Training Report Day-28

8 July 2024

Generative Adversarial Networks (GANs) are a revolutionary type of neural network architecture in AI, designed to generate new data samples that resemble a given training set. Developed by Ian Goodfellow in 2014, GANs consist of two networks—a generator and a discriminator—that work against each other in a “game,” learning to generate realistic outputs. GANs have found applications in a variety of fields, including image synthesis, style transfer, data augmentation, and even AI-driven art creation.

1. How GANs Work

GANs involve two competing neural networks:

- **Generator**: The generator network’s job is to create new data instances that resemble the training data. It starts with random noise and generates data samples based on this input.
- **Discriminator**: The discriminator is trained to distinguish between real data samples (from the training set) and fake samples (generated by the generator).

The GAN training process involves both networks playing a “zero-sum” game:

1. The generator creates fake samples to try and "fool" the discriminator.
2. The discriminator receives both real and generated samples and tries to correctly classify them as real or fake.
3. The generator learns by receiving feedback from the discriminator and adjusts to produce more realistic data, while the discriminator improves its classification ability.

Over time, the generator becomes better at creating realistic samples, while the discriminator improves its ability to detect fakes—ideally leading to a point where the generated samples are indistinguishable from the real data.

2. Training a GAN

Training GANs can be challenging because it requires a careful balance between the generator and discriminator. If one network becomes too strong, it can overpower the other, causing the training to fail. Common problems include:

- **Mode Collapse**: The generator may produce a limited variety of outputs, known as "mode collapse."
- **Vanishing/Exploding Gradients**: Due to the adversarial nature of GANs, gradients can become unstable.

To overcome these, techniques like **Wasserstein GAN** (WGAN), **gradient penalty**, and **batch normalization** are often used.

3. Types of GANs and Their Applications

3.1 Vanilla GAN

The original GAN architecture is the basic model where both generator and discriminator are simple feed-forward neural networks. Although basic, Vanilla GANs paved the way for more complex architectures.

3.2 Deep Convolutional GAN (DCGAN)

DCGANs use convolutional layers instead of fully connected layers, making them more suitable for image data. They are especially good at generating high-resolution images and are a popular choice for applications in image generation.

3.3 Conditional GAN (cGAN)

Conditional GANs allow for conditioning on labels or specific data points, so the generator creates samples conditioned on a certain attribute (e.g., generating images of a specific category, like cats or dogs).

3.4 StyleGAN

StyleGANs, developed by NVIDIA, are advanced GANs used to generate high-quality, realistic images. They introduced a style-based generator architecture, allowing for fine-grained control over the features of the generated images, such as hair color or facial features.

StyleGAN is behind projects like "thispersondoesnotexist.com," which generates photorealistic faces of nonexistent people.

3.5 CycleGAN

CycleGANs are designed for tasks that involve image-to-image translation without needing paired data (e.g., converting horses to zebras, or photographs to paintings). They use a "cycle consistency loss" to enforce that translating an image to a different domain and then back results in the original image.

3.6 Super-Resolution GAN (SRGAN)

SRGANs are GANs designed for image super-resolution tasks, improving the resolution of low-quality images. This has applications in fields like medical imaging, satellite imagery, and digital forensics.

4. Common Applications of GANs

4.1 Image Synthesis

GANs are extensively used to generate high-quality, realistic images. Applications range from creating new art and enhancing video game graphics to training data augmentation by creating synthetic images.

4.2 Image-to-Image Translation

GANs perform tasks like turning sketches into realistic photos, transforming black-and-white images to color, and converting daytime scenes to nighttime scenes. For example, **Pix2Pix** and **CycleGAN** architectures are used to translate images between different styles.

4.3 Text-to-Image Generation

GANs can also generate images from textual descriptions, such as "a yellow bird with black wings." Models like **StackGAN** and **AttnGAN** generate images that match the semantic content described in a text input, useful for creative industries and rapid prototyping.

4.4 Data Augmentation

In fields like medical imaging, GANs can augment limited datasets by generating synthetic images that help models generalize better. This is crucial for domains where collecting labeled data is challenging.

4.5 Super-Resolution and Image Restoration

GANs, especially SRGAN, are used for image enhancement by generating high-resolution versions of low-quality images. They’re also applied to remove artifacts, restore old photographs, and sharpen images.

4.6 Video and Animation

GANs can be extended to generate video frames in sequence, which is useful for animations, video compression, and even creating synthetic training data for action recognition in videos.

4.7 Anomaly Detection

GANs can learn to generate "normal" data distributions, making them useful for identifying anomalies by detecting samples that don't match the learned distribution. This is used in fraud detection, defect detection in manufacturing, and other security applications.

5. Code Example: Building a Simple GAN for Image Generation

Here’s an example of a simple GAN to generate images similar to the MNIST dataset (handwritten digits).

CODE:-

```
import tensorflow as tf

from tensorflow.keras.layers import Dense, Reshape, Flatten, LeakyReLU, BatchNormalization

from tensorflow.keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt
```

```
# Load and preprocess MNIST data

(x_train, _), (_, _) = mnist.load_data()

x_train = x_train / 127.5 - 1.0  # Normalize to [-1, 1]

x_train = np.expand_dims(x_train, axis=-1)



# Generator model

def build_generator():

    model = tf.keras.Sequential([

        Dense(256, input_shape=(100,)),

        LeakyReLU(0.2),

        BatchNormalization(),

        Dense(512),

        LeakyReLU(0.2),

        BatchNormalization(),

        Dense(1024),

        LeakyReLU(0.2),

        BatchNormalization(),

        Dense(28 * 28 * 1, activation='tanh'),

        Reshape((28, 28, 1))
    ])

    return model



# Discriminator model

def build_discriminator():

    model = tf.keras.Sequential([

        Flatten(input_shape=(28, 28, 1)),

        Dense(512),

        LeakyReLU(0.2),

        Dense(256),

        LeakyReLU(0.2),

        Dense(1, activation='sigmoid')

    ])

    return model



# Compile models

generator = build_generator()

discriminator = build_discriminator()

discriminator.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# GAN model

discriminator.trainable = False

gan_input = tf.keras.Input(shape=(100,))

generated_image = generator(gan_input)

gan_output = discriminator(generated_image)

gan = tf.keras.Model(gan_input, gan_output)

gan.compile(optimizer='adam', loss='binary_crossentropy')



# Training

epochs = 10000

batch_size = 64

half_batch = batch_size // 2



for epoch in range(epochs):

    # Train discriminator

    idx = np.random.randint(0, x_train.shape[0], half_batch)

    real_images = x_train[idx]

    noise = np.random.normal(0, 1, (half_batch, 100))

    fake_images = generator.predict(noise)
```

```
d_loss_real = discriminator.train_on_batch(real_images, np.ones((half_batch, 1)))

d_loss_fake = discriminator.train_on_batch(fake_images, np.zeros((half_batch, 1)))

d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)



# Train generator

noise = np.random.normal(0, 1, (batch_size, 100))

g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))



# Print progress and display generated images occasionally

if epoch % 1000 == 0:

    print(f"{epoch} [D loss: {d_loss[0]}, acc.: {100 * d_loss[1]}] [G loss: {g_loss}]")

    noise = np.random.normal(0, 1, (16, 100))

    generated_images = generator.predict(noise)

    plt.figure(figsize=(4, 4))

    for i in range(generated_images.shape[0]):

        plt.subplot(4, 4, i + 1)

        plt.imshow(generated_images[i, :, :, 0] * 0.5 + 0.5, cmap='gray')

        plt.axis('off')

    plt.show()
```