CS 229 Final Project: A Foray into GANs

Madeleine Yip (mjyip) and Kevin Tan (tankevin)

June 3, 2021

Abstract

This project explores different types of Generative Adversarial Networks on the MNIST dataset, from fully connected networks to variations of convolutional networks. We examine different strategies to improve GAN training and convergence, including (1) adding skip connections and (2) adding a parallel branch to our baseline DCGAN model. Even though all three achieved image quality at the end, the parallel branch DCGAN demonstrated earlier loss convergence.
Convolutional GAN Code:
`https://colab.research.google.com/drive/1lwUOwZdZKaYE1i2N7r4VHlF6JWFbXhOS?usp=sharing`
Fully-Connected GAN Code:
`https://github.com/kevtan/exploring-GANs`

1 Introduction

Although much of machine learning is focused on classification and regression tasks, generative models are an exciting sub-field worth exploring. Generative Adversarial Networks (GANs) are especially relevant to the CS 229 curriculum because they are a natural outgrowth of Gaussian Discriminant Analysis (GDA) in which representations of the training data are learned. Ideas seen in the problem sets like self-supervised and semi-supervised learning techniques are also present in GANs. In particular, the training of unconditional GANs are a form of self-supervised learning in that the supervisory signals are derived from the training algorithm itself and the training of conditional GANs are a form of semi-supervised learning in that some of the supervisory signals are derived from the training dataset while the others are derived from the training algorithm.

2 Related Work

The GAN generator and discriminator template builds off of Ian Goodfellow's initial GAN work [2] Our convolutional GAN is inspired from the DCGAN paper [5]. Originally introduced in the ResNet architecture, skip connections perform well for various applications by preserving features learned across various layers. [3]. We used the MNIST Handwritten digit dataset to keep our overall network on the smaller side for faster iteration and experimentation [4].

3 Methods

3.1 The Big Idea

The high-level idea of GANs is to have 2 adversarially-competing networks: the generator and the discriminator. The function of the generator is to take in noise vectors ξ from some latent space and, in our case, generate fake but highly-realistic 28x28 1-channel handwritten digits $\hat{X}$. The function of the discriminator

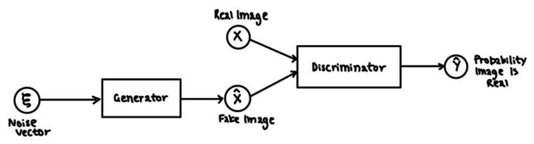


Figure 1: The unconditional GAN architecture. Noise vectors are fed into the generator to create fake handwritten digits. These are mixed with the real images from the MNIST dataset and fed to the discriminator. The discriminator then outputs probabilities that the images it received was real.

is to take in 28x28 1-channel handwritten digits from both the generator $\hat{X}$ and the training dataset X and output a probability that the image it received was real $\hat{Y}$. This is illustrated in Figure 1

The discriminator and generator are trained in an alternating fashion resembling coordinate descent. To train the discriminator, noise vectors are fed to the generator to get fake images. The fake images are combined with real ones and posed to the discriminator as a supervised learning problem of distinguishing between real and fake images. The BCE loss function is used here. To train the generator, noise vectors are again fed to the generator to get fake images. These fake images are artificially earmarked as real so that the generator gets penalized when the discriminator marks them as false and rewarded otherwise; this highlights the adversarial nature of the networks.

3.2 A Dead-Simple Unconditional FCGAN

For our initial experiment, we used incredibly simple fully-connected neural networks for both the generator and the discriminator. The purpose was to test that our GAN training infrastructure (e.g. the training loop, the saving of checkpoints, the monitoring of performance metrics) was set up correctly and working as expected. The remarkable thing was that, even with such a simple network with so few parameters, we were able to get surprisingly good results.

3.2.1 The Architecture

Our first pass at a generator architecture included a single hidden layer with a `LeakyReLU` function. We chose to use the `LeakyReLU` activation function instead of the more-traditional `ReLU` activation function because of the *vanishing gradient problem* where, when the inputs to the `ReLU` activation function is negative, the derivative of the activation is zero.

Our first pass at a discriminator architecture include a single hidden layer with a sigmoid activation function. We chose to use the `LeakyReLU` activation function in the discriminator for the same reason we chose it in the generator. The single sigmoid-activation output neuron intuitively represents the probability that the input was real.

3.2.2 Training Convergence

In the beginning, we found that the generator's loss spiked while the discriminator's loss dropped. This is shown in Figure 2 and can be explained by the fact the relative ease of the discriminator's task. However, at around 20 epochs, the generator loss drops while the discriminator loss spikes. During this phase, the generator has learned to generate more convincing images; this is corroborated by Figure 3, where the generation results look dramatically more believable between epochs 33 and 66.

3.2.3 Results

We use ξ as the notation used for representing the noise vectors. In order to track the learning progress of the generator, we randomly sampled a fixed set of noise vectors $\{\xi^{(i)}\}_{i=1}^{k}$ and fed them to the generator

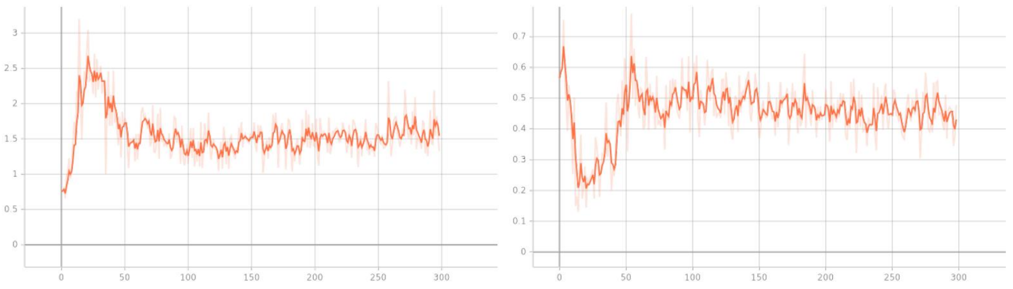


Figure 2: On the left is the generator loss plotted against training epochs. On the right is the discriminator loss plotted against training epochs.

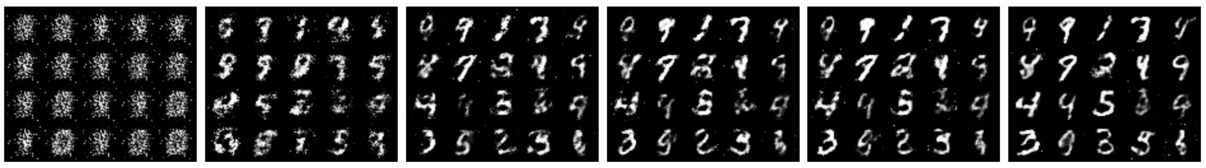


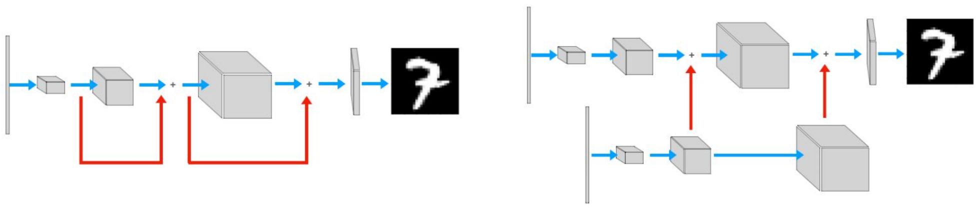
Figure 3: From left to right and top to bottom are the generation results for a fixed set of 20 random noise vectors after 33, 66, 105, 128, 151, and 298 epochs.

at the end of each epoch. It was extremely insightful to see how the generated images for these fixed noise vectors crystallized as the generator improved.

3.3 Convolutional Generators

After having seen promising results with even a simple fully-connected GAN, we then moved on to integrating CNNs. Our Tensorflow [1] implementation of the baseline model is inspired by DCGAN [5]. Because our results from our baseline surpassed those of our direct DCGAN implementation, we decided to use a variation of DCGAN (without batch normalization) as our baseline. Our second difference was that our baseline remains consistently 128 feature maps deep at each convolutional layer. The last difference was that our fixed noise fed into the generator was a 100-dimensional vector.

We experimented with two other generator models: one with skip connections and another with a parallel CNN branch. For the latter-proposed model, we were inspired by skip connections but we also wanted to see if adding a branch could also encourage stability and earlier convergence than our baseline. The figure below shows our two proposed CNN architecture side by side. The baseline model is simply the model on the left with the red arrows (representing the skip connection) removed. In both images, the noise input is scaled up by a factor of two with each transposed convolution block. The first convolutional block reshapes the noise to be 7x7, for the second block 14x14 and for the third 28x28. Each of these convolutional blocks are 128 feature maps deep. Note how addition operation is performed at scales 14x14 and 28x28.



The vanilla skip connection generator (left) and the parallel branch generator (right). The red arrows represent the intermediate operations needed for addition (for the left, an upsample and for the right, a transposed convolution). The blue arrows represent the flow of data between transposed convolution blocks.

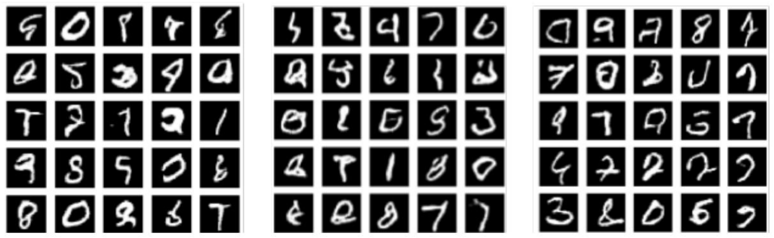
3.4 Loss Function

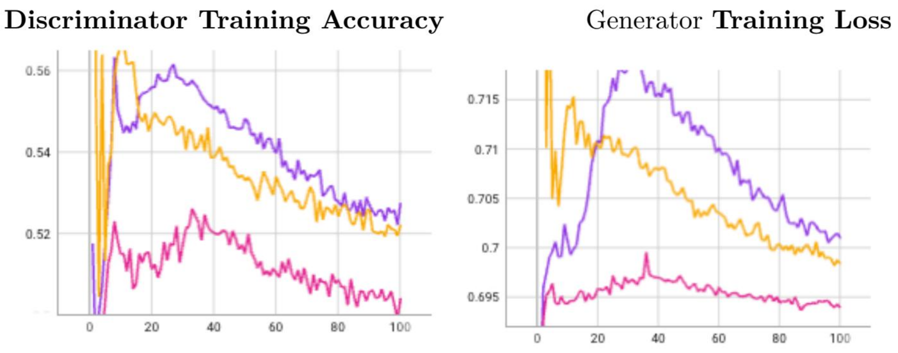
Like the DCGAN paper, we used the binary cross entropy loss function for both our generator and discriminator. The discriminator loss is $L_D = \frac{1}{m}\sum_{i=1}^{m}\Big[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))\Big]$ while the generator loss is $L_G = \frac{1}{m}\sum_{i=1}^{m} - \log(D(G(z^{(i)})))\Big]$, where D represents the discriminator model, G represents the generator model, $x^{(i)} \in \mathbb{R}^{28x28}$ is the real image example, $z^{(i)} \in \mathbb{R}^{100}$ is the noise vector fed into the generator and m is the size of the batch.

4 Experiments

4.1 Convolutional Generator Architectures

Since GANs cannot be evaluated by metrics other than loss, we compared the overall quality of images between the three modesl. Although it was difficult to pinpoint any marked improvements in our results, the loss curve trends did indicate interesting differences between the architectures. We only managed to train for 100 epochs for each experiment due to time constraints.

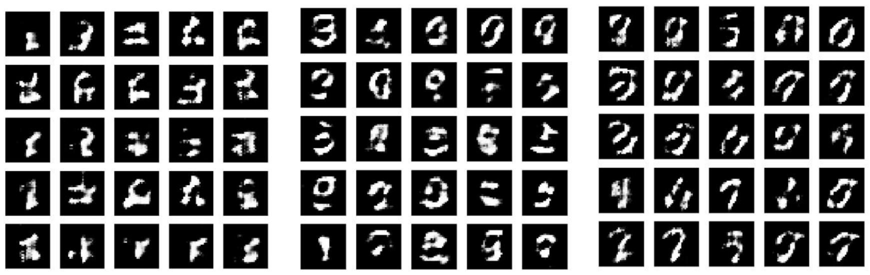

The baseline results (left), the skip generator results, and the parallel generator results (right) after 100 epochs



Yellow line is the baseline, purple is the skip generator, and magenta is the new parallel architecture.

Because our proposed parallel architecture was inspired by skip connections, we predicted that this generator would perform and converge similarly as the vanilla skip connection generator. However, it actually took longer time for the vanilla skip generator to converge while our model took shorter time. Also, compared to the baseline and vanilla skip connection model, our parallel model tends to favor the generator more as indicated by the lower discriminator accuracy. For the discriminator, we see that there is a jump in initial training accuracy for the baseline and skip generators. However, there is less of a jump for the parallel generators. Visually we can compare the "convergence" by analyzing earlier generated fake images. The figure below compares the image quality around 2 epochs. Around epoch 2, baseline results are less well formed compared to the skip and parallel architecture results.

Fake images generated after 2 epochs

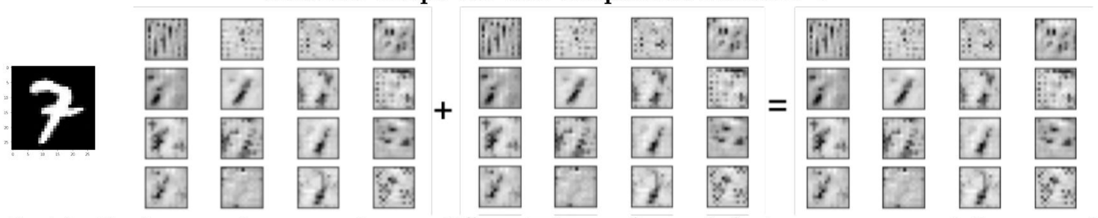


For the skip and parallel architectures, even though the generator loss numbers are significantly more spread apart, the results are both similar quality. This is because binary cross entropy loss depends on discriminator performance. Because the skip discriminator at epoch two achieves higher accuracy, the generator loss is subsequently higher for the skip generator. As a result, even generator loss may not be the most accurate metric in quantitatively comparing convergence.

4.2 Feature Level Analysis

To further explore how this parallel architecture encouraged convergence (as opposed to the baseline), we looked at the final feature maps between the two branches. Upon examining the addition of individual feature maps, we saw that the parallel architecture outputted similar feature maps between branches.

Feature Maps for the outputted number 7



Notice that in the images above we observe different shapes for each feature map around the general shape of a 7 (the final outputted number is on the left of the diagram above). We think that having two similar branches allows for different representations for a similar model to occur and therefore encourages convergence.

5 Conclusion and Future Work

Our project was a tantalizing foray into the world of GANs. Although the joint training of two adversarial networks is notoriously difficult (e.g. back-propagating errors through both networks exacerbates the vanishing gradient problem and both networks must be at comparable performance levels for learning to progress smoothly), we were able to get convincing results with even simple fully connected networks with no spatial awareness. In order to fix the artifacts and blurriness of the images generated by our most simple networks, we used more complex CNNs with far more parameters and with spatial awareness. The images generated using DCGAN-inspired networks yielded crisper images with little to no artifacts.

There were many things we didn't have enough bandwidth to implement. Future work would include doing conditional generation and using higher resolution datasets to see how the skip connection results accumulate over various scales. Another promising avenue of exploration would be the exploration of the latent space learned by the generator. For instance, we could discover which portions of the latent space belong to which numbers and whether there exist semantic directions (e.g. boldness and slant) in the latent space that would allow us to tweak the outputs generated.

6 Contributions

1. Madeleine handled all the parts related to convolutional generators/GANs, including implementing the convolutional GANs, training metric monitoring and feature map analysis.

2. Kevin implemented the fully-connected GAN and training infrastructure including training metric monitoring, restarting training from checkpoints, and conditional generation.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[4] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[5] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.