XSam's Blog Documentation

发布 ***1.0***

Sam

2018 年 **11** 月 **21** 日

Contents:

1 前言 **1**

2 第一章 python基础知识 **3**
- 2.1 1.1 python简介 . . . 3
- 2.2 1.2 Anaconda的下载与安装 . . . 4
- 2.3 1.3 python基础语法 . . . 4

3 第二章 深度学习与TensorFlow简介 **5**
- 3.1 2.1 什么是深度学习 . . . 5
- 3.2 2.2 TensorFlow . . . 5

4 第三章 TensorFlow入门 **7**

5 第四章 卷积神经网络 **9**

6 第五章 生成对抗网络 **11**
- 6.1 5.1 生成对抗网络简介 . . . 11
- 6.2 5.2 GAN . . . 14
- 6.3 5.3 DCGAN . . . 21
- 6.4 5.4 LSGAN . . . 30
- 6.5 5.5 WGAN . . . 30
- 6.6 5.6 CycleGAN . . . 30
- 6.7 5.7 CGAN . . . 30
- 6.8 5.8 StarGAN . . . 30

CHAPTER 1

前言

记录自己深度学习的学习过程。

CHAPTER 2

第一章 python基础知识

- *1.1 python*简介
- *1.2 Anaconda*的下载与安装
- *1.3 python*基础语法

2.1 1.1 python简介

Python是著名的“龟叔”Guido van Rossum在1989年圣诞节期间，为了打发无聊的圣诞节而编写的一个编程语言。

Python就为我们提供了非常完善的基础代码库，覆盖了网络、文件、GUI、数据库、文本等大量内容，被形象地称作“内置电池（batteries included）”。用Python开发，许多功能不必从零编写，直接使用现成的即可。除了内置的库外，Python还有大量的第三方库，也就是别人开发的，供你直接使用的东西。当然，如果你开发的代码通过很好的封装，也可以作为第三方库给别人使用。许多大型网站就是用Python开发的，例如YouTube、Instagram，还有国内的豆瓣。很多大公司，包括Google、Yahoo等，甚至NASA（美国航空航天局）都大量地使用Python。龟叔给Python的定位是“优雅”、“明确”、“简单”，所以Python程序看上去总是简单易懂，初学者学Python，不但入门容易，而且将来深入下去，可以编写那些非常非常复杂的程序。

总的来说，Python的哲学就是简单优雅，尽量写容易看明白的代码，尽量写少的代码。如果一个资深程序员向你炫耀他写的晦涩难懂、动不动就几万行的代码，你可以尽情地嘲笑他。

那Python适合开发哪些类型的应用呢?

1. 网络应用，包括网站、后台服务等等;
2. 许多日常需要的小工具，包括系统管理员需要的脚本任务等等;
3. 把其他语言开发的程序再包装起来，方便使用。

最后说说Python的缺点。任何编程语言都有缺点，Python也不例外。优点说过了，那Python有哪些缺点呢?

1. 运行速度慢，和C程序相比非常慢，因为Python是解释型语言，你的代码在执行时会一行一行地翻译成CPU能理解的机器码，这个翻译过程非常耗时，所以很慢。而C程序是运行前直接编译成CPU能执行的机器码，所以非常快。但是大量的应用程序不需要这么快的运行速度，因为用户根本感觉不出来。例如开发一个下载MP3的网络应用程序，C程序的运行时间需要0.001秒，而Python程序的运行时间需要0.1秒，慢了100倍，但由于网络更慢，需要等待1秒，你想，用户能感觉到1.001秒和1.1秒的区别吗？这就好比F1赛车和普通的出租车在北京三环路上行驶的道理一样，虽然F1赛车理论时速高达400公里，但由于三环路堵车的时速只有20公里，因此，作为乘客，你感觉的时速永远是20公里。
2. 代码不能加密。如果要发布你的Python程序，实际上就是发布源代码，这一点跟C语言不同，C语言不用发布源代码，只需要把编译后的机器码（也就是你在Windows上常见的xxx.exe文件）发布出去。要从机器码反推出C代码是不可能的，所以，凡是编译型的语言，都没有这个问题，而解释型的语言，则必须把源码发布出去。

这个缺点仅限于你要编写的软件需要卖给别人挣钱的时候。好消息是目前的互联网时代，靠卖软件授权的商业模式越来越少了，靠网站和移动应用卖服务的模式越来越多了，后一种模式不需要把源码给别人。

再说了，现在如火如荼的开源运动和互联网自由开放的精神是一致的，互联网上有无数非常优秀的像Linux一样的开源代码，我们千万不要高估自己写的代码真的有非常大的"商业价值"。那些大公司的代码不愿意开放的更重要的原因是代码写得太烂了，一旦开源，就没人敢用他们的产品了。

2.2 1.2 Anaconda的下载与安装

Anaconda。

2.3 1.3 python基础语法

python基础语法

CHAPTER 3

第二章 深度学习与TensorFlow简介

- *2.1* 什么是深度学习
- *2.2 TensorFlow*

3.1 2.1 什么是深度学习

提到人工智能，人们往往会想到深度学习，然而，深度学习不像人工智能那样容易从字面上理解。这是因为深度学习是从内部机理来阐述的，而人工智能是从其应用的角度来阐述的，即深度学习是实现人工智能的一种方法。

人工智能领域，起初是进行神经网络的研究。但神经网络发展到一定阶段后，模型越来越庞大，结构也越来越复杂，于是人们将其命名为“深度学习”。可以这样理解————深度学习属于后神经网络时代。

深度学习近年来的发展突飞猛进，越来越多的人工智能应用得以实现。其本质为一个可以模拟人脑进行分析、学习的神经网络，它模仿人脑的机制来解释数据（如图像、声音和文本），通过组合低层特征，形成更加抽象的高层特征或属性类别，来拟合人们日常生活中的各种事情。

深度学习被广泛应用于与人们生活息息相关的各种领域，可以实现机器翻译、人脸识别、语音识别、信号恢复、商业推荐、金融分析、医疗辅助和智能交通等。

在国内乃至世界，越来越多的资金涌向人工智能领域，人工智能领域新成立的创业公司每年呈递增趋势，越来越多的学校也开始开设与深度学习相关的课程。这个时代，正像是移动互联网的前夜。

3.2 2.2 TensorFlow

CHAPTER 4

第三章 TensorFlow入门

CHAPTER 5

第四章 卷积神经网络

CHAPTER 6

第五章 生成对抗网络

- *5.1* 生成对抗网络简介
- *5.2 GAN*
- *5.3 DCGAN*
- *5.4 LSGAN*
- *5.5 WGAN*
- *5.6 CycleGAN*
- *5.7 CGAN*
- *5.8 StarGAN*

6.1 5.1 生成对抗网络简介

6.1.1 1.简介

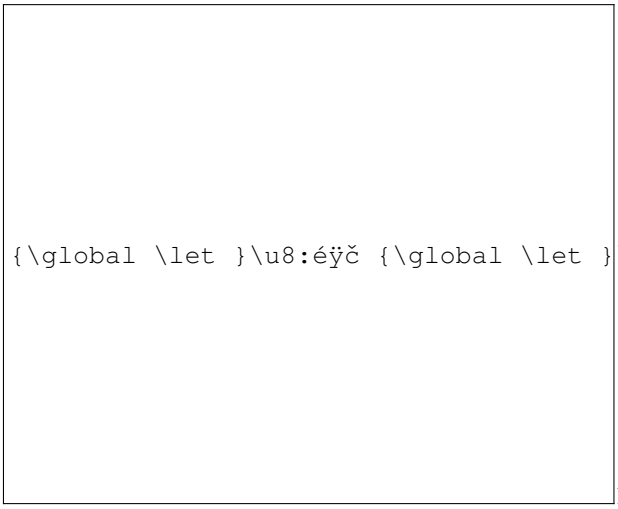
生成对抗网络(GAN, Generative Adversial Network)是生成模型的一种，主要用于通过数据分布生成样本。

GAN之父Ian J. Goodfellow等人于2014年10月在 *Generative Adversarial Networks* 中提出了一个通过对抗过程估计生成模型的新框架。框架中同时训练两个模型：捕获数据分布的生成模型G，和估计样本来自训练数据的概率的判别模型D。G的训练程序是将D错误的概率最大化。这个框架对应一个最大值集下限的双方对抗游戏。可以证明在任意函数G和D的空间中，存在唯一的解决方案，使得G重现训练数据分布，而D=0.5 。在G和D由多层感知器定义的情况下，整个系统可以用反向传播进行训练。在训练或生成样本期间，不需要任何马尔科夫链或展开的近似推理网络。实验通过对生成的样品的定性和定量评估证明了本框架的潜力。

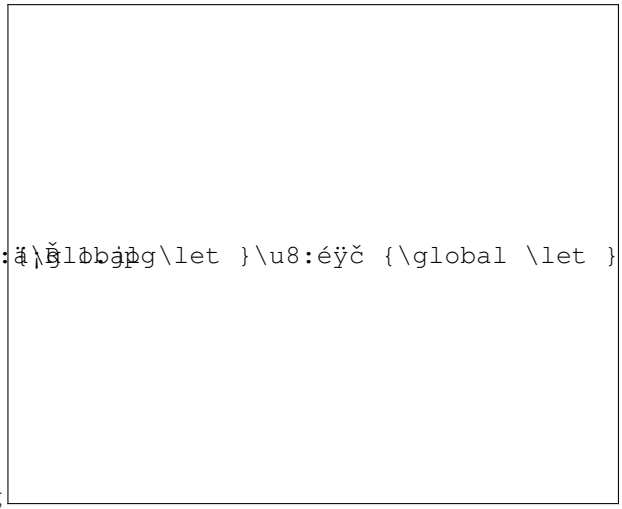
学习生成模型的理由：

1. 生成样本，这是最直接的理由。
2. 训练并不包含 最大似然估计 (MLE, Maximum Likelihood Estimation)。
3. 生成器对训练数据不可见，过拟合的风险较低。
4. GAN十分擅长捕获模式的分布。

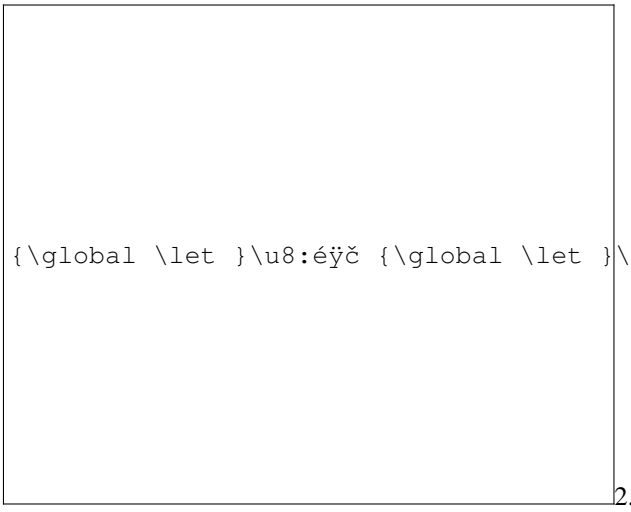
GAN描述的是一种"读作敌人，写作朋友"的观念，就好比Naruto和Sasuke。



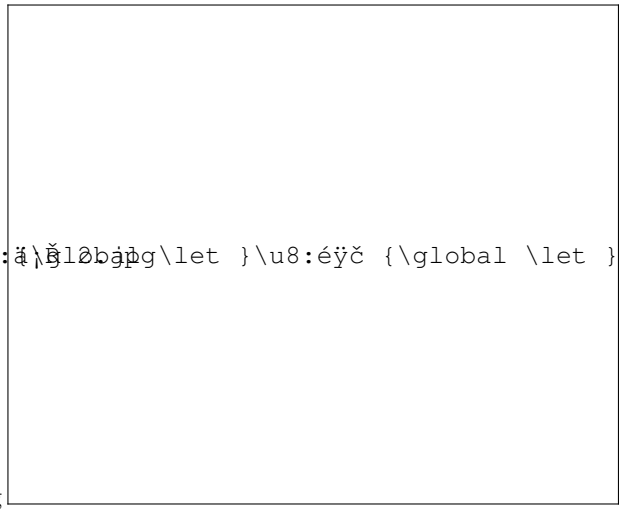
1.jpg



1.jpg



2.jpg



2.jpg

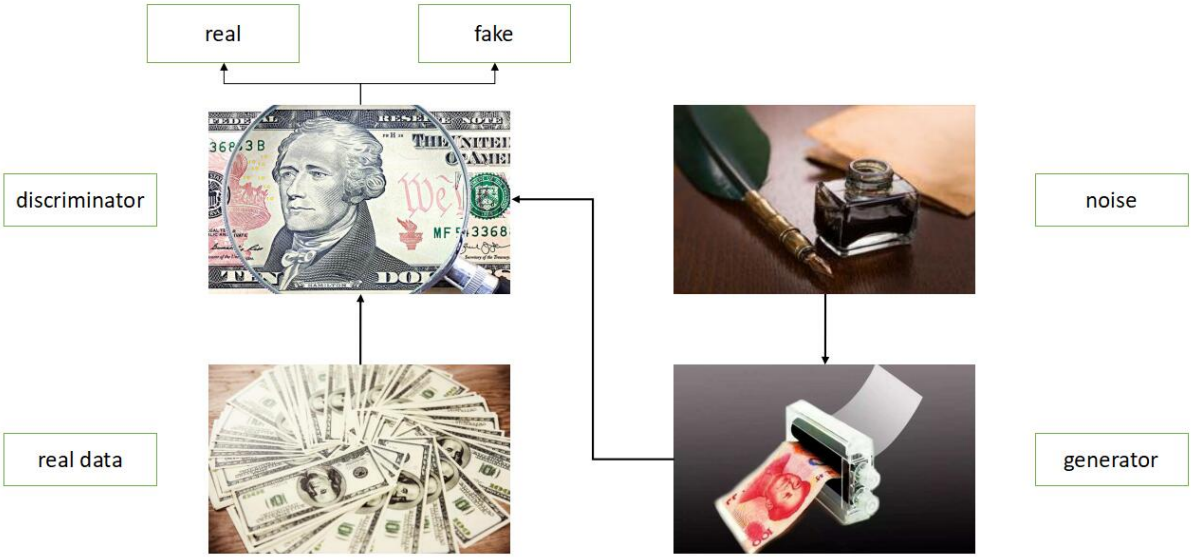
6.1.2 2.原理

GAN原始论文 给出了一个制假钞的例子。

犯罪嫌疑人与警察对于钞票的关注点：

1. 想要成为一名成功的伪钞制作者，犯罪嫌疑人需要可以混淆警察，使得警察无法分辨真钞与伪钞。

2. 警察需要尽可能高效地分辨钞票的真伪。



这个场景表现为博弈论中的极大极小博弈，整个过程被称为对抗性过程(Adversarial Process)。GAN是一种由两个神经网络相互竞争的特殊对抗过程。第一个网络生成数据，第二个网络试图区分真实数据与第一个网络创造出来的假数据。第二个网络会生成一个在[0, 1]区间内的标量，代表数据是真实数据的概率。

在GAN中，第一个网络通常被称为生成器(Generator)并且以G(z)表示，第二个网络通常被称为判别器(Discriminator)并且以D(x)表示。

在平衡点，也就是极大极小博弈的最优点，生成器生成数据，判别器认为生成器生成的数据是真实数据的概率为0.5，整个过程可以用如下公式表示:

$$min_G max_D V(D,G) = E_{x \sim p_{data}(x)}[logD(x)] + E_{x \sim p_Z(z)}[log(1 - D(G(z)))]$$

在一些情况下这两个网络可以达到平衡点，但是在另一些情况下却不能，两个网络会继续学习很长时间。

- 生成器

生成器网络以随机的噪声作为输入并试图生成样本数据。通常，生成器G(z)从概率分布p(z)中接收输入z，并且生成数据提供给判别器网络D(x)。

- 判别器

判别器网络以真实数据或生成器生成的数据为输入，并试图预测当前输入是真实数据还是生成数据。其中一个输入x从真实的数据分布pdata(x)中获取，接下来解决一个二分类问题，产生一个在[0, 1]区间内的标量。

由于真实世界中无标记的数据量远远大于标记数据，而GAN十分擅长无监督学习任务，因此近些年来GAN变得越来越流行。它得以流行的另一个原因在于在多种生成模型中，GAN可以生成最为逼真的图像。尽管这是一个很主观的评价，但这已经成为所有从业者的共识。

此外，GAN还有着强大的表达能力：它可以在潜在空间(向量空间)内执行算数运算，并将其转换为对应特征空间内的运算。如，在潜在空间内有一张戴眼镜男人的照片，减去一个神经网络中男人的向量，再加上一个神经网络中女人的向量，最后会得到特征空空间内一张戴眼镜女人的图像。这种表达能力的确令人叹为观止。

6.2 5.2 GAN

1. GAN论文：https://arxiv.org/abs/1406.2661
2. MNIST数据集下载：http://yann.lecun.com/exdb/mnist

   下载后，将t10k-images-idx3-ubyte.gz，t10k-labels-idx1-ubyte.gz，train-images-idx3-ubyte.gz与train-labels-idx1-ubyte.gz四个压缩包放在项目所在文件夹的"MNIST_data"文件夹中。
3. GAN的实现：

一共四个文件，ops.py，discriminator.py，generator.py以及gan.py。

一. 定义初始文件结构，ops.py，discriminator.py，generator.py以及gan.py。

① ops.py

```
import tensorflow as tf
import numpy as np
import tensorflow.contrib.slim as slim
import scipy

# Help function for linear function
def linear(x, output_size, name='linear', stddev=0.02):
    shape = x.get_shape().as_list()
    with tf.variable_scope(name):
        matrix = tf.get_variable("matrix", [shape[1], output_size], tf.float32, tf.
↪random_normal_initializer(stddev=stddev))
        bias = tf.get_variable("bias", [output_size], initializer=tf.constant_
↪initializer(0.0))
        output = tf.matmul(x, matrix) + bias
        return output

# Help function for relu
def relu(z):
    return tf.nn.relu(z)

# Help function for flatten
def flatten(z, name='flatten'):
    return tf.layers.flatten(z, name=name)

# Help function for dense layer
def dense(z, units=1, activation=None, name='dense'):
    return tf.layers.dense(z, units=units, activation=activation, name=name)

# Help function for sigmoid
```

(continues on next page)

(续上页)

```
def sigmoid(z):
    return tf.nn.sigmoid(z)

# Help function for printing trainable_variables
def show_all_variables():
  model_vars = tf.trainable_variables()
  slim.model_analyzer.analyze_vars(model_vars, print_info=True)

# Help function for reading image
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, resize_height, resize_width,␣
↪crop)

# Help function for saving images
def save_images(images, size, image_path):
  return imsave(inverse_transform(images), size, image_path)

# Help function for imread
def imread(path, grayscale=False):
  if (grayscale):
    return scipy.misc.imread(path, flatten=True).astype(np.float)
  else:
    return scipy.misc.imread(path).astype(np.float)

# Help function for merging images
def merge_images(images, size):
  return inverse_transform(images)

# Help function for merging
def merge(images, size):
  h, w = images.shape[1], images.shape[2]
  if (images.shape[3] in (3,4)):
    c = images.shape[3]
    img = np.zeros((h * size[0], w * size[1], c))
    for idx, image in enumerate(images):
      i = idx % size[1]
      j = idx // size[1]
      img[j * h:j * h + h, i * w:i * w + w, :] = image
    return img
  elif images.shape[3]==1:
    img = np.zeros((h * size[0], w * size[1]))
    for idx, image in enumerate(images):
      i = idx % size[1]
      j = idx // size[1]
      img[j * h:j * h + h, i * w:i * w + w] = image[:,:,0]
    return img
  else:
    raise ValueError('in merge(images,size) images parameter must have dimensions:␣
↪HxW or HxWx3 or HxWx4')

# Help function for imsave
def imsave(images, size, path):
  image = np.squeeze(merge(images, size))
  return scipy.misc.imsave(path, image)
```

(continues on next page)

(续上页)

```
81
82  # Help function for center_crop
83  def center_crop(x, crop_h, crop_w, resize_h=64, resize_w=64):
84    if crop_w is None:
85      crop_w = crop_h
86    h, w = x.shape[:2]
87    j = int(round((h - crop_h)/2.))
88    i = int(round((w - crop_w)/2.))
89    return scipy.misc.imresize(x[j:j+crop_h, i:i+crop_w], [resize_h, resize_w])
90
91  # Help function for transform
92  def transform(image, input_height, input_width, resize_height=64, resize_width=64,␣
    ↪crop=True):
93    if crop:
94      cropped_image = center_crop(image, input_height, input_width, resize_height,␣
    ↪resize_width)
95    else:
96      cropped_image = scipy.misc.imresize(image, [resize_height, resize_width])
97    return np.array(cropped_image)/127.5 - 1.
98
99  # Help function for inverse_transform
100 def inverse_transform(images):
101   return (images + 1.) / 2.
102
103 # Help function for calculating image_manifold_size
104 def image_manifold_size(num_images):
105   manifold_h = int(np.floor(np.sqrt(num_images)))
106   manifold_w = int(np.ceil(np.sqrt(num_images)))
107   assert manifold_h * manifold_w == num_images
108   return manifold_h, manifold_w
```

② discriminator.py

```
1  import tensorflow as tf
2  import numpy as np
3  from ops import *
4
5  class Discriminator:
6      def __init__(self):
7          pass
8
9      def forward(self, x, reuse=False, name='discriminator'):
10         pass
```

③ generator.py

```
1  import tensorflow as tf
2  import numpy as np
3  from ops import *
4
5  class Generator:
6      def __init__(self):
7          pass
8
9      def forward(self, x, reuse=False, name='generator'):
10         pass
```

④ gan.py

```
import tensorflow as tf
import numpy as np
import os
import glob
import time
from ops import *
from random import shuffle
from discriminator import Discriminator
from generator import Generator
from tensorflow.examples.tutorials.mnist import input_data

class GAN:
    def __init__(self, img_shape, train_folder, sample_folder, model_folder, grayscale=False, crop=True, iterations=10000, lr_dis=0.0002, lr_gen=0.0002, beta1=0.5, batch_size=64, z_shape=100, sample_interval=100):
        pass

    def train(self):
        pass

    def load_dataset(self):
        mnist = input_data.read_data_sets("./MNIST_data/", one_hot=True)
        return mnist

if __name__ == '__main__':
    pass
```

二. 逐步完善discriminator.py，generator.py以及gan.py。

① discriminator.py

```
import tensorflow as tf
import numpy as np
from ops import *

class Discriminator:
    def __init__(self):
        pass

    def forward(self, x, reuse=False, name='discriminator'):
        with tf.variable_scope(name, reuse=reuse):
            z = tf.reshape(x, [-1, 28 * 28 * 1])

            # Layer1
            z = linear(z, output_size=128, name='d_linear_1')
            z = relu(z)

            # Layer2
            z = linear(z, output_size=1, name='d_lineaer_2')
            z = sigmoid(z)

            return z
```

② generator.py

```python
import tensorflow as tf
import numpy as np
from ops import *

class Generator:
    def __init__(self):
        pass

    def forward(self, x, reuse=False, name='generator'):
        with tf.variable_scope(name, reuse=reuse):
            # Layer1
            z = linear(x, output_size=128, name='g_linear_1')
            z = relu(z)

            # Layer2
            z = linear(z, output_size=28 * 28 * 1, name='g_linear_2')
            z = tf.reshape(z, [-1, 28, 28, 1])
            z = sigmoid(z)

            return z
```

③ gan.py

```python
import tensorflow as tf
import numpy as np
import os
import glob
import time
from ops import *
from random import shuffle
from discriminator import Discriminator
from generator import Generator
from tensorflow.examples.tutorials.mnist import input_data

class GAN:
    def __init__(self, train_folder, sample_folder, model_folder, grayscale=False, crop=True, iterations=10000, lr_dis=0.0002, lr_gen=0.0002, beta1=0.5, batch_size=64, z_shape=100, sample_interval=100):
        if not os.path.exists(train_folder):
            print("Invalid dataset path.")
            return
        if not os.path.exists(sample_folder):
            os.makedirs(sample_folder)
        if not os.path.exists(model_folder):
            os.makedirs(model_folder)

        self.height, self.width, self.channel = img_shape
        self.train_folder = train_folder
        self.sample_folder = sample_folder
        self.model_folder = model_folder
        self.grayscale = grayscale
        self.crop = crop
        self.iterations = iterations
        self.lr_dis = lr_dis
        self.lr_gen = lr_gen
        self.beta1 = beta1
        self.batch_size = batch_size
```

(continues on next page)

(续上页)

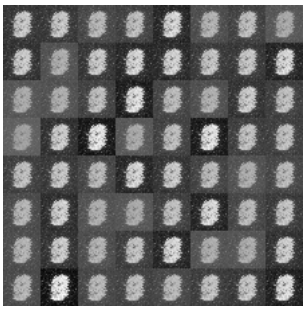
```
        self.z_shape = z_shape
        self.sample_interval = sample_interval
        self.discriminator = Discriminator()
        self.generator = Generator()

        # load dataset
        self.X = self.load_dataset()

        # placeholders
        self.phX = tf.placeholder(tf.float32, [self.batch_size, self.height, self.width, self.channel], name='phX')
        self.phZ = tf.placeholder(tf.float32, [self.batch_size, self.z_shape], name='phZ')

        # forward
        self.gen_out = self.generator.forward(self.phZ, reuse=False)
        self.dis_real = self.discriminator.forward(self.phX, reuse=False)
        self.dis_fake = self.discriminator.forward(self.gen_out, reuse=True)
        self.sampler = self.generator.forward(self.phZ, reuse=True)

        # loss
        self.d_loss = -tf.reduce_mean(tf.log(self.dis_real) + tf.log(1. - self.dis_fake))
        self.g_loss = -tf.reduce_mean(tf.log(self.dis_fake))

        # vars
        train_vars = tf.trainable_variables()
        self.dis_vars = [var for var in train_vars if 'discriminator' in var.name]
        self.gen_vars = [var for var in train_vars if 'generator' in var.name]

        # optimizer
        self.dis_train = tf.train.AdamOptimizer(self.lr_dis, beta1=beta1).minimize(self.d_loss, var_list=self.dis_vars)
        self.gen_train = tf.train.AdamOptimizer(self.lr_gen, beta1=beta1).minimize(self.g_loss, var_list=self.gen_vars)

    def train(self):
        self.sess = tf.Session()
        self.sess.run(tf.global_variables_initializer())

        saver = tf.train.Saver(max_to_keep=1)
        savedir = self.model_folder

        sample_z = np.random.uniform(-1, 1, size=(self.batch_size, self.z_shape))

        for epoch in range(self.iterations):
            batch_X, _ = self.X.train.next_batch(self.batch_size)
            batch_X = np.reshape(batch_X, [-1, 28, 28, 1])
            batch_Z = np.random.uniform(-1, 1, size=(self.batch_size, self.z_shape))
            _, d_loss = self.sess.run([self.dis_train, self.d_loss], feed_dict={self.phX: batch_X, self.phZ: batch_Z})

            batch_Z = np.random.uniform(-1, 1, size=(self.batch_size, self.z_shape))
            _, g_loss = self.sess.run([self.gen_train, self.g_loss], feed_dict={self.phZ: batch_Z})

```

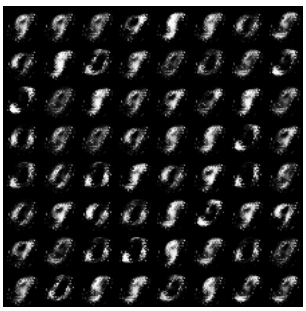
(continues on next page)

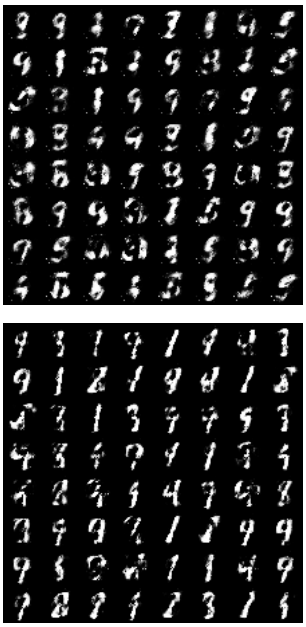
(续上页)

```
            if epoch % 100 == 0:
                print("Epoch: {}. D_loss: {}. G_loss: {}".format(epoch, d_loss, g_
↪loss))

                samples = self.sess.run(self.sampler, feed_dict={self.phZ: sample_z})
                save_images(samples, image_manifold_size(samples.shape[0]), '{}/{}.png
↪'.format(self.sample_folder, epoch))
                saver.save(self.sess, "{}/gan.ckpt".format(self.model_folder), global_
↪step=epoch)

    def load_dataset(self):
        mnist = input_data.read_data_sets("./MNIST_data/", one_hot=True)
        return mnist

if __name__ == '__main__':
    img_shape = (28, 28, 1)
    train_folder = "MNIST_data"
    sample_folder = "samples"
    model_folder = "models"
    gan = GAN(train_folder=train_folder, sample_folder=sample_folder, model_
↪folder=model_folder)
    gan.train()
```

三. 结果







注解： 从左到右分别是0 epoch，3100 epoch，6100 epoch，9100 epoch时的测试结果。

注解： 运行结果

6.3 5.3 DCGAN

1. DCGAN论文：https://arxiv.org/pdf/1511.06434

2. 何之源 提供的Anime faces数据集下载：https://pan.baidu.com/share/init?surl=eSifHcA 提取码：g5qa
3. DCGAN的实现：

一共四个文件，ops.py，discriminator.py，generator.py以及dcgan.py。

一. 定义初始文件结构，ops.py，discriminator.py，generator.py以及dcgan.py。

① ops.py

```
import tensorflow as tf
import tensorflow.contrib.slim as slim
import scipy
import numpy as np

# Help function for creating convolutional layers
def conv2d(x, output_dim, filter_size=5, stride=2, padding='SAME', stddev=0.02, name=
↪'conv2d'):
    input_dim = x.get_shape().as_list()[-1]
    with tf.variable_scope(name):
        filter = tf.get_variable('w', [filter_size, filter_size, input_dim, output_
↪dim], initializer=tf.truncated_normal_initializer(stddev=stddev))
        conv = tf.nn.conv2d(x, filter=filter, strides=[1, stride, stride, 1],␣
↪padding=padding)
        bias = tf.get_variable('bias', [output_dim], initializer=tf.constant_
↪initializer(0.0))
        conv = tf.nn.bias_add(conv, bias)
        return conv

# Help function for creating deconvolutional layers
def deconv2d(x, filter_size, output_dim, stride, padding='SAME', stddev=0.02, name=
↪'deconv2d'):
    params = x.get_shape().as_list()
    batch_size = params[0]
    width = params[1]
    height = params[2]
    input_dim = params[-1]
    with tf.variable_scope(name):
        filter = tf.get_variable('w', [filter_size, filter_size, output_dim, input_
↪dim], initializer=tf.random_normal_initializer(stddev=stddev))
        deconv = tf.nn.conv2d_transpose(x, filter=filter, output_shape=[batch_size,␣
↪width * stride, height * stride, output_dim], strides=[1, stride, stride, 1],␣
↪padding=padding)
        bias = tf.get_variable('bias', [output_dim], initializer=tf.constant_
↪initializer(0.0))
        return deconv

def linear(x, output_size, name='linear', stddev=0.02):
    shape = x.get_shape().as_list()
    with tf.variable_scope(name):
        matrix = tf.get_variable("matrix", [shape[1], output_size], tf.float32, tf.
↪random_normal_initializer(stddev=stddev))
        bias = tf.get_variable("bias", [output_size], initializer=tf.constant_
↪initializer(0.0))
        output = tf.matmul(x, matrix) + bias
```

(continues on next page)

(续上页)

```
        return output

# Help function for batch_norm
def batch_norm(x, train=True, momentum=0.9, epsilon=1e-5, name="batch_norm"):
    return tf.contrib.layers.batch_norm(x, decay=momentum, updates_collections=None, epsilon=epsilon, scale=True, is_training=train, scope=name)

# Help function for relu
def relu(z):
    return tf.nn.relu(z)

# Help function for leaky_relu
def leaky_relu(z):
    return tf.nn.leaky_relu(z)

# Help function for sigmoid
def sigmoid(z):
    return tf.nn.sigmoid(z)

# Help function for tanh
def tanh(z):
    return tf.nn.tanh(z)

# Help function for residual block - identity
def identity_block(X, filter_sizes, output_dims, strides, stage, block, reuse=False, trainable=True):
    block_name = 'res_identity_' + str(stage) + '_' + block

    with tf.variable_scope(block_name, reuse=reuse):
        X_shortcut = X

        z = conv2d(X, filter_size=filter_sizes[0], output_dim=output_dims[0], stride=strides[0], padding='SAME', name=block_name + "_conv1")
        z = batch_norm(z, name=block_name + "_bn1", train=trainable)
        z = relu(z)

        z = conv2d(z, filter_size=filter_sizes[1], output_dim=output_dims[1], stride=strides[1], padding='SAME', name=block_name + "_conv2")
        z = batch_norm(z, name=block_name + "_bn2", train=trainable)
        z = relu(z)

        z = conv2d(z, filter_size=filter_sizes[2], output_dim=output_dims[2], stride=strides[2], padding='SAME', name=block_name + "_conv3")
        z = batch_norm(z, name=block_name + "_bn3", train=trainable)

        z = tf.add(X_shortcut, z)
        z = relu(z)
        return z

# Help function for printing trainable_variables
def show_all_variables():
  model_vars = tf.trainable_variables()
  slim.model_analyzer.analyze_vars(model_vars, print_info=True)

# Help function for reading image
def get_image(image_path, input_height, input_width, resize_height=64, resize_width=64, crop=True, grayscale=False):
```

(continues on next page)

(续上页)

```
  image = imread(image_path, grayscale)
  return transform(image, input_height, input_width, resize_height, resize_width, crop)

# Help function for saving images
def save_images(images, size, image_path):
  return imsave(inverse_transform(images), size, image_path)

# Help function for imread
def imread(path, grayscale=False):
  if (grayscale):
    return scipy.misc.imread(path, flatten=True).astype(np.float)
  else:
    return scipy.misc.imread(path).astype(np.float)

# Help function for merging images
def merge_images(images, size):
  return inverse_transform(images)

# Help function for merging
def merge(images, size):
  h, w = images.shape[1], images.shape[2]
  if (images.shape[3] in (3,4)):
    c = images.shape[3]
    img = np.zeros((h * size[0], w * size[1], c))
    for idx, image in enumerate(images):
      i = idx % size[1]
      j = idx // size[1]
      img[j * h:j * h + h, i * w:i * w + w, :] = image
    return img
  elif images.shape[3]==1:
    img = np.zeros((h * size[0], w * size[1]))
    for idx, image in enumerate(images):
      i = idx % size[1]
      j = idx // size[1]
      img[j * h:j * h + h, i * w:i * w + w] = image[:,:,0]
    return img
  else:
    raise ValueError('in merge(images,size) images parameter must have dimensions: HxW or HxWx3 or HxWx4')

# Help function for imsave
def imsave(images, size, path):
  image = np.squeeze(merge(images, size))
  return scipy.misc.imsave(path, image)

# Help function for center_crop
def center_crop(x, crop_h, crop_w, resize_h=64, resize_w=64):
  if crop_w is None:
    crop_w = crop_h
  h, w = x.shape[:2]
  j = int(round((h - crop_h)/2.))
  i = int(round((w - crop_w)/2.))
  return scipy.misc.imresize(x[j:j+crop_h, i:i+crop_w], [resize_h, resize_w])

# Help function for transform
```

(continues on next page)

(续上页)

```
def transform(image, input_height, input_width, resize_height=64, resize_width=64,␣
↪crop=True):
  if crop:
    cropped_image = center_crop(image, input_height, input_width, resize_height,␣
↪resize_width)
  else:
    cropped_image = scipy.misc.imresize(image, [resize_height, resize_width])
  return np.array(cropped_image)/127.5 - 1.

# Help function for inverse_transform
def inverse_transform(images):
  return (images + 1.) / 2.

# Help function for calculating image_manifold_size
def image_manifold_size(num_images):
  manifold_h = int(np.floor(np.sqrt(num_images)))
  manifold_w = int(np.ceil(np.sqrt(num_images)))
  assert manifold_h * manifold_w == num_images
  return manifold_h, manifold_w

# Help function for cost function
def cost(logits, labels):
    return tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=logits,␣
↪labels=labels))
```

② discriminator.py

```
import tensorflow as tf
import numpy as np
from ops import *

class Discriminator:
    def __init__(self):
        pass

    def forward(self, x, momentum=0.9, df_dim=32, trainable=True, name='discriminator
↪', reuse=False):
        pass
```

③ generator.py

```
import tensorflow as tf
import numpy as np
from ops import *

class Generator:
    def __init__(self, img_shape):
        self.width, self.height, self.channels = img_shape

    def forward(self, x, momentum=0.9, gf_dim=64, trainable=True, name='generator',␣
↪reuse=False):
        pass
```

④ dcgan.py

```
import tensorflow as tf
```

(continues on next page)

(续上页)

```
import numpy as np
import os
import glob
import time
from ops import *
from generator import Generator
from discriminator import Discriminator
from random import shuffle

class DCGAN:
    def __init__(self, input_shape, output_shape, train_folder, sample_folder, model_folder, grayscale=False, crop=True, iterations=500, lr_dis=0.0002, lr_gen=0.0002, beta1=0.5, batch_size=64, z_shape=128, sample_interval=100):
        pass

    def load_dataset(self):
        x_imgs_name = glob.glob(os.path.join(self.train_folder, '*'))
        return x_imgs_name

    def train(self):
        pass

if __name__ == '__main__':
    pass
```

二. 逐步完善discriminator.py，generator.py以及dcgan.py。

① discriminator.py

```
import tensorflow as tf
import numpy as np
from ops import *

class Discriminator:
    def __init__(self):
        pass

    def forward(self, x, momentum=0.9, df_dim=32, trainable=True, name='discriminator', reuse=False):
        with tf.variable_scope(name) as scope:
            if reuse:
                scope.reuse_variables()

            # Layer1
            z = leaky_relu(conv2d(x, filter_size=5, output_dim=df_dim, stride=2, padding='SAME', name='d_conv_1'))
            # Layer2
            # z = leaky_relu(batch_normalization(conv2d(z, filter_size=5, output_dim=df_dim * 2, stride=2, padding='SAME', name='d_conv_2'), trainable=trainable, momentum=momentum, name='d_bn_2'))
            z = leaky_relu(batch_norm(conv2d(z, filter_size=5, output_dim=df_dim * 2, stride=2, padding='SAME', name='d_conv_2'), train=trainable, name='d_bn_2'))
            # Layer3
            # z = leaky_relu(batch_normalization(conv2d(z, filter_size=5, output_dim=df_dim * 4, stride=2, padding='SAME', name='d_conv_3'), trainable=trainable, momentum=momentum, name='d_bn_3'))
```

(continues on next page)

(续上页)

```
            z = leaky_relu(batch_norm(conv2d(z, filter_size=5, output_dim=df_dim * 4, stride=2, padding='SAME', name='d_conv_3'), train=trainable, name='d_bn_3'))
            # Layer4
            # z = leaky_relu(batch_normalization(conv2d(z, filter_size=5, output_dim=df_dim * 8, stride=2, padding='SAME', name='d_conv_4'), trainable=trainable, momentum=momentum, name='d_bn_4'))
            z = leaky_relu(batch_norm(conv2d(z, filter_size=5, output_dim=df_dim * 8, stride=2, padding='SAME', name='d_conv_4'), train=trainable, name='d_bn_4'))
            # Layer5
            z = tf.reshape(z, [x.get_shape().as_list()[0], -1])
            z = linear(z, output_size=1, name='d_linear_5')
            return sigmoid(z), z
```

② generator.py

```
import tensorflow as tf
import numpy as np
from ops import *

class Generator:
    def __init__(self, img_shape):
        self.width, self.height, self.channels = img_shape

    def forward(self, x, momentum=0.9, gf_dim=64, trainable=True, name='generator', reuse=False):
        with tf.variable_scope(name) as scope:
            if reuse:
                scope.reuse_variables()
            # Layer1
            w = self.width // (2 ** 4)
            h = self.height // (2 ** 4)
            z = linear(x, output_size=gf_dim * 8 * w * h, name='g_linear_1')
            z = tf.reshape(z, [-1, w, h, gf_dim * 8])
            z = relu(batch_norm(z, train=trainable, name='g_bn_1'))

            # Layer2
            z = relu(deconv2d(z, filter_size=5, output_dim=gf_dim * 4, stride=2, padding='SAME', name='g_deconv_2'))

            # Layer3
            z = relu(batch_norm(deconv2d(z, filter_size=5, output_dim=gf_dim * 2, stride=2, padding='SAME', name='g_deconv_3'), train=trainable, momentum=momentum, name='g_bn_3'))

            # Layer4
            z = relu(batch_norm(deconv2d(z, filter_size=5, output_dim=gf_dim * 1, stride=2, padding='SAME', name='g_deconv_4'), train=trainable, momentum=momentum, name='g_bn_3'))

            # Layer5
            z = relu(batch_norm(deconv2d(z, filter_size=5, output_dim=gf_dim // 2, stride=2, padding='SAME', name='g_deconv_5'), train=trainable, momentum=momentum, name='g_bn_3'))

            # Layer6
            z = conv2d(z, filter_size=7, output_dim=self.channels, stride=1, padding='SAME', name='g_conv_6')
```

(continues on next page)

(续上页)

```
            return tanh(z)
```

③ dcgan.py

```
import tensorflow as tf
import numpy as np
import os
import glob
import time
from ops import *
from generator import Generator
from discriminator import Discriminator
from random import shuffle

class DCGAN:
    def __init__(self, input_shape, output_shape, train_folder, sample_folder, model_folder, grayscale=False, crop=True, iterations=500, lr_dis=0.0002, lr_gen=0.0002, beta1=0.5, batch_size=64, z_shape=128, sample_interval=100):
        if not os.path.exists(train_folder):
            print("Invalid dataset path.")
            return
        if not os.path.exists(sample_folder):
            os.makedirs(sample_folder)
        if not os.path.exists(model_folder):
            os.makedirs(model_folder)

        self.in_width, self.in_height, self.in_channels = input_shape
        self.out_width, self.out_height, self.out_channels = output_shape
        self.train_folder = train_folder
        self.sample_folder = sample_folder
        self.model_folder = model_folder
        self.grayscale = grayscale
        self.crop = crop
        self.iterations = iterations
        self.lr_dis = lr_dis
        self.lr_gen = lr_gen
        self.beta1 = beta1
        self.batch_size = batch_size
        self.z_shape = z_shape
        self.sample_interval = sample_interval
        self.discriminator = Discriminator()
        self.generator = Generator(output_shape)

        # load dataset
        self.X = self.load_dataset()

        # placeholders
        self.phX = tf.placeholder(tf.float32, [self.batch_size, self.out_width, self.out_height, self.out_channels], name='phX')
        self.phZ = tf.placeholder(tf.float32, [self.batch_size, self.z_shape], name='phZ')

        # forward
        self.gen_out = self.generator.forward(self.phZ, reuse=False)
        self.dis_real, self.dis_real_logits = self.discriminator.forward(self.phX, reuse=False)
```

(continues on next page)

(续上页)

```
        self.dis_fake, self.dis_fake_logits = self.discriminator.forward(self.gen_out, reuse=True)
        self.sampler = self.generator.forward(self.phZ, reuse=True, trainable=False)

        # loss
        self.dis_loss_real = cost(self.dis_real_logits, tf.ones_like(self.dis_real))
        self.dis_loss_fake = cost(self.dis_fake_logits, tf.zeros_like(self.dis_fake))
        self.d_loss = self.dis_loss_fake + self.dis_loss_real
        self.g_loss = cost(self.dis_fake_logits, tf.ones_like(self.dis_fake))

        # vars
        train_vars = tf.trainable_variables()
        self.dis_vars = [var for var in train_vars if 'discriminator' in var.name]
        self.gen_vars = [var for var in train_vars if 'generator' in var.name]

        # optimizer
        self.dis_train = tf.train.AdamOptimizer(self.lr_dis, beta1=beta1).minimize(self.d_loss, var_list=self.dis_vars)
        self.gen_train = tf.train.AdamOptimizer(self.lr_gen, beta1=beta1).minimize(self.g_loss, var_list=self.gen_vars)

    def load_dataset(self):
        x_imgs_name = glob.glob(os.path.join(self.train_folder, '*'))
        return x_imgs_name

    def train(self):
        run_config = tf.ConfigProto()
        run_config.gpu_options.allow_growth = True
        self.sess = tf.Session(config=run_config)
        self.sess.run(tf.global_variables_initializer())

        saver = tf.train.Saver(max_to_keep=1)
        savedir = self.model_folder

        counter = 0

        sample_z = np.random.uniform(-1, 1, size=(self.batch_size, self.z_shape))
        sample_files = self.X[0: self.batch_size]
        sample = [get_image(sample_file, input_height=self.in_height, input_width=self.in_width, resize_height=self.out_height, resize_width=self.out_width, crop=self.crop, grayscale=self.grayscale) for sample_file in sample_files]
        if (self.grayscale):
            sample_inputs = np.array(sample).astype(np.float32)[:, :, :, None]
        else:
            sample_inputs = np.array(sample).astype(np.float32)

        start_time = time.time()
        for i in range(self.iterations):
            batch_idxs = len(self.X) // self.batch_size
            shuffle(self.X)
            for idx in range(batch_idxs):
                batch_files = self.X[idx * self.batch_size: (idx + 1) * self.batch_size]
                batch_X = [get_image(batch_file, input_height=self.in_height, input_width=self.in_width, resize_height=self.out_height, resize_width=self.out_width, crop=self.crop, grayscale=self.grayscale) for batch_file in batch_files]
```

(continues on next page)

(续上页)

```
                if self.grayscale:
                    batch_X = np.array(batch_X).astype(np.float32)[:, :, :, None]
                else:
                    batch_X = np.array(batch_X).astype(np.float32)

                # batch_X = self.read_img(self.X)
                batch_Z = np.random.uniform(-1, 1, (self.batch_size, self.z_shape)).astype(np.float32)
                _, d_loss = self.sess.run([self.dis_train, self.d_loss], feed_dict={self.phX: batch_X, self.phZ: batch_Z})
                _, g_loss = self.sess.run([self.gen_train, self.g_loss], feed_dict={self.phZ: batch_Z})

                print("Epoch:{} {}/{}. Time: {}. Discriminator loss: {}. Generator loss: {}".format(i, idx, batch_idxs, time.time() - start_time, d_loss, g_loss))
                if counter % self.sample_interval == 0:
                    samples, d_loss, g_loss = self.sess.run([self.sampler, self.d_loss, self.g_loss], feed_dict={self.phZ: sample_z, self.phX: sample_inputs})
                    save_images(samples, image_manifold_size(samples.shape[0]), '{}/{}.png'.format(self.sample_folder, counter))
                    saver.save(self.sess, "{}/dcgan.ckpt".format(self.model_folder), global_step=counter)
                counter += 1

if __name__ == '__main__':
    input_shape = (96, 96, 3)
    output_shape = (96, 96, 3)
    train_folder = "D:/Project/Python_Project/21 deep learning/chapter_8/faces/"
    sample_folder = "samples"
    model_folder = "models"
    dcgan = DCGAN(input_shape=input_shape, output_shape=output_shape, train_folder=train_folder, sample_folder=sample_folder, model_folder=model_folder)
    dcgan.train()
```

三. 结果

6.4 5.4 LSGAN

6.5 5.5 WGAN

6.6 5.6 CycleGAN

6.7 5.7 CGAN

6.8 5.8 StarGAN