Teaching machines to write

Talor Anderson

CS 4990 (Prof. Hao Ji), Cal Poly Pomona

Abstract

Machine Learning has been gaining traction in many areas of computation. One of the earliest and most well-known applications has been correctly identifying a collection of handwritten numbers known as the MNIST dataset. Matching the correct number to one of these written characters is often an exercise for new students in Machine Learning.

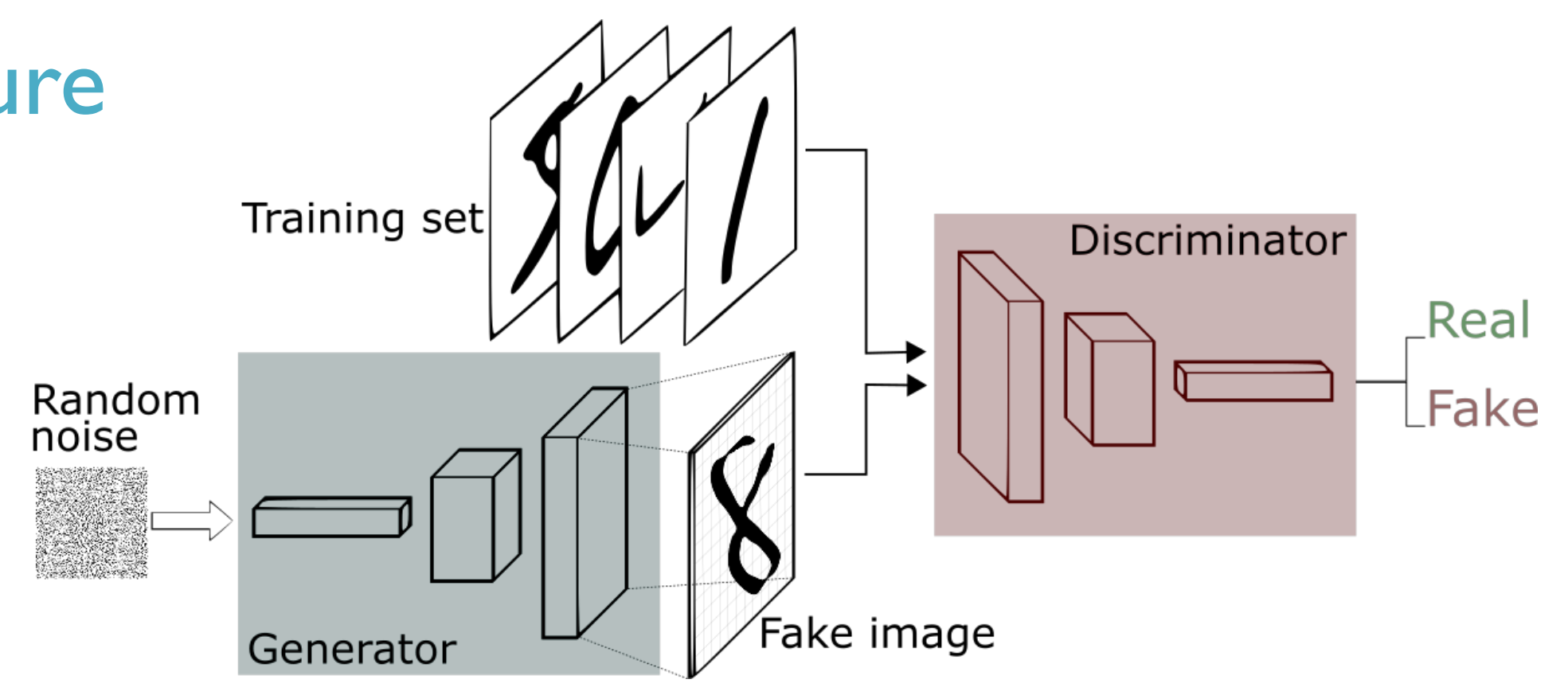
I chose to turn this task on its head. Rather than guessing what number was displayed in an image, could a machine instead create images of numbers that looked as if they had been written by hand? In short, yes!

This project focused on use of a GAN (Generative Adversarial Network) to mimic human handwriting as closely as possible. Machine Learning pioneer Yann LeCun called adversarial training "the most interesting idea in the last 10 years in ML." Details on my GAN implementation are provided below.

Introduction

The GAN architecture is described as follows. Two competing networks are created, named the Generator (blue, in figure below) and the Discriminator. Inputs to the Generator are an image of random noise and a number label. The Generator then attempts to transform the image to fit the given label using what it has learned from previous trials. It outputs an image that it believes will mimic training data enough to fool the Discriminator.
The Discriminator takes a sampling of real training images and fake generated images, and attempts to decipher which is real and which is fake. Rewards are given to the Discriminator when it identifies real or fake samples correctly, while the Generator is rewarded for fooling the discriminator. Because one's loss is the other's gain, this gives the network its name of 'Adversarial'.

GAN Architecture Visualization



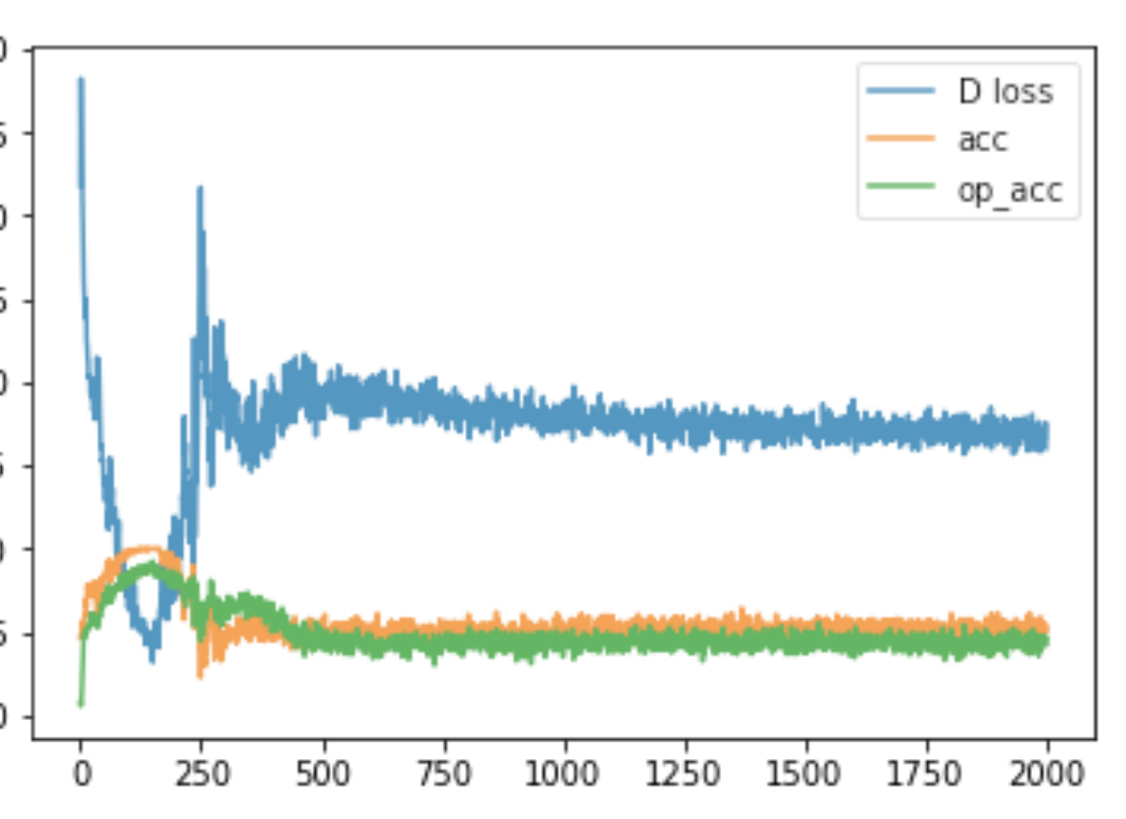
Data Used

My project used the widely known MNIST dataset to train on. This dataset, consisting of 28x28 px images of handwritten numbers and their corresponding label, has been studied hundreds of times making it easy to implement and judge my results.

Methods

My project was built off of a class, ACGAN, that encompassed both Keras models for the Generator and Discriminator, training them together. Both models are fighting to beat the other, and training alternates in batches.

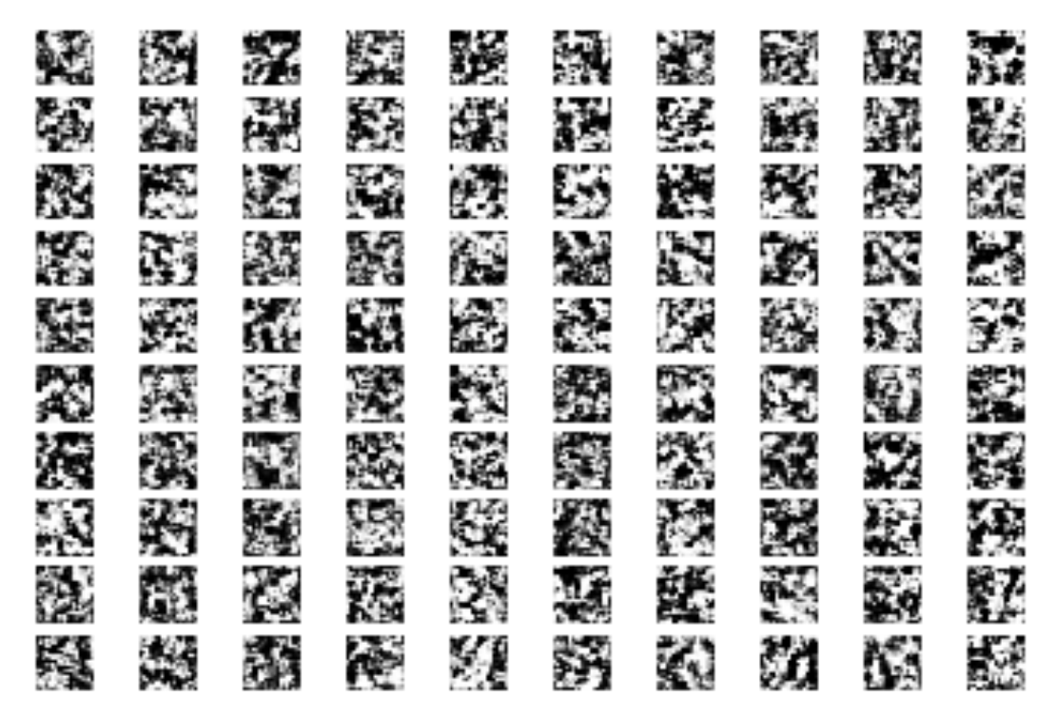
Training Accuracy

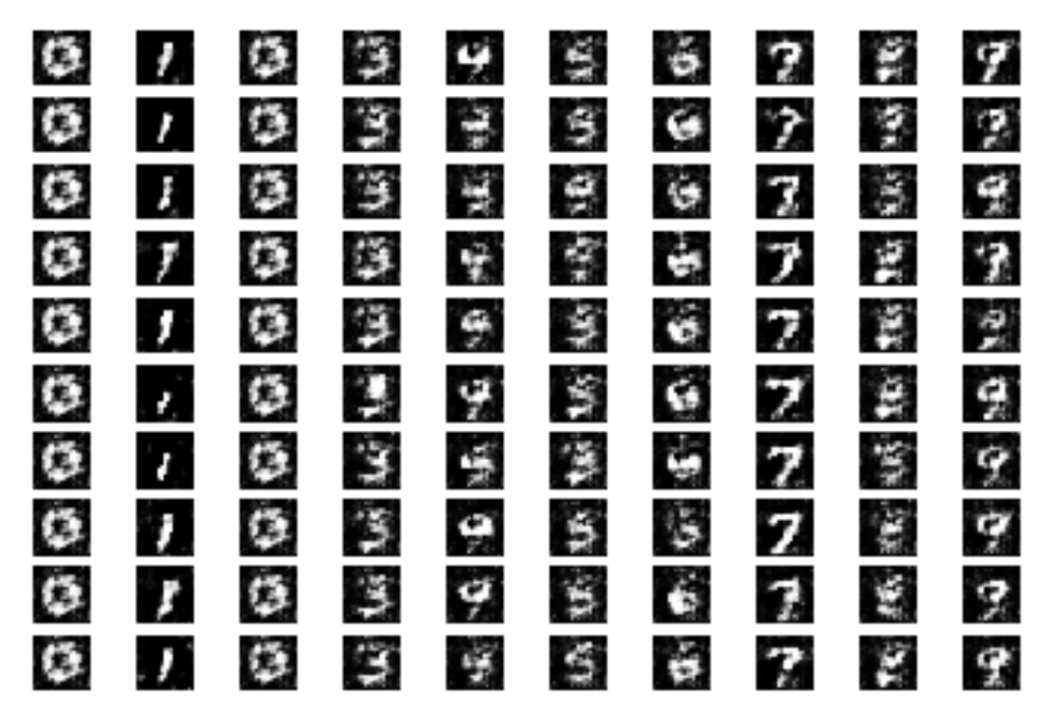


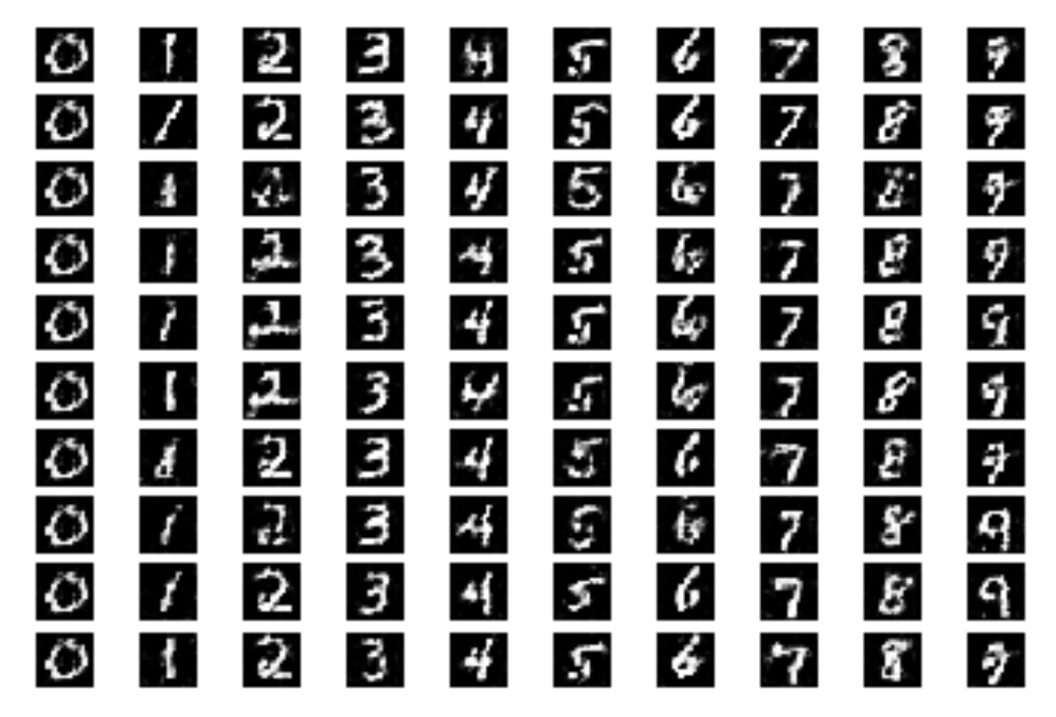
In the training graph (left), the Discriminator quickly learns to identify false samples, noted by the jump in its accuracy (orange) to 1.0(100%). At this point, the Generator is still outputting formless blobs. As the Generator learns, the accuracy converges to 0.5(50%), meaning that the discriminator can't tell the difference between an equal amount of real and fake samples, so we have reached our goal.

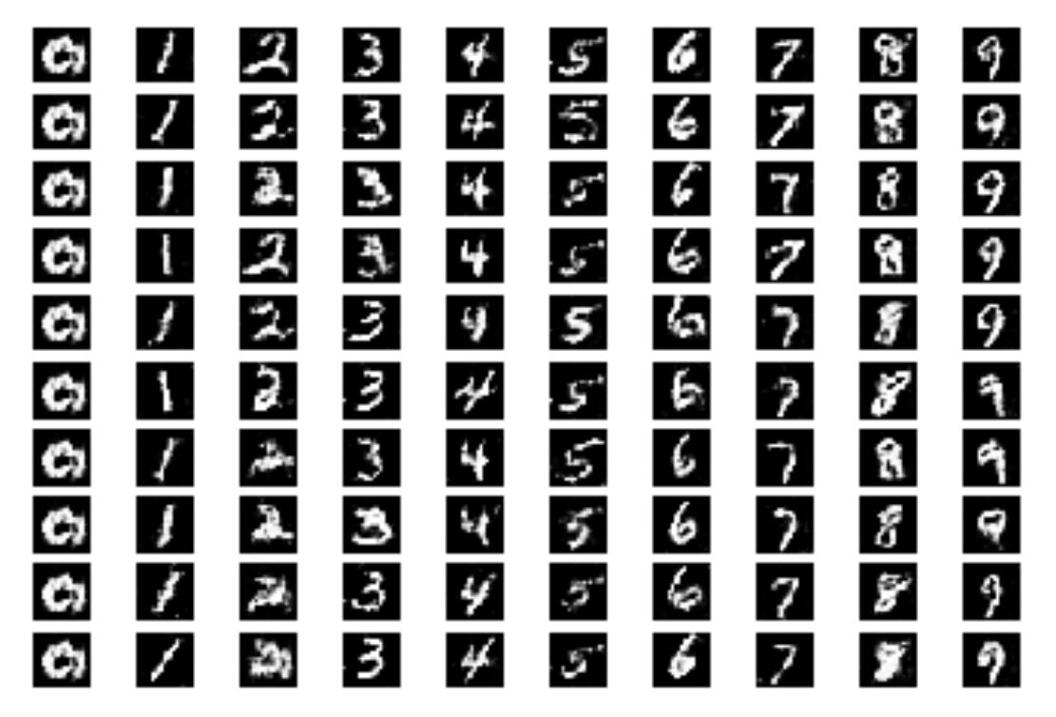
Results

The generator starts out creating random noise, and over many iterations learns how to best fool the discriminator, consequently generating more and more realistic images after many hundreds of iterations.


0 iterations


400 iterations


1000 iterations


2000 iterations

Results (Continued)

As seen in the figure, the generator slowly improves more and more until it generates legible characters. Once training is ended, the discriminator can no longer tell the difference between any training samples vs generator output. This means we have successfully reached our goal of synthesizing handwritten numbers.

After training, I noticed several quirks of my network that were characteristic of GAN training. For one, the generator suffered from mode collapse on certain digits (zero). This is when the generator outputs the same sample every time because it will always fool the discriminator, which is why all the generated zeros look the same. This is a common problem with GANs.

Second, the discriminator also was coded to predict the correct label of samples, not just their validity. The discriminator got generally good accuracy guessing correct numerical labels (>50%), but not as good as can be achieved (>95%). In my research, I learned that GANs can be notoriously difficult to tune and train, so I think with some simple optimization these problems could be overcome.

Conclusion

Using GANs, we can generate new data based on previous data we have seen. While GANs have widespread potential, I noticed one application while browsing a Kaggle competition for MNIST classification. Since the entire dataset is public, contenders could train their models on the whole dataset and achieve machine learning models with 100% accuracy via overfitting. If a competition were to use synthesized digits via a GAN, this would be impossible. This would help Kaggle keep their competitions more fair and fun for newcomers. This is just one application of many for the new field of GANs.

References

1. Odena, A. (2017). Conditional Image Synthesis with Auxiliary Classifier GANs. arXiv: 1610.09585v4 [stat.ML].

email@university.edu