DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2017

On Depth and Complexity of Generative Adversarial Networks

HIROYUKI VINCENT YAMAZAKI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

KTH Computer Science
and Communication

On Depth and Complexity of Generative Adversarial Networks

HIROYUKI VINCENT YAMAZAKI

Master's Thesis at CSC
Supervisors: Yamaguchi Takahira, Arvind Kumar
Examiner: Örjan Ekeberg

Abstract

Although generative adversarial networks (GANs) have achieved state-of-the-art results in generating realistic looking images, they are often parameterized by neural networks with relatively few learnable weights compared to those that are used for discriminative tasks. We argue that this is suboptimal in a generative setting where data is often entangled in high dimensional space and models are expected to benefit from high expressive power. Additionally, in a generative setting, a model often needs to extrapolate missing information from low dimensional latent space when generating data samples while in a typical discriminative task, the model only needs to extract lower dimensional features from high dimensional space. We evaluate different architectures for GANs with varying model capacities using shortcut connections in order to study the impacts of the capacity on training stability and sample quality. We show that while training tends to oscillate and not benefit from additional capacity of naively stacked layers, GANs are capable of generating samples with higher quality, specifically for images, samples of higher visual fidelity given proper regularization and careful balancing.

Referat

Djup och komplexitet hos generativa motstridanade nätverk

Trots att Generative Adversarial Networks (GAN) har lyckats generera realistiska bilder består de än idag av neurala nätverk som är parametriserade med relativt få tränbara vikter jämfört med neurala nätverk som används för klassificering. Vi tror att en sådan modell är suboptimal vad gäller generering av högdimensionell och komplicerad data och anser att modeller med högre kapaciteter bör ge bättre estimeringar. Dessutom, i en generativ uppgift så förväntas en modell kunna extrapolera information från lägre till högre dimensioner medan i en klassificeringsuppgift så behöver modellen endast att extrahera lågdimensionell information från högdimensionell data. Vi evaluerar ett flertal GAN med varierande kapaciteter genom att använda shortcut connections för att studera hur kapaciteten påverkar träningsstabiliteten, samt kvaliteten av de genererade datapunkterna. Resultaten visar att träningen blir mindre stabil för modeller som fått högre kapaciteter genom naivt tillsatta lager men visar samtidigt att datapunkternas kvaliteter kan öka, specifikt för bilder, bilder med hög visuell fidelitet. Detta åstadkoms med hjälp utav regularisering och noggrann balansering.

Contents

Chapter 1

Introduction

Machine learning is the science of creating algorithms and programs without having humans explicitly defining each instruction. In this context, *create* can be defined as training a parameterized model based on training data collected from the real world, usually by updating the parameters in the negative directions of the gradients of some error surface given the observed data. With machine learning, we can model the world around us by assuming and exploiting statistical regularities purely from observations.

It is for instance reasonable to assume that images of handwritten digits follow certain statistical patterns, such as consisting of strokes of similar widths against a background with a single or few colors with flat texture. At the same time, while images of the same digits are expected to look somewhat similar, images of different digits should be distinguishable. By collecting images of handwritten digits together with corresponding target labels telling which digit each image represents, we can train a model to extract certain features from the images that are characteristic for each digit to correctly classify them. The task of optimizing the parameters of a model to correctly classify images as just described is an example of supervised learning and is a typical discriminative machine learning task. There are other machine learning tasks including semi-supervised learning where only some of the data samples have associated labels and unsupervised learning where there are no target labels at all. In fact, inferring not limited to but including digits from handwritten images with todays' algorithms can be performed with almost perfect accuracy and is considered more or less a solved task because of recent advances in optimization techniques, computational power and largely available training data [8, 33].

On the other hand, in order to work with high-dimensional data such as images, we should not limit ourselves to discriminating them. Another set of tasks in machine learning to consider is generative modeling which in recent years have seen vast progress with the introduction of the Variational Autoencoder (VAE) and the Generative Adversarial Network (GAN), the latter GAN being the technique we in this work choose to focus on [9, 18]. With generative modeling we usually want to learn the underlying data distributions that the data is representing. This could

be the distribution of images of handwritten digits over the intensities in the pixel space. By doing so, we could estimate probability distributions of these images using a trainable parameterized function. If this function allows for sampling which is the case for VAEs and GANs, previously unseen data samples can be generated. In other words, completely new images that resemble images from the training data can be generated. Additionally, by conditioning the functions on different inputs, one could train a generative model to perform denoising, fill in missing information or project data to different modalities by e.g. mapping 2-dimensional images to 3-dimensional space or transforming objects in images [4, 22, 26]. Although dependent on the problem instance, this machine learning task could be considered more difficult than inferring labels as in the discriminative setting. Studies in generative models are motivated by all these applications to mention a few but is again, on a more general level fundamental to understanding high-dimensional data and tests our abilities to manipulate them.

1.1 Problem

Generative modeling takes an important step towards understanding high dimensional data that is difficult for us humans to interpret. Specifically for GANs, although successfully being applied to image data, the training algorithm is known to suffer from instabilities which has lead to proposals of various architectures, heuristics and alternative training objectives alleviating these drawabacks [2, 3, 26, 38]. However, these models are often relatively small in terms of number of parameters compared to neural networks used for other discriminative classification tasks. We argue that this is suboptimal in the setting of modeling real image distributions and that GANs are capable of generating images of higher visual fidelity given more capacities. The problem lies in the fact that the modest capacities of the networks could be hindering the generative models to fully estimate complex and high dimensional data distributions. This could also be one of the reasons why GANs have yet scaled to complex datasets with high diversity over image space, such as CIFAR-10 or ImageNet.

1.2 Objective

The aim of this report is to evaluate the importance of model complexity in the standard GAN setting with a generator and a discriminator network, in terms of training stability, convergence as well as quality of generated samples. We attempt to train larger models inspired by various techniques suggested in the community of deep leraning to generate images of higher visual fidelity. Concretely, we train deeper GANs with residual blocks inspired by the residual network [12]. Models with different capacities are compared by varying the number of layers and inter-layer connections.

1.3 Contribution and Delimitations

The main contribution of this work is an evaluation and an comparison based on the Inception score [30] for GANs with different capacities. Instead of proposing a new architecture, based on our findings, we intend to aid designing future non-trivial architectures for GANs. It is important to note that this evaluation may yield different results if the objective functions is changed from the standard GAN setting. In this report, we choose to study the basic model described in the original paper by Goodfellow et al. [9].

1.4 Outline of Thesis

This report is organized as follows. Chapter 2 explains the background that is relevant to this work starting from probability theory and neural networks before extending to convolutional neural networks (CNNs) and continuing to presenting two methods for generative modeling using neural networks. The last section introduces the GAN. Chapter 3 lists related work, such as a concrete implementation of the GAN, namely the Deep Convolutional Generative Adversarial Network (DCGAN) and techniques for training deeper neural networks. Chapter 4 describes the experimental setups and is followed by Chapter 5 that presents the results. Chapter 6 discusses the results and attempts to analyze the different experimental outcomes. Chapter 7 concludes this work by summarizing the previous chapters.

1.5 Ethical Considerations

Machine learning or artificial intelligence (AI) as it often is referred to has seen enormous progress in recent years. As of today, existing techniques are being improved and new methods are being proposed which simultaneously widens the possible applications for machine learning. Andrew Ng, a professor at Stanford University who by MIT Technology Review TR35 in 2008 was named one of the top innovators in the world states that AI will become the new electricity. Machine learning and AI might in other words change the way we live. Additionally, we may reasonably assume that machine learning will be implemented in many software and hardware solutions just as electricity is today as we are already seeing applications in healthcare for detecting cancer, automated manufacturing and autonomous driving to mention a few. In factories and server clusters, machine learning solutions are able to through experience based data improve cooling efficiencies and optimize for energy usage which would lead to a more sustainable environment as well. While these applications hold high potential, they are not perfectly accurate at all times. For instance, autonomous cars relies on discriminative models that through visual input data from cameras decide which action to take next. An adversarial example of say a shirt that to us humans looks like any shirt may to an autonomous car look like a road or a beach [10, 20]. Such vulnerabilities of these discriminative models

can be exploited to cause harm. We hope that better understandings of generative models will in addition to giving us new perspectives on complex high dimensional data, contribute to addressing weaknesses in current classifiers.

Chapter 2

Background

2.1 Information Theory

In generative modeling and specifically in the context of GANs, we are interested in estimating the underlying data probability P_{data} from which the training data is expected to be sampled from, using a distribution P_{g} which is parameterized by a neural network. By minimizing the divergence between the two distributions, the neural network is trained to generate samples that are indistinguishable from the training data. More concretely, we are approximately minimizing the Kullback-Leibler divergence and the Jensen-Shannon divergence, which are two divergences among a family of f-divergences. This section explains the basic information theory required in order to understand how these probability divergences are measured.

2.1.1 Information Content

In information theory, the self-information or the Shannon information content $\mathrm{I}(X)$ associated with an random variable X with probability $P(X)$ measures the amount of information associated with the event.

$$\mathrm{I(X)} = -\log P(X) \tag{2.1}$$

Since $0 \leq P(X) \leq 1$, the self-information is non-negative and reaches its minimum value 0 when $P(X = x) = 1$, i.e. when an event is guaranteed to occur. Then, the self-information is 0. For an event with $P(X) < 1$, the information content is $\mathrm{I}(X) > 0$ and the less likely the event, the more information it contains. If this log is in base 2, the measuring unit is bits. For instance, an outcome of a fair coin flip contains $\mathrm{I}(X = \text{tail}) = -\log 0.5 = 1$ bit of information.

Often, we are interested in the expected, or the average information content which is called the entropy. The entropy $\mathrm{H}(X)$ of a random variable X with prob-

ability $P(X)$ is defined as follows.

$$\begin{aligned} \mathrm{H}(X) &= \mathbb{E}_X[\mathrm{I}(X)] \\ &= \mathbb{E}_X[-\log P(X)] \\ &= -\sum_{x \in X} p(x) \log p(x) \end{aligned} \tag{2.2}$$

Where $p(x)$ is the probability mass function evaluated at x. It can be interpreted as the mean of the self-information weighted by the probabilities of each possible outcome. If the probability distribution is continuous, the sum is simply replaced by an integral as in Equation 2.3. In the following explanations, we will stick to the discrete domain for simplicity, but the reasoning is analogous.

$$-\int_{-\infty}^{\infty} p(x) \log p(x) dx \tag{2.3}$$

2.1.2 Measuring Probability Distribution Divergences

Although there are several different f-divergences, we choose to focus on two of them, namely the Kullback-Leibler divergence and the Jensen-Shannon divergence since these are the divergences being minimized in the GAN setting. It is important to note that a divergence is not a true distance metric, i.e. the divergence from P to Q is not necessarily equal to the divergence from Q to P.

Kullback-Leibler Divergence

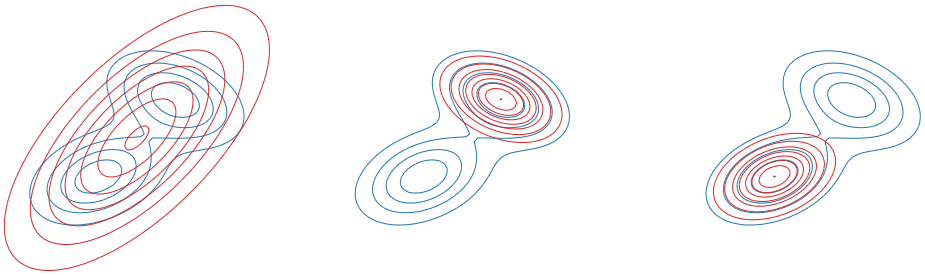
The Kullback-Leibler divergence D_{KL} is a non-symmetric measure of divergence between two probability distributions P and Q. It is non-negative and reaches its minimum value 0 if and only if P is identical to Q. It is therefore suited as a measure for measuring divergences between probability distributions. By minimizing the Kullback-Leibler divergence between these two distributions, we can essentially estimate P using Q.

$$\begin{aligned} D_{\mathrm{KL}}(P \mid\mid Q) &= \mathrm{H}(P, Q) - \mathrm{H}(X) \\ &= -\sum_{x \in X} p(x) \log q(x) + \sum_{x \in X} p(x) \log p(x) \\ &= \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} \\ &= \mathbb{E}_{x \sim P(X)}[\log P(X) - \log Q(X)] \end{aligned} \tag{2.4}$$

Here, $\mathrm{H}(P, Q)$ is the cross-entropy of P and Q which looks a lot like the entropy but instead represents the average number of bits required to identify the outcome of the random variable X using a probability Q.

Implications of the fact that this divergence is non-symmetric arises as shown in Figure 2.1. Minimizing $D_{\mathrm{KL}}(P \mid\mid Q)$ as in Equation 2.4 is called the forward

Kullback-Leibler divergence and this divergence becomes infinite if for any $p(x) > 0$, $q(x) = 0$. Therefore Q tends to cover not only all of the support of P but too much. Conversely, the reverse divergence $D_{\mathrm{KL}}(Q \parallel P)$ becomes infinite if for any $q(x) > 0$, $p(x) = 0$. This often leads to Q underestimating the support of P.



(a) Forward $D_{\mathrm{KL}}(P \parallel Q)$ (b) Reverse $D_{\mathrm{KL}}(Q \parallel P)$ (c) Reverse $D_{\mathrm{KL}}(Q \parallel P)$

Figure 2.1: Minimizing different Kullback-Leibler divergences between the bivariate bimodal Gaussian distribution P represented by the blue contours and the estimate Q represented by the red contours. Plots are based on [25].

Jensen-Shannon Divergence

The Jensen-Shannon divergence D_{JS} is similar to the former divergence but is in contrast symmetric. The square root of the Jensen-Shannon divergence is a proper distance metric. This divergence is well-behaved and does not become infinite where P and Q have non-overlapping supports.

$$\begin{aligned} D_{\mathrm{JS}}(P \parallel Q) &= \frac{1}{2} D_{\mathrm{KL}}(P \parallel M) + \frac{1}{2} D_{\mathrm{KL}}(Q \parallel M) \\ M &= \frac{1}{2}(P + Q) \end{aligned} \tag{2.5}$$

2.2 Neural Networks

Neural networks, or artificial neural networks are mathematical models that partially build upon previously mentioned concepts from information theory. They are widely adopted in machine learning, including generative settings, to approximate functions by extracting statistical regularities from data. They are parameterized functions with initially random parameters that are iteratively optimized towards local or ideally global minima of some error surface. This is referred to as *training* a neural network and is further explained in 2.2.1.

A neural network is a combination of connected artificial neurons, similar to the biological neurons in the human brain. Hence the name. Figure 2.2 represents such a neuron in its simplest form.

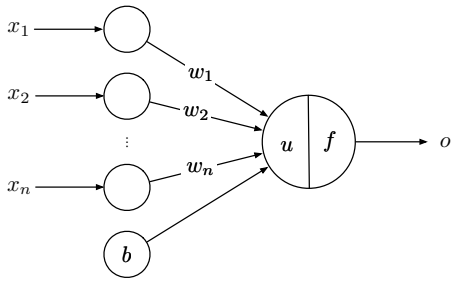


Figure 2.2: A single neuron with n inputs, n learnable weights and a single bias term. In this example, the input is a vector but the dimensions of the input can be generalized to tensors of higher dimensions.

Each neuron is responsible for computing a weighted sum u of the given inputs $x = \{x_1, x_2, .., x_n\}^{\mathrm{T}}$ by an affine transformation. The parameters $w_1, w_2, .., w_n$ are therefore referred to as weights. The bias term b lets the neuron not only scale but also shift its entire output.

$$u = \sum_{i=1}^{n} x_i w_i + b \tag{2.6}$$

A function f then takes this weighted sum and applies a transformation. The transformed weighted sum becomes the final output o of this neuron. f is often referred to as the activation functions since it resembles the biological activations of a neuron in the human brain.

$$o = f(u) \tag{2.7}$$

By connecting multiple neurons in layers using non-linear activation functions, these neural networks can learn complex mappings to e.g. classify non-linearly separable data. In particular, a neural network with at least three layers including the input and the output layers with non-linear sigmoidal activations functions can in theory approximate any function according to the Universal Approximation Theorem (Section 3.3). The operation of mapping inputs x to outputs o using a neural network is called a forward pass and a neural network as just described is often referred to as a multi-layer perceptron [16, 29, 34]. Figure 2.3 illustrates such a neural network.

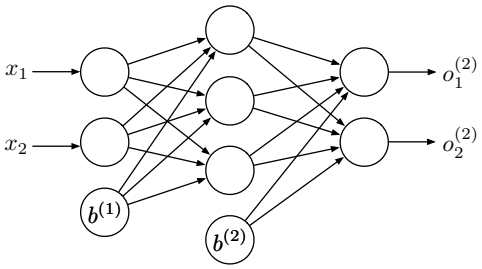


Figure 2.3: A three-layered multi-layer perceptron with two inputs, three hidden neurons and two outputs where each layer uses a shifting bias.

In the Figure 2.3, $b^{(l)}$ represents the bias in the l-th layer and is used in each layer except in the input layer. In the following sections, the superscript (l) will be used to refer to the l-th layer in the model. Additionally, L will refer to the last layer of a neural network. The input layer can be thought of as the 0-th layer and does not depend on any bias. Note that the bias is a parameter that is optimized along with the weights during training.

2.2.1 Optimization with Gradient Descent

The initially random weights and biases need to be updated such that some predefined error function, or loss denoted $\mathcal{L}$ is minimized. Gradient descent is an optimization algorithm that by using a differentiable loss function tries to find a minimum by iteratively updating the parameters in the negative direction of the derivative of the loss landscape.

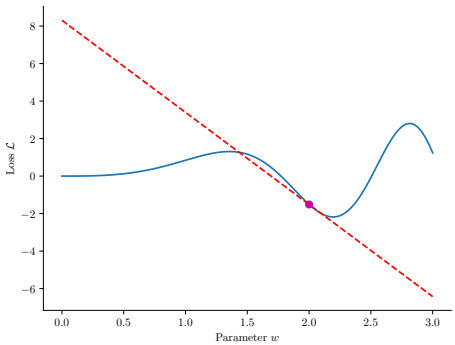


Figure 2.4: The loss on the y-axis with the dashed line illustrating the gradient at the point where $w = 2.0$. In this example, the loss would be minimized by slightly increasing w.

Assume that for an input $x \in X$, that the target value that the network is expected to output is y. The loss $\mathcal{L}$ can be defined as follows.

$$\mathcal{L} = \frac{1}{2}||y - o^{(L)}||_2^2 \tag{2.8}$$

This particular loss functions is called the Mean Squared Error (MSE) and is a concave quadratic function with a global minimum when $o^{(L)} = y$, i.e. when the output of the last layer L equals the target value. Note that the MSE is everywhere differentiable with respect to $o^{(L)}$, is non-negative and becomes 0 only at its global minimum. Since this loss depends on $o^{(L)}$ which in turn depends on the weights and biases, by iteratively taking the partial derivatives of the loss with respect to each parameter and then updating them in the negative direction, the loss can be minimized. The partial derivatives are computed using the backpropagation algorithm which the rest of this section explains. Backpropagation essentially computes the errors at the output layer based on $\mathcal{L}$, then propagates these errors to earlier layers using the chain rule.

$$w_{ji,t+1}^{(l)} = w_{ji,t}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial w_{ji,t}^{(l)}} \tag{2.9}$$

$$b_{j,t+1}^{(l)} = b_{j,t}^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial b_{j,t}^{(l)}} \tag{2.10}$$

Here, $w_{ji,t}^{(l)}$ denotes the weight from the i-th neuron in layer $l-1$ to the j-th neuron in layer l at time t. The α in the equation above is called the learning rate and decides the fraction of the derivative of which each update should take into account. If α is too large, the training tend to oscillate or diverge while a too small α slows down convergence and might cause the algorithm to get stuck in a bad local minimum hindering it from fully exploring the parameter space.

The partial derivatives with respect to the parameters in the last layer L is easily computed assuming that the activation function is differentiable. We omit the t since it is fixed in the following equations.

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(L)}} = \delta_j^{(L)} o_i^{(L-1)} \tag{2.11}$$

$$\delta_j^{(L)} = (o_j^{(L)} - y_j) \frac{\partial o_j^{(L)}}{\partial u_j^{(L)}} \tag{2.12}$$

Where $\delta_j^{(L)}$ depends on the partial derivative of the loss functions, in this case the MSE, and is the local error at the j-th neuron in the last layer. For a weight $w_{ji}^{(l)}$ where $l < L$, the partial derivative becomes as follows.

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} o_i^{(l-1)} \tag{2.13}$$

$$\delta_j^{(l)} = \sum_k \left[w_{kj}^{(l)} \delta_k^{(l+1)} \right] \frac{\partial o_j^{(l)}}{\partial u_j^{(l)}} \tag{2.14}$$

Here, k denotes the index of a neuron in the $(l+1)$-th layer and the sum is over contributed errors to the next layer weighted by the weights. Note that if $l = 1$ that $o^{(l-1)}$ in Equation 2.13 becomes the input.

The bias updates are the δ elements themselves and this holds true also for the last layer L.

$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \delta_j^{(l)} \tag{2.15}$$

Updating the parameters after each training sample x is called online training and usually causes high oscillation. In contrast, updating the parameters after averaging the partial derivatives over the whole training set is called batch training. Batch training often converges to nearby local minima from its initial position and tends to yield suboptimal results. Often in practice, parameters are updated by computing partial derivatives based on randomly sampled minibatches of data samples, often ranging in size from 32 to 512 depending on the problem and hardware limitations. This is called minibatch training and provides a fair trade-off between online and batch training.

2.2.2 Non-Linearities

A neural network, no matter how many layers it consists of, can be represented by an equivalent single-layer neural network if the activation functions are the identity function $f(x) = x$. Let us for simplicity omit the bias term and let $W^{(l)}$ be the weight matrix of layer l. This way, the activations of all neurons in the l-th layer can be expressed by a single matrix multiplication.

$$\begin{aligned} o^{(l)} &= f^{(l)}(u^{(l)}) = u^{(l)} \\ u^{(l)} &= W^{(l)} o^{(l-1)} \end{aligned} \tag{2.16}$$

And the activations of the next layer becomes as follows.

$$\begin{aligned} o^{(l+1)} &= u^{(l+1)} \\ u^{(l+1)} &= W^{(l+1)} o^{(l)} \\ &= W^{(l+1)} W^{(l)} o^{(l-1)} \end{aligned} \tag{2.17}$$

Subsequent transformations through M layers can be expressed with a single matrix multiplication with a weight matrix $\hat{W}$ that is the product of all previous weight matrices.

$$\begin{aligned} o^{(l+M)} &= \left(\prod_{m=0}^{M} W^{(l+m)} \right) o^{(l-1)} \\ &= \hat{W} o^{(l-1)} \end{aligned} \tag{2.18}$$

This limits the expressive power of the neural network to linear models. The bias term does not change this property. To learn more sophisticated non-linear functions, we must benefit from additional layers using non-linear activation functions. The following sections describe non-linear transformations that are often found in literature and specifically in the contexts of GANs.

Sigmoid

The sigmoid function has the domain of all real numbers and the range [0, 1]. It is a monotonically increasing function.

$$f_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-x}} \tag{2.19}$$

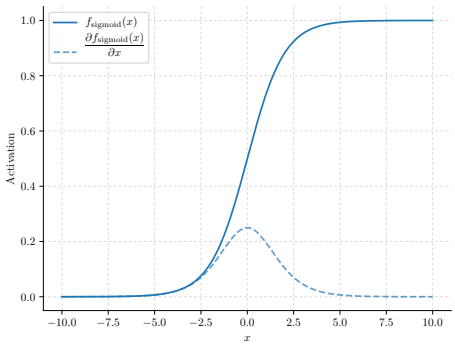


Figure 2.5: The sigmoid activation function.

This function can be intuitively thought of as a biological neuron firing or not. However, it suffers from the saturated regions near activations of 0 and 1 where the gradients are almost 0. This means that sigmoids may during backpropagation cause the gradient to vanish, preventing any earlier layers from receiving meaningful updates. The sigmoid function can still be used to output probabilities where the range is expected to be constrained within the bounds of [0, 1].

Tanh

The tanh, or the hyperbolic tangent function has similar characteristics to the sigmoid function but has a larger range of [-1, 1] and is symmetric about the origin.

$$f_{\tanh}(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.20}$$

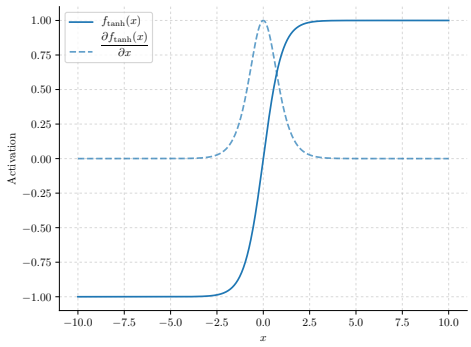


Figure 2.6: The tanh activation function.

This latter property is favorable since this means that outputs may have different signs depending on x and are on average closer to 0. This was shown to converge faster [21]. Additionally, the tanh function provides larger gradients than the sigmoid function.

ReLU

The ReLU, or the Rectified Linear Unit is simply a linear function in the positive domain but is thresholded at 0 for all $x < 0$. This means that the gradient is either 1 or 0. It is computationally efficient and preferable over many other activation functions. Additionally, it does not suffer from saturated regions which has proven to improve the performance in terms of convergence rate and accuracy in discriminative settings [19]. On the other hand, a ReLU activation may cause a neuron to never fire if it is pushed too far in the negative region during an update. This would mean that the neuron always outputs 0 with a constant gradient of 0 that would not allow the neuron to recover.

$$f_{\text{ReLU}}(x) = \max(0, x) \tag{2.21}$$

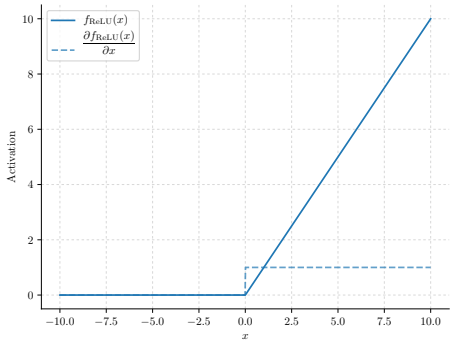


Figure 2.7: The ReLU activation function.

Leaky ReLU

The leaky ReLU is similar the ReLU activation function but prevent a neuron from never firing by allowing gradients to propagate in all of its domain ensuring that weights are continuously updated.

$$f_{\text{LReLU}}(x) = \max(x, ax) \tag{2.22}$$

$a \leq 1$ is a scalar that defines the steepness of the slope in the domain $x < 0$.

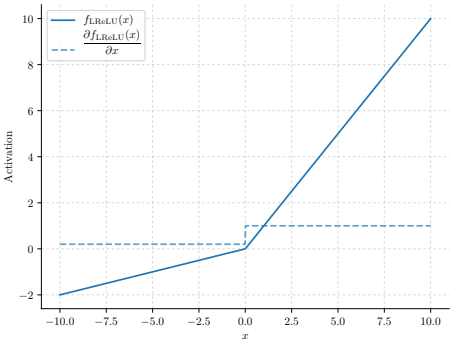


Figure 2.8: The leaky ReLU (LReLU) activation function with $a = 0.2$.

2.3 Convolutional Neural Networks

The sections up until now have assumed that all neurons in layer l are connected to all neurons in layer $l + 1$. These types of connections are called fully connected, dense or linear and can be thought of as the most basic form of neural connections.

We will in the rest of this work refer to these layers as linear layers. Convolutional neural networks (CNNs) on the other hand are neural networks with different neural connections operating on discrete grids of data. CNNs has been successfully applied to image data since images can be represented by grids of pixels, each pixel representing the color intensity at that particular position. The handwritten digit recognition task is one such example. If the image is a grayscale image, we have one such grid and if the image is a colored RGB image, we have one grid per channel and a total of three. CNNs are in often preferable over linear layers with high dimensional data with spatial correlations. This is due to the fact that convolutional layers often require less parameters than linear layers and are more efficient in terms of memory. Additionally, convolutions take the spatial information into account using the convolution operation, i.e. by sliding kernels over the input data. Since computations can be parallelized, it is also computationally efficient using GPUs. However most importantly, the convolutional operation is translation invariant since the same kernel is applied to all position in the input. This is in contrast to the linear layer which would result in completely different outputs if the inputs were to be shifted. This is another reason why convolutions have been successful with image data.

2.3.1 Convolution

We explain the convolution operation using a concrete example. Given an image with a single channel with 3×3 pixel dimensions and a set of learnable weights, a kernel with dimensions 2×2, convolving the kernel over the input yields the following output.

Input

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$

Kernel

$w_{1,1}$	$w_{1,2}$
$w_{2,1}$	$w_{2,2}$

Feature map

$\sum_{i=1}^{2}\sum_{j=1}^{2} w_{i,j}x_{i,j}$	$\sum_{i=1}^{2}\sum_{j=1}^{2} w_{i,j}x_{i,j+1}$
$\sum_{i=1}^{2}\sum_{j=1}^{2} w_{i,j}x_{i+1,j}$	$\sum_{i=1}^{2}\sum_{j=1}^{2} w_{i,j}x_{i+1,j+1}$

The output of a convolutional layer is called a feature map and is computed by sliding the kernel over the input. Note that the kernel has 4 unique position on the input image, which is why the output feature map has a size of 4. The kernel is often relatively small as in this example, meaning that the number of parameters are mostly determined by the number of input and output feature maps. In this example, we also used a stride of 1 and padding of 0. The stride decides how many

steps the kernel is moved before computing each weighted sum. A larger stride yields a smaller feature map. The padding decides the number of pixels that are padded to the input before convolution, usually with the value 0. A larger padding yields a larger feature map. Also, if the input consist of multiple feature maps, the same number of kernels are prepared. Each kernel is responsible for computing the feature map corresponding to its own input feature map. When all output feature maps are computed, they are added elementwise resulting in a single feature map. Each kernel, or set of kernels in the case of inputs with multiple feature maps can be seen as one single neuron in the linear case.

The above convolution operation can similarly to the linear connection be expressed with a single matrix multiplication by flattening the input to a 1-dimensional vector. The output feature map is also flattened so by reshaping it to the appropriate size, the same operation as above can be achieved with the following multiplication.

$$\begin{pmatrix} w_{1,1} & w_{1,2} & 0 & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & w_{1,1} & w_{1,2} & 0 & w_{2,1} & w_{2,2} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{1,1} & w_{1,2} & 0 & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & w_{1,1} & w_{1,2} & 0 & w_{2,1} & w_{2,2} \end{pmatrix} \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ x_{2,1} \\ x_{2,2} \\ x_{2,3} \\ x_{3,1} \\ x_{3,2} \\ x_{3,3} \end{pmatrix} \tag{2.23}$$
$$= \begin{pmatrix} w_{1,1}x_{1,1} + w_{1,2}x_{1,2} + w_{2,1}x_{2,1} + w_{2,1}x_{2,2} \\ w_{1,1}x_{1,2} + w_{1,2}x_{1,3} + w_{2,1}x_{2,2} + w_{2,1}x_{2,3} \\ w_{1,1}x_{2,1} + w_{1,2}x_{2,2} + w_{2,1}x_{3,1} + w_{2,1}x_{3,2} \\ w_{1,1}x_{2,2} + w_{1,2}x_{2,3} + w_{2,1}x_{3,2} + w_{2,1}x_{3,3} \end{pmatrix}$$

The first term in Equation 2.23 is a sparse matrix with the same number of rows as unique positions in the input and the same number of columns as the total size of the input. Biases can be included in the operation by simply concatenating the bias values to the first matrix and 1s to the input vector.

Since all computations are differentiable, backpropagation is performed in a similar fashion to the linear case. The error propagated from the upper layer has the same shape as the output feature map. These errors are multiplied by the transpose of the first term to compute the errors that can be propagated further to the previous layer.

2.3.2 Deconvolution

The convolution operation often involves decreasing the dimensionality of the input feature map. By doing so multiple times, low dimensional features can be extracted from high dimensional data that can be used for e.g. classification. In a generative settings as in the GAN, we are interested in generating images from latent variables.

This means that we need to upsample representations to higher dimensions which is the opposite to what CNNs are used for. This opposite operation can be accomplished by deconvolution, or transposed convolution, as the name suggest, by simply transposing the sparse matrix. This operation is equivalent to the multiplication in the backpropagation of a CNN. Clearly, the backpropagation in a deconvolution layer is equivalent to the forward pass in a convolution.

2.4 Generative Modeling

Generative modeling involves learning probability distributions of given sets of data. While a discriminative model is concerned with inferring labels y from data samples x, i.e. optimizing its parameters for $p(y|x)$, a generative model, usually conditioned on some latent state, generates previously unseen data samples by optimizing for $p(x|y)p(y)$. This section will explain two such models, namely the Variational Autoencoder (VAE) and the model that we in this work focus on, the GAN. Both model are trained unsupervised meaning that training data does not need to be labeled.

2.4.1 Variational Autoencoder

The VAE is a generative model that is based on Bayesian inference [18]. It is trained to maximize the likelihood of observing the training data X.

$$\prod_{x \in X} p(x) \tag{2.24}$$

Or equivalently, to maximize the sum of the log-likelihoods.

$$\sum_{x \in X} \log p(x) \tag{2.25}$$

To do so, it uses an encoder neural network $Q(z|X)$ to approximate the intractable true distribution $P(z|X)$ of observing the latent variable z given a sample $x \in X$. The assumption is that any sample x is generated based on a hidden state z that we cannot directly observe. Each term in the sum in Equation 2.25 can then be marginalized but is still an intractable integral. By introducing a decoder network $P(X|z)$ that is responsible for generating data given this hidden state z and by formulating the divergence between the former two distributions as follows,

we get a different expression for the log-likelihood $\log P(X)$ that can be optimized.

$$\begin{aligned} D_{\mathrm{KL}}(Q(z|X) \mid\mid P(z|X)) &= \mathbb{E}_{z\sim Q}[\log Q(z|X) - \log P(z|X)] \\ &= \mathbb{E}_{z\sim Q}\left[\log Q(z|X) - \log \frac{P(X|z)P(z)}{P(X)}\right] \\ &= \mathbb{E}_{z\sim Q}[\log Q(z|X) - \log P(X|z) - \log P(z) + \log P(X)] \\ &= \mathbb{E}_{z\sim Q}[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X) \\ &= \mathbb{E}_{z\sim Q}[\log Q(z|X) - \log P(z)] - \mathbb{E}_{z\sim Q}[\log P(X|z)] + \log P(X) \\ &= D_{\mathrm{KL}}(Q(z|X) \mid\mid P(z)) - \mathbb{E}_{z\sim Q}[\log P(X|z)] + \log P(X) \end{aligned} \tag{2.26}$$

The second row uses Bayes' theorem. Rearranging the equation so that $\log P(X)$ is the only term on one side.

$$\log P(X) = \mathbb{E}_{z\sim Q}[\log P(X|z)] - D_{\mathrm{KL}}(Q(z|X) \mid\mid P(z)) + D_{\mathrm{KL}}(Q(z|X) \mid\mid P(z|X)) \tag{2.27}$$

This is the fundamental equation in VAE. Since the last term on the RHS is non-negative, the first and the second term is called the variational lower bound of the LHS log-likelihood. We can train the VAE by maximizing the lower bound with respect to the parameters of $Q(z|X)$ and $P(X|z)$, the encoder and decoder networks. Note that although the first term in the lower bound is an expectation, it can be optimized using stochastic gradient descent by updating the parameters using minibatches of training samples. This expectation can be maximized by minimizing the elementwise reconstruction error of samples using e.g. the L2 loss or the Bernoulli log-likelihood if training data is in range $[0, 1]$. Similarly, the second term, the Kullback-Leibler divergence can be minimized by choosing a normal distribution $\mathcal{N}(0, 1)$ for $P(z)$. This distribution is convenient since the divergence can be computed in closed form. In this case, the encoder $Q(z|X)$ is a neural network with two outputs, namely the mean and the variance of the normal distribution that we want to approximate. z can be sampled from this distribution using the reparameterization trick, which allows us to backpropagate the error through this stochastic process [18].

To generate data samples once the VAE is trained, $z \sim \mathcal{N}(0, 1)$ is sampled and fed through the decoder network.

2.4.2 Generative Adversarial Network

In the settings of generating images, VAEs suffer from blurry samples caused by pixel-wise reconstruction losses and often fail to generate realistic sharp looking images [39]. GANs are generative models that similar to VAEs allow sampling based on latent z but alleviate this problem. They are trained to directly estimate data distributions using two functions, a data generating function and an adversarial function called the generator and the discriminator [9].

While not strictly required by the model, the generator and the discriminator are often implemented as neural networks and are jointly trained with the objective function defined by the following minimax game.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim P_z}[\log(1 - D(G(z)))] \tag{2.28}$$

G is the generative model that allows us to sample data conditioned on latent state z. z is a noise vector obtained from an easily sampled distribution P_z such as $\mathcal{N}(0,1)$ or $\mathcal{U}(-1,1)$. The objective of the generator is to map z to samples in the data space that look like data from the true distribution P_{data}.

D is the scalar value output of the discriminator and represents the probability that it assigns to given data that it is sampled from the true data distribution. It is constrained to [0, 1] using a sigmoid output activation. The discriminator is an adversarial classifier, trained to discriminate between samples from the true distribution and samples from the generator and can be thought of as an trainable loss functions. The parameters are updated using alternative stochastic gradient descent between the two models and GANs converge when those functions reach an equilibrium in this minimax game, a saddle point in the value function. The losses for respective models are based on minibatches of m samples and is shown below.

$$\mathcal{L}_D = \frac{1}{m}\sum_{i=1}^{m}\left[-\log\left(D(x^{(i)})\right) - \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right] \tag{2.29}$$

$$\mathcal{L}_G = \frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \tag{2.30}$$

Minimizing Divergences

Intuitively, one may think of training the discriminator until optimality before updating the generator to compute the gradients that truly minimize the divergence between P_g and P_{data}. Here, P_g is simply the distribution represented by the generator network. This is partially true. The optimal discriminator D^* for a fixed generator, given a sample x can be defined as follows [9].

$$D^*(x) = \frac{p_{data}(x)}{p_{\text{data}}(x) + p_g(x)} \tag{2.31}$$

If the two distributions are equal, then the optimal discriminator has no choice but to output 0.5 and can do no better than randomly guessing whether the given x came from the training data or the generative distribution. Assuming that both neural networks have enough capacity, updating the generator based on D^* is known to minimize the following Jensen-Shannon divergence [9].

$$2D_{\text{JS}}(P_{\text{data}} \mid\mid P_g) - \log 2 \tag{2.32}$$

However, often noticed when training GANs is that the discriminator rejects generated samples with high confidence, especially early in training when the generator is weak. As a result, the absolute value of the gradients become small due to the sigmoid activation, hindering the generator from improving. To alleviate this problem and allow gradients to flow, the authors of GAN suggest replacing the original loss with a non-saturating loss which has larger gradients when the discriminator output is near 0.

$$\mathcal{L}_G = \frac{1}{m}\sum_{i=1}^{m} -\log\left(D\left(G\left(z^{(i)}\right)\right)\right) \tag{2.33}$$

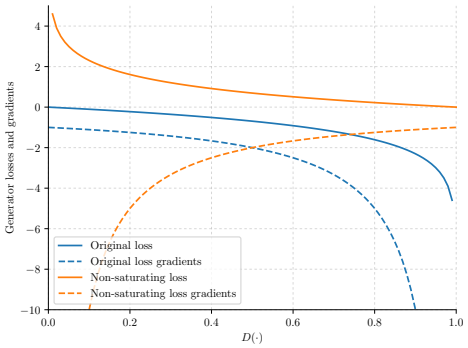


Figure 2.9: Comparison between the original GAN loss and the non-saturating loss. The dashed lines are the gradients with respect to the discriminator input.

Figure 2.9 illustrates the differences. Although this loss has the same fixed point as the original loss, the norm of the gradients are now larger when D is near 0. On the other hand, the training objective is different. Instead of only minimizing the previous divergence, the Kullback-Leibler divergence is minimized as well as the negative Jensen-Shannon divergence, which results in less stable gradients [1]. In practice, P_g generates samples of higher fidelity when trained on images using this loss rather than the original loss and this compromise is required in order to get meaningful gradients at all.

$$D_{\mathrm{KL}}(P_g||P_{\mathrm{data}}) - 2D_{\mathrm{JS}}(P_g||P_{\mathrm{data}}) \tag{2.34}$$

Failure Cases

Training is in general known to be difficult for GANs and is not guaranteed to converge since the existence of an equilibrium is not guaranteed. It could be the case that the discriminator pushes the generator outputs around in the data space never allowing the generator to spread its mass to cover the support of P_{data}. Another

failure case is having the generator collapsing to a single or few modes, basically mapping all possible inputs to nearby points that the discriminator thinks is real. Because of the training instability, various architectures and techniques have been proposed to alleviate these problems [3, 26, 38].

Overfitting

An overfitted GAN is only able to generate samples that too much resemble or is identical to samples in the training data. Such a model would be useless since one could simply take any random sample from the training data instead. Although this could happen in practice, practitioners of GAN and in particular the main author of [9] argue that this is not an issue. The reason for this is that the generator never directly sees a sample from the training data but merely receives information about the true distribution via gradients from the discriminator. Despite this fact, common methods for testing for overfitting exist. For instance, for each generated sample one could find the nearest neighbor in the training data in terms of pixel-wise distance (L1 or L2). If the distance is close to 0, the image is similar to some sample in the training data. Another approach to test for overfitting is to linearly interpolate the noise vector or alternatively use spherical linear interpolations [35]. The generator has not overfitted if the transition is smooth.

Chapter 3

Related Work

This chapter presents previous work in the context of deep learning this is related to our experiments and also briefly introduces techniques that are often seen in similar literature.

3.1 Deep Convolutional Generative Adversarial Networks

The DCGAN was proposed as an extension to GANs with an emphasis on convolutional and deconvolutional layers to generate images [28]. They used deconvolutions in the generator to upsample noise and used batch normalization (see Section 3.1.1) in both the generator and the discriminator to successfully train generative models that could generate sharp realistic looking images of bedrooms and human faces. In addition, the DCGAN is known for being relatively robust to changes in hyperparameters.

3.1.1 Batch Normalization

Batch normalization was first introduced in [15] and was shown to significantly speed up the convergence of neural networks in discriminative settings, achieving state of the art accuracy on the ImageNet classification task.

Batch normalization was motivated by the fact that the output distributions of any layer in a neural network would change during optimization, forcing subsequent layers to continuously having to adapt to those changes ultimately leading to slow and oscillating optimization. The authors of batch normalization called this phenomena the internal covariate shift. To alleviate this problem, batch normalization approximates the mean and the variance of the features in any given layer over the training data by continuously computing the mean and the variance over each minibatch. These values are computed independently over each dimension. It then uses these statistics to normalize the activation to have the mean of 0 and unit variance for each respective minibatch. To further ensure that the model expressivity is not reduced, a scaling parameter γ and a shifting parameter β with equal dimensions

as the activation themselves are introduced such that the batch normalization layer may represent the identity transformation essentially undoing the normalization if necessary. This allows batch normalization in addition to normalization to be fully differentiable, included in the computational graph and thus optimized along with all other parameters in a network.

In short, instead of having all parameters of a transformation layer being responsible for the overall distribution of its output, the distribution is entirely controlled by two parameter γ and β. Therefore, the batch normalization layer is often added after the transformation and before the activation function. The exact algorithm is described in Appendix D.

3.2 Residual Networks

The accuracy of a classifier with l layers should in theory not be lower than a classifier with $l + i$ layers where $i > 0$. For instance, if each additional layer is given enough capacity, they should be able to recover the identity mapping to neither improve nor degrade the model. However in practice, classifier accuracies starts to suffer after certain depth. Possible reasons are vanishing gradients caused by saturating activation functions or badly initialized weights. But same degradation is observed using batch normalization which ensures that gradients are propagated properly even for saturating non-linearities such as the sigmoid function. Shortcut connections, or skip connections were introduced in the residual network architecture and allowed training of deep architectures with hundreds and thousands of layers with state of the art accuracy [12].

Residual networks group layers into so called residual blocks. A residual block uses a shortcut connections to directly pass its input to its output with an element-wise addition at the end, thereby reparameterizing the block to learn the residual. In practice, two or three layers are often grouped together constituting a block that is then trained to learn the residual function.

One possible reason for the success of shortcut connections is the breakage of the symmetry in in the loss landscape, reducing the number of local minima [27]. However, the main reasons for the increased performance is not entirely clear. Since its introduction, the shortcut connection has been widely adopted in various architectures improving existing benchmarks [6,36]. We in this work study the effects of those architectures when put in the settings of GANs.

Residual Learning

Given a set set of connected layers in a neural network, if the optimal transformation to be learned by those layers is defined as $\mathcal{H}(x)$ where x is the input to the first layer, then we can define the residual $\mathcal{F}(x)$ as the difference between the optimal transformation and its input.

$$\mathcal{F}(x) = \mathcal{H}(x) - x \tag{3.1}$$

With residual learning, instead of directly trying to optimize for $\mathcal{H}$, we instead optimize for $\mathcal{F}$ and add its input to the last layer.

$$\mathcal{H}(x) = \mathcal{F}(x) + x \tag{3.2}$$

The capacity of the model will remain unchanged but the conditioning will be different. Since weights can be initialized to be centered at 0 with any variance, training $\mathcal{F}$ to produce a non-contributing 0 output is equivalent to an identity mapping for this set of layers, which is identical to training $\mathcal{H}$ to be the identity mapping but much easier.

3.3 The Universal Approximation Theorem

The universal approximation theorem states based on theoretical analysis that any neural network with at least a single hidden layer with sigmoid non-linearities is capable of approximating any continuous function under mild assumptions using a finite set of parameters [5, 13]. Although this theorem states that deeper architectures should not be necessary, how to optimize such a model is unclear and recent advances in deep learning has at the same time been driven by the success of deeper architectures as in Section 3.2, which leads to the motivation of this work.

Chapter 4

Methodology

This chapter explains the experimental setups for evaluating the impacts of the model capacities on the generative distribution P_g represented by the generator $G(z)$ where $z \sim \mathcal{U}(-1, 1)$ and training stability. Two sets of experiments were conducted, the first one in 2-dimensional space with datasets that allow visual inspection directly in the data space without any reduction of dimensionality. The second set of experiments were conducted on image data with higher dimensionality than the 2-dimensional data, optimizing more sophisticated models. The motivation for conducting two separate experiments are twofold.

1. Before training deeper models with image data, which ultimately is our goal, we can with experiments in 2-dimensional space directly visualize samples from P_{data} and P_g in order to study the impacts of the model capacities to address and eliminate unnecessary experimental setups for image data.

2. How to evaluate the performances of GANs is an ongoing discussion but the Inception score has been widely adopted in the community of GANs that are trained on image data. By not only limiting ourselves to 2-dimensional data, we can use the Inception score to properly compare different models using this score.

The first following sections describe the setups that apply to both experiments. Later sections explain each experiment in detail including model architectures and evaluation metrics.

4.1 Increasing the Capacity of Generative Adversarial Networks

In order to find out if samples generated from P_g can be improved by increasing the model capacity, we increase the number of layers in both the generator and the discriminator. Alternatively, one could increase the width, i.e. the number of neuron per layer in a linear layer, or the number of feature maps and kernel sizes for

convolutional layer, but work such as [7] shows that exponentially many neurons are required for non-linear neural networks with 2 layers as compared to similar neural networks with 3 layers. In other words, increasing the depth is more efficient in order to improve the expressive power of neural networks using the same number of parameters than increasing their widths.

On the other hand, as pointed out by [12], simply increasing the number of layers in a neural network does not necessarily increase its performance in a discriminative setting. More precisely, after a certain number of layers, the performance starts degrading. We assume that this holds true for GANs as well since the main cause for the degradation is the depth itself, which is not unique to discriminative settings but applies generally to any neural network. In order to increase the depth and thereby increasing the expressive power of each model, shortcut connections inspired by the residual network are thus used. The shortcut connections are attached such that each residual block consists of two non-linear transformations. These connections or blocks allow the gradients to propagate to early layers even when the models are deep. We then vary the number of residual blocks in order to evaluate models of different capacities. More residual blocks means deeper models which means higher capacity. We do not use residual blocks in the first and the last layers. These choices are based on [12].

If not stated otherwise, the models used in our experiments are based by the DCGAN [28] architecture. This architecture is robust to the choice of hyperparameters and is well suited as a baseline model [14, 23]. Therefore, our models employ the ReLU activations in the generator and the Leaky ReLU activations in the discriminator as in the DCGAN. Leaky ReLUs are used in the discriminator to allow more gradients to flow to the generator, than using ReLU activations for the reasons explained in Section 2.2.2. To further improve training batch normalizations are applied before non-linearities in all layers except the output layer of the generator and the first layer of the discriminator. These layers are expected to learn the statistics of the data that are not normalized based on the batch statistics. For instance, if batch normalization is applied after the last layer of the generator, its outputs will always be normalized which may be different from the true distribution of the data. This would prevent the generator from generating high quality samples, unless P_{data} is a standard Gaussian. To assign probabilities to samples using the discriminator, the last layer is constructed using a linear layer with a single output neuron, $D(\cdot)$ constrained to $[0, 1]$ using a sigmoid non-linearity.

The models are updated using the binary cross entropy loss with the Adam update rule [17] with $\beta_1 = 0.5, \beta_2 = 0.999$ and varying learning rate α depending on the experiment. Details can be found in the Appendix. These settings are all similar to the settings in [28]. All models are trained with weight decay (L2 regularization) with a decay rate of 0.00001 to prevent the parameters from exploding and from having the discriminator overfitting to the training data. Both the generator and the discriminator are trained using the same training configurations and they use the same number of layers.

For details that are specific to each set of experiments, see Section 4.4 and 4.5.

4.2 Non-Saturating Heuristic

We employ the non-saturating loss as described in [9] in all of our experiments for the same reason as the main author in the original work, i.e. to allow gradients to propagate early in training when the generator is performing poorly.

4.3 Measuring Convergence and Training Progress

We record the outputs of the discriminator given $x \sim P_{data}$ and $x \sim P_g$ respectively during the course of training. Additionally, we plot the losses for the discriminator and the generator throughout the training. Although these metrics does not directly correlate to the training convergence, they allow us to compare different models and study the impacts of different capacities. The reason these metrics are not measuring the convergences is that the adversarial losses for the two networks does not directly tell us whether or not the training is converging towards a saddle point in the value function of the minimax game or not.

4.4 Experiments in $\mathbb{R}^2$

The first set of experiments is conducted on 2-dimensional data. This section explains the datasets, the models and the evaluation methods used in these experiments.

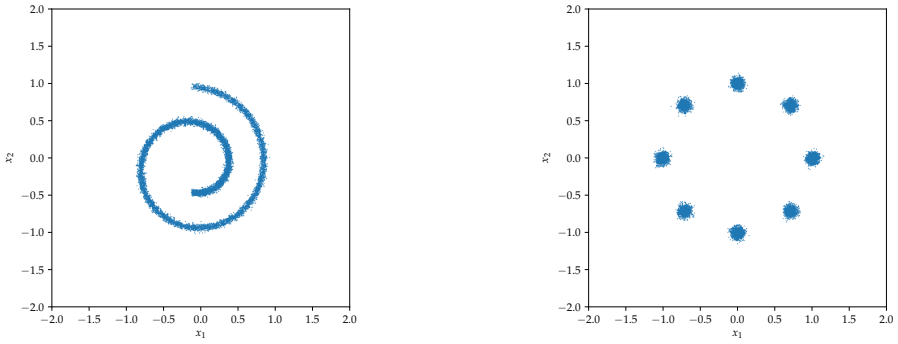
4.4.1 Datasets

The datasets consist of 10000 samples from two bivariate distributions each where a training data sample can be represented by a point in 2-dimensional space $x = \{x_1, x_2\}$.

The first dataset is the 2-dimensional Swiss Roll dataset as seen in the Figure 4.1 (a) centered at the origin with a noise of 0.2 [24]. To also train the GAN on P_{data} with disjoint modes, we employ a bivariate Gaussian mixture with 8 components. The Gaussian mixture is a dataset commonly used in the GAN setting since it represents multiple modes which a model suffering from mode collapse will fail to learn. These datasets are not preprocessed but rather generated such that they are all centered at the origin at $(0, 0)$.

4.4.2 Models

The baseline model is a multi-layer perceptron with an input, hidden and output layer. It is compared agains a residual GAN with input, output layers and 7 intermediate residual blocks. Figure 4.2 illustrates the two models. Since each residual block uses two linear layers with non-linear transformations. Both the generator and the discriminator have 16 layers each and the backpropagation algorithm has to go through 32 layers in total from the loss based on the discriminator output to



(a) The 2-dimensional Swiss Roll dataset. (b) The bivariate Gaussian mixture model.

Figure 4.1: P_{data}, or datasets in $\mathbb{R}^2$. Each dataset consist of 10000 samples from different distributions.

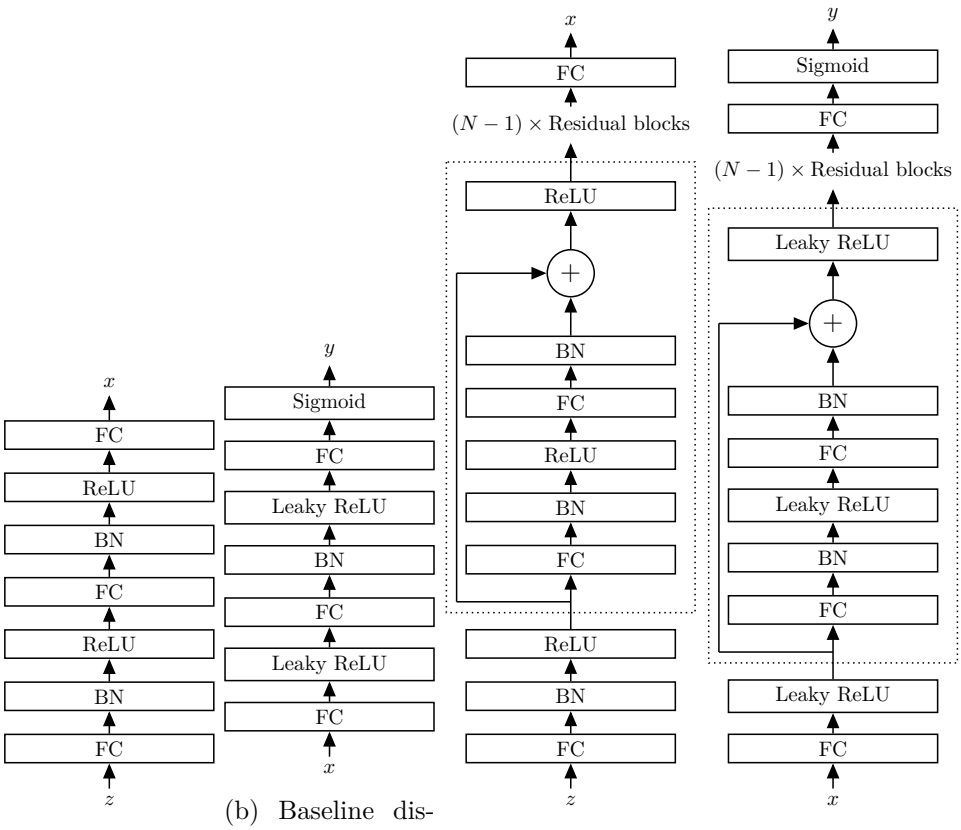
the input layer of the generator. With M standing for *model*, we denote the two models for convenience as follows.

- M0_A - Baseline $\mathbb{R}^2$ model with 3 layers.
- M1_A - Residual $\mathbb{R}^2$ model with 16 layers.

What early results show is that the giving the discriminator the same capacity as the generator usually leads to the discriminator becoming too powerful meaning that the generator fails to learn. In order to avoid this from happening, we propose to remove the batch normalization from the discriminator while keeping the capacities unchanged. This weakens the discriminator and stabilizes the training in the cases where the discriminator outperforms the generator.

4.4.3 Evaluation

The performance of the models are evaluated based on visual inspection of generated samples. Several times during the course of training, the same number of samples as the training data are generated from the generator. They are plotted on the 2-dimensional data space along with the training data which allows us to directly observe how well the generative distribution P_g fits P_{data}. Note that although this evaluation method does not allow us to compare different models precisely, it is sufficient in order to study the impacts of model capacities before proceeding to the experiments on image data where a different evaluation method is used with proper scores.



(a) Baseline generator from $M0_A$.

(b) Baseline discriminator from $M0_A$.

(c) Residual generator from $M1_A$.

(d) Residual discriminator from $M1_A$.

Figure 4.2: Compared models for the $\mathbb{R}^2$ datasets. The layers inside the dashed line constitute a residual block. The N denotes the number of residual blocks and we in our experiments use 7.

4.5 Experiments with Images

The second set of experiments is conducted on image data. This section explains the datasets, the models and the evaluation methods used in these experiments.

4.5.1 Datasets

The two image datasets used for training are the MNIST dataset and the CIFAR-10 dataset. Both datasets are frequently used in computer vision for image classification. The introduction to this report with handwritten digits is based on MNIST.

MNIST is a dataset consisting of 60000 training images and 10000 test images. They are grayscale images with a single channel with dimensions 28×28 pixels. Since each pixel represents the grayscale intensity at that particular position, an image can be represented as $x \in \mathbb{R}^{1\times 28\times 28}$. The CIFAR-10 dataset consists of a

total of 60000 images. Each image is an RBG image with 3 channels with 32×32 pixels and is thus represented as $x \in \mathbb{R}^{3\times32\times32}$.

(a) The MNIST dataset.

(b) The CIFAR-10 dataset.

Figure 4.3: Samples from image datasets.

Images are usually represented by 8-bit per-channel intensity values in the range of $[0, 255]$. In order to feed these images into the neural networks, images are preprocessed and scaled such that each pixel value is in the range $[-1, 1]$. Besides the fact that they are now closer to a 0-mean, this allows us to use the tanh activation function at the last layer of the generator to always make sure that the output is constrained within the same range as the actual training data. Although similar constraint could be achieved using the sigmoid function, we argue that the a 0-centered activation function such as the tanh is preferable since we do not have to shift the output mean to the positive region.

4.5.2 Models

The models for the MNIST dataset and CIFAR-10 differ due to the different dimensions of the image data. This section briefly explains the different baseline models as well as the deeper residual models. For full details of the architecture, please refer to Appendix A.

The baseline model is a DCGAN, for the MNIST experiment with 3 deconvolutional layers in the generator and 2 convolutional layers follower by a linear layer in the discriminator. The models for CIFAR-10 has an additional deconvolution, convolution layer in the generator and discriminator respectively. These models are compared agains residual GANs where the hidden layers are replaced by residual blocks. We denote the models as follows.

- Mc_{B} - MNIST models where $c = 0$ is the baseline DCGAN model and $c = \{1, 2, 3, 4, 5\}$ are the residual models with c residual blocks.
- Md_{C} - CIFAR-10 models where $d = 0$ is the baseline DCGAN model and $d = \{1, 2, 3\}$ are the residual models with $2d$ residual blocks.

4.5.3 Evaluation

The training performance is measured using the Inception score which has been used for evaluating many other generative models that generate images [14, 30, 37, 38]. The Inception score is defined as

$$\exp(\mathbb{E}_x[D_{\mathrm{KL}}(P_{inception}(Y|X) \;||\; P_{inception}(Y))]) \tag{4.1}$$

which is the exponential of the expected Kullback-Leibler divergence from $P_{inception}(Y)$, the marginal distribution over all predictions in a set of samples to $P_{inception}(Y|X)$, the conditional distribution over classes given by a pre-trained Inception network [31]. Intuitively, this means that a classifier should make highly confident predictions, assigning low entropy to $P_{inception}(Y|X)$, but with variations, high entropy to $P_{inception}(Y)$, for well optimized generative models that yield realistic images. Higher is therefore better.

While most work related to GANs have used the Inception score to evaluate fully trained models, we in this work plot the inception scores during the course of training. By doing so, the training stability can be studied by looking at the rate in which the Inception score oscillates. However, in order to achieve this in reasonable time, the number of samples used to compute the score must be reduced since inference using the Inception model and computing the actual score otherwise becomes the bottleneck of the training. We base each score on 5000 samples in all of our experiments instead of the originally suggested 50000.

4.5.4 Nearest Neighbor

A desirable property of P_g is to be able to sample data points that while resembling the characteristics of the training data $x \sim P_{\mathrm{data}}$ are previously unseen. If P_g is only able to generate identical data points to the training data, we could simply sample from P_{data} instead. Although unlikely as described in Section 2.4.2, the GAN objective does not penalize this behavior. We therefore try to detect signs of overfitting by finding for each generated sample the nearest neighbor in terms of the L2 or pixel-wise Euclidean distance. At the same time, since small perturbations in the image may lead to significantly different results, e.g. shifting the image by one or two pixels, we in this work visually inspect the nearest neighbors for randomly samples images, but only to find the starkest forms of overfitting.

Chapter 5

Results

For each experiment, the losses for the generator and the discriminator are plotted during the course of training. The orange and blue losses correspond to the generator and discriminator losses respectively. We observe that when the generator and the discriminator are balanced, that the losses of both networks stay relatively stable. But more often than not, the generator loss increases and the discriminator loss decreases as the training progresses given similar capacities to both networks. The rate in which this occurs is elevated as the models are given additional capacity. In theory, this is preferred since in order to get meaningful gradients that approximately minimizes the Jensen-Shannon and Kullback-Leibler divergences the discriminator loss should be relatively small. We see however in practice that if the discriminator is too strong that the quality of P_g can suffer. This could be a result of the increased variances of the gradients as the discriminator becomes too strong. By removing batch normalization from the discriminator, this behavior is relaxed.

In addition to the losses, the output of the discriminator, i.e. the probability that the discriminator assigns to given data is plotted. Here, the orange and blue curves correspond to the output of the discriminator given generated and true data respectively. Note that the losses are determined solely based on the discriminator outputs which means that the two plots are correlated. We visualize both for the sake of interpretation of the results and specifically in order to find out when the discriminator starts assigning values close to 0 or 1. We observe using these plots that when the losses stay stable, that the discriminator outputs stay relatively close to 0.5 for both $x \sim P_g$ and $x \sim P_{\text{data}}$ which means that the discriminator cannot distinguish between generated samples and samples from the training data but that P_g often is improved.

For the 2-dimensional experiments, generated samples are visualized in the data space and for image experiments, Inception scores are plotted over the course of training. The following sections present specific details to each experiment.

5.1 Results in $\mathbb{R}^2$

For the 2-dimensional experiments with the Swiss roll dataset and the Gaussian mixture using $\mathrm{M0_A}$ and $\mathrm{M1_A}$, data points are sampled and plotted on the data space from the generator at iteration 100, 1000, 2000, 3000, 4000, 6000, 8000, 100000, 15000, 20000. To generate the samples, a fixed set of 10000 random noise vectors z, one for each data point, is sampled and used throughout the training.

As visualized in Figure 5.1 and Figure 5.2, the baseline models are stable and seem to converge towards an equilibrium. P_g has high density where training date is present as seen in the top two rows of the visualization. However, the final results are poor since a lot of data that is clearly absent in the training data is generated. By increasing the capacities of the models, many of those samples are instead mapped by the generator closer to the true data manifold. In other words, P_g using a deeper model yields better estimates of P_{data}. Albeit the results, training suffers from oscillation as observed both in the loss curves and sample plots if capacities are increased naively using residual blocks. Since these oscillations are likely to be caused by unstable gradients from the discriminator, we remove the batch normalization layers from the discriminator and observe that the training instability is partially alleviated and that the final model more accurately approximates the true data distribution than the naive high capacity model.

5.1.1 Loss and Sample Quality Correlation

The result from training the baseline model $\mathrm{M0_A}$ in Figure 5.3 (left) shows that training is relatively stable although the generator loss slowly continues to diverge. This stability can be verified in the visualizations of the sampled data from the generator in Figure 5.1, which show that the mass of generated samples are attracted to the true distribution. Unfortunately, the fact that the model suffers from low capacity cannot be derived from this figure. The right figure with the residual model instead shows a unstable training scenario where, especially early in training, the discriminator rejects generated samples with high confidence. This manifest itself in the form of high losses for the generator. The early samples from iteration 1000 and 2000 in Figure 5.1 shows that this leads to the discriminator pushing the generator around in the data space.

To make the discriminator with residual blocks $\mathrm{M1_A}$ weaker in order to balance the training, batch normalization is removed from the discriminator. Figure 5.4, 5.6 presents the results. This time, the discriminator performance is hampered and the overall training is more stable.

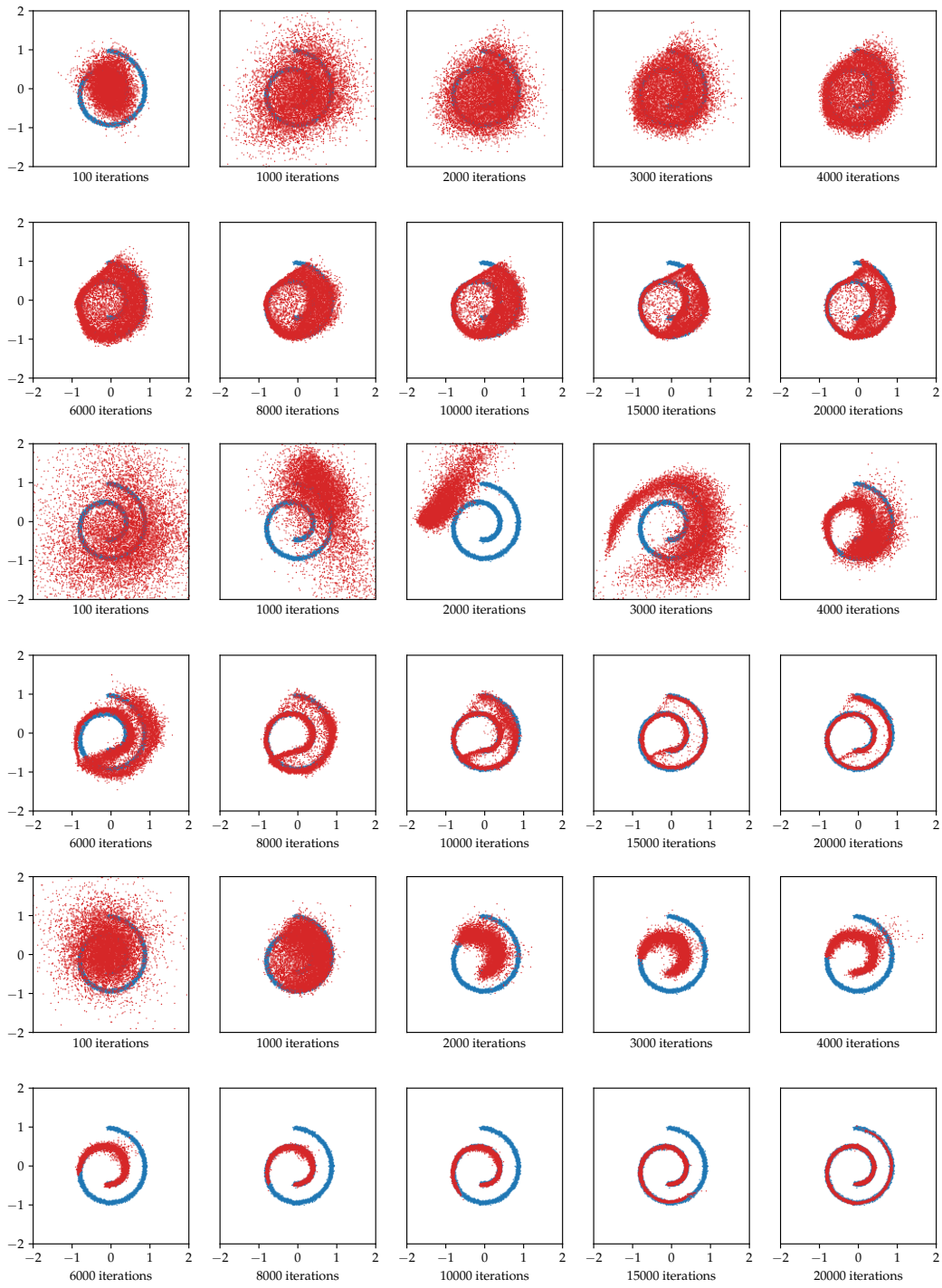


Figure 5.1: Training GANs with the Swiss Roll dataset. First two rows show the training progress with $M0_A$, followed by $M1_A$ and $M1_A$ without batch normalization in the discriminator.

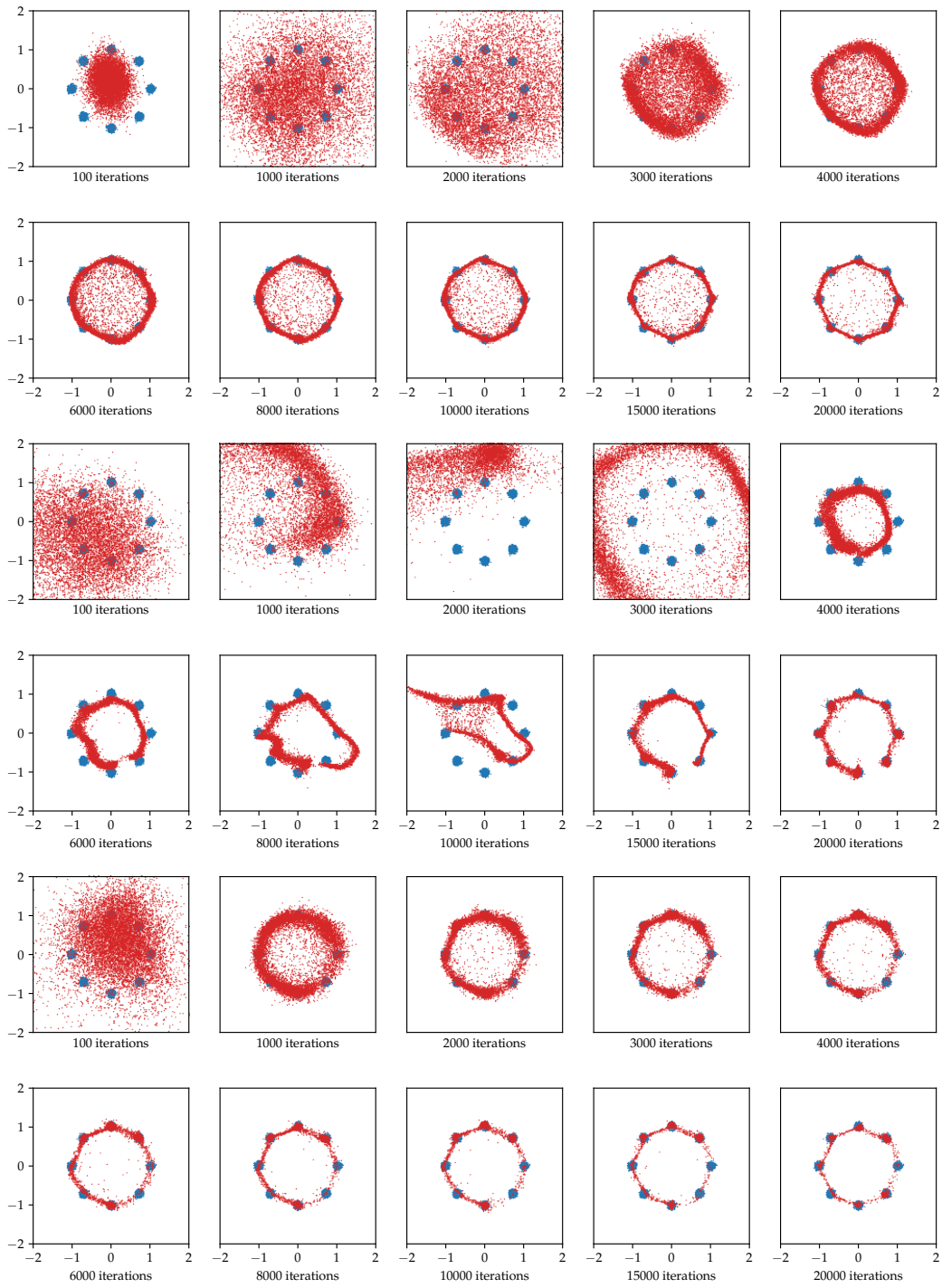


Figure 5.2: Training GANs with the Gaussian mixture. First two rows show the training progress with $M0_A$, followed by $M1_A$ and $M1_A$ without batch normalization in the discriminator.

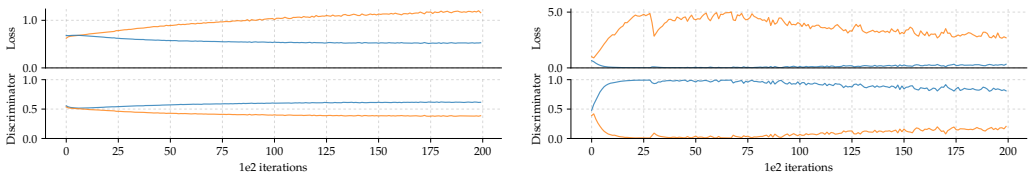


Figure 5.3: Losses and discriminator outputs from the Swiss roll dataset. The orange losses correspond to the generator loss and the blue losses correspond to the discriminator. The blue discriminator output is based on samples from the training data and the orange plot is based on generated data. The left plots are from training the $M0_A$ model and and the right plots are trained with $M1_A$.

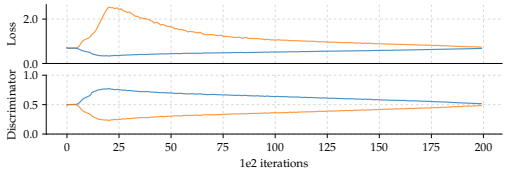


Figure 5.4: Similar plots to Figure 5.3 except that the batch normalization is removed from the discriminator.

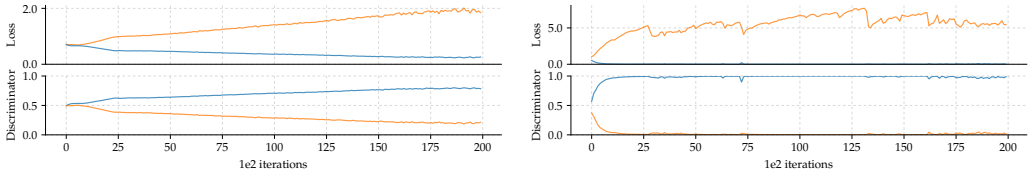


Figure 5.5: Similar plots to Figure 5.3 but with the Gaussian mixture.

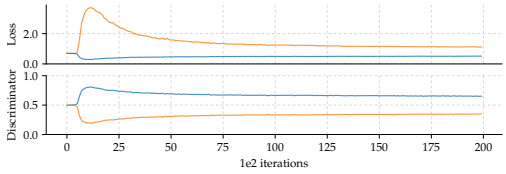


Figure 5.6: Similar plots to Figure 5.5 except that batch normalization is removed from the discriminator.

5.2 Results with Images

To extend the results from the 2-dimensional space to images, similar experiments are conducted using MNIST and CIFAR-10. The models for MNIST are trained for 30000 iterations and the models for CIFAR-10 are trained for 80000 iterations. Early experiments show that that the CIFAR-10 dataset is more difficult for the GAN to learn than the MNIST dataset, which is expected due to the higher data dimensionality. The number of iterations are chosen through experiments and are based on the convergences of the Inception scores.

The losses and discriminator outputs presented in Figure 5.7, 5.8, 5.9 and 5.10 are similar to the results in the previous experiments in 2-dimensions. As the model capacities are naively increased, the divergences of the losses become prominent. In addition, we clearly see that the variances of the losses are increased. Again, by removing the batch normalization layers from the discriminator, the training becomes more stable.

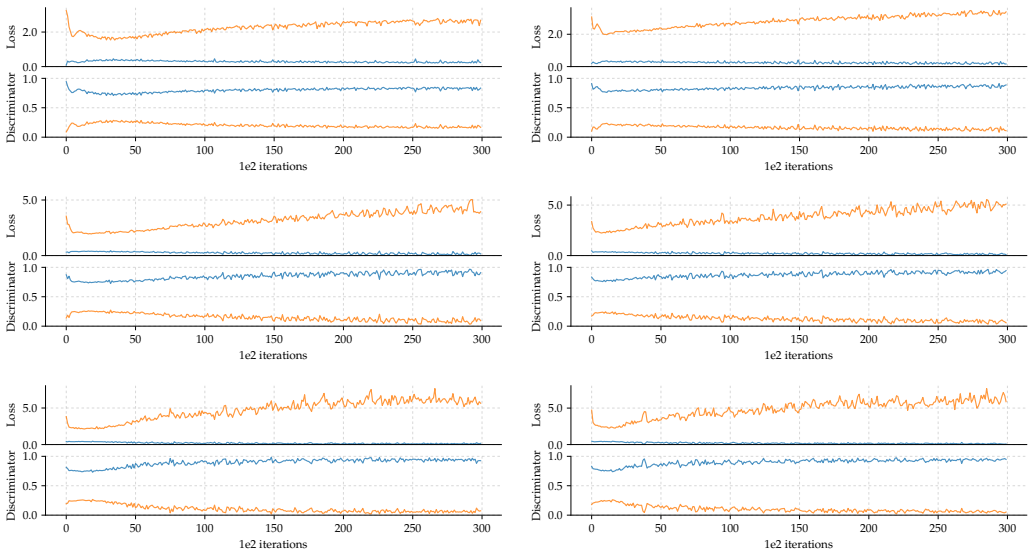


Figure 5.7: Training GANs with MNIST. From top left to bottom right, the training results for $M0_B$, $M1_B$, $M2_B$, $M3_B$, $M4_B$, $M5_B$. The orange loss corresponds to the generator loss and the blue loss corresponds to the discriminator. The orange discriminator prediction is the probability assigned to the generated data and the blue prediction is the probability assigned to the training data.

Since the densities cannot be directly visualized due to the high dimensionality, instead of plotting the generated samples against the true data distribution, the Inception scores are plotted during the course of training. The scores are based on batches of 5000 samples and similarly to the 2-dimensional experiments where generated samples are plotted, a fixed set of 5000 noise vectors are sampled before

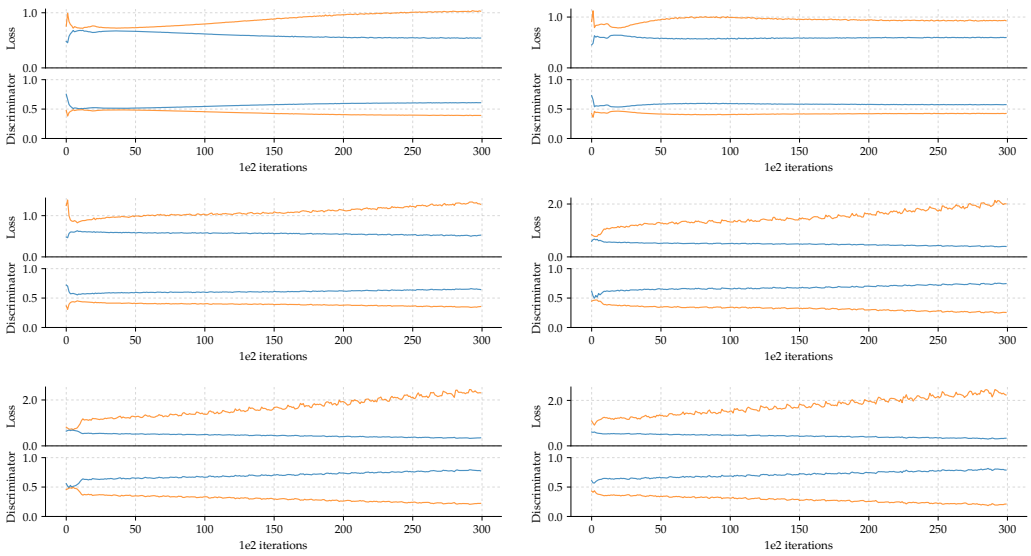


Figure 5.8: Training GANs with MNIST without batch normalization in the discriminator. The plots are ordered similarly to Figure 5.7.

training and then reused throughout the training. Since computing the Inception scores is still computationally demanding, they are computed once every 100-th iteration.

For the MNIST experiments, a pre-trained MNIST classifier network with an 0.9809 accuracy on the test data was prepared that assigned an Inception score of 9.97 to the original dataset. The baseline model $M0_B$ achieves a score of 9.14 while the deeper model with 3 residual blocks, $M3_B$ achieves the highest score of 9.73. This latter score is close to 9.97 meaning that the residual GAN likely estimates the probability distribution of the true data well and in particular, better than the baseline model. Scores are based on models where the discriminator uses batch normalization. The rest of the scores are shown in Figure 5.11 which shows that even deeper models yield lower scores. Removing batch normalizing from the discriminator severely degrades the results which is inconsistent with the 2-dimensional experiments but we argue that the true distribution of the handwritten images in the MNIST dataset are represented by a distribution that the convolutional generator with high capacity easily learns, even early in training. If this is true, then it should prevents the discriminator from ever becoming too strong and hence there should be no need for removing the batch normalization layers. We also notice that increasing the number of layer even further doesn't improve the quality.

For the CIFAR-10 experiments, the parameters for the Inception model from [31] are used as suggested in [30]. The CIFAR-10 experiments in contrast to MNSIT show similar behavior to the 2-dimensional experiments where removing the batch normalization layers from the discriminator stabilizes training. In addition, the

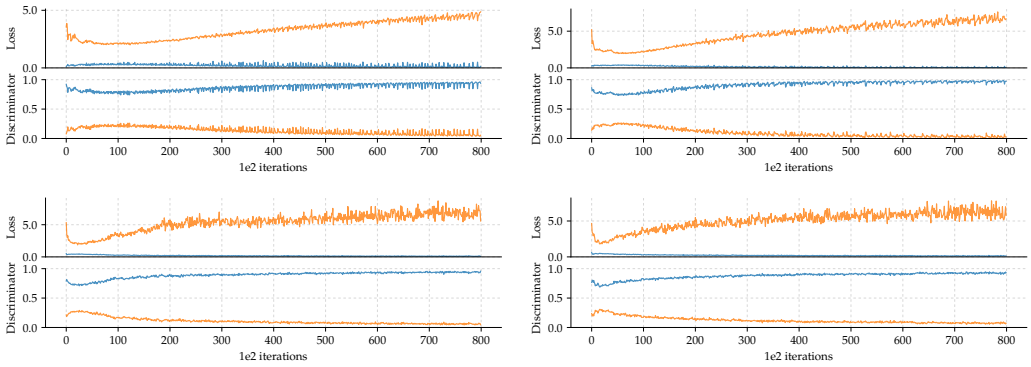


Figure 5.9: Training GANs with CIFAR-10. From top left to bottom right, the training results for $M0_C$, $M1_C$, $M2_C$, $M3_C$. The orange loss corresponds to the generator loss and the blue loss corresponds to the discriminator. The orange discriminator prediction is the probability assigned to the generated data and the blue prediction is the probability assigned to the training data.

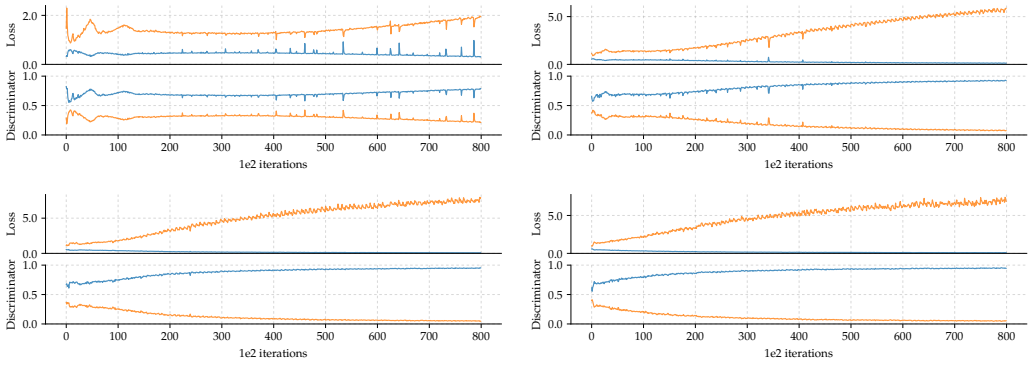
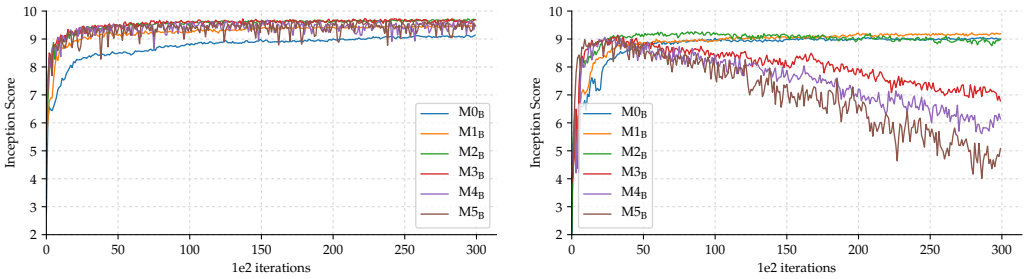


Figure 5.10: Training GANs with CIFAR-10 without batch normalization in the discriminator. The plots are ordered similarly to Figure 5.9.

Inception scores are increased with the capacities of the networks up until a certain point. More specifically, the $M2_C$ model with 4 residual blocks without batch normalization in the discriminator achieves the highest Inception score of 7.60. Increasing the capacity even further did not improve the results and instead caused mode collapse. This was also the case for the models with highest capacities in the MNIST experiments.

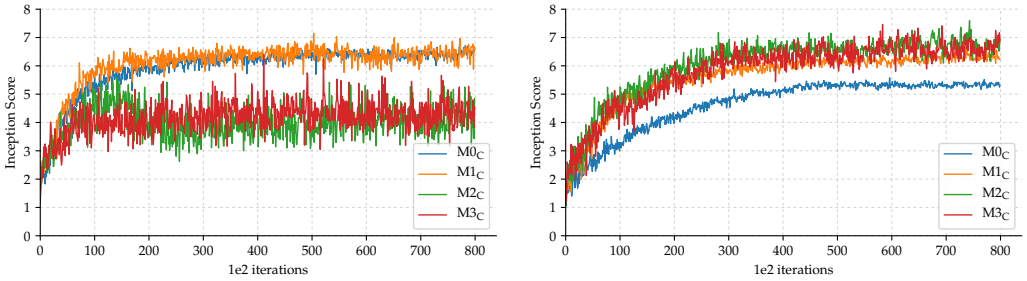
5.2.1 Nearest-Neighbor Samples

From the best performing generators in each image experiment with respect to the Inception scores, 100 images are randomly sampled. For each of these images,



(a) Batch normalization in both networks. (b) Without batch normalization in the discriminator.

Figure 5.11: The Inception scores for MNIST.



(a) Batch normalization in both networks. (b) Without batch normalization in the discriminator.

Figure 5.12: The Inception scores for CIFAR-10.

nearest neighbors from the training data in terms of the L2-loss or the Euclidean pixel-wise loss is retrieved. Figure 5.15 shows the nearest neighbors visualized next to the generated samples. $M3_B$ generates samples that highly resembles the training samples, both in terms of visual fidelity but also in terms of L2-loss. This is not as prominent in $M2_C$ that was trained on CIFAR-10 as the images in general does not achieve the same visual quality.

MNIST			CIFAR-10		
Model	BN	Score	Model	BN	Score
$M0_B$	G, D	9.14 ± 0.14	$M0_C$	G, D	6.71 ± 0.17
$M1_B$	G, D	9.50 ± 0.074	$M1_C$	G, D	7.15 ± 0.26
$M2_B$	G, D	9.72 ± 0.046	$M2_C$	G, D	5.59 ± 0.18
$M3_B$	G, D	$\mathbf{9.73 \pm 0.058}$	$M3_C$	G, D	6.41 ± 0.22
$M4_B$	G, D	9.68 ± 0.068	$M0_C$	G	5.58 ± 0.16
$M5_B$	G, D	9.70 ± 0.070	$M1_C$	G	6.65 ± 0.19
$M0_B$	G	9.05 ± 0.10	$M2_C$	G	$\mathbf{7.60 \pm 0.31}$
$M1_B$	G	9.22 ± 0.058	$M3_C$	G	7.46 ± 0.21
$M2_B$	G	9.26 ± 0.12	$\text{CIFAR-10}_{\text{train}}$		12.00 ± 0.10
$M3_B$	G	9.14 ± 0.094			
$M4_B$	G	9.07 ± 0.15			
$M5_B$	G	9.11 ± 0.14			
$\text{MNIST}_{\text{train}}$		9.97 ± 0.010			

Table 5.1: The Inception scores for each model on the MNIST and CIFAR-10 datasets. The best score in each experiment is marked bold.

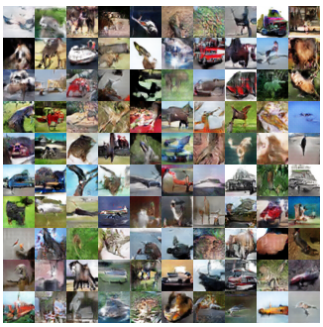
(a) Good samples from $M3_B$ with high capacity.

(b) Bad samples from $M0_B$ due to low capacity.

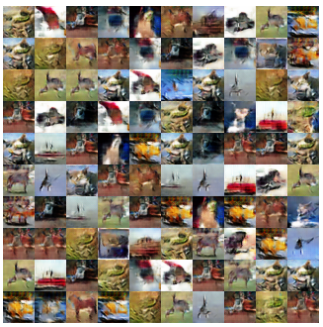
(c) Mode collapse with $M5_B$ high capacity model without batch normalization in the discriminator.

Figure 5.13: Sampled random images from different models trained with MNIST.

(a) Good samples from $M2_C$ with high capacity.



(b) Bad samples from $M0_C$ due to low capacity.



(c) Mode collapsed $M3_C$ that was trained with batch normalization in the discriminator.

Figure 5.14: Sampled random images from different models trained with CIFAR-10.

(a) Samples from $M3_B$ and nearest neighbors.

(b) Samples from $M2_C$ and nearest neighbors.

Figure 5.15: Nearest-neighbor visualizations for generated images from $M3_B$ and $M2_C$. For each generated image (left), the nearest image in the training data (right) is shown.

Chapter 6

Discussion

While GANs are known for being difficult to evaluate due to their lack of proper convergence and non-heuristic evaluation metrics, we clearly see that the sample quality can be improved by increasing their capacities. These observations were made through empirical studies, visualizations in the data space for 2-dimensional experiments as well using the Inception scores and image samples using MNIST and CIFAR-10. We will in this section try to break down the underlying factors and analyze the causes for different observations in the previous chapters.

6.1 Sample Quality and Training Stability

The sample quality in terms of how well the generating distribution matches the true distribution in the 2-dimensional experiments, and the Inception scores in the image experiments, does improve in all experiments with additional capacity. This shows that GANs can benefit from additional expressive power although being defined by adversarial minimax games. To estimate high dimensional distributions, models with few parameters are simply not enough, mapping noise z to data points that are far way from the true data distribution P_{data}.

This is true until the model reaches a certain depth that causes the training to instead become too unstable and stops P_g from improving. The instability could be the result of the fact that finding a Nash equilibrium becomes more difficult as the number of parameters increase and thus causes the loss landscapes to become too complex. Additionally, as described in [1], the non-saturating loss although being successfully applied in practice, is by definition unstable and is known to cause the variances of the gradients to increase as the training progresses. This behavior is observed in the losses during the course of training. It is especially prominent in the image experiments with residual high capacity GANs as shown in Figure 5.7, 5.8, 5.9, 5.10.

The oscillations caused by the high capacity models trained on the image dataset with batch normalization in the discriminator resemble the behavior of the 2-dimensional experiment with the same setup. What happens is that the discrimina-

tor pushes the samples from the generator around in the data space. This is most likely caused by the discriminator rejecting $x \sim P_g$ with high confidence (assigning low probabilities to $D(G(z))$) that leads to large gradients propagated to the generator due to the non-saturating loss. Although the discriminator needs to be near optimal in order for the training to approximately minimize the divergences between the distributions, given the same amount of high capacity with residual blocks, the discriminator simply becomes too strong. In order to balance the networks and stabilize the training, removing batch normalization from the discriminator is a heuristic that in practice seems to work well. While batch normalization greatly helps gradients to propagate to early layers in deeper architectures, we note that the leaky ReLU non-linear activations in this case is enough since it does not suffer from saturated regions and is less prone to dead ReLU units. In our experiments, removing batch normalization layers from the discriminator lead to better samples in the 2-dimensional experiment as well as with CIFAR-10.

To further decrease the amount of oscillation, we could simply alter the learning rates. However, this would not necessarily make the training more stable. Additionally, it would make the interpretations of the results of the comparisons between different models less representative of the effects of capacity. In order to study and isolate the effects of model capacities on the training stability and sample quality, we want to keep other training parameters fixed, including the learning rate. Instead, we argue that overall lower learning rates could have helped improving P_g. Alternatively, different learning rates could be used for the generator and the discriminator but this is an issue of balancing the two networks. Why the variances are not increasing to the same extent in the 2-dimensional experiments is unclear and might require us to look at the actual gradient norms during training. In general, we see that the adversarial training becomes more unstable as the models get more complex.

6.2 Mode Collapse

Mode collapse, i.e. when the generator starts generating samples that all look similar to each other, the generator has collapsed into a single or few modes of the true data distribution. Once this happens, the generator usually has a hard time escaping these modes since the non-saturating objective function does not incentivize the generator to explore other modes.

Results show that for the best performing models in all experiments, that this is not an issue. $\mathrm{M3_B}$ successfully generates all 10 digits that are present in the MNIST dataset and images from $\mathrm{M2_C}$ similarly show highly varied samples in terms of colors and objects. On the other hand, the models with highest capacities such as the $\mathrm{M5_B}$ and $\mathrm{M3_C}$ suffer from this problem. For instance as shown in Figure 5.13 (c), randomly sampled images from the fully trained $\mathrm{M5_B}$ model are mostly images of the digit 1. Similarly, the deepest model trained with CIFAR-10 show little diversity in its generated samples. Although not immediately obvious by these

figures, it was observed during experiments that when these mode collapses occur, that each iteration altered the generated samples completely which meant that the discriminator was pushing the generated samples around in the data space. This is similar to he Swiss roll example with $M1_A$ where the discriminator is strong with batch normalization (Figure 5.1). Whether this is caused by the high capacity generator, discriminator or both is not clear and remains as future work but could be a result of the heavily oscillating training in the deeper models and might be alleviated, again by lowering the learning rates.

6.3 Nearest Neighbor Analysis

For the image experiments, the nearest neighbor images were retrieved for each generated image in the best performing models in order to detect signs of overfitting. Since the generator never sees an example from the training data, but merely receives information in terms of gradients from the discriminator, it should in practice not suffer from overfitting. This is also stated by the author of GAN [9]. What we see in our results are different, especially for the MNIST experiment. The best performing model in terms of the Inception score generates almost identical images to the actual training data. In fact, this is also true for the 2-dimensional experiments. The difference is that the image data spans many more dimensions and the generator still manages to almost perfectly capture the density. We assume that both the generator and the discriminator in this case are able to use their additional parameters to estimate the distributions with high precision which could be interpreted as overfitting. A possible cause is that the size of the dataset is too small in comparison to the number of parameters in the models.

Chapter 7

Summary and Conclusions

In this work, we have shown that GANs are capable of estimating data distributions and generating samples of higher quality by increasing the capacities of the generator and the discriminator by roughly the same number of parameters.

While we still do not know how efficiently GANs make use of their given capacities, our experiments show that the lack of capacity may prevent P_g from improving and at the same time that generative models can be improved by having their capacities increased. The important observation is that this holds true in an adversarial setting. In an adversarial setting, in this case a GAN, two neural networks are optimized with opposite objectives to find Nash equilibria, which makes optimization difficult. However, this improvement in sample quality of P_g does not come for free and due to the instability in the learning algorithm caused by the non-saturating objective function, naively increasing the capacities tended to cause oscillation. Additionally to this intrinsic obstacle, we argue that the instability is further amplified by the discriminator being too confident in rejecting generated samples. We observed through experiments that indeed by making the discriminator weaker, the training becomes stable. In order to weaken the discriminator without actually altering the number of parameters, we propose to remove the batch normalization layers. This simple heuristic helped convergence for many of the experiments.

For data with relatively simple densities such as our 2-dimensional datasets and MNIST, a high capacity GAN can learn to generate almost identical samples to the training data. Whether these models turn out useful or not depend on the application, but we have experimentally shown that GANs are powerful models that given enough capacities can estimate these distributions.

7.1 Future Work

The original saturating loss suffers from vanishing gradients and the non-saturating heuristic, although mitigating this problem introduces one that the objective now includes the negative Jensen-Shannon divergence. As of writing this report, many papers are being published that try to allay this instability of the training objective

of GANs and the divergences that they approximately minimize [2,11,23]. While in practice being able to generate samples, specifically for images, that are sharp and realistic, there are theoretical concerns that need to be addressed. Our experimental findings show that there might be yet undiscovered potential behind GANs with higher capacities with alternative objective functions to the non-saturating loss, in order to generate high dimensional data oh higher fidelity.

Bibliography

[1] Martin Arjovsky and Léon Bottou. Towards Principled Methods for Training Generative Adversarial Networks, 2017, arXiv:1701.04862.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN, 2017, arXiv:1701.07875.

[3] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode Regularized Generative Adversarial Networks, 2016, arXiv:1612.02136.

[4] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets, 2016, arXiv:1606.03657.

[5] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[6] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks, 2016, arXiv:1605.06409.

[7] Ronen Eldan and Ohad Shamir. The Power of Depth for Feedforward Neural Networks, 2015, arXiv:1512.03965.

[8] Xavier Gastaldi. Shake-Shake regularization, 2017, arXiv:1705.07485.

[9] 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. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[10] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples, 2014, arXiv:1412.6572.

[11] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs, 2017, arXiv:1704.00028.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, 2015, arXiv:1512.03385.

[13] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks Are Universal Approximators. *Neural Netw.*, 2(5):359–366, July 1989.

[14] Xun Huang, Yixuan Li, Omid Poursaeed, John Hopcroft, and Serge Belongie. Stacked Generative Adversarial Networks, 2016, arXiv:1612.04357.

[15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015, arXiv:1502.03167.

[16] Orbach JJ. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. *Archives of General Psychiatry*, 7(3):218–219, 1962.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2014, arXiv:1412.6980.

[18] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. 2013, arXiv:1312.6114.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[20] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world, 2016, arXiv:1607.02533.

[21] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[22] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, 2016, arXiv:1609.04802.

[23] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least Squares Generative Adversarial Networks, 2016, arXiv:1611.04076.

[24] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

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

[26] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional Image Synthesis With Auxiliary Classifier GANs, 2016, arXiv:1610.09585.

[27] A. Emin Orhan and Xaq Pitkow. Skip Connections Eliminate Singularities, 2017, arXiv:1701.09175.

[28] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015, arXiv:1511.06434.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[30] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs, 2016, arXiv:1606.03498.

[31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. 2015, arXiv:1512.00567.

[32] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a Next-Generation Open Source Framework for Deep Learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

[33] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. Regularization of Neural Networks using DropConnect. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058–1066. JMLR Workshop and Conference Proceedings, May 2013.

[34] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[35] Tom White. Sampling Generative Networks, 2016, arXiv:1609.04468.

[36] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks, 2016, arXiv:1605.07146.

[37] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, 2016, arXiv:1612.03242.

[38] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based Generative Adversarial Network, 2016, arXiv:1609.03126.

[39] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Towards Deeper Understanding of Variational Autoencoding Models, 2017, arXiv:1702.08658.

Appendix A

Detailed Experimental Setup

A.1 Model Overviews

See Table A.1 and Table A.2 for the model architectures used in all experiments.

A.2 Optimization Algorithm and Hyperparameters

The Adam optimization algorithm [17] was used in all of our experiments with hyperparameters according to Table A.3. Those are similar to [28]. The generators and the discriminators always used the same parameters. Each parameter update was based on the gradients of a minibatch of 64 samples. The number of updates were experimentally configured with respect to computational time, sample quality and convergence speed for each dataset. Models trained on 2-dimensional data were trained for 200000 iterations, MNIST for 300000 iterations and CIFAR-10 for 800000 iterations.

A.3 Regularization

A weight decay, or L2 regularization term with a rate of 0.00001 was added to the to each loss in all experiments in order to avoid parameters from becoming too large or high capacity discriminators from overfitting.

A.4 Parameter Initialization

All weights were initialized with a 0-centered Gaussian with the standard deviations according to Table A.4. Biases were always initialized to 0. Since the models became more sensitive to parameter initializations as they grew, values were experimentally chosen to yield fair results for all model capacities. How to select the optimal configuration is outside the scope of this work.

A.5 Hardware and Software Setup

Experiments were conducted on a GPU server with a Xeon E5-1620v3 4core/8thread 3.5Ghz processor, 4 GeForce GTX TITAN X 12GB GPUs and 128GB DDR4-2133 RAM. Each image experiment utilized two GPUs of which one was used to train the actual model and the other to compute the Inception scores. The pre-trained Inception model for CIFAR-10 required 8.3 GB of GPU memory to store its parameters. On top of this, each minibach needed to be transferred to the same GPU before inference. The 2-dimensional experiments on the other hand only used the CPU since no additional performance could be gained from using the GPU due to linear layers that cannot be parallelized to the same extent as the convolutional layers for the image experiments.

To implement the neural networks, the Python-based framework Chainer was used [32]. Chainer provides automatic differentiation based on the computational graph defined during run-time. This allowed us to prototype fast and focus on the architectures.

With the above configurations, the 2-dimensional experiments took less than an hour while the image experiments could take up to 10 hours.

Generator, Input $z \in \mathcal{R}^{32}$					
Model	Operation	Output	BN	Non-Linearity	Repetitions
	Linear	32	Yes	ReLU	1
$M0_A$	Linear	32	Yes	ReLU	1
$M1_A$	Linear	32	Yes	ReLU	7
	Linear	32	Yes	ReLU	
	Linear	2	No	N/A	1
Discriminator, Input $x \in \mathcal{R}^2$					
Model	Operation	Output	BN	Non-Linearity	Repetitions
	Linear	32	No	Leaky ReLU	1
$M0_A$	Linear	32	Yes	Leaky ReLU	1
$M1_A$	Linear	32	Yes	Leaky ReLU	7
	Linear	32	Yes	Leaky ReLU	
	Linear	1	No	Sigmoid	1

Table A.1: Architectures used in all $\mathbb{R}^2$ experiments. The first row is the layer closest to the input layer and the bottom layer is the output layer.

Generator, Input $z \in \mathcal{R}^{100}$							
Model	Operation	Kernel	Stride	Output	BN	Non-Linearity	Repetitions
	Deconvolution	7×7	1×1	128	Yes	ReLU	1
$M0_B$	Deconvolution	4×4	2×2	64	Yes	ReLU	1
Mc_B	Deconvolution	4×4	2×2	64	Yes	ReLU	$c > 0$
	Deconvolution	3×3	1×1	64	Yes	ReLU	
	Deconvolution	4×4	2×2	1	No	Tanh	1

Discriminator, Input $x \in \mathcal{R}^{1 \times 28 \times 28}$							
Model	Operation	Kernel	Stride	Output	BN	Non-Linearity	Repetitions
	Convolution	4×4	2×2	64	No	Leaky ReLU	1
$M0_B$	Convolution	4×4	2×2	128	Yes	Leaky ReLU	1
Mc_B	Convolution	4×4	2×2	128	Yes	Leaky ReLU	$c > 0$
	Convolution	3×3	1×1	128	Yes	Leaky ReLU	
	Linear	N/A	N/A	1	No	Sigmoid	1

Generator, Input $z \in \mathcal{R}^{100}$							
Model	Operation	Kernel	Stride	Output	BN	Non-Linearity	Repetitions
	Deconvolution	4×4	1×1	256	Yes	ReLU	1
$M0_C$	Deconvolution	4×4	2×2	128	Yes	ReLU	1
Md_C	Deconvolution	4×4	2×2	128	Yes	ReLU	$d > 0$
	Deconvolution	3×3	1×1	128	Yes	ReLU	
$M0_C$	Deconvolution	4×4	2×2	64	Yes	ReLU	1
Md_C	Deconvolution	4×4	2×2	64	Yes	ReLU	$d > 0$
	Deconvolution	3×3	1×1	64	Yes	ReLU	
	Deconvolution	4×4	2×2	3	No	Tanh	1

Discriminator, Input $x \in \mathcal{R}^{3 \times 32 \times 32}$							
Model	Operation	Kernel	Stride	Output	BN	Non-Linearity	Repetitions
	Convolution	4×4	2×2	64	No	Leaky ReLU	1
$M0_C$	Convolution	4×4	2×2	128	Yes	Leaky ReLU	1
Md_C	Convolution	4×4	2×2	128	Yes	Leaky ReLU	$d > 0$
	Convolution	3×3	1×1	128	Yes	Leaky ReLU	
$M0_C$	Convolution	4×4	2×2	256	Yes	Leaky ReLU	1
Md_C	Convolution	4×4	2×2	256	Yes	Leaky ReLU	$d > 0$
	Convolution	3×3	1×1	256	Yes	Leaky ReLU	
	Linear	N/A	N/A	1	No	Sigmoid	1

Table A.2: Architectures used in MNIST and CIFAR-10 experiments. c and d represents the number of residual blocks in each model. The first row is the layer closest to the input layer and the bottom layer is the output layer.

	α	β_1	β_2
$\mathbb{R}^2$	0.0001	0.5	0.999
Images	0.0002	0.5	0.999

Table A.3: Parameters for the Adam optimization algorithm.

	Mean	Standard Deviation
$\mathbb{R}^2$	0	0.1
Images	0	0.02

Table A.4: Means and standard deviations for the Gaussians that were used to initialize the weights for our models. Biases were always initialized to 0.

Appendix B

MNIST Inception Scores

The following table shows the architecture of the pre-trained MNIST classifier that was used to compute the Inception scores for the MNIST experiments. The model was trained using the training dataset of MNIST and the parameters were saved once the accuracy of the test dataset reached 0.9809.

MNIST classifier, Input $x \in \mathcal{R}^{1\times28\times28}$					
Operation	Kernel	Stride	Output	BN	Non-Linearity
Convolution	5×5	1×1	32	No	ReLU
MaxPooling	2×2	2×2	32	No	N/A
Convolution	5×5	1×1	64	No	ReLU
MaxPooling	2×2	2×2	64	No	N/A
Linear	N/A	N/A	1024	No	ReLU
Dropout(0.5)	N/A	N/A	1024	No	N/A
Linear	N/A	N/A	10	No	Softmax

Table B.1: Architecture overview of the pre-trained MNIST classifier used to compute the Inception scores. Since all convolutions in this model are size preserving with a padding of 2 in each dimension, max pooling was applied to decrease the sizes of the feature maps. Dropout was included with a drop rate of 0.5 which on average causes half of the activations to randomly become 0. The layers introduced in this table that are not explained in our work are beyond the scope of this report but are not immediately correlated to our findings or results. The outputs of this neural network are vectors with the logits for the probabilities for each digit ranging from 0 to 9 after an additional softmax operation.

Appendix C

Extreme Unbalancing of the Generator and the Discriminator

This section shows the results of unbalancing the generator and the discriminator unfairly for the Swiss roll and Gaussian mixture datasets. The purpose of this experiment was to get an intuition behind the importance of balancing the two networks. Batch normalization was used in the layers of the discriminators.

In the first experiments, the baseline generator from $\mathrm{M0_A}$ was trained against the discriminator from $\mathrm{M1_A}$. The results in Figure C.1 show that the generator loss increased rapidly and that the discriminator rejected generated samples throughout most of the training with near perfect confidence. As long as there are any overlaps between the supports of P_g and P_{data}, this is expected to be highly unlikely since the the discriminator should not be able to tell these overlapping samples apart. A possible reason for this behavior was that the dataset of 10000 samples was too small with respect to the distribution that it was being sampled from and the discriminator was able to learn each and every sample. This would mean that larger datasets are needed in order to successfully train deeper GANs. Also, as observed in the previous experiments, we again see that the highly confident predictions of the discriminator lead to oscillating $x \sim P_g$ as in Figure C.3.

In the second experiments, the generator was given more capacity instead. This time, the generator used the same architecture as the model in $\mathrm{M1_A}$ and the discriminator used the model from $\mathrm{M0_A}$. The losses and discriminator outputs were stable compared to the other experimental setups as seen in Figure C.2. The generated samples visualized in Figure C.4 verifies the stability. In fact, the results resemble the baseline model experiments but where the rate of convergence is significantly improved. This means that the the discriminator still gives meaningful gradients. Although stability is addressed, the final results of the generated samples were still not on par with the experiments where both the generator and the discriminator were given high capacities, meaning that again, the capacity was hindering P_g from improving once it became strong enough and the discriminator could no longer can tell the samples apart.

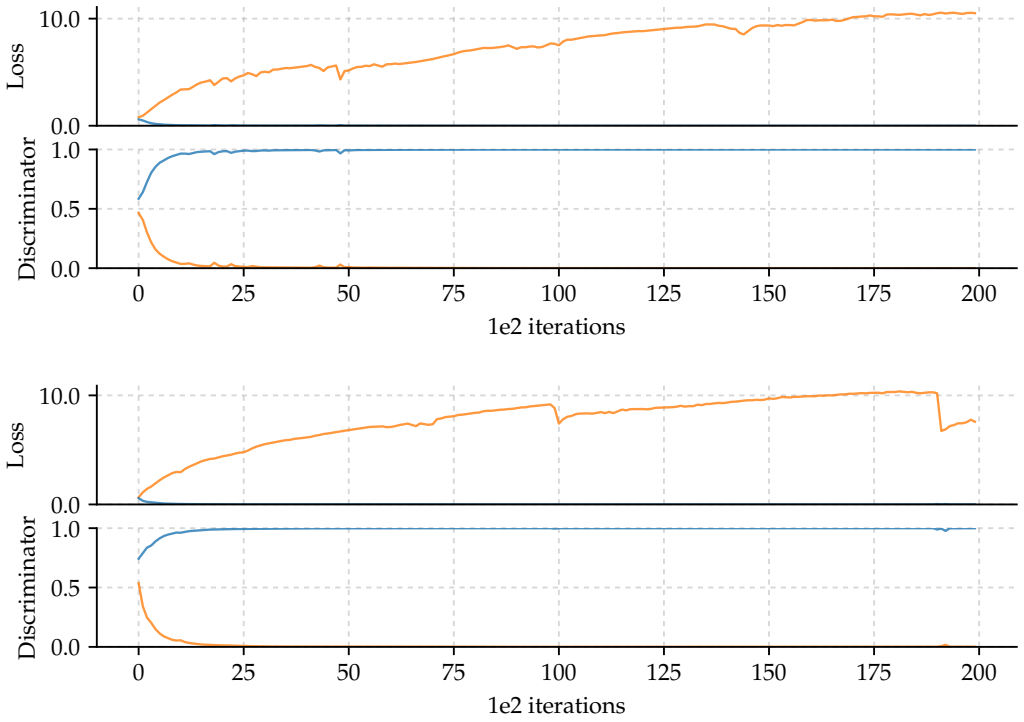


Figure C.1: Losses and discriminator outputs from training a GAN where the discriminator is given higher capacity with the Swiss roll dataset (top) and the Gaussian mixture (bottom).

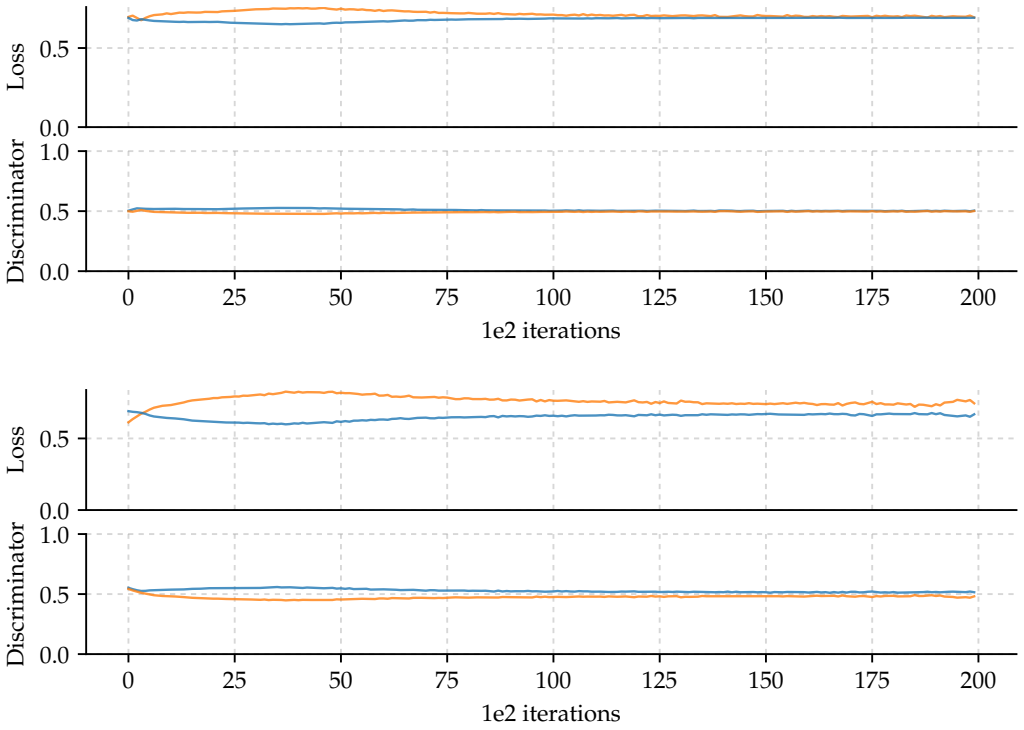


Figure C.2: Losses and discriminator outputs from training a GAN where the generator is given higher capacity with the Swiss roll dataset (top) and the Gaussian mixture (bottom).

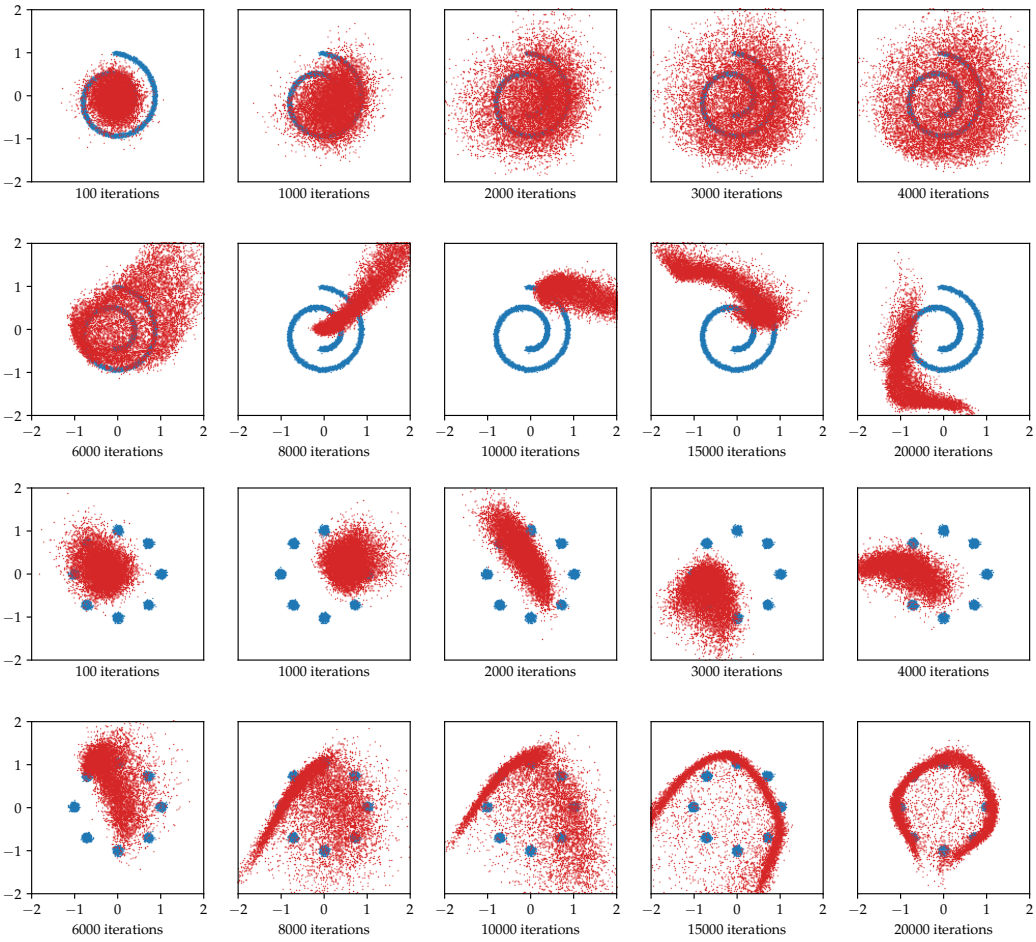


Figure C.3: Generated samples during the course of training from Figure C.1 where the discriminator is given more capacity.

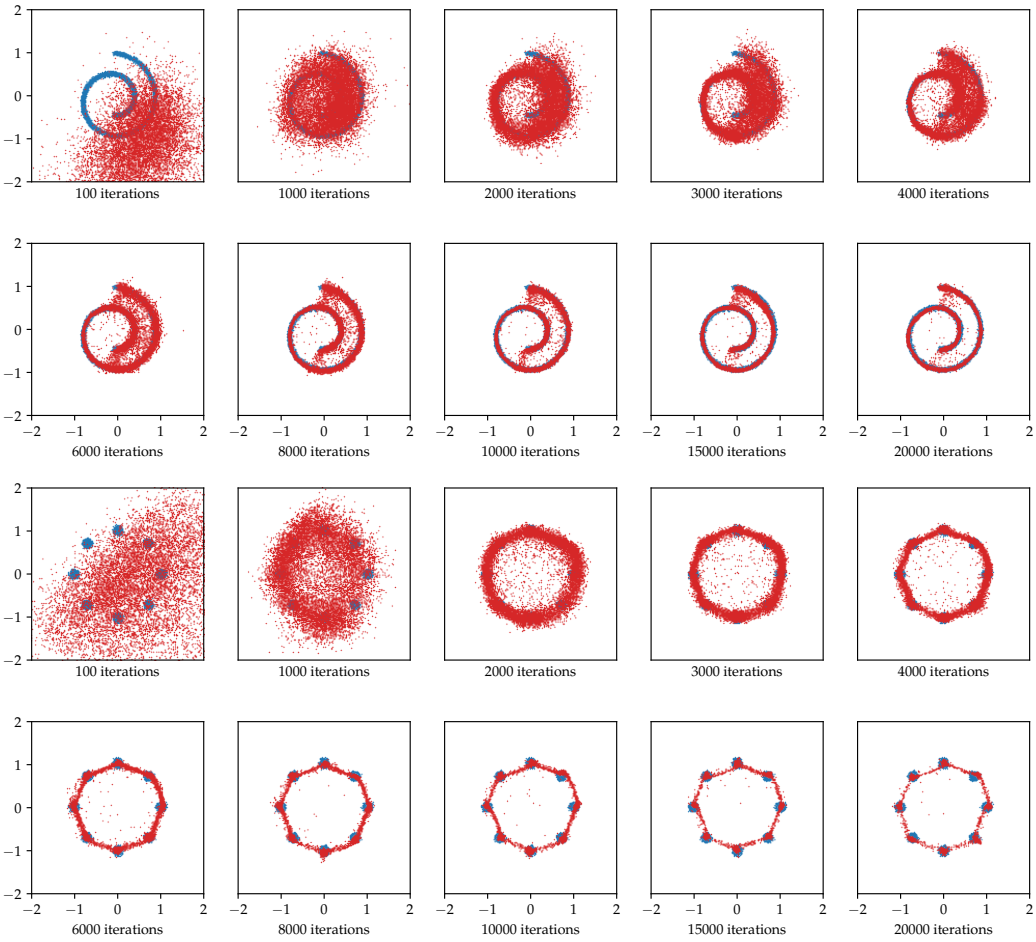


Figure C.4: Generated samples during the course of training from Figure C.2 where the generator is given more capacity.

Appendix D

Batch Normalization

The bath normalization algorithm is described in Algorithm 1. ϵ is a small constant added for numerical stability.

Algorithm 1: Batch Normalization

input : Minibatch $\mathcal{B} = \{x_1, x_2, .., x_m\}$
Parameters γ, β

output: Normalized, scaled and shifted minibatch $\{y_1, y_2, .., y_m\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\widehat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$y_i \leftarrow \gamma\widehat{x_i} + \beta$$

During inference, the activations y_i for a particular sample in the minibatch x_i should not depend on the other $x_j, j \neq i$ in the same minibatch. This is achieved by replacing $\mu_{\mathcal{B}}$ with $\mathbb{E}_{\mathcal{B}}[x]$ which simply is the moving average over the minibatch activation means and $\sigma^2_{\mathcal{B}}$ with $\frac{m}{m-1}\mathbb{E}[\sigma^2_{\mathcal{B}}]$ which is the unbiased variance observed during the training phase. Thus in addition to its learnable parameters γ and β, these scalars are being kept track of.