Analyze GAN's performance in custom celebrity dataset

Masnoon Nafees

Department of Information Systems

University of Maryland, Baltimore County

masnoon1@umbc.edu

ABSTRACT Generative Adversarial Network (GAN) is relatively new topic in the field of Machine Learning. The basic idea behind GAN is having two neural networks. One of them works as a discriminator and other is generator. The goal for the GAN is a two-player game where generator's task is to create something similar to actual dataset to fool the discriminator and discriminator's task is to identify whether the data is actually coming from generator or discriminator. After proposing GAN by Ian Goodfellow in 2014, several other types of GAN are proposed. For example, Wasserstein GAN, DCGAN, Wasserstein GAN with Zero Gradient Penalty, LSGAN, StyleGAN etc. in this project, I tested a dataset into GAN, DCGAN and WGAN and analyze their performance. While analyzing I have observed that GAN works better on Grayscale Images but not very well in color images. On the other hand, DCGAN generate more natural images where WGAN failed to generate images for constant gradient crushing towards zero. Rest of the paper will state the motivation, problem statement, dataset, experimental part and conclusion.

KEYWORDS

GAN, Face Generation, Overfitting, Accuracy, Loss Function, WGAN, MNIST

Motivation :

The real motivation for doing this project is to explore the new technologies and innovations in the field of Artificial Intelligence. GAN is a new architecture in Deep Learning. There are lots of area yet to be explored and also keep in minds that GAN's aren't perfected yet. There are several applications can be done with GAN like generate fake images, text, cartoon transformation, style images etc. Though this project, we can easily generate someone's look alike face instead of going for photoshop and all the stuff. This heavily reduces cost and resources. Also, this can be used as a security purposes like to detect someone who is having a disguise or forensic material whether the image is computer generated or not.

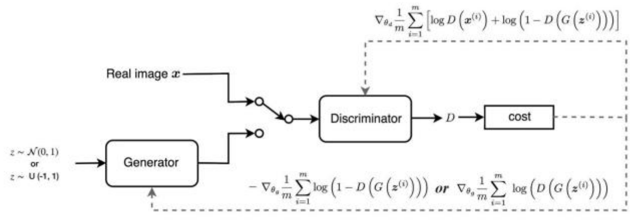
$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(x^{(i)}\right)+\log\left(1-D\left(G\left(z^{(i)}\right)\right)\right)\right]$
Real image x
Discriminator
D
cost
$z\sim\mathcal{N}(0,1)$ or $z\sim U(-1,1)$
Generator
$-\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(z^{(i)}\right)\right)\right)$ or $\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(D\left(G\left(z^{(i)}\right)\right)\right)$

Figure 1 : GAN Architecture

From the Figure 1[1], we can see the architecture of GAN. Discriminator takes data from real dataset and labeled it as 1 where Generator produce random noise from distribution mostly Gaussian and labeled as 0. To trick the discriminator, fake data has been pushed to label as 1. Each iteration, loss has been adjusted for both network. In this project, DCGAN architecture has manage to generate image after few hundreads iteration.

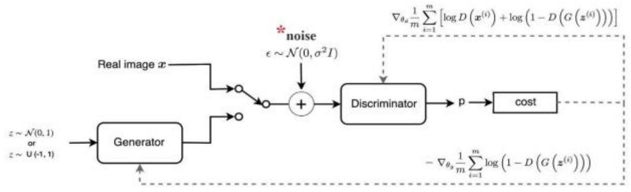
$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(x^{(i)}\right)+\log\left(1-D\left(G\left(z^{(i)}\right)\right)\right)\right]$
*noise
$\epsilon\sim\mathcal{N}(0,\sigma^2 I)$
Real image x
+
Discriminator
p
cost
$z\sim\mathcal{N}(0,1)$ or $z\sim U(-1,1)$
Generator
$-\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log\left(1-D\left(G\left(z^{(i)}\right)\right)\right)$

Figure 2 : WGAN Architecture

Figure 2 [2] is a WGAN archtecture which is diffetent than GAN. WGAN used a linear activation function in the output layer of the critic model (instead of sigmoid). For label, -1 labels for real images and 1 labels for fake images (instead of 1 and 0). Wasserstein loss to train the critic and generator models. Constrain critic model weights to a limited range after each mini batch update (e.g. [-0.01,0.01]).

Critic model has to be updated more times than the generator each iteration (e.g. 5). And RMSProp version of gradient descent with a small learning rate and no momentum (e.g. 0.00005).

Problem:

The problem I tried to overcome in this project is to build a data preprocessing model which can take input user's directory and process into network model. Image manipulation is hefty and time consuming. You have to use 3rd party software like photoshop which aren't free. Another important aspect of the project is about finding out the best combination of GAN. After first introduced in 2014, there lot's GAN are introduced like WGAN, WGAN-ZP, DCGAN, LSGAN etc. I experimented the dataset I collect with DCGAN and WGAN and analyze their performance on custom colorful images. A simplistic way of generated image later can be used in various purposes.

Dataset:

The dataset I used for this project is from Facescrub. The FaceScrub dataset was created using this approach, followed by manually checking and cleaning the results. It comprises a total of 106,863 face images of male and female 530 celebrities, with about 200 images per person. As such, it is one of the largest public face databases. The images were retrieved from the Internet and are taken under real-world situations (uncontrolled conditions). Name and gender annotations of the faces are included [7]. For experimental purposes, I took some of the part of the dataset. 137 images of Actor Aaron Eckart have been used for the experiment. Most of the images are 300*300 color images. It will take huge amount of time if I can this on CPU. So, I used Google Colab's free GPU which Nvidia K80. Here are some images of the actor. The images are extracted a python script the dataset team and also faces are extracted at the same and saves the images in a directory.

Figure 4: Pictures of Aaron

Data Preprocessing:

Data preprocessing is the most challenging part of this project. It's mainly because images need to be converted into floating points then confined into 0 to 1. At first, I took all the images from the directory and append it in a list. Then I adjust the color by adding additional axis. After reshaping the image into 80*80 the shape of the array become (80,80,3). After that, the pixel points of RGB are converted into Float32. Then the Numpy array is ready for next step.

Methodology:

Methodology part was tricky. We need to compile two neural networks and work as a one. For this, I have taken help from several web sources [4]. First, we declare latent variable as 100. These latent points are the 100-dimensional vector. The generated images will be plotted on these later. Then, for discriminator model, layers are added. 5 layers with 128 neurons are being used. We have used transposed technique to downsample the images. For example, each layer the size of the images will be cut into half. As the architecture is derived from DCGAN, the network is convolutional neural network and fully connected. Dropout is set to 0.4. Padding was same and strides was (2,2). LeakyRelu and alpha = 0.2 was used. The last layer was single nodes with sigmoid activation function. For Generator, the model was identical with slight changes. The layers are designed for upsampling discriminators last output which was 5*5 images and convert into 80*80 with channel 3 color images. Activation function Tanh was used here.

After that, we set methods for fake sample generations and real sample generations. When compilation is complete, the epoch was set to 1000 and the batch was 15. Each 10 iteration, performance of the model was computed and also for the loss functions. Discriminator real, fake and Generators loss has been calculated and later that has been used to visualize the loss. Later, images and models are saved. Saved model can be later used for calculation which will skip the whole iterations again. For WGAN implementation, the architecture is slightly different. Instead of using Binary Cross Entropy, Wasserstein Loss are being used. WGAN uses more iteration on Critic (Discriminator) than Generator. WGAN gives the score instead of probability than GAN. Also, kernel constraint is set. In the next part, we will discuss the experiment.

Experiment:

Nvidia K80 GPU takes around 15 minutes to complete the processing of 137 images. This is the generated image for 100 iteration.

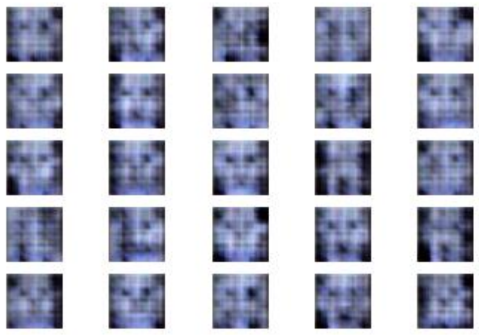

Figure 5: After 100 iteration

Here we can see the shapes are just taking. Loss and accuracy are calculated. The following image is after 200 iteration. We can see the face is getting identical of the actor. All the generated images have blue effect unfortunately. This will be tried to fix in future.

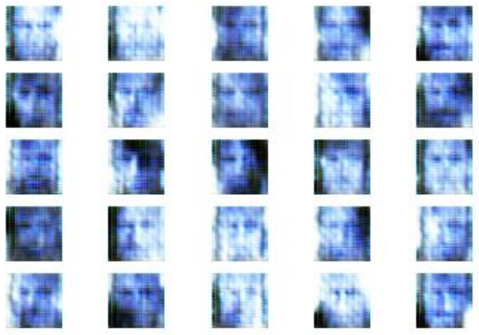

Figure 6: After 200 iteration

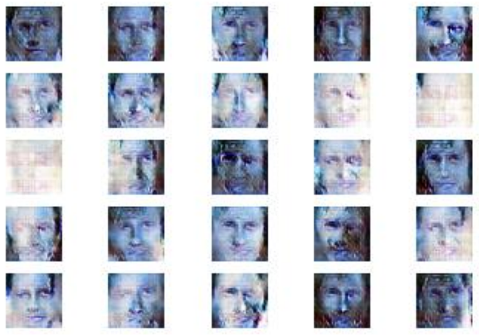

Figure 7: After 500 Iterations

After 500 iterations the images are taking into good shape. Though the images look ugly, but the computer is learning well to get close to actual image.

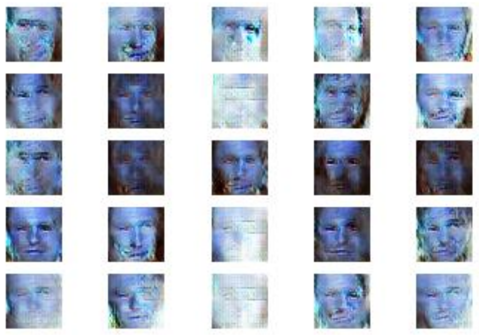

Figure 8: After 540 iteration

After 540 iterations, images are bit sharper and some of them are clearer.

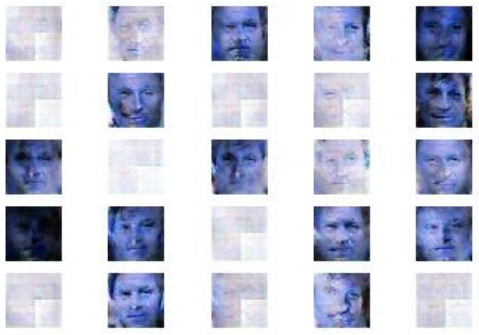

Figure 9: After 660 iteration

In this iteration, we can see the best possible generated images. Eyes, nose, face are clearer. Even some of the images have hair also. During the experiment, I observed that more iteration doesn't mean more clearer picture. This is the table for loss and accuracy.

Iteration	Dis(Real)	Dis(Fake)	Generator	Accuracy (Real)	Accuracy (Fake)
100	.407	.244	3.307	90%	88%
230	.813	.66	2.83	96%	78%
540	.020	.033	4.3	97%	96%
610	.341	.025	4.70	99%	98%
800	.004	.096	6.4	100%	100%

Figure 10: Loss and Accuracy Table

As we can see in the table, the loss for loss for Discriminator fluctuates but generator's loss started to increase. We observed that, there is a connection between batch size and epoch combination. In this experiment, around 600-700 iteration produce best images. After that losses are reduced for Discriminator and increased for Generator. And accuracy became 100 percent for both Real and Fake. This means model has already overfitted and nothing to optimize.

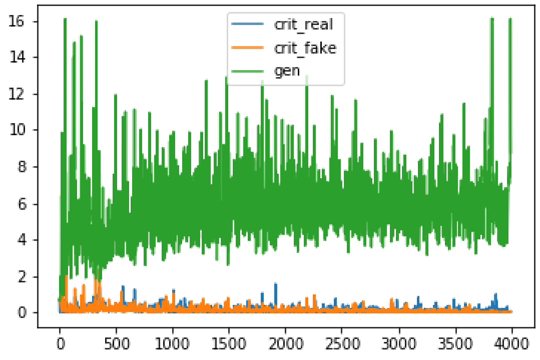

Figure 11: Loss

For loss visualization, X axis has been used to calculate all the iteration by accounting batch size needed to complete one epoch. Y axis is the value of loss.

So, after 700 iteration, the generated images are fading away.

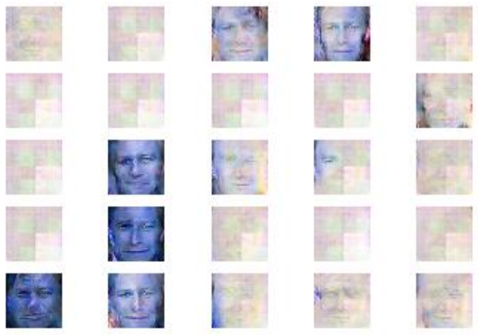

Figure 12: After 800 iteration

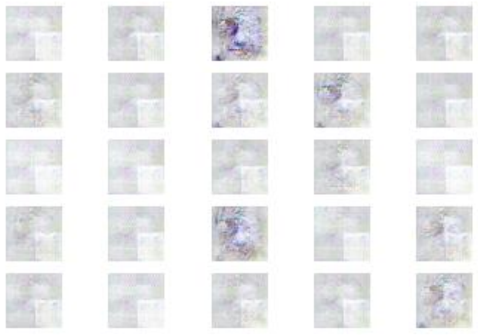

Figure 13: After 990 iteration

So, we can see in 990 iteration, the images are barely visible.

We tested the dataset in WGAN also. But the performance was so poor. WGAN doesn't even generate any images. The discriminator's loss got stuck into -1 and for generator it was 0. And the accuracy for both wasn't changing. The resulted images were just single color after

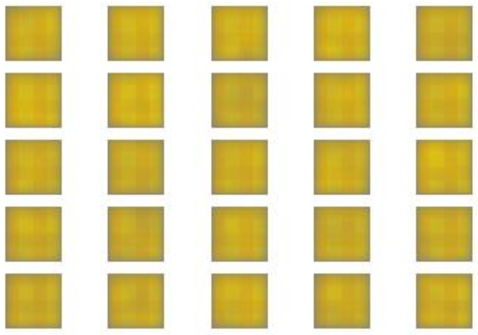

Figure 14: WGAN Image generation

But I ran the architecture in MNIST dataset which has grayscale image. It means the channel is set to 1. Both GAN and WGAN performs better on MNIST dataset but not on colorful image dataset. DCGAN performs best among them.

Observation:

After the experiment, we found out that batch size with number of epochs are important. Finding out the best combination may give best generated images. More images may give better result. During experiment, WGAN took more time than DCGAN. In Colab, DCGAN takes around 15 minutes to complete iteration while WGAN take around 20-22 minutes.

Conclusion

Gan's aren't perfected yet. Though, many novel researches are going on. From 2014's GAN, within few years we got WGAN, DCGAN, STYLEGAN, LSGAN etc. As WGAN clips the constraint some issues may arrived in some particular scenario. This issue has been tackled by WGAN-ZP. It is WGAN but with 0 gradient penalty. It means when the gradient is going towards 0 or already in 0, the model penalizes the penalty and push the gradient up. This is the one way to solve the issue. GAN's are getting better and better for Image Generation, Text Generation etc.

Future Work

For future work, I'd like to take more look on blueish color on generated image. Also, implementation of WGAN needed to change to get a work on Color images.

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. Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

3. Fang, W., Zhang, F., Sheng, V. S., & Ding, Y. (2018). A method for improving CNN-based image recognition using DCGAN. *Comput., Mater. Continua*, *57*(1), 167-178.

4. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in neural information processing systems* (pp. 5767-5777).

5. Hsu, R. L., Abdel-Mottaleb, M., & Jain, A. K. (2002). Face detection in color images. *IEEE transactions on pattern analysis and machine intelligence*, *24*(5), 696-706.

6. Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., & Paul Smolley, S. (2017). Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 2794-2802).

7. http://vintage.winklerbros.net/facescrub.html [Dataset]

8. https://machinelearningmastery.com