**这个文件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。**

import math

import numpy as np

import tensorflow as tf

from tensorflow.python.framework import ops

**首先导入tensorflow.python.framework模块，包含了tensorflow中图、张量等的定义操作**

from utils import \*

**第9行到第20行的代码是为了保持tf0.x和tf1.x版本的兼容性。tf0.x版本使用tf.xxx\_summary风格的函数，而tf1.x版本则使用tf.summary.xxx风格的函数。为了保持一致性，通过重命名统一成tf.xxx\_summary风格了**

try:

image\_summary = tf.image\_summary

scalar\_summary = tf.scalar\_summary

histogram\_summary = tf.histogram\_summary

merge\_summary = tf.merge\_summary

SummaryWriter = tf.train.SummaryWriter

except:

image\_summary = tf.compat.v1.summary.image

scalar\_summary = tf.compat.v1.summary.scalar

histogram\_summary = tf.compat.v1.summary.histogram

merge\_summary = tf.compat.v1.summary.merge

SummaryWriter = tf.compat.v1.summary.FileWriter

**22行到27行重新定义了concat函数，也是为了兼容性考虑, if "concat\_v2" in dir(tf): 这句话是说如果tf有concat\_v2这个方法的话，tf0.x中使用concat\_v2函数，而tf1.x版本中使用concat函数**

if "concat\_v2" in dir(tf):

def concat(tensors, axis, \*args, \*\*kwargs):

return tf.concat\_v2(tensors, axis, \*args, \*\*kwargs)

else:

def concat(tensors, axis, \*args, \*\*kwargs):

return tf.concat(tensors, axis, \*args, \*\*kwargs)

**定义了batch\_norm类。需要注意的是37-44行定义了类的\_\_call\_\_特殊方法，这个方法的作用是可以将类像普通的函数那样直接调用，而不用先构造一个对象再调用方法，这是常用的一个技巧。tf中的batch normalization 是函数 tf.contrib.layers.batch\_norm**

class batch\_norm(object):

**定义一个batch\_norm类，包含两个函数init和call函数。首先在init(self, epsilon=1e-5, momentum = 0.9, name=”batch\_norm”)函数中，定义一个name参数名字的变量，初始化self变量epsilon、momentum 、name。在call(self, x, train=True)函数中，利用tf.contrib.layers.batch\_norm函数批处理规范化**

def \_\_init\_\_(self, epsilon=1e-5, momentum = 0.9, name="batch\_norm"):

with tf.compat.v1.variable\_scope(name):

self.epsilon = epsilon

self.momentum = momentum

self.name = name

**定义了class 的\_\_call\_\_ 方法，可以把类像函数一样调用**

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)

**函数的作用是将conv(卷积)和cond(条件)concat起来。在~~mnist的generator~~和discriminator中会用到定义conv\_cond\_concat(x,y)函数。**

**连接x,y与Int32型的[x\_shapes[0], x\_shapes[1], x\_shapes[2], y\_shapes[3]]维度的张量乘积**

def conv\_cond\_concat(x, y):

"""Concatenate conditioning vector on feature map axis."""

x\_shapes = x.get\_shape()

y\_shapes = y.get\_shape()

**沿axis = 3(最后一个维度连接)**

return concat([

x, y\*tf.ones([x\_shapes[0], x\_shapes[1], x\_shapes[2], y\_shapes[3]])], 3)

**conv2d函数重新定义了卷积操作，主要是封装了 tf.nn.conv2d 函数**

**义conv2d(input\_, output\_dim, k\_h=5, k\_w=5, d\_h=2,d\_w=2, stddev=0.02,name=”conv2d”)函数。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**

def conv2d(input\_, output\_dim,

k\_h=5, k\_w=5, d\_h=2, d\_w=2, stddev=0.02,

name="conv2d"):

with tf.compat.v1.variable\_scope(name):

w = tf.compat.v1.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.compat.v1.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

**定义了deconv2d(反卷积)函数。tf0.x的反卷积函数为 tf.nn.deconv2d ,tf1.x的反卷积函数为 tf.nn.conv2d\_transpose 。最后还加上了一个bias( tf.nn.bias\_add )**

**定义deconv2d(input\_, output\_shape,k\_h=5, k\_w=5, d\_h=2, d\_w=2, stddev=0.02,name=”deconv2d”, with\_w=False):函数。解卷积函数：获取随机正态分布权值、解卷积，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，真则返回解卷积、权值、偏置值，否则返回解卷积**

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.compat.v1.variable\_scope(name):

# filter : [height, width, output\_channels, in\_channels]

w = tf.compat.v1.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.compat.v1.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

**定义了leaky-relu函数lrelu。其实就一行代码: tf.maximum(x, leak\*x)**

**定义lrelu(x, leak=0.2, name=”lrelu”)函数。定义一个lrelu激励函数**

def lrelu(x, leak=0.2, name="lrelu"):

return tf.maximum(x, leak\*x)

**定义了linear函数，其实就是一个fully\_connected layer**

**定义linear(input\_, output\_size, scope=None, stddev=0.02, bias\_start=0.0, with\_w=False)函数。进行线性运算，获取一个随机正态分布矩阵，获取初始偏置值，如果with\_w为真，则返回xw+b，权值w和偏置值b；否则返回xw+b**

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.compat.v1.variable\_scope(scope or "Linear"):

try:

matrix = tf.compat.v1.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.compat.v1.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