Generative Adversarial Networks as a Method to Produce Elliptic Curves by Rank

by

John Cox

Dr. Jeremy Jacobson
Adviser

The Department of Quantitative Theory and Methods

2021

Generative Adversarial Networks as a Method to Produce Elliptic Curves by Rank

By

John Cox

Dr. Jeremy Jacobson

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

The Department of Quantitative Theory and Methods

2021

Abstract

Generative Adversarial Networks as a Method to Produce Elliptic Curves by Rank

By John Cox

Elliptic curves are studied in many areas of mathematics, yet there is still much to learn about their properties. One key property—rank—is particularly tricky. Conjecture states that nearly all elliptic curves are of rank zero or one, with relatively few exceeding rank one.(1) However, it is believed that there is no bound on how high the rank of elliptic curves can be, and the current discovered record is a curve of rank 28. Because of their varied uses in mathematics, it is common for researchers to search for elliptic curves with specific properties such as a rank value. One approach to searching for elliptic curves by rank is to randomly check curves for their rank, a time-consuming and inefficient method. Generative Adversarial Networks (GANs) are an emerging machine learning technique that allows for the creation of fake data that imitates real data. Using GANs, it is possible to create a model that generates elliptic curves by specified rank, given a sufficient training set of elliptic curves to feed the model. This project proposes GANs as a new method for generating elliptic curves and creates a GAN model that generates elliptic curves of rank one at a more effective rate than a guess and check approach.

Generative Adversarial Networks as a Method to Produce Elliptic Curves by Rank

By

John Cox

Dr. Jeremy Jacobson

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

The Department of Quantitative Theory and Methods

2021

Acknowledgements

Thank you to my advisor, Dr. Jacobson, who inspired me to pursue a thesis on this topic. Over the last year, he devoted countless hours out of his busy schedule to guide me through this project, and from him I've learned so much. I would also like to thank my parents for their ongoing encouragement and support throughout my academic endeavors. Without their backing, I never would have made it to this point.

Table of Contents

Tables and Figures

Introduction and Background Information

Generative Adversarial Networks (GANs) are a machine learning framework gaining popularity for their accuracy as a generative model. Common uses for GANs include the creation of realistic images and videos that are computer-generated, and have even been used in popular videos called "deep fakes".(2) The power of GANs can be applied in many fields of study and is already having an impact on the medical field.(3,4)

Although GANs are most often used to synthesize images, they can be used to generate just about any type of data. The only requirements to create a GAN model are to have a sufficient training dataset and computing power to train the model. In this project, I look to apply the GAN model architecture on elliptic curve data, with the goal being to produce elliptic curves of specified rank values. Particularly, I implement the GAN structure to synthesize elliptic curves of rank one. The following section provides a brief introduction to both elliptic curves and GANs.

Elliptic Curves

Elliptic curves and their properties have been studied for hundreds of years and are relevant to a variety of fields. One property of elliptic curves, rank, is one of the most well-known yet one of the most difficult to determine. There is no simple pattern to the rank of curves, and there is no easy way to produce elliptic curves by their rank. Rank is the subject of many theorems, notably proofs on the bounds to the average rank of all curves.(5, 6) The rank distribution conjecture states that about half of curves are rank zero, about half are rank one, and curves of greater than rank one are of almost no density, however, evidence for this conjecture is inconclusive.

Elliptic Curves Over the Rational Numbers

For this project, we will only focus on elliptic curves over rational numbers. An elliptic curve over the rational numbers is the set of rational number solutions to the following equation:

$$y^2 = x^3 + ax + b$$

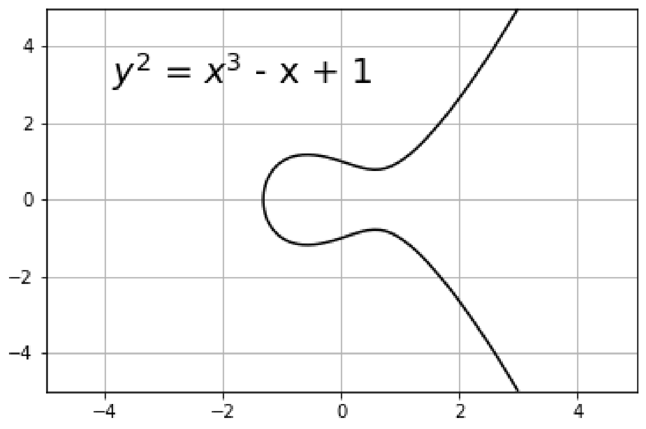
Given $4a^3 + 27b^2 \neq 0$ to avoid singularity

In Weierstrass form, which we will be using later in this project, this becomes:

$$y^2 + ay = x^3 + bx^2 + cxy + dx + e$$

Elliptic curves over the rational numbers play an important role in many fields of mathematics including theoretical math, algebraic geometry, number theory, and cryptography. An example of what such a curve might look like plotted on a graph is shown in figure 1.

Figure 1. Visual example of elliptic curve



Group Law

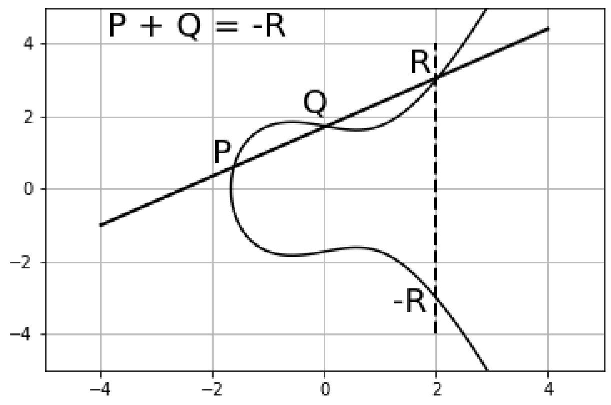
One of the most recognized properties of elliptic curves is its geometric additive property under group law. The group law can be defined as follows:

1. All elements are points on the elliptic curve
2. The identity element is the origin.
3. The inverse of a point P is its reflection across the x-axis.
4. The sum of three geometrically aligned, non-zero points on the curve is equal to the origin.

Under this law, three aligned points on an elliptic curve, when added together, are equal to zero. Note that this group law forms an abelian group, and thus the order of three aligned points does not matter in point addition. Thus, if we take three aligned points on an elliptic curve P, Q, and R, we can say P + Q + R = 0. Equivalently, we can rewrite this as P + Q = -R. This is an important feature that is vital to understanding the rank of elliptic curves.

To visualize this geometric addition, take three aligned points on an elliptic curve P, Q, and R, shown in figure 2. P + Q in this figure is equal to -R. Note that -R is just R reflected over the x-axis.

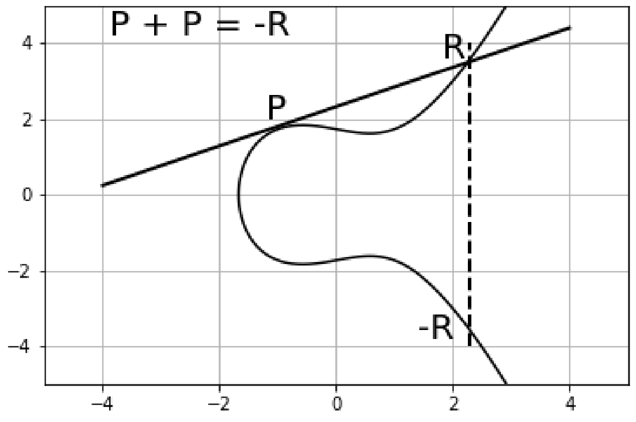
Figure 2. Geometric addition of points on elliptic curve



Rank of Elliptic Curves

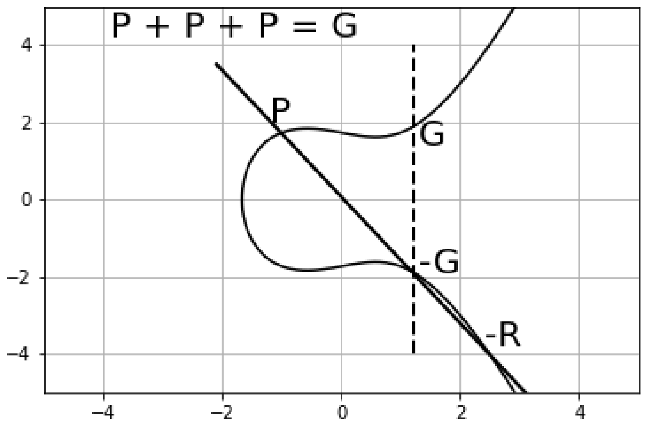
Rank is defined for elliptic curves over the rational numbers Q and is one measure of the size of their solution set. To understand rank, we must revisit the group law. From group law, we know P + Q = -R. But what happens when Q becomes infinitely close to P? As the difference between P and Q approaches zero, the line connecting P and Q becomes tangent to the curve. As such, we can say P + P = -R. An example of this is shown in figure 3 with new P and R values.

Figure 3. Geometric addition on the tangent line



Now that we have P + P = -R, we can add P to itself again. This will result in a new value, which we can call G. We can keep adding P to itself over and over and obtaining new values on the curve. At some tangency points on some curves, this process of adding P to itself can go on infinitely without a solution ever returning back to P. The number of tangency points on a curve where this is true is the rank of that curve. Many curves don't have any tangency points of that nature and are thus rank zero. Rank zero curves have a finite number of solutions while rank one or higher curves have infinitely many solutions.

Figure 4. Repeated geometric addition of P to itself



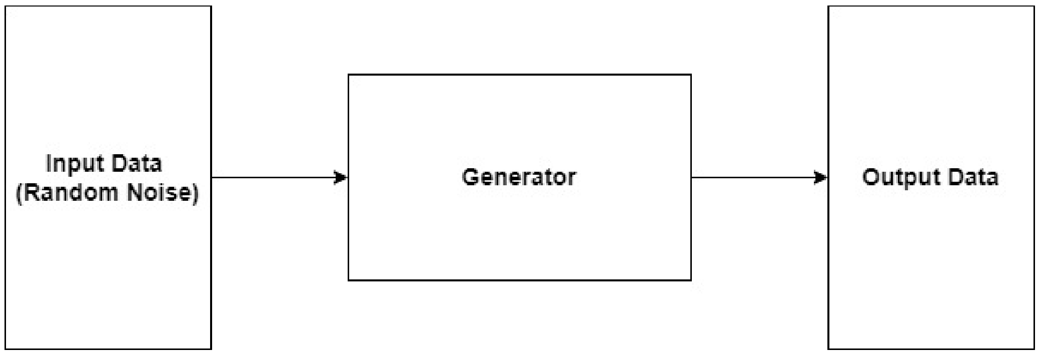
Generative Adversarial Networks (GANs)

The idea for GANs as a framework for generating data was first published in 2014 by Ian Goodfellow et al.(7) In his paper, Goodfellow describes a training process structured behind two neural networks: a generator and a discriminator. These two neural networks, when trained together in an adversarial fashion, achieve a powerful generative model. A brief description of each of these two neural networks and how they interact in a GAN framework is given below.

The Generator

The goal of the first network—the generator—is to create data that is indistinguishable from the training data. The generator is typically a deep neural network that takes in a random noisy input and outputs data in the dimension of our desired result. When trained properly, the generator can take in a batch of random noisy data and transform it into data that closely imitates the training data. A simple framework for a generator is displayed in figure 1.

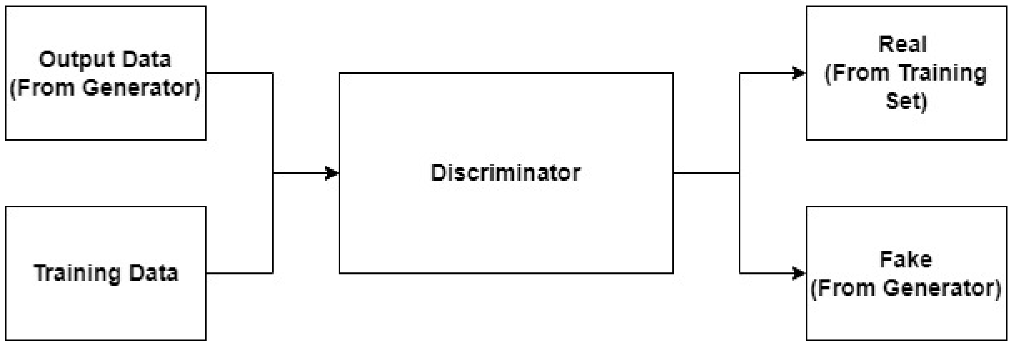
Figure 5. Generator flowchart



The Discriminator

The goal of the second network—the discriminator—is to appropriately label data as either created from the generator or chosen from the training data. This network typically consists of a binary classifier and takes as input batches of data consisting of both generated data from the generator and real data from the training set. The discriminator then labels each input it receives as either real (meaning it is from the training set) or fake (meaning it is from the generator). A simple framework for a discriminator is shown in figure 2.

Figure 6. Discriminator flowchart



The Adversarial Relationship Between the Generator and Discriminator

The clever feature of GANs that makes them so effective is the adversarial relationship between the generator and discriminator. Indeed, these two neural networks compete against each other during their training process. The generator strives to create data that can fool the discriminator into labeling it as real, while the discriminator aims to outsmart the generator and correctly classify all input it receives. Throughout each training step, the loss functions of the generator incorporate its ability to fool the discriminator, and likewise, the loss functions of the discriminator incorporate its accuracy of labeling data. These networks compete in a zero-sum game, with the goal being to have a generator that produces accurate and unique data that closely resembles the training data.

Data

Source

The required data for this project is training data for the GAN. This data was pulled from the publicly available online database The L-Functions and Modular Forms Database (LMFDB).(8) LMFDB provides a wide selection of data of elliptic curves over the rational numbers. This data includes an inventory of elliptic curves defined by their coefficients in Weierstrass form, with information on their properties including rank. Much of this data is hosted on GitHub, where I was able to use a bash script to download it onto my computer.(9) The specific variables that I was interested in were the five coefficients that define each curve and the rank value of that curve. In all, I was able to download 480,517 elliptic curves and their respective information for analysis.

It is important to note that these curves are not a random sample. Curves hosted on LMFDB tend to have certain properties, such as specific conductor or torsion order values at a different distribution level than what you would expect randomly generated curves to have. Thus, the distribution of coefficients of certain rank curves I pulled from this project may differ from randomly generated curves of that same rank. This problem may alter my results, but it should not be a barrier for producing curves by rank from the generator.

Filtering

Not all the available curves were suitable for analysis. With the scope of this project revolving around rank one curves, the training data only required rank one curves. Thus, I filtered out any curve that was not rank one. Due to the nature of the GAN model created in this project, the coefficients should be in binary format. Thus, I filtered out any curve that had coefficients magnitude larger than 2^{13} so that the coefficients were more manageable to deal with in their binary form. Once in binary form, each binary digit of each coefficient represents a data point to be fed into the model. After the data filtering process, I was left with 91,430 rank one curves.

Each curve is represented by its five coefficients. The range of values for each coefficient differs. The first and third coefficients are either zero or one. The second coefficient can be negative one, zero, or one. The fourth and fifth coefficients can range from -2^{13} to 2^{13} due to the restraints I implemented. Thus, the first and third coefficients can be represented in one binary digit, the second coefficient can be represented in two binary digits (the first digit represents whether the number is positive or negative), and the fourth and fifth coefficients can be represented in 14 binary digits. In all, each of the 91,430 curves used in the training set was represented as lists of 32 binary digits.

Methods

By utilizing the framework of a GAN, it is feasible to train a generator that can produce the coefficients of elliptic curves by rank. To test this, I propose a GAN model that generates elliptic curves of rank one. The goal is to create a GAN model that produces the coefficients of elliptic curves of rank one at a higher rate than random.

The GAN models I use in this project were designed and run on a python script hosted on Jupyter and use the PyTorch library. Many versions of the model were tried and tested to obtain the best-known parameters to generate elliptic curves of rank one. The specific functions and parameters that were tested that led to the final model are explained in the following section.

The accuracy of each GAN model was tested by feeding its trained generator random noise and saving the output as a sample of generated elliptic curves. The rank of the curves in these samples was then calculated using built-in functions in Magma software. The sample from the final trained model was compared to a sample output from an untrained model, and the differences between the two samples were calculated.

Loss Functions, Backpropagation, and Optimization

Loss functions are one of the most important factors that go into a GAN model. The loss functions are how the generator and discriminator are evaluated in the training loop, and from these loss values, the back-propagation algorithm calculates the gradient vector used to update the neural networks. Binary Cross-Entropy (BCE) loss was used to calculate the loss values. BCE loss is given by the following formula, where y is the true label of the data and $d(x)$ is the predicted label of data:

$$loss = -\left[y \log d(x) + (1 - y)\log\left(1 - d(x)\right)\right]$$

Four loss values were calculated and tracked. In this formula, y represents the label of the data, and $d(x)$ represents the predicted label chance (number between zero and one) as found by the discriminator for that same data. The first loss value of note, generator loss, is calculated where $d(x)$ is the predicted labels of the output of the generator, and y is 'real' label one. This loss value represents how well the generator generates data that fools the discriminator into labeling it 'real'. The second loss value, true discriminator loss, is calculated where $d(x)$ is the predicted labels of the training data, and y is the true label for 'real' data, one. This value represents how well the discriminator is labeling the training data as 'real'. The third, fake discriminator loss, is where $d(x)$ represents the predicted labels of the output of the generator, and y is the 'fake' label zero. This value represents how well the discriminator is labeling generated data from the generator as 'fake'. Fourth, discriminator loss is the average of true discriminator loss and fake discriminator loss. This loss value incorporates the entire performance of the discriminator over both 'real' and 'fake' data.

Once the loss values are calculated, a process to update the neural network weights begins. Through the backpropagation algorithm, gradient vectors are calculated from the loss functions. This process is performed for the previously defined generator and discriminator losses. True discriminator loss and fake discriminator loss values are not used directly in this process, although their values are used in the calculation of discriminator loss. With the calculated gradient vectors, the Adam optimization algorithm is deployed to update the weights of the generator and discriminator respectively. The generator weights are updated using the gradient from generator loss, and the discriminator is updated using the gradient from discriminator loss.

Parameters

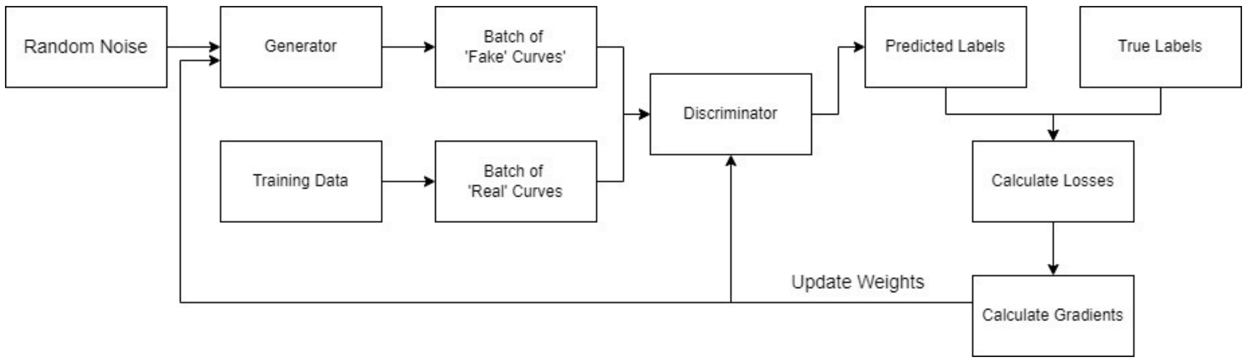
Due to the complex architecture of the GAN, there are many different parameters to be tested to optimize the model. The main parameters that were tested throughout model iterations were: number of epochs, number of layers, number of nodes, batch size, learning rate. After running into a few problems with the training steps, one more parameter was added in –variance weight.

Parameters were tested by trial and error from a baseline model. The baseline model was made up of ten epochs, four layers, 1000 nodes, 256 batch size, and 0.001 learning rate. Models were tested by changing one parameter from the baseline model, typically by a factor of ten, and the quality of the new model was subjectively evaluated on a few factors. These factors were the average loss values of the generator and discriminator over each epoch, the accuracy of the model results, and the number of unique curves that were generated by the model. If changing one parameter led to an improvement, it was considered for the final model. More fine-tuned updates began to take shape after learning what parameters were effective and which weren't.

GAN Architecture

An overview of the architecture for the GAN model is displayed in figure 7. The process begins with random noise data that is fed to the Generator. The random noise is a batch of randomly generated curves in binary form. The generator, a fully connected neural network with multiple layers and thousands of nodes, takes in this random noise, applies its weights and biases to the noise, and outputs a batch of elliptic curves. This batch of curves is then used as input to the discriminator along with a random batch of curves from the training set. The discriminator, a neural network of the same layers, nodes, and batch size as the generator, takes in these inputs and outputs a number for each curve that correlates to either 'real' or 'fake' when the sigmoid function is applied to its output. In this case, 'real' is one, and 'fake' is zero. The output of the discriminator is a number between zero and one for each curve and can be thought of as the chance of a curve being real or fake according to the discriminator. These predictions from the discriminator are fed into the BCE loss function with their respective true labels. The loss values are calculated, and from those, the gradients are calculated. The rectified linear activation function (ReLU) is used as the activation function throughout the neural networks and the Adam optimization algorithm is used to update the weights of each model, see PyTorch documentation for details.(10) Once the gradients are calculated, the weights of the generator and discriminator are appropriately updated. These weights are adjusted by moving in the opposite direction of the gradient vector of their respective loss functions (generator loss and discriminator loss). The goal is to minimize the loss of these functions, that is have the generator produce curves to which the discriminator thinks is real, and to have the discriminator become adept at labeling generated curves as ‘fake’ and curves from the training set as ‘real’. This process is repeated for every batch of data in the training set and then again over each epoch

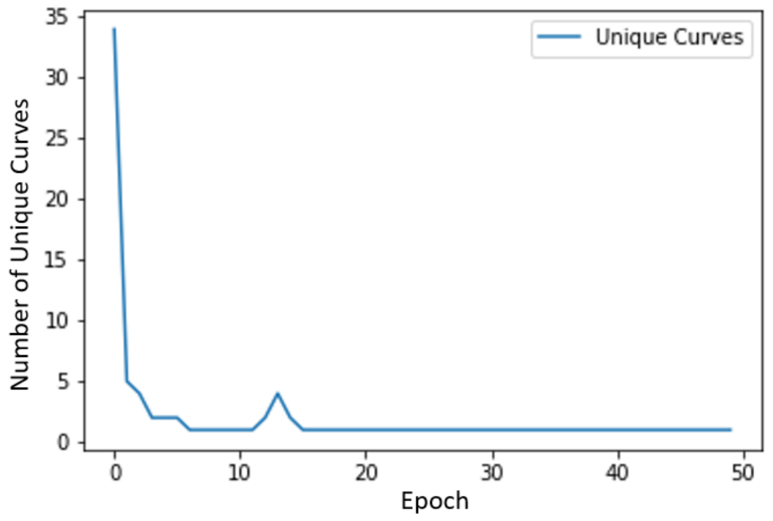
Figure 7. GAN architecture



Issues During Training

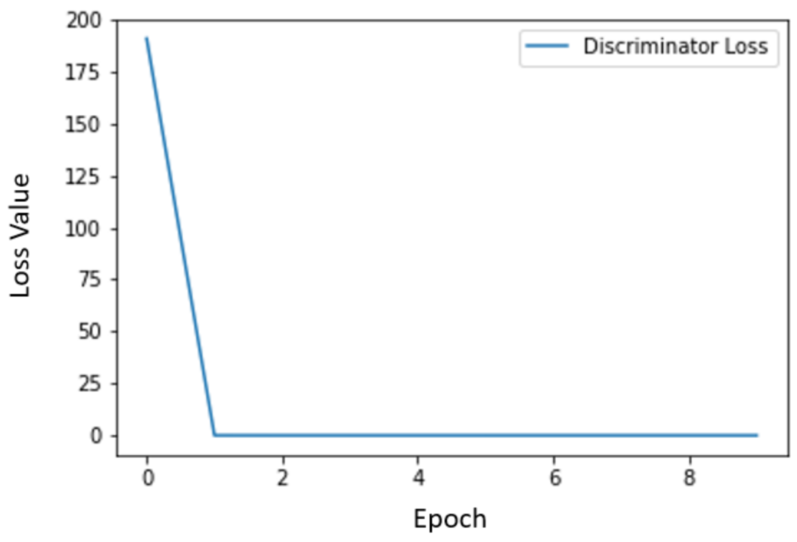
I ran into two common problems during the training of the GAN. The first problem was mode collapse. When testing some of my models, the generator would converge into creating just one unique curve. This can happen if the generator finds a single curve that is likely to fool the discriminator and it learns to continuously create that curve for any given input. An example of how mode collapse takes shape over each epoch is shown in figure 5 below. To fix this issue, I added a new term to the generator loss. This new term incorporates the variance of the training set, with the idea to help the generator output have similar variance. Specifically, the term is the sum of mean squared errors between each binary digit in the generated data and training data. This term was added to the BCE loss function for generator loss and was multiplied by a parameter called variance weight. Adding this term to the generator loss incentivizes the generator to produce results that are as diverse as the training data, and this term was effective in solving mode collapse in my training process.

Figure 8. Mode collapse over epochs



The second problem was discriminator loss convergence. The discriminator loss values would very quickly converge to zero, meaning that the discriminator became quite accurate very quickly. This is not always productive for the generator, if the discriminator is too good at labeling, the generator might not have room to improve. An example of discriminator loss convergence over each epoch in one of my models is shown in figure 6 below. To fix this issue, I implemented noisy labels into the loss functions. Instead of having the true labels be either zero or one, I had the labels be a random number between 0.9 to 1.0 for ‘real’ and 0.0 to 0.1 for ‘fake’. Adding in these noisy labels ensured the discriminator would never become too accurate and allowed for more continuous training for the generator without getting caught in local minima.

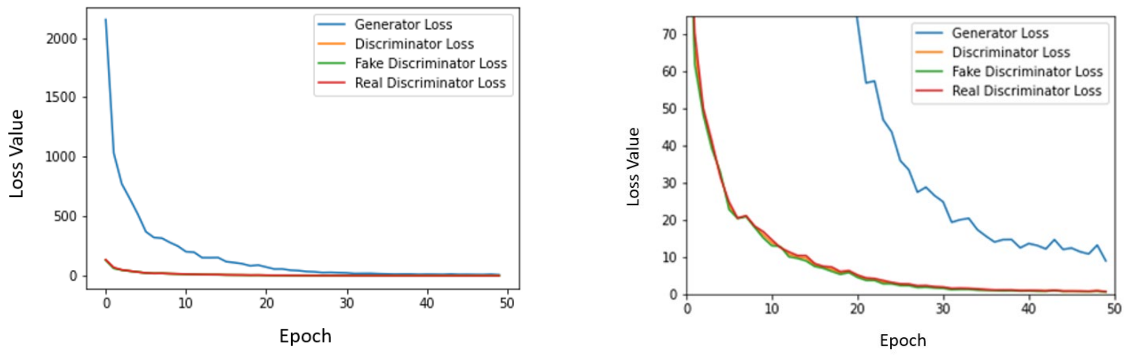
Figure 9. Discriminator loss convergence over epochs



Final Model

The final model was put together using four layers, 50 epochs, 5,000 nodes, 256 batch size, 0.001 learning rate, and 0.0 variance weight. These parameters were the same in both the generator and discriminator. Noisy labels were used for the discriminator loss but not the generator loss. This model was chosen subjectively based on the stability of its loss values, the accuracy of the model, and the diversity of curves it produced. The different loss values of this model over each epoch are shown below.

Figure 10. Loss values of final model over epochs



Results

Table 1. The percent of generated curves that fall into each rank category by model

Rank	Trained Model	Untrained Model
0	27.73%	37.80%
1	47.27%	40.94%
>1	25.0%	21.26%

The result of my final model was a generator that could create rank one elliptic curves at a rate of 47.27%, compared to the untrained control model which produced curves of rank one at a rate of 40.94%. This marked a 6.33 absolute percentage increase in production of rank one curves from an untrained model. On a relative scale, the trained model was 15.46% more accurate at producing rank one curves than the untrained model. Each curve the model created was unique, so 256 curves were generated in total. The control produced 254 unique curves in comparison. As well as an increase in rank one curves, the trained model also created curves of higher than rank one at a higher rate compared to the control model, 25.00% to 21.26%. The amount of rank zero curves produced decreased in the trained model compared to the control, going from 37.80% of generated curves to just 27.73% of curves.

Many of the models I tested yielded similar accuracy over the untrained model. To confirm that these numbers the generator consistently scored were significant, a two-sample t-test was performed on this sample to test if the rank values from the trained model followed a different distribution from the untrained model. The t-test yielded a t-statistic of -2.31, correlating to a p-value of 0.02. This indicates that the trained model produced significantly different curves than the untrained model at the 0.05 alpha level, with that difference coming from the trained model creating rank one or higher curves with more frequency.

Discussion

GANs show promise to produce elliptic curves by rank value. The model in this project demonstrated that GANs can generate elliptic curves of certain rank at a significantly higher rate than random. However, there is still much room to improve, as my model only produced desired curves at a rate lower than half. More research is needed to discover the extent to which GANs can be used to produce elliptic curves of desired properties, and what methods can achieve such a model. Similarly, more research needs to look at how effectively these models can be trained on elliptic curve data of rank not equal to one. Future studies might look at training a GAN on elliptic curves of higher rank, such as two or greater. If effective, GANs might even have the power to advance research on discovering curves of record-breaking rank.

References

1. Goldfeld D. Conjectures on elliptic curves over quadratic fields. 2006. p. 108-18.

2. Shen T, editor "Deep Fakes" using Generative Adversarial Networks (GAN)2018.

3. Lan L, You L, Zhang Z, Fan Z, Zhao W, Zeng N, et al. Generative Adversarial Networks and Its Applications in Biomedical Informatics. Front Public Health. 2020;8:164-.

4. Kazeminia S, Baur C, Kuijper A, van Ginneken B, Navab N, Albarqouni S, et al. GANs for medical image analysis. Artificial Intelligence in Medicine. 2020;109:101938.

5. Bhargava M, Shankar A. Binary quartic forms having bounded invariants, and the boundedness of the average rank of elliptic curves. Annals of Mathematics. 2015:191-242.

6. Bhargava M, Shankar A. Ternary cubic forms having bounded invariants, and the existence of a positive proportion of elliptic curves having rank 0. Annals of Mathematics. 2015;181(2):587-621.

7. Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative adversarial nets. Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2; Montreal, Canada: MIT Press; 2014. p. 2672–80.

8. The LMFDB Collaboration, *The L-functions and modular forms database,* http://www.lmfdb.org, 2021, [Online; accessed 6 December 2021]

9. John Cremona. (2016). JohnCremona/ecdata: All conductors to 400000 (2016-10-17). Zenodo. https://doi.org/10.5281/zenodo.161341

10. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

11. Cox, J. (2021). Elliptic Curve GAN Files (Version 0.1.0) [Computer software]. https://doi.org/10.5281/zenodo.5762842