University of Southampton
Faculty of Engineering and Physical Sciences
Electronics and Computer Science

Quantitative Analysis of the Evolutionary GAN on Synthetic Datasets

by

Edward Kitcher

September 2019

Supervisor: Dr. Richard Watson
Second Examiner: Prof. Harold M H Chong

A dissertation submitted in partial fulfilment of
the degree MSc Artificial Intelligence

Abstract

Generative Adversarial Networks (GANs) are a type of generative model capable of producing incredibly realistic data, without the generative network ever being exposed to the data is it trying to generate. New GAN frameworks and algorithms are being created all the time with the main focus on increase the performance and training stability. In this paper we investigate one of these new GAN frameworks, the Evolutionary- GAN (E-GAN). This framework creates and evolves a population of generators through different mutations, using a fitness function to decide which generators survive into the next generation. Authors claim that these evolutionary techniques increase generative quality and training stability. This is backed up by the use of synthetic data sets, however, no quantitative analysis was done on the generated samples from these data sets. We therefore propose further analysing the performance of E-GAN against other GANs on these synthetic data sets, using a slightly different network structure and our own metric to quantify performance. Experiments show that these claims do not to hold up. While E-GAN is able to produce comparatively good quality data, it is not noticeably different from some less computationally expensive GANs. Crucially, we find that a random version of the E-GAN, which does not utilise the proposed fitness function, is able to perform at a similar level. Although not conclusive, our work suggests the evolutionary framework is not the reason for the improved generative quality and stability.

Acknowledgements

I'd like to thank my friends and family for their support. As well as my lecturers and tutor for their guidance.

Also my examiners for taking the time to read my work.

Statement of Originality

I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.

I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.

I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

I have acknowledged all sources, and identified any content taken from elsewhere.

I have not used any resources produced by anyone else.

I did all the work myself, or with my allocated group, and have not helped anyone else.

The material in the report is genuine, and I have included all my data/code/designs.

I have not submitted any part of this work for another assessment.

My work did not involve human participants, their cells or animals.

Contents

Chapter 1

Introduction

We begin by taking time to explain what a GAN actually is, as it can be quite hard to explain and understand the process accurately. This section will include very limited mathematical terminology and will focus more on describing the overall process, as we find the mathematical notation initially creates a lot of confusion and unnecessary confusion. We will then go on to briefly describe the Evolutionary GAN (E-GAN) and outline what our research question is and how we will answer it.

1.1 What is a GAN?

A Generative Adversarial Network (GAN) is a model proposed by Goodfellow et al [4] which consists of two networks, a generator and a discriminator. The basic idea behind the model is that given a data set (the real data), we want to train the generator to produce data (fake data) which the discriminator finds indistinguishable from the real data. Once the generator has done this is has essentially captured the essence of that data (i.e the data distribution).

It's helpful to employ a metaphor to describe the training process of GANs. The most popular is that of counterfeiters (the generator) creating fake currency and the police (the discriminator) trying to detect the counterfeit currency. However, this description does not properly capture the subtle details of the process and implies that one network must beat the other, when in fact, we want both of them to grow and learn together until G is producing convincing enough data that D (and human observers) believe to be real. To fix this, we propose a different metaphor, that of a lazy student (generator) and a clumsy teacher (discriminator).

Imagine a scenario in which a student has to take a series of exams, but is completely unprepared and has no knowledge of the subject matter. The stu-

dents grand strategy is to simply guess the answers, in the hopes they can fool the teacher into thinking they know what they're talking about. After each exam the student will be able to see which questions they got right and which they got wrong, allowing them to improve their subject knowledge and their ability to fool the teacher. Luckily for the student, the teacher is just as useless. They have little or no knowledge of the subject matter, and so must mark the questions right/wrong based on their own judgement. The teacher is not only looking for correct answers, but is also looking out for answer which they believe are just guesses. After marking the paper they give a transcript back to the student, only then realising they had a mark scheme for the exam. Using this mark scheme they see how accurate they were and quickly try to learn a bit more about the subject matter to improve their ability for next time. This process of taking exams, marking and reflection is repeated until the student can write answers which the teacher believes are correct, and cannot accurately tell if the student is guessing or not.

Figure 1.1: Image taken from 'Progressive Growing of GANs for Improved Quality, Stability, and Variation' [7]. Images generated from the CELEBA-HQ data set. All images were created by the generator without it ever seeing the real data set.

As you can see from figure 1.1 GANs are able to produce amazing quality, even though the generator never gets to see the data set it is trying to learn. GANs are not only used for image creation but can be used on a wide variety of data types such as: video, text, sound etc. The student-teacher metaphor is useful to gain a rough idea of training but a more visual explanation can be seen in figure 1.2. It's hard to believe but the generator is able to produce such incredible output from only ever using random noise (z) as its input. Specifically, random noise generated from either a uniform or normal distribution. D takes either the real data, drawn from the given data set (X), or the fake data ($G(z)$) produced by G. D will then output a probability which represents its belief that the input data was real. The output from D is then used to update the networks

via a process called back propagation. This updates the weights of the network according to a given loss function. While G and D have different loss functions, these loss functions themselves differ depending on the GAN architecture being used, more on this in chapter 2.

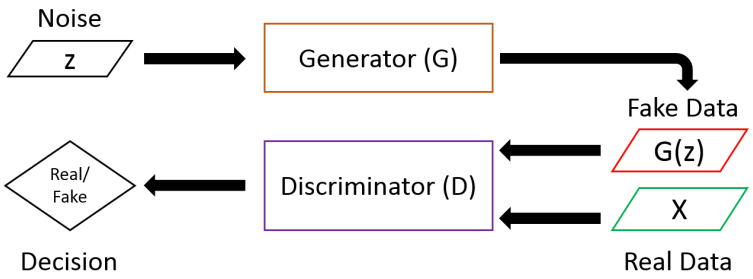

Figure 1.2: Illustration of the training process. Generator (G) recieves input noise (z) to create output G(z). This fake output or the real data (X) is then given to D, to decide if the data was real or fake.

If GANs are sounding too good to be true, unfortunately, that's because they are. The training of GANs is fraught with difficulty as the training process itself can be very unstable. More detail and explanation on this can be found in chapter 2, but For now we shall give an overview of the main problems. We mentioned in the student-teacher metaphor that initially the teacher has poor understanding of the subject (the data), which means that early on in the process they are likely to give the student misleading feedback. As a consequence, the student will likely start going down the wrong path and continue to output nonsensical data. To solve this one might think that having a knowledgeable teacher would be much more efficient (theoretically, this is equivalent to have an optimal discriminator). However, this also causes problems and it too makes training unstable. This is because the teacher will always be able to tell when the student is guessing and will always give 0 marks. Meaning that the student is unable to increase their understanding and therefore never improve. This inability to learn is referred to as **gradient loss** which we discuss further in chapter 2. Therefore, we require a balance between having a teacher good enough to give appropriate feedback, but not too good such that it leaves the student unable to improve. Much less likely is when G becomes much better than D, this too is undesirable as it has a similar impact. As D is unable to improve, G's output stagnates. Ideally we want the student to gain knowledge of the whole subject, but it's quite common for a student to focus on one or a few subsets (i.e modes) of the subject and ignore the rest. This is called **mode collapse**. If the student only learns a subset of the subject they will use this knowledge to answer all questions regardless of whether or not it makes sense. Meaning that after a while, the teacher will catch on that the student is just guessing and mark all answers (even when they are correct)

as wrong. This also leads to gradient loss because the student is receiving zero marks and so has nothing to work with.

If the training is so unstable, how do they produce such convincing data as in Figure 1.1? Research has been mainly focused on stabilising training, and in doing so, allows the process to create much greater output. In chapter 2 we describe the progress of GANs as they improve the stability and quality of the generated data, as well as the problems unrelated to training, such as a lack of reliable metrics. In this paper we shall be focusing on one of these proposed improvements, that of the Evolutionary GAN (E-GAN) [15]. Authors Wang et al. propose an algorithm with a population of generators, which compete against each other, produce offspring and after each training step the fittest are selected to continue on into the next generation. The differences between the generator offspring is how they update their network (i.e update their understanding). A synthetic data set is used to show the improved quality of E-GAN compared to other GANs. In our work we shall see if this evolutionary approach really does improve training stability compared to these other GANs by measuring its performance on these synthetic data sets using a simple metric of our own creation. Analysis on the synthetic data was previously only done qualitatively. We also use a slightly different network structure to see if E-GAN is a good general framework, or if its parameters have just been optimised for that problem.

Chapter 2

Background

As the theoretical analysis and proof of the GAN models is out of scope for this project we do not detail them here, but more detail can be found in their respective papers.

2.1 Development of GANs

The original GAN proposed by Ian Goodfellow et al. [4] laid the groundwork upon which many others have build a wide range of new algorithms and frameworks. The original GAN works by having the two networks play each other in an *adversarial* minimax game, in which D tries to maximise its probability of correctly guessing the class of the input, while G wants to minimise D's ability to correctly class its output as fake. As mentioned in the introduction, the game ends when G has matched the real data distribution meaning that D is unable to distinguish real from fake. This is achieved by minimising the distance between the generated data distribution and the real data distribution. Specifically, when D is optimal, this equivalent to the minimising the Jensen Shannon Divergence (JSD). The loss functions use binary cross entropy (BCE) as seen in equation 2.1. However there are issues with this as BCE only declares the data as right or wrong, and not by how much. This results in very little gradient information for the generator to follow and ultimately causes vanishing gradients.

$$L_D = -\mathbf{E}_x[\log(D(x))] - \mathbf{E}_z[\log(1 - D(G(z)))] \tag{2.1}$$

Since the original GAN there has been an rapid increase in new algorithms based on this foundation. These new algorithms sought to deal with the many issues of training GANs. The main problem was that the original GAN does not perform well on more complicated data (e.g. more complicated images such as higher resolution faces). To overcome this, convolutional layers were introduced

to the network structures of G and D. This was first done with the Laplacian Pyramid of Adversarial Networks [3] and soon after with the Deep Convolution GAN (DCGAN) [13]. The DCGAN won out in popularity as it continues the same architectural theme as the original GAN and introduces some important differences such as batch normalisation, which helps overcome some of the training instability.

The next notable contribution was that of the Wasserstein GAN [1]. It's main contributions were to introduce a new way of measuring the distance between the two probability distributions, i.e the Wasserstein distance. To achieve this distance, the authors propose clipping the weights of the discriminator. This was later found to actually damage the performance of the discriminator [5]. They show that better results can be achieved if instaed of clipping the weights, one should penalise the norm of the discriminator.

Since then another main contributor was the least-squares GAN (LSGAN) [10]. LS GAN, like the name suggests, uses a least squared loss function for both the generator and discriminator. This was motivated by the fact that traditional loss function of the discriminator only informs the generator of whether or not the generated sample was convincingly real or fake, but not by how much. This is not the case for LSGAN and as a result it helps move fake data closer to the real data distribution and therefore creates more stable gradients for the generator.

2.2 Problems with training GANs

As mentioned in the introduction, the training of GANs is riddled with problems. These problems will be discussed in the following section. The main problems we shall address in this section are mode collapse, lack of reliable metrics and performance dependency on hyperparameters and network architectures.

Mode collapse

This is when the generator fails to produce diverse output and only focuses on a small subset of the data distribution (modes). This can manifest itself in a few ways. Consider a situation in which we are trying to produce images of cats. While the test set contains a wide verity of cat pictures which are fed to D, during training the generator might start to focus on outputting a certain type of cat (e.g. a ginger cat) or even worse, only outputting the same image of a cat. While this is technically correct as it has managed to capture the data, it is only a subset of the distribution (a single mode). This will eventually lead D to class

any ginger cat it receives as fake, even if it isn't. This can happen multiple times in the same training session, with G focusing on a single mode until D learns to discard it. Of course this causes training to become very unstable and usually results in very poor output. Almost all GANs suffer from mode collapse in one form or another but many provide different ways of tackling the problem, mainly through either new objective functions or model architectures. The synthetic data sets seen later on originate from the Unrolled GAN [11] which set out to show how they could avoid mode collapse by training G to produce data which is able to fool not just the current discriminator but the updated discriminator a few training steps in the future. This was found to make the generator produce more diverse output and therefore reduce the mode collapse problem. The E-GAN [15] later proposes to overcome mode collapse through the use of a fitness function which places emphasis on the generator producing diverse output.

Metrics

The major issue currently with GANs is that there is no evaluation metric which is widely recognised as the best way to evaluate performance. Until recently, the most commonly used metric was inception score (IS), proposed by Salimans et al. [14]. IS was found to correlate well with ratings given by human observers [14] and aims to measure the quality and diversity of generated images. However, IS score only works with images. To calculate IS a set of generated images are fed to a classifier network called Inception Net, which has been pre-trained on Image Net (a huge data set of real-world images). Each generated image is given a label distribution by the classifier. If this image is easily identifiable (good quality) then it will have a narrow distribution (i.e a high probability of the image belonging to a single label). Each images' label distribution is then compared against the combined label distribution for the whole set of generated images (the diversity). The Kullback–Leibler (KL) divergence is then computed and exponentiated, giving us the inception score.

IS has recently come under criticism [2, 9] as it has a number of inaccuracies. The fundamental issue with IS is that it is not an accurate metric when using it to evaluate results from models trained on anything other than Image Net. This is due to the fact that the labels for some of the popular databases (such as CIFAR 10) are not easily identifiable and cause the classifier to give incorrect label distributions. A metric which is gaining popularity is Fréchet Inception Distance (FID), proposed by Heusel et al. [6] as it is less sensative to noise.

Hyper-parameters and neural architectures

GAN performance is heavily hyper-parameter dependant. Ignoring the issues of comparing GANs with a suitable metric, it is also hard to compare performance as it is unclear if a GAN framework is working well because of the new features proposed, or just because the hyper parameters have been optimised for that framework. A large study conducted by M. Lucic et al [9] found that out of the GAN models they tested, none of them were noticeably more stable than the others when each GANs' hyper-parameters were optimised for the given data set. Meaning that each data set requires its own optimised hyper parameters in order for the model to perform at its best. Previously, there has been attempts to allow for comparisons between different models by using the same network set ups as DCGAN [13] like with the E-GAN [15] but this may not provide a good comparison [9].

2.3 Evolutionary GAN

The evolutionary GAN (EGAN) was proposed by Wang et al. [15] with the aim of stabilising GAN training and improving generator performance. The framework takes inspiration from evolution by utilising different properties of the evolutionary process, namely; variation, evaluation and selection.

The proposed framework uses a population of generators which evolve over time, with their fitness being determined by the discriminator. In other words, they are thinking about the generator as a species and the discriminator as the environment. One could possibly invoke the concept of Niche Construction [12] here as both G and D are changing over time as a result of the other. Although, perhaps it is more closely related to the idea of an evolutionary arms race (admittedly, D is not being selected for).

The algorithm works by using each training step as a generation. At the beginning of each generation we have a population of generators (i.e the parents) which create offspring via asexual reproduction. These offspring are perfect clones of their parents and so inherit all the same genes (network parameters). To create variation, each child is mutated. This is achieved by assigning each child their own unique loss function (discussed further below). Once this loss has been calculated, based on the current discriminator, each child generator will be updated via back propagation. As a result, the children will have slightly different network parameters (genes) compared to their parents and siblings.

After each child has had their loss calculated and networks updated, they are

evaluated based on a given fitness function as seen in equation 2.4. This fitness function is comprised of two parts, quality (F_q) and diversity (F_d). The n most fit children are selected (elitist selection) to become the new parents in the next generation, and the others are discarded. This process of variation, evaluation and selection is repeated for a given number of generations.

$$F_q = \mathbf{E}_z[D(G(z))] \tag{2.2}$$

$$F_d = -\log \|\nabla_D - \mathbf{E}_x[\log D(x)] - \mathbf{E}_z[\log(1 - D(G(z)))]\| \tag{2.3}$$

$$F = F_q + \gamma F_d \tag{2.4}$$

While the algorithm does allow for a population of any size, in all of the authors experiments only one parent generator is used at each generation. This is because training GANs is already computationally expensive and having a population would require far too much computational power. Due to there only being one generator at the beginning of each generation, the algorithm is essentially an 'evolutionary hill climber', meaning that at each generation the algorithm climbs a little closer towards the solution at the top of the current hill in solution space.

As the algorithm uses multiple loss functions for the generator which are all calculated against the same discriminator, there has not been any theoretical analysis of why this approach should work, unlike many others [4, 1, 10, 11]. As such, this is more of an engineered solution rather than a sound theoretically one. The E-GAN is still able to produce good output, their results on a synthetic data set can be seen in Figure 2.1.

The mutations in the E-GAN are what defines each child. Meaning that at each iteration 3 children are created and evaluated on the fitness function. The first mutation is the Minimax loss function, as seen in equation 2.5. This is loss function proposed in the original GAN paper [4]. The second mutation is the Heuristic, referred to later on as 'LogD'. This can be seen in equation 2.6. The last mutation is least-sqaures. This was proposed in the Least-sqaures GAN (LSGAN) [10] but the difference being that in this set up, the discriminator loss is using binary cross entropy, not mean squared error. It is possible to incorporate more loss functions although the authors claim to have investigated others without giving specifics.

$$M_{minimax} = \mathbf{E}_z[\log(1 - D(G(z))] \tag{2.5}$$

$$M_{heuristic} = -\mathbf{E}_z[\log(D(G(z))] \tag{2.6}$$

$$M_{least-squares} = \mathbf{E}_z[(D(G(z)) - 1)^2] \tag{2.7}$$

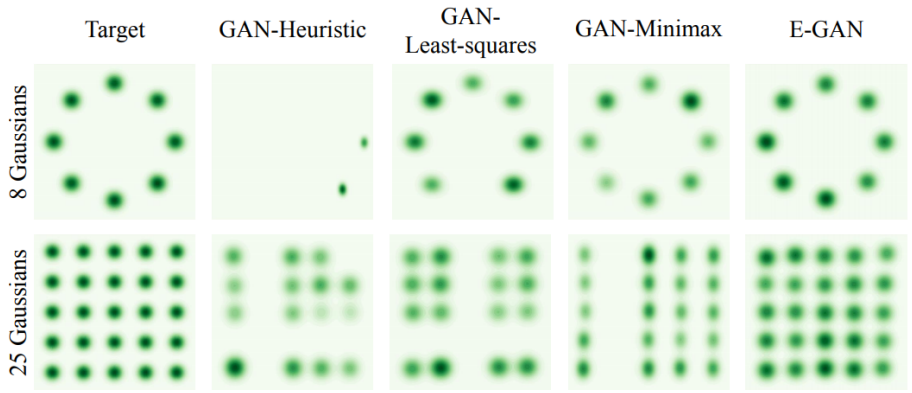

Figure 2.1: Image taken from the original E-GAN paper [15]. The images show the Kernel Density Estimate (KDE) plots for different GAN models on two different data sets: 8 Gaussians and 25 Gaussians.

In the E-GAN paper, the authors illustrate the effectiveness a fitness function and multiple loss functions by running it on two simple synthetic data sets (see Figure 2.1), originally proposed in [11]. As there are multiple modes, we want the generator to be able to capture all of them without suffering from mode collapse. We can clearly see that the 'GAN-Heuristic' greatly suffers from mode collapse, where as the others still do but to a lesser degree, with E-GAN performing the best and with seemingly even distribution across all modes. The authors use this as an insight into why their algorithm performs well on more complicated data. However, these results on the synthetic data were not analysed quantitatively, only their experiments on image data sets were, with the use of IS. While it appears E-GAN out performs the other loss function groups on the synthetic data sets, it can not be determined without a large enough sample.

Chapter 3

Method

Our research is focused on evaluating the performance of E-GAN on synthetic data sets using quantitative analysis as current the analysis is only qualitative. However, as this was not our initial research direction we will also use this section to outline the reasoning behind the changes in direction and how we came to the current research question. We appreciate that some of this might be better suited to the Time Management chapter, however, we believe it is better to describe the process here as its easier to understood in context.

3.1 Assessing Viability

We had originally hoped to improve the E-GAN [15] algorithm by adding more complexity to the current fitness function. This would be achieved by employing a process called 'simulated annealing' [8], and possibly adding in additional components. Simulated annealing [8] is an optimisation process used to find the optimum of a function, in this case, the fitness function. To see how viable this was we set out to see what effect each component (F_d , F_q) currently has on the fitness function. We experimented with using a fitness function entirely comprised of either F_d or F_q on the two synthetic data sets (8 and 25 Gaussians). Strangely, the output from the generators was not noticeably different. We would expect to see that a fitness function only comprised of F_q would be susceptible to mode collapse as there would be a lack of diversity, and similarly F_d would be diverse but less focused on hitting the modes (a Gaussian). This was not the case.

To better understand the effect of the components, we recorded the fitness scores for each mutation at each iteration. We found that for the 8 Gaussian data set, 70% of the time the fittest child had both the highest F_q and F_d score, meaning that even if we were to use simulated annealing, it would only change the outcome of who is the fittest individual (at most) 30% of the time. For the 25 Gaussian data set, the chance for the fittest child to have both highest F_d and F_q

was roughly 55% of the time. This correlation was not dependant on time either, further making the use of simulated annealing unappealing. We tested to see how mutations were selected over time, with only a handful of runs (not enough for a statistical test), and it seemed like there was no correlation between number of steps and mutation selection. For instance, one mutation might be selected early on and then picked less later on in training, while in another run, it would be dominant throughout training. This further demonstrated the instability of the training process, and meant that any change we made to the algorithm would not be easily determined unless we used many runs to see what significance (if any) our changes have made to the data.

Throughout these basic experiments we became increasingly aware of how dependent the output is on the network structures. Dispute having almost identical networks, we were unable to match the quality of their output without increasing the variance in the weights. We discuss why we were unable to match their network structure more in the Project management chapter. We decided to keep the network structure as it was and use this as an opportunity to investigate the claims of the E-GAN paper.

3.2 Change of direction

The E-GAN claims to stabilise training and consequently improve the generators performance. If E-GAN really does stabilise training through the use of evolutionary techniques we would expect to see E-GAN out perform other GAN frameworks even with different network structure. This is where our research will focus. This kind of analysis is only feasible on synthetic data sets as it is much quicker to train.

The authors of the E-GAN use quantitative analysis (inception score) on more complicated data sets such as CIFAR-10, but only qualitative on the synthetic data sets. As discussed in section 2, inception score has since been shown to be unreliable as a metric ,on data sets other than Image-Net, and while it is easy to see that E-GAN performs well on the synthetic data, these do not necessarily illustrate that the evolutionary framework itself is responsible for the output quality. On top of this, due to the instability of training and the fact that performance is heavily dependent on hyper parameters, it is not sufficient to only use qualitative analysis. Therefore, we propose evaluating performance on these synthetic data sets.

As a metric doesn't exist we have created a simple evaluation metric of our

own, which can be used on known distributions to give a rough estimate of performance. It works by counting the amount of generated data points close to the mean of each of the Gaussians within a certain radius. If a Gaussian (mode) has more than its fair share of data points (mode collapse) its score begins to decrease. Overall score ranges from 0 to 1. Where 0 means no generated data was within any of the radii, and 1 means all data points are evenly distributed across all modes, which is incredibly unlikely. For instance, when we tested the score on a random sample from the actual data distribution, we found scores of roughly 0.95, the score wasn't perfect due to having uneven distribution. In summary, this metric scores high for data which is focused on the modes, while still being spread out (quality and diversity). This metric cannot be used on data which has an unknown distribution, but our aim is not to create a new metric to evaluate GANs, it is simply a means to an end in order to quantify performance for these synthetic data sets. Also, this score can obviously not be used in the fitness function has it requires knowledge of the data G is trying to capture.

3.3 Research Method

To test its performance of the E-GAN algorithm we will compare its performance against the following groups of mutations:
1) E-GAN: LogD, Minimax, Least Square
2) RND: Randomly selecting a mutation from 1) at each training step
3) LogD only
4) Minimax only
5) Least Square only
Groups 3,4 and 5 are the same as we saw in Figure 2.1. We are introducing the random group because we are interested in seeing if random mutations are any better than selected mutation
.

Each of these groups were ran 50 times on each data set, and after each run they output log files detailing the metric score, the fitness at each iteration, mutation selected at each iteration (where appropriate) and the final data plot and KDE plot. For data set we will create a box-plot based on their metric scores, along with their KDE output and mutation selected (where appropriate).

For each of the single mutation groups (3,4 and 5) we use the same discriminator loss function (i.e binary cross entropy) as used in the E-GAN framework [15]. It is not specifically stated but we assume this how the synthetic data sets were conducted originally. This means that we are testing the affect of each mu-

tation on the same type of discriminator.

Chapter 4

Results & Analysis

4.1 Experiments

In this section we wish to investigate the E-GAN framework to see if it can produce more stable training and better output than other set ups through quantitative analysis on the same data sets. We will present our results and follow up with analysis of what these represent, rather than having results and analysis in two separate chapters. We end the analysis with a discussion of the limitations of our research.

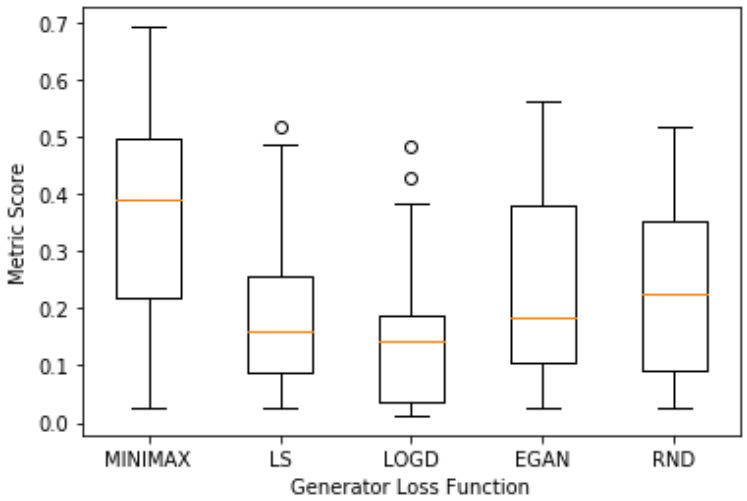

Figure 4.1: Box plot for the various mutation groups, produced from 50 samples each trained on the **8** Gaussian data set. Circles denote outliers.

There is a noticeable difference in score between the two data sets. The poor performance on the 25 Gaussian is to be expected though as its harder for the generator to 1) find all the modes, and 2) evenly distribute data across them. Surprisingly, E-GAN did not perform very well. We can clearly see from figure 4.1 and 4.2 that E-GANs performance is comparatively underwhelming, consid-

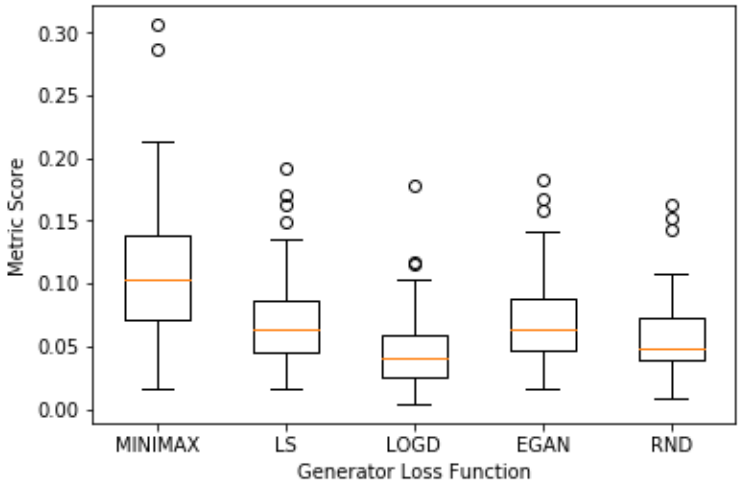

Figure 4.2: Box plot for the various mutation groups, produced from 50 samples each trained on the **25** Gaussian data set. Circles denote outliers.

ering its performance in the original E-GAN paper [15] (see figure 2.1). We would expect E-GAN to perform the best as it is able to choose which mutation to select at each iteration based on the fitness function. However this does not appear to be the case for either data set. Most noticeable is how E-GANs performance is dominated by minimax. Minimax is able to consistently produce better output, as can be seen by comparing their medians. This could be because the minimax loss function has been proven [4] (assuming an optimal discriminator) to produce effective gradients for the generator to follow, where as the E-GAN is an engineered solution. However, E-GAN does have a smaller range than minimax, across both data sets which suggests its output is more consistent and therefore more stable. This makes sense as minimax is known [4] to suffer from the vanishing gradient problem when D is able to spot fake data at a high rate. Furthermore, the fitness function could be making the training more stable as it has a verity of updates to choose from. Although, if we look at the output from both Minimax and E-GAN in Figures 4.3 & 4.4 we can see that on 8 Gaussians minimax does indeed seem to outperform E-GAN and that its top performance here is comparable to the original KDE plots in Figure 2.1. However, on 25 Gaussians this does not seem to be the case. As the evaluation metric punishes modes with more than their fair share of data, and as the E-GAN has quite dense concentration, this could be the reason for the low score. Which would suggest that E-GAN is actually performing better than the metric is reporting.

E-GANs performance is even more disappointing when we consider the fact that the least squares group is not using the same discriminator loss function as given in the original LSGAN paper [10], meaning that the pairing of these two

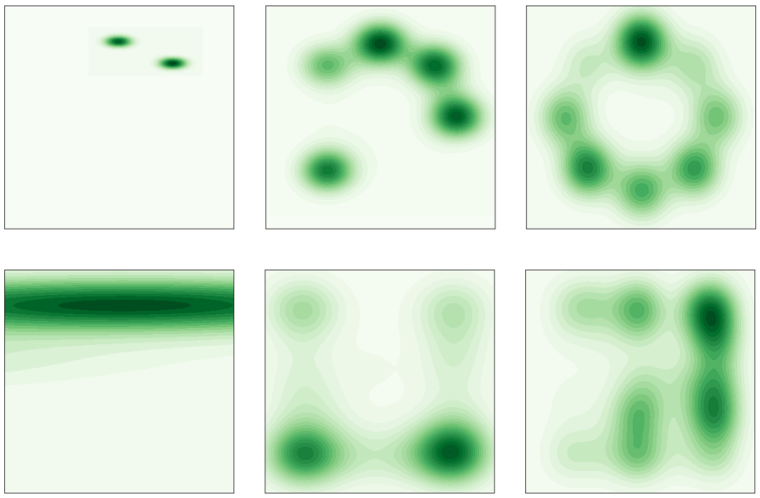

Figure 4.3: Minimax output for 8 Gaussian (top row) and 25 Gaussian (bottom row), with the minimum, median and maximum KDE plots respectively.

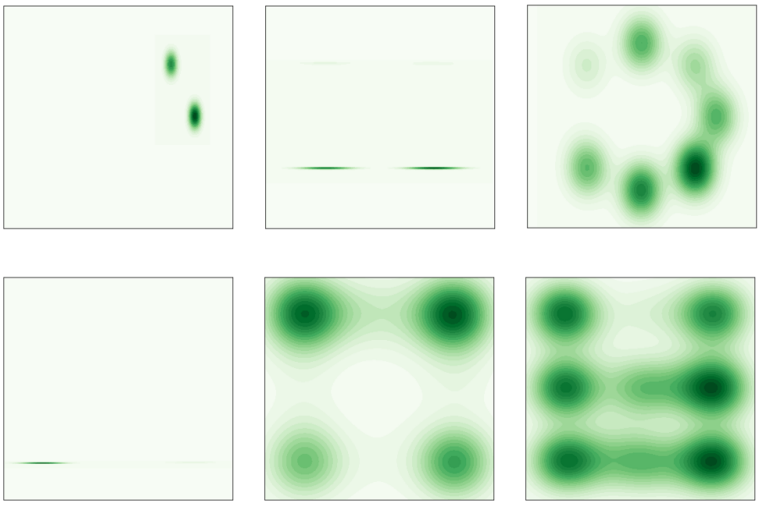

Figure 4.4: E-GAN output for 8 Gaussian (top row) and 25 Gaussian (bottom row), with the minimum, median and maximum KDE plots shown respectively

has no theoretical basis as to why they should work and yet its performance is comparable to E-GAN. However, the metric doesn't appear to fairly represent the actual difference in performance between the two. If we look at Figures 4.6 & 4.4 for the 25 Gaussian, there is a clear difference in performance. While the box plot for 25 Gaussian seems to show they are almost equal. The KDE plots could be misleading as the max result by E-GAN shown in the KDE might not be usual. While E-GAN does have noticably different interquartile range for the 8 Gaussian, this range seems similar for the 25 Gaussian which is likely due to there being more modes and so both end up collapsing on the roughly the same number of nodes each time.

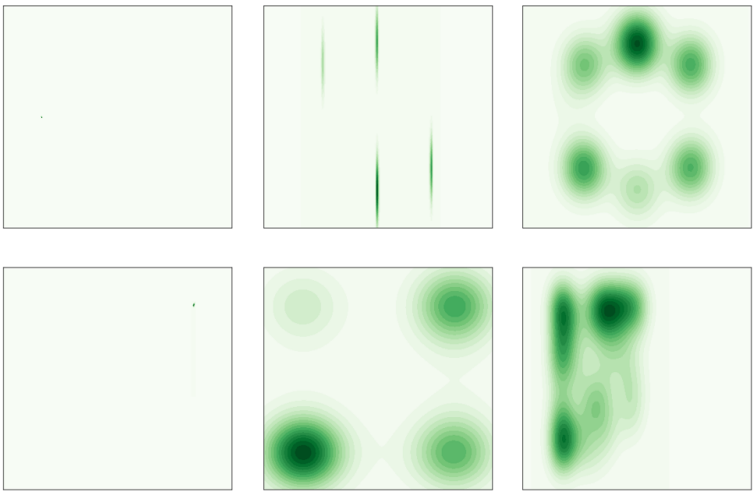

Figure 4.5: logD output for 8 Gaussian (top row) and 25 Gaussian (bottom row), with the minimum, median and maximum KDE plots shown respectively

The poor performance of the LogD group was expected but not to this degree. While the logD group should avoid the vanishing gradient problem [4], it can suffer from generative quality fluctuations [5]. It also must be quite susceptible to mode collapse as the consistently low scoring is likely due to the metric punishing it for focusing entirely on a subset of modes.

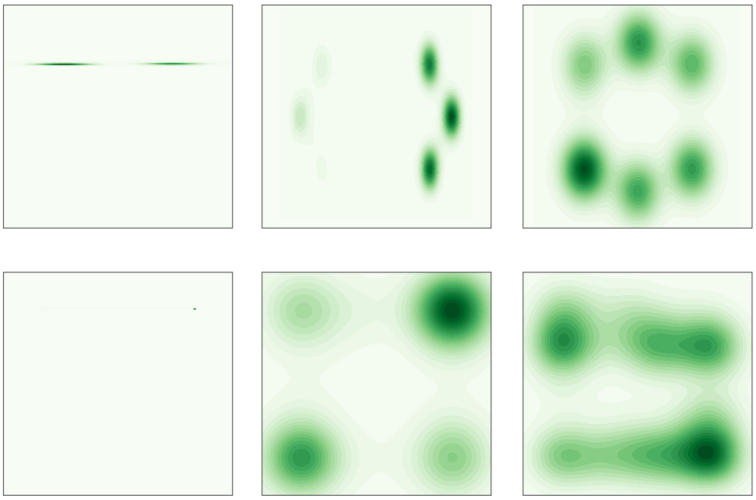

Figure 4.6: Least-Sqaures output for 8 Gaussian (top row) and 25 Gaussian (bottom row), with the minimum, median and maximum KDE plots shown respectively

The order of performance quality of Minimax, LS and LogD is similar to the ordering as seen in the original E-GAN paper see figure 2.1. This suggests that minimax is more robust than the other 2 as the performance is noticeably better. This could be because these data sets are quite simple, and it has been found

that minimax does struggle to work well on more complicated data [4].

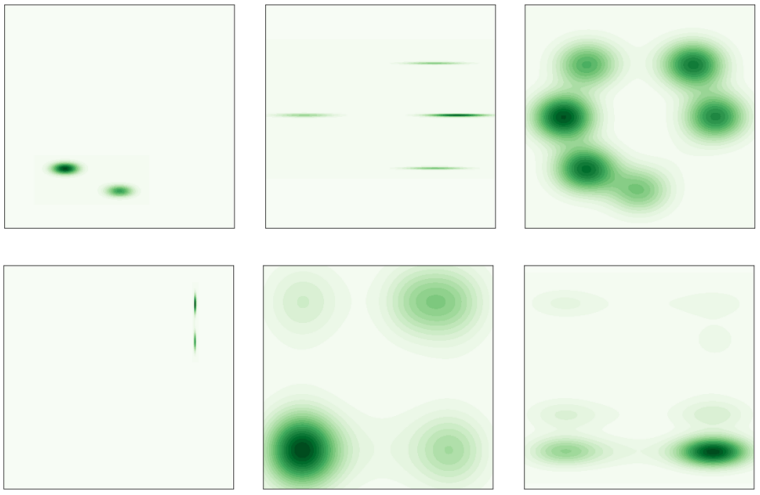

Figure 4.7: Random mutation output for 8 Gaussian (top row) and 25 Gaussian (bottom row), with the minimum, median and maximum KDE plots shown respectively

Interestingly, the random group (RND) was not that much worse than the E-GAN and it appears that it was able to actually outperform E-GAN on the 8 Gaussian data set if we compare their medians. This would suggest that the fitness function is not actually that effective. If picking a random mutation produces similar results, if not better, results this could be due to there being little difference between each loss function at each iteration. We did observe during training of the E-GAN that the fitnesses of each child at each iteration do not vary that much. This would suggest that the fitness function is unable to perform effectively as there is little difference between the children. If this is the case, then more diversity is needed in order for a fitness function to perform efficiently. Alternatively, the lack of E-GAN performance could just be a case of parameter optimisation, as the E-GAN [15] does have the fitness hyper-parameter of γ which was found to be optimal at $\gamma = 1$(for their network structure), and so if this parameter was to be optimised there would then be appropriate variety in fitness. However, this probably doesn't explain the whole story. If we look at their respective outputs from Figures 4.4 & 4.7 we can see that E-GAN was able put more emphasis on the modes it discovered, while RND had less focus. This may suggest that for more complicated data sets the fitness function *does* make a difference.

To further examine the role of the fitness function in performance we will now refer to Figures 4.8 & 4.9 which show how the mutations were selected over time. Interestingly, there seems to be a pattern of selection, which in our initial

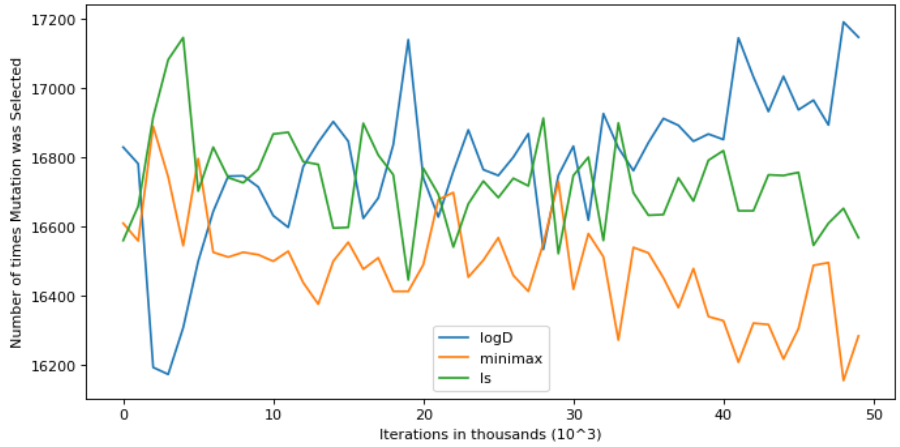

Figure 4.8: Selected mutations averaged over 1000 iterations for the E-GAN algorithm on the 8 Gaussian data set.

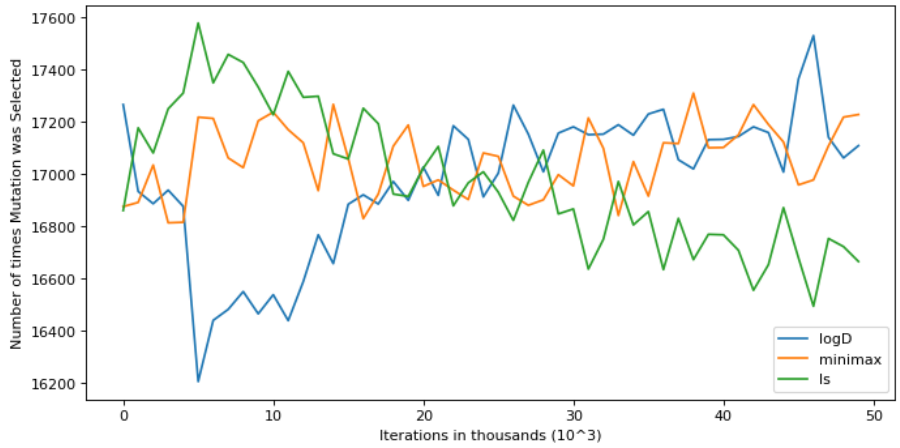

Figure 4.9: Selected mutations averaged over 1000 iterations for the E-GAN algorithm on the 25 Gaussian data set.

analysis seemed random. We can see that logD was initially selected for on both data sets but becomes less popular at different rates. A similar result was found in the E-GAN paper [15] on the CIFAR-10 data set, with the reason being that minimax is poor early on and is selected against. Of-course, these are selections from two different data sets, on two different network setups so they are not exactly comparable, but it is interesting to note the pattern of early selecting LogD. As D has an advantage early on in training due to the fact that it can understand the real data faster, this can cause G to struggle to find effective gradients. In doing so, G may be applying useless updates which is putting it on the wrong path, leading to disappointing output. This might explain why E-GANs and LogDs performance was quite poor. The LogD updates are giving the generator useless updates early on in training, but as E-GAN has access to more mutations it is able to recover from this, unlike LogD.

	Minimum	LQ	Median	UQ	Maximum	Range	No. Of Outliers
Minimax	0.025	0.2173	0.391	0.4981	0.691	0.666	0
LS	0.025	0.0869	0.1589	0.2545	0.4844	0.4594	1
LogD	0.0125	0.0367	0.1413	0.1889	0.3834	0.3709	2
E-GAN	0.025	0.1035	0.1832	0.3807	0.5624	0.5374	0
RND	0.025	0.091	0.2256	0.3522	0.5156	0.4906	0

Table 4.1: Box plot data results for the 8 Gaussians data set, taken at 4 d.p.

	Minimum	LQ	Median	UQ	Maximum	Range	No. Of Outliers
Minimax	0.0157	0.0716	0.1037	0.1386	0.2130	0.1972	2
LS	0.0158	0.0444	0.0628	0.0858	0.1349	0.1191	4
LogD	0.0040	0.0246	0.0403	0.0590	0.1037	0.0997	3
E-GAN	0.0158	0.0471	0.0599	0.0906	0.1408	0.125	3
RND	0.0080	0.0392	0.0481	0.0721	0.1076	0.0996	3

Table 4.2: Box plot data results for the 25 Gaussians data set, taken at 4 d.p.

Although there is a pattern to the data this does not explain how random mutations were able to perform similarly to E-GAN. One answer could be that the update steps are so small that we require the mutation to be selected many times in a row for it to have a meaningful effect. The selection of minimax is also unclear, as it was usually selected against in 8 Gaussian but remained fairly consistent in the 25 Gaussian. This could be that minimax is just better on more complicated data but this goes against the findings of the original GAN [4] which found its performance to be lacking on more complicated data.

As there are so many factors when considering performance, it is hard to pin point the exact cause of certain behaviours. A very useful analysis of many GANs on many data sets was conducted by Lucic, M et. al. [9], they found that most GANs could perform fairly equally given enough computational power and hyper-parameter optimisation. This could go some way to explaining the results found here as it could be the case that certain loss functions are just better suited to the network parameters on this problem.

It's also worth noting that the E-GAN was the most computationally expensive out of the groups as 3 losses were calculated at each training step, this doesn't seem to be a good trade off in terms of performance when we consider the quality of minimax.

4.2 Limitations

While it is tempting to draw concise conclusions about the results found here it is important to remember that the analysis was conduced on 1) synthetic data sets and not a sophisticated image data set (such as CIFAR10), and 2) the metric used was designed to give a good representation of the performance quality but obviously has its faults. After seeing the quality of images produced we are confident that the metric does correlate with performance, however it does seem to punish mode collapse far too much. For instance images with a wide spread of data across multiple modes but with little concentration score higher than data which focuses on a handful of modes with high concentration. In reality this is not ideal, as we want the generator to be confident in its output in order to create good quality data.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

The claim that E-GAN improves quality and stabilises training does not seem to hold up. By analysing E-GANs performance against other GANs with varying loss functions we find that E-GAN does not reliably produce better output than existing GANs such as the minimax formulation. This is likely due to the fact that it has a stronger theoretical founding. Although, the LogD also has theoretical foundation and it performed the worst. It is likely the case that if we optimised each groups' hyper-parameters we would find similar performance across all groups, similar to the findings of [9]. This further suggests that the performance found in the E-GAN paper [15] is more likely due to the hyper-parameters being generally optimised, than the actual evolutionary framework.

Crucially, we found that random mutations were able to perform at a similar level. This is important as it suggests that the fitness function isn't performing properly. This could be because it doesn't accurately measure quality and diversity in a useful way. Or more likely, that there isn't enough variation of fitness produced by the mutations. Meaning that each loss function needs to be selected multiple times in a row (compound effect) for it to show any meaningful impact. As mentioned before, the E-GAN with a population of 1 is essentially an evolutionary hill climber, and if we use random mutations it becomes a random hill climber. If these two are producing similar results it would suggest that the steps the algorithm is making are too small and/or don't actual point towards a optimal solution. Further supporting the idea that the fitness function doesn't work. This could be improved by allowing the algorithm to take bigger steps across the fitness landscape through selecting a mutation multiple times in a row and/or adding in more components to point the fitness score in the right direction.

We had originally thought that there was no correlation between training

step and mutation select (discussed in methodology). This is clearly not the case. However, the exact reason behind the selection is not clear. But what we can say is that the poor performance of E-GAN and LogD could be being caused by the LogD mutation. Suggesting that without it, E-GAN might be able to perform better as it will have more useful gradients to follow.

5.1.1 Future work

In-line with the work above, it would follow that testing would need to be done on the image data sets. However, due to the large amount of computational time required, gaining a large enough sample would be infeasible. We therefore propose the use of FID, rather than IS, as it has been found to be a more accurate measure of performance [9]. We are most interested in how the random mutation group performs on the larger data sets, as if it is still able to produce good quality output this means the evolutionary framework is not useful and/or the actual network structure is more important than the loss functions used.

Similar to our original research direction, we believe more work should be done to improve the fitness function as it does seem to be slightly better than random updates. If the fitness function was to point towards solutions for effectively, the evolutionary framework could act as a general framework for multiple problems, reducing the need to optimise hyper-parameters for each individual data set.

Chapter 6

Project management

Originally our research focused was on Niche Construction [12] and this is where we spent our time researching and writing a literature review. However a clear and testable research question which we could incorporate into existing models on the subject didn't seem feasible. This meant that the original GANT chart (Figure 6.1) and other preparations had to be revisited and adapted.

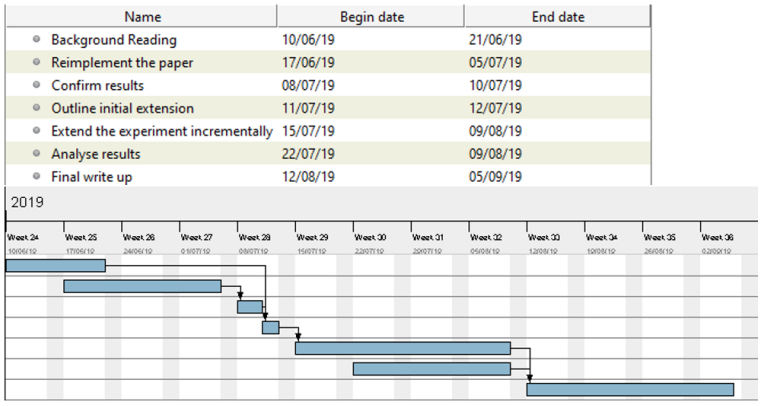

Figure 6.1: Original GANT chart from project preparation.

We therefore spent the first two weeks of the project exploring other routes until we settled on the E-GAN [15]. This continued the evolutionary theme while incorporating more advanced A.I technologies and techniques than would have been explored originally. In this paper the authors make use of a synthetic data sets which shows their algorithm working. This makes it a good candidate for re-implementation as it seems manageable in the time frame.

As there was limited time, and our previous work was not entirely applicable, we needed to begin work on re-implementing the model and develop background knowledge so we could properly outline and define a good research question. We decided to do both simultaneously as the authors provided code of their algorithm

and the background knowledge of GANs would be useful but not critical to start re-implementing it ourselves. We gave ourselves a 4 week period to complete this in as this would leave as ample time (around 6 weeks) to conduct experiments and write up.

Re-implementation turned out to be more complicated than we had originally thought. As the Python code for the experiments can be found on their GitHub page and they give pseudo code for their algorithm in their paper [15] we believed it would be straight forward to compare the two and understand what was going on. However, we did not find the code to be very easily readable as it was implemented in the 'Theano' deep learning framework, which we had no experience with and was not a very intuitive framework to understand. On top of this the dependencies to run the code didn't seem to work with each other, mainly because one of the library's was a developer mode and not formally released yet.

As we planned on extending the code, we needed it to be in a framework which was more easily understandable and editable. We decided to therefore try implimentation with Tensorflow, as this is the framework used in many deep learning experiments, including the original GAN by Ian Goodfellow [4]. Unfortunately this too brought about its own difficulties. While it was straight forward to create a GAN algorithm working on the toy Gaussian data sets, extending this to incorporate the E-GAN algorithm was not straight forward, despite the E-GAN framework being theoretically quite a simple extension. Despite our best efforts, and after spending a few weeks on this, we decided to cut our losses with TensorFlow and use Pytorch instead due to the fact it allowed networks to be created in a much more Python-esque way.

We were able to create the E-GAN algorithm quite quickly and managed to get some initial results with the Pytorch implementation. Although the parameters of the model were not entirely clear as some of the parameters given in the E-GAN paper [15] differ from those shown in the authors Theano GitHub code. Namely: Learning rate, discriminator training iterations and batch size. We decided to use the parameters as given in the code [15]. The authors ask to reference their paper rather than their GitHub page if any code was used, we do not use any code but do use their parameters.

We were able to create similar results, but only when using the default weight initialisation. If we use the same weight initialisation as theirs, the output of the generator becomes very constrained. It is likely that there is something not quite right about our network which makes it slightly different to theirs. However, even

after spending a week on this we were unable to determine the exact difference between our set up and theirs.

It was at this point we start to investigate simulated annealing on the algorithm (as detailed in methods). Once it became clear that this was not a viable option we had to change our plans again and work on a new extension. From our literature research it became clear that a big problem with GANs was the difficulty in evaluating them on a accepted metric, and seeing as the analysis on the synthetic data sets was only qualitative, this gave us the idea for our research project. However at this point we only had about 4 weeks left.

When it came to gathering the data the university GPU servers were down and so we could either wait and hope they come back up soon or try to gather data elsewhere. We decided on gathering the data elsewhere as we were aware of how much time had been spent so far on things which haven't gone to plan. This turned out to be the right decision as the servers were down for another 5 days.

This project has required us to change plans multiple times and adapt to circumstances faced either through a lack of our own knowledge, or external factors.

Bibliography

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[2] S. Barratt and R. Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

[3] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[5] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[6] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.

[7] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[9] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709, 2018.

[10] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[11] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2016.

[12] F. J. Odling-Smee, K. N. Laland, and M. W. Feldman. *Niche construction: the neglected process in evolution (MPB-37)*, volume 61. Princeton university press, 2013.

[13] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[14] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[15] C. Wang, C. Xu, X. Yao, and D. Tao. Evolutionary generative adversarial networks. *IEEE Transactions on Evolutionary Computation*, 2019.

Appendix

6.1 Table of contents

The code in the zip file contains 1) the code for the model and the log files (including the metric code), and 2) the code for getting the results such as boxplots and mutation selections.