Generative Adversarial Networks

Conditional GAN and Image translation

Daniel Yukimura

yukimura@impa.br

October 17, 2018

Generative Adversarial Networks (GANs)

Review:

We have two neural networks competing

- **Generator**: $G : \mathcal{H} \to \mathcal{X}$.
- **Discriminator**: $D : \mathcal{X} \to [0, 1]$

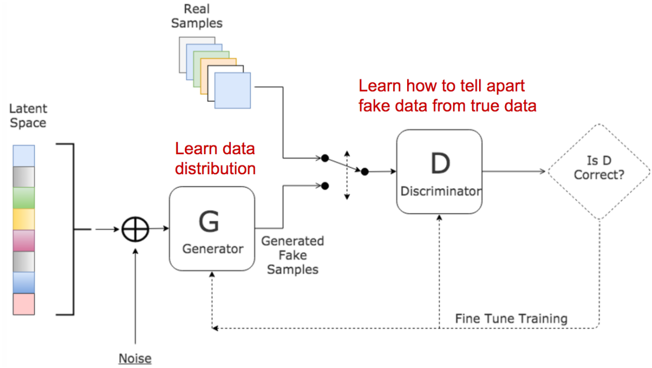
We want to find the parameters that reach equilibrium for the minmax game

$$G^* = \operatorname*{argmin}_G \max_D \frac{1}{2} \left(\mathbb{E}[\log D(X)] + \mathbb{E}[\log (1 - D(X_G))] \right). \tag{1}$$

where $X_G = G(H)$ are the "fake" samples, given by the distribution induced by G from p_H.

Generative Adversarial Networks (GANs)

Review:



Conditional GANs

The majority of real applications involving generative models requires some sort of **conditioning**, such as

- Segmentation,
- "in-painting",
- Next frame prediction,
- Style Transfer.

Example: Our images have a label associated

$$(X, Y) \sim p_{data} \tag{2}$$

We learn to sample from $p(X \mid Y)$.

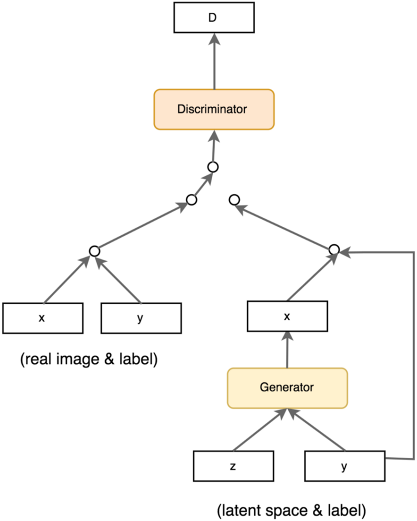
Conditional GANs

The **Conditional GAN (CGAN)** approach proposed by *Mirza and Osidero* (2014), consists on allowing both networks, G and D, to directly carry the extra information.

- We consider a condition as an event coming from a related random variable $E = [Y = y]$, $y \in \mathcal{Y}$, where y could represent
  - class label
  - encoded text sentence
  - matrix of pixels
  - other...
- Adversarial value function:

$$v(G, D) = \mathbb{E}_{(X,Y)\sim p_{data}}[\log D(X|Y)] + \mathbb{E}_{\substack{Y\sim p_Y \\ H\sim p_H}}[\log(1 - D(G(H|Y)|Y))]. \tag{3}$$

Conditional GANs



Conditioanl GANs

Example: MNIST

In this example we want to generate handwriting digits conditioned on the class they belong $y \in \{0, 1, \ldots, 9\}$:

```
class cond_Generator(nn.Module):
    def __init__(self, l_dim=128, z_dim=100, y_dim=10):
        super(cond_Generator, self).__init__()
        self.layer1_z = nn.Sequential(## 1x1 to 4x4
            nn.ConvTranspose2d(z_dim, l_dim*2, 4, 1, 0),
            nn.BatchNorm2d(l_dim*2),
            nn.ReLU())
...
```

Conditional GANs

```
...
        self.layer1_y = nn.Sequential(
            nn.ConvTranspose2d(y_dim, l_dim*2, 4, 1, 0),
            nn.BatchNorm2d(l_dim*2),
            nn.ReLU())
        self.layer2 = nn.Sequential(## 4x4 to 7x7
            nn.ConvTranspose2d(l_dim*4, l_dim*2, 3, 2, 1),
            nn.BatchNorm2d(l_dim*2),
            nn.ReLU())
...
```

Conditional GANs

```
...
        self.layer3 = nn.Sequential(## 7x7 to 14x14
            nn.ConvTranspose2d(l_dim*2, l_dim, 4, 2, 1),
            nn.BatchNorm2d(l_dim),
            nn.ReLU())
        self.layer4 = nn.Sequential(## 14x14 to 28x28
            nn.ConvTranspose2d(l_dim, 1, 4, 2, 1),
            nn.Tanh())

    def weight_init(self, mean, std):
        for m in self._modules:
            normal_init(self._modules[m], mean, std)
...
```

Conditional GANs

```
...
    def forward(self, z, y):
        z = self.layer1_z(z)
        y = self.layer1_y(y)
        out = torch.cat([z,y],1)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        return out
```

Conditional GANs

```
class cond_Discriminator(nn.Module):
    def __init__(self, l_dim=128, y_dim=10):
        super(cond_Discriminator, self).__init__()
        self.layer1_x = nn.Sequential(## 28x28 to 14x14
            nn.Conv2d(1, int(l_dim/2), 4, 2, 1),
            nn.LeakyReLU(0.2))
        self.layer1_y = nn.Sequential(
            nn.Conv2d(y_dim, int(l_dim/2), 4, 2, 1),
            nn.LeakyReLU(0.2))
```

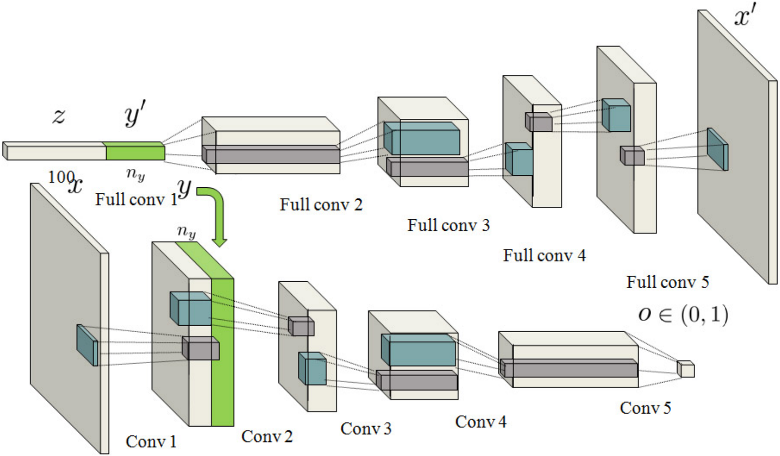
Conditional GANs

```
self.layer2 = nn.Sequential(## 14x14 to 7x7
    nn.Conv2d(l_dim, l_dim*2, 4, 2, 1),
    nn.BatchNorm2d(l_dim*2),
    nn.LeakyReLU(0.2)
)
self.layer3 = nn.Sequential(## 7x7 to 4x4
    nn.Conv2d(l_dim*2, l_dim*4, 3, 2, 1),
    nn.BatchNorm2d(l_dim*4),
    nn.LeakyReLU(0.2))
self.layer4 = nn.Sequential(
    nn.Conv2d(l_dim*4, 1, 4),
    nn.Sigmoid())
```

Conditional GANs

```
def weight_init(self, mean, std):
    for m in self._modules:
        normal_init(self._modules[m], mean, std)
def forward(self, x, y):
    x = self.layer1_x(x)
    y = self.layer1_y(y)
    out = torch.cat([x,y],1)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    return out
```

Conditional GANs



Conditional GANs

```
for epoch in range(n_epochs):
    for idx, (img_batch, y_batch) in enumerate(train_loader):
        # Training Discriminator
        x = to_cuda(img_batch)
        y = to_cuda(onehot_fill[y_batch])
        x_disc = D(x,y)
        D_x_loss = criterion(x_disc, D_labels)

        z = to_cuda(torch.randn(mbatch_size, z_dim).view(-1,100,1,1))
        y_rd = (torch.rand(mbatch_size,1)*10).type(torch.LongTensor).squeez
        y_label = to_cuda(onehot_encoder[y_rd])
        y_fill = to_cuda(onehot_fill[y_rd])
...
```

Conditional GANs

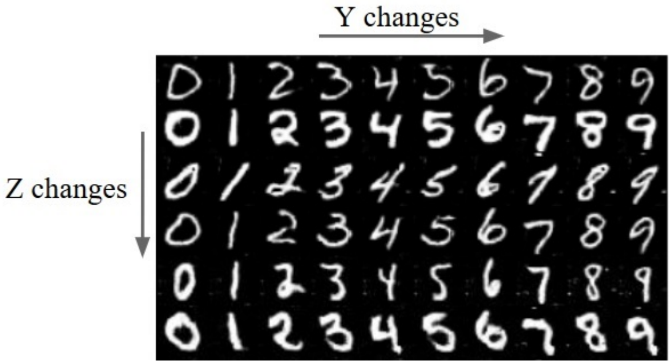
```
...
    z_disc = D(G(z,y_label),y_fill)
    D_z_loss = criterion(z_disc, D_fakes).squeeze()
    D_loss = D_x_loss + D_z_loss
    D.zero_grad()
    D_loss.backward()
    D_opt.step()
...
```

Conditional GANs

```
...
        # Training Generator
        z = to_cuda(torch.randn(mbatch_size, z_dim).view(-1,100,1,1))
        y_rd = (torch.rand(mbatch_size,1)*10).type(torch.LongTensor).squeez
        y_label = to_cuda(onehot_encoder[y_rd])
        y_fill = to_cuda(onehot_fill[y_rd])
        z_disc = D(G(z,y_label),y_fill)
        G_loss = criterion(z_disc, D_labels)

        D.zero_grad()
        G.zero_grad()
        G_loss.backward()
        G_opt.step()
...
```

Conditional GANs



Latent Space Optimization

An alternative for conditioning GANs

- First train G and D as usual, and define
- **Contextual Loss:**

$$\mathcal{L}_{contextual}(z) = \|M \odot G(z) - M \odot y\|_1 \tag{4}$$

- **Perceptual Loss:**

$$\mathcal{L}_{perceptual}(z) = log(1 - D(G(z))) \tag{5}$$

Yeh et al. 2016

Latent Space Optimization

An alternative for conditioning GANs

- First train G and D as usual, and define
- **Contextual Loss:**

$$\mathcal{L}_{contextual}(z) = \|M \odot G(z) - M \odot y\|_1 \quad (4)$$

- **Perceptual Loss:**

$$\mathcal{L}_{perceptual}(z) = log(1 - D(G(z))) \quad (5)$$

Yeh et al. 2016

Latent Space Optimization

An alternative for conditioning GANs

- First train G and D as usual, and define
- **Contextual Loss:**

$$\mathcal{L}_{contextual}(z) = \|M \odot G(z) - M \odot y\|_1 \tag{4}$$

- **Perceptual Loss:**

$$\mathcal{L}_{perceptual}(z) = log(1 - D(G(z))) \tag{5}$$

Yeh et al. 2016

Latent Space Optimization

An alternative for conditioning GANs

- Find the best $z \in \mathcal{Z}$ that gives the best sample for the condition.

$$\hat{z} = \operatorname*{argmin}_{z} \left(\mathcal{L}_{contextual}(z) + \mathcal{L}_{perceptual}(z)\right) \tag{6}$$

- Then reconstruct

$$x_{reconstructed} = M \odot y + (M^{-1}) \odot G(\hat{z}) \tag{7}$$

Yeh et al. 2016

Latent Space Optimization

An alternative for conditioning GANs

- Find the best $z \in \mathcal{Z}$ that gives the best sample for the condition.

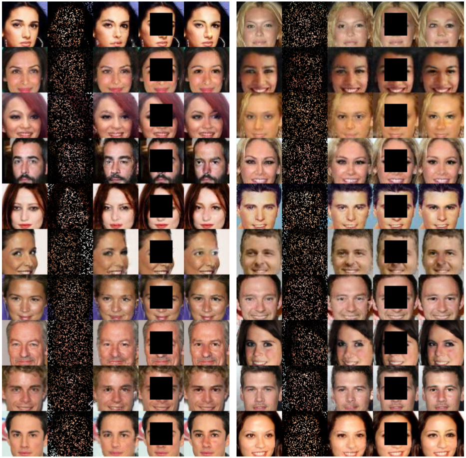
$$\hat{z} = \underset{z}{\operatorname{argmin}} \left(\mathcal{L}_{contextual}(z) + \mathcal{L}_{perceptual}(z)\right) \tag{6}$$

- Then reconstruct

$$x_{reconstructed} = M \odot y + (M^{-1}) \odot G(\hat{z}) \tag{7}$$

Yeh et al. 2016

Latent Space Optimization



Yeh et al. 2016

Formulation

- We want to learn mapping functions from two domains $\mathcal{X}$ and $\mathcal{Y}$ given training set of images.

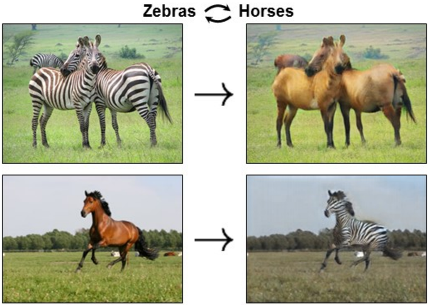


Image-to-Image translation

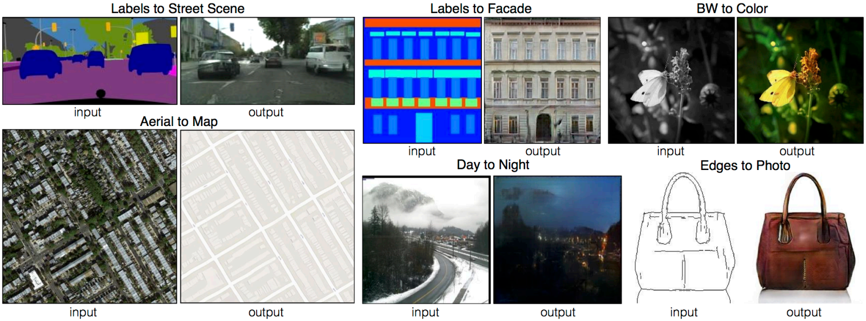
Formulation

- We want to learn mapping functions from two domains $\mathcal{X}$ and $\mathcal{Y}$ given training set of images.
- $\{x_i\}_{i=1}^{N}$ from $\mathcal{X}$ and
- $\{y_j\}_{j=1}^{M}$ from $\mathcal{Y}$.
- Now we want mappings between distributions $X \sim p_X$ in $\mathcal{X}$ and a distribution $Y \sim p_Y$ in $\mathcal{Y}$.

Image-to-Image translation

Pixel-to-Pixel

Conditioning GANs similarly to the original form:



Isola et al. 2016

Image-to-Image translation

Pixel-to-Pixel

Using conditional GANs:

- Train, as usual, a conditional GAN to approximate $y \approx G(z|x)$.
- Since this is a very complex condition, sometimes we pair it with a regularized loss (a pixel-wise condition)

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{\substack{x,y \sim p_{data} \\ z \sim p_Z}} ||y - G(z|x)|| \tag{8}$$

building the minmax game

$$G^* = \underset{G}{\operatorname{argmin}} \max_{D} \mathcal{L}_{CGAN}(G, D) + \mathcal{L}_{L1}(G). \tag{9}$$

Isola et al. 2016

Image-to-Image translation

Pixel-to-Pixel

Using conditional GANs:

- Train, as usual, a conditional GAN to approximate $y \approx G(z|x)$.
- Since this is a very complex condition, sometimes we pair it with a regularized loss (a pixel-wise condition)

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{\substack{x,y \sim p_{data} \\ z \sim p_Z}} \|y - G(z|x)\| \tag{8}$$

building the minmax game

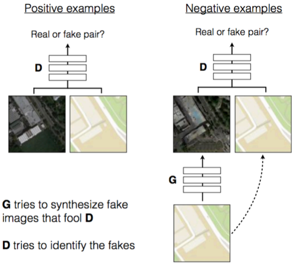
$$G^* = \arg\min_G \max_D \mathcal{L}_{CGAN}(G, D) + \mathcal{L}_{L1}(G). \tag{9}$$

Isola et al. 2016

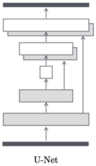
Image-to-Image translation

From the paper of *Isola et al. 2016*:

- The result is improved by **adding skip connections**, to the generator, from layer i to $L - i$, this network is known as $U - net$.



Discriminator

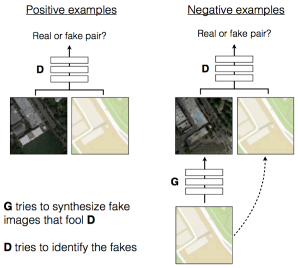


Generator

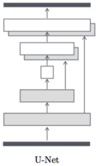
Image-to-Image translation

From the paper of *Isola et al. 2016*:

- Randomness is added to the process by using **dropout** instead of adding a sample $z \sim p_Z$.



Discriminator

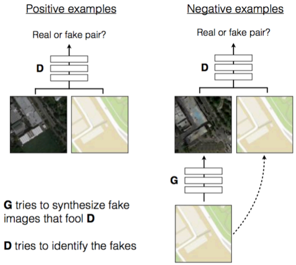


Generator

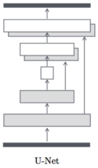
Image-to-Image translation

From the paper of *Isola et al. 2016*:

- The discriminator has the **PatchGAN** architecture, the output is a pixel matrix in $[0,1]^{N\times N}$ representing how believable each corresponding patch is.



Discriminator



Generator

Image-to-Image translation

CycleGAN

Unpaired Image-to-Image translation (Zhu et al. 2017)

Image-to-Image translation

CycleGAN

Unpaired Image-to-Image translation (Zhu et al. 2017)

- In this approach we build two maps

$$G : \mathcal{X} \to \mathcal{Y} \text{ and } F : \mathcal{Y} \to \mathcal{X} \tag{10}$$

  i.e. we want a map from distribution p_X to p_Y and also an inverse one.

- We also end up with two distinct discriminators

$$D_X : \mathcal{X} \to [0, 1] \text{ and } D_Y : \mathcal{Y} \to [0, 1]. \tag{11}$$

  For $x \sim p_X$ and $y \sim p_Y$, D_X distinguish between x and $F(y)$ and D_Y between y and $G(x)$

Image-to-Image translation

CycleGAN

Unpaired Image-to-Image translation (Zhu et al. 2017)

- In this approach we build two maps

$$G : \mathcal{X} \to \mathcal{Y} \text{ and } F : \mathcal{Y} \to \mathcal{X} \tag{10}$$

  i.e. we want a map from distribution p_X to p_Y and also an inverse one.

- We also end up with two distinct discriminators

$$D_X : \mathcal{X} \to [0, 1] \text{ and } D_Y : \mathcal{Y} \to [0, 1]. \tag{11}$$

  For $x \sim p_X$ and $y \sim p_Y$, D_X distinguish between x and $F(y)$ and D_Y between y and $G(x)$

Image-to-Image translation

CycleGAN

Unpaired Image-to-Image translation (Zhu et al. 2017)

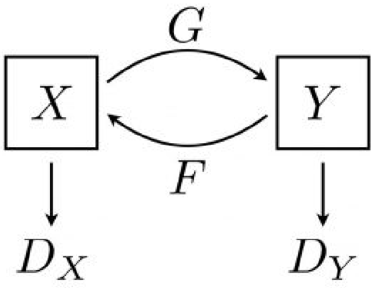


Image-to-Image translation

CycleGAN

- We end with a loss for G and other for F

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y\sim p_Y} \log D_Y(y) + \mathbb{E}_{x\sim p_X} \log\left(1 - D_Y(G(x))\right) \quad (12)$$

  The other is similar taking values as $\mathcal{L}_{GAN}(F, D_X, Y, X)$.

- Cycle consistency loss

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x\sim p_X} \|F(G(x)) - x\|_1 + \mathbb{E}_{y\sim p_y} \|G(F(y)) - y\|_1 \quad (13)$$

Image-to-Image translation

CycleGAN

- We end with a loss for G and other for F

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y\sim p_Y} \log D_Y(y) + \mathbb{E}_{x\sim p_X} \log\left(1 - D_Y(G(x))\right) \quad (12)$$

The other is similar taking values as $\mathcal{L}_{GAN}(F, D_X, Y, X)$.

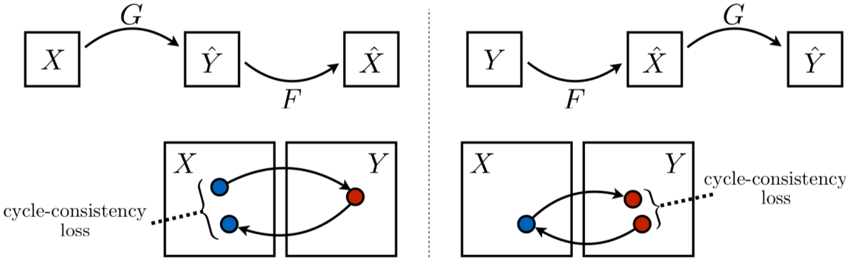
- **Cycle consistency loss**

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x\sim p_X} \|F(G(x)) - x\|_1 + \mathbb{E}_{y\sim p_y} \|G(F(y)) - y\|_1 \quad (13)$$

Image-to-Image translation

CycleGAN

- **Cycle consistency loss**



(b) (c)

CycleGAN

- Finally our full objective is

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) \\ + \mathcal{L}_{cyc}(G, F) \tag{14}$$

- Our minmax game is now given by

$$G^*, F^* = \underset{G,F}{\operatorname{argmin}} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y). \tag{15}$$

Image-to-Image translation

CycleGAN

- Finally our full objective is

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) \\ + \mathcal{L}_{cyc}(G, F)\end{aligned} \tag{14}$$

- Our minmax game is now given by

$$G^*, F^* = \operatorname*{argmin}_{G,F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y). \tag{15}$$

Image-to-Image translation

CycleGAN

Image-to-Image translation

CycleGAN

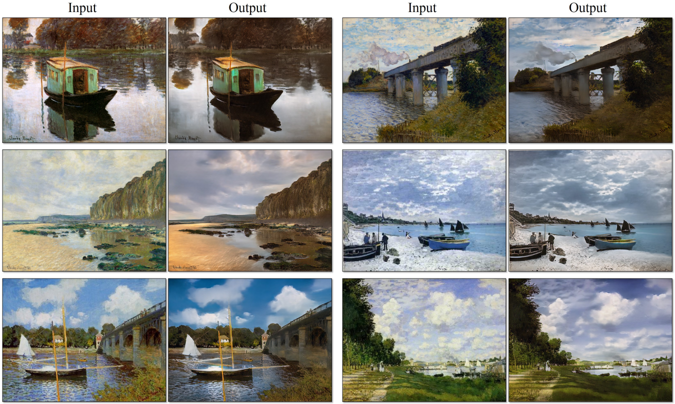


Image-to-Image translation

CycleGAN

Image-to-Image translation

Code in PyTorch:

- pix-to-pix and CycleGAN
- Pixel-to-Pixel HD
- Video-to-Video Synthesis
- CycleGAN - Colaboratory (tensorflow)