Generative Adversarial Networks: Part II

Nuha Mohammed

June 2019

1 Introduction

We will explore common problems in GAN training and take a deeper dive into a few different iterations of GANs that have been introduced after Ian Goodfellow's initial paper.

2 Review

Generative Adversarial Networks (GANs) consist of two neural networks: a generator and a discriminator. These two models are trained at the same time; the generator tries to recover the training set while the discriminator estimates the probability that the a given sample came from the training set rather than the generator. Ultimately, the generator learns to recover the training distribution and the discriminator outputs 0.5 for all samples, meaning it was equally likely to have come from the generator as from the training set.

3 Problems in Training

3.1 Non-convergence

The convergence of GAN training near an equilibrium can be assessed by looking at the Jacobian matrix of eigenvalues (i.e. the matrix of partial derivatives) at that point.

$$\upsilon(\theta, \psi) = \begin{pmatrix} -\nabla_\theta L(\theta, \psi) \\ \nabla_\theta L(\theta, \psi) \end{pmatrix} \tag{1}$$

The Nash equilibrium occurs at a specific parameter assignment where the neither the generator nor the discriminator will change its action regardless of what the other will do. Note that GANs may not always converge and this is where problems in training originate.

3.2 Mode Collapse

Mode Collapse occurs when there is no progress in optimization and all input images map to the same output image

3.3 Diminished Gradient

In many GANs, in the cost function, the gradient vanishes as the discriminator becomes optimal which means learning gradually becomes harder.

$$\nabla_{\theta_g} \log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \rightarrow \boldsymbol{0}$$

In a neural network, the gradient is integral for gradient descent to back-propagate the signal. Therefore, diminished/vanishing gradients make learning harder.

4 DCGAN

4.1 Strided Convolutions

Deep Convolutional Generative Adversarial Networks (DCGANs) use strided convolutions for the Discriminator and fractional-strided convolutions for the Generator (instead of pooling layers). You can refer to the CNN lecture to refresh your memory on strided convolutions which is a method of spatial downsampling. Using strided convolutions instead of pooling allows the network to "learn" its own downsampling, contributing to greater stability in training this GAN architecture.

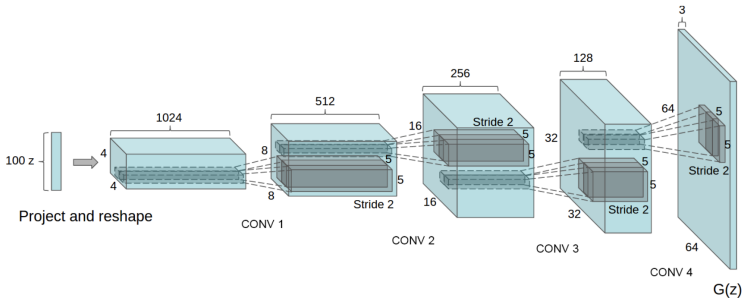


Figure 2. DCGAN Generator with strided convolutions for spatial downsampling.

4.2 Batch Normalization

Another feature of DCGAN is the use of batch normalization (batchnorm) in both the generator and discriminator. With batchnorm, the output of a previous activation layer is normalized by subtracting the batch mean from it and

dividing by the batch standard deviation. In practice, batchnorm helps prevent mode collapses and helps with gradient flow in deeper models.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

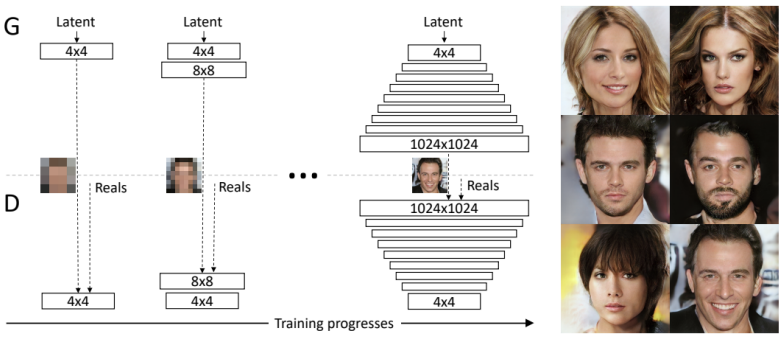
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

4.3 Activation Functions

DCGAN uses ReLU activation in all layers of the generator except for output layer which uses Tanh. The discriminator uses the leaky ReLU function which, in contrast to the regular ReLU function, allows small negative values to pass through (*ReLU truncates negative values to 0). Therefore, leaky ReLU helps with gradient flow in the discriminator.

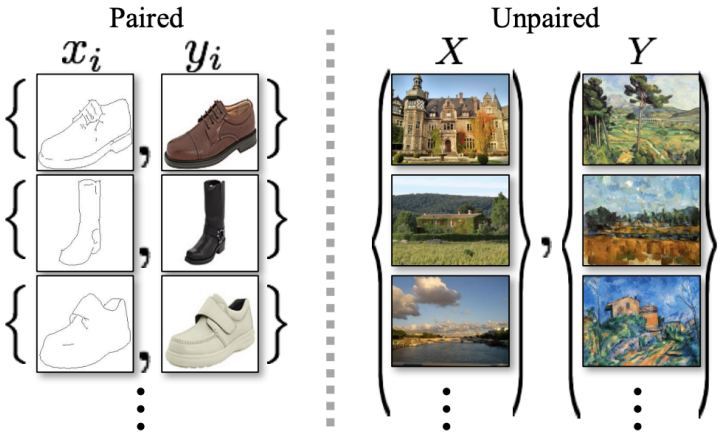
5 ProGAN

ProGAN was a major improvement to GANs by NVIDIA. It proposed the progressive growing of the GAN, a layer at a time. The GAN starts out by producing very low resolution images. Everytime training stablizes, a new layer is added and the resolution is doubled. This process continues until the desired resolution is reached. ProGAN provides a stable way of training GANs at higher resolutions.



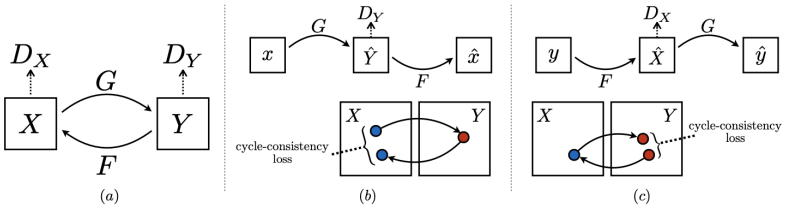
6 CycleGAN

CycleGAN is different from conventional GANs as it has to do with style transfer. Furthermore, CycleGAN deals with image-to-image translation with unpaired training samples which consist of a source set and target set with no pairs between the two datasets.



This type of mapping is prone to mode collapse. CycleGAN combats mode collapse by introducing a new loss function for "cycle consistency".

6.1 Loss Function



If we define a translator F and another translator G, the cycle consistency loss promotes F and G to be inverses of each other. It follows that F(G(X))x and G(F(x))y. This cycle consistency loss is combined with the adversarial loss. Adversarial Loss is expressed as that of a conventional GAN.

$$\begin{aligned}\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = &\mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] \\ &+ \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))],\end{aligned} \tag{1}$$

While, cyclic consistency loss is expressed as such:

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = &\mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] \\ &+ \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].\end{aligned} \tag{2}$$

Both are combined to yeild this loss function:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = &\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ &+ \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ &+ \lambda \mathcal{L}_{\text{cyc}}(G, F),\end{aligned} \tag{3}$$

and the GAN aims to solve the following:

$$G^*, F^* = \arg\min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y). \tag{4}$$

7 Challenge!

Complete the two challenges below!
1. Implement a DCGAN
2. Build a CycleGAN and use it to translate emojis between Windows and Apple emojis.

Instruction for both assignments are here:
https://www.cs.toronto.edu/ rgrosse/courses/csc$321_2$018/$assignments/a4-handout.pdf$

Shell Code here (scroll down to "Programming Assignment 4"):
http://www.cs.toronto.edu/ rgrosse/courses/csc$321_2$018/

8 Acknowledgements

I did not create any of the images in this lecture. I acquired images and information from the following sources:

- DCGAN paper
- CycleGAN paper
- ProGAN paper
- OpenAI's blog on GANs
- Medium's articles on GANs
- ML Club's GAN lecture

*The Challenge problem was taken from Univeristy of Toronto's CSC321 course.