

Learning Deep Learning with PyTorch

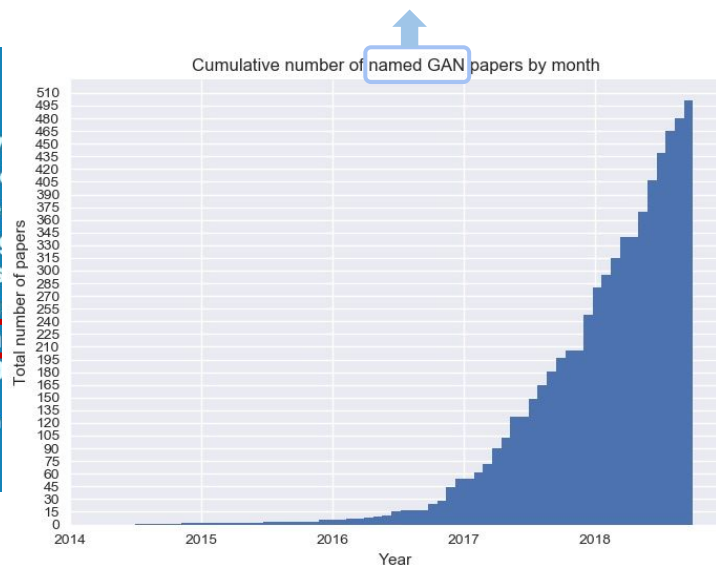
(7) GANs

Qiyang Hu
UCLA IDRE
Nov 3rd, 2020

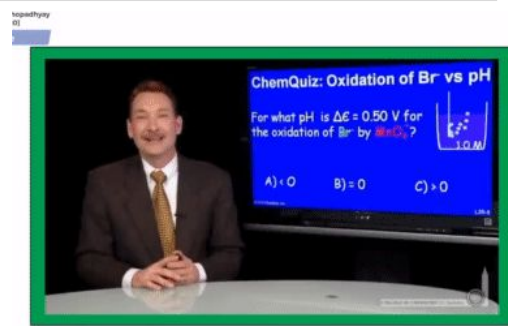
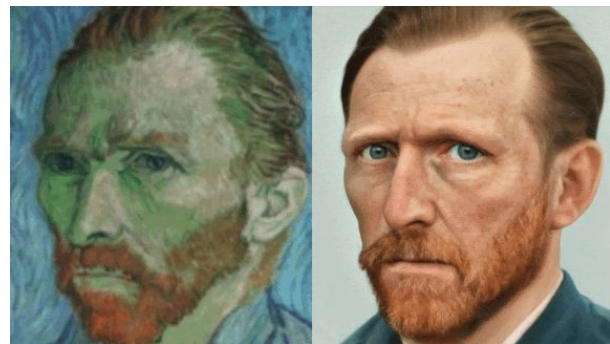
Generative Adversarial Networks



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



The only limit of GAN is our imagination



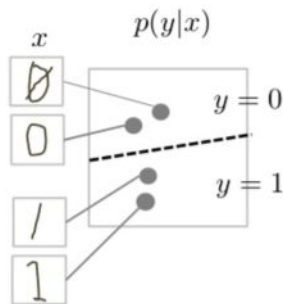
At each signal loss, the speech is being generated from lip movements of the speaker

GAN: Generative

Naive Bayes, LDA, GMM,
RBM, DBN, VAE, **GAN**

- Discriminative model

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

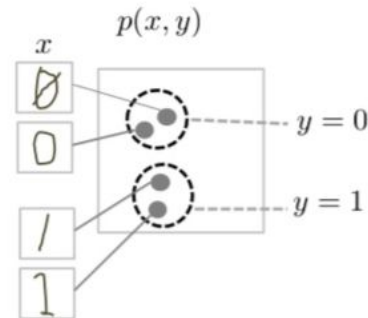


- capture the conditional prob. $p(Y | X)$
- measure the misfit of points
- learn the difference, ignore correlations

VS

- Generative model

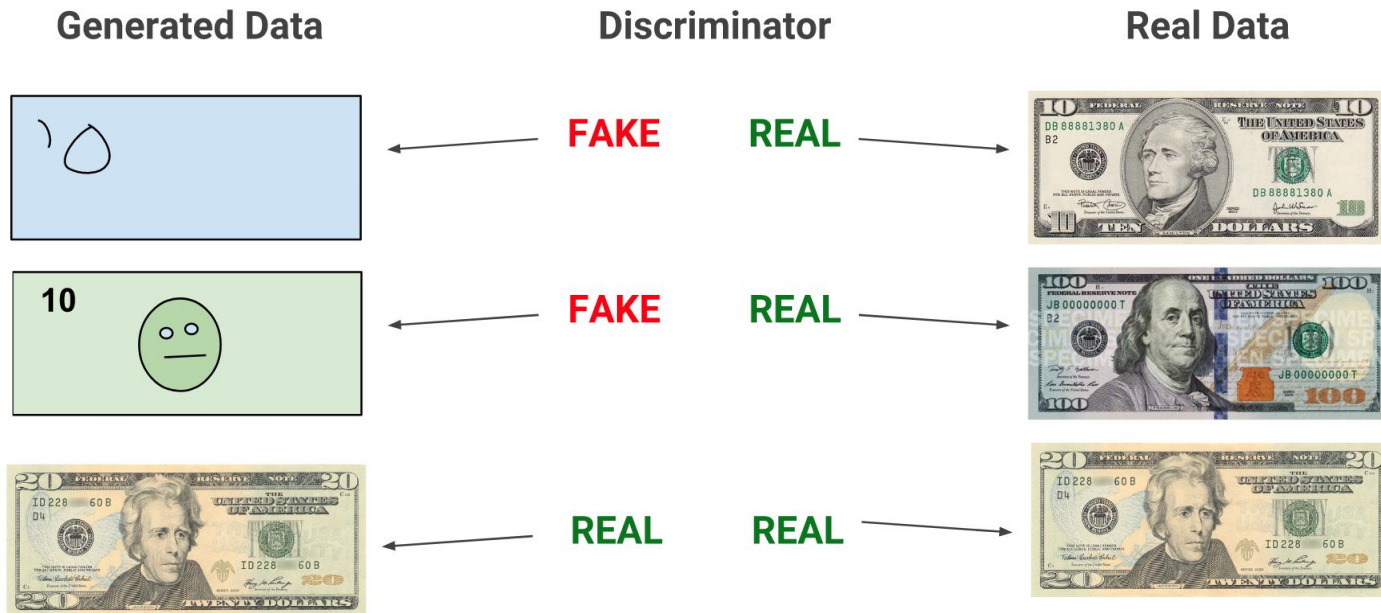
- generate new data instances
- try to model how data is placed



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

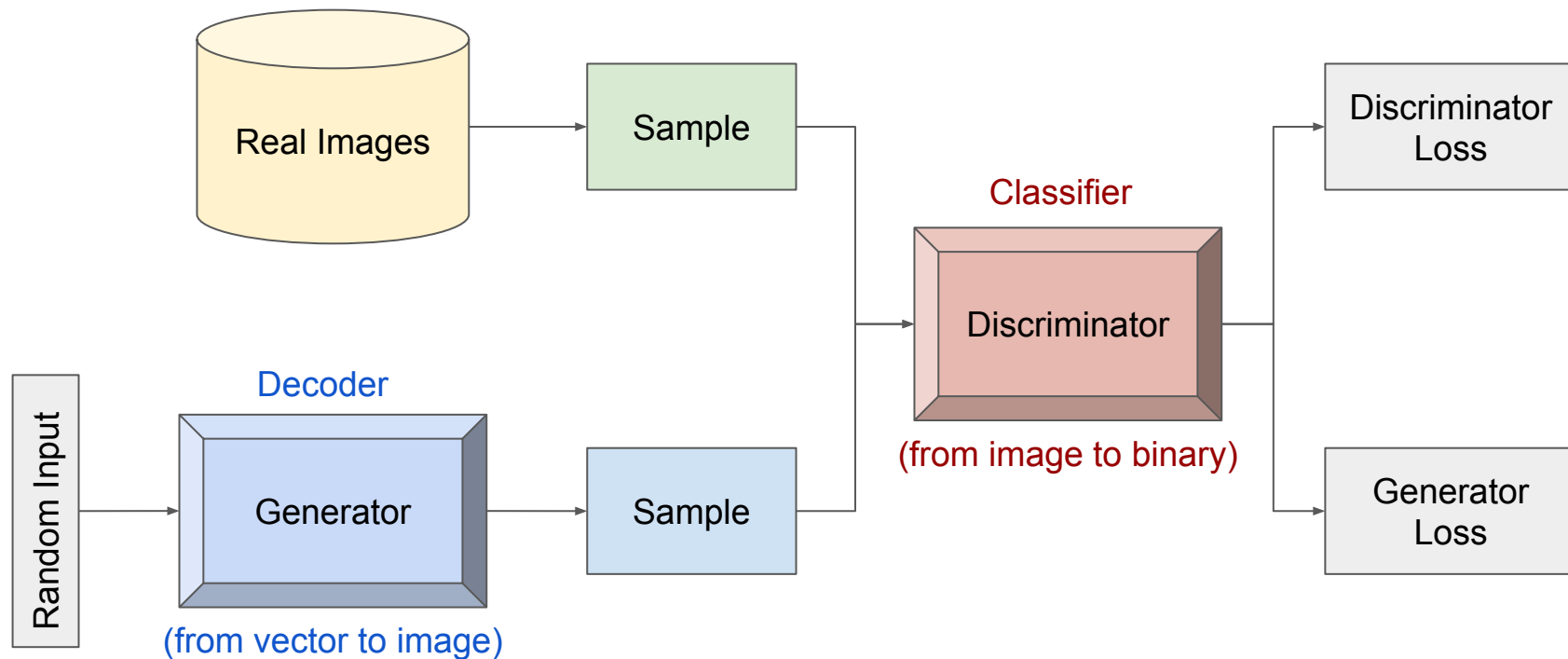
GAN: Adversarial

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

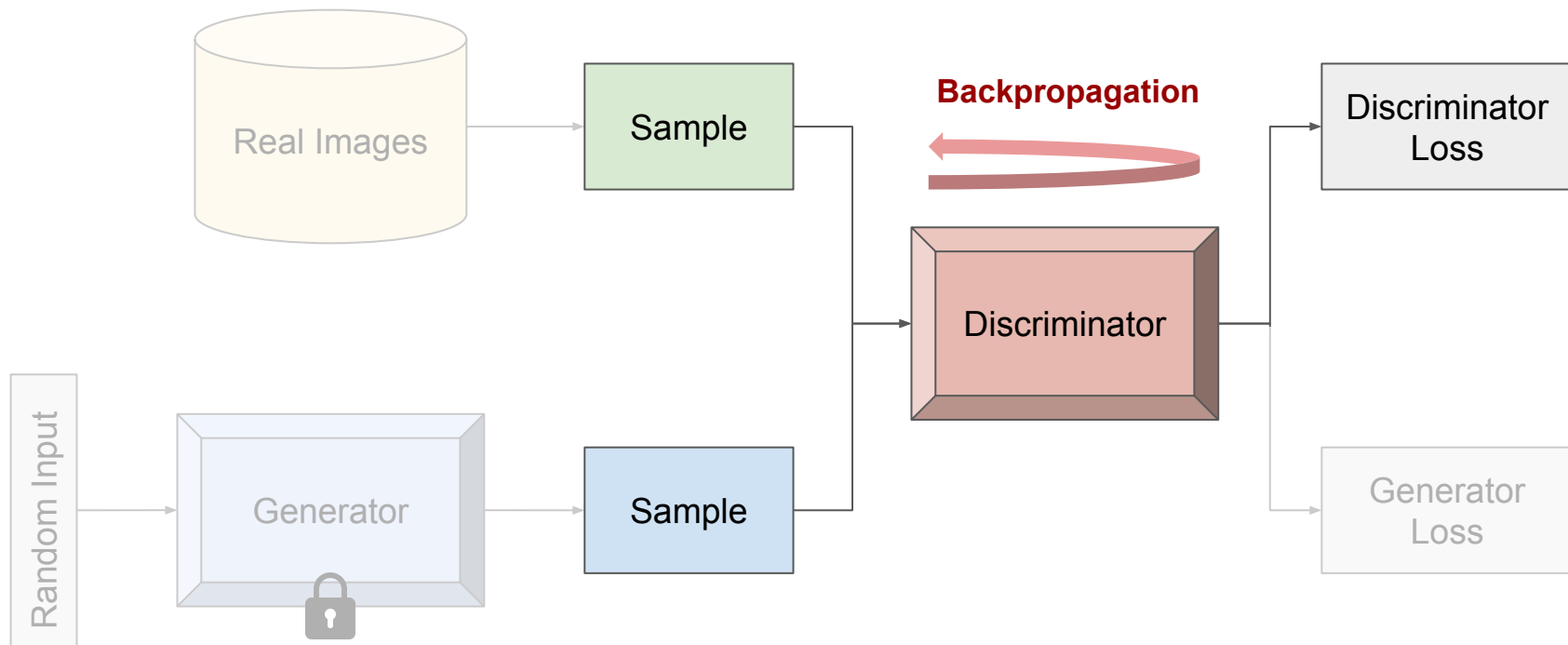


[Image Source](#)

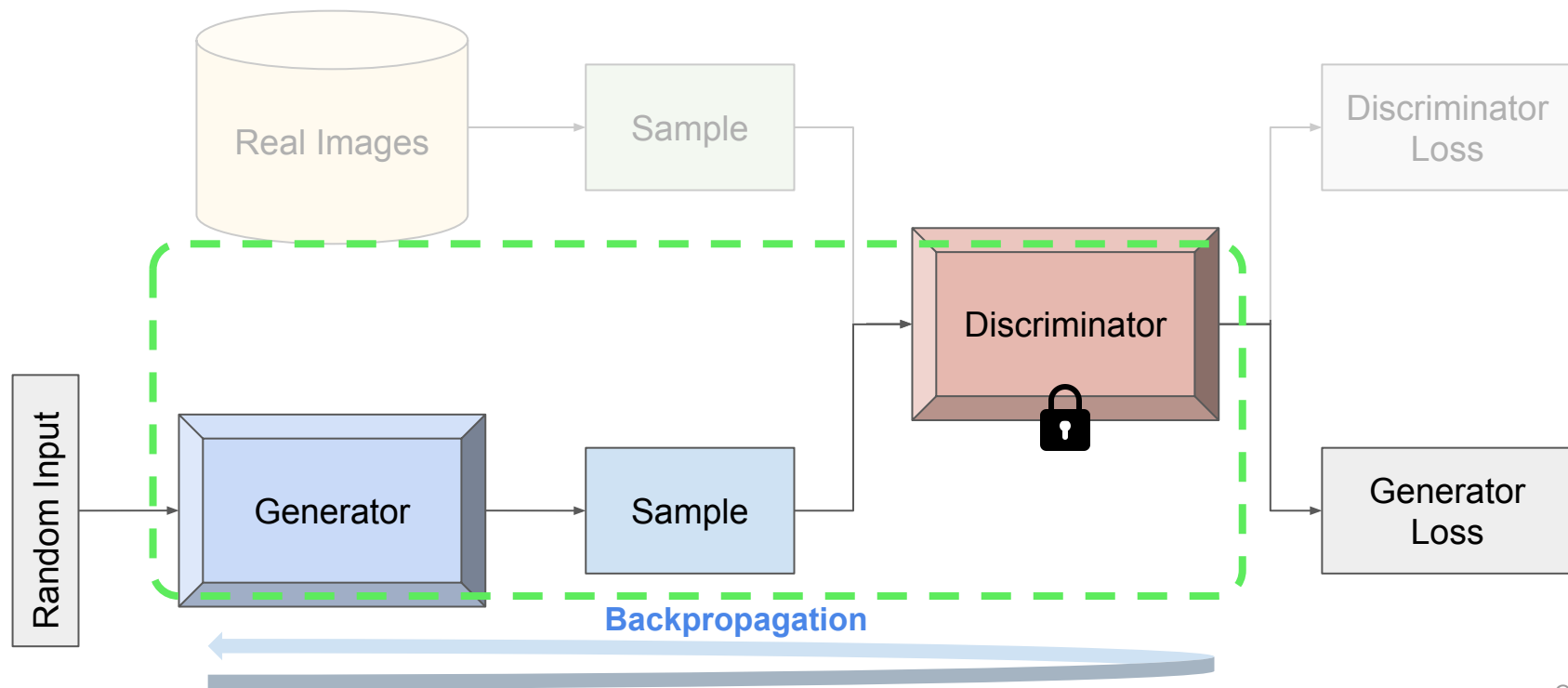
GAN: Network



Training of GAN (1): update discriminator



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 step a and b
- When to stop
 - While generator improves, discriminator performance gets worse
 - Generator becomes perfect \Rightarrow 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 are 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 are 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](#)

The diagram illustrates the minimax loss function for a Generative Adversarial Network (GAN). The equation is
$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} (\log D(x)) + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$
 Annotations include: 'Real image sample' (green box) pointing to x ; 'Noise sample' (grey box) pointing to z ; 'Fake image sample' (blue box) pointing to $G(z)$; 'Probability of the real image is real' (yellow box) pointing to $D(x)$; and 'Probability of the fake image is real' (orange box) pointing to $D(G(z))$. The \min_G and \max_D terms are highlighted in blue and orange boxes respectively.

- 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 \Rightarrow 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 \Rightarrow 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 $\sim \text{Uniform}(0.7, 1.2)$
- Fake $\sim \text{Uniform}(0.0, 0.3)$

As homework



Use DCGAN or Hybrid

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

Implemented in Demo: Yes

ADAM

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

Today's Demo — Generative Dog Images from Kaggle

- Experiment with creating puppy pics
 - A Kernels-only competition (total 10K prize, expired a year 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 $< \frac{1}{5}$ of the picture
 - With various texts (from memes, magazine, etc)
 - Even wild predators included

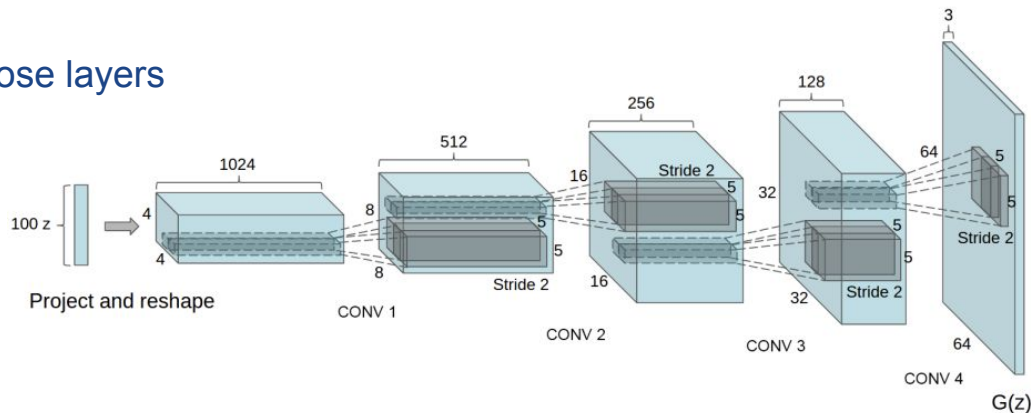


Neural Networks for Generator and Discriminator

- 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 $\neq 1$ to make
the discriminator not
learn too quickly

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

```
#####
# (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_05

A lot of different GANs!

- Various design of network structures

- Conditional GANs
- CycleGANs
- ProGANs
- InfoGANs
- EB-GANs
- VAE-GANs
- BiGANs
- Triple-GANs
- ...

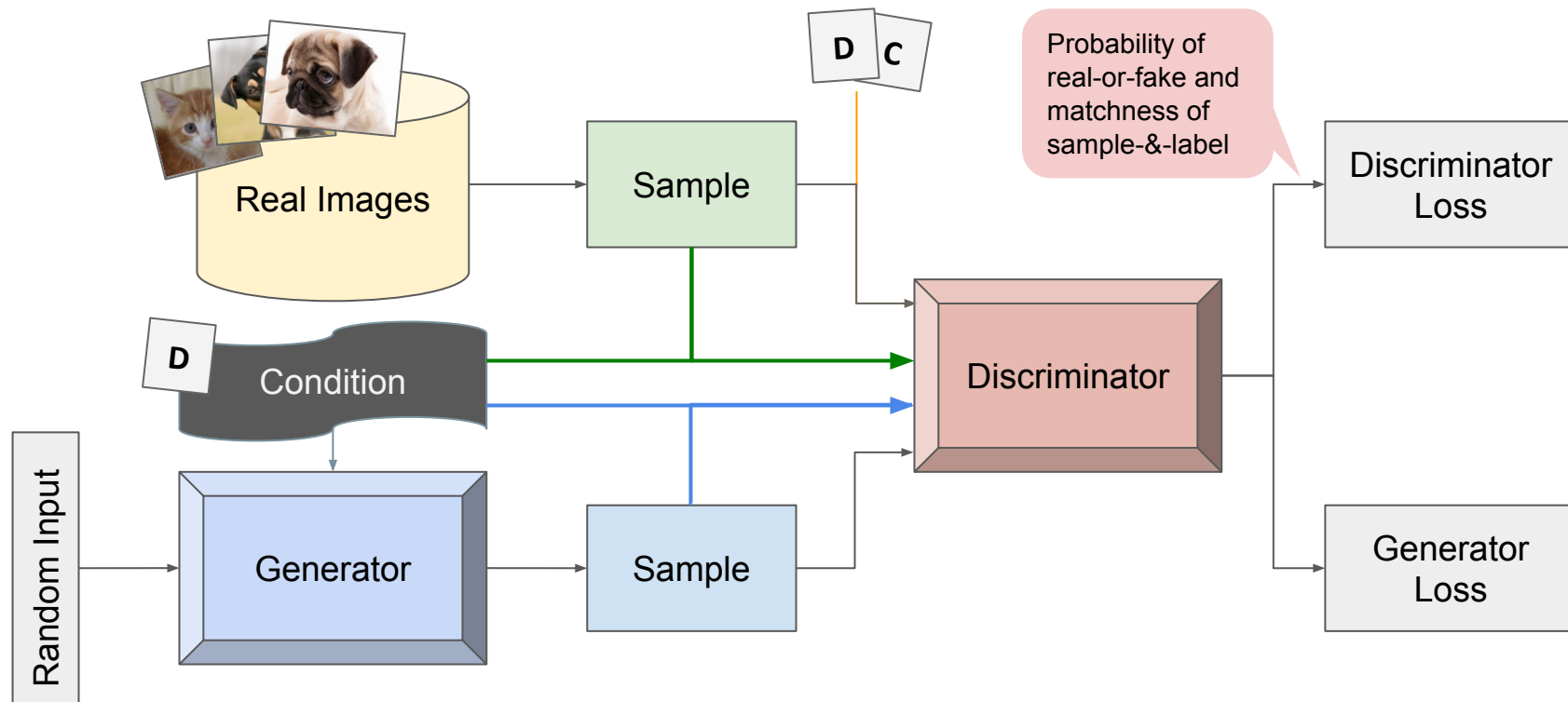
- Various metrics for objective functions

- LSGANs
- WGANs
- RGANs
- Cramer GANs
- Fisher GANs
- MMD GANs
- McGANs
- HingeGANs
- ...

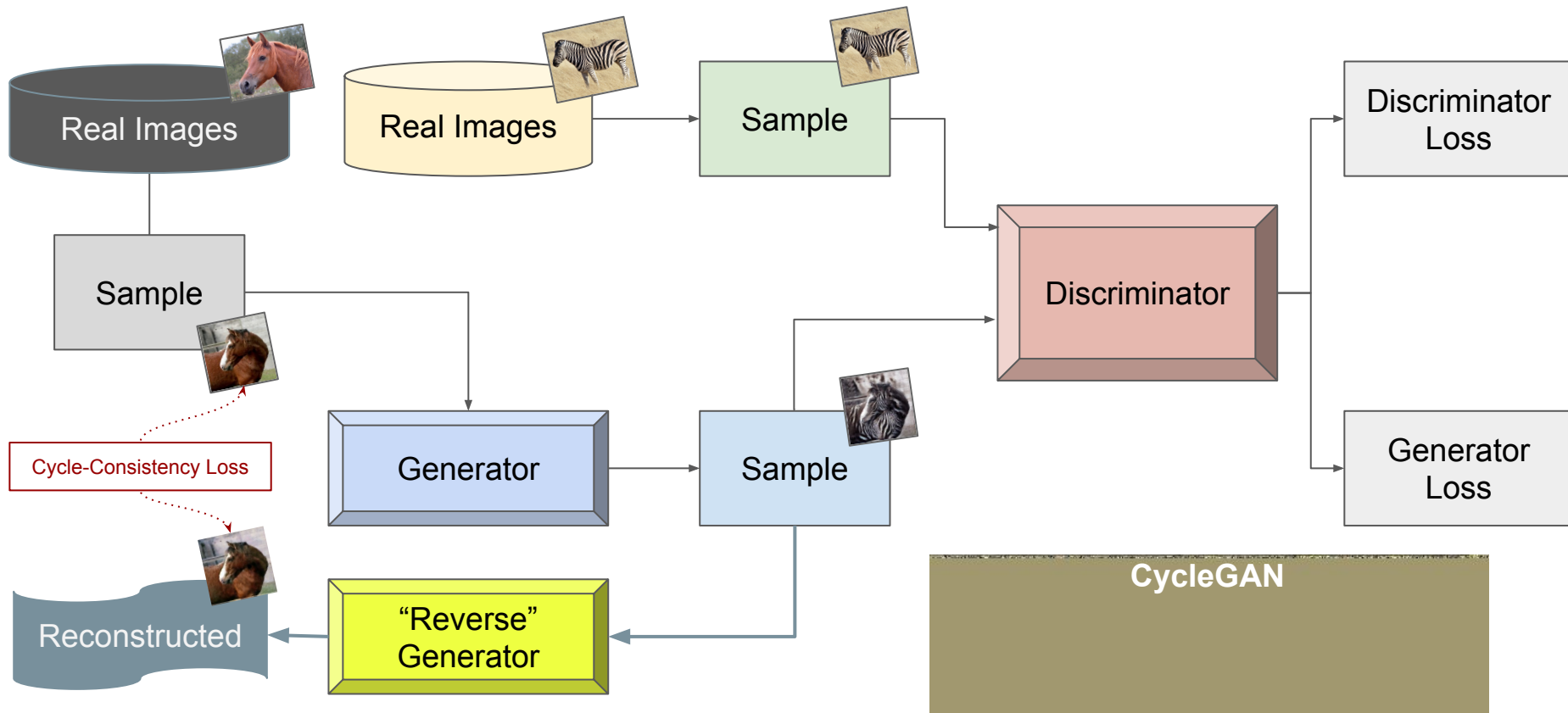
- Combining the two

- BEGAN
- MAGANs
- ...

Conditional GAN: generate images with specific class



CycleGAN: *unsupervised* conditional GAN



Some GAN loss function variations

SGAN (non-saturating)

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

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

RSGAN

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

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

RaSGAN

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

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

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

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

LSGAN

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

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

RaLSGAN

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

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

HingeGAN

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

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

RaHingeGAN

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

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

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

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

WGAN-GP

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

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

$\mathbb{P}_{\hat{x}}$ 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

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

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

$\mathbb{P}_{\hat{x}}$ 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

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

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

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

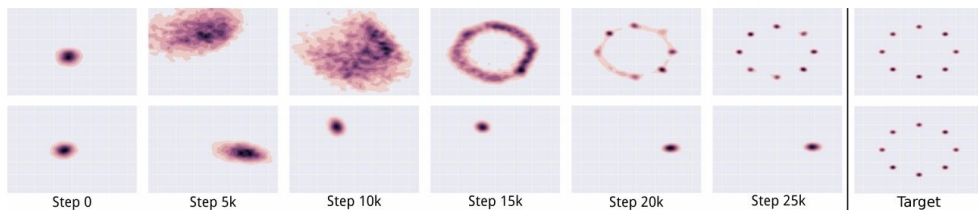
$$\tilde{D}(x_f) = \text{sigmoid}(C(x_f) - \mathbb{E}_{x_r \sim \mathbb{P}} C(x_r))$$

$\mathbb{P}_{\hat{x}}$ 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



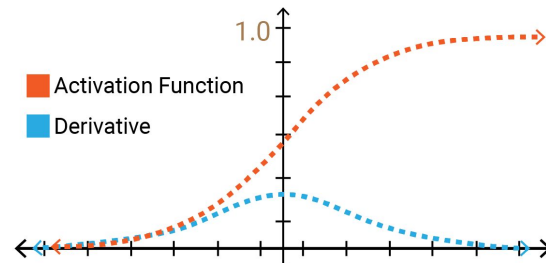
[Source](#)

- ✓ Wasserstein loss
- ✓ Unrolled and packing

- Convergence failure

- ✓ Adding noise to discriminator inputs
- ✓ Penalizing discriminator weights
- ✓ Relativistic metrics

- Vanishing gradient



- ✓ Gradient Penalty
- ✓ Spectral Normalization

- Result evaluation

- ✓ Inception Score
- ✓ Fréchet Inception Distance (FID, MiFID)

