Learning Generative Adversarial Networks

Qiyang Hu
UCLA Office of Advanced Research Computing
Feb 25, 2022

In this talk

bit.ly/**LDL_repo**

Introduction

01

- What is GANs?
- Training of GANs

GANs Hands-On

02

- Today's project
- DCGANs
- GANs in PyTorch

GANs World

03

- Variant GANs
  - SAGAN, CGAN, CycleGAN
- Challenges in GANs

In this talk

Introduction

- What is GANs?
- Training of GANs

01

GANs Hands-On

- Today's project
- DCGANs
- GANs in PyTorch

02

GANs World

- Variant GANs
  - SAGAN, CGAN, CycleGAN
- Challenges in GANs

03

Generative Adversarial Networks

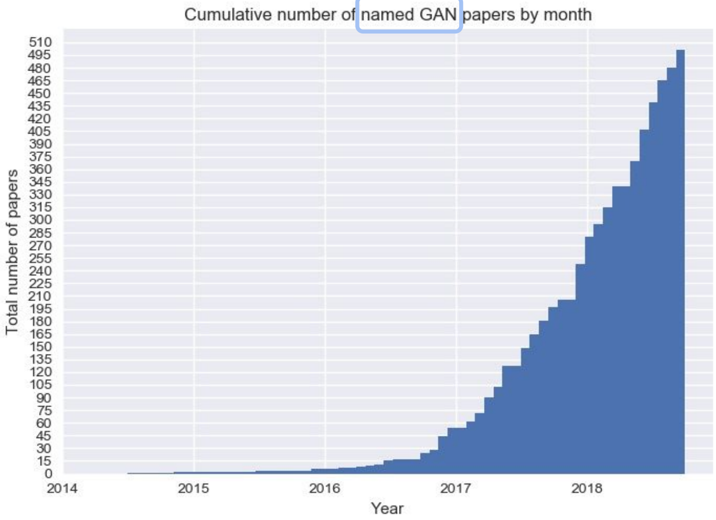
"GAN-Father"

Ian Goodfellow @goodfellow_ian · Jan 14, 2019

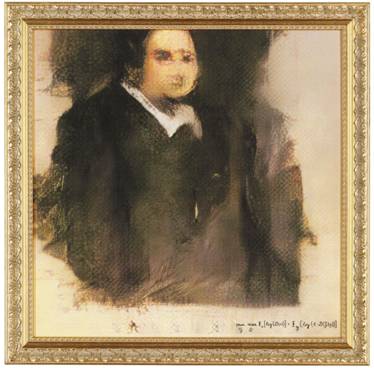
4.5 years of GAN progress on face generation. arxiv.org/abs/1406.2661 arxiv.org/abs/1511.06434 arxiv.org/abs/1606.07536 arxiv.org/abs/1710.10196 arxiv.org/abs/1812.04948

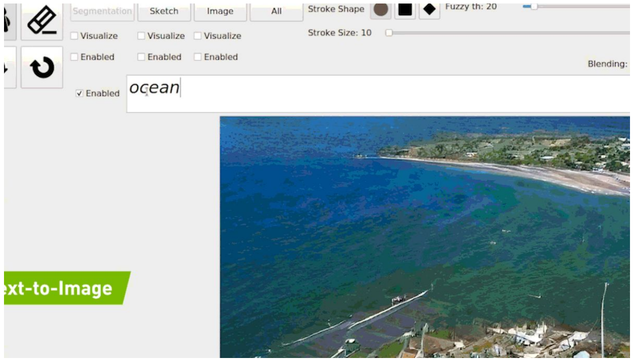
42 1.5K 3.7K

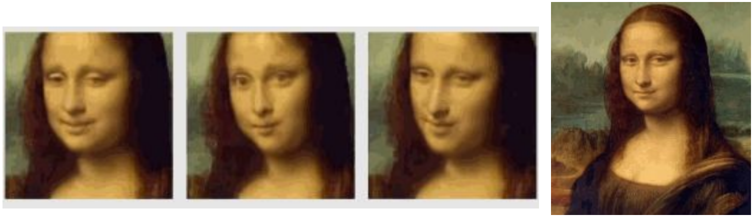
GAN, CGAN, DCGAN, RGAN, SGAN...

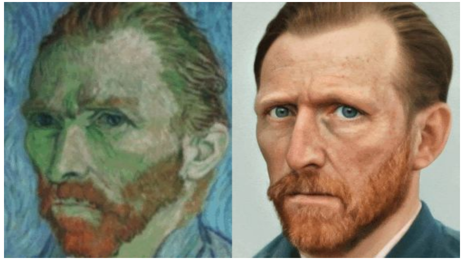


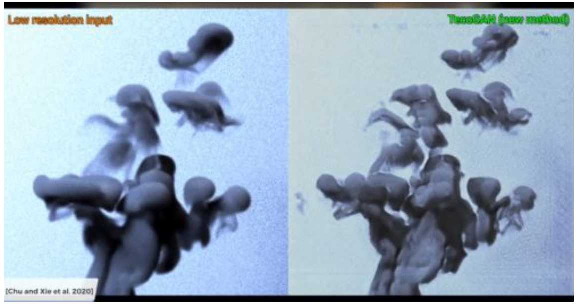
The only limit of GAN is our imagination











Qiyang Hu

GAN: Generative

Training Data (source)

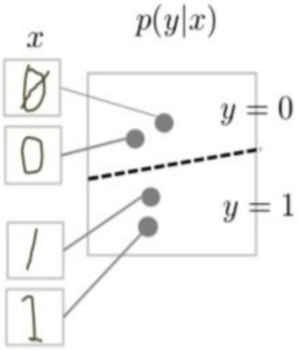
$p_{\text{data}}(\mathbf{x})$

Generated Samples (source)

$p_{\text{model}}(\mathbf{x})$

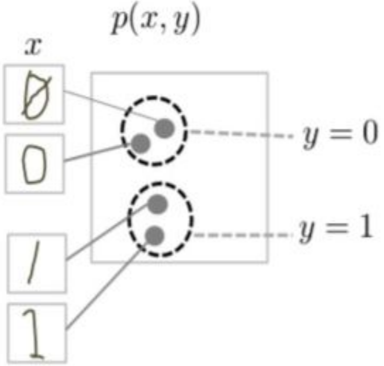
$\sim$

- Discriminative model
  - discriminate labels of data instances
  - try to draw boundaries in the data space



  - capture the conditional prob. *p(Y | X)*
  - measure the misfit of points
  - learn the difference, ignore correlations

- Generative model
  - generate new data instances
  - try to model how data is placed



  - capture the joint prob. *p(X, Y)*
  - measure the misfit of prob distributions
  - learn distributions to capture correlations

Taxonomy of Generative Models

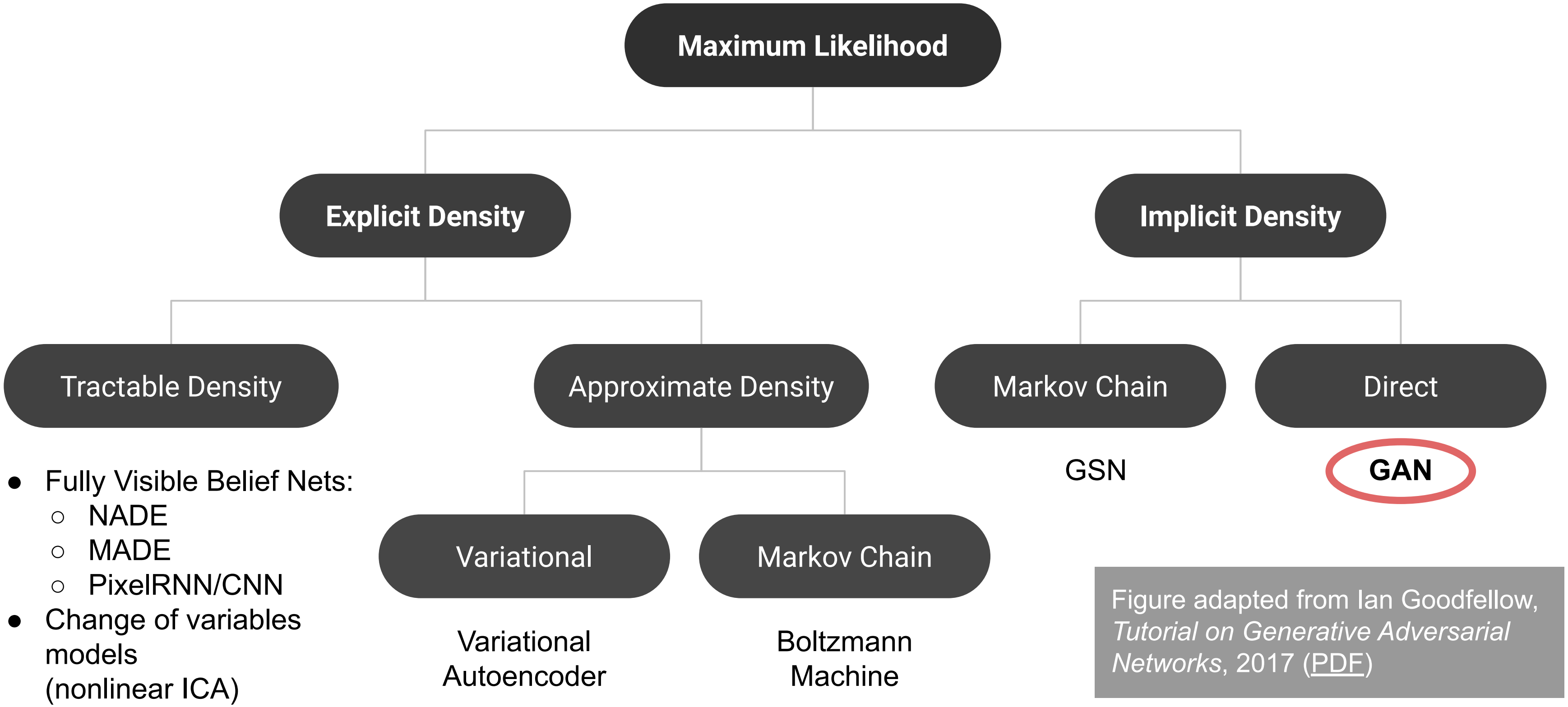


Figure adapted from Ian Goodfellow, *Tutorial on Generative Adversarial Networks*, 2017 (PDF)

Qiyang Hu

GAN: Adversarial

- Generator: generate plausible data
- Discriminator: distinguish the generator's fake data from real data

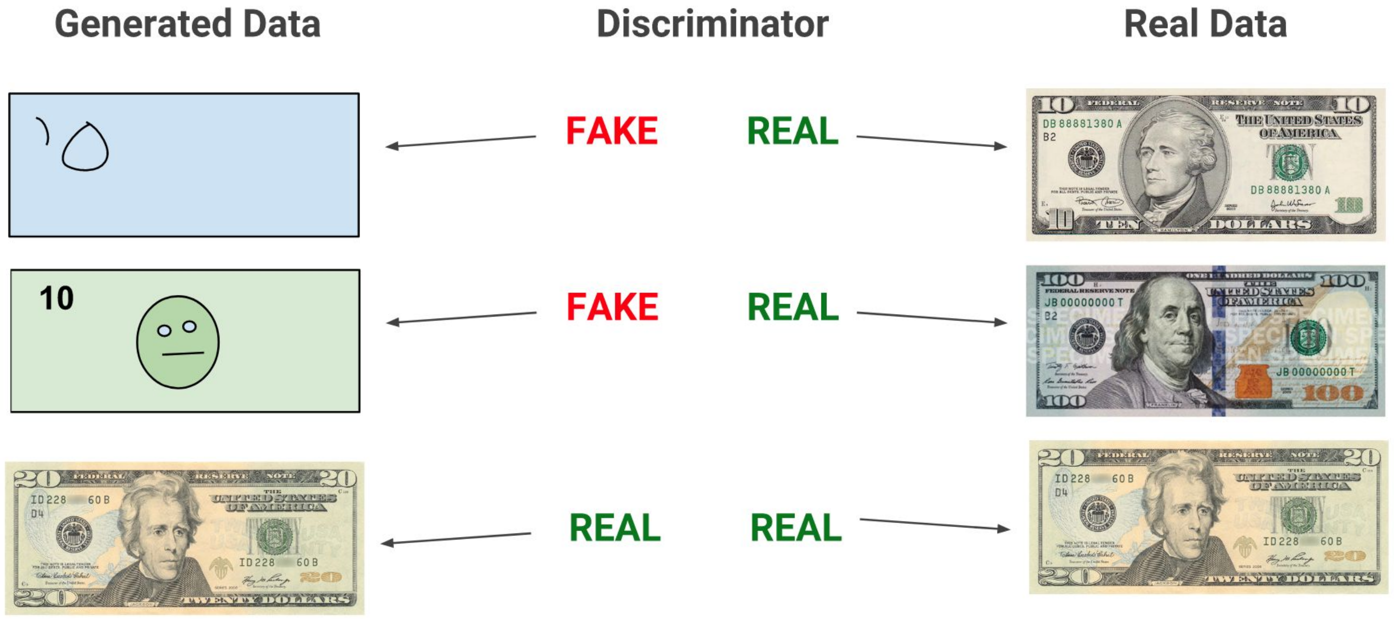
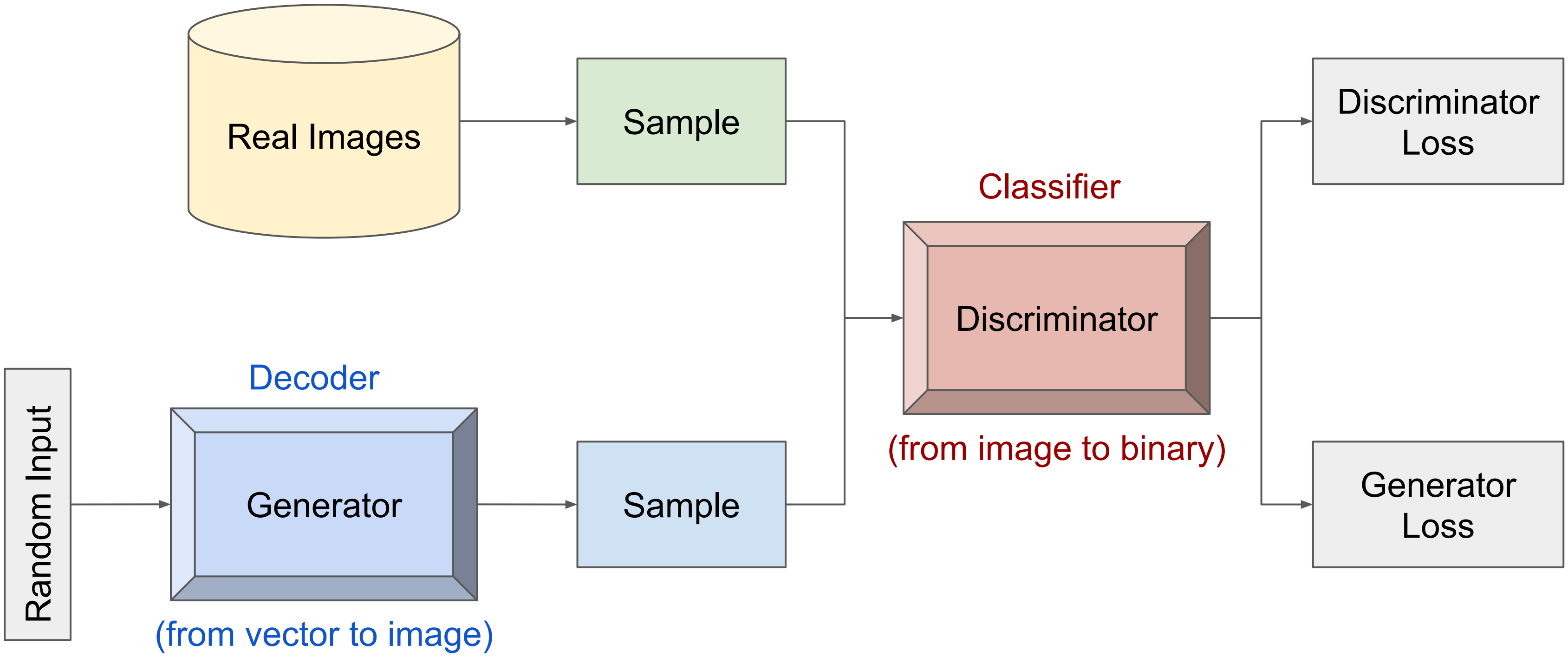


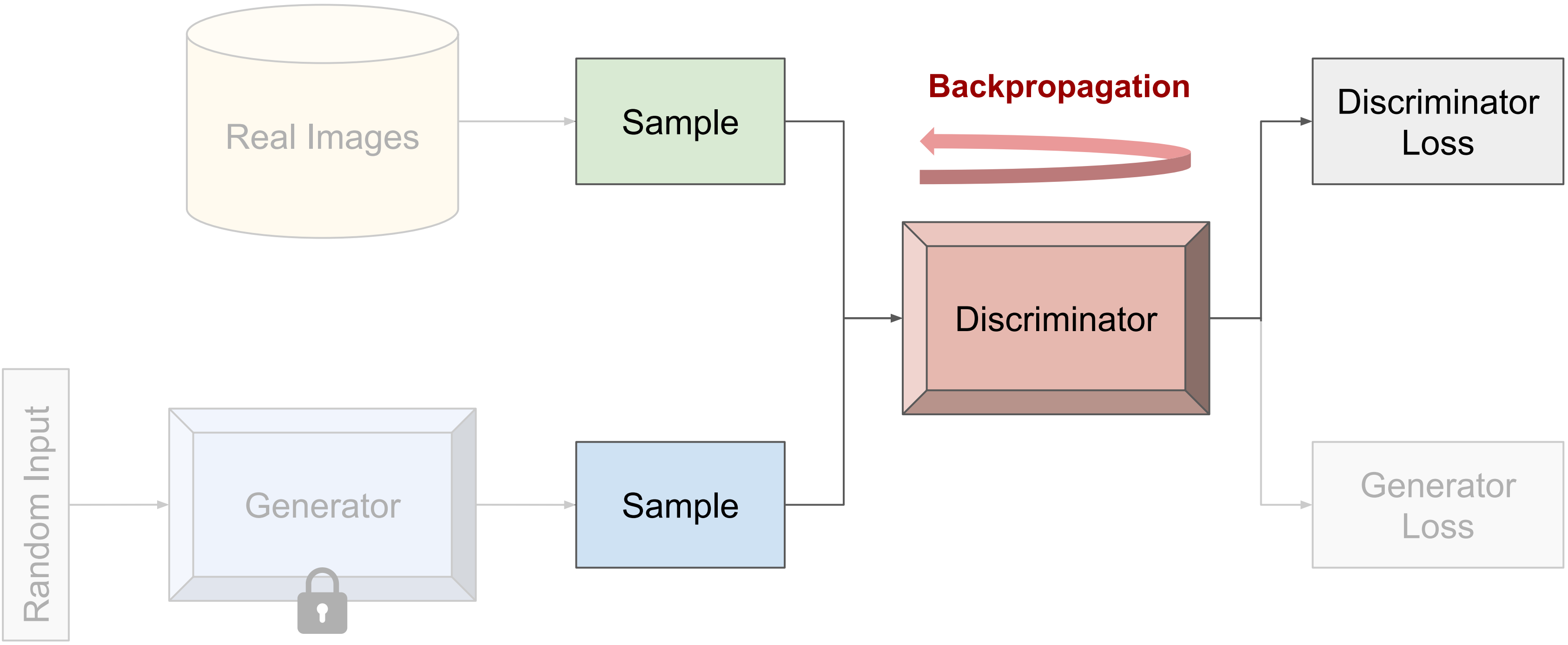
Image Source

GAN: Network



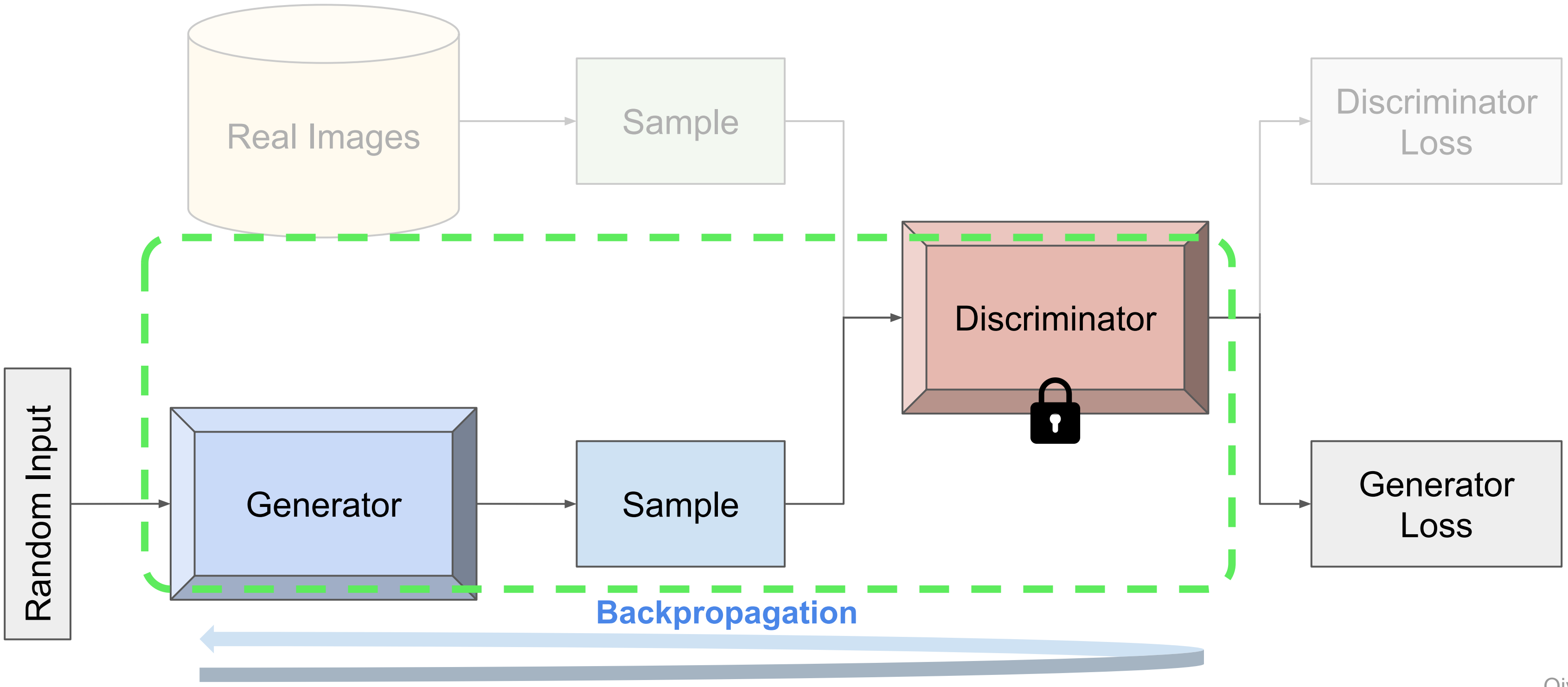
Qiyang Hu

Training of GAN (1): update discriminator



Qiyang Hu

Training of GAN (2): update generator



Training of GAN (3): iterate the 2 steps to converge

- Alternate the training periods
  - The discriminator trains for one or more epochs with locking generator
  - The generator trains for one or more epochs with locking discriminator
  - Repeat the above steps
- When to stop
  - While generator improves, discriminator performance gets worse
    - Generator becomes perfect ⇒ discriminator gets 50% accuracy
  - Feedback from discriminator is less meaningful over time
    - At some point discriminator starts giving completely random feedback
    - Generator starts to train on junk feedback, and its own quality may collapse
  - Convergence of GANs is unstable, very hard to identify

The design logic behind the GAN structure

- Why do we need discriminator ?
  - There <u>are</u> generative models that can learn without discriminator
    - e.g. Variational Autoencoder (VAE)
  - Generator constructs the images in a bottom-up way
    - Very hard to capture the higher-level correlations
  - The discriminator can guide the generator with correlation info in a criticizing way
- Why do we need generator?
  - There <u>are</u> generative models that can learn without generator
    - e.g. Energy based model
  - Discriminator constructs the images in a top-down way
    - Very hard to learn from constructing negative sampling
  - The generated instances become negative training examples for the discriminator.

Standard Loss function for GAN

- Minimax Loss
  - Proposed in the original Goodfellow's paper

Real image sample

Noise sample

Fake image sample

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

Probability of the real image is real

Probability of the fake image is real

  - Derives from a single measure of distance (BCE) between the real and generated distributions

- In practice
  - Discriminator loss: maximize $\frac{1}{n}\sum_{i=0}^{n} \log(D(x_i)) + \frac{1}{n}\sum_{i=0}^{n} \log[1 - D(G(z_i))]$
  - Generator (not-saturating) loss: maximize $\frac{1}{n}\sum_{i=0}^{n} \log[D(G(z_i))]$

GANs are *very* difficult to train.

- Discriminator shouldn't be too good.
  - Good discriminator ⇒ always 100% accuracy
  - Generator has no positive case to follow for learning.
  - Mathematically, falling into the vanishing-gradient zone
  - Generator needs some success, esp. in early stages
- Discriminator shouldn't be too bad.
  - Bad discriminator ⇒ random guess
  - Generator cannot get helpful feedback, esp. in late stages

Training Tips For GANs

- https://github.com/soumith/ganhacks
- Need experiences as always

Input Normalizing

- normalize the images to (-1, 1)
- Tanh as the last layer of the generator output

Implemented in Demo: Yes

Tune the learning rates

- Make D not improve too fast
- Make D not improve too slow

Implemented in Demo: Yes

A modified loss function

- Generator loss function to be max log(D)
- Flip labels when training generator: real = fake, fake=real

Implemented in Demo: Yes

BatchNorm

Construct different mini-batches for real and generated samples

Implemented in Demo: Yes

Add noise to inputs

- Perturb the both real and fake images when training D
- Decay the noise over time.

Implemented in Demo: No

• • •

Avoid Sparse Gradients: ReLU, MaxPool

- LeakyRL is good for G and D
- Use stride, not pooling

Implemented in Demo: Yes

Use Soft and Noisy Labels

- Real ~ Uniform(0.7, 1.2)
- Fake ~ Uniform(0.0, 0.3)

As homework

Use DCGAN or Hybrid

- Use DCGAN if possible
- If not, use hybrid of KL + GAN or VAE + GAN

Implemented in Demo: Yes

Use the ADAM Optimizer

optim.Adam rules.

Implemented in Demo: Yes

Use Dropouts in G

- Provide noise in the form of dropout (50%)
- Apply at both training and test time

Implemented in Demo: No

In this talk

Introduction

- What is GANs?
- Training of GANs

01

GANs Hands-On

- Today's project
- DCGANs
- GANs in PyTorch

02

GANs World

- Variant GANs
  - SAGAN, CGAN, CycleGAN
- Challenges in GANs

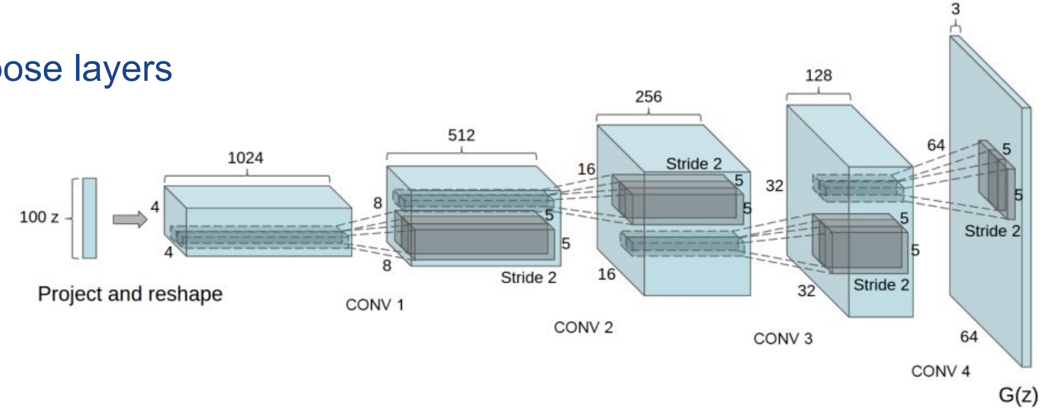
03

Today’s Demo — Generative Dog Images from Kaggle

- Experiment with creating puppy pics
  - A Kernels-only competition (total 10K prize, expired years ago)
  - [Evaluation]
    - Using a pre-trained model (Inception)
    - Calculating MiFID scores
- Using [Stanford Dogs Dataset]
  - 20,580 images with annotation info (120 breeds, bounding box)
  - Some dog pictures are very *tricky*
    - Only part of the dogs body
    - Having multiple dogs
    - Having multiple persons
    - Dogs may occupy <⅕ of the picture
    - With various texts (from memes, magazine, etc)
    - Even wild predators included

Neural Networks for Generator and Discriminator

- Deep Convolutional Generative Adversarial Networks (DCGANs)
- Generator
  - Input: a std-norm latent vector
  - Strided 2D Convolutional-transpose layers
  - Batch norm layers
  - ReLU activations
  - Convtrans+*Tanh* before output
  - Output: a 3x64x64 RGB image



- Discriminator
  - Input: 3x64x64 input image
  - Strided convolution layers, batch norm layers, LeakyReLU activations
  - Conv+*Sigmoid* before output
  - Output: a scalar probability

Generator Implementation in PyTorch

```
class Generator(nn.Module):
    def __init__(self, nz=128, channels=3):
        super(Generator, self).__init__()

        self.nz = nz
        self.channels = channels

        def convlayer(n_input, n_output, k_size=4, stride=2, padding=0):
            block = [
                nn.ConvTranspose2d(n_input, n_output, kernel_size=k_size, stride=stride, padding=padding, bias=False),
                nn.BatchNorm2d(n_output),
                nn.ReLU(inplace=True),
            ]
            return block

        self.model = nn.Sequential(
            *convlayer(self.nz, 1024, 4, 1, 0), # Fully connected layer via convolution.
            *convlayer(1024, 512, 4, 2, 1),
            *convlayer(512, 256, 4, 2, 1),
            *convlayer(256, 128, 4, 2, 1),
            *convlayer(128, 64, 4, 2, 1),
            nn.ConvTranspose2d(64, self.channels, 3, 1, 1),
            nn.Tanh()
        )

    def forward(self, z):
        z = z.view(-1, self.nz, 1, 1)
        img = self.model(z)
        return img
```

Discriminator Implementation in PyTorch

```
class Discriminator(nn.Module):
    def __init__(self, channels=3):
        super(Discriminator, self).__init__()

        self.channels = channels

        def convlayer(n_input, n_output, k_size=4, stride=2, padding=0, bn=False):
            block = [nn.Conv2d(n_input, n_output, kernel_size=k_size, stride=stride, padding=padding, bias=False)]
            if bn:
                block.append(nn.BatchNorm2d(n_output))
            block.append(nn.LeakyReLU(0.2, inplace=True))
            return block

        self.model = nn.Sequential(
            *convlayer(self.channels, 32, 4, 2, 1),
            *convlayer(32, 64, 4, 2, 1),
            *convlayer(64, 128, 4, 2, 1, bn=True),
            *convlayer(128, 256, 4, 2, 1, bn=True),
            nn.Conv2d(256, 1, 4, 1, 0, bias=False),  # FC with Conv.
        )

    def forward(self, imgs):
        logits = self.model(imgs)
        out = torch.sigmoid(logits)

        return out.view(-1, 1)
```

Training loop

```
############################
# (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
###########################
# train with real
netD.zero_grad()
real_images = real_images.to(device)
batch_size = real_images.size(0)
labels = torch.full((batch_size, 1), real_label, device=device)

output = netD(real_images)
errD_real = criterion(output, labels)
errD_real.backward()
D_x = output.mean().item()

# train with fake
noise = torch.randn(batch_size, nz, 1, 1, device=device)
fake = netG(noise)
labels.fill_(fake_label)
output = netD(fake.detach())
errD_fake = criterion(output, labels)
errD_fake.backward()
D_G_z1 = output.mean().item()
errD = errD_real + errD_fake
optimizerD.step()
```

```
    real_label = 0.9
    fake_label = 0
```

real_label **≠ 1** to make the discriminator not learn too quickly

```
############################
# (2) Update G network: maximize log(D(G(z)))
###########################
netG.zero_grad()
labels.fill_(real_label)  # fake labels are real for generator cost
output = netD(fake)
errG = criterion(output, labels)
errG.backward()
D_G_z2 = output.mean().item()
optimizerG.step()
```

Colab Hands-on

bit.ly/**LDL_gan**

In this talk

Introduction

- What is GANs?
- Training of GANs

01

GANs Hands-On

- Today's project
- DCGANs
- GANs in PyTorch

02

GANs World

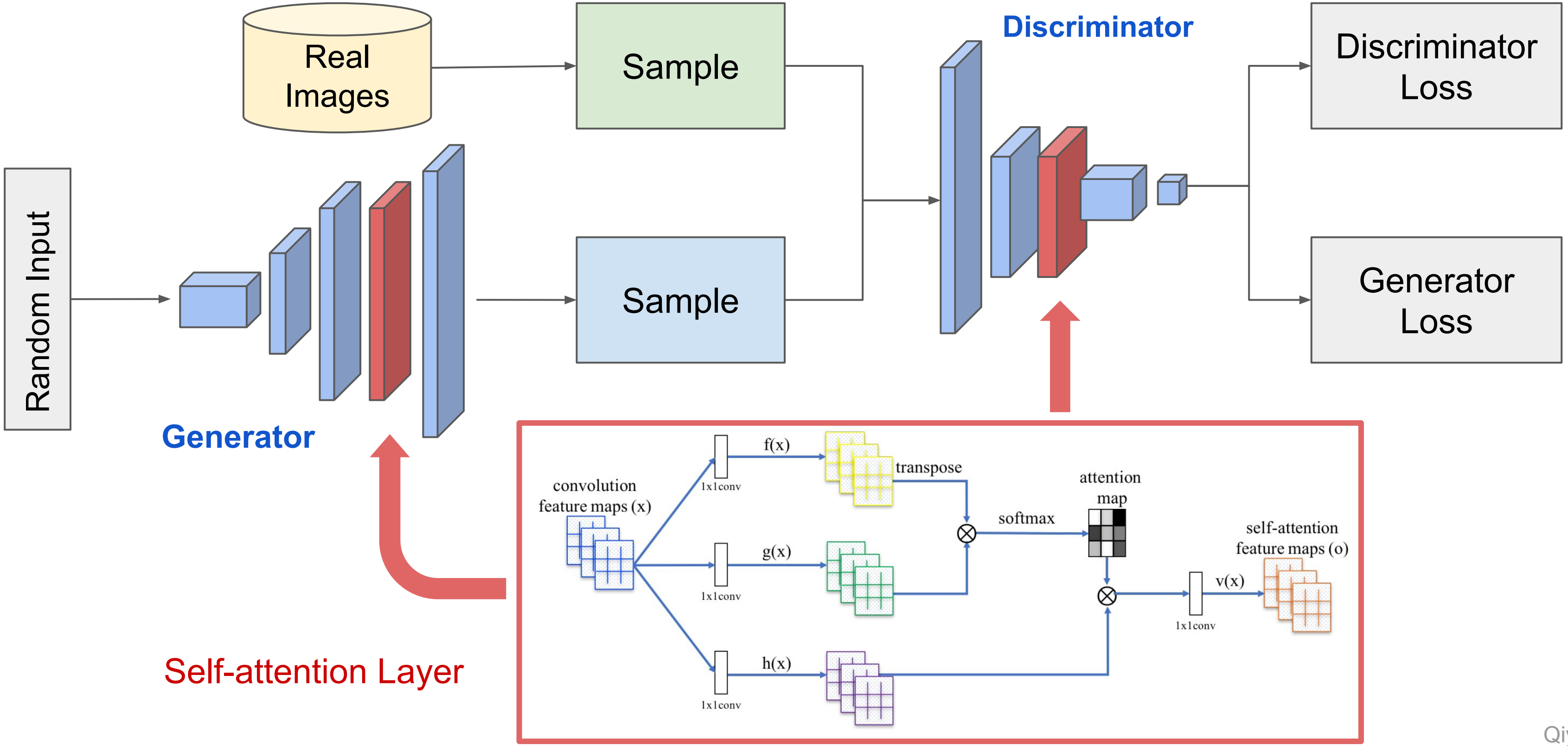
- Variant GANs
    - SAGAN, CGAN, CycleGAN
- Challenges in GANs

03

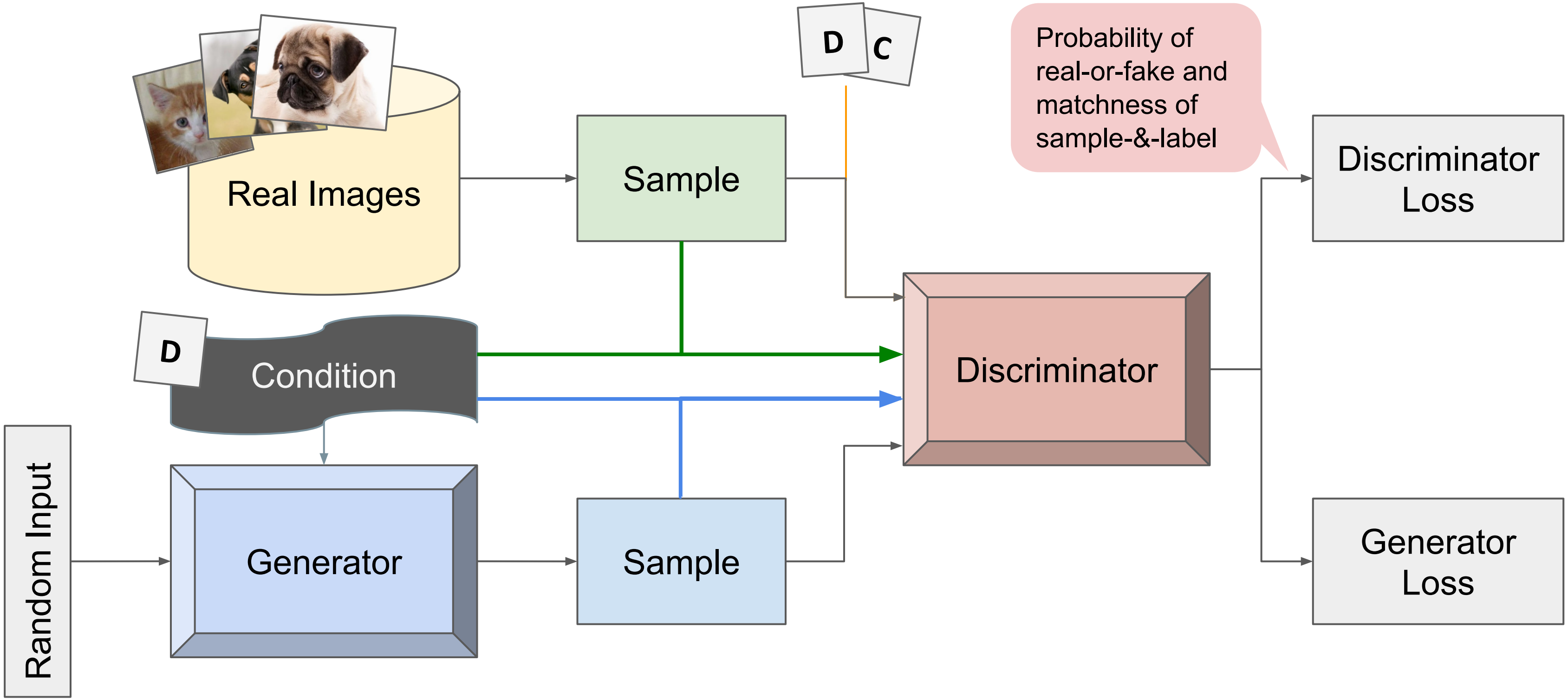
A *lot* of different GANs!

- Various design of network structures
  - SAGANs
  - Conditional GANS
  - CycleGANs
  - InfoGANs
  - EB-GANs
  - VAE-GANs
  - BiGANs
  - Triple-GANs
  - …
- Various metrics for objective functions
  - WGANs
  - LSGANs
  - RGANs
  - Cramer GANs
  - Fisher GANs
  - MMD GANs
  - McGANs
  - HingeGANs
  - ...
- Combining the two
  - BEGAN
  - MAGANs
  - ...

SAGAN: DCGANs + Self-Attention Layer (Zhang, et al. 2018)

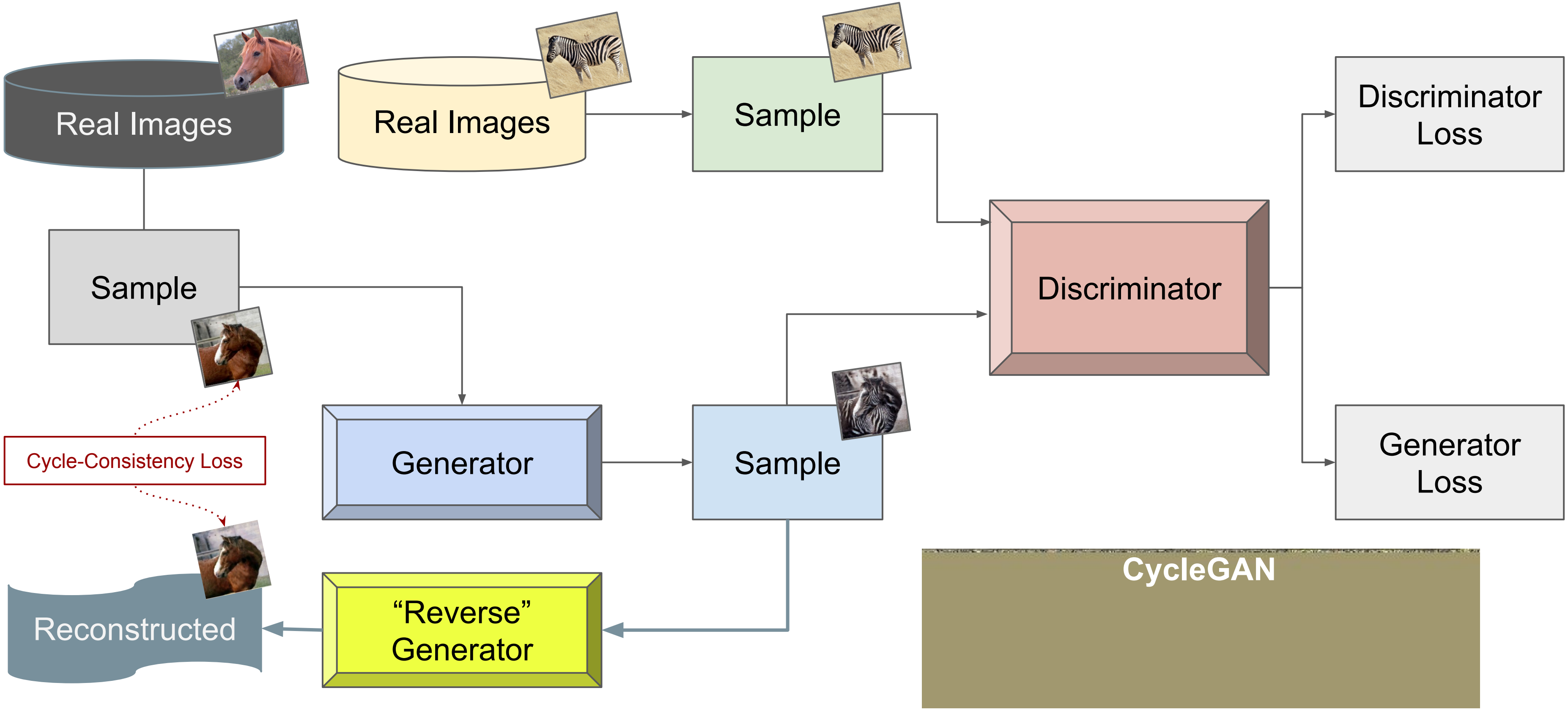


Conditional GAN: generate images with specific class



Qiyang Hu

CycleGAN: *unsupervised* conditional GAN



Qiyang Hu

Some GAN loss function variations

SGAN (non-saturating)

$$L_D^{SGAN} = -\mathbb{E}_{x_r\sim\mathbb{P}}\left[\log\left(\text{sigmoid}(C(x_r))\right)\right] - \mathbb{E}_{x_f\sim\mathbb{Q}}\left[\log\left(1-\text{sigmoid}(C(x_f))\right)\right]$$

$$L_G^{SGAN} = -\mathbb{E}_{x_f\sim\mathbb{Q}}\left[\log\left(\text{sigmoid}(C(x_f))\right)\right]$$

RSGAN

$$L_D^{RSGAN} = -\mathbb{E}_{(x_r,x_f)\sim(\mathbb{P},\mathbb{Q})}\left[\log(\text{sigmoid}(C(x_r)-C(x_f)))\right]$$

$$L_G^{RSGAN} = -\mathbb{E}_{(x_r,x_f)\sim(\mathbb{P},\mathbb{Q})}\left[\log(\text{sigmoid}(C(x_f)-C(x_r)))\right]$$

RaSGAN

$$L_D^{RaSGAN} = -\mathbb{E}_{x_r\sim\mathbb{P}}\left[\log\left(\tilde{D}(x_r)\right)\right] - \mathbb{E}_{x_f\sim\mathbb{Q}}\left[\log\left(1-\tilde{D}(x_f)\right)\right]$$

$$L_G^{RaSGAN} = -\mathbb{E}_{x_f\sim\mathbb{Q}}\left[\log\left(\tilde{D}(x_f)\right)\right] - \mathbb{E}_{x_r\sim\mathbb{P}}\left[\log\left(1-\tilde{D}(x_r)\right)\right]$$

$$\tilde{D}(x_r) = \text{sigmoid}\left(C(x_r) - \mathbb{E}_{x_f\sim\mathbb{Q}}C(x_f)\right)$$
$$\tilde{D}(x_f) = \text{sigmoid}\left(C(x_f) - \mathbb{E}_{x_r\sim\mathbb{P}}C(x_r)\right)$$

LSGAN

$$L_D^{LSGAN} = \mathbb{E}_{x_r\sim\mathbb{P}}\left[(C(x_r)-0)^2\right] + \mathbb{E}_{x_f\sim\mathbb{Q}}\left[(C(x_f)-1)^2\right]$$

$$L_G^{LSGAN} = \mathbb{E}_{x_f\sim\mathbb{Q}}\left[(C(x_f)-0)^2\right]$$

RaLSGAN

$$L_D^{RaLSGAN} = \mathbb{E}_{x_r\sim\mathbb{P}}\left[(C(x_r) - \mathbb{E}_{x_f\sim\mathbb{Q}}C(x_f) - 1)^2\right] + \mathbb{E}_{x_f\sim\mathbb{Q}}\left[(C(x_f) - \mathbb{E}_{x_r\sim\mathbb{P}}C(x_r) + 1)^2\right]$$

$$L_G^{RaLSGAN} = \mathbb{E}_{x_f\sim\mathbb{P}}\left[(C(x_f) - \mathbb{E}_{x_r\sim\mathbb{P}}C(x_r) - 1)^2\right] + \mathbb{E}_{x_r\sim\mathbb{P}}\left[(C(x_r) - \mathbb{E}_{x_f\sim\mathbb{Q}}C(x_f) + 1)^2\right]$$

HingeGAN

$$L_D^{HingeGAN} = \mathbb{E}_{x_r\sim\mathbb{P}}\left[\max(0, 1-C(x_r))\right] + \mathbb{E}_{x_f\sim\mathbb{Q}}\left[\max(0, 1+C(x_f))\right]$$

$$L_G^{HingeGAN} = -\mathbb{E}_{x_f\sim\mathbb{Q}}\left[C(x_f)\right]$$

RaHingeGAN

$$L_D^{HingeGAN} = \mathbb{E}_{x_r\sim\mathbb{P}}\left[\max(0, 1-\tilde{D}(x_r))\right] + \mathbb{E}_{x_f\sim\mathbb{Q}}\left[\max(0, 1+\tilde{D}(x_f))\right]$$

$$L_G^{HingeGAN} = \mathbb{E}_{x_f\sim\mathbb{P}}\left[\max(0, 1-\tilde{D}(x_f))\right] + \mathbb{E}_{x_r\sim\mathbb{Q}}\left[\max(0, 1+\tilde{D}(x_r))\right]$$

$$\tilde{D}(x_r) = C(x_r) - \mathbb{E}_{x_f\sim\mathbb{Q}}C(x_f)$$
$$\tilde{D}(x_f) = C(x_f) - \mathbb{E}_{x_r\sim\mathbb{P}}C(x_r)$$

WGAN-GP

$$L_D^{WGAN-GP} = -\mathbb{E}_{x_r\sim\mathbb{P}}\left[C(x_r)\right] + \mathbb{E}_{x_f\sim\mathbb{Q}}\left[C(x_f)\right] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(||\nabla_{\hat{x}}C(\hat{x})||_2 - 1)^2\right]$$

$$L_G^{WGAN-GP} = -\mathbb{E}_{x_f\sim\mathbb{Q}}\left[C(x_f)\right]$$

$\mathbb{P}_{\hat{x}}$ is the distribution of $\hat{x} = \epsilon x_r + (1-\epsilon)x_f$, where $x_r \sim \mathbb{P}$, $x_f \sim \mathbb{Q}$, $\epsilon \sim U[0,1]$.

RSGAN-GP

$$L_D^{RSGAN} = -\mathbb{E}_{(x_r,x_f)\sim(\mathbb{P},\mathbb{Q})}\left[\log(\text{sigmoid}(C(x_r)-C(x_f)))\right] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(||\nabla_{\hat{x}}C(\hat{x})||_2 - 1)^2\right]$$

$$L_G^{RSGAN} = -\mathbb{E}_{(x_r,x_f)\sim(\mathbb{P},\mathbb{Q})}\left[\log(\text{sigmoid}(C(x_f)-C(x_r)))\right]$$

$\mathbb{P}_{\hat{x}}$ is the distribution of $\hat{x} = \epsilon x_r + (1-\epsilon)x_f$, where $x_r \sim \mathbb{P}$, $x_f \sim \mathbb{Q}$, $\epsilon \sim U[0,1]$.

RaSGAN-GP

$$L_D^{RaSGAN} = -\mathbb{E}_{x_r\sim\mathbb{P}}\left[\log\left(\tilde{D}(x_r)\right)\right] - \mathbb{E}_{x_f\sim\mathbb{Q}}\left[\log\left(1-\tilde{D}(x_f)\right)\right] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(||\nabla_{\hat{x}}C(\hat{x})||_2 - 1)^2\right]$$

$$L_G^{RaSGAN} = -\mathbb{E}_{x_f\sim\mathbb{Q}}\left[\log\left(\tilde{D}(x_f)\right)\right] - \mathbb{E}_{x_r\sim\mathbb{P}}\left[\log\left(1-\tilde{D}(x_r)\right)\right]$$

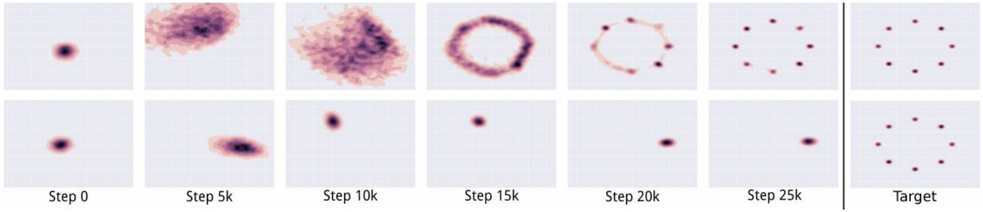
$$\tilde{D}(x_r) = \text{sigmoid}\left(C(x_r) - \mathbb{E}_{x_f\sim\mathbb{Q}}C(x_f)\right)$$
$$\tilde{D}(x_f) = \text{sigmoid}\left(C(x_f) - \mathbb{E}_{x_r\sim\mathbb{P}}C(x_r)\right)$$

$\mathbb{P}_{\hat{x}}$ is the distribution of $\hat{x} = \epsilon x_r + (1-\epsilon)x_f$, where $x_r \sim \mathbb{P}$, $x_f \sim \mathbb{Q}$, $\epsilon \sim U[0,1]$.

Challenges in GANs

- Mode collapse

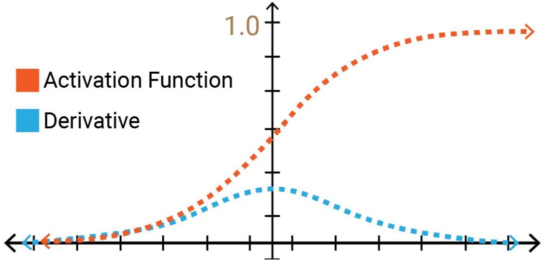
  Generator produces samples with a limited set of modes



Source

  ✔ Wasserstein loss
  ✔ Unrolled and packing

- Vanishing gradient



  ✔ Gradient Penalty
  ✔ Spectral Normalization

- Convergence failure
  ✔ Adding noise to discriminator inputs
  ✔ Penalizing discriminator weights
  ✔ Relativistic metrics

- Result evaluation
  ✔ Inception Score
  ✔ Fréchet Inception Distance (FID, MiFID)

Survey

bit.ly/**survey_gan**