INTRODUCTION - GENERATIVE ADVERSARIAL NETWORKS

In this lecture, we will be familiar with an advanced deep learning technique named Generative Adversarial Networks (GANs). This lecture note contains *background*, *application*, *concept*, *algorithm*, *architecture*, *tuning tips*, *potential problems*, *evaluation* of typical GANs. We only focus on two typical GANs: original GANs [1] and Deep Convolutional GAN (DCGAN [2]).

1. Background

Generative Adversarial Networks (GANs) is a cutting-edge technique of deep neural networks, which was first come up by Ian Goodfellow in 2014. In 2016, Yann LeCun (Facebook AI research director) described GAN as

"the most interesting idea in the last 10 years in Machine Learning."

GAN is a very new stuff and has a promising future. Especially in the recent two years, GAN was developed with an exponential increment (Fig.1). Although it is an infant technique, there are bunch of models proposed with the suffix "GAN", such as ACGAN, DCGAN, WGAN, BEGAN, CycleGAN, StackGAN. There is a website called "The GAN Zoo" (`https://github.com/hindupuravinash/the-gan-zoo`), where includes hundreds variants of GAN. Fortunately, we can still catch up the development of GAN now. Actually, GAN is not a complicated model because we have learned every single component in GAN if we break down it.

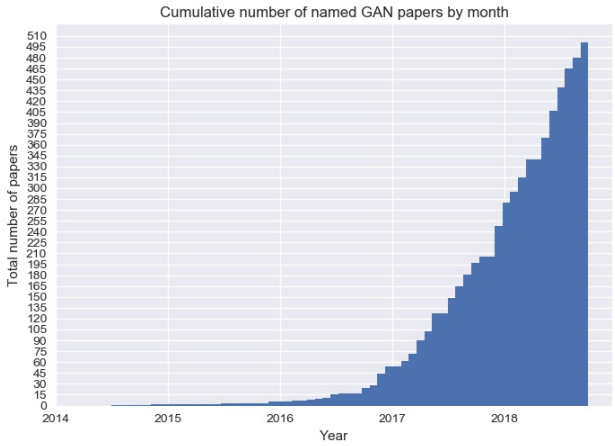

Figure 1. The GAN papers counts by The GAN Zoo

Date: September 7, 2020.
Author: Gaofeng Huang.
Institution: George Washington University.

2. Application

2.1. **Image Generation.** (Fig.2) Most of deep learning techniques we learned previously are used for recognition. GAN technique is going to let the machine learn to generate new stuff. To note, GAN does not simply memorize the given dataset. Generation is always harder than recognition. For example, we first learned how to recognize digits like 0-9, then we tried to mimic the shape of digits and created digits in our styles, which is called generation.
As discussed in *Chapter 20 of Deep Learning* [3], there are many other generative models, such as Restricted Boltzmann Machine (RBM), Generative Stochastic Networks (GSNs), and Variational auto-encoder (VAE). Comparing with these generative models, GAN has the state-of-the-art performance in the filed of image generation. In other words, GAN is the most powerful image generative model.

Figure 2. Generated facial images by BEGAN

2.2. **Image-to-Image Translation.** (Fig.3) GAN can learn the features of two image collections and translate one images from one domain to another. CycleGAN is one representative unsupervised GAN with such functionality. To note, CycleGAN does not require these two datasets to be paired. For example, in $Summer \rightleftarrows Winter$ translation, we need to collect photos of the same scene in different seasons. However, paired datasets are very expensive and often not available. Without paired datasets, CycleGAN can translate landscape photos into a particular painting style, such as Monet, Van Gogh. Besides, it can also translate zebras into horses.

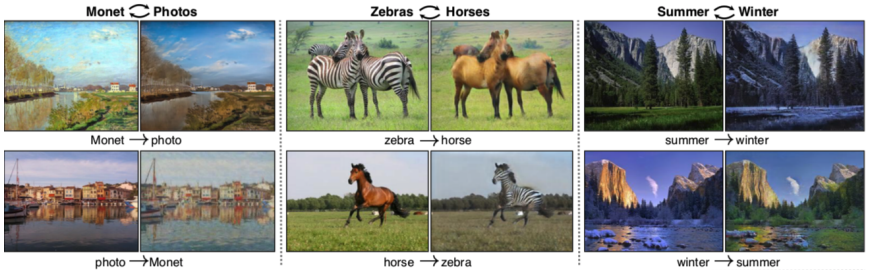

Figure 3. Image-to-Image translation by CylceGAN

2.3. **Text-to-Image Translation.** (Fig.4) If we embed text information as the label of corresponding images, GAN will learn the mapping between sentiment vectors and image features. StackGAN is a conditional GAN which can generate images based on text description.

FIGURE 4. Text-to-Image translation by StackGAN

2.4. **Image Manipulation.** There are many other applications in image manipulation. e.g. Super resolution (Fig.5): recovering the photo-realistic texture for the low resolution image. Photo inpainting (Fig.6): filling the missing area in a given image.

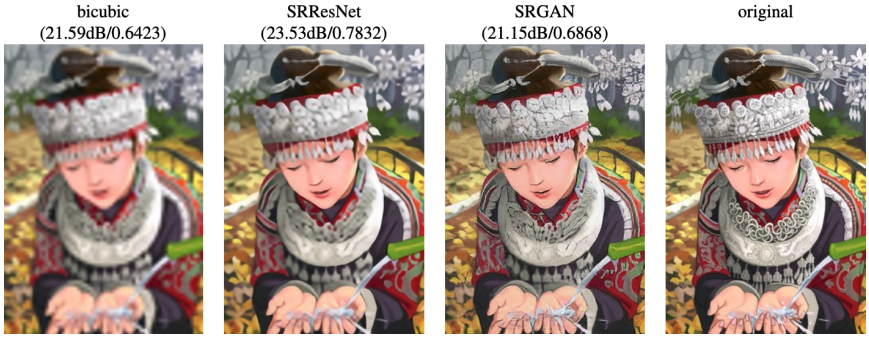

FIGURE 5. Super resolution by SRGAN

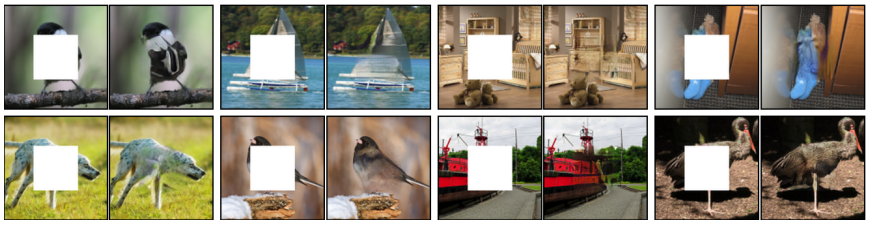

FIGURE 6. Photo inpainting with GAN

Section Summary. The underlying functionality of GAN is transforming data distribution from domain X to domain Y. If we define domain X as a normal (noise) distribution, GAN is going to generate images. If domain X is another image distribution, e.g. different style images, lower resolution images or masked

images, GAN is going to transform input images $X \rightarrow$ target images Y. If domain X is a text data distribution, GAN is going to generate images based on the input text query. After we learn the GAN technique, you are highly encouraged to propose some creative ideas on further applications.

3. Concept

3.1. **What is a GAN.** It is like a zero-sum game in *Game Theory* (Example 1). i.e. One becomes better means the opponent must be worse. In a typical GAN, the "*Criminal*" is named as Generator G while the "*Investigator*" is Discriminator D. In this game, G is trying to generate real-like images to fool D while D is trying to figure all fake images out. These two models compete with each other and eventually reach to a *Nash equilibrium*, where both G and D cannot get better or worse any more.

Example 1. How to be a master of producing fake dollars?
Criminal: Produced 1st version fake dollars and the *investigator* cannot figure them out.
Investigator: Found the new fake dollars and successfully figured all 1st version fake dollars out.
Criminal: Produced 2nd version fake dollars and the *investigator* cannot figure them out.
Investigator: Found the new fake dollars and successfully figured all 2nd version fake dollars out.
...
n-th version fake dollars
...
$\Rightarrow$ *Criminal*: An expert in making fake dollars.
$\Rightarrow$ *Investigator*: An expert in figuring fake dollars.

3.2. **Why GAN works.** If a model only learns the data points rather than the whole distribution, this model can only memorize the dataset. A model can generate new data samples when it learns the whole distribution. GAN applies a zero-sum game to help G to simulate the real distribution. Then, we can sample any random points from this distribution as the generated data.

3.3. **Objective function.**

- **Discriminator**

Discriminator D is going to distinguish all fake images. Intuitively, we can scratch an objective function of D:

$$\max \ J^{(D)} = \mathbb{E}_{x_r \sim X_r}[\text{score}(x_r)] - \mathbb{E}_{x_f \sim X_f}[\text{score}(x_f)] \tag{1}$$

where x_r is sampled from the real distribution X_r, x_f is sampled from the fake distribution X_f, score() denotes an evaluation function which gives high scores on real images and low scores on fake images. Actually, Discriminator D is like a binary classifier to distinguish real or fake.

Let's replace the score() function by D. If the output $D(x)$ is a real number, we hope $D(x_r) \rightarrow \infty$ and $D(x_f) \rightarrow -\infty$ to achieve the maximum $J^{(D)}$. However, the

objective $J^{(D)} \to \infty$ is not applicable in optimization. Thus, we prefer to map the range to probability space, $(-\infty, \infty) \mapsto (0, 1)$. In the objective function, we add a sigmoid transfer function to the output layer.

$$sigmoid(x) \equiv \frac{1}{1 + e^{-x}}$$

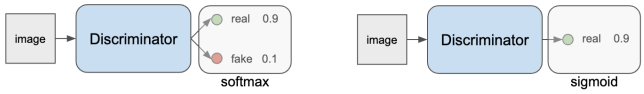

FIGURE 7. Discriminator with softmax/sigmoid output. In binary classification, these two methods have the same effect. For the following discussion, we use sigmoid output in D by default.

With the sigmoid function, the output denotes the probability of real (Fig.7), $D(x) = p(real)$. We hope the output probability of real images close to 1, $D(x_r) \to 1$, while the probability of fake images close to 0, $D(x_f) \to 0$. The later one can also be reformulated as $(1 - D(x_f)) \to 1$. Then, we modify the objective function (Eq.1) scratched at the beginning. Besides, maximizing the objective function (Eq.2.a) is equivalent to minimizing the negative objective function, which is called loss function (Eq.2.b).

$$\max \ J^{(D)} = \mathbb{E}_{x_r \sim X_r}[D(x_r)] + \mathbb{E}_{x_f \sim X_f}[1 - D(x_f)] \tag{2.a}$$

$$\Leftrightarrow \min \ L^{(D)} = -\mathbb{E}_{x_r \sim X_r}[D(x_r)] - \mathbb{E}_{x_f \sim X_f}[1 - D(x_f)] \tag{2.b}$$

The loss function (Eq.2.b) is still based on mean absolute error. As for the loss functions in classification, we often prefer to use cross-entropy (CE) loss rather than mean absolute error (MAE, L1 loss) and mean squared error (MSE, L2 loss). As shown in Fig.8, CE loss can mitigate the gradient vanishing problem when we apply the sigmoid to the output layer.

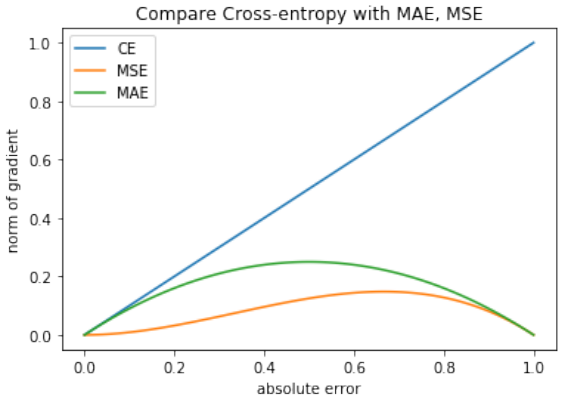

FIGURE 8. MAE/MSE loss: gradient vanishing when the error is large. CE loss: the norm of gradient $\|\nabla\|$ is positively related to the error, a large error leads to a fast learning speed.

Definition 1. The cross-entropy of the distribution q relative to a distribution p over a given set is defined as follows:

$$CE(p, q) = -\mathbb{E}_p[\log q]$$

Let's apply the CE loss into the loss function for D. For real images, p is $[1\ 0]^T$, q is $[D(x_r)\ 1 - D(x_r)]^T$. For fake images, p is $[0\ 1]^T$, q is $[D(x_f)\ 1 - D(x_f)]^T$. Eq.4 is a typical binary cross-entropy (BCE) loss.

$$BCE = -\mathbb{E}_{x_r \sim X_r}\left[\begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}\log D(x_r) \\ \log(1 - D(x_r))\end{bmatrix}\right] - \mathbb{E}_{x_f \sim X_f}\left[\begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}\log D(x_f) \\ \log(1 - D(x_f))\end{bmatrix}\right]$$

With algebraic operations, the loss function for D can be derived as below. For notation consistency, we replace x_f, X_f by x_g, X_g because generated images x_g are exactly identified as fake images x_f by D. θ_D denotes the parameters in D model, which are the weights in neural networks. Our goal is to find the optimal weights to minimize the following BCE loss function.

$$\Rightarrow \min_{\theta_D} L^{(D)} = -\mathbb{E}_{x_r \sim X_r}[\log D(x_r)] - \mathbb{E}_{x_g \sim X_g}[\log(1 - D(x_g))] \tag{3}$$

Remark 1. Any loss consisting of a negative log-likelihood is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by model. For example, mean squared error is the cross-entropy between the empirical distribution and a Gaussian model.

— Page 132, *Deep Learning*, 2016.

The nature of Eq.3: Cross-entropy is a divergence to measure the difference between two distributions. $-\mathbb{E}_{x_r \sim X_r}[\log D(x_r)]$ is minimizing the divergence between real distribution X_r in the training set and the probability distribution defined by D model; and $-\mathbb{E}_{x_f \sim X_f}[\log(1 - D(x_f))]$ is maximizing the divergence between generated distribution X_f and the probability distribution defined by D model.

- **Generator**

Generator G is going to generate real-like images to fool D. Similarly, we can scratch an objective function for G:

$$\max\ J^{(G)} = \mathbb{E}_{x_g \sim X_g}[\text{score}(x_g)] \tag{4}$$

where score() still denotes the D function, x_g is sampled from the generated distribution X_g. Here, we use G to map a normal distribution with the generated distribution, $N(0, 1) \overset{G}{\mapsto} X_g$. In other words, we feed a noise vector $z \sim N(0, 1)$ into the G function and get a generated image, $x_g = G(z)$.

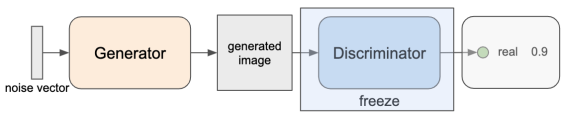

FIGURE 9. The training process for Generator. As we have trained D to classify real/fake images. The objective of G is to make D classify the generated images into real type.

Corresponding to $L^{(D)}$ (Eq.3), we also use CE loss on the generated images. There is a trick that $L^{(G)}$ can simply borrow the second term from $L^{(D)}$. i.e. D is maximizing $-\mathbb{E}_{x_g \sim X_g}[\log(1 - D(x_g))]$ and G can minimizing this term to compete with D. This method is exactly a minimax game and is proposed in the original GAN paper. We refer it (Eq.5.a) as Saturating loss function. However, Saturating loss function cannot provide sufficient gradient for G to learn. For example, in the early learning step, D can easily distinguish generated images and real images. Thus, $D(x_g)$ is close to 0, $\log(1 - D(x_g))$ will saturate to 0. (Fig.10)

(5.a) $$\text{Saturating: } \min_{\theta_G} L^{(G)} = \mathbb{E}_{x_g \sim X_g}[\log(1 - D(x_g))]$$

Later, Ian Goodfellow proposed a more stable and efficient loss function for G. We can calculate the CE of generated images independently, p is $[1\ 0]^T$, q is $[D(x_g)\ 1 - D(x_g)]^T$. We refer it (Eq.5.b) as Non-Saturating loss function. With this loss function, $-\log D(x_g)$ is very large and non-saturating at the early learning period. (Fig.10)

(5.b) $$\text{Non-Saturating: } \min_{\theta_G} L^{(G)} = -\mathbb{E}_{x_g \sim X_g}[\log D(x_g)]$$

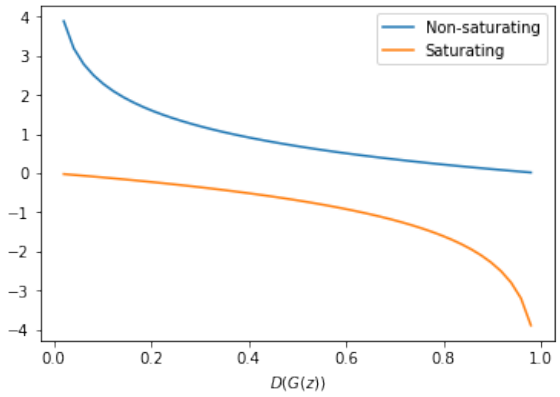

FIGURE 10. Saturating $L^{(G)}$ vs Non-Saturating $L^{(G)}$

In practice, Non-Saturating $L^{(G)}$ is better than Saturating $L^{(G)}$. Intuitively, it would be better to set a large learning rate at the early stage of gradient descent while a small learning rate when closing to the optima.

4. ALGORITHM

If we combine $L^{(G)}$ of Eq.5.a and $L^{(D)}$ of Eq.3 together, the aggregate objective function for GAN, $L^{(GAN)}$, can be derived as:

(6.a) $$\text{minimax: } \min_{\theta_G} \max_{\theta_D} L^{(GAN)} = \mathbb{E}_{x_r \sim X_r}[\log D(x_r)] + \mathbb{E}_{x_g \sim X_g}[\log(1 - D(x_g))]$$

It is the objective function of original GAN with a minimax game. However, we cannot optimize this combined loss function by changing G and D simultaneously. In practice, we can only train these two models alternatively. In each training iteration, we train D and G sequentially. (Algorithm 1)

Training Discriminator (Fig.11). First, we feed noise vectors z into the G from

previous iteration to generate images $x_g = G(z)$. Then, we feed half batch of generated images x_g and half batch of real images x_r into the D from previous iteration. With the loss function (Eq.3), back-propagating gradients to update the parameters in Discriminator θ_D. To note, the parameters in Generator θ_G is frozen. Up to now, we only update D in current iteration.

Training Generator (Fig.12). Next, we freeze θ_D and going to update θ_G. Similarly, we feed noise vectors z into the G from previous iteration to generate images $x_g = G(z)$. Then, we feed a batch of generated images x_g into the updated D. With the loss function (Eq.5.b), back-propagating gradients to update the parameters in Generator θ_G. After that, we complete one training iteration for GAN.

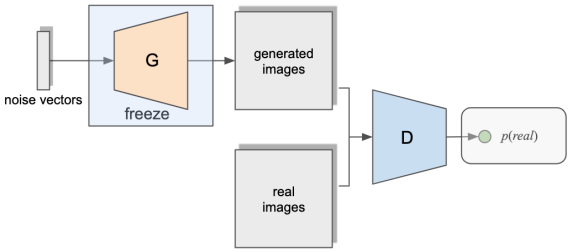

FIGURE 11. Training Discriminator

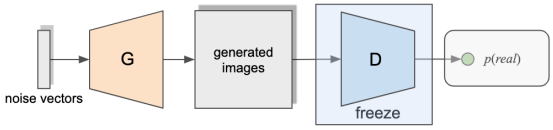

FIGURE 12. Training Generator

Algorithm 1 GAN Training

procedure GAN(θ_G, θ_D) ▷ Parameters of GAN
  Initialization: random θ_G^0 and θ_D^0
  loop($iteration = i$)
    —— Train θ_D (Freeze θ_G) ——
    Input: real images x_r, noise vectors $z \sim N(0, 1)$
    Output of G: generated images $x_g^{i-1} = G^{i-1}(z)$
    Output of D: $D^{i-1}(x_r), D^{i-1}(x_g^{i-1})$ ▷ D: $(i-1)$-th version
    Optimization: $\min -\log D^{i-1}(x_r) - \log(1 - D^{i-1}(x_g^{i-1}))$
    Update: θ_D^i
    —— Train θ_G (Freeze θ_D) ——
    Input: noise vectors $z \sim N(0, 1)$
    Output of G: generated images $x_g^{i-1} = G^{i-1}(z)$
    Output of D: $D^i(x_g^{i-1})$ ▷ D: i-th version
    Optimization: $\min -\log D^i(x_g^{i-1})$
    Update: θ_G^i

5. Architecture

- **Discriminator**

Downsampling networks (Input: images $\mapsto$ Output: probability)

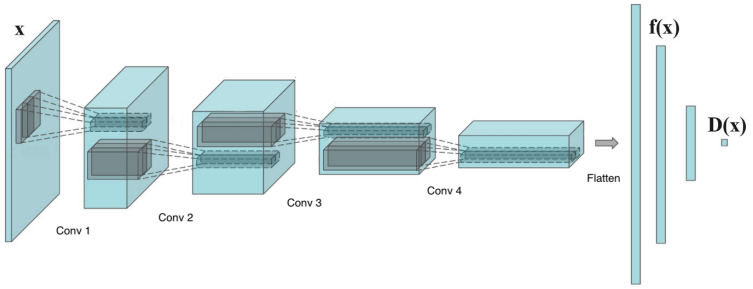

FIGURE 13. An example architecture of the Discriminator in DCGAN

(1) Dense / Fully connected layer: i) Decreasing number of neurons for layers. ii) It cannot be deep and is not good at extracting features.
(2) Maxpooling2D. i) The output is just selecting the maximum input value within the `[height, width]` window of input values. ii) There is no weight and no trainable parameters introduced by this operation.

$$\texttt{Maxpooling2D([2,2])}: \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 8 & 7 & 4 & 3 \\ 6 & 5 & 2 & 1 \end{bmatrix} \mapsto \begin{bmatrix} 6 & 8 \\ 8 & 4 \end{bmatrix}$$

(3) Convolution2D (`stride=2`). i)The output is a linear combination of the input values times a weight for each cell in the `[height, width]` kernel/filter. ii) These weights become trainable parameters in your model.

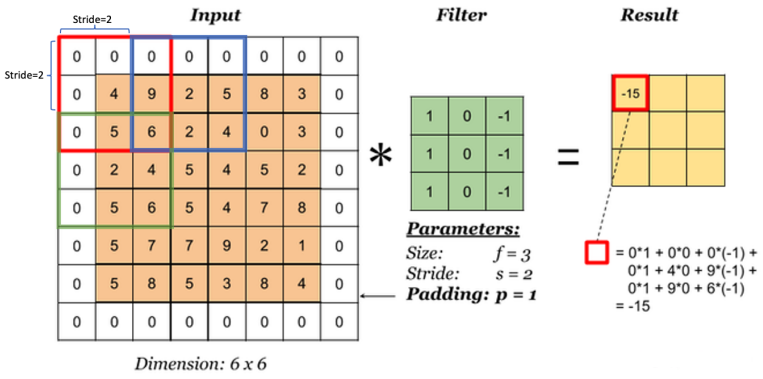

FIGURE 14. Convolution operation. (Modified from `indoml.com`)

- **Generator**

Upsampling networks. (Input: noise vectors $\mapsto$ Output: images)

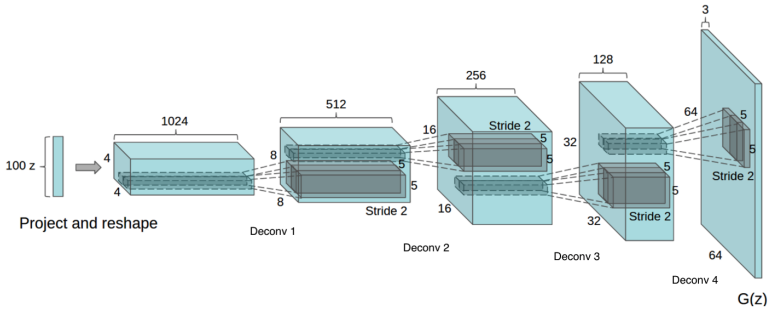

FIGURE 15. An example architecture of the Generator in DCGAN

(1) Dense / Fully connected layer: i) Increasing number of neurons for layers. ii) It cannot be deep and is not good at generating features.
(2) Upsampling2D. i) The `[height, width]` window of output values is just repeating the corresponding input value. ii) There is no weight and no trainable parameters introduced by this operation.

$$\texttt{Upsampling2D([2,2])}: \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix}$$

(3) ConvTranspose2D / Deconvolution2D (`stride=2`). i) It is also named as fractional-strided convolution. Stride here is the reciprocal of the moving step. e.g. `stride=2` $\Rightarrow$ moving step $= \frac{1}{2}$. How to move $\frac{1}{2}$ step? Inserting zero columns and rows to the original input. ii) Similar to convolution operation, these weights become trainable parameters in your model.

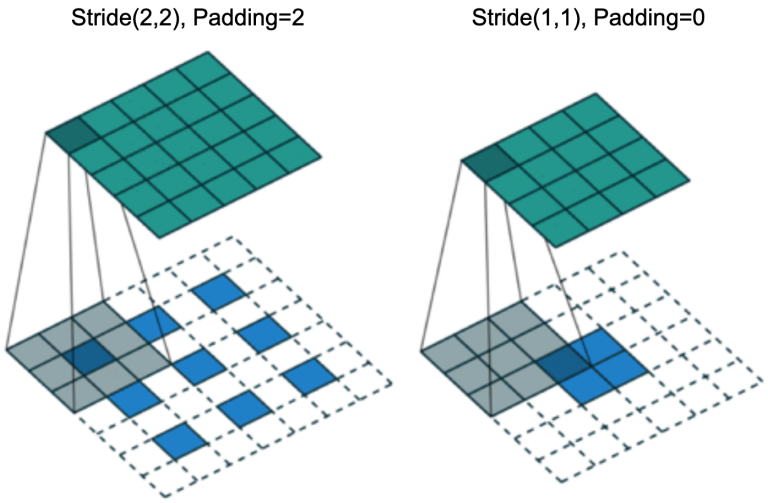

FIGURE 16. Transposed convolution operation. (View more visualizations at `https://github.com/vdumoulin/conv_arithmetic` by Vincent Dumoulin, Francesco Visin.)

6. Tuning Tips

Hard to train. GAN has two networks and many hyper-parameters. Thus, GAN is very hard to train in practice. This section will share some useful tuning tips. With these tips, there is a good chance (not necessary) to improve your GAN performance.

Architecture. In the previous section (Sec.5), we have many choices to design our GAN architecture. However, fully connected networks is not good at generating sharp images (Fig.). As we know, convolutional neural networks (CNN) is good at extracting features with less parameters and deep architectures. In DCGAN paper [2], they first proposed CNN based GAN and provide some architecture guidelines for stable DCGAN. The guidelines are partially modified as following:

- Replace any pooling layers with strided convolutions (Discriminator) and fractional-strided convolutions (Generator).
  Remark: Maxpooling2D and Upsampling2D are not suggested because GAN training is highly sensitive and requires continuous differentiable functions of D and G. Maxpooling2D and Upsampling2D have no parameters in training, but the parameters in Conv2D and ConvTransposed2D are trainable.
- Use BatchNorm (BN) in G. [2] suggests to also use BN in D. However, we suggest not to use Normalization layer in D, otherwise, to use LayerNorm (LN) or SpectrumNorm (SN) in D (beyond introduction level).
  Remark: Normalization layer is an efficient way to avoid gradient vanishing problem and help to create a deeper network.
- Remove fully connected hidden layers for deeper architectures.
  Remark: Fully connected layers can only be occurred in the input part of G and output part of D.

Activation function. Except the saturating activation functions (sigmoid, tanh), we suggest to use non-saturating activation functions (ReLU, LeakyReLU) in the hidden layers. LeakyReLU is an advanced ReLU function which allows the pass of negative cases with a small value (Fig.17). In practice, LeakyReLU has a significant effect on stablizing GAN training (Fig.18).

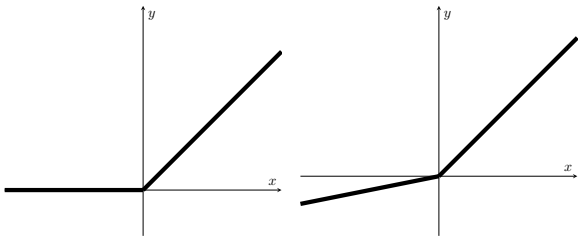

Figure 17. ReLU vs LeakyReLU

- Use LeakyReLU activation in G for all layers except for the output, which uses tanh. Note: in GAN training, it is suggested to rescale the color range of images to $[-1, 1]$.
- Use LeakyReLU activation in D for all layers except for the output, which uses sigmoid/softmax.

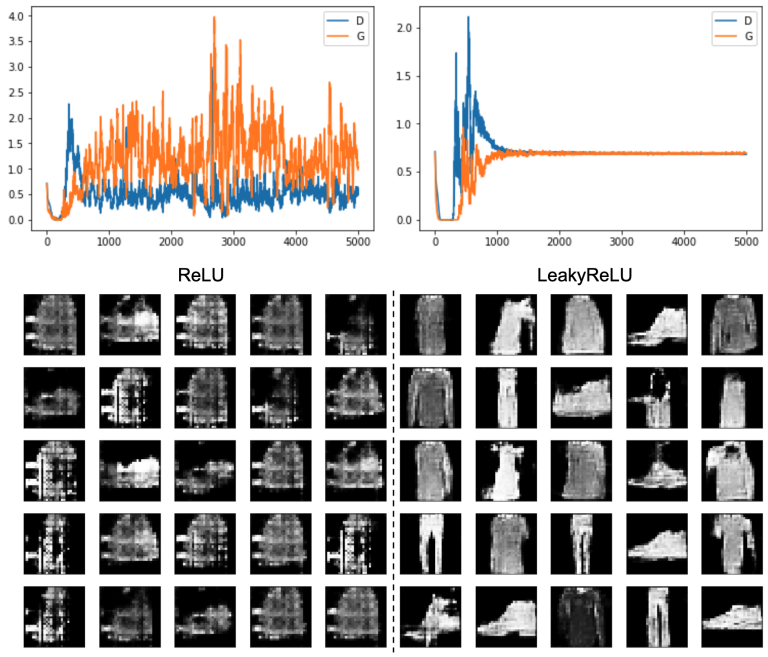

FIGURE 18. Compare training stability and performance on *MNIST-Fashion* based on DCGAN. With LeakyReLU activation, the game between G and D will converge to a *Nash equilibrium*. With ReLU, GAN cannot converge to a *Nash equilibrium* and the generated images suffer mode collapse (defined in Sec.7) problem.

7. Potential Problems

Mode collapse. (Fig.19) Generator only produces low-diversity outputs. A complete mode collapse, which is not common, means the Generator just makes a trick to create only one type of image to fool the Discriminator. A partial mode collapse, which always happens, is a hard problem in GAN to solve.

Imbalanced capacity between *G* and *D*. In practice of constructing a GAN model, the architecture of Discriminator is not suggested to be very powerful. Due to the limitation of loss function, a powerful Discriminator cannot give meaningful gradients when training its Generator. However, it is contradictory that a powerful Generator must require a powerful Discriminator. In other words, Discriminator is like a teacher while Generator is like a student. A primary student fails to improve himself when he sits in the undergraduate lectures. A Ph.D student can hardly improve himself when he sits in the undergraduate lectures. We hope the Discriminator can lead the Generator to grow up at all stages, including early round and final round. It is also one of the reasons why training a GAN is so hard.

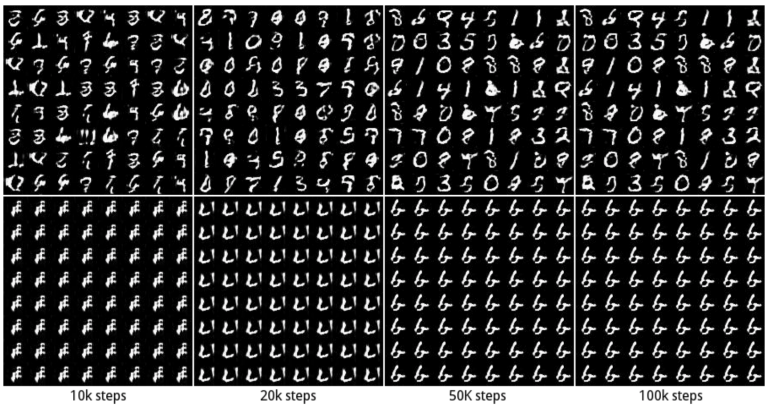

FIGURE 19. First row: GAN can generate all modes of images. Second row: GAN can only generate a single mode of images, which suffers mode collapse problem.
(From `https://arxiv.org/pdf/1611.02163.pdf`)

Non-convergence for training. In the minimax problem, some loss functions may not lead to a convergence by nature. For example, two minimax games with

$$L^C = \min_x \max_y x^2 y^{-1}$$

$$L^{NC} = \min_x \max_y xy$$

With gradient descent (Fig.20), the training process of L^{NC} will converge to 0, but the training process of L^{NC} is oscillated and cannot converge to an equilibrium.

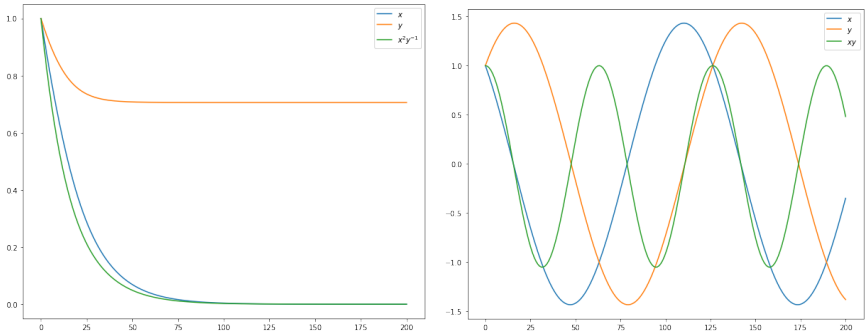

FIGURE 20. Convergence game vs Non-convergence game

8. EVALUATION

GAN is a very new topic, it is still an open problem to find a perfect evaluation of GAN or generated images. Generally, we measure the generated images in two dimensions: quality of images and diversity of images.

Apart from comparing the generated images with real images by our eyes, there are two common mathematical metrics to evaluate the quality of the generated

images: Inception Score (IS) and Fréchet Inception Distance (FID). Both of these two measurements are based on the Inception V3 network, which is pretrained on ImageNet dataset. IS is derived from the classification output while FID is derived from the feature layer.

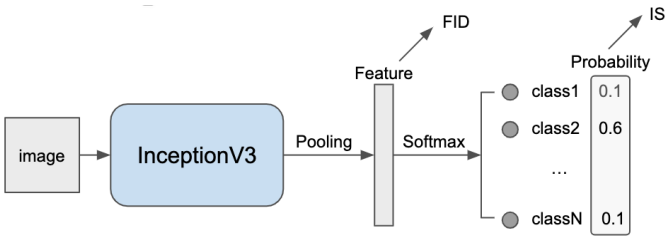

FIGURE 21. FID vs IS

Inception Score. IS measures the KL divergence (similar to cross-entropy, see Def.2) between the generated sample distribution and the ImageNet distribution, whereas FID calculates the feature-level distance between the generated sample distribution and the real sample distribution.
Inception Score is defined as:

$$IS(X_g) = \exp\left(\mathop{\mathbb{E}}_{x_g \sim X_g}\left[D_{\mathrm{KL}}\left(\ p\left(y|x_g\right) \parallel p\left(y\right)\ \right)\right]\right)$$

where $x_g \sim X_g$ indicates that x_g is an image sampled from X_g, $D_{\mathrm{KL}}(P \parallel Q)$ is the KL divergence between the distributions P and Q, $p(y|x_g)$ is the conditional class distribution, and $p(y) = \mathbb{E}_{x_g \sim X_g}\left[p(y|x_g)\right]$ is the marginal class distribution.
Remark: InceptionV3 network is trained on ImageNet and used to predict the class of given images. If the softmax output is sharp (p(one class) $\to 1, p$(other classes) $\to 0$), the IS will be small and the quality of this image is high. Besides, the diversity of softmax output indicates the diversity of images.
D_{KL} calculation will output a small number. The exp in the expression is there to make the values easier to compare, so it will be ignored if we use $\ln(IS)$ without loss of generality.

Definition 2. In mathematical statistics, the Kullback–Leibler (KL) divergence (also called relative entropy) is a measure of how one probability distribution is different from a second, reference probability distribution.

$$\begin{aligned} D_{\mathrm{KL}}(P \parallel Q) &= \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \\ &= \underbrace{\left(-\sum_{x \in \mathcal{X}} P(x) \log Q(x)\right)}_{\text{Cross-entropy}(P,Q)} - \underbrace{\left(-\sum_{x \in \mathcal{X}} P(x) \log P(x)\right)}_{\text{Cross-entropy}(P,P)} \end{aligned}$$

This equation is a discrete case of KL divergence, where P and Q are two probability distributions, $\mathcal{X}$ is the probability space. For the continuous case, $P(x), Q(x)$ are probability density functions.

Fréchet Inception Distance. IS totally depends on the InceptionV3 knowledge on ImageNet. If we input a high quality image which cannot be classified into ImageNet classes to InceptionV3 network, then the softmax output will not be a sharp value. In other words, IS can only evaluate the generated images which should belong to ImageNet dataset. In comparison, FID is a more general evaluation and applicable to new datasets. Besides, feature-level information far surpasses classification-level information.
Fréchet Distance is defined as:

$$FID = \|\mu_r - \mu_g\|^2 + Tr\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r\Sigma_g\right)^{1/2}\right)$$

where μ_r is the mean of the real features, μ_g is the mean of the generated features, Σ_r is the covariance matrix of the real features, Σ_g is the covariance matrix of the generated features.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, *Generative adversarial nets*, Neural Information Processing Systems (2014), 2672–2680.
[2] Alec Radford, Luke Metz, and Soumith Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, arXiv preprint arXiv:1511.06434 (2015).
[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, Vol. 1, MIT press Cambridge, 2016.