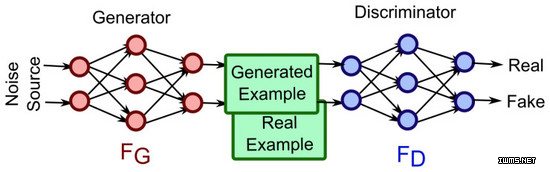
**不到 200 行代码，教你如何用 Keras 搭建生成对抗网络（GAN）**

|  |  |  |
| --- | --- | --- |
| [日期：2017-04-01] | 来源：雷锋网  作者： | [字体：[大](javascript:ContentSize(16)) [中](javascript:ContentSize(0)) [小](javascript:ContentSize(12))] |



生成对抗网络（Generative Adversarial Networks，GAN）最早由 Ian Goodfellow 在 2014 年提出，是目前[深度学习](http://www.dlworld.cn/XueXiSuanFa/90.html" \t "_blank)领域最具潜力的研究成果之一。它的核心思想是：同时训练两个相互协作、同时又相互竞争的深度神经网络（一个称为生成器 Generator，另一个称为判别器 Discriminator）来处理无监督学习的相关问题。在训练过程中，两个网络最终都要学习如何处理任务。

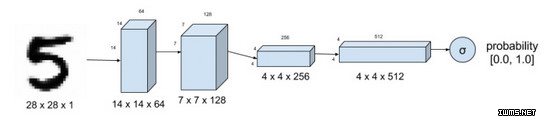
通常，我们会用下面这个例子来说明 GAN 的原理：将警察视为判别器，制造假币的犯罪分子视为生成器。一开始，犯罪分子会首先向警察展示一张假币。警察识别出该假币，并向犯罪分子反馈哪些地方是假的。接着，根据警察的反馈，犯罪分子改进工艺，制作一张更逼真的假币给警方检查。这时警方再反馈，犯罪分子再改进工艺。不断重复这一过程，直到警察识别不出真假，那么模型就训练成功了。

虽然 GAN 的核心思想看起来非常简单，但要搭建一个真正可用的 GAN 网络却并不容易。因为毕竟在 GAN 中有两个相互耦合的深度神经网络，同时对这两个网络进行梯度的反向传播，也就比一般场景困难两倍。

为此，本文将以深度卷积生成对抗网络（Deep Convolutional GAN，DCGAN）为例，介绍如何基于 Keras 2.0 框架，以Tensorflow 为后端，在 200 行代码内搭建一个真实可用的 GAN 模型，并以该模型为基础自动生成 MNIST 手写体数字。

**判别器**

判别器的作用是判断一个模型生成的图像和真实图像比，有多逼真。它的基本结构就是如下图所示的卷积神经网络（Convolutional Neural Network，CNN）。对于 MNIST 数据集来说，模型输入是一个 28x28 像素的单通道图像。Sigmoid 函数的输出值在 0-1 之间，表示图像真实度的概率，其中 0 表示肯定是假的，1 表示肯定是真的。与典型的 CNN 结构相比，这里去掉了层之间的 max-pooling，而是采用了步进卷积来进行下采样。这里每个 CNN 层都以 LeakyReLU 为激活函数。而且为了防止过拟合和记忆效应，层之间的 dropout 值均被设置在 0.4-0.7 之间。具体在 Keras 中的实现代码如下。



self.D = Sequential()

depth = 64

dropout = 0.4

# In: 28 x 28 x 1, depth = 1

# Out: 10 x 10 x 1, depth=64

input\_shape = (self.img\_rows, self.img\_cols, self.channel)

self.D.add(Conv2D(depth\*1, 5, strides=2, input\_shape=input\_shape,\

padding='same', activation=LeakyReLU(alpha=0.2)))

self.D.add(Dropout(dropout))

self.D.add(Conv2D(depth\*2, 5, strides=2, padding='same',\

activation=LeakyReLU(alpha=0.2)))

self.D.add(Dropout(dropout))

self.D.add(Conv2D(depth\*4, 5, strides=2, padding='same',\

activation=LeakyReLU(alpha=0.2)))

self.D.add(Dropout(dropout))

self.D.add(Conv2D(depth\*8, 5, strides=1, padding='same',\

activation=LeakyReLU(alpha=0.2)))

self.D.add(Dropout(dropout))

# Out: 1-dim probability

self.D.add(Flatten())

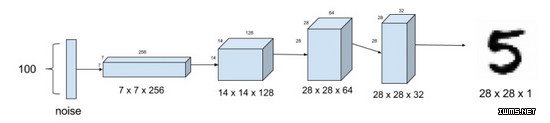
self.D.add(Dense(1))

self.D.add(Activation('sigmoid'))

self.D.summary()

**生成器**

生成器的作用是合成假的图像，其基本机构如下图所示。图中，我们使用了卷积的倒数，即转置卷积（transposed convolution），从 100 维的噪声（满足 -1 至 1 之间的均匀分布）中生成了假图像。如在 DCGAN 模型中提到的那样，去掉微步进卷积，这里我们采用了模型前三层之间的上采样来合成更逼真的手写图像。在层与层之间，我们采用了批量归一化的方法来平稳化训练过程。以 ReLU 函数为每一层结构之后的激活函数。最后一层 Sigmoid 函数输出最后的假图像。第一层设置了 0.3-0.5 之间的 dropout 值来防止过拟合。具体代码如下。



self.G = Sequential()

dropout = 0.4

depth = 64+64+64+64

dim = 7

# In: 100

# Out: dim x dim x depth

self.G.add(Dense(dim\*dim\*depth, input\_dim=100))

self.G.add(BatchNormalization(momentum=0.9))

self.G.add(Activation('relu'))

self.G.add(Reshape((dim, dim, depth)))

self.G.add(Dropout(dropout))

# In: dim x dim x depth

# Out: 2\*dim x 2\*dim x depth/2

self.G.add(UpSampling2D())

self.G.add(Conv2DTranspose(int(depth/2), 5, padding='same'))

self.G.add(BatchNormalization(momentum=0.9))

self.G.add(Activation('relu'))

self.G.add(UpSampling2D())

self.G.add(Conv2DTranspose(int(depth/4), 5, padding='same'))

self.G.add(BatchNormalization(momentum=0.9))

self.G.add(Activation('relu'))

self.G.add(Conv2DTranspose(int(depth/8), 5, padding='same'))

self.G.add(BatchNormalization(momentum=0.9))

self.G.add(Activation('relu'))

# Out: 28 x 28 x 1 grayscale image [0.0,1.0] per pix

self.G.add(Conv2DTranspose(1, 5, padding='same'))

self.G.add(Activation('sigmoid'))

self.G.summary()

return self.G

**生成 GAN 模型**

下面我们生成真正的 GAN 模型。如上所述，这里我们需要搭建两个模型：一个是判别器模型，代表警察；另一个是对抗模型，代表制造假币的犯罪分子。

判别器模型

下面代码展示了如何在 Keras 框架下生成判别器模型。上文定义的判别器是为模型训练定义的损失函数。这里由于判别器的输出为 Sigmoid 函数，因此采用了二进制交叉熵为损失函数。在这种情况下，以 RMSProp 作为优化算法可以生成比 Adam 更逼真的假图像。这里我们将学习率设置在 0.0008，同时还设置了权值衰减和clipvalue等参数来稳定后期的训练过程。如果你需要调节学习率，那么也必须同步调节其他相关参数。

optimizer = RMSprop(lr=0.0008, clipvalue=1.0, decay=6e-8)

self.DM = Sequential()

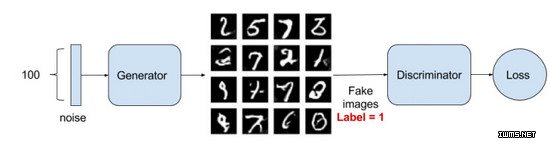
self.DM.add(self.discriminator())

self.DM.compile(loss='binary\_crossentropy', optimizer=optimizer,\

metrics=['accuracy'])

对抗模型

如图所示，对抗模型的基本结构是判别器和生成器的叠加。生成器试图骗过判别器，同时从其反馈中提升自己。如下代码中演示了如何基于 Keras 框架实现这一部分功能。其中，除了学习速率的降低和相对权值衰减之外，训练参数与判别器模型中的训练参数完全相同。



optimizer = RMSprop(lr=0.0004, clipvalue=1.0, decay=3e-8)

self.AM = Sequential()

self.AM.add(self.generator())

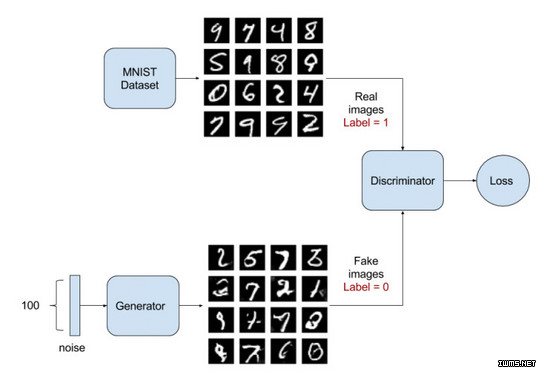
self.AM.add(self.discriminator())

self.AM.compile(loss='binary\_crossentropy', optimizer=optimizer,\

metrics=['accuracy'])

训练

搭好模型之后，训练是最难实现的部分。这里我们首先用真实图像和假图像对判别器模型单独进行训练，以判断其正确性。接着，对判别器模型和对抗模型轮流展开训练。如下图展示了判别器模型训练的基本流程。在 Keras 框架下的实现代码如下所示。



images\_train = self.x\_train[np.random.randint(0,

self.x\_train.shape[0], size=batch\_size), :, :, :]

noise = np.random.uniform(-1.0, 1.0, size=[batch\_size, 100])

images\_fake = self.generator.predict(noise)

x = np.concatenate((images\_train, images\_fake))

y = np.ones([2\*batch\_size, 1])

y[batch\_size:, :] = 0

d\_loss = self.discriminator.train\_on\_batch(x, y)

y = np.ones([batch\_size, 1])

noise = np.random.uniform(-1.0, 1.0, size=[batch\_size, 100])

a\_loss = self.adversarial.train\_on\_batch(noise, y)

训练过程中需要非常耐心，这里列出一些常见问题和解决方案：

问题1：最终生成的图像噪点太多。

解决：尝试在判别器和生成器模型上引入 dropout，一般更小的 dropout 值（0.3-0.6）可以产生更逼真的图像。

问题2：判别器的损失函数迅速收敛为零，导致发生器无法训练。

解决：不要对判别器进行预训练。而是调整学习率，使判别器的学习率大于对抗模型的学习率。也可以尝试对生成器换一个不同的训练噪声样本。

问题3：生成器输出的图像仍然看起来像噪声。

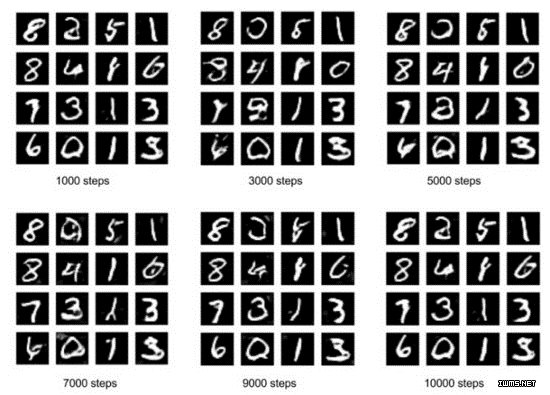
解决：检查激活函数、批量归一化和 dropout 的应用流程是否正确。

问题4：如何确定正确的模型/训练参数。

解决：尝试从一些已经发表的论文或代码中找到参考，调试时每次只调整一个参数。在进行 2000 步以上的训练时，注意观察在 500 或 1000 步左右参数值调整的效果。

**输出情况**

下图展示了在训练过程中，整个模型的输出变化情况。可以看到，GAN 在自己学习如何生成手写体数字。



完整代码地址：

<https://github.com/roatienza/Deep-Learning-Experiments/blob/master/Experiments/Tensorflow/GAN/dcgan_mnist.py>

