Deep Learning Crash Course

Hui Xue

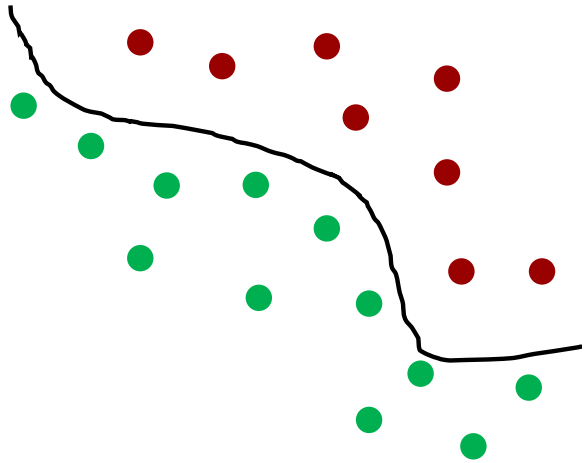
Fall 2021

www.deeplearningcrashcourse.org

Outline

- Generative and discriminative model
- Generative model, introduction
- Key ideas motivating Generative Adversarial Network (GAN)
- G/D game
- Optimal solution of G/D gaming
- Conditional GAN and Cycle GAN
- Further extension of GAN framework

Generative and discriminative model



Discriminative: learn the decision boundary to make prediction, often with a functional form

$$p(y = red|x = \bullet)$$

Example: logistic regression, CNN …

$p(y = red|x = \bullet)$ $p(y = green|x = \bullet)$

$p(y = red|x = \bullet)$ $p(y = green|x = \bullet)$

$p(y = red|x = \bullet)$ $p(y = green|x = \bullet)$

$p(y = red|x = \bullet)$ $p(y = green|x = \bullet)$

Generative: learn the joint distribution between data and label to make prediction

$$p(y = red, x = \bullet)$$

Example: autoregression, GAN, variational autoencoder …

Linked by the Bayes Rule

$$p(y = red|x = \bullet) = \frac{p(y = red, x = \bullet)}{p(x = \bullet)} = \frac{p(x = \bullet\ |y = red)p(y = red)}{p(x = \bullet)}$$

$p(y = red|x = \bullet) = 0.9$

$p(y = green|x = \bullet) = 0.1$

Discriminative model takes the short-cut.

Very suitable for supervised learning

$p(x = \bullet)$ is a constant, given the dataset

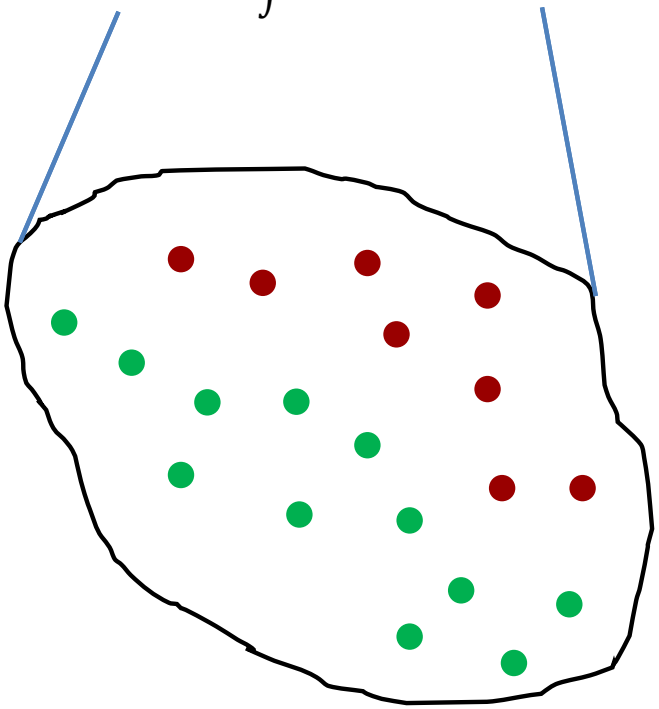
$p(y = red, x = \bullet) > p(y = green, x = \bullet)$

This dot is red …

We can use joint probability to make prediction too

Generative model can do more

$$p(x) = \int p(x|y)p(y)dy$$



Data distribution p(x)

p(y) is the class prior, often can be estimated from counting class frequency

- **Sample**: get a new sample if we learned the data distribution
  $x \sim p(x)$
- **Density estimation or prediction**: compute $p(x = x^*)$ or $p(y|x = x^*)$
- **Data representation**: learn effective representation from dataset
- **Make prediction for incomplete data**:

$$p(y = 1|x_1 = red, x_2 = ?) = p(y = 1|x_1 = red, x_2 = red)p(x_2 = red) + p(y = 1|x_1 = red, x_2 = green)p(x_2 = green)$$

make prediction from partial data is hard for discriminative model

Generative model

Given a dataset $\boldsymbol{X} = \{x_0 \quad x_1 \quad \cdots \quad x_{N-1} \quad x_N\}$, assume i.i.d.

Data likelihood is $p(\boldsymbol{X}) = \prod_{k=0}^{N-1} p(x_k)$

Log data likelihood is $\log(p(\boldsymbol{X})) = \sum_{k=0}^{N-1} log(p(x_k))$

Some generative model relies on maximizing data likelihoods:

Autoregressive model : $p_{\boldsymbol{\theta}}(\boldsymbol{X}) = \prod_{k=0}^{N-1} p(x_k|x_{<k})$, conditional dependence

Variational autoencoders : $p_{\boldsymbol{\theta}}(\boldsymbol{X}) = \int p(\boldsymbol{x}|z)p(z)dz$, latent variable

Flow model : $p_{\boldsymbol{\theta}}(\boldsymbol{X}) = p(\mathrm{z} = f_{\theta}^{-1}(x)) \left|\det(\frac{\partial f_{\theta}^{-1}(x)}{\partial x})\right|$, functional transformation between latent and data variable

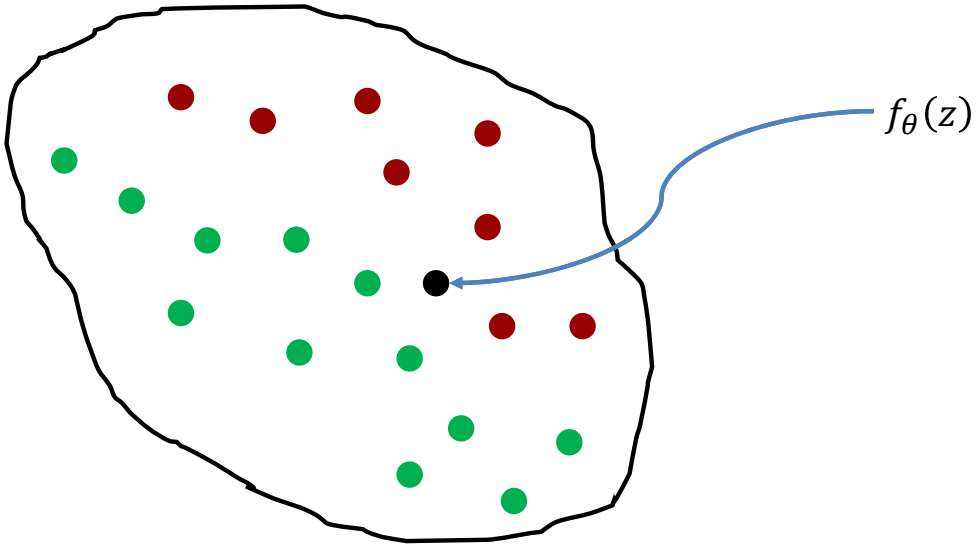
Differences are how to represent data likelihood and how to conduct learning

Implicit generative model

Given a dataset $\boldsymbol{X} = \{x_0 \quad x_1 \quad \cdots \quad x_{N-1} \quad x_N\}$, assume i.i.d.

Do not explicitly represent data likelihood $p(\boldsymbol{X}) = \prod_{k=0}^{N-1} p(x_k)$

But allow sampling from data distribution → create new samples



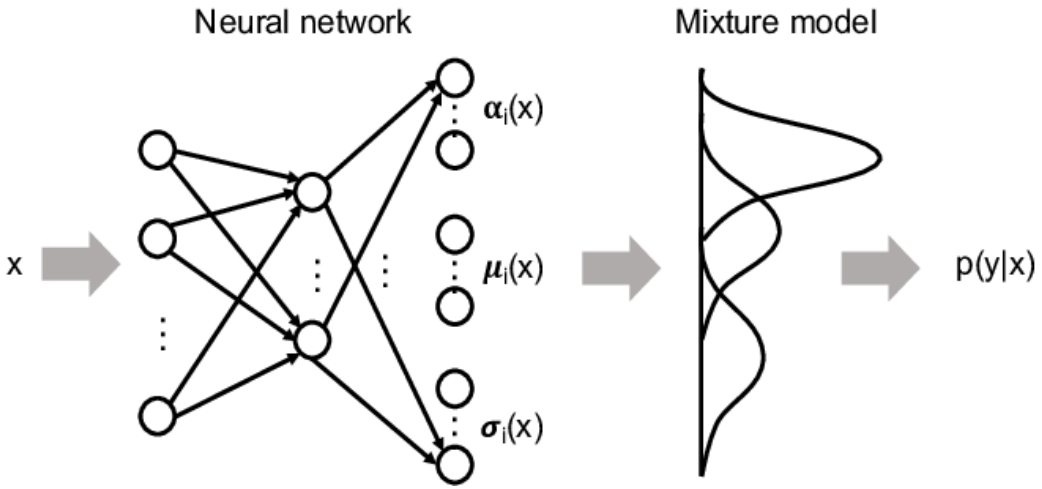
Data distribution p(x)

- Model can turn a latent vector to a data sample by learning the underneath data distribution
- Do not allow density estimation
- Can provide good data representation

Early attempt: density network

Given a latent vector $\boldsymbol{z}$, density network is a model to output parametric data distribution, e.g.

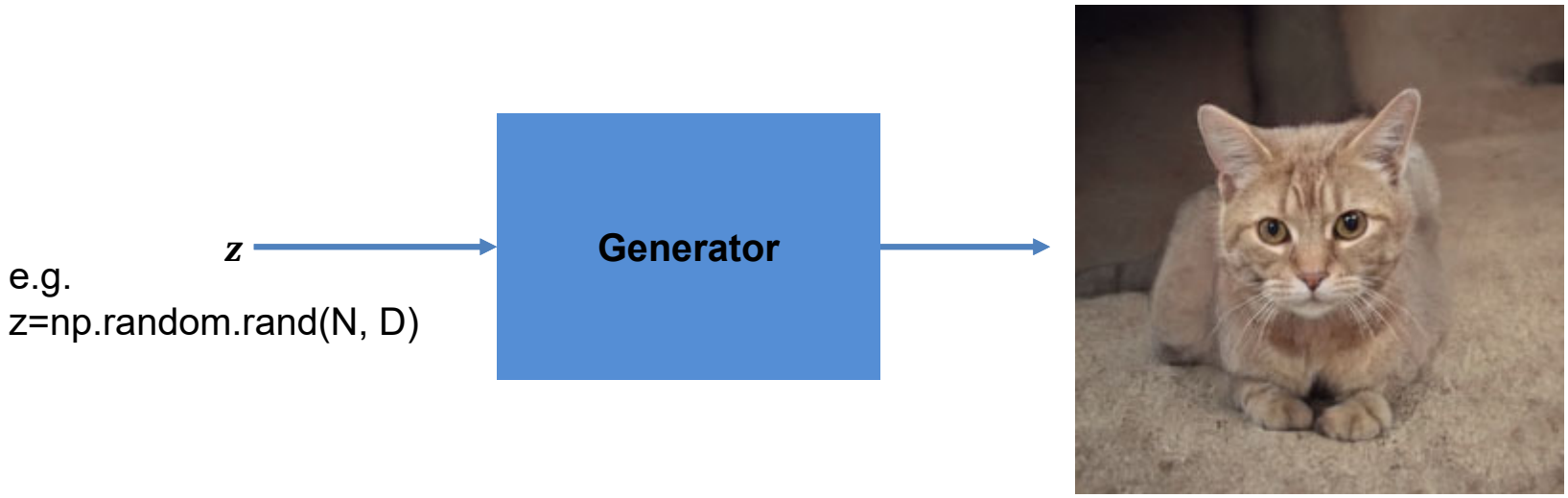
mean and variance of multi-modality Gaussian distribution



- Allow effective sampling
- Model is simple and easy to train
- Limited representation power

Modern generative model

- Use deep neural network to learn data distribution
- Directly generate sample, implicit modeling

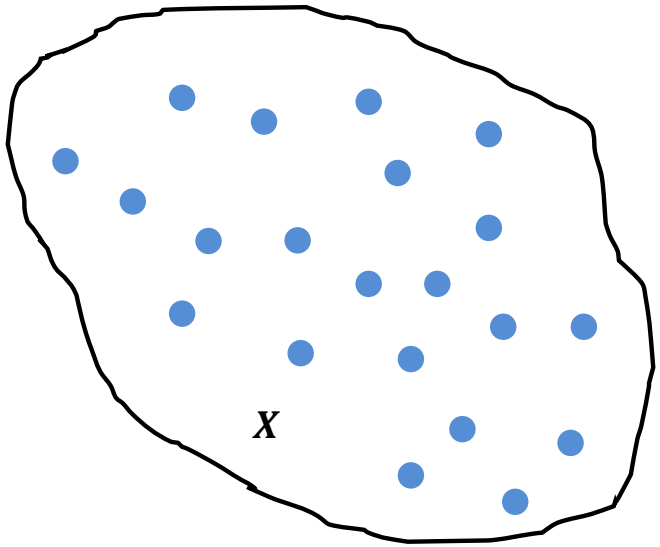


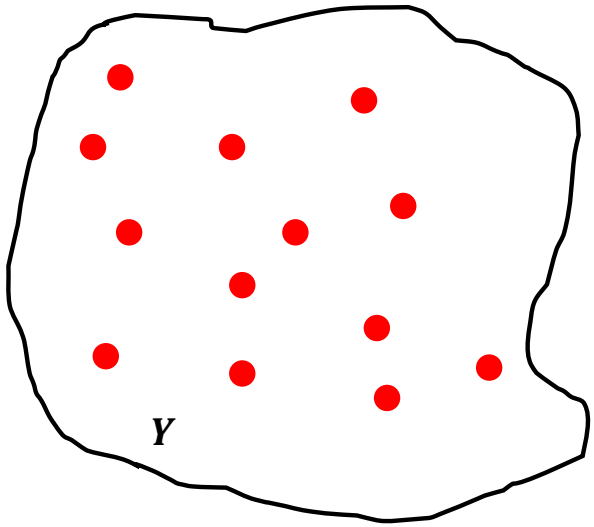
Picture is from Nvidia, style GAN
https://github.com/NVlabs/stylegan2

How to evaluate samples?

Two sample tests

Given two datasets $\boldsymbol{X}$ and $\boldsymbol{Y}$, assume the underlying data distribution is p_x and p_y

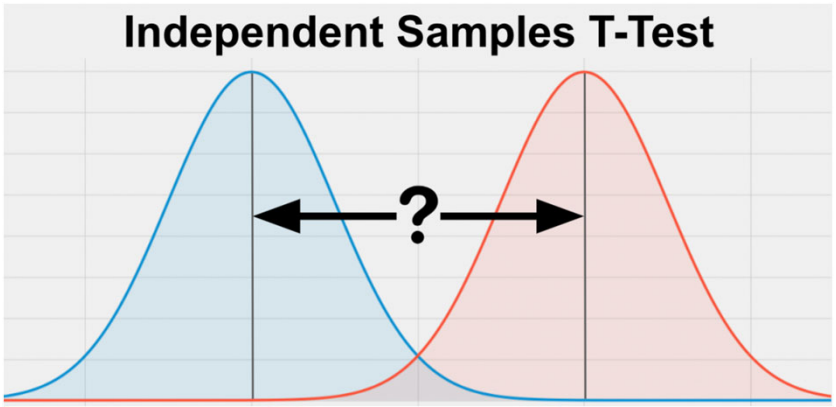




Are $p_x = p_y$?

Two-sample tests

For low dimensional data, it is often performed by designing and comparing some statistics, e.g. t-test



Consider two sets of samples, are they from the same normal distribution?

- Easy to perform
- Very limited representation power
- Difficult to design statistics for high dimensional data, e.g. images, musical waveforms …

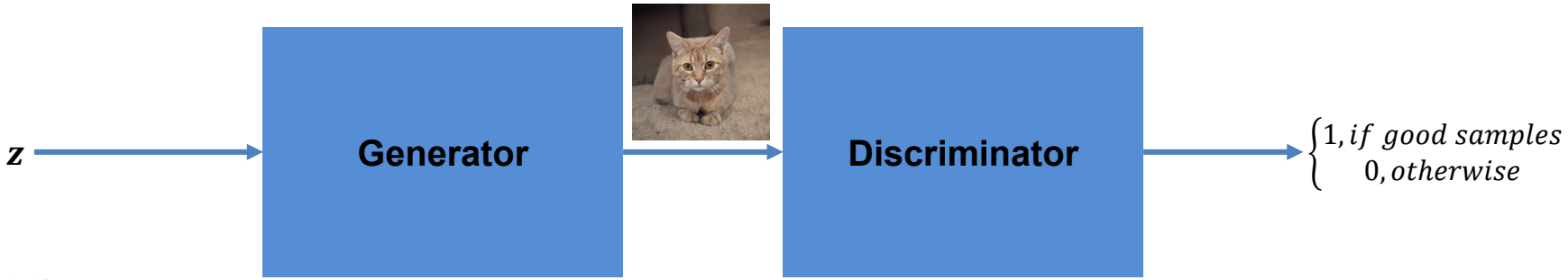
Use DNN for two-sample tests

- Avoid the difficulty to design statistics
- Use DNN representation power for this classification task

https://www.thehappycatsite.com/wp-content/uploads/2019/02/Cartoon-Cat-Names-long.jpg

A linear design

A linear design

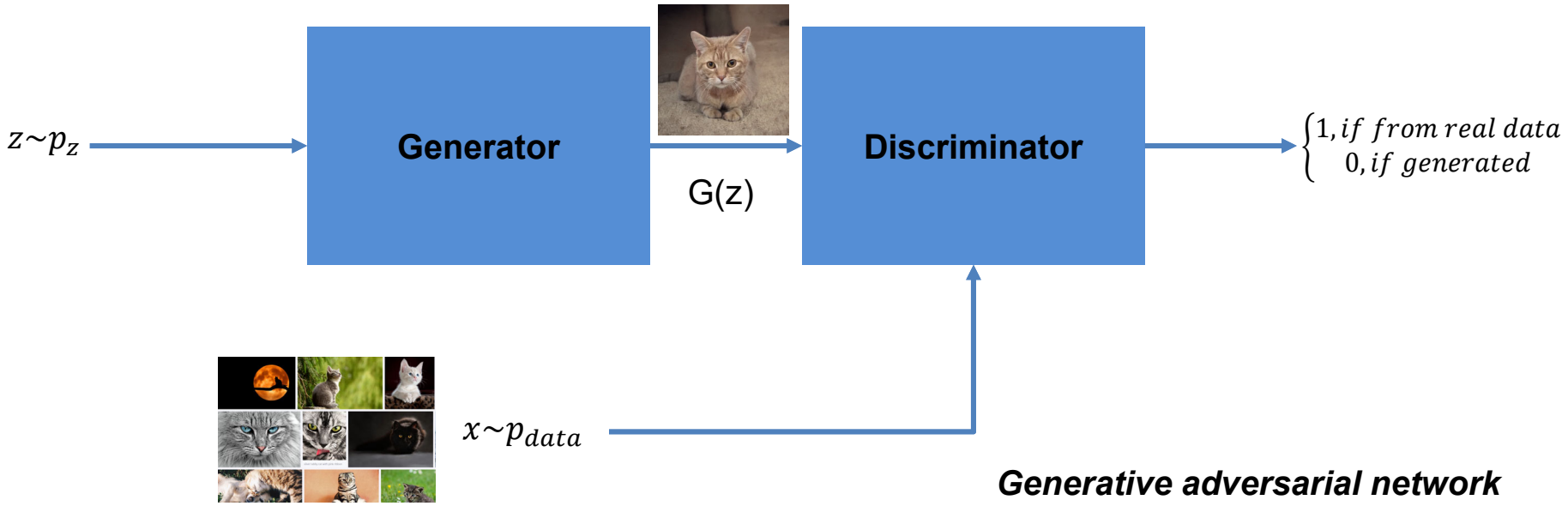


e.g.
z=np.random.rand(N, D)

- Two key ideas
  - Use DNN for generator and discriminator
  - Directly generate samples

Still hard to evaluate sample quality

G/D gaming design

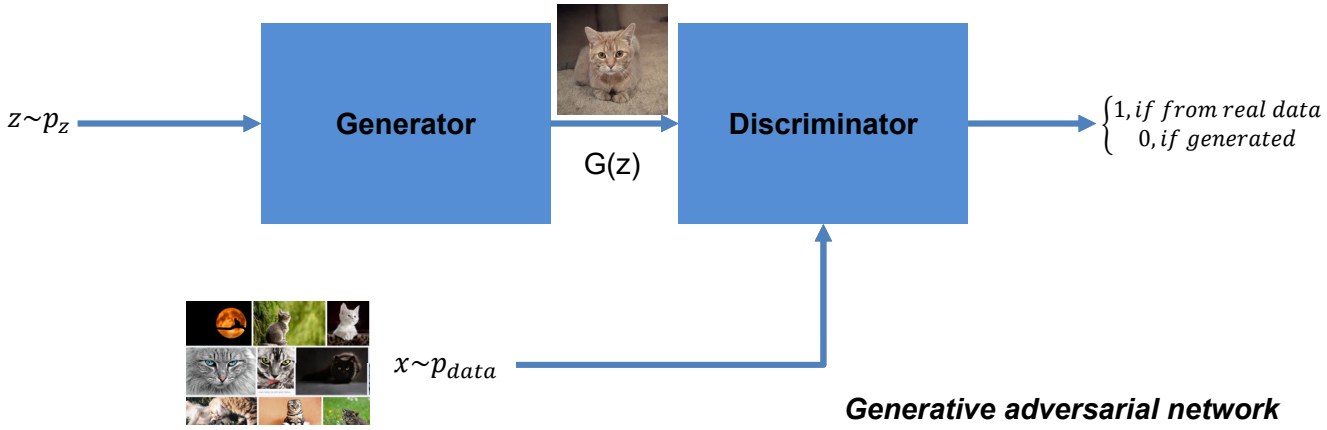


Generative adversarial network

- Three key ideas
  - Use DNN for generator and discriminator
  - Directly generate samples
  - G/D gaming design, give discriminator real samples to train

Ian J. Goodfellow et al. Generative Adversarial Nets. NIPS 2014.

G/D gaming design



Generative adversarial network

- Generator model is used to generate new samples by learning data distribution
- Discriminator is used to classify whether inputs are from real data distribution p_{data} or generator $p_{G(z)}$
- Generator tries to generate “real” sample to fool discriminator; Discriminator tries to classify real sample from generated ones to the highest accuracy

Ian J. Goodfellow et al. Generative Adversarial Nets. NIPS 2014.

GAN loss: Discriminator

$$\max_D V(G,D) = E_{x\sim p_{data}}[logD(x)] + E_{x\sim p_G}[\log(1-D(x)]$$

When training D, G is fixed

- Recall the binary cross-entropy loss : $\log(y) + \log(1-y)$, y is the model probability

- For a batch of samples from **Real** data distribution: $X\sim p_{data}$

Since $D(x) \in [0.0, 1.0]$, this term will be maximized when D(x)=1.0

For **real** samples from data distribution, discriminator assigns probability 1.0

- For a batch of samples generated from generator: $X\sim p_G$ and $x = G(z)$

For **fake** samples generated by G, discriminator assigns probability 0.0

GAN loss: Discriminator

$$\max_D V(G,D) = E_{x\sim p_{data}}[logD(x)] + E_{x\sim p_G}[\log(1-D(x)]$$

When training D, G is fixed

$$E_{x\sim p_{data}}[logD(x)] + E_{x\sim p_G}[\log(1-D(x)] = \int p_{data}(x)logD(x)dx + \int p_G(x)\log(1-D(x))dx =$$

$$\int \left[p_{data}(x)logD(x) + p_G(x)\log\left(1-D(x)\right)\right]dx$$

To find the discriminator to maximize cost:

Give a specific x, take the derivative to D:

$$\frac{d\{p_{data}(x)logD(x) + p_G(x)\log\left(1-D(x)\right)\}}{dD} = \frac{p_{data}(x)}{D(x)} - \frac{p_G(x)}{1-D(x)} = 0$$

Let the derivative be 0, we get the optimal discriminator: $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$

GAN loss: Generator

$$\min_G V(G,D) = E_{x\sim p_{data}}[logD(x)] + E_{x\sim p_G}[\log(1-D(x)] = E_{x\sim p_{data}}[logD(x)] + E_{z\sim p_z}[\log(1-D(G(z))]$$

When training G, D is fixed

- Only the 2nd term has G

- If the generator is so good that D is 100% fooled, with D(G(z))=1.0, loss is minimized ($-\infty$)

$$\min_G \max_D V(G,D) = E_{x\sim p_{data}}[logD(x)] + E_{x\sim p_G}[\log(1-D(x)] = E_{x\sim p_{data}}[logD(x)] + E_{z\sim p_z}[\log(1-D(G(z))]$$

GAN loss is often written as this min-max problem: the goal is to jointly train two networks (D and G), maximize loss to train D and minimize loss to train G

GAN loss: at the optimal point

$$V(G,D) = E_{x\sim p_{data}}[logD(x)] + E_{x\sim p_G}[\log(1-D(x)]$$

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$$

$$\begin{aligned}
V(G,D^*) &= E_{x\sim p_{data}}[log\frac{p_{data}(x)}{p_{data}(x)+p_G(x)}] + E_{x\sim p_G}[\log(1-\frac{p_{data}(x)}{p_{data}(x)+p_G(x)})] \\
&= E_{x\sim p_{data}}[log\frac{p_{data}(x)}{p_{data}(x)+p_G(x)}] + E_{x\sim p_G}[\log\frac{p_G(x)}{p_{data}(x)+p_G(x)}] \\
&= E_{x\sim p_{data}}[log\frac{p_{data}(x)}{\frac{p_{data}(x)+p_G(x)}{2}}] + E_{x\sim p_G}[\log\frac{p_G(x)}{\frac{p_{data}(x)+p_G(x)}{2}}] - log4 \\
&= D_{KL}\left(p_{data}, \frac{p_{data}(x)+p_G(x)}{2}\right) + D_{KL}\left(p_G, \frac{p_{data}(x)+p_G(x)}{2}\right) - log4
\end{aligned}$$

$D_{KL}\left(p_{data}, \frac{p_{data}(x)+p_G(x)}{2}\right)$ is the KL divergence

GAN loss: at the optimal point

$$V(G, D^*) = D_{KL}\left(p_{data}, \frac{p_{data}(x) + p_G(x)}{2}\right) + D_{KL}\left(p_G, \frac{p_{data}(x) + p_G(x)}{2}\right) - log4$$

$D_{KL}\left(p_{data}, \frac{p_{data}(x)+p_G(x)}{2}\right)$ and $D_{KL}\left(p_G, \frac{p_{data}(x)+p_G(x)}{2}\right)$ are the KL divergence

$D_{KL}(p, q) = 0$ if p=q

Therefore, when we achieve the best discriminator and the best generator, which minimizes the loss is

$$p_G = p_{data}$$

$$V(G, D^*) = -log4$$

Theorem: If the generator can be updated in the **functional space** and discriminator is **optimized in each step**, this optimal loss can be achieved.

Ian J. Goodfellow et al. Generative Adversarial Nets. NIPS 2014.

GAN loss: Jenson-Shannon Divergence

$$V(G, D^*) = D_{KL}\left(p_{data}, \frac{p_{data}(x) + p_G(x)}{2}\right) + D_{KL}\left(p_G, \frac{p_{data}(x) + p_G(x)}{2}\right) - log4$$

$D_{KL}(p, q) = E_{x\sim p}[log\frac{p(x)}{q(x)}]$ KL Divergence $D_{KL}(p, q) \geq 0$, $D_{KL}(p, q) == 0\ if\ p = q$, $D_{KL}(p, q) \neq D_{KL}(q, p)$

$$D_{JSD}(p, q) = \frac{1}{2}[D_{KL}\left(p, \frac{p+q}{2}\right) + D_{KL}\left(q, \frac{p+q}{2}\right)]$$ Jenson-Shannon Divergence

$D_{JSD}(p, q) \geq 0$, $D_{JSD}(p, q) == 0\ if\ p = q$, $D_{JSD}(p, q) == D_{JSD}(q, p)$

$$V(G, D^*) = 2D_{JSD}(p_{data}, p_G) - log4$$

When the generator is optimized,

$$V(G^*, D^*) = -log4$$

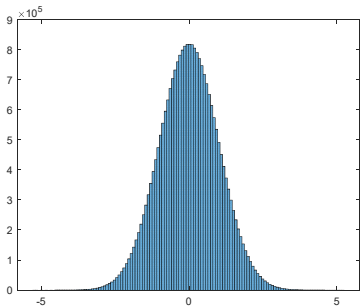
GAN : Sampling for Training

- For a batch of samples from **Real** data distribution: $X \sim p_{data}$

$$E_{x \sim p_{data}}[logD(x)] \approx \frac{1}{B}\sum_{i=0}^{B-1} logD(x^{(i)})$$

- For a batch of samples generated from generator: $Z \sim p_z$ and $G(z)$

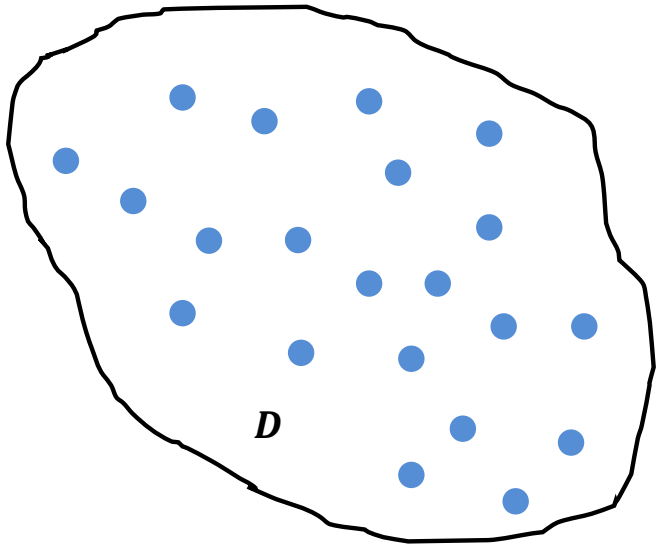
$$E_{z \sim p_z}[\log(1 - D(G(z))] \approx \frac{1}{B}\sum_{i=0}^{B-1} logD(G(z^{(i)}))$$



p_z

Z = np.random.rand(B, 256)

Given a prior distribution,
we sample latent vectors Z



Given a training set, we sample a minibatch of samples $x \sim p_{data}$

GAN : Training

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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Ian J. Goodfellow et al. Generative Adversarial Nets. NIPS 2014.

For every epoch

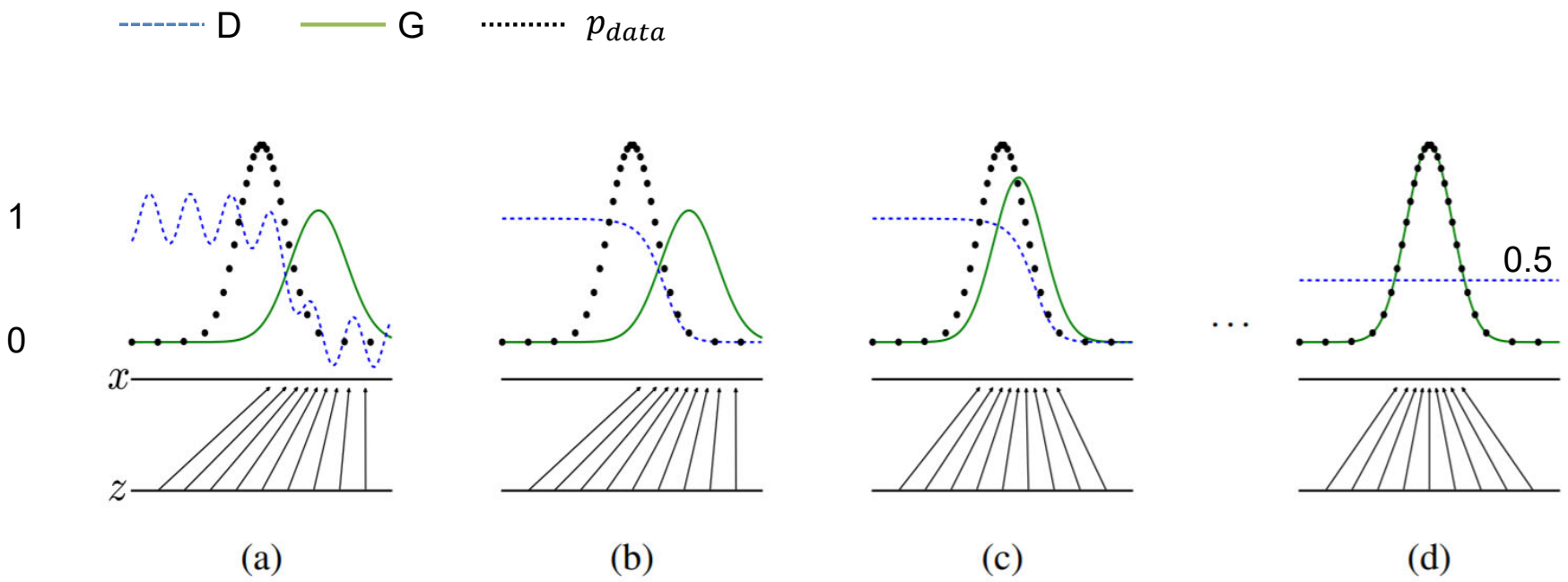
  for k=0:K-1
    Sample z
    Sample x

    Train D

  Sample z
  Train G

GAN : Training

In every iteration, G learns the data distribution to challenge D



Ian J. Goodfellow et al. Generative Adversarial Nets. NIPS 2014.

GAN : Example architectures

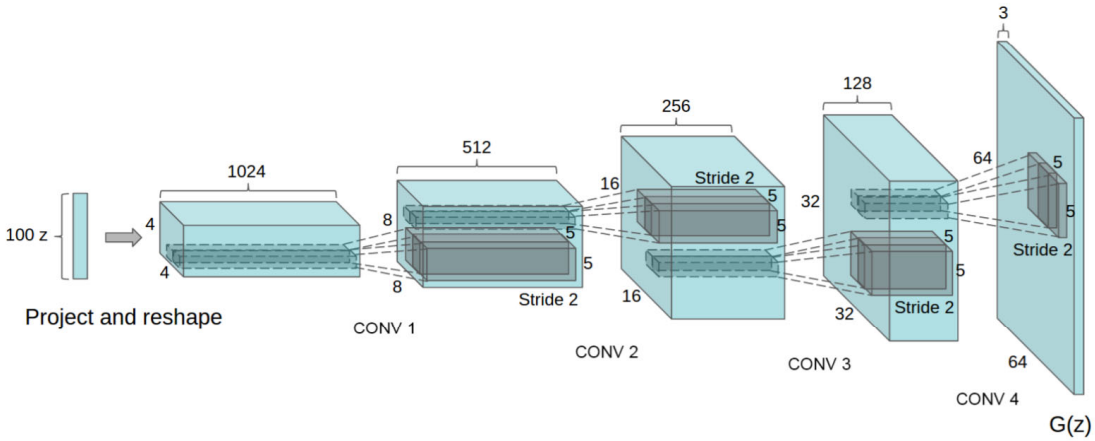
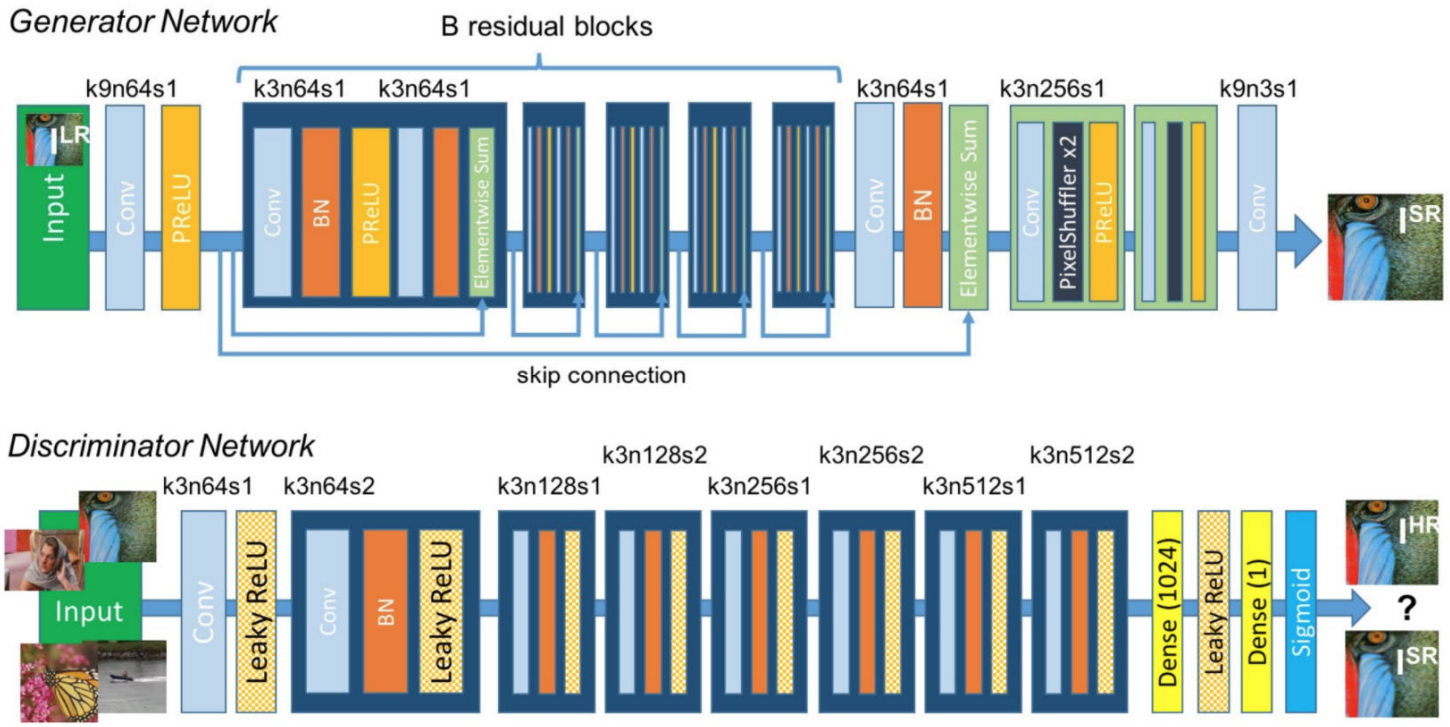


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)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

- Input vector reshape
- Stride convolution to increase spatial resolution
- Gradually reduce CONV channels with higher spatial resolution
- BatchNorm, ReLU or LeakyReLU

Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. 2016. https://arxiv.org/pdf/1511.06434.pdf

GAN : Example architecture for Super-resolution

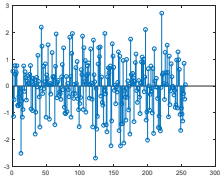


- Skip connections
- Residual blocks
- 3x3 conv kernel

- VGG type architecture for discriminator

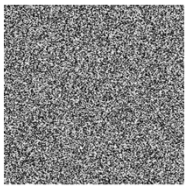
Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. 2017. https://arxiv.org/pdf/1609.04802.pdf

GAN : Training



random vector z

Generator

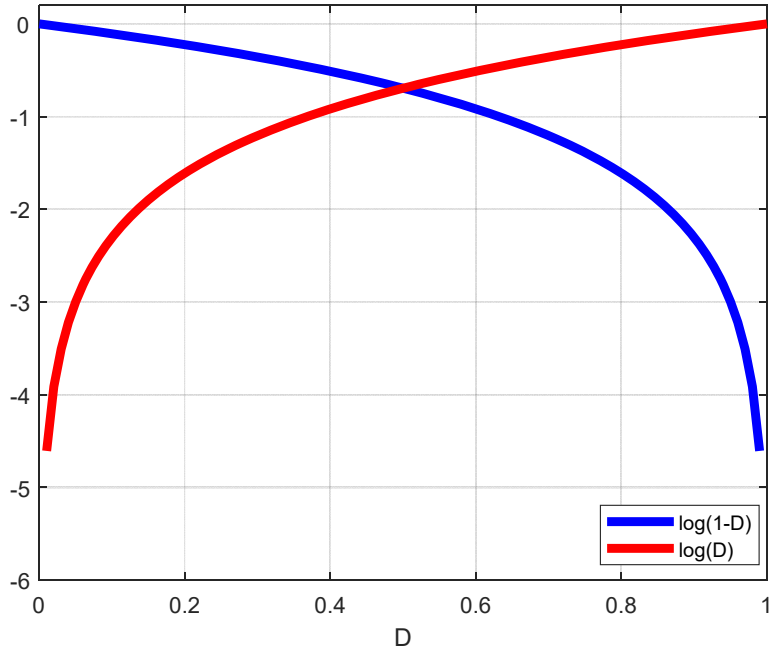


At the beginning, generator was not trained and was randomly initialized

Most likely, the images generated will not be realistic

GAN : Training

$E_{z\sim p_z}[\log(1 - D(G(z))]$ D will have easy time to differentiate that G(z) is faked (D(G(z))=0) at the beginning



$E_{z\sim p_z}[\log(1 - D(G(z))]$ reaches minima when D=1

$E_{z\sim p_z}[\log(D(G(z))]$ reaches maxima when D=1

$$\max_G V(G, D) = E_{x\sim p_G}[\log(D(x)]$$

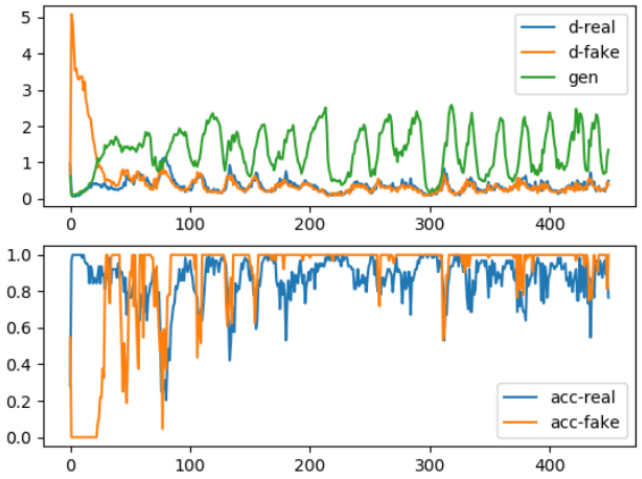
Non-saturated Generator loss

GAN : Training

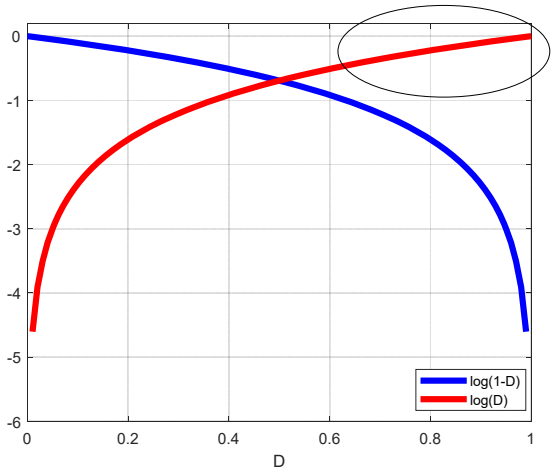
- Vanishing gradient for generator
- Model collapse
- Unstable training
- Sensitive to hyper parameters …

Often requires to check generated samples to evaluate the training process

D and G needs synchronized updates



If D gets trained faster that the generator, gradient from D to G will vanish



https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

GAN : Training

- Model collapse

G collapses to generate limited sample variety; these samples often are good, so D may be fooled

Example of model partially collapse to produce same sample

Some tricks can be found at
https://github.com/soumith/ganhacks

Figure 4. Wasserstein GAN. https://arxiv.org/abs/1701.07875

GAN : Progress is significant

2014

Ian J. Goodfellow et al. Generative Adversarial Nets. NIPS 2014.

Analyzing and Improving the Image Quality of StyleGAN. 2020

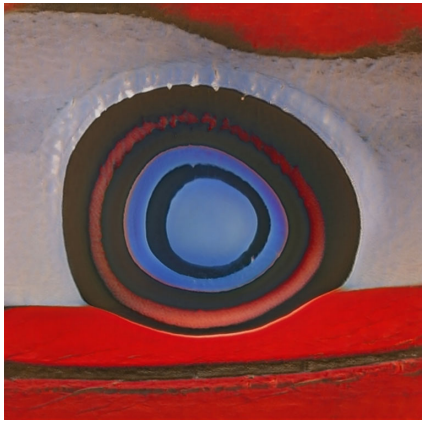
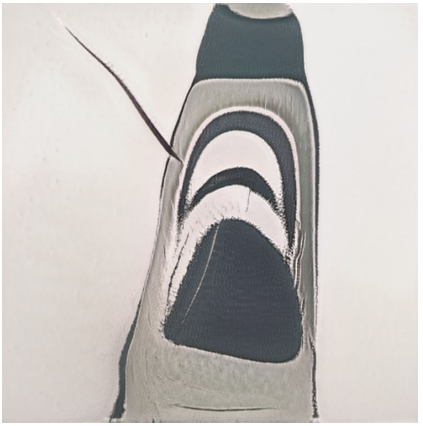
https://thispersondoesnotexist.com/ Try this website

GAN : Other objects

Trained with ImageNet images

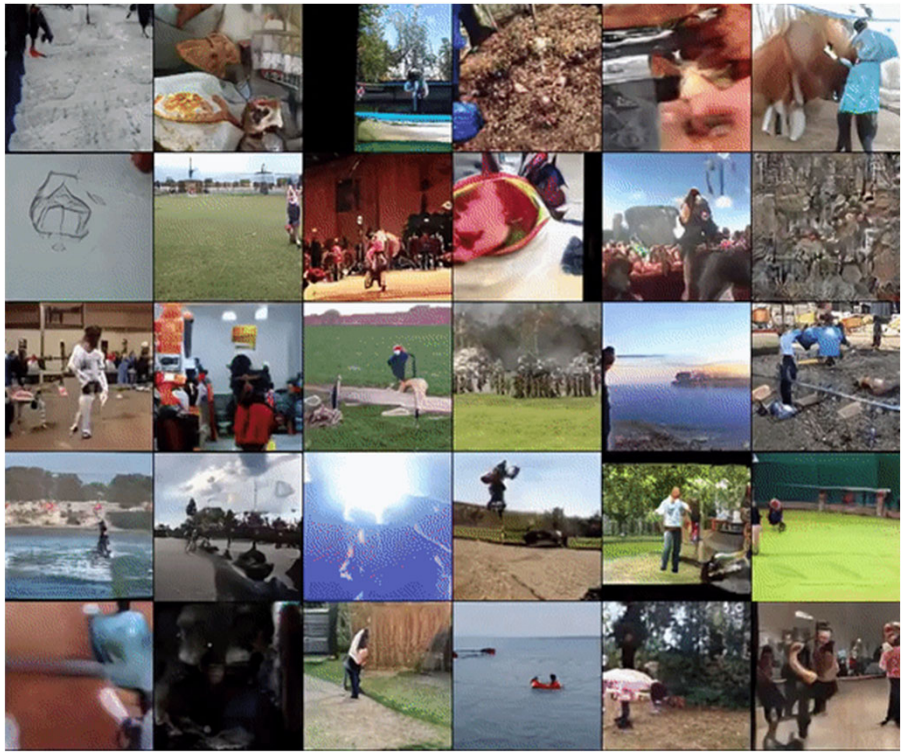
GAN : Art generation



StyleGAN2. 2020

GAN : Video creation

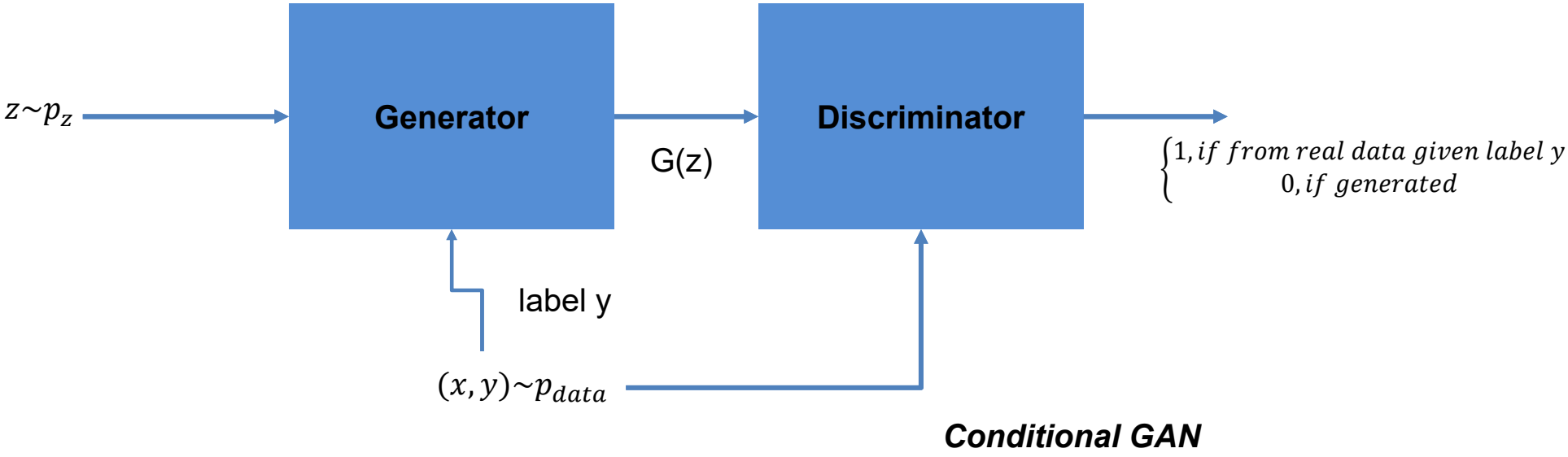


DVD-GAN

DeepMind : Impressive Step Toward Realistic Video Synthesis, 2019

Conditional GAN

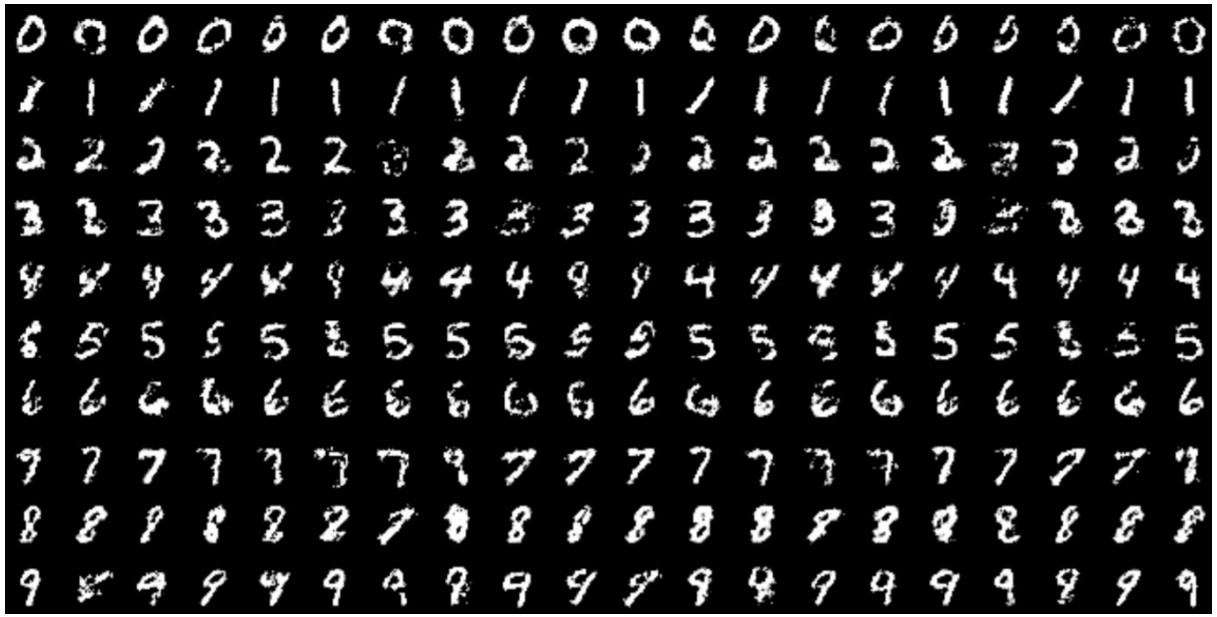
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$



Conditional GAN

Conditional Generative Adversarial Nets. https://arxiv.org/abs/1411.1784. 2014

Conditional GAN



Conditional Generative Adversarial Nets. https://arxiv.org/abs/1411.1784. 2014

Image to image translation: Conditional GAN with complicated label

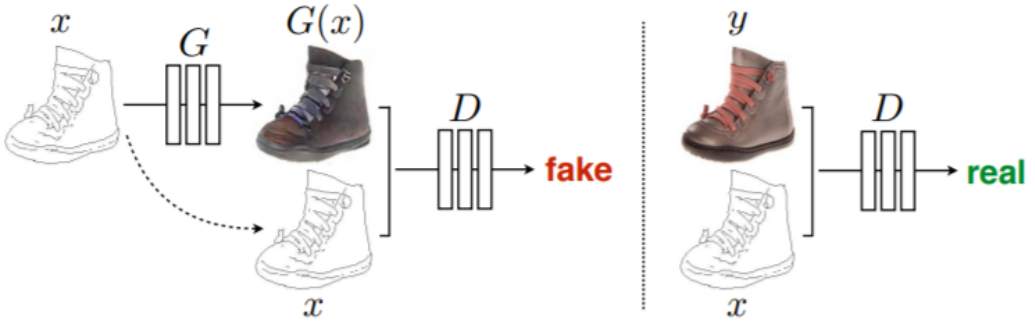


Figure 2: Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

Image-to-Image Translation with Conditional Adversarial Nets. https://arxiv.org/abs/1611.07004. 2018

Image to image translation: Conditional GAN with complicated label

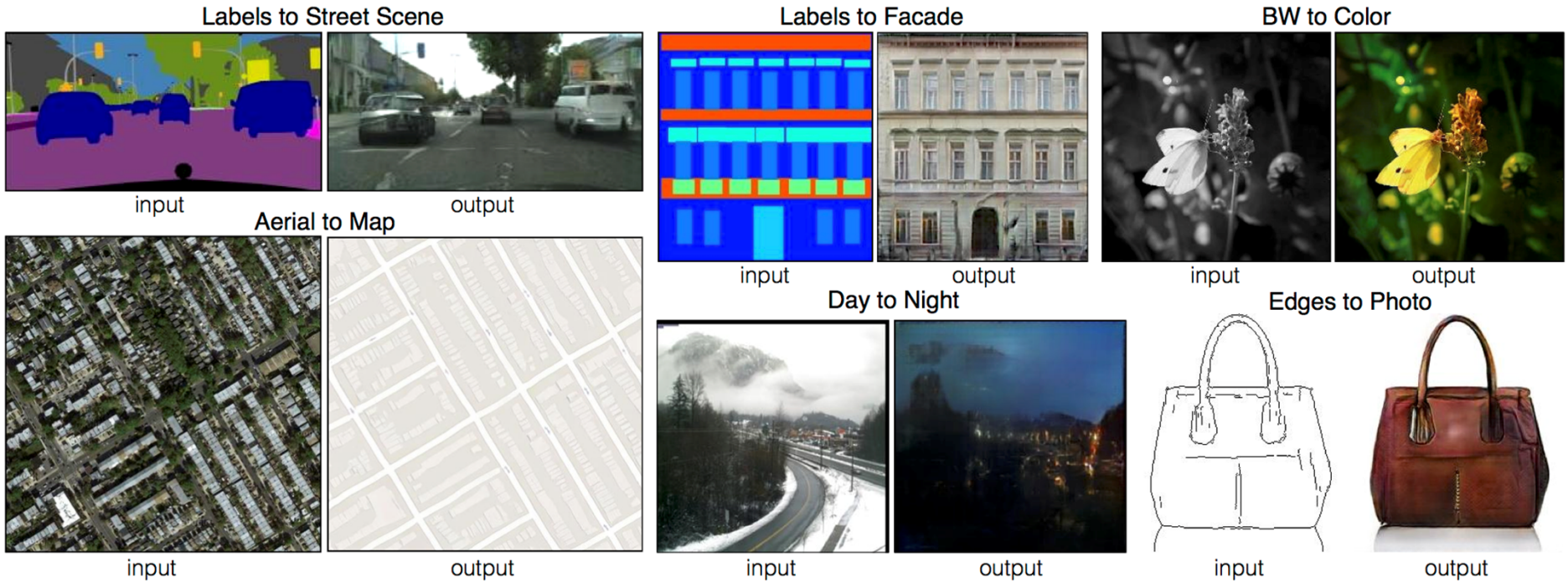
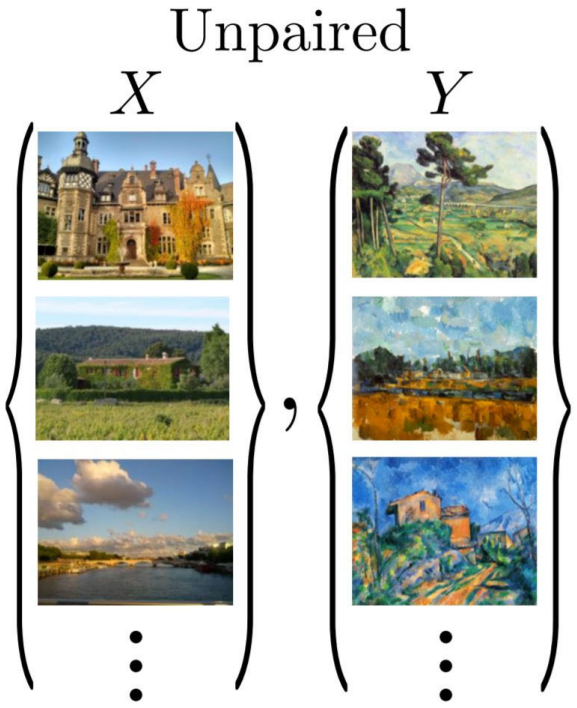
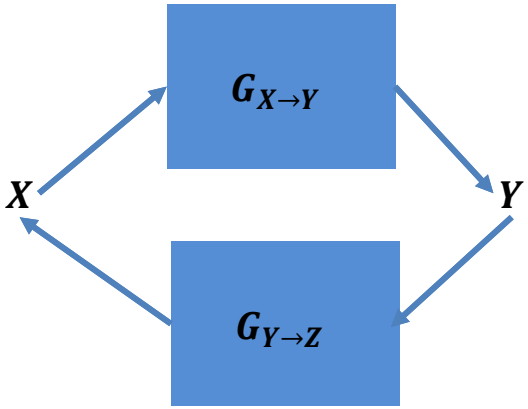


Image-to-Image Translation with Conditional Adversarial Nets. https://arxiv.org/abs/1611.07004. 2018

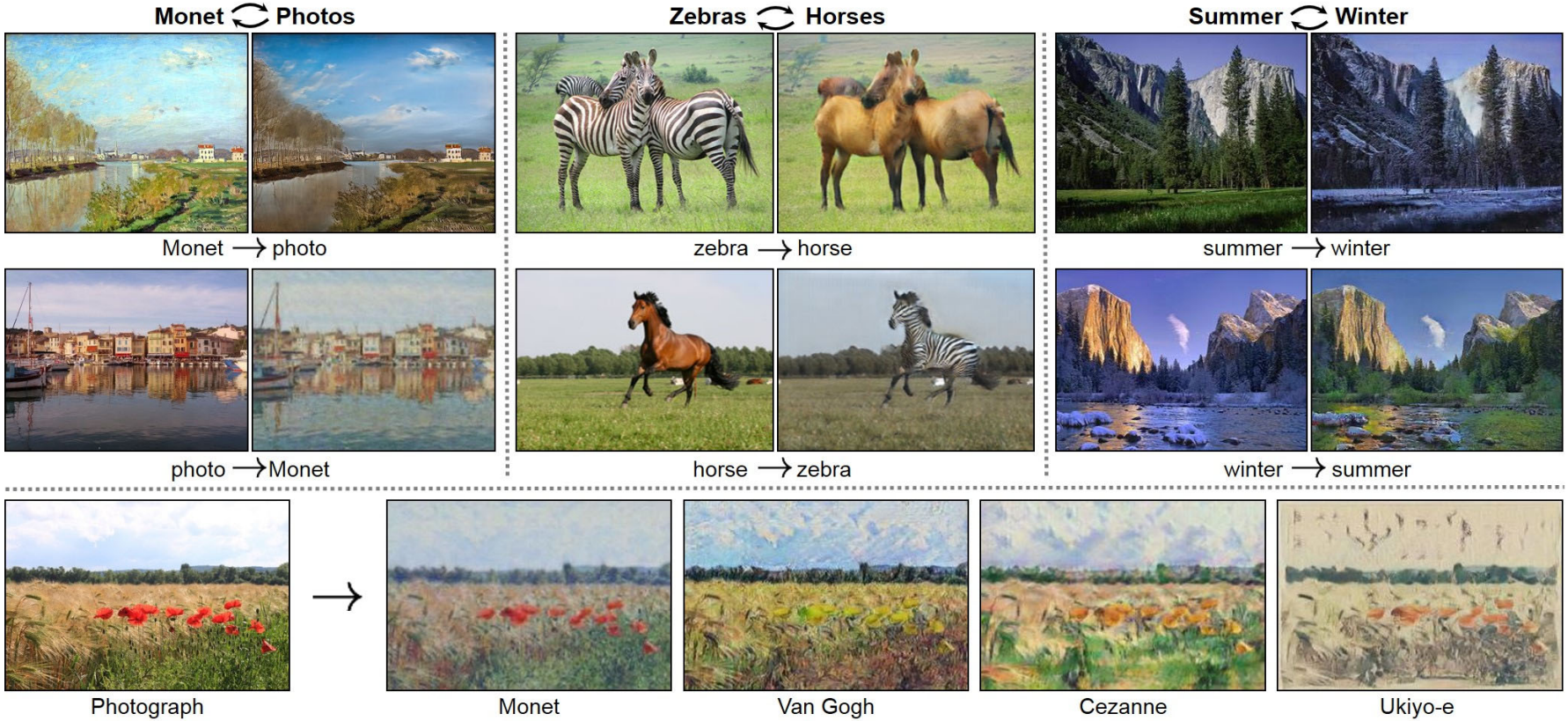
CycleGAN : avoid 1-to-1 mapping between label and sample





Domain transformation with strong consistency

CycleGAN : avoid 1-to-1 mapping between label and sample



https://github.com/junyanz/CycleGAN/

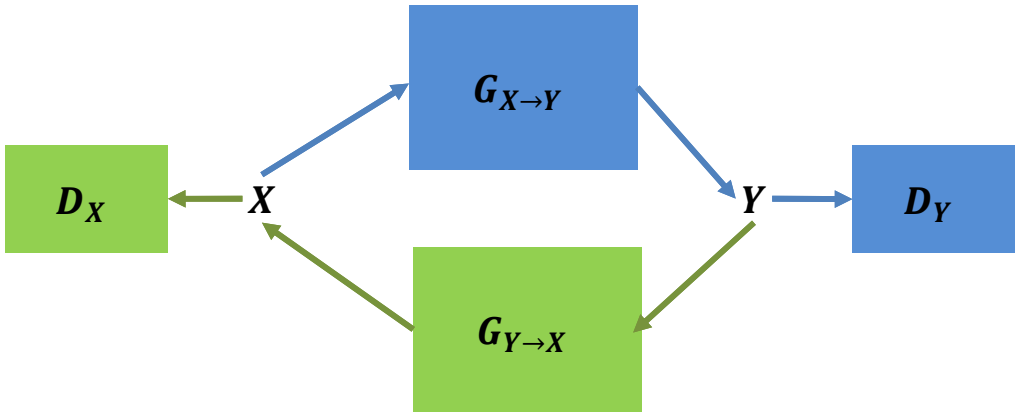
CycleGAN : avoid 1-to-1 mapping between label and sample

Apply to every frame separately

https://github.com/junyanz/CycleGAN/

CycleGAN : avoid 1-to-1 mapping between label and sample

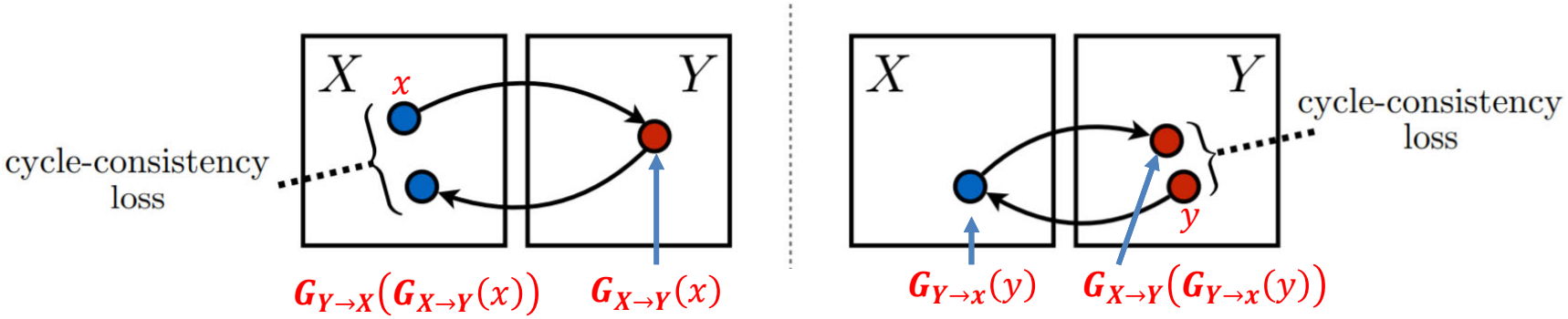
- Give two domains X and Y and corresponding samples
- Two generators $\boldsymbol{G_{X\to Y}}$ and $\boldsymbol{G_{Y\to X}}$
- Two discriminators $\boldsymbol{D_X}$ and $\boldsymbol{D_Y}$
- Two GANs jointly optimized:



$$\min_{\boldsymbol{G_{X\to Y}},\boldsymbol{G_{Y\to X}},\boldsymbol{D_X},\boldsymbol{D_Y}} \ell_{GAN}(X, Y, \boldsymbol{G_{X\to Y}}, \boldsymbol{D_Y}) + \ell_{GAN}(X, Y, \boldsymbol{G_{Y\to X}}, \boldsymbol{D_X})$$

CycleGAN : avoid 1-to-1 mapping between label and sample

- Cycle consistency needed



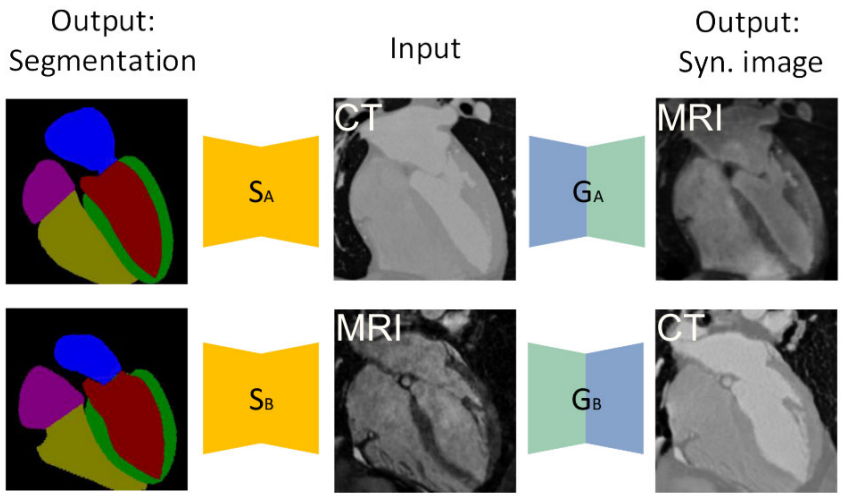
$$\min_{G_{X\to Y},G_{Y\to X},D_X,D_Y}\left\{\begin{array}{c}\ell_{GAN}(X,Y,G_{X\to Y},D_Y)+\ell_{GAN}(X,Y,G_{Y\to X},D_X)\\ +\lambda(E_{x\sim X}\left[\left|G_{Y\to X}(G_{X\to Y}(x))-x\right|_1\right]+E_{y\sim Y}\left[\left|G_{X\to Y}(G_{Y\to x}(y))-y\right|_1\right])\end{array}\right\}$$

https://junyanz.github.io/CycleGAN/ ⟵ for great examples

GAN framework is flexible

- GAN is a flexible framework
- Unique power is from the G/D gaming of two DNNs
- Ability to sample with high resolution and details

- The CycleGAN idea can be further extended to be conditioned on labels too



Train GAN to transfer CT to MRI and MRI to CT

For medical imaging, the key anatomical structures should be kept consistent in generated images

Combine GAN loss, cycle loss and segmentation loss to achieve this

Translating and Segmenting Multimodal Medical Volumes with Cycle- and Shape-Consistency Generative Adversarial Network. https://arxiv.org/abs/1802.09655

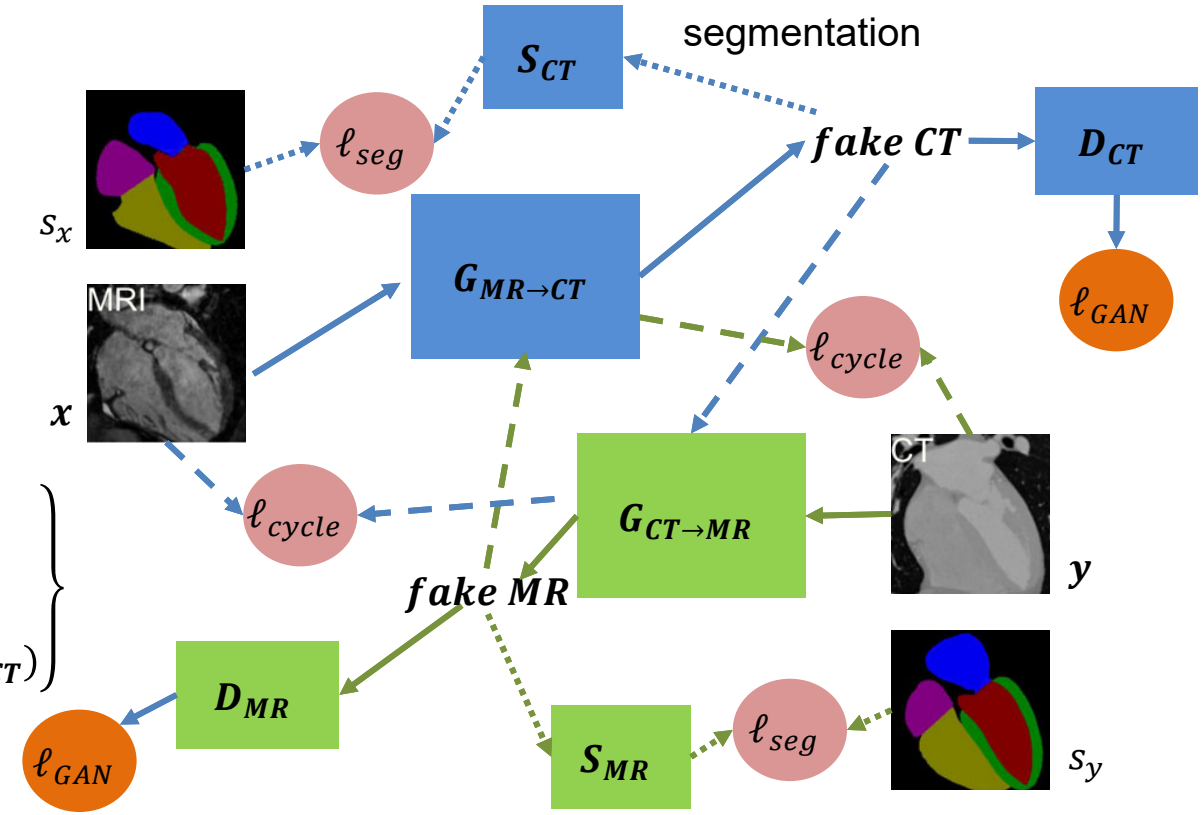
CycleGAN with segmentation consistency

Every sample comes with its segmentation label (x, s_x), (y, s_y)

Train two more segmentation networks (e.g. use U-Net) : $\boldsymbol{S_X}$, $\boldsymbol{S_y}$, receiving the images and generate segmentation results

$$\min_{\substack{\boldsymbol{G_{CT\to MR}},\\ \boldsymbol{G_{MR\to CT}},\\ \boldsymbol{D_{MR}},\boldsymbol{D_{CT}},\\ \boldsymbol{S_{MR}},\boldsymbol{S_{CT}}}} \left\{ \begin{array}{c} \ell_{GAN}(CT, MR, \boldsymbol{G_{CT\to MR}}, \boldsymbol{D_{MR}}) \\ +\ell_{GAN}(CT, MR, \boldsymbol{G_{MR\to CT}}, \boldsymbol{D_{CT}}) \\ +\lambda\ell_{cycle}(CT, MR, \boldsymbol{G_{CT\to MR}}, \boldsymbol{G_{MR\to CT}}) \\ +\beta\ell_{seg}(CT, MR, \boldsymbol{S_{MR}}, \boldsymbol{S_{CT}}, \boldsymbol{G_{CT\to MR}}, \boldsymbol{G_{MR\to CT}}) \end{array} \right\}$$

ℓ_{seg} can be pixel-wise cross-entropy loss or soft Dice loss or combined



Translating and Segmenting Multimodal Medical Volumes with Cycle- and Shape-Consistency Generative Adversarial Network. https://arxiv.org/abs/1802.09655

CycleGAN with segmentation consistency

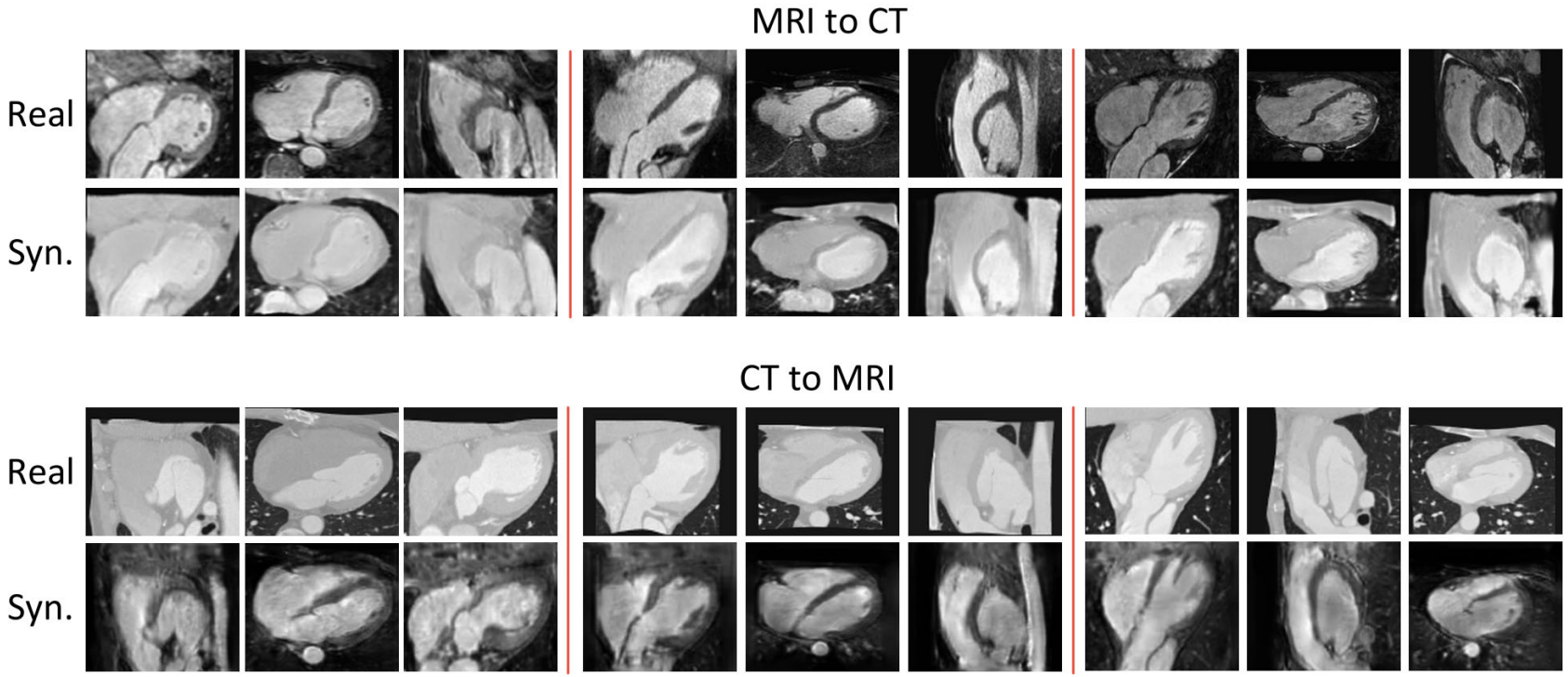
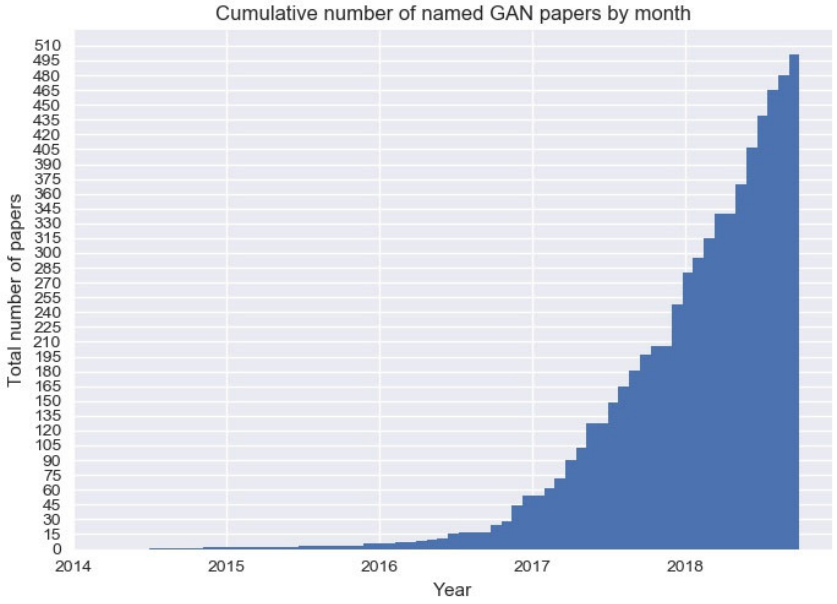


Figure 4: Qualitative results of our translation from MRI to CT (first row) and from CT to MRI (second row). For each sample (in one out of six grids), we show three orthogonal cuts through the center of 3D volumes.

Translating and Segmenting Multimodal Medical Volumes with Cycle- and Shape-Consistency Generative Adversarial Network.
https://arxiv.org/abs/1802.09655

More resources



https://github.com/hindupuravinash/the-gan-zoo