GANs - Generative Adversarial Network

- Introduction
- Generative Models
- GAN Structure
- Discriminator
- Generator
- Training
- Loss Functions

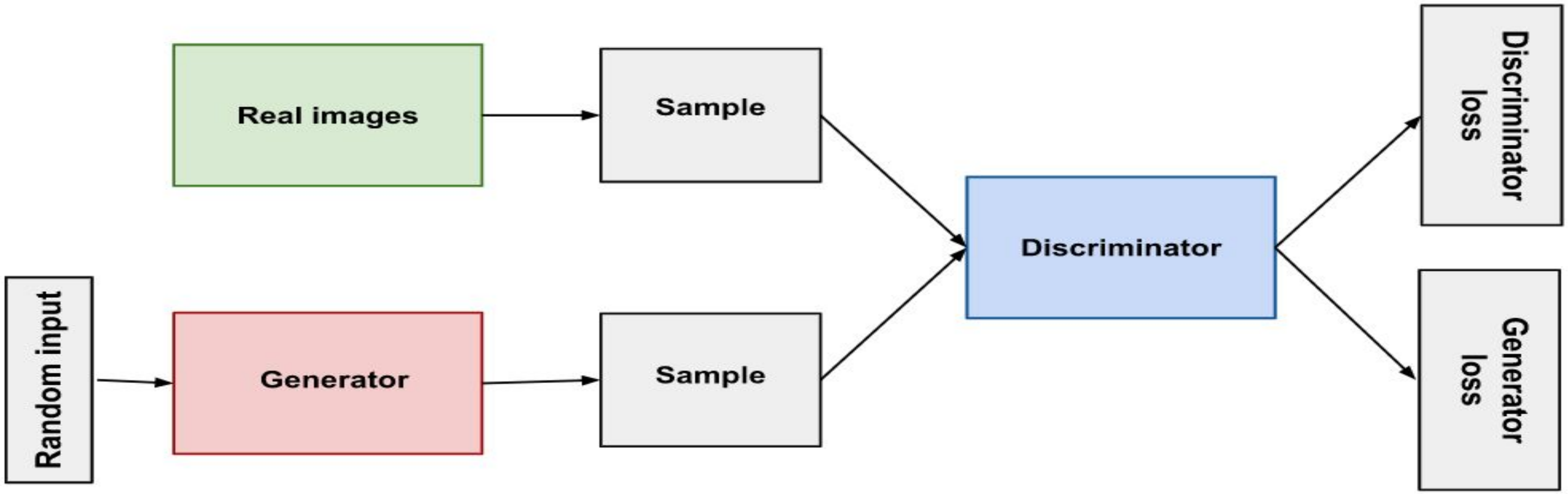
Introduction

- GANs Create instances of data which resembles the training data
- Example: GANs can create images that look like photographs of Cloths, human faces, etc.
- GANs do this with the help of two neural networks, Generator and Discriminator

Generative Models

- Generative models can generate new data instances.
- Discriminative models discriminate between different kinds of data instances
- For data set instances X and a set of labels Y:
    - Generative models capture the joint probability $p(X, Y)$, / $p(X)$ if there are no labels.
    - Discriminative models capture the conditional probability $p(Y \mid X)$.

GAN Structure



Ref:https://developers.google.com/

Discriminator

The discriminator is a classifier, It Distinguishes the Real Data from data generated from Generator

Discriminator Training Data

- Real data instances, (real pictures of cloths/people).
  - The discriminator uses these instances as positive examples during training.
- Fake data instances created by the generator.
  - The discriminator uses these instances as negative examples during training.

Discriminator Training

Discriminator Penalization:

The discriminator loss does penalty for real as fake / fake as real

The discriminator:

Updates its weights through backpropagation from the discriminator loss

Generator

Learns from feedback from discriminator

Learns to make the discriminator classify its output as real.

Generator Training

- Sample random noise. Produce generator output from sampled random noise.
- Get discriminator "Real" or "Fake" classification for generator output.
- Calculate loss from discriminator classification.
- Backpropagate through both the discriminator and generator to obtain gradients.
- Use gradients to change only the generator weights.

GAN Training

Generator and Discrimininator training alternating periods:

- The discriminator trains for one or more epochs.
- The generator trains for one or more epochs.
- Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

Training ---> Discriminator accuracy get worse (can't tell real or fake)

At Generator (Success) → Discriminator (50% Accuracy)

Convergence is difficult to find as the discriminator feedback gets less meaningful over time.

Loss Functions

- Difference between distribution of the data generated by the GAN and the distribution of the real data.
- A GAN can have two loss functions: one for generator training and one for discriminator training.

Digits Recognition using GANs

Preparing the Training Data

```
4]: transform = transforms.Compose(
        [transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))]
    )
```

```
5]: # Load the data
    train_set = torchvision.datasets.MNIST(
        root=".", train=True, download=True, transform=transform
    )
```

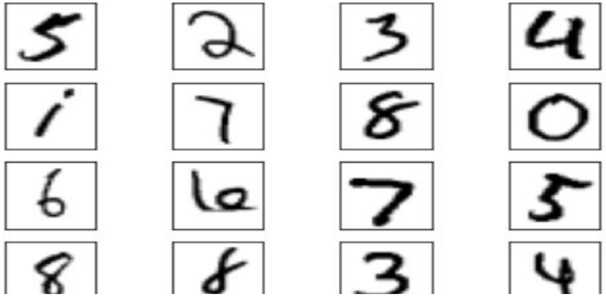
```
6]: batch_size = 32
    train_loader = torch.utils.data.DataLoader(
        train_set, batch_size=batch_size, shuffle=True
    )
```

transforms.ToTensor() converts the data to a PyTorch tensor.

transforms.Normalize() converts the range of the tensor coefficients.

```
In [6]: batch_size = 32
        train_loader = torch.utils.data.DataLoader(
            train_set, batch_size=batch_size, shuffle=True
        )
```

```
In [7]: # plot sample data

        real_samples, mnist_labels = next(iter(train_loader))
        for i in range(16):
            ax = plt.subplot(4, 4, i + 1)
            plt.imshow(real_samples[i].reshape(28, 28), cmap="gray_r")
            plt.xticks([])
            plt.yticks([])
```



Plot sample data

yoona.ai

Implementing the Discriminator

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(784, 1024),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        x = x.view(x.size(0), 784)
        output = self.model(x)
        return output
```

Implementing the Generator

```
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 784),
            nn.Tanh(),
        )

    def forward(self, x):
        output = self.model(x)
        output = output.view(x.size(0), 1, 28, 28)
        return output

generator = Generator().to(device=device)
```

Training the Models

yoona.ai

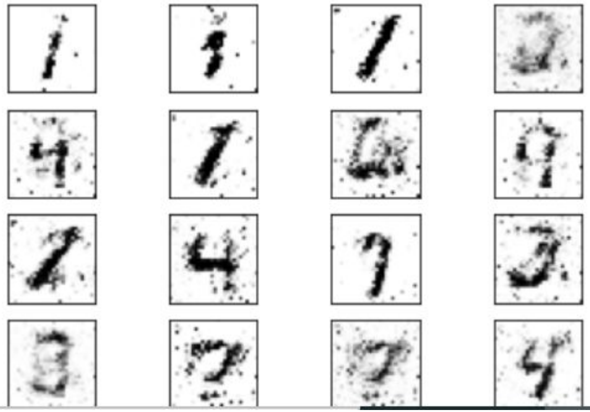
[11]:
```
lr = 0.0001
num_epochs = 10
loss_function = nn.BCELoss()

optimizer_discriminator = torch.optim.Adam(discriminator.parameters(), lr=lr)
optimizer_generator = torch.optim.Adam(generator.parameters(), lr=lr)
```

[13]:
```
for epoch in range(num_epochs):
    for n, (real_samples, mnist_labels) in enumerate(train_loader):
        # Data for training the discriminator
        real_samples = real_samples.to(device=device)
        real_samples_labels = torch.ones((batch_size, 1)).to(
            device=device
        )
        latent_space_samples = torch.randn((batch_size, 100)).to(
            device=device
        )
        generated_samples = generator(latent_space_samples)
        generated_samples_labels = torch.zeros((batch_size, 1)).to(
            device=device
        )
        all_samples = torch.cat((real_samples, generated_samples))
```

Checking the Samples Generated by the GAN

```
latent_space_samples = torch.randn(batch_size, 100).to(device=device)
generated_samples = generator(latent_space_samples)
```

```
generated_samples = generated_samples.cpu().detach()
for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(generated_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
```



yoona.ai

For Fashion cloths data:

Final output should be like this

Industry Ready AI Fashion Designs