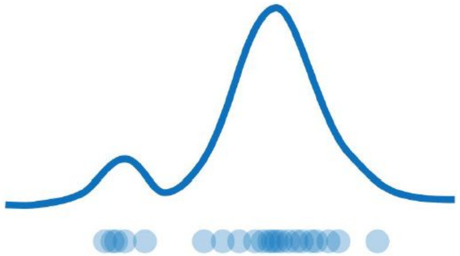
Conditional GANs

JAMAL TOUTOUH
jamal@lcc.uma.es
jamal.es
@jamtou

Review basic ideas about GANs

We want to learn a **generative model** without computing the **density estimation** function



GANs construct a generative model by raising an arms race between two neural networks, a **generator** and a **discriminator**

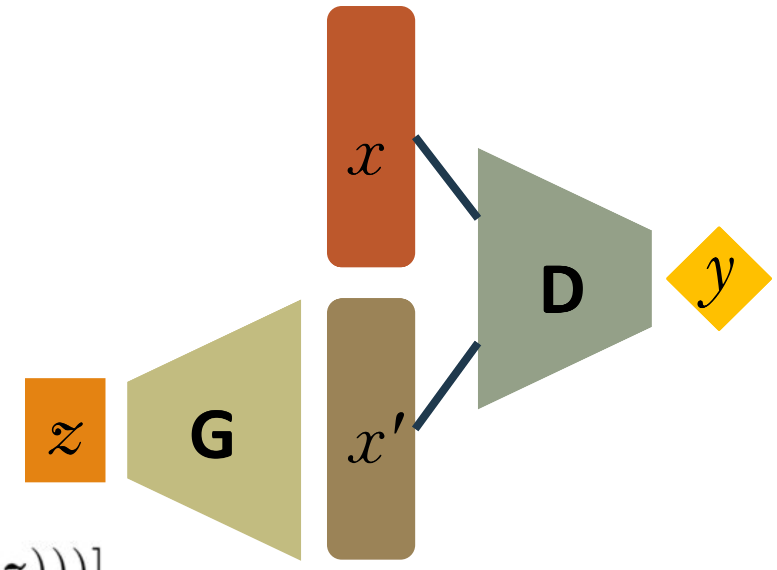
Training samples

Synthetic samples

Review basic ideas about GANs

GANs construct a generative model by raising an arms race between two neural networks, a **generator** and a **discriminator**

- Discriminator (**D**) tries to distinguish between real data (X) from the real data distribution and fake data (X') from the generator (**G**)
- Generator (**G**) learns how to create synthetic/fake data samples (X') by sampling random noise (Z) to fool the discriminator (**D**)



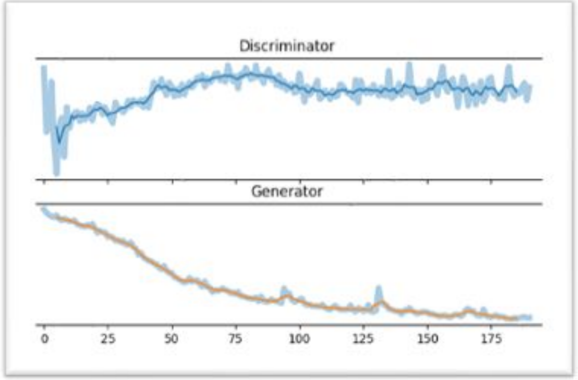
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

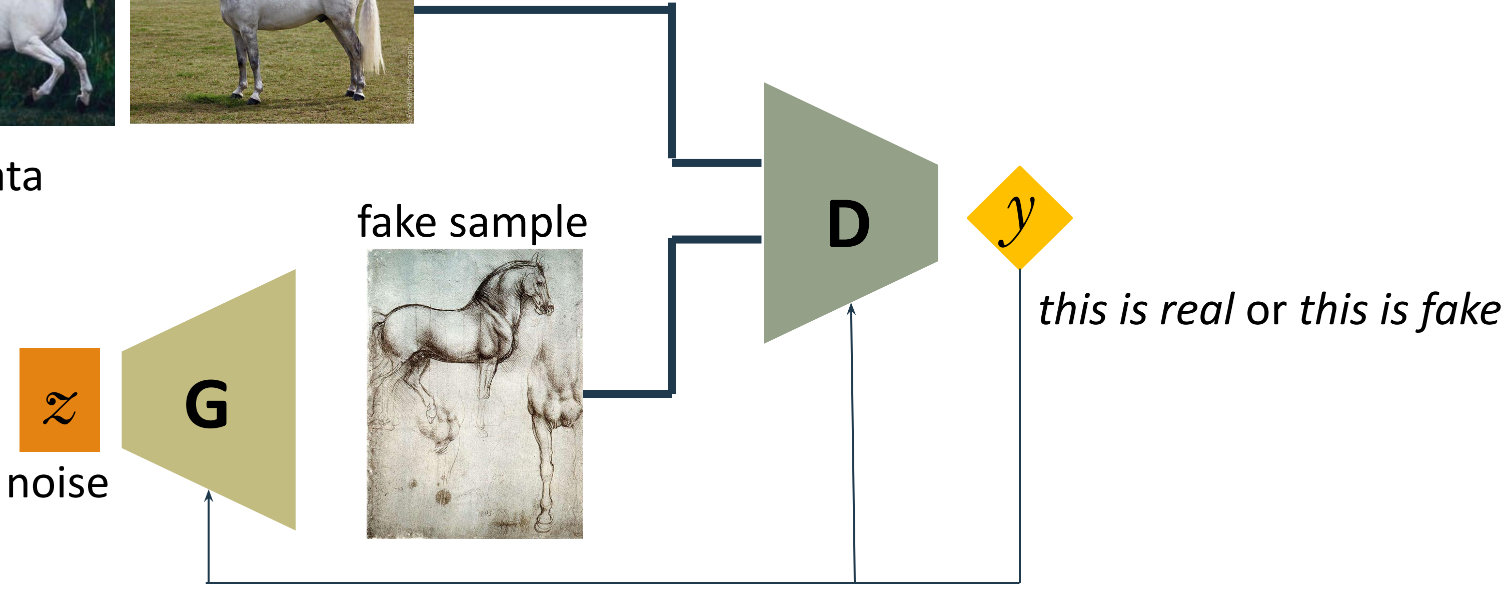
Review basic ideas about GANs

GANs construct a generative model by raising an arms race between a **generator** and a **discriminator**

Goodfellow et al. 2014. **Generative Adversarial Nets**

real data



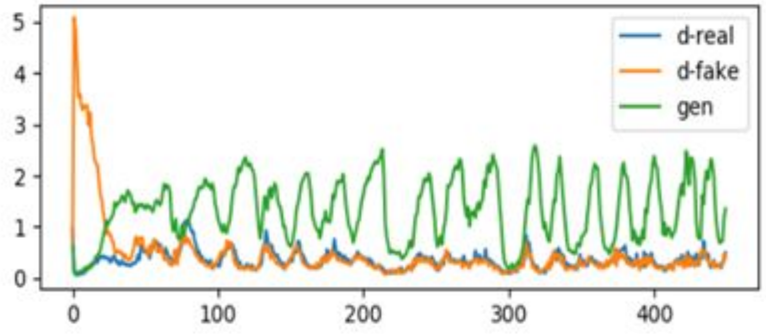


$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

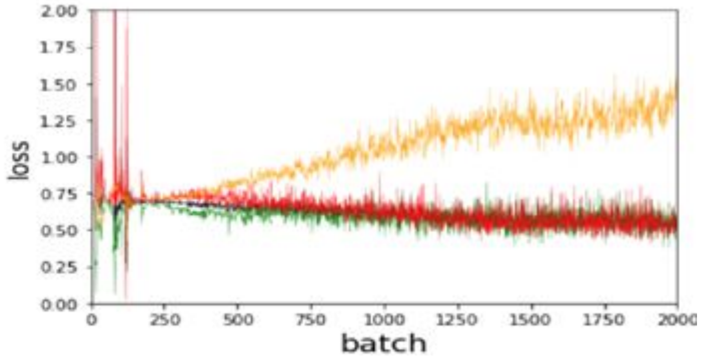
Review basic ideas about GANs

GAN Training Pathologies

- **Non-convergence:** the model parameters oscillate, destabilize, and never converge

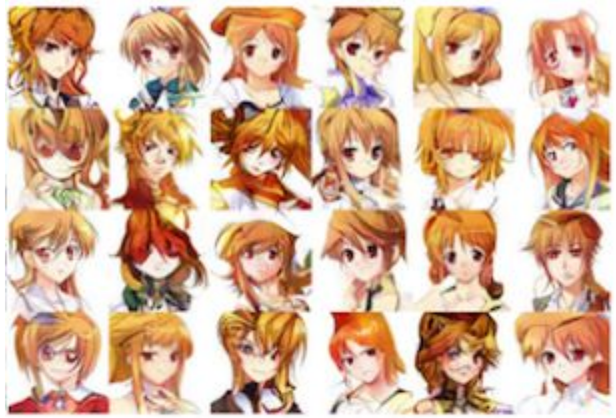


- **Mode collapse:** the generator collapses which produces limited varieties of samples



- **Diminished gradient:** the discriminator gets too successful that the generator gradient vanishes and learns nothing

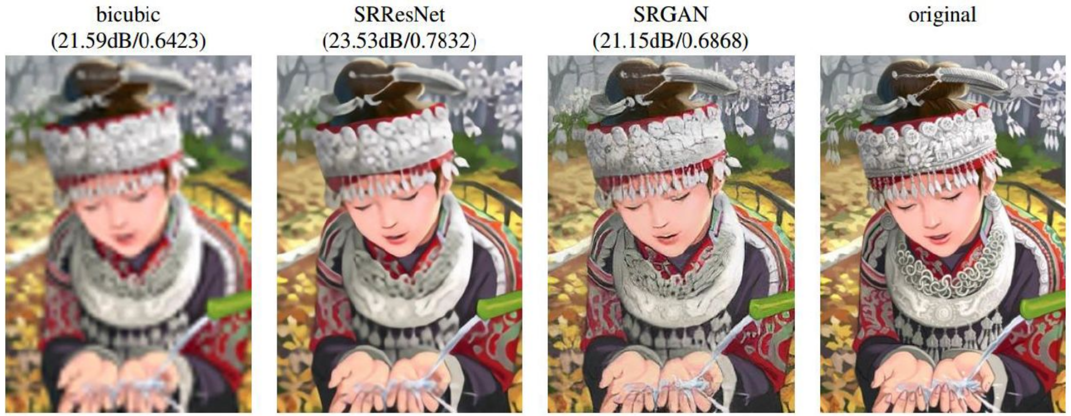
Review basic ideas about GANs

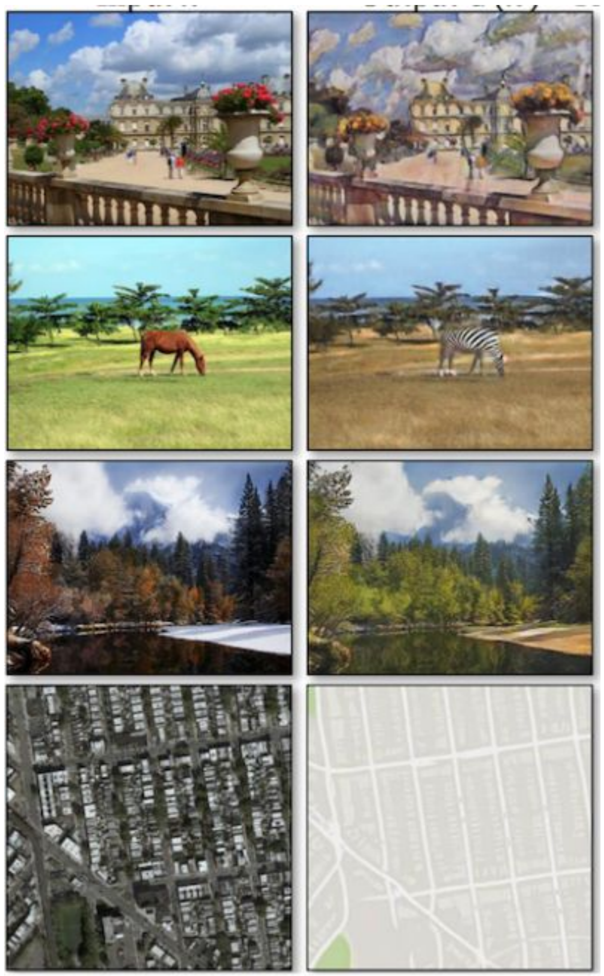


Review basic ideas about GANs

- GAN is trained in a completely unsupervised and unconditional fashion, meaning no labels involved in the training process.
- We have zero control over the type of samples generated.
- What if we want our GAN model to generate samples of given specific type (e.g., generate just the digit 9 in MNIST).
  - We could try to control the random vectors sampled from the latent space

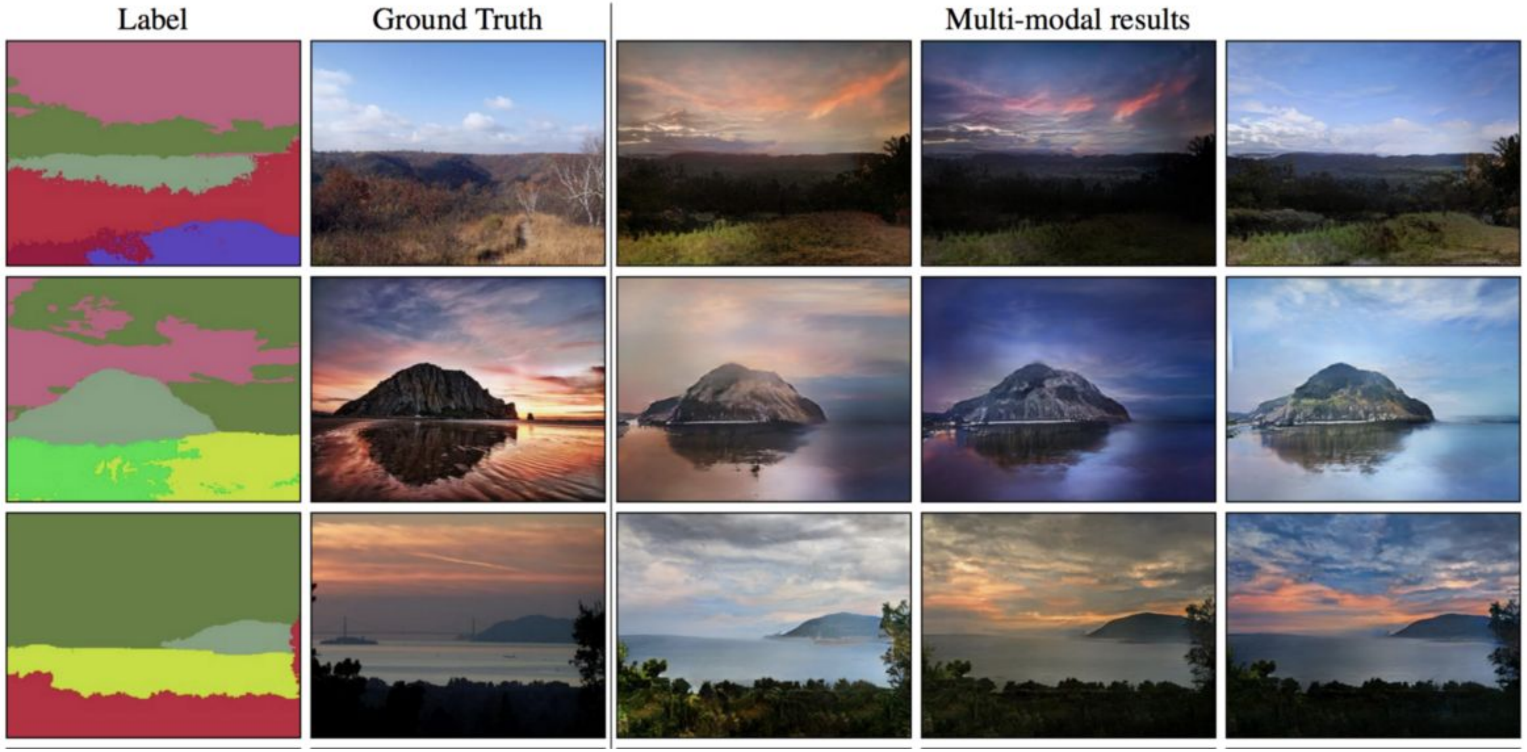
How to Image-to-Image Translation





Text-to-Image Translation

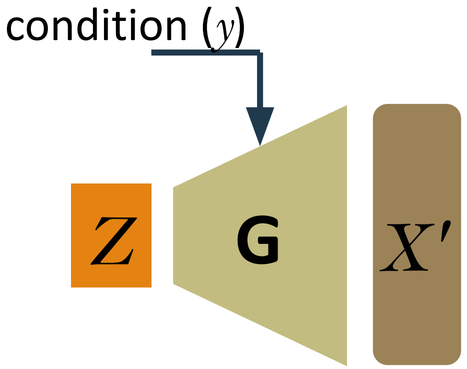
Semantic-Image-to-Photo Translation

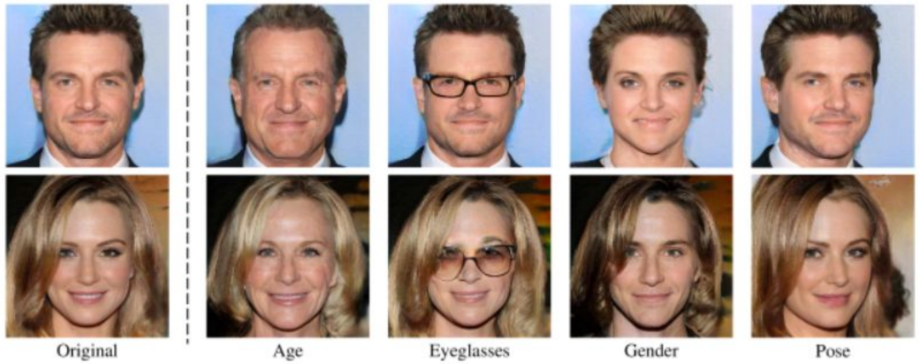


http://nvidia-research-mingyuliu.com/gaugan

Conditional GANs

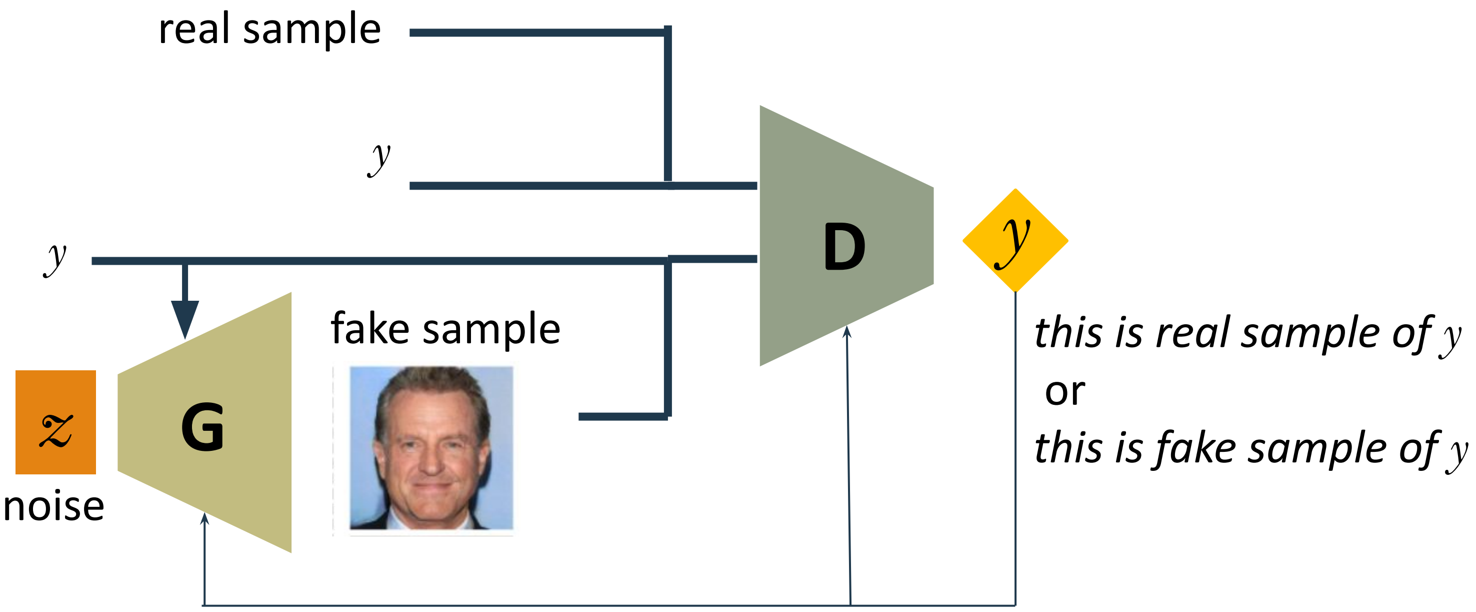
- **Goal:** Better control of the generation
- **Idea:** Add information about the generated samples (e.g., labels) to train the generator





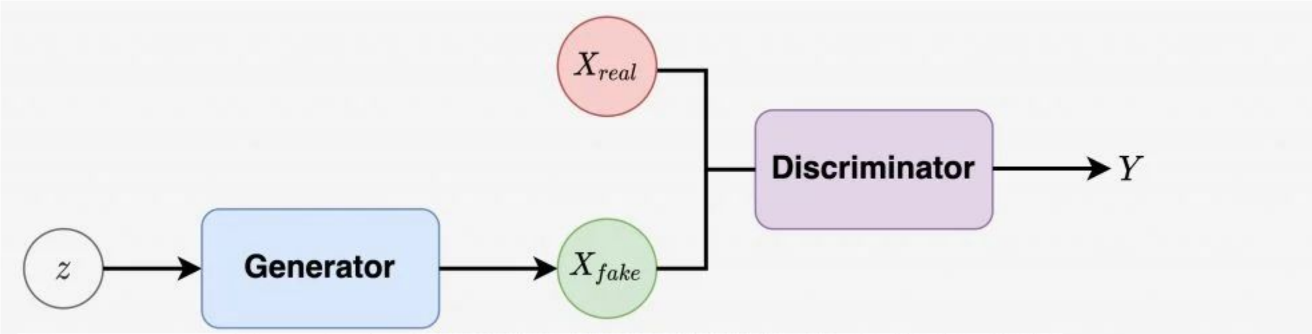
Training cGANs

- **Main difference:** Discriminator gets data sample and condition (e.g., label)

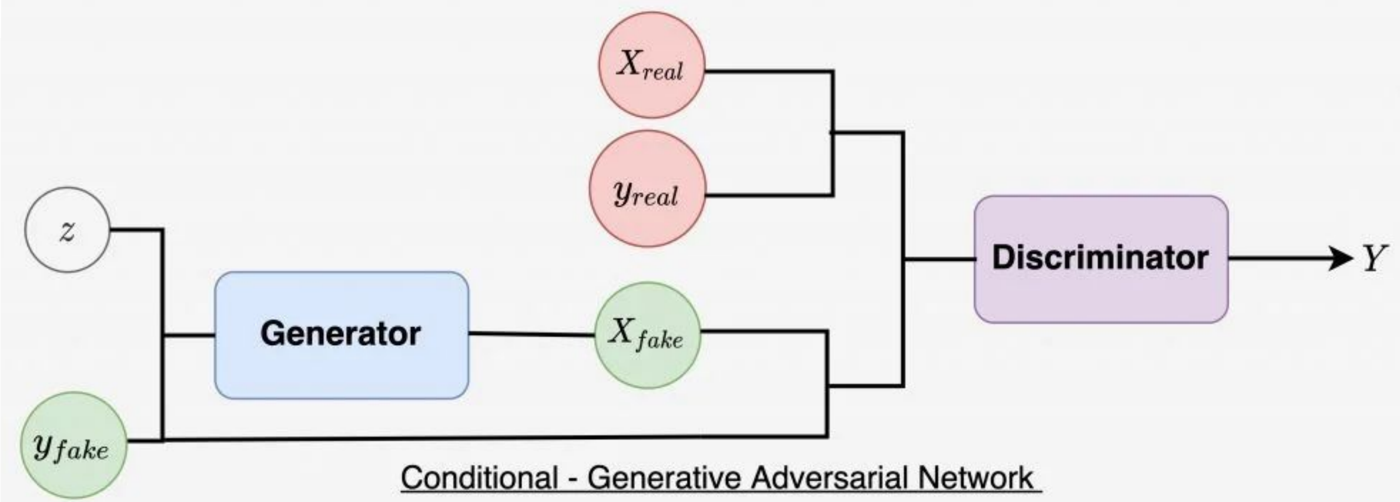


$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x,y)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z,y),y))]$$

GANs vs cGAN



$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$



Conditional - Generative Adversarial Network

$$\mathop{\mathbb{E}}_{x\sim p_{data}(x)}[\log D(x, y)] + \mathop{\mathbb{E}}_{z\sim p_z(z)}[\log(1 - D(G(z, y), y))]$$

Conditional GANs in Pytorch and Tensorflow

INCLUIR

https://learnopencv.com/conditional-gan-cgan-in-pytorch-and-tensorflow/

GAN Training. General Code

0. Create ANNs

cGAN

```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_emb = nn.Embedding(10, 10)

        self.model = nn.Sequential(
            nn.Linear(110, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(1024, 784),
            nn.Tanh()
        )

    def forward(self, z, labels):
        z = z.view(z.size(0), 100)
        c = self.label_emb(labels)
        x = torch.cat([z, c], 1)
        out = self.model(x)
        return out.view(x.size(0), 28, 28)
```

GAN

```
class Generator(nn.Module):
    """
    Class that defines the the Generator Neural Network
    """
    def __init__(self, input_size, hidden_size, output_size):
        super(Generator, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.SELU(),
            nn.Linear(hidden_size, hidden_size),
            nn.SELU(),
            nn.Linear(hidden_size, output_size),
            nn.SELU(),
        )

    def forward(self, x):
        x = self.net(x)
        return x
```

GAN Training. General Code

0. Create ANNs

cGAN

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_emb = nn.Embedding(10, 10)

        self.model = nn.Sequential(
            nn.Linear(794, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x, labels):
        x = x.view(x.size(0), 784)
        c = self.label_emb(labels)
        x = torch.cat([x, c], 1)
        out = self.model(x)
        return out.squeeze()
```

GAN

```
class Discriminator(nn.Module):
    """
    Class that defines the the Discriminator Neural Network
    """
    def __init__(self, input_size, hidden_size, output_size):
        super(Discriminator, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ELU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ELU(),
            nn.Linear(hidden_size, output_size),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.net(x)
        return x
```

GAN Training. General Code

1. Train discriminator

GAN

```
# 1. Train the discriminator
discriminator.zero_grad()
# 1.1 Train discriminator on real data
input_real = get_data_samples(batch_size)
discriminator_real_out = discriminator(input_real.res
discriminator_real_loss = discriminator_loss(discrimi
discriminator_real_loss.backward()
# 1.2 Train the discriminator on data produced by the
input_fake = read_latent_space(batch_size)
generator_fake_out = generator(input_fake).detach()
discriminator_fake_out = discriminator(generator_fake
discriminator_fake_loss = discriminator_loss(discrimi
discriminator_fake_loss.backward()
# 1.3 Optimizing the discriminator weights
discriminator_optimizer.step()
```

cGAN

```
# 1 Train discriminator on real data
real_validity = discriminator(real_images, labels)
real_loss = criterion(real_validity, Variable(torch.ones(batch_size)).to(device))

# 2 Train the discriminator on data produced by the generator
z = Variable(torch.randn(batch_size, 100)).to(device)
fake_labels = Variable(torch.LongTensor(np.random.randint(0, 10, batch_size))).to(device)
fake_images = generator(z, fake_labels)
fake_validity = discriminator(fake_images, fake_labels)
fake_loss = criterion(fake_validity, Variable(torch.zeros(batch_size)).to(device))

# 3 Get loss to train the discriminator from both losses
d_loss = real_loss + fake_loss
d_loss.backward()
```

GAN Training. General Code

2. Train generator

GAN

```
# 2.1 Create fake data
input_fake = read_latent_space(batch_size)
generator_fake_out = generator(input_fake)
# 2.2 Try to fool the discriminator with fak
discriminator_out_to_train_generator = discr
discriminator_loss_to_train_generator = gene

discriminator_loss_to_train_generator.backwa
# 2.3 Optimizing the generator weights
generator_optimizer.step()
```

cGAN

```
# 1 Create fake data
z = Variable(torch.randn(batch_size, 100)).to(device)
fake_labels = Variable(torch.LongTensor(np.random.randint(0, 10, batch_size))).to(devic
fake_images = generator(z, fake_labels)

# 2 Try to fool the discriminator with fake data
validity = discriminator(fake_images, fake_labels)
g_loss = criterion(validity, Variable(torch.ones(batch_size)).to(device))
g_loss.backward()

# 3 Optimize the generator weights
g_optimizer.step()
```

GAN Training. Source Code Example 1

Example: Train a generator to create samples of Fashion-MNIST (grayscale clothing 28x28 images).
Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.
It shares the same image size and structure of MNIST training and testing splits

- Source code: https://colab.research.google.com/drive/1sG_Yfw2_Q38BUxNLjZsk7CLtlIBdCQ-Q

Thanks!

Comments?

JAMAL TOUTOUH
toutouh@mit.edu
jamal.es
necol.net
@jamtou