第 5 章

CHAPTER 5

生成对抗网络

本章介绍生成对抗网络(Generative Adversarial Network,GAN)[3]。生成对抗网络在AI界书写了一个以假乱真的剧本。近年来AI换脸等技术火爆全球,离不开这个网络的点滴贡献。生成对抗网络能够学习数据的分布规律,并创造出类似我们真实世界的物件如图像、文本等。从以假乱真的程度上看,它甚至可以被誉为深度学习中的艺术家。好了,闲言少叙,我们这就走进生成对抗网络的世界。

5.1 生成对抗网络的原理

视频讲解

相信大家都会画画,不管画得好坏与否,但总归会对着图案勾上两笔。我们临摹的次数越多,画得也就越像。最后,临摹到了极致,我们的画就和临摹的那幅画一模一样了,以至于专家也无法分清到底哪幅画是赝品。好了,在这个例子中,我们将主人公换成生成对抗网络,画画这个操作换成训练,其实也是这么一回事。总体来说,就是这个网络学习数据分布的规律,然后弄出一个和原先数据分布规律相同的数据。这个数据可以是语音、文字和图像等。

生成对抗网络的网络结构拥有两个部分,一个是生成器(Generator),另一个是辨别器(Discriminator)。现在我们拿手写数字图片来举个例子。我们希望GAN能临摹出和手写数字图片一样的图,达到以假乱真的程度。生成对抗网络结构图如图5.1所示。

它整体的流程如下:

(1) 首先定义一个生成器,输入一组随机噪声向量(最好符合常见的分布,一般的数据分布都呈现常见分布规律),输出为一个图片。

(2) 定义一个辨别器,用它来判断图片是否为训练集中的图片,是为真,否为假。

(3) 当辨别器无法分辨真假,即判别概率为0.5时,停止训练。

其中,生成器和辨别器就是我们要搭建的神经网络模型,可以是CNN、RNN或者全连接神经网络等,只要能完成任务即可。

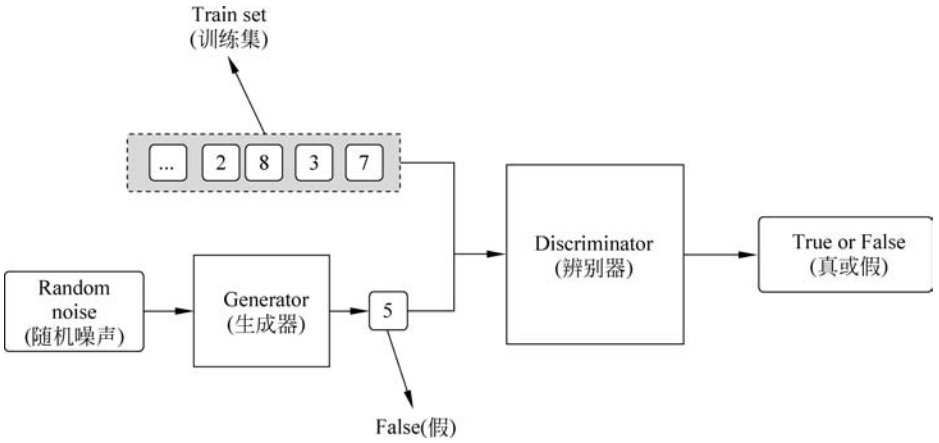


图 5.1　生成对抗网络

5.2　生成对抗网络的训练过程

(1) 初始化生成器 G 和辨别器 D 两个网络的参数。

(2) 从训练集抽取 n 个样本,以及生成器利用定义的噪声分布生成 n 个样本。固定生成器 G,训练辨别器 D,使其尽可能区分真假。

(3) 循环更新 k 次辨别器 D 之后,更新 1 次生成器 G,使辨别器尽可能区分不了真假。

多次更新迭代后,在理想状态下,最终辨别器 D 无法区分图片到底是来自真实的训练样本集合,还是来自生成器 G 生成的样本,此时辨别的概率为 0.5,完成训练。

论文的作者尝试用这个框架分别对 MNIST[4]、Toronto Face Database (TFD)[5] 和 CIFAR-10[6] 训练了 4 个生成对抗网络模型。如图 5.2～图 5.5 所示。样本是公平的随机抽签,并非精心挑选。最右边的列显示了模型预测生成的示例。

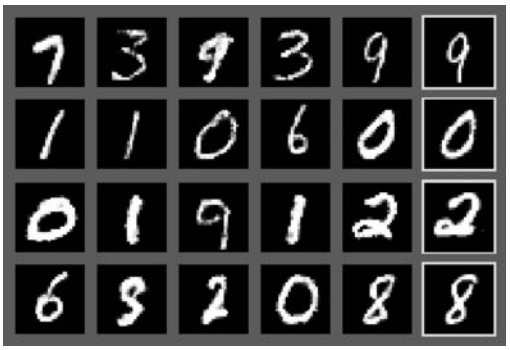


图 5.2　MNIST 数据集[4]

图 5.3　Toronto Face Database 数据集[5]

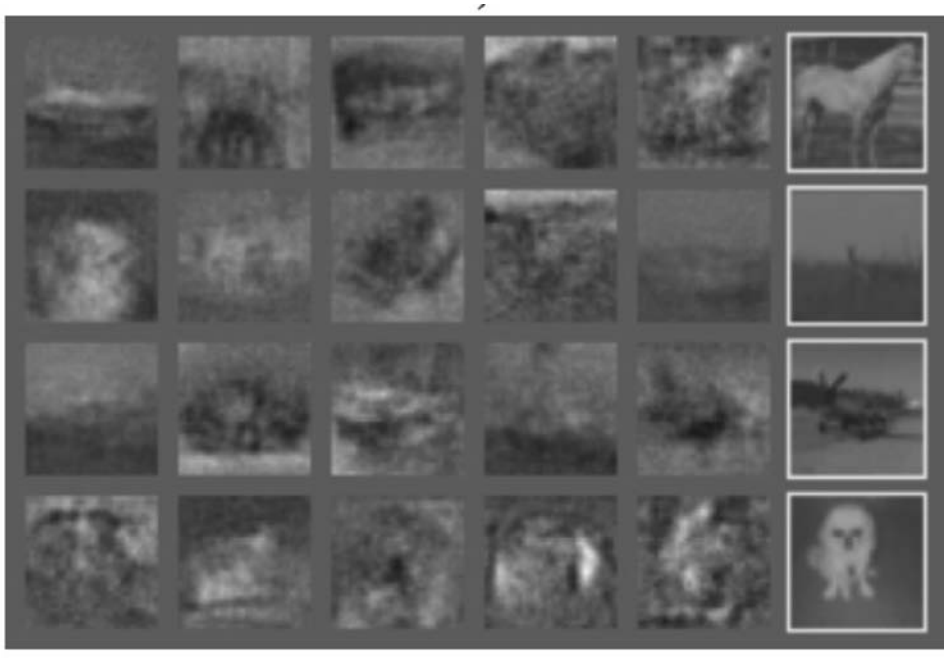


图 5.4　CIFAR-10（全连接层网络）[6]

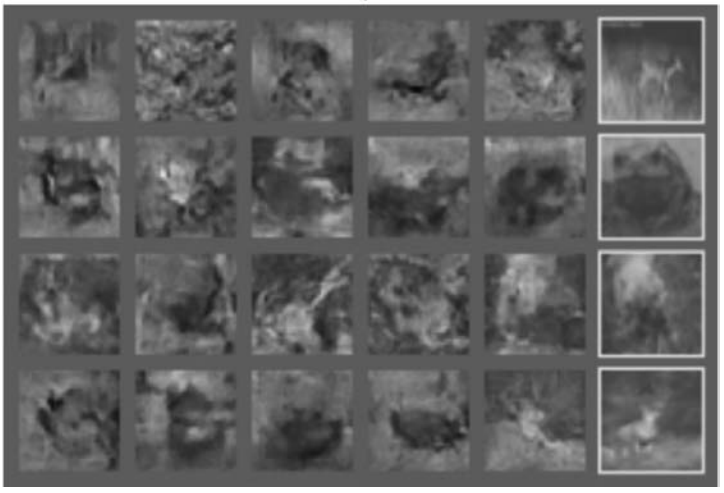


图 5.5　CIFAR-10（卷积辨别器和反卷积生成器）[6]

5.3　实验

视频讲解

同样地，我们依旧通过实验来巩固我们刚刚所学的知识点。本次实验基于 Jupyer Notebook、Anaconda Python3.7 与 Keras 环境。数据集利用 Minst 手写体图像数据集。

5.3.1　代码

```
1.  # chapter5/5_3_GAN.ipynb
2.  import random
3.  import numpy as np
4.  from keras.layers import Input
5.  from keras.layers.core import Reshape,Dense,Dropout,Activation,Flatten
6.  from keras.layers.advanced_activations import LeakyReLU
7.  from keras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D,
    Deconv2D, UpSampling2D
8.  from keras.regularizers import *
9.  from keras.layers.normalization import *
10. from keras.optimizers import *
11. from keras.datasets import mnist
12. import matplotlib.pyplot as plt
13. from keras.models import Model
14. from tqdm import tqdm
15. from IPython import display
```

1. 读取数据集

```
img_rows, img_cols = 28, 28

# 数据集的切分与混洗(shuffle)
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

print(np.min(X_train), np.max(X_train))
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
0.0 1.0
X_train shape: (60000, 1, 28, 28)
60000 train samples
10000 test samples
```

2. 超参数设置

```
shp = X_train.shape[1:]
dropout_rate = 0.25

# 优化器
opt = Adam(lr=1e-4)
dopt = Adam(lr=1e-5)
```

3. 定义生成器

```
K.set_image_dim_ordering('th')  # 用 theano 的图片输入顺序
# 生成 1×28×28 的图片
nch = 200
g_input = Input(shape=[100])
H = Dense(nch*14*14, kernel_initializer='glorot_normal')(g_input)
H = BatchNormalization()(H)
H = Activation('relu')(H)
H = Reshape( [nch, 14, 14] )(H)     # 转成 200×14×14
H = UpSampling2D(size=(2, 2))(H)
H = Convolution2D(100, (3, 3), padding="same", kernel_initializer='glorot_normal')(H)
H = BatchNormalization()(H)
```

```
H = Activation('relu')(H)
H = Convolution2D(50, (3, 3), padding = "same", kernel_initializer = 'glorot_normal')(H)
H = BatchNormalization()(H)
H = Activation('relu')(H)
H = Convolution2D(1, (1, 1), padding = "same", kernel_initializer = 'glorot_normal')(H)
g_V = Activation('sigmoid')(H)
generator = Model(g_input,g_V)
generator.compile(loss = 'binary_crossentropy', optimizer = opt)
generator.summary()
```

4. 定义辨别器

```
#辨别是否来自真实训练集
d_input = Input(shape = shp)
H = Convolution2D(256, (5, 5), activation = "relu", strides = (2, 2), padding = "same")
(d_input)
H = LeakyReLU(0.2)(H)
H = Dropout(dropout_rate)(H)
H = Convolution2D(512, (5, 5), activation = "relu", strides = (2, 2), padding = "same")(H)
H = LeakyReLU(0.2)(H)
H = Dropout(dropout_rate)(H)
H = Flatten()(H)
H = Dense(256)(H)
H = LeakyReLU(0.2)(H)
H = Dropout(dropout_rate)(H)
d_V = Dense(2,activation = 'softmax')(H)
discriminator = Model(d_input,d_V)
discriminator.compile(loss = 'categorical_crossentropy', optimizer = dopt)
discriminator.summary()
```

5. 构造生成对抗网络

```
# 冷冻训练层
def make_trainable(net, val):
    net.trainable = val
    for l in net.layers:
        l.trainable = val
make_trainable(discriminator, False)

# 构造 GAN
gan_input = Input(shape = [100])
H = generator(gan_input)
gan_V = discriminator(H)
GAN = Model(gan_input, gan_V)
GAN.compile(loss = 'categorical_crossentropy', optimizer = opt)
GAN.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_33 (InputLayer)        (None, 100)               0
model_18 (Model)             (None, 1, 28, 28)         4341425
_________________________________________________________________
model_19 (Model)             (None, 2)                 9707266
=================================================================
Total params: 14,048,691
Trainable params: 4,262,913
Non-trainable params: 9,785,778
_________________________________________________________________
```

6. 训练

```
# 描绘损失收敛过程
def plot_loss(losses):
        display.clear_output(wait=True)
        display.display(plt.gcf())
        plt.figure(figsize=(10,8))
        plt.plot(losses["d"], label='discriminitive loss')
        plt.plot(losses["g"], label='generative loss')
        plt.legend()
        plt.show()


# 描绘生成器生成图像
def plot_gen(n_ex=16,dim=(4,4), figsize=(10,10) ):
    noise = np.random.uniform(0,1,size=[n_ex,100])
    generated_images = generator.predict(noise)

    plt.figure(figsize=figsize)
    for i in range(generated_images.shape[0]):
        plt.subplot(dim[0],dim[1],i+1)
        img = generated_images[i,0,:,:]
        plt.imshow(img)
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# 抽取训练集样本
ntrain = 10000
trainidx = random.sample(range(0,X_train.shape[0]), ntrain)
XT = X_train[trainidx,:,:,:]
```

```
30.
31. # 预训练辨别器
32. noise_gen = np.random.uniform(0,1,size=[XT.shape[0],100])
33. generated_images = generator.predict(noise_gen)# 生成器产生样本
34. X = np.concatenate((XT, generated_images))
35. n = XT.shape[0]
36. y = np.zeros([2*n,2])                          # 构造辨别器标签 One-hot encode
37. y[:n,1] = 1
38. y[n:,0] = 1
39.
40. make_trainable(discriminator,True)
41. discriminator.fit(X,y, epochs=1, batch_size=32)
42. y_hat = discriminator.predict(X)
```

```
Epoch 1/1
20000/20000 [==============================] - 288s 14ms/step - loss: 0.0469
```

```
1.  # 计算辨别器的准确率
2.  y_hat_idx = np.argmax(y_hat,axis=1)
3.  y_idx = np.argmax(y,axis=1)
4.  diff = y_idx-y_hat_idx
5.  n_total = y.shape[0]
6.  n_right = (diff==0).sum()
7.
8.  print("(%d of %d) right"  % (n_right, n_total))
```

```
(20000 of 20000) right
```

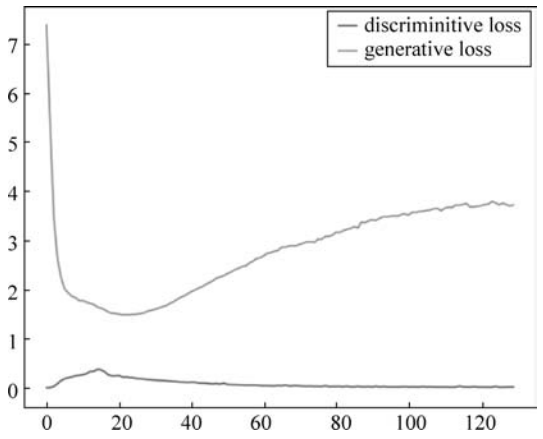
```
1.  def train_for_n(nb_epoch=5000, plt_frq=25,BATCH_SIZE=32):
2.      for e in tqdm(range(nb_epoch)):
3.
4.          # 生成器生成样本
5.          image_batch = X_train[np.random.randint(0,X_train.shape[0],size=BATCH_
            SIZE),:,:,:]
6.          noise_gen = np.random.uniform(0,1,size=[BATCH_SIZE,100])
7.          generated_images = generator.predict(noise_gen)
8.
9.          # 训练辨别器
10.         X = np.concatenate((image_batch, generated_images))
11.         y = np.zeros([2*BATCH_SIZE,2])
12.         y[0:BATCH_SIZE,1] = 1
13.         y[BATCH_SIZE:,0] = 1
14.
15.         # 存储辨别器损失(loss)
```

```
        make_trainable(discriminator,True)
        d_loss  = discriminator.train_on_batch(X,y)
        losses["d"].append(d_loss)

        # 生成器生成样本
        noise_tr = np.random.uniform(0,1,size=[BATCH_SIZE,100])
        y2 = np.zeros([BATCH_SIZE,2])
        y2[:,1] = 1

        # 存储生成器损失(loss)
        make_trainable(discriminator,False)  # 关掉辨别器的训练
        g_loss = GAN.train_on_batch(noise_tr, y2)
        losses["g"].append(g_loss)

        # 更新损失(loss)图
        if e%plt_frq == plt_frq-1:
            plot_loss(losses)
            plot_gen()
train_for_n(nb_epoch=1000, plt_frq=10,BATCH_SIZE=128)
```



5.3.2 结果分析

从模型输出的 loss 我们可以知道生成器与辨别器两者拟合的 loss 并不是特别好，因此我们可以通过调参来解决。主要调参方向有以下 4 点：

(1) batch size。

(2) adam 优化器的 learning rate。

(3) 迭代次数 nb_epoch。

(4) 生成器和辨别器的网络结构。

5.4　总结

本章讲解了生成对抗网络的知识点。大家在掌握了整个流程之后，就可以将笔者的代码修改成自己所需要的场景，进而训练自己的 GAN 模型了。

笔者在本章介绍的 GAN 只是 2014 年的开山之作，之后有很多人基于 GAN 提出了许多有趣的实验，但是所用的网络原理都差不多，这里就不一一赘述了。而且 GAN 的应用范围非常广阔，例如市面上很火的“换脸”软件，大多都是基于 GAN 的原理。甚至我们也可以利用 GAN 去做数据增强，例如在我们缺少训练集的时候，可以考虑用 GAN 生成一些数据，扩充我们的训练样本。