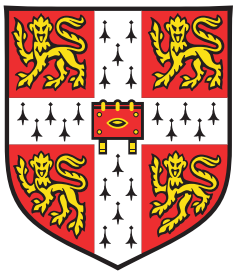
Wasserstein Generative Adversarial Network



Wenbo Gong

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Gonville and Caius College August 2017

I would like to dedicate this thesis to my loving parents and my fiancée for their invaluable supports during my project.

Declaration

I, Wenbo Gong of Gonville and Caius College, being a candidate for the M.Phil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

The thesis contains total words of 13814 excluding bibliography, photographs and diagrams but including tables, footnotes, and appendices.

Signed: Wenbo Gong

Date: 11/08/2017

Wenbo Gong
August 2017

Acknowledgements

First, I would like to thank my supervisor Dr. Richard Turner for his valuable supports and indispensable advices on this project. Working with him has been a wonderful experience for me.

Also I would like to thank Yingzhen Li, Mark Rowland and Sihui Wang. The discussions with them are extremely helpful and inspiring.

Abstract

Recent advances in deep generative models give us new perspective on modeling high-dimensional, nonlinear data distributions. Especially the GAN training can successfully produce sharp, realistic images. However, GAN sidesteps the use of traditional maximum likelihood learning and instead adopts an two-player game approach. This new training behaves very differently compared to ML learning. There are still many remaining problem of GAN training. In this thesis, we gives a comprehensive review of recently published methods or analysis on GAN training, especially the Wasserstein GAN and FlowGAN model. We also discuss the limitation of the later model and use this as the motivation to propose a novel generator architecture using mixture models. Furthermore, we also modify the discriminator architecture using similar ideas to allow 'personalized' guidance. We refer the generator mixture model as Mixflow and mixture of discriminators as 'personalized GAN' (PGAN). In experiment chapter, we demonstrate their performance advantages using toy examples compared to single flow model. In the end, we test their performance on MNIST dataset and the Mixflow model not only achieves the best log likelihood but also produce reasonable images compared to state-of-art DCGAN generation.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

In the context of *Deep Learning*, the goal is to discover and approximate the complex, high-dimensional and intractable probabilistic distributions. The striking success has been mainly based on the **discriminative** models which maps the real data to label. The **generative** model has less impact due to the difficulty of approximating complex distributions.

The Generative Adversarial Network(GAN) (Goodfellow et al., 2014) sidesteps these difficulties and adopts a *two-player* game approach that implicitly change the generator distribution towards targeting distribution. However, this model setup suffers from well-known problems. First, the original GAN is notoriously difficult to train stably, like the **vanishing gradient** problem. In addition, the GAN tends to only fits parts of the distribution called mode-dropping problem. Last but not least, due to the implicit property of the generator, it is difficult to explicitly evaluate the sample log likelihood, thus leaving the 'gap' between the implicit model and explicit generator likelihood.

The objective of this project is to review the current published methods on GAN and try to propose an improvements towards one of the above problems.

1.2 Contribution

First, we will give comprehensive a review of some of the popular methods on GAN and how they handle the proposed problems. Then based on the review (especially **FlowGAN** method (Grover et al., 2017)), we will propose a novel network using mixtures models and

CNN that will further improve the capacity and performance of FlowGAN. Last but not least, we will also conduct some toy experiments to illustrate the properties of GAN training and compare the difference of *maximum likelihood training* and *GAN training*.

1.3 Thesis Layout

In Chapter 2, we will first give a detailed introduction to the **Original GAN** model and discuss its limitations. Then, **Wasserstein GAN**, a major improvement, will be explained. In the end, we will discuss the variations of GAN that can be categorized into three major classes.

In Chapter 3, a detailed review of **Normalizing Flow** will be given and how they are used in ML training. Then, the limitations of Flow will be discussed.

In Chapter 4, we will first propose the novel generator architecture that can be used for both ML and GAN training. Then followed by a discriminator mixture model, it allows for 'personalized' guidance for generator component. The combined discriminator and generator mixture is called PGAN.

In Chapter 5, we will conducts some experiments to show the new architecture will indeed increase the performance compared to the original **FlowGAN** model. We also test the log likelihood and generation quality for MNIST data set, which shows our model can both achieve high likelihood and reasonable generation quality compared to state-of-art DCGAN(Radford et al., 2015).

In the last chapter, we will talk about the possible future work and give a conclusion of the thesis.

Chapter 2

Generative Adversarial Network

2.1 Background

The goal of **deep learning** is to discover the rich, complex distributions encountered in the real life. The major success has been mainly based on the discriminative model. The unsupervised learning, especially the **generative models**, are less successful. The main difficulty is to approximate high dimensional, non-linear distributions during maximum likelihood training. This requires the tractability and high capacity of the model. Normally, these two properties are in opposite directions.

The **Generative Adversarial Networks**(Goodfellow et al., 2014) uses a completely different training approaches. This allows us to use a model with high modeling power and not need to worry about explicit likelihood. Specifically, the model has a discriminator and a generator. The discriminator will determine whether the samples are 'real' or 'fake'. The generator will use this information to improve generation quality. The competition (like **Figure** 2.1.) will drive both 'players' to improve their performance. It has been shown that the equilibrium in this game is when the generated samples are indistinguishable from the real samples and discriminator cannot do a better job than a random guess.

This training algorithm sidesteps the use of likelihood and any approximate inference techniques (e.g. MCMC, variational inference, etc.). Thus, we can use the *universal function approximator*, i.e. Multi-layer perceptron (MLP) or Convolution Neural Network (CNN) as our discriminator and generator.

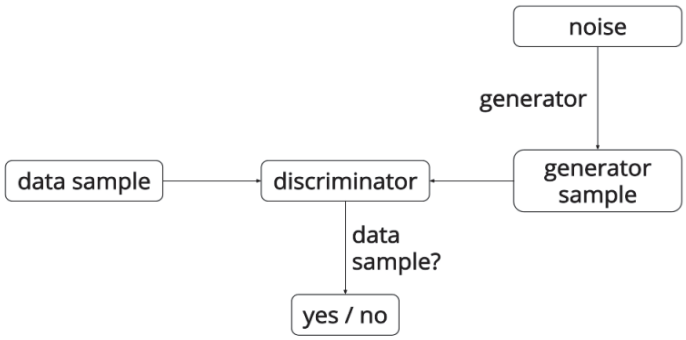


Fig. 2.1 The Original GAN model structure

2.2 Original GAN

Assume we have our **discriminator** $D(\cdot;\theta_d)$ and **generator** $G(\cdot;\theta_g)$ parametrized by θ_d and θ_g respectively. We also have true data distribution p_d defined on space $\boldsymbol{X}$ and **base distribution** p_z defined on space $\boldsymbol{Z}$. The **generator** $G(\cdot,\theta_g)$ defines mapping between space $\boldsymbol{Z}$ and $\boldsymbol{X}$, $\boldsymbol{z} \longmapsto \boldsymbol{x}$ where $\boldsymbol{x} \in \boldsymbol{X}$ and $\boldsymbol{z} \in \boldsymbol{Z}$. The generator also implicitly define the **generator distribution** p_g through the push $p_g = G_{\#}(p_z)$.

We want to **maximize** the probability of assigning correct labels for both real data and generated data. In addition, we also want to train $G(\cdot,\theta_g)$ simultaneously to **minimize** the probability of assigning generated data with the 'fake' label. Thus, the objective function is defined as

$$\min_{\theta_g}\max_{\theta_d} V(D,G) = E_{\boldsymbol{x}\sim p_d}[\log D(\boldsymbol{x})] + E_{\boldsymbol{z}\sim p_z}[\log(1-D(G(\boldsymbol{z})))] \tag{2.1}$$

In fact, we can treat the **generator** $G(\cdot,\theta_g)$ as the decoder which maps from the latent space $\boldsymbol{Z}$ to data space $\boldsymbol{X}$ in the context of **Autoencoder**. This setup can be generalized to different variations on GAN which will be discussed later.

We can derive this objective through a probabilistic framework.

Proposition 1. *For fixed generator G, the objective function is equivalent to maximize the log likelihood of assigning correct labels to binary labeled data.*

Proof. Assume we have data distribution $p(\boldsymbol{x})$ defined on a space $\boldsymbol{X}$ with label information $y \in \{0,1\}$. We have a classifier that will assign labels to the input data $\boldsymbol{x}$, which is defined as $p(y|\boldsymbol{x})$.

Thus, to maximize the log likelihood of assigning correct labels,

$$\begin{aligned}&\max_{\theta_d} E_{\boldsymbol{x}\sim p(\boldsymbol{x})}[\log p(y|\boldsymbol{x})]\\ =&\max[E_{x\sim p(\boldsymbol{x}|y=1)}[\log p(y=1|\boldsymbol{x})]+E_{x\sim p(\boldsymbol{x}|y=0)}[\log p(y=0|\boldsymbol{x})]]\end{aligned} \tag{2.2}$$

Thus, if we define $p(\boldsymbol{x}|y=0)=G_{\#}(p_z)$, we will recover the original objective function. □

In the original GAN paper(Goodfellow et al., 2014), they proved the convergence of this minimax training procedure. The sketch of the proof is first we fix the generator, we can derive the optimal discriminator $D^*(\boldsymbol{x})$ to be $\frac{p_d(\boldsymbol{x})}{p_d(\boldsymbol{x})+p_g(\boldsymbol{x})}$. Then, by using this optimal discriminator, we can show that the minimum of the objective is achieved if and only if $G_{\#}(p_z)=p_d$. The detailed proof are in the **Appendix 1**.

From the Theorem 4 in **Appendix 1**, we show that minimizing the training objective for generator is equivalent to minimizing the Jensen-Shannon distance (JSD) (defined in the proof) between p_d and p_g. There are some good properties of JSD compared to the traditional KL divergence.

- KL divergence $KL(p||q)$ is not symmetric by definition, which means $KL(p||q)\neq KL(q||p)$, whereas the JSD(p||q) is symmetric.
- For minimizing $KL(p||q)=\int_{\boldsymbol{X}} p(\boldsymbol{x})\log(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})})d\boldsymbol{x}$, this is equivalent to maximizing the log likelihood $q(\boldsymbol{x})$ (Proof is trivial, just use Monte Carlo approximation to KL(p||q)). From the definition of $KL(p||q)$, if $\boldsymbol{x}$ is from the high probability region of $p(\boldsymbol{x})$ and low probability region of $q(\boldsymbol{x})$, then, the $KL(p||q)$ will quickly expand to infinity. On the other hand, if $q(\boldsymbol{x})>p(\boldsymbol{x})$, the KL term will tends to be small. Thus, if we minimize $KL(p||q)$, the model will tends to cover the whole support of $p(\boldsymbol{x})$ rather than generating realistic images. The situation is reversed if we minimize $KL(q||p)$.
- Based on the previous point, the KL divergence is not always defined by definintion.
- The JSD is always defined. In addition, JSD can be treated as a symmetric middle ground to the two KL divergence. From (Theis et al., 2015), it is conjectured the reason of success in GAN at producing real-look images is due to the switch from the ML training.

The training algorithm for GAN is introduced in **Algorithm** 1.
The algorithm basically trains a measure represented by $D(\cdot)$ to estimate the Jensen-Shannon

Algorithm 1 Original GAN Training Algorithm

1: **for** training iterations **do**
2: **for** discriminator steps **do**
3: Sample batches of m samples $\boldsymbol{z} \sim p_z(\boldsymbol{z})$
4: Sample batches of m samples real data $\boldsymbol{x} \sim p_d(\boldsymbol{x})$
5: Maximize and update the discriminator parameters using gradient descent optimization:

$$\max_D \frac{1}{m} \sum_{i=1}^{m} [\log(D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i)))] \tag{2.3}$$

6: **end for**
7: Sample batches of m samples $\boldsymbol{z} \sim p_z(\boldsymbol{z})$
8: Minimize and update the generator:

$$\min_G \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\boldsymbol{z}_i))) \tag{2.4}$$

9: **end for**

distance between p_d and p_g, then we try to reduce the JSD by updating the generator until convergence. However, in practice, the generated images will be rejected with high confidence by the discriminator at first, therefore, $\log(1 - D(G(\boldsymbol{z})))$ will saturates and won't provide useful gradient. Instead, we can maximize $\log(D(G(\boldsymbol{z})))$ which will end with the same stationary point but provides a better gradients.

2.2.1 Theoretical Limitation

This original GAN architecture is known for its unstable training and often suffers from 'mode-dropping' problem. To be specific, the 'mode-dropping' means the generator distribution is highly concentrated on parts of targeting distribution and loses the generation diversity. Especially, it is counter-intuitive that the better we train the discriminator, the worse the generator will be due to vanishing gradients. This is claimed to be due to saturation of $\log(1 - D(G(\boldsymbol{z})))$ according to (Goodfellow et al., 2014), however, even we replace it with $\log(D(G(\boldsymbol{z})))$ as suggested, this still happens.

The instability of training is theoretically investigated in (Arjovsky and Bottou, 2017). In this subsection, we will omit the formal detailed proofs, instead, an overview of the ideas will be presented.

First, we explain why the GAN training suffers from vanishing gradient problem. *Lemma*

1 in (Arjovsky and Bottou, 2017) proves that the generator distribution is only supported by a union of low dimensional manifold if $G(\cdot)$ is well-behaved neural net. The intuition is that the generator distribution is defined as $p_g = G_{\#}(p_z)$, thus, the support of p_g should be contained in space $\boldsymbol{Z}$ where the dimension is much smaller than $\boldsymbol{X}$.

If the distribution is supported by low dimensional manifold, then it is not **absolutely continuous**. The **absolute continuous** indicates that if a set A has a Lebesgue measure 0, then the measure $\mu(A) = 0$, in our context, measure μ is probability measure. So, based on the definition, if the distribution is supported by low dimensional manifold, then the Lebesgue measure of the manifold is 0 however, the probability of the manifold is not zero obviously. Now we introduce two key concepts where the vanishing gradients theorem (Theorem 2.1 (Arjovsky and Bottou, 2017)) is based on.

Definition 1. *Normal Topology Space: The topology space is normal if any two closed disjoint sets have open disjoint neighborhoods.*

Lemma 1. *Urysohn's lemma: A topological space is normal if and only if any two closed disjoint subsets can be separated by a continuous function.*

Thus, in **Theorem 2.1** (Assume the supports are disjoint), it first show that the supports of p_d and p_g contains open disjoint neighborhoods and the space is normal. Then Urysohn's lemma states there exists a continuous function that perfectly separate them and have accuracy 1 (or 0) under the support of p_d (or p_g). Thus, the gradient of the discriminator w.r.t input is 0.

Then **Theorem 2.2** further generalize that if the support of p_d and p_g are not perfectly aligned, then by similar methods, it can be shown that there exists a perfect discriminator that separates the support. Thus, overall, this explains the vanishing gradient problem of GAN training. Further, **Theorem 2.4** shows if the discriminator is closed to optimal discriminator, then the gradient will be very small. Thus, these theorems theoretically explain the reason why the better the discriminator, the worse the gradients for generator.

If the alternative training objective $E_{\boldsymbol{z}\sim p_z}[-\log(D(G(\boldsymbol{z})))]$ is used, **Theorem 2.5** shows that with optimal discriminator and fixed generator parameter θ_0, then

$$E_{\boldsymbol{z}\sim p_z}[-\nabla_\theta \log(D^*(G_\theta(\boldsymbol{z})))|_{\theta=\theta_0}] = \nabla_\theta[KL(p_g||p_d) - 2JSD(p_g||p_d)]|_{\theta=\theta_0} \tag{2.5}$$

Thus, based on the properties of KL divergence, this is not equivalent to maximum likelihood, which will encourage the generator to generate 'real-look' images but won't assign high

cost if p_g does not cover the whole p_d. In addition, due to the symmetrical property of JSD, so it won't alter this behavior. Thus, we will see **mode-dropping** of GAN. In addition, **Theorem 2.6** shows that with imperfect discriminator (different by ε), then the distribution of the gradient $E_{z\sim p_z}[-\nabla_\theta \log(D(G_\theta(z)))]$ is actually a **Cauchy distribution**. However, the Cauchy distribution does not have finite moments of order greater than or equal to one. Therefore, this means it will have infinite expectation and variance, causing the instability of training.

Based on the theorems stated above, the main assumptions are that the generator distributions has low dimensional supports. So one simple way to break this assumption proposed by Arjovsky and Bottou (2017) is to add **absolutely continuous** noise to the input of the discriminator. In fact, the generator will be like the decoder in VAE (Kingma and Welling, 2013). Luckily, there is more elegant alternative measure called Wasserstein metric.

2.3 Wasserstein GAN

In this section, we will first introduce some basics about Wasserstein distance, followed by a new algorithm proposed by Arjovsky et al. (2017), called Wasserstein GAN (WGAN). Then, we will discuss some variants based on the WGAN.

The Wasserstein distance is also called Earth Mover's distance(EMD). The intuition comes from a simple idea: How to move things to a desired shape and location with minimum energy consumption. In terms of probability distributions, if we regard each distribution as a heap of earth and we want to transform them into another heap with minimum effort as shown in **Figure** 2.2. This distance is often be used as a distance measure between distributions.

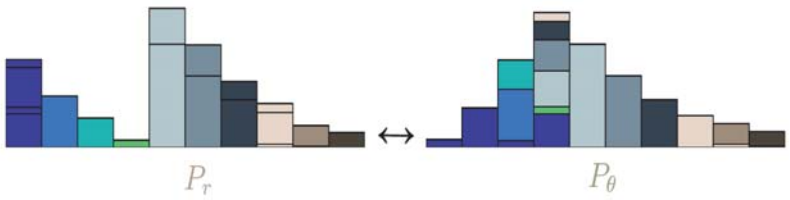


Fig. 2.2 EMD for two discrete distributions

This metric has been widely used in computer vision. For example, it has been used for image and texture retrieval (Rubner et al., 2000).

2.3.1 Primal Form

Based on the intuition of EMD, we can easily formulate the equation. We first define the transporting plan which is equivalent to the joint distribution of two probabilities. This states how we distribute amount of earth $\mathbf{x}$ in domain $\mathbf{Y}$, and vice versa. For valid plan, we need constraints to ensure it produce the correct marginal probability.

$$\begin{aligned} \int_{\boldsymbol{X}} \gamma(\boldsymbol{x},\boldsymbol{y}) d\boldsymbol{x} &= p_\theta(\boldsymbol{y}) \\ \int_{\boldsymbol{Y}} \gamma(\boldsymbol{x},\boldsymbol{y}) d\boldsymbol{y} &= p_r(\boldsymbol{x}) \end{aligned} \tag{2.6}$$

Therefore, the EMD can be defined as a constrained optimization problem:

$$\begin{gathered} EMD(p_r, p_\theta) = \inf_{\gamma \in \Pi} E_{(x,y)\sim\gamma(\boldsymbol{x},\boldsymbol{y})}[||\boldsymbol{x}-\boldsymbol{y}||\gamma(\boldsymbol{x},\boldsymbol{y})] \\ \text{s.t. constraints (2.6)} \end{gathered} \tag{2.7}$$

where Π represents the set containing all possible joint distributions and $||\cdot||$ is the distance measure between two points (e.g. Euclidean distance). With samples $(\boldsymbol{x},\boldsymbol{y})$ from distribution with finite support, we often denote $\boldsymbol{\Gamma} = \gamma(\boldsymbol{x},\boldsymbol{y})$ and $\boldsymbol{D} = ||\boldsymbol{x}-\boldsymbol{y}||$ where $\boldsymbol{\Gamma},\boldsymbol{D} \in \mathbb{R}^{l\times l}$. Thus, equation 2.7 can be written as

$$EMD(p_r, p_\theta) = \inf_{\gamma \in \Pi} < \boldsymbol{D},\boldsymbol{\Gamma} >_F \tag{2.8}$$

where $< \cdot,\cdot >_F$ means the sum of element-wise product. This formulation will be useful in **primal-form** WGAN (Sinkhorn Autodiff) with finite support distributions. However, in some problems, we only care about the distances rather than the transporting plan. Thus, there is a well-known dual formulation of the EMD called Kantorovich-Rubinstein duality. Before we dive into this dual form EMD, we first show that the primal form EMD can be formulated as **Linear Programming** if distributions have **finite** supports (like sampled empirical distribution).

Assume we flatten the distance matrix $\boldsymbol{D}$ as vector $\boldsymbol{c}$, thus, we have a corresponding flattened joint probability vector $\boldsymbol{x}$, then the objective function will be $\boldsymbol{c}^T\boldsymbol{x}$. Similarly, we can construct the constraints through matrix $\boldsymbol{A}$ containing 0 or 1 which will pick particular joint probability and sum them to the desired marginalized probability. The corresponding marginalized

probabilities are stored in vector $\boldsymbol{b}$. Thus, overall, the **linear programming formulation**:

$$\begin{aligned} &\min_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \\ &\text{s.t.} \, \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ &\boldsymbol{x} \geq 0 \end{aligned} \tag{2.9}$$

2.3.2 Dual Form

Based on the linear programming formulation and duality theorem, there exists a dual formulation where its optimal value defines the bound of the primal optimal value (in our formulation it is the lower bound if both solutions are bounded and feasible). However, we can show that this formulation has zero duality gap, thus, this indicates **strong duality**.

Theorem 1. *There exists a strong duality formulation to the original linear programming (2.9). The formulation is* $\max_{\boldsymbol{y}} [\boldsymbol{b}^T \boldsymbol{y}]$ *such that* $\boldsymbol{A}^T \boldsymbol{y} \leq \boldsymbol{c}$.

Proof. Refer to Appendix. □

Thus, we can define the strong duality formulation

$$\begin{aligned} &\max_{\boldsymbol{y}} \boldsymbol{b}^T \boldsymbol{y} \\ &\text{s.t.} \, \boldsymbol{A}^T \boldsymbol{y} \leq \boldsymbol{c} \end{aligned} \tag{2.10}$$

Then, we assume

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{bmatrix} \tag{2.11}$$

By formulation of $\boldsymbol{c}$ and $\boldsymbol{A}$ and we regard $\boldsymbol{f}$ and $\boldsymbol{g}$ as functions evaluated at $\boldsymbol{x}$, thus, the constraints can be summarized as

$$f(\boldsymbol{x}_i) + g(\boldsymbol{x}_j) \leq \boldsymbol{D}_{i,j} \tag{2.12}$$

If i=j, then we have $g(\boldsymbol{x}_i) \leq -f(\boldsymbol{x}_i)$. Thus, the maximum is achieved if $g = -f$. Thus, the objective function is changed to

$$\max_{f} \mathbb{E}_{\boldsymbol{x} \sim p_r}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x} \sim p_\theta}[f(\boldsymbol{x})] \tag{2.13}$$

As for the constraints, we have

$$\begin{aligned} f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) &\leq \boldsymbol{D}_{i,j} \\ f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j) &\geq -\boldsymbol{D}_{i,j} \end{aligned} \tag{2.14}$$

If we use Euclidean distance as distance measure, then this indicates the maximum and minimum slope are 1 and -1. Thus, this indicates $||f||_L \leq 1$, in other words, Lipschitz continuity with constraints 1. Therefore, the Kantorovich-Rubinstein duality is formulated as

Definition 2. *Kantorovich-Rubinstein duality: The Wasserstein distance between two distributions p_r and p_θ can be defined as*

$$\max_{f:||f||_L \leq 1} \mathbb{E}_{\boldsymbol{x}\sim p_r}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim p_\theta}[f(\boldsymbol{x})] \tag{2.15}$$

Now, we introduce the Wasserstein GAN objective based on this duality formulation:

$$\min_{G(\cdot)} \max_{f(\cdot):||f||_L \leq 1} \mathbb{E}_{\boldsymbol{x}\sim p_d}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p_z}[f(G(\boldsymbol{z}))] \tag{2.16}$$

However, the implementation is not trivial due to the Lipschitz continuity constraints.

2.3.3 Lipschitz Constraints

In the WGAN paper (Arjovsky et al., 2017), they propose a simple way to deal with the Lipschitz constraints. First, we can easily see that if we replace $||f||_L \leq 1$ with $||f||_L \leq K$, then the EMD will be multiplied by constant K (this is equivalent to multiplying K to the distance matrix $\boldsymbol{D}$ and from the primal form EMD, the optimal distance will also be multiplied by K). Therefor, first we assume the discriminator function f is parameterized by $\boldsymbol{w}$, then we define a compact parameter space $\boldsymbol{W}$ such that the elements in the set $\boldsymbol{F} = \{f_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\boldsymbol{W}}$ are all K-Lipschitz continuous. Thus, the problem is changed to

$$\max_{\boldsymbol{w}\in\boldsymbol{W}} \mathbb{E}_{\boldsymbol{x}\sim p_d}[f_{\boldsymbol{w}}(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p_z}[f_{\boldsymbol{w}}(G(\boldsymbol{z}))] \tag{2.17}$$

To actually solve above problem, we need to choose a powerful function approximator. Typically, a deep neural network with weights $\boldsymbol{w}$ are chosen. To make sure elements in $\boldsymbol{F}$ are the K-Lipschitz function, we can clip the weight within a small fixed range, say

$\boldsymbol{W} = [-0.01, 0.01]^l$ after each updates. The intuition behind this is if the weights are small enough, then the gradients with respect to input will also be small. However, this is not a good solution because it severally damages the modeling power of the neural nets and often requires a careful tuning of the range. In the end of this section, we will demonstrate these phenomenon using toy examples.

The algorithm of this WGAN-WeightClipping(WGAN-WC) are described in the following.

Algorithm 2 WGAN-WC, Clipping parameter: c=0.01,learning rate: l=0.00005,Discriminator iteration: nc=5,batch size: m=64

while Not converge **do**
  for $t = 0 \ldots, nc$ **do**
    Sample $\{\boldsymbol{x}^{(i)}\}_{i=1}^{m} \sim p_d$ from real data
    Sample $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p_z$ from noise distribution
    Using RMSProp with learning rate lr to update weight $\boldsymbol{w}$ based on objective $\frac{1}{m}\sum_{i=1}^{m} f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_{\boldsymbol{w}}(G_\theta(\boldsymbol{z}^{(i)}))$
    Clip weight $\boldsymbol{w}$ with range $[-c, c]$
  end for
  Sample $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p_z$
  Update generator using RMSProp with learning rate lr based on objective $-\frac{1}{m}\sum_{i=1}^{m} f_{\boldsymbol{w}}(G_\theta(\boldsymbol{z}^{(i)}))$
end while

There exists another way proposed by Gulrajani et al. (2017) to satisfy this constrains. The intuition is to add an additional term to the objective function as a regularizer such that it will penalize the objective if any gradient of the discriminator is away from 1.

They showed that the optimal discriminator have gradient 1 at almost everywhere under distribution p_d and p_g. To be precise,

Lemma 2. *p_r and p_g are two distributions defined on compact metric space $\boldsymbol{X}$, then there exists a 1-Lipschitz function f^* which is the solution of dual form EMD. Let π is the optimal coupling of the primal form EMD, then if f^* is differentiable and $\boldsymbol{x_t} = t\boldsymbol{x} + (1-t)\boldsymbol{y}$ with $0 \leq t \leq 1$,*

$$P_{(\boldsymbol{x},\boldsymbol{y})\sim\pi}[\nabla f^*(\boldsymbol{x}_t) = \frac{\boldsymbol{y} - \boldsymbol{x}_t}{||\boldsymbol{y} - \boldsymbol{x}_t||}] = 1 \tag{2.18}$$

Proof. Sketch of the proof: Based on the fundamental Theorem 5.10 of Villani (2008), it shows that

$$|f^*(\boldsymbol{x}_t) - f^*(\boldsymbol{x})| = t|\boldsymbol{y} - \boldsymbol{x}| \tag{2.19}$$

Then define

$$v = \frac{\boldsymbol{y} - \boldsymbol{x}_t}{||\boldsymbol{y} - \boldsymbol{x}_t||} = \frac{\boldsymbol{y} - \boldsymbol{x}}{||\boldsymbol{y} - \boldsymbol{x}||} \tag{2.20}$$

Thus, by definition of derivative and equation 2.19, we can show

$$\frac{\partial}{\partial v} f^*(\boldsymbol{x}_t) = 1 \tag{2.21}$$

Thus, we use Pythagoras theorem and definition of Lipschitz constraints, they show

$$\begin{aligned} 1 &\leq ||\nabla f^*(\boldsymbol{x}_t)||^2 \\ &= |\frac{\partial}{\partial v} f^*(\boldsymbol{x}_t)|^2 + |\nabla f^*(\boldsymbol{x}_t) - v\frac{\partial}{\partial v} f^*(\boldsymbol{x}_t)|^2 \\ &= 1 + |\nabla f^*(\boldsymbol{x}_t) - v|^2 \\ &\leq 1 \end{aligned} \tag{2.22}$$

Thus, $\nabla f^*(\boldsymbol{x}_t) = v$ □

Therefore, based on this **Lemma**, the new objective function is

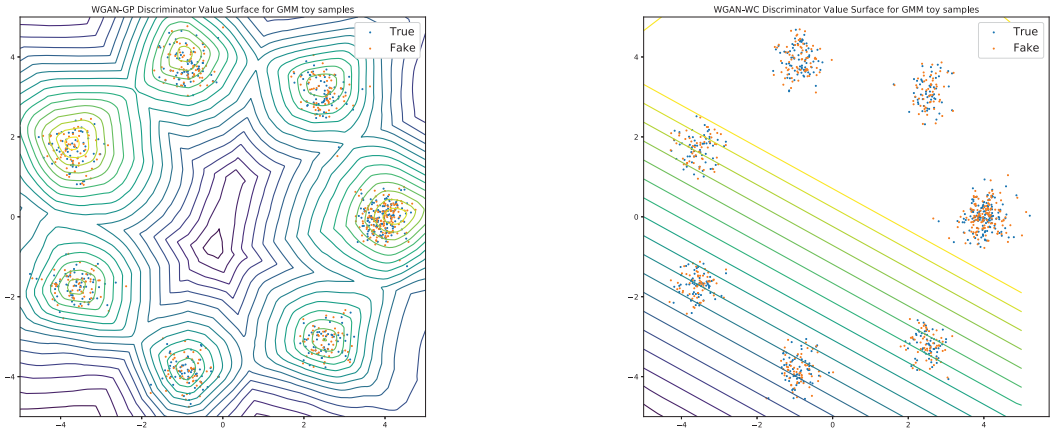
$$L = \mathbb{E}_{\boldsymbol{x} \sim p_d}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z} \sim p_z}[f(G(\boldsymbol{z}))] - \lambda \mathbb{E}_{\hat{\boldsymbol{x}} \sim p_{\hat{x}}}[(||\nabla_{\hat{\boldsymbol{x}}} f(\hat{\boldsymbol{x}})||_2 - 1)^2] \tag{2.23}$$

The first two terms are WGAN loss function without constraints and the third term are the regularizer. $\hat{\boldsymbol{x}}$ are the interpolated points between true data $\boldsymbol{x}$ and generated sample $G(\boldsymbol{z})$. In limit of the high λ, the optimal discriminator under this objective will be still optimal under true dual EMD. Therefore, in the non-parametric limits (enough capacity of discriminator, infinite training samples, infinite training epoch), the optimal cost recovered by this objective will be true EMD between p_r and p_g. We call this method WGAN-GP. The algorithm is described below.

Algorithm 3 WGAN-GP,$\lambda = 10$, nc=5, $lr = 0.0001$, $\beta_1 = 0.5$,$\beta_2 = 0.9$

while Not converge **do**
  for $t = 1, \ldots, nc$ **do**
    Sample $\{\boldsymbol{x}^{(i)}\}_{i=1}^m \sim p_d$, $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p_z$ and $\{\varepsilon^{(i)}\}_{i=1}^m \sim U[0,1]$
    $\hat{\boldsymbol{x}}^{(i)} = \varepsilon^{(i)}\boldsymbol{x}^{(i)} + (1 - \varepsilon^{(i)})G(\boldsymbol{z}^{(i)}))$ for $i = 1, \ldots, m$
    $L = \frac{1}{m}\sum_{i=1}^m f_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) - f_{\boldsymbol{w}}(G(\boldsymbol{z}^{(i)})) + \lambda\,(||\nabla_{\hat{\boldsymbol{x}}^{(i)}} f_{\boldsymbol{w}}(\hat{\boldsymbol{x}}^{(i)})||_2 - 1)^2$
    Update $\boldsymbol{w}$ using Adam$(L, lr, \beta_1, \beta_2)$
  end for
  Sample $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p_z$
  Update $G(\cdot)$ using Adam$(\frac{1}{m}\sum_{i=1}^m -f_{\boldsymbol{w}}(G(\boldsymbol{z}^{(i)})), lr, \beta_1, \beta_2)$
end while

Now we do some simple toy examples to demonstrate the difference of the two methods. The first example is to show that WGAN-WC severally damages the modeling power and tends to over-simplify the true EMD surfaces, whereas the WGAN-GP behaves much better. The toy examples are sampled from 7 component GMM and the generator is fixed by adding noise to the true data followed by a random shuffle. **Figure** 2.3 shows the discriminator value surface learned by two different methods. From the this plot, the value

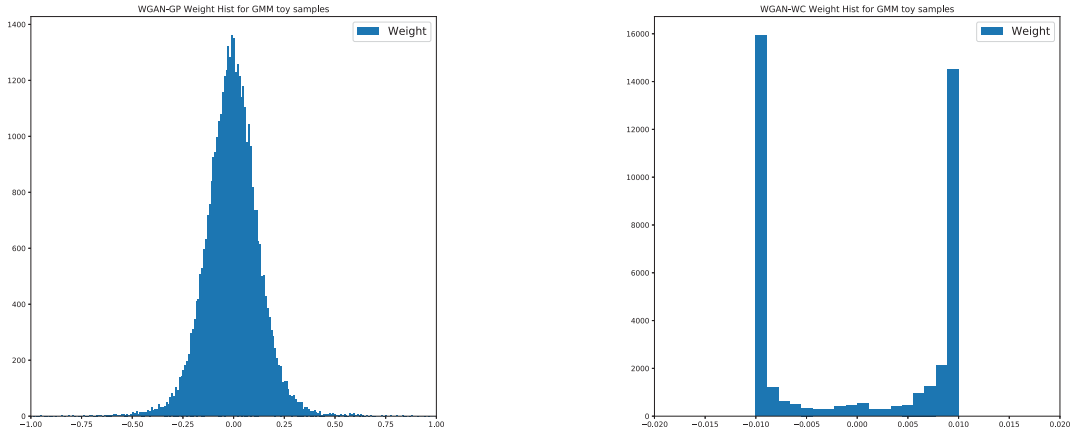


(a) WGAN-GP (b) WGAN-WC

Fig. 2.3 The learned discriminator value surface. The true samples are drawn from 7-component GMM and fake samples are generated by adding Gaussian noise with $\mu = 0$ and $\sigma = 0.1$ to the true data.

surface learned by WGAN-WC over-simplify the actual data distribution. In other words, the weight clipping tends to stop the neural network to learn the actual EMD surface, instead, it encourages it to learn a over-smoothed surface. WGAN-WC cannot capture the high moment features of the data. On the other hand, WGAN-GP method successfully capture all correct modes of the data with reasonable smoothness. This example shows that the weight-clipping method strongly discourage the discriminator to learn complicated surface. The reason for this behavior can be seen in the next experiment.

The second experiment is to show the weights distributions for the same toy examples. **Figure** 2.4 shows the distributions of the learned weights through a histogram. We can see that the WGAN-WC method tries to push the weights to the limits to capture the data structure. Thus, if we restrict our weight within a small range, then the modeling power of



(a) WGAN-GP (b) WGAN-WC

Fig. 2.4 The weights distribution of the two methods. The training data are the same as before. The generator is also fixed by adding noise to true data.

neural nets are hindered. On the contrast, the WGAN-GP does not directly put a constraints on the weights, instead, it put restrictions on the smoothness of the neural net. Thus, most of the weights are concentrated on the low value and few of them are in the high range. So the modeling power is reasonably reserved and it can capture more complicated data features.

The above algorithms are based on the dual form Wasserstein distance and the assumption is that the distance measure between points is Euclidean distance. This limits the possibilities of using other alternative distance measures. In the next section, we will introduce the primal-form based WGAN using entropy regularized linear programming (Cuturi, 2013) called **Sinkhorn Algorithm** (Genevay et al., 2017).

2.3.4 Primal form based WGAN

In subsection 2.3.1, the equation 2.9 defines the primal-form EMD. In primal form, choice of distance measure is not restricted and we don't need to worry about Lipschitz continuity. Thus, this gives us more design freedom. In addition, the LP formulation is a convex optimization with a unique optimum that gives the true EMD between two distributions under finite supports. However, the LP formulation is known to be computationally expensive for high dimensional data. In addition, the solution of LP is the optimal vertices in a high dimensional convex polytope. Thus, it won't be very smooth. This will cause problems for

the update in generator.

These potential difficulties are the primary reason that the dual EMD is adopted in the original WGAN. However, a new approximation method for optimal transport problem has been proposed by Cuturi (2013) intended to solve the above problems. The intuitive idea is to use entropy regularization in LP to enforce smoothness of the solution and reduce the search space. The regularized problem is strongly convex.

Apart from the method mentioned above, there is another way proposed by Bousquet et al. (2017) to solve the primal-from EMD by introducing a encoder network and a regularizer to enforce similarity between aggregated posterior and prior distribution . In fact, this method is very similar to the Autoencoder training with a regularizer. The objective function is

$$\inf_{Q(\boldsymbol{z}|\boldsymbol{x})} \mathbb{E}_{\boldsymbol{x}\sim p_d}\mathbb{E}_{\boldsymbol{z}\sim Q(\boldsymbol{z}|\boldsymbol{x})}[c(\boldsymbol{x},G(\boldsymbol{z}))] + \lambda D(Q(\boldsymbol{z})||\boldsymbol{Z})) \tag{2.24}$$

where $Q(\boldsymbol{z}|\boldsymbol{x})$ is the encoder network, $c(\cdot,\cdot)$ is the distance measure and $D(\cdot,\cdot)$ is the distance measure between distributions. The first term is the reconstruction loss in the Autoencoder objective function (e.g. $c(\cdot,\cdot)$ is l_2 norm). The second term is the penalty to ensure aggregated posterior is like the prior. This forces the $Q(\boldsymbol{z}|\boldsymbol{x})$ to be true posterior. Because the $Q(\boldsymbol{z}|\boldsymbol{x})$ are implicit model defined by MLP, so information-based metric is hard to use. Analogous to regularizer used in Adversarial Autoencoder (Makhzani et al., 2015), the loss $D(\cdot||\cdot)$ are replaced to GAN loss. In this thesis, we will not explore this methods in details, though, this methods provides a unifying framework for GAN, Adversarial Autoencoder and Adversarial Variational Bayes (Mescheder et al., 2017). In addition, it also gives us another reason to use primal form GAN: they showed if the true optimum f^* in dual form WGAN is not reached exactly, the effect on the gradient in the dual formulation can be arbitrarily large (**Proposition 3** (Bousquet et al., 2017)).

Now, we will focus on the entropy regularized primal EMD. Before we dive into the Sinkhorn algorithm, we need to slightly change the LP formulation. From the section 2.3.1, one of the constraints of LP is $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ where vector $\boldsymbol{b}$ contains marginal likelihood of p_d and p_g. However, for distributions with continuous infinite support, we do not know these two probabilities. Thus, we can use the idea of **Mini-batch sampling loss**. To be specific, the EMD loss $W(p_d, p_g)$ is replaced by $W(\hat{p}_d, \hat{p}_g)$, where $\hat{p}_d = \frac{1}{m}\sum_{i=1}^{m}\delta_{\boldsymbol{x}_i}$ and $\hat{p}_g = \frac{1}{m}\sum_{i=1}^{m}\delta_{G(\boldsymbol{z}_i)}$. The distance vector $\boldsymbol{c}$ is defined by taking the distance measure $c(\cdot,\cdot)$ on all possible pairs of sampled data. Therefore, the constraints is changed to $\boldsymbol{A}\boldsymbol{x} = \frac{1}{m}\mathbf{1}_{2m}$ where $\mathbf{1}_n = [1,1,\ldots,1]^T$

with n elements. Thus, the LP formulation for empirical distribution $\hat{p}_d$ and $\hat{p}_g$ is

$$\begin{aligned} &\min_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \\ &\text{s.t. } \boldsymbol{A}\boldsymbol{x} = \frac{1}{m}\mathbf{1}_{2m} \\ &\boldsymbol{x} \geq 0 \end{aligned} \tag{2.25}$$

Solving the above LP problem is both numerically unstable and expensive. So the entropy regularization LP is introduced by Cuturi (2013). First, we define a probability simplex $\Sigma_d = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T \mathbf{1}_d = 1\}$, thus $\hat{p}_d, \hat{p}_g \in \Sigma_d$. Then we further define a transportation polytope for $\hat{p}_d$ and $\hat{p}_g$ called $U(\hat{p}_d, \hat{p}_g)$:

$$U(\hat{p}_d, \hat{p}_g) = \{P \in \mathbb{R}^{d \times d} | P\mathbf{1}_d = \hat{p}_d, P^T \mathbf{1}_d = \hat{p}_g\} \tag{2.26}$$

From the basic information theoretical inequality (Cover and Thomas, 2012), we have

$$\forall \hat{p}_d, \hat{p}_g \in \Sigma_d,\ \forall P \in U(\hat{p}_d, \hat{p}_g),\ h(P) \leq h(\hat{p}_d) + h(\hat{p}_g) \tag{2.27}$$

This inequality indicates the entropy of joint probability table P is less than the independence table h($\hat{p}_d \hat{p}_g^T$) (Good et al., 1963). Now define a subset $U_\alpha(\hat{p}_d, \hat{p}_g) \subset U(\hat{p}_d, \hat{p}_g)$:

$$U_\alpha(\hat{p}_d, \hat{p}_g) = \{P \in U(\hat{p}_d, \hat{p}_g) | KL(P || \hat{p}_d \hat{p}_g^T) \leq \alpha\} \tag{2.28}$$

Due the the convexity of entropy, this subset is also convex. This subset can be interpreted as a set of tables P in $U(\hat{p}_d, \hat{p}_g)$ which have sufficient entropy with respect to $h(\hat{p}_d)$ and $h(\hat{p}_g)$. We can see if the table P only contains independent joint probabilities, then the KL term is zero and the search space is reduced to a 'single point' in the polytope $U(\hat{p}_d, \hat{p}_g)$. Therefore, the α controls the size of the search space. When it is large enough, the subset $U_\alpha(\hat{p}_d, \hat{p}_g) = U(\hat{p}_d, \hat{p}_g)$ (like **figure** 2.5) According to the max-entropy principle, the objective function with an entropic constraint will look for the most smooth solution P^* given level of the cost. The intuition is that for a given empirical distribution $\hat{p}_d$ and $\hat{p}_g$, finding plausible transporting plan P with low cost (plausibility is measured by $h(P)$) is more robust and informative than the extreme plan with very low cost. This alleviates the smoothness problem. Next, we show that the solution to the above entropy regularized LP formulation can be found by Sinkhorn-Knopp fixed point algorithm.

Theorem 2. *Sinkhorn-Knopp Theorem: For a square matrix $\boldsymbol{A}$ with strictly positive elements, there exists a strictly positive diagonal matrix $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ such that $\boldsymbol{D}_1\boldsymbol{A}\boldsymbol{D}_2$ is doubly stochastic.*

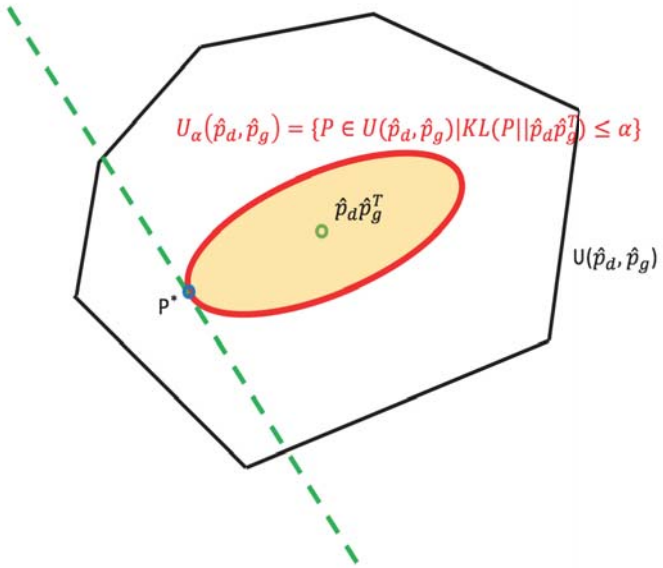


Fig. 2.5 The schematic view of the entropy regularization LP

Theorem 3. *For a pair of empirical distributions $\hat{p}_d$, $\hat{p}_g$, distance matrix $\boldsymbol{M} \in M = \{\boldsymbol{M} \in \mathbb{R}^{d\times d} : \forall i \leq d, m_{ii} = 0; \forall i,j,k \leq d, m_{ij} \leq m_{ik} + m_{kj}\}$ and α, there exist a λ such that Sinkhorn distance $d_{M,\alpha}(\hat{p}_d, \hat{p}_g) = d_M^\lambda(\hat{p}_d, \hat{p}_g)$. And there is a unique optimum solution P^λ to $d_M^\lambda(\hat{p}_d, \hat{p}_g)$ which is in the form $u_i e^{-\lambda m_{ij}} v_j$. u and v are two non-negative vector uniquely defined up to multiplicative factor.*

Proof. Refer to the appendix. □

Thus, based on the theorem, we can use a iterative procedure to approximate the solution P^λ called **Sinkhorn-Knopp fixed point algorithm**. Then with the optimal transportation matrix, we can evaluate the EMD between $\hat{p}_d$ and $\hat{p}_g$. With the re-parametrization trick introduced by Kingma and Welling (2013), we can update the generator by reducing the EMD. The proposed algorithm for Generative models called **Sinkhorn Autodiff** formally proposed by Genevay et al. (2017). The flow diagram of Sinkhorn Autodiff is shown in **Figure** 2.6.

For simplicity, we denote λ as $\frac{1}{\varepsilon}$, this is because as λ increases the entropy of solution $h(P)$ decreases, this represents increasing α. Therefore, ε in some sense represents the 'error' between true EMD and Sinkhorn EMD.

The detailed algorithm for Sinkhorn Autodiff are shown in Algorithm 4. The $\odot$ represents the element-wise product, $\div$ represents the component-wise division and $\mathbf{1}_n$ is the n

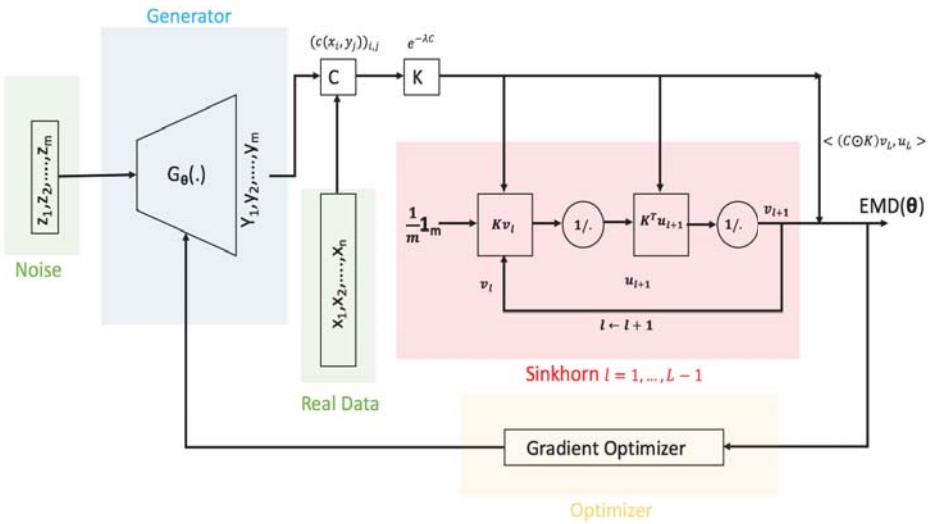


Fig. 2.6 Flow diagram of Sinkhorn Autodiff algorithm with mini-batch noise $\{\boldsymbol{z}_{(i)}\}_{i=1}^{m}$ and real data sample $\{\boldsymbol{x}_{(i)}\}_{i=1}^{n}$. The θ are the parameters of generator and the optimizer for θ are normally chosen to be Adam.

Algorithm 4 Sinkhorn Autodiff

Input: Noise $\{\boldsymbol{z}_{(j)}\}_{j=1}^{m}$, error ε, Sinkhorn iteration L, distance measure $c(\cdot)$

for $k = 1, 2, \ldots$ **do**
  Sample mini-batch $\{\boldsymbol{y}_{(j)}\}_{j=1}^{m}$ using $\boldsymbol{y}_{(j)} = G(\boldsymbol{z}_{(j)})$.
  Sample read data $\{\boldsymbol{x}_{(i)}\}_{i=1}^{n}$
  $\forall(i,j)$, $C_{i,j} = c(\boldsymbol{x}_{(i)}, \boldsymbol{y}_{(j)})$ and $K_{i,j} = \exp(-\frac{C_{i,j}}{\varepsilon})$
  $v_0 = \frac{1}{n}\mathbf{1}_n$
  for $l = 1, \ldots, L$ **do**
    $u_l = \frac{\frac{1}{n}\mathbf{1}_n}{Kv_{l-1}}$ and $v_l = \frac{\frac{1}{n}\mathbf{1}_n}{K^T u_l}$
  end for
  E=$< (K \odot C)v, u >$
  Update Generator using Adam(E,θ)
end for

dimensional vector with elements 1.

2.4 Related work and variations

Apart from the methods mentioned above, there are large number of variations of GAN. These methods can be roughly categorized into three different areas. We will focus on the

first two area, i.e. **Network Architecture** and **Autoencoder-based GAN**, especially the **FlowGAN** method.

2.4.1 Network Architecture

In the original GAN, the discriminator and generator are chosen to be MLP.Instead, fully connected CNN can also be used as proposed by Radford et al. (2015). They use the same training objective as the original GAN and propose some practical suggestions on the network topologies. This method is called DCGAN which achieves very good results on generating sharp and realistic images and is also a strong candidate for unsupervised learning.

The unsupervised learning capacity of GAN is further investigated and improved by Chen et al. (2016). A new method called **InfoGAN** is proposed to learn disentangled representations in a completely unsupervised manner. The network architecture are based on DCGAN and conditional GAN (Mirza and Osindero, 2014) but an additional mutual information between generator output and input is added to the objective function. Thus, the generator will try to both generate realistic images and also be highly relevant to the latent code inputed to the generator. Further improvement to this method is proposed just before this thesis using a semi-supervised training to improve the convergence and synthetic image quality (Spurr et al., 2017).

Apart from the unsupervised learning, another problem of the original GAN has also been investigated. Specifically, the evaluation of original GAN training is tricky and is mainly assessed by generation quality. Apart from human judgment, inception score and MODE score Che et al. (2016) can be also used. However, these are indirect methods to assess training quality. Therefore, a new generator structure called **FlowGAN** (Grover et al., 2017) is proposed which allows for explicit likelihood evaluation of generator output. The generator is chosen to be **Normalizing Flow**. One useful property of normalizing flow is that the output density can be explicitly evaluated using change of variable rule. The details of normalizing flow will be introduced in Chapter 3.

One of the problems of GAN is mode-dropping, which means the generated images are only from part of the distribution. This means the generator distribution will highly concentrate on high probability areas of targeting distribution. We will show this behavior using toy examples in experiment chapter. In fact recent work in approximate inference suggests that the GAN training obtains much worse likelihood than the MLE (Wu et al., 2016) and the generation quality and log likelihood are largely uncorrelated (Theis et al., 2015). Thus, with

the exact likelihood evaluation of flow, FlowGAN proposes a alternative objective function which combines WGAN and MLE objective. Therefore, ideally, a middle ground between realistic image and reasonable likelihood can be obtained. The objective function is

$$\min_{G(\cdot)} \max_{f(\cdot)} \mathbb{E}_{\boldsymbol{x}\sim p_d}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p_z}[f(G(\boldsymbol{z}))] - \lambda_1 \mathbb{E}_{\hat{\boldsymbol{x}}\sim p_{\hat{x}}}[(||\nabla_{\hat{\boldsymbol{x}}} f(\hat{\boldsymbol{x}})||_2 - 1)^2] - \lambda_2 \mathbb{E}_{\boldsymbol{x}\sim p_d}[\log p_G(\boldsymbol{x})] \tag{2.29}$$

The λ_1 and λ_2 controls the weight of gradient penalty and ML term. With high λ_2, it will recover the MLE solution. In the experiment section, we will demonstrate some basic property of FlowGAN using toy examples.

2.4.2 Autoencoder-based GAN

Another research direction tries to combine VAE (Kingma and Welling, 2013) and GAN training together. Some of the well-known work is to regard the generator as the decoder and then introduce another encoder network. In the context of VAE, the objective function can be split into a reconstruction error which will penalize if the reconstructed images $\hat{\boldsymbol{x}}$ where $\boldsymbol{x} \to \boldsymbol{z} \to \hat{\boldsymbol{x}}$ are different from the real image $\boldsymbol{x}$ and a regularizer penalizing the difference between aggregated posterior $\mathbb{E}_{\boldsymbol{x}}[q(\boldsymbol{z}|\boldsymbol{x})]$ and $p(\boldsymbol{z})$.

The VAE uses KL as the distance measure between $q(\boldsymbol{z})$ and $p(\boldsymbol{z})$. The Adversarial Autoencoder (AAE) (Makhzani et al., 2015) uses GAN training as alternative to drive $q(\boldsymbol{z})$ to be like $p(\boldsymbol{z})$. This Auto-encoder based GAN can alleviate the mode-dropping problem. Intuitively, if mode-dropping happens, then the decoder can only generate limited variety of images, thus, the encoder will only map these images to limited latent vectors, which will induce large penalty to the objective function to avoid mode-dropping

In fact, a recent proposed method called CycleGAN (Zhu et al., 2017) achieve excellent results in unpaired image-to-image translation. The assumption is the cycle-consistency of the image, which means the decoder and encoder are consistent to each other. This consistency means the cycle translation of each image domain should remain similar to the original domain, i.e. $F(G(\boldsymbol{x}))$ should remain same as $\boldsymbol{x}$ where $G(\cdot)$ and $F(\cdot)$ are translations of $\boldsymbol{X} \to \boldsymbol{Y}$ and $\boldsymbol{Y} \to \boldsymbol{X}$ respectively. This loss is exactly the same as the reconstruction error in the VAE and Adversarial Autoencoder (if l_2 norm is used as distance measure in CycleGAN). The objective function of CycleGAN consists of two cycle-consistency loss and two GAN loss (equivalent to regularizer in Adversarial Autoencoder) between translated image domain and target domain. This objective function is the sum of two Adversarial Autoencoder objectives in two domains.

The AAE method is also closely related to the Wasserstein GAN. Specifically, the VE-GAN (Bousquet et al., 2017) shows that the primal form Wasserstein GAN with l_2 distance measure and GAN loss regularizer is equivalent to objective function of AAE up to multiplicative constant.

The additional ML term in FlowGAN model can also be interpreted in this Autoencoder-based GAN framework. The additional ML term behaves like the regularizer to make sure $q(\boldsymbol{z})$ is similar to $p(\boldsymbol{z})$. Due to the bijective mapping property of normalizing flow, this is equivalent to maximizing the log likelihood.

2.4.3 Alternative distances

Another popular variations of GAN is to design alternative distance measures between distributions. For example, Wasserstein GAN can fall into this category. Apart from Wasserstein distance, there are many alternative distance. For example, relaxed Wasserstein distance which uses Bregman divergence as a distance measure between points. This methods shows faster convergence if the Bregman divergence is chosen to be KL.

Another distance alternative to Wasserstein is called Cramer distance (Bellemare et al., 2017). They showed that Wasserstein distance is biased for sampled gradients whereas Cramer distance is unbiased. There are still many other objective functions alternative to the one we mentioned in this thesis like MMD GAN (Li et al., 2017), McGAN (Mroueh et al., 2017), etc. We will not give details about these distances because they are irrelevant to the following materials.

Chapter 3

Normalizing Flow

In this section, we will give a short introduction on a model which we will later be used as generator called **normalizing flow**. Specifically, we will focus on a specific flow called **Nonlinear Independent Component Estimation** (NICE) proposed by Dinh et al. (2014), where ancestral sampling and likelihood evaluation are easy. At the end of this chapter, we will discuss one limitation of NICE and use this as the motivation for the novel improvement which can greatly enhance its performance.

3.0.1 Introduction

To model complicated data structures, one of the choices is to use neural network. However, the implicit property of neural network makes it hard to evaluate the probability of its output. In fact, the **Lemma 1** in Arjovsky and Bottou (2017) showed that the distribution of the output from MLP is supported by joint of low dimensional manifolds, thus it is not **absolute continuous** According to **Radon–Nikodym** theorem, it does not have a proper density.

One alternative candidate uses the principle of normalizing flow (Tabak and Turner, 2013; Tabak et al., 2010) and is essentially a sequence of **bijective** mappings which will map the random variable from simple distribution to incrementally complicated one. The key idea is **bijective** mapping which will allow us to compute the likelihood using change of variable rule. For example, for a bijective mapping $\boldsymbol{x} = g(\boldsymbol{z})$ and $\boldsymbol{z} \sim p_z$, the likelihood of mapped variable is

$$p_x(\boldsymbol{x}) = p_z(\boldsymbol{z}) |\det \frac{\partial g}{\partial \boldsymbol{z}}|^{-1} \tag{3.1}$$

The above equation is directly from Jacobian property of invertible functions. Thus with a sequence mapping, the log probability is defined as

$$\begin{aligned} \boldsymbol{x}_K &= f_K \circ f_{K-1} \circ \ldots \circ f_1(\boldsymbol{z}) \\ \log p_x(\boldsymbol{x}_K) &= \log p_z(\boldsymbol{z}) - \sum_{k=1}^{K} \log |\det \frac{\partial g_k}{\partial \boldsymbol{x}_{k-1}}| \end{aligned} \tag{3.2}$$

In fact, this transformation allows us to evaluate the log probability without explicitly knowing transformed variable $\boldsymbol{x}_K$. This is often referred as law of the unconscious statistician (LOTUS). Thus, there are two kinds of normalizing flows. The first kinds of flows allows us to explicitly sample from the transformed distribution such as NICE and Real-NVP (Dinh et al., 2016). Another type of flow only allows us to evaluate the log probability whereas sampling from transformed distribution is difficult. These type of flows are normally used in variational inference such as planar flow and rotation flow (Rezende and Mohamed, 2015). As in the context of GAN, we need the samples from the transformed distribution, thus, only the first type of flow is considered.

The evaluation of Jacobian determinant is computationally expensive, normally $O(n^3)$. Therefore, the challenge is to design flexible flows with simple Jacobian structures. One solution is to design the flow with lower triangular Jacobian matrix. Especially, in the next section, we will show by using special structures, the **determinant** of Jacobian is 1.

3.0.2 Nonlinear Independent Component Estimation (NICE)

In this section, we will give details about a specific flow called NICE. The core idea of NICE is to split the variables into two blocks $(\boldsymbol{z}_1, \boldsymbol{z}_2)$ called (plain, key) pair. The transformation will retain the original key value and encode the plain by an special invertible function $e(\text{plain}, \text{m(key)})$ and m is an arbitrary complex function called coupling function. The model structure is shown in **Figure** 3.1.

The computations described above are grouped in one layer called **coupling layer**. Formally, let $\boldsymbol{z} \in \mathbb{R}^D$ and a partition $[I_1, I_2]$ on $[1, D]$. The function $m(\cdot)$ is defined on $\mathbb{R}^d$ where $d = |I_1|$. We can also define a special invertible mapping with respect to the first argument given the second $e(\boldsymbol{z}_{I_2}, m(\boldsymbol{z}_{I_1}))$. It is defined on $\mathbb{R}^{D-d} \times \mathbb{R}^d \to \mathbb{R}^{D-d}$. We call this function

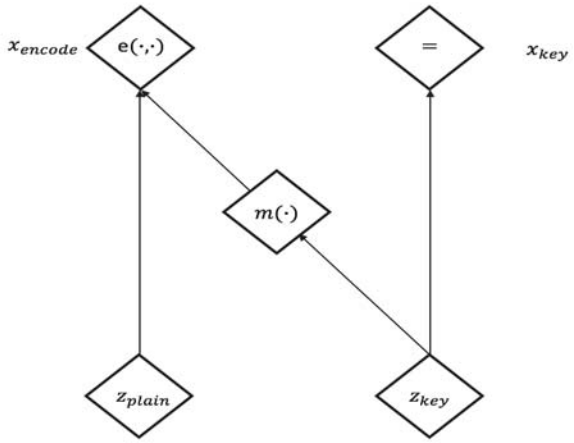


Fig. 3.1 The computational structure of the coupling layer

$e(\cdot,\cdot)$ as **coupling law**. Thus inside the coupling layer, the transformation is defined as

$$\begin{aligned} \boldsymbol{x}_{I_1} &= \boldsymbol{z}_{I_1} \\ \boldsymbol{x}_{I_2} &= e(\boldsymbol{z}_{I_2}, m(\boldsymbol{z}_{I_1})) \end{aligned} \tag{3.3}$$

Therefore, the Jacobian of this transformation can be easily evaluated as

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}} = \begin{bmatrix} \frac{\partial \boldsymbol{x}_{I_1}}{\partial \boldsymbol{z}_{I_1}} & \frac{\partial \boldsymbol{x}_{I_1}}{\partial \boldsymbol{z}_{I_2}} \\ \frac{\partial \boldsymbol{x}_{I_2}}{\partial \boldsymbol{z}_{I_1}} & \frac{\partial \boldsymbol{x}_{I_2}}{\partial \boldsymbol{z}_{I_2}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_d & \boldsymbol{0} \\ \frac{\partial \boldsymbol{x}_{I_2}}{\partial \boldsymbol{z}_{I_1}} & \frac{\partial \boldsymbol{x}_{I_2}}{\partial \boldsymbol{z}_{I_2}} \end{bmatrix} \tag{3.4}$$

Therefore, the Jacobian matrix is a lower triangle matrix with its determinant $\det \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}} = \frac{\partial \boldsymbol{x}_{I_2}}{\partial \boldsymbol{z}_{I_2}}$.

The ancestral sampling is also trivial.

$$\begin{aligned} \boldsymbol{z}_{I_1} &= \boldsymbol{x}_{I_1} \\ \boldsymbol{z}_{I_2} &= e^{-1}(\boldsymbol{x}_{I_2}, m(\boldsymbol{x}_{I_1})) \end{aligned} \tag{3.5}$$

Due to the fact that we do not need to evaluate the inverse of function $m(\cdot)$, thus, we can use arbitrary differentiable function $m(\cdot)$ such as MLP or CNN.

We can choose the coupling law as simple addition. This is called **additive coupling layers**. The transformation is

$$\begin{aligned} \boldsymbol{x}_{I_2} &= \boldsymbol{z}_{I_2} + m(\boldsymbol{z}_{I_1}) \\ \boldsymbol{z}_{I_2} &= \boldsymbol{x}_{I_2} - m(\boldsymbol{x}_{I_1}) \end{aligned} \tag{3.6}$$

Therefore, as defined in equation 3.4, the diagonal elements of Jacobian for additive coupling layer are 1, thus, the det $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{z}} = 1$.

The above procedure defines single coupling layer, in fact, we can stack single layer on top of each other to form a multi-layer structure. This is identical as applying a sequence of invertible mappings to the initial random variables. Each coupling layer has a unit Jacobian determinant, therefore, the multi-layer structure will still have the same determinant value. This represents the constant volume in each of transformation. To address this, we can explicitly introduce a diagonal scaling matrix $\boldsymbol{S}$. Each output dimension x_i will become $S_{ii}x_i$. This term controls the 'importance' of each dimension just like the eignspectrum of PCA. The important dimensions are the manifolds learned by algorithm. The overall log likelihood of NICE is

$$\log(p_x(\boldsymbol{x})) = \sum_{i=1}^{D} \log(p_z(f_i(x_i))) + \log(|S_{ii}|) \tag{3.7}$$

Notice that the transformation is from real data $\boldsymbol{x}$ to latent variable $\boldsymbol{z}$ where $\boldsymbol{z}$ has simple distributions. $f(\boldsymbol{x}) = g^{-1}(\boldsymbol{x})$ is the inverse mapping. This formulation is in opposite direction as the one we introduced above, but due to the simplicity of ancestral sampling and likelihood evaluation, transformation from other direction is trivial. Based on the objective function and simplicity of based distribution p_z, this will encourage the S_{ii} to be small while the $\log(S_{ii})$ will prevent S_{ii} to reach 0. Thus, the larger the S_{ii} is, the less important the corresponding dimension is.

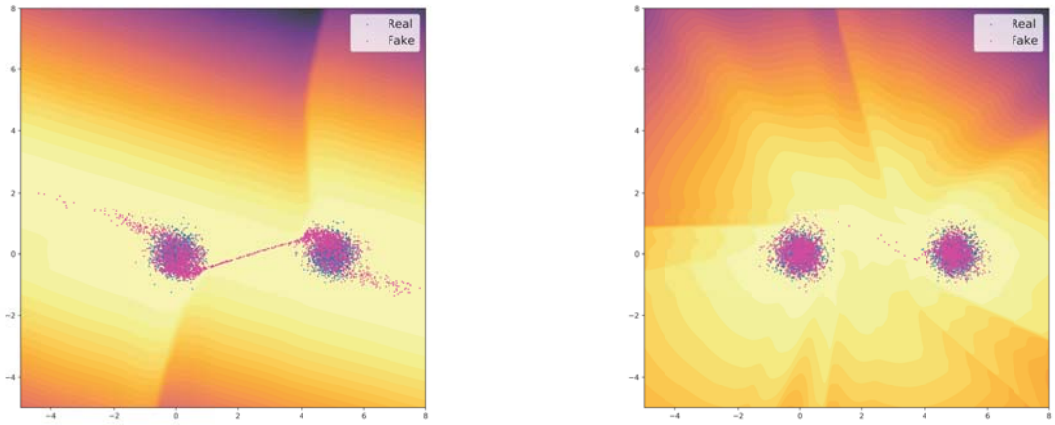
In the following chapters, for single flow, the default number of coupling layer is 4 and the choice for function $m(\cdot)$ is MLP with ReLu activation function or CNN with fully connected network. The default coupling law is additive coupling.

3.0.3 Limitations

Although the normalizing flow has achieved excellent results on modeling complicated data distributions, there are still some limitations on this model. In this section, we will focus one particular problem, that is, normalizing flow is difficult to model distributions with separated mode structures.

To be precise, the distribution from normalizing flow is generated by using a sequence of invertible mappings on the base distributions. In the context of distribution with separated mode and Gaussian base distribution, the flow has to separate part of the Gaussian distribution to account for the different modes. However, due to the continuous of base distribution, they

tends to generated a 'bridge' between the modes. In other words, the normalizing flow cannot output clean separations with few coupling layers. **Figure** 3.2 shows a simple demonstration with 2-component GMM, we can clearly see a bridge between modes. If more coupling layers are used, the situation can be alleviated but it also requires more data for stable training and may suffer from vanishing gradient if the layer is too deep. In fact, later in the experiment section, we will demonstrate more complicated examples and even with deep coupling layers, it is still difficult for the flow to capture correct structures.



(a) 2 Coupling Layer

(b) 4 Coupling Layer

Fig. 3.2 Simple GMM example to demonstrate the limitation of normalizing flow. Although the bridging effect is less severe for 4 coupling layers, but the contour surface is less smooth and is quite different from the true 2-component GMM surface.

Chapter 4

Mixture Flow Model

In this chapter, we will present a novel improvement to the existing normalizing flow model which achieves much better results. In addition, we also gives the objective function and training procedure of this model for GAN and GAN+ML training. In the end, based on the mixture of generator, we will give new GAN structure which allows the 'personalized' discriminator to guide each flow component.

4.1 Motivation

Based on the previous chapter, we give a brief introduction to a specific normalizing flow called NICE. In the end of that chapter, we discuss the potential limitation when modeling the distributions with separated modes. In fact, we can try to increase the modeling capacity of NICE to solve this issue. The most simple solution is to increase the number of coupling layers, capacity of function $m(\cdot)$. However, this approach often requires more training data and potentially suffers from vanishing gradient problem.

Another alternative approach is to use the idea of **mixture** models. The most well-known and simple example is GMM. In fact, this idea in the context of GAN is not brand-new, the MIX+GAN model proposed by Arora et al. (2017) uses a mixture of generators and discriminators. The weight term are leaned through GAN training objective via back-propagation. Empirically, this model shows an improvement compared to single generator and discriminator. DeLiGAN (Gurumurthy et al., 2017) also uses the idea of mixture model which can learn much faster with limited number of data. Our approach exploits the third mixture model options by exploiting the maximum likelihood training. The difference between our model with the MIX+GAN and DeLiGAN will be discussed in the related work section.

4.2 Mixture of Flow

Assume we have real data $\boldsymbol{x} \in \mathbb{R}^D$ defined in the space $\boldsymbol{X}$ and latent noise $\boldsymbol{z} \in \mathbb{R}^D$ defined in space $\boldsymbol{Z}$. Due to the bijective property of flow, the dimension of latent noise and real data have to be the same. We can also define a bijective mapping $g_i(\cdot) : \boldsymbol{Z} \to \boldsymbol{X}$ where $i \in [1,T]$ and T is the total number of component in the mixture model. Similarly, the inverse mapping $f_i(\cdot) : \boldsymbol{X} \to \boldsymbol{Z}$ Thus, each bijective mapping $g_i(\cdot)$ can be interpreted as a NICE model with additive coupling law and multiple coupling layers. Thus, we can explicitly evaluate the log likelihood of i^{th} component:

$$\log p(\boldsymbol{x}|c=i) = \sum_{d=1}^{D} \log(p_z(f_i(\boldsymbol{x}))) + \log(|S_{i,dd}|) \tag{4.1}$$

Therefore, the overall log likelihood is defined as

$$\log(p(\boldsymbol{x})) = \log(\sum_{i=1}^{T} p(\boldsymbol{c=i}) p(\boldsymbol{x}|c=i)) \tag{4.2}$$

Each NICE component can have similar model topologies but each has their own training parameters.

This mixture models will increase the modeling capacity compared to single flow and ideally, it can greatly alleviate the 'bridging effect' discussed in the single flow model.

4.2.1 ML Training Procedure

Due to the explicit likelihood evaluation, we can train the model using maximum likelihood. First, we give a **Expectation-Maximization** based training algorithm and then an alternative gradient-based training is given which can later be used in GAN+ML training.

Asssume the component weights are parametrized by θ and NICE is parameterized by ϕ. The formulation of the training is

$$\max_{\theta,\phi} \; log(p(\boldsymbol{x})) \tag{4.3}$$

Therefore, if we introduce a density $q(c|\boldsymbol{x})$ and use Jensen inequality, we have

$$\begin{aligned}
\log(p(\boldsymbol{x})) &= \log(\sum_{i=1}^{T} p(c=i)p(\boldsymbol{x}|c=i)) \\
&= \log(\sum_{i=1}^{T} q(c=1|\boldsymbol{x})\frac{p(\boldsymbol{x},c=i)}{q(c=i|\boldsymbol{x})}) \\
&\geq \sum_{i=1}^{T} q(c=i|\boldsymbol{x})\log(\frac{p(\boldsymbol{x},c=i)}{q(c=i|\boldsymbol{x})}) \\
&= \sum_{i=1}^{T} q(c=i|\boldsymbol{x})\log(\frac{p(c=i|\boldsymbol{x})p(\boldsymbol{x})}{q(c=i|\boldsymbol{x})}) \\
&= \log(p(\boldsymbol{x})) - KL(q(c=i|\boldsymbol{x})||p(c=i|\boldsymbol{x}))
\end{aligned} \tag{4.4}$$

Thus, we can see when $q(c=i|\boldsymbol{x})$ equals to the true posterior $p(c=i|\boldsymbol{x})$, the equality holds. Thus, in the **expectation** step at time t, we can let

$$q_t(c=i|\boldsymbol{x}) = \frac{p_{\phi_{t-1}}(\boldsymbol{x}|c=i)p_{\theta_{t-1}}(c=i)}{\sum_{i=1}^{T} p_{t-1}(\boldsymbol{x},c=i)} = p_{t-1}(c=i|\boldsymbol{x}) \tag{4.5}$$

This can be explicitly evaluated, therefore, in the maximization step,

$$\max_{\theta,\phi} \sum_{i=1}^{T} q_t(c=i|\boldsymbol{x})\log(\frac{p(c=i)p(\boldsymbol{x}|c=i)}{q_t(c=i|\boldsymbol{x})}) \tag{4.6}$$

Analogues to the GMM EM algorithm, at time t with N training data points, the maximized component weight

$$p(c=i) = \frac{1}{N}\sum_{n=1}^{N} q_t(c=i|\boldsymbol{x}_n) \tag{4.7}$$

The maximization of $p(\boldsymbol{x}|c=i)$ are achieved by using gradient descent optimizer like Adam.

Notice that at the beginning of the training, there will be huge oscillations in the updated value $p(c=i)$ due to the hard assignment and sensitivity to initialization. Therefore, we can achieve a more stable training by allowing the model to learn without changing component weight for certain number of iterations called C_{start}. Further, we can modify the update rule using

$$p_t(c=i) = p_{t-1}(c=i) + \lambda_c(p_t(c=i) - p_{t-1}(c=i)) \tag{4.8}$$

Thus, if $\lambda_c = 1$, it will recover the true EM updates.

Alternative to this EM approach, we can also train Mixflow by purely gradient based optimizer. The maximization of NICE component $p(\boldsymbol{x}|c=i)$ is done through gradient optimization. As for the component weight, we can parametrize them using a **softmax** function.

$$p(c=i)=\frac{\exp(\theta_i)}{\sum_{t=1}^{T}\exp(\theta_t)} \tag{4.9}$$

Therefore, by updating θ_i we will guarantee the sum-to-one constraints of prior weight. Therefore the algorithm for maximum likelihood training of Mixflow is given below.

Algorithm 5 Mixflow ML training

Input: Training data $\boldsymbol{x}$, component learning rate λ_ϕ,weight learning rate λ_c, C_{start}, epoch L, component number T, training data number N, EM training flag EM_flag

for $l=1\ldots,L$ **do**
  Sample $\{\boldsymbol{x}\}_{n=1}^{N}\sim p_x$
  Evaluate $q_t(c=i|\boldsymbol{x}_n)$ for $n=1,\ldots,N$ and $i=1,\ldots,T$
  if flag_EM=True **then**
    Update $p_\phi(\boldsymbol{x}|c=i)$ using Adam($\beta_1=0.5,\beta_2=0.9$) according to equation 4.6 for $i=1,\ldots,T$
    Update component weight according to equation 4.8 if $l>C_{start}$
  else
    Update both $p(\boldsymbol{x}|c=i)$ and $p(c=i)$ using Adam($\beta_1=0.5,\beta_2=0.9$) with **softmax** parametrization
  end if
end for

4.2.2 GAN+ML Training Procedure

We can also incorporate the GAN training with this model analogous to the **FlowGAN** (Grover et al., 2017) method. In the following, we assume the ML training is done using **softmax** parametrization and GAN objective is Wasserstein loss. Then the objective function is

$$\min_{g_i(\cdot),p(c)}\max_{f(\cdot)}\mathbb{E}_{\boldsymbol{x}\sim p_x}[f(\boldsymbol{x})]-\mathbb{E}_{\boldsymbol{z}\sim p_z}\mathbb{E}_c[f(g_c(\boldsymbol{z}))]-GP-\lambda_{ml}\mathbb{E}_{\boldsymbol{x}\sim p_x}[\log(\sum_{i=1}^{T}p(c=i)p(\boldsymbol{x}|c=i))] \tag{4.10}$$

$f(\cdot)$ and $g_i(\cdot)$ are the discriminator and i^{th} generator respectively, λ_{ml} is the weight for the ML term and GP is the gradient penalty term used in WGAN. The minimization of $g_i(\cdot)$ can be done using re-parametrization and back-propagation. The learning algorithm is presented below.

Algorithm 6 Mixflow GAN+ML Training

Input: Discriminator Training iteration n_c,learning rate for discriminator, generator and component c (l_{critic}, l_{gen}, l_c), epoch L, component number T, C_{start}, λ_{ml} and λ_{gp}

1: **for** epoch$= 1, \ldots, L$ **do**
2: Sample $\{\boldsymbol{x}\}_{n=1}^{N} \sim p_x$
3: Sample $\{\boldsymbol{z}\}_{n=1}^{N} \sim p_z$ and $\{c\}_{n=1}^{N} \sim p_c$
4: Calculate the gradient penalty term as Algorithm 3
5: Maximize $f(\cdot)$ with objective $\frac{1}{N}\sum_{n=1}^{N} f(\boldsymbol{x}_n) - f(g_c(\boldsymbol{z}_n)) - GP$ using Adam
6: Minimize $g_c(\cdot)$ with objective $-\frac{1}{N}\sum_{n=1}^{N} f(g_c(\boldsymbol{z}_n)) - \frac{\lambda_{ml}}{N}\sum_{n=1}^{N}\log(\sum_{i=1}^{T} p(c=i)p(\boldsymbol{x}_n|c=i))$ using Adam optimizer and algorithm 5 for all $c = 1, \ldots, T$
7: Update component prior $p(c=i)$ by maximizing the objective $\frac{1}{N}\sum_{n=1}^{N}\log(\sum_{i=1}^{T} p(c=i)p(\boldsymbol{x}_n|c=i))$
8: **end for**

4.2.3 Entropic Regularization

During the experiment with Mixflow GAN+ML training, we observe that the model will collapse to few dominant component even with large C_{start} and small λ_c. Thus, to encourage the model to use more component, we can manually add a entropy regularization term which will penalize the objective if component are away from uniform distribution. This idea is also used in MIX+GAN (Arora et al., 2017) to regularize the generator and discriminator. The regularization is

$$ER = -\frac{1}{T}\sum_{i=1}^{T}\log(p(c=i)) \tag{4.11}$$

Adding this term to the objective may damage the ML learning because the component weight is now learned to maximize ML term and regularization. However, adding this should improve the generation quality because the dominant component obtained through ML tends to generate blurred images even with the GAN training. Due to the time limits, we did not do enough experiments on the effect of entropic regularization but preliminary results are given.

4.3 Mixture of Discriminator

In the above section, we discuss the ML and GAN+ML training with single discriminator. However, it has to be powerful enough to capture the high dimensional data structure. This may take several training epochs. Thus, we will introduce the 'personalized' discriminators for each generator and give individualized guidance.

One simple approach is to pair each generator with its own discriminator. Thus, in the context of GAN training, the WGAN loss will be

$$\begin{aligned} &\max_{f_i} \mathbb{E}_{\boldsymbol{x}\sim p_x}[f_{\delta(x)}(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p_z}\mathbb{E}_c[f_c(g_c(\boldsymbol{z}))] - \lambda \mathbb{E}_{\hat{\boldsymbol{x}}\sim p_{\hat{x}}}[(||\nabla_{\hat{\boldsymbol{x}}} f_{\delta(\hat{\boldsymbol{x}})}(\hat{\boldsymbol{x}})||_2 - 1)^2] \\ &\delta(\boldsymbol{x}) = \text{argmax}_i\ p(\boldsymbol{x}|c=i) \end{aligned} \tag{4.12}$$

Hard component assignment will result into a discontinued discriminator value surface which should be avoided to maintain the smoothness and training stability.

Thus, instead of using deterministic assignment, we can use posterior probability to smooth the discriminator value surface. Thus, instead of using $f_{\delta(x)}(\cdot)$, we propose the following discriminator

$$f(\boldsymbol{x}) = \sum_{i=1}^{T} p(c=i|\boldsymbol{x}) f_i(\boldsymbol{x}) \tag{4.13}$$

Thus, in the limiting case, this will recover the hard component assignment. In the following, we assume each discriminator f_i is parametrized by ω and f_{mix} represents the posterior-smoothed mixture of discriminators.

We call this model **Personalized GAN (PGAN)**. The objective function for PGAN with GAN+ML training is analogous to Mixflow GAN (equation 4.10) with the only difference of replacing $f(\cdot)$ with $f_{mix}(\cdot)$.

Note that the definition of mixture of discriminators involves part of the generator through the posterior component, and the sole purpose of posterior probability is to assign the weights to each discriminator components. Thus this term should not be involved in updating the generator or discriminator. Therefore, we need to treat the posterior term as a constant during the update and it should not contribute to the gradients. The **training algorithm** is identical to Mixflow GAN+ML training (**algorithm** 6) with only difference of using f_{mix} instead.

4.3.1 Weight Sharing

If the single discriminator is replaced by mixture models, then we significantly increase the model complexity and number of training parameters. Even we reduce the topology of each discriminator, the training is still unstable due to the scarcity of the data especially high dimensional data like images. Therefore, to alleviate this problem, we can share parts of the model parameters across the components and only allow high level weights to be trained individually.

For example, if each component is a fully connected CNN, then we could share the CNN parameters across all components and only train the top-level fully connected MLP for each component. Therefore, this will significantly reduce the number of training parameters. In fact, this makes sense because CNN only extracts the features and these features can be used as the representations for later tasks. Thus, we do not need to train individual feature extractor.

Due to the time limits and hyper-parameter tuning, we **did not** do any experiments of PGAN on MNIST. However, the toy examples should illustrate some properties and advantages of PGAN.

4.4 Related Work

As mentioned in the motivation section, MIX+GAN and DeLiGAN adopt the idea of mixture model to improve the performance of GAN.

To be specific, the MIX+GAN uses the MLP/CNN as the discriminator component and generator component. Each of the component are weighted by a prior term which are similar to what we introduced in Mixflow model. However, the difference between our model and MIX+GAN is

- MIX+GAN still uses the neural network based generator which will not give a tractable density, in other words, the training is still entirely based on GAN training and may produce good quality image but poor likelihood. Our Mixflow model allow us to train the model with both GAN and ML training which tends to produce reasonable images with much better likelihood. In addition, the PGAN model allows for individual discriminator to guide specific generator whereas the MIX+GAN does not have this property. The sole purpose of discriminator mixture in MIX+GAN is to increase the modeling power.

- The prior weight in MIX+GAN are trained under the GAN loss function which is still lack of understanding how the mixture model behaves in such training procedure. On the other hand, the Mixflow model updates its prior weight purely on ML training which has been extensively researched.

The DeLiGAn model uses a mixture of latent space rather than the model structure. To be precise, they parametrize the latent space by a GMM model. They uses the re-parametrization trick (Kingma and Welling, 2013) to translate a stardard Gaussian noise to specific GMM

component with uniform weights. The DeLiGAN achieves better results in terms of diversity with limited amount of data.

Chapter 5

Experiments

In this section, we will conduct some experiments to demonstrate the properties of models and compare the training methods. The chapter are divided into two parts, the first part is focused on **toy examples** including **GMM samples** and **modified Swiss Roll data**. Notice for swiss roll dataset, we only have the generating process rather than the data density, thus, we cannot compare the learned likelihood with ground truth. For each data set, we will compare 6 different setups including

- ML Single Flow
- ML Mixflow
- GAN Single Flow
- GAN+ML Single Flow
- GAN+ML Mixflow
- GAN+ML PGAN (only toy examples)

First, we will compare the flow and Mixflow models using ML. Then, we will show the difference between GAN and GAN+ML to demonstrate some basic properties of these training methods. We will also show the results on Mixflow GAN+ML training and compare it with the PGAN model. In the second part, we will conduct experiments on MNIST data set. The evaluation is based on visual quality of generated images and learned negative log likelihood.

5.1 Toy Examples

5.1.1 GMM

We will conduct experiment on GMM toy data, this example intends to demonstrate the advantages of Mixflow model when modeling distributions with separated modes compared to single flow. We also show mode-dropping happens during GAN training.

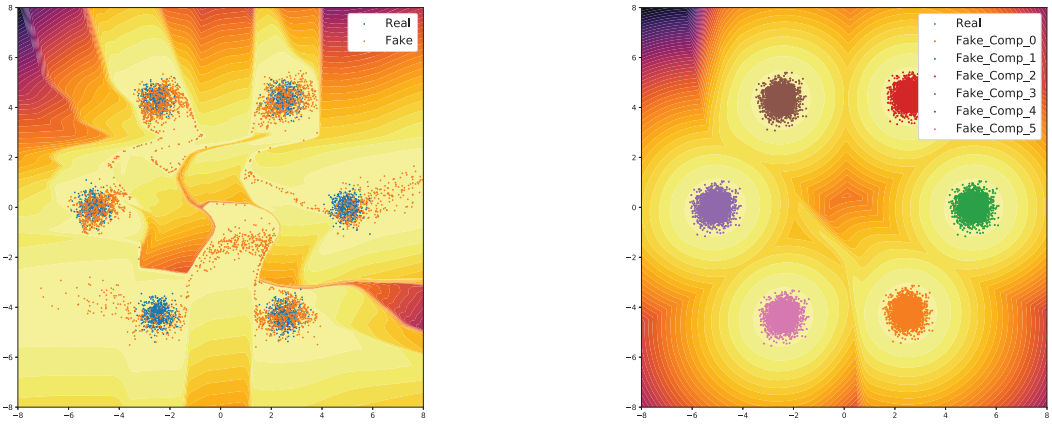
Data Set The training data are generated by 2D GMM with **6** Gaussian components and uniform weight. The mean of components are positioned at points which equally separate the circle with **radius 5**. Each Gaussian component has a **diagonal convariance matrix** with diagonal elements **0.1**. The batch size of training data is 300.

Model Topology The generator used is NICE and mixture of NICE. The coupling law of them is **additive coupling**. For single flow, it has 4 coupling layers and the function $m(\cdot)$ is a 5 layer-MLP with 100 node per layer. For Mixflow and PGAN, the default number of components is 6 and each flow has 2 coupling layers. Function $m(\cdot)$ is 3-layer MLP with 50 node per layer. The activation function for both single flow and Mixflow is ReLu. The discriminator are chosen to be 5-layer MLP with 100 node per layer and ReLu activation function. For PGAN, each discriminator component is a 3-layers MLP with 50 node per layer. The activation function is the same as single discriminator.

Hyper-parameters The training epoch is 5000. We set $\beta_1 = 0.5$ and $\beta_2 = 0.9$ for Adam optimizer. The learning rate for NICE is 0.005. For GAN and GAN+ML training, the gradient penalty weight $\lambda_{gp} = 0.1$ and ML weight $\lambda_{ml} = 1$. The learning rate for discriminator is 0.002 and for each epoch, we train discriminator 5 times before we update the generator. The learning rate for component weight is 0.005 and we use **softmax** parametrization for weight.

Figure 5.1 shows the generation quality and contour comparison between single flow and Mixflow with ML training.

We can clearly see even with deep coupling layers, the single flow model still contains the 'bridge' between different modes. The generation quality is not very good especially for components at 0^o and 225^o anti-clockwise. In addition, the contour learned by single flow contains sharp changes and it does not resemble the true GMM contour. On the other hand, the samples generated by Mixflow almost perfectly cover the true GMM samples and each



(a) Single Flow ML training

(b) Mixflow ML training

Fig. 5.1 The contour and generated samples for single flow and Mixflow with ML training. For Mixflow model, we plot generated samples for each components. The contour is the log likelihood value.

components capture one mode of the GMM distribution. As for the contour, the Mixflow model looks much more like the true GMM contour. This can also be verified by checking the training curve (**Figure** 5.2).

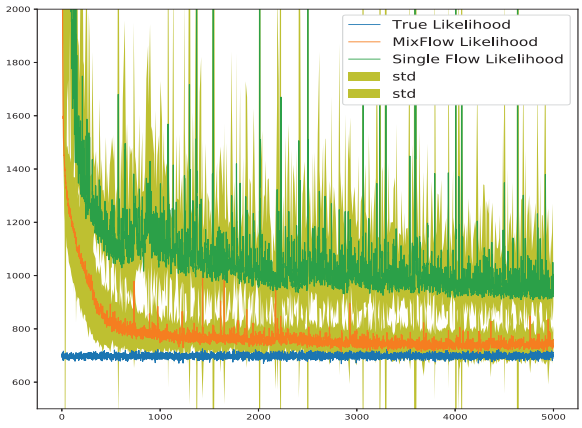
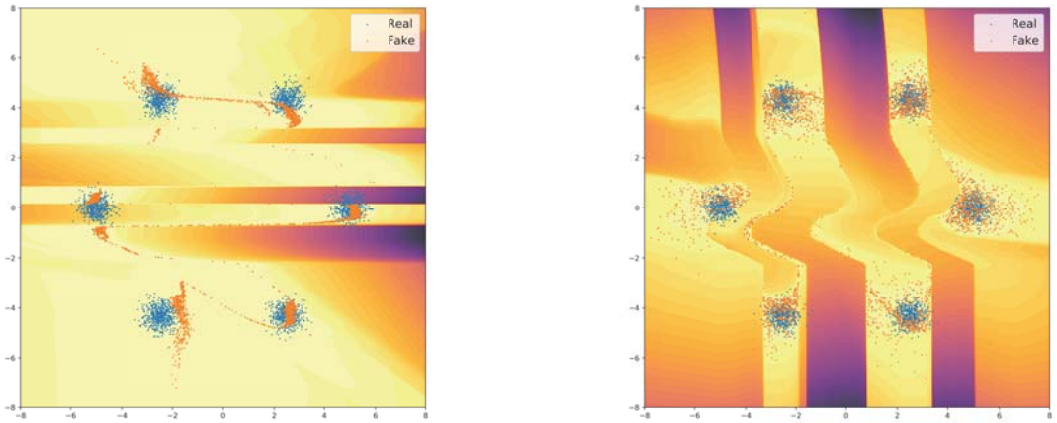


Fig. 5.2 The negative log likelihood plot for single flow and Mixflow with ML training. The curve are obtained by averaging 5 runs and ± 1 std.

We can see from the training curve, the Mixflow obtains lower NLL much faster than single flow and the value is very close to the ground truth with smaller variance. These two plots confirms that the Mixflow model indeed performs better when distributions have separated modes.

Now we compare the difference between GAN training and GAN+ML training. For single flow, **Figure** 5.3 shows the generated samples and contour plots for both training methods.



(a) Single Flow GAN training

(b) Single Flow GAN+ML training

Fig. 5.3 The contour and sample generation for GAN and GAN+ML training.

We can observe that pure GAN training will highly push the generated sample towards the high density region. This may produce the good quality samples but the diversity is not encouraged. This explains why model-dropping often happens during GAN training and why it tends to produce poor likelihood. In addition, the contour learned by GAN also contains sharp changes and does not look like GMM contour. With the extra ML term included, the generation diversity is much better than pure GAN training. The contour learned by GAN+ML still contain large changes. **Figure** 5.4 shows the training curve of GAN+ML with single flow model.

From the training curve, we can see by adding extra ML term, the training curve of GAN+ML is very similar to pure ML training. In fact, from results in **Table** 5.1, the log likelihood is much better for GAN+ML training.

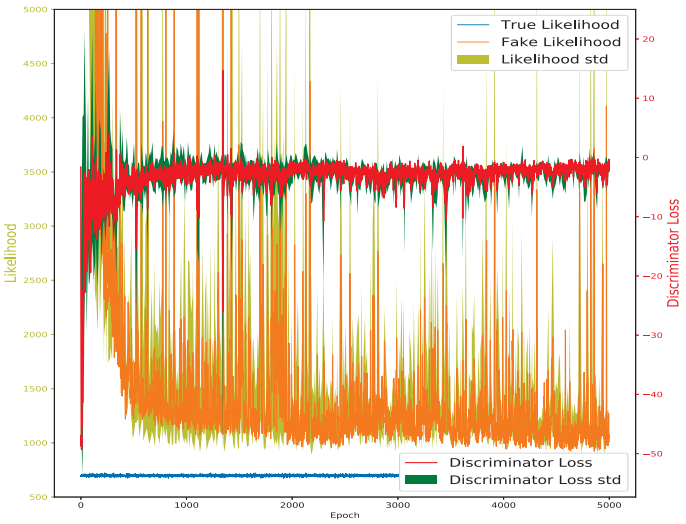
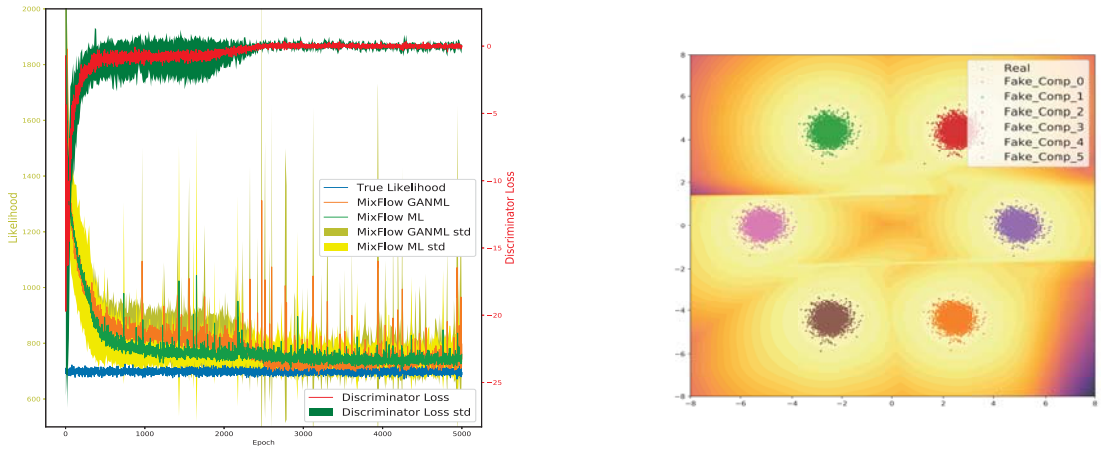


Fig. 5.4 The training curve of GAN+ML with single flow. The red line is the discriminator loss.

Now we show the sample generation for Mixflow model with GAN+ML training and training curve comparison with Mixflow ML training (**figure** 5.5).



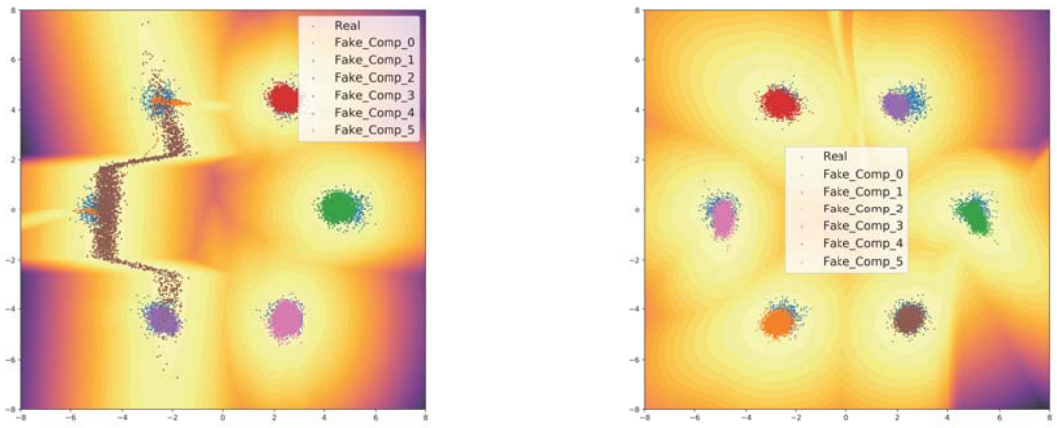
(a) (b)

Fig. 5.5 (a)The training curve comparison between GAN+ML and ML training with Mixflow model. The curve is obtained by averaging 5 runs. (b)Contour and sample generation plots for Mixflow model with GAN+ML learning

We can see the GAN+ML training obtains similar training curve as pure ML training and this is much better than both single flow ML or GAN+ML training. As for the generation quality, we can see it achieves similar quality as Mixflow ML training but the contour surface has

sharp changes due to the GAN training. The PGAN model produces similar results as the Mixflow GAN+ML training, thus, we won't put the plot for PGAN (refer to Appendix B).

In order to compare the performance between PGAN and Mixflow GAN+ML, we will slightly change the hyper-parameters to increase the difficulty. Change $\lambda_{ml} = 0.01$ to reduce the importance of ML learning and batch size is changed to 100. **Figure** 5.6 shows the generation quality comparison.



(a) (b)

Fig. 5.6 (a) The sample generation of Mixflow with GAN+ML training. (b) Same plot with PGAN model.

We can see that the generation quality of PGAN is better than Mixflow GAN+ML. The possible reason is that the personalized discriminator can provide individual guidance even with scarce training data and little help from ML learning. On the other hand, single discriminator need more training data and helps from ML learning to successfully capture the entire data structure, thus, without these, the discriminator tends to produce inaccurate guidance to the generator component and results into generation artifacts.

Table 5.1 shows the results obtained on GMM toy examples.

Therefore, this tables shows that the Mixflow and PGAN model achieves the lowest NLL compared to others. In addition, it shows that the pure GAN training will generate much worse likelihood compared to GAN+ML training.

Training Method	NLL	Discriminator Loss	Dominant component
MLE Single Flow	968.70	-	1
GAN Single Flow	1.25e7	-2.43	1
GAN+ML Single Flow	1127.5	-1.99	1
MLE MixFlow	**742.85**	-	6
GAN+ML MixFlow	**742.45**	**0.0044**	6
PGAN	**743.61**	0.0132	6

Table 5.1 The results for GMM toy examples. The NLL and Discriminator loss is obtained by averaging the last 500 epochs. The ground true NLL is **723.35**. Note: due to the difference in discriminator topology between PGAN and other models, the PGAN discriminator loss cannot be directly compared

Note: Appendix B contains more completed plots for GMM toy examples.

5.1.2 Swiss Roll

For this toy examples, we uses modified swiss roll as the training data. We choose this because spirals is difficult for simple normalizing flow to model and it does not contain explicitly separated modes like GMM. Thus, this gives us the opportunity to test other advantages of Mixflow model.

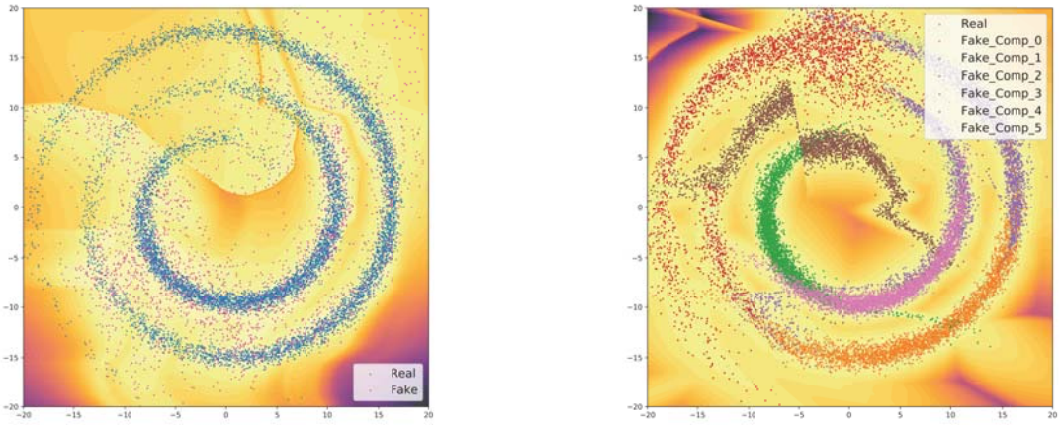
Data Set The training data is the swiss roll data with more spirals to increase the difficulty. The batch size is 300 by default.

Model Topology The model topology is **identical** to the one used for GMM examples.

Hyper-parameters Identical to GMM examples.

Figure 5.7 shows the ML training comparisons on generated samples.

We can clearly observe that the ML training of singe flow fails to capture the structure of swiss roll and instead it tries to cover the whole area. On the other hand, Mixflow ML learning can successfully capture most of the data structure except for some area with sparse data. One interesting observation is that for most of the component, they can successfully find their own 'modes' even though the data does not have explicit modes. In the swiss roll data, each component will try to model part of the spiral. However, there is still some



(a) (b)

Fig. 5.7 (a)The generated samples and contour plots for single flow with ML learning. (b)The same plot for Mixflow with ML learning

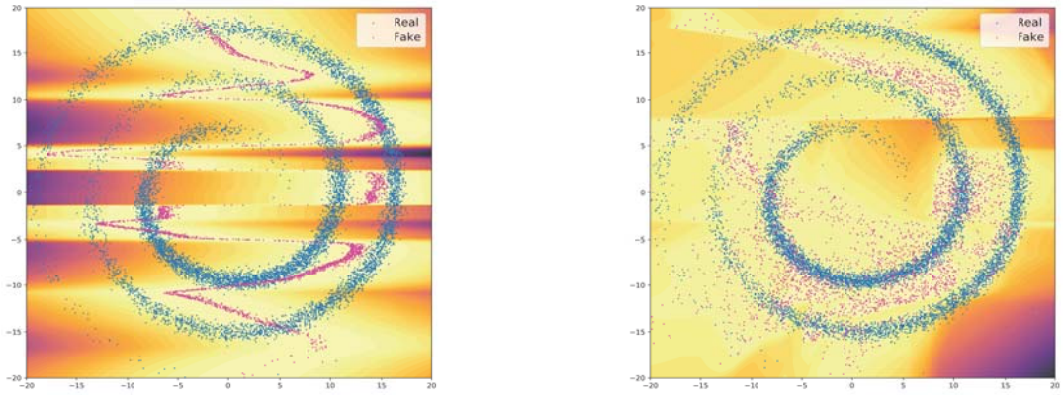
artifacts such as 'gray' component, it will try to connect between arcs, this is probably due to the initialization, for example, if the this component is initialized at position between the arcs, then it will try to model both parts simultaneously.

Figure 5.8 shows the comparison between GAN training and GAN+ML training for single flow.

Similarly, for GAN training, it also fails to capture the data structures and produces sharp changes in contour plots. We can guess the GAN training wants to 'stretch' the flow to specific part of the spirals, thus, it can generates realistic samples but severe mode dropping will happen. This observation is consistent with the GMM GAN training. GAN+ML training generates samples resembling the ML training samples. However, for the contour of GAN+ML, we can still observe the 'stretch' induced by GAN training.

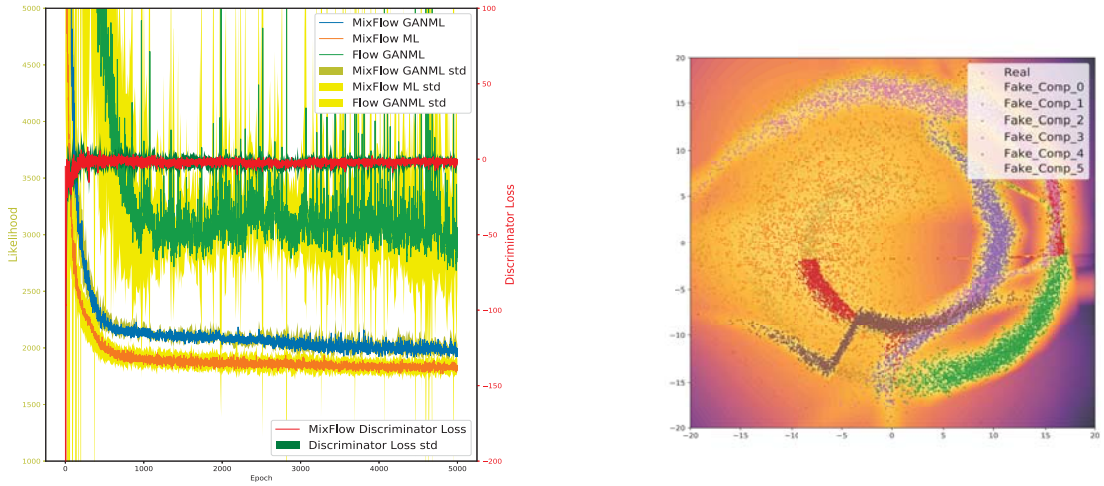
Now we will show the Mixflow GAN+ML training results in **Figure** 5.9.

The GAN+ML training achieves similar generation quality as ML training but achieves slightly worse log likelihood. One interesting observation is that the contour of GAN+ML is more contractive than pure ML learning (figure 5.7), this contractive behavior is because the GAN training will focus on the high density regions, thus it will try to contract the generator



(a) (b)

Fig. 5.8 (a)The generated samples and contour plots for single flow with GAN training. (b)The same plot for single flow with GAN+ML learning



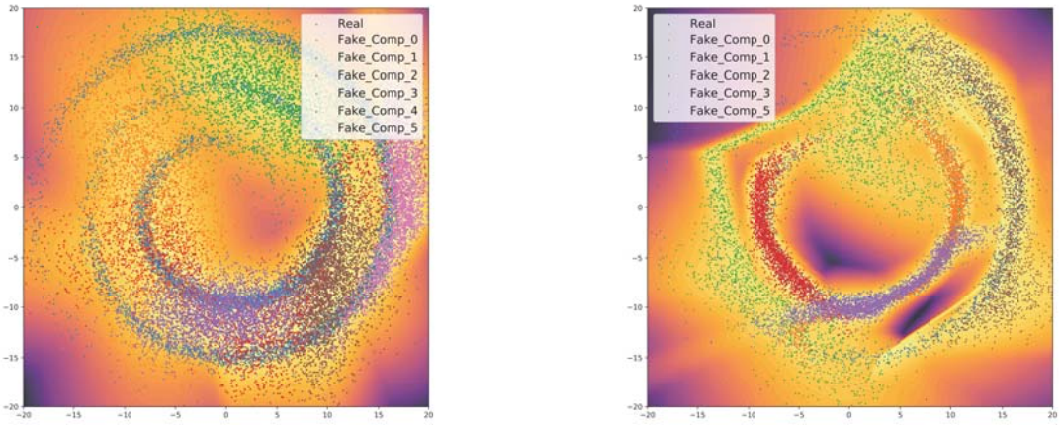
(a) (b)

Fig. 5.9 (a)The training curve comparison between Mixflow ML and Mixflow GAN+ML training. (b)The sample generation and contour plot for Mixflow GAN+ML

density and discourage the coverage. Thus the ML learning and GAN learning can establish a middle point between coverage and contraction in order to produce good images with high likelihood.

Similarly, we compare the performances between Mixflow GAN+ML and PGAN. Analogous to GMM toy examples, we will change the hyper-parameter settings, the λ_{ml} is changed to

0.1 and training epoch is 3000. Both models obtain the similar log likelihood with similar convergence speed (see Appendix B). However, **Figure** 5.10 shows the generated samples for both models.



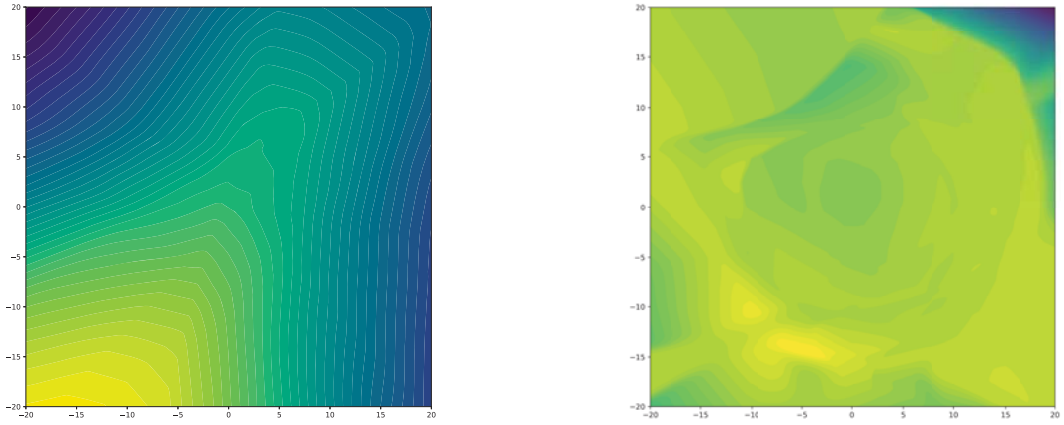
(a) (b)

Fig. 5.10 (a) Sample Generation of Mixflow GAN+ML training (b) PGAN sample generation. Note that for PGAN, component 4 has very small weight after training, thus, we did not plot the samples generated by that component.

Even with similar likelihood, the sample generation quality of PGAN is much better than Mixflow. This can be explained by the discriminator contour surface shown in **Figure** 5.11.

We can observe that the single discriminator fails to capture the swiss roll data structure. Thus, it will not provides very useful guidance for generator. On the other hand, PGAN captures more features than single discriminator. For example, it puts the high values on lower left corner where the generation quality is not very good.

Table 5.2 shows the NLL and discriminator loss for the above models.



(a) (b)

Fig. 5.11 (a) The discriminator value surface learned by single discriminator (b) Contour learned by posterior smoothed mixture of discriminators

Training Method	NLL	Discriminator Loss	Dominant component
MLE Single Flow	2039.68	-	1
GAN Single Flow	2.74e6	-4.18	1
GAN+ML Single Flow	3075.47	-9.65	1
MLE MixFlow	**1828.72**	-	6
GAN+ML MixFlow	**1992.65**	**-1.47**	5
PGAN	**1989.34**	-0.95	6

Table 5.2 The results for Swiss roll toy examples. The NLL and Discriminator loss is obtained by averaging the last 500 epochs. There is no ground truth for NLL. The PGAN discriminator loss cannot be directly compared to the rest of the models.

5.1.3 Conclusion

In above sections, we conduct some experiments and comparisons on toy examples. Based on the results, we can safely make the following points for low dimensional data.

- Single Flow is difficult to model distributions with separated modes or complicated distributions no matter ML, GAN or GAN+ML training are used.
- Mixflow models achieves much better log likelihood, generation quality and convergence speed compared to single flow for both ML or GAN+ML training.
- Pure GAN training tends to concentrate the generator distribution to the high density regions, thus, it will produce high quality samples but poor log likelihood. This is also called the mode-dropping. With the extra ML term, the generator will tends to find a middle ground between the high generation quality and distribution coverage.
- PGAN model can generated higher quality samples compared to Mixflow GAN+ML training with fewer training epochs. This advantage comes from the individual guidance of mixture of discriminators. From the contour plots of discriminator value surface, we can observe the PGAN captures more complicated data structures than single discriminator.

5.2 MNIST

In this section, we will focus on the model performance on MNIST data set. Due to the time limits, we did not do experiments on the PGAN model. The comparison are based on visual quality of sample generation and learned NLL.

Data Set The MNIST data set contains 55000 training images, 10000 test images and 5000 validation points. Each images is $28 \times 28 = 784$ dimensional vector with the value between 0 and 1. The batch size for training these models are 200.

Model Topology All the flow models uses **logistic distribution** as the base distribution. Single flow uses 4 coupling layers and additive coupling law. For Mixflow model, the default number of component is 10 and each component has 3 coupling layers and additive couping law. The $m(\cdot)$ function is chosen to be ReLu MLP or CNN. For MLP, single flow has 5 layer and 1000 node per layer which is identical to the settings in (Dinh et al., 2014). For Mixflow, the MLP has 3 layers and 200 node per layer. For CNN, the single flow has 2-layer

fully connected network with 1000 and $7 \times 7 \times 64 = 3136$ node, followed by a convolutional layer with 4×4 filter size and 64 feature maps, then the output layer is another convolutional layer with 1 feature map and same filter size. The intermediate convolutional layer has batch normalization and ReLu activation function. For Mixflow, the CNN has same structure as single flow but with 300 and $7 \times 7 = 784$ nodes for fully connected network and 16 feature maps for intermediate CNN layer.

Hyper-parameter The training epoch is 100000. The learning rate of generator is 0.0005 and 0.0001 for discriminator. For Mixflow, the learning rate of component weight is 0.01. The Adam optimizer has the same settings as the toy example. The gradient penalty weight is set to 10 and ML weight is 0.001 to roughly balance the ML learning and GAN learning. The entropic regularization weight is set to 1 and only applied to Mixflow GAN+ML learning. The rest of the settings is the same as the toy examples.

Pre-Training Pre-training is used for all GAN or GAN+ML method. For single flow, it will be pre-trained by all MNIST data for 1000 epochs using ML learning. For Mixflow, each component will be trained using specific digit data for 1000 epochs.

Table 5.3 shows the learned NLL of these models using MLP as coupling function $m(\cdot)$.

Model	Training NLL	Test NLL	Discriminator Loss	Major Component
Single Flow ML	-2034.46	-1910.31	-	1
Single Flow GAN	3144.79	3431.66	2.21	1
Single Flow GAN+ML	-1741.21	-1650.53	1.91	1
Mixflow ML	**-3021.35**	**-2984.03**	-	4
Mixflow GAN+ML	**-2812.34**	**-2728.74**	-2.15	7

Table 5.3 The results are obtained using the average of last 5000 training epochs. The results obtained in my experiment for single flow is different from Grover et al. (2017). But the ML results are similar to original NICE paper(Dinh et al., 2014). The reason still need to be investigated. The Major components are the number of component with weight larger than 0.9

Figure 5.12 shows the MNIST generation for the above models. Note that this generation does not conditioned on any label information, thus, we have no control on the digits it generates, thus we can only compare the overall generation quality. More generated MNIST digits can be found in Appendix B.

We can observe that the image from ML is blurred and difficult to distinguish. This is

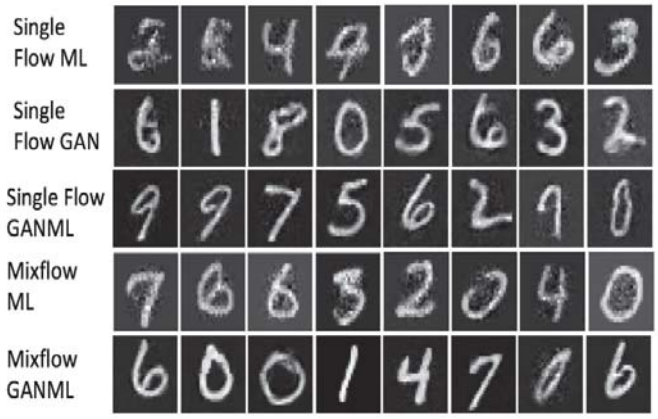


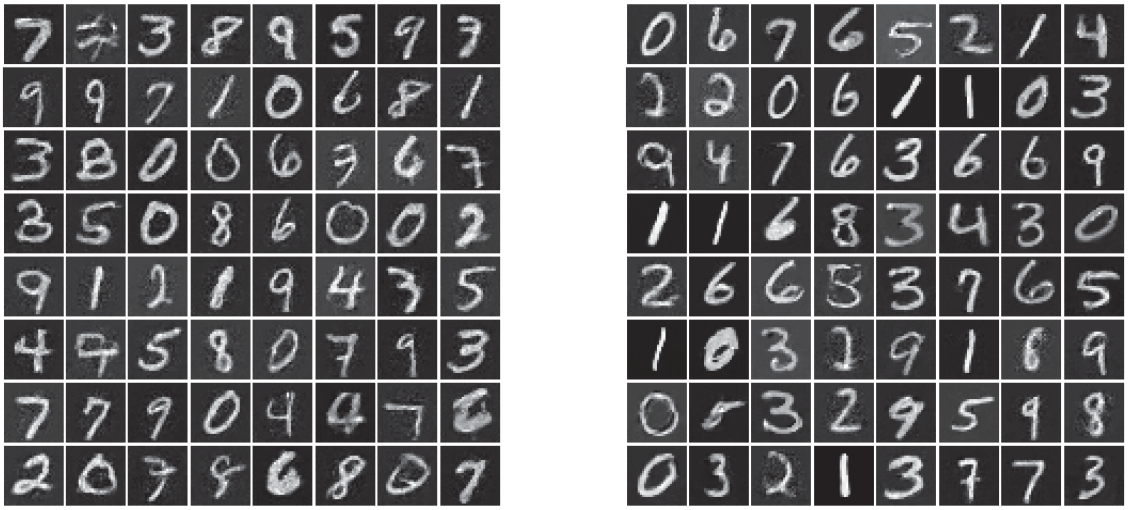
Fig. 5.12 The images are generated using the above models. The digits are randomly selected from the batch. The coupling function is MLP.

because the ML learning tends to cover the whole distributions. Thus, the image will be noisy and blurry. If GAN training is involved, the images is much sharper due to the 'contraction' in the distribution. We can observe this through the better image quality. However, if only GAN training is used, the poor log likelihood is obtained due to the property of GAN discussed in toy examples. If extra ML term is included, the image quality does not suffer, instead, it may achieve better generation quality (third row and last row compared to second row. For clear comparison, refer to Appendix B for more generated images.) and reasonable log likelihood.

If we compare the single flow and Mixflow model, we can see the Mixflow ML learning indeed achieves lower likelihood than single flow as expected. But the generation quality is not significant better than single flow. This may due to the property of ML learning as discussed above. If we include the GAN training, we can see Mixflow GAN+ML model achieves slightly better quality than single flow GAN+ML and GAN especially for some digits like 1,7 (**Figure** 5.13).

Another interesting observation is if we inspect the prior weight of each components, we will notice that the 8^{th} component is the most dominant one. This is expected, because the digit 8 has the most complicated structures and can be easily transformed to other digits like 0, 6, 5,etc. In fact, in figure 5.12, the last row are generated by component [6,0,0,1,4,7,**8**,6], this mean at least, 8^{th} component are also responsible for generating digit 0.

The generated image quality can be significantly improved if we change the coupling function to CNN (in original NICE paper (Dinh et al., 2014) and FlowGAN paper (Grover et al., 2017),



(a) (b)

Fig. 5.13 (a)Single Flow GAN+ML training (b) Mixflow GAN+ML training.

they only mentioned the MLP not CNN). Due to the time limits, we only test the single flow GAN+ML and Mixflow GAN+ML. We also compare their generation quality with state-of-art DCGAN generation. **Table** 5.4 shows the results obtained using CNN coupling function. **Figure** 5.14 shows the generation comparison between True data, DCGAN, Single

Model	Training NLL	Test NLL	Discriminator Loss	Major Component
Single Flow GAN+ML	-2628.08	-1756.98	0.92	1
Mixflow GAN+ML	**-3245.61**	**-2695.74**	**0.67**	7

Table 5.4 The results are obtained using the average of last 5000 training epochs. The Major components are number of component with component weight larger than 0.9

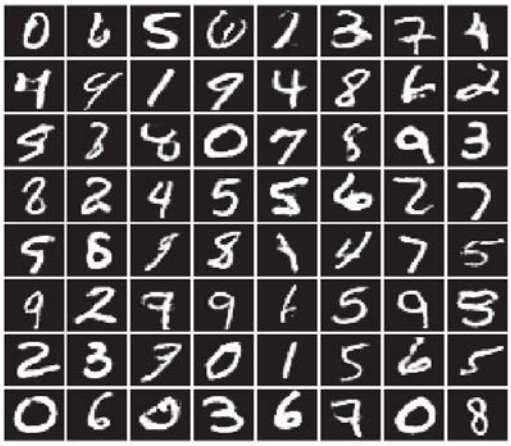
Flow GAN+ML and Mixflow GAN+ML.

We can see from the generated images, the Mixflow GAN+ML and single Flow GAN+ML achieves reasonable image quality compared to real MNIST or DCGAN. In addition, these model can give explicit log likelihood evaluation which can be used for many other tasks like image inpainting.

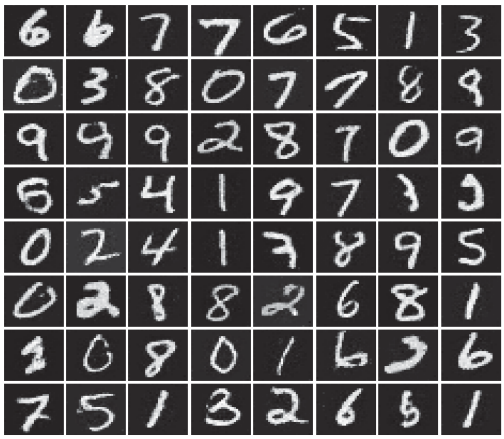
5.2.1 Conclusion

In this section, we conduct the experiments on MNIST data set. It shows that using Mixflow can achieve better likelihood than single flow which is consistent with the observation in

(a) Real MNIST data



(b) DCGAN MNIST



(c) Single Flow GAN+ML

(d) Mixflow GAN+ML

Fig. 5.14 MNIST generation using CNN based network.

toy examples. As for the generation quality, the Mixflow does not significantly outperform the single flow model, however, the Mixflow with pre-training allow each component to learn specific digits. For some digits, it can generate clearly better images. In fact, based on the dominant component number of Mixflow ML, we guess although MNIST has 10 digits, the actual separated mode is much smaller than 10. The manifold of different digits may be connected with reasonable probability in the high dimensional space. Thus, even with pre-training, some of the generator components are merged together like 8 and 0.

The generation quality can be significantly improved if CNN coupling function is used. By comparison to DCGAN and real images, the model proposed in this thesis can produce reasonable images with high likelihood. This GAN+ML training method establish a middle

ground between high image quality and good likelihood, which should be further investigated to see its performance in other tasks.

Chapter 6

Future Work and Conclusion

6.1 Future Work

In the experiment chapter, we have obtained some preliminary results on toy examples and MNIST data set. However, there are still many things left to be investigated.

6.1.1 PGAN

The initial attempt of PGAN on MNIST dataset is not very successful. The possible reason is because the scarcity of training data for each discriminator component. Thus, the unstable training of discriminator results in the inaccurate gradients for generator. One way to alleviate this issue is to share the CNN structure and parameters across all discriminator components. The only trainable weights are the fully connected network. Thus, we can obtain a more stable training.

In addition, as discussed in Chapter 4, the reason we use posterior smoothing is to avoid the sharp changes in discriminator value surface. However, in high dimensional spaces, this may still happen even with posterior smoothing. Thus, we can use 'Tempering' to further smooth the boundary. In other words, we can divided the posterior logits by a 'temperature' and put them through the sofrmax function. This may further increase the training stability.

6.1.2 Entropic Regularization

We introduce the idea of entropic regularization in Chapter 4. This extra term is to avoid the collapsing of the component weights. In the experiment section ,we only implement this

idea on Mixflow GAN+ML training. Thus the actual influence of this term still lacks of understanding. Therefore, further experiment should be conducted to investigate its influence towards log likelihood, GAN loss and image generation quality.

6.1.3 Flow with Convolution Structure

In this thesis, we use NICE throughout the whole experiments. However, this flow does not have a convolutional structure, therefore, it won't perform well on more complicated dataset like CIFAR-10, LSUN, etc. Another normalizing flow called Real-NVP (Dinh et al., 2016) mimics the convolutional structure and it is more suitable for this kind of task.

Similarly, we can use CNN as the coupling function instead of ReLu MLP. Thus, we can test our model on more data set to obtain convincing evidence that the new model performs better than previous methods.

6.1.4 Quantitative Metric

In the experiment section, the assessment of generation quality is through human. This does not give us a quantitative measure on generation quality and is strongly biased if the number of examiners is not enough. Thus, we prefer to use the quantitative measure about image quality. One suggestion is MODE score (Che et al., 2016) for MNIST data set. This can be calculated as

$$\exp(\mathbb{E}_{\boldsymbol{x}\sim p_g}[KL(p(y|\boldsymbol{x})||p^*(y)))] - KL(p^*(y)||p(y))) \tag{6.1}$$

where $\boldsymbol{x}$ is the generated image and $p(y|\boldsymbol{x})$ is the pre-trained classifier, $p^*(y)$ is the distribution of the true label and $p(y)$ is the label distribution of generated data.

For CIFAR-10, LSUN dataset, we can use the inception score calculated as

$$\exp(\mathbb{E}_{\boldsymbol{x}\sim p_g}[KL(p(y|\boldsymbol{x})||p(y))]) \tag{6.2}$$

Thus, we can compare the generation quality in a more consistent way.

6.1.5 Component Number

In the above experiment, we know the roughly correct number of components. However when the component number is much smaller/larger than this, the behavior of the Mixflow model is not fully investigated in this thesis. Thus, more experiment should be conducted to

test its performance.

In addition, we could also investigate the possibility of using the ideas from **infinite mixture model and Dirichlet Process**, thus, we may be able to determine how many component we should use based on the data. In fact, this idea has been implemented for VAE (Abbasnejad et al., 2016) to perform semi-supervised learning.

6.1.6 Primal form WGAN

We use dual form Wasserstein distance for Mixflow GAN+ML and PGAN training. In fact, our Mixflow model can be easily adapted to primal form WGAN. We can test if the distance measure influences our model performance. Unfortunately, the PGAN method are based on the idea of personalized discriminator, thus, it cannot be used for primal form WGAN.

6.2 Conclusion

In this thesis, we give a comprehensive review of Generative adversarial network and its related variants. The focus is on different types of Wasserstein GAN which is more stable to train compared to original GAN. Particularly, we introduce the dual and primal form WGAN. In the end of the review, we give brief introduction to the recently published GAN variants and their relationship with the Wasserstein metric.

Then we introduce a different type of generator called normalizing flow which has been used in FlowGAN model. Based on the review of normalizing flow and simple toy examples, we argue that the normalizing flow, particularly NICE, is difficult to model distributions with separated modes. Thus, we propose a novel generator structures using the idea from mixture models, called Mixflow, which has better performance than single flow according to toy examples. Furthermore, for GAN training, we also propose a personalized discriminator to guide each generator component. The toy examples shows the PGAN method obtains better generation quality compared to Mixflow GAN+ML when model is crippled.

The final MNIST dataset shows that not only our proposed model achieved the best log likelihood but it also produces more reasonable generation quality compared to single flow training. This opens the door for tasks when both generation quality and good log likelihood are required.

References

Ehsan Abbasnejad, Anthony Dick, and Anton van den Hengel. Infinite variational autoencoder for semi-supervised learning. *arXiv preprint arXiv:1611.07800*, 2016.

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (gans). *arXiv preprint arXiv:1703.00573*, 2017.

Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The cramer distance as a solution to biased wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017.

Olivier Bousquet, Sylvain Gelly, Ilya Tolstikhin, Carl-Johann Simon-Gabriel, and Bernhard Schoelkopf. From optimal transport to generative modeling: the vegan cookbook. *arXiv preprint arXiv:1705.07642*, 2017.

Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*, 2016.

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

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013.

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

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

Aude Genevay, Gabriel Peyré, and Marco Cuturi. Sinkhorn-autodiff: Tractable wasserstein learning of generative models. *arXiv preprint arXiv:1706.00292*, 2017.

Irving J Good et al. Maximum entropy for hypothesis formulation, especially for multidimensional contingency tables. *The Annals of Mathematical Statistics*, 34(3):911–934, 1963.

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

Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Bridging implicit and prescribed learning in generative models. *arXiv preprint arXiv:1705.08868*, 2017.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

Swaminathan Gurumurthy, Ravi Kiran Sarvadevabhatla, and Venkatesh Babu Radhakrishnan. Deligan: Generative adversarial networks for diverse and limited data. *arXiv preprint arXiv:1706.02071*, 2017.

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

Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. *arXiv preprint arXiv:1705.08584*, 2017.

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

Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *arXiv preprint arXiv:1701.04722*, 2017.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Youssef Mroueh, Tom Sercu, and Vaibhava Goel. Mcgan: Mean and covariance feature matching gan. *arXiv preprint arXiv:1702.08398*, 2017.

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

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

A. Spurr, E. Aksan, and O. Hilliges. Guiding InfoGAN with Semi-Supervision. *ArXiv e-prints*, July 2017.

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

Esteban G Tabak, Eric Vanden-Eijnden, et al. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

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

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273*, 2016.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.

Appendix A

Theoretical Proof

Original GAN

Proposition 2. *For fixed generator $G(\cdot)$, the optimal discriminator $D^*(\boldsymbol{x})$ achieved using objective function 2.1 is $\frac{p_d(\boldsymbol{x})}{p_d(\boldsymbol{x})+p_g(\boldsymbol{x})}$.*

Proof. The training objective for discriminator $D(\cdot)$ is to maximize

$$\begin{aligned} V(G,D) &= \int_{\boldsymbol{X}} p_d(\boldsymbol{x}) \log(D(\boldsymbol{x}))d\boldsymbol{x} + \int_{\boldsymbol{Z}} p_z(\boldsymbol{z}) \log(1-D(G(\boldsymbol{z})))d\boldsymbol{z} \\ &= \int_{\boldsymbol{X}} p_d(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1-D(\boldsymbol{x}))d\boldsymbol{x} \end{aligned} \tag{A.1}$$

Thus, based on the Euler–Lagrange equation and calculus of variations, for any $(p_d, p_g) \in (0,1]^2$, the maximum in the range of $[0,1]$ is achieved by

$$D^*(\boldsymbol{x}) = \frac{p_d(\boldsymbol{x})}{p_d(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

□

Theorem 4. *The global minimum achieved by the training objective equation 2.1 is achieved if and only if $p_g = p_d$ and the minimum value is $-\log 4$ and it is equivalent to minimizing the Jensen-Shannon divergence between p_g and p_d.*

Proof. Assume the discriminator is trained to be at its optimum $D^* = \frac{p_d}{p_d+p_g}$. Then, the training objective becomes

$$\begin{aligned}
&\min_{p_g} E_{\boldsymbol{x}\sim p_d}[\log(D^*(\boldsymbol{x}))] + E_{\boldsymbol{x}\sim p_g}[\log(1-D^*(\boldsymbol{x}))] \\
=&\min_{p_g} E_{\boldsymbol{x}\sim p_d}[\log(\frac{p_d(\boldsymbol{x})}{p_d(\boldsymbol{x})+p_g(\boldsymbol{x})})] + E_{\boldsymbol{x}\sim p_g}[\log(\frac{p_g(\boldsymbol{x})}{p_d(\boldsymbol{x})+p_g(\boldsymbol{x})})] \\
=&\min_{p_g} E_{\boldsymbol{x}\sim p_d}[\log(\frac{p_d(\boldsymbol{x})}{\frac{p_d(\boldsymbol{x})+p_g(\boldsymbol{x})}{2}})] + E_{\boldsymbol{x}\sim p_g}[\log(\frac{p_g(\boldsymbol{x})}{\frac{p_d(\boldsymbol{x})+p_g(\boldsymbol{x})}{2}})] - \log 4 \\
=&\min_{p_g} KL(p_d||\frac{p_d+p_g}{2}) + KL(p_g||\frac{p_d+p_g}{2}) - \log 4 \\
=&-\log 4 + 2 * JSD(p_d||p_g)
\end{aligned} \tag{A.2}$$

Thus, the minimizing training objective for generator with optimal discriminator is equivalent to minimize the Jensen-Shannon distance. The minimum is achieved if and only if $p_d = p_g$ and the minimum is 0. Thus, the overall minimum for the training objective is $-\log 4$ □

Wasserstein GAN

Proof. Proof of Theorem 1: Assume we the primal optimal solution is obtained by $\boldsymbol{c}^T\boldsymbol{x}^* = z^*$ such that $\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{b}$. Then we construct the following matrix:

$$\hat{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A} \\ -\boldsymbol{c}^T \end{bmatrix}, \quad \hat{\boldsymbol{b_\varepsilon}} = \begin{bmatrix} \boldsymbol{b} \\ -z^* + \varepsilon \end{bmatrix}, \quad \hat{\boldsymbol{y}} = \begin{bmatrix} \boldsymbol{y} \\ \alpha \end{bmatrix} \tag{A.3}$$

If $\varepsilon = 0$, then

$$\hat{\boldsymbol{A}}\boldsymbol{x}^* = \hat{\boldsymbol{b}_0} \tag{A.4}$$

This indicates matrix $\hat{\boldsymbol{A}}$ span the vector $\hat{\boldsymbol{b}_0}$, or in other words, the $\hat{\boldsymbol{b}_0}$ are inside the convex cone made by column vector of $\hat{\boldsymbol{A}}$. If $\varepsilon > 0$, then there is no such $\boldsymbol{x}$ that

$$\hat{\boldsymbol{A}}\boldsymbol{x} = \hat{\boldsymbol{b}_\varepsilon} \tag{A.5}$$

because $\max\ -\boldsymbol{c}^T\boldsymbol{x} = -z^* < -z^* + \varepsilon$. This represents $\hat{\boldsymbol{A}}$ cannot span the vector $\hat{\boldsymbol{b}_\varepsilon}$. Therefore, it is not inside the convex cone defined by $\hat{\boldsymbol{A}}$. Thus, by Farkas' lemma, there exists a $\boldsymbol{y}$ and α such that

$$\hat{\boldsymbol{A}}^T\hat{\boldsymbol{y}} \leq 0, \quad \hat{\boldsymbol{b_\varepsilon}}^T\hat{\boldsymbol{y}} > 0 \tag{A.6}$$

in other words,

$$\boldsymbol{A}^T\boldsymbol{y} \leq \alpha\boldsymbol{c}, \quad \boldsymbol{b}^T\boldsymbol{y} > z^* - \varepsilon \tag{A.7}$$

without loss of generality, we can set $\alpha = 1$. We can also show that

$$\boldsymbol{b}^T\boldsymbol{y} = \boldsymbol{x}^{*T}\boldsymbol{A}^T\boldsymbol{y} \leq \boldsymbol{x}^{*T}\boldsymbol{c} = \boldsymbol{z}^* \tag{A.8}$$

Thus, we establish the bound of $\boldsymbol{b}^T\boldsymbol{y}$:

$$z^* - \varepsilon < \boldsymbol{b}^T\boldsymbol{y} \leq z^* \tag{A.9}$$

Therefore, if we maximize the objective function $\boldsymbol{b}^T\boldsymbol{y}$ with constraints $\boldsymbol{A}^T\boldsymbol{y} < \boldsymbol{c}$, we can get arbitrarily close to z^*, especially with $\varepsilon = 0$, the $\hat{\boldsymbol{b}}_0$ lies on the boundary of convex cone defined by $\hat{\boldsymbol{A}}$ and the solution of $\boldsymbol{y} = \boldsymbol{x}^*$ □

Proof. Theorem 3: By the theory of Lagrange Multiplier, the original optimization with equality constraints can be reformulated to unconstrained problem with Lagrange Multiplier $\lambda \in [0,\infty]$ such that the optimum for the original problem will corresponds to the stationary points in the new problem. In addition, due to the convexity of the constraints, the problem is convex optimization defined on the convex set $U_\alpha(\hat{p}_d, \hat{p}_g)$, thus there is only one unique optimum P^λ and it is the stationary point.

We define

$$d_M^\lambda(\hat{p}_d, \hat{p}_g) = < P^\lambda, M >, \text{ where } P^\lambda = \text{argmin}_{P \in U(\hat{p}_d, \hat{p}_g)} < P, M > -\frac{1}{\lambda}h(P) \tag{A.10}$$

By duality theory, for a given α, there exists a λ that $d_{M,\alpha}(\hat{p}_d, \hat{p}_g) = d_M^\lambda(\hat{p}_d, \hat{p}_g)$. The objective function with Lagrange Multiplier λ, γ, β is

$$\mathscr{L}(P, \gamma, \beta, \lambda) = \sum_{ij} \frac{1}{\lambda} p_{ij} \log(p_{ij}) + p_{ij} m_{ij} + \gamma^T (P\mathbf{1}_d - r) + \beta^T (P^T \mathbf{1}_d - c) \tag{A.11}$$

Thus, based on the analysis, the optimum is achieved if $\frac{\partial \mathscr{L}}{\partial p_{ij}} = 0$ Thus,

$$p_{ij}^\lambda = \exp(-\frac{1}{2} - \lambda\gamma_i) \exp(-\lambda m_{ij}) \exp(-\frac{1}{2} - \lambda\beta_j) \tag{A.12}$$

Thus, based on the Sinkhorn-Knopp Theorem, there exists a diagonal matrix $diag(\boldsymbol{v})$ and $diag(\boldsymbol{v})$(unique due to convexity in this settings) such that

$$\exists \boldsymbol{u}, \boldsymbol{v} > \mathbf{0}_d : P^{\lambda} = diag(\boldsymbol{u}) \exp(-\lambda M) diag(\boldsymbol{v}) \tag{A.13}$$

Proof complete. □

Appendix B

Experiment Results

B.1 Toy Examples

B.1.1 GMM

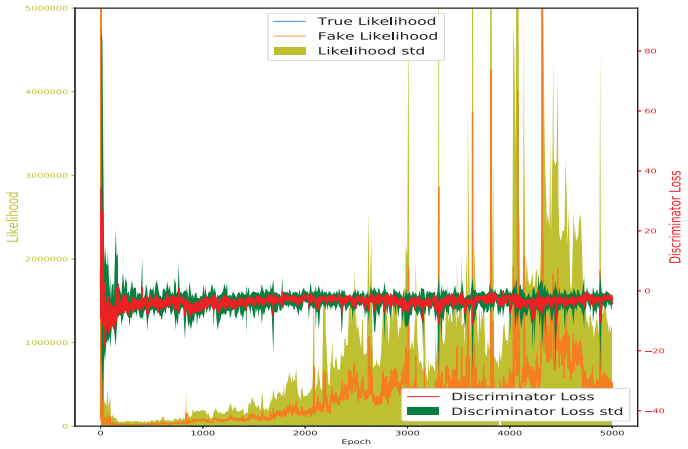
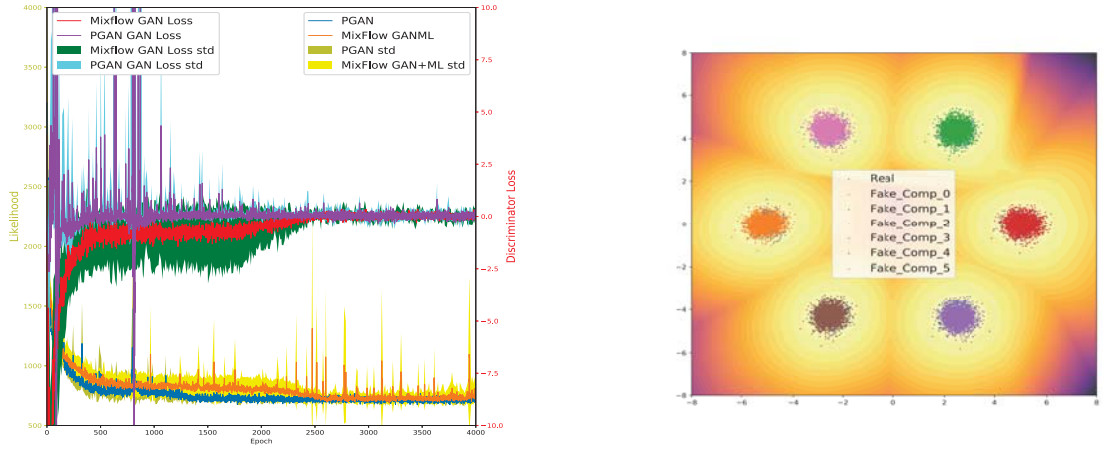


Fig. B.1 The Single Flow GAN training curve for GMM examples



(a) (b)

Fig. B.2 (a) Training Curve comparison between PGAN and Mixflow GAN+ML. The results are obtained by averaging 5 runs and ± 1 std. (b) The generated sampled obtained by PGAN.

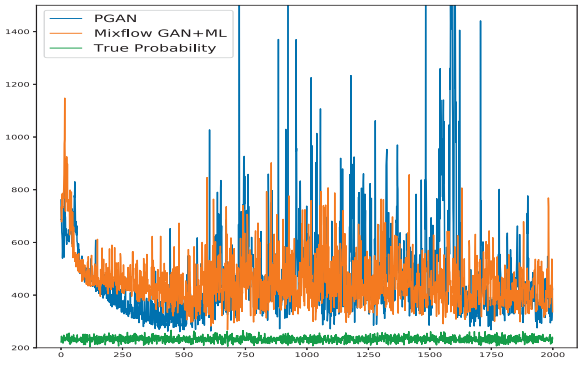
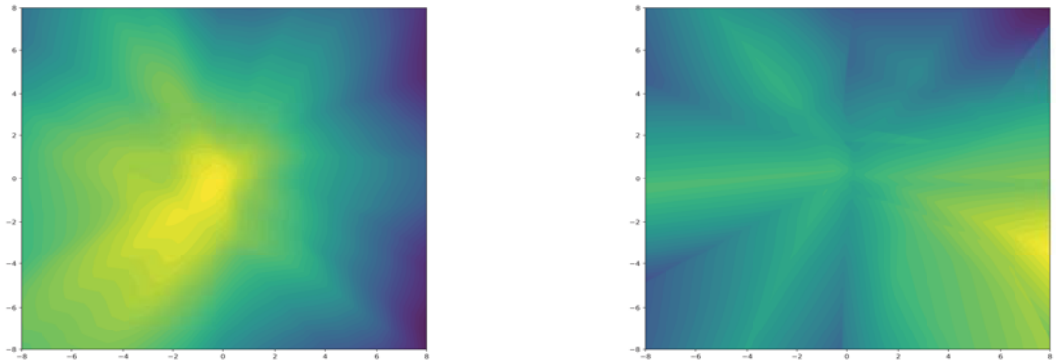


Fig. B.3 The training likelihood comparison of Mixflow and PGAN when model is crippled



(a) (b)

Fig. B.4 (a) The discriminator value surface learned by Mixflow GAN+ML training. (b) The discriminator value surface of PGAN. We observe the more complicated structure obtained by PGAN method as expected.

B.1.2 Swiss Roll

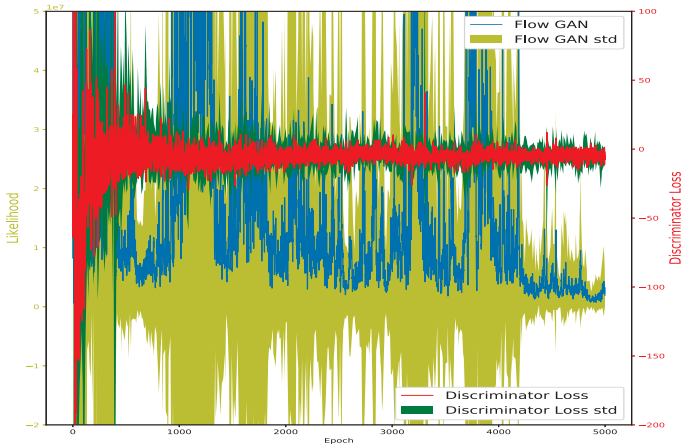
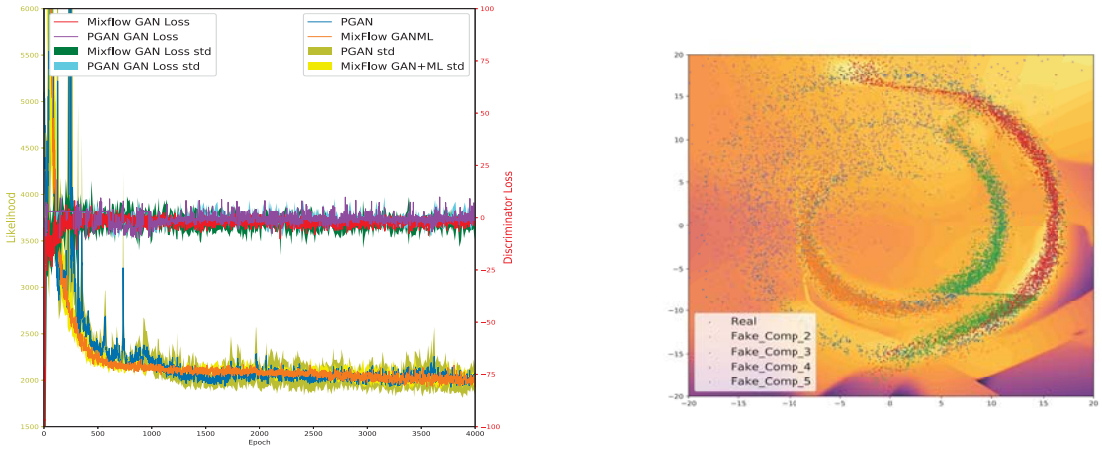


Fig. B.5 The training curve and GAN loss of single flow GAN training



(a) (b)

Fig. B.6 (a) The training curve comparison between PGAN and Mixflow GAN+ML. The results are obtained by averaging 5 runs and ± 1 std. (b) The sample generation of PGAN methods

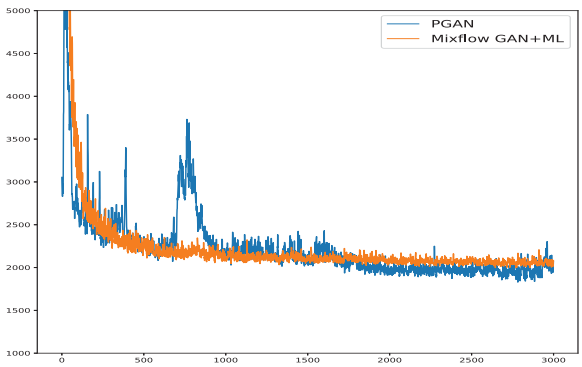


Fig. B.7 The training curve of PGAN and Mixflow GAN+ML when model is crippled.

B.2 MNIST

B.2.1 ReLu MLP Coupling function

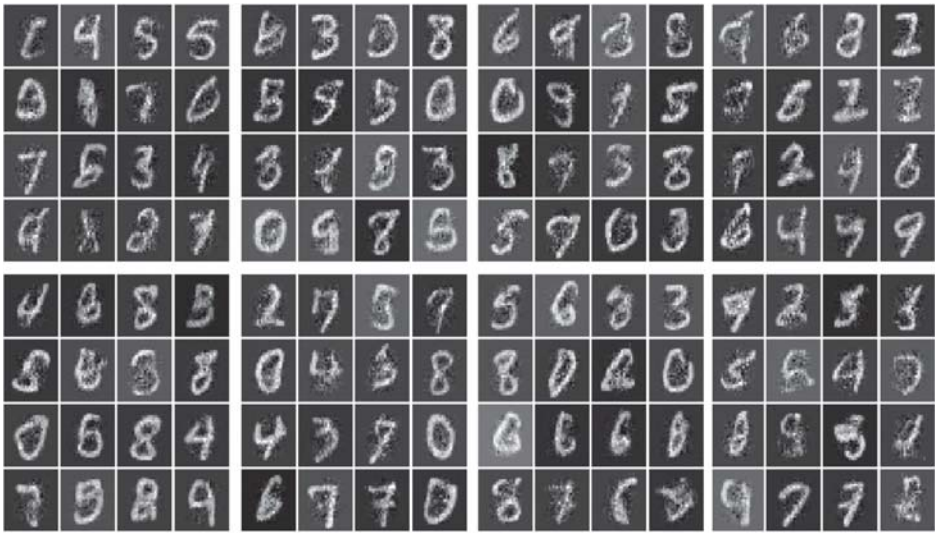


Fig. B.8 The generated image from single flow model with ML learning

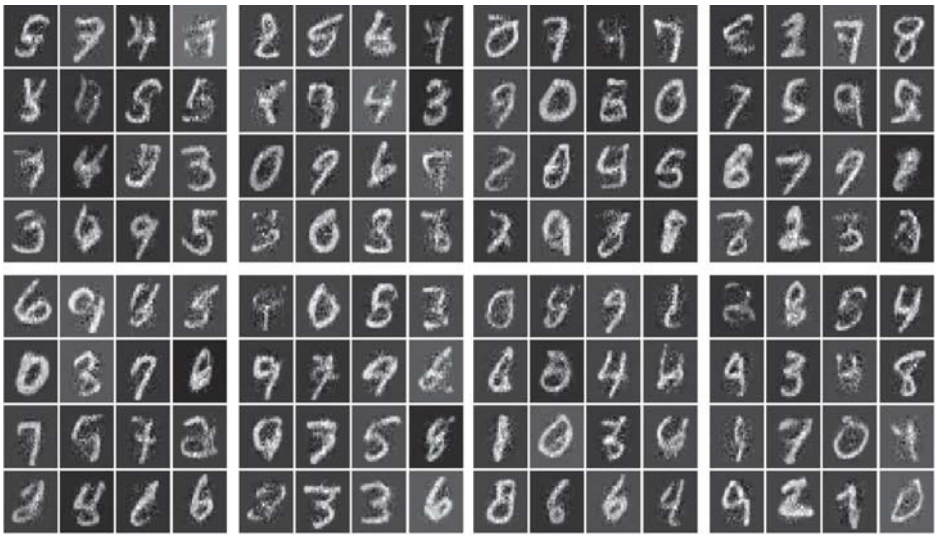


Fig. B.9 The generated image from Mixflow model with ML learning

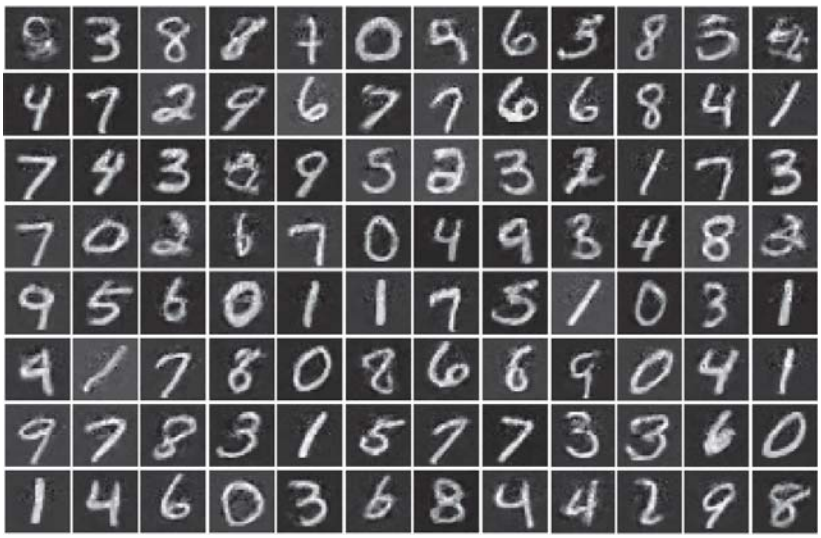


Fig. B.10 The generated image from Single flow model with pure GAN learning

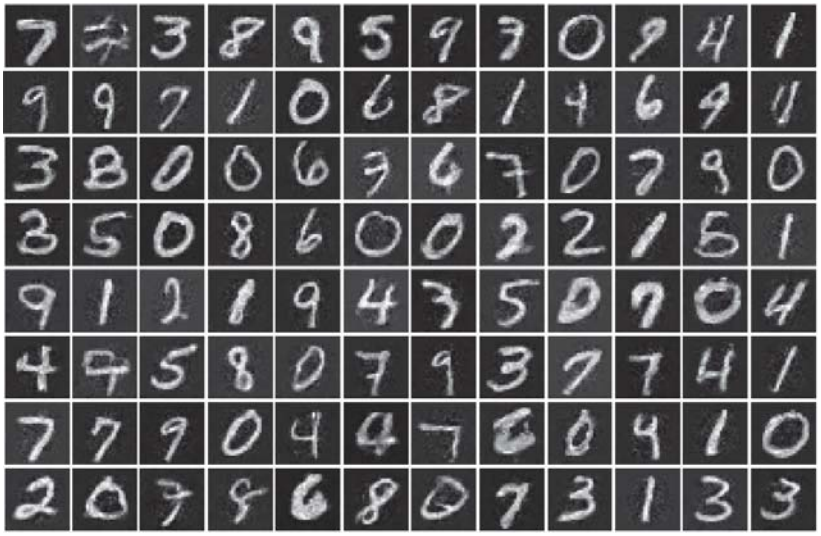


Fig. B.11 The generated image from Single flow model with GAN+ML learning

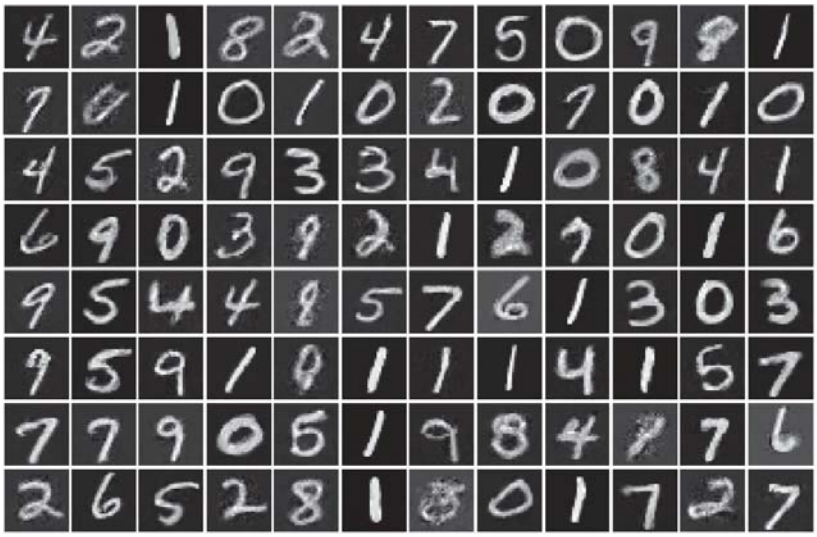


Fig. B.12 The generated image from Mixflow model with GAN+ML learning

B.2.2 CNN Coupling function

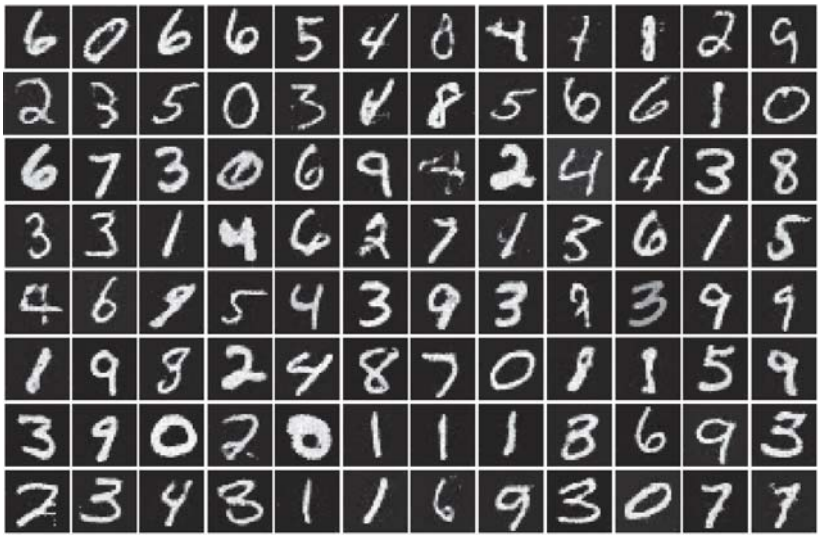


Fig. B.13 The generated image from single flow model with GAN+ML learning using CNN coupling function

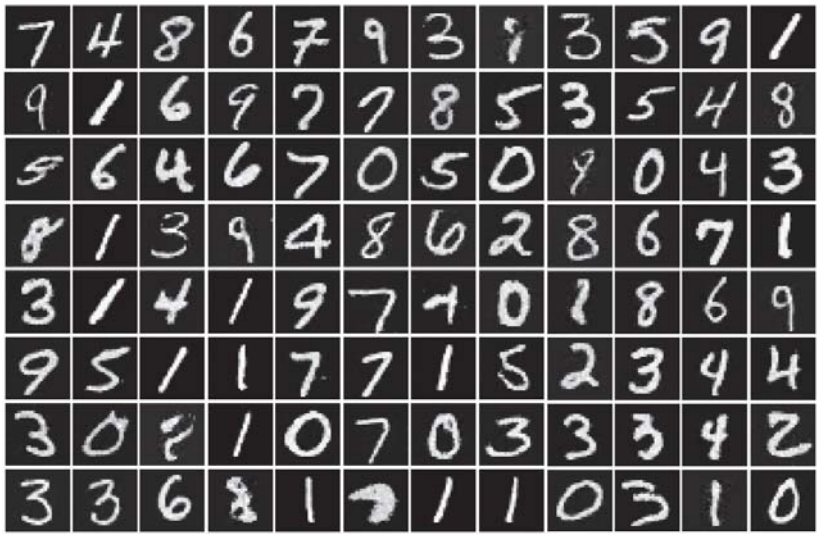


Fig. B.14 The generated image from Mixflow model with GAN+ML learning using CNN coupling function

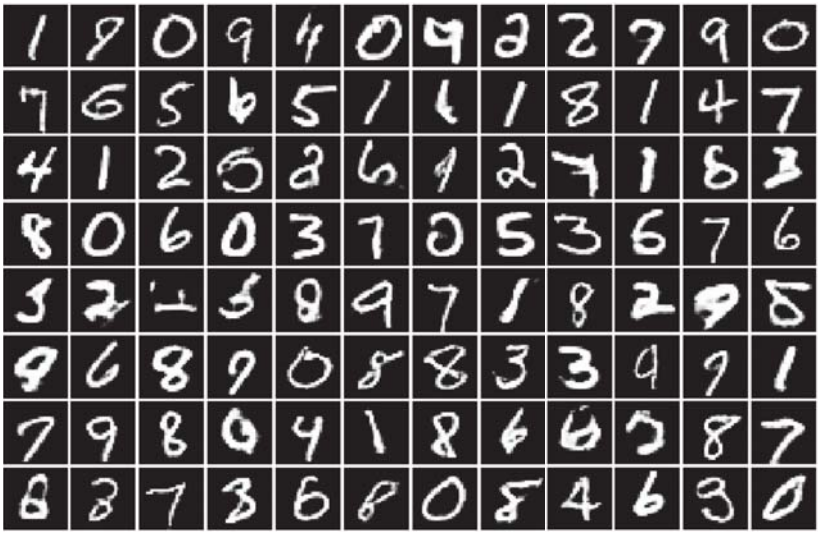


Fig. B.15 The generated image using DCGAN model