model.py

初始化参数

model.py定义了DCGAN类，包括9个函数

\_\_init\_\_()

参数初始化，已讲过的input\_height, input\_width, crop, batch\_size, output\_height, output\_width, dataset\_name, input\_fname\_pattern, checkpoint\_dir, sample\_dir就不再说了

sample\_numsample\_num：大小和batch\_size一样

y\_dimy\_dim：输出通道。训练mnist数据集时，y\_dim=10训练mnist数据集时，y\_dim=10，我想可能是因为mnist是图片数字，分为10类。如果不是mnist，则默认为none。

z\_dimz\_dim：噪声z的维度，默认为100

gf\_dimgf\_dim：G第一个卷积层的过滤器个数G第一个卷积层的过滤器个数，默认为64

df\_dimdf\_dim：D第一个卷积层的过滤器个数D第一个卷积层的过滤器个数，默认为64

gfc\_dimgfc\_dim：G第一个全连接层的G单元个数G第一个全连接层的G单元个数，默认为1024

dfc\_dimdfc\_dim：D第一个全连接层的D单元个数D第一个全连接层的D单元个数，默认为1024

c\_dimc\_dim：颜色通道，灰度图像设为1，彩色图像设为3，默认为3

其中self.d\_bn1, self.d\_bn2, g\_bn0, g\_bn1, g\_bn2是batch标准化，见ops.py的batch\_norm(object)。

如果是mnist数据集，d\_bn3, g\_bn3都要batch\_norm。

self.data读取数据集。

然后建立模型(build\_model)

build\_model()

inputs的形状为[batch\_size, input\_height, input\_width, c\_dim]。

如果crop=True，inputs的形状为[batch\_size, output\_height, output\_width, c\_dim]。

输入分为样本输入inputs和抽样输入sample\_inputs。

噪声z的形状为[None, z\_dim]，第一个None是batch的大小。

然后取数据：

self.G = self.generator(self.z)#返回[batch\_size, output\_height, output\_width, c\_dim]形状的张量,也就是batch\_size张图

self.D, self.D\_logits = self.discriminator(inputs)#返回的D为是否是真样本的sigmoid概率，D\_logits是未经sigmoid处理

self.sampler = self.sampler(self.z)#相当于测试，经过G网络模型，取样，代码和G很像，没有G训练的过程。

self.D\_, self.D\_logits\_ = self.discriminator(self.G, reuse=True)

#D是真实数据，D\_是假数据

用交叉熵计算损失，共有：d\_loss\_real、d\_loss\_fake、g\_loss

self.d\_loss\_real = tf.reduce\_mean(

sigmoid\_cross\_entropy\_with\_logits(self.D\_logits, tf.ones\_like(self.D)))

self.d\_loss\_fake = tf.reduce\_mean(

sigmoid\_cross\_entropy\_with\_logits(self.D\_logits\_, tf.zeros\_like(self.D\_)))

self.g\_loss = tf.reduce\_mean(

sigmoid\_cross\_entropy\_with\_logits(self.D\_logits\_, tf.ones\_like(self.D\_)))

tf.ones\_like：新建一个与给定tensor大小一致的tensor，其全部元素为1

d\_loss\_real是真样本输入的损失，要让D\_logits接近于1，也就是D识别出真样本为真的

d\_loss\_fake是假样本输入的损失，要让D\_logits\_接近于0，D识别出假样本为假

d\_loss = d\_loss\_real + d\_loss\_fake是D的目标，要最小化这个损失

g\_loss：要让D识别假样本为真样本，G的目标是降低这个损失，D是提高这个损失

summary这几步是关于可视化，就不管了

train()

通过Adam优化器最小化d\_loss和g\_loss。

sample\_z为从-1到1均匀分布的数，大小为[sample\_num, z\_dim]

从路径中读取原始样本sample，大小为[sample\_num, output\_height, output\_width, c\_dim]

接下来进行epoch个训练：

将data总数分为batch\_idxs次训练，每次训练batch\_size个样本。产生的样本为batch\_images。

batch\_z为训练的噪声，大小为[batch\_num, z\_dim]

d\_optim = tf.train.AdamOptimizer(config.learning\_rate, beta1=config.beta1) \

.minimize(self.d\_loss, var\_list=self.d\_vars)

g\_optim = tf.train.AdamOptimizer(config.learning\_rate, beta1=config.beta1) \

.minimize(self.g\_loss, var\_list=self.g\_vars)

首先输入噪声z和batch\_images，通过优化d\_optim更新D网络。

然后输入噪声z，优化g\_optim来更新G网络。G网络更新两次，以免d\_loss为0。这点不同于paper。

这样的训练，每过100个可以生成图片看看效果。

if np.mod(counter, 100) == 1

discriminator()

代码自定义了一个conv2d，对tf.nn.conv2d稍加修改了。下面贴出tf.nn.conv2dtf.nn.conv2d解释如下：

tf.nn.conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None, name=None)

除去name参数用以指定该操作的name，与方法有关的一共五个参数：

第一个参数input：指需要做卷积的输入图像，它要求是一个Tensor，具有[batch, in\_height, in\_width, in\_channels]这样的shape，具体含义是[训练时一个batch的图片数量, 图片高度, 图片宽度, 图像通道数]，注意这是一个4维的Tensor，要求类型为float32和float64其中之一

第二个参数filter：相当于CNN中的卷积核，它要求是一个Tensor，具有[filter\_height, filter\_width, in\_channels, out\_channels]这样的shape，具体含义是[卷积核的高度，卷积核的宽度，图像通道数，卷积核个数]，要求类型与参数input相同，有一个地方需要注意，第三维in\_channels，就是参数input的第四维

第三个参数strides：卷积时在图像每一维的步长，这是一个一维的向量，长度4

第四个参数padding：string类型的量，只能是”SAME”,”VALID”其中之一，这个值决定了不同的卷积方式（后面会介绍）

第五个参数：use\_cudnn\_on\_gpu:bool类型，是否使用cudnn加速，默认为true

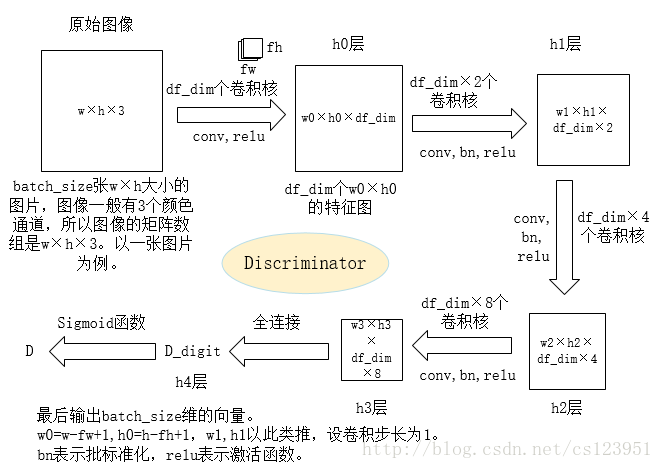
结果返回一个Tensor，这个输出，就是我们常说的feature map

batch\_norm(object)batch\_norm(object)

tf.contrib.layers.batch\_norm的代码见https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/layers/python/layers/layers.py

batchnormalization来自于http://arxiv.org/abs/1502.03167

加快训练。



激活函数lrelu见ops.py。四次卷积（其中三次卷积之前先批标准化）和激活之后。然后线性化，返回sigmoid函数处理后的结果。h3到h4的全连接相当于线性化，用一个矩阵将h3和h4连接起来，使h4是一个batch\_size维的向量。

generator()

self.h0 = tf.reshape(self.z\_, [-1, s\_h16, s\_w16, self.gf\_dim \* 8])改变z\_的形状。-1代表的含义是不用我们自己指定这一维的大小，函数会自动计算，但列表中只能存在一个-1。（当然如果存在多个-1，就是一个存在多解的方程了）

deconv2d()deconv2d()

引用tf的反卷积函数tf.nn.conv2d\_transpose或tf.nn.deconv2d。以tf.nn.conv2d\_transpose为例。

defconv2d\_transpose(value, filter, output\_shape, strides,padding=”SAME”, data\_format=”NHWC”, name=None):

value: 是一个4维的tensor，格式为[batch, height, width, in\_channels] 或者 [batch, in\_channels,height, width]。

filter: 是一个4维的tensor，格式为[height, width, output\_channels, in\_channels]，过滤器的in\_ channels的维度要和这个匹配。

output\_shape: 一维tensor，表示反卷积操作的输出shapeA

strides: 针对每个输入的tensor维度，滑动窗口的步长。

padding: “VALID”或者”SAME”，padding算法

data\_format: “NHWC”或者”NCHW” ，对应value的数据格式。

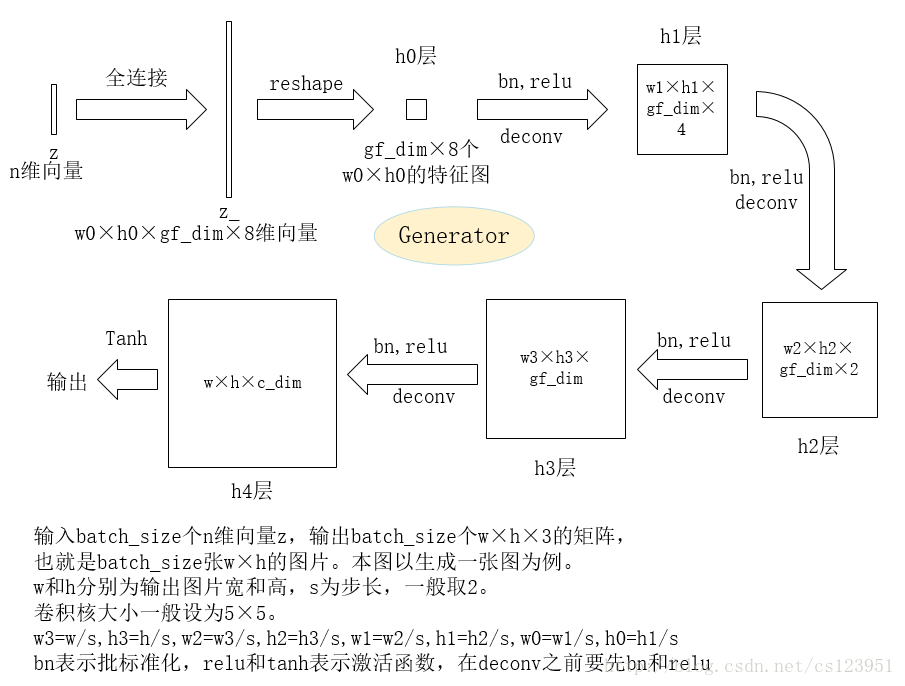
name: 可选，返回的tensor名。

deconv= tf.nn.conv2d\_transpose(input\_, w, output\_shape=output\_shape,strides=[1,d\_h, d\_w, 1])

第一个参数是输入，即上一层的结果，

第二个参数是输出输出的特征图维数，是个4维的参数，

第三个参数卷积核的移动步长，[1, d\_h, d\_w, 1]，其中第一个对应一次跳过batch中的多少图片，第二个d\_h对应一次跳过图片中多少行，第三个d\_w对应一次跳过图片中多少列，第四个对应一次跳过图像的多少个通道。这里直接设置为[1，2，2，1]。即每次反卷积后，图像的滑动步长为2，特征图会扩大缩小为原来2\*2=4倍。



sampler()

和generator结构一样，用的也是它的参数。存在的意义可能在于共享参数？

将self.sampler = self.sampler(self.z, self.y)改为self.sampler = self.generator(self.z, self.y)

参考：

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806, 2014.

Mordvintsev, Alexander, Olah, Christopher, and Tyka, Mike. Inceptionism : Going deeper into neural networks.http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html. Accessed: 2015-06-17.

Radford A, Metz L, Chintala S. UnsupervisedRepresentation Learning with Deep Convolutional Generative AdversarialNetworks[J]. Computer Science, 2015.

http://blog.csdn.net/nongfu\_spring/article/details/54342861

http://blog.csdn.net/solomon1558/article/details/52573596