Generative Adversarial Networks for Population Genetics

Mohamed Ali
Adviser: Sara Mathieson

Apr 29, 2022

Abstract

Population genetics can be defined as the study of distributions and changes in the genetic data of populations through time. This field relies heavily on simulated data for validation. Simulating a population involves knowledge of demographic parameters such as mutation rate, recombination rate, population size, and times when changes in population has occurred. In the paper (Wang et al., 2021), an algorithm to estimate these demographic parameters and adapt to data from different populations is proposed, a Generative Adversarial Network that learns the probability distribution of a population so that it can estimate what parameters have led to such configuration. There are two problems with the approach presented in the paper, as is the case with the classical GAN approach from the original paper, (Goodfellow et al., 2014). The problems are that the learning process of a classical GAN is not stable and relies heavily on hyperparameters and the loss function is not useful for understanding the development of the model. Another problem that arises when trying to classify special parameters such as natural selection is mode collapse, a situation where a generator can only generate a single aspect or type of output, without fully exploring the probability space. The last problem we will address in this paper is evaluation. It's very difficult to evaluate the accuracy of a model if you can't "view" the final product. This is mostly the case in population genetics, where parameters and simulations might not be enough to validate a model.

Contents

1 Introduction

As population genetics datasets grow in size, understanding and analyzing the information becomes more and more of a difficult task. Machine Learning has been an indispensable tool for recognizing patterns and understand how evolutionary events take place. As put in (Schrider & Kern, 2018), "many advances in the field [of population genetics] have come from the introduction of new stochastic population genetic models, often of increasing complexity, that describe how population parameters (e.g., recombination or mutation rates) might generate specific features of genetic polymorphism." This shows the power of Machine Learning (ML) algorithms, as it allows researchers to infer these population parameters while also being agnostic about how the dataset is created. While classical statistical estimation and approximation can roughly estimate such parameters, Machine Learning techniques aim to optimize the accuracy of an algorithm by learning the weights of a map from the input data to the output, which allows us to learn more about the probability space even if the labelling of the dataset is not perfect, and even if the dataset is not labelled at all, as in the case of unsupervised learning. ML can also somewhat address the problem of the ever-increasing size of population genetics datasets. As mentioned in Schrider & Kern, "The ML paradigm enables the efficient use of high-dimensional inputs which act as dependent variables, without specific knowledge of the joint probability distribution of these variables. Inputs that consist of thousands of variables (also known as 'features' in the ML world) have been used with great success and increases in the number of features can often yield greater predictive power." This demonstrates the ability of ML algorithms to utilize high-dimensional data to aid in the processes of pattern recognition and parameter estimation.

To simulate population genetic data, researchers rely on evolution simulators. A very capable simulator is msprime (Kelleher, Etheridge, & McVean, 2016). msprime takes as input genetic and demographic parameters of a genetic population, such as mutation and recombination rates, and outputs a tree sequence representing the simulated population. The problem with msprime, and most evolution simulators, is the reliance on the few genetic parameters provided by researchers. This assumes that the parameters needed by the simulator are the only factors in the evolution of any population, which is most probably not the case.

In this paper, we explore the methods and techniques that have been devised to learn the evolutionary parameters passed to msprime using ML methods. The goal is create a framework that can automatically and accurately select the evolutionary parameters and use it to generate simulated data that accurately mirrors the real population genetic data. In the next subsections, we will provide population genetics and machine learning background information to facilitate understanding the status quo of research in this area and our methods. We will further discuss msprime and the evolutionary parameters it requires and we will

examine a data generation machine learning method, Generative Adversarial Networks (GANs)(Goodfellow et al., 2014) and their vast applications in population genetics, including the promise of learning the probability distribution of data without resorting to labelled data.

1.1 ML Background

Machine Learning can be categorized into two major categories: Supervised Learning and Unsupervised Learning. While supervised learning depends on annotated (labeled) data to learn to predict the label or class of test data points, unsupervised learning explores the probability space of a dataset to uncover patterns and determine the underlying structure of a given dataset. Both supervised and unsupervised learning are applied on a **training dataset**, with the difference that unsupervised learning doesn't require the dataset to have **labels**. This means that supervised learning is more fit for classification problems where training data is classified into a set of classes with the goal of training a model that can classify new data into one or more of these classes, only based on the structure and organization of the training set. In contrast, unsupervised learning can tell us more about the nature and structure of a dataset without any pre-determined human categorization of the data. This opens the door for a multitude of applications including, clustering, visualization, and dimensionality reduction.

Unsupervised learning is especially advantageous in the context of population genetics. Population genetics is concerned with exposing unintuitive, inconspicuous relations between individuals. Unsupervised learning makes this possible by exploring patterns and structural similarities that reveal these sought-after relations. It can further categorize the data into similar groups based on the relatedness relations it has uncovered. This circumvents the human bias and error resulting from data labelling. As put by Schrider & Kern, "Perhaps most important among [ML adavantages] is the ability to circumvent using idealized, parametric models of the data when labeled training data can be obtained from empirical observation $\cdots$ we can use ML to train algorithms to recognize phenomena as they are in nature, rather than how we choose to represent them in a model." In the next subsections, we will examine three essential parts of a ML model: **training**, **testing**, and **model fitting**. We will then explore three concepts that are of special importance to us in understanding what has been done in the field and our approach to solving the problem, **Deep Learning**, **Convolutional Neural Networks** and **GANs**.

1.1.1 Training

A machine learning model is a representation of a dataset as the relationship between its data points. The model is essentially a mapping from the data points, or **examples**, to an output. This mapping is essentially a function with multiple layers and thousands of variables and coefficients. Training a ML model

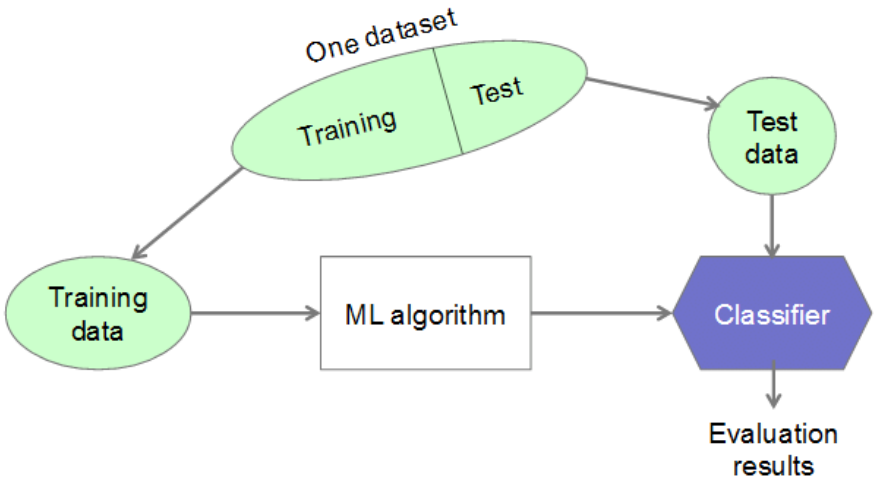


Figure 1: Training and testing a machine learning algorithm. One dataset is split into two parts, training and test sets. The training set is used in the learning process, while the test set in independent of the training set and is only used for evaluation. The idea behind this split is to make sure the model is not just memorizing the examples of the dataset. The test set makes sure that the model has approximated the weights well enough to predict examples that it hasn't seen before, which is what we are after. The figure is attributed to (Pousada et al., 2018)

is essentially approximating the best coefficients that makes the mapping from input to output as accurate as possible. Such coefficients are called **weights**. The input for model training is a **training set**, a subset of the entire dataset used for model training and testing. The dataset is split into the training set and the **test set**. Common split percentages range from 67% training, 33% test to 90% training, 10% test, depending on the size of the dataset and the application. We define the training process as learning the weights of the mapping from the training dataset to the output of the model.

1.1.2 Testing

Using the test set we defined above, a model is evaluated and scored on various metrics to determine the success of the learning process. Common performance metrics are **accuracy**, **false positive rate**, **false negative rate**, **recall**, **precision**, and **confusion matrix**. Although these metrics are widely used, they mostly apply to supervised learning with an output of two classes, usually 1 or 0, with the exception of accuracy. However, accuracy is not the most accurate evaluation measure in unsupervised learning either. We will explore more relevant evaluation metrics for the context of unsupervised learning and population genetics later in this paper. Figure 1 shows the train-test split model.

1.1.3 Model Fitting

Fitting a ML model is the process of building the mapping f that approximates the weights from input to output. In order to create a good mapping, a **loss function** has to defined. The loss function tells the model how good or bad its prediction for a specific test example was. Model fitting can be defined as the process of minimizing the average loss over the training set. **Overfitting** is a situation where a model is more sensitive to the structure of each data example in the dataset rather than the underlying structure of the entire dataset at hand. this leads to the model being extremely accurate in recognizing examples from the training set, but not as good in classifying other examples, which emphasizes the importance of using an independent test set for evaluation. On the otherhand, **underfitting** refers to a model that doesn't do well on the training set as well as new examples. A **good fit** is a model that's neither underfit nor overfit; that is, a model that can generalize the underlying structure of the training set so that it's able to recognize and model both the training and the test sets.

1.1.4 Deep Learning (DL)

Deep Learning is a class of Machine Learning that is characterized with complex, non-linear functions. DL is typically used with large datasets where features are not provided. A DL algorithm uses multiple processing layers to abstract data into machine-generated features. These features are then used to understand the underlying structure of the dataset and classify new examples if necessary. A very well-known application of Deep Learning in Convolutional Neural Networks, which we will discuss next.

1.1.5 Convolutional Neural Networks (CNNs)

As described in Jiuxiang et al., "Convolutional Neural Network (CNN) is a well-known deep learning architecture inspired by the natural visual perception mechanism of the living creatures." A CNN has three basic components: **convolutional layers**, **pooling layers**, and **fully connected layers**. A convolutional layer is used to compute feature maps by sliding a **kernel** over the sample input. A kernel is a window of set size. The convolution layer looks at the input through these window to identify similarities between training examples. Refer to figure 2 for a detailed example of how convolution works. A pooling layer reduces the dimension of the data through averaging, choosing the max data point, or choosing the min data point, among other techniques. This allows for faster processing and a shift-invariant result. The pooling layer is typically placed between two convolutional layers. The fully connected layer is typically the last step in a classification task. As put by (Jiuxiang et al., 2018), the goal of the FC layer is "to perform high-level reasoning. They take all neurons in the previous layer and connect them to every single neuron of current layer to generate global semantic information."

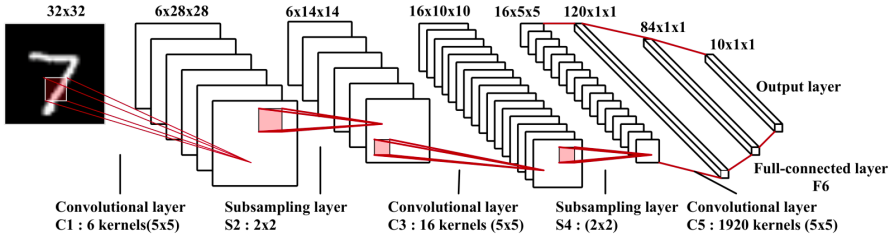


Figure 2: The architecture of a digit classification convolutional neural network known as LeNet-5. The first layer in this pipeline is a convolutional layer with 6 (5×5) kernels. Each (5×5) kernel slides over the entire example, in our case a 32×32 image of the number 7, producing 28×28 different configurations (placements of a 5×5 kernel on a 32×32 image). The result is a layer with $6 \times 28 \times 28$ data points. The next layer is a 2×2 pooling, or subsampling, layer. This layer shrinks 2×2 data points into one data point, resulting in a simpler layer of $6 \times 14 \times 14$ data points. Multiple convolutional and pooling layers follow to further strengthen the feature map learnt by the network. At the end, a fully connected layer maps from the last convolutional into the output layer, a 10 x 1 x 1 layer representing the classes of numbers from 0 to 9. This figure is adopted from (Jiuxiang et al., 2018)

1.1.6 Generative Adversarial Networks (GANs)

A Generative Adversarial Network, or GAN, is a method of generating synthetic data that closely mimics a real distribution of data. GANs employ two neural networks, a discriminator and a generator. While the discriminator's job is to tell apart real and fake data, the generator's job is to fool the discriminator into thinking that what it generates is real. Figure 3 illustrates this pipeline. GANs' ability to produce synthetic data benefits us greatly in the field of population genetics, which we will talk about next.

1.2 Population Genetics Background

Population genetics is the study of the set of genomes that compose individuals of a natural population. The genetic composition of a population is ever-changing. Multiple factors affect the genetic composition, some of which are **migration of individuals**, **segregation of genes**, **recombination of genes**, **genetic drift**, and **gene mutations**. See figure 4 for an overview of those events. These factors, among others, combined with time, shape what we know as **evolution**. Studying these factors is in the crux of population genetics. As put by (Coop, 2013), "Population genetics is the study of the genetic composition of natural populations. It seeks to understand how this composition has been changed over time by the forces of mutation, recombination, selection, migration, and genetic drift." The **ground truth** of a population is empirical evidence that is known to be real about all the factors, or **genetic parameters** of a population. The field of population genetics suffers from very limited ground truth, for the obvious reason that no one has been able to record the genetic

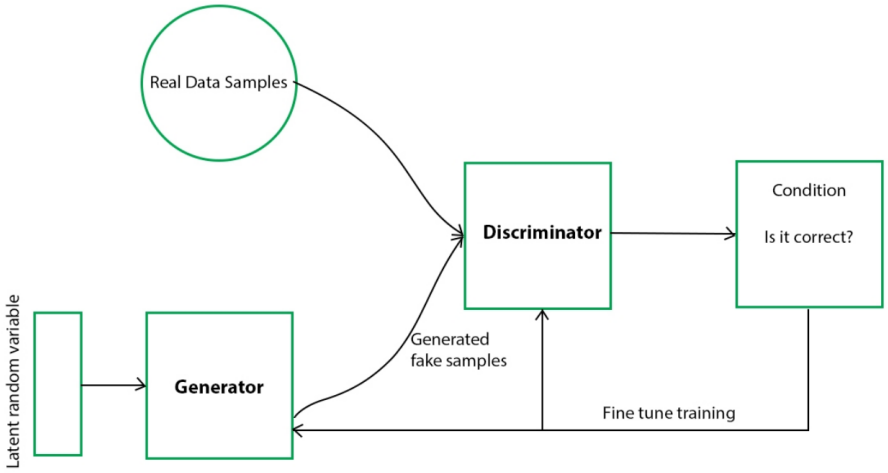


Figure 3: A schematic of the GAN pipeline. The generator network starts with random noise as input and produces a data sample. The discriminator takes as input both real samples and fake samples from the generator. At first, the generator samples will be nowhere near the real data samples. After several iterations, the generator will learn, through a loss function and updating its weights to minimize the loss, to generate better samples. The discriminator will learn to better classify into real and fake. This back and forth between the discriminator and the generator eventually yields a generator that is capable of producing samples that mirror the real data samples. The ultimate goal with GANs is to produce a generator that is capable of generating synthetic data that fools the discriminator; that is, produces a discriminator accuracy of 50%. This figure is adopted from (Roy, 2019)

composition of individuals who lived thousands of years ago. This problem of limited ground truth is also the main reason unsupervised learning methods are more preferred in the field, as the ground truth serves as **labels** for the genetic composition of a population. In order to exploit the ML techniques available for us in the field of population genetic, we need very large datasets so that we can circumvent using labels and resort to unsupervised learning methods to analyze the underlying structure of the genetic composition of populations to learn their genetic composition. In other words, we care about creating synthetic data that resembles specific populations in order to provide a large database for ML algorithms to study and analyze such populations, overcoming the limited ground truth problem in the field. The problem is that, in order to simulate such populations, we need to infer historical parameters specific for the population. This process of generating fake data governed by historical parameters has been the focus of the field for years. As put by Schrider & Kern, "Population genetics over the past 50 years has been squarely focused on reconciling molecular genetic data with theoretical models that describe patterns of variation produced by a combination of evolutionary forces." These parameters interact in so many ways, some of which are well known "models", others are not as well-known but possible. Figure 5 shows four well-studied models presented in Wang et al., 2021.

1.2.1 msprime

msprime (Kelleher et al., 2016) is a population genetics simulator based on tskit. Msprime can simulate random ancestral histories for a sample of individuals. Msprime results are limited by the choice of input parameters, the genetic parameters of the population. In order to produce simulated data that represents a real population data, these input parameters have to be selected carefully. Given the very limited ground truth in the field, selecting these parameters is not an easy task and requires the use of ML methods to accomplish. Msprime is a powerful tool; it can be used in different contexts, but we will focus more on its use as a generator in the GAN context.

1.3 Goals

We will review some of the most prominent GAN approaches to tackle the issue of limited ground truth in population genetics. We will start be reviewing the original GAN paper (Goodfellow et al., 2014) to see how the original algorithm fits in the population genetics context. Then, we will review PG-GAN (Wang et al., 2021) and WGAN (Arjovsky, Chintala, & Bottou, 2017). PG-GAN proposes an approach for automatic inference of demographic parameters using GANs, which ties in closely with our ultimate goal of facilitating the creation of synthetic data for population genetics. WGAN is a modified GAN algorithm with a novel distance measure, the Earth Mover distance. They propose ways around mode collapse and irrelevance of the loss metric, which are problematic in PG-GAN and the original GAN algorithm. Finally, we will review Borji, 2019, a

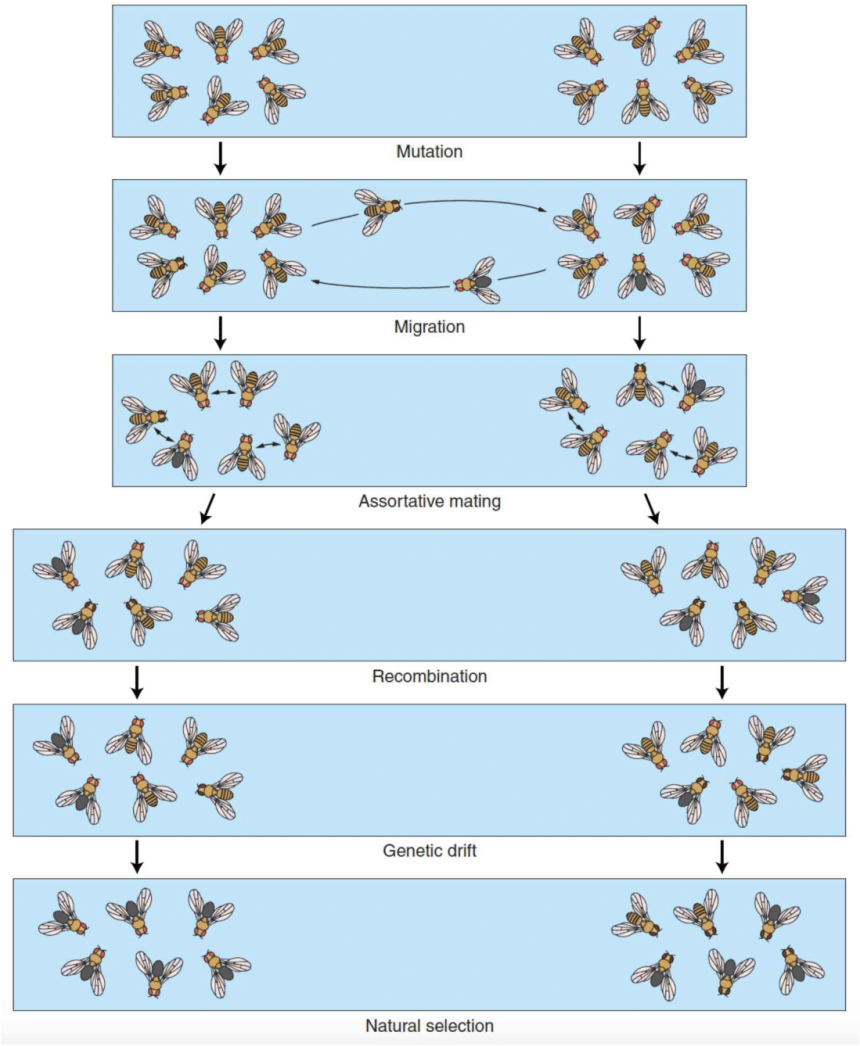


Figure 4: An illustration of the factors causing the change of the genetic composition of a population. The figure shows how various historic events can lead to genetic variations within a population of flies. The figure is attributed to (F. et al., 2020)

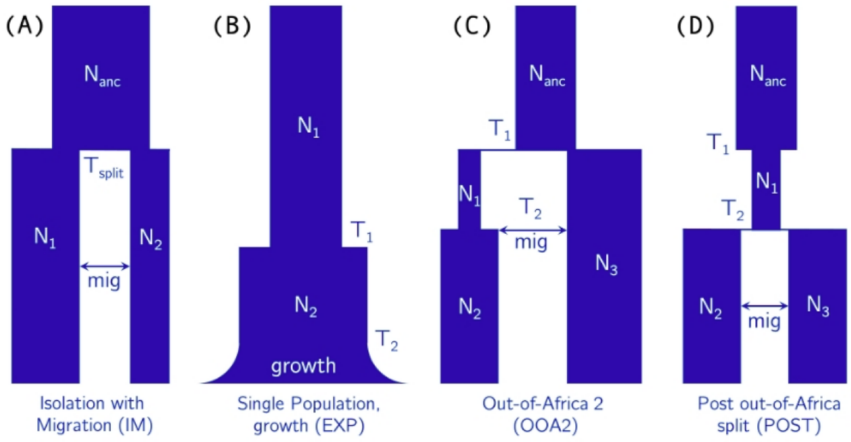


Figure 5: Four models studied and used in (Wang et al., 2021). Population genetics simulators require an evolutionary model to follow in producing simulated data, an additional layer that needs to be selected, along with genetic parameters, in order to simulate a real population. These four models are examples of models that can be used by msprime for simulating population. Model (A), Isolation with Migration (IM) represents a population that split into 2 populations at time T_{split}. Before the split, the size of the population is represented by the variable N_{anc}. The split resulted in two populations of size N_1 and N_2. Migration events took place between the two populations, which is represented by *mig*, the value of **migration rate**. Model (B) demonstrates a population that had a change in size event at time T_1 then an exponential growth event at time T_2. The size of the original population before the size change and after the size change are the parameters N_1 and N_2 respectively. The **growth rate** of the population after time T_2 is represented by *growth*. Model (C) is a more realistic model of population evolution originally presented in (Gutenkunst et al., 2009). The model houses three events, a split, a change of size, and migration. Model (D) is similar to Model (C) except in the fact that the change in size happens before the population splits.

literature review of every GAN evaluation metric that has been proposed. We aim to find better evaluation measures that work well with population genetic applications.

Our ultimate goal is to devise a new method based on the structure of PG-GAN, the novel approaches and distance measure of WGAN, and the exhaustive list of evaluation metrics of Borji's GAN evaluation metrics paper and tailor it to the needs of population genetics by making sure the new algorithm solves the problems of PG-GAN and provides a relevant metric that can easily measure how good the produced synthetic data is.

2 Literature Review

We present an overview of each of the papers presented in section 1.3. We then explain the algorithm and procedure in details. Finally, we summarize the advantages and disadvantages of the approach in hand and preview its applications in population genetics.

2.1 Generative Adversarial Nets

An Unsupervised Deep Learning Model for Generative Modeling.

2.1.1 Overview

A Generative Adversarial Net, as first proposed by Goodfellow et al., 2014, is a "a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G." In this approach, the training is defined as the procedure of teaching the generative of model to maximize the probability that the discriminative model misclassify. The authors represent the model as a minimax two-player game, with the generator and the discriminator as players. They claim that a unique solution exists. This solution, as put in (Goodfellow et al., 2014), is "G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere." In other words, the training process is successful when the generator is able to fool the discriminator rendering its predictions as good as a coin flip.

2.1.2 The Algorithm

In the design by Goodfellow et al., 2014, both the discriminator and the generator are multilayer perceptrons (i.e. neural networks). The generator is represented by $G(z;\theta_g)$, where G is the mapping represented by a neural network, z is the input noise variable with a prior $p_z(z)$ defined on it, and θ_g is the network parameters. The discriminator is defined similarly, $D(x;\theta_d)$, with x being the

real data. D's objective is simple: to maximize its accuracy, that is, correctly classify real and fake as much as possible. We define $D(x)$, the probability that x is real and we make our goal to maximize $\log D(x)$. That said, G's objective becomes to minimize the probability that D classifies the generated data, $G(z)$, as fake. In other words, G minimizes $\log(1 - D(G(z)))$. Now we can see how the algorithm boils down to the following minimax two-player game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \log D(x) + \log(1 - D(G(z))),$$

where x comes from the real data distribution $p_{data}(x)$ and z comes from $p_z(z)$.

The paper emphasizes the importance of the back and forth between the discriminator and the generator. If the discriminator is trained to completion first, the generator will not be able to learn anything, as all its attempts will be classified as fake and its loss function would be meaningless. If the generator learnt in a faster rate than the discriminator, the generator will be able to fool the discriminator with lower quality data, relying on the fact that the discriminator is not well-trained. Thus (Goodfellow et al., 2014) proposes an alternative approach: "we alternate between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly enough" The promise is that a unique solution does exist and it will be reached following these steps. Figure 6 formally presents this procedure.

2.1.3 Results & Advantages

GANs have led to huge leaps in the fields of machine learning and population genetics. GANs eliminated the need for Markov chains and inference during training, which were the limitations of previous generation methods like generative autoencoders. This opened the door for accurate data generation only using backpropagation and any differentiable function. A huge advantage in the context of population genetics is GAN's ability to generate data without using the training set examples as basis for the generation. As explained in (Goodfellow et al., 2014), "Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples, but only with gradients flowing through the discriminator. This means that components of the input are not copied directly into the generator's parameters." In other words, GANs can be our best hope in understanding the underlying structure of different population genetics data set distributions.

2.1.4 Disadvantages

GANs have two major problems: The absence of an explicit representation of $p_g(x)$, the generator's distribution over the data set, and the instability of the training process. The afore-mentioned back and forth between the generator

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Figure 6: Algorithm for the Generative Adversarial Net approach presented by (Goodfellow et al., 2014).

and the discriminator entails that the number of iterations each gets to run has to be chosen carefully chosen. Failing to do so can lead to **mode collapse**, a situation where the generator generates the same class of the real data too many times given different inputs. This mostly happens as a result of the generator learning some aspects of the real data very well so that it can always fool the discriminator by generating such aspects, eliminating the need to explore the entire probability space.

2.2 PG-GAN

Automatic inference of demographic parameters using generative adversarial networks.

2.2.1 Overview

The PG-GAN framework, as presented in (Wang et al., 2021), is "a parametric GAN framework that combines the ability to create realistic data with the interpretability that comes from an explicit model of evolution." The discriminator of a PG-GAN is a Convolutional Neural Network that takes an evolutionary mode and a genotype matrix as input, where a genotype matrix is a computational representation of the genome of individuals. As with classic GANs, the discriminator's job is to classify its input as real or fake. While the discriminator

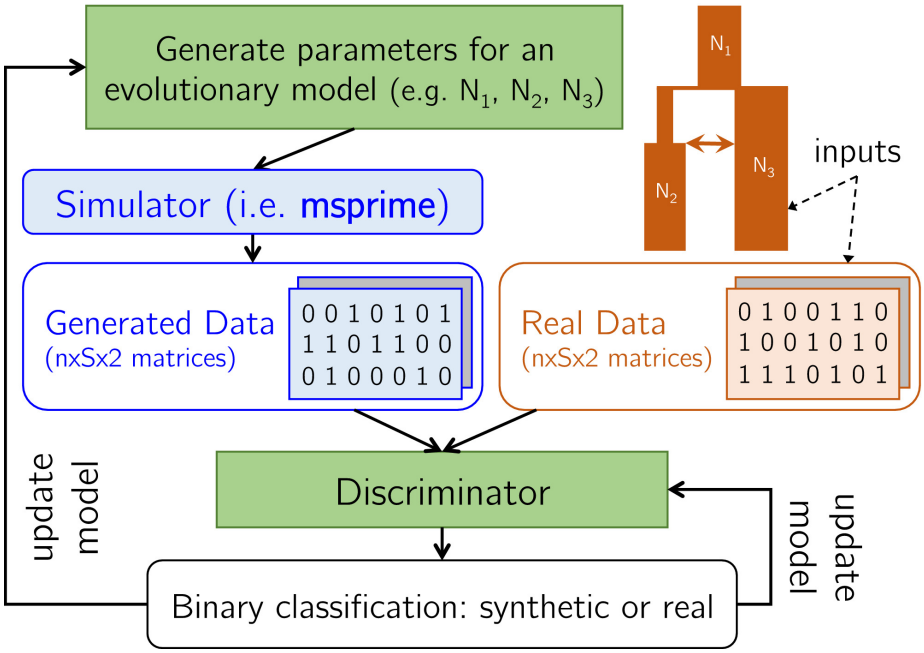


Figure 7: An overview of the pg-gan algorithm presented in (Wang et al., 2021). The inputs to the algorithm are the genotype matrix and an evolutionary model (refer to 5 for a description of the model in this figure.) The generator passes its generated genetic parameters to msprime, which generates the population that is then translated into a genotype matrix to be classified by the discriminator. Both the discriminator and the generator are updated in an alternating fashion, as suggested by (Goodfellow et al., 2014).

of a PG-GAN aligns well with that from a normal GAN, the generator of PG-GAN is entirely different. PG-GAN uses msprime, a "coalescent simulator that generates genotypes data from a parametrized demographic history." While the discriminator is trained using **gradient descent**, a standard approach in GAN discriminators, the generator uses **simulated annealing**, a probabilistic technique used to approximate the global optimum of a given function. This is crucial because while gradient descent does the same thing, it doesn't work for non-continuous functions, which the generator here is. The promise is that PG-GAN can train both the generator and the discriminator successfully so that the generator is able to produce populations of the same genomic composition as the real data. The outputs of PG-GAN are the evolutionary parameters at the end of the training process, as these parameters are the most successful in fooling the well-trained discriminator. An illustration of the PG-GAN approach is shown in figure 7.

2.2.2 The Algorithm

The PG-GAN approach simplifies the minimax two-player game idea presented in (Goodfellow et al., 2014) by presenting two separate loss functions that needs to be minimized, one for the generator and one for the discriminator. The generator loss function is

$$L_G(\theta) = -\frac{1}{M}\sum_{m=1}^{M} \log D(z^{(m)}),$$

where M is the number of regions of simulated data $z^1, \cdots, z^M$, and θ is the input parameters of the generator.

With real data $X = x^1, \cdots, x^M$, the discriminator loss function is

$$L_D(\theta, X) = -\frac{1}{M}\sum_{m=1}^{M} [\log D(x^{(m)}) + \log(1 - D(z^m))]$$

This approach leads to a similar setup as in the minimax game, only that both players are trying to minimize opposite values. Since this can map to a minimax setup, it gains the reassurance that a unique solution exists, which is proved in (Goodfellow et al., 2014). The PG-GAN algorithm is demonstrated in figure 8.

2.2.3 Evaluation

One of the biggests issues of Generative Adversarial Newtorks is the lack of a natural, informative evaluation metric. It's especially difficult when the data being denerated cannot be evaluated qualitatively. For example, a GAN generating images of a car can be evaluated by any human who knows what a car is (though this method end up being biased towards generators that memorize specific data examples and use them without exploring the entire space, see (Borji, 2019)). This makes evaluating the performance of PG-GAN a real challenge. To overcome this obstacle, (Wang et al., 2021) resorted to visualizing **summary statistics**, a way to visualize the data simply and quickly using a few chosen measures. In PG-GAN (Wang et al., 2021), seven types of summary statistics were used: Site Frequency Spectrum, Inter-SNP distancces, Linkage Equilibrium, Pairwise Heterozygosity, Tahima's D, Number of Haplotypes, and Hudson's F_{st}. The problem is that these summary statistics are not inclusive to everything the model can learn from the raw data; in other words, an excellent model wouldn't necessarily have much better summary statistics than a good one, but could potentially have much better results in tens of different summary statistics that are either not computed for PG-GAN or we don't even know anything about them or how to compute them at all. An example of the summary statistics of a good PG-GAN model is shown in figure 9. As we mentioned, the close matching of the training data and simulated data curves is not conclusive

Algorithm 1: Training `pg-gan`.

Input: evolutionary model parametrized by Θ, real data X, architectures of generator G and discriminator D
Output: optimal parameters Θ^* (which can be fed into G to produce synthetic data), trained discriminator D

for *num pre-training iterations* **do**
  Initialize evolutionary parameters randomly to Θ
  Train discriminator using real data and simulated data under $G(\Theta)$
end

Initialize $\Theta^{(0)}$ to the parameters that caused the highest accuracy during pre-training
for *each training iteration* i **do**
  Use simulated annealing to select parameters Θ near $\Theta^{(i)}$ that minimize the generator loss

$$\mathcal{L}_G(\Theta) = -\frac{1}{M}\sum_{m=1}^{M} \log D(z^{(m)}),$$

  where $\{z^{(1)}, \cdots, z^{(M)}\}$ is a mini-batch of data simulated under $G(\Theta)$

  if *best parameters* Θ *are accepted* **then**
    for *num mini-batches* **do**
      Sample mini-batch of M regions $X = \{x^{(1)}, \cdots, x^{(M)}\}$ from the real data
      Simulate mini-batch of M regions $\{z^{(1)}, \cdots, z^{(M)}\}$ under $G(\Theta)$
      Update the discriminator by minimizing binary cross-entropy loss (below) using gradient descent

$$\mathcal{L}_D(\Theta, X) = -\frac{1}{M}\sum_{m=1}^{M} \Big[\log D(x^{(m)}) + \log(1 - D(z^{(m)}))\Big]$$

    end
    Set $\Theta^{(i+1)} \leftarrow \Theta$
  end
end
Run the above procedure K times; select the parameters that minimize discriminator accuracy on the simulations.

Figure 8: Algorithm for the pg-gan approach presented by (Wang et al., 2021).

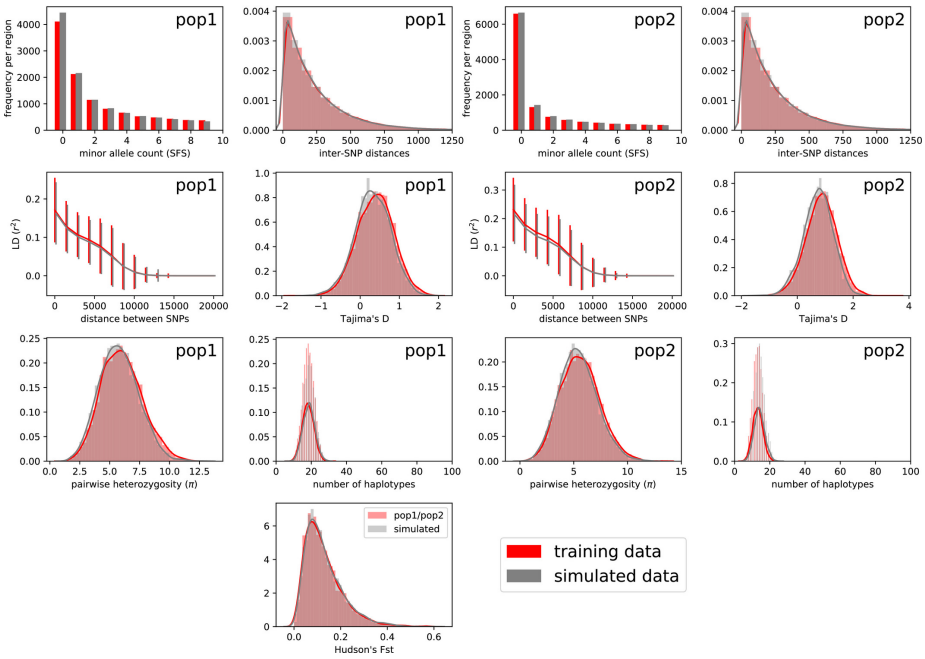


Figure 9: Summary statistics of an IM model trained by (Wang et al., 2021). The model is trained on two populations, hence the presence of two graphs for each type of summary statistics mentioned in 2.2.3, except that for Hudson's F_{st}, which is computed on two populations. As the figure illustrates, the trained and simulated data match closely in almost all summary statistics, giving us empirical evidence that the model that created this data is a well-trained model.

evidence that the model that created the simulated data is an optimal model, but it's enough proof that the model is good enough to produce data that is at least similar in structure to the original data.

2.2.4 Results & Advantages

PG-GAN presents a very solid holistic approach for automatically inferring genetic parameters that can be used for simulating any population. One big advantage of PG-GAN is introducing a natural way of refining simulation architectures, as PG-GAN have no trouble reaching 50% discriminator accuracy and producing a well-trained simulation when the input training data is itself simulated; in other words, PG-GAN can be used with simulated data (that we know exactly the set of genetic parameters that produced such data) so that the quality of the simulation pipeline can be evaluated.

Another strong pro of PG-GAN is that it has created a solid structure through which other evolutionary models, summary statistics, and even generator and simulator pipelines can be experimented with. For example, geneticists can

create evolutionary models that are tailored for the populations they are researching and can use PG-GAN on their new model to simulate it. This is huge because many populations, especially ones from different species, may not fit any of the evolutionary models of PG-GAN, thus the benefit of being able to add more evolutionary models to the pool of models the generator can choose from.

2.2.5 Disadvantages

While PG-GAN can produce perfect fits on simulating simulated data, on real data the fit is not always perfect. As explained in (Wang et al., 2021), "this could be because there are features of the real data that our models do not include, for example false negatives and other genotyping errors, phasing errors, missing data and inaccessible regions of the genome." Fortunately, these effects can be incorporated in the generator and accounted for. Another problem is what we mentioned in 2.2.3 about the summary statistics not being conclusive. This was an issue with classical GANs, as well as PG-GAN. Although summary statistics is a well-grounded qualitative approach, only a natural quantitative approach can be conclusive about the quality of the model.

2.3 WGAN

An alternative to traditional GAN training with stable learning, no mode collapse, and meaningful learning curves.

2.3.1 Overview

Before delving deeper into what the framework of WGANs look like, we first carefully look into a novel distance metirc presented by (Arjovsky et al., 2017), the *Earth-Mover* (EM) distance. To understand the EM distance, we first define $\mathbb{P}_r, \mathbb{P}_g \in \text{Prob}(X)$, where X is a compact metric set and $\text{Prob}(X)$ denotes the space of probability measures defined on X. We denote the set of all joint distributions $\gamma(x, y)$ with $\mathbb{P}_r$ and $\mathbb{P}_g$ marginals as $\Pi(\mathbb{P}_r, \mathbb{P}_g)$. The Earth-Mover distance, $W(\mathbb{P}_r, \mathbb{P}_g)$ is defined as follows:

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

where $\mathbb{E}$ signifies the Euclidean Space. The intuitive definition of this distance is well-explained in (Arjovsky et al., 2017): "$\gamma(x, y)$ indicates how much 'mass' must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into the distribution $\mathbb{P}_g$. The EM distance then is the 'cost' of the optimal transport plan"

2.3.2 The Algorithm

WGAN utilizes the fact that the EM distance is continuous and differentiable to train the critic (another name for the discriminator inroduce in (Arjovsky et al., 2017)) to optimality. But to do so, the infimum in the EM equation has to be computed, which is a very difficult task, as the infimumis highly intractable. Instead, the paper benefits from the Kantorovich-Rubinstein duality (Villani,2009) to conclude that computing the EM distance boils down to computing

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z))],$$

where $\mathbb{P}_r$ is any distribution, $\mathbb{P}_\theta$ is the distribution of $g_\theta(Z)$ with g a function that follows assumption 1 for interchanging limits and integrals (see (Dobelman, 2012)) and Z a random variable with density p. f is computed by solving the following maximization problem:

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{x\sim \mathbb{P}_\theta[f(x)]},$$

where $\|f\|_L \leq 1$ represents all the 1-Lipschitz functions. The transformation from having to compute the infimum into the simpler problem above is proved in Appendix C of (Arjovsky et al., 2017).

The WGAN approach is to use a neural network to optimize the max problem of finding f and using it to minimize the Earth-Mover distance, allowing the real and generated probability distributions to be as close as possible. Thw WGAN algorithm is demonstrated in figure 10.

2.3.3 Evaluation

WGANs provide a natural way of evaluating and visualizing the quality of the generator. Due to the fact that the EM distance is differentiable almost everywhere, reliable gradient information is always available. This means that the Wasserstein loss function, an estimate of the Earth-Mover distance, will correlate well with the *quality* of the generated samples. Figure 11 shows how accurate this correlation is on three experiments with three different generators.

2.3.4 Results & Advantages

WGAN shows amazing results compared to other generative approaches. We can summarize the advantages of WGANs in three main points:

1. **Meaningful loss metric** As mentioned in 2.3.3, the quality of the generated data can be quantitatively estimated using the loss curve of WGAN. This can be helpful for population genetics since, unlike images or text, the generated data cannot be visualized and assessed by humans. This feature is thanks to the persistent gradients of WGAN that, unlike those of regular GANs, don't face the problem of **vanishing gradients**, where

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of priors.
    $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Figure 10: Algorithm for the WGAN approach presented by (Arjovsky et al., 2017).

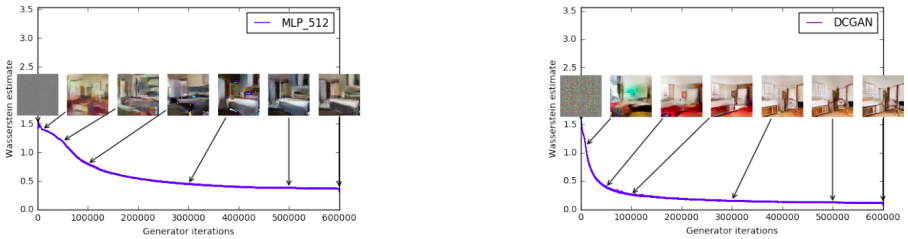


Figure 11: The figure shows the Wasserstein estimate training curve and sample generated images at different stages of training. It's clear that the quality of the generated image is inversely related to the Wasserstein estimate (loss). In other words, as the loss decreases, the quality of the generated data increases. This figure is adapted from (Arjovsky et al., 2017).

gradients of the loss function vanish (approach zero) and thus become maeningless to the learning process. Figure 12 compares a regular GAN discriminator and a WGAN critic and their gradients.

2. **Improved stability** Another benefit of WGANs is the ability to train the critic to optimality. This means that we no longer need to balance the training of the critic and the generator in alternating steps. This makes the relationship between the quality of training of the critic and the generator as simple as it gets: as put in (Arjovsky et al., 2017), "the better the critic, the higher quality the gradients we use to train the generator."

3. **Eliminating mode collapse** This is an immediate result of the improve stability of WGANs, as mode collapse is a result of an instability in the learning process. (Arjovsky et al., 2017) presents an interesting experiment (see figure 13) to show that WGANs don't suffer from mode collapse. They also claim to have never see evidence of mode collapse in all of their experiments with the WGAN algorithm.

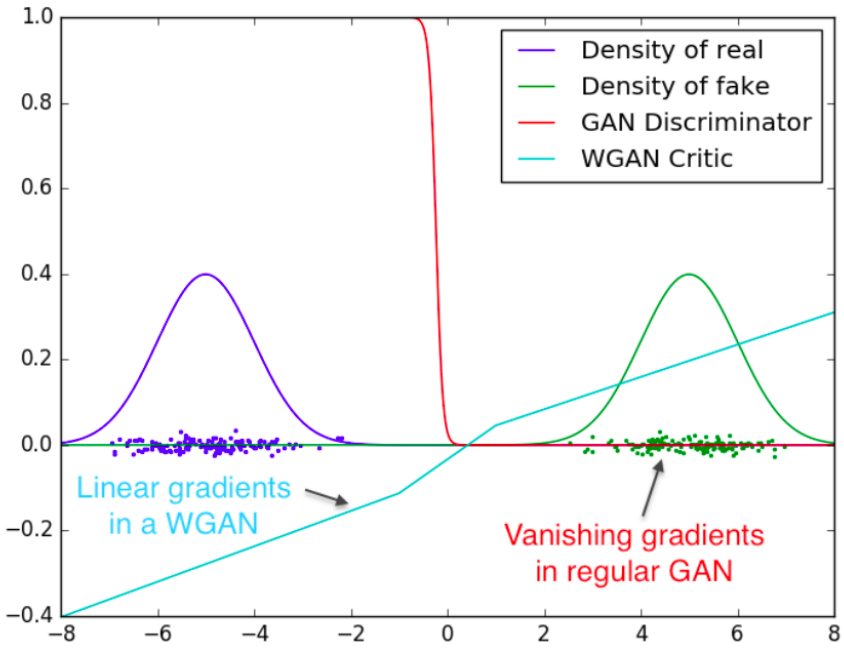


Figure 12: The figure, originally viewed in (Arjovsky et al., 2017), depicts two optimal GAN discriminator, with one being a normal GAN discriminator and one being a WGAN critic. The figure shows that the gradient of the normal GAN discriminator end up vanishing (becoming zero), rendering them useless for generator training. On the other hand, the gradients of WGAN are linear and stay informative on all parts of the space.

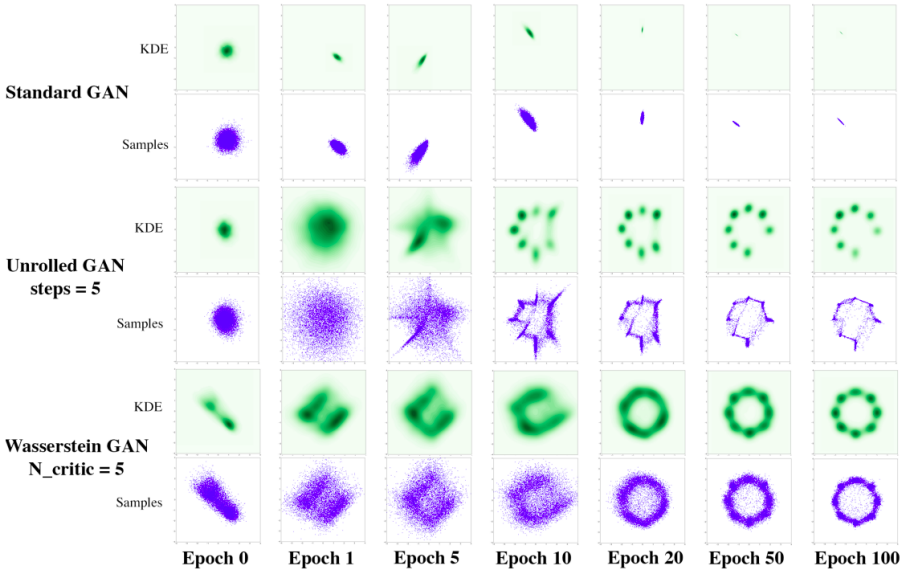


Figure 13: Comparison between Standard GAN, Unrolled GAN, and WGAN presented in (Arjovsky et al., 2017). Seven samples and their corresponding Kernel Distribution Estimates are provided for each GAN method where the seven samples correspond to epochs 0, 1, 5, 10, 20, 50, and 100. The figure shows that WGAN captures the low dimensional structure of the data before matching the higher-dimensional details and is able to learn the distribution without mode collapse.

3 Methods

3.1 Overview

We propose a recreation of PG-GAN, the algorithm in (Wang et al., 2021), using the WGAN technique presented in (Arjovsky et al., 2017). We hope to produce learning curves that closely relate the loss of the model to the quality of the generated data. We aim to verify this relationship with a fast, automatic evaluation metric that eliminates the uncertainty that stems from plotting summary statistics of the real and generated data and comparing them manually.

Theoretically, WGAN provides solutions to all of the problems PG-GAN is facing, the biggest of which being visualizing and evaluating the quality of the generated data. WGANs solve this problem by introducing the Earth-Mover distance that allows the training curves to correlate to the distance between the probability distributions of the raw data and the generated data. Moreover, tuning the parameters of PG-GAN wouldn't be a problem anymore if the Wasserstein critic is used instead of PG-GAN's discriminator.

3.2 Procedure

In transforming the GAN algorithm of (Wang et al., 2021) into wgan, we follow 5 principal steps introduced in (Brownlee, 2021). These 5 steps are described in the subsections below.

3.2.1 Linear Activation

One aspect where WGANs differ from normal GANs is that the critic of a WGAN doesn't just predict if data is real or fake; instead, it scores that data based on realness. That is, WGAN critics can't use a sigmoid function in the output layer of their architecture, which is the case for normal GANs. Instead, a linear activation function is used to give information on not only whether data is real, but also how much does it score on a fake-to-real scale.
If we are using the Model class of Keras, we would have a last layer that looks something like this:

```
model.add(Dense(1, activation='linear'))
```

3.2.2 Opposite Sign Labels

GAN discriminators use the class 0 to represent labels of fake data and the class 1 to represent labels of real data. This poses a problem for the WGAN critic, as it doesn't classify data in such way. While GAN discriminators can only output 1 or 0 for any given example, WGAN critics are expected to output a score of realness, not a class or a specific label. If that range were to be from 0 to 1, it would be difficult to infer which inputs are real and which are fake. Instead, WGANs use $+1$ as labels for real data and -1 as labels for generated

ones. This way, the critic can be encouraged to score real data on the positive side and fake data on the negative side, making it easier to distinguish between the two classes. A well-trained critic is not expected to output $+1$ for real data and -1 for fake ones; instead, it's expected to *separate* the predictions for the two classes as much as possible. For example, a critic that scores a real example as 5.2 and a fake one as -4.9 is doing well, as it is maximizing the distance between the two classes. With this in place, the focus of the WGAN critic can shift from outputting precise labels to just assigning scores that are different for real and fake data.

To implement this change, we simply assign real data samples labels of $+1$ and fake data samples labels of -1. The way WGAN utilizes these opposite sign labels is described in more details in the Wasserstein loss procedure below.

3.2.3 Wasserstein Loss

The chief contribution of (Arjovsky et al., 2017) is the novel Wasserstein loss function that transforms a normal GAN discriminator, one that is only capable of binary classification, into a critic, one that can score data on a scale from -1 to 1, -1 being real (with great confidence) and 1 being fake, also with great confidence. Scores between -1 and 1 determine how fake the data is. Although this is theoretically correct, encouraging the critic to score data beyond these values has the practical benefit of making the separation of the data more obvious. The idea is to separate the real and fake data as much as possible, where most of the fake data scores is on the negative side of the scale, and most of the real data scores is on the positive side.

To incorporate the Wasserstein loss function in PG-GAN, we need to implement the loss function in Keras and use it. An example of how that can be done is illustrated in (Brownlee, 2021):

```
from keras import backend

def wasserstein_loss(y_true, y_pred):
        return backend.mean(y_true * y_pred)
```

Notice that we multiply by y_true, the class label of the input, to make sure the loss is minimizing even for real examples. Now that the loss function is defined, it can be simply used in the compile method of the Keras model module:

```
model.compile(loss=wasserstein_loss, ...)
```

For PG-GAN, the critic is trained on real data and fake data separately. The Wasserstein loss function has to be modified to account for that. The implementation of wasserstein_loss in PG-GAN is shown below:

```
def wasserstein_loss(real_output, fake_output):
    real_loss = backend.mean(real_output)
```

```
    fake_loss = backend.mean(fake_output)
    total_loss = - real_loss + fake_loss
    return total_loss
```

In the implementation above, the real loss is negated in computing the total loss, encouraging the critic to *maximize* the real loss in order to minimize the loss. The only way the critic can maximize the real loss is by scoring real data with large positive values, which is our goal. In contrast, the fake loss is not negated, and thus need to be *minimized* in order to achieve a lower total loss. This encourages the critic to give fake data large negative scores, minimizing the loss and maximizing the separation between the two classes.

In order to engage the generator in this minimax game, the loss function of the generator has to be modified to maximize the realness score of the generated data. In other words, the generator aims to get scored as real data does; that is, with large positive values. Thus, the loss of the generator is simply defined as the negation of the score given by the critic to the fake data generated by the generator. If this score is large, the generator loss is the negation of that large number and therefore is a small number. This adversarial nature allows the generator to learn from the scores of the critic which data samples look more real and thus improve the quality of the data.

To implement the generator loss function in PG-GAN, we generate a batch of fake data and give it to the critic for prediction. We use the score returned by the critic as an indicator of realness. The negation of that score defines our loss. The implementation in PG-GAN is presented below:

```
def generator_loss(generated_regions):
    # not training when we use the critic here
    fake_output = critic(generated_regions, training=False)
    generator_loss = -backend.mean(fake_output)
    return generator_loss
```

3.2.4 Weight Clipping

Weight clipping is a way to combat **exploding gradients**, a phenomenon where gradients get too large so that they stop being meaningful for the learning process. Weight clipping is simple: weights that get past a predefined constraint gets *clipped* back to that constraint and is never allowed to increase or decrease past it. To implement this idea and integrate it with PG-GAN, we can use the Keras *constraint* class, as suggested by (Brownlee, 2021). The idea is to define a class that *extends* the Keras constraint class and implements its __call__ and get_config methods, which are used for applying the clipping operation and returning the clipping factor, respectively. An implementation of this class would look something like:

```
class ClipConstraint(Constraint):
```

```
        # set clip value when initialized
        def __init__(self, clip_value):
                self.clip_value = clip_value

        # clip model weights
        def __call__(self, weights):
                return backend.clip(weights,
    -self.clip_value, self.clip_value)

        # get the config
        def get_config(self):
                return {'clip_value': self.clip_value}
```

With this in place, an instance of this class can be initialized with the desired clipping factor, say 0.01, and we can use that object as the value for the built-in kernel_constraint parameter of Keras' convolutional layer API. For example:

```
constraint = ClipConstraint(0.01)
...
# use the constraint object in a layer
model.add(Conv2D(..., kernel_constraint=constraint))
```

3.2.5 Update Critic more than Generator

A crucial difference between WGAN and PG-GAN is the strive for balance between the discriminator and generator training. In PG-GAN, choosing the number of iterations per network (discriminator/generator) can be a delicate process that involves trial and error. In WGAN, though, it is much simpler: the critic (discriminator) has to be updated more times than the generator. This is mainly because of the nature of the critic and how it scores input data, as increasing the number of iterations of the critic per one generator iteration can only aid the learning process, while doing the same with a normal GAN usually lead to the discriminator being more advanced in classifying fake data, leaving the generator no choice but to fail.
Implementing this change requires creating a nested loop for updating the critic in a main loop for updating both the critic and the generator. With *n_steps* as the number of iterations of the generator and *n_critic* the number of critic iterations per one generator iteration, the code should look something like this:

```
# main gan training loop
for i in range(n_steps):
        # update the critic
        for _ in range(n_critic):
                # code to update critic
        # code to update generator
```

3.3 Proposed Algorithm

We propose an adaptation of WGAN for implementing the PG-GAN algorithm. In particular, a Wasserstein critic will replace the GAN discriminator of PG-GAN and the Wasserstein estimate will be used as the loss function. We deem this procedure the Wasserstein PG-GAN. The procedure is described in Algorithm 1.

Algorithm 1 Wasserstein PG-GAN in the style of (Arjovsky et al., 2017).

Require: α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: w_0, initial critic parameters, θ_0, initial generator parameters.
Require: evolutionary model parametrized by θ. X, real data. Architectures of generator G and critic C.
  while θ has not converged **do**
    for $t = 0, \cdots, n_{critic}$ **do**
      Sample ${x^{(i)}}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
      Sample ${z^{(i)}}_{i=1}^{m} \sim p(z)$ a batch of priors.
      $L_C(w, X) \leftarrow \nabla_w[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$
      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
      $w \leftarrow \text{clip}(w, -c, c)$
    end for
    Sample ${z^{(1)}}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $L_G(\theta) \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
    $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
  end while

4 Experiments

In order to assess the performance of the WGAN implementation of PG-GAN, we introduce the following experiments, with different hyperparameters, model architectures, and gradient control techniques. For all four experiments, the generator architecture is the same, with MSPrime and deconvolutional layers. Experiment 1 will be used as a reference to indicate improvement/deterioration of performance for the other experiments, as it represents the original implementation of WGAN. The core idea of the experiment is as follows:
Since we don't know the evolutionary parameters (ground truth) of real data, we will not be able to evaluate the performance of the algorithm if we experiment on real data. Instead, we use MSPrime to create a database of fake data that we will use as our real database. We choose an isolation-with-migration two-population model to be simulated with the following parameters:

- Recombination Rate (reco): 1.25^{-8}

- Ancestor Population (N_anc): 15000
- Time of Population Split (T_split): 2000
- Migration Rate (mig): 0.05
- Population 1: 9000
- Population 2: 5000

MSPrime takes these parameters as input and outputs a database of simulated DNA data of individuals that are supposed to mimic that of individuals who experienced the same evolutionary parameters. We use this database to train the critic. This allows us to have a very clear measure of performance for this experiment, as the goal of the algorithm becomes learning the parameters above form the simulated data.

4.1 Experiment 1: Gradient Clipping

For this experiment we use the original implementation of WGAN with gradient clipping using a clipping constraint. The implementation is simple; when ever the magnitude of the critic gradient gets larger than a value (0.01 in our case), we clip it back to that value. This forces the critic function to be a **1-Lipschitz** function, a function with a bounded derivative, which is essential to the completion of the proof that WGAN converges (Arjovsky et al., 2017).
Since we are using simulated annealing for the generator, we propose 10 different values for each parameter that we want to infer. This leaves us with 60 different proposals. The generator *learns* which parameter values are better through the score given by the discriminator. For this experiment we use an n_critic multiplier of 5 as suggested by (Arjovsky et al., 2017). This means that, for each generator iteration, we train the critic for 5 iterations. This experiment assumes that one generator iteration packs as much training as 60 epochs (since we propose 60 different sets of parameters). Therefore, we use an n_critic of 300.

4.2 Experiment 2: Gradient Penalty

This experiment uses an alternative to gradient clipping. As shown by (Gulrajani, Ahmed, Arjovsky, Dumoulin, & Courville, 2017), "Sometimes [WGANs] can still generate only poor samples or fail to converge. We find that these problems are often due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to undesired behavior." The paper proposes enforcing the Lipschitz constraint by penalizing the norm of the gradient with respect to the critic's input. In other words, if the gradient gets out of bound, the loss of the critic increases, forcing the critic to bound its own gradient in order to reduce the loss. We use gradient penalty instead of gradient clipping for this experiment.

4.3 Experiment 3: Layer Normalization

This experiment is similar to Experiment 2, except that **layer normalization** is used. Layer normalization normalizes the distribution of the critic layers, leading to less expensive computations per layer and smoother gradients.

4.4 Experiment 4: Less Critic Epochs

As explained in experiment 1, we chose n_critic to be 5 $\times$ generator proposals; that is 300. We experiment with an n_critic of just 5 epochs. Although the generator proposes 60 different sets of parameters, it only learns the one with the smallest loss. This means that an n_{critic} of 300 might be way off. We experiment to try to find a balance between how much each network learns. We increase the number of total iterations (generator epochs) to 20000 (instead of 1000), to account for the small number of critic epochs per iteration.

5 Results

In this section, we view the results of each experiment proposed above. In addition to graphing the generators predictions of the 6 parameters over the epochs, we also plot the loss of both the critic and the generator. The critic loss is separated into two parts, fake loss and real loss. Fake critic loss is the loss of the critic in evaluating generated data. Real critic loss is the loss of the critic in evaluating real data, the data generated with the specified parameters described in section 4.

5.1 Experiment 1 results

The results of this experiment show the inefficiency of gradient clipping for deeper networks. (Gulrajani et al., 2017) observed that deep critics fail to converge under gradient clipping and the results are inconsistent. This is clearly the case here. As shown in figure 14, The generator infers none of the parameters correctly. In some cases, the parameter choice is trending away from the target value.

In addition, figure 15 shows the failure in distribution separation between real and generated data. Observe that the critic real loss is just the average score given by the critic to real data. Similarly, the critic fake loss is the average score given by the critic to fake data. This shows that a perfect implementation will separate those values as much as possible; thus increasing the discrimination between real and fake values and reducing the critic's loss. We can clearly see from figure 15 that this distinction is not present, as the two distributions overlap and fluctuate at the end. We can also see the exploding gradients problem showing up, as the values for fake and real critic loss soar to a whopping 10^9. This occurs because of interactions between the clipping constraint and the loss

function, as explained by (Gulrajani et al., 2017)

The generator loss also exhibits exploding gradients and is not showing a downward trend, which signifies that the generator might not be learning as much as possible from the process, since it's loss is not generally decreasing.

5.2 Experiment 2 results

This experiment uses gradient penalty and is expected to eliminate the exploding gradients problem from the previous experiment. However, the stability of the learning process is not guaranteed, as hyperparameter choice plays a role. As shown in figure 16, the results of the experiment are slightly better than the previous one, with evolutionary parameter predictions closer to the target values and trends mainly towards the targets. However, figure 17 shows that there is no real separation between the real and fake distributions, meaning that convergence of the generator is unattainable.

5.3 Experiment 3 results

This is the same as experiment 2 except with layer normalization. The only difference observed in results is smoother gradients and a better downward trend in generator loss (shown in figure 19). The critic loss still shows sign altering, which means that the critic keeps changing the signs of the scores given to real and fake samples. This definitely confuses the generator and doesn't allow it to learn in a consistent manner. Figure 18 confirms that, as the proposed values are still not close to the target one.

5.4 Experiment 4 results

This experiment resulted in perfect critic separation between real and fake data, as shown by figure 21. This is directly related to the choice of n_{critic} as 5 and not 300. Less critic epochs led to more balance in the learning process. When the critic had much more epochs to learn before the generator, it has the choice to switch the scores of fake and real data while achieving the same loss. With less epochs per generator iteration, the generator learned more about the data and produced more challenging fake data, forcing the critic to be consistent as well.

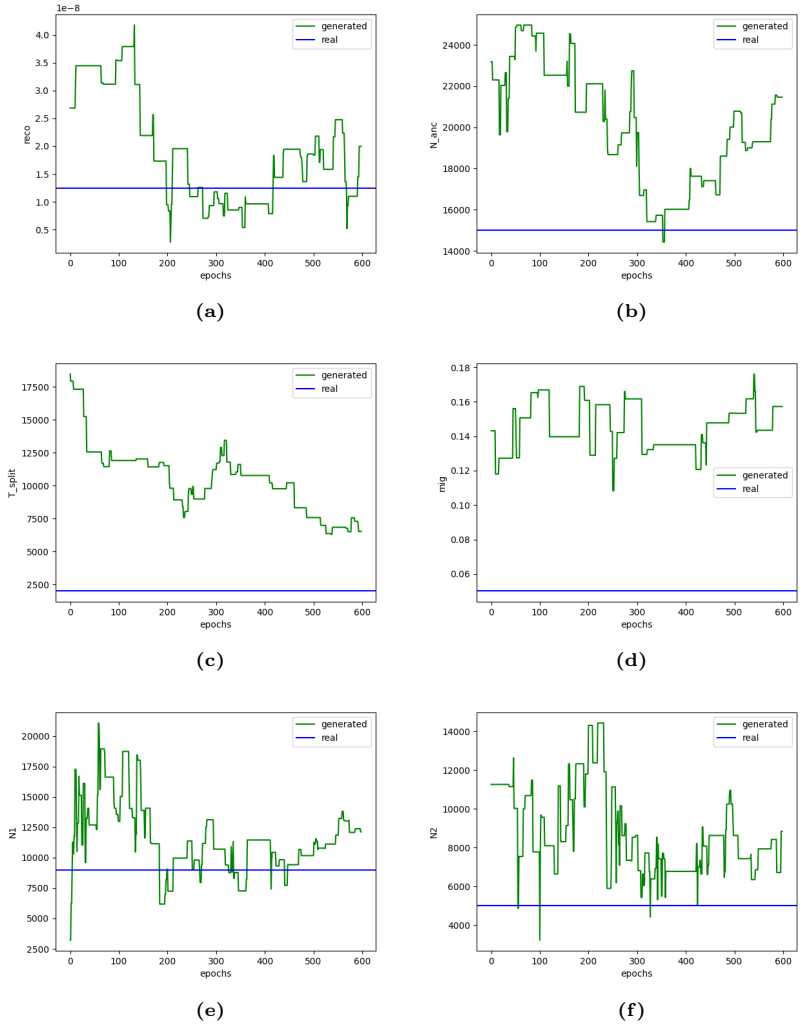
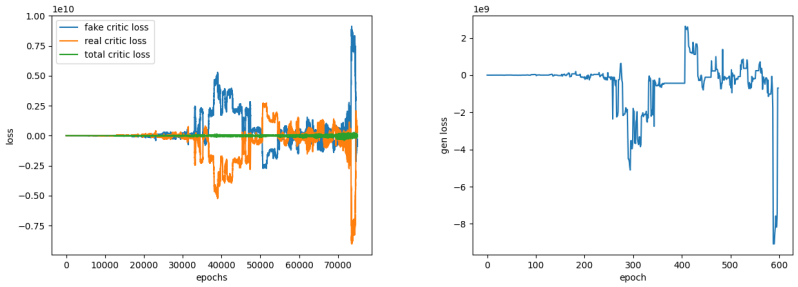


Figure 14: Experiment 1: Parameter values chosen by the generator over the epochs.



(a) (b)

Figure 15: Experiment 1: Critic and Generator loss. (a) The figure shows real, fake, and total critic loss over the epochs. The real loss (orange line) is fluctuating and changing signs, showing a lack of consistency. The fake loss is similar. There is no separation between the scores from real and fake samples, showing that the critic is not well-trained. Exploding gradients are clearly present. (b) Generator loss is fluctuating around 0, with an anomaly near the 600^{th} epoch.

6 Discussion

Based on the results from the 4 experiments, we conclude that experiment 4 resulted in the best performance and discuss the advantages and disadvantages of the algorithm regarding the following aspects:

6.1 The stability of the learning process

- The clear separation between real and fake data shows the consistency of the learning process and the ability to optimize the generator network
- In terms of the stability of the learning process, this is definitely an area of improvement. The algorithm shows almost the same trends in terms of critic loss on fake data, critic loss on real data, and generator loss at each run with the same parameters. Put simply, running the same model multiple times guarantees similar learning curves.
- We ran experiment 4 several times and it showed similar results every time, proving the consistency of the WGAN approach and how it prevents mode collapse.

6.2 The relation between the learning curves and the quality of the generated data

- We are able to look at the critic loss graphs and determine if the model is well-trained or not, which is not a feature of normal GANs. This allows us to quantitatively assess the quality of the generated data, which is very helpful in the field of population genetics where the generated data can neither be assessed quantitatively nor qualitatively.

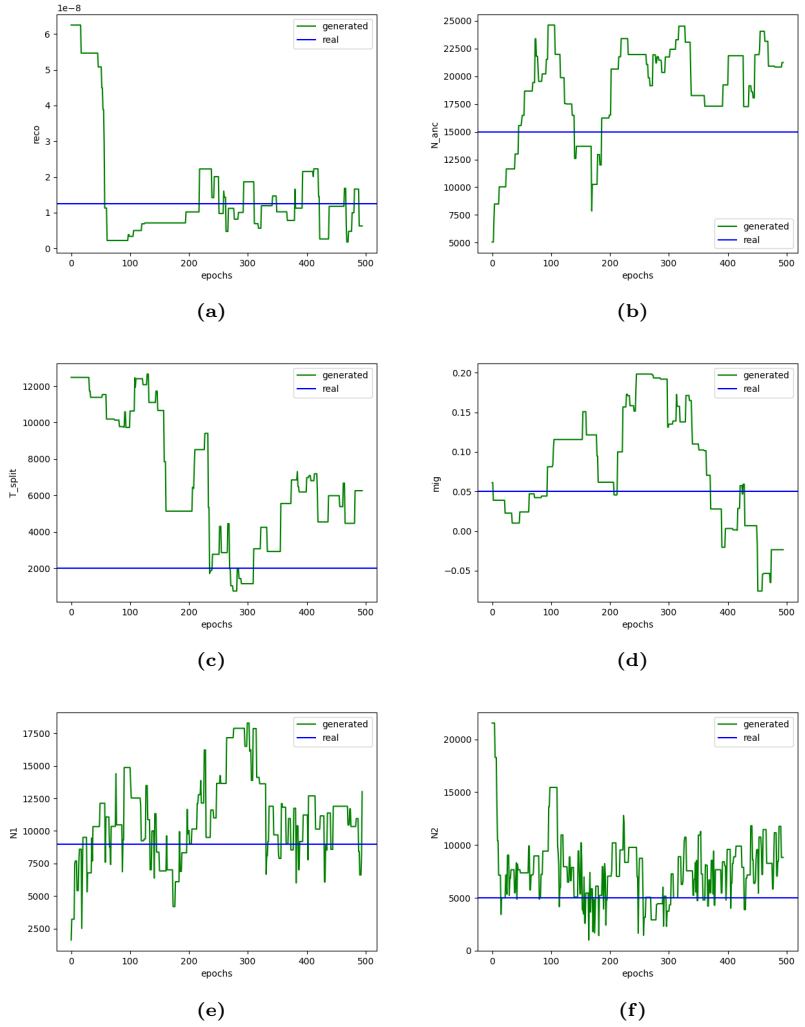
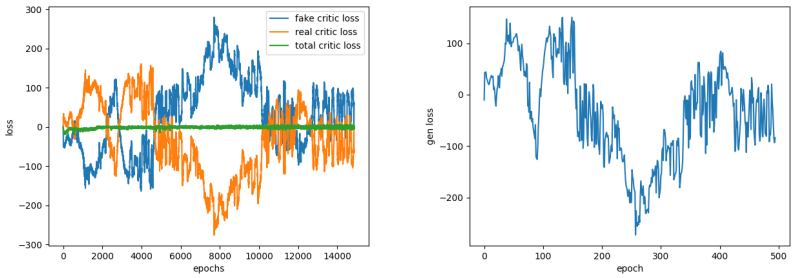


Figure 16: Experiment 2: Parameter values chosen by the generator over the epochs.



(a) **(b)**

Figure 17: Experiment 2: Critic and Generator loss. (a) Real and fake critic loss keeps altering signs, showing uncertainty and lack of discrimination between the two distributions. Gradients don't explode. (b) Generator loss has a downward trend followed by an upward one. This is because of the instability of the critic.

- Generator loss can also be used to assess the quality of the generated data. The smaller the generator loss is, the better the results.

6.3 How parameter choice affects the learning process

- Per each generator iteration, the critic has to take enough time to *completely learn* the fake data presented by the generator. This means that choosing the critic multiplier poorly **will** result in poor performance. With a low critic multiplier, the generator will easily fool the critic and the critic will never get the chance to learn the data that fooled it. With a high critic multiplier, the process will take much more time than needed, adding to what is already a very lengthy process and will eventually confuse the generator and lead to a failed run.

- WGANs are evidently less prone to failing due to the choice of parameters. We saw that the choice of *n_critic* was crucial to the quality of generated data. This is because it was way off before (60 times more than what it should be). However, small changes to *n_critic* didn't affect the quality of learning.

- Other hyperparameter changes, like number of generator proposals and batch size, didn't drastically affect the process, leading us to believe the consistency and stability of the WGAN approach.

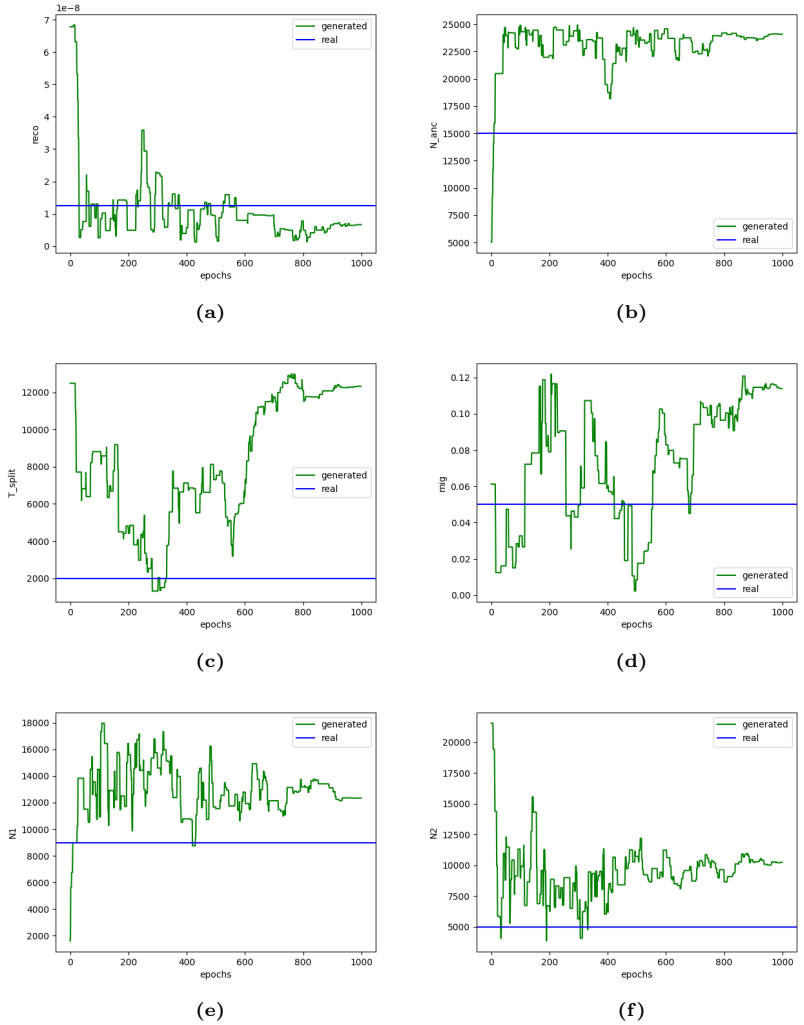
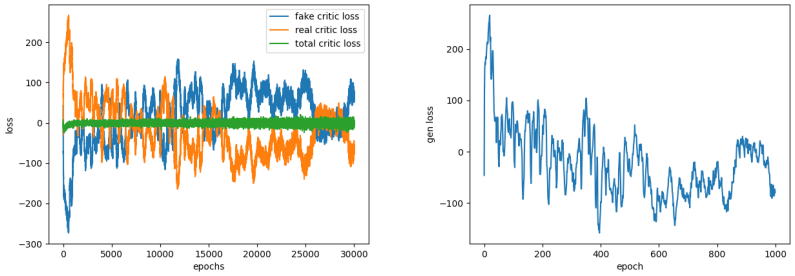


Figure 18: Experiment 3: Parameter values chosen by the generator over the epochs. We observe the smoother gradients because of layer normalization



(a) **(b)**

Figure 19: Experiment 3: Critic and Generator loss. (a) The figure shows similar results to experiment 2, where the real and fake distributions are not separated by the critic. No exploding gradients. (b) Generator loss is trending trending downwards.

7 Conclusions and future work

We conclude that a WGAN approach to PG-GAN provides a reliable way to visualize and assess the performance of generative networks in population genetics. The critic's continuous gradients prevents vanishing gradients and gradient penalty prevents exploding gradients. With a good balance between the generator and the critic, the algorithm proves to be stable and reliable in generating data that closely mimic the real sample. The approach admits the following extensions:

1. Applying layer normalization to the critic architecture with the parameters of experiment 4

2. running the algorithm for more iterations.

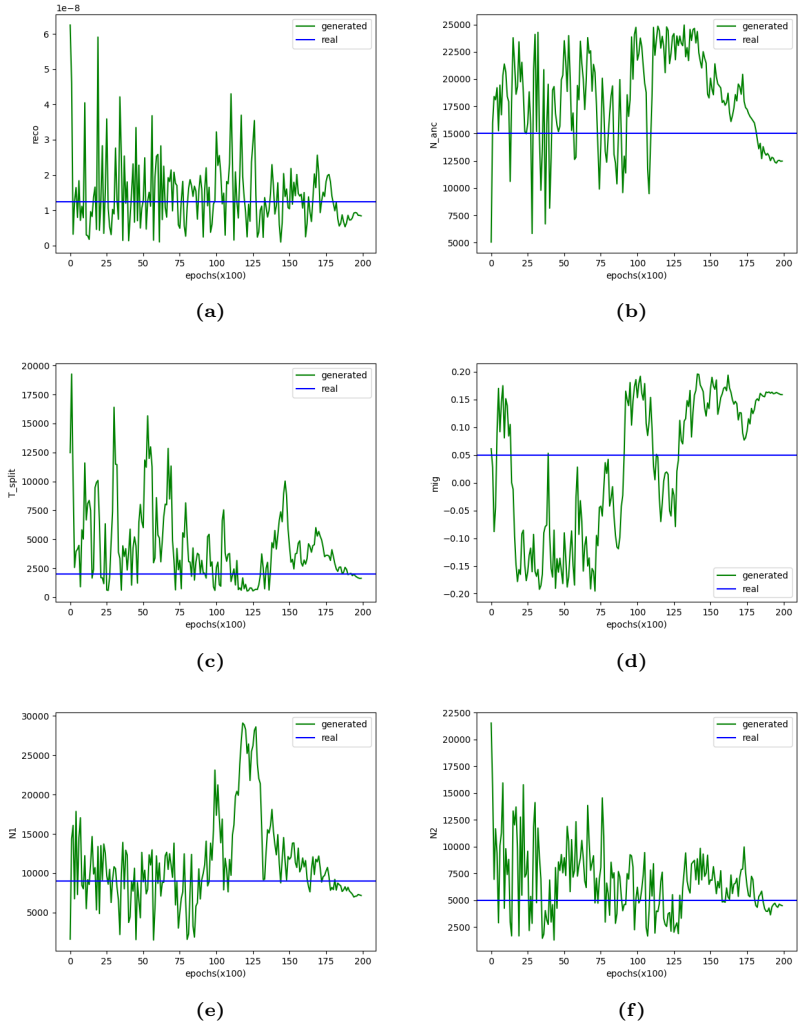
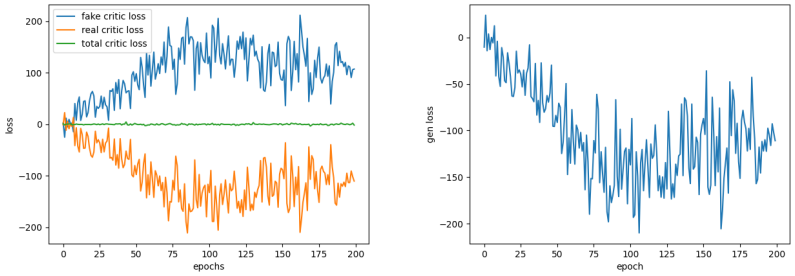


Figure 20: Experiment 4: Parameter values chosen by the generator over the epochs. We observe the very close predictions due to the balance between the critic and the generator. Notice that the total number of epochs is 20000, so a number n on the x-axis is in fact $n \times 100$



(a) **(b)**

Figure 21: Experiment 4: Critic and Generator loss. Notice that the total number of epochs is 20000, so a number n on the x-axis is in fact $n \times 100$ (a) The figure shows clear separation between the real and fake data. This distinction means that fake data are scored higher on the positive scale and real data are scored lower on the negative scale, allowing for quality and consistent learning for the generator. (b) We observe the downward trend of the generator. Fluctuations are normal because between each iteration of the generator, the critic has 5 epochs of training, which can sometimes pump the generator loss up if the critic learns a crucial feature. It's crucial that the generator loss goes down immediately afterwards and not keep going up, which means the critic is consistent and the generator immediately catches up.

References

Arjovsky, M., Chintala, S., & Bottou, L. (2017, Jul). *Wasserstein generative adversarial networks.* PMLR. Retrieved from `https://proceedings.mlr.press/v70/arjovsky17a.html`

Borji, A. (2019). Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, *179*, 41–65. doi: 10.1016/j.cviu.2018.10.009

Brownlee, J. (2021, Jan). *How to develop a wasserstein generative adversarial network (wgan) from scratch.* Retrieved from `https://machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scratch/`

Coop, G. (2013). University of California, Davis. Retrieved from `http://cooplab.github.io/popgen-notes/`

Dobelman, J. (2012, Nov). *Standard regularity conditions for statistics.* Rice. Retrieved from `http://www.stat.rice.edu/~dobelman/courses/Regularity.pdf`

F., G. A. J., Doebley, J. F., Peichel, C., & Wassarman, D. A. (2020). *Introduction to genetic analysis.* Macmillan International Higher Education.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Proceedings of the 27th international conference on neural information processing systems - volume 2* (p. 2672–2680). Cambridge, MA, USA: MIT Press.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017). *Improved training of wasserstein gans.* arXiv. Retrieved from `https://`

arxiv.org/abs/1704.00028 doi: 10.48550/ARXIV.1704.00028

Gutenkunst, R. N., Hernandez, R. D., Williamson, S. H., & Bustamante, C. D. (2009, 10). Inferring the joint demographic history of multiple populations from multidimensional snp frequency data. *PLOS Genetics*, *5*(10), 1-11. Retrieved from https://doi.org/10.1371/journal.pgen.1000695 doi: 10.1371/journal.pgen.1000695

Jiuxiang, G., Zhenhua, W., Jason, K., Lianyang, M., Amir, S., Bing, S., ... Tsuhan, C. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, *77*, 354-377. Retrieved from https://www.sciencedirect.com/science/article/pii/S0031320317304120 doi: https://doi.org/10.1016/j.patcog.2017.10.013

Kelleher, J., Etheridge, A. M., & McVean, G. (2016, 05). Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLOS Computational Biology*, *12*(5), 1-22. Retrieved from https://doi.org/10.1371/journal.pcbi.1004842 doi: 10.1371/journal.pcbi.1004842

Pousada, M., Caballé, S., Conesa, J., Bertrán, A., Gómez-Zúñiga, B., Hernández Encuentra, E., ... Moré, J. (2018, 05). Towards a web-based teaching tool to measure and represent the emotional climate of virtual classrooms. In (p. 314-327). doi: 10.1007/978-3-319-59463-7_32

Roy, R. (2019, Jan). *Generative adversarial network (gan).* Retrieved from https://www.geeksforgeeks.org/generative-adversarial-network-gan/

Schrider, D. R., & Kern, A. D. (2018). Supervised machine learning for population genetics: A new paradigm. *Trends in Genetics*, *34*(4), 301-312. Retrieved from https://www.sciencedirect.com/science/article/pii/S0168952517302251 doi: https://doi.org/10.1016/j.tig.2017.12.005

Wang, Z., Wang, J., Kourakos, M., Hoang, N., Lee, H. H., Mathieson, I., & Mathieson, S. (2021). Automatic inference of demographic parameters using generative adversarial networks. *Molecular Ecology Resources*. doi: 10.1111/1755-0998.13386