfortunately, due to the intractability of the marginal density model $p_\theta(\mathbf{x}_0)$, we have to apply the

variational upper bound commonly seen in variational autoencoders for training:

$$\mathbb{E}_{q(\mathbf{x}_0)}\big[-\log p_\theta(\mathbf{x}_0)\big] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_1,\ldots,\mathbf{x}_T|\mathbf{x}_0)}\left[-\log\frac{p_\theta(\mathbf{x}_0,\ldots,\mathbf{x}_T)}{q(\mathbf{x}_1,\ldots,\mathbf{x}_T|\mathbf{x}_0)}\right] =: L, \tag{3.28}$$

where the $q(\mathbf{x}_0)$ is the clean image distribution, $q(\mathbf{x}_1,\ldots,\mathbf{x}_T|\mathbf{x}_0)$ is the joint distribution of forward process and $p_\theta(\mathbf{x}_0,\ldots,\mathbf{x}_T)$ is the joint distribution of reverse process given by the denoising density model.

It can be further shown that the variational upper bound L can be decomposed into three components:

$$L = \mathbb{E}_q\Bigg[\underbrace{D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)\|p(\mathbf{x}_T))}_{L_1} + \sum_{t>1}\underbrace{D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_2}\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_3}\Bigg], \tag{3.29}$$

where L_1 term is the KL divergence between $q(\mathbf{x}_T|\mathbf{x}_0)$ and $p(\mathbf{x}_T)$. Since both of them follow the standard Gaussian distribution, L_1 is simply zero. Furthermore, L_3 is the negative log likelihood of the denoising model $p_\theta(\mathbf{x}_0|\mathbf{x}_1)$ of the last step of the denoising process. Specifically, if parameter β_t is very small and the time steps T is very large, it's easy to show that the L_3 term can be regarded as a constant with respect to our model parameter θ. As a result, in order to minimise L, we only need to consider the term L_2.

The L_2 term is the KL divergence between $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, where $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ is a tractable inverse (posterior) distribution conditioned on $\mathbf{x}_t,\mathbf{x}_0$ with density:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1};\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0),\tilde{\beta}_t\boldsymbol{I}), \tag{3.30}$$

where $\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\alpha_{t-1}}{1-\alpha_t}\beta_t$.
In contrast to $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, Equation 3.30 shows that given both the clean image $\mathbf{x}_0$ and the noisy image $\mathbf{x}_t$ at the current time step, we will be able to infer the distribution of the less noisy image $\mathbf{x}_{t-1}$ at the previous time step.

Now that we have access to both $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, we can compute the KL divergence of these two models. Note that both distributions follow the Gaussian distribution. Therefore, the KL divergence simplifies to:

$$L_2 = D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}\|\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) - \mu_\theta(\mathbf{x}_t,t)\|^2\right] + C. \tag{3.31}$$

Algorithm 1 Diffusion Model Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathcal{U}(1, ..., T)$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
5: Take gradient descent step on
$\nabla_\theta \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, t)\|^2$
6: **until** converged

where $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$ is the mean of the inverse data distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$, $\mu_\theta(\mathbf{x}_t, t)$ is the mean of our denoising model distribution. As shown in Section 3.2.2, given the samples $\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{(1-\alpha_t)}\,\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, we can easily derive the mean of the inverse data distribution:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}\right). \tag{3.32}$$

Moreover, the mean of the denoising model is:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right). \tag{3.33}$$

As discussed in Section 3.2.2, we could have directly trained a neural network $\mu_\theta(\mathbf{x}_t, t)$ to predict the ground-truth mean $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$, which we refer to as *direct mapping*. However, predicting the intermediate link – true noise $\boldsymbol{\epsilon}$ (*indirect mapping* proves to give us much better result (Ho et al., 2020).

By substituting Equation 3.32 and Equation 3.33 into L_2 term, we get:

$$L_2 = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{x}_0, \boldsymbol{I})}\left[\underbrace{\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\alpha_t)}}_{\lambda_t}\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\underbrace{\sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}}_{\mathbf{x}_t}, t)\|^2\right] + C. \tag{3.34}$$

Therefore, we can show that the maximum likelihood-based training objective boils down to minimise Equation 3.34. Specifically, the λ_t term makes sure that the training objective L_2 is weighted appropriately to align with the initial maximum likelihood training objective $\mathbb{E}_{q(\mathbf{x}_0)}\big[-\log p_\theta(\mathbf{x}_0)\big]$.

Nevertheless, the time dependent λ_t is monotonically decreasing with respect to t. Since β_t is designed to be very small, λ tend to become very large for small t and likewise very small for large t, which is adversarial for the training of our model.

Therefore, based on empirical heuristics, the authors in the paper Ho et al. (2020) simply set λ_t to be 1 and refer to the training objective as L_{simple}:

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), t \sim \mathcal{U}(1,T)} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, t)\|^2, \tag{3.35}$$

where $t \sim \mathcal{U}(1, T)$ follows a uniform distribution on $\{1, ..., T\}$. Interestingly, the authors observe better results from the model trained on the weighted objective function L_{simple} than on the original objective L_2. However, it still remains as an open question regarding why a model trained on weighted maximum data likelihood outperform their pure maximum likelihood-based counterparts.

To summarise, we present the training algorithm for L_{simple} adopted by Ho et al. (2020) in Algorithm 1. At each iteration, we first sample clean image $\mathbf{x}_0$ from true data distribution $q(\mathbf{x}_0)$. Then we sample a particular time step t from the uniform distribution $\mathcal{U}(1, ..., T)$. Given time step t and clean image $\mathbf{x}_0$, we compute the noisy image $\mathbf{x}_t$. After that, we feed both $\mathbf{x}_t$ and $\mathbf{x}_0$ into the neural network to predict the true noise $\boldsymbol{\epsilon}$. Finally gradient descent is utilised to update the parameters. We repeat the above process until the convergence of the parameters.

3.2.4 Sampling Approach

In this section, we recapitulate the sampling procedure mentioned in Section 3.2.2 in Algorithm 2.

In order to generate image from the reverse denoising process, we start from the last time step T, where we sample $\mathbf{x}_T$ from the standard normal distribution. Then we walk back the diffusion process from that sample by forming a for loop from T to 1. For each iteration within the for loop, we first **(*i*)** subtract the rescaled predicted noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ from the image at the current time step to get the mean of the denoising model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, then **(*ii*)** add the standard deviation σ_t of the denoising model to derive the less noisy image $\mathbf{x}_{t-1}$ at the previous time step. By following the for loop step by step, we are able to generate clean images $\mathbf{x}_0$ from true data distribution $q(\mathbf{x}_0)$ at time step $t = 0$.

Algorithm 2 Diffusion Model Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
2: **for** $t = T, ..., 1$ **do**
3: $\quad z \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \boldsymbol{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

3.2.5 Neural Network Architecture

So far, we have looked at the foward process, the reverse process, training objective as well as the sampling procedure. In this section, we introduce the neural network architecture used to train our noise model $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$. Specifically, we focus on the techniques that incorporate the timestep embedding t into the neural network. Furthermore, we present the formulations of two crutial hyperparameters – variance β_t in the forward process and variance σ_t^2 in the backward process.

The rest of the section is organised as follows: in Section 3.2.5.1, we present the overview of the U-Net architecture as well as the design of key hyperparameters. The time embedding learning approach is discussed in Section 3.2.5.2.

3.2.5.1 U-Net

Ho et al. (2020) proposed the U-Net architecture for DDPMs, which was shown to significantly enhance the quality of generated samples compared to previous achitectures. Dhariwal and Nichol (2021) further improved the U-Net design by performing several ablations of the model architecture. We provide a brief overview of both original and refined architectures in this section. In addition, we introduce the design of two key hyperparameters β_t and σ_t^2 of the neural network.

The U-Net, a neural network architecture widely applied in medical image segmentation, was first proposed in the paper Ronneberger et al. (2015). It employs an encoder-decoder architecture that resembles an autoencoder, but with a crucial addition: the skip connection. Specifically, the architecture consists of a downsampling path which captures high-level representations of the input data, and a symmetric upsampling path that recovers the latent projection back to the original spacial size.

The skip connections between two paths are used to combine the high-level representations from the upsampling path with corresponding low-level representations from the downsampling path. It is shown that the U-Net is capable of seize rich spacial and position information of images, while maintaining computational efficiency (Huang et al., 2020).

The 32×32 resolution U-Net model in DDPMs uses four feature map resolutions that decay sequentially, including 32×32, 16×16, 8×8 and 4×4. At each resolution level, the model uses two convolutional residual blocks (He et al., 2016), where the group normalisation is applied as the normalisation layer (Wu and He, 2018). In addition, a self-attention block using a single head is added at 16×16 resolution to better take account of the overall semantic information of the image (Dosovitskiy et al., 2020). The timestep information t is learnt by the network through injecting the Transformer sinusoidal position embedding into every residual block.

Aside from the architecture of the neural network, hyperparameters of the diffusion model are also chosen to improve the sample quality in DDPMs. Among them, the total time steps T is set to be 1000. The variance β_t in the forward process is specified to increase linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$. Furthermore, the variance σ_t^2 in the reverse process is fixed to be equal to the β_t.

3.2.5.2 Time Embedding Learning

We discuss the time embedding methodologies for the U-Net in this section. Learning the time information of data is a well-studied topic in statistical learning models. Among them, time-aware neural networks are of particular appeal to data scientists.

In DDPMs, the authors applied the same approach as the positional embedding in Transformer, where the time information of the input data is embedded in the trigonometric functions, as shown in figure 3.4. The method was first developed to solve the problem of learning the relative position of word in a long sentence. It is capable of dealing with the sentences with variable lengths as it would become very hard to normalise them in a uniform scale. For its application in diffusion model, each image in the batch is assigned with a particular sinusoidal curve, where its

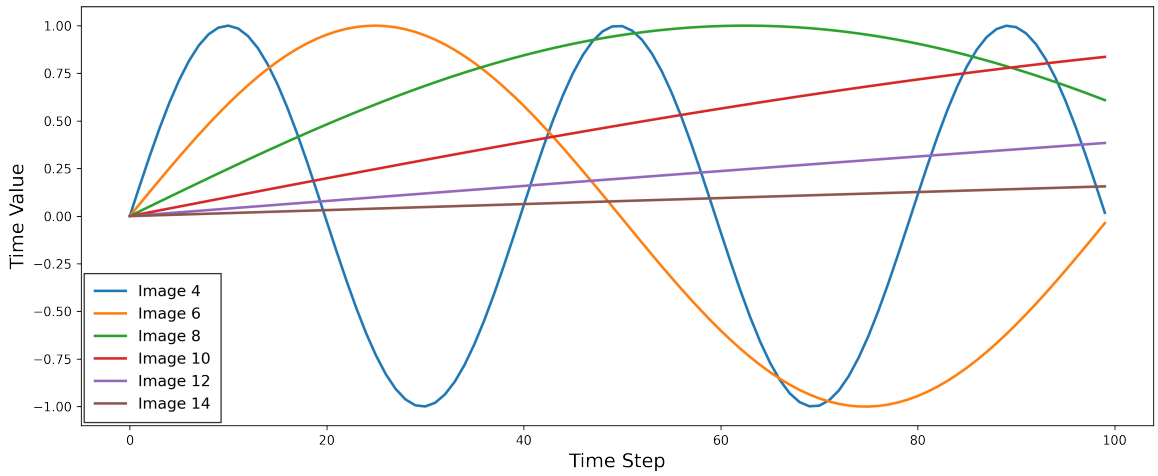

Figure 3.4: Examples of sinusoidal position embeddings used in Transformer and diffusion model. Each curve represents the state information of the Markov Chain for one specific image of index $4, 6, 8, 10, 12, 14$ in the mini-batch. By varying the period of the trigonometric functions, the method is able to efficiently encode the time information.

replica with different noise level correspond to differnt value in the curve.

Despite the effectiveness achieved by sinusoidal position embedding in encoding the timestep information, there is actually a method that obtains much better result in terms of learning the time information for the neural network. The method is proposed in the paper Dhariwal and Nichol (2021) in order to improve the diffusion model performance. It is termed as Adaptive Group Normalisation, where the sinusoidal position embedding is replaced by group normalisation in each residual block. Specifically, the image data after the first convolution layer is scaled and shifted by the time information t:

$$\text{AdaGN}(y_s, y_b) = y_s \text{GroupNorm}(h) + y_b,$$

where h is the output of the first convolution layer in the residual block, and $y = [y_s, y_b]$ is computed from a linear transformation of time steps. In other words, the intermediate output is adapted by the time information while performing the group normalisating. In some sense, the method serves as an enhanced version of group normalisation. This approach is shown to give a substantial boost to the sample quality than the addition plus normalisation method used in the original architecture. The comparison of these two approaches provides us with an useful insight when

designing the time learning methods in statistical models. First, encoding time using periodic functions, such as sinusoidal function, gives an efficient way of learning data with variable time step lengths. Further, when the time step length is sufficiently large and stay constant, time-adapted normalisation can achieve better learning outcome.

Chapter 4

Results

In this chapter, several experiments are conducted to demonstrate the effectiveness of the normalising flow and diffusion model. Our experiments employ the normalising flow, specifically contractive residual flow (CRF) based on the implementation (Chen et al., 2020) provided by Chen et al. (2019). Besides, We implement the U-Net as well as the diffusion model (DDPM) (Ho et al., 2020) based on Varuna Jayasiri (2020). For the convenience of distinguishing different diffusion models, we refer to the original DDPM in Ho et al. (2020) as *Large DDPM*, and the diffusion model implemented in our experiments is termed as DDPM, due to the difference in the number of parameters. The computer code and relevant references are available at Appendix A for reproduction. All the experiments are employed in Python 3 and run on a Linux machine with 2 P100 GPUs, 32GB GPU memory.

The rest of the chapter is organised as follows. We present a brief introduction of the datasets used in our experiments in Section 4.1. Section 4.2 introduces the evaluation metrics of the generative models. In Section 4.3, we give detailed configurations of the normalising flow model and diffusion model depolyed in the experiments. The generated images with sample quality evaluation results are detailed in Section 4.4.

4.1 Data Sets

We experiment the normalising flow and diffusion model on MNIST (LeCun et al., 1998) and CIFAR10 datasets (Krizhevsky and Hinton, 2009). A brief description of

Table 4.1: Properties of the training datasets used in the experiments.

Data Set	Total instances	Training instances	Colour channels	Pixel format	Classes
MNIST	70,000	60,000	1	28×28	10
CIFAR10	60,000	50,000	3	32×32	10

the datasets is listed below with some important statistics in Table 4.1:

MNIST A dataset of handwritten digits containing 60,000 training images and 10,000 testing images, both sampled from the same data distribution. There are 10 classes of digits, *0, 1, 2, 3, 4, 5, 6, 7, 8* and *9*. Each image is of 28×28 pixel format. Therefore, each sample image is a vector of $28 \times 28 = 784$ dimensionality. This is an entry-level dataset for testing the machine learning methods. Given its simplicity and no demanding efforts on preprosessing, it is one of the most widely-used datasets in the machine learning literature. However, due to its small capacity and limited complexity, the methods that are effective for this dataset often suffer from scaling issues on more complicated tasks.

CIFAR10 A dataset of colour images containing 50,000 training images and 10,000 testing images. It has 10 different classes with 6,000 images for each class. Specifically, the 10 classes are *airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships*, and *trucks*. Each image is of 32×32 pixel format with 3 colour channels. The CIFAR10 dataset is specifically built for training image generative models. It is a labeled subset of 80 million tiny images dataset collected from the WorldNet. Compared to the MNIST dataset, it is a more challenging dataset to learn interesting features since the geometry of objects are much more complicated and the dataset requires more preprocessing and formatting.

For both datasets, we use training images as training data for our generative model. Specifically, we use 60,000 training images in MNIST as training data, and 50,000 training images in CIFAR10 as training data. In addition, we use the 32×32 pixel format as the uniform unit for image processing. Therefore, MNIST images are resized to 32×32 at the beginning of the training. Finally, all images are normalised to have values in the $[0, 1]$.

4.2 Evaluation Metrics

For evaluation of the generative models, we use **Fréchet inception distance** (FID) (Heusel et al., 2017) and **Inception Score** (IS) (Salimans et al., 2016) to measure the quality of the generated images.

The FID metric was introduced in Heusel et al. (2017) to characterise the convergence of GAN training, and capture the similarity of generated images to real ones. It is the current standard metric for evaluating the performance of a generative models. Given two probability distributions s, s' over $\mathbb{R}^n$ having finite mean and variances, FID fits two multivariate Gaussian distributions $\mathcal{N}(\mu, \Sigma)$, $\mathcal{N}(\mu', \Sigma')$, respectively for s, s'. The Fréchet distance is then defined as follows:

$$d_F(\mathcal{N}(\mu, \Sigma), \mathcal{N}(\mu', \Sigma'))^2 = \|\mu - \mu'\|_2^2 + \mathrm{tr}\left(\Sigma + \Sigma' - 2\left(\Sigma^{\frac{1}{2}} \cdot \Sigma' \cdot \Sigma^{\frac{1}{2}}\right)^{\frac{1}{2}}\right),$$

where $\mathcal{N}(\mu, \Sigma)$ is estimated from Inception v3 model (without its final classification layer) trained on the true dataset, and $\mathcal{N}(\mu', \Sigma')$ is obtained from Inception v3 based on the generated images. The FID compares the true dataset and generated images based on the distribution of the deepest layer in Inception v3. These layers stand for the high-level representations for images. A lower FID is interpreted as 'better', as it means that the distribution of generated images is closer to that of the true dataset.

Despite the consistency of FID with human perception, there are cases where FID fails to evaluate the quality of generative models unbiasedly (Liu et al., 2018). Therefore, we apply another metric Inception score for evaluating the generative models (Salimans et al., 2016). The score is derived based on the Inception v3 model as well. Specifically, Inception v3 is applied to a large number of generated images. Given two spaces, the image space Ω_X and the label space Ω_Y. The space of labels is finite. The Inception score is defined as:

$$\mathrm{IS} = \exp\left(\mathbb{E}_{\mathbf{x}}\left[D_{KL}\left(p(\boldsymbol{y}|\mathbf{x}) \| p(\boldsymbol{y})\right)\right]\right),$$

where the conditional distribution $p(\boldsymbol{y}|\mathbf{x})$ represents the probability of the image $\mathbf{x}$ has label $\boldsymbol{y}$ according to the Inception v3. The marginal distribution $p(\boldsymbol{y})$ is the distribution of the whole image space Ω_X on Ω_Y. In practice, it is computed empirically by summing up the conditional distribution $p(\boldsymbol{y}|\mathbf{x})$. A higher IS means that the

generated images are 'sharp' and 'diverse'.

4.3 Experimental details

4.3.1 Normalising Flow

We apply the contractive residual flow model proposed in Chen et al. (2019) without modification. Firstly, the input data is passed through a logit transform. After this, we process the data (Devlin et al., 2018) through a series of residual blocks and squeeze layers. Activation normalisation (Kingma and Dhariwal, 2018) is used before and after every residual blocks. Each residual connection in the residual blocks is comprised of Chen et al. (2019):

$$\text{LipSwish} \rightarrow 3\times3\text{Conv} \rightarrow \text{LipSwish} \rightarrow 1\times1\text{Conv} \rightarrow \text{LipSwish} \rightarrow 3\times3\text{Conv}.$$

For both the MNIST and CIFAR10 datasets, we use the following architectures:

$$\text{Input Data} \rightarrow \text{LogitTransform}(\alpha) \rightarrow 16\times\text{ResBlock} \rightarrow \big[\text{Squeeze} \rightarrow 16\times\text{ResBlock}\big]\times2,$$

where the α is $1e-5$ and 0.05 respectively for the MNIST and CIFAR10 dataset.

The contractive residual flow models f_θ are configured to possess $T = 198$ nonlinear transformations (layers). There are 48 residual blocks in the architecture. The models are trained with batch size of 64, and channel size of 128. Furthermore, the Lipschitz constant L is configured to be 0.98.

Our MNIST model has 16 million parameters and our CIFAR10 model has 25 million parameters. In addition, in the training process, the MNIST model can be fit on a single GPU, while CIFAR10 model needs two GPU to satisfy the memory consumption.

According to Chen et al. (2019), contractive residual flow models converge in roughly $300-350$ epochs for MNIST and CIFAR10 datasets. Our MNIST model trains at 1.98 seconds per batch of 64 images (31 minutes per epoch), and sampling 128 images takes 0.77 seconds. The CIFAR10 model trains at 3.56 seconds per batch (56 minutes per epoch). Theoretically, it takes 259 hours (15 days) to train the MNIST models for 500 epochs, and 502 hours (21 days) to train the CIFAR10

model. Therefore, we only apply the pretrained models (500 epochs) on MNIST and CIFAR10 datasets in our experiments.

4.3.2 Diffusion Model

We adapt the U-Net used in the Large DDPM (Ho et al., 2020) as the neutral network architecture in our experiments. In this section, we give experimental details of the model architectures. For more information of the U-Net model, please refer to 3.2.5.

Due to limited GPU memory and computational time, our models have significantly fewer parameters, equivalently, less capacity compared to Large DDPM model. However, by adopting two important architecture improvements, our U-Net models are able to compensate the loss in model size and achieve impressive results in MNIST data generation tasks. Specifically, **(*i*)** in order to improve the learning of timestep information in the training data, our neutral network architecture uses the layer of AdaGan (Dhariwal and Nichol, 2021) (see Section 3.2.5.2) in place of the addition of sinusoidal timestep embedding adopted in Large DDPM. In addition, **(*ii*)** the Large DDPM U-Net uses a self-attention layer with a single head at the 16×16 resolution level. We reinforce the attention design by applying a two-head self-attention layer. Dhariwal and Nichol (2021) shows that multi-head self-attention is able to better capture the spatial information of the input data while only increasing the training time slightly due to the advantage of parallal computing.

We list the architecture and training parameters of our U-Net model in Table 4.2 and Table 4.3, in comparison of Large DDPM U-Net model. Our MNIST model has 5.7 million parameters, while CIFAR10 model has 8.9 million parameters. We used two P100 GPUs for the training of all experiments. Our MNIST model trains at 8.1 iterations per second, and CIFAR model trains at 4.5 iterations per second. In addition, We use one P100 GPU for sampling. Our MNIST model takes 92.6 seconds to sample a batch of 128 images, and sampling the same number of images takes 158.1 seconds for the CIFAR10 model.

We employ the implementation of U-Net as well as the diffusion model based on Varuna Jayasiri (2020).

Table 4.2: Comparison between our U-Net and Large DDPM U-Net in terms of architecture.

Architecture	Our U-Net	Large DDPM U-Net
Resolution levels	32, 16, 8, 4	32, 16, 8, 4
Residual blocks per resolution	2	2
Base channels	64	128
Channel multiplier	1, 2, 2, 4	1, 2, 3, 4
Attention at resolutions	16	16
Attention heads	2	1
β_t schedule	linear from 10^{-4} to 10^{-2}	linear from 10^{-4} to 10^{-2}
Time embedding	Adaptive group normalisation	Sinusoidal positional embedding
Parameters (millions)	5.7 (MNIST)/ 8.9 (CIFAR)	35.7 (CIFAR)

Table 4.3: Comparison between our U-Net and Large DDPM U-Net in terms of training parameters for CIFAR10 dataset.

Training Parameters	Our Model	Large DDPM
Batch size	128	128
Number of timesteps	1000	1000
Learning rate	2×10^{-3}	2×10^{-4}
Epochs	500	1500
Iterations per second	4.5	21
Training time (hours)	8.3	10.6

4.4 Samples and Evaluation

In this section, we present the results of images generated by normalising flow and diffusion model on MNIST and CIFAR10 dataset. In addition, we evaluate the sample quality of the images generated by two models using FID and Inception score.

4.4.1 MNIST Samples

We experiment on the CRF and DDPM trained on MNIST dataset both for 500 epochs. We present the unconditionally generated MNIST images by normalising flow (CRF) in Figure 4.1. As we can observe from the figure, some generated digit images can be well classified, for example, several 1, 3, 6 displaying distinct geometric structure. However, the CRF model still gets confused with some of the digit classes, such as 0

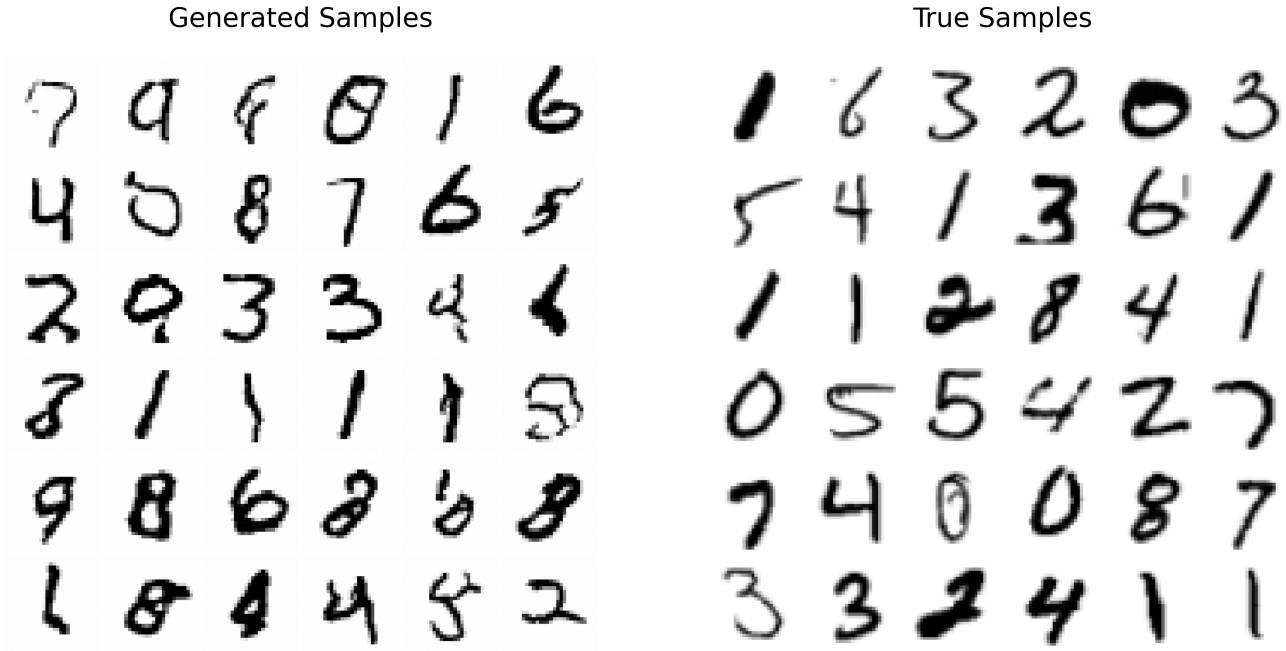

Figure 4.1: MNIST images unconditionally generated by the **normalising flow (CRF)**. The images on the left column are generated samples, and images on the right column are true samples drawn from the MNIST dataset.

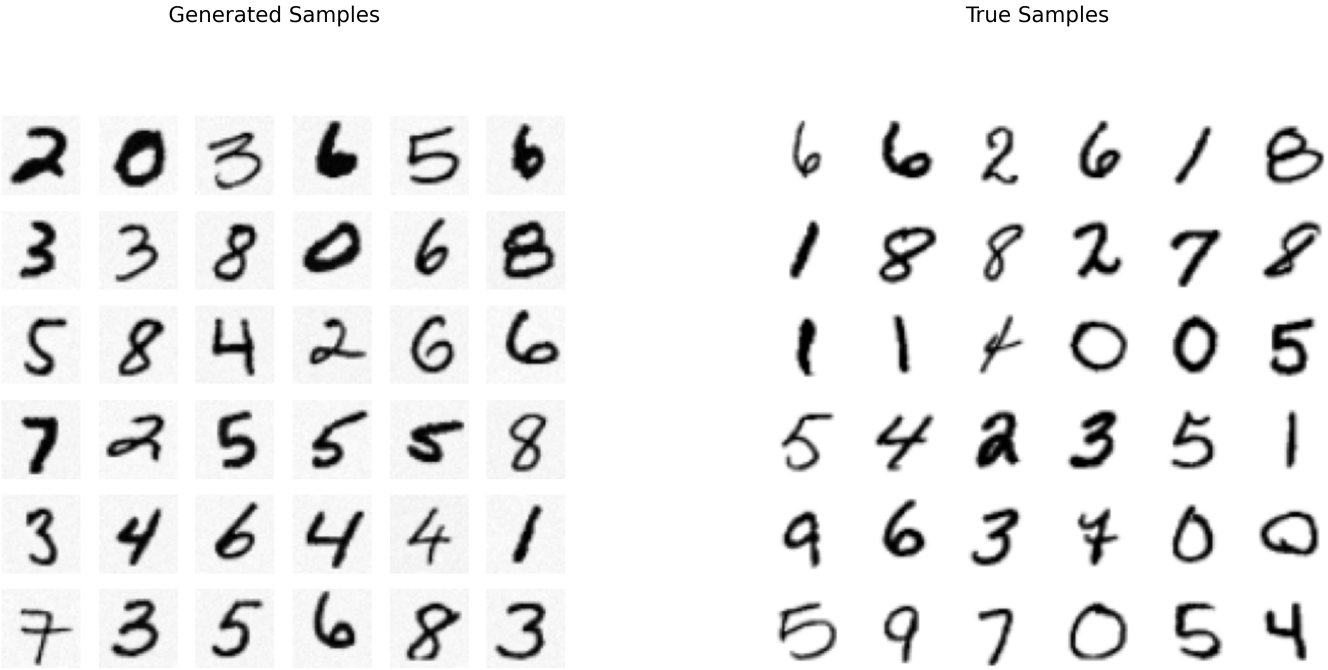

Figure 4.2: MNIST images unconditionally generated by **diffusion model (DDPM)**. The images on the left column are generated samples, and images on the right column are true samples drawn from the MNIST dataset.

Table 4.4: Inception score and FID of the MNIST images generated by the normalising flow and diffusion model. The lower FID is interpreted as better, and the higher IS is explained as superior. (For reference, the $10,000$ true images sampled from MNIST obtained FID of 0, Inception score of 2.004).

Evaluation Metrics (MNIST)	Normalising Flow	Diffusion Model
FID	20.55	**12.62**
Inception score	**2.07**	1.88

with 8, 3 with 8 and 3 with 9, which illustrates the fact that the model only manages to learn a portion of the class embedding of the 10 digits.

For DDPM, the results of the unconditionally generated images are shown in Figure 4.2. Compared with the true samples drawn from MNIST dataset, the generated images exhibit very alike statistical structure, as well as diverse categories. Further, each class of the digit shown in the image is distinct from the others. It is almost unlikely to misclassify one class of the digit to another class. This is equivalent to the fact that DDPM successfully learns the class embedding of the 10 digits in MNIST dataset. diversity of generated images is validated as we use VGG5 to classify $10,000$ images generated by the diffusion model. The distribution of the images over the label space is very close to a discrete uniform distribution $\mathcal{U}\{0,9\}$, i.e. any generated image $\mathbf{x}$ has equal probability $(1/N = 10)$ to be one of the classes in the label space. However, we notice that there is still a small amount of noise remaining in the generated samples, even though the images can be distinctly classified. We believe that this problem can be tackled either by increasing the training epochs, or by further processing the generated images with some denoising tools.

From our perspective, the mixing of digit classes of the normalising flow is attributed to the fact that the transformation in flow-based models is trained as a whole. Equivalently, the model is only trained with the true images. This gives the advantage of fast sampling. Nevertheless, the model cannot get enough knowledge of the intermediate variations of the semantics of the images, therefore cannot very well guide the transformation of images through the generation process. On the side of diffusion model, the model is fed with images in the different steps of the transformation, hence is able to capture the semantic variation of the transformation. Also, the inverse transformation is performed step by step, allowing the model to

guide the generation of images to corresponding classes.

Besides the generated images of digits, we present the FID and Inception Score of the normalising flow and diffusion model on MNIST in Table 4.4. We convert the grey-scale MNIST images to RGB images by replicating the single colour channel three times. For each value of the metrics presented in the table, we repeat the sampling 10 times, and take the average value.

As demonstrated in the table, the FIDs of the diffusion model (12.62) is much lower than that of the normalising flow (20.55), which is consistent with what we can intuitively observe from the genrated images. The images in 4.2 shows similar geometric shape and distinct classes than the images in 4.1 when compared to the true samples. However, there is still a huge gap in FID (12.617) between the distribution of generated MNIST images and true MNIST images. We infer the reason may well be the noise in the background of the generated images which deviates the distribution of generated samples from the true samples.

With regard to the Inception score, the images generated by the normalising flow strikingly achieves better score (2.007) than the true samples (2.004), though the difference is very small. On the one hand, it shows that the normalising flow generates more diverse and distinct images than the diffusion model (1.88) in some aspects. On the other hand, it demonstrates that the Inception score can sometimes be inconsistent with the human perception and pratical scenarios, as it only reflects the perception of the pre-trained Inception neural network and suggests the diversity and sharpness of the generated images from the standpoint of a classifier.

Compared to the normalising flow, one major advantage of the diffusion model as mentioned earlier is that the generation process can be guided easily using a classifier. Since the generation process is composed of many small transformations, the diffusion model can be guided by a classifier to generate specific classes of images. This gives the diffusion model a unique capability to generate conditional images without making big modifications to the model.

We show in Figure 4.3 the MNIST images generated using the classifier guidance by the diffusion model. Specifically, the classifier guidance is a type of conditional image generation $p(\mathbf{x}|y)$, where conditional on the label y, the generator is guided by a saparate traianed classifier to generate the image $\mathbf{x}$ with label y in the sampling

Figure 4.3: Conditionally generated MNIST images using classifier guidance by the **diffusion model**. The images are configured to generate with specific labels.

process. It is shown in Rombach et al. (2022) that the classifier guidance mechanism can be flexibly incorporated into the diffusion model to generate targeted images, which has become one of the major appeals of the diffusion model.

4.4.2 CIFAR10 Samples

In order to test the boundary of our generative models, we move on to train the normalising flow (CRF) and diffusion model (DDPM) on the CIFAR10 dataset, which is significantly more challenging to generate since it has three color channels instead of one. We keep the basic configurations of diffusion model unchanged apart from increasing the colour channels of the residual blocks in the U-Net. Moreover, we train both models on CIFAR10 for 500 epochs. We display the generated images in Figure 4.4 and 4.5 in contrast with the true images.

In terms of the normalising flow model, the generated CIFAR10 images are shown in Figure 4.4. These images are sharp, especially in terms of the outline of the objects

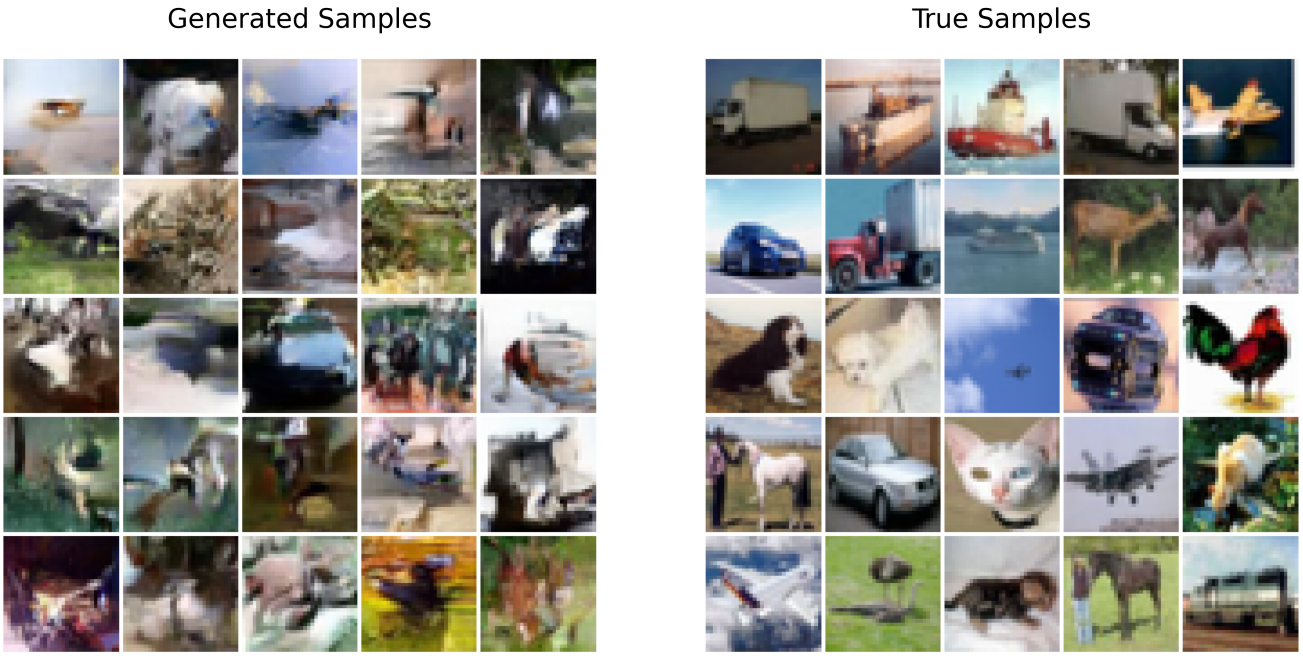

Figure 4.4: Images generated by **normalising flow (CRF)** trained on the CIFAR10. The images on the left column are generated samples. Compared to the true images on the right column, each generated image is equally sharp in terms of the outline of objects against the background colour, though most of the objects are unrecognisable.

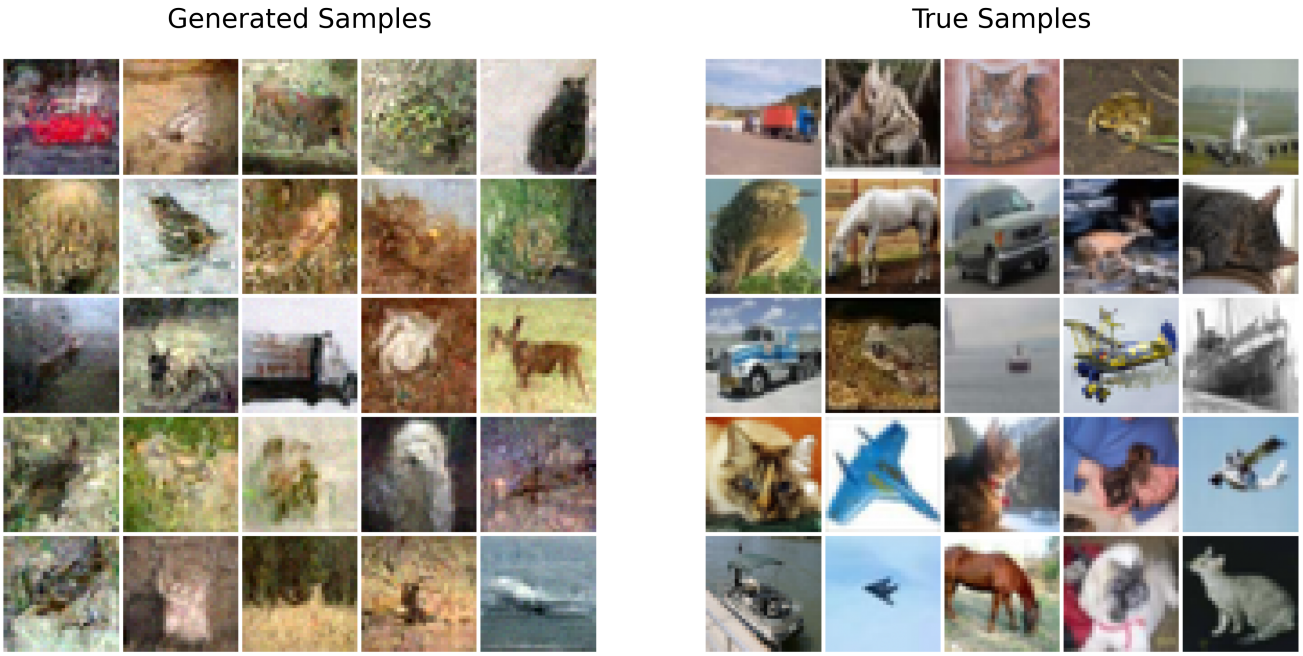

Figure 4.5: Images generated by **diffusion model (DDPM)** trained on the CIFAR10. The images on the left column are generated samples. Compared to the true images on the right column, each generate image is just a blur of certain object with lots of noise. However, it is possible to tell the labels of the objects in some generated images.

Table 4.5: Inception score and FID of the CIFAR10 images generated by the normalising flow and diffusion model, compared with the Large DDPM. The lower FID is interpreted as better, and the higher IS is explained as superior. (For reference, the $10,000$ true images sampled from CIFAR10 obtained FID of 2.17, Inception score of 11.16).

Evaluation Metrics (CIFAR10)	Normalising Flow	DDPM	Large DDPM
FID	**51.63**	60.85	12.14
Inception score	5.19	**8.31**	9.46

against the colour of the background. More strikingly, it displays equal precision of application of colour compared to the true samples, and significantly outperform the images generated by the diffusion model with respect to image quality from our intuitive perception. Nevertheless, apart from a few images shown in Figure 4.4, such as the middle one displaying the shape of a car, most of the objects are unrecognisable. By examining the images carefully, it is easy to infer the reason for this phenomenon, which is that the normalising flow model generates objects with mixed classes similar to what it does with regard to MNIST digits. In other words, the normalising flow often suffers from the problem of learning the class embedding of the input data.

Apparently, the diffusion model (DDPM) performs very poorly to generate sharp and distinct CIFAR10 images, as displayed in Figure 4.5. Generally, it performs much worse in CIFAR10 image generation task than the normalising flow in terms of image quality. Although in some images the label of the object is roughly recognisable, the model is largely overwhelmed by the complexity of CIFAR10 images. Unlike in the MNIST generating tasks, often the model fails to figure out the class of object in CIFAR10 it is going to generate, which leads to completely blurry images. However, compared to the normalising flow, the class of each image in diffusion model is more distinct, which demonstrates the capacity of DDPM to learn the class embedding of input data sufficiently.

In addition, we give the Inception score and FID of calculated on the $10,000$ generated CIFAR10 images by the normalising flow and diffusion model in Table 4.5 in comparison with the Large DDPM.

As we can see from the table, the FID score of the normalising flow (51.63) is better

than that of diffusion model (60.85), which is consistent with what we can intuitively observe from the generated images. The quality of the images generated by the normalising flow is overall superior than that of the diffusion model due to the bad quality and loss of details of the generated images by the diffusion model, especially in terms of the sharpness of colour. Nevertheless, although only possessing one third of the parameters of the normalising flow, the diffusion model still achieves competitive FID score (60.85) compared to the normalising flow (51.85). To this, we believe it might be attributed to the effectiveness of diffusion model in terms of learning the class embedding of the CIFAR10 dataset, as each image generated by diffusion model tends to display a unique class of object among the class set of CIFAR10 dataset. Furthermore, from the FID score achieved by the Large DDPM model (12.14), we can see that the number of base channels play a major role in the performance of diffusion model, since it is the primary difference between our DDPM (64 base channels) and Large DDPM (128 base channels). In addition, it demonstrates that with increased channel number, our DDPM model may well get rid of large amount of the noise in the generated images. This provides us with valuable insights regarding the improvment of our diffusion model.

With respect the Inception score, the normalising flow (CRF) (5.19) performs worse than the diffusion model (8.31). One explanation is that IS awards more in terms of the diversity of the generated images, which means our diffusion model is able to generate images with diverse categories, nevertheless with no details in the images. We believe this might also be caused by the problem of mixing of different classes of objects together encountered in the normalising flow. For the diffusion model, this is less of a concern since the model is able to successfully learn the class embedding, and generate images with unique class from class set of the input dataset. Therefore, the DDPM achieves relatively high mark (8.31) in Inception Score, which is close to the Large DDPM model (9.46). This further demonstrates our argument that the with large channel number to better capture the details of the input images, our DDPM is able to generate much clearer images , *i.e.*improve the FID score significantly, since it already proves it can generate distinct and sharp images of objects from the standpoint of a classifier.

For the computational reasons, we are unable to train the Large DDPM model

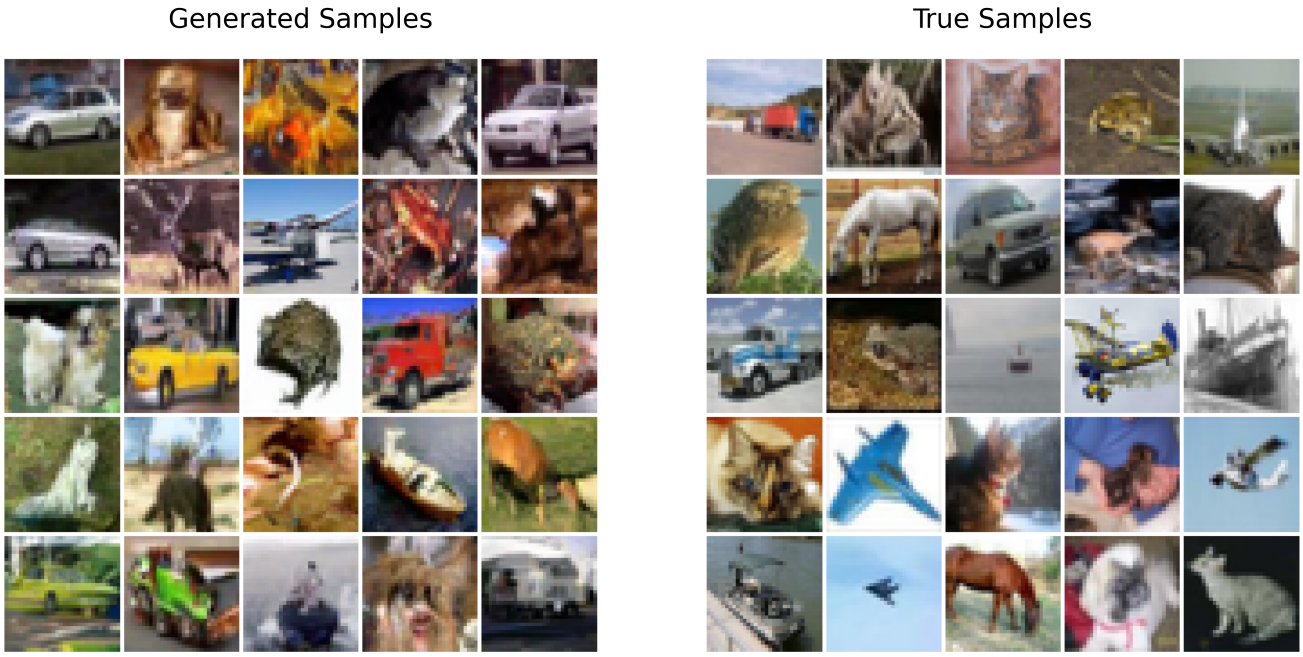

Figure 4.6: Images generated by **Large DDPM** trained on the CIFAR10. The images on the left column are generated samples. Compared to the true images on the right column, each generated image is equally high quality and display almost the same distribution.

which has over 37.5 parameters on the CIFAR10 dataset (Ho et al., 2020) in our device. For the sake of the readers who are interested in this actual results of Large DDPM, we apply the pre-trained Large DDPM provided by pesser (2023) on the CIFAR10 dataset and display the generated samples in Figure 4.6. Compared to the true images, each generated image is equally high quality and display almost the same distribution.

We show the training loss of our DDPM model over the 500 epochs in Figure 4.7 in hope of revealing more useful insights about the improvement of the diffusion model. It is worth noting that We are able to train the diffusion model, since we manage to fit the DDPM model into our hardware and the training time is tractable. However, for the normalising flow, as mentioned in Section 4.3.1, it takes theoretically 15 days to train the model, therefore we could not train the model from scratch in our device. From the perspective of the loss curves, the loss value of 0.002 demonstrates the fitness of our diffusion model in terms of generating MNIST images, as can be seen in Figure 4.7. In contrast, the loss value of the CIFAR10 training converges to

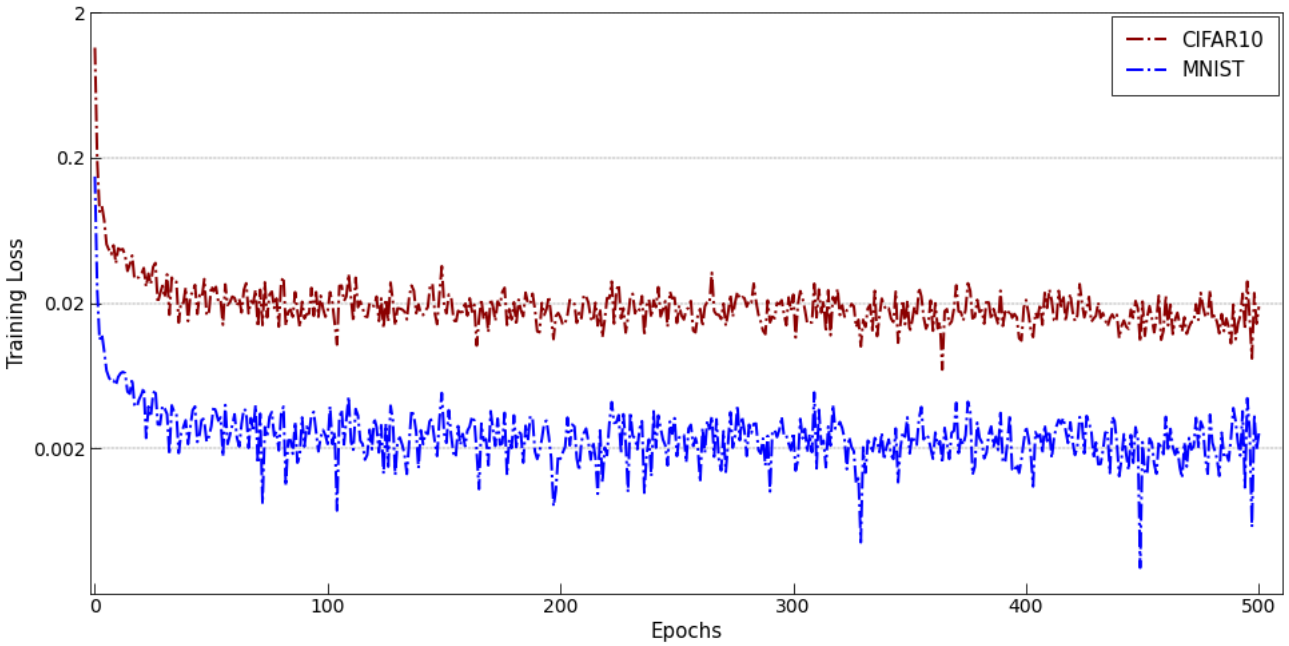

Figure 4.7: The loss curve of the MNIST and CIFAR10 training process of the diffusion model on 500 epochs. The loss of the model trained on the MNIST reaches as low as 0.002. The loss on the CIFAR10 converges to 0.02, which is significantly higher than the ideal value.

approximately 0.02, which is far from the ideal loss value for a generative model. Moreover, for both dataset sets, the traning loss curve converges in approximately 50 epochs, after that the loss curve seems to stay stationary. However, in actual training stage, we observe improved image quality as the number of epoch increases, even though there is no significant decay in the loss value.

Chapter 5

Discussion

In this section, we discuss the findings of the experiments in Chapter 4. Firstly, we present the discussion about the experimental results in Section 5.1. The Section 5.2 the gives the further improvements of the generative models and promising future work in this field.

5.1 Experimental Results Discussion

The performance of the normalising flow (CRF) and the diffusion model (DDPM) are reported in Chapter 4. In this section, we discuss the findings of the experiments run in Chapter 4 by comparing the performance of two deep generative models, namely normalising flow and diffusion model. We discuss a number of interesting points revealed by these experiments from three aspects: generated images, training and sampling speed and applications as follows:

Generated Images

- Considering the image quality of the generated samples, the normalising flow outperforms the diffusion model. Additionally, the performance gap is larger when the structure of input data is more complex. This is demonstrated by both the sharper images intuitively observed from the generated CIFAR10 images, and the better FID score of the CIFAR10 samples generated by the normalising flow than the diffusion model. We believe the better image quality shown by

the normalising flow is enpowered by its larger number of parameters and larger channel number (128 versus 64 in diffusion model). By increasing the number of base channels, the Large DDPM is able to significantly improve its sample quality on top of excellent class embedding learning. This gives one insight about how we can improve the diffusion model, which is to reinforce the part of architecture that can capture more detailed content from the input data.

- The diffusion model demonstrates better class embedding learning ability than the normalising flow. Specifically, in most cases the images generated by the diffusion model display a unique class of object from the class set of the training data, in contract to the mixture of different classes of objects generated by the normalising flow. As discussed in Chapter 4 Section 4.3.1, the better class embedding ability of diffusion model arises from its time embedding approach into the training process. Specifically, instead of training the transformation as a whole similar to the normalising flow, the diffusion model choose to train the U-Net at different stages of the transformation. Therefore, the model becomes aware of the semantic variation of the input images through the transformation process, which gives better guidance when the model generates samples step by step during the sampling stage. We believe changing the training of flow to be step-wise can facilitate the learning of the class embedding, however, it will inevitably compromise the conciseness of the flow-based models.

Training and Sampling Speed

- There is no doubt that the normalising flow is much faster in terms of sampling speed than the diffusion model. Theoretically, its one-pass density calculation allows it to instantly transform the samples drawn from base distribution to the samples from desired distribution ($2,000$ images in approximately 30s). In contrast, the step-wise sampling approach in diffusion model gives rise to the transformation being applied sequentially, which results in much slower sampling ($2,000$ images in approximately 2h). Numerous methods have been proposed to accelarate the sampling speed of diffusion model by trading off

between sampling speed and sample quality, such as Kong and Ping (2021); Okamoto et al. (2021); San-Roman et al. (2021).

- The diffusion model shows faster training speed. It runs much faster than the normalising flow in both MNIST and CIFAR10 training (87 seconds per epoch versus 56 minutes per epoch for normalising flow in the CIFAR10 training environment). It normally takes our device 8 hours to train the diffusion model for 500 epochs in the fine-tuning stage of CIFAR10 dataset, while normalising flow will require around 15 days to be fully trained for 500 epochs. Moreover, the diffusion model exhibits much faster convergence than the normalisng flow during our experiments. Specifically, as illustrated in Figure 4.7, the DDPM converges in approximately 50 epochs for both datasets. In contrast, the normalising flow will take around 300 epochs, which is quite cumbersome to implement during the fine-tuning stage of training. Comparing the complexity of the objective function in the normalising flow and diffusion model, it is not surprising that the normalising flow trains much slower. Explicitly, the normalising flow deploys a complicated maximum likelihood-based training objective function that which takes the whole transformation into account and involves the calculation of Jacobian determinant, whereas the objective function applied by the diffusion model focuses on learning each small mapping function in the large transformation process, therefore taking a much simpler form even though it is developed based on the maximum likelihood. In order to improve the training speed of the flow-based model, one way is to split the training of the whole flow into small pieces, and trains separately analogous to the approach in diffusion model. However, it will impair the smooth sampling of the normalising flow.

Applications

- Normalising flow possess two intrinsic operations: density estimation and sampling. Therefore, it is well-suited for many applications that requires probabilistic modelling of the data. The most direct way to apply a flow-based

model is to generate images, audio or other structured objects. However, what makes the normalising flow exceptional from other generative models is that it can be leveraged to perform inference: estimating concealed quantities in a model, such as the computation of the normalising constant and the calculation of expectation over the posterior. Specifically, normalising flows have been employed in Hamiltonian Monte Carlo to initialize the chain (Hoffman et al., 2019), and act as posterior approximations for global variables (Louizos and Welling, 2017).

- One of the most prominent features of the diffusion model is that it can be easily adjusted to generate conditional images based on its internal class embedding and step-wise sampling approach. Specifically, a guiding mechanism using a pre-trained classifier can be incorporated into the model to control the sampling process (Rombach et al., 2022). The principle of diffusion model can be regarded as the decomposition of image formation process into a sequential application of denoising autoencoders (Rombach et al., 2022). Denoising autoencoders are a variant of regular autoencoders that learn to denoise the corrupted version of the data to recover the original data, which is essentially the reverse generating process of the denoising diffusio model. Combined with the high quality output from the model, it makes the diffusion model a powerful tool for generation tasks such as generating high-quality videos for the file industry.

As we can see from the above comparison, the normalising flow and diffusion model have their own advantages and drawbacks in terms of sample quality and the sampling and training speed. It means that we need to use them appropriately under different situations, and the comparison provides us with useful insights about how to improve the two generative models in the future work presented in the following section.

5.2 Further Improvements and Future Work

For future reference, we give the potential next steps for works in this thesis.

- It is difficult to model discrete modes with the flow-based models, since the models must apply continuous transformations onto a continuous distribution. Often the flow-based models allocates false density to connections between these disconnected points, leading to potential inaccuracies (Rasul et al., 2021). Specifically, the use of normalising flows in discrete valued data is achieved by dequantising (restoring the missing data) the data, by attaching uniform noise to the data. Future work could explore methods of learning the disconnected distributions in a clear-cut manner without the dequantisation.

- From the perspective of the flow transformation, normalising flow learns the whole process, while the diffusion model learns each small transformation on the high-level view of the generative model. They seem to take two ends of the generative approaches in the maximum likelihood-based generative models. By sacrificing the class embedding learning ability and training efficiency, the normalsing flow achieves maximum sampling speed and excellent sharpness in the samples. Meanwhile, the diffusion model attains the class embedding learning of the input data, fast training and high-quality images through the compromise in sampling time. By drawing inspirations from the normalising flow, we propose to reduce the number of time steps in the reverse generation process of the diffusion model in the future work. Another promising future work direction is to apply the generating process in the latent space instead of the pixel space to accomplish complexity reduction (Rombach et al., 2022).

Chapter 6

Conclusions

This thesis concentrates on two deep generative models: diffusion models and normalising flows. First, we give a high-level view of the generative models (Chapter 1). Then we present the backgrounds and brief history for the development of the generative models and their link with the representation learning (Chapter 2. Second, we describe the foundational ideas of the two types of generative models: diffusion models and normalising flows (Chapter 3). In particular, we focus on two specific models, namely denoising diffusion probabilistic model (DDPM) and contractive residual flow (CRF). Third, we implement these two specific models on two common datasets in computer vision, and present the experimental results (Chapter 4). Lastly, we compare the DDPM and CRF based on the experimental results, and discuss the further improvements and future work for the normalising flow and diffusion model.

We exhibits that the normalising flow is a very universal and expressive probabilistic model. By parameterising flows with deep neural network, the flow-based models fully leverage the compositional nature of the computation. In the literature of flow-based modes, most work focuses on maximising the expressivity of the flow while retaining the invertible transformation and tractable Jaocobian determinant computation. The specific model presented in this thesis utilises the idea of Lipschitz continuity to construct an invertible flow composed of the activation functions in deep neural network. This enpowers the flow to be extremely expressive, beating the diffusion model in our experiments in terms of the sample quality.

Furthermore, we illustrately shows that the denoising diffusion probabilistic

model is comprised of two process: a fixed forward diffusion process, and a reverse generative process. We provide the fundamental probabilistic framework under the diffusion model and further deduce the final form of the objective function based on maximum likelihood principle. Besides, we provide the implementation of the algorithm introduced in Chapter 3 in Chapter 4. More importantly, we make several key improvement to the original Large DDPM architecture. By doing so, we successfully fit the model into our hardware and manage to effectively compensate the loss in performance caused by smaller number of parameters. Finally, we show that the number of base channels in the diffusion model is crucial to lower the noise level in the generated images and capture finer detail of training data.

Through the experimental results, we demonstrate that with our given hardware, the normalising flow can achieve better sample quality than the diffusion model, although the diffusion model outperforms in terms of class embedding learning ability. In addition, the sampling speed of the normalising flow is significant faster than that of the diffusion model (240× faster), while the diffusion model is faster in training (38.62× faster). We detail the reasons for the difference in Chapter 4 and Chapter 5.

Looking forward, we propose future directions for the two deep generative models. For the normalising flow, firstly, we propose to split the training of the complete flow into small piece, after that we trains them separately. Besides, we suggest to explore methods of learning the discrete distributions explicitly without dequantisation. With regard to diffusion model, we put forward the idea to reduce the number of time steps in the reverse process in order to alleviate the computational burden. Moreover, we draw inspirations from Rombach et al. (2022) and advocate that the future work could explore the idea of applying the generation process in the latent space.

Bibliography

Alain, G., Bengio, Y., Yao, L., Yosinski, J., Thibodeau-Laufer, E., Zhang, S., and Vincent, P. (2016). Gsns: generative stochastic networks. *Information and Inference: A Journal of the IMA*, 5(2):210–249.

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. (2019). Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR.

Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.

Bond-Taylor, S., Leach, A., Long, Y., and Willcocks, C. G. (2021). Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *IEEE transactions on pattern analysis and machine intelligence*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Chen, R. T., Behrmann, J., Duvenaud, D. K., and Jacobsen, J.-H. (2019). Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*, 32.

Chen, R. T. Q., Behrmann, J., Duvenaud, D., and Jacobsen, J. (2020). Residual flows for invertible generative modeling.

Chen, S. and Gopinath, R. (2000). Gaussianization. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Chen, X. and He, K. (2021). Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15750–15758.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dhariwal, P. and Nichol, A. (2021). Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *stat*, 1050:10.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

Hinton, G. (2014). 2014 ama on reddit quote. https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton/.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. (2019). Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pages 2722–2730. PMLR.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851.

Hoffman, M., Sountsov, P., Dillon, J. V., Langmore, I., Tran, D., and Vasudevan, S. (2019). Neutra-lizing bad geometry in hamiltonian monte carlo using neural transport. *arXiv preprint arXiv:1903.03704*.

Huang, H., Lin, L., Tong, R., Hu, H., Zhang, Q., Iwamoto, Y., Han, X., Chen, Y.-W., and Wu, J. (2020). Unet 3+: A full-scale connected unet for medical image segmentation. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1055–1059. IEEE.

Hutchinson, M. F. (1990). A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kong, Z. and Ping, W. (2021). On fast sampling of diffusion probabilistic models. *arXiv preprint arXiv:2106.00132*.

Kreis, K., Gao, R., and Vahdat, A. (2022). Tutorial on denoising diffusion-based generative modeling: Foundations and applications. `https://www.youtube.com/watch?v=cS6JQpEY9cs`.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA. Curran Associates Inc.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Liu, S., Wei, Y., Lu, J., and Zhou, J. (2018). An improved evaluation framework for generative adversarial networks. *arXiv preprint arXiv:1803.07474*.

Louizos, C. and Welling, M. (2017). Multiplicative normalizing flows for variational bayesian neural networks. In *International Conference on Machine Learning*, pages 2218–2227. PMLR.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27.

Okamoto, T., Toda, T., Shiga, Y., and Kawai, H. (2021). Noise level limited sub-modeling for diffusion probabilistic vocoders. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6029–6033. IEEE.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, 22(1):2617–2680.

pesser (2023). Pytorch pretrained diffusion models. `https://github.com/pesser`.

Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training. https://gwern.net/doc/www/s3-us-west-2.amazonaws.com/d73fdc5ffa8627bce44dcda2fc012da638ffb158.pdf.

Rasul, K., Seward, C., Schuster, I., and Vollgraf, R. (2021). Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning*, pages 8857–8868. PMLR.

Ravuri, S., Lenc, K., Willson, M., Kangin, D., Lam, R., Mirowski, P., Fitzsimons, M., Athanassiadou, M., Kashem, S., Madge, S., et al. (2021). Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 597(7878):672–677.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., et al. (2022a). Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494.

Saharia, C., Ho, J., Chan, W., Salimans, T., Fleet, D. J., and Norouzi, M. (2022b). Image super-resolution via iterative refinement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *Advances in neural information processing systems*, 29.

San-Roman, R., Nachmani, E., and Wolf, L. (2021). Noise estimation for generative diffusion models. *arXiv preprint arXiv:2104.02600*.

Schuhmann, C., Vencu, R., Beaumont, R., Kaczmarczyk, R., Mullis, C., Katta, A., Coombes, T., Jitsev, J., and Komatsuzaki, A. (2021). Laion-400m: Open dataset of clip-filtered 400 million image-text pairs. *arXiv preprint arXiv:2111.02114*.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR.

Song, J., Zhao, S., and Ermon, S. (2017). A-nice-mc: Adversarial training for mcmc. *Advances in Neural Information Processing Systems*, 30.

Song, Y. and Ermon, S. (2019). Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32.

Strimmer, K. (2022). Multivariate statistics and machine learning. `https://strimmerlab.github.io/publications/lecture-notes/MATH38161/transformations-and-dimension-reduction.html`.

Tabak, E. G. and Turner, C. V. (2013). A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164.

Vahdat, A. and Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *Advances in neural information processing systems*, 33:19667–19679.

Varuna Jayasiri, N. W. (2020). labml.ai annotated paper implementations.

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103.

Wikipedia, the free encyclopedia (2002). Japanese castle. `https://upload.wikimedia.org/wikipedia/commons/thumb/3/35/Himeji_Castle_`

The_Keep_Towers.jpg/640px-Himeji_Castle_The_Keep_Towers.jpg. [Online; accessed March 10, 2023].

Wikipedia, the free encyclopedia (2022). European castle in japan. https://upload.wikimedia.org/wikipedia/commons/thumb/9/99/X-Y_plot_of_algorithmically-generated_AI_art_of_European-style_castle_in_Japan_demonstrating_DDIM_diffusion_steps.png/800px-X-Y_plot_of_algorithmically-generated_AI_art_of_European-style_castle_in_Japan_demonstrating_DDIM_diffusion_steps.png. [Online; accessed March 10, 2023].

Wu, Y. and He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Appendix A

Computer code

The computer code for this thesis is made available from: `https://github.com/yqjing/sem2project`. We thank the following authors for making the code publicly available.

For code in `Diffusion_models`, we follow the code based on the collection of works:

1. labml diffusion model,
`https://nn.labml.ai/diffusion/ddpm/index.html`
2. openai/guided-diffusion,
`https://github.com/openai/guided-diffusion`
3. hojonathanho/diffusion,
`https://github.com/hojonathanho/diffusion`
4. lucidrains/denoising-diffusion-pytorch,
`https://github.com/lucidrains/denoising-diffusion-pytorch`
5. huggingface/The Annotated Diffusion Model,
`https://huggingface.co/blog/annotated-diffusion`
6. openai/chatgpt,
`https://chat.openai.com/auth/login?next=/chat`
7. pytorch-gan-metrics 0.5.2,
`https://pypi.org/project/pytorch-gan-metrics/`
8. pesser/pytorch_diffusion,
`https://github.com/pesser/pytorch_diffusion`

For code in `Normalising_flows`, we follow the code based on the collection of works:

1. rtqichen/residual-flows,
`https://github.com/rtqichen/residual-flows`
2. Pretrained models,
`https://github.com/rtqichen/residual-flows/releases/tag/v1.0.0`

For code in `FID_calculation`, we follow the code based on the collection of works:

1. Newbeeer/stf,
`https://github.com/Newbeeer/stf`