CMSC498L Final Project Report: Generative Adversarial Networks and Their Applications

Andrew Mao, Aman Jaiman, Jagan Prem
Department of Computer Science
University of Maryland
College Park, MD 20742

Abstract

We review the basic theory of generative adversarial networks (GANs), their motivation and potential applications. We discuss the formulation of the optimization problem and proof of convergence. We explain the problems associated with training GANs, their cause and attempts to remedy them. We then turn to deep convolutional GANs (DCGANs) to look at specifically image generation with convolutional layers. We examine the various guidelines of the model to ensure more stable training. Lastly, we implement a variant of DCGAN on CIFAR-10, STL-10, and a custom dataset of Nintendo Smash Spirits. We discuss the results, including an ablation study on the hyperparameters.

1 Introduction

Generative adversarial networks (GANs) are part of the class of generative models, which attempt to model the joint probability distribution $P(X, Y)$. GANs were first proposed by Goodfellow et al. [2014]; they proposed pitting the generative model against a discriminator, which learns to distinguish real from fake data samples. Hence, the generator and discriminator can be thought of as adversaries playing a zero-sum game.

Later, Radford et al. [2015] discusses using the CNN architecture with GANs (Goodfellow et al. [2014]) to build image representations. In their research they show how certain constraints on the historical Convolutional GAN can make them more stable for training. They call this new architecture Deep Convolutional GANs (DCGAN).

We decide to implement our own DCGAN in PyTorch to deepen our learning of generative networks. In doing so, we wanted to understand the architecture of a DCGAN, learn the constraints that make them stable in training, modify the architecture and hyperparameters to achieve better results, and lastly test with our own data to observe different results while making the content relevant to our interests.

1.1 Motivation

Generative models are useful in understanding the ability of models to represent and generalize their inputs. They are a powerful tool in semi-supervised learning. In vision, image synthesis has applications for data augmentation, style transfer, and artwork.

Other generative models include Autoregressive models (PixelCNN), Deep Belief Networks (DBNs) and Variational Autoencoders (VAEs).

Along with testing with some "standard" datasets, we thought it would be cool to generate some new images based on some of our interests. We decided to use a newly assembled dataset of video game

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

sprites. The video game sprites came from Super Smash Bros. Ultimate (SSBU), a video game by Nintendo, and the sprites are referred to as "spirits."

2 Data

We worked with three different datasets for this project: CIFAR-10, STL-10, and a newly created spirits dataset.

The CIFAR-10 and STL-10 datasets were included in PyTorch's torchvision library. We decided to work with these datasets because of familiarity, ease of accessibility, and the amount of images they contain.

We also decided to work with SSBU spirits because of our familiarity with them in the game, and because we found it to be a new application to DCGAN generation. To get the spirits dataset, a web scraping script was written to download all 1395 images from http://smashbros-ultimate.com/spirits. The images were kept in a directory and accessed using a custom PyTorch dataloader.

3 Related Work

3.1 Paper 1: Introduction of GANs

Goodfellow et al. [2014] are the first to propose GANs in the modern form. They formulate the loss function as a two player minimax game, and present an algorithm to optimize it. They show the loss function has a global optimum when the generator distribution equals the data distribution. They accomplish this by solving for the optimal discriminator, substituting it into the loss function and rewriting it in terms of JS divergence. The authors also show that their algorithm converges to this optimum.

The authors then perform some experiments on small datasets, such as MNIST, the Toronto Face Database, and CIFAR-10. They use kernel density estimation on samples from generator to estimate its pdf, then measure the log-likelihood (probability) of the real (test) data under it.

3.1.1 Notation

Let p_g be the distribution learned by the generator G, p_r be the distribution of the real data, and p_z be the input distribution to G, usually Gaussian noise.

Let $D(x)$ be the discriminator's confidence that x came from p_r instead of p_g.

D and G are typically deep neural networks.

The goal of the discriminator D is to maximize the log likelihood of real data, $\mathbb{E}_{x\sim p_r(x)}[\log D(x)]$. At the same time, we want $D(G(z))$ to be close to zero, so D also tries to maximize $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$. On the other hand, we want G to minimize the same loss, so minimize $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$.

The generator and discriminator optimize the following minimax loss function:

$$\begin{aligned}\min_G \max_D L(D, G) &= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \\ &= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log(1 - D(x)]\end{aligned}$$

3.1.2 Limitations and Extensions

The authors note a number of advantages and disadvantages with GANs. First, GANs do not explicitly represent the data distribution, instead implicitly representing it using a deep neural network. They also note the problem of mode collapse (which they call the "Helvetica scenario"), where the generator gets stuck creating images with low diversity.

Another limitation of the paper is how to evaluate GANs. The authors used kernel density estimation to approximate the generator distribution, but note that it doesn't perform well in high dimensional spaces. Since exact likelihood estimation isn't tractable and visual examination is not rigorous, an alternative metric for evaluation should be developed.

3.2 Deep Convolutional GANs

Radford et al. [2015] discusses using the CNN architecture for unsupervised learning by using GANs (Goodfellow et al. [2014]) to build image representations. In their research they show how certain constraints on the historical Convolutional GAN can make them more stable for training. They call this new architecture Deep Convolutional GANs (DCGAN).

The first change they make is by replacing spatial pooling functions with strided convolutions in both the discriminator and the generator. Next, they remove fully connected hidden layers from the CNN architecture. Then they implement batch normalization in the discriminator and the generator, which they found to alleviate problems due to poor initialization. The authors also found that using Tanh activation in the output layer of the generator, and ReLU activation elsewhere, was able to achieve faster learning. For the discriminator, LeakyReLU activation worked well in all layers.

The authors trained DCGANs on three image datasets: Large-scale Scene Understanding (LSUN), Imagenet-1k, and a Faces dataset created by the authors. The generator of the model is able to transform a vector to a 64x64 pixel image by learning the representation of the dataset.

4 Our Implementation

We modify code from Nathan Inkawhich, which may be found at

`https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html`

The architecture is a standard DCGAN. The generator repeats a common module: transposed convolution, followed by batch normalization, followed by ReLU. This is different for the final activation, which is tanh. The discriminator uses a different module: strided convolution, then batchnorm, then leaky ReLU. The final activation is sigmoid.

The first test is on CIFAR-10, which consists of 60K 32*32*3 RGB images. The input to the generator is a Gaussian noise vector of size 100. The architecture is four layers of transposed convolutions (stride 2, padding 1) followed by batchnorm. The channel depth starts at 32*8=256, then repeatedly halves until reaching 3 for RGB. The kernel size is always 4. The discriminator is similar to the generator in reverse: 4 modules, repeatedly doubles the channel depth, kernel size 4, stride 2, padding 1. The final activation is sigmoid.

For weight initialization, the paper specifies drawing from a normal distribution with mean=0, stdev=0.02.

For training, Two separate Adam optimizers are used for each network (with learning rate 0.0002 and Beta1 = 0.5). Following tips from ganhacks (Salimans et al. [2016]): mini-batches will contain either all real or all fake images.

To help compute our losses, we use Pytorch's BCELoss (binary cross entropy loss), which is defined as

$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -\left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)\right]$$

BCELoss contains both log components in the objective function. We can use the label y (0 or 1) to select which component is used.

For the discriminator, batch of real samples is used to calculate $log(D(x))$, and a batch of fake samples is used to calculate $log(1 - D(G(z)))$. The gradients are accumulated for each pass, before updating.

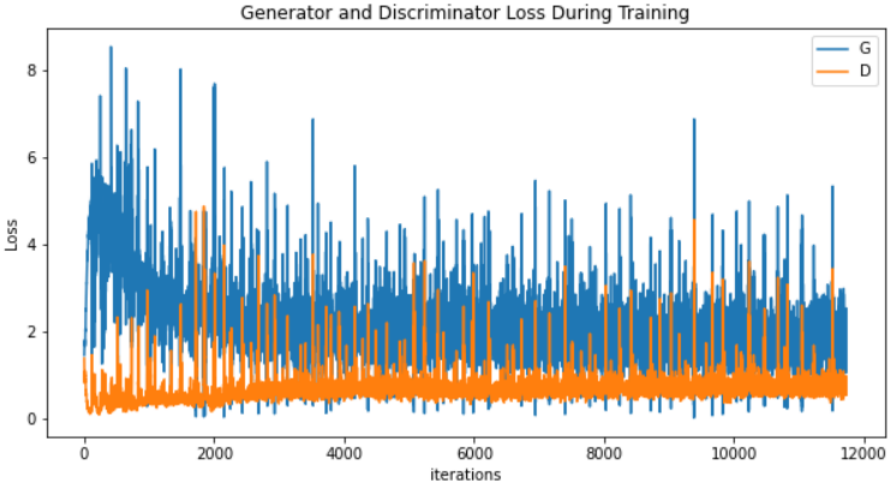
G will maximize $log(D(G(z))$, as mentioned by Goodfellow, which provides a stronger gradient signal.

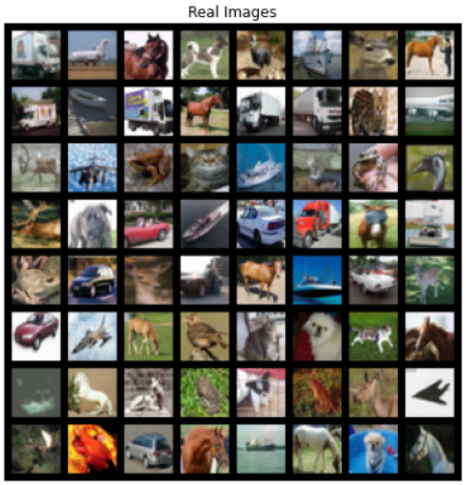
We train for 30 epochs, using mini-batches of 128 images. We plot the loss of both D and G over time, and save the generated images.

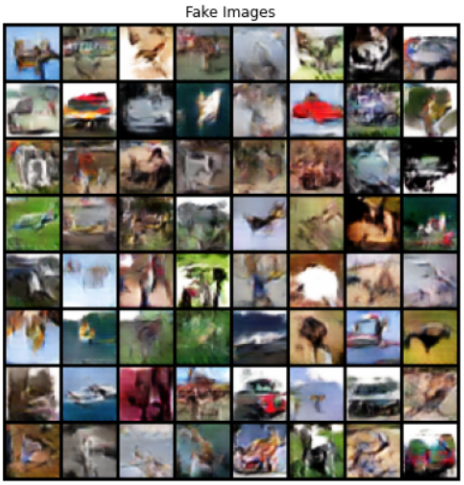
We also use this model on the STL-10 and spirits dataset.

5 Experiments and Results

Our model generated compelling samples on CIFAR-10 and STL-10. Images show diversity in color, form and shape, and sometimes can be mistaken for real images. Separation of foreground and background is clear, showing a recognition of objects. No mode collapse is observed. However on close inspection, the generated samples still fail to look as clear and identifiable as real images. We theorize that the diversity and different classes of the datasets make it difficult for the generator to disentangle their representations; conditioning on class labels should lead to quality improvements.







6 Discussion

6.1 Ablation Study

Getting GANs to converge during training was difficult. To understand the sensitivity, we conducted an informal ablation study on the model architecture. We tested tweaking certain hyperparameters and their effect on the generated samples and loss functions.

- Use ReLU instead of Leaky ReLU in the discriminator
- remove all batchnorm layers
- Use the default Adam optimizer parameters (lr=.001 instead of .0002, beta1=0.9 instead of 0.5

The most impactful change was tweaking the optimizer parameters - the model completely failed to converge, both visually and with the loss function. Removing batchnorm also had a significant impact - the images were noticeably "washed out" of color, and the objects were degraded in quality. We observe this may be an instance of mode collapse - this is supported by the low loss in the generator, and the high uncertainty of the discriminator (can't tell real from fake).

Removing Leaky ReLU had the smallest impact on the visual quality, but interestingly hurt the training time. The model took more than double the time to converge (200s/epoch v 90), and the images were noticeably degraded in quality (many of them also had green artifacts).

6.2 Scaling Up

After seeing promising results on CIFAR-10 and STL-10, we wanted to scale up to a larger dataset of a single class. This turned out to be much harder than expected. We looked into ImageNet, but the entire dataset size was enormous, required requesting access, and it wasn't easy to obtain a single class.

We tried the LSUN bedroom dataset. We ran into numerous problems: the zip file (42 GB compressed) wouldn't extract properly on Colab, or it would partially extract, exceed the disk quota and the resulting lmdb database was corrupted.

We tried LSUN classrooms which was much smaller (3 GB); this time we suceeded in downloading and extracting, but trying to read the file would cause I/O errors; we only got it working in the CPU runtime, but then training became unbearably slow.

Finally we tried Celeb-A, but ran into I/O errors again; we eventually discovered that Drive has trouble dealing with a large number of files in a single directory (100K+), and will even miss finding files completely. Interestingly, Celeb-A was the only dataset to use this directory structure.

In the end this cost a couple days. So lessons learned, big data makes everything harder, and free resources like Colab have their limitations.

6.3 Difficulties

We observed the loss functions of G and D oscillated wildly during training, with G's loss generally decreasing in the early stages of training. Hence it is difficult to observe convergence or debug when things go wrong.

We theorize that the diversity and different classes of the datasets make it difficult for the generator to disentangle their representations; conditioning on class labels should lead to quality improvements. To test this we trained the same model on a single class of CIFAR-10 (cat), for 50 epochs. The results were not promising - images were blurry and barely resembled cats. We believe that this can be improved with more data and a higher resolution. We also theorize that a full conditional GAN would be able to transfer learn features from other classes (like dogs) and hence generate better quality images.

6.4 Spirit Generation

To wrap up our testing, we used our model to generate the SSBU spirits. Learning curves and generated samples are included in the appendix. We noticed that while the generator was able to pick up on key features like shape, foreground coloring, and background coloring, the images were much more pixelated than the samples generated on CIFAR-10 or STL-10, and not as indicative of actual spirits. This is likely a result on the lack of data, only 1395 training images, compared to the other data sources. The results did not show any mode collapse. Another thing to notice is that the images seemed to have higher contrast, which wasn't as prevalent in testing with CIFAR-10.

7 Future Work

There were many tasks that we were unable to complete, or wanted to complete if given more time.

- Train the model on larger datasets like LSUN and Celeb-A. Due to the aforementioned issues with Colab, we would need to set up a local environment to train, use a different

environment like GCP, or split the dataset directory into multiple directories and write a custom Dataloader.

- Implement more advanced GANs, in particular conditional GAN, WGAN-GP and Progressive GAN. This was the goal after training on a larger (higher-resolution) dataset, where the difference in sample quality may become more apparent.
- Quantify the quality of generated samples using Inception score or FID.
- Train other generative models, in particular PixelRNN and VAEs, and compare results against the DCGAN.
- Experiment with the latent space, like interpolation and vector arithmetic. Again, we only wanted to try this after obtaining samples of sufficiently high quality to be clearly distinguishable.
- Use the discriminator in a classification task as mentioned in Radford et al. [2015].

8 Conclusion

In this paper we discussed the basic theory behind GANs and DCGANs, and implemented one in Pytorch. We experimented with different datasets and trained GANs to varying levels of success. Through an ablation study, we showed the importance of good hyperparameters, as well as the fragility of GAN training. We were glad to have gotten working models that can generate realistic images, and especially glad to get something working for a custom dataset to show the flexibility of learning GANs can achieve (just 1395 examples!). GANs have shown great potential to be very effective generative models from their outset and continuing research on them has discovered increasingly more ways to optimize their performance. Further research will undoubtedly continue into GANs to overcome more of their shortcomings.

References

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2234–2242. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf`.

9 Appendix

You can see our full presentation at https://go.umd.edu/SmashSpiritsWithGANS

9.1 Smash Spirits GAN, learning curve and generated samples