Generative Adversarial Networks

Contents

0 Introduction

Generative models are an ongoing area of research for both the mathematical and the deep learning community. Due to recent advances in computational power (GPUs and Cloud computing, in particular), it is now possible to explore deep learning methods which were intractable just a few decades prior. When correctly implemented, they are capable of learning the representation of the training data, allowing for the generation of new data, which are indistinguishable from the original.

This essay will focus on a particularly recent yet successful method – the Generative Adversarial Network (GAN)[2]. GANs are a range of unsupervised and semi-supervised machine learning models, characterised by the combination of two feed forward neural networks – one dedicated to creating counterfeits of the training data, and one for detecting counterfeits, in order to generate data which is identical to the data set. This presents a range of practical and technical challenges since it introduces some game-theoretic elements to the usual objective of a feedforward network.

Before diving into a mathematical definition of GANs, this essay will first go through some basics of deep learning, with brief summaries on feed-forward neural nets, dense nets and convolutional networks.

After that, this essay will formally define a GAN, before exploring some of the basic theoretical results which come immediately from the definition. We will then unravel the assumptions made in the basic results in order to have a better understanding of some of the theoretical challenges involved in the analysis of GANs.

We will also implement a GAN in order to learn some simple datasets. This will allow us to observe a GAN in practice, in addition to illustrating some of the practical problems involved.

To follow up on the implementation, we shall quickly techniques which can be used to improve the performance GANs. This will involve some heuristic techniques which have been observed to work, in addition to some methods which can be used to detect failure modes.

We shall conclude the essay by analysing some of the game theoretic aspect of GANs.

This essay is aimed at individuals with a mathematical background and assumes material from the Modern Statistical Methods course and the Bayesian Modelling and Computation course in part III. Hence topics such as gradient techniques, concentration bounds, MCMC and certain other generative models such as belief networks will be stated without proof.

1 Recap on Feed Forward Networks

Let's begin by quickly defining some of the basic components of deep feed-forward networks[1] – the functions that will eventually make up the generator and the discriminator of a GAN. A feed-forward network $\boldsymbol{F} : \mathbb{R}^{d_{in}} \times \Theta_1 \times ... \times \Theta_n \to \mathbb{R}^{d_{out}}$ is a composition of n differentiable functions $\boldsymbol{f}^{(i)} : \mathbb{R}^{d_i} \times \Theta_i \to \mathbb{R}^{d_{i+1}}$, where $i \in \{1, ..., n\}$. Θ_i is a space of parameters and $d_{in} = d_1, ..., d_{i+1} = d_{out}$ are the dimensions of each layer:

$$\boldsymbol{F}(\boldsymbol{x}; \boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_n) = f^{(n)}(...f^{(2)}(f^{(1)}(\boldsymbol{x}; \boldsymbol{\theta}_1); \boldsymbol{\theta}_2)...; \boldsymbol{\theta}_n)$$

Here, n is said to be the *depth* of the network and d_i are said to be the *width*. The image of $f^{(n)}$ is the *output layer*, the images of $f^{(1)}, ..., f^{(n-1)}$ are the *hidden layers* and the domain of $f^{(1)}$ is the input layer. The *capacity* of the $\boldsymbol{F}$ is the set of functions which may be approximated by F, and will depend on the number and type of layers present in the network.

The goal of the feed-forward network is to to approximate some function $\boldsymbol{F}^*$ by altering the parameters $\boldsymbol{\theta}_1 \in \Theta_1, ..., \boldsymbol{\theta}_n \in \Theta_n$. Like most other optimisation problems, the parameters are tuned with respect to some objective function. Since the functions involved are differentiable, it is possible to train the feed-forward network with some variant of gradient descent. Due to the compositional nature of the functions involved, with the correct choice of $f^{(1)}, ..., f^{(n)}$, updating the weights with gradient methods is greatly simplified using the chain rule. In the context of feed forward networks, this gradient descent step is called **Back propogation**.

1.1 Dense Layer

A fully connected layer (also called a dense layer) is the a function $\boldsymbol{f} : \mathbb{R}^n \times \Theta \to \mathbb{R}^m$ of the form:

$$f_i(\mathbf{x}; \boldsymbol{\theta}) = g\left(\theta_{i0} + \sum_{j=1}^{n} \theta_{ij} x_j\right)$$

In this case, $\Theta = \mathbb{R}^{n+1} \times \mathbb{R}^m$, θ_{ij} are the components of $\boldsymbol{\theta}$ and g is some activation function. Note that every component of the input can have some effect on each component of the the output.

The purpose of the activation function is to introduce non-linearity into the system (so that the capacity of the network isn't just a set of linear functions). $g(x)$ is often taken as a differentiable sigmoid function such as $\tanh(x)$ or $\frac{1}{1+e^{-x}}$. These functions have a convenient known derivative, hence the derivative of the entire dense layer can be easily computed via the chain rule.

Another common activation function is the rectifier $g(x) = \max(0, x)$ (the reLU, for rectifier linear unit) or $g(x) = \log(1+e^x)$. In the case of the former, the derivative is discontinuous at zero, so most implementations simply assign a value of zero at that point.

1.2 Convolution Layers and Pooling Layers

A convolutional layer applies a convolution on the data with respect to a series of kernels in order to identify specific spacial features[1]. In the two dimensional case (for example, when the input is in the form of a single channel grey-scale image), the components of the convolutional layer $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^n \times \Theta \to \mathbb{R}^{n-m} \times \mathbb{R}^{n-m} \times \mathbb{R}^l$ satisfies:

$$f_{ijk}(\boldsymbol{x}; K^{(1)}, ..., K^{(l)}) = g\left(\sum_{i',j'=0}^{m-1} K^{(k)}_{i'j'} x_{i+i',j+j'}\right)$$

where the parameters consist of l kernels $K^{(i)} \in \mathbb{R}^m \times \mathbb{R}^m$, where $i = 1, ..., l$. Note that the convolution operation is linear, so this output is usually then passed through a rectifier or some other activation function g, again to introduce non-linearity into the system. Of course, this entire procedure generalises readily to higher dimensions and additional channels in the input.

Convolutional layers are often followed by a pooling layer which effectively down-samples the data with respect to the space dimension with respect of a function (e.g. max-pooling divides the input into a grid and then takes the largest value in each square). The most immediate effect of this pooling is that the dimensionality of the output is significantly reduced. Intuitively, the pooling also removes some of the redundant spacial information from the features output, introducing a small degree of translational invariance.

Note that the capacity of a convolutional layer is strictly smaller than that of a fully connected layer with the same dimensional input and outputs. However, it is significantly easier to train due to the lower dimension of parameters. Convolutional layers are particularly powerful when one is attempting to process image data – intuitively, the kernels will specialise in order to detect features which may be present throughout the image.

Empirically, convolutional layer kernels are known to exhibit specialisation. For example in the first convolutional layer the kernels might learn to identify vertical and horizontal edges, the second layer learns to identify loops and so on. Once one digs deeper into a CNN, it is even possible to identify filters dedicated to non-abstract features such as eyes and noses (if said CNN is to identify faces).

De-convolutional and unpooling layers are effectively the inverse of convolutional and pooling layers. i.e. de-convolutional layers combine a number of feature kernels to form an image, and the unpooling layer up-samples the input. In a DCGAN architecture (Described in [3]), the generator would be implemented using the de-convolutional net, and the discriminator would be implemented with a convolutional net.

2 Generative Adversarial Networks

Now, we are ready to talk about Generative Adversarial Networks. First of all, it is important to make the distinction between a generator and a discriminator. Let (X, Y) be the data and its label, drawn from some distribution. A *discriminator* learns the conditional distribution $f(Y|X)$. In other words, it can act like a classifier – given the data X, it will return with the probability distribution of Y. The strategy involved in training a discriminator is a relatively straightforward supervised task– obtain labelled data and then apply gradient descent.

On the other hand, a *generator* attempts to learn the joint distribution $f(Y, X)$. This may be done either explicitly or implicitly – some generative systems, such as Boltzmann machines, are capable of express the distribution learned. In other systems, a generator is able to directly capable of sampling from the distribution. When this is the case, it is still possible to recover the joint distribution via Parzen kernel density estimation.

This means it will be able to sample from the distribution, and if done correctly, generated data will indistinguishable from data the original source.

In addition to being of interest mathematically, Generative models have a broad range of practical applications. This includes, but is not limited to:

1. **Image Segmentation** – This is the act of partitioning an image into components which may be useful. For example, when looking at medical images, it would useful if one can automatically identify which areas are fat, bone and muscle.
2. **Style Transfer** – This is similar to image segmentation, but here we may convert the style of one image into another. The creative applications are immediate, but this can also be used for supervised models. For example, in the context of self driving cars – instead of training with real data, which is expensive and potentially dangerous, we can train with data created by a generative model[17].
3. **Generating Audio and Video** – This could be used for speech synthesis more realistic audio playback[10].
4. **Reconstruction** – By conditioning on the joint distribution, it is possible to recover areas of images which have been obfuscated.

2.1 Defining a GAN

It's not immediately obvious how one should go about creating a generative model with feed-forward neural networks, since it is difficult to define a sensible objective function. However, we know how to train a discriminator relatively easily. GANs exploit this fact by defining the objective function for a generative model in terms of a discriminator.

Suppose that we have data which is drawn independently from some distribution $p_{\text{data}}(\boldsymbol{x})$ over the space $\boldsymbol{X}$. The goal of our GAN is to learn and sample from this distribution.

To do this, let $\boldsymbol{Z}$ be some latent space, with $p_z(\boldsymbol{z})$ a prior density over this space (preferably a distribution we can already sample from). We also define the generator $G : \boldsymbol{Z} \times \boldsymbol{\Theta}_G \to \boldsymbol{X}$ and the discriminator $D : \boldsymbol{X} \times \boldsymbol{\Theta}_D \to [0, 1]$. G and D are differentiable functions, and in most implementations they will be feed-forward neural networks of some kind, with parameters $\boldsymbol{\Theta}_G$ and $\boldsymbol{\Theta}_D$ respectively.

$G(\boldsymbol{z}, \theta_G)$ maps the random noise sampled from $p_z(\boldsymbol{z})$ to $\boldsymbol{X}$ – in other words, it draws samples from $\boldsymbol{X}$ with respect to some distribution $p_G(\boldsymbol{z})$. Meanwhile, $D(\boldsymbol{x}, \theta_D)$ attempts to return the probability that a particular piece of data is sampled from $p_{\text{data}}(\boldsymbol{z})$ (the real distribution) as opposed to $p_G(\boldsymbol{z})$.

2.1.1 Training a GAN

We define the **value function**:

$$V(G, D) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\phi(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}[\phi(1 - D(G(\boldsymbol{z})))]$$

Where $\phi(x)$ is a monotonic and concave **measuring function** (In Goodfellow's original paper, $\phi(\boldsymbol{x}) = \log(x)$, meaning maximising V is effectively minimising the cross-entropy). Assuming $\phi(\boldsymbol{x})$ increasing, when training D, we want to maximise this quantity, since we want D to maximise the probability of assigning the correct labels to the two classes. However, when training G, we want to minimise $\phi(1 - D(G(\boldsymbol{z})))$ and hence V – we want it to fool the discriminator. By pitting the two networks against each-other, it is hoped that the discriminator will learn features the generator have missed, in turn forcing the generator to incorporate it.

Ideally, training terminates when the discriminator can no longer distinguish between the generated and the actual distribution, and D always return $\frac{1}{2}$. When this happens, it is hoped that $p_G(\boldsymbol{x})$ is now a good approximation to $p_{\text{data}}(\boldsymbol{x})$. With the training is complete, the discriminator may be discarded.

Note that these competing objectives introduces game theoretic concepts into the training process. Instead of simply minimising a value, a GAN is trying to obtain the Nash Equilibrium of the discriminator and the generator.

2.1.2 Other Objectives for G

Note that in the case above, the objectives define a minimax zero-sum game, with the optimum result $(G^*, D^*) = \arg\min_G \max_D V(G, D)$. This allows some of the basic theoretical results in the next section to be proved, but in practice, other objective functions may be used for G. One thing to consider is that in most implementations, G and D are feed forward networks which are trained with gradient methods, hence if the gradients vanish, then training will slow down significantly[3].

Heuristically, even when this is not engineered to be the case, the discriminator tends to train much more quickly than the generator since it has a simpler goal. This means that it is likely (especially early in the training) that $D(G(\boldsymbol{z}))$ will be very small. This could lead to a vanishing gradient in the objective for G $\mathbb{E}_{\boldsymbol{z} \sim p_z} \log(1 - D(G(\boldsymbol{z}))$, preventing the generator from training further.

One way to prevent this is to maximise $\mathbb{E}_{\boldsymbol{z} \sim p_z} \log(D(G(\boldsymbol{z}))$ in when training G instead of minimising $\mathbb{E}_{\boldsymbol{z} \sim p_z} \log(1 - D(G(\boldsymbol{z}))$. This exploits the logarithm's high gradient near 0 in order to allow the generator to continue learning. This breaks the assumption that the game is zero sum, which is required for some of the theoretical results later, but is used practically because of the improvements in performance.

Alternatively, if one wants to do maximum likelihood learning, then it is possible to modify the objective function for this purpose (In general GANs do not do this). In this case, we can try to minimise:

$$-\mathbb{E}_{\boldsymbol{z} \sim p_z} e^{\sigma^{-1}(D(G(\boldsymbol{z})))} = \mathbb{E}_{\boldsymbol{z} \sim p_z} \frac{D(G(\boldsymbol{z}))}{D(G(\boldsymbol{z})) - 1}$$

Where $\sigma(x) = \frac{1}{1+e^{-x}}$. Note that similar to the minimax game, this does not avoid the vanishing gradients problem.

2.1.3 Advantages over other generative models

Other generative models have existed prior to the introduction of GANs, such as the fully visible belief networks, Boltzman nachines and bariational auto-encoders. Each of these methods have their disadvantages, and GANs we invented in part as a response to these shortcomings. We will not discuss the inner workings of these alternate methods, but the basic results can be found in the final chapter of [1].

For example, A GAN is able to sample from a distribution whilst completely avoiding the use of Markov

Chain Monte Carlo, which is necessary in many other generative models (e.g. Boltzmann Machines). This is advantageous, since it is difficult to assess whether a MCMC method has converged to is stationary distribution. This allows GANs to be used for high dimensional data – a domain in which MCMC suffers[14].

Furthermore, the generation of samples is can be easily parallelised – it's just a feed forward network. Whilst the performance of fully visible belief networks are good, it is computationally prohibitive to use in real time settings[14], since the interconnected nature of the conditional probabilities render parallelisation impossible.

Finally, a feedwardward network is very general, hence the method can be adapted to a broad range of problems. This, in addition to the fact that they are capable of generating high data samples mean that GANs are a very promising area of research. However GANs replace these advantages with its own set of problems in convergence, which we will discuss further later.

2.2 Basic theoretical results

2.2.1 Optimal solution for D and G

It's simple, but important, to verify that $p_{\text{data}}(\boldsymbol{x})$ is actually the optimal solution for $p_G(\boldsymbol{x})$ – after all, this is the whole point of a generative model. We first prove this result for $\phi(x) = \log(x)$, using the methods and assumptions described in Goodfellow's original paper[2].

To do this, it is first necessary to find the optimal solution for D.

Proposition 1. *For a given G, if $\phi(x) = \log(x)$, then the optimal solution for D is:*

$$D^*_G(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_G(\boldsymbol{x}) + p_{data}(\boldsymbol{x})}$$

This result backs up the assertion made in the previous section that once the training is complete, the discriminator will end up at $D(\boldsymbol{x}) = \frac{1}{2}$.

Proof. This can be proved simply by plugging the result into the definition of the value function and then substituting one of the integrals.

$$V(G, D) = \int_{\boldsymbol{X}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))d\boldsymbol{x} + \int_{\boldsymbol{Z}} p_z(\boldsymbol{z}) \log(1 - D(G(\boldsymbol{z})))d\boldsymbol{z}$$

Note that $p_z(\boldsymbol{z}) = \left|\det(dG)\right| p_G(G(\boldsymbol{z}))$, where dG is the Jacobian of G. Hence:

$$V(G, D) = \int_{\boldsymbol{X}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_G(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))d\boldsymbol{x}$$

Note that for or non-zero a, b, $a \log(x) + b \log(1 - x)$ attains its maximum at $x = \frac{a}{a+b}$. Hence

$$D^*_G(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_G(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}$$

□

Suppose that it is always possible to train D in order for it to take on this value. The objective for G can may then be defined as:

$$C(G) = \max_D V(G, D)$$

$$= \mathbb{E}_{x \sim p_{\text{data}}} \left[\phi \left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_G(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})} \right) \right] + \mathbb{E}_{x \sim p_{\text{G}}} \left[\phi \left(\frac{p_{\text{G}}(\boldsymbol{x})}{p_G(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})} \right) \right]$$

Theorem 1. *The global minimum of $C(G)$ is attained iff $p_G = p_{data}$ if $\phi(x) = \log(x)$.*

Proof. Recall the definition of Kullback-Leibler divergence:

$$KL(P||Q) = \int_X p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

and that of the Jenson-Shannon divergence:

$$JS(P||Q) = \frac{KL(P||M) + KL(Q||M)}{2}$$

Where $M = \frac{P+Q}{2}$. Note that JS divergence is non-negative, attaining the minimum iff $P = Q$. Now go back to our virtual training criterion $C(G)$:

$$\mathbb{E}_{x\sim p_{\text{data}}}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_G(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}\right)\right] + \mathbb{E}_{x\sim p_{\text{G}}}\left[\log\left(\frac{p_{\text{G}}(\boldsymbol{x})}{p_G(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}\right)\right]$$

$$= -\log 4 + \mathbb{E}_{x\sim p_{\text{data}}}\left[\log\left(\frac{2p_{\text{data}}(\boldsymbol{x})}{p_G(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}\right)\right] + \mathbb{E}_{x\sim p_{\text{G}}}\left[\log\left(\frac{2p_{\text{G}}(\boldsymbol{x})}{p_G(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}\right)\right]$$

$$= -\log 4 + 2 \cdot JS(p_{\text{data}}||p_G) \geq -\log 4$$

Hence a global minimum of $-\log 4$ is attained iff $p_G = p_{\text{data}}$. □

Note that this means that when $\phi(x) = \log(x)$, the GAN is attempting to minimise the JS-divergence of the distributions. Other measure functions can lead to the minimisation of some other distance for distributions. For example, by choosing $\phi(x) = x$ and assuming the neural nets are 1-Lipschitz, we obtain the Wasserstein GAN[6], which instead minimises the Wasserstein distance between the two distributions.

Definition. Recall that if μ and ν are distributions, then Wasserstein distance is defined as:

$$d_W(\mu, \nu) = \sup_{D \text{ 1-Lipschitz}} \left|\mathbb{E}_{x\sim\mu} D(x) - E_{x\sim\mu} D(x)\right|$$

This is also called the earth mover distance, since it is the shortest distance one needs to move material from one density to form to other density.

Proposition 2. *If $\phi(x)$ then the generator is minimising the Wasserstein distance.*

Proof. Just plug $\phi(x) = x$ into the definition:

$$C(G) = \max_{D\text{1-Lipschitz}} \int_{\boldsymbol{X}} p_{\text{data}}(\boldsymbol{x}) D(\boldsymbol{x}) + p_G(\boldsymbol{x})(1 - D(\boldsymbol{x}))d\boldsymbol{x}$$

$$= \sup_{D \text{ 1-Lipschitz}} \left|\mathbb{E}_{x\sim p_{\text{data}}} D(x) - E_{x\sim p_G} D(x)\right| + 1$$

$$= d_W(p_{\text{data}}, p_G) + 1$$

□

More generally, concavity of $\phi(x)$ ensures that when $p_{\text{data}} = p_G$ the minimum is achieved at $2\phi(\frac{1}{2})$.

Now that the uniqueness and existence of a sensible global minimum has been verified, one can attempt to come up with an algorithm to train a GAN which might be able to be implemented practically. After that, we can investigate convergence.

Algorithm 1: Training a GAN

for *number of training iterations* **do**
  for *number of discriminator training iterations* **do**
    Take m samples $(\boldsymbol{z}_1, ..., \boldsymbol{z}_m)$ from $\boldsymbol{Z}$ using the distribution $p_G(\boldsymbol{z})$
    Take m samples $(\boldsymbol{x}_1, ..., \boldsymbol{x}_m)$ from $\boldsymbol{X}$ using the distribution $p_{\text{data}}(\boldsymbol{x})$ Update $D(\boldsymbol{x}, \theta_D)$ using minibatch stochastic gradient ascent with gradient:

$$\frac{1}{m}\nabla_{\theta_D}\sum_{i=1}^{m}\left[\log D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i)))\right]$$

  end
  Take m samples $(\boldsymbol{z}_1, ..., \boldsymbol{z}_m)$ from $\boldsymbol{Z}$ using the distribution $p_G(\boldsymbol{z})$ Update G using minibatch stochastic gradient descent with gradient:

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

end

2.2.2 Convergence for Minimax GAN

Unlike convex optimisation using gradient descent, it is possible for gradient descent in a minimax game to to be non-convergent. For example, if $V = xy$, with agent 1 in control of x trying to maximise V, agent 2 in control of y trying to minimise V, then if the two agents update simultaneously via gradient descent the trajectory would circle the saddle point at the origin.

It is possible to show that Algorithm 1, under some assumptions, converges to the optimal solution detailed in Theorem 1. Note that the additional loop of training for the discriminator means that one can train it for additional time, so we make the assumption that D is always optimal. Again, in practice this is not always used due to computational constraints.

Theorem 2. *If:*

- *G and D have sufficient capacity to approximate p_{data} and D^*_G respectively.*
- *The number of discriminator training iterations allows the discriminator to reach D^*_G.*
- *p_G is updated with respect to:*

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_G(\boldsymbol{x})}[\log(1 - D^*_G(x))]$$

Then p_G converges to p_{data} under Algorithm 1.

Proof. Consider $V(G, D) = U(p_G, D)$:

$$U(p_G, D) = \int_{\boldsymbol{X}} p_{\text{data}}(\boldsymbol{x})\log D(\boldsymbol{x}) + p_G(\boldsymbol{x})\log(1 - D(\boldsymbol{x}))d\boldsymbol{x}$$

$$\implies U(tp_G + (1-t)p'_G, D) = tU(p_G, D) + (1-t)U(p'_G, D)$$

Hence $U(p_G, D)$ is convex.

Note that if a family of functions $f(x, \theta)$ is convex for every θ, and $f(x) = \sup_\theta f(x, \theta)$, then $\partial f^*_\theta(x) \in \partial f(x)$ where $\theta^* = \arg\sup_\theta f(x, \theta)$. ($\partial f^*_\theta(x)$ is the derivative of f^*_θ at x and $\partial f(x)$ is the sub-derivative of f at x. We use the sub-derivative here since $f(x)$ is not necessarily differentiable)

Since U is convex in p_G, we can use this fact to show that gradient descent specified above is identical to updating at the optimal D for a given G.

This means that $C(G) = \sup_D U(p_G, D)$ is convex in p_G with a unique global minimum. These conditions ensure that with sensible updates of p_G (e.g. small or decaying learning rates), the distribution converges to p_{data}. □

These assumptions are not easy to meet practically. Furthermore, the proof optimises the function p_G directly. In actuality, the gradient descent algorithm will be manipulating a feed forward net representing p_G viaa set of parameters. This means the results in convexity which the proof exploits do not apply in general.

Furthermore, these results, which were developed early on in the inception of GANs, assume that the GANs are trained with an infinite number of data samples, in addition to assuming that the neural nets can approximate any function. In Section 3 we will show that this interpretation is somewhat flawed.

2.2.3 Objective for MLE GAN

Finally, the results in Proposition 1 allows us to show why it is possible to modify the objective function for MLE learning[3].

Theorem 3. *Assuming the discriminator is optimal, training the generator by minimising the using the gradient of the objective* $-\mathbb{E}_{\boldsymbol{z}\sim p_z} e^{\sigma^{-1}(D(G(\boldsymbol{z}))} = \mathbb{E}_{\boldsymbol{z}\sim p_z} \frac{D(G(\boldsymbol{z}))}{D(G(\boldsymbol{z}))-1}$ *is equivalent to minimising using the gradient of* $D_{KL}(p_{data}, p_G)$.

Proof. Maximum likelihood estimation is equivalent to minimising the KL divergence between the real and generator distribution:

$$D_{KL}(p_{\text{data}}, p_G)$$

Note that:

$$\frac{\partial}{\partial\theta} D_{KL}(p_{\text{data}}, p_G) = -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \frac{\partial}{\partial\theta} \log p_G$$

Also, assuming $p_G(\boldsymbol{x}) \geq 0$ that and the functions involved are all continuously differentiable:

$$\frac{\partial}{\partial\theta} \mathbb{E}_{x\sim p_G} f(\boldsymbol{x}) = \int_X f(\boldsymbol{x}) \frac{\partial}{\partial\theta} e^{\log(p_G(\boldsymbol{x}))} d\boldsymbol{x}$$

$$= \int_X p_G(\boldsymbol{x}) f(\boldsymbol{x}) \frac{\partial}{\partial\theta} \log(p_G(\boldsymbol{x})) d\boldsymbol{x}$$

Hence the gradients of the objective match that of the KL divergence if $f(\boldsymbol{x}) = -\frac{p_{\text{data}}(\boldsymbol{x})}{p_G(\boldsymbol{x})} = \frac{D_G^*(\boldsymbol{x})}{D_G^*(\boldsymbol{x})-1}$. This gives the desired result. □

3 Limitations of the Basic Results

3.1 Finite Training Data and Generalisation Theory

The basic results in Section 2 were obtained using the "true" distributions of p_G and p_{data}. In practice, these distributions are obfuscated and it is only possible to obtain estimates of them using a finite number of samples from the true distribution. Furthermore, we had assumed that the neural nets can express a broad range of function mapping to $[0, 1]$, which is not realistic.

We want our GAN to be able to minimise the difference between p_G and p_{data} with respect to some distance measure. Since we actually train GANs using some finite set of data, we are implicitly using the empirical distribution in the training process. So in training, it is actually trying minimise the distance between some empirical $\hat{p}_G$ and $\hat{p}_{\text{data}}$.

However, we do not actually know if the convergence between the two empirical distributions imply the convergence of the real distributions. This motivated a study by Arora et al.[4], which investigated the ability of errors calculated from these empirical distributions to generalise to the true distributions.

First, let's define the notion of generalisation.

Definition. Let $\hat{p}_{\text{data}}$ be the empirical distribution of $p_{\text{data}}(x)$ (i.e. the uniform distribution over the training samples) with m samples. Let $\hat{p}_G(x)$ be the empirical distribution of $p_G(x)$ (i.e. the uniform distribution over a set of random samples taken from $p_G(x)$) with a polynomial number of samples. p_G is said to generalise with error ϵ with respect to some distance function $d(\cdot, \cdot)$ if:

$$\left|d(p_G, p_{\text{data}}) - d(\hat{p}_G, \hat{p}_{\text{data}})\right| < \epsilon$$

with high probability over the choice of $\hat{p}_G$.

Here, the support of $\hat{p}_{\text{data}}$ can be interpreted as the set of training data.

This definition encapsulates the ability for the empirical distribution's error to track that of the true distribution. If a GAN is able to generalise with small error, then it is possible to ascertain whether it has learned the true distribution. The polynomial constraint ensures that the training is doable in polynomial time (computing the gradients of a batch of m data takes $O(m)$ time).

We will now see that some of the more commonly used distances, like the Wasserstein distance and JS distance, do not uphold this criterion by finding a counterexample.

Lemma 1. *Suppose μ is an d dimensional multivariate Gaussian with distribution $N(0, \frac{1}{d}I)$ and $\hat{\mu}$ be an empirical distribution of μ, then $d_{JS}(\mu, \hat{\mu}) = \log 2$.*

For the Wasserstein distance, $d_W(\mu, \hat{\mu}) > 1.1$.

Proof. Suppose $x_1, ..., x_m$ are the samples drawn from μ with which $\hat{\mu}$ is defined. μ and $\hat{\mu}$ are defined on different supports, therefore it maximises the value of JS divergence at $\log 2$

An equivalent definition of the Wasserstein distance is:

$$d_W(\mu, \nu) = \inf_{\gamma \in \Pi(\mu,\nu)} \mathbb{E}_{(x,y)\sim\gamma}||x - y||$$

Where $\Pi(\mu, \nu)$ is the set of joint distributions with marginal distributions are μ, ν.
Therefore:

$$d_W(\mu, \hat{\mu}) \geq 1.2\mathbb{P}(\forall i = 1, .., m, ||y - x_i|| \geq 1.2))$$

Note that $||y - x_i||^2$ is sub-exponential. Applying the sub-exponential concentration inequality and then a union bound we obtain:

$$\mathbb{P}(\forall i = 1, .., m, ||y - x_i|| \geq 1.2) \geq 1 - me^{-\Omega(d)}$$

Where $f(x) = \Omega(g(x)) \implies \lim_{n\to\infty} \sup \left|\frac{f(x)}{g(x)}\right| > 0$. For m bounded by some polynomial in d, $me^{\Omega(d)} = o(1)$, so $d_W(\mu, \hat{\mu}) > 1.1$. □

This is not the strongest bound, but it sufficient for the purposes of the proof.

Theorem 4. *Let μ , ν be Gaussian distributions $N(0, \frac{1}{d}I)$ and let $\hat{\mu}$, $\hat{\nu}$ be empirical distributions of μ , ν respectively with m samples. Then:*

$$d_{JS}(\mu, \nu) = 0, \quad d_{JS}(\hat{\mu}, \hat{\nu}) = \log 2$$

and

$$d_W(\mu, \nu) = 0, \quad \mathbb{P}(d_W(\hat{\mu}, \hat{\nu}) \geq 1.1) \geq 1 - m^2 e^{-\Omega(d)}$$

Proof. For the Jensen-Shannon distance, the result follows trivially because $\hat{\mu}$ and $\hat{\nu}$ will have no intersection in their supports.

The result is more complicated for the Wasserstein distance, but like the previous theorem, it can be obtained by applying the appropriate union bounds and concentration inequalities. Consider $x, y \sim N(0, \frac{1}{d}I)$. Note that $||x - y||_2^2$ is a rescaling of a χ^2 distribution with mean 2, and is therefore sub-exponential. Hence:

$$\mathbb{P}(||x - y||_2^2 - 2 \leq -\epsilon) \leq e^{-\Omega(\epsilon^2 d)}$$

Fixing ϵ, it is possible to union bound over the m^2 pairs distances between the supports of $\hat{\mu}$ and $\hat{\nu}$.

$$\begin{aligned} 1 - \mathbb{P}(\forall i \in \text{Supp}(\hat{\mu}), j \in \text{Supp}(\hat{\nu}), ||i - j||_2^2 - 2 \leq -\epsilon) &\geq 1 - m^2 \mathbb{P}(||x - y||_2^2 - 2 \leq -\epsilon) \\ &\geq 1 - m^2 e^{-\Omega(d)} \end{aligned}$$

Therefore with probability $1 - m^2 e^{-\Omega(d)}$, the closest distance between the two the supports is at least one. From the earth mover interpretation of Wasserstein distance, this implies that $d_W(\hat{\mu}, \hat{\nu}) > 1.1$. **(REVIEW THIS LATER – TYPO IN PAPER)** □

Because JS divergence and Wasserstein distance do not generalise, it is possible to overfit with a polynomial number of samples. For example, suppose $p_{\text{data}} = \mu$, and the generator simply memorises the training data (i.e. $p_G = \hat{p}_{\text{data}}$), then $d_W(\hat{p}_G, \hat{p}_{\text{data}}) \approx 0$ but $d_W(p_G, p_{\text{data}}) \approx 1$. Hence it is not possible to say if the GAN has learned the distribution, even if the empirical results are close together.

Note that the proofs for the JS divergence relied on the fact that the empirical distributions have discrete. In practice, there will be a small amount of noise when sampling, and so the supports involved may overlap. This means it is prudent to check the result for this case too.

Theorem 5. *Let $\tilde{\mu}$ and $\tilde{\nu}$ be the convolution of $\hat{\mu}$, $\hat{\nu}$ with a Gaussian $N(0, \frac{\sigma^2}{d}I)$, then if $\sigma < \frac{c}{\sqrt{\log m}}$ for sufficiently small c, then:*

$$\mathbb{P}\left(d_{JS}(\tilde{\mu}, \tilde{\nu}) > \log 2 - \frac{1}{m}\right) \geq 1 - m^2 e^{-\Omega(d)}$$

$\tilde{\mu}$ and $\tilde{\nu}$ are the average of a number of normal distributions centred at the points in the supports of $\hat{\mu}$, $\hat{\nu}$ respectively.

Proof. Let $\rho_1(x)$ and $\rho_2(x)$ be the density functions of $\tilde{\mu}$ and $\tilde{\nu}$ respectively. Define:

$$g(x) = \rho_1(x) \log \frac{2\rho_1(x)}{\rho_1(x) + \rho_2(x)} + \rho_2(x) \log \frac{2\rho_2(x)}{\rho_1(x) + \rho_2(x)}$$

This is effectively double the function one integrates over in order to determine the JS divergence of $\tilde{\mu}$ and $\tilde{\nu}$. Let:

$$z_x \sim \text{Bern}\left(\frac{\rho_1(x)}{\rho_1(x) + \rho_2(x)}\right)$$

Define the entropy of a random variable X to be $H(X) = E(-\log(\mathbb{P}(X)))$. Note that:

$$(\rho_1(x) + \rho(2))(\log 2 - H(z_x)) = g(x)$$

Entropy of a Bernoulli variable is bounded between 0 and $\log 2$, so $0 \leq g(x) \leq \log 2(\rho_1(x) + \rho(2))$.

Now let B be the union of balls of radius 0.2 near the samples from $\hat{\mu}$ and $\hat{\nu}$. As proved in the previous theorem, the probability of all these distances being at least 1 is $1 - m^2 e^{-\Omega(d)}$, and so do not intersect with high probability. Also, since the distribution is locally Gaussian, the choice of σ means $\frac{\max(\rho_1(x),\rho_2(x))}{\min(\rho_1(x),\rho_2(x))} \geq m^2$ inside the balls. Finally, if one is sampling from $\frac{\tilde{\mu}+\tilde{\nu}}{2}$, then the probability of being inside the union of the balls is at least $1 - \frac{1}{2m}$, since the balls are centred around the peaks. Hence $\forall x \in B$:

$$H(z_x) \leq \frac{m^2}{1+m^2} \log \frac{m^2+1}{m^2} + \frac{\log(1+m^2)}{1+m^2}$$

$$= O\left(\frac{\log m}{m^2}\right) = o\left(\frac{1}{m}\right)$$

Using this result:

$$d_{JS}(\tilde{\mu}, \tilde{\nu}) = \frac{1}{2} \int g(x)dx \geq \frac{1}{2} \int_{x \in B} g(x)dx$$

$$= \int_{x \in \beta} \frac{(\rho_1(x) + \rho(2))}{2} \left(\log 2 - o\left(\frac{1}{m}\right)\right) dx$$

$$= \log 2 - \frac{1}{m} - o\left(\frac{1}{m}\right) \geq \log 2 - \frac{1}{m}$$

□

This means that even if there is a small amount of noise added to the empirical distributions (due to sampling or intentionally with kernels), JS divergence does not generalise.

3.2 Limited Capacity and Neural Net Distance

Recall that the goal of the game is $\min_G \max_D V(G, D)$. Observe that in Section 2, we were only able to invoke the JS-divergence because we had a free choice of D. This is also the case for the Wasserstein GAN, where it is assumed that D can take on any function which is 1-Lipschitz. Because D is actually a neural net with limited capacity, the generator is in fact trying to minimise some other quantity. This motivates the definition of the neural net distance.

In order to do this, we first need to define F-distance, which generalises the notion of distances in a manner which is conducive for investigating GANs.

Definition. Let F be a class of function such that:

1. $f \in F \implies f : \mathbb{R}^d \to [0, 1]$
2. $f \in F \implies 1 - f \in F$

Then the F-divergence with respect to ϕ is

$$d_{F,\phi}(\mu, \nu) = \sup_{D \in F} \mathbb{E}_{x \sim \mu}(\phi(D(x))) + \mathbb{E}_{x \sim \nu}(1 - \phi(D(x))) - 2\phi\left(\frac{1}{2}\right)$$

Note that this is very similar to the objective function of the discriminator with the measure function ϕ, shifted by $2\phi\left(\frac{1}{2}\right)$ – the value attained when the two distributions are identical.

By choosing different F, we can show that F-distance is equivalent to other distances. For example, by choosing $\phi(x) = \log(x)$ and F the set of all functions mapping to $[0, 1]$, we recover the JS-divergence.

Now we impose a restriction on the capacity of a neural net cannot represent all functions. This gives us our definition for the neural net Distance.

Definition. If F is a set of neural networks with at most p parameters, then $d_{F,\phi}$ is informally defined as the **neural net distance**. We make the additional assumption that the distance is L_ϕ-Lipschitz and takes values in $[-\Delta, \Delta]$.

This is a more realistic representation of what a real GAN is trying to minimise, since it takes into account the limitations of the neural networks making up its components. Furthermore, it is possible to show that this new distance generalises.

Theorem 6. *Suppose $F = \{D_v, v \in V\}$ is the set of L-Lipschitz discriminators with parameters v. p is the number of parameters in v. Again, let $\hat{\mu}$, $\hat{\nu}$ be empirical distributions of μ , ν respectively with m samples. There exists c such that if $m \geq \frac{cp\Delta^2 \log(LL_\phi p/\epsilon)}{\epsilon^2}$.*

$$\mathbb{P}(|d_{F,\phi}(\mu, \nu) - d_{F,\phi}(\hat{\mu}, \hat{\nu})| \leq \epsilon) \geq 1 - e^{-p}$$

Proof. First, we want to prove that for every discriminator:

$$|E_{x\sim\mu}(\phi(D_v(x))) - E_{x\sim\hat{\mu}}(\phi(D_v(x)))| \leq \frac{\epsilon}{2}$$

$$|E_{x\sim\nu}(\phi(1 - D_v(x))) - E_{x\sim\hat{\nu}}(\phi(1 - D_v(x)))| \leq \frac{\epsilon}{2}$$

If these two inequalities are true for every discriminator, then the result follows by restricting to the optimal discriminator and then applying the triangle inequality.

Now to prove the first inequality using a Chernoff bound:

$$\mathbb{P}\left(\left|E_{x\sim\mu}(\phi(D_v(x))) - E_{x\sim\hat{\mu}}(\phi(D_v(x)))\right| \geq \frac{\epsilon}{4}\right) \leq 2e^{-\frac{\epsilon^2 m}{2\Delta^2}}$$

Let χ to be a $\frac{\epsilon}{8LL_\phi}$-net of V (every point in χ is within a distance of $\frac{\epsilon}{8LL_\phi}$ of a point in V). The size of this net can be constructed such that $\log|\chi| \leq O(p \log(LL_\phi p/\epsilon))$. Hence if you union bound over all $v \in \chi$, if $m > \frac{Cp\Delta^2 \log(LL_\phi p/\epsilon)}{\epsilon^2}$ with sufficiently large C you have:

$$\mathbb{P}\left(\exists v \in \chi \text{ s.t. } \left|E_{x\sim\mu}(\phi(D_v(x))) - E_{x\sim\hat{\mu}}(\phi(D_v(x)))\right| \leq \frac{\epsilon}{4}\right) \leq 1 - e^{-p}$$

Now it is possible to use the triangle inequality to prove the result for all $v \in V$. For all $v \in V \exists v' \in X$ s.t. $||v - v'|| \leq \frac{\epsilon}{8\epsilon LL_\phi}$. Using the Lipschitz properties of ϕ and D_v:

$$|\mathbb{E}_x(\phi(D_v(x)) - E_x(\phi(D_{v'}(x))| \leq \frac{\epsilon}{8}$$

Hence:

$$\begin{aligned}
&|E_{x\sim\mu}(\phi(D_v(x))) - E_{x\sim\hat{\mu}}(\phi(D_v(x)))| \\
&\leq |E_{x\sim\mu}(\phi(D_{v'}(x))) - E_{x\sim\hat{\mu}}(\phi(D_{v'}(x)))| \\
&+|E_{x\sim\mu}(\phi(D_v(x))) - E_{x\sim\mu}(\phi(D_{v'}(x)))| + |E_{x\sim\hat{\mu}}(\phi(D_v(x))) - E_{x\sim\hat{\mu}}(\phi(D_{v'}(x)))| \\
&\leq \epsilon\left(\frac{1}{4} + \frac{1}{8} + \frac{1}{8}\right) = \frac{\epsilon}{2}
\end{aligned}$$

The inequality $|E_{x\sim\nu}(\phi(1 - D_v(x))) - E_{x\sim\hat{\nu}}(\phi(1 - D_v(x)))| \leq \frac{\epsilon}{2}$ can be proved using the same method, and combining this with the previous inequality gives the result. □

The corollary to this result gives us generalisation throughout the training process.

Corollary 1. *Suppose $G^{(1)}, G^{(2)}, ..., G^{(K)}$ are the K during the K training steps , where $\log K << d$ and $\log K \leq p$ Then there is some constant c such that $m > \frac{Cp\Delta^2 \log(LL_\phi p/\epsilon)}{\epsilon^2}$ implies that for all $t = 1, ..., K$:*

$$\mathbb{P}\left(|d_{F,\phi}(p_{data}, p_{G^{(t)}}) - d_{F,\phi}(\hat{p}_{data}, \hat{p}_{G^{(t)}})| \leq \epsilon\right) \geq 1 - e^{-p}$$

Proof. Consider the inequalities:

$$|E_{x\sim p_{\text{real}}}(\phi(D_v(x))) - E_{x\sim\hat{p}_{\text{real}}}(\phi(D_v(x)))| \leq \frac{\epsilon}{2}$$

$$|E_{x\sim p_{G^{(t)}}}(\phi(1-D_v(x))) - E_{x\sim\hat{p}_{G^{(t)}}}(\phi(1-D_v(x)))| \leq \frac{\epsilon}{2}$$

The first one does not depend on G, and hence is held with high probability as with the proof of the previous theorem. In the second inequality, $\hat{p}_G^{(t)}$ is generated with fresh sample each time, hence the inequality also holds. Combining the two results using the triangle inequality as in the proof of the previous theorem gives the result. □

This means that convergence in neural net distance for the empirical distributions imply convergence in the true distributions.

However, note that the neural net distance also captures the discriminator's limitations. Intuitively, if a discriminator is unable to discern a lot of features features, it is possible for it to be fooled by distributions which are completely unlike the real one.

This means that just because the neural net distance between two distributions is small, doesn't mean they are similar. This can be captured by the following corollary to the above theorem.

Corollary 2. *Let $\hat{\mu}$ be the empirical distribution of μ with at least m samples. Then there exists some constants c such that for $m > \frac{Cp\Delta^2\log(LL_\phi p/\epsilon)}{\epsilon^2}$:*

$$\mathbb{P}(d_{F,\phi}(\hat{\mu},\mu) \leq \epsilon) \geq 1 - e^{-p}$$

Proof. Follows by applying the triangle inequality to the result in addition to the two inequalities in the proof of the previous theorem. □

This means that, because of the limited capacity of the neural nets, once m is of order $\tilde{O}(\frac{p}{\epsilon^2})$ it will be unable to distinguish the between the real and empirical distribution [1]. This is why sometimes GANs can perform poorly even when the generator wins.

[1] Notation: $f(x) = \tilde{O}(g(x)) \implies \exists k > 0 s.t. f(x) = O(g(x)\log g(x))$

4 Implementation

In this section, some simple examples of GANs will be trained in order to demonstrate its practical capabilities, in addition to illustrating some of the common problems which might arise when training.

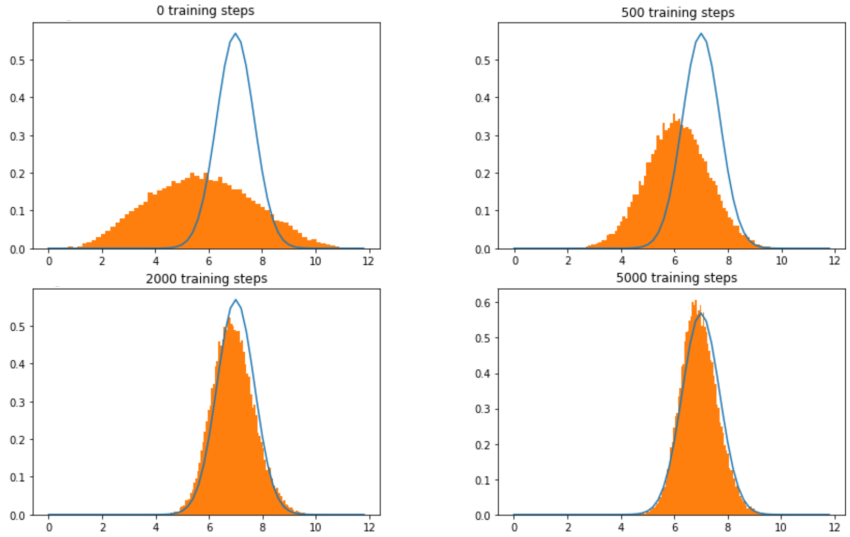
4.1 Toy Examples with Exponential and Normal Distributions

Whilst it is possible to train GANs to do complicated tasks such as generating or reconstructing images, it is much more instructive to first demonstrate a GAN learning a known distribution.

A simple GAN was implemented with Keras 2.0 (Tensorflow backend) and then trained with data sampled from some known, one-dimensional distribution (In this case, a normal and exponential distribution). The code is equivalent to Algorithm 1, other than the fact that the Generator training step uses tries to maximise $\log(D(G(\boldsymbol{z}))$ instead of minimising $\log((1 - DG(\boldsymbol{z}))$ – as stated before, at the start of training, this objective gives stronger gradients for the generator to train on.

The generator and discriminators are both fully connected dense nets. ReLUs were used in the exponential distribution, and sigmoid functions were used for learning the normal distribution. Observing fig. 1 and fig. 2, it is clear that GANs are able to replicate the distributions, albeit with much higher computation costs than more traditional techniques such as kernel density estimation.

Figure 1: Example of a GAN learning a normal distribution with parameters $\mu = 7, \sigma = 0.7$



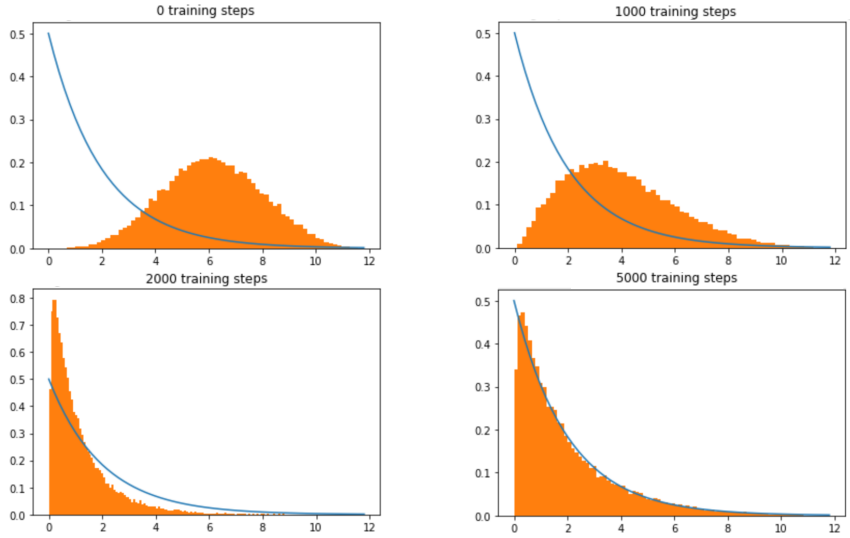
4.2 MNIST DCGAN

A more informative dataset would be the MNIST, a dataset consisting of 60,000 labelled 28×28 images of handwritten single digit numbers. A GAN was trained using this dataset with a AWS EC2 p2.xlarge server. This GAN used a DCGAN model, hence the generator is implemented with a deconvolutional net, whereas the discriminator is implemented with a convolutional net. The code of this GAN was modified from [18].

In this case, the discriminator is a convolutional network with ReLU activation, and the generation is a deconvolutional net, also with ReLU activation. The GAN is trained with the ADAM gradient descent algorithm [20] for 10000 epochs, at which point the generator wins (See fig. 4 for the discriminator accuracy for the first 5000 epochs, and the discriminator and generator losses are displayed in fig. 5).

Some results are displayed in fig. 3. From inspection, the GAN has performed admirably. Even with a

Figure 2: Example of a GAN learning an exponential distribution with parameters $\mu = 2$



relatively modest amount of computational power, it can create some pretty convincing images of handwritten characters.

Figure 3: Left: MNIST DCGAN, 1000 training iterations. Middle: 5000 training iterations. Right: MNIST Training data

Figure 4: Left: MNIST DCGAN Training Loss, moving average window size 10

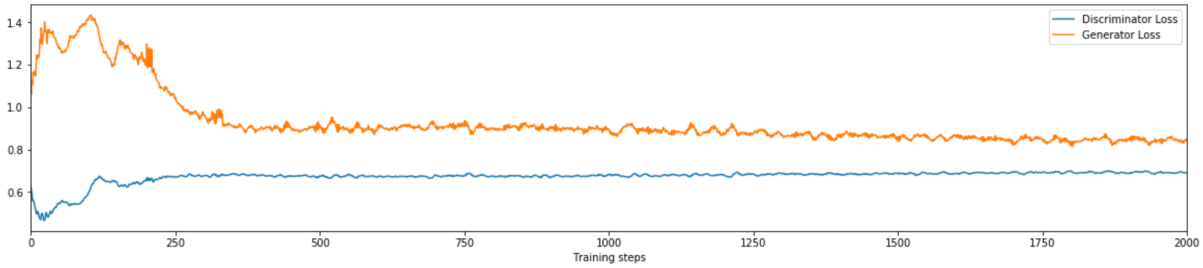
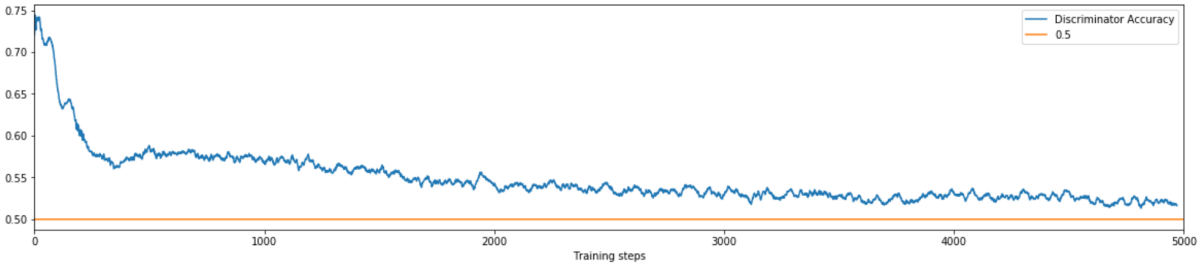


Figure 5: Left: MNIST DCGAN discriminator accuracy converging to 0.5, moving average window size 30



5 Training Tips and Debugging

5.1 Heuristic Tips When Training

GANs are very difficult to train properly, due to the fact that it is dependent upon the Nash Equilibrium. It is very sensitive to initial conditions, and often do not converge.

This section will go over some heuristic methods to improve the performance of GANs. These methods are observed to work well enough to warrant attention. However, because they often do not have a theoretical basis, and do not always work, this section will be kept brief. Most of the results here are documented in [9].

1. **Training with Labels** – This was observed to work in a study by [REFERENCE]. Intuitively, the class information may help the GAN to learn the more pertinent features of each class.

2. **One-Sided Label Smoothing** – Instead of training the discriminator with the labels of 1 for the real, train it with labels which are close to 1. This prevents the discriminator from classifying with values which are too close to 1 or 0, which could lead to very large gradients in the training.

3. **Use Dropout** – Dropout is a technique used to regularise a neural network, where hidden nodes are removed stochastically (usually Bernoulli) during training . This prevent the hidden nodes from collaborating, and it is hoped that they will individually learn more useful features. This is a general tip for training neural networks, and is also used in training GANs.

4. **Virtual Batch Normalisation** – Batch normalisation is a method to reduce instability in a neural network, by normalising the mean and standard deviations of the activations of a particular layer in the neural network on a per batch basis. This prevents values which are very large from dominating the training process by causing imbalanced gradients, which is dangerous because the large values can cascade through the neural network and cause instability. The mean and standard deviation is then also used in training, so no information is lost.

   This method works well, but for GANs need a different minibatch at each step of training, which

can cause the mean and standard deviation values used to reparametrise the model to fluctuate. To avoid this, Virtual Batch Normalisation uses the union of a reference batch and the sample batch in order to reduce the effects of these fluctuations. Again, this method is observed to work well by [REFERENCE].

5. **Feature Matching** – Feature matching is a technique which prevents the generator from from over-training on the discriminator detailed by [REFERENCe]. This is done by minimising some :

$$|\mathbb{E}_{x\sim p_{\text{data}}} f(\boldsymbol{x}) - \mathbb{E}_{x\sim p_G} f(\boldsymbol{x})|_2^2$$

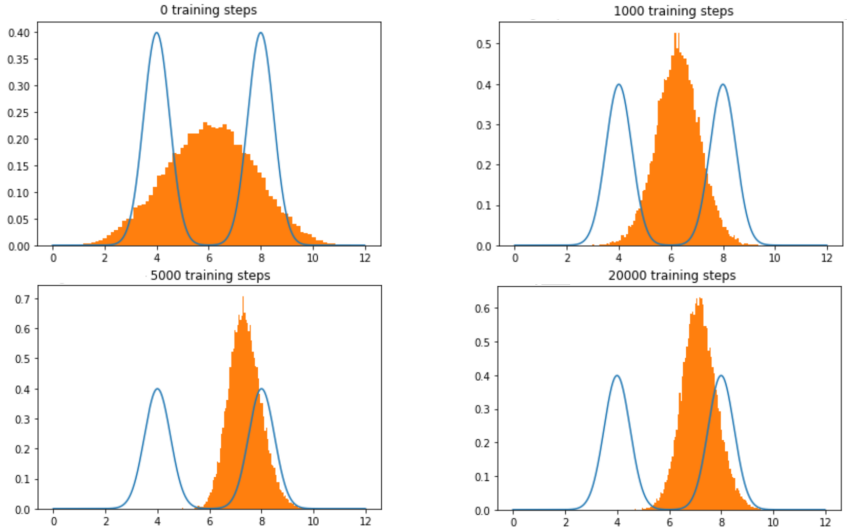
Where $f(x)$ is some intermediate layer of the discriminator. Again, there is little theoretical basis for the method's success, but it was observed to be effective in reduce instability by the authors.

5.2 Common Failure Modes

It was proved in [SECTION] that minimax GANs are able to converge. However, due to the assumptions made in that section, this is not the case in practice.

One common failure mode is mode collapse, also called the Helvetica Scenario[2] in Goodfellow's original

Figure 6: Mode collapse when learning two normal distributions with $\mu_1 = 4, \mu_2 = 8$ and $\sigma = 0.5$



paper. This occurs when the generator allocates too high a probability to a particular region of the domain. When multiple points in Z are mapped to the same point, it is difficult for the generator to disentangle them.

This is most visible when attempting to learn a distribution with two peaks which are far apart (see fig. 6). When this happens:

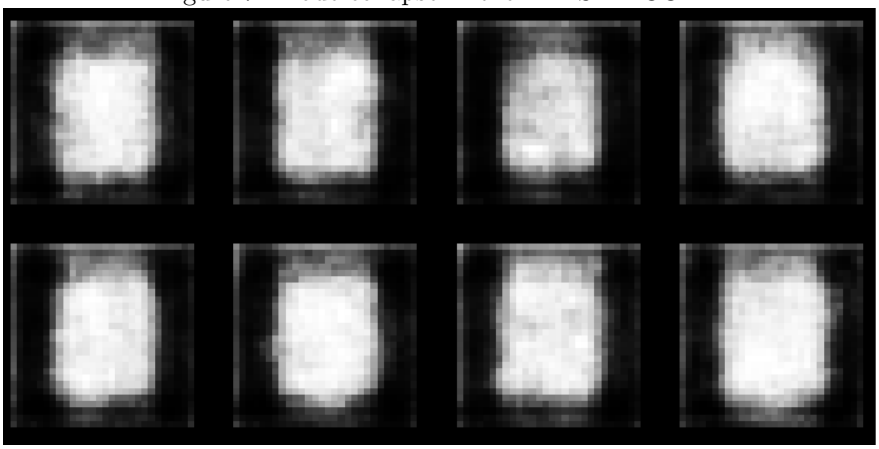
1. The discriminator makes its decision based the existence of samples on one mode.
2. The generator accumulates probability at that mode.
3. The discriminator starts to makes its decision based on the other mode. As the two nets fight, the learned distribution either gets stuck on current mode or move to new mode, but never learns both.

[2]This appears to be a reference to *Look Around You*, a series of satirical videos made by the BBC. In particular, in the pilot episode *Calcium*, a high concentration of calcium in the same place results in the amalgamation of a subject's facial features. Meanwhile, in the context of GANs undergoing mode collapse when trained over facial datasets such as CelebA, a high concentration of probability mass in the same place results in the amalgamation of a subject's facial features.

This may happen for a number or reasons. For example, if the discriminator or generator lack the capacity to represent the real distribution then of course it can only represent a simpler function. Further, suppose we swap the order of the optimisation, so instead of finding $\min_G \max_D V(G, D)$ we are doing $\max_D \min_G V(G, D)$. When we do this, the generator is incentivised to concentrate all the probability to a single point where it can fool the discriminator. Since we are not always optimising D when we're training in practice, it is possible during training this is what's actually happening.

It is possible for a generator to simply memorise the training data, leading to over-fitting. This may

Figure 7: Mode collapse in the MNIST DCGAN



occur is the training dataset if too small.

5.2.1 Heuristic Methods for Detecting Failure

Some simple methods for detecting over-fitting exists, but they are heuristic in nature. For example, for a given data point used for training, one can sample from the generated distribution and observe the closest generated data point to said training data.

We do not want the GAN to simply memorise the input data, so if the closest generated data point from a modest sample is almost identical to a piece of training data, then it is a warning sign that the GAN has overfit. To demonstrate this technique, it was implemented on the MNIST DCGAN (See fig. 8). With this technique, one can see that the generated data is not identical, which suggests that the GAN has not simply memorised the training data.

Alternatively, one can interpolate between different points in the space to observe the shift from one

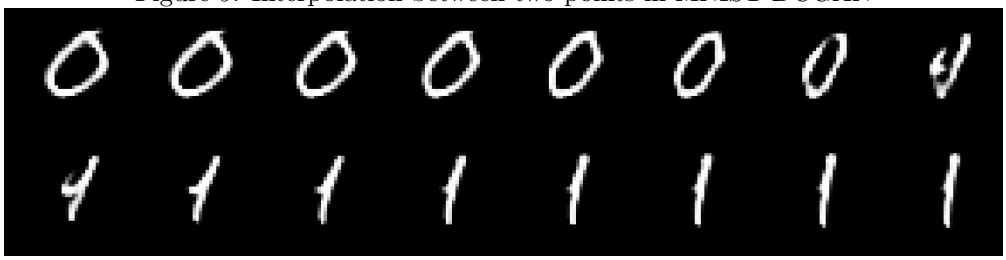
Figure 8: Data generated MNIST DCGAN closest to training data

class to another. If there is a smooth transition with lots of different images, it is hoped that the GAN has learned the representation. The method is implemented in fig. 9 for the MNIST DCGAN.

For mode collapse specifically, it is possible to modify the generators training such that it (Metz et al.) minimises:

$$V(\theta_G, \theta_D^k)$$

Figure 9: Interpolation between two points in MNIST DCGAN



Where θ_D is the parameters of the discriminator after training for k steps of gradient ascent. This gives us an **Unrolled GAN** – by allowing the generator to look into the future, it is able to anticipate the discriminator's actions, potentially avoiding Helvetica.

Note that this is different from simply training a discriminator k steps before training the generator, like in Algorithm 1. Here, the discriminator is only updated once using the usual gradients.

5.3 Birthday Paradox Test

This section will describe a relatively simple but powerful test for detecting mode collapse and over-fitting.

One symptom of failure for the distribution learned by a GAN is that the learned distribution fails to have the desired support size, or is simply too concentrated on a small subset of the domain. This can be a result of local minima in the training, insufficient training data samples or simply insufficient training time.

Instead of attempting to sample the entire distribution,[5] proposed a more computationally inexpensive technique to estimate the support size of a distribution, with the intuition grounded in the famous birthday paradox.

Theorem 7. *(The Birthday Paradox) Suppose $k << N$ i.i.d. samples are drawn form a probability distribution defined on some domain Ω. Also, suppose there exist $S \subseteq \Omega$ such that $\mathbb{P}(x \in S) \geq \rho$ and $|S| = N$. Then:*

$$\mathbb{P}(\textit{At least one collision}) \geq 1 - e^{\frac{-k^2\rho}{2N}}$$

Proof.

$$\mathbb{P}(\text{At least one collision after } k \text{ drawn}) \geq \mathbb{P}(\text{At least one collision in } S \text{ after } k \text{ drawn})$$

$$= 1 - \mathbb{P}(\text{No collision in } S \text{ after } k \text{ drawn}) = 1 - k! \sum_{\substack{\omega \in \Omega \\ |\omega| = k}} \prod_{i \in \omega} \mathbb{P}(i)$$

By the method of Lagrange multipliers, it is possible so show that, when drawing from a distribution over some discrete space Φ:

$$k! \sum_{\substack{\phi \subseteq \Phi \\ |\phi| = k}} \prod_{i \in \phi} \mathbb{P}(i) \leq \prod_{i=0}^{k-1} (1 - \frac{i}{n})$$

with equality is achieved if the probability given that something is chosen from S is uniformly distributed. By first applying this inequality to S, and then applying the inequality $e^x \geq 1 + x$:

$$\implies \mathbb{P}(\text{At least one collision}) \geq 1 - \prod_{i=0}^{k-1} (1 - \frac{i\rho}{n})$$

$$\geq 1 - \prod_{i=0}^{k-1} e^{\frac{-i\rho}{n}} \geq 1 - e^{\frac{-k(k-1)\rho}{2n}}$$

□

This is the formal statement (and generalisation) of the Birthday Paradox – the fact that the probability of a collision is counter-intuitively high, and the k only needs to be order of $\sqrt{n}$ in order for there to be high probability to a collision. However, in order to test the support size of a distribution (or at least, the degree of concentration in a distribution), we want to know the size of S when the probability of a distribution is known.

Theorem 8. *Suppose k i.i.d. samples are drawn form a probability distribution defined on some domain Ω, and there is probability of γ for there to be collisions in these samples. Then for $\rho = 1 - o(1)$ there exists $S \subseteq \Omega$ such that $\mathbb{P}(x \in S) > \rho$ and:*

$$|S| \leq \frac{2k\rho^2}{\left(-3 + \sqrt{9 + \frac{24}{k}\log\frac{1}{1-\gamma}}\right) - 2k(1-\rho)^2}$$

This result gives the degree of concentration of a distribution given the probability of collision. This is makes for a useful test for mode collapse, since the collision probability can be measured relatively easily.

To prove this theorem, we require some other results and definitions. First, define $T = \inf\{t \geq 2 : X_t \in \{X_1, ..., X_{t-1}\}\}$, the time taken for the first collision to occur. Also, define $\beta = \frac{1}{\mathbb{P}(T=2)} = \frac{1}{\sum_{x\in\Omega}\mathbb{P}(x)^2}$. Note that this value is maximised at $|\Omega|$ if the distribution is perfectly uniform and minimised at 1 if the distribution is concentrated at one value. Finally, let $P_{m,k} = \prod_{i=0}^{k-1}(1 - \frac{i}{n})$, the probability of no collisions after k draws.

Lemma 2. *If $\beta \geq 1000$, and $0 \leq k \leq 2\sqrt{\beta \log \beta}$, then:*

$$\mathbb{P}(T \geq k) \geq e^{-\frac{k^2}{2\beta} - \frac{k^3}{6\beta^2}}$$

This result involves a large number of supporting lemmas, whose results can be found in [8].

Proof. We'll first show this result for the uniform case, and then generalise it to all distributions which satisfy the conditions. Let the uniform distribution have support $n \geq 1000$ (recall $\beta = n$ if the distribution is uniform), and $k \leq 2\sqrt{n \log n}$. To prove this result, we use the Taylor series $-\log(1-x) = \sum_{i=1}^{\infty}\frac{x^i}{i}$:

$$-\log\big(\mathbb{P}(T > k)\big) = -\log\prod_{j=0}^{k-1}\left(1 - \frac{j}{n}\right)$$

Expand out and swap the sums:

$$\sum_{j=0}^{k-1}\sum_{i=1}^{\infty}\frac{j^i}{in^i} = \frac{1}{n}\sum_{j=0}^{k-1} j + \frac{1}{2n^2}\sum_{j=0}^{k-1} j^2 + ...$$

The first term allows us to recover the bound we obtained in theorem 7. However, instead we use $\sum_{j=0}^{k-1} j^m \leq \frac{k^{m+1}}{m+1}$ from lemma 5 on all but the last terms:

$$\begin{aligned}
&\leq \frac{k(k-1)}{2n} + \frac{k^3}{3}\cdot\frac{1}{2n^2} + \frac{k^4}{4}\cdot\frac{1}{3n^3} + ... \\
&= \frac{k(k-1)}{2n} + \frac{k^3}{6n^2} + \sum_{i=3}^{\infty}\frac{k^{i+1}}{i(i+1)n^i} \\
&\leq \frac{k(k-1)}{2n} + \frac{k^3}{6n^2} + \frac{k^4}{12n^2}\sum_{i=0}^{\infty}\frac{k}{n}
\end{aligned}$$

With the given constraints on n and k, the geometric series will sum to less than $\frac{6}{5}$. Hence:

$$-\log\big(\mathbb{P}(T > k)\big) \leq \frac{k(k-1)}{2n} + \frac{k^3}{6n^2} + \frac{k^4}{10n^2}$$

$$\leq \frac{k^2}{2n} + \frac{k^3}{6n^2} + \frac{k}{n}\left(\frac{k^3}{10n^2} - \frac{1}{2}\right)$$

Apply the constraints again:

$$-\log P_{n,k} \leq \frac{k^2}{2n} + \frac{k^3}{6n^2}$$

Which gives us the desired result. Note that this holds even if n is not an integer.

Now it is possible to consider the general case. We're going to use the worst case stated in lemma 6 to prove this result. Note that $T > 1$, so this statement holds trivially for $k < 2$. Reuse the definitions of m ω from that lemma, and define $\alpha = \sqrt{\frac{\frac{m}{\beta}-1}{m-1}}$, $x = \frac{1-\alpha(m-1)}{m}$, $y = \frac{1+\alpha}{m}$ such that the distribution in lemma 6 can be restated as:

$$\mathbb{P}(\omega_1) = x, \qquad \mathbb{P}(\omega_i) = y \quad i = 2, ..., m$$

The distribution is uniform except at ω_1. Hence if there are no collision after $k > 2$ samples, then either are no samples in ω_1 or there is exactly one. The probability of the former case is $(1-x)^k$, with conditional probability $P_{m-1,k}$. There is a $kx(1-x)^{k-1}$ probability of the second case, with conditional probability $P_{n-1,k-1}$. After some calculation, it is possible to apply lemma 7:

$$\mathbb{P}(T > k) \geq (1-x)^k P_{m-1,k} + kx(1-x)^{k-1} P_{m-1,k-1}$$

$$= \left(\frac{(\alpha+1)(m-1)}{m}\right)^k \prod_{i=0}^{k-1}\left(1 - \frac{i}{m-1}\right) + k\frac{1-\alpha(m-1)}{m}\left(\frac{(\alpha+1)(m-1)}{m}\right)^{k-1} \prod_{i=0}^{k-1}\left(1 - \frac{i}{m-1}\right)$$

$$= (1+\alpha)^{k-1}(1-(k-1)\alpha)P_{m,k} \geq P_{\beta,k}$$

This bounds the general case below with the uniform case, and we have the result. □

We now have everything we need to prove theorem 8.

Proof. (Proof of theorem 8) Let $X_1, X_2, ...$ be i.i.d. samples drawn from the probability distribution. From lemma 2 we know that if $\beta > 1000$ and $k \leq 2\sqrt{\beta \log \beta}$:

$$\mathbb{P}(T \geq k) \geq e^{-\frac{k^2}{2\beta} - \frac{k^3}{6\beta^2}}$$

Note that:

$$\mathbb{P}(T \geq k) = 1 - \mathbb{P}(\text{At least one collision in first } k)$$

$$= 1 - \gamma \geq e^{-\frac{k^2}{2\beta} - \frac{k^3}{6\beta^2}}$$

Inverting the expression gives:

$$\beta \leq \frac{2k}{-3 + \sqrt{9 + \frac{24}{k}\log\frac{1}{1-\gamma}}} = \beta^*$$

Suppose S is the smallest subset such that $\mathbb{P}(x \in S) \geq \rho$, and let $|S| = N$. By considering the distribution where the probability to be inside S to be perfectly uniform and perfectly concentrated outside, we can obtain a bound the biggest possible size of N:

$$\beta^* \geq \frac{1}{\sum_{x \in S}\mathbb{P}(x)^2 + \mathbb{P}(x \in \Omega \setminus S)^2} = \frac{1}{N\left(\frac{\rho}{N}\right)^2 + (1-\rho)^2}$$

$$\implies N \leq \frac{\rho^2}{(\frac{1}{\beta*} - (1-\rho)^2)} = \frac{2k\rho^2}{\left(-3 + \sqrt{9 + \frac{24}{k}\log\frac{1}{1-\gamma}}\right) - 2k(1-\rho)^2}$$

Which gives us the desired result. □

Observe that for values of ρ and γ close to 1, this N is actually bounded by a quantity which is $O(k^{\frac{3}{2}})$. The paper continued to use the coarser $N = O(k^2)$ bound, since it is still sufficient to detect discrepancies between support sizes.

The additional assumptions of $\beta > 1000$ and $k \leq 2\sqrt{\beta \log \beta}$ are easily met, due to the high dimensional nature of the data with which GANs are often trained.

Also, note that whilst these results are only proved for discrete distributions, but it is still possible to use the method on continuous distributions by using some metric (e.g. Euclidean distance when looking at images) to determine the similarity of different points which have been sampled from the distribution. Since all the test requires is the difference between the quantity obtained from the true distribution and learned distribution, having to use such a metric does not impact the method's effectiveness.

Using this method, Arora et. al were able to show in [5] that a lot of modern GANs which are capable of producing sharp images in fact fail at learning the desired support sizes. Furthermore, they were able to verify the Corollary 2, which stated that if the size support necessary to fool a discriminator is $\tilde{O}(p)$, by steadily increasing the diversity of the discriminator and observing the linear increase in support in the learned distribution.

6 Why Does the Generator Win?

We shall conclude the essay by investigating some results in game theory relating to GANs.

In our implementation of the GAN, the generator was able to win the game, and the discriminator slowly converged towards $\frac{1}{2}$. In fact, the fact that the generator wins is observed to happen quite often. This is rather interesting, since it implies that the p_{data} is very close to p_G in neural net distance once training is complete. It's note immediately clear why that this should be the case, given that we expect p_{data} to be quite complicated. This section will thereby be dedicated to a result by [16] which shows that for sufficiently complicated GANs, this will indeed happen more often than not.

To see why this is the case, first restate the definition of the value function:

$$V(u,v) = \mathbb{E}_{x\sim p_{\text{data}}}\phi(D_v(x)) + \mathbb{E}_{x\sim p_{G_u}}\phi(1-D_v(x))$$

This time, let's restrict D_v to a class of discriminators with parameters $v \in \mathcal{V}$ and G_u belong to a class of generators with parameters $u \in \mathcal{U}$. We know from section Section 3 that in practice, the distributions are empirical, but this does not affect the results on the equilibrium.

We also assume that the discriminator and generators are L'-Lipschitz with respect to its inputs and L-Lipschitz with respect to its parameters.

In order to investigate this further, it is necessary to introduce some results in game theory.

6.1 Useful Definitions in Game Theory

Definition. A pure strategy for a game completely determines a player's moves for every situation in the game. A mixed strategy is a set of pure strategies with equipped with a probability distribution. In the context of GANs, a mixed strategy for the generator equips a probability distribution S_u over the parameters, with a similar definition for the discriminator.

We require this definition in order to invoke the next theorem by Von Neumann[19]. Later on, we shall show that we can use a single neural net to represent the appropriate mixture of generators. Effectively what a mixture of generators and discriminators is doing is use S_v and S_u to chose a random discriminator and generator, then play the game.

Theorem 9. *(V. Neumann, 1928) There exists V^* and (S_u, S_v) be (possibly infinite) mixed strategies such that:*

$$\forall v, \mathbb{E}_{u\sim S_u}(V(u,v)) \leq V^*, \qquad \forall u, \mathbb{E}_{v\sim S_v}(V(u,v)) \geq V^*$$

What this theorem states is that the zero-sum game has some mixed solution. This is definition is a little unwieldy for our purposes, since the mixed strategies might be infinite. In order to ensure that our mixtures are not infinite we use a weaker definition by [Lipton and Young, 1994].

Definition. A pair of mixed strategies (S_u, S_v) is said to be in ϵ-approximate equilibrium if for some V^*:

$$\forall v, \mathbb{E}_{u\sim S_u}(V(u,v)) \leq V^* + \epsilon, \qquad \forall u, \mathbb{E}_{v\sim S_v}(V(u,v)) \geq V^* - \epsilon$$

If the strategies are point masses, then we say the equilibrium is pure.

6.2 Equilibrium for a Mixture of Generators and Discriminators

We first prove that an equilibrium exists for a mixed GAN.

Theorem 10. *There is a constant $C > 0$ such that for any $\epsilon > 0$ there exists a uniform mixture of $T = \frac{C\Delta^2 \log(\frac{pLL'L_\phi}{\epsilon})}{\epsilon^2}$ generators and mixture of T discriminators where the strategies achieve equilibrium. In other words, there exists $G_{u_1}, G_{u_2}, ..., G_{u_T}$, $D_{v_1}, ..., D_{v_T}$ and (S_u, S_v) is in ϵ-approximate equilibrium, where S_u is uniform on the parameters $u_1, ..., u_T$ and S_v is uniform on the parameters $v_1, ..., v_T$.*

Proof. Let (S'_u, S'_v) and V^* satisfy the result by Von Neumann.

Similar to the proof of the generalisation of neural net distances, we construct $\frac{\epsilon}{4LL'L_\phi}$ -nets $\chi_\mathcal{V}$ for $\mathcal{V}$ and $\chi_\mathcal{U}$ for U, the set of parameters for the discriminator and generator respectively. Again, $\log(|\chi_\mathcal{V}| + |\chi_\mathcal{U}|) \leq C'p\log(\frac{LL'L_\phi\phi}{\epsilon})$ for some C'. Using the Chernoff bound, for any $u' \in \chi_\mathcal{U}$:

$$\mathbb{P}(\mathbb{E}_{i=1,..,T}V(u', v_i) \leq \mathbb{E}_{v\in\mathcal{V}}V(u', v) - \frac{\epsilon}{2}) \leq e^{-\frac{\epsilon^2 T}{2\Delta^2}}$$

Where $u_1, ..., u_T$ is sampled randomly from $\mathcal{U}$. By choosing $T = \frac{C\Delta^2\log(\frac{pLL'L_\phi}{\epsilon})}{\epsilon^2}$ and $C \geq C'$, we ensure that this inequality holds with high probability. Further, by using a union bound we know that it holds for all $u' \in \chi_\mathcal{U}$ with high probability.

Again, we're use the net along with a triangle inequality. From the definition, we have $\forall u \in \mathcal{U}, \exists u' \in \chi_\mathcal{U}$ s.t. $||u - u'|| \leq \frac{\epsilon}{4LL'L_\phi}$. Via the Lipschitz property of the functions making up $V(u, v)$, it is clear that V has Lipschitz constant $2LL'L_\phi$ in u and v. Hence:

$$|\mathbb{E}_{i=1,..,T}V(u', v_i) - \mathbb{E}_{i=1,..,T}V(u, v_i)| \leq \frac{\epsilon}{2}$$

Combining the two results with the triangle inequality gives for $v \in \mathcal{V}$:

$$\mathbb{E}_{i=1,..,T}V(u', v_i) \geq V^* - \epsilon$$

The upper bound of $V^* + \epsilon$ can be proved using identical methods. Combining the two gives approximate equilibrium. □

This theorem effectively states that when there are two such distributions, the game is "winnable". Now we just need to show that this state is in fact when the discriminator is $\frac{1}{2}$, under some additional assumptions.

Theorem 11. *In the same setting of the previous theorem, if the generators are able to approximate Gaussian distributions and the discriminator constant functions, and ϕ is concave, then $V^* = 2\phi(\frac{1}{2})$.*

Proof. This is easy to show for the discriminator – setting $D_u(\boldsymbol{x}) = \frac{1}{2}$ automatically sets $V = 2\phi(\frac{1}{2})$. Since the generator is nested over by the discriminator in the value function, this will always be the case.

We have assumed that the generator can approximate Gaussians. Consider $G_{x,\xi}$, where $\xi > 0$ and x is sampled from p_{data}. Consider we add Gaussian noise of standard deviation ξ to the real distribution, resulting in a distribution $p_\xi(\boldsymbol{x})i$. This is a convolution of the real distribution and $\xi N(0, I)$.

Note that p_ξ may be expressed as a infinite mixture of Gaussian distributions (the intuition is similar to Gaussian kernel density estimation). Further the Wasserstein between a distribution before and after Gaussian noise $O(\xi)$, because the Wasserstein distance between a point mass at 0 and $\xi N(0, I)$ is $O(\xi)$, and by the earth mover interpretation, this upper bounds the Wasserstein distance for the convolution of other distributions and $\xi N(0, I)$. The discriminator is L'-Lipschitz with respect to its inputs, and ϕ is L-Lipschitz. Hence:

$$\begin{aligned}
&|\mathbb{E}_{x\sim p_{G_\xi}}\phi(1 - D_v(x)) - \mathbb{E}_{x\sim p_{\text{data}}}\phi(1 - D_v(x))| \leq L_\phi L' d_W(p_{\text{data}}, p_{G_\xi}) = O(L_\phi L'\xi) \\
&\implies \max_v \mathbb{E}_{x\sim p_{\text{data}}}(\phi(D_v(\boldsymbol{x}))) + \mathbb{E}_{x\sim p_{G_\xi}}(\phi(1 - D_v(\boldsymbol{x}))) \\
&\leq O(L_\phi L'\xi) + \max_v \mathbb{E}_{x\sim p_{\text{data}}}(\phi(D_v(\boldsymbol{x})) + \phi(1 - D_v(\boldsymbol{x})))
\end{aligned}$$

By concavity, $(\phi(D_v(\boldsymbol{x})) + \phi(1 - D_v(\boldsymbol{x}))) \leq 2\phi(\frac{1}{2})$. Hence, by letting $\xi \to 0$, $V \to 2\phi(\frac{1}{2})$.

□

Practically, the fact that generator is able to approximate a Gaussian is not a difficult assumption to hold. In fact, the prior random noise used by the generator is often Gaussian. The assumption for the discriminator is also trivially held.

The above result means that, with a sufficiently large support on the mixture of generators, there exists a scenario in the game where the mixture of generators wins. This scenario is also relatively easy to achieve, since we only need T generators.

6.3 Equilibrium with a single Neural Net

In the previous section, we showed that the generator can always win if it is a mixture of generators which can approximate a Gaussian. However, when implementing a GAN, we do not use a mixture of generators in practice – we use a single neural net. This section will show that a single neural net with the appropriate number of parameters is able to approximate such a mixture, allowing for pure equilibrium to be achieved.

Theorem 12. *Suppose the individual generators and discriminators making up the mixtures of generators and discriminators are neural networks with $k > 2$ layers and p parameters. For convenience, assume that the last layer of the neural net is a linear Rectifier. Then there exists $k+1$-layer neural networks which can represent generators G and D with $O(\frac{\Delta^2 p^2 \log(\frac{LL'L_\phi p}{\epsilon})}{\epsilon^2})$ parameters such that there exists a ϵ-approximate pure equilibrium with $V^* = 2\phi(\frac{1}{2})$.*

We just need the rectifier to approximate a step function for the purposes of the proof. It is easy for most activation function to approximate step functions, but we use the rectifier for the sake of simplicity.

In the statement of the proof, we have already assumed the existence of k-layer generators capable of representing Gaussian distributions. So it's simply a matter of creating a neural network which can represent a mixture using these existing distributions.

Lemma 3. *Let $q \in \mathbb{N}$ and $z_1 < z_2 < ... < z_q \in \mathbb{R}$ and $0 < \delta < \min z_{i+1} - z_i$. There exists a 2-layer ReLU neural network $\boldsymbol{S} : \mathbb{R} \to \mathbb{R}^{q+1}$ such that:*

1. *$\sum_{i=1}^{q+1} S_i(z) = 1$, where i are the components of $\boldsymbol{S}(z)$.*
2. *$z \in [z_{i-1} + \frac{\delta}{2}, z_i - \frac{\delta}{2}]$, $S_i(z) = 1$ and $S_j(z) = 0$ for $j \neq i$.*
3. *$z < z_1 - \frac{\delta}{2}$ $S_1(z) = 1$ and $S_j(z) = 0$ for $j \neq 0$.*
4. *$z > z_q + \frac{\delta}{2}$ $S_{q+1}(z) = 1$ and $S_j(z) = 0$ for $j \neq q+1$.*

Proof. Note that a step function at z_i can be approximated with two ReLUs:

$$f(z; z_i) = \max\left\{0, \frac{z - z_i - \frac{\delta}{2},}{\delta}\right\} - \max\left\{0, \frac{z - z_i + \frac{\delta}{2},}{\delta}\right\}$$

Hence we define $\boldsymbol{S}(z)$ such that:

$$S_1 = 1 - f(z; z_1), \qquad S_{q+1} = f(z; z_q), \qquad S_i(z) = f(z; z_i) - f(z; z_i - 1), \quad i = 2, .., q$$

□

This defines a step function with which we can use to select which Gaussian generator to activate. By choosing z_i, we can also ensure that each generator can be selected with equal probability.

Lemma 4. *Given the setting of the above theorem, we can create a $k+1$-layer neural network with $O(\frac{\Delta^2 p^2 \log(\frac{LL'L_\phi p}{\epsilon})}{\epsilon^2})$ parameters that can generate a distribution within δ' of the mixture of the k-layer generators $G_{u_1}, ..., G_{u_T}$ in terms of total variation.*

Proof. Let $p'_z : \boldsymbol{Z'} \to [0, 1]$ be the prior distribution for each of G_{u_1} and let p_z be some one dimensional prior distribution on $Z = \mathbb{R}$. Choose $z_1, ..., z_T$ such that it partitions the probabilities of p'_z evenly.

Define $G : Z \times Z' \to \mathbb{R}^d$ to be our generator, such that G first obtains the values of $\boldsymbol{S}(z)$ (as defined in Lemma 3) and the outputs of each of $G_{u_i}(z')$, where $z \in Z$ and $z' \in Z'$. Then at the rectifier of the

output of $G_{u_i}(z')$ we subtract $K(1 - S_i(z))$, where K is a very large constant, sufficient to overpower the outputs of the generator. This will give the ReLU a negative input and force this generator to contribute nothing to the output if $S_i(z) = 0$.

However the rectifier will return the usual vaue if $S_i(z) = 1$. This means we have created a neural net which is approximately the mixture of the Gaussian generators. The additional layer of summation means it is $k + 1$ layered.

Choose $\delta = \frac{\delta'}{100T}$ for our step function. Note that the probability that $\boldsymbol{S}$ as defined in Lemma 3 has non-zero entries in more than one point is less than δ'.

Recall that the total variation between two distributions μ and ν is:

$$d_{TV} = \sup_{\text{Event } A} |\mu(A) - \nu(A)|$$

This means that the difference between the two distributions is bounded by δ'.

We have $\tilde{O}(p^2)$ parameters in this neural net since we have used T generators of size p. □

This gives us a distribution which is arbitrarily close to the mixture model. Now we just need to verify that this small discrepancy of δ' does not break the result of Theorem 11.

Proof. (Proof of Theorem 12) Suppose T gives us a $\frac{\epsilon}{2}$-approximate mixed equilibrium. Construct G as in Lemma 4 such that $\delta' \leq \frac{\epsilon}{4\Delta}$. Let $V^\dagger(G, D)$ be the payoff for this game.

For any D_v, we have:

$$V^\dagger(G, D_v) \leq \mathbb{E}_{i=1,..,T} V(G_{u_i}, D_v) + |V(G, D_v) - \mathbb{E}_{i=1,..,T} V(G_{u_i}, D_v)|$$

Note that because ϕ is defined on a support of size 2Δ, the change of δ' in the total variation distance of the generator mean at most an alteration of $2\Delta\delta'$ in the value function. Hence:

$$V^\dagger(G, D_v) \leq V^* + \frac{\epsilon}{2} + 2\Delta\frac{\epsilon}{4\Delta} = V^* + \epsilon$$

Which proves the upper bound. For the other bound required to prove equilibrium, just choose $D = \frac{1}{2}$. This proves the result.

□

So we can conclude that there is a pure solution for the equilibrium, and a generator which consists of just a single neural net is indeed capable to winning again the discriminator, as long as the number of parameters is $\tilde{O}(p^2)$, as long as neural nets of size p can approximate Gaussians.

7 Appendix

Lemma 5. *If $m \geq 0$, then:*

$$\sum_{j=0}^{k-1} j^m \leq \frac{k^{m+1}}{m+1}$$

Lemma 6. *Consider the non-zero probability distributions on $\Omega = \{\omega_1, ..., \omega_n\}$. For fixed β, the $\mathbb{P}(T > k)$ is minimised when:*

$$\mathbb{P}(\omega_1) = \frac{\left(1 - \sqrt{(\frac{m}{\beta} - 1)(m-1)}\right)}{m}$$

$$\mathbb{P}(\omega_i) = \frac{\left(1 + \sqrt{\frac{\frac{m}{\beta} - 1}{m-1}}\right)}{m} \qquad i = 2, ..., m$$

and zero everywhere else, with $\lceil \beta \rceil = m$.

Lemma 7. *If $\beta_0 > 1$, $m = \lceil \beta \rceil$, and $\alpha = \sqrt{\frac{\frac{m}{\beta} - 1}{m-1}}$, then:*

$$(1+\alpha)^{k-1}(1 - (k-1)\alpha)P_{m,k} \geq P_{\beta,k}$$

Where $0 \leq k \leq m$

Proof. Note that $k < 2 \implies (1+\alpha)^{k-1}(1-(k-1)\alpha)$. When this is the case, $P_{m,k} \geq P_{\beta,k}$ holds because $m \geq \beta$. Now that we have shown a base case, we can try to prove the result by induction. Suppose that the result holds for $k = i - 1$, i.e.:

$$(1+\alpha)i - 2(1-(i-2)\alpha) \geq \frac{P_{\beta_0,i-1}}{P_{m,i-1}}$$

Then it is also true for $k = i$. We can do this by proving:

$$\frac{(1+\alpha)i - 1(1-(i-1)\alpha)}{(1+\alpha)i - 2(1-(i-2)\alpha)} \geq \frac{P_{\beta_0,i}}{P_{\beta_0,i-1}} \frac{P_{m,i-1}}{P_{m,i}} = \frac{1 - \frac{i-1}{\beta}}{1 - \frac{i-1}{m}}$$

Define $\delta = m - \beta_0$ (one minus the fractional part of β_0). This simplifies the above to:

$$\frac{(1+\alpha)(1-(i-1)\alpha)}{1-(i-2)\alpha} \geq 1 - \frac{(i-1)\delta}{\beta(m-(i-1))}$$

Note that $\delta < 1$ and $\beta > m - 1$ by definition, hence:

$$\alpha = \sqrt{\frac{\delta}{\beta(m-1)}} < \frac{1}{m-1}$$

Note that $2 < i \leq m$. So multiply bother sides by $-(i-2)(m-1)$ flips the sign and makes the inequality non-strict, returning:

$$\frac{1}{(m-1)(1-(i-2)\alpha)} \leq \frac{1}{m-(i-1)}$$

Multiplying both sides by $\frac{\delta}{\beta}(1-i)$ and then adding 1 to both sides gives the desired inequality above. □

8 Appendix

8.1 Code

The code use in Section 4 can be found in the following repository:

8.2 Acknowledgements

I would like to thank Dr Sergio Bacallado for advising me for this essay.

8.3 References

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. Available at http://www.deeplearningbook.org.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. NIPS 2014, arXiv:1406.2661.

[3] Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. arXiv:1701.00160.

[4] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and Equilibrium in Generative Adversarial Nets (GANs). ICML 2017, arXiv:1703.00573.

[5] Sanjeev Arora and Yi Zhang. Do GANs actually learn the distribution? An empirical study. arXiv:1706.08224.

[6] Martin Arjovsky, Soumith Chintala, and L´eon Bottou. Wasserstein GAN. arXiv:1701.07875v3

[7] Luke Metz, Ben Poole, David Pfau,Jascha Sohl-Dickstein. Unrolled Generative Adversarial Networks. arXiv:1611.02163

[8] Michael J. Wiener. Bounds on Birthday Attack Times. https://eprint.iacr.org/2005/318.pdf

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

[10] Chris Donahue, Julian McAuley, Miller Puckette. Synthesizing Audio with Generative Adversarial Networks. arXiv:1802.04208

[11] Soumith Chintala. How to Train a GAN? Tips and tricks to make GANs work. Github Repository, available here: `github.com/soumith/ganhacks`

[12] Tengyao Wang. Statistical Learning in Practice, Lent 2018. Available at: `www.statslab.cam.ac.uk/~tw389/teaching/SLP18`

[13] Rajen Shah. Teaching – Modern Statistical Methods. Available at: `www.statslab.cam.ac.uk/~rds37/modern_stat_methods.html`

[14] Sergio Bacallado. Bayesian Modelling and Computation, Lent 2018. Available at: `www.statslab.cam.ac.uk/~sb2116/bayesian_computation/`

[15] Sanjeev Arora. Generalization and Equilibrium in Generative Adversarial Networks (GANs). Available here: `www.offconvex.org/2017/03/30/GANs2/`

[16] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and Equilibrium in Generative Adversarial Nets (GANs) – Supporting Document. Proceedings of the 34th International Conference on Machine Learning, PMLR 70:224-232, 2017.

[17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. arXiv:1611.07004

[18] Rowel Atienza. (2017) GAN by Example using Keras on Tensorflow Backend. Available at: `towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0`

[19] J v. Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295?320, 1928.

[20] Diederik P. Kingma, Jimmy Lei Ba. Adam: A Method for Stochastic Optimisation. arXiv:1412.6980