上一回我们只是简单地展示了基于keras框架、MNIST数据集的DCGAN模型的结果，下面我们来详细地看一下这个代码的实现。

**生成模型的结构**

def generator\_model():

model = Sequential()

model.add(Dense(input\_dim=100, output\_dim=1024))

model.add(Activation('tanh'))

model.add(Dense(out\_dim=128\*7\*7))

model.add(BatchNormalization())

model.add(Activation('tanh'))

model.add(Reshape((128, 7, 7), input\_shape=(128\*7\*7,)))

model.add(UpSampling2D(size=(2, 2)))

model.add(Convolution2D(out\_channel=64, kernel\_height=5, kernel\_width=5, border\_mode='same'))

model.add(Activation('tanh'))

model.add(UpSampling2D(size=(2, 2)))

model.add(Convolution2D(out\_channel=1, kernel\_height=5, kernel\_width=5, border\_mode='same'))

model.add(Activation('tanh'))

return model

直接上代码了。keras的代码总体上比较直观，我在里面加了一些参数对应的描述，应该编译不过，但是会比较好理解。

这里需要说明的一点是，这个实现中的激活函数都是双曲正切，和论文中的描述不一样。当然，和论文中的模型架构也不一样，不过两者的数据集也不一样。

**判别模型的结构**

判别模型的结构如下所示，仔细地读一遍就可以理解，这里不需要赘述了。

def discriminator\_model():

model = Sequential()

model.add(Convolution2D(

64, 5, 5,

border\_mode='same',

input\_shape=(1, 28, 28)))

model.add(Activation('tanh'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Convolution2D(128, 5, 5))

model.add(Activation('tanh'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(1024))

model.add(Activation('tanh'))

model.add(Dense(1))

model.add(Activation('sigmoid'))

return model

**训练**

这里的训练的一轮迭代可以用下面的流程表示：

1 2 优化D，

1. 利用G生成一批generated\_image
2. 将真实的数据和generated\_image合并，并放入D中进行一轮训练，其中真实数据的label为1，generated\_image的label为0。

3 4 优化G

1. 利用G再生成一批generated\_image
2. 这一次将G和D连起来，并给第3步的gereated\_image的label设为1（为了以假乱真），固定D的参数不变(好像也可以变吧，这样在这一步可以同时优化D和G)，进行一轮训练。

1利用G生成一批generated\_image

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

2将真实的数据和generated\_image合并，

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

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

#上半部分是真实图片images\_train，标记为1，下半部分是生成的图片，标记为0

y[batch\_size:, :] = 0

3并放入D中进行一轮训练，

##2、训练判别器（二分类） 为了最小化损失函数d\_loss，使得判别器越来越准

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

4利用G再生成一批generated\_image

5将G和D连起来，并给第4步的gereated\_image的label设为1（为了以假乱真）固定D的参数不变，进行一轮训练。

这里4和5用一个串联好的模型adversarial代替，不过也可以自己写

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

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