Application of GANs and DCGANs

STA 561 Final Project

Jiayi Ding jd402, Steve Shao ls362, Yangfan Ren yr47

April 28, 2019

Abstract

This report summarizes our implementation of generative adversarial networks(GANs) to train a generator model that can create fake human-looking letters/digits from the EMNIST dataset. In this paper we will discuss: (1) an introduction to GANs and recent related works on the extension of GANs, (2) a detailed explanation on the methodology of GANs and deep convolutional generative adversarial networks (DCGANs), (3) a comparison between the results from GAN and DCGAN trained on the dataset. Finally the report contains our thoughts on GANs and its usefulness in real-life application.

Key words: Generative Adversarial Nets, Deep Convoluted Neural Networks, MNIST dataset, Deep Learning, Computer Vision

1. Introduction

Generative adversarial networks (GANs) are generative models devised by Goodfellow et al. in 2014[1]. It provides a way to learn deep representations without extensively annotated training data, by deriving back propagation signals through a competitive process involving a pair of networks. Specifically, the two adversarial parts are a generator that generates images and a discriminator that tries to determine whether images are real or generated. A good analogy of the generative model is a team of counterfeiters, trying to create fake currency and use it without detection, while the discriminative model can be thought as the police, trying to detect the counterfeit currency. GANs are useful in a wide variety of applications, including image super resolution, image synthesis, semantic image editing, classification, and style transfer.

Deep Convolutional Generative Adversarial Networks (DCGANs), first introduced by Radford et. al. in 2015[2] is a direct extension of GANs. The discriminator is explicitly made up of strided convolution layers, batch norm layers, and LeakyReLU activations. The generator is comprised of convolutional-transpose layers, batch norm layers, and ReLU activations. Compared to GAN, DCGAN specifically focuses on using Deep Convolutional networks in place of those fully-connected networks. Since convolutional nets find areas of correlation within an image, i.e. spatial correlations, DCGAN is likely to be more fitting for image data, and generate more stable and higher quality image.

2. Related Work

Currently, there are several different extensions and improvement of GANs. Conditional Generative Adversarial Networks (cGANs) introduced by Mirza et. al. in 2014[3], are conditional versions of generative adversarial nets, where both generator and discriminator are conditioned on either class label or data from some other modality. These GANs use extra label information and result in better quality images are able to control how generated images will look. Stacked Generative Adversarial Networks (StackGANs) brought by Zhang et. al. in 2017[4] are proposed as a solution to the problem of synthesizing high-quality images from text descriptions in computer visions. StackGANs decompose the hard problem into more manageable sub-problems through a sketch-refinement process. Information. InfoGAN is an information-theoretic extension to GAN, first introduced by Chen et. al. in 2016[5]. InfoGAN, especially useful when data is very complex, and one would like to see the most important features, maximizes the mutual information between a small subset of the latent variables and the observation. Wasserstein GANs(WGAN), a variation of GANs that has got a lot of attention introduced by Arjovsky et. al. in 2017[6]. It minimizes a reasonable and efficient approximation of the Earth Mover's distance, and can be helpful in the pursuit of dimensionality reduction.

3. Methodology

3.1. Adversarial Nets

The following methodologies are from the paper "Generative Adversarial Nets" by Goodfellow et al.(2014)[1]. We can define a prior on input noise variables $p_z(z)$ and provide a mapping to the data $\boldsymbol{x}$ as $G(z;\theta_g)$, where G is a differentiable function related to a multilayer perceptron. By this way, we can obtain the generator's distribution p_g. Furthermore, we need to define another multilayer perceptron $D(x;\theta_d)$ which represents the probability that $\boldsymbol{x}$ come from the data instead of p_g. Then we train D in order to maximize the probability that the labels are correctly assigned to training data and samples from G. In the meantime, we train G to minimize $\log(1-D(G(z)))$. Generally speaking, we present a minimax game of D and G as followed:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))].$$

3.2. Theoretical Results

3.2.1. Global Optimality of $p_g = p_{data}$

First, given any generator G, we consider the optimal disriminator D. We can obtain this by maximize the quantity $V(G,D)$:

$$\begin{aligned} V(G,D) &= \int_x p_{data}(x)\log(D(x))dx + \int_x p_z(z)\log(1-(D(g(z))))dx \\ &= \int_x p_{data}(x)\log(D(x))dx + p_g(x)\log(1-(D(g(x))))dx \end{aligned}$$

For any $(a,b)\in\mathbb{R}^2\ 0,0$, the function $y = a\log(y) + b\log(1-y)$ avhieves its maximum in $[0,1]$ at $\frac{a}{a+b}$. Then the minimax game can be interpretated as:

$$\begin{aligned} C(G) &= \max_D V(D,G) \\ &= \mathbb{E}_{x\sim p_{data}}[\log D^*(x)] + \mathbb{E}_{z\sim p_z}[\log(1-D^*(G(z)))] \\ &= \mathbb{E}_{x\sim p_{data}}[\log D^*(x)] + \mathbb{E}_{x\sim p_g}[\log(1-D^*(x)] \\ &= \mathbb{E}_{x\sim p_{data}}[\log \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}] + \mathbb{E}_{x\sim p_g}[\log \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}] \end{aligned}$$

that is, for fixed G, the optimal discriminator D is $D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$. We can then obtain Theorem 1 as followed. The proof can be referenced in the paper by Goodfellow et al.

Theorem 1. The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$ and $C(G) = -\log 4$ at that point.

3.2.2. Convergence of Algorithm

We must use an iterative and numerical method to implement the minimax game practically. The procedure is presented in Algorithm 1.

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 $z^{(1)}, \dots, z^{(m)}$ from noise prior $p_g(z)$.
     • Sample minibatch of $m$ examples $x^{(1)}, \dots, x^{(m)}$ from data generating distribution $p_{data}(x)$.
     • Update the discriminator by ascending its stochastic gradient:
           $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\, \log D(x^{(i)}) + \log(1 - D(G(z^{(1)})))]$.
   end for
   • Sample minibatch of $m$ noise samples $z^{(1)}, \dots, z^{(m)}$ from noise prior $p_g(z)$.
   • Update the generator by descending its stochastic gradient:
           $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(1)})))$.
 end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
The paper shows that given G, the discriminator will reach its potimum at each step of Algorithm 1 if G and D have enough capacity, and p_g is updated to improve the criterion

$$\mathbb{E}_{x \sim p_{data}}[\log D^*_G(x)] + \mathbb{E}_{x \sim p_g}[\log(1 - D^*_G(x))],$$

then p_g converges to p_{data}.

3.3. Deep Convolutional Generative Adversarial Networks(DCGANS)

DCGANG is a class of CNNs that have certain architectural constraints. It is one of the popular and successful network design for GAN. It mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling.

The paper written by Radford et al.(2016)[2] presents the architecture guidelines for stable Deep Convolutional GANs are as followed:

• Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).

• Use batchnorm in both the generator and the discriminator.

• Remove fully connected hidden layers for deeper architectures.

• Use ReLU activation in generator for all layers except for the output, which uses Tanh.

• Use LeakyReLU activation in the discriminator for all layers.

In Figure 1, a 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used[2].

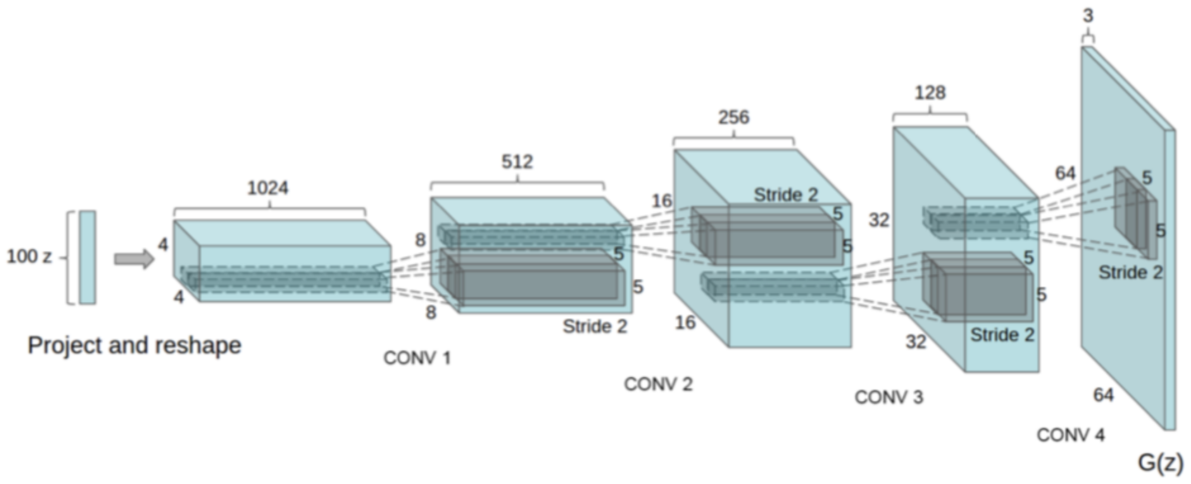

Figure 1: DCGAN generator used for LSUN scene modeling.

4. Implementation

We implemented in Python the algorithm of Generative Adversarial Nets and Deep Convoluted Generative Adversarial Nets based on pytorch and tensorflow. Specificall, we proceed as the following steps:

- Take the training set of MNIST data and preprocess the data by stretching the gray matrix (28×28) into a long vector ($28^2 \times 1$) as the features of the image.
- Define a function to randomly split the data into minibatches.
- Construct a neural network in pytorch as our generator, which takes a random noise as input and through the neural network spits out a feature vector ($28^2 \times 1$) that could potentially represent a image of "artificially generated" handwritten digit.
- Construct a neural network in pytorch as our discriminator, which takes the features of a image and discriminate whether the image is a real data or "artificially generated" data by giving a probability of being real.
- Compute the cross entropy of classifying both the real data and fake data in each iteration based on a minibatch we obtrain and backpropogate the error signals to update the weights in both our genrator and discriminator.
- Repeat the last step untill we run out of minibatches and we call it an epoch. Train the GANs/DCGANs for many epoches until the loss (cross entropy) converges and Output generator.

After we trained the generator, we could have a generative neural network that can potentially give us very perfect fake data which can be arbitrarily close to the original data if everything goes right.

The codes implementation by python are available in our Github repository.

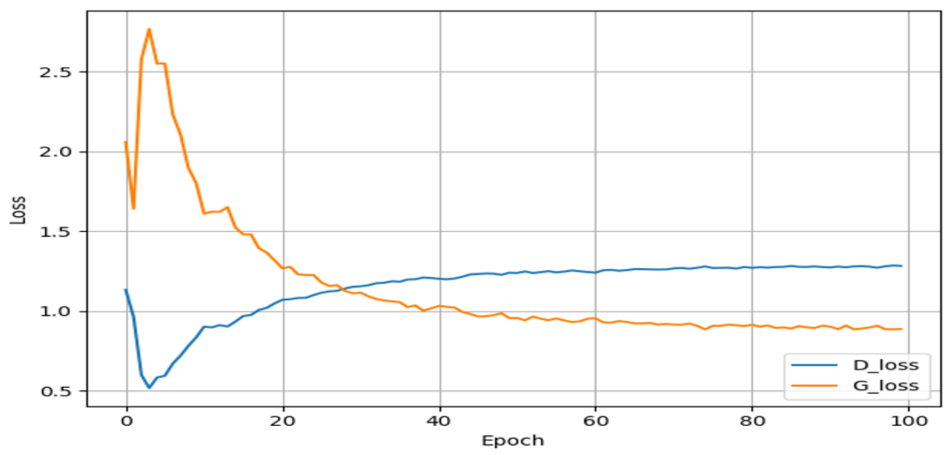

Figure 2: Loss by epoch in GANs

5. Results and analysis

For our project, we would like to do a comparative analysis between GANs and DCGANs to test their capability of creating image generative models in general. In terms of the main results we have, we trained 2 image generative models using GANs and DCGANs on the MNIST data. Below is the Loss by epoch figure in GANs

As the Figure 2 shows, the loss for discriminator and generator have converged to some stable state where the generator does a quite decent job of generating the fake data from a random noise which the discriminator can not distinguish the true data from. The loss plot is very similar in DCGAMs, whereby it takes less epochs to reach to a more efficient (lower loss for both generator and discriminator) stable state.

A more convincing result would be the "fake data" we generated. Ideally, by either of both models, we are supposed to have a generator which takes a random noise as input and generate images that are like real ones written by a human being.

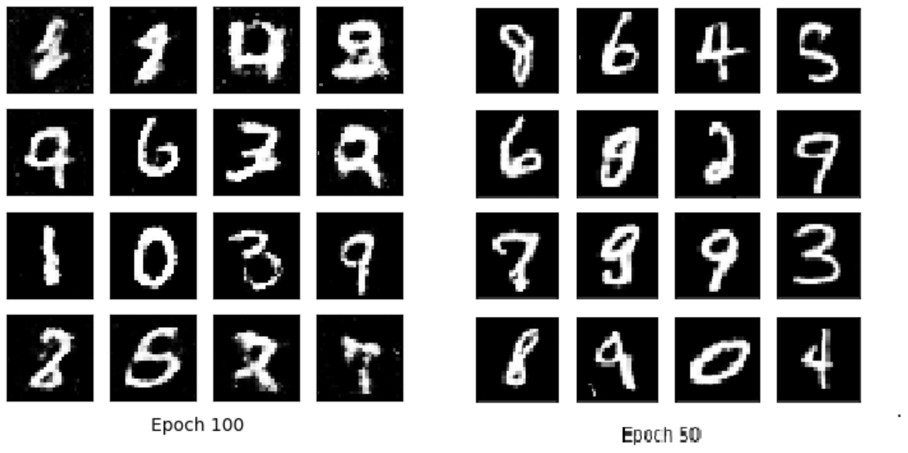

Figure 3: Generated Digits Images. Left: GANs, right: DCGANs

Above is our main result, where the left image is generated by the generator from GANs after 100 epochs and the right image is generated by the generator from DCGANS after 50 epochs. As expected, the result from DCGANs outperforms that from naive GANs (even with half the number of epochs) since we used convoluted neural networks as our discriminator and generator which are generally more powerful in image recognition problems.

As the figure shows, the generatorq from DCGANs can generate figures with clearer edges and also with more implicit meaning (from which we can tell what number it is). Generally, DCGANs would be more fitting for image data, whereas the general idea of GANs can be applied to wider domains, as the model specifics are left open to be addresses by individual model architectures.

6. Conclusion and discussion

In this project, we went through the basics of GANs and DCGANs and implemented them in python on the MNIST data and trained 2 image generative models to generate handwritten digit images.

In our attempts to train image generators with GANs, only the handwritten digits images were used, which are essentially gray scale matrix data with only one color channel. But more often in real world scenarios, we would have to deal with colored images, which might require more sophisticated architecture of our generator and discriminator as well as tricks like one-sided label smoothing and reference batch normalization to get desirable results. We will continue working on examples of more complicated colored images and use more tricks to optimize the model.

Reference

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

[2] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

[3] Mirza, M., & Osindero, S. (2016). Conditional Generative Adversarial Nets. arXiv:1411.1784

[4] Zhang, H., Xu, T., & Li, H. (2017). StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks. 2017 IEEE International Conference on Computer Vision (ICCV). doi: 10.1109/iccv.2017.629

[5] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. (2016, June 12). InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. arXiv: 1606.03657

[6] Arjovsky, M., & Leon, S. (2017). Wasserstein GAN. airXiv:1701.07875

[7] Goodfellow, I. (2016). NIPS 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160.

[8] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative adversarial networks: An overview. IEEE Signal Processing Magazine, 35(1), 53-65.

[9] Murphy, K. P. (2012). Machine learning: a probabilistic perspective. MIT press.