生成对抗网络 GAN 实验

一、实验目的

学习掌握生成对抗网络（GAN）和深度卷积生成对抗网络（DCGAN）和 BIGGAN 基本原理；利用 DCGAN 和 BIGGAN 进行图片生成。

二、实验内容

通过 PC 上位机连接服务器，登陆 SimpleAI 平台，熟悉并利用 DCGAN 算法进行 MNIST 数字手写体图像生成。在此基础上，通过查找资料，分析 BIGGAN 网络，分别使用 DCGAN 和 BIGGAN 两种网络进行图像生成和对比。

三、实验环境

硬件：x86_64 Centos 3.10.0 服务器/GPU 服务器、GPU、PC 上位机
软件：SimpleAI 实验平台、Docker 下 Ubuntu16.04 镜像、python3.5

四、实验原理

生成式对抗网络（GAN, Generative Adversarial Networks ）是一种深度学习模型，是近年来复杂分布上无监督学习最具前景的方法之一。模型通过框架中（至少）两个模块：生成模型（Generative Model）和判别模型（Discriminative Model）的互相博弈学习产生相当好的输出。原始 GAN 理论中，并不要求 G 和 D 都是神经网络，只需要是能拟合相应生成和判别的函数即可。但实用中一般均使用深度神经网络作为 G 和 D 。

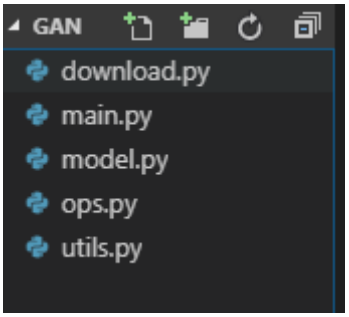
DCGAN，Deep Convolutional Generative Adversarial Networks 是生成对抗网络（Generative Adversarial Networks）的一种延伸，将卷积网络引入到生成式模型当中来做无监督的训练，利用卷积网络强大的特征提取能力来提高生成网络的学习效果。

DCGAN 算法特点:

1. 在判别器模型中使用 strided convolutions 来替代空间池化（pooling），而在生成器模型中使用 fractional strided convolutions，即 deconv，反卷积层。
2. 除了生成器模型的输出层和判别器模型的输入层，在网络其它层上都使用了 Batch Normalization，使用 BN 可以稳定学习，有助于处理初始化不良导致的训练问题。
3. 去除了全连接层，而直接使用卷积层连接生成器和判别器的输入层以及输出层。
4. 在生成器的输出层使用 Tanh 激活函数，而在其它层使用 ReLU；在判别器上使用 leaky ReLU。

五、实验步骤

1. 点击“打开数据集”下载本次实验需要的源代码，里面已经有下载好的数据集，下载后放到文件系统里。命令行进入相应目录，可以看到 data 文件夹和五个 python 文件，data 文件夹下存放本次实验所需的数据，五个 python 文件分别是 main.py，download.py，model.py，ops.py，util.py。
   1) download.py 的功能是下载数据，本次实验的数据已经下载好，不需要另外下载，代码仅供了解。
   2) main.py 是主函数，用于配置命令行参数以及模型的训练和测试。
   3) utils.py 定义很多有用的全局辅助函数。
   4) ops.py 定义了很多构造模型的重要函数，比如 batch_norm(BN 操作),conv2d(卷积操作),deconv2d(翻卷积操作)等。
   5) model.py 是定义 DCGAN 模型的地方，也是我们要重点关注的代码。



2. download.py 中定义如下所示的“download_mnist()”函数，下载 MINIST 数据集，包括图片和标签集'train-images-idx3-ubyte.gz'，'train-labels-idx1-ubyte.gz'，'t10k-images-idx3-ubyte.gz' ， 't10k-labels-idx1-ubyte.gz' 。 其 中 subprocess.call(cmd)函数是执行由参数提供的命令，此处用数组作为参数运行命令，用于数据包的下载和解压。

```
def download_mnist(dirpath):
  data_dir = os.path.join(dirpath, 'mnist')
  if os.path.exists(data_dir):
    print('Found MNIST - skip')
    return
  else:
    os.mkdir(data_dir)
  url_base = 'http://yann.lecun.com/exdb/mnist/'
  file_names = ['train-images-idx3-ubyte.gz',
                'train-labels-idx1-ubyte.gz',
                't10k-images-idx3-ubyte.gz',
                't10k-labels-idx1-ubyte.gz']
  for file_name in file_names:
    url = (url_base+file_name).format(**locals())
    print(url)
    out_path = os.path.join(data_dir,file_name)
    cmd = ['curl', url, '-o', out_path]
    print('Downloading ', file_name)
    subprocess.call(cmd)
    cmd = ['gzip', '-d', out_path]
    print('Decompressing ', file_name)
    subprocess.call(cmd)
```

3. main.py 调用前面定义好的模型、图像处理方法，来进行训练测试，程序的入口。其中使用 DCGAN 生成 MNIST 数字手写体图像，注意这里的 y_dim=10，表示 0 到 9 这 10 个类别，c_dim=1，表示灰度图像。

```
with tf.Session(config=run_config) as sess:
  if FLAGS.dataset == 'mnist':
    dcgan = DCGAN(
        sess,
        input_width=FLAGS.input_width,
        input_height=FLAGS.input_height,
        output_width=FLAGS.output_width,
        output_height=FLAGS.output_height,
        batch_size=FLAGS.batch_size,
        sample_num=FLAGS.batch_size,
        y_dim=10,
        z_dim=FLAGS.generate_test_images,
        dataset_name=FLAGS.dataset,
        input_fname_pattern=FLAGS.input_fname_pattern,
        crop=FLAGS.crop,
        checkpoint_dir=FLAGS.checkpoint_dir,
        sample_dir=FLAGS.sample_dir,
        data_dir=FLAGS.data_dir)
```

4. utils.py主要是定义了各种对图像处理的函数，相当于其他3个文件的头文件。主要实现了 3 个图像操作功能： get_image()负责读取图像，返回图像裁剪后的新图像；save_images()负责将一个 batch 中所有图像保存为一张大图像并返回；merge_images()负责翻转，返回新图像。

```
def show_all_variables():…

def get_image(image_path, input_height, input_width,
              resize_height=64, resize_width=64,
              crop=True, grayscale=False):
    image = imread(image_path, grayscale)
    return transform(image, input_height, input_width,…

def save_images(images, size, image_path):
    return imsave(inverse_transform(images), size, image_path)

def imread(path, grayscale = False):…

def merge_images(images, size):
    return inverse_transform(images)

def merge(images, size):…

def imsave(images, size, path):…

def center_crop(x, crop_h, crop_w,…

def transform(image, input_height, input_width,  …

def inverse_transform(images):…

def to_json(output_path, *layers):…

def make_gif(images, fname, duration=2, true_image=False):…

def visualize(sess, dcgan, config, option):…


def image_manifold_size(num_images):…
```

5. ops.py 主要定义了一些变量连接的函数、批处理规范化的函数、卷积函数、解卷积函数、激励函数、线性运算函数。定义一个 batch_norm 类，包含两个函数 init 和 call 函数。首先在 init 函数中，初始化变量在 call 函数中，利用 tf.contrib.layers.batch_norm 函数批处理规范化。

```python
class batch_norm(object):
  def __init__(self, epsilon=1e-5, momentum = 0.9, name="batch_norm"):
    with tf.variable_scope(name):
      self.epsilon  = epsilon
      self.momentum = momentum
      self.name = name

  def __call__(self, x, train=True):
    return tf.contrib.layers.batch_norm(x,
                        decay=self.momentum,
                        updates_collections=None,
                        epsilon=self.epsilon,
                        scale=True,
                        is_training=train,
                        scope=self.name)
```

1) 定义 conv2d 函数，即卷积函数：获取随机正态分布权值、实现卷积、获取初始偏置值，获取添加偏置值后的卷积变量并返回。

```python
def conv2d(input_, output_dim,
       k_h=5, k_w=5, d_h=2, d_w=2, stddev=0.02,
       name="conv2d"):
  with tf.variable_scope(name):
    w = tf.get_variable('w', [k_h, k_w, input_.get_shape()[-1], output_dim],
                        initializer=tf.truncated_normal_initializer(stddev=stddev))
    conv = tf.nn.conv2d(input_, w, strides=[1, d_h, d_w, 1], padding='SAME')

    biases = tf.get_variable('biases', [output_dim], initializer=tf.constant_initializer(0.0))
    conv = tf.reshape(tf.nn.bias_add(conv, biases), conv.get_shape())

    return conv
```

2) 定义 deconv2d 函数，即转置卷积函数，获取随机正态分布权值、转置卷积，获取初始偏置值，获取添加偏置值后的卷积变量，判断 with_w 是否为真，真则返回转置卷积、权值、偏置值，否则返回转置卷积。

```python
def deconv2d(input_, output_shape,
             k_h=5, k_w=5, d_h=2, d_w=2, stddev=0.02,
             name="deconv2d", with_w=False):
  with tf.variable_scope(name):
    # filter : [height, width, output_channels, in_channels]
    w = tf.get_variable('w', [k_h, k_w, output_shape[-1], input_.get_shape()[-1]],
                        initializer=tf.random_normal_initializer(stddev=stddev))

    try:
      deconv = tf.nn.conv2d_transpose(input_, w, output_shape=output_shape,
                                      strides=[1, d_h, d_w, 1])

    # Support for verisons of TensorFlow before 0.7.0
    except AttributeError:
      deconv = tf.nn.deconv2d(input_, w, output_shape=output_shape,
                              strides=[1, d_h, d_w, 1])

    biases = tf.get_variable('biases', [output_shape[-1]], initializer=tf.constant_initializer(0.0))
    deconv = tf.reshape(tf.nn.bias_add(deconv, biases), deconv.get_shape())

    if with_w:
      return deconv, w, biases
    else:
      return deconv
```

3) 定义 linear 函数，进行线性运算。

```python
def linear(input_, output_size, scope=None, stddev=0.02, bias_start=0.0, with_w=False):
  shape = input_.get_shape().as_list()

  with tf.variable_scope(scope or "Linear"):
    try:
      matrix = tf.get_variable("Matrix", [shape[1], output_size], tf.float32,
                               tf.random_normal_initializer(stddev=stddev))
    except ValueError as err:
        msg = "NOTE: Usually, this is due to an issue with the image dimensions.  Did you correctly set '--crop' or '--input_height' or '--output_height'?"
        err.args = err.args + (msg,)
        raise
    bias = tf.get_variable("bias", [output_size],
      initializer=tf.constant_initializer(bias_start))
    if with_w:
      return tf.matmul(input_, matrix) + bias, matrix, bias
    else:
      return tf.matmul(input_, matrix) + bias
```

6. model.py 定义了 DCGAN 的类，完成了生成判别网络的实现，剩余代码都是在写 DCGAN 类，所以下面几步都是在这个类里面定义进行的。DCGAN 的构造方法除了设置网络中的超参数和函数参数外，还要注意区分 dataset 是否是 MINIST，因为 MINIST 数据集是单通道灰度图像，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。

```
class DCGAN(object):
    def __init__(self, sess, input_height=108, input_width=108, crop=True, ···

    def build_model(self):···

    def train(self, config):···

    def discriminator(self, image, y=None, reuse=False):···

    def generator(self, z, y=None):···

    def sampler(self, z, y=None):···

    def load_mnist(self):···

    @property
    def model_dir(self):···

    def save(self, checkpoint_dir, step):···

    def load(self, checkpoint_dir):···
```

1) 定义构建模型函数 build_model(self)。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。定义 sigmoid 交叉熵损失函数 sigmoid_cross_entropy_with_logits(x, y)。都是调用 tf.nn.sigmoid_cross_entropy_with_logits 函数，计算训练过程和测试过程的损失。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。生成器是一个生成式网络，它接收一个随机的噪声，通过这个噪声生成图像，本次实验中生成的是手写数字图像。判别器用来判断一张图片是不是真实的，它的输入是一张图片，输出图片为真实图片的概率，如果为 1，就代表完全可能是真实的图片，如果为 0，就代表不可能是真实的图片。训练过程中，生成器的目标就是尽量生成真实的图片去欺骗判别器，而判别器的目标就是尽量辨别出生成器生成的假图像和真实的图像。这样生成器和判别器构成了一个动态的“博弈过程”，相互促进，最终达到平衡点。

```
def build_model(self):
  if self.y_dim:
    self.y = tf.placeholder(tf.float32, [self.batch_size, self.y_dim], name='y')
  else:
    self.y = None
  if self.crop:
    image_dims = [self.output_height, self.output_width, self.c_dim]
  else:
    image_dims = [self.input_height, self.input_width, self.c_dim]
  self.inputs = tf.placeholder(
    tf.float32, [self.batch_size] + image_dims, name='real_images')
  inputs = self.inputs
  self.z = tf.placeholder(
    tf.float32, [None, self.z_dim], name='z')
  self.z_sum = histogram_summary("z", self.z)
  self.G                  = self.generator(self.z, self.y)
  self.D, self.D_logits   = self.discriminator(inputs, self.y, reuse=False)
  self.sampler            = self.sampler(self.z, self.y)
  self.D_, self.D_logits_ = self.discriminator(self.G, self.y, reuse=True)
  self.d_sum = histogram_summary("d", self.D)
  self.d__sum = histogram_summary("d_", self.D_)
  self.G_sum = image_summary("G", self.G)
  def sigmoid_cross_entropy_with_logits(x, y):
    try:
      return tf.nn.sigmoid_cross_entropy_with_logits(logits=x, labels=y)
    except:
      return tf.nn.sigmoid_cross_entropy_with_logits(logits=x, targets=y)
  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_)))
  self.d_loss_real_sum = scalar_summary("d_loss_real", self.d_loss_real)
  self.d_loss_fake_sum = scalar_summary("d_loss_fake", self.d_loss_fake)
  self.d_loss = self.d_loss_real + self.d_loss_fake
  self.g_loss_sum = scalar_summary("g_loss", self.g_loss)
  self.d_loss_sum = scalar_summary("d_loss", self.d_loss)
  t_vars = tf.trainable_variables()
  self.d_vars = [var for var in t_vars if 'd_' in var.name]
  self.g_vars = [var for var in t_vars if 'g_' in var.name]
  self.saver = tf.train.Saver()
```

2) 定义训练函数 train(self, config)。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,

```
def train(self, config):
  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)
```

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。

```
sample_z = np.random.uniform(-1, 1, size=(self.sample_num , self.z_dim))

if config.dataset == 'mnist':
  sample_inputs = self.data_X[0:self.sample_num]
  sample_labels = self.data_y[0:self.sample_num]
```

开始 for epoch in xrange(config.epoch)循环训练。根据数据集是否是 mnist，获取批处理的大小。

```
for epoch in xrange(config.epoch):
  if config.dataset == 'mnist':
    batch_idxs = min(len(self.data_X), config.train_size) // config.batch_size
  else:
    self.data = glob(os.path.join(
      config.data_dir, config.dataset, self.input_fname_pattern))
    np.random.shuffle(self.data)
    batch_idxs = min(len(self.data), config.train_size) // config.batch_size
```

开始 for idx in xrange(0, batch_idxs)循环训练，判断数据集是否是 mnist，来定义初始化批处理图像和标签。然后定义初始化噪音 z。判断数据集是否是 mnist，来更新判别器网络和生成器网络，运行生成器优化器两次，以确保判别器损失值不会变为 0。

```
if config.dataset == 'mnist':
  # Update D network
  _, summary_str = self.sess.run([d_optim, self.d_sum],
    feed_dict={
      self.inputs: batch_images,
      self.z: batch_z,
      self.y:batch_labels,
    })
  self.writer.add_summary(summary_str, counter)

  # Update G network
  _, summary_str = self.sess.run([g_optim, self.g_sum],
    feed_dict={
      self.z: batch_z,
      self.y:batch_labels,
    })
  self.writer.add_summary(summary_str, counter)

  # Run g_optim twice to make sure that d_loss does not go to zero (different from paper)
  _, summary_str = self.sess.run([g_optim, self.g_sum],
    feed_dict={ self.z: batch_z, self.y:batch_labels })
  self.writer.add_summary(summary_str, counter)

  errD_fake = self.d_loss_fake.eval({
      self.z: batch_z,
      self.y:batch_labels
  })
  errD_real = self.d_loss_real.eval({
      self.inputs: batch_images,
      self.y:batch_labels
  })
  errG = self.g_loss.eval({
      self.z: batch_z,
      self.y: batch_labels
  })
```

每 100 次 batch 训练后，根据数据集是否是 mnist 的不同，获取样本、判别器损失值、生成器损失值，调用 utils.py 文件的 save_images 函数，保存训练后的样本，并以 epoch、batch 的次数命名文件。然后打印判别器损失值和生成器损失值。每 500 次 batch 训练后，保存一次检查点。

```
if np.mod(counter, 100) == 1:
  if config.dataset == 'mnist':
    samples, d_loss, g_loss = self.sess.run(
      [self.sampler, self.d_loss, self.g_loss],
      feed_dict={
          self.z: sample_z,
          self.inputs: sample_inputs,
          self.y:sample_labels,
      }
    )
    save_images(samples, image_manifold_size(samples.shape[0]),
          './{}/train_{:02d}_{:04d}.png'.format(config.sample_dir, epoch, idx))
    print("[Sample] d_loss: %.8f, g_loss: %.8f" % (d_loss, g_loss))
  else:
    try:
      samples, d_loss, g_loss = self.sess.run(
        [self.sampler, self.d_loss, self.g_loss],
        feed_dict={
            self.z: sample_z,
            self.inputs: sample_inputs,
        },
      )
      save_images(samples, image_manifold_size(samples.shape[0]),
            './{}/train_{:02d}_{:04d}.png'.format(config.sample_dir, epoch, idx))
      print("[Sample] d_loss: %.8f, g_loss: %.8f" % (d_loss, g_loss))
    except:
      print("one pic error!...")

if np.mod(counter, 500) == 2:
  self.save(config.checkpoint_dir, counter)
```

3) 定义判别器函数。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。

```
def discriminator(self, image, y=None, reuse=False):
  with tf.variable_scope("discriminator") as scope:
    if reuse:
      scope.reuse_variables()

    if not self.y_dim:
      h0 = lrelu(conv2d(image, self.df_dim, name='d_h0_conv'))
      h1 = lrelu(self.d_bn1(conv2d(h0, self.df_dim*2, name='d_h1_conv')))
      h2 = lrelu(self.d_bn2(conv2d(h1, self.df_dim*4, name='d_h2_conv')))
      h3 = lrelu(self.d_bn3(conv2d(h2, self.df_dim*8, name='d_h3_conv')))
      h4 = linear(tf.reshape(h3, [self.batch_size, -1]), 1, 'd_h4_lin')

      return tf.nn.sigmoid(h4), h4
    else:
      yb = tf.reshape(y, [self.batch_size, 1, 1, self.y_dim])
      x = conv_cond_concat(image, yb)

      h0 = lrelu(conv2d(x, self.c_dim + self.y_dim, name='d_h0_conv'))
      h0 = conv_cond_concat(h0, yb)

      h1 = lrelu(self.d_bn1(conv2d(h0, self.df_dim + self.y_dim, name='d_h1_conv')))
      h1 = tf.reshape(h1, [self.batch_size, -1])
      h1 = concat([h1, y], 1)

      h2 = lrelu(self.d_bn2(linear(h1, self.dfc_dim, 'd_h2_lin')))
      h2 = concat([h2, y], 1)

      h3 = linear(h2, 1, 'd_h3_lin')

      return tf.nn.sigmoid(h3), h3
```

4) 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)。

```python
def generator(self, z, y=None):
  with tf.variable_scope("generator") as scope:
    if not self.y_dim:
      s_h, s_w = self.output_height, self.output_width
      s_h2, s_w2 = conv_out_size_same(s_h, 2), conv_out_size_same(s_w, 2)
      s_h4, s_w4 = conv_out_size_same(s_h2, 2), conv_out_size_same(s_w2, 2)
      s_h8, s_w8 = conv_out_size_same(s_h4, 2), conv_out_size_same(s_w4, 2)
      s_h16, s_w16 = conv_out_size_same(s_h8, 2), conv_out_size_same(s_w8, 2)
      # project `z` and reshape
      self.z_, self.h0_w, self.h0_b = linear(
          z, self.gf_dim*8*s_h16*s_w16, 'g_h0_lin', with_w=True)
      self.h0 = tf.reshape(self.z_, [-1, s_h16, s_w16, self.gf_dim * 8])
      h0 = tf.nn.relu(self.g_bn0(self.h0))
      self.h1, self.h1_w, self.h1_b = deconv2d(
          h0, [self.batch_size, s_h8, s_w8, self.gf_dim*4], name='g_h1', with_w=True)
      h1 = tf.nn.relu(self.g_bn1(self.h1))
      h2, self.h2_w, self.h2_b = deconv2d(
          h1, [self.batch_size, s_h4, s_w4, self.gf_dim*2], name='g_h2', with_w=True)
      h2 = tf.nn.relu(self.g_bn2(h2))
      h3, self.h3_w, self.h3_b = deconv2d(
          h2, [self.batch_size, s_h2, s_w2, self.gf_dim*1], name='g_h3', with_w=True)
      h3 = tf.nn.relu(self.g_bn3(h3))
      h4, self.h4_w, self.h4_b = deconv2d(
          h3, [self.batch_size, s_h, s_w, self.c_dim], name='g_h4', with_w=True)
      return tf.nn.tanh(h4)
    else:
      s_h, s_w = self.output_height, self.output_width
      s_h2, s_h4 = int(s_h/2), int(s_h/4)
      s_w2, s_w4 = int(s_w/2), int(s_w/4)
      # yb = tf.expand_dims(tf.expand_dims(y, 1),2)
      yb = tf.reshape(y, [self.batch_size, 1, 1, self.y_dim])
      z = concat([z, y], 1)
      h0 = tf.nn.relu(
          self.g_bn0(linear(z, self.gfc_dim, 'g_h0_lin')))
      h0 = concat([h0, y], 1)
      h1 = tf.nn.relu(self.g_bn1(
          linear(h0, self.gf_dim*2*s_h4*s_w4, 'g_h1_lin')))
      h1 = tf.reshape(h1, [self.batch_size, s_h4, s_w4, self.gf_dim * 2])
      h1 = conv_cond_concat(h1, yb)
      h2 = tf.nn.relu(self.g_bn2(deconv2d(h1,
          [self.batch_size, s_h2, s_w2, self.gf_dim * 2], name='g_h2')))
      h2 = conv_cond_concat(h2, yb)
      return tf.nn.sigmoid(
          deconv2d(h2, [self.batch_size, s_h, s_w, self.c_dim], name='g_h3'))
```

7. 训练

1) 点击“打开数据集”下载实验源码，注意保持数据文件夹 data 与 py 文件的位置关系及 data 文件夹内的目录结构，命令行进入相应目录

2) 训练过程中需要依赖 Pillow 库，实验平台没有安装，需要自己手动安装。

pip3 install -i https://pypi.douban.com/simple/ Pillow

3) 命令行下输入 python main.py --dataset mnist --input_height=28 --output_height=28 --train，使用 mnist 训练 DCGAN。预设训练 25 轮（main.py 11 行），同学们可以根据实际训练速度进行调整，使得整个训练时间不至于太长。训练过程中采样得到的生成图片保存在 samples 文件夹下，第一次采样和最后一次采样得到图片分别为下图 1 和图 2 所示。

4) python main.py --dataset mnist --input_height=28 --output_height=28，用训练好的模型进行测试

8. 实验结果

第 1 个 epoch 第 99 个 batch：

第 13 个 epoch 第 483 个 batch：

第 25 个 epoch 第 1067 个 batch：

六、扩展实验

1）进一步分析代码，理解 DCGAN 结构，特别是熟悉构建生成网络和判别网络的过程。用自己感兴趣的数据集（比如汽车、飞机或者人物等）进行训练和

测试。

2）GAN 变体 BIGGAN 在图片生成方面有巨大的进展，可生成几乎以假乱真的图片。分析和使用 BIGGAN（https://github.com/AaronLeong/BigGAN-pytorch）对扩展实验 1）的图像进行训练和测试，并对两个算法结果进行对比分析。