Evolutionary Generative Adversarial Networks

Andrej Ivanov

Third Year Project
B.Sc. in Computer Science
The University of Manchester, School of Computer Science
Supervisor: Tingting Mu

April 2019

Abstract

In recent years, Generative Adversarial Networks (GANs) have been the state-of-the-art generative models for a number of image-generation-related tasks. Despite their success, GANs suffer from training instability and are notoriously hard to train. My work extends a recently proposed modification of the GAN model, called the Evolutionary GAN (E-GAN).

The paper starts off with an intuitive introduction to GANs and evolutionary techniques, followed by a description of the evolutionary GAN method. Then, I present a detailed Tensorflow implementation of the model and highlight a number of optimizations. Furthermore, I compare the performance of my model with respect to the performance of the current most popular modification of GAN.

The paper shows that the E-GAN is able to produce distinguishable samples much faster and with a simpler architecture for the discriminator and the generator, but after a point destabilizes and starts producing worse-looking images. In addition to presenting the comparison results, I also aim to provide an analysis on the nature of the advantages and disadvantages of E-GAN. Finally, I conclude the report with suggestions for future research on the evolutionary model.

Contents

List of Figures

List of Algorithms

List of Abbreviations

X	vector of real numbers
$E_{X \sim P}$	Expected value of a random variable X with respect to P
$log(x)$	Natural logarithm of x
P_{real}	the data probability distribution in the context of generative modelling
P_{model}	the distribution learned by the model in the context of generative modelling
P_z	noise distribution in the context of GANs
z	a noise vector sampled from P_z
$D(x)$	The function representing the discriminator in a GAN
$G(x)$	The function representing the generator in a GAN
$f(x) \rightarrow y$	as the value of the function $f(x)$ gets close to y
$D_{KL}(p \parallel q)$	KL divergence between the distributions p and q
GAN	Generative Adversarial Network
EGAN	Evolutionary Generative Adversarial Network
DCGAN	Deep Convolutional Generative Adversarial Network
WGAN	Wasserstein Generative Adversarial Network
EA	Evolutionary algorithms

Chapter 1

Background

This chapter provides the background knowledge necessary to understand the more intricate methods explained in later chapters. It introduces generative modelling in machine learning, and explains how generative adversarial networks are used to tackle this problem. Furthermore, the chapter provides a brief introduction to evolutionary algorithms.

1.1 Generative Modelling

In order to understand GANs and E-GAN in particular, one must first be comfortable with the more general problem these methods aim to solve, that is, generative modelling. The first section of this chapter provides a gentle introduction to the problem of generative modelling, and the second one establishes a more robust, theoretical description of the problem. The last section gives examples of actual use cases of generative modelling.

1.1.1 What is generative modelling ?

The problem E-GAN is trying to solve is that of generative modelling. In essence, generative modelling is a collection of methods which take a dataset of samples describing some real world phenomenon, and attempt to generate new samples from the phenomenon. For instance, given a dataset of pictures

of cats, the task of a generative model would be to learn to generate new pictures of cats.

To truly comprehend the nature of the difficulty of this problem, it is crucial to grasp the subtle difference between discriminative and generative models in the context of machine learning. While a discriminative task only requires learning the difference between real world classes, generative modelling is much more involved, as it is necessary to build a fundamental representation of each class in order to generate samples from it. To clarify this idea, consider a dataset of samples from cats and dogs. A discriminative model only outlines the difference between cats and dogs (for example, noticing that dogs are generally larger than cats) and does not bother with their individual properties. On the other hand, generative modelling is concerned with representing cats and dogs individually (for example, observing that cats have four legs, have fur and are able to moew).

1.1.2 Common approaches

Mathematically, the problem is defined as follows: given a dataset of data points $X = x^{(1)}, ..., x^{(n)}$ (where each $x^{(i)}$ is a vector) sampled from a data distribution P_{real}, we need to somehow model a distribution P_{model} approximating P_{real}. The best we can do is approximate P_{real} because P_{real} is the unknown underlying distribution of the phenomenon described by the dataset. In the previous example with the cats, P_{real} would be the underlying distribution of all cat pictures in the world (obviously unknown), and each data point would be the pixel representation of an image of a cat.

The main dichotomy in generative models is: *explicit* models that try to estimate P_{model} explicitly, and *implicit* models that only learn to sample from P_{model} without learning the distribution explicitly. The most popular examples of the former are Fully visible belief networks [5] and Variational encoders [10], and the most popular implicit model is the Generative adversarial network, which is the area of interest for this project.

1.1.3 Importance

Although representing and manipulating high dimensional probability distributions provides profound theoretical insights for numerous natural science and engineering fields, it is not entirely obvious why the ability of generating samples (and especially images) wouble be useful in real life. To convince the reader that this problem is worth exploring in depth, here are some applications of generative modelling which have found great success in practice in recent years:

- *Image Super-Resolution*: up-scaling low resolution images to high resolution images. [13]
- *Style transfer*: transforming images from a collection X to the "style" of images of collection Y. Use cases include converting pictures of nature in the style of famous artists, or converting aerial satellite images into maps. [26]
- *Text-to-image synthesis*: given a sentence, generating an image described by the sentence. [25]

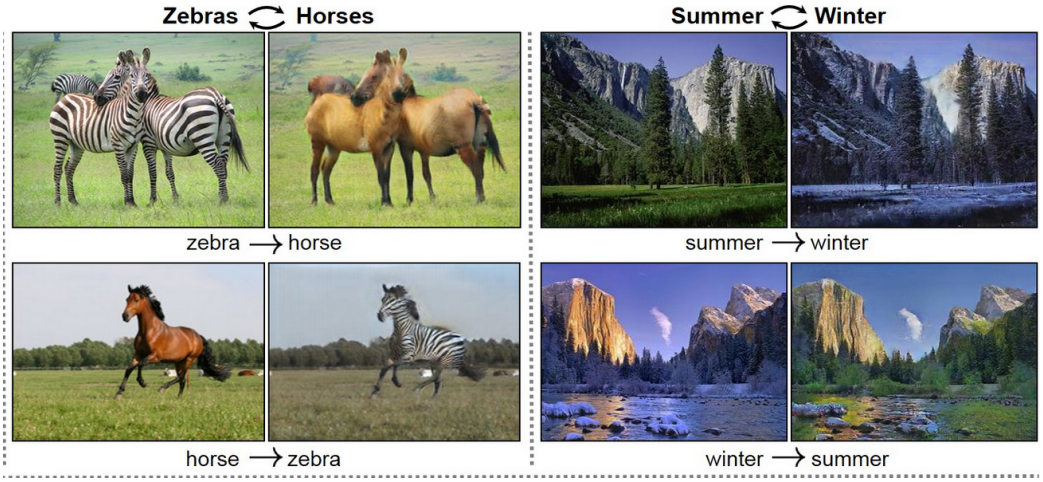


Figure 1.1: Results from style transfer

Example of using generative modelling to perform style transfer between images. On the left, the pattern of a zebra is transferred to a horse, and vice versa. On the right, an image of a landscape in the summer is transformed to its equivalent in winter, and vice versa. Image taken from the Cycle GAN paper [26].

Having covered generative modelling in general, from the next section the paper will focus on GANs as a generative model. Furthermore, for the purposes of my work, I will restrict the problem of generating abstract samples to generating images specifically.

1.2 Generative Adversarial Networks

Generative adversarial networks are one of the main generative models that can learn complex real world data distributions. They perform better than other approaches to generative modelling, and consequently GANs have been the state-of-the-art generative models for numerous image-generation-related tasks. In a similar fashion to the previous section, I first provide a high-level explanation of a GAN, and then I describe the more involved theoretical properties of the approach.

1.2.1 How do GANs work?

A GAN consists of two neural networks: a generator and a discriminator. The goal of the generator is to learn to generate samples from the training data distribution, and the goal of the discriminator is to learn to distinguish between real samples from the dataset and fake samples from the generator. The two networks play an adversarial game where the generator aims to trick the discriminator into thinking its generated samples are actually real. In each iteration, the discriminator receives real images from the training dataset and fake images from the generator, and is punished for not distinguishing the fake images. Conversely, the generator is punished if the discriminator does manage to separate out the fake images. This way, the generator and discriminator iteratively improve and learn from each other. The training ends when the discriminator can no longer distinguish between generated and real samples.

1.2.2 Theoretical description of GANs

Mathematically, as neural networks, the discriminator and generator represent parameterized functions D and G, respectively. The input to G is a noise vector z, and the output is $G(z)$ - a sample similar to the samples from the training data. The input to D is a sample x (either generated by G or real from the training dataset), and the output is $D(x)$ - a score between 0 and 1 representing the discriminator's confidence that the sample x is real.

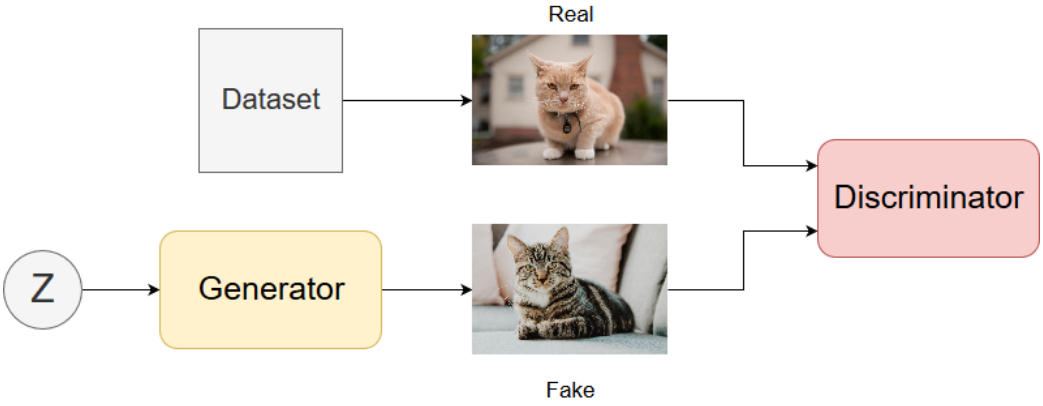


Figure 1.2: Architecture of a GAN

In each training step, we sample a batch of noise z from P_z and a batch of data samples x from P_{real}, and then update the discriminator and the generator according to their loss functions:

- Intuitively, the discriminator aims to assign high scores to real samples and low scores to generated samples. Theoretically, this means that D seeks to to minimize $D(G(z))$ and maximize $D(x)$. Hence, the loss function of D is:

$$J^{(D)} = -\frac{1}{2}E_{x\sim p_{data}}log(D(x)) - \frac{1}{2}E_{z\sim p_z}log(1 - D((G(z))). \quad (1.1)$$

- By similar reasoning, the generator aims to maximize $D(G(z))$ and the loss function of G is:

$$J^{(G)} = \frac{1}{2} E_{z \sim p_z} log(1 - D((G(z))) \tag{1.2}$$

In future sections, I will use the terms training adversarial objective and loss function interchangeably - they refer to the same concept. For the aforementioned adversarial objective (called the *minimax* loss), it can be theoretically proven that the training minimizes the KL-divergence between p_{data} and p_{model}, denoted by $D_{KL}(p_{data}||p_{model})$, which has a unique minimum at $p_{model} = p_{data}$ [1]. This property is a theoretical guarantee that with enough training data, G will eventually learn p_{real}. However, in practice this loss functions performs very poorly, which is further explained in the next section.

1.2.3 Problems with training of GANs

Despite their success, GANs are notoriously hard to train. For example, if p_{data} and p_{model} only have a small overlap (for example at the beginning of training when the generator produces random images and p_{data} and p_{model} are completely different) the KL-divergence distance measure between the two distributions is undefined, and consequently, the generator does not receive any useful feedback from the loss function. Most of recent GAN research focuses on proposing novel adversarial objective functions, which aim to minimize different probability distance measures between p_{data} and p_{model}, but all of them have their own drawbacks. These issues give rise to the following instabilities during GAN training:

- *Divergence*, the parameters of the model destabilize, and the generator and discriminator keep getting worse.
- *Mode collapse*, the generator assigns all its probability mass at a small region. Results in the generator outputting the same result for any input.
- *Discriminator too powerful*, if the discriminator becomes significantly stronger than the generator, it rejects all of the generator's samples with high confidence, and the generator does not receive useful feedback.

- *Hyper-parameter tuning*, GAN training requires a significant amount of heuristic tricks and hyper-parameter tuning to make work.

This concludes the overview of generative modelling and GANs in particular. The next chapter presents the basics of evolutionary algorithms, and is not directly connected to GANs.

1.3 Evolutionary Algorithms

The Evolutionary GAN is unique in its use of evolutionary techniques to help improve GAN training. Because of this, it is important for the reader to understand the basics of evolutionary algorithms.

1.3.1 Motivation

Evolutionary algorithms (EAs) are a collection of population-based optimization methods inspired by biological evolution. Much like in nature animals create offspring and the fittest offspring continue to the next generation, in evolutionary algorithms, we start with a solution in the search space, create possible (children) solutions, and the most optimal (fittest) solutions proceed to the next generation. A parallel can be drawn between the process of biological species evolving to the optimal state in their environment and solutions in a search space evolving towards the optimum in the space. The most popular classes of EAs are: genetic algorithms, evolution strategies and differential evolution. For the purposes of the method presented in this paper, we only need to focus on genetic algorithms.

1.3.2 Genetic algorithms

Genetic algorithms are perhaps the simplest form of EAs. A genetic algorithm can be divided into 3 main stages, performed iteratively:

- *Initialization*: we initialize a starting generation of parents. From a biological view, each parent is described by a number of genes. From

a mathematical view, we consider that the parent is described by a vector of features.

- *Crossover and mutation*: in this stage, we perform crossover to some of the parents to obtain children, and apply mutations to some of the children. Mathematically, we create children by combining the vectors of features (genes) of parents.
- *Selection*: Finally, we use a fitness function to evaluate how well each child performs in the environment. The fittest children proceed to the next generation.

1.3.3 Gradient descent and evolution

To bridge the gap between evolutionary algorithms and GANs, it is critical to grasp the subtlety of how EAs relate to gradient descent (the conventional way of optimizing GANs). Although EAs and gradient descent, both being optimization methods, share the common goal of finding an optimum in a search space, they reach the goal in two very different ways. Gradient descent takes single small steps towards the optimum (from a local point of view). On the other hand, EAs take multiple steps in different directions, and then choose the most optimal one.

By this point I have fully covered the background knowledge necessary to understand the E-GAN. The next chapter starts explaining the evolutionary approach.

Chapter 2

Introduction

This chapter first describes the evolutionary GAN (E-GAN) model in detail, as proposed in [24], and then lists my objectives on how to work on top of it. Although I incorporate my own intuition and interpretations, the basics of the method described in this chapter are completely developed by the aforementioned paper.

2.1 EGAN

In this section, I first emphasize the particular weaknesses of the original GAN model which the E-GAN is aiming to improve, and then continue with a description of the evolutionary architecture which implements these improvements.

2.1.1 Motivation

Despite their success at producing sharp and visually appealing images, GANs still suffer from a range of training problems (see section 1.2.3). Employing evolutionary techniques aims to improve problems inherent in existing GANs in two main aspects:

- Most recent work on stabilizing training of GANs focuses on developing new adversarial training objectives for the generator. However, each of the popular training objectives suggested so far has its own downsides (see chapter 3). Evolutionary techniques will allow us to combine multiple training objectives and choose the best one (will be explained in detail in the next section).
- GANs often suffer from mode collapse. With evolutionary techniques we can make sure we select the children with higher diversity of generated samples, thus overcoming the issue of mode collapse.

The next section describes the model inspired by these observations.

2.1.2 The evolutionary model

The main difference between current popular models and our model is that E-GAN utilizes a generation of parent generators instead of a single generator. The principle idea is that each of the parent generators creates children, then each of the children is evaluated according to a fitness function, and finally the fittest children proceed to be the parents in the next iteration (generation). More precisely, a generator parent creates children by updating its weights with regards to different adversarial objectives. This process is called mutation and the different adversarial objectives are hence called mutations. The details of the choice of mutations and fitness functions are presented in the next two sections.

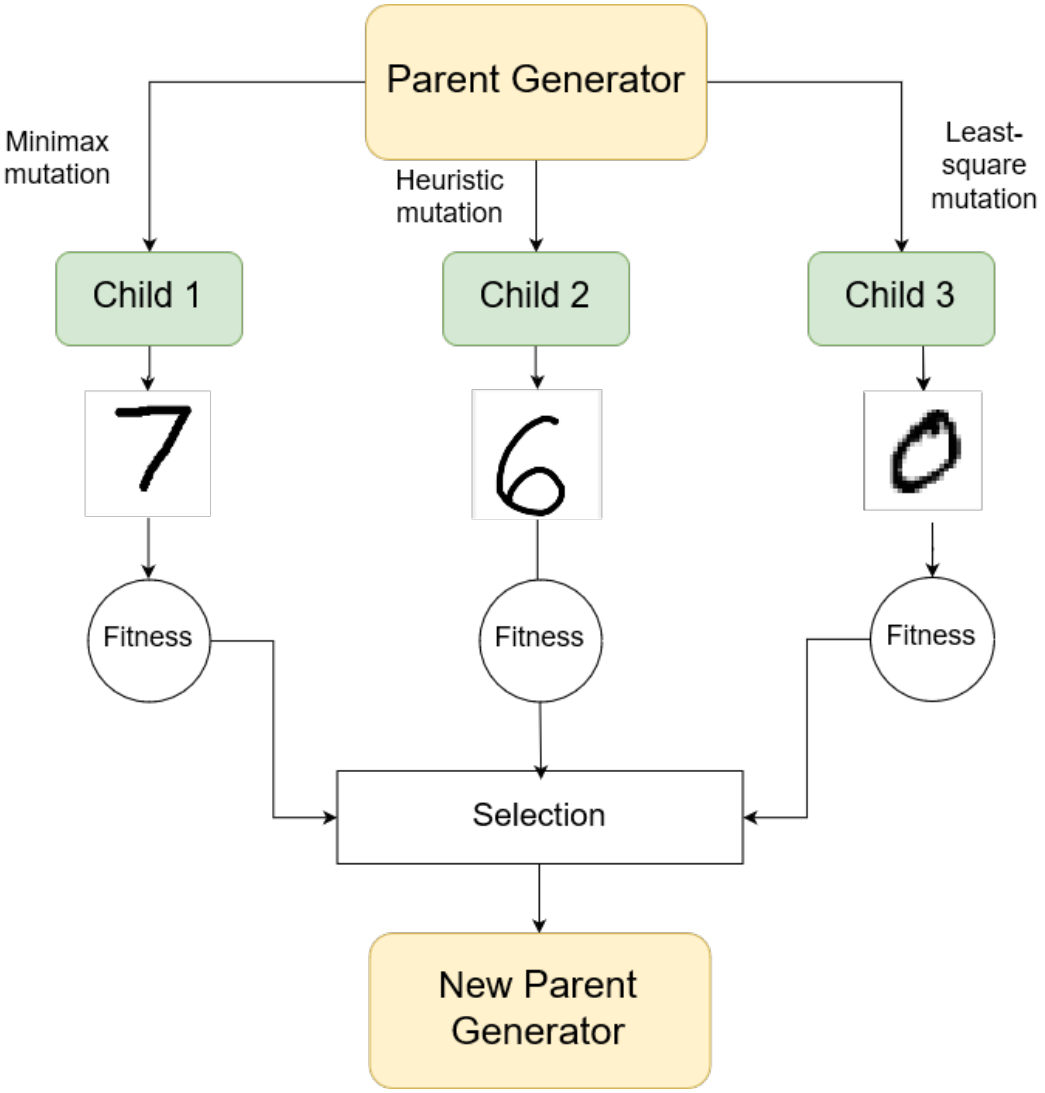


Figure 2.1: The E-GAN architecture

The figure illustrates a single evolutionary training step in a simplified E-GAN. In this example, the model has a single parent generator and three mutations: minimax, heuristic and least-square mutation. Each of the children produces samples (handwritten digits in this case), and is evaluated on its generated samples. In the last stage, we select the best performing offspring to be the parent in the next generation.

2.1.3 Mutations

The mutations lie at the heart of what makes E-GAN perform well. Each mutation corresponds to a different adversarial training objective - the point of combining multiple mutations is to overcome their individual weaknesses. Here are the three most common mutations used in GANs:

- **Minimax mutation**: intuitively minimizes the log-probability of the discriminator being correct. A major disadvantage of this mutation is that it saturates as $D(G(z)) \to 0$. In other words, when the discriminator distinguishes the generator's samples with high confidence, the generator's gradient vanishes [6].

$$M_{minimax} = \frac{1}{2} E_{z \sim p_z} log(1 - D(G(z))) \tag{2.1}$$

- **Heuristic mutation**: intuitively maximizes the log-probability of the discriminator being "wrong". In practise, this mutation exhibits inverse properties to those of the minimax mutation: it has unrestricted gradient flow when $D(G(z)) \to 0$ (discriminator easily recognizes the fake samples), but saturates when $D(G(z)) \to 1$ (discriminator has trouble differentiating fake and real samples).

$$M_{heuristic} = -\frac{1}{2} E_{z \sim p_z} log(D(G(z))) \tag{2.2}$$

- **Least-squares mutation**: this mutation is similar to the heuristic mutation. The most notable difference is that is assigns smaller gradients than the heuristic mutation both when the discriminator is at an advantage, and when the generator is at an advantage.

$$M_{least} = \frac{1}{2} E_{z \sim p_z} [D(G(z)) - 1]^2 \tag{2.3}$$

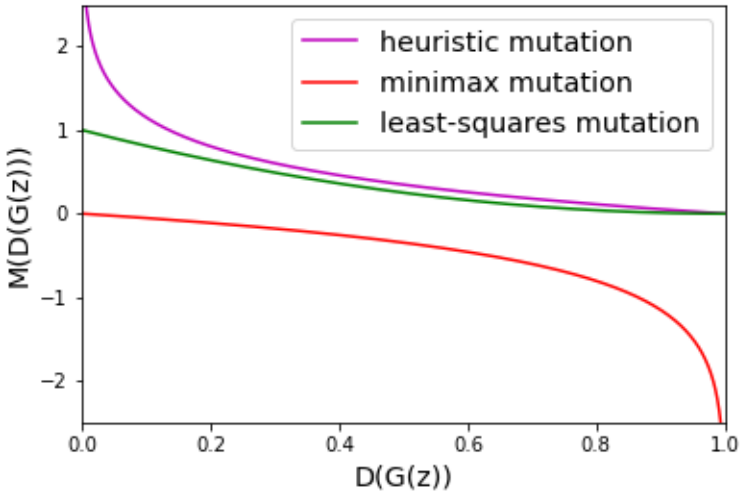


Figure 2.2: The values of the minimax, heuristic and least-squares mutations for varying discriminator scores

The minimax mutation if very flat and thus receives vanishing gradients as $D(G(z))$ gets close to 0, but drops quickly (and thus receives large gradients) when $D(G(z))$ gets close to 1. The heuristic mutation and the least-squares mutation exhibit the opposite property: they are flat as $D(G(z))$ approaches 1 and rise freely when $(D(G(z))$ approaches 0.

2.1.4 Fitness

The fitness score of a child is a measure F given by the discriminator that expresses how well the child is performing. In this sense, the generators evolve in an environment regulated by the discriminator. Through the fitness score the discriminator evaluates how good the child's generated samples are, F_q, and how diverse they are, F_d. More precisely, $F = F_q + \gamma F_d$, where γ is the diversity factor. To calculate F, we first generate a batch of images from the child. Then:

- The quality score is simply the average score the discriminator assigns to the generated samples (remember - the discriminator assigns high scores to realistic-looking images). Mathematically:

$$F_q = E_{z \sim p_z} log(D(G(z))) \tag{2.4}$$

- Estimating the diversity score of a child is more involved. A recent theoretical analysis on the stability of GANs observed that mode collapse is accompanied with the discriminator rejecting the collapsed region with high probability (high gradients). In simpler terms, if a generator gets stuck on one solution, the discriminator quickly learns to reject it with high confidence, and training stops. Hence, to endorse diversity in the generator, we want to minimize the size of the discriminator's gradients:

$$F_d = -log||\frac{\partial}{\partial D}[-\frac{1}{2}E_{x\sim p_{data}}log(D(x)) - \frac{1}{2}E_{z\sim p_z}log(1 - D((G(z)))]|| \tag{2.5}$$

2.1.5 Selection

Finally, the fittest children move on to be the parents in the next iteration. This evaluation and selection process guarantees that with every iteration, we are getting better generators both in terms of quality and diversity of their generated samples.

In the last few sections, I described the evolutionary process of my model from the perspective of GAN training only. Because of this, it might not be immediately obvious to the reader how the E-GAN relates to conventional evolutionary algorithms. The next section aims to clarify the E-GAN as an evolutionary algorithm.

2.1.6 E-GAN as an evolutionary algorithm

One can think of the E-GAN as a "degenerate" case of a genetic algorithm. My approach uses the genetic algorithm as a platform that allows it to combine multiple adversarial objectives and enforce diversity among the generated samples. Because of this, the E-GAN does not make use of the *crossover* part of the genetic algorithm, and instead only performs mutations to obtain children. In fact, in later chapters I show that the model favors mostly from only one parent, a configuration where crossover is not even possible.

With this remark I conclude my description of the E-GAN model as proposed in the paper. In the next section I concisely define my goals for the

project.

2.2 Aims and objectives

What was explained earlier is a relatively novel approach to using both GANs and evolutionary techniques. Furthermore, the original paper only provides high-level description of the method, does not supply an implementation and does not have any additional work on top of it as of the time of this report. As such, the E-GAN has a lot of room for research and experimentation. This sections defines my plan on how to extend the work on E-GANs.

2.2.1 List of objectives

I have three central objectives for this project:

- Gain a firm understanding on the fundamental problems of training GANs, and how evolutionary techniques can be employed to overcome these problems.
- Implement a DCGAN (the current most popular modification of GAN) and an E-GAN. Optimize and fine-tune the details of the E-GAN as much as possible.
- Compare the performance of E-GAN against the performance of DCGAN in terms of training stability, training speed and quality of generated samples.

2.2.2 Evaluation

I evaluated my modification to the E-GAN on the benchmark datasets MNIST (handwriten digits) and FashionMNIST (hand-drawn fashion items). All my comparisons introduced in later chapters are against DCGAN.

Chapter 3

Related works

The following chapter analyzes the place of our model in the vast array of modifications of GANs proposed in the past few years. The first two discussed works, DCGAN [17] and WGAN [2] are the most common approaches that share the same goal as the E-GAN, but aim to achieve it in completely different ways. The third section compares our model to other less popular, but more similar models.

3.1 Deep Convolutional GAN

As explained in the previous chapter, the original minimax GAN setup fails to converge for most non-trivial generative problems, and mostly serves as the basis for theoretical analysis of GANs. As a response to this predicament, the most common (and "default") current GAN architecture is DCGAN. This GAN architecture proposes a number of heurstic-based modifications to stabilize GAN training, such as batch normalization in both the generator and discriminator and using a heuristic loss function instead of the minimax loss for the generator. Although this set of tricks provides good results in practice, DCGAN still relies on careful architecture design and hyper-parameter tuning unique for each problem. In contrast, our model aims to generalize training stability.

3.2 Wasserstein GAN

The goal of the WGAN is to minimize the Wasserstein distance (also known as the Earth Mover's distance) between the model distribution and the real distribution. The Wasserstein distance is favorable because it exhibits superior theoretical properties over the KL distance minimized by the minimax approach. In particular, the advantage of the aforementioned distance is that it is defined and smooth everywhere, even if P_{model} and P_{real} do not substantially overlap. The WGAN yields a more generic solution to GAN training stability than DCGAN, but it still has the potential to suffer from vanishing or exploding gradients towards equilibrium [8].

3.3 Multi-generator GANs

This section presents a list of works on stabilizing GANs similar to the E-GAN. More specifically, these approaches employ a multiple generator architecture, but differ from E-GAN significantly as they do not use evolutionary techniques or multiple adversarial training objectives, which are at the core of my model. The recently proposed Mixture GAN [9] focuses on tackling the mode collapse problem by enforcing a mixture of generators to cover a wider range of data modes. Some approaches, such as the Dual Generator GAN [23] and the Coupled GAN [14] use a pair of generators to introduce advancements in the context of image-to-image translation in particular. Slightly differing from the other works in this section in its architecutre, the Generative Multi-Adversarial Network [4] operates with a varying number of discriminators to balance the power gap between the generator and the discriminator.

Chapter 4

Implementation

The following chapter explains how I realized the ideas proposed in the original E-GAN paper. Note that the details of my implementation and the lower-level optimizations go beyond what is outlined in the previous work. The first section covers the technology used in the project, and the next three sections delve into the intricacies of my implementation.

4.1 Technologies

The whole framework is written in Python with the help of the deep learning library Tensorflow. Training of models was performed both locally on a CPU and on a GPU on Google Cloud. The subsequent subsections cover the reasoning behind the choice of these technologies.

4.1.1 Python

Python is an interpreted programming language which enables users to develop solutions quickly and focus on the core of the problem at hand, unobscured by software engineering paradigms such as types. This makes it particularly suitable for machine learning tasks, and especially my project, where the goal was not to develop a fixed robust software solution, but rather analyze and test numerous modifications of a model. Even though this ease

of use makes Python slower than most other programming languages, it is compatible with a number of popular libraries for data handling, data visualisation and deep learning, which "under the hood" use faster compiled languages like C++.

4.1.2 Tensorflow

Tensorflow is my deep learning framework of choice primarily because it has a significantly larger community than its contenders and thus has the benefit of more resources and tutorials. The second reason I find this library superior is *Tensorboard*: an extension which allowed me to monitor and visualize the progress of my models during training, such as the loss of the discriminator and the generators, or the fitness of the children generators throughout iterations. Additionaly, Tensorflow is developed by Google, and accordingly enjoys compatibility with other products from Google, such as the Google Cloud API.

4.1.3 Google Cloud

The Google Cloud platform provides a wide range of services, but I used two in particular:

- *Google Cloud ML Engine*: Tensorflow's operations are exceptionally fast on a GPU, and this engine allowed me to train my models on an NVIDIA Tesla K80 GPU on the cloud, instead of my laptop's CPU. Another advantage of the engine is that it allowed me to run multiple jobs simultaneously, which was extremely useful when I compared different approaches, or different values for a hyper-parameter (such as the diversity factor in the fitness method).
- *Google Cloud Storage*: I mainly used this storage to save my trained models, and persist all the generated samples and graphs from my experiments for analysis in the future.

4.2 The framework

The main focus implementation-wise was flexibility in terms of different training options. As a result, I developed a general framework for training of GANs, where I could easily choose between an E-GAN and a regular GAN, switch between training datasets (MNIST and FashionMNIST), and test various model configurations and hyper-parameter values. My research focused mainly on the following aspects of the GAN:

- *Number of parents*: One, two and three parents in a generation were tested. My evolutionary model does not include crossover between parents, so we only obtain a minor increase in quality as we increase the number of parents, while the computation time is obviously significantly increased. Theoretically, each parent generator learns a probability distribution over a set of modes in P_{real}, so with a higher number of parents we have a higher coverage over P_{real} (higher diversity), but this is already taken care of by the diversity fitness. Hence, one parent in a generation is the most optimal with our architecture.
- *Mutation functions*: The three considered mutation functions were: minimax, heuristic and least-squares mutation (see section 2.1.3). The framework was able to test combinations of these mutations, and visualize each mutation's contribution to the development of the GAN.
- *Types of optimizers*: As a bit of background, optimizers in Tensorflow are simply instances of optimization algorithms. In the context of an optimization problem in a multi-dimensional space, one can think of the gradients as pointers toward the general direction of the minimum of the space, and of the optimizer as something which decides how large the steps in the pointed direction need to be. For my GAN, I focused my experiments mainly on the stochastic gradient decent optimizer (SGD) and the Adam optimizer.
- *Complexity of discriminator and generator architectures*: the framework provides 3 neural network models (with varying layer complexity) for the discriminator and the generator.
- *Discriminator train steps*: the number of times we update the discriminator for each evolutionary step of the generators. One, two and three discriminator train steps were tested - two steps are superior with any

configuration, as one step and three steps make the discriminator too weak and too strong, respectively.

Algorithm 1 A general implementation of the E-GAN training algorithm.

Hyper-parameters: *num_epochs* - number of training epochs, *batch_size* - the batch size in each training iteration, *num_parents* - the number of generator parents in a generation, *num_mutations* - the number of mutations (children) per parent, *disc_optimizer* - optimizer for the discriminator, $child\_optimizer_i$ - optimizer for i^{th} mutation.

Formulas:
$J^{(D)}(x, z) = -\frac{1}{2}E_{x\sim p_{data}}log(D(x)) - \frac{1}{2}E_{z\sim p_z}log(1 - D((G(z)))$
$M_i^{(G)}(x) = i^{th}$ mutation
$F_q(z) = E_{z\sim p_z}log(D(G(z)))$
$F_d(z) = -log|| - \frac{1}{2}E_{z\sim p_z}log(1 - D((G(z)))]||$

function EGAN(x_{real})
  for *epoch* $\leftarrow$ 1 to *num_epochs* **do**
    while end of x_{real} not reached **do**
      for *train_step* $\leftarrow$ 1 to *disc_train_steps* **do**
        $x \leftarrow$ sample batch from x_{real}
        $z \leftarrow$ sample batch from p_z
        $loss \leftarrow J^{(D)}(x, z)$
        $D.variables \leftarrow$ update from $disc\_optimizer(lr_D, loss, D.variables)$
      for $j \leftarrow 1$ to *num_parents* **do**
        $z \leftarrow$ sample batch from p_z
        $loss_i \leftarrow M_i^{(G)}(G_j(z))$, $1 \leq i \leq num\_mutations$
        $child_{i,j}.variables \leftarrow$ update from $child\_optimizer_i(lr_G, loss_i, G_j.variables)$
        $F_{i,j} = Fq_{i,j} + \gamma Fd_{i,j}$
      Sort children $child_{i,j}$ according to their $F_{i,j}$
      $G_1, G_2... \leftarrow child_1, child_2, ..$

4.3 Optimization

This section highlights three lower-level optimizations.

4.3.1 Reuse discriminator outputs for all children

In my version of GAN, we update a single parent generator according to different mutations in order to create its children. More precisely, the mutations only consider the discriminator's score on the parent's generated samples, $D(G(z))$, to evaluate how "wrong" the parent is. Since this score is common among all the mutations, we can pre-compute it and use it to generate all the children (see Algorithm 1 line 11).

4.3.2 Remove redundancy in diversity fitness

Recall that the diversity fitness of a child calls for calculating the gradients of the discriminator, which in turn requires calculating the discriminator's loss function. Specifically, for $child_i$ the diversity score is :

$$F_d = -log||\frac{\partial}{\partial D}[-\frac{1}{2}E_{x\sim p_{data}}log(D(x)) - \frac{1}{2}E_{z\sim p_z}log(1 - D((child_i(z)))]|| \tag{4.1}$$

Since the term $-\frac{1}{2}E_{x\sim p_{data}}log(D(x))$ is common across all $child_i$, and we only use F_d to compare the performance of each $child_i$ with respect to the other children, in practice we can simplify the diversity score to:

$$F_d = -log||\frac{\partial}{\partial D}[-\frac{1}{2}E_{z\sim p_z}log(1 - D((child_i(z)))]|| \tag{4.2}$$

4.3.3 Parallelization

As an evolutionary algorithm, the E-GAN is inherently parallelizable. Essentially, the model consists of a sequence of operations performed on multiple children generators, such as updating the parent with respect to different mutations or calculating the fitness of each child. My implementation groups the operations at each stage together and performs them in parallel. Furthermore, as each operation is designed to be a set of heavy linear algebra computations with little logic, Tensorflow is able to enhance the optimization further by parallelizing the computations on a GPU.

4.4 Difficulties

My model's unconventional approach to training neural networks made it particularly difficult to implement in Tensorflow. In ordinary use cases of deep learning frameworks, where we have a single neural model and a single loss function, the framework couples the optimizer to the model's variables (we can look at the variables as remaining conceptually fixed, and only their values changing). For the purpose of my implementation, the optimizer cannot be bound to each parent because the parent is replaced by some child at the end of each iteration. What should happen instead is: the optimizer makes updates to a different model (the new parent) in each iteration but still recognizes that each new parent is the product of the previous one, which is not directly supported in Tensorflow. Working around this predicament required some source-code level manipulation of model and optimizer variables.

Chapter 5

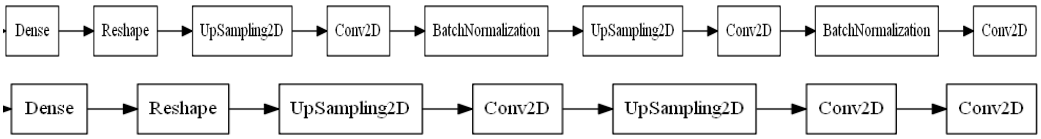
Results

The following chapter presents the results of my experiments comparing E-GAN and DCGAN on the MNIST and FashionMNIST datasets. The experiments highlight two improvements the evolutionary model brings to GAN training, but also expose a significant disadvantage of my model. The structure of the chapter is as follows: the first two sections show that the E-GAN does not rely on a complex neural architecture and it is able to produce good samples faster than DCGAN, and the last two sections show how after a certain point the E-GAN destabilizes and starts producing worse-looking images, whereas the DCGAN continues learning properly.

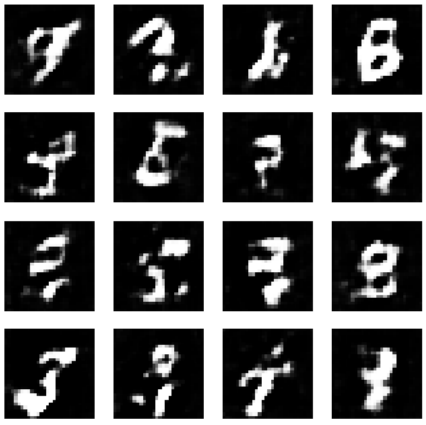
Before I begin presenting the ways the E-GAN performs better than the DCGAN, I wish to clarify an aspect of the evaluation of a GAN model. Since the output of a GAN cannot be correct or incorrect, the only way to evaluate it is visually (in fact, this is one of the difficulties of training GANs). Luckily, with the MNIST and FashionMNIST datasets determining the better-looking sample out of two is trivial. For the rest of the chapter, I will use terms like "recognizable" and "distinguishable" samples to describe generated images which resemble the training images, but are a bit noisy and blurry, and the term "good-looking" samples to describe images which very clearly resemble the training images.

5.1 Towards universal GAN training

The E-GAN's approach to stabilizing the training process abstracts away the neural network architecture of the discriminator and the generator. Experiments show that even though the quality of the generated images increases by adding heuristic-based layers to the neural networks of the discriminator and the generator, my model reaches discernible results even with the simplest architectures. This is a notable improvement over most current approaches, where we need to extend the architecture of the models with a set of tricks (such as batch normalization and dropout) to obtain any recognizable samples. The evolutionary model does not produce sharp and state-of-the-art images with the simple architecture (as shown on Figure 5.1), but it is still a significant step towards universal GAN training.



(a) DCGAN architecture (top) and E-GAN architecture (bottom)



(b) E-GAN generated samples on MNIST with the minimalist neural architecture

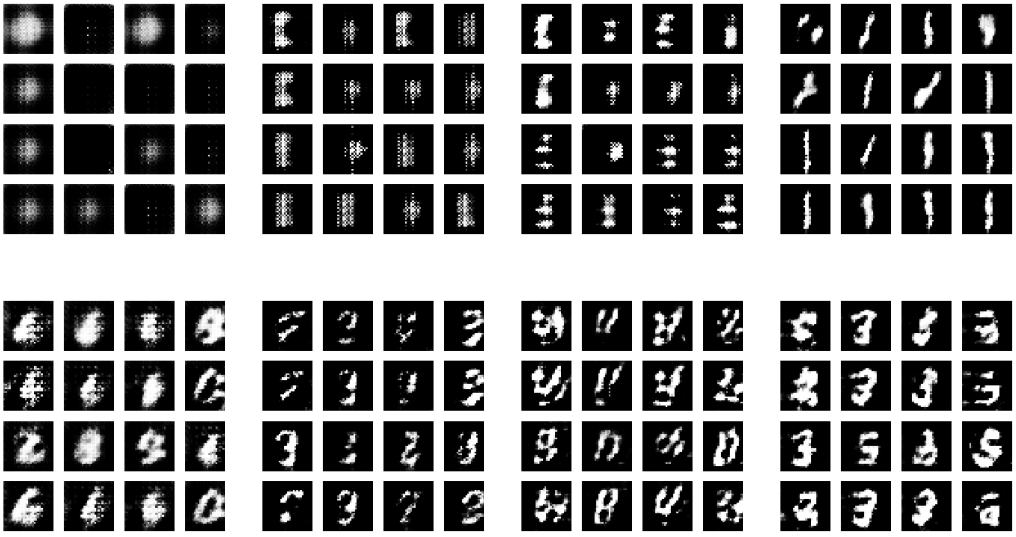
Figure 5.1: Comparison of complexity of architectures

The figure above compares my model and DCGAN in terms of the minimal network complexity to get discernible generated samples on MNIST. For DCGAN (architecture on top), we need a sequence of Batch Normalization layers, but for EGAN (bottom architecture) they are not necessary. With the simplest architecture, the DCGAN only produces noise, but the E-GAN's outputs resemble a six, a three and an eight.

5.2 Speed

The E-GAN also introduces a significant improvement in terms of speed of training. Although my model is slower than the DCGAN per iteration (because it performs updates on multiple generators instead of one), it starts producing good-looking images much faster the DCGAN. To be more specific, let's consider training on the MNIST dataset on an NVIDIA Tesla K80 GPU.

In this setting, E-GAN is roughly twice slower than the DCGAN, as it takes 11 minutes to complete an epoch compared to DCGAN's 5.5 minutes per epoch. However, our model reaches decent solutions in the first few epochs (20-40 minutes of training), while the DCGAN takes 20 epochs to get the same quality of images (more than an hour of training). Also worth noting is the fact that with my model, we observe samples roughly resembling digits already in the first epoch, while during the first 10 epoch the DCGAN produces only noise and meaningeless shapes. So far we have only discussed the speed-up on a single simple GPU. As we increase the computational power (use a more powerful GPU or use a cluster of GPUs), the speed-up compared to rivaling methods will only become more superior (see section 4.3.3).



(a) 10 minutes (b) 20 minutes (c) 30 minutes (d) 40 minutes

Figure 5.2: Speed comparison of DCGAN and E-GAN

The figures displays the generated samples of a DCGAN (top) and an E-GAN (bottom) in the first 40 minutes of training on MNIST. The former only manages to generate unidentifiable shapes. The latter already exhibits distinguishable digits after the first epoch (a two, a zero, and a nine).

5.3 Image quality

Even though my model reaches solutions much faster, the DCGAN eventually ends up producing better-looking images. The E-GAN tends to reach a certain "peak" of image quality, after which the quality of generated samples slowly degrades: by the 10th epoch of training, the GAN simply produces random noise. On the other hand, the DCGAN takes a lot more time to start producing distinguishable sample, but makes steady progress and by the 35th epoch produces clear images.

Figure 5.3: Image quality comparison of E-GAN and DCGAN
The E-GAN (top) produces distinguishable clothes in the 2nd epoch of training (we can notice a blouse, pants and shoes), but after that point the quality of the clothes slowly degrades. At the end of training, the DCGAN produces good-looking images alike the images from the training set.

5.4 Stability

To examine the training stability of a GAN, one must look 'under the hood' and learn to interpret the loss functions of the discriminator and the generator, which is not at all obvious, especially in an environment with multiple generators like ours. In a regular neural network, the loss function quantitatively tells the network how "wrong" it is - the lower the loss the better-performing the network is. In a GAN, however, we cannot draw such a straightforward conclusion. The generator loss represents how "wrong" the generator is *according to the discrmininator*. Under this definition, low generator loss does not necessarily mean a good generator - it could be the case that the discriminator is just bad and easily fooled. Because of this, the most favorable scenario in training GANs is when both the generator and discriminator loss are stable around some intermediate value (neither too high nor too low) - this indicates that both networks are successfully learning from each other and getting better. The same philosophy holds in my evolutionary setting: the only difference is that instead of a single generator loss function we have multiple losses represented by the mutations. Here are the most common anomalies of the described "equilibrium" which slow down or completely destabilize training:

- *Discriminator loss drops too low*: this phenomenon indicates that the discriminator became too powerful and now rejects all generated samples, thus not providing the generator with any useful feedback.
- *Generator loss drops too low*: this means that the generator became much more powerful than the discriminator.
- *Both losses explode*: this case is basically the divergence issue described in section 1.2.3.

Now, back to the analysis of the training stability of my model. Various experiments showed that the losses are stable at the beginning of training, up to a point when the losses start destabilizing, usually permanently (see example of this in Figure 5.5). This process corresponds to the reaching of a "high" of image quality followed by gradual regression of quality described in the previous section. Conversely, the DCGAN makes slow but always steady progress, reflected by stability in its discriminator and generator loss (demonstrated in Figure 5.4).

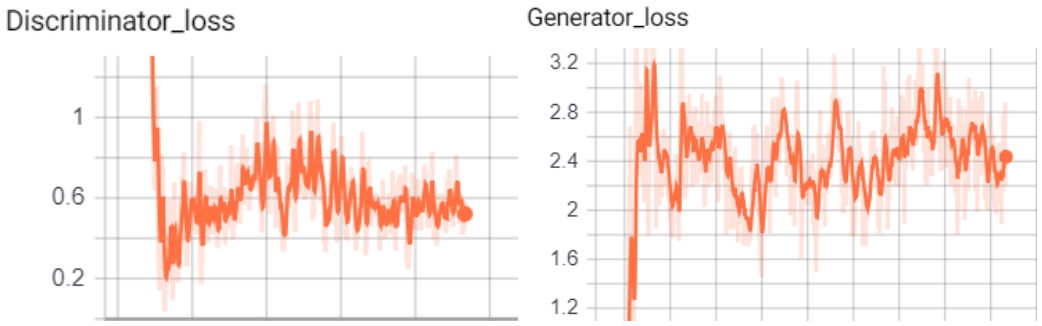


Figure 5.4: Graph of discriminator and generator loss on a DCGAN

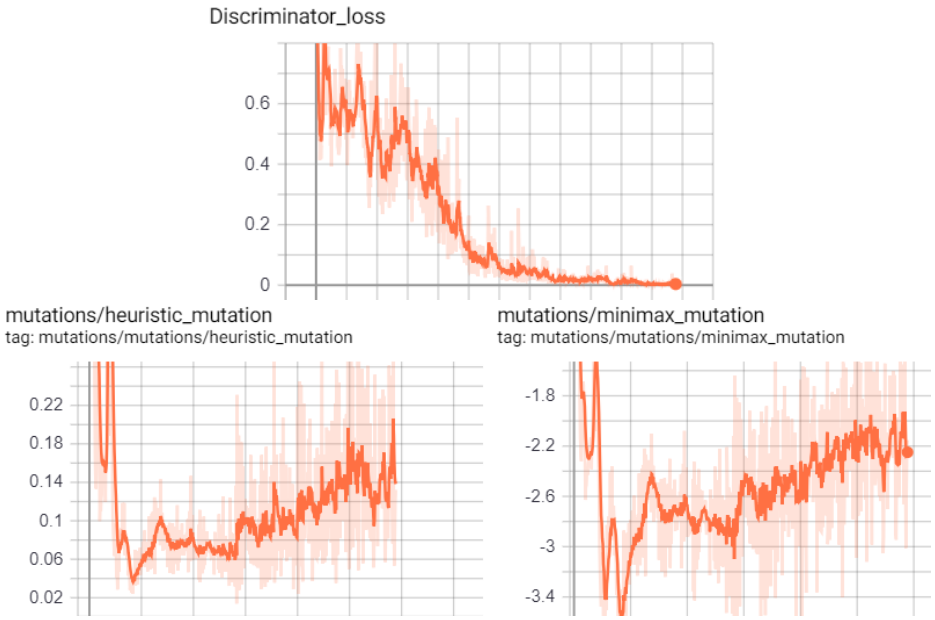


Figure 5.5: Graph of discriminator and generators loss on an E-GAN
The figure shows the discriminator loss and the mutations of an E-GAN over 10 epochs (this particular one only uses the heuristic and minimax mutation for the sake of demonstration simplicity). In this instance, the discriminator loss drops too low - the discriminator becomes too powerful. This results in the generators not learning anything - the generator mutations increase instead of decrease.

Chapter 6

Analysis

The following chapter aims to provide intuitive and theory-based explanations of the properties of an E-GAN. The chapter focuses on two key properties of the E-GAN: the choice of mutation functions and the choice of optimization algorithms for the discriminator and the generator. There are a few other highly influential aspects of my model, but I decided to only cover the aforementioned two in depth, as they capture the essence of the evolutionary training process.

6.1 Choice of mutations

My experiments on using combinations of the three tested mutations demonstrate that the most optimal configuration both in terms of training stability and generated image quality is to use all three mutations. What follows is an intuitive explanation of this empirical conclusion - before proceeding forward, the reader is advised to revise section 2.1.3, where the individual properties of each mutation are discussed.

It has been theoretically proven that the discriminator has an easy job distinguishing the generator's samples at the beginning of training [1]. These theoretical results are also supported by my experiments on training E-GAN with numerous configurations. On the other hand, with training the generator gets increasingly better and in later stages it is much harder for the

discriminator to separate out the generator's samples. These two relatively simple observations expose a natural pattern in GAN training: at the beginning, the generator gets low discriminator scores, and towards the end, it gets high scores.

Because of this, at the initial stages of training, when $D(G(z)) \to 0$, the minimax mutation makes negligible progress while the heuristic mutation makes significant progress. Conversely, towards the end of training, the heuristic mutation stagnates because of vanishing gradients but the minimax mutation offers unrestrained improvement. Thus, in theory, the evolutionary model benefits the most from the heuristic mutation in the initial stages of training, from the least-square mutation in the intermediate stages (one can see this mutation as the "middle-ground" between minimax and heuristic), and from the minimax mutation in the later stages of training. Looking at the rate at which each mutation produced the fittest child in each epoch of training, we can observe that in practice the E-GAN truly does utilize the mutations in the aforementioned fashion (see Figure 6.1). This indicates all three mutations are necessary and play an individual role in training of an E-GAN.

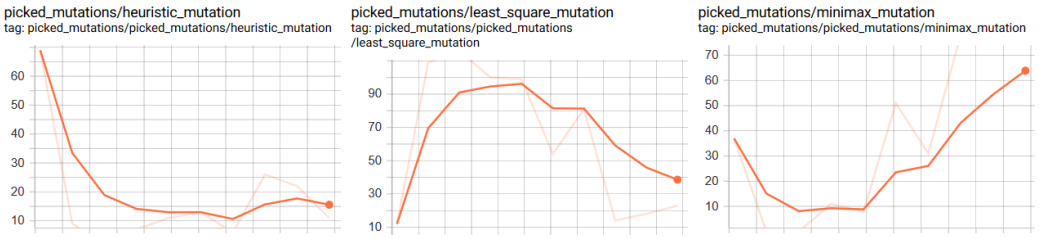


Figure 6.1: Number of times each mutation is picked per epoch
The heuristic mutation (left) is picked the most in the first 2 epochs. The least-squares mutation (middle) is picked the most in epochs 2-5. Minimax mutation (right) is picked the most in the final epochs of training.

6.2 Optimizers

Choosing the right optimizer for the discriminator and the generator is arguably the leading cause for the training instability in E-GAN (presented in section 5.4). Before I dive in the analysis of the impact of different optimizers, I will outline the properties of each optimizer most relevant to my work.

In my experiments, I mainly focused on the SGD optimizer (as an example of a simple parameter-less algorithm) and the Adam optimizer (representing a more complex class of algorithms). The SGD optimizer simply takes fixed steps in the direction of the gradients. The Adam optimizer, however, keeps track of information about past updates to make more informed decisions about the next step size. For instance, an Adam optimizer will make larger steps in a dimension in which we previously made large steps (known as momentum), and vice versa. The difference between these two approaches is visualized in Figure 6.2.

The Adam optimizer operates with a single loss function, or in other words, a single search space. As such, it does not have an immediate definition in my evolutionary environment, where we have three mutations, or equivalently, three search spaces. My work primarily focused on the following three solutions to this predicament:

- **SGD for the generators**: this is the worst performing approach. On one hand, if the discriminator uses an SGD optimizer as well, training is relatively stable but also extremely slow, and the E-GAN loses its benefits. However, if the discriminator uses an Adam optimizer instead, it becomes too dominant over the generators.
- **Separate Adam optimizer for each mutation**: a much better performing approach, but suffers from serious training destabilization. The issue can be summarized as follows: if the evolutionary model has not picked mutation X in a long time (and instead was making updates according to some other mutation Y), the Adam optimizer for X will have outdated historic information. The next time the algorithm chooses mutation X, its Adam algorithm will not be able to make a meaningful update.
- **One Adam optimizer for all mutations**: the best approach in practice in terms of quality of generated samples, even though it does not have a precise theoretical definition. Essentially, we have a single optimizer which makes updates according to three different search spaces (corresponding to the three different mutations). Similar to the previous example, we get inconsistent updates when the evolutionary model chooses a mutation which hasn't been picked for a larger number of iterations.

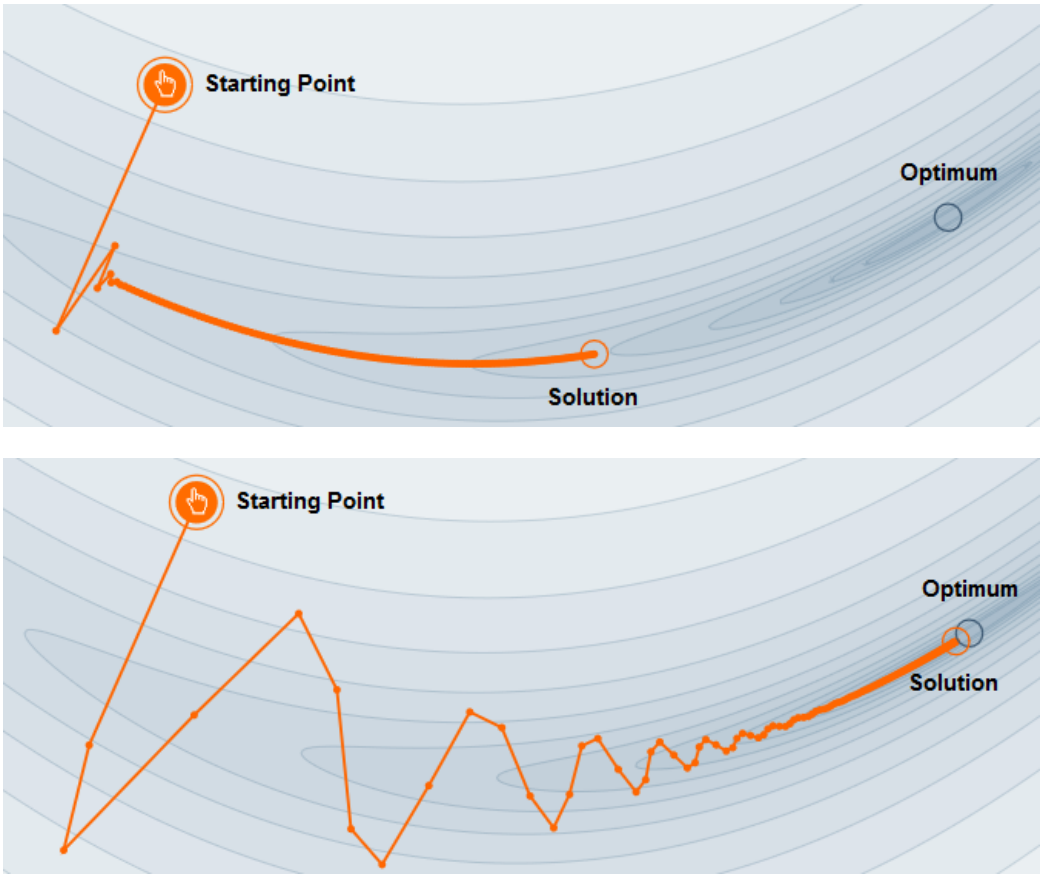


Figure 6.2: Comparison of SGD and Adam optimizer
The SGD optimizer (top) gets stuck on tiny oscillatory updates and does not even reach the optimum. The Adam optimizer (bottom) is smarter: over time it learns to make larger steps in the horizontal direction, and tiny steps in the vertical direction, eventually converging near the optimum. The figure was created with the visualisation tool provided in [12].

This concludes the analysis on the two most important traits of my model. Note that this is not an exhaustive analysis by any means: the E-GAN is affected by a number of other features such as the diversity factor in the fitness score, but their interpretation follows a similar reasoning. The next chapter summarizes the results and conclusion from my work, and proposes a set of aspects of the E-GAN worth exploring in more depth in the future.

Chapter 7

Conclusion

In previous chapters, I demonstrated the results from my experiments and my analysis of the core properties of my model. In this chapter, I outline the main conclusions that can be drawn from these results and discuss what the conclusions insinuate about the current position of the E-GAN in the world of deep learning, and in the world of GANs specifically. Finally, I complete this report with a list of promising research topics as an extension to my work.

7.1 Summary

In this project, I extend the work on the previously proposed evolutionary GAN. The E-GAN primarily aims to mitigate the problem of *mode collapse* and the *individual weaknesses* of currently popular adversarial objectives, by creating an evolutionary environment where a generation of generators utilize different adversarial objectives to create children, and the best performing children are passed on to the next generation.

In addition to the originally observed improvements, my trials demonstrate that the E-GAN is able to achieve meaningful results much faster than the DCGAN, and is able to perform with almost any discriminator and generator architecture, rather than relying on a specific one like previous works. I also expose a significant hindrance to its training: after reaching distinguish-

able samples, the E-GAN destabilizes and starts producing worse samples. Finally, my work analyzes the E-GAN from perspectives such as the choice of an optimization algorithm and the role of each mutation to provide basic intuition for the advantages and disadvantages of the model.

7.2 The importance of the E-GAN

Although this approach is still in its initial phase and does not produce state-of-the-art sharp-looking images, it exhibits a lot of potential because of two main reasons:

- **It is a step towards a universal GAN model**: The ideal in each area in machine learning is to develop a general model which can easily translate between problems. For GANs in particular, the long-term goal is to have a single GAN architecture which can train on any problem, but as explored in Chapter 3 most current approaches fail to achieve this goal. On the other hand, the E-GAN does exhibit some universality (see section 5.1), and although in its current state it suffers from severe instability, I believe that this problem can be solved with future research on the topic. After all, the original GAN work itself [7] was unable to produce any realistic samples until later changes, such as the improved techniques for GANs [21] and the introduction of the DCGAN [17].
- **It is a novel approach to evolutionary techniques**: In light of the huge success of gradient-descent-based approaches in the past decades, the deep learning community has somewhat neglected evolution as an optimization strategy. In recent years evolution has experienced a minor resurgence in a few deep learning areas such as reinforcement learning [19]. The EGAN's unconventional approach to handling these techniques can potentially be translated to other neural networks, and can open the door to other evolutionary techniques in deep learning.

7.3 Future research

Being such an unexplored topic, the E-GAN offers a lot of room for research. In this section, I propose three instances of future work that could improve the model significantly:

- **Mathematical definition**: Although both the original paper and my work are influenced by theoretical insight, the E-GAN is still lacking a firm mathematical description. The fact that a lot of fundamental details of the model are theoretically vague is perhaps the biggest obstacle to overcoming the serious training problems. For instance, it is well-known which probability distribution distance measure is minimized by each individual mutation, but the distance measure minimized by combining these mutations in our evolutionary environment is still theoretically ambiguous. Having this kind of a robust mathematical definition would help future research produce more educated ideas on how to improve E-GAN training. The next proposals rely on such a definition.
- **Wasserstein and f-divergence mutations**: The WGAN [2] and the f-GAN [18] are arguably the most sophisticated GAN models used heavily in practise. These two approaches aim to minimize the Wasserstein distance and the f-divergence, respectively. With the Wasserstein adversarial objective, the GAN sometimes suffers from low-quality images and convergence problems, and the f-divergence adversarial objective has difficulty with areas where P_{real} and P_{model} do not overlap (the same problem as the minimax mutation - in fact, the minimax loss is a special case of the f-divergence loss). It is worth exploring if these problems can be overcome by utilizing these adversarial objectives as mutations in the E-GAN.
- **Other evolutionary techniques**: As described in section 2.1.6, the E-GAN as a concept is not motivated by some biological insight. Instead, it employs evolutionary methods as a means to combine multiple adversarial objectives and enforce generated sample quality and diversity on the generator. The family of evolutionary algorithms has a number of approaches that can possibly replace the current minimalist genetic algorithm used in my work. For instance, a previous work on NEAT [22] suggests several approaches to combining neural networks -

future work on E-GANs could exploit this to include *crossover between parent generators* in the process of creating children.

Bibliography

[1] Martín Arjovsky and Léon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *CoRR* abs/1701.04862 (2017).

[2] Martín Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017, pp. 214–223. URL: http://proceedings.mlr.press/v70/arjovsky17a.html.

[3] David W. Corne and Michael A. Lones. "Evolutionary Algorithms". In: *CoRR* abs/1805.11014 (2018). arXiv: 1805.11014. URL: http://arxiv.org/abs/1805.11014.

[4] Ishan P. Durugkar, Ian Gemp, and Sridhar Mahadevan. "Generative Multi-Adversarial Networks". In: *CoRR* abs/1611.01673 (2016). arXiv: 1611.01673. URL: http://arxiv.org/abs/1611.01673.

[5] Brendan J Frey, Geoffrey E Hinton, and Peter Dayan. "Does the Wake-sleep Algorithm Produce Good Density Estimators?" In: *Advances in Neural Information Processing Systems 8*. Ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. MIT Press, 1996, pp. 661–667. URL: http://papers.nips.cc/paper/1153-does-the-wake-sleep-algorithm-produce-good-density-estimators.pdf.

[6] Ian J. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *CoRR* abs/1701.00160 (2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[7] Ian Goodfellow et al. "Generative Adversarial Nets". In: (2014). Ed. by Z. Ghahramani et al., pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[8] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. In: *CoRR* abs/1704.00028 (2017). arXiv: 1704.00028. URL: http://arxiv.org/abs/1704.00028.

[9] Quan Hoang et al. “Multi-Generator Generative Adversarial Nets”. In: *CoRR* abs/1708.02556 (2017). arXiv: 1708.02556. URL: http://arxiv.org/abs/1708.02556.

[10] Diederik P. Kingma. “Fast Gradient-Based Inference with Continuous Latent Variable Models in Auxiliary Form”. In: *CoRR* abs/1306.0733 (2013). arXiv: 1306.0733. URL: http://arxiv.org/abs/1306.0733.

[11] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[12] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[13] Christian Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: *CoRR* abs/1609.04802 (2016). arXiv: 1609.04802. URL: http://arxiv.org/abs/1609.04802.

[14] Ming-Yu Liu and Oncel Tuzel. “Coupled Generative Adversarial Networks”. In: *CoRR* abs/1606.07536 (2016). arXiv: 1606.07536. URL: http://arxiv.org/abs/1606.07536.

[15] Xudong Mao et al. “Multi-class Generative Adversarial Networks with the L2 Loss Function”. In: *CoRR* abs/1611.04076 (2016). arXiv: 1611.04076. URL: http://arxiv.org/abs/1611.04076.

[16] Vaishnavh Nagarajan and J. Zico Kolter. “Gradient descent GAN optimization is locally stable”. In: (2017). Ed. by I. Guyon et al., pp. 5585–5595. URL: http://papers.nips.cc/paper/7142-gradient-descent-gan-optimization-is-locally-stable.pdf.

[17] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016. 2015. URL: http://arxiv.org/abs/1511.06434.

[18] Kevin Roth et al. “Stabilizing Training of Generative Adversarial Networks through Regularization”. In: *CoRR* abs/1705.09367 (2017). arXiv: 1705.09367. URL: http://arxiv.org/abs/1705.09367.

[19] Tim Salimans et al. “Evolution Strategies as a Scalable Alternative to Reinforcement Learning”. In: (2017). eprint: arXiv:1703.03864.

[20] Tim Salimans et al. “Evolution Strategies as a Scalable Alternative to Reinforcement Learning”. In: *CoRR* abs/1703.03864 (2017).

[21] Tim Salimans et al. “Improved Techniques for Training GANs”. In: *CoRR* abs/1606.03498 (2016). arXiv: 1606.03498. URL: http://arxiv.org/abs/1606.03498.

[22] Kenneth O. Stanley and Risto Miikkulainen. “Evolving Neural Networks Through Augmenting Topologies”. In: *Evol. Comput.* 10.2 (June 2002), pp. 99–127. ISSN: 1063-6560. DOI: 10.1162/106365602320169811. URL: http://dx.doi.org/10.1162/106365602320169811.

[23] Hao Tang et al. “Dual Generator Generative Adversarial Networks for Multi-Domain Image-to-Image Translation”. In: *CoRR* abs/1901.04604 (2019). arXiv: 1901.04604. URL: http://arxiv.org/abs/1901.04604.

[24] Chaoyue Wang et al. “Evolutionary Generative Adversarial Networks”. In: *CoRR* abs/1803.00657 (2018). arXiv: 1803.00657. URL: http://arxiv.org/abs/1803.00657.

[25] Han Zhang et al. “StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks”. In: *CoRR* abs/1612.03242 (2016). arXiv: 1612.03242. URL: http://arxiv.org/abs/1612.03242.

[26] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”. In: *CoRR* abs/1703.10593 (2017). arXiv: 1703.10593. URL: http://arxiv.org/abs/1703.10593.