* from \_*future\_* import division 这句话当python的版本为2.x时生效，可以让两个整数数字相除的结果返回一个浮点数(在python2中默认是整数,python3默认为浮点数)。glob可以以简单的正则表达式筛选的方式返回某个文件夹下符合要求的文件名列表。
* DCGAN的构造方法除了设置一大堆的属性之外，还要注意区分dataset是否是mnist,因为mnist是灰度图像,所以应该设置channel = 1( self.c\_dim = 1 ),如果是彩色图像，则 self.c\_dim = 3 or self.c\_dim = 4 。然后就是**build\_model**。
* self.generator 用于构造生成器; self.discriminator 用于构造鉴别器; self.sampler 用于随机采样(用于生成样本)。这里需要注意的是, self.y 只有当dataset是mnist的时候才不为None,不是mnist的情况下,只需要 self.z 即可生成samples。
* sigmoid\_cross\_entropy\_with\_logits 函数被重新定义了，是为了兼容不同版本的tensorflow。该函数首先使用sigmoid activation，然后计算cross-entropy loss。
* self.g\_loss 是生成器损失; self.d\_loss\_real 是真实图片的鉴别器损失; self.d\_loss\_fake 是虚假图片(由生成器生成的fake images)的损失; self.d\_loss 是总的鉴别器损失。
* 这里的 histogram\_summary 和 scalar\_summary 是为了在后续在tensorboard中对各个损失函数进行可视化。
* tf.trainable\_variables() 可以获取model的全部可训练参数,由于我们在定义生成器和鉴别器变量的时候使用了不同的name,因此我们可以通过variable的name来获取得到**self.d\_vars**(鉴别器相关变量),**self.g\_vars**(生成器相关变量)。 self.saver = tf.train.Saver() 用于保存训练好的模型参数到checkpoint。
* train 函数是核心的训练函数。这里optimizer和DCGAN的原文保持一直，选用**Adam**优化函数, lr=0.0002 , beta1=0.5 。 merge\_summary 函数和 SummaryWriter 用于构建summary,在tensorboard中显示。
* sample\_z 是从[-1,1]的均匀分布产生的。如果dataset是mnist,则可以直接读取**sample\_inputs**和**sample\_labels**。否则需要手动逐个处理图像, get\_image  
  返回的是取值为(-1,1)的,shape为(resize\_height,resize\_width)的ndarray。如果处理的图像是灰度图像,则需要再增加一个dim,表示图像的**channel=1**，对应的代码是 sample\_inputs = np.array(sample).astype(np.float32)[:, :, :, None] 。
* 接下来通过 self.sess.run([d\_optim,... 和 self.sess.run([g\_optim,...) 来更新鉴别器和生成器。 self.writer.add\_summary(summary\_str, counter) 增加summary到writer。由于同样的原因,这里仍然需要区分mnist和其他的数据集，所以计算最优化函数的过程需要一个**if**和一个**else**。
* np.mod(counter, config.print\_every) == 1 表示每*print\_every*次生成一次samples; np.mod(counter, config.checkpoint\_every) == 2 表示每*checkpoint\_every*次保存一下checkpoint file。
* 下面是discriminator(鉴别器)的具体实现。首先鉴别器使用**conv**(卷积)操作，激活函数使用**leaky-relu**,每一个layer需要使用batch normalization。tensorflow的batch normalization使用 tf.contrib.layers.batch\_norm 实现。如果不是mnist,则第一层使用**leaky-relu+conv2d**,后面三层都使用**conv2d+BN+leaky-relu**,最后加上一个one hidden unit的linear layer,再送入sigmoid函数即可；如果是mnist,则 yb = tf.reshape(y, [self.batch\_size, 1, 1, self.y\_dim]) 首先给y增加两维，以便可以和image连接起来，这里实际上是使用了conditional GAN(条件GAN)的思想。 x = conv\_cond\_concat(image, yb) 得到condition和image合并之后的结果，然后 h0 = lrelu(conv2d(x, self.c\_dim + self.y\_dim, name='d\_h0\_conv')) 进行卷积操作。第二次进行**conv2d+leaky-relu+concat**操作。第三次进行**conv2d+BN+leaky-relu+reshape+concat**操作。第四次进行**linear+BN+leaky-relu+concat**操作。最后同样是**linear+sigmoid**操作。
* 下面是generator(生成器)的具体实现。和discriminator不同的是,generator需要使用deconv(反卷积)以及relu 激活函数。generator的结构是:1.如果不是mnist:**linear+reshape+BN+relu---->(deconv+BN+relu)x3 ---->deconv+tanh**;2.如果是mnist,则除了需要考虑输入z之外，还需要考虑label y,即需要将z和y连接起来(Conditional GAN),具体的结构是:reshape+concat---->linear+BN+relu+concat---->linear+BN+relu+reshape+concat---->deconv+BN+relu+concat---->deconv+sigmoid。注意的最后的激活函数没有采用通常的tanh,而是采用了sigmoid(其输出会直接映射到0-1之间)。
* sampler函数是采样函数，用于生成样本送入当前训练的生成器，查看训练效果。其逻辑和generator函数基本类似,也是需要区分是否是mnist,二者需要采用不同的结构。不是mnist时,y=None即可；否则mnist还需要考虑y。
* load\_mnist 函数用于加载mnist数据集; save 函数用于保存checkpoint; load 函数用于加载checkpoint。