

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

Santiago Duque
May 2020

**Evolutionary Training in Generative Adversarial
Networks**

Project supervisor: Srinandan Dasmahapatra
Second examiner: Xu Fang

A project report submitted for the award of BSc in Computer Science

Abstract

Ensemble methods in Machine Learning have been shown to be very powerful for classification and regression problems. Generative Adversarial Networks (GANs) are powerful tools consisting of two “competing” neural networks, the Generator, which -in the context of image production- tries to produce an image as close to the real ones as possible, and the Discriminator, which tries to classify images produced by the Generator as genuine or fake. These networks are then updated in each iteration based on their interactions, to perfect their image generation, in case of the Generator, and their classification power, in case of the Discriminator.

The goal of this project is to introduce Evolutionary Algorithms in the training of a GAN to improve the quality of its results and alleviate some of their inherent difficulties.

TensorFlow and various Python and Machine Learning / Data Science libraries will be used. Up to this point, research has been done and existing implementations of GANs and Evolutionary Algorithms have been run and analysed.

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 data, or animals.

Contents

1	Project Introduction	5
1.1	Project Description	5
1.1.1	Generative Adversarial Networks	5
1.1.2	Evolutionary Algorithms	6
1.1.3	System outline	7
2	Background and Literature Review	7
2.1	Approaches to enhancing GANs	7
2.1.1	Deep Convolutional GANs (DCGANs)	7
2.1.2	Wasserstein GAN	7
2.1.3	Least Squares GANs (LSGANs)	8
2.1.4	Consistent Adversarial Training Enhanced GANs (CAGANs)	8
2.1.5	Variational Encoder Enhancement to Generative Adversarial Networks (VEE-GAN)	8
2.2	Evolutionary Generative Adversarial Networks	8
3	Analysis of the problem and solution	9
3.1	Problems with GANs	9
3.1.1	Vanishing gradients	9
3.1.2	Mode Collapse	10
4	System design and implementation	11
4.1	Development Framework	11
4.2	Design iterations	12
4.2.1	Structure	12
4.3	First steps	12
4.4	First goals	13
4.5	Evolutionary Algorithms on CIFAR10 and CelebA	13
4.5.1	Diminishing mutation rate	14
4.6	Intelligent design	14
4.7	Pseudocode	15
5	Evaluation and results	16
5.1	Evaluation on mixture of Gaussians	16
5.2	Inception Scores	16
5.3	Fréchet Inception Distance	17
5.4	Evaluation constraints	18
5.5	Inception Score results	18
5.6	FID score results	18
5.6.1	FID scores and Diminishing Evolution	19
5.7	Other observations	20
5.7.1	Collapse recovery	20
5.7.2	Drawbacks	20

6	Conclusions and future work	20
6.1	Constraints and elements to improve	20
6.2	Future Work	21
7	Project management	21
7.1	External issues	21
7.2	Progress	21
7.2.1	GANTT Chart	22
7.2.2	Project Brief	22

1 Project Introduction

A Generative Adversarial Network (henceforth referred to as 'GAN') is a machine learning system with two parts: a Generator and a Discriminator. The Generator is given a training dataset, and is tasked with producing data that resembles the true data distribution as much as possible, and the Discriminator attempts to discern whether a particular candidate belongs to the actual data distribution or was faked by the Generator. They were first proposed in 2014 by Ian Goodfellow and his team from the University of Montreal (Goodfellow et al. 2014). GANs have found several uses in the last few years, including astronomy (Schawinski et al. 2017), image-to-image translation (Zhu et al. 2018), 3D modelling and face generation (the notorious "This person does not exist" website, created with StyleGAN)(Karras et al. 2019).

Evolutionary Algorithms are a class of algorithms inspired by the processes that drive biological evolution. A set of candidate solutions is randomly generated; each evaluated through some fitness function. Depending on the score given by the fitness function to each candidate, it will be selected or discarded for the next generation; passing on its traits through recombination and mutation. (Vikhar 2016) Genetic Algorithms are a subclass of Evolutionary Algorithms. They are specifically inspired by the biological concepts of mutation, crossover and natural selection (Vikhar 2016). Given a distribution of candidate solutions, a problem-dependent fitness function selects the best ones to pass their traits onto the next generation. This is done through the operators of mutation (unary) and crossover (binary or n-ary, that is, two or more parents for each child). Evolutionary and specifically Genetic Algorithms have also found a variety of uses since they were first proposed in 1960 (Mitchell 1996), including: Image processing (dos Santos-Paulino et al. 2014), automated design (Li et al. 2004), vehicle routing (Vidal et al. 2012) and anti-terrorism systems (Buurman et al. 2009).

Although effective in their results, GANs have certain difficulties and problems in their training. If the original data distribution and the generated data distribution do not overlap substantially, the Generator update gradients might be inadequate and even too small, resulting in the vanishing gradients problem (Wang et al. 2018). The other, arguably even more prevalent problem is mode collapse: the Generator focuses on a very limited space of samples, or even only one sample (Metz et al. 2017). While the produced images are good, a desired quality of the final result is diversity, that is, images belonging to as many different categories as possible. Traditional GANs tend to collapse to only one category.

Introducing Evolutionary metaheuristics to GANs could be a way to alleviate or even fix these problems, as well as producing better results in the form of more convincing images.

1.1 Project Description

In this section, GANs and Evolutionary Algorithms are described in more detail, and then we outline the system that pieces both together.

1.1.1 Generative Adversarial Networks

A GAN is a Machine Learning system consisting of two Neural Networks, the Generator and the Discriminator. The Generator produces samples that attempt to be as close as possible to the training data. It first starts with a noise distribution, in order to introduce a stochastic element. The Discriminator is tasked with determining whether a certain sample belongs to the training data

(is "real") or has been produced by the Generator (is "fake"). The output of the Discriminator for each sample is either a 0, if it believes it is fake or a 1, if it believes it is genuine. In each iteration, therefore, the Discriminator is fed a batch of real images, and its output compared to a string of 1s, and a batch of images created by the Generator, and its output compared to a string of 0s. This results in the loss function, which is then used to update the weights of both the Discriminator and the Generator. In each iteration, then, the Generator and Discriminator update each other, and get progressively better at their tasks. The desired result is generally the output of the Generator.

Formally, GANs are a structured probabilistic model. The input to the Generator is a random variable z sampled from a noise distribution $Pz(z)$. The Discriminator is represented by a function D , with input x (the output of the Generator and the real data distribution), which uses $\theta^{(D)}$ as parameters. The Generator is represented by a function G with input z and parameters $\theta^{(G)}$. Both players have their own cost functions. The Discriminator wants to minimise the cost function $J^{(D)}(\theta^{(D)}, \theta^{(G)})$, while only having control of $\theta^{(D)}$; the Generator wants to minimise $J^{(G)}(\theta^{(D)}, \theta^{(G)})$, while only having control of $\theta^{(G)}$. This means that the process is a game-like problem, and its solution is a Nash equilibrium. (Goodfellow 2016). This solution is represented by the value function:

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

where D is the Discriminator and G is the Generator (Goodfellow et al. 2014).

Both Discriminator and Generator have a cost/loss function with which they update their gradients.

The Discriminator function:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2} \mathbb{E}_{x \sim p_{data}(x)} \log D(x) - \frac{1}{2} \mathbb{E}_z \log(1 - D(G(z))) \quad [1]$$

The Generator function depends on the way one wishes to update the Generator, for example: as a zero-sum game (Minimax), heuristically... The function chosen in this project is the second one proposed by Goodfellow: the heuristic, non-saturating function. The reason being that a Minimax approach does not work well in practice, and using cross-entropy minimisation for the Generator highly reduces cost saturation (Goodfellow 2016):

$$J^{(G)} = -\frac{1}{2} \mathbb{E}_z \log D(G(z)) \quad [2]$$

1.1.2 Evolutionary Algorithms

Evolutionary Algorithms are inspired by the processes that drive biological evolution, namely reproduction, mutation, recombination and natural selection. A set of candidate solutions is randomly generated; each evaluated through some fitness function. Depending on the score given by the fitness function to each candidate, it will be selected or discarded for the next generation; passing on its traits through recombination and mutation.

The most common evolutionary methods (i.e., ways of evolving the population for selection) are mutation and crossover (also known as asexual and sexual evolution, respectively).

1.1.3 System outline

The base idea behind a GAN with evolutionary training would be as follows. Multiple Generators act as the evolutionary population. A single Discriminator acts as the evolutionary environment. The Generators are then evolved, either asexually (mutations) or sexually (as offspring of two or more parent Generators). As with Evolutionary Algorithms, a fitness function evaluates the performance of each Generator. As with standard GANs, this fitness function is obtained from the performance of the Discriminator with each Generator.

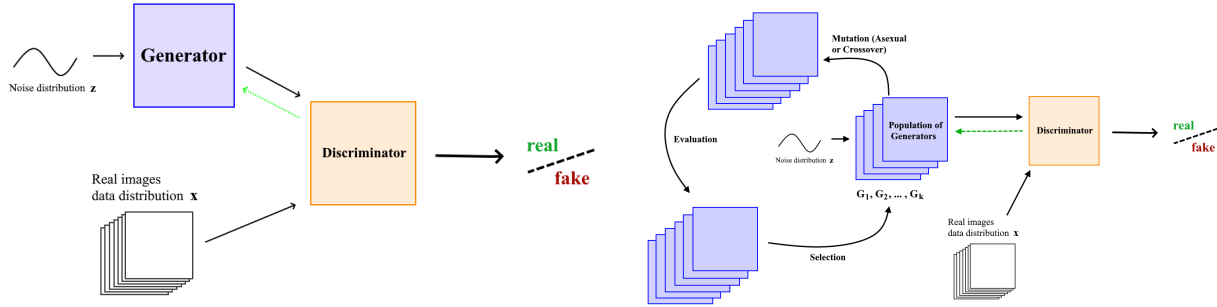


Fig 1 - Frameworks for the original GAN and a GAN with Evolutionary Algorithms, respectively.

The latter has a collection of Generators which evolve in each iteration, based on their performance against the Discriminator. *Mutation* consists of varying the traits (layers and weights) of the Generators, either individually, stochastically altering certain weights (asexual evolution), or by mixing layers and/or weights of other Generators (sexual evolution).

This system will be further detailed in section 3.

2 Background and Literature Review

2.1 Approaches to enhancing GANs

There have been numerous attempts at improving the performance of GANs, improving the quality of their outputs, and solving or alleviating some of their inherent issues, such as mode collapse.

2.1.1 Deep Convolutional GANs (DCGANs)

(Radford et al. 2016) introduced a GAN model for unsupervised learning, on which most GAN models are currently based (Goodfellow 2016). It uses an all-convolutional net (Springenberg et al. 2015) in the network, which replaces deterministic spatial pooling functions with strided convolutions, thus allowing the Generator to learn its own spatial upsampling. The model reshapes fully connected layers of the Discriminator and Generator, improving stability in training (global average pooling, Mordvintsev et al. (2015)). It also employs batch normalisation (Ioffe & Szegedy 2016), which helps with training stability by normalising the inputs to zero mean and unit variance.

2.1.2 Wasserstein GAN

(Arjovsky et al. 2017) proposed using the Wasserstein Distance (also known as the Earth Mover's Distance) to evaluate the discrepancy between the real data and the produced distribution. This

measure would reduce instability in the training of the GAN and alleviate or even solve mode collapse. A further description of this measure is given in section 3.1.1.

2.1.3 Least Squares GANs (LSGANs)

(Mao et al. 2017) proposed a GAN model that employs the Least Squares function as a loss function for the Discriminator, to improve results and increase stability in training, by alleviating problems such as the 'vanishing gradient', where the backpropagation gradient becomes too small for the training rate.

2.1.4 Consistent Adversarial Training Enhanced GANs (CAGANs)

(Ni et al. 2018) implemented a GAN system where one Generator produces output for an exponential number of Discriminators. This way, Discriminators are encouraged to be consistent with real data samples, whereas Generators are encouraged to produce consistent samples for the different Discriminators. The Discriminators are generated via dropout - Discriminator subnetworks are created from the Discriminator network by adding hidden dropout layers, which allows for infinite Discriminators in practice, thereby avoiding Generator overfitting.

2.1.5 Variational Encoder Enhancement to Generative Adversarial Networks (VEE-GAN)

(Srivastava et al. 2017) proposed a reconstruction network to resist mode collapse in training and to produce "more realistic samples". This additional network is trained to map the real data distribution to a Gaussian, and also to invert the Generator network.

2.2 Evolutionary Generative Adversarial Networks

Similar approaches towards improving GANs with Evolutionary heuristics have been undertaken over the past few years. C. Wang et al first proposed E-GANs in 2018 (Wang et al. 2018). They used three types of unary mutations - minimax, heuristic and least-squares mutations; and the Adam gradient descent optimisation for updates, a very common algorithm in current implementations of GANs, as it is computationally efficient and relatively intuitive (Kingma & Ba 2015). In an E-GAN, Generators are considered the evolutionary population, and the Discriminator is the evolutionary environment. In each iteration, Generators mutate in accordance with the current Discriminator environment. The Generators receive a score through the fitness function, and only the n-best performing, or all those with a score above a certain threshold, proceed to the next iteration. A standard GAN has a static training objective. An E-GAN, in contrast, dynamically evolves a solution, which helps suppress or potentially eliminate inherent problems such as mode collapse or vanishing gradient of standard GANs.

(Al-Dujaili et al. 2018) introduced Coevolutionary Algorithms to enhance the performance of GANs. A Coevolutionary Algorithm is an Evolutionary Algorithm where the fitness of an evolutionary actor is subjective, that is, it comes from its interaction with other evolutionary actors (Popovici et al. 2018). The GAN consists of two independent populations, the population of Generators $P_u = \{U, ..., Ut\}$ and of Discriminators $P_v = \{V, ..., Vt\}$, t being the size of the population. The coevolutionary dynamic utilised is predator-prey: The Generator population aims to find Generators which evaluate to low L values (L being the minmax function of the GAN) within the Discriminator

population, and the goal of the Discriminator population is to find Discriminators that evaluate to high L value within the Generators. Throughout several iterations, the fitness of each Generator and each Discriminator is evaluated based on their interactions with other Generators $ui \in Pu$ and Discriminators $vj \in Pv$.

(Toutouh et al. 2019) introduced an enhanced version of GANs that utilises both Evolutionary Algorithms and the Lipizzaner system. The Lipizzaner system employs a "spatial distributed competitive coevolutionary algorithm" (Toutouh et al. 2018): Elements of the Generator and Discriminator populations are placed on a grid, and each actor gets evaluated against its neighbouring elements, thus making use of neighbour communication.

The proposed "Mustangs" (*MU*tation *Spa*Tial *g*ANs) combine the diversity in mutation and genome space training of both E-GANs and Lipizzaner GANs.

3 Analysis of the problem and solution

GANs suffer from inherent problems, which will be discussed in the next subsection. However, the intent here is not so much to completely solve those problems, which have been tackled in numerous occasions since the advent of GANs in 2014, but rather to explore the possibilities of introducing evolutionary elements in GANs, e.g.: to find out which problems they solve, whether they introduce any problems, their caveats and nuances, the advantages and disadvantages they bring over other alterations made to GANs, etc.

3.1 Problems with GANs

3.1.1 Vanishing gradients

The *vanishing gradient* problem occurs when the gradient of a neural network becomes too small (i.e., almost horizontal), preventing the weights of the system from changing their value, and thus potentially halting training completely. In the context of GANs, this happens when the Discriminator tends to optimality (that is, it is too good). This is a problem with the initial objective function (the Jensen-Shannon, or JS function), originally defined in (Goodfellow 2016).

As proven in Theorem 2.4 of (Arjovsky & Bottou 2017), as the Discriminator gets increasingly better and approaches optimality, the gradients of the Generator vanish, which causes this problem.

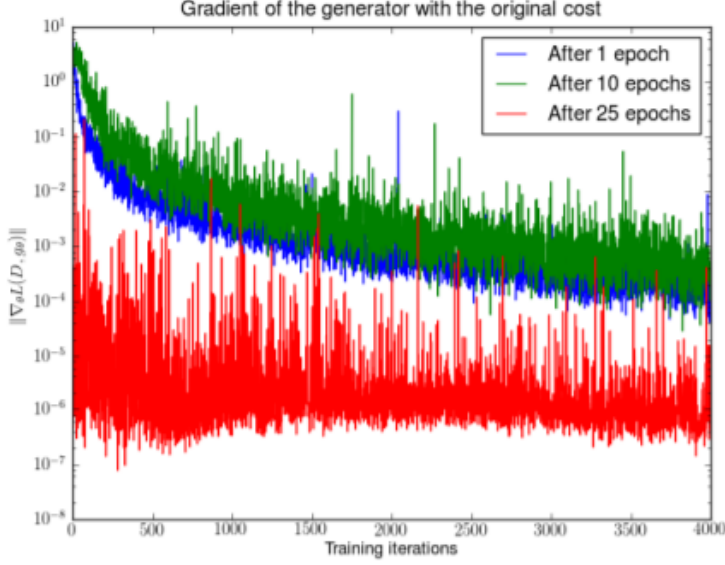


Fig 2 - Experiment from Arjovsky’s and Bottou’s research paper, plotting the cost function gradients per training iteration.

A DCGAN is trained for 1, 10 and 25 epochs. Then, a Discriminator is trained from scratch, and with a fixed Generator, its cost function gradients are measured. The gradients do indeed decay very quickly (note the logarithmic scale) (Arjovsky & Bottou 2017).

To solve this problem, the most common approach is to utilise a different objective function, such as the Wasserstein-1 distance, used in the aforementioned ‘Wasserstein GANs’. (Arjovsky et al. 2017) Briefly, this metric, also referred to as the ‘Earth-Mover’s Distance’, is defined as:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|] ,$$

where X is a compact metric set, $\text{Prob}(X)$ is the set of probability measures on X , $P_r, P_g \in \text{Prob}(X)$ are two distributions between which the distance and divergence will be measured, and $\Pi(P_r, P_g)$ is the set of all joint distributions $\gamma(x, y)$ whose marginals are P_r and P_g , respectively. $\gamma(x, y)$ indicates the amount of ‘mass’ needed to transport from x to y in order to transform P_r into P_g . Wasserstein-1 thus defines the cost of the optimal ‘transport’ plan. (Arjovsky et al. 2017) contains a more thorough explanation, along with examples for the Wasserstein-1 distance compared to other metrics.

3.1.2 Mode Collapse

Goodfellow, in the 2016 NIPS Tutorial for GANs, describes Mode Collapse, also known as the *Helvetica scenario*, as the problem that occurs when the generator ends up mapping several different input \mathbf{z} values to the same output point. That is, it produces images from only one category (total mode collapse) or considerably fewer categories than the input data distribution (partial mode collapse) (Goodfellow 2016). Total mode collapse is rare, but partial mode collapse is very much not so.

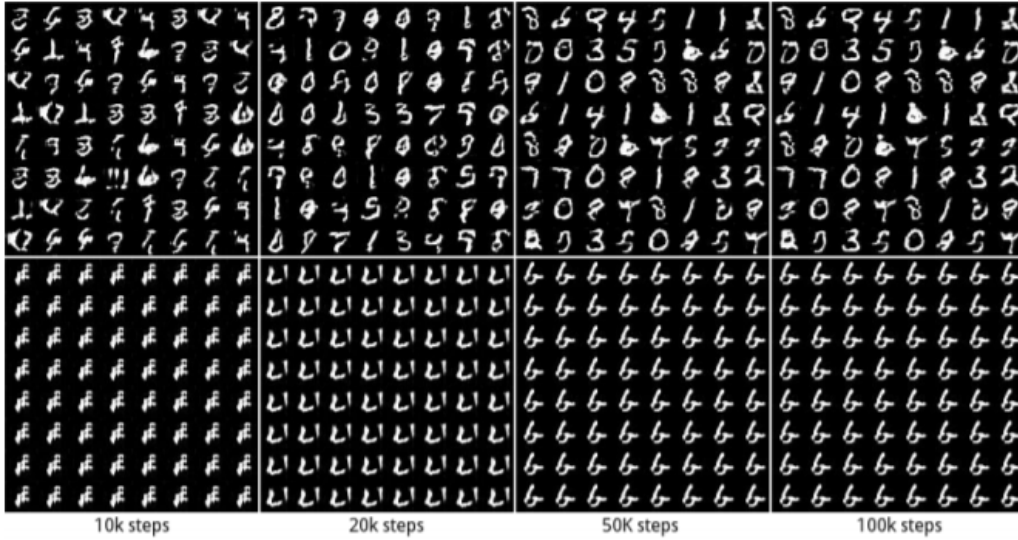


Fig 3 - Images produced by a GAN with the MNIST Digits Dataset, taken from Unrolled GANs (Metz et al. 2017).

The top row shows an uncollapsed GAN, while the bottom row shows a GAN that has suffered from total mode collapse: it is only producing a digit from the same category.

This is arguably the most common problem with GANs, and consequently the one with the most mitigations.

For example, AdaGAN (Tolstikhin et al. 2017) is inspired by Boosting as a Machine Learning meta-algorithm: the GAN is trained sequentially by adding a new Generator based on the performance of the Discriminator and the previous Generator, thereby decreasing the probabilities of missing the categories that were missed in the previous iteration.

In a different approach, MAD-GANs (Ghosh et al. 2017) also employs multiple Generators, but the Discriminator is also tasked with discerning which Generator produced each sample, in addition to determining whether it is real or fake. The only way for the Discriminator to do this is to push the training of each Generator towards diverse identifiable categories, thus reducing mode collapse.

4 System design and implementation

The final system was built via an iterative process, with the previous work and problems to solve in mind. In this section we will describe the framework and environment used for development, and detail the process taken to develop the system, with its different iterations across time, as well as the proposed final system.

4.1 Development Framework

The project was developed in the TensorFlow framework using the Anaconda package management distribution for Python 3. The main library is Keras, which is an extensively used library for accessible development of Neural Networks and other Machine Learning tasks. The main work was developed using Jupyter Notebook, a very flexible virtual environment that facilitates the iterative task of development.

4.2 Design iterations

4.2.1 Structure

The first design iteration consists of a working standard DCGAN (Radford et al. 2016) for the MNIST handwritten digits datasets(LeCun et al. n.d.). Future iterations are built around this first system, and all will use the DCGAN as a baseline.

A DCGAN is comprised, as most GANs, of a Generator and a Discriminator. The Discriminator contains Convolutional Layers, which have a set of learnable filters, commonly known as kernels, that extend the depth of the input volume. The Generator contains Convolutional Transpose Layers which, in addition to convolving the input, apply a 2D transpose operator. These are used to upsample the image shape to the desired size.

4.3 First steps

The initial changes made to the standard DCGAN was increasing the number of Generators. The first challenge was already clear: how to best make every Generator be properly updated in each iteration, as the performance of the Discriminator on its images impacts how its weights change. Some ideas were carried out, like taking the mean or the sum of the losses of all Generators as the update to the Discriminator. These alternatives were quickly discarded in favour of simply iteratively training each Generator with regards to the Discriminator.



Fig 4 - The produced images of three MNIST Generators

The model has been trained with the "fake" loss passed to the Discriminator as the average of the losses of the Generators. Mode collapse occurred, as each Generator ended up producing very similar digits, from no more than three different categories.

Then, a few simple mutations were introduced. These consisted of randomly changing a few weights of each layer of each Generator, without any selection factor. The MNIST Digits Dataset is generally used to confirm that a model works properly and does not break, so no further changes were made at this point.

It should be mentioned that "mutations" in this model are understood in a different way than mutations in other Evolutionary GANs, like (Wang et al. 2018). In those models, a mutation takes the place of the update performed according to the loss gradient. In our case, this update is

performed normally, as would be on any other non-evolutionary GAN. Mutations here are literal changes to the weights of the model, in addition to the weight updates from each loss iteration, and completely independent from them.

4.4 First goals

One central objective of this project was devised soon after the first iterations. This goal would be to accelerate convergence of the mode, that is, to reach a certain level of performance faster (in fewer training iterations) than a standard GAN model. The reasoning behind this objective is that mutations serve as the natural weight changes that occur after every model update in an epoch. Since mutations are random, they could very well hinder or undo the progress of the standard weight changes. However, the expectation is that the selection factor would get rid of harmful mutations, and keep beneficial ones, thus acting as an "accelerator" of the weight update of sorts.

4.5 Evolutionary Algorithms on CIFAR10 and CelebA

The changes proper were introduced in models fit to produced images based on the CIFAR-10 dataset, which consists of 60,000 32x32 images of real objects, animals, etc., divided in 10 classes (Krizhevsky 2009), and the CelebA dataset, which consists of more than 200,000 images of faces, aligned to 218x178 (Liu et al. 2015). The images from the latter were resized to 32x32, as to ease the usage of computational resources.

The other main evolutionary components, selection and crossover, are introduced here. The n worst Generators are selected and eliminated. They are then substituted by offspring from the surviving Generators. This offspring is produced semi-randomly, mimicking actual biological reproduction: each individual weight is chosen randomly to be inherited from either one of the parents.

Individual mutations are also refined. Several factors are introduced: m , the mutation amount (i.e., the range from which the mutation amount will be randomly applied), p , the mutation probability, and s , the mutation severity (i.e., how many layers are affected by the mutations, and how deeply). All these factors allow for a great potential of fine-tuning to achieve the best results possible.



Fig 5 - Some faces generated by the evolutionary model trained on the CelebA dataset.

A single generator can produce faces of different ethnicities, both genders and varied facial expressions (though the most common is a smile, since it is overwhelmingly the most frequent expression in the original dataset). Some faces even include glasses or hats. The Generators, thus, encompass a wide variety of modes, and so the diversity of the fake images is excellent.

4.5.1 Diminishing mutation rate

After some iterations with these configurations, one conclusion was reached after a key observation: mutations close to convergence decrease performance. That is, after a certain number of epochs, mutations do not help the model converge, but instead hinder it. This is logical, since the closer the model is to convergence, the less its weights change, and harmful mutations which would otherwise be swiftly overshadowed by the regular updates and evolutionary selection, takes the model many more iterations to get rid of. Models with a constant rate of mutation and crossover/selection performed as expected until a certain point (which varied depending on the configuration and dataset used), and after that point, its output quality would slow down considerably, compared to a standard non-evolutionary model, or even worsen in some cases.

The concept of diminishing evolution was then introduced: the Generators mutate more at the beginning of the training process and get eliminated or breed more often, and the frequency with which they do so, decreases as more epochs elapse.



Fig 6 - Some (cherry-

picked) images produced by the CIFAR-10-trained evolutionary model.

Many images clearly depict an object of a particular category; however, many other seem to display two or three different objects, lowering the overall quality.

4.6 Intelligent design

A different idea was to store a history of the weights which change the most during a run of a standard DCGAN, and then take those weights into account when performing the mutations. This concept was named 'Intelligent Design' (after the pseudo-scientific notion that a deity purposefully guides evolution). Furthermore, the model would keep track of its most changing weights, and then factor that into the mutations, e.g. by giving said weights a higher probability of mutating. This addition did not radically change the functioning of the GAN, though it was observed that a consistent use of this feature did slightly improve the performance of the model. As detailed in section 6.1, further parameter tuning could significantly improve this inclusion, and thus potentially make it a very beneficial feature.

4.7 Pseudocode

The following would be a rough pseudocode of a GAN with the previously described evolutionary components - mutation, selection and crossover.

Pseudocode 4.1: Evolutionary GAN.

```
1  input: Batch size  $m$ , number of Generators  $n_g$ , sexual mutation hyperparameter  $s$  (offspring
2  and selection frequency), asexual mutation hyperparameter  $a$  (mutation frequency),
3  image shape  $d$ , number of training epochs  $n_e$ 
4
5  begin
6    for (epoch  $e$  in  $n_e$ ) do:
7      sample a batch of training data  $x$  and noise  $z$  of dimension  $d$ 
8      for (Generator  $g$  in  $n_g$ ) do:
9        generator_image_list  $\leftarrow g.train(z)$ 
10     end
11     discriminator.train_valid( $x$ ) # Discriminator's performance on the real images
12     for (image batch  $f$  in generator_image_list) do:
13       discriminator.train_fake( $f$ ) # Discriminator's performance on the fake images
14     end
15     if ( $e \% a = 0$ ): # asexual mutation
16       for (Generator  $g$  in  $n_g$ ) do:
17          $g.mutate()$ 
18       end
19     end
20     if ( $e \% s = 0$ ): # crossover offspring production / selection
21       for (Generator  $g$  in  $n_g$ ) do:
22         if ( $g = \text{worst\_generator}$ ):
23            $g.kill$ 
24         else:
25           parent_list.add( $g$ )
26         end
27       end
28       parent_list.reproduce()
29     end
30   end
31 end
```

5 Evaluation and results

Since it is not in our interest to compare the performance of our models with other GAN alternatives, but strictly to analyse the results of adding certain evolutionary components, we perform our evaluation comparing the respective performances of a standard DCGAN, and a DCGAN with our evolutionary traits.

Models with different parameters were tested. The overall expectation for the final models was that mutations would accelerate convergence during the middle stages of training, and not harm performance due to negative mutations when nearing the end of training. Indeed, that expectation was mostly satisfied. To achieve this, a certain amount of parameter tuning was carried out. However, given the limitations of the project, it is most likely not the optimal combination of parameters, as will be expanded upon in the conclusion (section 6).

5.1 Evaluation on mixture of Gaussians

Mixtures of 2D Gaussians are generally used to evaluate GANs with regards to their convergence speed and susceptibility to mode collapse.

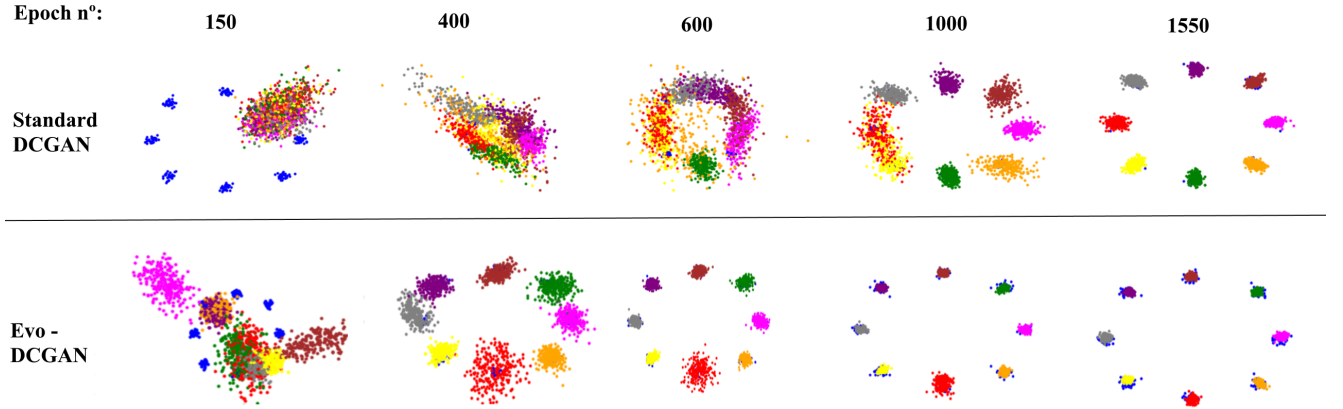


Fig 7 - Evaluation performed on a mixture of 8 2D Gaussian distributions, with 8 Generators.

The evaluation was carried out on a model with limited mutations and crossover, as the architecture is much simpler than that of a model which generates images. The evolutionary components do not affect mode distribution (as mode collapse is not even present in the original, unchanged DCGAN). However, the speed with which it converges to each node is much faster, and when it does, the distribution of points becomes much tighter, much sooner.

5.2 Inception Scores

One of the inherent problems with GANs and other image-generation tools is that there is no objective way of evaluating the quality of the generated images. That is, some images might "look better" or "more realistic" than others, but that is not an objective and reproducible criterion by which to compare different types of architectures.

For this reason, the Inception Score (henceforth IS) was devised by Tim Salimans, along with Ian Goodfellow -the creator of GANs- and others, in an attempt to remove the subjective factor in the evaluation of GANs (Salimans et al. 2016). This score aims to analyse two key properties of

generated images: quality (how closely the produced image resembles a real object) and diversity (how wide is the range of objects generated).

Entropy is the average level of uncertainty regarding the possible outcomes of a certain variable. It is calculated as:

$$-\sum_i P_X(x_i) \log P_X(x_i) \quad (1)$$

where $I_X(x_i)$ is the self-information associated with particular outcome; I_X is the self-information of the random variable X in general, treated as a new derived random variable; $E[I_X]$ is the expected value of this new random variable, equal to the sum of the self-information of each outcome, weighted by the probability of each outcome occurring. (Pathria 2011). The IS is based on this entropy property. Generated images that show meaningful objects should have a conditional label distribution $p(y|x)$ with low entropy. This is the image quality. The model should also produce images from a varied range of categories, so the marginal $\int p(y|x = G(z))dz$ should have high entropy. The resulting formula for the IS would thus be

$$E_x KL(p(y|x) || (p(y))) \quad (2)$$

(Salimans et al. 2016). In simpler terms, the Inception model will attempt to classify an image into a certain category, with a certain confidence score of each class. The more "unbalanced" this distribution is, the better (that is, the model is very confident that the image pertains to a particular category, and not at all confident that it pertains to other categories). This is the measure of quality, i.e., the image looks like something in particular. The model then performs a sum of the label distributions. This time, the more "balanced" this sum of distributions is, the better (that is, the model contains roughly guesses that encompass many categories). This is the measure of diversity, i.e., the set of images contains objects that belong to many or all modes or categories. The IS therefore captures both image quality and diversity.

5.3 Fréchet Inception Distance

It has been argued that the IS is not suitable for evaluating the product from certain datasets, such as the LSUN Bedroom Dataset (Yu et al. 2015) or the CelebA Dataset. The reason for this is that there is not a defined number of classes or categories in these datasets, and the Inception function knows nothing about the desired distribution of the model (Binkowski et al. 2018).

Thus, another metric was developed to evaluate these datasets: the Fréchet-Inception Distance (henceforth referred to as FID), also known as the Wasserstein-2 distance (Heusel et al. 2017). This score does not evaluate the quality of a particular set of images; instead, it measures a certain distance between two collections of images (the lower the score the better). From the paper, the difference of two Gaussians (synthetic and real images) is measured by the Fréchet distance. The FID is defined by the Fréchet distance $d(., .)$ between the Gaussian with mean (m, C) from $p(.)$ and the Gaussian with mean (m_w, C_w) obtained from $p_w(.,)$, given by:

$$d^2((m, C), (m_w, C_w)) = ||m - m_w||_2^2 + Tr(C + C_w - 2(CC_w)^{1/2}).$$

m and m_w refer to the mean feature vectors of the real and fake images, respectively, while C and C_w are the Covariance matrices for the real and fake feature vectors, respectively. The Tr function performs the sum of the elements along the diagonal of the square matrix (Trace operation). The

FID thus compares the statistics of the produced images to that of real images, instead of evaluating the generated samples on their own, as the IS does.

5.4 Evaluation constraints

Due to physical constraints, which are expanded upon in section 6.1, a completely thorough evaluation of the models could not be performed. Evaluations using the Inception Score and Fréchet-Inception Distance are computationally expensive for large sets of images. Given the necessary restrictions on computational equipment, only image sets of limited size were able to be evaluated. For this reason, some changes to the evaluation procedure had to be made, and thus the evaluation does not aim to compare our model to other existing, state-of-the-art systems in terms of performance, but to study the performance of distinct evolutionary elements, compared to a standard DCGAN. The scores, therefore, should not be taken as a standard measure (the FID Score utilised with the CelebA dataset, for example, reaches considerably lower than the score of state-of-the-art methods, such as DCGAN with Elastic Weight Consolidation (Atkinson et al. 2018)). Instead, the score should be taken strictly as a comparison with a non-evolutionary GAN model.

5.5 Inception Score results

The standard IS model on the available image sets of restricted size was too low to be of any use, for both the standard existing DCGAN and the modified versions - the larger the dataset, the better the score will be, generally. This is an ongoing problem when comparing different GAN models, as there is no standardised size for the evaluated sets among different papers (Jean 2018).

Although the performance of the models on the CIFAR-10 dataset was visually satisfactory, the aforementioned limitation precluded an objective comparison, and thus it was decided that it should be discarded.

5.6 FID score results

The FID results do not reflect real, standardised scores (they are much lower than state-of-the-art systems), but they are consistent among themselves, so they can be used to establish a comparison between a regular DCGAN and a modified, evolutionary GAN.

The CelebA model was evaluated using this FID measure. The results can be summarised as follows: the first few thousand epochs are very erratic and not consistent in their score at all - this can be attributed to the rate of mutation being very high, which necessarily means that many mutations will not be beneficial or will even be harmful for the performance of the model, and the natural selection cannot account for most bad mutations. In the middle of the run, the model starts to stabilise quite rapidly: the rate of mutation decreases slightly, and the model has enough information about changing weights (in the context of the aforementioned Intelligent Design feature) so that future mutations will be more beneficial than harmful. This indeed accelerates the convergence of the model. Toward the end of the convergence curve, the model does not differ greatly from a standard GAN. That is, they are both stabilised, and the rate of mutation of the evolutionary model is now very low, since it is nearing convergence and attempting accelerating this process would be futile. The following figure is a plot of the average FID scores of three Generators of each type of model respectively -standard and mutated-, at different points in the training process.

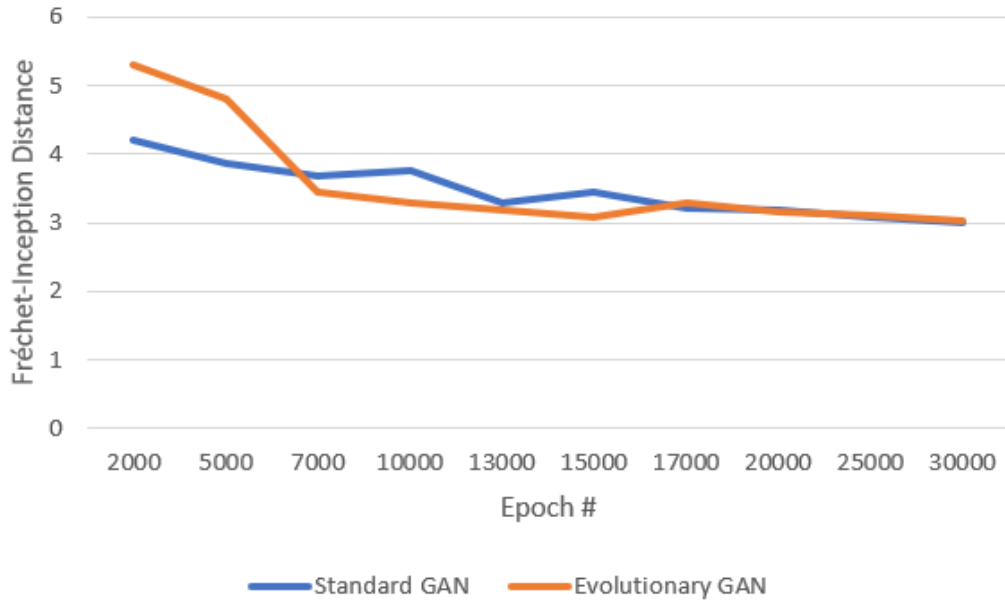


Fig 8 - FID scores comparison for the CelebA dataset (lower is better).

Note: The FID results were obtained using the code from Brownlee (2019).

5.6.1 FID scores and Diminishing Evolution

It would not be feasible to include every single parameter-related evaluation performed; however, the scores obtained by an evolutionary model without Diminishing Evolution and Intelligent Design serves to illustrate the purpose of these features:

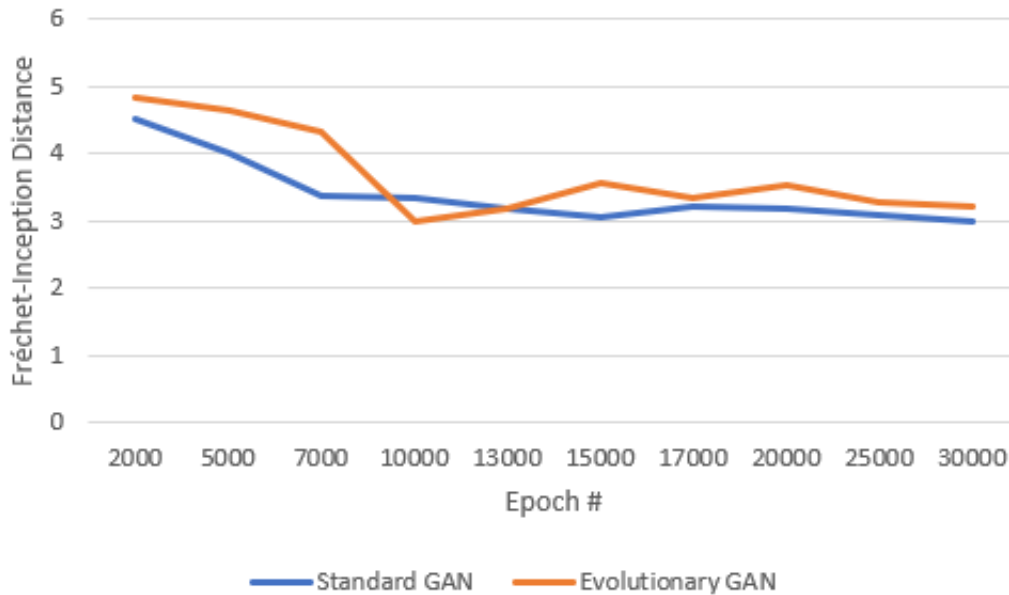


Fig 9 - FID scores comparison for the CelebA dataset, the Evolutionary model does not include Intelligent Design or Diminishing Evolution

The difference is not stark, but it is noticeable that after the middle stages of the training phase, the performance of the model starts to stagnate and even worsen. This is theorised to be because mutations close to convergence do not help performance improvement, but instead hinder it, as explained in section 4.5.1. A more optimal combination of parameters might help to further alleviate this problem.

5.7 Other observations

5.7.1 Collapse recovery

Probably the most interesting result observed from the evolutionary GAN was the ability of one Generator to quickly recover from what would have otherwise been a total mode collapse. Fig 1 in the Appendix shows an example of this, which occurred during an early iteration of the project, with mutations and infrequent crossovers. On the left, a Generator which completely collapses in an early epoch, generating a blank background. On the right, a Generator which follows a standard progression. By virtue of mutations and crossovers, being clearly always selected to die as the worst Generator, it was able to recover from complete collapse into a standard functioning Generator. While rare, these collapses are much more harsh in a standard GAN, where it is much less likely that the Generator recovers from collapse, and even if it does, it would do so much more slowly.

5.7.2 Drawbacks

As mentioned in earlier sections, the clearest disadvantage of introducing random mutations is precisely the stochastic element. That is, mutations can very well hinder the progress of the model, and their randomness makes the model performance much more inconsistent than that of a standard GAN.

Other drawbacks include the difficulty of determining which weights to be mutated (which is partially solved by Intelligent Design), and an added computational load (albeit small) caused by the additional operations of mutating weights and altering the Generator population.

6 Conclusions and future work

There are several key lessons to learn from this study.

6.1 Constraints and elements to improve

Circumstances external to the development of the project, detailed in section 7.1, have precluded a comprehensive parameter tuning, feature exploration and testing. The main shortcoming of the project was a severe lack of parameter exploration - a more thorough parameter tuning would have resulted in much more satisfactory results. Some interesting features were not expanded on nearly enough; certain features even needed to be completely discarded due to time constraints. Some of these features seem extremely promising (see 'Intelligent Design', section 4.5), and it is the author's belief that future research and work could yield very interesting results.

Certain aspects were also left undeveloped due to time and resource constraints, such as a fitness function for the Generators that takes diversity into account, as to reduce mode collapse. Many other GAN variants have developed alternatives with this aim, such as aforementioned Wasserstein-1 distance. It was preferred, however, not to blatantly utilise other people's unchanged work, so if

a novel fitness function couldn't be developed, it was chosen to proceed without one, and simply analyse the differences between a standard GAN and one with evolutionary components.

6.2 Future Work

Much research remains to be done on this subject, as several novel features were found to have a great deal of potential.

Firstly, parameter tuning. As mentioned in the earlier section, a more thorough parameter tuning, not only for the parameters of the general training function (such as the frequency and degree of mutation, and the frequency and mode of crossover and selection), but also for the parameters of specific new features, like Intelligent Design or Diminishing Evolution (such as which weights to conserve, and to which degree, for Intelligent Design; or the method and extent in which the ratio of mutation and selection is progressively reduced, for Diminishing Evolution).

Secondly, a more comprehensive evaluation of the models would be rather necessary. A proper standardised evaluation using the FID, and a functioning version of the IS, for the CelebA and CIFAR datasets, respectively. Another beneficial element would be to test the model on more datasets, such as the LSUN Bedroom or the STL-10 datasets (Coates et al. 2011). This would be to ensure the robustness of the model, such that it is effective in very different scenarios.

Finally, some additional features could be implemented, which would greatly enhance the performance of the models. One of these could be a fitness function that takes into account the diversity of the generated samples, as mentioned in the previous section; or different types of mutations on which to perform parameter tuning, as to find out the most effective one in each case.

7 Project management

This project was undertaken as the Third Year Individual Project (IP) for a BsC in Computer Science for the University of Southampton.

7.1 External issues

The most important event exterior to the project was the outbreak of the Coronavirus COVID-19, which caused a global pandemic and forced many governments to take outstanding action. The United Kingdom and Spain -the author's native country- were amongst the most affected countries. Together with other issues, the closure of the University and its computing labs, the necessary relocation of the author, as well as the forced total lockdown undertaken by the Spanish government, hindered considerably the progress of the project. The sheer amount of computational resources (mainly GPU power) and time needed for each iteration of the project make it completely non-viable to perform a satisfactory number of runs needed for parameter-tuning, or a satisfactory amount of testing, without access to high-power machines.

7.2 Progress

As detailed in the Progress Report, the process of developing the project was iterative: the first steps were figuring out the intricacies of both GANs and Evolutionary Algorithms. Then, creating

a functioning DCGAN for different datasets. And finally, progressively introduce more evolutionary elements and iteratively fine-tune these elements.

7.2.1 GANTT Chart

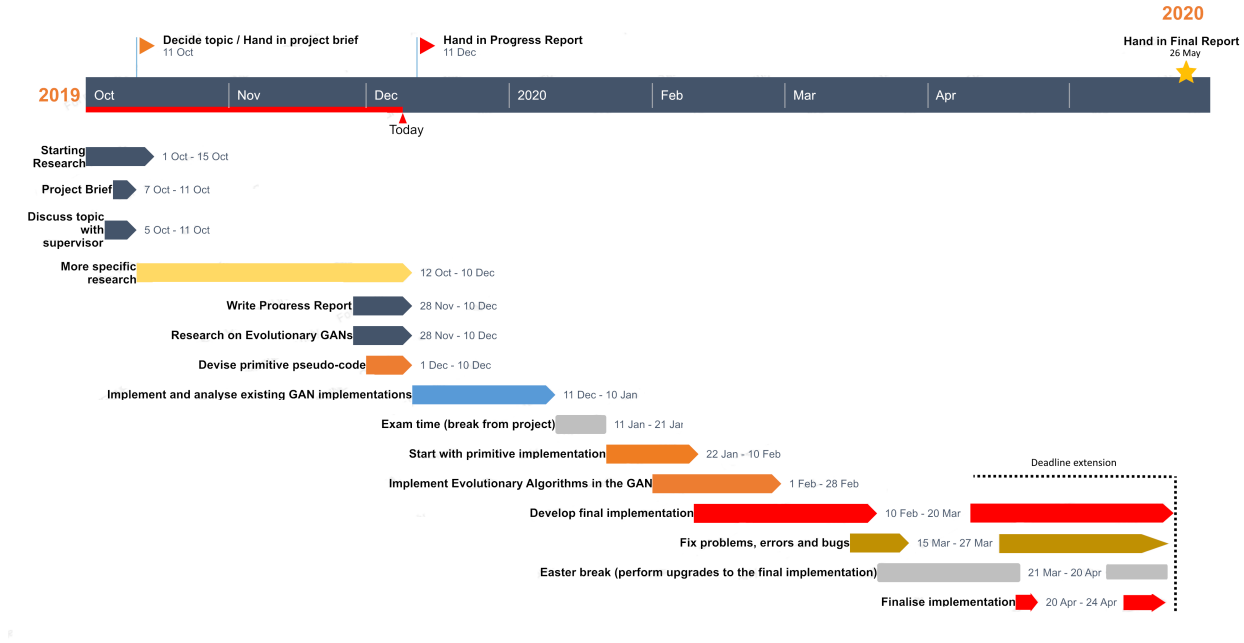


Fig 10 - GANTT progress chart, developed at the beginning of the project.

Everything up to 'Implement Evolutionary Algorithm in the GAN' was achieved in time. The aforementioned external circumstances, however, pushed the final four tasks (developing, improving and finalising the final implementation, and bug fixing) and dragged them significantly. Indeed, a deadline extension was requested and obtained, so the new deadline was set on the 26th of May, and all four tasks were extended until said date. This is reflected in the new GANTT chart. Work previous to the first serious consequences of the aforementioned outbreak was done as outlined in the progress chart. After that, there was a significant overlap between the different tasks, much more than planned, which made progress less efficient and fruitful.

7.2.2 Project Brief

A copy of the Project Brief agreed in October 2019 is attached in the Appendix (Fig 4). The goal has indeed been realised within the established scope, insofar as the development of a system which combines Generative Adversarial Networks and Evolutionary Algorithms, and the analysis of its advantages and drawbacks.

Appendix

Images and figures

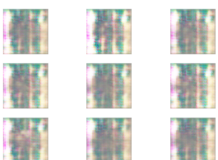
	Collapsed Generator	Non-collapsed Generator
Epoch #400		
Epoch #800		
Epoch #1200		
Epoch #1400		
Epoch #1600		
Epoch #1800		
Epoch #2500		

Fig 1: Collapse recovery

GENERATOR				
Operation	Kernel	Strides	Feature maps	Activation
Input			100	
Dense			256 x 4 x 4	Leaky ReLU
Transposed Convolution	4x4	2x2	128	Leaky ReLU
Transposed Convolution	4x4	2x2	128	Leaky ReLU
Transposed Convolution	4x4	2x2	128	Leaky ReLU
Convolution (Output)	3x3		3	tanh

Fig 2: Structure of the Generator model

DISCRIMINATOR				
Operation	Kernel	Strides	Feature maps	Activation
Input	3x3		32x32x3	
Convolution	3x3	2x2	128	Leaky ReLU
Convolution	3x3	2x2	128	Leaky ReLU
Convolution	3x3	2x2	256	Leaky ReLU
Dense (Output)			1	Sigmoid

Fig 3: Structure of the Discriminator model

Project Brief

Evolutionary training in Generative Adversarial Networks

By Santiago Duque

Supervised by Dr Srinandan Dasmahapatra

October 2019

1. Problem

Ensemble methods in Machine Learning have been shown to be very powerful for classification and regression problems. In addition, Generative Adversarial Networks (GANs) are powerful tools consisting of two “competing” neural networks that progressively improve at some adversarial game; for unsupervised and supervised learning. There have been attempts and models that fuse both concepts (Ensemble learning and GANs).

2. Goal

The goal of this project is to devise a novel system of training and improving GANs, which would consist of a “tournament” or evolution-like “game”, where multiple generators send their product to multiple discriminators who make a decision, upon which the weighting of each generator will be decided, and such its probability to pass on its traits to the next iteration, comparable to genetic algorithms.

3. Scope

The project will use TensorFlow and various Python and Machine Learning / Data Science libraries. It will start out with simple networks and problems, to build the structure of the system, and then progressively make it more complex, and then analyse the results and the viability of the system as a whole.

Fig 4: Project Brief

Archive table of contents and execution instructions

The code consists of two main folders - one for the files as .ipynb Notebooks, and one for the .py Python files. These files are the two main models, one for CIFAR-10 and one for CelebA.

To run the main models, the desired parameters can be set near the top of the file, before the main class and after the imports and dependencies.

The framework requirements are listed in the requirements.txt file. For most frameworks, simply doing, install requirements.txt will work - (for example, !pip install requirements.txt). Most requirements are redundant, but it was preferred to err on the side of caution. If a particular requirement is not included, for some reason, it will not necessitate more than a simple install in the particular framework of the user.

Word count

Word count: 6,168, using <https://wordcounter.net>

References

- Al-Dujaili, A., Schmiedlechner, T., Hemberg, E. & O'Reilly, U.-M. (2018), 'Towards distributed coevolutionary gans'.
- Arjovsky, M. & Bottou, L. (2017), 'Towards principled methods for training generative adversarial networks'.
- Arjovsky, M., Chintala, S. & Bottou, L. (2017), 'Wasserstein gan'.
- Atkinson, C., McCane, B., Szymanski, L. & Robins, A. V. (2018), 'Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks', *CoRR* **abs/1802.03875**.
URL: <http://arxiv.org/abs/1802.03875>
- Binkowski, M., Sutherland, D. J., Arbel, M. & Gretton, A. (2018), 'Demystifying mmd gans', *ArXiv* **abs/1801.01401**.
- Brownlee, J. (2019), 'How to implement the frechet inception distance (fid) for evaluating gans'.
URL: <https://web.archive.org/web/20200526050652/https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/>
- Buurman, J., Zhang, S. & Babovic, V. (2009), 'Reducing risk through real options in systems design: The case of architecting a maritime domain protection system', *Risk analysis : an official publication of the Society for Risk Analysis* **29**, 366–79.
- Coates, A., Ng, A. & Lee, H. (2011), An analysis of single-layer networks in unsupervised feature learning, in 'Proceedings of the fourteenth international conference on artificial intelligence and statistics', pp. 215–223.
- dos Santos-Paulino, A. C., Nebel, J.-C. & Flórez-Revuelta, F. (2014), 'Evolutionary algorithm for dense pixel matching in presence of distortions'.
- Ghosh, A., Kulharia, V., Namboodiri, V. P., Torr, P. H. S. & Dokania, P. K. (2017), 'Multi-agent diverse generative adversarial networks', *CoRR* **abs/1704.02906**.
URL: <http://arxiv.org/abs/1704.02906>
- Goodfellow, I. (2016), Nips 2016 tutorial: Generative adversarial networks, OpenAI.
- Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014), 'Generative adversarial nets'.
- Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G. & Hochreiter, S. (2017), 'Gans trained by a two time-scale update rule converge to a nash equilibrium', *CoRR* **abs/1706.08500**.
URL: <http://arxiv.org/abs/1706.08500>
- Ioffe, S. & Szegedy, C. (2016), 'Batch normalization: Accelerating deep network training by reducing internal covariate shift'.
- Jean, N. (2018), 'Fréchet inception distance'.
URL: <https://web.archive.org/web/20190608135524/https://nealjean.com/ml/frechet-inception-distance/>

- Karras, T., Laine, S. & Aila, T. (2019), ‘A style-based generator architecture for generative adversarial networks’.
- Kingma, D. P. & Ba, J. L. (2015), Adam: A method for stochastic optimization, ICLR.
- Krizhevsky, A. (2009), Learning multiple layers of features from tiny images, Technical report.
- LeCun, Y., Cortes, C. & Burges, C. J. (n.d.), ‘The minst database of handwritten digits’.
URL: <http://yann.lecun.com/exdb/mnist/>
- Li, Y., Ang, K. H., Chong, G. C. Y., Feng, W., Tan, K. C. & Kashiwagi, H. (2004), ‘Cautocsd - evolutionary search and optimisation enabled computer automated control system design’,
<https://doi.org/10.1007/s11633-004-0076-8>.
- Liu, Z., Luo, P., Wang, X. & Tang, X. (2015), Deep learning face attributes in the wild, *in* ‘Proceedings of International Conference on Computer Vision (ICCV)’.
- Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z. & Smolley, S. P. (2017), ‘Least squares generative adversarial networks’.
- Metz, L., Poole, B., Pfa, D. & Sohl-Dickstein, J. (2017), ‘Unrolled generative adversarial networks’.
- Mitchell, M. (1996), *An Introduction to Genetic Algorithms*, MIT Press.
- Mordvintsev, A., Olah, C. & Tyka, M. (2015), ‘Inceptionism: Going deeper into neural networks’.
URL: <https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>
- Ni, Y., Song, D., Zhang, X., Wu, H. & Liao, L. (2018), ‘Cagan: Consistent adversarial training enhanced gans’.
- Pathria, R. K. (2011), *Statistical mechanics*, Academic Press, Boston.
- Popovici, E., Bucci, A., Wiegand, R. P. & de Jong, E. D. (2018), ‘Coevolutionary principles’.
- Radford, A., Metz, L. & Chintala, S. (2016), ‘Unsupervised representation learning with deep convolutional generative adversarial networks’.
- Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A. & Chen, X. (2016), ‘Improved techniques for training gans’, *CoRR* **abs/1606.03498**.
URL: <http://arxiv.org/abs/1606.03498>
- Schawinski, K., Zhang, C., Zhang, H., Fowler, L. & Santhanam, G. K. (2017), ‘Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit’.
- Springenberg, J. T., Dosovitskiy, A., Brox, T. & Riedmiller, M. (2015), ‘Striving for simplicity: The all convolutional net’.
- Srivastava, A., Valkov, L., Russell, C., Gutmann, M. U. & Sutton, C. (2017), ‘Veegan: Reducing mode collapse in gans using implicit variational learning’.
- Tolstikhin, I. O., Gelly, S., Bousquet, O., Simon-Gabriel, C.-J. & Schölkopf, B. (2017), Adagan: Boosting generative models, *in* ‘NIPS’.

- Toutouh, J., Hemberg, E. & O'Reilly, U. (2019), 'Spatial evolutionary generative adversarial networks', *CoRR* **abs/1905.12702**.
URL: <http://arxiv.org/abs/1905.12702>
- Toutouh, J., Hemberg, E. & O'Reilly, U.-M. (2018), 'Lipizzaner: A system that scales robust generative adversarial network training'.
- Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N. & Rei, W. (2012), 'A hybrid genetic algorithm for multidepot and periodic vehicle routing problems', <https://doi.org/10.1287/opre.1120.1048>.
- Vikhar, P. A. (2016), 'Evolutionary algorithms: A critical review and its future prospects'.
- Wang, C., Xu, C., Yao, X. & Tao, D. (2018), 'Evolutionary generative adversarial networks', *arXiv preprint arXiv:1803.00657*.
- Yu, F., Zhang, Y., Song, S., Seff, A. & Xiao, J. (2015), 'LSUN: construction of a large-scale image dataset using deep learning with humans in the loop', *CoRR* **abs/1506.03365**.
URL: <http://arxiv.org/abs/1506.03365>
- Zhu, J.-Y., Park, T., Isola, P. & Efros, A. A. (2018), 'Unpaired image-to-image translation using cycle-consistent adversarial networks'.