Analysing the Loss Landscape Features of Generative Adversarial Networks

Jarrod Moses
University of Pretoria
Pretoria, South Africa
u19056665@tuks.co.za

Katherine M. Malan
University of South Africa
Pretoria, South Africa
malankm@unisa.ac.za

Anna S. Bosman
University of Pretoria
Pretoria, South Africa
annar@cs.up.ac.za

ABSTRACT

Generative adversarial networks (GANs) have recently gained popularity in artificial intelligence research due to their superior generation, enhancement and style transfer of content compared to other generative models. Introduced in 2014, GANs have been used in the fields of computer vision, natural language processing, medical applications, and cyber security, with the number of use cases rapidly growing. GANs are, however, difficult to train in practice due to their inherent high dimensionality, and the complexity associated with the adversarial learning task. Loss landscape analysis can aid in unravelling reasons for difficulty in training GANs as the analysis creates a topology of the search space. The vanilla and deep convolutional GAN architectures are examined in this study to gain a deeper understanding of their loss landscapes during training. The GAN loss landscape features are visualised through the use of loss gradient clouds (LGCs). The LGC analysis showcases the importance of volatility in the training of GANs, as a range of gradient magnitudes allows more exploration in finding an appropriate middle ground in balancing the loss objectives of the GAN.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Continuous space search.*

KEYWORDS

loss landscapes, fitness landscape analysis, generative adversarial networks, loss gradient clouds.

ACM Reference Format:
Jarrod Moses, Katherine M. Malan, and Anna S. Bosman. 2021. Analysing the Loss Landscape Features of Generative Adversarial Networks. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3449726.3463132

1 INTRODUCTION

GANs represent a relatively new approach in generating, enhancing or transferring styles of imagery, textual and video content. The architecture was developed by Goodfellow et al. [9], where two neural networks (NNs) compete adversarially against one another. The first NN is trained to generate noisy samples initially, and then over a number of iterations, to generate content from reference input training data. This model is referred to as the generator, and its outputs are commonly dubbed fakes in the context of GANs.

A second network, called the discriminator, is then fed a sample of data from the generator and training data. The discriminator aims to distinguish whether the sample originates from the training data or the fakes generated by the generator model. Initially, the discriminator can easily distinguish the fakes from the training data, which indirectly feeds back to the generator. The generator then adjusts to produce samples which makes it harder for the discriminator to distinguish between fake and real input. This loop is repeated as the generator attempts to produce content which the discriminator struggles to tell apart from the real training data. Figure 1 depicts the generator progressively producing improved imagery of a handbag over training.

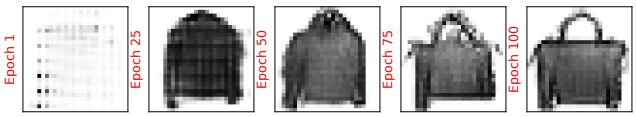

Figure 1: Generator producing the image of a handbag over training

The most successful application of GANs has been in the field of computer vision [22]. GANs have shown particularly impressive performance in enhancing low resolution imagery, style conversions between images, and producing non-existing realistic human faces. The applications of GANs continue to expand in the content generation fields of imagery, audio, video, natural language processing, cyber security through anomaly detection, and medical applications. Many GAN variations have been, and continue to be developed, including the Wasserstein GAN [2], Conditional GAN [20], Deep Convolutional GAN [23], CycleGAN [2] and SeqGAN [31].

An equilibrium solution in the context of GANs is where the generator produces a wide variety of samples which the discriminator fails to distinguish from real input data. GAN training is a high dimensional problem, which leads to difficulty in training, as the generator and discriminator networks may not end up with an equilibrium solution. In addition, the learning from the generator may not produce a wide diversity of outputs for input into the discriminator, known as mode collapse.

Loss landscape analysis (LLA) provides a framework to visualise the training process of GANs by characterising the topological features of the loss function in the context of a defined search space. This gives insight into the training of the GAN as the GAN searches

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07.
https://doi.org/10.1145/3449726.3463132

for an optimised solution. Visualisations of the training process can aid in understanding the most prominent features of the GANs during training. LLA also allows one to study what impact the algorithms' parameters have on the resultant landscape. The loss gradient cloud (LGC) [6] is a recently proposed tool for LLA which can be used to visualise and gain insight into the training process, and is the chosen method of analysis for this research.

By understanding the loss landscapes of the GANs through LLA, this study aims to achieve the following;

- Gain an understanding of loss landscape features which contribute to the volatility of training GANs.
- Investigate the usefulness of applying a recently proposed visualisation called loss gradient clouds to the loss landscapes of GANs.
- Visualise how the landscapes change by investigating the effect of adversarial training on the landscapes' dynamics.

The remainder of this paper is set out by firstly covering a review of the vanilla GAN and deep convolutional GAN (DCGAN), the difficulty in training GANs, followed by the practical use cases of GANs in Section 2. Section 3 conducts a literature review on loss landscapes and discusses the value of visualising training results through a LGC. Section 4 covers the experimental setup, followed by Section 5 which summarises the modelled results. Section 6 concludes the paper and discusses the limitations of this research and topics for future investigation.

2 GENERATIVE ADVERSARIAL NETWORKS

GANs were introduced in 2014 by Goodfellow et al. [9] and represent a unique approach to generating samples by learning the data distribution from sampled data. GAN research is a rapidly evolving field with many architectural improvements suggested since its introduction. This section firstly looks at the vanilla GAN's loss function to gain a better understanding of the training logic. This is followed by a discussion of the training difficulty, and the practical use cases of GANs.

2.1 GAN

A vanilla GAN consists of two NNs competing adversarially against each other. The NNs can range from fully-connected networks, convolutional networks, recurrent networks to auto-encoders with the differing architectures attempting to improve on the GAN's training. This research will focus on the fully-connected and convolutional network architectures.

The generator network is trained to produce samples which capture the underlying distribution of the training data. The discriminator network is fed a sample of training and generator data in order to estimate the probability of whether the sample originates from the training data or the fakes generated by the generator model. The discriminator output will initially result in a large loss for the generator in its early training iterations. The generator would then adjust its weights in order to minimise this loss. This process is then repeated with the generator aiming to produce samples which make it harder for the discriminator to distinguish between fake and real input. This loop is repeated in the hopes of finding an equilibrium state, where the generator produces fakes which the discriminator struggles to tell apart from the training data.

The objective of the discriminator network is to maximise its ability to distinguish between real and fake inputs. The objective of the generator network is to minimise the discriminator's ability to tell the difference between the fake and real data. These objectives are summarised with the following objective function [9].

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]. \quad (1)$$

where;

- G represents the generator model.
- D represents the discriminator model.
- $G(z)$ represents the data generated by the generator model.
- $p_{data}(x)$ is the probability distribution of the training data.
- $p_z(z)$ is the probability distribution of input noise variables.
- $D(x)$ is the probability that x comes from the training data not produced by the generator.
- $D(G(z))$ is the probability that $G(z)$ comes from the training data not produced by the generator.
- $x \sim p_{data}(x)$ represents x being sampled from the distribution $p_{data}(x)$.
- $z \sim p_z(z)$ represents z being sampled from the distribution $p_z(z)$.

The deep convolutional GAN (DCGAN) replaces the fully connected multi-layer perceptron structure with transposed convolutional layers in the generator network and convolutional layers in the discriminator network [23]. The DCGAN showed improved image generation performance compared to the vanilla GAN [23].

2.2 Training Difficulty

Training GANs is difficult in practice for the following reasons [21, 22]:

- The generator and discriminator networks may not end up with an equilibrium solution.
- The gradient of the generator tends to vanish as the discriminator improves to optimality, resulting in the generator producing a decreasing diversity of fakes for input into the discriminator. This is known as the GAN's mode collapsing.
- The adversarial nature of training may lead to unstable parameters, e.g. the weights of the NN changing drastically for small changes in the input space.

The search space of a GAN is composed of the set of weights used in the discriminator and generator. The loss landscape of a GAN can be defined as the loss function applied to each of the sampled weights during GAN training. By sampling from the search space and calculating the loss function for each of the sampled points, loss landscape properties can be examined by analysing the relationships between the calculated loss values [5].

2.3 GANs in Practice

Pan et al. [22] categorise the use cases of GANs primarily in the fields of computer vision and natural language processing and to a lesser extent in medical applications and cyber security through anomaly detection. The use of GANs in computer vision includes

enhancing the resolution of low-resolution images [17], converting the image style of one image to another, producing images with texture similar to input images and producing non-existing realistic looking faces. The application of GANs in the field of natural language processing includes the generation of text from speech, dialogue, poetry and music. The SeqGAN [31] showcased creative generation across poetry, speech and music input data.

3 LOSS LANDSCAPE ANALYSIS

For most optimisation problems in the machine learning context there is a loss function that captures the objectives of the problem at hand. Potential solutions are evaluated at each training iteration based on their loss values, which are determined using the loss function. The aim of a training algorithm is to solve the optimisation problem, which would usually be to find the solution that minimises the loss value after multiple iterations. The loss function can be broadened into a loss landscape by characterising the topological features of the loss function and some defined surrounding search space to gain insight into the training algorithm as it searches for an optimised solution.

3.1 Fitness Landscapes

The concept of fitness landscapes comes from the evolutionary context [29] and was later applied in the understanding of evolutionary search algorithms [12]. Fitness landscape analysis can be applied to any optimisation problem which has a defined objective function. Stadler [26] defines fitness landscapes as consisting of three elements, namely:

- A set X of possible solutions to the problem.
- A neighbourhood around the elements in X which can usually be defined by some distance, nearness or accessibility metric d to X.
- A fitness function.

This definition can be used for both continuous and discrete problem sets. Continuous optimisation problems can be defined in the case where X is all points in $\mathbb{R}^n$ (where n is the dimension of the search space) and d is the Euclidean distance [3]. The evaluation of continuous landscapes is more difficult in practice due to the infinite number of neighbours and the distances between a point and its neighbours compared to the discrete case which functions within a finite set.

3.2 Continuous Loss Landscapes

By applying fitness landscape analysis techniques in continuous spaces, the loss landscapes of the GAN can be studied. LLA provides a means of studying the behaviour of the loss function by estimating topological characteristics. Merkuryeva and Bolshakovs [19] liken a landscape to a surface in the search space defined by the loss values at each possible solution. In the context of NNs, Rakitianskaia et al. [24] highlight the search space representing all possible weight combinations forming the landscape with the resultant loss values corresponding to the weights. As a result, the search space of NNs is unbounded as the weights can be any real value. This requires a subset of the search space to be selected in order to estimate the loss landscape of NNs.

Bosman et al. [4] suggest using a range of regions when conducting LLA on NNs, with emphasis on the regions where the weights are initialised and the areas explored in training the NN. Loss landscapes can be particularly useful in trying to explain how and why NNs behave in certain ways during training due to the inherent black box nature and high dimensionality. LLA can be used to better understand optimisation problems by highlighting the most essential features in the landscape while training the algorithm. By studying these features, insight can be provided into why the algorithm performs well or struggles in the training phase. This research uses training samples to plot LGCs to study the continuous loss landscapes of GANs. Other sampling techniques and LLA metrics are excluded from this study.

3.3 Loss Gradient Clouds

Bosman et al. [3, 6, 7] introduce the concept of loss gradient clouds (LGCs), which provide a means of visually representing the clusters of low error values of a sampled search space in NN loss landscapes. The plot is produced by calculating the gradients and loss values for each of the sampled points. A scatter plot is then produced by plotting the norm of the gradient (y-axis) against its corresponding loss value (x-axis). When the norm of the gradient is zero, this represents a stationary point in the search space, which could be indicative of minima or saddle points. The LGC visualisation can therefore be used to identify the presence of 'basins' of attraction and can assist in understanding the nature of the search landscapes in terms of gradients and losses.

Although LGCs are a new idea, similar scatter plot visualisations have been used in two well known fitness landscape analysis techniques, fitness distance correlation [13] and fitness clouds [28]. While fitness distance correlation can also be used to visualise basins of attraction, it does not display gradient information and relies on knowledge of the global optimum. Fitness clouds, on the other hand, focus on displaying evolvability, which is a different aim to visualising the clusters of good solutions in error landscapes.

4 EXPERIMENTAL PROCEDURE

The aim of this study is to visually analyse features of the loss landscape of the vanilla GAN and DCGAN architectures in training, through the plotting of LGCs. This section details the experiments conducted to illustrate those loss landscape features.

4.1 Data

The vanilla GAN and DCGAN architectures were trained on the well-known machine learning MNIST [16], Fashion MNIST [30] and Cifar-10 [15] datasets. A summary of the datasets is briefly outlined below.

(1) MNIST: The dataset consists of 70,000 examples of 28×28 greyscale handwritten digits from 0 to 9. For the purposes of this study, a training sample consisting of 60,000 randomly sampled data points was used for analysis.
(2) Fashion MNIST: The dataset contains 70,000 examples of greyscale fashion images, labelled on a scale from 0 to 9, where each digit indicates a different type of fashion item (such as shirt or bag). The dataset shares the same image size and split between training and test sets as MNIST.

(3) Cifar-10: The dataset consists of 60,000 32×32 colour images in 10 classes ranging from vehicles, air planes and various animals. There are 50,000 training images and 10,000 test images. The random sample of 50,000 training images was used for further analysis.

4.2 GAN Architectures

The vanilla GAN is a simple architecture to implement, and is similar in structure compared to the original GAN introduced by Goodfellow et al. [9], while the DCGAN includes convolutional layers in the generator and discriminator networks.

4.2.1 Vanilla GAN. The vanilla GAN consists of fully connected generator and discriminator networks with one hidden layer sized to 100 units. A uniform random distribution ranging from $[-1, 1]$ is then used to initialise the weights of the generator. The size of the input vector is set to 20 and referred to as vector $\vec{z}$, while the size of the output vector is equal to the size of the image to be generated. The discriminator is fed the images $\tilde{x}$ generated by the generator, and the training image data x found in either the MNIST, Fashion MNIST or Cifar-10 datasets. The leaky ReLU activation function is applied to the hidden layers of both the generator and discriminator to improve their robustness [18]. A dropout layer with probability 0.5 is applied after the hidden layer in the discriminator to reduce overfitting.

The output layer of the generator uses the tanh activation function for improved learning, as recommended by Raschka and Mirjalili [25]. No activation function is applied to the final layer of the discriminator in order to get the non-normalized predictions of the network referred to as the logits of the prediction. Training on the Cifar-10 dataset has the same structure, however, the image dimensions are adjusted to 32×32×3 to account for the length, width and RGB colour palette of the Cifar-10 images. The total number of trainable parameters in the generator and discriminator networks for the vanilla GAN architecture is shown in Table 1.

Table 1: Vanilla GAN trainable parameters.

Network	Dataset	Number Parameters
Generator	MNIST	81, 184
Generator	Fashion MNIST	81, 184
Generator	Cifar-10	312, 272
Discriminator	MNIST	78, 601
Discriminator	Fashion MNIST	78, 601
Discriminator	Cifar-10	307, 401

4.2.2 DCGAN. The DCGAN consists of a series of transposed convolutional layers in the generator network and convolutional layers in the discriminator network. The generator initialises its weights from a uniform random distribution ranging from $[-1, 1]$. The size of the input vector is set to 20 and referred to as vector z. The size of vector z is then reshaped by applying a fully connected layer to z. A series of transposed convolutions then upsamples over four layers until the feature maps to the dimension of the required image size (28×28×1 for MNIST, Fashion MNIST and 32×32×3 for Cifar-10). The channel's size is halved at each transposed convolutional layer until the last layer which uses a single filter.

The parameters are set according to the DCGAN seen in Raschka and Mirjalili [25]. The first transposed convolutional layer uses a kernel size of $(5, 5)$ and strides of $(1, 1)$ as parameters. The second and third layers use a kernel size of $(5, 5)$ and strides of $(2, 2)$ as parameters followed by the last layer which uses a kernel size of $(5, 5)$ and strides of $(1, 1)$. All transposed convolutional layers use the same padding process to preserve the size of the outputs to that of the inputs into a layer. The transposed convolutional layers are followed by batch normalization and leaky ReLU activation functions up until the last layer, which uses the tanh activation function without batch normalization.

The discriminator is fed input from the output of the generator and the training datasets of either MNIST, MNIST Fashion or Cifar-10. A series of four convolutional layers then downsamples to the size of the kernel parameter (7×7×1 for MNIST and Fashion MNIST, 8×8×1 for Cifar-10). The last layer then fits a fully connected dense network to provide the logits (non-normalized predictions). The first transposed convolutional layer uses a kernel size of five and strides of $(1, 1)$ as parameters. The second and third layers use a kernel size of $(5, 5)$ and strides of $(2, 2)$ as parameters followed by the last layer which uses a kernel size of $(7, 7)$. All convolutional layers use the same padding process to preserve the size of the outputs to that of the inputs into a layer.

In addition, convolutional layers one to three are followed by batch normalization and leaky ReLU activation functions. Layers two and three also have a dropout probability of 0.3. The last layer is a fully connected dense layer with one unit which does not use any activation function in order to get the logits of the discriminator's prediction. Training on the Cifar-10 dataset has the same structure, however the image dimensions are adjusted to 32×32×3. The total number of trainable parameters in the generator and discriminator networks for the DCGAN architecture is shown in Table 2.

Table 2: DCGAN trainable parameters.

Network	Dataset	Number Parameters
Generator	MNIST	804, 832
Generator	Fashion MNIST	804, 832
Generator	Cifar-10	848, 672
Discriminator	MNIST	1, 039, 539
Discriminator	Fashion MNIST	1, 039, 539
Discriminator	Cifar-10	1, 042, 754

4.3 Data Preprocessing

The input pixels of the datasets lie in the range of $[0, 255]$. The datasets are preprocessed to a floating data type and scaled by a factor of 2 and shifted by -1 in order to rescale the pixel in the range of $[-1, 1]$. The rescaling is required due to the output layer of the generator model using the tanh activation function.

4.4 Training Procedure

The generator and discriminator use the Adam optimizer algorithm as the optimization method as it is well suited to high dimensional parameter problems due to its computational efficiency [14]. The Adam optimiser uses the default parameter settings of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-7}$. A batch (size 128) of input is

uniformly generated in range $[-1, 1]$, and sent through the generator to get output. The output is then fed to the discriminator model to classify whether the output is from the training data or from the generator. The generator loss is then calculated using the binary cross entropy loss function, which measures how close output from the generator is to being classified as real from the discriminator's classification. Small losses indicate the generator's ability to produce samples similar to the training data distribution, effectively fooling the discriminator's classification. Further training attempts to minimize the loss result in the generator iteratively producing images which the discriminator classifies as being real.

Processed data from the training datasets of MNIST, Fashion MNIST or Cifar-10 are then fed (for each separated training session) to the discriminator model. The discriminator's loss is then calculated using the binary cross entropy loss function, which measures how close the output from the discriminator is to classifying inputs from the training data as real and inputs from the generator model as fake. Small losses indicate that the discriminator is able to effectively tell apart the real and fake input.

Further training attempts to minimize the difference between these two losses. The hyper-parameter settings used is based on the work of Raschka and Mirjalili [25]. A summary of the generator and discriminator networks hyper-parameters for the vanilla GAN and DCGAN architectures is shown in Table 3 and Table 4, respectively.

Table 3: Vanilla GAN hyper-parameters.

Parameter	Value
Generator: Number hidden layers	1
Discriminator: Number hidden layers	1
Generator: Input size z	20
Number Epochs	100
Generator: Hidden layer size	100
Discriminator: Hidden layer size	100
Batch size	128

Table 4: DCGAN hyper-parameters.

Parameter	Value
Generator: Input size z	20
Number Epochs	100
Batch size	128

Runs are performed for each dataset and architecture for 100 epochs each. The losses and magnitude of gradients are stored at each iteration of training for both the discriminator and the generator. The output of the discriminator's classification is also stored at each iteration. A LGC is then plotted with the magnitude of gradients on the y-axis and losses on the x-axis across each iteration.

4.5 Implementation Tools

The implementation of the experimental setup is carried out in Jupyter notebooks based in Google Colab [10]. The notebook uses Python 3.6.7, which has a vast number of libraries, documentation and community support for the deep learning and plotting tools required to conduct this experiment. In addition, Google Colab provides limited access to machines with GPUs for model training, which reduces the resource constraints on running the models on local machinery. All the Python libraries utilised in this research are summarised below.

- Tensorflow 2.0, which is a machine learning framework [1], was used to build the GAN models and source the datasets.
- Numpy, which is a Python library for working with arrays [27], was used for all array and vector manipulation.
- Matplotlib, which is Python plotting library [11], was used to plot all the figures produced in this research.

5 EMPIRICAL RESULTS

The generator aims to learn the probability distribution of the training datasets over 100 epochs. LGC plots are then made by scatter plotting the norm of the gradient (y-axis) against the corresponding loss value (x-axis) of the generator and the discriminator's combined losses. The points on the plot are colour coded according to the training stage to track the movement of the cloud over its iterations. Training is subdivided into $1/5, 2/5, 3/5, 4/5$ and $5/5$ of completed iterations with colour coding of purple, orange, black, green, and yellow, respectively. For example, points coloured purple fall into the first $1/5$ of total iterations in the training of the 100 epochs.

Figures 2 - 4 show the LGCs for the vanilla GAN generator producing high gradients and losses in the initial iterations of training (purple dots). As the iterations increase, there is a gradual descent into areas of lower gradient and loss by epoch 100 (green to yellow dots). Stationary points (gradient equal to zero) are not reached, however the trend suggests that with further training stationarity could be reached. Once the gradients descended into the light green area, the LGC suggests that further iterations may find it increasingly difficult for the gradients to escape this area due to its localisation and smaller space size compared to earlier iteration points.

The LGC for the vanilla GAN discriminator shows a gradual descent from area of high gradient to lower gradient with the losses remaining within a narrow range throughout. The combined discriminator loss showcases gradual movement to defined ranges for both the gradients and losses with a broader area space compared to the generator, suggesting more scope for movement to other areas of loss with further training.

It is interesting to note convergence appears to occur away from zero gradients or loss even though these areas are approached in the initial iterations of training. This may be a result of the adversarial nature of training the generator against the discriminator. As the generator network produces improved fakes, the discriminator has a harder job of identifying the fakes from the real inputs leading to larger losses as the training progresses. Improvements in the one network come at the expense of the other network in GANs.

The DCGAN generator, shown in Figures 5 - 7, displays more outlier gradient magnitudes in the initial iterations of training compared to the vanilla GAN. However, the movement from areas of high to low gradients and losses is comparable. Another interesting observation is the more funnel-like shape the LGC forms through the iterations when compared to the movement of the vanilla GAN.

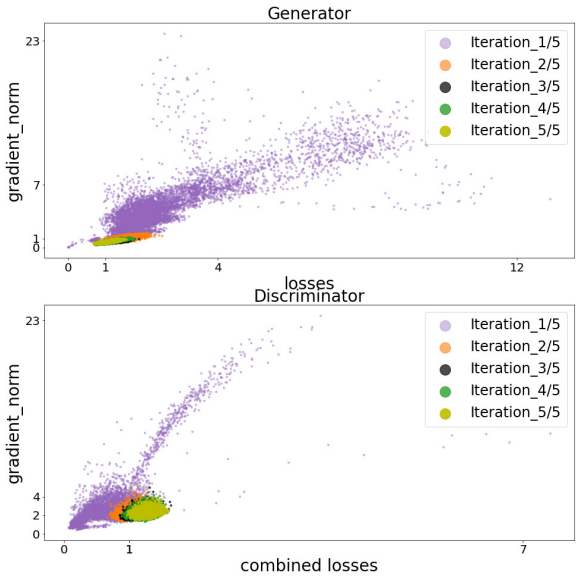

Figure 2: MNIST vanilla GAN LGC plots with unbounded ranges

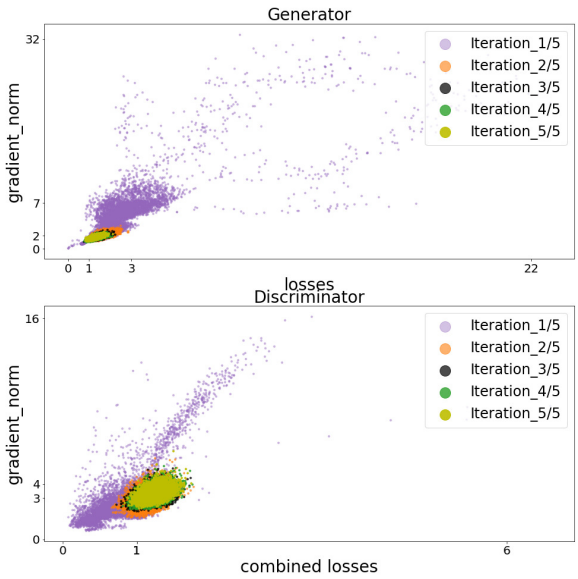

Figure 3: Fashion MNIST vanilla GAN LGC plots with unbounded ranges

Stationary points (gradient equal to zero) are not reached as the LGC suggests movement away from these points.

The generator plots confirms the movement away from stationarity in the last training iterations. The comparative size of the last stages of iterative training (light green area in the LGC) suggests that further iterations have scope for the gradients to escape compared to the more condensed vanilla GAN at this juncture of training.

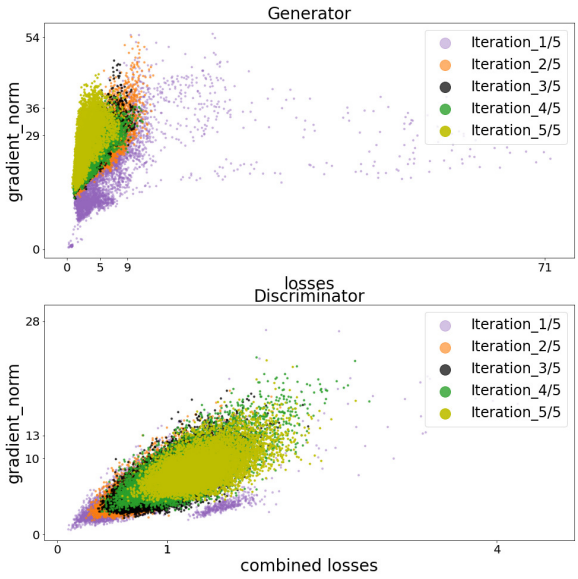

Figure 4: Cifar-10 vanilla GAN LGC plots with unbounded ranges

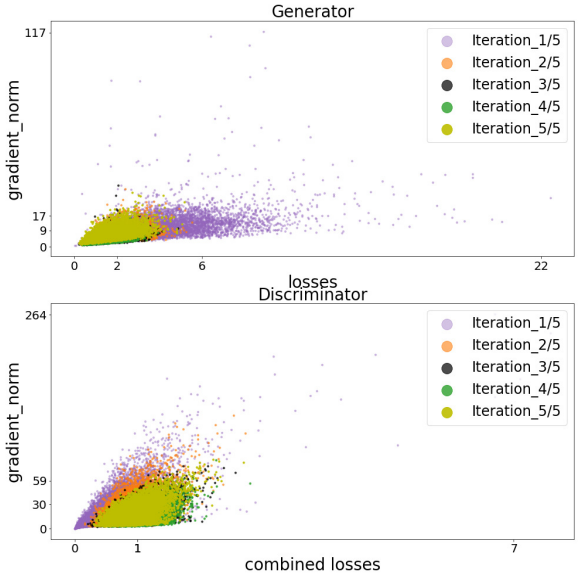

Figure 5: MNIST DCGAN LGC plots with unbounded ranges

The DCGAN combined discriminator loss showcases gradual movement to a large range for the gradient magnitudes and a more narrowly defined loss range. This suggests the landscape of the generator has few peaks of attraction judging from the low variation of the losses range. The discriminator reaches few points of stationarity in earlier iterations but never settles in this area due to the volatility associated with the abrupt changes of the calculated gradients and losses with each training iteration. The discriminator has a broader range of gradient norms but a narrower loss range compared to the generator.

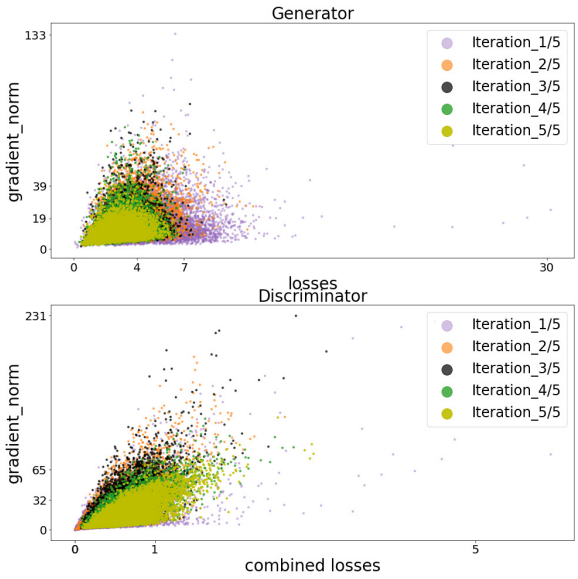

Figure 6: Fashion MNIST DCGAN LGC plots with unbounded ranges

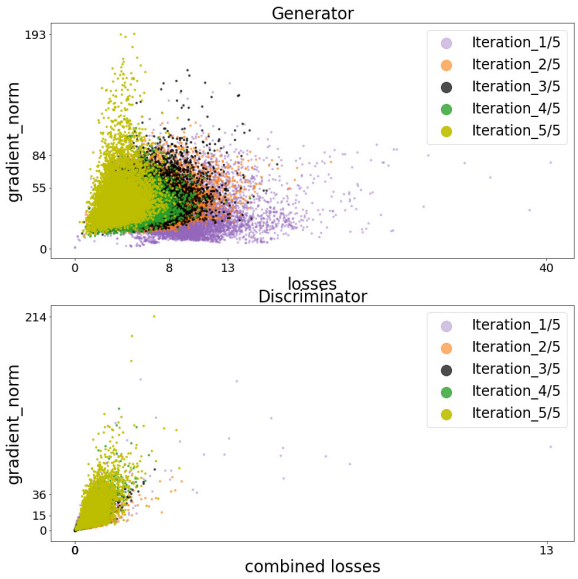

Figure 7: Cifar-10 DCGAN LGC plots with unbounded ranges

For all datasets the DCGAN generates samples which visually outperform the vanilla GAN. The LGC of the generator is more funnel shaped in comparison to the vanilla GAN with a broader range of gradient magnitudes and losses. As a result, the generator appears more capable of exploring a wider range of possible image generations in order to fool the discriminator in training.

LGC plots for the DCGANs show a wider range of gradient magnitudes compared to the vanilla GAN, allowing the discriminator to be more volatile in its prediction at each iteration. This volatility appears to boost the GAN's performance as the learning in correcting poorly generated samples is steeper, forcing the generator to perform better at each passing iteration. Samples produced from the generator's output for epochs 1, 25, 50, 75 and 100 are shown for the vanilla GAN and DCGAN in Figures 8 - 9.

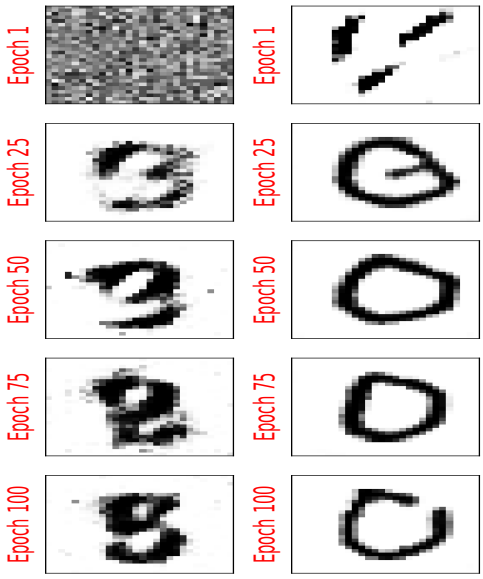

Figure 8: MNIST vanilla GAN (left) and DCGAN (right) generated samples

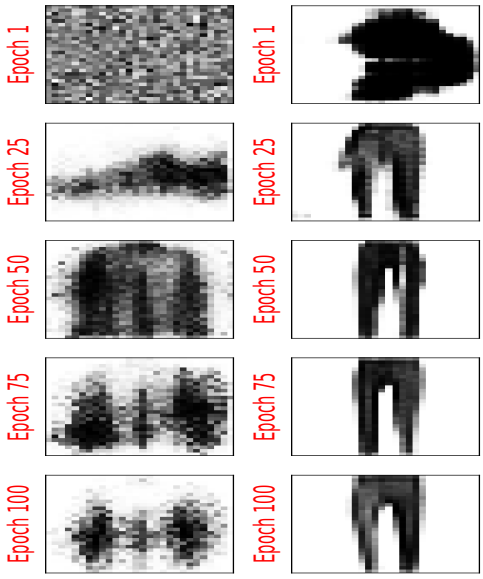

Figure 9: Fashion MNIST vanilla GAN (left) and DCGAN (right) generated samples

6 CONCLUSIONS

The LGC revealed the importance of volatility in the training of GANs. This is illustrated by DCGAN's broader range of gradient

magnitudes, allowing more exploration to find an appropriate middle ground in balancing the loss objectives of both the discriminator and generator. The vanilla GAN appeared to learn more gradually over each passing iteration in a more linear fashion. The movement of the vanilla GAN LGC is more distinct for each passing iteration, moving from a more expansive area to an area of greater constriction (shown by the colour palette of the LGC plots). The LGC for each passing training iteration is less distinguishable for the DCGAN, meaning the scale of learning remains fairly constant from the beginning to the end of the GAN's training.

The limitations and avenues for further work of this research include the following:

- This study only considered two GAN architectures. Doing comparisons on more GAN architectures may provide more insight than is currently presented. The Wasserstein GAN is of particular interest due to its use of the Earth-Mover distance which allows for improved discriminator training.
- The MNIST and Fashion MNIST datasets are similar in structure and shape. Choosing a wider variety of datasets may serve to further validate the findings in this research.
- A scalability study which uses more than 100 epochs for training can be explored.
- The findings may differ with the use of other optimization techniques e.g. RMSPROP.
- The study does not focus on hyper-parameter tuning of the GAN architectures.
- This research is limited to LGCs to understand the loss landscape of GAN training, various other loss landscape techniques can be employed to corroborate or question the findings presented here.

Further research can extend on the landscape analysis of GANs by analysing the curvature by taking the eigenvalues of the Hessian matrix [8]. As a result, the LGC can be classified as convex, concave, saddle or singular [6]. The number of GAN architectures is ever increasing as new techniques attempt to alleviate the difficulties associated with GAN training. Loss landscape analysis can be performed on a more varied array of GAN architectures to further understand these approaches by visualising the training by means of a LGC. A comparative study between the architectures can then be made.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] M. Arjovsky, S. Chintala, and L. Bottou. 2017. Wasserstein GAN. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70. Proceedings of Machine Learning Research, 214–223.

[3] A. S. Bosman. 2019. *Fitness Landscape Analysis of Feed-Forward Neural Networks*. Ph.D. Dissertation. University of Pretoria.

[4] A. S. Bosman, A. Engelbrecht, and M. Helbig. 2016. Search space boundaries in neural network error landscape analysis. In *IEEE Symposium Series on Computational Intelligence*. 1–8.

[5] A. S. Bosman, A. Engelbrecht, and M. Helbig. 2018. Progressive Gradient Walk for Neural Network Fitness Landscape Analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Kyoto, Japan). Association for Computing Machinery, 1473–1480. https://doi.org/10.1145/3205651.3208247

[6] A. S. Bosman, A. Engelbrecht, and M. Helbig. 2020. Visualising basins of attraction for the cross-entropy and the squared error neural network loss functions. *Neurocomputing* 400 (2020), 113–136. https://doi.org/10.1016/j.neucom.2020.02.113

[7] Anna Sergeevna Bosman, Andries Petrus Engelbrecht, and Marde Helbig. 2020. Loss Surface Modality of Feed-Forward Neural Network Architectures. In *2020 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN48605.2020.9206727

[8] C.H. Edwards. 1973. In *Advanced Calculus of Several Variables*. Academic Press, 142 – 159. https://doi.org/10.1016/B978-0-12-232550-2.50005-4

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[10] Google. 2018. Colaboratory: Frequently Asked Questions. Available: https://research.google.com/colaboratory/faq.html.

[11] J. D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering* 9, 3 (2007), 90–95.

[12] T. Jones. 1995. *Evolutionary Algorithms, Fitness Landscapes and Search*. Ph.D. Dissertation. The University of New Mexico.

[13] Terry Jones, Stephanie Forrest, et al. 1995. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms.. In *ICGA*, Vol. 95. 184–192.

[14] D. P. Kingma and J. Ba. 2015. Adam: A Method for Stochastic Optimization. In 3rd International Conference for Learning Representations, San Diego.

[15] A. Krizhevsky. 2012. *Learning Multiple Layers of Features from Tiny Images*. Technical Report.

[16] Y. LeCun and C. Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010).

[17] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *IEEE Conference on Computer Vision and Pattern Recognition*. 105–114.

[18] A. L. Maas, A. Y. Hannun, and A. Y. Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

[19] G. Merkuryeva and V. Bolshakovs. 2010. Comparative Analysis of Statistical and Information Measures for Benchmark Fitness Landscapes. In *2010 Fourth UKSim European Symposium on Computer Modeling and Simulation*. 96–101.

[20] M. Mirza and S. Osindero. 2014. Conditional Generative Adversarial Nets. *CoRR* abs/1411.1784 (2014). arXiv:1411.1784 http://arxiv.org/abs/1411.1784

[21] Z. Pan, W. Yu, B. Wang, H. Xie, V. S. Sheng, J. Lei, and S. Kwong. 2020. Loss Functions of Generative Adversarial Networks (GANs): Opportunities and Challenges. *IEEE Transactions on Emerging Topics in Computational Intelligence* (2020), 1–23.

[22] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng. 2019. Recent Progress on Generative Adversarial Networks (GANs): A Survey. *IEEE Access* 7 (2019), 36322–36333.

[23] A. Radford, L. Metz, and S. Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 [cs.LG] cite arxiv:1511.06434.

[24] A. S. Rakitianskaia, E. Bekker, K. M. Malan, and A. Engelbrecht. 2016. Analysis of error landscapes in multi-layered neural networks for classification. In *2016 IEEE Congress on Evolutionary Computation*. 5270–5277. https://doi.org/10.1109/CEC.2016.7748360

[25] S. Raschka and V. Mirjalili. 2019. Generative Adversarial Networks for Synthesizing New Data. In *Python Machine Learning 3rd Edition*. Packt Publishing Ltd., Chapter 17. https://github.com/rasbt/python-machine-learning-book-3rd-edition

[26] P.F. Stadler. 2002. Fitness landscapes. *In: Lässig M., Valleriani A. (eds) Biological Evolution and Statistical Physics*. 585 (2002), 183 – 204. https://doi.org/10.1007/3-540-45692-9_10

[27] S. van der Walt, S. C. Colbert, and G. Varoquaux. 2011. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering* 13, 2 (2011), 22–30.

[28] S. Verel, P. Collard, and M. Clergue. 2003. Where are bottlenecks in NK fitness landscapes?. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.* 273–280 Vol.1. https://doi.org/10.1109/CEC.2003.1299585

[29] S. Wright. 1932. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. *Proceedings of the International Congress on Genetics*, 1 (1932), 356–366.

[30] H. Xiao, K. Rasul, and R. Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* abs/1708.07747 (2017). arXiv:1708.07747 http://arxiv.org/abs/1708.07747

[31] L. Yu, W. Zhang, J. Wang, and Y. Yu. 2016. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *CoRR* abs/1609.05473 (2016). arXiv:1609.05473 http://arxiv.org/abs/1609.05473