Generative Adversarial Networks Based on a General Parameterized Family of Generator Loss Functions

by

Justin Veiner

A thesis submitted to the
Department of Mathematics and Statistics
in conformity with the requirements for
the degree of Master of Applied Science

Queen's University
Kingston, Ontario, Canada
September 2023

Abstract

This thesis introduces a unifying parameterized generator loss function for generative adversarial networks (GANs). We establish an equilibrium theorem for our resulting GAN system under a canonical discriminator in terms of the so-called Jensen-f-divergence, a natural generalization of the Jensen-Shannon divergence to the f-divergence. We also show that our result recovers as special cases several GANs from the literature, including the original GAN, least square GAN (LSGAN), α-GAN and others. Finally, we systematically conduct experiments on three image datasets for different manifestations of our GAN system to illustrate their performance and stability.

Acknowledgments

I would first like to thank Professors Fady Alajaji and Bahman Gharesifard. I am extremely grateful for all the guidance, support, and mentorship you have offered me these past two years, and for your patience and encouragement whenever I hit a roadblock with my work. My undergraduate courses with both of you deepened my passion for mathematics and allowed me to be intellectually challenged.

I would like to also thank my family for their love and support, and for being there for me throughout the course of my degree. I am grateful to my friends for their immense support throughout my time at Queen's, and for always being a phone call or text away.

Thank you to the Department of Mathematics and Statistics and the Natural Sciences and Engineering Research Council of Canada for their financial support. Finally, I would like to thank Jennifer Read for her assistance and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Generative adversarial networks (GANs) were devised by Ian Goodfellow *et al.* in 2014 [11] as a novel unsupervised learning technique to generate synthetic data. A GAN consists of two components. The generator neural network generates data from stochastic noise. The discriminator neural network assigns the generated data and real data a score between 0 and 1, with a score closer to 1 corresponding to a higher belief that the data is real as opposed to synthetic. The adversarial component arises in the GAN's minimax game, which aims to maximize the discriminator's loss, diminishing its performance capabilities and allowing for the discriminator to confuse real data with synthetic data. Deep Convolutional GAN (DCGAN) [26] was subsequently introduced to use convolutional layers to extract higher-dimensional feature representations that are prevalent in more complex image datasets. DCGAN improves on the baseline GAN's ability to capture an underlying data distribution.

The GAN's competing objectives render it notoriously difficult to train. Particularly, GANs tend to exhibit mode collapse, which occurs when the generator finds a single output that can fool the discriminator, thus only producing that one output. As a result, there has been an increased focus on designing GANs that improve

stability and encourage sample diversity. InfoGAN [6] modifies the generator's input to include a latent code in addition to the noise component. The generated images are fed into a recognition network, which guesses the value of the latent code the generator used for the image. The objective function includes a mutual information term to maximize the mutual information between generated data and latent codes, thus encouraging diverse samples. Least Squares GAN (LSGAN) uses a squared-error based loss function to penalize outputs that deviate from their correct labels, creating more gradients for the generator and improving stability [25]. LSGAN has also been shown to generate better quality data than the original GAN. LSGAN was generalized through LkGAN [5], replacing the squared error loss for the generator's loss function with the absolute error distortion measure of order $k \geq 1$. Finally, RényiGAN [5] uses Rényi cross-entropy-based loss functions of order $\alpha > 0, \alpha \neq 1$. It has been shown that RényiGANs exhibit stable performance for $\alpha \geq 2$ [5].

GANs have also been studied in the context of f-divergences [23]. The original GAN problem reduces to minimizing the Jensen-Shannon divergence (JSD), which is itself an f-divergence. In [31], f-GAN is introduced using a class of loss functions parameterized by a lower semicontinuous function f, with the resulting problem reducing to minimizing an f-divergence. The f-GAN's loss function requires computing the Fenchel conjugate of a function f, resulting in a more involved computation. α-GAN [20, 21] uses a class-probability estimation (CPE)-based loss function parameterized by $\alpha > 0$. The resulting minimax game reduces to the minimization of the Arimoto divergence, itself an f-divergence. Both f-GAN and α-GAN can interpolate between various well-known GANs, including VanillaGAN, HellingerGAN, LSGAN and total variation GAN (TVGAN). The α-GAN work was expanded on in [41] with

the (α_D, α_G)-GAN, which utilizes a distinct α parameter for the generator and discriminator, thus posing a dual-objective optimization problem.

1.1 Contributions

The motivation of this thesis is to utilize a GAN system with a network of generators whose outputs are simultaneously judged by a single (centralized) discriminator. The network of generators is unified under a family of parameterized loss functions, from which each generator selects a loss function suited to its own problem. The novel contributions to this thesis are as follows. We introduce the Jensen-f-divergence, a generalization of the Jensen-Shannon divergence applied to f-divergences. We expand on the α-GAN work of [20,21,41], introducing a theoretical framework classifying the set of α-parameterized CPE-based loss functions. We then introduce $\mathcal{L}_\alpha$-GAN, which uses a generator loss function from this class, and a discriminator loss function that achieves the same maximum as the original GAN of [11] (also known as Vanilla-GAN). We prove that the minimax game played by the two loss functions results in the minimization of a Jensen-f_α-divergence, with the generating function f_α directly computed from the generator's loss function. We provide examples of existing GANs that are recovered by $\mathcal{L}_\alpha$-GAN and determine their equilibrium points. Finally, we conduct experiments on various realizations of $\mathcal{L}_\alpha$-GAN on three image datasets.[1]

1.2 Organization of Thesis

This thesis is organized as follows. We conduct a review of information measures, machine learning, and neural networks in Chapter 2. Notably, we introduce the

[1] Our simulation codes can be found at `https://github.com/justin-veiner/MASc`.

Jensen-f-divergence as a generalization of the Jensen-Shannon divergence. In Chapter 3, we present VanillaGAN and RényiGAN and derive their solutions. We also present InfoGAN, and reproduce the original work's results on the MNIST dataset. In Chapter 4, we present the theoretical basis for $\mathcal{L}_\alpha$-GAN. We provide examples of existing GANs that are encapsulated under this framework, and conduct experiments for different $\mathcal{L}_\alpha$-GAN manifestations. We provide a summary and avenues of further research in Chapter 5.

Chapter 2

Information Measures and Deep Learning Fundamentals

2.1 Information Measures

We begin by defining the information measures that will be used throughout this thesis. We follow the treatment in [1] unless otherwise stated. We also work in the standard probability space $(\mathbb{R}, \mathcal{B}(\mathbb{R}), \mu)$, where $\mathcal{B}(\mathbb{R})$ is the Borel σ-algenra on $\mathbb{R}$ and μ is the Lebesgue measure. Finally, we use the short-form $\int f \, d\mu = \int f(x) \, d\mu(x)$ for any measurable function f.

2.1.1 Entropies

Entropy measures the amount of information gained in an experiment after observing one of its outcomes. The definition was first devised by Claude Shannon in 1948 [38], which we recall below.

Definition 1. *Let $\mathcal{R}$ be a discrete set. The **Shannon entropy** of a discrete random*

variable X with probability mass function (pmf) P_X and support $\mathcal{R}$ is given by

$$H(X) = \mathbb{E}_{A\sim P_X}[-\log P_X(A)] = -\sum_{a\in\mathcal{R}} P_X(a)\log P_X(a), \tag{2.1}$$

assuming the sum exists.

Another notation for entropy $H(X)$ is $H(P_X)$. The units used to measure entropy depends on the base of the logarithm used in (2.1). The most commonly used units are bits (base 2) and nats (base e). For an arbitrary $D > 1$, we measure entropy in D-ary units. We can convert the units in (2.1) to D-ary units using the formula $H_D(X) = \frac{H(X)}{\log D}$. The Shannon entropy is nonnegative, and bounded above by $\log|\mathcal{R}|$ (when $\mathcal{R}$ is finite).

We now present the conditional Shannon entropy, which measures the entropy of a random variable X given that we know the value of another random variable Y.

Definition 2. *Let X and Y be two random variables with probability mass functions P_X and P_Y, respectively. Then the* ***conditional Shannon entropy of X given Y*** *is denoted by $H(X|Y)$ and is given by*

$$H(X|Y) = \mathbb{E}_{(A,B)\sim P_{X,Y}}[-\log P_{X|Y}(A|B)], \tag{2.2}$$

where $P_{X|Y}(\cdot|\cdot)$ denotes the conditional pmf of X given Y.

It can be shown that conditioning on another random variable never increases entropy, i.e.,, for random variables X and Y, we have that $H(X|Y) \leq H(X)$. Intuitively this makes sense, as obtaining additional information about an experiment will not increase the uncertainty of its outcome. We now present the chain rule for

entropy, which states that the joint entropy of n random variables can be expressed as a sum of conditional entropies.

Proposition 1. *Let $\mathbf{X} = (X_1, ..., X_n)$ be a tuple of $n \geq 2$ random variables, i.e., a random vector of size n. Then the Shannon entropy of $\mathbf{X}$ can be written as*

$$H(\mathbf{X}) = H(X_1, ..., X_n) = H(X_1) + \sum_{i=2}^{n} H(X_i|\mathbf{X}^{i-1}), \tag{2.3}$$

where $\mathbf{X}^{i-1} = (X_1, ..., X_{i-1})$ is the tuple consisting of the first $i-1$ random variables of the collection.

Proof. We first prove that (2.3) holds, assuming that $n = 2$, and then generalize it to an arbitrary collection of random variables; let X and Y be two random variables with marginal pmfs P_X and P_Y respectively, and joint pmf $P_{X,Y}$. Then by (2.2) we have that

$$\begin{aligned} H(Y|X) &= \mathbb{E}_{(A,B)\sim P_{X,Y}}[-\log P_{Y|X}(B\,|A)] \\ &= \mathbb{E}_{(A,B)\sim P_{X,Y}}\left[-\log \frac{P_{X,Y}(A,B)}{P_X(A)}\right] \\ &= \mathbb{E}_{(A,B)\sim P_{X,Y}}[-\log P_{X,Y}(A,B)] - \mathbb{E}_{(A,B)\sim P_{X,Y}}[-\log P_X(A)] \\ &= H(X,Y) - H(X). \end{aligned}$$

Rearranging this expression we get that

$$H(X,Y) = H(X) + H(Y|X). \tag{2.4}$$

This shows that the entropy chain rule holds for two random variables. We now

obtain the chain rule in (2.3) by applying (2.4) recursively; for a collection of random variables $X_1, ..., X_n$, we have that

$$\begin{aligned} H(X_1, ..., X_n) &= H(X_1, (X_2, ..., X_n)) \\ &= H(X_1) + H(X_2, ..., X_n|X_1) \\ &= H(X_1) + (H(X_2|X_1) + H(X_3, ..., X_n|\mathbf{X}^2)) \\ &= H(X_1) + H(X_2|X^1) + H(X_3|\mathbf{X}^2) + H(X_4, ..., X_n|\mathbf{X}^3) \\ &= \cdots \\ &= H(X_1) + \sum_{i=2}^{n} H(X_i|\mathbf{X}^{i-1}). \end{aligned}$$

□

The entropy chain rule states that joint entropy can be written as the sum of conditional entropies. Since conditioning never increases entropy, we have for $\mathbf{X} = (X_1, ..., X_n)$ that $H(\mathbf{X}) \leq \sum_{i=1}^{n} H(X_i)$, with equality if and only if (iff) $X_1, ..., X_n$ are independent.

Definition 3. *The **differential Shannon entropy** of a continuous random variable X with probability density function (pdf) f_X and support $\mathcal{R} \subseteq \mathbb{R}$ is denoted by $h(X)$ and given by*

$$h(X) = \mathbb{E}_{A\sim f_X}[-\log f_X(A)] = -\int_{\mathcal{R}} f_X \log f_X \, d\mu, \tag{2.5}$$

assuming the integral exists.

The conditional differential Shannon entropy formula is analagous to the discrete setup in Defintion 2. The chain rule presented in Proposition 1 also applies in the

continuous case. Unlike the discrete Shannon entropy, differential Shannon entropy can be negative, which we demonstrate with an example.

Example 1. *Let X be a uniform random variable on the interval $[0, 0.5]$. The pdf of X is given by*

$$f_X(a) = \begin{cases} 2, & 0 \leq a \leq 0.5 \\ 0, & \textit{otherwise.} \end{cases}$$

Then the differential Shannon entropy (measured in bits) is

$$\begin{aligned} h(X) &= -\int_{\mathbb{R}} f_X(x) \log_2 f_X(x)\, dx \\ &= -\int_0^{0.5} 2 \log_2 2\, dx \\ &= -1 \textit{ bit.} \end{aligned}$$

Alfred Rényi expanded on Shannon's definitions of entropy in 1961 as a way to generalize the original definition while retaining the additivity property for independent random variables from Shannon [34]. We now present the notions of Rényi's entropy.

Definition 4. *Let $\alpha > 0$ such that $\alpha \neq 1$, and let $\mathcal{R}$ be a discrete set. The **Rényi entropy** of a discrete random variable X with probability mass function P_X and support $\mathcal{R}$ is given by*

$$H_\alpha(X) = \frac{1}{1-\alpha} \log \left(\sum_{a \in \mathcal{R}} P_X(a)^\alpha \right). \tag{2.6}$$

Definition 5. *Let $\alpha > 0$ such that $\alpha \neq 1$. The **Rényi differential entropy** of a continuous random variable X with pdf f_X and support $\mathcal{R} \subseteq \mathbb{R}$ is given by*

$$h_\alpha(X) = \frac{1}{1-\alpha} \log \left(\int_{\mathcal{R}} f_X^\alpha \, d\mu \right). \tag{2.7}$$

It can be shown that the Rényi entropy approaches the Shannon entropy as α goes to 1. We state this result, providing its proof for completeness.

Proposition 2. *We have the following statements:*

(i) For a discrete random variable X with pmf P_X, we have that

$$\lim_{\alpha \to 1} H_\alpha(X) = H(X). \tag{2.8}$$

(ii) For a continuous random variable $\mathbf{Y}$ with pdf f_Y, we have that

$$\lim_{\alpha \to 1} h_\alpha(Y) = h(Y). \tag{2.9}$$

Proof. Let X be a discrete random variable with support $\mathcal{R}$ and pmf P_X. Then we have that

$$\begin{aligned}
\lim_{\alpha \to 1} H_\alpha(X) &= \lim_{\alpha \to 1} \frac{1}{1-\alpha} \log \left(\sum_{a \in \mathcal{R}} P_X(a)^\alpha \right) \\
&\overset{(a)}{=} \lim_{\alpha \to 1} \frac{\frac{\partial}{\partial \alpha} \left(\log \left(\sum_{a \in \mathcal{R}} P_X(a)^\alpha \right) \right)}{\frac{\partial}{\partial \alpha}(1-\alpha)} \\
&= \lim_{\alpha \to 1} \frac{\frac{\sum_{a \in \mathcal{R}} P_X(a)^\alpha \log P_X(a)}{\sum_{a \in \mathcal{R}} P_X(a)^\alpha}}{-1} \\
&= - \lim_{\alpha \to 1} \frac{\sum_{a \in \mathcal{R}} P_X(a)^\alpha \log P_X(a)}{\sum_{a \in \mathcal{R}} P_X(a)^\alpha}
\end{aligned}$$

$$\stackrel{(b)}{=} -\sum_{a\in\mathcal{R}} P_X(a)\log P_X(a)$$
$$= H(X),$$

where L'Hôpital's rule is applied in (a), and (b) holds since $\sum_{a\in\mathcal{R}} P_X(a) = 1$. □

The proof of the differential entropy case is similar.

2.1.2 Cross-Entropies

Cross-entropy measures the difference between two probability distributions p and q when conducting an experiment. We assume the true data distribution for the experiment is p, and that q is some approximation of p. Cross-entropies are useful in generative models, where we aim to measure the similarity between the real data with distribution p, and the generated data with distribution q. We begin by presenting the Shannon cross-entropy and its continuous analogue.

Definition 6. *The **Shannon cross-entropy** between two pmfs p and q with common discrete support $\mathcal{R}$ is given by*

$$H(p;q) = \mathbb{E}_{A\sim p}[-\log q(A)] = -\sum_{a\in\mathcal{R}} p(a)\log q(a). \tag{2.10}$$

It can easily be shown that $H(p;q) = H(p)$ iff $p = q$ almost everywhere ($a.e.$).

Definition 7. *The **differential Shannon cross-entropy** between two pdfs p and q with common support $\mathcal{R}\subseteq\mathbb{R}$ is given by*

$$h(p;q) = \mathbb{E}_{A\sim p}[-\log q(A)] = -\int_{\mathcal{R}} p\log q\, d\mu. \tag{2.11}$$

We next present Rényi's formulations of cross-entropy [34].

Definition 8. *Let $\alpha > 0$ such that $\alpha \neq 1$, and let $\mathcal{R}$ be a discrete set. The* ***Rényi cross-entropy*** *between two pmfs p and q with common support $\mathcal{R}$ is given by*

$$H_\alpha(p;q) = \frac{1}{1-\alpha} \log \left(\sum_{a \in \mathcal{R}} p(a) q^{\alpha-1}(a) \right). \tag{2.12}$$

Definition 9. *Let $\alpha > 0$ such that $\alpha \neq 1$. The* ***Rényi differential cross-entropy*** *of two pdfs p and q with common support $\mathcal{R} \subseteq \mathbb{R}$ is given by*

$$h_\alpha(p;q) = \frac{1}{1-\alpha} \log \left(\int_{\mathcal{R}} p q^{\alpha-1} \, d\mu \right). \tag{2.13}$$

As is the case with entropies, it can be shown that the Rényi cross-entropy of order α converges to the Shannon cross-entropy as α goes to 1. The Rényi cross-entropy as well as the Rényi divergence (which will be defined in the next section) were recently used in [13] within an information bottleneck framework for designing fair classification algorithms.

We conclude by stating that the second argument of cross-entropy does not have to be restricted to a distribution, but can be a non-negative function. When using a functional argument, we use the notation $\mathcal{H}(\cdot\,;\,\cdot)$.

2.1.3 Divergences

Like cross-entropy, divergence is also a measure of similarity between two distributions p and q. It also measures the inefficiency of conducting an experiment assuming the data's distribution is q when the true data distribution is actually p.

Definition 10. *Let $\mathcal{R}$ be a discrete set. The **Kullback-Leibler divergence** between two pmfs p and q with common support $\mathcal{R}$ is given by*

$$\mathrm{KL}(p||q) = \mathbb{E}_{A\sim p}\left[\log \frac{p(A)}{q(A)}\right] = \sum_{a\in\mathcal{R}} p(a)\log\frac{p(a)}{q(a)}. \tag{2.14}$$

Definition 11. *Let p and q be two probability density functions with supports S_p and S_q respectively, where $S_p \subseteq S_q \subseteq \mathbb{R}$. The **differential Kullback-Leibler divergence** between p and q is denoted by $\mathrm{KL}(p||q)$ and given by*

$$\mathrm{KL}(p||q) = \mathbb{E}_{A\sim p}\left[\log \frac{p(A)}{q(A)}\right] = \int_{S_p} p\log\frac{p}{q}\,d\mu. \tag{2.15}$$

We clearly have that $\mathrm{KL}(p||q) = 0$ iff $p = q$ (a.e.). We now present the Jensen-Shannon divergence, which is the arithmetic average of two Kullback-Leiber divergences, where each individual distribution is compared to the mixture of the two.

Definition 12. *The **Jensen-Shannon divergence** between two distributions p and q is denoted by $\mathrm{JSD}(p||q)$ and given by*

$$\mathrm{JSD}(p||q) = \frac{1}{2}\mathrm{KL}\left(p\middle|\middle|\frac{p+q}{2}\right) + \frac{1}{2}\mathrm{KL}\left(q\middle|\middle|\frac{p+q}{2}\right). \tag{2.16}$$

Definition 13. *Let $\alpha > 0$ such that $\alpha \neq 1$, and let $\mathcal{R}$ be a discrete set. The **Rényi divergence** between two pmfs p and q with common support $\mathcal{R}$ is denoted by $D_\alpha(p||q)$ and given by*

$$D_\alpha(p||q) = \frac{1}{\alpha-1}\log\left(\sum_{a\in\mathcal{R}} p^\alpha(a)q^{1-\alpha}(a)\right). \tag{2.17}$$

Definition 14. *Let $\alpha > 0$ such that $\alpha \neq 1$. The **differential Rényi divergence** between two pdfs p and q with common support $\mathcal{R} \subseteq \mathbb{R}$ is denoted by $D_\alpha(p||q)$ and given by*

$$D_\alpha(p||q) = \frac{1}{\alpha - 1} \log \left(\int_{\mathcal{R}} p^\alpha q^{1-\alpha} \, d\mu \right). \tag{2.18}$$

We next present the Jensen-Rényi divergence, which is the Rényi analogue of the Jensen-Shannon divergence [5].

Definition 15. *Let $\alpha > 0$ such that $\alpha \neq 1$. The **Jensen-Rényi divergence** between two distributions p and q with common support $\mathcal{R} \subseteq \mathbb{R}$ is denoted by $\mathrm{JR}_\alpha(p||q)$ and given by*

$$\mathrm{JR}_\alpha(p||q) = \frac{1}{2} D_\alpha \left(p \middle|\middle| \frac{p+q}{2} \right) + \frac{1}{2} D_\alpha \left(q \middle|\middle| \frac{p+q}{2} \right). \tag{2.19}$$

Definition 16. [29] *The **Pearson χ^2 divergence** between two pdfs p and q with common support $\mathcal{R} \subseteq \mathbb{R}$ is denoted by $\chi^2(p||q)$ and given by*

$$\chi^2(p||q) = \int_{\mathcal{R}} \frac{(q-p)^2}{p} \, d\mu. \tag{2.20}$$

Definition 17. [29] *For a fixed $k > 1$, the **Pearson-Vajda divergence of order k** between two pdfs p and q with common support $\mathcal{R} \subseteq \mathbb{R}$ is denoted by $|\chi|^k(p||q)$ and given by*

$$|\chi|^k(p||q) = \int_{\mathcal{R}} \frac{|q-p|^k}{p^{k-1}} \, d\mu. \tag{2.21}$$

It directly follows from the above definitions that the Pearson-Vajda divergence

recovers the Pearson χ^2 divergence when $k = 2$.

Definition 18. [2, 23, 32] *Let $\alpha > 0$ such that $\alpha \neq 1$. The **Arimoto divergence of order $\boldsymbol{\alpha}$** between two pdfs p and q with common support $\mathcal{R} \subseteq \mathbb{R}$, is denoted by $\mathcal{A}_\alpha(p||q)$, and is given by*

$$\mathcal{A}_\alpha(p||q) = \frac{\alpha}{\alpha - 1}\left(\int_{\mathcal{R}} (p^\alpha + q^\alpha)^{\frac{1}{\alpha}}\, d\mu - 2^{\frac{1}{\alpha}}\right). \tag{2.22}$$

Definition 19. [15, 23] *Let $\alpha > 0$ such that $\alpha \neq 1$. The **Hellinger divergence of order** α between two pdfs p and q with common support $\mathcal{R} \subseteq \mathbb{R}$ is denoted by $\mathscr{H}_\alpha(p||q)$ and given by*

$$\mathscr{H}_\alpha(p||q) = \frac{1}{\alpha - 1}\left(\int_{\mathcal{R}} p^\alpha q^{1-\alpha}\, d\mu - 1\right). \tag{2.23}$$

We next discuss f-divergences, a class of divergences that encompasses a large number of defined divergences.

Definition 20. [23] *Let $f : [0, \infty) \to \mathbb{R}$ be a continuous convex function such that $f(1) = 0$. The **f-divergence** between two probability densities p and q with common support $\mathcal{R} \subseteq \mathbb{R}$ is denoted by $D_f(p||q)$, and given by*

$$D_f(p||q) = \mathbb{E}_{A\sim q}\left[f\left(\frac{p(A)}{q(A)}\right)\right] = \int_{\mathcal{R}} q\, f\left(\frac{p}{q}\right)\, d\mu. \tag{2.24}$$

We require that $f(1) = 0$ to satisfy the definiteness property of divergences, i.e., $D_f(p||q) = 0$ iff $p = q$ (a.e.).

Lemma 1. *The Kullback-Leibler divergence and Pearson χ^2 divergence are both f-divergences.*

Proof. First, let p and q be two distributions with common support $\mathcal{R}$. Next, define $f_1 : [0, \infty) \to \mathbb{R}$ as $f_1(x) = x \log x$, with the convention that $0 \log 0 = 0$. f is clearly continuous on $[0, \infty)$. Furthermore, $f_1''(x) = \frac{1}{x} > 0$ for every $x \in [0, \infty)$. Therefore, f_1 is also a convex function. We now compute the required expectation:

$$\mathbb{E}_q \left[f_1 \left(\frac{p}{q} \right) \right] = \int_{\mathcal{R}} q \, f_1 \left(\frac{p}{q} \right) \, d\mu \tag{2.25}$$

$$= \int_{\mathcal{R}} q \left(\frac{p}{q} \log \frac{p}{q} \right) \, d\mu \tag{2.26}$$

$$= \int_{\mathcal{R}} p \log \frac{p}{q} \, d\mu \tag{2.27}$$

$$= \mathrm{KL}(p||q). \tag{2.28}$$

Since f_1 is a continuous convex function with $f(1) = 0$, and we have shown we can write the KL divergence according to Definition 20, we have that $\mathrm{KL}(\cdot||\cdot)$ is an f-divergence.

We now show that the Pearson χ^2 divergence is an f-divergence. Define $f_2 : [0, \infty) \to \mathbb{R}$ as $f_2(x) = \frac{1}{x} - 2 + x = \left(\sqrt{x} - \frac{1}{\sqrt{x}} \right)^2$. Note that f_2 is continuous on $[0, \infty)$ (by convention, we take $f_2(0) = +\infty$). Furthermore, we have that $f_2''(x) = \frac{2}{x^3} > 0$ for every $x \in [0, \infty)$. Therefore, f_2 is a convex function. Proceeding as we did before, we have that

$$\mathbb{E}_q \left[f_2 \left(\frac{p}{q} \right) \right] = \int_{\mathcal{R}} q f \left(\frac{p}{q} \right) \, d\mu \tag{2.29}$$

$$= \int_{\mathcal{R}} q \left(\frac{q}{p} - 2 + \frac{p}{q} \right) \, d\mu \tag{2.30}$$

$$= \int_{\mathcal{R}} \left(\frac{q^2}{p} - 2q + p \right) \, d\mu \tag{2.31}$$

$$= \int_{\mathcal{R}} \frac{p(q^2/p - 2q + p)}{p} d\mu \tag{2.32}$$

$$= \int_{\mathcal{R}} \frac{q^2 - 2pq + p^2}{p} d\mu \tag{2.33}$$

$$= \int_{\mathcal{R}} \frac{(q-p)^2}{p}\, d\mu \tag{2.34}$$

$$= \chi^2(p||q). \tag{2.35}$$

Since f_2 is a continuous convex function with $f_2(1) = 0$, and we have shown we can write the KL divergence according to Definition 20, we have that $\chi^2(\cdot||\cdot)$ is an f-divergence. □

We summarize more examples of f-divergences under various choices of their generating function f in Table 2.1. We will be invoking these divergence measures in different parts of this thesis.

f-Divergence	Symbol	Formula	$f(u)$
Kullback-Leibler [19]	KL	$\int_{\mathcal{R}} p \log\left(\frac{p}{q}\right) d\mu$	$u\log(u)$
Jensen-Shannon [28]	JSD	$\frac{1}{2}\mathrm{KL}\left(p\middle\|\middle\|\frac{p+q}{2}\right) + \frac{1}{2}\mathrm{KL}\left(q\middle\|\middle\|\frac{p+q}{2}\right)$	$\frac{1}{2}\left(-(u+1)\log\left(\frac{u+1}{2}\right) + u\log(u)\right)$
Pearson χ^2 [29]	χ^2	$\int_{\mathcal{R}} \frac{(q-p)^2}{p}\, d\mu$	$\left(\sqrt{x} - \frac{1}{\sqrt{x}}\right)^2$
Pearson-Vajda ($k \geq 1$) [29]	$\lvert\chi\rvert^k$	$\int_{\mathcal{R}} \frac{\lvert q-p\rvert^k}{p^{k-1}}\, d\mu$	$u^{1-k}\lvert 1-u\rvert^k$
Arimoto ($\alpha > 0$, $\alpha \neq 1$) [32] [23]	$\mathcal{A}_\alpha$	$\frac{\alpha}{\alpha-1}\left(\int_{\mathcal{R}}(p^\alpha + q^\alpha)^{\frac{1}{\alpha}}\, d\mu - 2^{\frac{1}{\alpha}}\right)$	$\frac{\alpha}{\alpha-1}\left((1+u)^{\frac{1}{\alpha}} - (1+u) - 2^{\frac{1}{\alpha}} + 2\right)$
Hellinger ($\alpha > 0$, $\alpha \neq 1$) [15, 23]	$\mathscr{H}_\alpha$	$\frac{1}{\alpha-1}\left(\int_{\mathcal{R}} p^\alpha q^{1-\alpha}\, d\mu - 1\right)$	$\frac{u^\alpha - 1}{\alpha - 1}$

Table 2.1: Examples of f-divergences.

Note that the Rényi divergence in Definition 14 is not an f-divergence; however, it can be expressed as a transformation of the Hellinger divergence (presented in Definition 19), which itself is an f-divergence.

Lemma 2. *The Rényi divergence is a transformation of the Hellinger divergence.*

Proof. Let p and q be two probability distributions with common support $\mathcal{R}$. We

define the transformation T on the set of f-divergences as $T(\mathscr{D}) = \frac{1}{\alpha-1}\log(1+(\alpha-1)\mathscr{D})$. Applying this transformation to the Hellinger divergence, we have that

$$T(\mathscr{H}_\alpha(p||q)) = \frac{1}{\alpha-1}\log(1+(\alpha-1)\mathscr{H}_\alpha(p||q)) \tag{2.36}$$

$$= \frac{1}{\alpha-1}\log\left(1+(\alpha-1)\left(\frac{1}{\alpha-1}\left(\int_{\mathcal{R}} p^\alpha q^{1-\alpha}\,d\mu - 1\right)\right)\right) \tag{2.37}$$

$$= \frac{1}{\alpha-1}\log\left(\int_{\mathcal{X}} p^\alpha q^{1-\alpha}\,d\mu\right) \tag{2.38}$$

$$= D_\alpha(p||q). \tag{2.39}$$

Therefore, we have shown that the Rényi divergence is a transformation of the Hellinger divergence, which itself is an f-divergence. □

We now introduce a new measure, the Jensen-f-divergence, which is analagous to the Jensen-Shannon and Jensen-Rényi divergences [39].

Definition 21. *Let $f : [0,\infty) \to \mathbb{R}$ be a continuous convex function with $f(1) = 0$, and let $D_f(\cdot||\cdot)$ be its respective f-divergence. Then, the* ***Jensen-f-divergence*** *between two probability distributions p and q with common support $\mathcal{R} \subseteq \mathbb{R}$ is denoted by $\mathrm{JD}_f(p||q)$, and given by*

$$\mathrm{JD}_f(p||q) = \frac{1}{2}D_f\left(p\middle|\middle|\frac{p+q}{2}\right) + \frac{1}{2}D_f\left(q\middle|\middle|\frac{p+q}{2}\right). \tag{2.40}$$

Remark 3. *Examining (2.40), we note that the Jensen-f-divergence between p and q involves the f-divergences between either p or q and their mixture $(p+q)/2$. In other words to determine $\mathrm{JD}_f(p||q)$, we only need $f\left(\frac{2p}{p+q}\right)$ and $f\left(\frac{2q}{p+q}\right)$ when taking the expectations in (2.24). Thus, it is sufficient to restrict the domain of the convex function f to the interval $[0,2]$.*

The next result shows that the Jensen-Shannon divergence is a Jensen-f-divergence.

Lemma 3. *Let p and q be two densities with common support $\mathcal{R} \subseteq \mathbb{R}$, and let $f : [0, \infty) \to \mathbb{R}$ be given by $f(u) = u \log u$. Then we have that*

$$\mathrm{JD}_f(p||q) = \mathrm{JSD}(p||q). \tag{2.41}$$

Proof. We first note that f is continuous on $[0, \infty)$ and that $f(1) = 0$. Furthermore, we have that $f''(u) = \frac{1}{u}$, which is positive for $u > 0$. Therefore, f is also convex, and is a suitable function to construct a Jensen-f-divergence. We then have that

$$\begin{aligned}
\mathrm{JSD}(p||q) &= \frac{1}{2}\mathrm{KL}\left(p\middle|\middle|\frac{p+q}{2}\right) + \frac{1}{2}\mathrm{KL}\left(q\middle|\middle|\frac{p+q}{2}\right) \\
&= \frac{1}{2}\int_{\mathcal{R}} p \log\left(\frac{2p}{p+q}\right)\, d\mu + \frac{1}{2}\int_{\mathcal{R}} q \log\left(\frac{2q}{p+q}\right)\, d\mu \\
&= \frac{1}{2}\int_{\mathcal{R}} \frac{p+q}{2}\left(\frac{2p}{p+q}\log\left(\frac{2p}{p+q}\right)\right)\, d\mu \\
&\quad + \frac{1}{2}\int_{\mathcal{R}} \frac{p+q}{2}\left(\frac{2q}{p+q}\log\left(\frac{2q}{p+q}\right)\right)\, d\mu \\
&= \mathrm{JD}_f(p||q).
\end{aligned}$$

□

2.1.4 Mutual Informations

Mutual information is a measure of how much information two distributions share, in turn quantifying the dependence between them. We first present the original Shannon mutual information, then present its generalizations. We assume herein discrete random variables, but the definitions also apply to continuous random variables (with

the usual modifications).

Definition 22. *The **Shannon mutual information** between two random variables X and Y with respective marginal distributions P_X and P_Y and joint distribution $P_{X,Y}$ is denoted by $\mathrm{I}(X;Y)$ and given by*

$$\mathrm{I}(X;Y) = \mathrm{KL}(P_{X,Y}||P_X P_Y) = \mathbb{E}_{(A,B)\sim P_{X,Y}}\left[\frac{P_{X,Y}(A,B)}{P_X(A)P_Y(B)}\right]. \tag{2.42}$$

From (2.42), we clearly have that $\mathrm{I}(X;Y) = \mathrm{I}(Y;X)$ (i.e., the Shannon mutual information is symmetric). Furthermore, if X and Y are independent random variables, then $\mathrm{I}(X;Y) = 0$, since $P_{X,Y} = P_X P_Y$, and $\mathrm{KL}(p||q) = 0$ iff $p = q$ (a.e.). We now present an alternative formula for Shannon mutual information, expressed in terms of Shannon entropies.

Proposition 4. *Let X and Y be random variables with respective marginal distributions P_X and P_Y and joint distribution $P_{X,Y}$. Then we have that*

$$\mathrm{I}(X;Y) = H(X) - H(X|Y), \tag{2.43}$$

where $H(\cdot)$ is the Shannon entropy and $H(\cdot|\cdot)$ is the Shannon conditional entropy.

Proof. We have that

$$\begin{aligned}
\mathrm{I}(X;Y) &= \mathbb{E}_{(A,B)\sim P_{X,Y}}\left[\log\frac{P_{X,Y}(A,B)}{P_X(A)P_Y(B)}\right] \\
&\overset{(*)}{=} \mathbb{E}_{(A,B)\sim P_{X,Y}}\left[\log\frac{P_{X|Y}(A|B)}{P_X(A)}\right] \\
&= \mathbb{E}_{(A,B)\sim P_{X,Y}}\left[-\log P_X(A)\right] + \mathbb{E}_{(A,B)\sim P_{X,Y}}\left[\log P_{X|Y}(A|B)\right] \\
&= \mathbb{E}_{A\sim P_X}\left[-\log P_X(A)\right] + \mathbb{E}_{(A,B)\sim P_{X,Y}}\left[\log P_{X|Y}(A|B)\right]
\end{aligned}$$

$$= H(X) - H(X|Y),$$

where $(*)$ holds since $P_{X|Y}(a|b) = \frac{P_{X,Y}(a,b)}{P_Y(b)}$. □

From Proposition 4, we can derive two more identities:

(i) By symmetry of (2.42), we have that

$$\mathrm{I}(X;Y) = \mathrm{I}(Y;X) = H(Y) - H(Y|X).$$

(ii) Applying the chain rule for entropies to (2.43), we have that

$$\begin{aligned}\mathrm{I}(X;Y) &= H(X) - H(X|Y) \\ &= H(X) - (H(X,Y) - H(Y)) \\ &= H(X) + H(Y) - H(X,Y).\end{aligned}$$

We now present three definitions of α-mutual information, which are formulated using Rényi information measures [40].

Definition 23. *Let $\alpha > 0$ such that $\alpha \neq 1$. The **Arimoto mutual information** between two random variables X and Y with respective supports $\mathcal{R}_X$ and $\mathcal{R}_Y$, marginal distributions P_X and P_Y and joint distribution $P_{X,Y} = P_{Y|X}\, P_X$ is denoted by $\mathrm{I}^a_\alpha(X;Y)$ and given by*

$$\mathrm{I}^a_\alpha(X;Y) = H_\alpha(X) - H^a_\alpha(X|Y) \tag{2.44}$$

where

$$H_\alpha^a(X|Y) = \frac{\alpha}{1-\alpha} \log \mathbb{E}||P_{Y|X}(\cdot|X)||_\alpha \tag{2.45}$$

is the ***Arimoto Conditional Rényi entropy****, and* $||\mathbf{x}||_\alpha = (\sum_i x_i^\alpha)^{\frac{1}{\alpha}}$ *is the* α*-norm.*

The Arimoto mutual information can also be expressed in closed form as

$$\mathrm{I}_\alpha^a(X;Y) = \frac{\alpha}{1-\alpha} E_0\left(\frac{1}{\alpha} - 1, P_{X_\alpha}\right), \tag{2.46}$$

where P_{X_α} and E_0 are given by the following expressions:

$$P_{X_\alpha}(a) = -\alpha \log P_X(a) - (1-\alpha) H_\alpha(X), \tag{2.47}$$

$$E_0(\rho, P_X) = -\log\left(\sum_{b\in\mathcal{R}_Y}\left(\sum_{a\in\mathcal{R}_X} P_X(a) P_{Y|X}^{\frac{1}{1+\rho}}(b|a)\right)^{1+\rho}\right). \tag{2.48}$$

Definition 24. [40] *The* ***Csiszár mutual information*** *between two random variables* $\mathbf{X}$ *and* $\mathbf{Y}$ *is given by the following optimization:*

$$\mathrm{I}_\alpha^c(X;Y) = \min_{Q_Y} \mathbb{E}[D_\alpha(P_{Y|X}(\cdot|X) \,||\, Q_Y]. \tag{2.49}$$

The minimization in (2.49) is over all possible distributions of Y. This optimization is complicated to solve; so the Cziszár mutual information is difficult to compute even in more trivial scenarios.

We next present Sibson's formulation of α-mutual information. Sibson wanted to quantify the dissimilarity between a finite number of probability measures in order to place a prior distribution on them.

Definition 25. *The **Sibson mutual information** between two random variables X and Y with respective marginal distributions P_X and P_Y and joint distribution $P_{X,Y} = P_{Y|X}P_X$ is given by the following optimization:*

$$\mathrm{I}_\alpha^s(X;Y) = \min_{Q_Y} D_\alpha(P_{Y|X}||Q_Y|P_X) \tag{2.50}$$

Remark 5. *All above information measures can be similarly defined for random vectors with common support in $\mathbb{R}^d$.*

2.2 Machine Learning

Machine learning is a subfield of artificial intelligence that aims to learn trends in datasets without human assistance [27]. In this section, we provide an overview of dataset structures and training objectives often used in machine learning algorithms, and briefly review supervised and unsupervised learning.

Definition 26. *Let $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^m$. A **dataset**, denoted by $\mathcal{D}$ is the set of tuples*

$$\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}_{i=1}^M \tag{2.51}$$

*where $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}$ for $i = 1, ..., M$. We call $\mathbf{x}_i \in \mathcal{X}$ a **feature** (or covariate), and $\mathbf{y}_i \in \mathcal{Y}$ a **response** (or label).*

Definition 27. *The **training data**, is the subset $\mathcal{T} \subset \mathcal{D}$ of the dataset*

$$\mathcal{T} = \{(\mathbf{x}_{i_1}, \mathbf{y}_{i_1}), (\mathbf{x}_{i_2}, \mathbf{y}_{i_2}), \ldots, (\mathbf{x}_{i_N}, \mathbf{y}_{i_N})\} \tag{2.52}$$

*where $(i_1, ..., i_N)$ is an N-permutation of $(1, ..., M)$ for $N < M$. Furthermore, we denote $\mathcal{S} \subset \mathcal{D}$ as the **testing data**, where*

$$\mathcal{S} = \mathcal{D} \setminus \mathcal{T} \tag{2.53}$$

has size $|\mathcal{S}| = M - N$.

The training data $\mathcal{T}$ is used as the reference for the neural network to learn. The testing data $\mathcal{S}$ is used to evaluate the neural network's performance after training. A common partition of the dataset is to use 80% of the data for training and 20% for testing [37]. A portion of the training data can also be set aside as the validation data. The validation set is used to evaluate a model's performance during training. It is important to leave a part of the dataset for testing to observe how well the model generalizes to data samples it was not trained on.

A loss function quantifies how well a model performs its designated task, by computing some function of the label for a feature, and the label the model predicts for a feature [12]. Ideally, the value of a loss function is high for a poor prediction, and low for a good prediction.

Definition 28. *A **loss function** is a function $V : \mathcal{Y} \times \hat{\mathcal{Y}} \to \mathbb{R}_+$*

$$(\mathbf{y}, \hat{\mathbf{y}}) \mapsto V(\mathbf{y}, \hat{\mathbf{y}}) \tag{2.54}$$

where $\mathbf{y}$ *is a real label, and* $\hat{\mathbf{y}}$ *is a predicted label.*

We can rewrite Definition 28 in terms of a statistical model $f_{\boldsymbol{\theta}}$: if $\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x})$ is the model's prediction, we can redefine the loss function as $V : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$:

$$(\mathbf{x}, \mathbf{y}) \mapsto V(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x})). \tag{2.55}$$

This reformulation is useful for stating the mathematical problem behind a neural network, which will be discussed later in this section.

One commonly-used loss function is the 0-1 loss. This loss function is ideal for classification problems, where there are a finite number of classes. The loss function's value is zero for a correct prediction, and one for an incorrect prediction, mathematically stated as

$$V_{0,1}(\mathbf{y}, \hat{\mathbf{y}}) = \begin{cases} 0, & \mathbf{y} = \hat{\mathbf{y}} \\ 1, & \mathbf{y} \neq \hat{\mathbf{y}}. \end{cases} \tag{2.56}$$

If we have a batch of real and predicted labels $\{(\mathbf{y}_1, \hat{\mathbf{y}}_1), \ldots, (\mathbf{y}_N, \hat{\mathbf{y}}_N)\}$, then we can obtain the model's error rate on this batch by computing the average 0-1 loss value amongst all pairs of labels:

$$\% \text{ Error} = \frac{1}{N} \sum_{i=1}^{N} V_{0,1}(\mathbf{y}_i, \hat{\mathbf{y}}_i).$$

Example 2. *Suppose we want to build a neural network that aims to classify emails as legitimate or spam. After training the neural network, we feed a batch of 7 emails*

to the neural network. The network's predictions are summarized in the following table:

#	Label	Prediction	$V_{0,1}(y, \hat{y})$
1	Legitimate	Legitimate	0
2	Spam	Spam	0
3	Legitimate	Legitimate	0
4	Legitimate	Spam	1
5	Legitimate	Legitimate	0
6	Spam	Spam	0
7	Spam	Legitimate	1

Table 2.2: Summary of spam classification network

The error rate of this batch is

$$\begin{aligned} \%\ \textit{Error} &= \frac{1}{7}\sum_{i=1}^{7} V_{0,1}(y, \hat{y}) \\ &= \frac{1}{7}(0+0+0+1+0+0+1) \\ &= \frac{2}{7}. \end{aligned}$$

Another popular choice of loss function for binary classification problems is the binary cross-entropy loss (BCE loss): if $\mathbf{y} \in \{0,1\}^N$ is a label vector, and $\hat{\mathbf{y}} \in [0,1]^N$ is the vector of predicted labels, the BCE loss is given by

$$V_{\mathrm{BCE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N} V_{\mathrm{BCE}}(y_i, \hat{y}_i) = \frac{1}{N}\sum_{i=1}^{N}(-y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)). \quad (2.57)$$

Note that $V_{\mathrm{BCE}}(y_i, \hat{y}_i) = H(p_i; q_i)$ given in Definition 6, where $\mathcal{R} = \{0,1\}$, $p_i(0) =$

$1 - p_i(1) = y_i$ and $q_i(0) = 1 - q_i(1) = \hat{y}_i$. For regression problems, we cannot use the 0-1 loss function, as we are now working in a continuous space, and cannot apply a discrete metric to such a problem. One popular loss function applied to regression problems is the **mean squared error loss** (MSE) [27], defined as

$$V_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}||\mathbf{y} - \hat{\mathbf{y}}||_2^2, \quad \mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^N, \tag{2.58}$$

where $||\cdot||_2$ denotes the 2-norm. The MSE loss computes the squared difference between a real and predicted label, therefore rewarding a prediction close to the actual value with a low loss value.

Often in generative models, the goal is to generate output resembling data in a dataset by building a network that approximates the probability distribution of the dataset. Therefore, we need a loss function that can compare two distributions. One such loss function is the KL loss [27]: in this case, we assume that our real data is normally distributed, i.e.,, $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Then, the neural network aims to predict $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ by making a prediction $(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$, and then constructing a distribution $\hat{\mathbf{y}} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$. We then compute the KL loss:

$$V_{\text{KL}}(\mathbf{y}, \hat{\mathbf{y}}) = \text{KL}(f_{\mathbf{y}}||f_{\hat{\mathbf{y}}}), \tag{2.59}$$

where $\text{KL}(\cdot||\cdot)$ is the Kullback-Leibler divergence. The closer the two distributions, the lower the value of the loss.

2.2.1 Supervised Learning

Supervised learning utilizes the dataset to answer specific questions given the data [27]. An example of this might be predicting one's future income given past census data. The inputs to the model are called covariates (or features), and the model's output is called the response variable. Supervised learning requires labelled data; without such data there is no way to evaluate the model's accuracy. Supervised learning problems can be divided into two subcategories: classification and regression. In a classification problem, there is a finite number of values the response variable can take. For example, when training a classification model on the MNIST dataset (consisting of handwritten digits) [8], the model's covariates are the 784 pixels of each image. The model is then only able to predict the image being one of ten digits between 0 and 9.

In contrast, a regression problem has a continuous response variable. For example, we can build a model to predict a student's final grade (a decimal number between 0 and 100) leveraging a dataset that includes past students' grades and education history. A supervised learning model uses a metric called a loss function to assess the model's prediction accuracy. We then use numerical methods to adjust the model's parameters based on the loss. We discuss these concepts in more detail in Section 2.3.

2.2.2 Unsupervised Learning

Unsupervised learning aims to uncover trends in the data without posing any specific question about the data [30]. It is often used for exploratory purposes to observe if anything of significance can be uncovered from the dataset. One common unsupervised learning algorithm is clustering, which aims to separate the data into a given

number of groups. If the clusters are distinct (i.e., have little overlap), then one can build a supervised learning model with the data that can predict some response variable with a high degree of accuracy (e.g. see Figure 2.1).

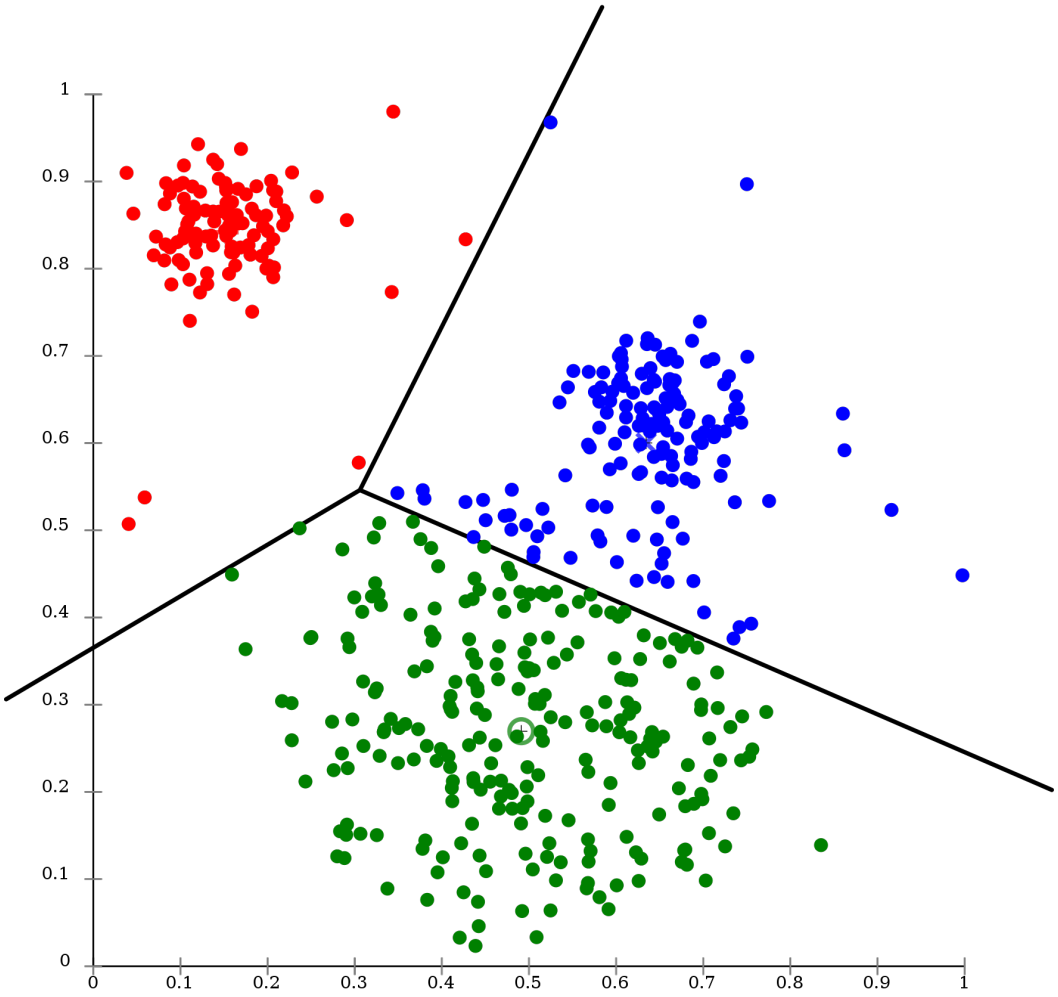


Figure 2.1: A clustering algorithm performed on a Gaussian dataset [7].

Unsupervised learning algorithms can also be applied to generative modelling. For instance, a generative adversarial network (GAN) aims to approximate the distribution of a real dataset to generate images that closely resemble the real data.

The choice of features is important for a model's performance. We want meaningful features that influence the outcome of the response variable. Redundant features can result in **overfitting**. This occurs when the model fits the noise in the training data. Consequently, the model fails to generalize to the testing data, and therefore performs poorly with samples it was not trained on. In contrast, too few features or lack of quality features can result in **underfitting**. This occurs when the lack of quality features results in too simple of a model. This increases the difficulty of training the model, and increases the probability of the model failing to generalize to the testing data.

2.3 Deep Learning and Neural Networks

We introduce the building blocks of neural networks and discuss the training process of a neural network. We follow the theory outlined in [12], [27] and [30].

Definition 29. *[22] An **activation function** is a continuous function* $\sigma : \mathcal{X} \subseteq \mathbb{R}^n \to \mathcal{Y} \subseteq \mathbb{R}^m$.

When we have a non-constant bounded activation function and a neural network with a sufficient number of hidden layers (to be defined next), a neural network can approximate any continuous function [22]. Furthermore, if we drop the continuity condition, a neural network can approximate any L^p function [22]. Common activation

functions include the sigmoid activation $S(x) = \frac{1}{1+e^{-x}}$, $\tanh(x)$, the ReLU (Rectified Linear Unit), and the LeakyReLU (Leaky Rectified Linear Unit) [42], respectively defined as

$$\textsc{ReLU}(x) = x\,\mathbf{1}_{\{x\geq 0\}}, \tag{2.60}$$

and

$$\textsc{LeakyReLU}_\alpha(x) = \alpha x\,\mathbf{1}_{\{x\geq 0\}} \tag{2.61}$$

respectively, where $\alpha \in \mathbb{R}_{>0}$.

Definition 30. *Let $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ be an input (column) vector, $\mathbf{w} \in \mathcal{X}$ be a weight (column) vector, and $b \in \mathbb{R}$ be a bias. A **neuron** is a function $f : \mathcal{X} \times \mathcal{X} \times \mathbb{R} \to \mathcal{Y} \subseteq \mathbb{R}^m$ defined by*

$$f(\mathbf{x}, \mathbf{w}, b) = \sigma(\mathbf{w}^\top \mathbf{x} + b) \tag{2.62}$$

where $\top$ denotes transposition and σ is an activation function.

A neural network organizes neurons into layers, with each layer connecting to the next. The first layer is called the input layer, and the last layer is called the output layer. The middle layers are referred to as hidden layers, as we are only able to observe the output layer given the input to the input layer; see Figure 2.2.

We now discuss how the neurons from one layer feeds forward to the next layer. We denote $N_i \in \mathbb{Z}_{>0}$ as the number of neurons in layer $\ell \in \{1, 2, ..., k\}$.

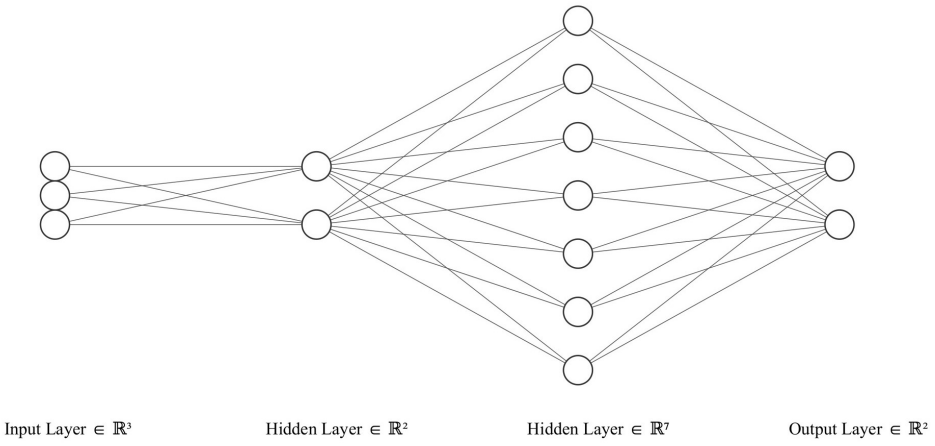


Figure 2.2: A neural network with three inputs, two outputs, and two hidden layers.

Definition 31. *The **weight matrix** of layer i is a matrix $\mathbf{w}^{(i)} \in \mathbb{R}^{N_{i-1} \times N_i}$. We denote $\mathbf{b}^{(i)} \in \mathbb{R}^{N_i}$ as the **bias vector** of layer i, where $i \in \{2, 3, ..., k\}$.*

Note that layer 1 does not have an associated weight matrix or bias vector, as we set the first layer as the input.

The entries of the weight matrix $\mathbf{w}^{(i)}$ are $\mathbf{w}^{(i)}_{j,k} = w^{(i)}_{j,k}$, which represents the weight of the connection between neuron j in layer $i-1$ to neuron k in layer i. The weights and biases are the parameters we update when we train a neural network.

Definition 32. [12] *Let $\mathcal{X} \subseteq \mathbb{R}^{N_1}$ be the input space, and $\mathcal{Y} \subseteq \mathbb{R}^{N_k}$ be the output space. A **k-layer neural network** is a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ defined by*

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma_k(\mathbf{w}^{(k)}\sigma_{k-1}(\mathbf{w}^{(\mathbf{k-1})}(\sigma_{k-2}(\mathbf{w}^{(k-2)}(\ldots\sigma_2(\mathbf{w}^{(2)}\mathbf{x} + \mathbf{b}^{(2)}))+ \tag{2.63}$$

$$\mathbf{b}^{(k-2)}) + \mathbf{b}^{(k-1)}) + \mathbf{b}^{(k)})) \tag{2.64}$$

where σ_i is the activation of layer i for $i = 1, ..., k$, applied identically to each input component, i.e.,, $\sigma(\mathbf{x}) = (\sigma_i(\mathbf{x}_1), ..., \sigma_i(\mathbf{x}_{N_i}))$. Furthermore,

$$\boldsymbol{\theta} = (\mathbf{w}^{(2)}, ..., \mathbf{w}^{(k)}, \mathbf{b}^{(2)}, ..., \mathbf{b}^{(k)})$$

denotes the collection of weights and biases of all k layers, called the ***parameters*** *of the neural network. Furthermore, for a loss function V and a dataset $\mathcal{D}$ with distribution $P_{\mathbf{D}}$, the* ***neural network problem*** *is*

$$\inf_{\boldsymbol{\theta} \in \Theta} \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim P_{\mathbf{D}}}[V(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))]. \tag{2.65}$$

The layers of neurons in (2.63) are referred to as **fully connected layers**. We can rewrite Definition 32 recursively: we denote the activation of layer 1 by $\mathbf{a}^{(1)} = \mathbf{x}$ (the input). From there, we can define subsequent activations recursively: for $i \in \{2, ..., k\}$, we have that

$$\mathbf{a}^{(i)} = \sigma_{i-1}(\mathbf{w}^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}). \tag{2.66}$$

We can write the argument of σ in vector form as follows:

$$\mathbf{z}^{(i)} = \begin{cases} \mathbf{x}, & i = 1 \\ \mathbf{w}^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)} & i \in \{2, ..., k\}. \end{cases} \tag{2.67}$$

This recursion is also referred as **forward propagation**. After obtaining the neural network's output through forward propagation, we compute the value of the loss function and compute the gradients of the loss function with respect to the weights

and biases. We do this through **backpropagation**, which works as follows [36]:

(i) After receiving the neural network's output for an input $\mathbf{x}$, compute the gradients of the loss function at the output layer with respect to its activations.

(ii) Starting at the second-last layer, compute the gradients of the loss function at each subsequent layer using the gradient computations from the successive layers.

(iii) Use the computations from the previous step to compute the gradients of the loss function with respect to each weight and bias.

We proceed by providing the equations for each step of backpropagation, following the treatment in [30].

Proposition 6. *(Step (i)) Let $f_{\boldsymbol{\theta}}$ be a k-layer neural network with loss function V. Then the gradient of V at the output layer with respect to the layer's inputs is given by*

$$\nabla_{\mathbf{z}^{(k)}} V = \nabla_{\mathbf{a}^{(k)}} V \odot \sigma'(\mathbf{z}^{(k)}), \tag{2.68}$$

where for vectors $\mathbf{s} = (s_1, ..., s_d)^\top$ and $\mathbf{t} = (t_1, ..., t_d)^\top \in \mathbb{R}^d$, $\odot$ is the Hadamard product given by $(\mathbf{s} \odot \mathbf{t})_i = s_i t_i$.

Proof. Assume that the output layer has $N_k \in \mathbb{Z}_{>0}$ neurons. Then for $j \in \{1, 2, ..., N_k\}$, we have that

$$\frac{\partial V}{\partial z_j^{(k)}} = \frac{\partial V}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial z_j^{(k)}} \tag{2.69}$$

$$= \frac{\partial V}{\partial a_j^{(k)}} \frac{\partial}{\partial z_j^{(k)}} \sigma(z_j^{(k)}) \tag{2.70}$$

$$= \frac{\partial V}{\partial a_j^{(k)}} \sigma'(z_j^{(k)}). \tag{2.71}$$

Putting this into vector form, we get that

$$\nabla_{\mathbf{z}^{(k)}} V = \nabla_{\mathbf{a}^{(k)}} V \odot \sigma'(\mathbf{z}^{(k)}).$$

□

Proposition 7. *(Step (ii)) Let $f_{\boldsymbol{\theta}}$ be a k-layer neural network with loss function V. Then for $\ell \in \{1, 2, ..., k-1\}$, we have that*

$$\nabla_{\mathbf{z}^{(\ell)}} V = ((\mathbf{w}^{(\ell+1)})^\top \nabla_{\mathbf{z}^{(\ell+1)}} V) \odot \sigma'(\mathbf{z}^{(\ell)}). \tag{2.72}$$

Proof. Let $f_{\boldsymbol{\theta}}$ be a k-layer neural network with loss function V. Let $\ell \in \{2, ..., k-1\}$ and assume that layer ℓ has N_ℓ layers. Then for $j \in \{1, 2, ..., N_\ell\}$ we have that

$$\frac{\partial V}{\partial z_j^{(\ell)}} = \sum_{k=1}^{N_\ell} \frac{\partial V}{\partial z_k^{(\ell+1)}} \frac{\partial z_k^{(\ell+1)}}{\partial z_j^{(\ell)}} \tag{2.73}$$

$$= \sum_{k=1}^{N_\ell} \frac{\partial V}{\partial z_k^{(\ell+1)}} \frac{\partial}{\partial z_j^{(\ell)}} \left(\sum_{i=1}^{N_{\ell+1}} w_{k,i}^{(\ell+1)} \sigma(z_i^{(\ell)}) + b_j^{(\ell+1)} \right) \tag{2.74}$$

$$\overset{(*)}{=} \sum_{k=1}^{N_\ell} \frac{\partial V}{\partial z_k^{(\ell+1)}} w_{k,j}^{(\ell+1)} \sigma'(z_j^{(\ell)}), \tag{2.75}$$

where $(*)$ holds since $\frac{\partial}{\partial z_j^{(\ell)}}(w_{k,i}^{(\ell+1)} \sigma(z_i^{(\ell)}) + b_i^{(\ell+1)})$ can only be non-zero for $i = j$.

Putting this in vector form, we have that

$$\nabla_{\mathbf{z}^{(\ell)}} V = ((\mathbf{w}^{(\ell+1)})^\top \nabla_{\mathbf{z}^{(\ell+1)}} V) \odot \sigma'(\mathbf{z}^{(\ell)}).$$

□

Now that we have the gradients of V with respect to the $\mathbf{z}^{(i)}$, we can compute the gradients of the V with respect to each weight and bias with the following proposition.

Proposition 8. *(Step (iii)) Let $f_{\boldsymbol{\theta}}$ be a k-layer neural network with loss function V. Let $\ell \in \{2, ..., k-1\}$ and assume that layer ℓ has N_ℓ layers. Then for $j \in \{1, 2, ..., N_{\ell-1}\}$ and $p \in \{1, 2, ..., N_\ell\}$ we have that*

$$\frac{\partial V}{\partial w_{j,p}^{(\ell)}} = a_p^{(\ell-1)} \frac{\partial V}{\partial z_j^{(\ell)}}, \tag{2.76}$$

and

$$\frac{\partial V}{\partial b_j^{(\ell)}} = \frac{\partial V}{\partial z_j^{(\ell)}}. \tag{2.77}$$

Note that we already obtain the quantities $\frac{\partial V}{\partial z_j^{(\ell)}}$ from the previous step.

Proof. We begin by showing that (2.76) holds. We have that

$$\frac{\partial V}{\partial w_{j,p}^{(\ell)}} = \frac{\partial V}{\partial z_j^{(\ell)}} \frac{\partial z_j^{(\ell)}}{\partial w_{j,p}^{(\ell)}} \tag{2.78}$$

$$= \frac{\partial V}{\partial z_j^{(\ell)}} \frac{\partial}{\partial w_{j,p}^{(\ell)}} \left(\sum_{i=1}^{N_\ell} w_{j,i}^{(\ell)} a_j^{(\ell-1)} + b_j^{(\ell)} \right) \tag{2.79}$$

$$\overset{(*)}{=} \frac{\partial V}{\partial z_j^{(\ell)}} a_p^{(\ell-1)} \tag{2.80}$$

$$= a_p^{(\ell-1)} \frac{\partial V}{\partial z_j^{(\ell)}} \tag{2.81}$$

where (*) holds since the only non-zero term of the sum's derivative is the p^{th} term. We now prove (2.77) in a similar fashion:

$$\frac{\partial V}{\partial b_j^{(\ell)}} = \frac{\partial V}{\partial z_j^{(\ell)}} \frac{\partial z_j^{(\ell)}}{\partial b_j^{(\ell)}} \tag{2.82}$$

$$= \frac{\partial V}{\partial z_j^{(\ell)}} \frac{\partial}{\partial b_j^{(\ell)}} \left(\sum_{i=1}^{N_\ell} w_{j,i}^{(\ell)} a_i^{(\ell-1)} + b_j^{(\ell)} \right) \tag{2.83}$$

$$= \frac{\partial V}{\partial z_j^{(\ell)}} \frac{\partial}{\partial b_j^{(\ell)}} (b_j^{(\ell)}) \tag{2.84}$$

$$= \frac{\partial V}{\partial z_j^{(\ell)}}. \tag{2.85}$$

□

2.3.1 Training a Neural Network

We aim to find the parameters $\boldsymbol{\theta}^*$ in (2.63) of the neural network that will minimize the expected value of the loss function. We use numerical methods to find a local minimum for (2.65). We discuss some gradient descent algorithms to obtain a critical point of a function.

2.3.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) finds the parameters $\boldsymbol{\theta}$ by updating them in the opposite direction of the gradient of the loss function V with respect to $\boldsymbol{\theta}$ [35]. If the difference between successive updates of $\boldsymbol{\theta}$ becomes arbitrarily small, it means we have reached a critical point of our loss function. We initialize a learning rate $\eta > 0$,

which determines the step size of the descent. Too large a learning rate can result in the SGD algorithm skipping over a critical point entirely, while too small a learning rate can result in a slower algorithm convergence or converging to a non-critical point. The SGD algorithm is outlined below.

Algorithm 1 Overview of Neural Network training using SGD

Require Loss function V, Learning rate $\eta > 0$, Number of epochs n_e, Training data $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$
Initialize Parameters $\boldsymbol{\theta}$, Neural network $f_{\boldsymbol{\theta}}$
for $i = 1$ to n_e **do**
  for $j = 1$ to n **do**
    Feed $\mathbf{x}_j$ through neural network $f_{\boldsymbol{\theta}}$ to retrieve $f_\theta(\mathbf{x}_j)$
    Compute the loss $V(\mathbf{y}_j, f_{\boldsymbol{\theta}}(\mathbf{x}_j))$
    Compute the gradient $\nabla_{\boldsymbol{\theta}} V(\mathbf{y}_j, f_{\boldsymbol{\theta}}(\mathbf{x}_j))$
    $\theta \leftarrow \theta - \eta \nabla_\theta V(\mathbf{y}_j, f_{\boldsymbol{\theta}}(\mathbf{x}_j))$
  end for
end for
$\theta^* \leftarrow \theta$
Return f_{θ^*}

One significant shortcoming of the SGD algorithm is its large computational cost; we update the weights and biases for each one of the n samples in the training set, and therefore compute the gradient of the loss n times. Many applications of neural networks utilize very large training sets, and the resulting additional gradient computations increases overall computation time. To remedy this, the **mini-batch stochastic gradient descent** was proposed. We divide our training data into batches of equal size, and update the parameters for each batch. Note that a batch size of 1 reduces to SGD. The Mini-Batch SGD algorithm is outlined below.

The Adam optimization algorithm [17] was introduced as a way to improve the

Algorithm 2 Overview of Neural Network training using MiniBatch-SGD.

Require Loss function V, Learning rate $\eta > 0$, Number of epochs n_e, Training data $\mathcal{T} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$, Batch size $B \in \{1, ..., n\}$
Initialize Parameters $\boldsymbol{\theta}$, Neural network $f_{\boldsymbol{\theta}}$
for $i = 1$ to n_e **do**
  for $j = 1$ to $\lfloor \frac{n}{B} \rfloor$ **do**
    Sample a mini-batch $(\mathbf{x}, \mathbf{y})$ of size B from $\mathcal{T}$, and yield a prediction $f_{\boldsymbol{\theta}}(\mathbf{x})$
    Compute the loss $V(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))$
    Compute the gradient $\nabla_{\boldsymbol{\theta}} V(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} V(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))$
  end for
end for
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
Return $f_{\boldsymbol{\theta}^*}$

SGD algorithm for higher-dimensional datasets. Unlike the SGD algorithm, which uses a constant learning rate to update $\boldsymbol{\theta}$, the Adam optimizer derives the learning rate for each timestep using the first and second moments of the objective function's gradients with exponential moving averages, which are controlled by hyperparameters β_1 and $\beta_2 \in [0, 1)$. We present the Adam optimization algorithm for a neural network below.

The bias-corrected terms $\hat{m}_t$ and $\hat{v}_t$ are necessary to avoid the first and second moment esimates m_t and v_t being biased towards zero in earlier iterations (since we initialize m_0 and v_0 as zero vectors). The recommended hyperparameter initialization for Algorithm 3 are $\alpha = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ [17].

Algorithm 3 Overview of the Adam optimization algorithm.

Require Step size $\alpha > 0$, exponential decay rates β_1, $\beta_2 \in [0, 1)$, Stopping parameter $\delta > 0$, Objective function $L(\boldsymbol{\theta})$, initial parameters (weights/biases) $\boldsymbol{\theta}_0 \in \mathbb{R}^d$
Initialize First and second moment vectors, $m_0 \leftarrow 0$, $v_0 \leftarrow 0$
Initialize Initial time step $t \leftarrow 0$
while $||\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}|| > \delta$ (i.e., $\boldsymbol{\theta}_t$ has not converged) **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{t-1})$ (gradient of L with respect to $\boldsymbol{\theta}_{t-1}$)
$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (estimation of the mean of the gradient of L)
$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t \odot g_t$ (estimation of the uncentered variance of the gradient of L)
$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ (bias correction of m_t)
$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ (bias correction of v_t)
$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$
end while
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}_t$
Return $\boldsymbol{\theta}^*$

2.3.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network mainly used for data that can be structured as a grid. They are mainly used in computer vision problems (image or video data), which can be visualized respectively as two- or three-dimensional grids, but also applies to time series data, which can be thought of as a one-dimensional grid of points. For image data, CNNs apply convolution operations to images using matrices called kernels to extract key visual features from the images. A CNN is trained in order to find the optimal kernels to minimize a given loss function. We begin by defining the convolution operation, then present the components of the convolutional layer that is the foundation for a CNN.

Kernels and Convolutions

Definition 33. *For $m, n, k \in \mathbb{Z}_{>0}$, an $\boldsymbol{m \times n \times k}$ **image** $\mathbf{I} = (\mathbf{I}_0, ..., \mathbf{I}_{k-1})$ is a collection of k superimposed $m \times n$ matrices in $[0,1]^{m \times n}$, where m and n is the length and width of the image respectively (in pixels), and k is the number of channels. We denote the **pixel** of image $\mathbf{I}$ at position (i, j) and channel p by $\mathbf{I}_p(i, j)$.*

We usually work with images with one channel ($k = 1$, black and white images), or three channels ($k = 3$, RGB images). For a fixed channel, a pixel value closer to 0 corresponds to a darker-coloured pixel, while a value closer to 1 corresponds to a lighter-colored pixel. For an RGB image, the three colour channels (red, green, and blue) are superimposed to create a colour.

Definition 34. [9] *A **kernel of size $\boldsymbol{m \times n \times k}$**, $\mathbf{K} = (\mathbf{K}_1, ..., \mathbf{K}_{k-1})$, is a collection of k $m \times n$ matrices in $\mathbb{R}^{m \times n}$. We denote the entry of matrix p of $\mathbf{K}$ at position (i, j) by $\mathbf{K}_p(i, j)$.*

Kernels are applied to images using a convolution operation, which we present next.

Definition 35. *Let I be an $m \times n \times k$ image and K be a kernel of size $M \times N \times k$. Then for a single channel $p \in \{0, ..., k-1\}$, the **convolution of $\boldsymbol{I}$ and $\boldsymbol{K}$ at channel $\boldsymbol{p}$** is denoted by $(\mathbf{I} * \mathbf{K})_p$, and given by the matrix*

$$[(\mathbf{I} * \mathbf{K})_p]_{i,j} = \sum_a \sum_b \mathbf{I}_p(i-a, j-b)\mathbf{K}_p(a, b) \tag{2.86}$$

$$= \sum_a \sum_b \mathbf{I}_p(a, b)\mathbf{K}_p(i-a, j-b), \tag{2.87}$$

where the sums are over all values of a and b such that they are well-defined.

The convolution in (2.86) is an operation between the image and the kernel rotated 180° [9]. In practice, the convolution operation is often replaced by the **cross-correlation** operation [12], which uses a non-rotated kernel, given by

$$[(\mathbf{I} * \mathbf{K})_p]_{i,j} = \sum_a \sum_b \mathbf{I}_p(i+a, j+b)\mathbf{K}_p(a,b). \tag{2.88}$$

If K_p is a symmetric matrix, then convolution and correlation are equivalent. Convolutions (and correlations) use the kernel to extract features from an image. These features vary by the choice of kernel. For example, kernels can be used to detect object edges, sharpen and blur an image. For a CNN, we aim to learn the kernels that will extract the most useful features from an image for the network to make an accurate prediction. The use of kernels in a CNN render them less computationally expensive than a standard neural network: a regular neural network would give each individual pixel its own weights and biases, while a kernel allows for its entries to be used across many pixels, significantly reducing the number of parameters. This notion is referred to as **parameter sharing** [12].

Padding

When convolving an image $\mathbf{I}$ with a kernel $\mathbf{K}$ at some channel p, we usually want every entry of $\mathbf{I}_p$ to interact with the center of $\mathbf{K}_p$, so that we do not lose any information about the image around its edges [9]. To accomplish this, we need to apply **padding** to the image. If I is an $m \times n \times k$ image and $\mathbf{K}$ is an $M \times N \times k$ kernel, then it is sufficient to pad each channel matrix $\mathbf{I}_p$ of $\mathbf{I}$ with at least $\frac{M-1}{2}$ rows of and $\frac{N-1}{2}$ of zeros around the edges of the matrix. This is referred to as **same convolution** [12], as the size of the resulting convolution matrix will match $\mathbf{I}_p$ to have size $m \times n$. [9].

One might also want every entry of $\mathbf{I}_p$ and $\mathbf{K}_p$ to interact so that the border pixels of an image have equal influence on the convolution operation, which requires $\mathbf{I}_p$ to be padded with M rows and N columns of zeros around the edges. This is referred to as **full convolution** [12], and the resultant convolution matrix will have size $(m+M-1)\times(n+N-1)$ [9]. For simplicity's sake, we assume that all convolutions are same convolutions.

Downsampling and Strides

To decrease the computational cost of the convolution operation, we can have the kernel skip over some groups of pixels, which is referred to as **downsampling** [9]. If the kernel skips over every s pixels, then we refer to $s \in \mathbb{Z}_{>0}$ as the **stride** [12]. We can also define a two-dimensional stride $(s_1, s_2) \in \mathbb{Z}_{>0} \times \mathbb{Z}_{>0}$ to determine how many of the image's pixels are sampled by the kernel both horizontally and vertically. The significant tradeoff to downsampling is a lower quality feature extraction, as the kernel does not sample every region of pixels.

Definition 36. [12] *Let I be an $m \times n \times k$ image, K be a kernel of size $M \times N \times k$ and $\mathbf{s} = (s_1, s_2) \in \mathbb{Z}_{>0} \times \mathbb{Z}_{>0}$. Then for a single channel $p \in \{0, ..., k-1\}$, the **downsampled convolution of I and K at channel p with strides $\mathbf{s}$** is denoted by $(I *_{\mathbf{s}} K)_p$, and given by the matrix*

$$[(\mathbf{I} *_{\mathbf{s}} \mathbf{K})_p]_{i,j} = \sum_a \sum_b \mathbf{I}_p(s_1 i - a, s_2 j - b)\mathbf{K}_p(a, b). \tag{2.89}$$

Pooling

After a convolution is performed on an image, the linear activations produced by the operation undergoes another activation function (usually a ReLU or LeakyReLU activation). From there, a CNN uses a pooling function $\rho : \mathbb{R}^n \to \mathbb{R}$ to further transform the image. A pooling function replaces the output produced by a layer with some summary statistic of neighbouring outputs. A common pooling function is the max pooling function, which obtains the maximum value amongst a region of pixels. Other pooling functions include the L^p distance between neighbouring pixels, min pooling, or a weighted average amongst a region of pixels [12]. We now present the convolutional neural network.

Definition 37. *Let $\mathcal{X} \subseteq \mathbb{R}^{m\times n\times k}$ be the input space and $\mathcal{Y} \subseteq \mathbb{R}^{N_p}$ be the output space. Let $\mathbf{s}_2, ..., \mathbf{s}_p \in \mathbb{Z}_{>0} \times \mathbb{Z}_{>0}$ be fixed, where $\mathbf{s}_i = (s_i^{(1)}, s_i^{(2)})$. Let $\rho : \mathbb{R}^n \to \mathbb{R}$ be a pooling function. A **p-layer convolutional neural network** is a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ given by*

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = r_p(\mathbf{K}^{(p)} *_{\mathbf{s}_p} r_{p-1}(\mathbf{K}^{(p-1)}(\dots r_2(\mathbf{K}^{(2)} *_{\mathbf{s}_2} \mathbf{x} + \mathbf{b}^{(2)})) + \mathbf{b}^{(p-2)}) + \mathbf{b}^{(p-1)}) + \mathbf{b}^{(p)})), \tag{2.90}$$

where $\sigma_i : \mathbb{R}^{N_i} \to \mathbb{R}$ is an activation function $r_i = \rho \circ \sigma_i$,

$$\boldsymbol{\theta} = (\mathbf{K}^{(2)}, ..., \mathbf{K}^{(p)}, ..., \mathbf{b}^{(2)}, ..., \mathbf{b}^{(p)})$$

denotes the parameters, $\mathbf{K}^{(i)}$ is a kernel, and $\mathbf{b}^{(i)}$ is a bias.

The experiments conducted in this thesis are based on DCGAN [33], and we therefore do not use pooling, (i.e., we can assume that ρ_i is the identity map).

Chapter 3

Generative Adversarial Networks

3.1 Generative Adversarial Networks

GANs were first devised by Goodfellow *et al.* in [10]. A GAN is comprised of two neural networks, a generator and a discriminator. The generator aims to produce data that mimics data from a dataset $\mathcal{D}$. The generator takes in random noise to output a generated data point. That data point is then fed into the discriminator, which outputs a score or label (varying from 0 to 1) for whether the data point belongs to the dataset. Note than an extreme score of 1 means that the discriminator is certain about a data point belonging to the real dataset (while a score of 0 means that the discriminator is certain that the data point is generated). The discriminator is also given samples from the real dataset. The generator aims to fool the discriminator by having the discriminator label points from the real dataset as generated. This chapter focuses on the application of GANs to image data, however, GANs can also be used to generate synthetic tabular or text data. We next present the notion of GANs.

Let $(\mathcal{X}, \mathcal{B}(\mathcal{X}), \mu)$ be the measure space of $n \times n \times m$ images (where $m = 1$ for black and white images and $m = 3$ for RGB images), and let $(\mathcal{Z}, \mathcal{B}(\mathcal{Z}), \mu)$ be a measure

space such that $\mathcal{Z} \subseteq \mathbb{R}^d$. The generator's noise input is sampled from a multivariate Gaussian distribution $P_{\mathbf{z}} : \mathcal{Z} \to [0, 1]$. We denote the probability distribution of the real data by $P_{\mathbf{x}} : \mathcal{X} \to [0, 1]$ and the probability distribution of the generated data by $P_{\mathbf{g}} : \mathcal{X} \to [0, 1]$. We also set $P_{\mathbf{x}}$ and $P_{\mathbf{g}}$ as the densities corresponding to $P_{\mathbf{x}}$ and $P_{\mathbf{g}}$. We follow the treatment of the original GAN paper [11] unless otherwise specified.

Definition 38. *The **generator** network is a neural network $G_{\boldsymbol{\theta}} : \mathcal{Z} \to \mathcal{X}$*

$$\mathbf{z} \mapsto G_{\boldsymbol{\theta}}(\mathbf{z}), \tag{3.1}$$

where $\boldsymbol{\theta}$ denotes the generator's parameters.

Definition 39. *The **discriminator** network is a neural network $D_{\boldsymbol{\theta}} : \mathcal{X} \to [0, 1]$*

$$\mathbf{x} \mapsto D_{\boldsymbol{\theta}}(\mathbf{x}), \tag{3.2}$$

where $\boldsymbol{\theta}$ denotes the discriminator's parameters.

For simplicity's sake, we denote the generator and discriminator by $G := G_{\boldsymbol{\theta}}$ and $D := D_{\boldsymbol{\theta}}$ respectively.

Definition 40. *Let G be a generator and D be a discriminator. The **GAN's loss function** is denoted by $V(D, G)$, and is given by*

$$V(D, G) = -\mathcal{H}(P_{\mathbf{x}}; D) - \mathcal{H}(P_{\mathbf{z}}; 1 - D \circ G), \tag{3.3}$$

where $\mathcal{H}(\cdot\,;\,\cdot)$ denotes the (differential) cross-entropy functional introduced after Definition 3 in the previous chapter. The ***GAN problem*** *is the minimax problem*

$$\inf_G \sup_D V(D,G). \tag{3.4}$$

We can rewrite the loss function of (3.3) in terms of expectations:

$$V(D,G) = -\mathcal{H}(P_{\mathbf{x}}; D) - \mathcal{H}(P_{\mathbf{z}}; 1 - D \circ G) \tag{3.5}$$

$$= \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}[\log D(\mathbf{A})] + \mathbb{E}_{\mathbf{Z}\sim P_{\mathbf{z}}}[\log(1 - D(G(\mathbf{Z})))] \tag{3.6}$$

$$= \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}[\log D(\mathbf{A})] + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}[\log(1 - D(\mathbf{B}))]. \tag{3.7}$$

We now present the solution to the GAN problem stated in (3.4), with its proof adopted from [4].

Proposition 9. *Consider the maximization over D in* (3.3)*. The discriminator D^* that maximizes $V(D,G)$ is given by*

$$D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}} + P_{\mathbf{g}}}. \tag{3.8}$$

Furthermore, when $D = D^*$, joint optimization of V reduces to the minimization of the Jensen-Shannon divergence:

$$V(D^*, G) = 2\mathrm{JSD}(P_{\mathbf{x}} || P_{\mathbf{g}}) - 2\log 2 \geq 2\log 2, \tag{3.9}$$

where the minimum is obtained iff $P_{\mathbf{x}} = P_{\mathbf{g}}$ (a.e.).

Proof. We first show that $V(D,G)$ is maximized over D when $D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$. We have

that

$$\begin{aligned}
\frac{\partial V}{\partial D}(D,G) &= \lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} P_{\mathbf{x}} \frac{\log(D+\epsilon) - \log(D)}{\epsilon}\, d\mu + \int_{\mathcal{X}} P_{\mathbf{g}} \frac{\log(1-D-\epsilon) - \log(1-D)}{\epsilon}\, d\mu \\
&= \lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} P_{\mathbf{x}} \frac{\log\left(\frac{D+\epsilon}{D}\right)}{\epsilon}\, d\mu + \int_{\mathcal{X}} P_{\mathbf{g}} \frac{\log\left(\frac{1-D-\epsilon}{1-D}\right)}{\epsilon}\, d\mu \\
&= \lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} P_{\mathbf{x}} \frac{\log\left(1+\frac{\epsilon}{D}\right)}{\epsilon}\, d\mu - \int_{\mathcal{X}} P_{\mathbf{g}} \frac{\log\left(\frac{1-D}{1-D-\epsilon}\right)}{\epsilon}\, d\mu \\
&= \lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} \frac{P_{\mathbf{x}}}{D} \frac{\log\left(1+\frac{\epsilon}{D}\right)}{\epsilon/D}\, d\mu - \int_{\mathcal{X}} \frac{P_{\mathbf{g}}}{1-D} \frac{\log\left(\frac{1-D}{1-D-\epsilon}\right)}{\epsilon/(1-D)}\, d\mu
\end{aligned}$$

We now show that the last line is bounded from above and below by $\int_{\mathcal{X}} \frac{P_{\mathbf{x}}}{D} - \frac{P_{\mathbf{g}}}{1-D}\, d\mu$. We first show the upper bound, using the bound $\log(1+x) \leq x$, $x > 0$, for the first term, and the bound $\log(y) \geq 1 - \frac{1}{y}$, $y > 0$, for the second term. This yields

$$\lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} \frac{P_{\mathbf{x}}}{D} \frac{\log\left(1+\frac{\epsilon}{D}\right)}{\epsilon/D}\, d\mu - \int_{\mathcal{X}} \frac{P_{\mathbf{g}}}{1-D} \frac{\log\left(\frac{1-D}{1-D-\epsilon}\right)}{\epsilon/(1-D)}\, d\mu \leq \int_{\mathcal{X}} \frac{P_{\mathbf{x}}}{D} - \frac{P_{\mathbf{g}}}{1-D}\, d\mu$$

We apply the same technique to obtain the lower bound:

$$\begin{aligned}
\frac{\partial V}{\partial D}(D,G) &= \lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} P_{\mathbf{x}} \frac{\log(D+\epsilon) - \log(D)}{\epsilon}\, d\mu + \int_{\mathcal{X}} P_{\mathbf{g}} \frac{\log(1-D-\epsilon) - \log(1-D)}{\epsilon}\, d\mu \\
&= \lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} P_{\mathbf{x}} \frac{\log\left(1+\frac{\epsilon}{D}\right)}{\epsilon}\, d\mu - \int_{\mathcal{X}} P_{\mathbf{g}} \frac{\log\left(\frac{1-D}{1-D-\epsilon}\right)}{\epsilon}\, d\mu \\
&= \lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} \frac{P_{\mathbf{x}}}{D+\epsilon} \frac{\log\left(1+\frac{\epsilon}{D}\right)}{\epsilon/(D+\epsilon)}\, d\mu - \int_{\mathcal{X}} \frac{P_{\mathbf{g}}}{1-D-\epsilon} \frac{\log\left(\frac{1-D}{1-D-\epsilon}\right)}{\epsilon/(1-D-\epsilon)}\, d\mu \\
&\geq \lim_{\epsilon \downarrow 0} \int_{\mathcal{X}} \frac{P_{\mathbf{x}}}{D+\epsilon}\, d\mu - \int_{\mathcal{X}} \frac{P_{\mathbf{g}}}{1-D-\epsilon}\, d\mu \\
&= \int_{\mathcal{X}} \frac{P_{\mathbf{x}}}{D} - \frac{P_{\mathbf{g}}}{1-D}\, d\mu
\end{aligned}$$

where the order of the limit and integral can be swapped by the monotone convergence

theorem. Therefore, we have that

$$\frac{\partial V}{\partial D}(D,G) = \int_{\mathcal{X}} \frac{P_{\mathbf{x}}}{D} - \frac{P_{\mathbf{g}}}{1-D}\, d\mu.$$

If $\frac{P_{\mathbf{x}}}{D^*} - \frac{P_{\mathbf{g}}}{1-D^*} = 0$ (a.e), then $\frac{\partial V}{\partial D}(D,G)|_{D=D^*} = 0$. Now solving for D^*, we obtain

$$D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}} + P_{\mathbf{g}}}.$$

Note that D^* yields a maximum for $V(D,G)$ as it can be shown (using a similar procedure) that $\frac{\partial^2 V}{\partial D^2}(D,G) < 0$. Finally, substituting $D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$ into (3.3), we have

$$\begin{aligned}
V(D^*,G) &= \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}[\log D^*(\mathbf{A})] + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}[\log(1-D^*(\mathbf{B}))] \\
&= \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}\left[\log \frac{P_{\mathbf{x}}(\mathbf{A})}{P_{\mathbf{x}}(\mathbf{A}) + P_{\mathbf{g}}(\mathbf{A})}\right] + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}\left[\log \frac{P_{\mathbf{g}}(\mathbf{B})}{P_{\mathbf{x}}(\mathbf{B}) + P_{\mathbf{g}}(\mathbf{B})}\right] \\
&= \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}\left[\log \frac{2P_{\mathbf{x}}(\mathbf{A})}{P_{\mathbf{x}}(\mathbf{A}) + P_{\mathbf{g}}(\mathbf{A})} - \log 2\right] \\
&\qquad + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}\left[\log \frac{2P_{\mathbf{g}}(\mathbf{B})}{P_{\mathbf{x}}(\mathbf{B}) + P_{\mathbf{g}}(\mathbf{B})} - \log 2\right] \\
&= \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}\left[\log\left(\frac{P_{\mathbf{x}}(\mathbf{A})}{(P_{\mathbf{x}}(\mathbf{A}) + P_{\mathbf{g}}(\mathbf{A}))/2}\right)\right] \\
&\quad + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}\left[\log\left(\frac{P_{\mathbf{g}}(\mathbf{B})}{(P_{\mathbf{x}}(\mathbf{B}) + P_{\mathbf{g}}(\mathbf{B}))/2}\right)\right] - 2\log 2 \\
&= 2\mathrm{JSD}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2\log 2.
\end{aligned}$$

Since $\mathrm{JSD}(P_{\mathbf{x}}||P_{\mathbf{g}}) \geq 0$, we have that $V(D^*,G) \geq 2\log 2$, with equality iff $P_{\mathbf{x}} = P_{\mathbf{g}}$ (a.e.). □

From now on, we refer to the GAN of (3.4) by the **VanillaGAN**. The Deep Convolutional GAN (DCGAN) uses the same loss function as the VanillaGAN, and

uses convolutional layers in the discriminator and generator architectures to optimize performance for computer vision problems [33]. A GAN can be evaluated using the Fréchet Inception Distance (FID) score [16], which we present next.

Definition 41. [16] *Assume that $P_{\mathbf{x}}$ and $P_{\mathbf{g}}$ have mean μ_x and covariance matrix Σ_x, and μ_g and Σ_g, respectively. The **Fréchet Inception Distance Score** (FID) is given by*

$$\text{FID} = ||\mu_x - \mu_g||_2^2 + \text{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{\frac{1}{2}}). \tag{3.10}$$

The FID score compares the real and generated distributions' means and covariances. The lower the FID score, the closer the generated images resemble images from the real dataset. When computing the FID score, we usually assume that $P_{\mathbf{x}}$ and $P_{\mathbf{g}}$ are multivariate Gaussian distributions. The FID score is an implementation of the Wasserstein-2 distance between the two multivariate Gaussian distributions and is typically computed using at least 10 000 samples [16].

3.2 RényiGANs

We now present the RényiGAN [5], which replaces the Shannon cross-entropies in the generator loss function with Rényi cross-entropies, allowing for an additional degree of freedom by introducing an α parameter that arises from the Rényi cross-entropy.

Definition 42. [5] *Let $\alpha > 0$ such that $\alpha \neq 1$. The **RényiGAN** is characterized by $(V_D, V_{G,\alpha})$, where $V_D : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}$ is the discriminator loss function and $V_{G,\alpha}$:*

$\mathcal{X} \times \mathcal{Z} \to \mathbb{R}$ *is the generator loss function such that*

$$V_D(D, G) = -\mathcal{H}(P_{\mathbf{x}}; D) - \mathcal{H}(P_{\mathbf{z}}; 1 - D \circ G) \tag{3.11}$$

and

$$V_{G,\alpha}(D, G) = -\mathcal{H}_\alpha(P_{\mathbf{x}}; D) - \mathcal{H}_\alpha(P_{\mathbf{z}}; 1 - D \circ G), \tag{3.12}$$

where $\mathcal{H}_\alpha(\cdot\,\cdot)$ *denotes the (differential) Rényi cross-entropy functional. The* ***RényiGAN problem*** *is the joint optimization problem*

$$\sup_D V_D(D, G) \tag{3.13}$$

$$\inf_G V_{G,\alpha}(D, G). \tag{3.14}$$

We now present the solution to the RényiGAN optimization problem of (3.13).

Proposition 10. [5] *Consider the maximization of* D *in* (3.13)*. The discriminator* D^* *that maximizes* $V_D(D, G)$ *is given by*

$$D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}} + P_{\mathbf{g}}}. \tag{3.15}$$

Furthermore, when $D = D^*$*, minimization of* $V_{G,\alpha}$ *over* G *reduces to minimizing a Jensen-Rényi divergence:*

$$V_{G,\alpha}(D^*, G) = 2\mathrm{JR}_\alpha(P_{\mathbf{x}} || P_{\mathbf{g}}) - 2\log 2 \geq -2\log 2, \tag{3.16}$$

where the minimum is obtained iff $P_{\mathbf{x}} = P_{\mathbf{g}}$ *(a.e.).*

Note that the RényiGAN and VanillaGAN share the same equilibrium point as $\alpha \to 1$, since it can be shown that $V_{G,a}(D^*, G)$ converges to $V(D^*, G)$.

Proof. The proof to show that V_D is maximized at $D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$ follows that of Proposition 9, as the VanillaGAN and RényiGAN utilize the same loss function for the discriminator. We show that when we fix $D = D^*$, minimizing $V_{G,\alpha}(D^*, G)$ reduces to minimizing a Jensen-Rényi divergence. We have that

$$
\begin{aligned}
V_{\alpha,G}(D^*, G) &= -\mathcal{H}_\alpha(P_{\mathbf{x}}; D^*) - \mathcal{H}_\alpha(P_{\mathbf{z}}; 1 - D^* \circ G) \\
&= -\mathcal{H}_\alpha(P_{\mathbf{x}}; D^*) - \mathcal{H}_\alpha(P_{\mathbf{g}}; 1 - D^*) \\
&= \frac{1}{\alpha - 1} \log \left(\mathbb{E}_{\mathbf{A} \sim P_{\mathbf{x}}} \left[\left(\frac{P_{\mathbf{x}}(\mathbf{A})}{P_{\mathbf{x}}(\mathbf{A}) + P_{\mathbf{g}}(\mathbf{A})} \right)^{\alpha - 1} \right] \right) \\
&\quad + \frac{1}{\alpha - 1} \log \left(\mathbb{E}_{\mathbf{B} \sim P_{\mathbf{g}}} \left[\left(\frac{P_{\mathbf{g}}(\mathbf{B})}{P_{\mathbf{x}}(\mathbf{B}) + P_{\mathbf{g}}(\mathbf{B})} \right)^{\alpha - 1} \right] \right) \\
&= \frac{1}{\alpha - 1} \log \left(\mathbb{E}_{\mathbf{A} \sim P_{\mathbf{x}}} \left[\left(\frac{2P_{\mathbf{x}}(\mathbf{A})}{P_{\mathbf{x}}(A) + P_{\mathbf{g}}(\mathbf{A})} \right)^{\alpha - 1} - 2 \right] \right) \\
&\quad + \frac{1}{\alpha - 1} \log \left(\mathbb{E}_{\mathbf{B} \sim P_{\mathbf{g}}} \left[\left(\frac{2P_{\mathbf{g}}(\mathbf{B})}{P_{\mathbf{x}}(\mathbf{B}) + P_{\mathbf{g}}(\mathbf{B})} \right)^{\alpha - 1} - 2 \right] \right) \\
&= 2 \left(\frac{1}{2} D_\alpha \left(P_{\mathbf{x}} \middle|\middle| \frac{P_{\mathbf{x}} + P_{\mathbf{g}}}{2} \right) + \frac{1}{2} D_\alpha \left(P_{\mathbf{g}} \middle|\middle| \frac{P_{\mathbf{x}} + P_{\mathbf{g}}}{2} \right) \right) - 2 \log 2 \\
&= 2\mathrm{JR}_\alpha(P_{\mathbf{x}} || P_{\mathbf{g}}) - 2 \log 2.
\end{aligned}
$$

Since $\mathrm{JR}_\alpha(P_{\mathbf{x}} || P_{\mathbf{g}}) \geq 0$, we have that $V_{G,\alpha}(D^*, G) \geq 2 \log 2$, with equality iff $P_{\mathbf{x}} = P_{\mathbf{g}}$ (a.e.). □

3.3 InfoGAN

In the previous chapter, we discussed the "classic" GAN architecture: the generator receives a noise vector $\mathbf{z} \sim p_{\mathbf{z}}$ as input, and outputs data $\mathbf{x} \in \mathcal{X}$. The data $\mathbf{x}$ is then given to the discriminator, which assigns the data a value between 0 and 1. The closer the output to 1, the more confidence the discriminator has that the data belongs to the real dataset as opposed to the generated set.

While the VanillaGAN architecture can yield quality results, the model lacks explainability. Specifically, we do not know how the generator uses the noise input in the model to generate the output. To remedy this issue, InfoGAN was proposed. In this section, we discuss the formulation of the InfoGAN, following the treatment of the original work [6].

Definition 43. *Suppose that the input to the generator is the tuple* $(\mathbf{z}, \mathbf{c}) \in \mathcal{Z} \times \mathcal{C}$. *We call* $\mathbf{z}$ *the* ***incompressible noise*** *and* $\mathbf{c}$ *the* ***latent codes****.*

We assume that the components of $\mathbf{c} = (c_1, ..., c_L)$ are independent of one another, i.e.,

$$p_{\mathbf{c}}(c_1, ..., c_L) = \prod_{i=1}^{L} p_{\mathbf{c}_i}(c_i). \tag{3.17}$$

The incompressible noise functions the same way as the generator's noise input in VanillaGAN; the noise is unstructured and used as a "black box input" for the generator. The latent codes target features of the real data distribution $p_{\mathbf{x}}$. With VanillaGAN, our generated output $\mathbf{x} = G(\mathbf{z}, \mathbf{c})$ is independent of the latent codes, since we treat the latent codes as unstructured Gaussian noise. To target features of the real data, we want to increase the mutual information between the latent codes

and the generated images, i.e., we want $\mathrm{I}(\mathbf{c}; G(\mathbf{z}, \mathbf{c}))$ to be high.

Definition 44. *The objective of the InfoGAN problem is the following optimization:*

$$\inf_G \sup_D -\mathcal{H}(p_{\mathbf{X}}; D) - \mathcal{H}(p_{\mathbf{z}}; 1 - D \circ G) - \lambda \mathrm{I}(\mathbf{c}; G(\mathbf{z}, \mathbf{c})). \tag{3.18}$$

The third term of the objective function contains a regularization coefficient $\lambda \geq 0$. A high value of λ penalizes the network if the latent codes are not sufficiently used to target features of the real data distribution. In contrast, lower values of λ decrease the influence of the mutual information term on the loss, with $\lambda = 0$ being analagous to the VanillaGAN's loss function.

The mutual information term is difficult to compute, as it requires the calculation of the posterior $P(\mathbf{c} = c \,|\, G(\mathbf{z}, \mathbf{c}) = \mathbf{x})$. Therefore, we derive a lower bound on the mutual information term by using an auxiliary distribution to approximate the posterior [6]. Utilizing the non-negativity of the Kullback-Leibler divergence, we have that

$$\begin{aligned}
\mathrm{I}(\mathbf{c}; G(\mathbf{z}, \mathbf{c})) &= H(\mathbf{c}) - H(\mathbf{c}|G(\mathbf{z}, \mathbf{c})) && (3.19)\\
&= H(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{g}}}[\mathbb{E}_{\mathbf{c}\sim p_{\mathbf{c}|\mathbf{x}}}[\log p(\mathbf{c}|\mathbf{x})]] && (3.20)\\
&= H(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{g}}}\left[\mathbb{E}_{\mathbf{c}\sim p_{\mathbf{c}|\mathbf{x}}}\left[\log \frac{p(\mathbf{c}|\mathbf{x})}{Q(\mathbf{c}|\mathbf{x})}\right] + \mathbb{E}_{\mathbf{c}\sim p_{\mathbf{c}|\mathbf{x}}}[\log Q(\mathbf{c}|\mathbf{x})]\right] && (3.21)\\
&= H(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{g}}}\left[\mathrm{KL}(p(\cdot||\mathbf{x})||Q(\cdot||\mathbf{x})) + \mathbb{E}_{\mathbf{c}\sim p_{\mathbf{c}|\mathbf{x}}}[\log Q(\mathbf{c}|\mathbf{x})]\right] && (3.22)\\
&\geq H(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{g}}}[\mathbb{E}_{\mathbf{c}\sim p_{\mathbf{c}|\mathbf{x}}}[\log Q(\mathbf{c}|\mathbf{x})]] && (3.23)\\
&= H(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{g}}, \mathbf{c}\sim p_{\mathbf{c}}}[\log Q(\mathbf{c}|\mathbf{x})] && (3.24)\\
&:= \mathrm{LI}(G, Q), && (3.25)
\end{aligned}$$

where the inequality holds by the non-negativity of the KL divergence, and Q is an auxiliary (conditional) distribution. All random variables in the derivation above are assumed to be discrete (if they are continuous then the entropies are replaced with differential entropies).

In practice, the auxiliary distribution is modelled as a neural network, called the **recognition network**, which takes in a generated image $G(\mathbf{z}, \mathbf{c})$ as input, and outputs the probabilities of each latent code being used to generate that image. In training, the recognition network encourages the generator to produce meaningful differences its output based on the latent code input.

Definition 45. *The InfoGAN problem is given by the following joint optimization:*

$$\inf_{G,Q} \sup_{D} \mathcal{H}(p_{\mathbf{x}}; D) - \mathcal{H}(p_{\mathbf{z}}; 1 - D \circ G) - \lambda(\mathcal{H}(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{g}}, \mathbf{c}\sim p_{\mathbf{c}}}[\log Q(\mathbf{c}|\mathbf{x})]), \quad (3.26)$$

where $\lambda \geq 0$ is the regularization coefficient.

3.3.1 Experiments

In the experiments, we aimed to reproduce the results of the original InfoGAN work. We also aimed to quantify the performance of InfoGAN, as the original work contained only qualitative results.

Experimental Setup

The MNIST dataset [8] was used to test the InfoGAN architecture. The generated images were evaluated using the FID score. The results were also examined qualitatively, as we aimed to observe the output of the generator as the latent codes were varied. Ideally, the Intra-FID score [43] would have also been utilized to score the

images by evaluating the FID score for each unique value of the latent codes. This was not performed as the algorithm to compute the Intra-FID scores has time complexity $\mathcal{O}(n^3)$, which is relative slow. Furthermore, as per [16], we require a minimum of 10 000 samples to reliably compute an FID score to not underestimate the score. Since there are only 7000 samples of each digit in the MNIST datset, the Intra-FID score is not a reliable metric. We use a batch size of 128 and an incompressible noise dimension of 62. We trained InfoGAN on 60 000 images for 800 epochs, for a total of 48 million images.

InfoGAN was trained using two setups: discrete and continuous. In the discrete setup, we use a single categorical latent code $\mathbf{c_1} \sim \text{Categorical}(k = 10)$. Each code $c \in \{0, 1, ..., 9\}$ is evenly distributed. For the MNIST dataset, we aim to have each code correspond to a unique digit between 0 and 9. It is important to note that codes and digits do not align, as InfoGAN does not use the latent code to recognize digits. Rather, it uses the categorical latent code to extract features from the dataset that uniquely identify each digit in order to maximize the mutual information between the code and the generated images.

In the continuous setup, we retain $\mathbf{c}_1$ with the categorical code from the discrete setup, and introduce two continuous codes, $\mathbf{c_2} \sim \text{Uniform}(-1, 1)$, and $\mathbf{c_3} \sim \text{Uniform}(-1, 1)$. We train the continuous codes separately from the discrete code to separate the discrete Shannon entropy for $\mathbf{c_1}$ from the differential Shannon entropy

for $\mathbf{c_2}$ and $\mathbf{c_3}$. Therefore, we aim to solve the following optimization:

$$\inf_{G,Q} \sup_{D} \mathcal{H}(p_{\mathbf{x}}; D) - \mathcal{H}(p_{\mathbf{z}}; 1 - D \circ G) - \lambda_d(H(\mathbf{c_1}) + \mathbb{E}_{\mathbf{x} \sim p_{\mathbf{g}}, \mathbf{c_1} \sim p_{\mathbf{c_1}}}[\log Q(\mathbf{c_1}|\mathbf{x})])$$
$$- \lambda_c(h(\mathbf{c_2}, \mathbf{c_3}) + \mathbb{E}_{\mathbf{x} \sim p_{\mathbf{g}}, \mathbf{c_2}, \mathbf{c_3} \sim p_{\mathbf{c_2}, \mathbf{c_3}}}[\log Q(\mathbf{c_2}, \mathbf{c_3}|\mathbf{x})]) \tag{3.27}$$

where λ_d and λ_c are the regularization coefficients for the discrete and continuous latent codes respectively. As per the original work, we set $\lambda_d = 1.0$ and $\lambda_c = 0.1$. The discrete code $\mathbf{c_1}$ is modelled as a ten-dimensional vector $(p_0, p_1, ..., p_9) \in [0, 1]^{10}$. We generate $\mathbf{c_1}$ by first randomly sampling an integer K between 0 and 9 inclusive from a discrete uniform distribution. We then generate $\mathbf{c_1} = (p_0, p_1, ..., p_9)$ using the formula $p_i = \mathbf{1}_{\{K=i\}}$. For $i = 0, 1..., 9$, p_i represents the probability of the generated image having discrete latent code i. If $p_i = 1$ for some $i = 0, 1, ..., 9$, then $\mathbf{c}_1 = i$ with probability one. We generate $\mathbf{c_2}$ and $\mathbf{c_3}$ by randomly sampling from a Uniform$(-1, 1)$ distribution.

We input the incompressible noise $\mathbf{z} \sim \mathcal{N}_{62}(\mathbf{0}, \mathbf{I})$ (sampled from a 62-dimensional Gaussian distribution) and latent codes to the generator to output an image. The image is then fed into the recognition network. In the discrete setup, we use a softmax activation to output a vector $(\hat{p}_0, \hat{p}_1, ..., \hat{p}_9) \in [0, 1]^{10}$, where for $i = 0, 1, ..., 9$, $\hat{p}_i$ is the probability that the generated image came from discrete latent code i. For continuous codes $\mathbf{c_2}$ and $\mathbf{c_3}$, the recognition network outputs the tuple (μ_k, σ_k^2) for $k = 2, 3$, where σ_k^2 is generated using an exponential transformation of the previous batch normalization layer to ensure positivity. We then use (μ_k, σ_k^2) to construct a random variable $T_k \sim \mathcal{N}(\mu_k, \sigma_k^2)$, to generate the probability $\hat{c}_k = P(T_k \leq \mathbf{c}_k)$, which is used to calculate the differential entropy term of the loss function.

The generator, discriminator, and recognition network were all trained using Adam optimizers [16]. For the generator's and discriminator's optimizers, we set $\alpha = 2 \times 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$. For the recognition network's optimizer, we set $\alpha = 10^{-3}$, $\beta_1 = 0.5$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$. Four experiments were conducted: we trained the InfoGAN in the discrete setup with and without a gradient penalty (GP) using the simplified gradient penalty. The gradient penalty aids in a neural network's stability by ensuring its gradients to have unit norm [14]. The same was done with the continuous setup. A coefficient of 5.0 was used for the gradient penalty. For each experiment, five trials were conducted, each with a different random seed. The random seeds used for the experiments were 123, 1600, 60677, 15859, and 79878. The experiments were run on one 6130 2.1 GHz 1xV100 GPU, 12 CPUs, and 175 GB of memory. The neural network architectures and algorithms used for InfoGAN training are highlighted in [6].

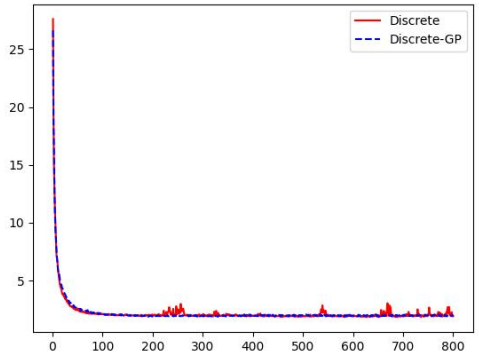
3.3.2 Results

The results of the reproduction are summarized in Table 3.1. For each experiment, we report the average best FID score across five trials. The best FID score of a trial is the lowest score computed across 800 epochs [16]. We report the variance of the best FID score across the five trials. We record the average epoch the best FID score occurred in across the five trials, and the variance of the best epoch.

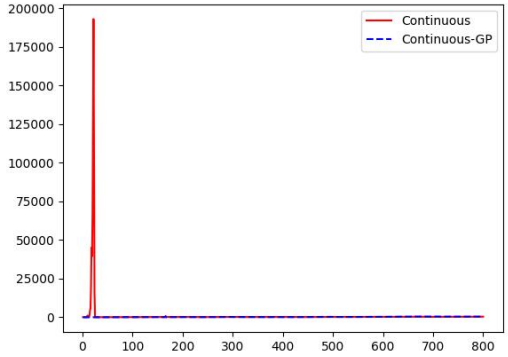
We plot the average FID score of each of the 800 epochs for all experiments. Finally, we present samples of images with the best FID scores, and plot samples of images to show how the output changes as the latent code varies.

Table 3.1: InfoGAN experiments on the MNIST dataset: the average and variance of the best FID scores and the average and variance of the epoch this occurs taken over five trials.

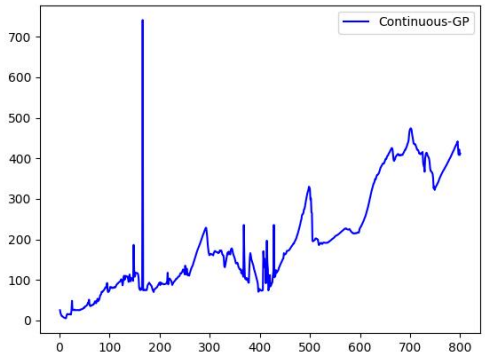
	Average best FID score	Best FID scores variance	Average epoch	Epoch variance
Discrete	**1.738**	1.614	584.40	$\mathbf{2.32 \times 10^{-3}}$
Discrete-GP	1.778	**1.522**	**488.75**	1.17×10^{-2}
Continuous	4.149	1.663	**258.67**	19.565
Continuous-GP	**2.967**	**1.462**	259.00	**4.790**



(a) FID plot for Discrete InfoGANs.



(b) FID plot for Continuous InfoGANs.



(c) FID plot for Continuous InfoGAN-GP.

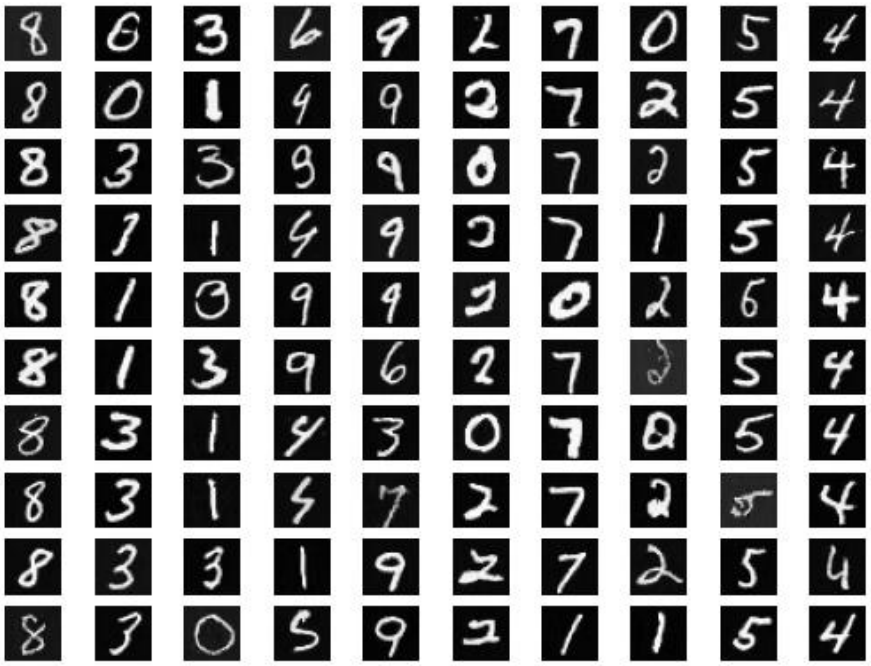


Figure 3.2: Output of Discrete InfoGAN, plotted on c_1.

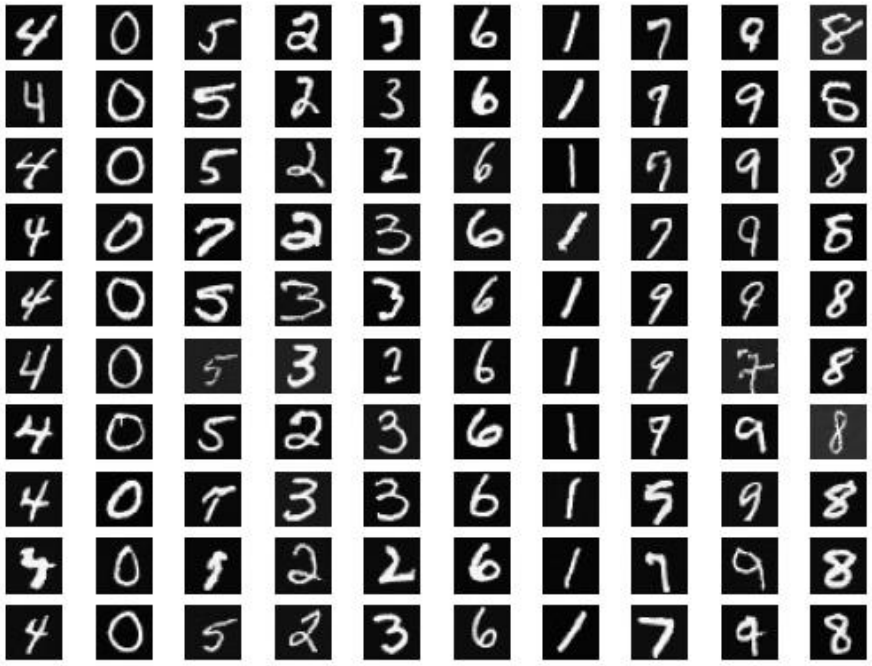


Figure 3.3: Output of Discrete InfoGAN, plotted on c_1.

Figure 3.4: Output of Continuous InfoGAN, plotted on $c_2 \in [-2, 2]$ with $c_1 = 0$, $c_3 = -2$ fixed.

Figure 3.5: Output of Continuous InfoGAN, plotted on $c_3 \in [-2, 2]$ with $c_1 = 0$, $c_2 = -0.9474$ fixed.

3.3.3 Discussion

From Table 3.1, we observe that the discrete InfoGAN does not experience gains in image quality with the addition of GP. However, GP is essential in improving the quality of images produced by the continuous InfoGAN; on average, GP improves the model's average FID score by approximately 140%.

Figures 3.1b and 3.1c show that the continuous InfoGAN can experience erratic instability during training. This might be due to the addition of the differential entropy term in the continuous InfoGAN loss function; it could be useful to train the continuous InfoGAN using a lower value of λ_c to see if decreasing λ_c improves overall stability.

The discrete InfoGAN does a decent job of achieving its mutual information-maximization objective. The model has slight difficulty differentiating between twos and threes (see the fourth and fifth columns of Figure 3.3). Note that 7/10 of the images in column 4 are twos, and 8/10 of images in column 5 are threes.

Finally, we observe the effects of the continuous latent codes on the digit 8 in Figures 3.4 and 3.5. It appears that varying c_2 (Figure 3.4) affects the slant of the digit, the digit slants more to the left for negative values of c_2, and more to the right for positive values of c_2. Varying c_3 does a decent job of changing the thickness of the digit; the InfoGAN tries to produce thicker digits for negative values of c_3 and thinner digits for positive values of c_3.

A future direction of this work could be to generalize the InfoGAN using Rényi information measures [34] [5] (i.e., a hybrid of RényiGAN and InfoGAN), by replacing the Shannon cross-entropies in the loss function with Rényi cross-entropies, and the Shannon mutual information term with one of the three α mutual information terms presented in Chapter 2.

Chapter 4

$\mathcal{L}_\alpha$-GANs

We now present our main contribution that unifies various generator loss functions under a CPE-based loss function $\mathcal{L}_\alpha$ for a dual-objective GAN, $\mathcal{L}_\alpha$-GAN, with a canonical discriminator loss function loss function that is optimized as in [11]. When some regularity conditions are satisfied, we show that under the optimal discriminator, our generator loss minimizes a Jensen-f-divergence. This chapter is taken from our preprint [39].

Let $(\mathcal{X}, \mathcal{B}(\mathcal{X}), \mu)$ be the measure space of $n \times n \times m$ images (where $m = 1$ for black and white images and $m = 3$ for RGB images), and let $(\mathcal{Z}, \mathcal{B}(\mathcal{Z}), \mu)$ be a measure space such that $\mathcal{Z} \subseteq \mathbb{R}^d$. The discriminator neural network is given by $D : \mathcal{X} \to [0, 1]$, and the generator neural network is given by $G : \mathcal{Z} \to \mathcal{X}$. The generator's noise input is sampled from a multivariate Gaussian distribution $P_{\mathbf{z}} : \mathcal{Z} \to [0, 1]$. We denote the probability distribution of the real data by $P_{\mathbf{x}} : \mathcal{X} \to [0, 1]$ and the probability distribution of the generated data by $P_{\mathbf{g}} : \mathcal{X} \to [0, 1]$. We also set $P_{\mathbf{x}}$ and $P_{\mathbf{g}}$ as the densities corresponding to $P_{\mathbf{x}}$ and $P_{\mathbf{g}}$, respectively. We begin by introducing the $\mathcal{L}_\alpha-$GAN system.

Definition 46. *For a fixed $\alpha \in \mathcal{A} \subseteq \mathbb{R}$, let $\mathcal{L}_\alpha : \{0, 1\} \times [0, 1] \to [0, \infty)$ such that $\mathcal{L}_\alpha$*

is symmetric in the sense that

$$\mathcal{L}_\alpha(1,\hat{y}) = \mathcal{L}_\alpha(0,1-\hat{y}), \qquad \hat{y}\in[0,1]. \tag{4.1}$$

The $\boldsymbol{\mathcal{L}_\alpha}$**−GAN** *system is characterized by* $(V_D, V_{\mathcal{L}_\alpha,G})$*, where* $V_D : \mathcal{X}\times\mathcal{Z}\to\mathbb{R}$ *is the discriminator loss function, and* $V_{\mathcal{L}_\alpha,G} : \mathcal{X}\times\mathcal{Z}\to\mathbb{R}$ *is the generator loss function, given by*

$$V_{\mathcal{L}_\alpha,G}(D,G) = \mathbb{E}_{\boldsymbol{A}\sim P_{\mathbf{x}}}[-\mathcal{L}_\alpha(1,D(\boldsymbol{A}))] + \mathbb{E}_{\boldsymbol{B}\sim P_{\mathbf{g}}}[-\mathcal{L}_\alpha(0,D(\boldsymbol{B}))]. \tag{4.2}$$

Moreover, the $\boldsymbol{\mathcal{L}_\alpha}$**−GAN *problem*** *is defined by*

$$\sup_D V_D(D,G) \tag{4.3}$$

$$\inf_G V_{\mathcal{L}_\alpha,G}(D,G). \tag{4.4}$$

4.1 Main Result

Theorem 11. *For a fixed* $\alpha\in\mathcal{A}\subseteq\mathbb{R}$ *and* $\mathcal{L}_\alpha : \{0,1\}\times[0,1]\to[0,\infty)$*, let* $(V_D, V_{\mathcal{L}_\alpha,G})$ *be the loss functions of a* $\mathcal{L}_\alpha-$GAN*, and consider the joint optimization in* (4.3)-(4.4)*. If* V_D *is a canonical loss function in the sense that it is maximized at* $D = D^*$*, where*

$$D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}, \tag{4.5}$$

then (4.4) *reduces to*

$$\inf_G V_{\mathcal{L}_\alpha,G}(D^*,G) = \inf_G 2a\mathrm{JD}_{f_\alpha}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2ab, \tag{4.6}$$

where $\mathrm{JD}_{f_\alpha}(\cdot||\cdot)$ *is the Jensen-*f_α*-divergence, and* $f_\alpha : [0,2] \to \mathbb{R}$ *is a continuous convex function*[1] *satisfying* $f_\alpha(1) = 0$ *and*

$$f_\alpha(u) = -u\left(\frac{1}{a}\mathcal{L}_\alpha\left(1, \frac{u}{2}\right) - b\right), \tag{4.7}$$

where $a, b \in \mathbb{R}$, $a \neq 0$. *Finally,* (4.6) *is minimized when* $P_{\mathbf{x}} = P_{\mathbf{g}}$ *(a.e.).*

Remark 12. *Note that not only* D^* *given in* (4.5) *is an optimal discriminator of the (original) VanillaGAN discriminator loss function, but it also optimizes the LSGAN/LkGAN discriminator loss function when their discriminator's labels for fake and real data,* γ *and* β*, respectively satisfy* $\gamma = 1$ *and* $\beta = 0$ *(see Section 4.2.3).*

Proof. Under the assumption that V_D is maximized at $D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$, we have that

$$\begin{aligned}
V_{\mathcal{L}_\alpha,G}(D^*, G) &= \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}[-\mathcal{L}_\alpha(1, D^*(\mathbf{A}))] + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}[-\mathcal{L}_\alpha(0, D^*(\mathbf{B}))] \\
&= -\int_{\mathcal{X}} P_{\mathbf{x}}\mathcal{L}_\alpha(1, D^*)\, d\mu - \int_{\mathcal{X}} P_{\mathbf{g}}\mathcal{L}_\alpha(0, D^*)\, d\mu \\
&= -\int_{\mathcal{X}} P_{\mathbf{x}}\mathcal{L}_\alpha\left(1, \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right) d\mu - \int_{\mathcal{X}} P_{\mathbf{g}}\mathcal{L}_\alpha\left(0, \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right) d\mu \\
&= -2\int_{\mathcal{X}} \left(\frac{P_{\mathbf{x}}+P_{\mathbf{g}}}{2}\right)\frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\mathcal{L}_\alpha\left(1, \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right) d\mu \\
&\quad - 2\int_{\mathcal{X}} \left(\frac{P_{\mathbf{x}}+P_{\mathbf{g}}}{2}\right)\frac{P_{\mathbf{g}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\mathcal{L}_\alpha\left(0, \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right) d\mu \\
&\overset{(a)}{=} -2\int_{\mathcal{X}} \left(\frac{P_{\mathbf{x}}+P_{\mathbf{g}}}{2}\right)\frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\mathcal{L}_\alpha\left(1, \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right) d\mu \\
&\quad - 2\int_{\mathcal{X}} \left(\frac{P_{\mathbf{x}}+P_{\mathbf{g}}}{2}\right)\frac{P_{\mathbf{g}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\mathcal{L}_\alpha\left(1, \frac{P_{\mathbf{g}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right) d\mu \\
&\overset{(b)}{=} -2\int_{\mathcal{X}} \left(\frac{P_{\mathbf{x}}+P_{\mathbf{g}}}{2}\right)\frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\left(\frac{-af_\alpha\left(\frac{2P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right)}{\frac{2P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}} + ab\right) d\mu
\end{aligned}$$

[1] It is implicitly implied by (4.7) that $\mathcal{L}_\alpha(1, u)$ is itself continuous and convex in u for $u \in [0, 1]$.

$$
\begin{aligned}
&- 2\int_{\mathcal{X}} \left(\frac{P_{\mathbf{x}}+P_{\mathbf{g}}}{2}\right) \frac{P_{\mathbf{g}}}{P_{\mathbf{x}}+P_{\mathbf{g}}} \left(\frac{-a f_\alpha\left(\frac{2P_{\mathbf{g}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right)}{\frac{2P_{\mathbf{g}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}} + ab\right) d\mu \\
&= 2a\left(\frac{1}{2}\int_{\mathcal{X}} \frac{P_{\mathbf{x}}+P_{\mathbf{g}}}{2} f_\alpha\left(\frac{2P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right) d\mu \right. \\
&\quad \left. + \frac{1}{2}\int_{\mathcal{X}} \frac{P_{\mathbf{x}}+P_{\mathbf{g}}}{2} f_\alpha\left(\frac{2P_{\mathbf{g}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}\right) d\mu\right) - 2ab \\
&= 2a\, \mathrm{JD}_{f_\alpha}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2ab,
\end{aligned}
$$

where:

- (a) holds since $\mathcal{L}_\alpha(1,u) = \mathcal{L}_\alpha(0,1-u)$, where $u = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$ (symmetry property of $\mathcal{L}_\alpha$).
- (b) holds by solving for $\mathcal{L}_\alpha(1,u)$ in terms of $f_\alpha(2u)$ in (4.7), where $u = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$ in the first term, and $u = \frac{P_{\mathbf{g}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$ in the second term.

□

4.2 Applications

We next show that the $\mathcal{L}_\alpha-$GAN of Theorem 11 recovers as special cases a number of well-known GAN generator loss functions and their equilibrium points (under an optimal classical discriminator D^*).

4.2.1 VanillaGAN

VanillaGAN [11] uses the same loss function V_{VG} for both generator and discriminator, given by

$$
V_{\mathrm{VG}}(D,G) = \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}[\log D(\mathbf{A})] + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}[\log(1-D(\mathbf{B}))], \tag{4.8}
$$

and can be cast as a saddle point optimization problem:

$$\inf_G \sup_D V_{\mathrm{VG}}(D, G). \tag{4.9}$$

It is shown in [11] that the optimal discriminator for (4.9) is given by $D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$, as in (4.5). When $D = D^*$, the optimization reduces to minimizing the Jensen-Shannon divergence:

$$\inf_G V_{\mathrm{VG}}(D^*, G) = \inf_G 2\mathrm{JSD}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2\log 2. \tag{4.10}$$

We next show that (4.10) can be obtained from Theorem 11.

Lemma 4. *Consider the optimization of the VanillaGAN given in* (4.9)*. Then we have that*

$$V_{\mathrm{VG}}(D^*, G) = 2\mathrm{JSD}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2\log 2 = V_{\mathcal{L}_\alpha, G}(D^*, G),$$

where $\mathcal{L}_\alpha(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$ *for all* $\alpha \in \mathcal{A} = \mathbb{R}$.

Proof. For any fixed $\alpha \in \mathbb{R}$, let the function $\mathcal{L}_\alpha$ in (4.2) be as defined in the statement:

$$\mathcal{L}_\alpha(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}).$$

Note that $\mathcal{L}_\alpha$ is symmetric, since for $\hat{y} \in [0, 1]$, we have that

$$\mathcal{L}_\alpha(1, \hat{y}) = -\log(\hat{y}) = \mathcal{L}_\alpha(0, 1-\hat{y}).$$

We now derive f_α from $\mathcal{L}_\alpha$ using (4.7): setting $a = 1$ and $b = \log 2$, we have that

$$\begin{aligned} f_\alpha(u) &= -u\left(\frac{1}{a}\mathcal{L}_\alpha\left(1, \frac{u}{2}\right) - b\right) \\ &= -u\left(-\log\frac{u}{2} - \log 2\right) \\ &= u\log u. \end{aligned}$$

Clearly f_α is continuous on $[0, \infty)$, and $f_\alpha(1) = 0$. Furthermore, we have that $f''(u) = \frac{1}{u} > 0$; hence, f_α is convex. By Lemma 3, we know that under the generating function $f(u) = u\log(u)$, the Jensen-f divergence reduces to the Jensen-Shannon divergence. Therefore, by Theorem 11, we have that

$$V_{\mathcal{L}_\alpha, G}(D^*, G) = 2a\mathrm{JD}_{f_\alpha}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2ab = 2\mathrm{JSD}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2\log 2 = V_{\mathrm{VG}}(D^*, G).$$

□

4.2.2 α-GAN

The notion of α-GANs is introduced in [20] as a way to unify several existing GANs using a parameterized loss function. We begin by describing this notion.

Definition 47. [20] *Let $y \in \{0, 1\}$ be a binary label, $\hat{y} \in [0, 1]$, and fix $\alpha > 0$. The **α-loss** between y and $\hat{y}$ is the map $\ell_\alpha : \{0, 1\} \times [0, 1] \to [0, \infty)$ given by*

$$\ell_\alpha(y, \hat{y}) = \begin{cases} \dfrac{\alpha}{\alpha - 1}\left(1 - y\hat{y}^{\frac{\alpha-1}{\alpha}} + (1-y)(1-\hat{y})^{\frac{\alpha-1}{\alpha}}\right), & \alpha \in (0, 1) \cup (1, \infty) \\ -y\log\hat{y} - (1-y)\log(1-\hat{y}), & \alpha = 1. \end{cases} \tag{4.11}$$

Definition 48. [20] *For $\alpha > 0$, the* $\boldsymbol{\alpha}-$**GAN** ***loss function*** *is given by*

$$V_\alpha(D,G) = \mathbb{E}_{\boldsymbol{A}\sim P_{\mathbf{x}}}[-\ell_\alpha(1, D(\boldsymbol{A}))] + \mathbb{E}_{\boldsymbol{B}\sim P_{\mathbf{g}}}[-\ell_\alpha(0, D(\boldsymbol{B}))]. \tag{4.12}$$

The joint optimization of the $\alpha-$GAN *problem is given by*

$$\inf_G \sup_D V_\alpha(D,G). \tag{4.13}$$

It is known that α-GAN recovers several well-known GANs by varying the α parameter, notably, the VanillaGAN ($\alpha = 1$) [11] and the HellingerGAN ($\alpha = \frac{1}{2}$) [31]. Furthermore, as $\alpha \to \infty$, V_α recovers a translated version of the WassersteinGAN loss function [3]. We now present the solution to the joint optimization problem presented in (4.13).

Proposition 13. [20] *Let $\alpha > 0$, and consider the joint optimization of the α-GAN presented in* (4.13)*. The discriminator D^* that maximizes the loss function is given by*

$$D^* = \frac{{P_{\mathbf{x}}}^\alpha}{{P_{\mathbf{x}}}^\alpha + {P_{\mathbf{g}}}^\alpha}. \tag{4.14}$$

Furthermore, when $D = D^*$ is fixed, the problem in (4.13) *reduces to minimizing an Arimoto divergence (as defined in Table 2.1) when $\alpha \neq 1$:*

$$\inf_G V_\alpha(D^*, G) = \inf_G \mathcal{A}_\alpha(P_{\mathbf{x}}||P_{\mathbf{g}}) + \frac{\alpha}{\alpha - 1}\left(2^{\frac{1}{\alpha}} - 2\right), \tag{4.15}$$

and a Jensen-Shannon divergence when $\alpha = 1$*:*

$$\inf_G V_1(D^*, G) = \inf_G \mathrm{JSD}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2\log 2, \tag{4.16}$$

where (4.15) *and* (4.16) *achieve their minima iff* $P_{\mathbf{x}} = P_{\mathbf{g}}$ *(a.e.).*

Recently, α-GAN was generalized in [41] to implement a dual objective GAN, which we recall next.

Definition 49. [41] *For* $\alpha_D > 0$ *and* $\alpha_G > 0$*, the* $(\boldsymbol{\alpha_D}, \boldsymbol{\alpha_G})-$**GAN***'s* ***optimization*** *is given by*

$$\sup_D V_{\alpha_D}(D, G) \tag{4.17}$$

$$\inf_G V_{\alpha_G}(D, G) \tag{4.18}$$

where V_{α_D} *and* V_{α_G} *are defined in* (4.12)*, with* α *replaced by* α_D *and* α_G *respectively.*

We now recall the solution to the (α_D, α_G)-GAN's optimization formulated in (4.17)-(4.18).

Proposition 14. [41] *Consider the joint optimization in* (4.17)-(4.18)*. Let parameters* α_D*,* $\alpha_G > 0$ *satisfy*

$$\left(\alpha_D \leq 1, \alpha_G > \frac{\alpha_D}{\alpha_D + 1}\right) \text{ or } \left(\alpha_D > 1, \frac{\alpha_D}{2} < \alpha_G \leq \alpha_D\right). \tag{4.19}$$

The discriminator D^* *that maximizes* V_{α_D} *is given by*

$$D^* = \frac{{P_{\mathbf{x}}}^{\alpha_D}}{{P_{\mathbf{x}}}^{\alpha_D} + {P_{\mathbf{g}}}^{\alpha_D}}. \tag{4.20}$$

Furthermore, when $D = D^*$ is fixed, the minimization of V_{α_G} in (4.18) *is equivalent to the following f-divergence minimization:*

$$\inf_G V_{\alpha_G}(D^*, G) = \inf_G D_{f_{\alpha_D,\alpha_G}}(P_{\mathbf{x}}||P_{\mathbf{g}}) + \frac{\alpha}{\alpha - 1}\left(2^{\frac{1}{\alpha}} - 2\right), \tag{4.21}$$

where $f_{\alpha_D,\alpha_G} : [0, \infty) \to \mathbb{R}$ is given by

$$f_{\alpha_D,\alpha_G}(u) = \frac{\alpha_G}{\alpha_G - 1}\left(\frac{u^{\alpha_D\left(1-\frac{1}{\alpha_G}\right)+1} + 1}{\left(u^{\alpha_D} + 1\right)^{1-\frac{1}{\alpha_G}}}\right). \tag{4.22}$$

We now apply the (α_D, α_G)-GAN to our main result in Theorem 11 by showing that (4.6) can recover (4.21) when $\alpha_D = 1$ (which corresponds to a VanillaGAN discriminator loss function).

Lemma 5. *Consider the $(\alpha_D, \alpha_G)-$*GAN *given in Definition 49. Let $\alpha_D = 1$ and $\alpha_G = \alpha > \frac{1}{2}$. Then, the solution to* (4.18) *presented in Proposition 14 is equivalent to minimizing a Jensen-f_α-divergence: specifically, if D^* is the optimal discriminator given by* (4.20)*, which is equivalent to* (4.5) *when $\alpha_D = 1$, then $V_{\alpha,G}(D^*, G)$ in* (4.21) *satisfies*

$$V_{\alpha,G}(D^*, G) = 2^{\frac{1}{\alpha}}\mathrm{JD}_{f_\alpha}(P_{\mathbf{x}}||P_{\mathbf{g}}) + \frac{\alpha}{\alpha - 1}(2^{\frac{1}{\alpha}} - 2) = V_{\mathcal{L}_\alpha,G}(D^*, G), \tag{4.23}$$

where $\mathcal{L}_\alpha(y, \hat{y}) = \ell_\alpha(y, \hat{y})$ and

$$f_\alpha(u) = \frac{\alpha}{\alpha - 1}\left(u^{2-\frac{1}{\alpha}} - u\right), \quad u \geq 0. \tag{4.24}$$

Proof. We want to show that Theorem 11 recovers Proposition 14. We set $\mathcal{L}_\alpha(y, \hat{y}) =$

$\ell_\alpha(y, \hat{y})$. Note that ℓ_α is symmetric, since we have that

$$\ell_\alpha(1, \hat{y}) = \frac{\alpha}{\alpha - 1}(1 - \hat{y}^{1-\frac{1}{\alpha}}) = \ell_\alpha(0, 1 - \hat{y}).$$

From Lemma 4, we know that when $\alpha = 1$, $f_\alpha(u) = u \log u$. For $\alpha \in (0, 1) \cup (1, \infty)$, setting $a = 2^{\frac{1}{\alpha}-1}$ and $b = \frac{\alpha}{\alpha-1}\left(2^{1-\frac{1}{\alpha}} - 1\right)$ in (4.7), we have that

$$\begin{aligned} f_\alpha(u) &= -u\left(\frac{1}{a}\mathcal{L}_\alpha\left(1, \frac{u}{2}\right) - b\right) \\ &= -u\left(2^{1-\frac{1}{\alpha}}\frac{\alpha}{\alpha-1}\left(1 - \left(\frac{u}{2}\right)^{1-\frac{1}{\alpha}}\right) - \frac{\alpha}{\alpha-1}(2^{1-\frac{1}{\alpha}} - 1)\right) \\ &= \frac{\alpha}{\alpha-1}(-u)[2^{1-\frac{1}{\alpha}} - u^{1-\frac{1}{\alpha}} - (2^{1-\frac{1}{\alpha}} - 1)] \\ &= \frac{\alpha}{\alpha-1}(u^{2-\frac{1}{\alpha}} - u). \end{aligned}$$

Clearly $f_\alpha(1) = 0$. Furthermore for $\alpha \neq 1$, we have that

$$f''_\alpha(u) = \frac{(2\alpha - 1)u^{\frac{-1}{\alpha}}}{\alpha}, \quad u \geq 0,$$

which is positive for $\alpha > \frac{1}{2}$, and f_α is convex for $\alpha > \frac{1}{2}$. Therefore, by Theorem 11, we have that

$$\begin{aligned} V_{\mathcal{L}_\alpha, G}(D^*, G) &= 2a\mathrm{JD}_{f_\alpha}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2ab \\ &= 2 \cdot 2^{\frac{1}{\alpha}-1}\mathrm{JD}_{f_\alpha}(P_{\mathbf{x}}||P_{\mathbf{g}}) - 2\frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}-1}(2^{1-\frac{1}{\alpha}} - 1) \\ &= 2^{\frac{1}{\alpha}}\mathrm{JD}_{f_\alpha}(P_{\mathbf{x}}||P_{\mathbf{g}}) + \frac{\alpha}{\alpha-1}(2^{\frac{1}{\alpha}} - 2). \end{aligned}$$

We now show that the above Jensen-f_α-divergence is equal to the $f_{1,\alpha}$-divergence

originally derived for the $(1,\alpha)$-GAN problem of Proposition 14 (note from Proposition 14, that if $\alpha_D = 1$, then $\alpha_G = \alpha > \frac{1}{2}$, so the range of α concurs with the range above required for the convexity of f_α). For any two distributions p and q with common support $\mathcal{X}$, we have that

$$
\begin{aligned}
D_{f_{1,\alpha}}(p||q) &= \frac{\alpha}{\alpha-1}\int_{\mathcal{X}} q\frac{\left(\frac{p}{q}\right)^{2-\frac{1}{\alpha}}+1}{\left(\frac{p}{q}+1\right)^{1-\frac{1}{\alpha}}}\,d\mu - \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}} \\
&= \frac{\alpha}{\alpha-1}\int_{\mathcal{X}} q\frac{\left(\frac{p}{q}\right)^{2-\frac{1}{\alpha}}+1}{\left(\frac{p+q}{q}\right)^{1-\frac{1}{\alpha}}}\,d\mu - \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}} \\
&= \frac{\alpha}{\alpha-1}\int_{\mathcal{X}} \left((p+q)\left(\frac{p}{p+q}\right)^{2-\frac{1}{\alpha}} + (p+q)\left(\frac{q}{p+q}\right)^{2-\frac{1}{\alpha}}\right)d\mu \\
&\quad - \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}} \\
&= \frac{\alpha}{\alpha-1}\frac{2}{2^{2-\frac{1}{\alpha}}}\int_{\mathcal{X}} \left(\frac{p+q}{2}\left(\frac{2p}{p+q}\right)^{2-\frac{1}{\alpha}} + \frac{p+q}{2}\left(\frac{2q}{p+q}\right)^{2-\frac{1}{\alpha}}\right)d\mu \\
&\quad - \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}} \\
&= \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}-1}\int_{\mathcal{X}} \left(\frac{p+q}{2}\left(\left(\frac{2p}{p+q}\right)^{2-\frac{1}{\alpha}} - \frac{2p}{p+q}\right) + p\right)d\mu \\
&\quad + \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}-1}\int_{\mathcal{X}} \left(\frac{p+q}{2}\left(\left(\frac{2q}{p+q}\right)^{2-\frac{1}{\alpha}} - \frac{2q}{p+q}\right) + q\right)d\mu \\
&\quad - \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}} \\
&= \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}}\frac{1}{2}\left(\int_{\mathcal{X}} \frac{p+q}{2}\left(\left(\frac{2p}{p+q}\right)^{2-\frac{1}{\alpha}} - \frac{2p}{p+q}\right)d\mu + 1\right) \\
&\quad + \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}}\frac{1}{2}\left(\int_{\mathcal{X}} \frac{p+q}{2}\left(\left(\frac{2q}{p+q}\right)^{2-\frac{1}{\alpha}} - \frac{2q}{p+q}\right)d\mu + 1\right) \\
&\quad - \frac{\alpha}{\alpha-1}2^{\frac{1}{\alpha}}
\end{aligned}
$$

$$= 2^{\frac{1}{\alpha}} \mathrm{JD}_{f_\alpha}(p||q) + \frac{\alpha}{\alpha-1} 2^{\frac{1}{\alpha}-1}(2) - \frac{\alpha}{\alpha-1} 2^{\frac{1}{\alpha}}$$
$$= 2^{\frac{1}{\alpha}} \mathrm{JD}_{f_\alpha}(p||q).$$

Therefore, $V_{\mathcal{L}_\alpha,G}(D^*, G) = V_\alpha(D^*, G)$. □

Note that this lemma generalizes Lemma 4; the VanillaGAN is a special case of the $(1, \alpha)$-GAN for $\alpha = 1$.

4.2.3 Shifted LkGANs and LSGANs

Least Squares GAN (LSGAN) was proposed in [25] to mitigate the vanishing gradient problem with VanillaGAN and to stabilize training performance. The LSGAN's loss function is derived from the squared error distortion measure, where we aim to minimize the distortion between the data samples and a target value we want the discriminator to assign the samples to. The LSGAN was generalized with the LkGAN in [5] by replacing the squared error distortion measure with the absolute error distortion measure of order $k \geq 1$, therefore introducing an additional degree of freedom to the generator's loss function. We first state the general LkGAN problem. We then apply the result of Theorem 11 to the loss functions of LSGAN and LkGAN.

Definition 50. [5] *Let γ, β, $c \in [0, 1]$ and let $k \geq 1$. The **LkGAN's loss functions**, denoted by $V_{LSGAN,D}$ and $V_{k,G}$ are given by*

$$V_{LSGAN,D}(D, G) = -\frac{1}{2}\mathbb{E}_{\boldsymbol{A}\sim P_{\mathbf{x}}}[(D(\boldsymbol{A}) - \beta)^2] - \frac{1}{2}\mathbb{E}_{\boldsymbol{B}\sim P_{\mathbf{g}}}[(D(\boldsymbol{B}) - \gamma)^2] \tag{4.25}$$

$$V_{k,G}(D, G) = \mathbb{E}_{\boldsymbol{A}\sim P_{\mathbf{x}}}[|D(\boldsymbol{A}) - c|^k] + \mathbb{E}_{\boldsymbol{B}\sim P_{\mathbf{g}}}[|D(\boldsymbol{B}) - c|^k]. \tag{4.26}$$

*The **LkGAN problem** is the joint optimization*

$$\sup_D V_{LSGAN,D}(D,G) \tag{4.27}$$

$$\inf_G V_{k,G}(D,G). \tag{4.28}$$

We next recall the solution to (4.27), which is a minimization of the Pearson-Vajda divergence $|\chi|^k(\cdot||\cdot)$ of order k (as defined in Table 2.1).

Proposition 15. [5] *Consider the joint optimization for the LkGAN presented in* (4.27)*. Then, the optimal discriminator D^* that maximizes $V_{LSGAN,D}$ in* (4.25) *is given by*

$$D^* = \frac{\gamma P_{\mathbf{x}} + \beta P_{\mathbf{g}}}{P_{\mathbf{x}} + P_{\mathbf{g}}}. \tag{4.29}$$

Furthermore, if $D = D^*$, and $\gamma - \beta = 2(c - \beta)$, the minimization of $V_{k,G}$ in (4.26) *reduces to*

$$\inf_G V_{k,G}(D,G) = \inf_G |c-\beta|^k |\chi|^k (P_{\mathbf{x}} + P_{\mathbf{g}} || 2P_{\mathbf{g}}). \tag{4.30}$$

Note that the LSGAN [25] is a special case of LkGAN, as we recover LSGAN when $k = 2$ [5].

By scrutinizing Proposition 15 and Theorem 11, we observe that the former cannot be recovered from the latter. However we can use Theorem 11 by slightly modifying the LkGAN generator's loss function. First, for the dual objective GAN proposed in Theorem 11, we need $D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$. By (4.29), this is achieved for $\gamma = 1$ and $\beta = 0$.

Then, we define the intermediate loss function

$$\tilde{V}_{k,G}(D,G) = \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}[|D(\mathbf{A}) - c_1|^k] + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}[|D(\mathbf{B}) - c_2|^k]. \tag{4.31}$$

Comparing the above loss function with (4.2), we note that setting $c_1 = 0$ and $c_2 = 1$ in (4.31) satisfies the symmetry property of $\mathcal{L}_\alpha$. Finally, to ensure the generating function f_α satisfies $f_\alpha(1) = 0$, we shift each term in (4.31) by 1. Putting these changes together, we propose a revised generator loss function, denoted by $\hat{V}_{k,G}$, given by

$$\hat{V}_{k,G}(D,G) = \mathbb{E}_{\mathbf{A}\sim P_{\mathbf{x}}}[|D(\mathbf{A})|^k - 1] + \mathbb{E}_{\mathbf{B}\sim P_{\mathbf{g}}}[|1 - D(\mathbf{B})|^k - 1]. \tag{4.32}$$

We call a system that uses (4.32) as a generator loss function a **Shifted LkGAN (SLkGAN)**. If $k = 2$, we have a shifted version of the LSGAN generator loss function, which we call the **Shifted LSGAN (SLSGAN)**. Note that none of these modifications alter the gradients of $V_{k,G}$ in (4.26), since the first term is independent of G, the choice of c_1 is irrelevant, and translating a function by a constant does not change its gradients. However, from Proposition 15, for $\gamma = 0$, $\beta = 1$ and $c = 1$, we do not have that $\gamma - \beta = 2(c - \beta)$, and as a result, this modified problem does not reduce to minimizing a Pearson-Vajda divergence. Consequently, we can relax the condition on k in Definition 50 to just $k > 0$. We now show how Theorem 11 can be applied to $\mathcal{L}_\alpha$-GAN using (4.32).

Lemma 6. *Let $k > 0$. Let V_D be a discriminator loss function, and let $\hat{V}_{k,G}$ be the*

generator's loss function defined in (4.32). Consider the joint optimization

$$\sup_D V_D(D,G) \tag{4.33}$$

$$\inf_G \hat{V}_{k,G}(D,G) \tag{4.34}$$

If V_D is optimized at $D^* = \frac{P_{\mathbf{x}}}{P_{\mathbf{x}}+P_{\mathbf{g}}}$ (i.e., V_D is canonical), then we have that

$$\hat{V}_{k,G}(D^*,G) = \frac{1}{2^{k-1}}\mathrm{JD}_{f_k}(P_{\mathbf{x}}||P_{\mathbf{g}}) + \frac{1}{2^{k-1}} - \frac{1}{2},$$

where f_k is given by

$$f_k(u) = u(u^k - 1), \qquad u \geq 0.$$

Examples of $V_D(D,G)$ that satisfy the requirements of Lemma 6 include the LkGAN discriminator loss function given by (4.25) with $\gamma = 1$ and $\beta = 0$, and the VanillaGAN discriminator loss function given by (4.8).

Proof. Let $k > 0$. We can restate the SLkGAN's generator loss function in (4.32) in terms of $V_{\mathcal{L}_\alpha,G}$ in (4.2): we have that $V_{\mathcal{L}_\alpha,G}(D^*,G) = \hat{V}_{k,G}(D^*,G)$, where $\alpha = k$ and $\mathcal{L}_k : \{0,1\} \times [0,1] \to [0,\infty)$ is given by

$$\mathcal{L}_k(y,\hat{y}) = -(y(\hat{y}^k - 1) + (1-y)((1-\hat{y})^k - 1)). \tag{4.35}$$

We have that $\mathcal{L}_k$ is symmetric, since

$$\mathcal{L}_k(1,\hat{y}) = -(\hat{y}^k - 1) = \mathcal{L}_k(0,1-\hat{y}).$$

Furthermore, setting $a = \frac{1}{2^k}$ and $b = 2^k - 1$ in (4.7), we have that

$$\begin{aligned} f_k(u) &= -u\left(\frac{1}{a}\mathcal{L}_k\left(1, \frac{u}{2}\right) - b\right) \\ &= -u\left(2^k\left(1 - \left(\frac{u}{2}\right)^k\right) - (2^k - 1)\right) \\ &= -u(2^k - u^k - 2^k + 1) \\ &= u(u^k - 1). \end{aligned}$$

We clearly have that $f_k(1) = 0$ and that f_k is continuous. Furthermore, we have that $f_k''(u) = k(k+1)u$, which is positive for $u > 0$. Therefore f_k is convex. As a result, by Theorem 11, we have that

$$\begin{aligned} \hat{V}_{k,G}(D^*, G) &= \frac{1}{2^{k-1}}\mathrm{JD}_{f_k}(P_{\mathbf{x}}||P_{\mathbf{g}}) - \frac{1}{2^{k-1}}(2^k - 1) \\ &= \frac{1}{2^{k-1}}\mathrm{JD}_{f_k}(P_{\mathbf{x}}||P_{\mathbf{g}}) + \frac{1}{2^{k-1}} - \frac{1}{2}. \end{aligned}$$

□

We conclude this section by emphasizing that Theorem 11 serves as a unifying result recovering the existing loss functions in the literature and moreover, provides a way for generalizing new ones. Our aim in the next section is to demonstrate the versatility of this result in experimentation.

4.3 Experiments

We perform two experiments on three different image datasets which we describe below.

Experiment 1. In the first experiment, we compare the (α, α)-GAN with the $(1, \alpha)$-GAN, controlling the value of α. Recall that $\alpha_D = 1$ corresponds to the canonical VanillaGAN (or DCGAN) discriminator. We aim to verify whether or not replacing an α-GAN discriminator with a VanillaGAN discriminator stabilizes or improves the system's performance depending on the value of α. Note that the result of Theorem 11 only applies to the (α_D, α_G)-GAN for $\alpha_D = 1$.

Experiment 2. We train two variants of SLkGAN, with the generator loss function as described in (4.32), parameterized by $k > 0$. We then utilize two different canonical discriminator loss functions to align with Theorem 11. The first is the VanillaGAN discriminator loss given by (4.8); we call the resulting dual objective GAN by **Vanilla-SLkGAN**. The second is the LkGAN discriminator loss, given by (4.25), where we set $\alpha = 0$ and $\beta = 1$ such that the optimal discriminator is given by (4.5). We call this system by **Lk-SLkGAN**. We compare the two variants to analyze how the value of k and choice of discriminator loss impacts the system's performance.

4.3.1 Experimental Setup

We run both experiments on three image datasets: MNIST [8], CIFAR-10 [18], and Stacked MNIST [24]. The MNIST dataset is a dataset of black and white handwritten digits between 0 and 9 of size $28 \times 28 \times 1$. The CIFAR-10 dataset is an RGB dataset of small images of common animals and modes of transportation of size $32 \times 32 \times 3$. The Stacked MNIST dataset is an RGB dataset derived from the MNIST dataset, constructed by taking three MNIST images, assigning each one of the three colour channels, and stacking the images on top of each other. The resulting images are

then padded so that each one of them have size $32 \times 32 \times 3$.

For Experiment 1, we use α values of 0.5, 5.0, 10.0 and 20.0. For each value of α, we train the (α, α)-GAN and the $(1, \alpha)$-GAN. We additionally train the DCGAN, which corresponds to the $(1, 1)$-GAN. For Experiment 2, we use k values of 0.25, 1.0, 2.0, 7.5 and 15.0. Note that when $k = 2$, we recover LSGAN. For the MNIST dataset, we run 10 trials with the random seeds 123, 500, 1600, 199621, 60677, 20435, 15859, 33764, 79878, and 36123, and train each GAN for 250 epochs. For the RGB datasets (CIFAR-10 and Stacked MNIST), we run 5 trials with the random seeds 123, 1600, 60677, 15859, 79878, and train each GAN for 500 epochs. All experiments utilize an Adam optimzer for the stochastic gradient descent algorithm, with a learning rate of 2×10^{-4}, and parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$ [17]. We also experiment with the addition of a gradient penalty (GP); we add a penalty term to the discriminator's loss function to encourage the discriminator's gradient to have a unit norm [14].

The MNIST experiments were run on one 6130 2.1 GHz 1xV100 GPU, 8 CPUs, and 16 GB of memory. The CIFAR-10 and Stacked MNIST experiments were run on one Epyc 7443 2.8 GHz GPU, 8 CPUs and 16 GB of memory. For each experiment, we report the best overall Fréchet Inception Distance (FID) score [16], the best average FID score amongst all trials and its variance, and the average epoch the best FID score occurs and its variance. The FID score for each epoch was computed over 10 000 images. For each metric, the lowest numerical value corresponds to the model with the best metric (indicated in bold in the tables). We also report how many trials

we include in our summary statistics, as it is possible for a trial to collapse and not train for the full number of epochs. The neural network architectures used in our experiments are presented in Appendix A.1. The training algorithms are presented in Appendix A.2.

4.3.2 Experimental Results

We report the FID metrics for Experiment 1 in Tables 4.1, 4.2 and 4.3, and for Experiment 2 in Tables 4.4, 4.5 and 4.6. We report only on those experiments that produced meaningful results. Models that utilize a simplified gradient penalty have the suffix "-GP". We display the output of the best-performing (α_D, α_G)-GANs in Figure 4.1 and the best-performing SLKGANs in Figure 4.3. Finally, we plot the trajectory of the FID scores throughout training epochs in Figures 4.2 and 4.4. Additional results are provided in Appendix A.3.

(α_D, α_G)-GAN	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Number of sucessful trials (/10)
(1,0.5)-GAN	1.264	1.288	2.979×10^{-4}	227.25	420.25	4
(0.5,0.5)-GAN	1.209	1.265	0.001	234.5	156.7	6
(1,5)-GAN	**1.125**	1.17	8.195×10^{-4}	230.3	617.344	10
(1,10)-GAN	1.147	**1.165**	7.984×10^{-4}	225.6	253.156	10
(10,10)-GAN	36.506	39.361	16.312	1.5	0.5	2
(1,20)-GAN	1.135	1.174	0.001	237.5	274.278	10
(20,20)-GAN	33.23	33.23	**0.0**	**1.0**	**0.0**	1
DCGAN	1.154	1.208	0.001	231.3	357.122	10

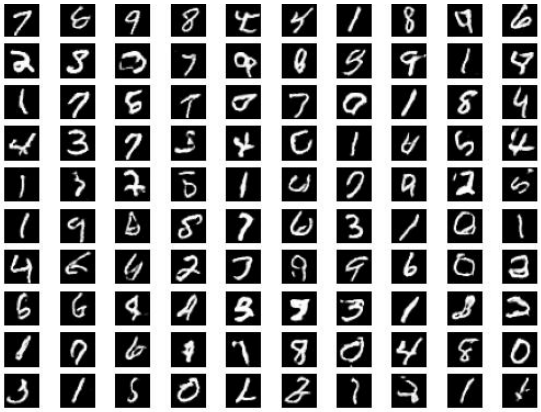
Table 4.1: (α_D, α_G)-GAN results for MNIST.

(α_D, α_G)-GAN	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Number of successful trials (/5)
(1,0.5)-GAN-GP	10.551	14.938	12.272	326.2	1808.7	5
(0.5,0.5)-GAN-GP	13.734	14.93	0.517	223.6	11378.3	5
(1,5)-GAN-GP	10.772	11.635	0.381	132.0	1233.5	5
(5,5)-GAN-GP	20.79	21.72	0.771	**84.8**	1527.2	5
(1,10)-GAN-GP	9.465	**10.187**	**0.199**	182.6	**1096.3**	5
(10,10)-GAN-GP	19.99	21.095	0.434	131.8	13374.7	5
(1,20)-GAN-GP	**8.466**	10.217	1.479	216.2	6479.7	5
(20,20)-GAN-GP	19.378	21.216	2.315	138.2	29824.2	5
DCGAN-GP	25.731	28.378	3.398	158.0	2510.5	5

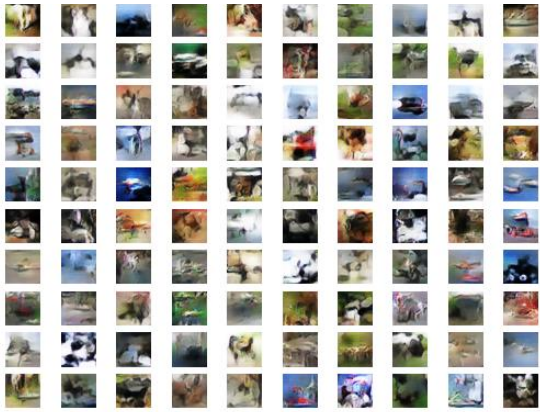
Table 4.2: (α_D, α_G)-GAN results for CIFAR-10.

(α_D, α_G)-GAN	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Number of succesful trials (/5)
(1,0.5)-GAN-GP	**4.833**	**4.997**	0.054	311.5	23112.5	2
(0.5,0.5)-GAN-GP	6.418	6.418	**0.0**	479.0	**0.0**	1
(1,5)-GAN-GP	7.98	7.988	1.357×10^{-4}	379.5	11704.5	2
(5,5)-GAN-GP	12.236	12.836	0.301	**91.5**	387.0	4
(1,10)-GAN-GP	7.502	7.528	0.001	326.5	14280.5	2
(10,10)-GAN-GP	14.22	14.573	0.249	95.0	450.0	2
(1,20)-GAN-GP	8.379	8.379	**0.0**	427.0	**0.0**	1
(20,20)-GAN-GP	16.584	16.584	**0.0**	94.0	**0.0**	1
DCGAN-GP	7.507	7.774	0.064	303.4	11870.8	5

Table 4.3: (α_D, α_G)-GAN results for Stacked MNIST.



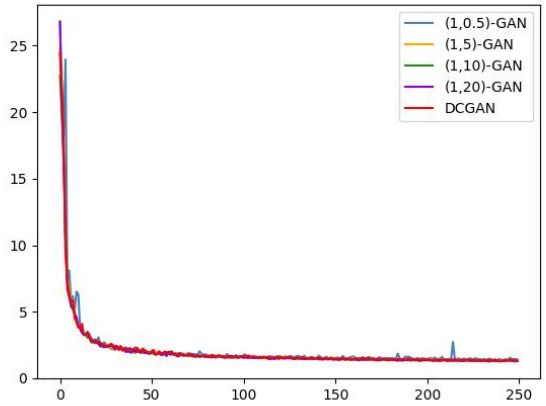
(a) (α_D, α_G)-GAN for MNIST, $\alpha_D = 1.0$, $\alpha_G = 5.0$, FID: 1.125.



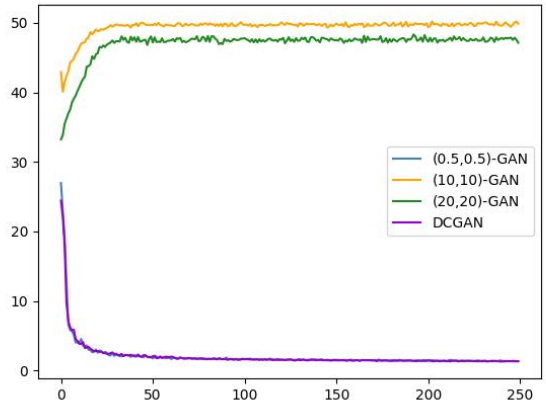
(b) (α_D, α_G)-GAN-GP for CIFAR-10, $\alpha_D = 1.0$, $\alpha_G = 20.0$, FID = 8.466.

(c) (α_D, α_G)-GAN-GP for Stacked MNIST, $\alpha_D = 1.0$, $\alpha_G = 0.5$, FID = 4.833.

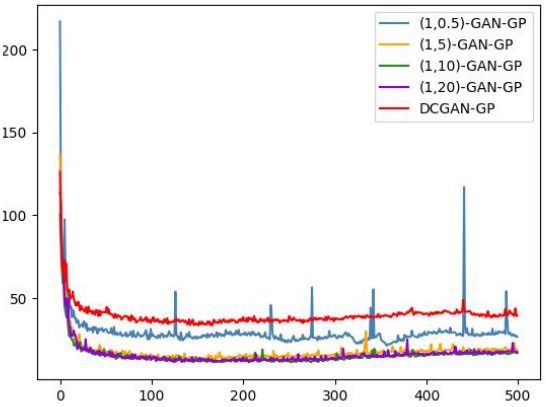
Figure 4.1: Generated images for the best-performing (α_D, α_G)-GANs.



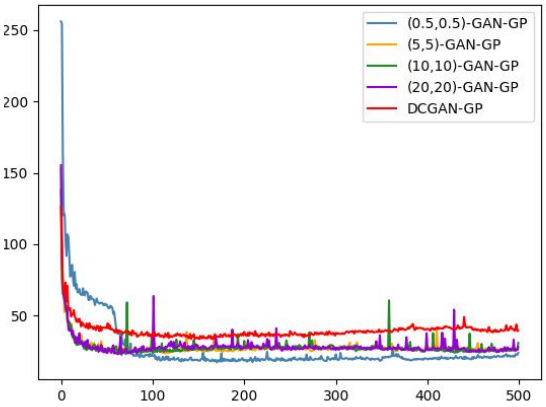
(a) $(1, \alpha)$-GANs for MNIST.



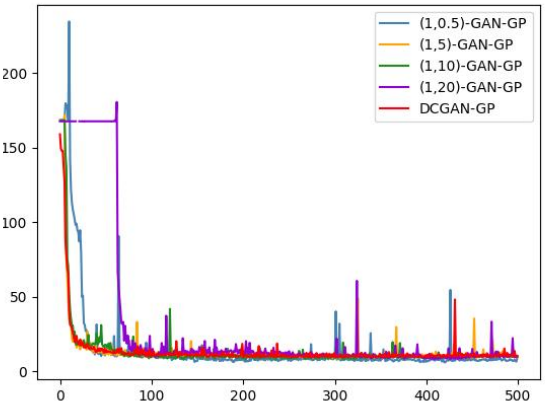
(b) (α, α)-GANs for MNIST



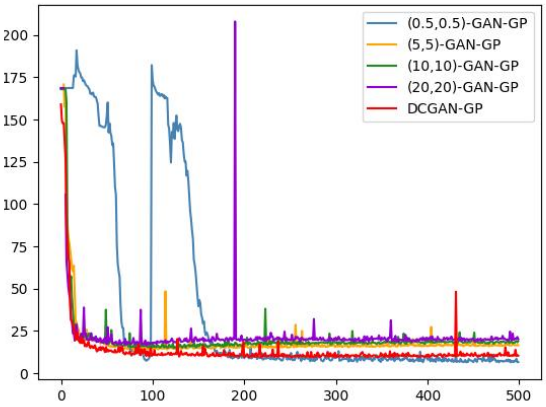
(c) $(1, \alpha)$-GAN-GPs, for CIFAR-10.



(d) (α, α)-GAN-GPs for CIFAR-10.



(e) $(1, \alpha)$-GAN-GPs for Stacked MNIST.



(f) (α, α)-GAN-GPs for Stacked MNIST.

Figure 4.2: Average FID scores vs. epochs for various (α_D, α_G)-GANs.

Variant-SLkGAN-k	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Number of successful trials (/10)
Lk-SLkGAN-0.25	1.15	1.174	6.298×10^{-4}	224.3	940.9	10
Vanilla-SLkGAN-0.25	**1.112**	**1.162**	0.001	237.0	**124.0**	10
Lk-SLkGAN-1.0	1.122	1.167	8.857×10^{-4}	233.0	124.0	10
Vanilla-SLkGAN-1.0	1.126	1.17	9.218×10^{-4}	226.2	1182.844	10
Lk-SLkGAN-2.0	1.148	1.198	5.248×10^{-4}	237.2	288.4	10
Vanilla-SLkGAN-2.0	1.124	1.184	8.933×10^{-4}	237.8	138.4	10
Lk-SLkGAN-7.5	1.455	1.498	$\mathbf{4.422 \times 10^{-4}}$	229.0	322.222	10
Vanilla-SLkGAN-7.5	1.439	1.511	0.001	212.2	1995.067	10
Lk-SLkGAN-15.0	1.733	1.872	0.005	198.8	1885.733	10
Vanilla-SLkGAN-15.0	1.773	1.876	0.005	**171.6**	3122.267	10
DCGAN	1.154	1.208	0.001	231.3	357.122	10

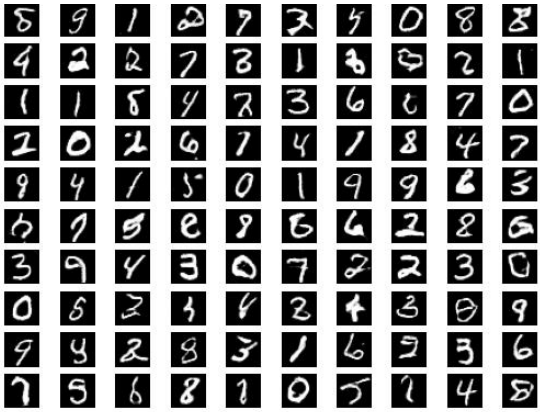
Table 4.4: SLkGAN results for MNIST.

Variant-SLkGAN-k	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Number of successful trials (/5)
Lk-SLkGAN-1.0	4.727	118.242	10914.643	**60.8**	1897.2	5
Vanilla-SLkGAN-1.0	4.821	5.159	**0.092**	88.0	506.5	5
Lk-SLkGAN-2.0	4.723	145.565	7492.26	73.2	3904.2	5
Vanilla-SLkGAN-2.0	**4.58**	**5.1**	0.261	105.4	740.8	5
Lk-SLkGAN-7.5	6.556	155.497	7116.521	254.6	18605.3	5
Vanilla-SLkGAN-7.5	6.384	48.905	8698.195	72.2	1711.7	5
Lk-SLkGAN-15.0	8.576	145.774	5945.097	263.0	36463.0	5
Vanilla-SLkGAN-15.0	7.431	50.868	8753.002	82.6	3106.8	5
DCGAN	4.753	5.194	0.117	88.6	**462.8**	5
Lk-SLkGAN-0.25-GP	17.366	18.974	2.627	87.8	1897.2	5
Vanilla-SLkGAN-0.25-GP	16.013	17.912	1.961	189.0	9487.5	5
Lk-SLkGAN-1.0-GP	10.771	12.567	1.083	77.8	**239.2**	5
Vanilla-SLkGAN-1.0-GP	8.569	9.588	**0.749**	197.6	2690.3	5
Lk-SLkGAN-2.0-GP	23.11	25.013	1.924	**75.4**	658.8	5
Vanilla-SLkGAN-2.0-GP	28.215	29.69	1.242	232.0	20438.5	5
Lk-SLkGAN-7.5-GP	33.304	41.48	49.187	82.8	1081.2	5
Vanilla-SLkGAN-7.5-GP	33.085	34.799	1.597	290.8	12714.7	5
Lk-SLkGAN-15.0-GP	9.157	12.504	3.839	310.4	6976.8	5
Vanilla-SLkGAN-15.0-GP	**7.283**	**8.568**	1.535	185.6	5978.3	5
DCGAN-GP	25.731	28.378	3.398	158.0	2510.5	5

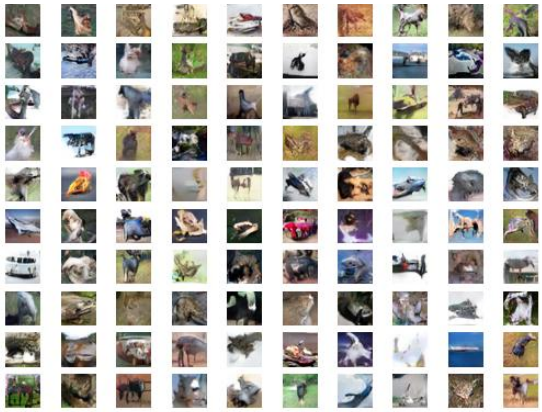
Table 4.5: SLkGAN results for CIFAR-10.

Variant-SLkGAN-k	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Number of successful trials (/5)
Lk-SLkGAN-0.25-GP	10.541	11.824	0.678	113.6	356.3	5
Vanilla-SLkGAN-0.25-GP	5.197	5.197	**0.0**	496.0	**0.0**	1
Lk-SLkGAN-1.0-GP	11.545	12.046	0.291	**89.0**	238.5	5
Vanilla-SLkGAN-1.0-GP	7.475	7.626	0.045	177.0	3528.0	2
Lk-SLkGAN-2.0-GP	10.682	12.782	2.12	180.2	28484.7	5
Vanilla-SLkGAN-2.0-GP	6.023	7.096	0.991	416.667	12244.333	3
Lk-SLkGAN-7.5-GP	8.912	9.906	0.577	239.0	35663.5	5
Vanilla-SLkGAN-7.5-GP	6.074	6.43	0.164	238.0	21729.5	5
Lk-SLkGAN-15.0-GP	4.458	4.74	0.029	253.4	11512.3	5
Vanilla-SLkGAN-15.0-GP	**3.836**	**3.873**	0.002	485.0	354.667	4
DCGAN-GP	7.507	7.774	0.064	303.4	11870.8	5

Table 4.6: SLkGAN results for Stacked MNIST.



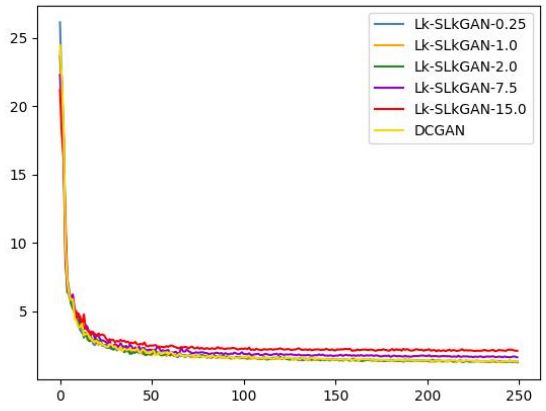
(a) Vanilla-SLkGAN-0.25 for MNIST, FID = 1.112.



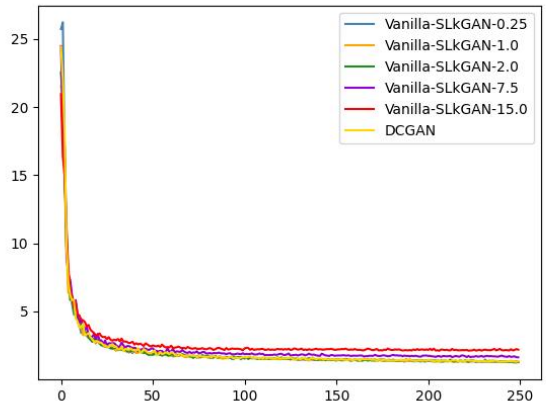
(b) Vanilla-SLkGAN-2.0 for CIFAR-10, FID = 4.58.

(c) Vanilla-SLkGAN-15.0-GP for Stacked MNIST, FID = 3.836.

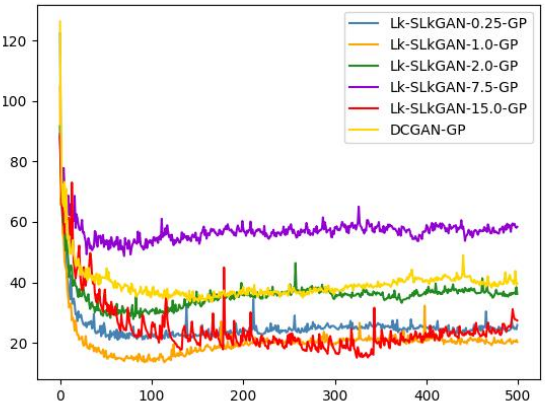
Figure 4.3: Generated images for best-performing SLkGANs.



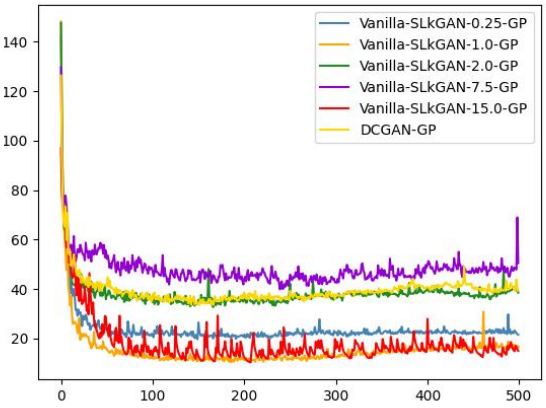
(a) Lk-SLkGANs for MNIST.



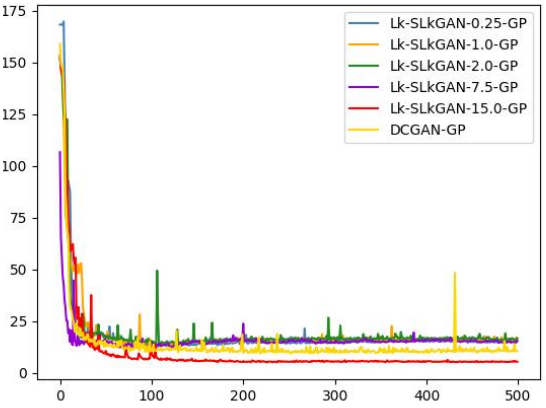
(b) Vanilla-SLkGANs for MNIST.



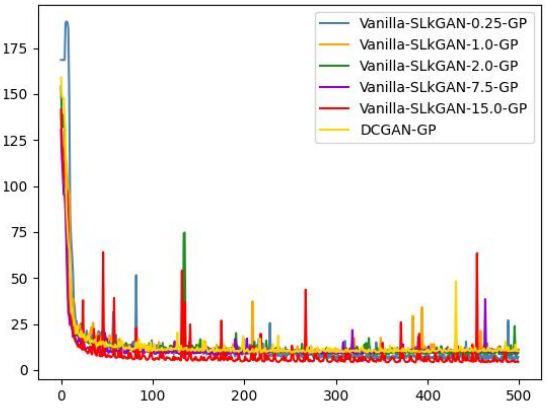
(c) Lk-SLkGAN-GPs for CIFAR-10.



(d) Vanilla-SLkGAN-GPs for CIFAR-10.



(e) Lk-SLkGAN-GPs for Stacked MNIST.



(f) Vanilla-SLkGAN-GPs for Stacked MNIST.

Figure 4.4: FID scores vs. epochs for various SLkGANs.

4.3.3 Discussion

Experiment 1

From Table 4.1, we note that 37 of the 90 trials collapse before 250 epochs have passed without a gradient penalty. The (5,5)-GAN collapses for all 5 trials, and hence it is not displayed in Table 4.1. This behaviour is expected, as the (α, α)-GAN is more sensitive to exploding gradients when α does not tend to 0 or $+\infty$ [20]. The addition of a gradient penalty could mitigate the discriminator's gradients diverging in the (5,5)-GAN by encouraging gradients to have a unit norm. Using a VanillaGAN discriminator with an α-GAN generator (i.e., the (1,α)-GAN) produces better quality images for all tested values of α, compared to when both networks utilize an α-GAN loss function. The (1,10)-GAN achieves excellent stability, converging in all 10 trials, and also achieves the lowest average FID score. The (1,5)-GAN achieves the lowest FID score overall, marginally outperforming DCGAN.

Likewise, for the CIFAR-10 and Stacked MNIST datasets, the (1,α)-GAN produces lower FID scores than the (α, α)-GAN (see Tables 4.2 and 4.3). However, both models are more stable with the CIFAR-10 dataset. With the exception of DCGAN, no model converged to its best FID score for all 5 trials with the Stacked MNIST dataset. Comparing the trials that did converge, both (α, α)-GAN and $(1, \alpha)$-GAN performed better on the Stacked MNIST dataset than the CIFAR-10 dataset. For CIFAR-10, the (1,10)- and (1,20)-GANs produced the best overall FID score and the best average FID score respectively. On the other hand, the (1,0.5)-α-GAN produced the best overall FID score and the best average FID score for the Stacked MNIST dataset. We also observe a tradeoff between speed and performance for the CIFAR-10 and Stacked MNIST datasets: the $(1, \alpha)$-GANs arrive at their lowest FID scores later

than their respective (α, α)-GANs, but achieve lower FID scores overall.

Comparing Figures 4.2c and 4.2d, we observe that the (α, α)-GAN-GP provides more stability than the $(1, \alpha)$-GAN for lower values of α (i.e. $\alpha = 0.5$), while the $(1, \alpha)$-GAN-GP exhibits more stability for higher α values ($\alpha = 10$ and $\alpha = 20$). Figures 4.2e and 4.2f show that the two α-GANs trained on the Stacked MNIST dataset exhibit unstable behaviour earlier into training when $\alpha = 0.5$ or $\alpha = 20$. However, both systems stabilize and converge to their lowest FID scores as training progresses. The (0.5,0.5)-GAN-GP system in particular exhibits wildly erratic behaviour for the first 200 epochs, then finishes training with a stable trajectory that outperforms DCGAN-GP.

A future direction might be explore how the complexity of an image dataset influences the best choice of α. For example, the Stacked MNIST dataset might be considered to be less complex than CIFAR-10, as images in the Stacked MNIST dataset only contain four unique colours (black, red, green, and blue), while the CIFAR-10 dataset utilizes significantly more colours.

Experiment 2

We see from Table 4.4 that all Lk-LkGANs and Vanilla-SLkGANs have FID scores comparable to the DCGAN. When $k = 15$, Vanilla-SLkGAN and Lk-SLkGAN arrive at their lowest FID scores slightly earlier than DCGAN and other SLkGANs.

The addition of a simplified gradient penalty is necessary for Lk-SLkGAN to achieve overall good performance on the CIFAR-10 dataset (see Table 4.5). Interestingly, Vanilla-SLkGAN achieves lower FID scores without a gradient penalty for lower k values ($k = 1, 2$), and with a gradient penalty for higher k values ($k = 7.5, 15$).

When $k = 0.25$, both SLkGANs collapsed for all 5 trials without a gradient penalty. Table 4.6 shows that Vanilla-SLkGANs achieve better FID scores than their respective Lk-LkGAN counterparts. However, Vanilla-SLkGANs are more stable, as no single trial collapsed, while 10 of the 25 Vanilla-SLkGAN trials collapsed before 500 epochs had passed. While all Vanilla-SLkGANs outperform the DCGAN with gradient penalty, Lk-SLkGAN-GP only outperforms DCGAN-GP when $k = 15$. Except for when $k = 7.5$, we observe that the Lk-SLkGAN takes less epochs to arrive at its lowest FID score. Comparing Figures 4.4e and 4.4f, we observe that the Lk-SLkGANs exhibit more stable FID score trajectories than their respective Vanilla-SLkGANs. This makes sense, as the LkGAN loss function aims to increase the GAN's stability compared to DCGAN [5].

Chapter 5

Conclusion

In this thesis, we have introduced $\mathcal{L}_\alpha$-GAN, a dual-objective GAN which uses a parameterized CPE-based generator loss function in tandem with a canonical discriminator loss function that achieves the same optimum as that of VanillaGAN. We proved that the minimax game played by $\mathcal{L}_\alpha$-GAN minimizes a Jensen-f_α-divergence that can be directly derived from the generator's loss function. We applied the $\mathcal{L}_\alpha$-GAN framework to recover VanillaGAN and $(1, \alpha)$-GAN. We proposed a translated version of LkGAN called Shifted-LkGAN (SLkGAN), and showed that the SLkGAN problem also reduced to the minimization of a Jensen-f-divergence. We conducted experiments with the three aforementioned $\mathcal{L}_\alpha$-GANs on three image datasets. The experiments indicated that $(1, \alpha)$-GAN exhibits better performance than (α_D, α)-GAN when $\alpha_D > 1$. They also showed that the devised SLkGAN system achieves lower FID scores and more stability with a VanillaGAN discriminator compared with an LkGAN discriminator.

Future work includes finding more examples of existing GANs that can be recovered by $\mathcal{L}_\alpha$-GAN. One can also try to apply our work to a non-canonical discriminator loss function (e.g., the LkGAN discriminator loss function with $\gamma \neq 1$ and/or $\beta \neq 0$),

which could result in the minimization of a different divergence measure. Additionally, one can attempt to apply the $\mathcal{L}_\alpha$-GAN to a newly-designed CPE loss $\mathcal{L}_\alpha$, derive its generating function for the Jensen-f-divergence, and evaluate the model's performance.

Appendix A

A.1 Neural Network Architectures

We outline the architecures used for the generator and discriminator. For the MNIST dataset, we use the architectures of [5]. For the CIFAR-10 and Stacked MNIST datasets, we base the architectures on [33]. We summarize some aliases for the architectures in Table A.1. For all models we use a batch size of 100 and noise size of 784 for the generator input.

Alias	Definition
FC	Fully Connected
UpConv2D	Deconvolutional Layer
Conv2D	Convolutional Layer
BN	Batch Normalization
LeakyReLU	Leaky Rectified Linear Unit

Table A.1: Summary of aliases used to describe neural network architectures.

We omit the bias in the convolutional and deconvolutional layers to decrease the number of parameters being trained, which in turn decreases computation times. We initialize our kernels using a normal distribution with zero mean and variance 0.01. We present the MNIST architectures in Tables A.2 and A.3, and the CIFAR-10 and Stacked MNIST architectures in Tables A.4 and A.5.

Layer	Output Size	Kernel	Stride	BN	Activation
Input	$28 \times 28 \times 1$	No			
Conv2D	$14 \times 14 \times 64$	5×5	2	No	LeakyReLU(0.3)
Dropout(0.3)				No	
Conv2D	$7 \times 7 \times 128$	5×5	2	No	LeakyReLU(0.3)
Dropout(0.3)				No	
FC	1			No	Sigmoid

Table A.2: Discriminator architecture for the MNIST dataset.

Layer	Output Size	Kernel	Stride	BN	Activation
Input	784				
FC	$7 \times 7 \times 256$				
UpConv2D	$7 \times 7 \times 128$	5×5	1	Yes	LeakyReLU(0.3)
UpConv2D	$14 \times 14 \times 64$	5×5	2	Yes	LeakyReLU(0.3)
UpConv2D	$28 \times 28 \times 1$	5×5	2	No	Tanh

Table A.3: Generator architecture for the MNIST dataset

Layer	Output Size	Kernel	Stride	BN	Activation
Input	$32 \times 32 \times 3$				
Conv2D	$16 \times 16 \times 128$	3×3	2	No	LeakyReLU(0.2)
Conv2D	$8 \times 8 \times 128$	3×3	2	No	LeakyReLU(0.2)
Conv2D	$4 \times 4 \times 256$	3×3	2	No	LeakyReLU(0.2)
Dropout(0.4)				No	
FC	1				Sigmoid

Table A.4: Discriminator architecture for the CIFAR-10 and Stacked MNIST datasets.

Layer	Output Size	Kernel	Stride	BN	Activation
Input	784				
FC	$4 \times 4 \times 256$				
UpConv2D	$8 \times 8 \times 128$	4×4	2	Yes	LeakyReLU(0.2)
UpConv2D	$16 \times 16 \times 128$	4×4	2	Yes	LeakyReLU(0.2)
UpConv2D	$32 \times 32 \times 128$	4×4	2	Yes	LeakyReLU(0.2)
Conv2D	$32 \times 32 \times 3$	3×3	1	No	Tanh

Table A.5: Generator architecture for the CIFAR-10 and Stacked MNIST datasets.

A.2 Algorithms

We outline the algorithms used to train our models in Algorithms 4, 5 and 6.

Algorithm 4 Overview of (α_D, α_G)-GAN training

Require α_D, α_G, Number of epochs n_e, Batch size B, Learning rate η
Initialize Generator G with parameters $\boldsymbol{\theta}_G$, Discriminator D with parameters $\boldsymbol{\theta}_D$.
for $i = 1$ to n_e **do**
Sample batch of real data $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_B\}$ from dataset
Sample batch of Gaussian noise vectors $\mathbf{z} = \{\mathbf{z}_1, ..., \mathbf{z}_B\} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Update the discriminator's parameters using an Adam optimizer with learning rate η by descending the gradient:

$$\nabla_{\boldsymbol{\theta}_D} \left(-\frac{1}{B} \sum_{i=1}^{B} (-\ell_\alpha(1, D(\mathbf{x}_i)) - \ell_\alpha(0, D(G(\mathbf{z}_i)))) \right)$$

or **update** the discriminator's parameters with a simplified GP:

$$\nabla_{\boldsymbol{\theta}_D} \left(-\frac{1}{B} \sum_{i=1}^{B} (-\ell_\alpha(1, D(\mathbf{x}_i)) - \ell_\alpha(0, D(G(\mathbf{z}_i)))) + 5 \left(\sum_{i=1}^{B} \left|\left| \nabla_{\mathbf{x}} \log \left(\frac{D(\mathbf{x})}{1 - D(\mathbf{x})} \right) \right|\right|_2^2 \right) \right)$$

Update the generator's parameters using an Adam optimizer with learning rate η and descending the gradient:

$$\nabla_{\boldsymbol{\theta}_G} \left(\frac{1}{B} \sum_{i=1}^{B} \ell_\alpha(0, D(G(\mathbf{z}_i))) \right)$$

end for

Algorithm 5 Overview of Lk-SLkGAN training

Require k, Number of epochs n_e, Batch size B, Learning rate η
Initialize Generator G with parameters $\boldsymbol{\theta}_G$, Discriminator D with parameters $\boldsymbol{\theta}_D$.
for $i = 1$ to n_e **do**
Sample batch of real data $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_B\}$ from dataset
Sample batch of Gaussian noise vectors $\mathbf{z} = \{\mathbf{z}_1, ..., \mathbf{z}_B\} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Update the discriminator's parameters using an Adam optimizer with learning rate η by descending the gradient:

$$\nabla_{\boldsymbol{\theta}_D} \left(\frac{1}{B} \sum_{i=1}^{B} \left(\frac{1}{2}(D(\mathbf{x}_i) - 1)^2 + \frac{1}{2}(D(G(\mathbf{z}_i))^2) \right) \right)$$

or **update** the discriminator's parameters with a simplified GP:

$$\nabla_{\boldsymbol{\theta}_D} \left(\frac{1}{B} \sum_{i=1}^{B} \left(\frac{1}{2}(D(\mathbf{x}_i) - 1)^2 + \frac{1}{2}(D(G(\mathbf{z}_i))^2) \right) + 5 \left(\sum_{i=1}^{B} \left|\left| \nabla_{\mathbf{x}} \log \left(\frac{D(\mathbf{x})}{1 - D(\mathbf{x})} \right) \right|\right|_2^2 \right) \right)$$

Update the generator's parameters using an Adam optimizer with learning rate η and descending the gradient:

$$\nabla_{\boldsymbol{\theta}_G} \left(\frac{1}{B} \sum_{i=1}^{B} \frac{1}{2}(|1 - D(G(\mathbf{z}_i))|^k - 1) \right)$$

end for

Algorithm 6 Overview of Vanilla-SLkGAN training

Require k, Number of epochs n_e, Batch size B, Learning rate η
Initialize Generator G with parameters $\boldsymbol{\theta}_G$, Discriminator D with parameters $\boldsymbol{\theta}_D$.
for $i = 1$ to n_e **do**
Sample batch of real data $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_B\}$ from dataset
Sample batch of noise vectors $\mathbf{z} = \{\mathbf{z}_1, ..., \mathbf{z}_B\} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Update the discriminator's parameters using an Adam optimizer with learning rate η by descending the gradient:

$$\nabla_{\boldsymbol{\theta}_D} \left(-\frac{1}{B} \sum_{i=1}^{B} \left(\log(D(\mathbf{x}_i)) + \log(1 - D(G(\mathbf{z}_i))) \right) \right)$$

or **update** the discriminator's parameters with a simplified (GP):

$$\nabla_{\boldsymbol{\theta}_D} \left(-\frac{1}{B} \sum_{i=1}^{B} \left(\log(D(\mathbf{x}_i)) + \log(1 - D(G(\mathbf{z}_i))) \right) + 5 \left(\sum_{i=1}^{B} \left|\left| \nabla_{\mathbf{x}} \log \left(\frac{D(\mathbf{x})}{1 - D(\mathbf{x})} \right) \right|\right|_2^2 \right) \right)$$

Update the generator's parameters using an Adam optimizer with learning rate η and descending the gradient:

$$\nabla_{\boldsymbol{\theta}_G} \left(\frac{1}{B} \sum_{i=1}^{B} \frac{1}{2} (|1 - D(G(\mathbf{z}_i))|^k - 1) \right)$$

end for

A.3 Additional Results

Supplementary experimental results for Chapter 4 are herein provided.

Table A.6: (α_D, α_G)-GAN results on MNIST.

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Num. successful trials
(1,0.5)-GAN	1.264	1.288	2.979×10^{-4}	227.25	420.25	4
(0.5,0.5)-GAN	1.209	1.265	0.001	234.5	156.7	6
(1,5)-GAN	**1.125**	1.17	8.195×10^{-4}	230.3	617.344	10
(1,10)-GAN	1.147	**1.165**	7.984×10^{-4}	225.6	253.156	10
(10,10)-GAN	36.506	39.361	16.312	1.5	0.5	2
(1,20)-GAN	1.135	1.174	0.001	237.5	274.278	10
(20,20)-GAN	33.23	33.23	**0.0**	**1.0**	**0.0**	1
DCGAN	1.154	1.208	0.001	231.3	357.122	10
(1,0.5)-GAN-GP	1.695	1.792	0.002	**87.3**	**78.456**	10
(0.5,0.5)-GAN-GP	**1.45**	**1.521**	**0.001**	205.1	611.878	10
(1,5)-GAN-GP	1.683	1.755	0.003	103.5	214.5	10
(5,5)-GAN-GP	2.306	2.472	0.014	118.0	7539.778	10
(1,10)-GAN-GP	1.613	1.761	0.004	93.0	296.222	10
(10,10)-GAN-GP	2.567	2.757	0.015	122.2	6920.4	10
(1,20)-GAN-GP	1.648	1.765	0.006	94.5	126.5	10
(20,20)-GAN-GP	2.788	3.016	0.02	116.3	5105.344	10
DCGAN-GP	1.549	1.724	0.007	92.0	144.667	10

Table A.7: (α_D, α_G)-GAN results on CIFAR-10.

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Num. successful trials
(1,0.5)-GAN	194.343	196.462	8.98	266.0	101250.0	2
(0.5,0.5)-GAN	196.005	196.005	**0.0**	**1.0**	**0.0**	1
(5,5)-GAN	200.504	200.504	**0.0**	369.0	**0.0**	1
(10,10)-GAN	195.905	195.905	**0.0**	497.0	**0.0**	1
DCGAN	**4.753**	**5.194**	0.117	88.6	462.8	5
(1,0.5)-GAN-GP	10.551	14.938	12.272	326.2	1808.7	5
(0.5,0.5)-GAN-GP	13.734	14.93	0.517	223.6	11378.3	5
(1,5)-GAN-GP	10.772	11.635	0.381	132.0	1233.5	5
(5,5)-GAN-GP	20.79	21.72	0.771	**84.8**	1527.2	5
(1,10)-GAN-GP	9.465	**10.187**	**0.199**	182.6	**1096.3**	5
(10,10)-GAN-GP	19.99	21.095	0.434	131.8	13374.7	5
(1,20)-GAN-GP	**8.466**	10.217	1.479	216.2	6479.7	5
(20,20)-GAN-GP	19.378	21.216	2.315	138.2	29824.2	5
DCGAN-GP	25.731	28.378	3.398	158.0	2510.5	5

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Num. successful trials
(1,0.5)-GAN	168.222	171.648	13.309	301.0	42936.667	4
(0.5,0.5)-GAN	168.561	176.991	56.649	251.667	56062.333	3
(1,5)-GAN	168.592	171.94	14.529	174.5	1108.333	4
(5,5)-GAN	168.706	168.716	$\mathbf{2.038 \times 10^{-4}}$	210.0	25088.0	2
(1,10)-GAN	159.114	167.646	42.966	150.0	30932.667	4
(10,10)-GAN	167.686	170.042	15.305	114.0	14344.667	4
(1,20)-GAN	168.296	174.546	36.038	210.667	41464.333	3
(20,20)-GAN	168.24	173.095	17.683	406.0	637.0	3
DCGAN	**125.156**	**130.566**	36.416	**17.8**	**48.7**	5
(1,0.5)-GAN-GP	**4.833**	**4.997**	0.054	311.5	23112.5	2
(0.5,0.5)-GAN-GP	6.418	6.418	**0.0**	479.0	**0.0**	1
(1,5)-GAN-GP	7.98	7.988	1.357×10^{-4}	379.5	11704.5	2
(5,5)-GAN-GP	12.236	12.836	0.301	**91.5**	387.0	4
(1,10)-GAN-GP	7.502	7.528	0.001	326.5	14280.5	2
(10,10)-GAN-GP	14.22	14.573	0.249	95.0	450.0	2
(1,20)-GAN-GP	8.379	8.379	**0.0**	427.0	**0.0**	1
(20,20)-GAN-GP	16.584	16.584	**0.0**	94.0	**0.0**	1
DCGAN-GP	7.507	7.774	0.064	303.4	11870.8	5

Table A.8: (α_D, α_G)-GAN results on Stacked-MNIST.

Table A.9: Lk-SLkGAN results on Stacked-MNIST.

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Num. successful trials
Lk-SLkGAN-0.25	1.15	1.174	6.298×10^{-4}	224.3	940.9	10
Lk-SLkGAN-1.0	**1.122**	**1.167**	8.857×10^{-4}	233.0	**124.0**	10
Lk-SLkGAN-2.0	1.148	1.198	5.248×10^{-4}	237.2	288.4	10
Lk-SLkGAN-7.5	1.455	1.498	$\mathbf{4.422 \times 10^{-4}}$	229.0	322.222	10
Lk-SLkGAN-15.0	1.733	1.872	0.005	**198.8**	1885.733	10
DCGAN	1.154	1.208	0.001	231.3	357.122	10
Lk-SLkGAN-0.25-GP	2.327	2.432	0.005	199.9	1921.878	10
Lk-SLkGAN-1.0-GP	2.448	2.556	0.007	170.5	2178.944	10
Lk-SLkGAN-2.0-GP	2.486	2.585	0.005	165.7	2984.9	10
Lk-SLkGAN-7.5-GP	2.263	2.542	0.026	**55.1**	1793.211	10
Lk-SLkGAN-15.0-GP	**1.203**	**1.278**	**0.001**	128.3	149.789	10
DCGAN-GP	1.549	1.724	0.007	92.0	**144.667**	10

Table A.10: Lk-SLkGAN results on MNIST.

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Num. successful trials
Lk-SLkGAN-1.0	4.727	118.242	10914.643	**60.8**	1897.2	5
Lk-SLkGAN-2.0	**4.723**	145.565	7492.26	73.2	3904.2	5
Lk-SLkGAN-7.5	6.556	155.497	7116.521	254.6	18605.3	5
Lk-SLkGAN-15.0	8.576	145.774	5945.097	263.0	36463.0	5
DCGAN	4.753	**5.194**	**0.117**	88.6	**462.8**	5
Lk-SLkGAN-0.25-GP	17.366	18.974	2.627	87.8	1897.2	5
Lk-SLkGAN-1.0-GP	10.771	12.567	**1.083**	77.8	**239.2**	5
Lk-SLkGAN-2.0-GP	23.11	25.013	1.924	**75.4**	658.8	5
Lk-SLkGAN-7.5-GP	33.304	41.48	49.187	82.8	1081.2	5
Lk-SLkGAN-15.0-GP	**9.157**	**12.504**	3.839	310.4	6976.8	5
DCGAN-GP	25.731	28.378	3.398	158.0	2510.5	5

Table A.11: Lk-SLkGAN results on CIFAR-10.

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Num. successful trials
Lk-SLkGAN-0.25	167.687	168.287	0.176	11.4	90.8	5
Lk-SLkGAN-1.0	166.63	167.842	0.552	8.0	**21.5**	5
Lk-SLkGAN-2.0	167.687	168.095	**0.104**	**6.8**	52.2	5
Lk-SLkGAN-k-7.5	154.015	160.841	48.995	17.8	1288.2	5
Lk-SLkGAN-15.0	**120.855**	151.866	514.982	**6.8**	28.2	5
DCGAN	125.156	**130.566**	36.416	17.8	48.7	5
Lk-SLkGAN-0.25-GP	10.541	11.824	0.678	113.6	356.3	5
Lk-SLkGAN-1.0-GP	11.545	12.046	0.291	**89.0**	**238.5**	5
Lk-SLkGAN-2.0-GP	10.682	12.782	2.12	180.2	28484.7	5
Lk-SLkGAN-7.5-GP	8.912	9.906	0.577	239.0	35663.5	5
Lk-SLkGAN-15.0-GP	**4.458**	**4.74**	**0.029**	253.4	11512.3	5
DCGAN-GP	7.507	7.774	0.064	303.4	11870.8	5

Table A.12: Lk-SLkGAN results on Stacked-MNIST.

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Num. successful trials
LkGAN-vanilla-k-0.25	**1.112**	**1.162**	0.001	237.0	**124.0**	10
Vanilla-SLkGAN-1.0	1.126	1.17	9.218×10^{-4}	226.2	1182.844	10
Vanilla-SLkGAN-2.0	1.124	1.184	$\mathbf{8.933 \times 10^{-4}}$	237.8	138.4	10
Vanilla-SLkGAN-7.5	1.439	1.511	0.001	212.2	1995.067	10
Vanilla-SLkGAN-15.0	1.773	1.876	0.005	**171.6**	3122.267	10
DCGAN	1.154	1.208	0.001	231.3	357.122	10
Vanilla-SLkGAN-0.25-GP	1.676	1.782	0.003	109.1	847.433	10
Vanilla-SLkGAN-1.0-GP	1.668	1.768	0.005	97.1	139.211	10
Vanilla-SLkGAN-2.0-GP	1.665	1.785	0.005	**86.9**	413.211	10
Vanilla-SLkGAN-7.5-GP	1.639	1.705	**0.002**	102.4	**104.489**	10
Vanilla-SLkGAN-15.0-GP	**1.208**	**1.301**	0.003	173.4	1488.489	10
DCGAN-GP	1.549	1.724	0.007	92.0	144.667	10

Table A.13: Vanilla-SLkGAN results on MNIST.

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Number of trials
Vanilla-SLkGAN-1.0	4.821	5.159	**0.092**	88.0	506.5	5
LkGAN-vanilla-k-2.0	**4.58**	**5.1**	0.261	105.4	740.8	5
Vanilla-SLkGAN-7.5	6.384	48.905	8698.195	**72.2**	1711.7	5
Vanilla-SLkGAN-15.0	7.431	50.868	8753.002	82.6	3106.8	5
DCGAN	4.753	5.194	0.117	88.6	**462.8**	5
Vanilla-SLkGAN-0.25-GP	16.013	17.912	1.961	189.0	9487.5	5
Vanilla-SLkGAN-GP	8.569	9.588	**0.749**	197.6	2690.3	5
Vanilla-SLkGAN-GP	28.215	29.69	1.242	232.0	20438.5	5
Vanilla-SLkGAN-7.5-GP	33.085	34.799	1.597	290.8	12714.7	5
Vanilla-SLkGAN-15.0-GP	**7.283**	**8.568**	1.535	185.6	5978.3	5
DCGAN-GP	25.731	28.378	3.398	**158.0**	**2510.5**	5

Table A.14: Vanilla-SLkGAN results on CIFAR-10.

	Best FID score	Average best FID score	Best FID scores variance	Average epoch	Epoch variance	Num. successful trials
Vanilla-SLkGAN-0.25	168.192	168.359	**0.082**	33.667	545.333	3
Vanilla-SLkGAN-1.0	138.779	164.768	237.278	186.6	58425.8	5
Vanilla-SLkGAN-2.0	153.149	167.618	115.968	237.8	55015.2	5
Vanilla-SLkGAN-7.5	**95.594**	150.473	968.235	144.4	32835.3	5
Vanilla-SLkGAN-15.0	158.273	163.027	10.579	**4.0**	**38.0**	5
DCGAN	125.156	**130.566**	36.416	17.8	48.7	5
Vanilla-SLkGAN-0.25-GP	5.197	5.197	**0.0**	496.0	**0.0**	1
Vanilla-SLkGAN-1.0-GP	7.475	7.626	0.045	**177.0**	3528.0	2
Vanilla-SLkGAN-2.0-GP	6.023	7.096	0.991	416.667	12244.333	3
Vanilla-SLkGAN-7.5-GP	6.074	6.43	0.164	238.0	21729.5	5
Vanilla-SLkGAN-15.0-GP	**3.836**	**3.873**	0.002	485.0	354.667	4
DCGAN-GP	7.507	7.774	0.064	303.4	11870.8	5

Bibliography

[1] Fady Alajaji and Po-Ning Chen. *An Introduction to Single-User Information Theory*. Springer Nature, Singapore, 2018.

[2] Suguru Arimoto. Information-theoretical considerations on estimation problems. *Information and Control*, 19(3):181–194, 1971.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings International Conference on Machine Learning*, pages 214–223. PMLR, 2017.

[4] Himesh Bhatia. Generalized Loss Functions for Generative Adversarial Networks. Master's thesis, Queen's University, Canada, 2020.

[5] Himesh Bhatia, William Paul, Fady Alajaji, Bahman Gharesifard, and Philippe Burlina. Least kth-order and Rényi generative adversarial networks. *Neural Computation*, 33(9):2473–2510, 2021.

[6] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon,

and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[7] C.. Chire. *KMeans-Gaussian-data.svg*. Wikimedia Commons, Oct 2011.

[8] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[9] Rafael C Gonzalez and Richard E Woods. *Digital Image Processing*. Pearson, Upper Saddle River, NJ, 4 edition, March 2017.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[12] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[13] Adam Gronowski, William Paul, Fady Alajaji, Bahman Gharesifard, and Philippe Burlina. Classification utility, fairness, and compactness via tunable information bottleneck and Rényi measures. *ArXiv preprint: 2206.10043v2*, 2023.

[14] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. 30, 2017.

[15] Ernst Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik*, 1909(136):210–271, 1909.

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

[17] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[18] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (Canadian Institute for Advanced Research).

[19] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[20] Gowtham R Kurri, Tyler Sypherd, and Lalitha Sankar. Realizing GANs via a tunable loss function. In *Proceedings IEEE Information Theory Workshop (ITW)*, pages 1–6. IEEE, 2021.

[21] Gowtham R Kurri, Monica Welfert, Tyler Sypherd, and Lalitha Sankar. α-GAN: Convergence and estimation guarantees. In *Proceedings IEEE International Symposium on Information Theory (ISIT)*, pages 276–281. IEEE, 2022.

[22] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[23] Friedrich Liese and Igor Vajda. On divergences and informations in statistics and information theory. *IEEE Transactions on Information Theory*, 52(10):4394–4412, 2006.

[24] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. PacGAN: The power of two samples in generative adversarial networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[25] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[26] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. 2017.

[27] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[28] Frank Nielsen. On a generalization of the Jensen-Shannon divergence and the Jensen-Shannon centroid. *Entropy*, 22(2):221, 2020.

[29] Frank Nielsen and Richard Nock. On the chi square and higher-order chi distances for approximating f-divergences. *IEEE Signal Processing Letters*, 21(1):10–13, 2013.

[30] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2018.

[31] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. *Advances in Neural Information Processing Systems*, 29, 2016.

[32] Ferdinand Österreicher. On a class of perimeter-type distances of probability distributions. *Kybernetika*, 32(4):389–393, 1996.

[33] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of the 9th International Conference on Image and Graphics*, pages 97–108, 2017.

[34] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, volume 4, pages 547–562. University of California Press, 1961.

[35] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[37] Robert Sanders. The Pareto principle: its use and abuse. *Journal of Services Marketing*, 1(2):37–40, 1987.

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

[39] Justin Veiner, Fady Alajaji, and Bahman Gharesifard. A unifying generator loss function for generative adversarial networks. *ArXiv preprint: 2308.07233*, 2023.

[40] Sergio Verdú. α-mutual information. In *Proceedings 2015 Information Theory and Applications Workshop (ITA)*, pages 1–6. IEEE, 2015.

[41] Monica Welfert, Kyle Otstot, Gowtham R Kurri, and Lalitha Sankar. (α_D, α_G)-GANs: Addressing GAN training instabilities via dual Objectives. In *Proceeedings IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2023.

[42] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *ArXiv preprint: 1505.00853*, 2015.

[43] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 7354–7363. PMLR, 09–15 Jun 2019.