### os.environ[**'TF\_CPP\_MIN\_LOG\_LEVEL'**]=**'0'**

**//** 大多中文博客：LOG 等级说明

import os  
os.environ[‘TF\_CPP\_MIN\_LOG\_LEVEL‘]=‘1‘ # 这是默认的显示等级，显示所有信息  
  
# 2级  
import os  
os.environ[‘TF\_CPP\_MIN\_LOG\_LEVEL‘]=‘2‘ # 只显示 warning 和 Error  
  
# 3级  
import os  
os.environ[‘TF\_CPP\_MIN\_LOG\_LEVEL‘]=‘3‘ # 只显示 Error

<https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error> :

* 0 = all messages are logged (default behavior)
* 1 = INFO messages are not printed
* 2 = INFO and WARNING messages are not printed
* 3 = INFO, WARNING, and ERROR messages are not printed

### Argparse模块最简单易懂，工程性介绍：

<https://www.jianshu.com/p/fef2d215b91d>

### Python的namedtuple使用详解

namedtuple是继承自tuple的子类。namedtuple创建一个和tuple类似的对象，而且对象拥有可访问的属性。

相比tuple，dictionary，namedtuple略微有点综合体的意味：直观、使用方便，墙裂建议大家在合适的时候多用用namedtuple。

### TensorFlow中global\_step的简单分析

global\_step在滑动平均、优化器、指数衰减学习率等方面都有用到，这个变量的实际意义非常好理解：代表全局步数，比如在多少步该进行什么操作，现在神经网络训练到多少轮等等，类似于一个钟表。

global\_step=tf.Variable(0, trainable=**False**)

### 读取文件【待填空】

**def** count\_text\_lines(file\_path):  
 f = open(file\_path, **'r'**)  
 lines = f.readlines()  
 f.close()  
 **return** len(lines)

**with** open(**'xm\_crct\_1.out'**,**'r'**) **as** f:  
 **while True**:  
 ff = f.readline()  
 search\_epochs = re.search(**r'[e][p][o][c][h][ ]+\d+.\d+'**,ff)  
 search\_loss = re.search(**r'[l][o][s][s][ ]+\d+.\d+'**,ff)  
 search\_lr = re.search(**r'lr 0.\d+'**,ff)  
  
 **if** search\_epochs **is not None**:  
 search\_epochs = search\_epochs.group()[6:]  
 train\_epochs.append(float(search\_epochs))  
  
 search\_loss = search\_loss.group()[5:]  
 train\_loss.append(float(search\_loss))  
  
 search\_lr = search\_lr.group()[3:]  
 train\_lr.append(float(search\_lr))  
  
 **if not** ff:  
 **break**

### Np.ceil()

* np.ceil: 计算比每一个元素大或相等的最小的整数.

### tf.train.exponential\_decay( ) & tf.train.piecewise\_constant( )

<https://blog.csdn.net/ghy_111/article/details/80591314>

根据boundaries分阶段，每阶段value由values指定的分段常数

例：前100000steps学习率为0.1，100000~110000steps学习率为0.01，剩余steps为0.001

### tensorflow的数据输入

<https://blog.csdn.net/zzk1995/article/details/54292859>

tensorflow有两种数据输入方法，比较简单的一种是使用feed\_dict，这种方法在画graph的时候使用placeholder来站位，在真正run的时候通过feed字典把真实的输入传进去。比较简单不再介绍。

比较恼火的是第二种方法，直接从文件中读取数据（其实第一种也可以我们自己从文件中读出来之后使用feed\_dict传进去，但方法二tf提供很完善的一套类和函数形成一个类似pipeline一样的读取线）：   
1.使用tf.train.string\_input\_producer函数把我们需要的全部文件打包为一个tf内部的queue类型，之后tf开文件就从这个queue中取目录了，要注意一点的是这个函数的shuffle参数默认是True，也就是你传给他文件顺序是1234，但是到时候读就不一定了，我一开始每次跑训练第一次迭代的样本都不一样，还纳闷了好久，就是这个原因。

2.搞一个reader，不同reader对应不同的文件结构，比如度bin文件tf.FixedLengthRecordReader就比较好，因为每次读等长的一段数据。如果要读什么别的结构也有相应的reader。

reader = tf.FixedLengthRecordReader(record\_bytes=1+32\*32\*3)

1

3.用reader的read方法，这个方法需要一个IO类型的参数，就是我们上边string\_input\_producer输出的那个queue了，reader从这个queue中取一个文件目录，然后打开它经行一次读取，reader的返回是一个tensor（这一点很重要，我们现在写的这些读取代码并不是真的在读数据，还是在画graph，和定义神经网络是一样的，这时候的操作在run之前都不会执行，这个返回的tensor也没有值，他仅仅代表graph中的一个结点）。

key, value = reader.read(files)

1

4.对这个tensor做些数据与处理，比如CIFAR1-10中label和image数据是糅在一起的，这里用slice把他们切开，切成两个tensor（注意这个两个tensor是对应的，一个image对一个label，对叉了后便训练就完了），然后对image的tensor做data augmentation。

data = tf.decode\_raw(value, tf.uint8)

label = tf.cast(tf.slice(data, [0], [1]), tf.int64)

raw\_image = tf.reshape(tf.slice(data, [1], [32\*32\*3]), [3, 32, 32])

image = tf.cast(tf.transpose(raw\_image, [1, 2, 0]), tf.float32)

lr\_image = tf.image.random\_flip\_left\_right(image)

br\_image = tf.image.random\_brightness(lr\_image, max\_delta=63)

rc\_image = tf.image.random\_contrast(br\_image, lower=0.2, upper=1.8)

std\_image = tf.image.per\_image\_standardization(rc\_image)

5.这时候可以发现，这个tensor代表的是一个样本（[高宽管道]），但是训练网络的时候的输入一般都是一推样本（[样本数高宽\*管道]），我们就要用tf.train.batch或者tf.train.shuffle\_batch这个函数把一个一个小样本的tensor打包成一个高一维度的样本batch，这些函数的输入是单个样本，输出就是4D的样本batch了，其内部原理似乎是创建了一个queue，然后不断调用你的单样本tensor获得样本，直到queue里边有足够的样本，然后一次返回一堆样本，组成样本batch。

images, labels = tf.train.batch([std\_image, label],

batch\_size=100,

num\_threads=16,

capacity=int(50000\* 0.4 + 3 \* batch\_size))

1

2

3

4

5.事实上一直到上一部的images这个tensor，都还没有真实的数据在里边，我们必须用Session run一下这个4D的tensor，才会真的有数据出来。这个原理就和我们定义好的神经网络run一下出结果一样，你一run这个4D tensor，他就会顺着自己的operator找自己依赖的其他tensor，一路最后找到最开始reader那里。