# Tensorflow API

## 1.tf.argmax(vector,1)

tf.argmax(vector, 1)：返回的是矩阵或者向量中的最大值的索引号，如果vector是一个向量，那就返回一个值，如果是一个矩阵，那就返回一个向量。

argmax()声明：

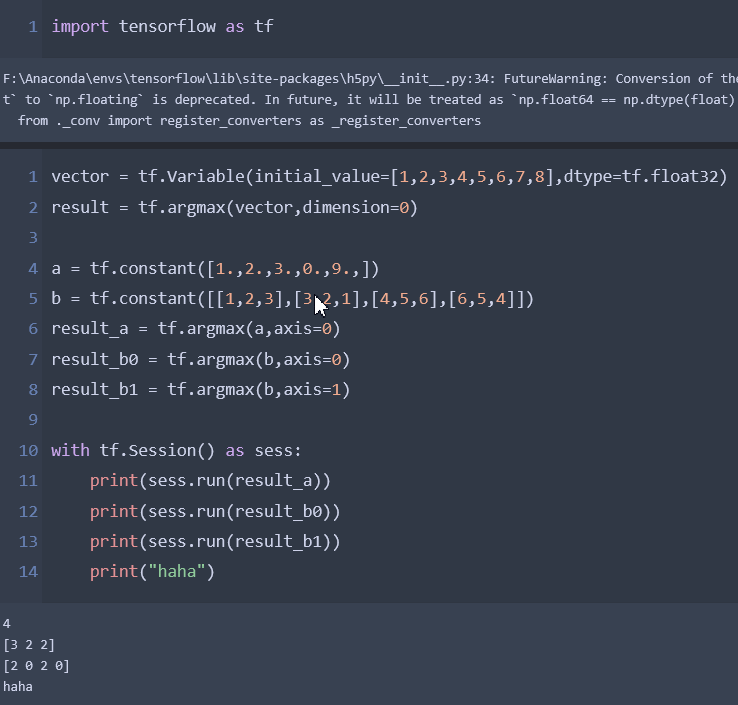
tf.argmax(input, dimension, name=None)

Returns: A Tensor of type int64.

当axis或者dimension=0 按列找

当axis或者dimension=1 按行找

示例：



## 2.tf.nn.softmax\_cross\_entropy\_with\_logits

TensorFlow中常用的求交叉熵的函数。

### 函数原型

tf.nn.softmax\_cross\_entropy\_with\_logits(

\_sentinel=None,

labels=None,

logits=None,

dim=-1,

name=None

)

这个函数的功能就是计算labels和logits之间的交叉熵（cross entropy）。

### 参数说明

第一个参数基本不用。此处不说明。

第二个参数label的含义就是一个分类标签，所不同的是，这个label是分类的概率，比如说[0.2,0.3,0.5]，labels的每一行必须是一个概率分布，就是说和为1。

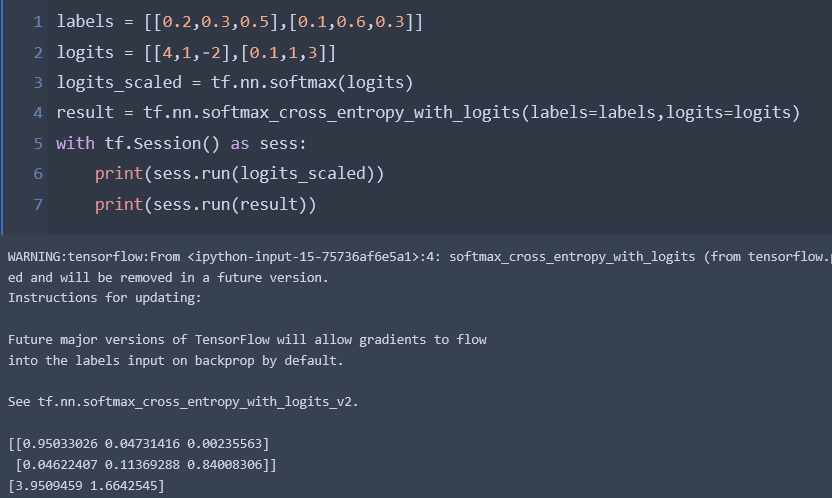
第三个参数logits，为原生态的、未经缩放的，可视为一种未归一化的log 概率，如是[4, 1, -2]。

### 该函数做的工作如下

（1）Softmax，把logits中的值从[-inf, +inf] 映射到[0,1]，并把所有参与映射的值累计之和等于1，变成诸如[0.95, 0.05, 0]的概率向量。这样一来，经过Softmax加工的数据可以当做概率来用。

（2）计算交叉熵。logits归一化后得到的概率向量与labels

### 示例



### 需要注意的是

（1）如果labels的每一行是one-hot表示，也就是只有一个地方为1（或者说100%），其他地方为0（或者说0%），还可以使用tf.sparse\_softmax\_cross\_entropy\_with\_logits()。之所以用100%和0%描述，就是让它看起来像一个概率分布。

（2）tf.nn.softmax\_cross\_entropy\_with\_logits（）函数已经过时 (deprecated)，它在TensorFlow未来的版本中将被去除。取而代之的是

tf.nn.softmax\_cross\_entropy\_with\_logits\_v2（）。

 （3）参数labels,logits必须有相同的形状 [batch\_size, num\_classes] 和相同的类型(float16, float32, float64)中的一种，否则交叉熵无法计算。

（4）tf.nn.softmax\_cross\_entropy\_with\_logits 函数内部的 logits 不能进行缩放（就是说不要对logits作多余的操作，softmax过程会自动完成），因为在这个工作会在改函数内部进行（注意函数名称中的 softmax ，它负责完成原始数据的归一化），如果 logits 进行了缩放，那么反而会影响计算正确性。

## 关于axis的理解

<https://blog.csdn.net/ningning52107/article/details/80227262>

总结：设axis=i，则numpy沿着第i个下标变化的放下进行操作。

规律：一个四维数组a =（4，3，2，3）

sum(a,axis=0)，结果维度就是(3,2,3)

sum(a,axis=1)，结果维度就是(4,2,3)

所以axis为哪个维度，得到的结果矩阵就是去掉那个维度的维度。

## 3.tf.truncated\_normal(shape, mean, stddev)

### 参数说明

shape表示生成张量的维度，mean是均值，stddev是标准差。

### 功能

这个函数产生正太分布，均值和标准差自己设定。这是一个**截断的**产生正太分布的函数，就是说产生正太分布的值如果与均值的差值大于两倍的标准差，那就重新生成。和一般的正太分布的产生随机数据比起来，这个函数产生的随机数与均值的差距不会超过两倍的标准差，但是一般的别的函数是可能的。

## 4.tf.summary()

其中tensorboard 作为一款可视化神器，可以说是学习tensorflow时模型训练以及参数可视化的法宝。

而在训练过程中，主要用到了tf.summary()的各类方法，能够保存训练过程以及参数分布图并在tensorboard显示。

tf.summary有诸多函数：

1、tf.summary.scalar

用来显示标量信息，其格式为：

tf.summary.scalar(tags, values, collections=None, name=None)

例如：tf.summary.scalar('mean', mean)

一般在画loss,accuary时会用到这个函数。

2、tf.summary.histogram

用来显示直方图信息，其格式为：

tf.summary.histogram(tags, values, collections=None, name=None)

例如： tf.summary.histogram('histogram', var)

一般用来显示训练过程中变量的分布情况

3、tf.summary.distribution

分布图，一般用于显示weights分布

7、tf.summary.merge\_all

merge\_all 可以将所有summary全部保存到磁盘，以便tensorboard显示。如果没有特殊要求，一般用这一句就可一显示训练时的各种信息了。

格式：tf.summaries.merge\_all(key='summaries')

## Python的Strip()



## 6.gfile API

Gfile和python中的os模块非常的相似

下面将分别介绍每一个gfile API

### 6.1 tf.gfile.Copy(oldpath, newpath, overwrite=False)

拷贝源文件并创建目标文件，无返回，其形参说明如下：

oldpath：带路径名字的拷贝源文件；

newpath：带路径名字的拷贝目标文件；

overwrite：目标文件已经存在时是否要覆盖，默认为false，如果目标文件已经存在则会报错

### 6.2 tf.gfile.MkDir(dirname)

创建一个目录，dirname为目录名字，无返回。

### 6.3 tf.gfile.Remove(filename)

删除文件，filename即文件名，无返回。

### 6.4 tf.gfile.DeleteRecursively(dirname)

递归删除所有目录及其文件，dirname即目录名，无返回。

### 2-5）tf.gfile.Exists(filename)

判断目录或文件是否存在，filename可为目录路径或带文件名的路径，有该目录则返回True，否则False。

### 2-6）tf.gfile.Glob(filename)

查找匹配pattern的文件并以列表的形式返回，filename可以是一个具体的文件名，也可以是包含通配符的正则表达式。

### 2-7）tf.gfile.IsDirectory(dirname)

判断所给目录是否存在，如果存在则返回True，否则返回False，dirname是目录名。

### 2-8）tf.gfile.ListDirectory(dirname)

罗列dirname目录下的所有文件并以列表形式返回，dirname必须是目录名。

### 2-9）tf.gfile.MakeDirs(dirname)

以递归方式建立父目录及其子目录，如果目录已存在且是可覆盖则会创建成功，否则报错，无返回。

### 2-10）tf.gfile.Rename(oldname, newname, overwrite=False)

重命名或移动一个文件或目录，无返回，其形参说明如下：

oldname：旧目录或旧文件；

newname：新目录或新文件；

overwrite：默认为false，如果新目录或新文件已经存在则会报错，否则重命名或移动成功。

### 2-11）tf.gfile.Stat(filename)

返回目录的统计数据，该函数会返回FileStatistics数据结构，以dir(tf.gfile.Stat(filename))获取返回数据的属性如下：



### 2-12）tf.gfile.Walk(top, in\_order=True)

递归获取目录信息生成器，top是目录名，in\_order默认为True指示顺序遍历目录，否则将无序遍历，每次生成返回如下格式信息(dirname, [subdirname, subdirname, ...], [filename, filename, ...])。

Walk方法详解:

os.walk() 方法用于通过在目录树中游走输出在目录中的文件名，向上或者向下。

os.walk() 方法是一个简单易用的文件、目录遍历器，可以帮助我们高效的处理文件、目录方面的事情。

在Unix，Windows中有效。

语法

walk()方法语法格式如下：

os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]])

参数

top -- 是你所要遍历的目录的地址, 返回的是一个三元组(root,dirs,files)。

root 所指的是当前正在遍历的这个文件夹的本身的地址

dirs 是一个 list ，内容是该文件夹中所有的目录的名字(不包括子目录)

files 同样是 list , 内容是该文件夹中所有的文件(不包括子目录)

topdown --可选，为 True，则优先遍历 top 目录，否则优先遍历 top 的子目录(默认为开启)。如果 topdown 参数为 True，walk 会遍历top文件夹，与top 文件夹中每一个子目录。

onerror -- 可选， 需要一个 callable 对象，当 walk 需要异常时，会调用。

followlinks -- 可选， 如果为 True，则会遍历目录下的快捷方式(linux 下是 symbolic link)实际所指的目录(默认关闭)。

返回值

该方法没有返回值。

### 2-13）tf.gfile.GFile(filename, mode)

获取文本操作句柄，类似于python提供的文本操作open()函数，filename是要打开的文件名，mode是以何种方式去读写，将会返回一个文本操作句柄。

tf.gfile.Open()是该接口的同名，可任意使用其中一个！

### 2-14）tf.gfile.FastGFile(filename, mode)

该函数与tf.gfile.GFile的差别仅仅在于“无阻塞”，即该函数会无阻赛以较快的方式获取文本操作句柄。

## 7.彻底弄懂tf.Variable、tf.get\_variable、tf.variable\_scope以及tf.name\_scope异同

### 7.1 tf.Variable与tf.get\_variable

tensorflow提供了通过变量名称来创建或者获取一个变量的机制。通过这个机制，在不同的函数中可以直接通过变量的名字来使用变量，而不需要将变量通过参数的形式到处传递。   
TensorFlow中通过变量名获取变量的机制主要是通过tf.get\_variable和tf.variable\_scope实现的。   
当然，变量也可以通过tf.Varivale来创建。当tf.get\_variable用于变量创建时，和tf.Variable的功能基本等价。

#以下两个定义是等价的

v = tf.get\_variable('v', shape=[1], initializer=tf.constant\_initializer(1.0))

v = tf.Variable(tf.constant(1.0, shape=[1], name='v')

1

2

3

tf.get\_varialbe和tf.Variable最大的区别在于：tf.Variable的变量名是一个可选项，通过name=’v’的形式给出。但是tf.get\_variable必须指定变量名。

### 7.2 tf.get\_variable与tf.variable\_scope

上面已经提到过了：TensorFlow中通过变量名获取变量的机制主要是通过tf.get\_variable和tf.variable\_scope实现的。在这里，我主要解释下大家深恶痛绝的reuse问题。   
其实只要记住一件事情就ok了：当reuse为False或者None时（这也是默认值），同一个tf.variable\_scope下面的变量名不能相同；当reuse为True时，tf.variable\_scope只能获取已经创建过的变量。   
下面我们通过代码来看下：

#reuse=False时会报错的情况：with tf.variable\_scope('foo'):

v = tf.get\_variable('v',[1],initializer=tf.constant\_initializer(1.0))

with tf.variable\_scope('foo'):

v1 = tf.get\_variable('v',[1])

1

2

3

4

5

6

在这种情况下会报错：Variable foo/v already exists, disallowed.Did you mean to set reuse=True in Varscope?   
其原因就是在命名空间foo中创建了相同的变量。如果我要在foo下创建一个变量v1，其name=‘v’，只需要将reuse设置为Ture就ok了。将上面第二部分代码修改为：

with tf.variable\_scope('foo', reuse=True):

v1 = tf.get\_variable('v',[1])

print(v1.name) #结果为foo/v

1

2

3

当reuse已经设置为True时，tf.variable\_scope只能获取已经创建过的变量。这个时候，在命名空间bar中创建name=‘v’的变量v3，将会报错：Variable bar/v dose not exists, diallowed. Did you mean to set reuse=None in VarScope?

with tf.variable\_scope('bar', reuse=True):

v3 = tf.get\_variable('v',[1])

1

2

简而言之，reuse=False时，tf.variable\_scope创建变量；reuse=True时，tf.variable\_scope获取变量。

### 7.3 tf.variable\_scope与tf.name\_scope

除了tf.variable\_scope，tf.name\_scope函数也提供了命名空间管理的功能。这两个函数在大部分情况下是等价的，唯一的区别是在使用tf.get\_variable函数时。   
tf.get\_variable函数不受tf.name\_scope的影响。   
我们从代码看下这句话的具体意思。   
首先是tf.variable\_scope：

with tf.variable\_scope('foo'):

a = tf.get\_variable('bar',[1])

print(a.name)#结果为foo/bar:0

1

2

3

再看tf.name\_scope：

with tf.name\_scope('a'):

a=tf.Variable([1])

print(a.name)#结果为a/Variable:0

b=tf.get\_variable('b',[1])

print(b.name)#结果为b:0

1

2

3

4

5

6

从这个结果中，我们能很清晰地看到，tf.get\_variable创建的变量并不是a/b:0，而是b:0。这就表示了在tf.name\_scope函数下，tf.get\_variable不受其约束。

## 8.tf.contrib.layers.batch\_norm()

### 函数定义

def batch\_norm(inputs,

decay=0.999,

center=True,

scale=False,

epsilon=0.001,

activation\_fn=None,

param\_initializers=None,

param\_regularizers=None,

updates\_collections=ops.GraphKeys.UPDATE\_OPS,

is\_training=True,

reuse=None,

variables\_collections=None,

outputs\_collections=None,

trainable=True,

batch\_weights=None,

fused=False,

data\_format=DATA\_FORMAT\_NHWC,

zero\_debias\_moving\_mean=False,

scope=None,

renorm=False,

renorm\_clipping=None,

renorm\_decay=0.99):

### 参数说明

**inputs**: A tensor with 2 or more dimensions, where the first dimension has `batch\_size`. The normalization is over all but the last dimension if `data\_format` is `NHWC` and the second dimension if `data\_format` is `NCHW`.代表输入，第一个维度为batch\_size

**dacay**:Decay for the moving average. Reasonable values for `decay` are close to 1.0, typically in the multiple-nines range: 0.999, 0.99, 0.9, etc. Lower `decay` value (recommend trying `decay`=0.9) if model experiences reasonably good training performance but poor validation and/or test performance. Try zero\_debias\_moving\_mean=True for improved stability.代表加权指数平均值的衰减速度，是使用了一种叫做加权指数衰减的方法更新均值和方差。一般会设置为0.9，值太小会导致均值和方差更新太快，而值太大又会导致几乎没有衰减，容易出现过拟合，这种情况一般需要把值调小点。

**center**: If True, add offset of `beta` to normalized tensor. If False, `beta` is ignored. 指定是否使用偏移beta。

**scale**: If True, multiply by `gamma`. If False, `gamma` is not used. When the next layer is linear (also e.g. `nn.relu`), this can be disabled since the scaling can be done by the next layer.是否进行变换(通过乘以一个gamma进行缩放)，我们习惯在BN后面接一个线性变化，如Relu，所以scale一般都设置为Flase，因为后面有对数据的转换处理，所以这里就不用再处理了。

**epsilon**: Small float added to variance to avoid dividing by zero.是为了避免分母为0的情况下，给分母加上的一个极小值，默认即可。

**activation**\_fn: Activation function, default set to None to skip it and maintain a linear activation.激活函数，默认为None，即使用线性激活函数。

**param**\_initializers: Optional initializers for beta, gamma, moving mean and moving variance.可选的初始化参数。

**param\_regularizers**: Optional regularizer for beta and gamma.可选的正则化项。

**updates\_collections:** Collections to collect the update ops for computation. The updates\_ops need to be executed with the train\_op. If None, a control dependency would be added to make sure the updates are computed in place.其变量默认是tf.GraphKeys.UPDATE\_OPS，在训练时提供了一种内置的均值和方差更新机制，即通过图中的tf.Graphs.UPDATE\_OPS变量来更新，但它是在每次当前批次训练完成后才更新均值和方差，这样就导致当前数据总是使用前一次的均值和方差，没有得到最新的更新。所以一般都会将其设置为None，让均值和方差即时更新。这样虽然相比默认值在性能上稍慢点，但是对模型的训练还是有很大帮助的。

**is\_training**: Whether or not the layer is in training mode. In training mode it would accumulate the statistics of the moments into `moving\_mean` and `moving\_variance` using an exponential moving average with the given `decay`. When it is not in training mode then it would use the values of the `moving\_mean` and the `moving\_variance`.当它为True，代表是训练过程，这时会不断更新样本集的均值与方差。当测试时，要设置成False，这样就会使用训练样本集的均值和方差。

**reuse:** Whether or not the layer and its variables should be reused. To be able to reuse the layer scope must be given.支持共享变量，与下面的scope参数联合使用。

variables\_collections: Optional collections for the variables.

outputs\_collections: Collections to add the outputs.

**trainable:** If `True` also add variables to the graph collection `GraphKeys.TRAINABLE\_VARIABLES` (see `tf.Variable`).

batch\_weights: An optional tensor of shape `[batch\_size]`, containing a frequency weight for each batch item. If present, then the batch normalization uses weighted mean and variance. (This can be used to correct for bias in training example selection.)

**used:** Use nn.fused\_batch\_norm if True, nn.batch\_normalization otherwise.

data\_format: A string. `NHWC` (default) and `NCHW` are supported.

zero\_debias\_moving\_mean: Use zero\_debias for moving\_mean. It creates a new air of variables 'moving\_mean/biased' and 'moving\_mean/local\_step'.

**scope:** Optional scope for `variable\_scope`.指定变量的作用域variable\_scope。

renorm: Whether to use Batch Renormalization https://arxiv.org/abs/1702.03275). This adds extra variables during raining. The inference is the same for either value of this parameter.

**renorm\_clipping:** A dictionary that may map keys 'rmax', 'rmin', 'dmax' to scalar `Tensors` used to clip the renorm correction. The correction `(r, d)` is used as `corrected\_value = normalized\_value \* r + d`, with `r` clipped to [rmin, rmax], and `d` to [-dmax, dmax]. Missing rmax, rmin, dmax are set to inf, 0, inf, respectively.

renorm\_decay: Momentum used to update the moving means and standard deviations with renorm. Unlike `momentum`, this affects training and should be neither too small (which would add noise) nor too large (which would give stale estimates). Note that `decay` is still applied to get the means and variances for inference.

### 做的工作包括

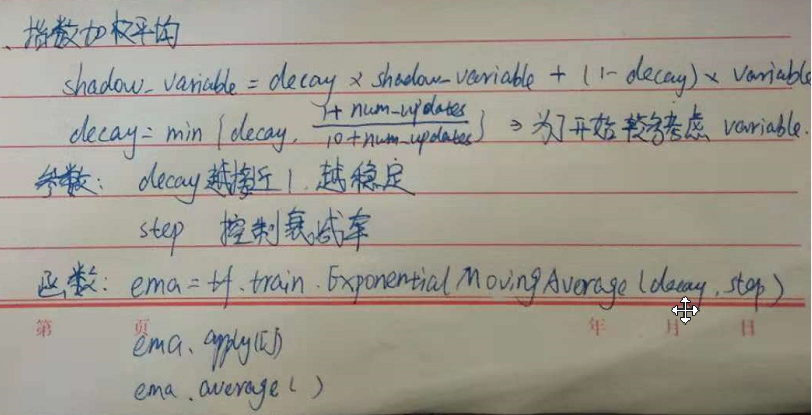
1. 求样本的均值和方差，可由tf.nn.moment()函数实现

def moments(x, axes, shift=None, name=None, keep\_dims=False)

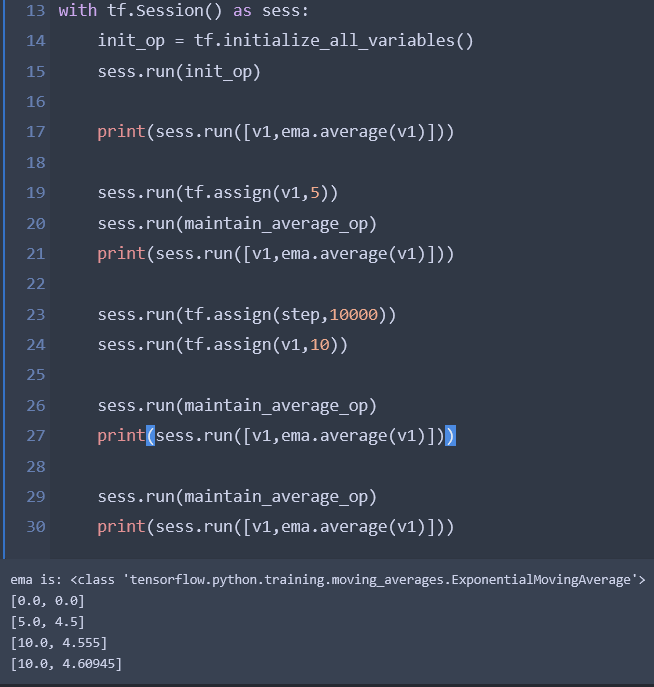
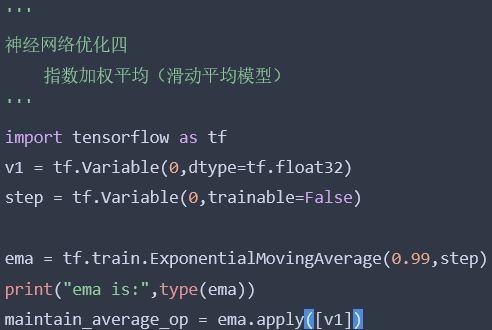
求得：mean和variance（方差）

1. 均值和方差的指数平均，由tf.train.ExponentialMovingAverage类实现

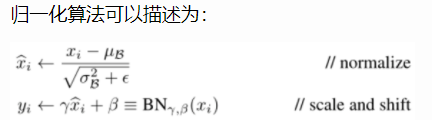
原理和用法：



示例:



（3）数据归一化



其中xi是batch\_size样本中的第i个样本，μB是求得的每个特征的平均值，σB2是求得每个特征的方差。

β代表偏移offset，即相加一个转化值；

γ代表缩放scale，即乘以一个转化值；

## tf.contrib.layers.flatten

### 定义

tf.contrib.layers.flatten(  
    inputs,  
    outputs\_collections=None,  
    scope=None  
)

### 功能

这个函数就是把inputs保留第一个维度，把第一个维度包含的每一子张量展开成一个行向量，返回张量是一个二维的， shape=（batch\_size，….）,一般用于卷积神经网络全链接层前的预处理。所以展开时默认第一个维度是batch维度。

t f.contrib.layers.fully\_connection(F，num\_output,activation\_fn)这个函数就是全链接成层,F是输入，num\_output是下一层单元的个数，activation\_fn是激活函数

## 10.tf.nn.conv2d

conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None, data\_format=None, name=None)

input: 卷积输入，Tensor（tf.Constant，tf.Variable，tf.placeholder），[batch, in\_height, in\_width, in\_channel]

filter: 卷积核，Tensor，[filter\_height, filter\_width, in\_channel, out\_channel]

strides: 卷积核在各个维度移动的步长，a 4-D list，[stride\_batch, stride\_height, stride\_width, stride\_channel]

**padding: 对输入input的填充方法，'SAME' 和 'VALID'两种**

**padding为'SAME'时,凑个整数（注意，same padding和吴恩达说的不一样啊）**  
输出Tensor大小: [batch, ceil(in\_height / strides\_h), ceil(in\_width / strides\_w), out\_channel]  
padding为'VALID'时,没有padding  
输出Tensor大小: [batch, ceil(in\_height - filter[1]) / strides[1] + 1, ceil(in\_width - filter[2]) / strides[2] + 1, out\_channel]

### 11.tf.nn.conv2d()

**方法定义**

tf.nn.conv2d (input, filter, strides, padding, use\_cudnn\_on\_gpu=None, data\_format=None, name=None)

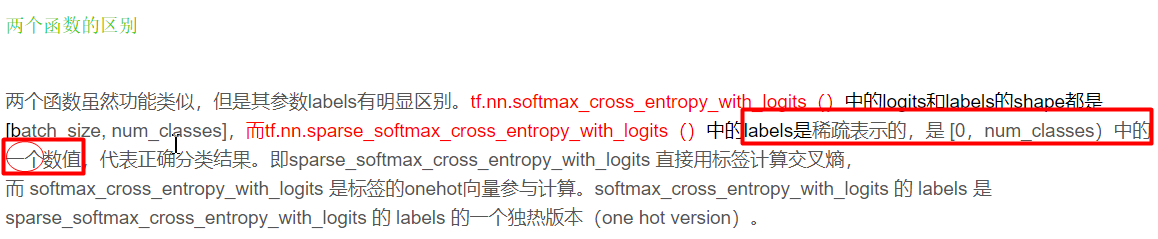
**参数**

(1)\*\*input : \*\* 输入的要做卷积的图片，要求为一个张量，shape为 [ batch, in\_height, in\_weight, in\_channel ]，其中batch为图片的数量，in\_height 为图片高度，in\_weight 为图片宽度，in\_channel 为图片的通道数，灰度图该值为1，彩色图为3。（也可以用其它值，但是具体含义不是很理解）

(2)filter： 卷积核，要求也是一个张量，shape为 [ filter\_height, filter\_weight, in\_channel, out\_channels ]，其中 filter\_height 为卷积核高度，filter\_weight 为卷积核宽度，in\_channel 是图像通道数 ，和 input 的 in\_channel 要保持一致，out\_channel 是卷积核数量。

(3)strides： 卷积时在图像每一维的步长，这是一个一维的向量，[ 1, strides, strides, 1]，第一位和最后一位固定必须是1

(4)padding： string类型，值为“SAME” 和 “VALID”，表示的是卷积的形式，是否考虑边界。"SAME"是考虑边界，不足的时候用0去填充周围，"VALID"则不考虑



## 11. Numy.concatenate——ndarray数组拼接

numpy提供了numpy.concatenate((a1,a2,...), axis=0)函数。能够一次完成多个多维数组的拼接。其中a1,a2,...是数组类型的参数

示例：

>>> a=np.array([1,2,3])

>>> b=np.array([11,22,33])

>>> c=np.array([44,55,66])

>>> np.concatenate((a,b,c),axis=0)  # 默认情况下，axis=0可以不写

array([ 1,  2,  3, 11, 22, 33, 44, 55, 66]) #对于一维数组拼接，axis的值不影响最后的结果

>>> a=np.array([[1,2,3],[4,5,6]])

>>> b=np.array([[11,21,31],[7,8,9]])

>>> np.concatenate((a,b),axis=0)

array([[ 1,  2,  3],

       [ 4,  5,  6],

       [11, 21, 31],

       [ 7,  8,  9]])

>>> np.concatenate((a,b),axis=1)  #axis=1表示对应行的数组进行拼接

array([[ 1,  2,  3, 11, 21, 31],

       [ 4,  5,  6,  7,  8,  9]])

## 12. tf.zeros\_like(tensor, dtype=None, name=None, optimize=True)

　创建一个shape和指定tensor相同的变量，但全部元素都为零。例如‘tensor’ =[[1,2,3], [4,5,6]]，那么tf.zeros\_like(tensor) ==>[[0,0,0],[0,0,0]]

tf.ones\_like()同理

## zeros(shape[, dtype, order])

　　依据给定形状和类型(shape[, dtype, order])返回一个新的元素全部为0的数组。

参数：

shape：int或者ints元组；定义返回数组的形状，形如：(2, 3)或2。

dtype：数据类型，可选。

返回数组的数据类型，例如：numpy.int8、默认numpy.float64。

order:{‘C’, ‘F’},可选,返回数组为多维时，元素在内存的排列方式是按C语言还是Fortran语言顺序(row- or columnwise)。

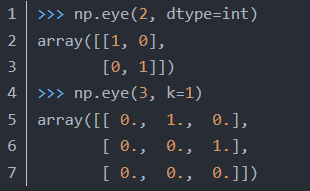
输出：ndarray给定形状，数据类型的数组。

## numpy.eye(*N*,*M=None*, *k=0*, *dtype=<type 'float'>*)

关注第一个第三个参数就行了

第一个参数：输出方阵（行数=列数）的规模，即行数或列数

第三个参数：默认情况下输出的是对角线全“1”，其余全“0”的方阵，如果k为正整数，则在右上方第k条对角线全“1”其余全“0”，k为负整数则在左下方第k条对角线全“1”其余全“0”。



## 读写csv文件

数组存储成CSV之类的区隔型文件：

下面代码给随机数生成器指定种子，并生成一个3\*4的NumPy数组

将一个数组元素的值设为NaN:

In [26]: import numpy as np

In [27]: np.random.seed(42)

In [28]: a = np.random.randn(3,4)

In [29]: a[2][2] = np.nan

In [30]: print(a)

[[ 0.49671415 -0.1382643 0.64768854 1.52302986]

[-0.23415337 -0.23413696 1.57921282 0.76743473]

[-0.46947439 0.54256004 nan -0.46572975]]

NumPy的savetxt()函数是与loadtxt()相对应的一个函数，它能以诸如CSV之类的区隔型文件格式保存数组：

In [31]: np.savetxt('np.csv',a,fmt='%.2f',delimiter=',',header="#1,#2,#3,#4")

上面的函数调用中，我们规定了用以保存数组的文件的名称、数组、可选格式、间隔符和一个可选的标题

## Np.random.choice方法

# 参数意思分别 是从a 中以概率P，随机选择3个, p没有指定的时候相当于是一致的分布

a1 = np.random.choice(a=5, size=3, replace=False, p=None)

# 非一致的分布，会以多少的概率提出来

a2 = np.random.choice(a=5, size=3, replace=False, p=[0.2, 0.1, 0.3, 0.4, 0.0])

print(a2)

# replacement 代表的意思是抽样之后还放不放回去，如果是False的话，那么出来的三个数都不一样，如果是

True的话， 有可能会出现重复的，因为前面的抽的放回去了。

## Python set()

Python set() 函数

[IMG_256 Python 内置函数](http://www.runoob.com/python/python-built-in-functions.html)

描述

set() 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据，还可以计算交集、差集、并集等。

语法

set 语法：

class set([iterable])

参数说明：

iterable -- 可迭代对象对象；

返回值

返回新的集合对象。

实例

以下实例展示了 set 的使用方法：

>>>x = set('runoob') >>> y = set('google') >>> x, y (set(['b', 'r', 'u', 'o', 'n']), set(['e', 'o', 'g', 'l'])) # 重复的被删除 >>> x & y # 交集 set(['o']) >>> x | y # 并集 set(['b', 'e', 'g', 'l', 'o', 'n', 'r', 'u']) >>> x - y # 差集 set(['r', 'b', 'u', 'n']) >>>

Tf.reduce\_mean()求平均数

Tf.reduce\_sum()求和

## 18.numpy中array和asarray的区别

array和asarray都可以将结构数据转化为ndarray，但是主要区别就是当数据源是ndarray时，array仍然会copy出一个副本，占用新的内存，但asarray不会。

## 19.Python的set()

python的set和其他语言类似, 是一个无序不重复元素集, 基本功能包括关系测试和消除重复元素. 集合对象还支持union(联合), intersection(交), difference(差)和sysmmetric difference(对称差集)等数学运算.

sets 支持 x in set, len(set),和 for x in set。作为一个无序的集合，sets不记录元素位置或者插入点。因此，sets不支持 indexing, slicing, 或其它类序列（sequence-like）的操作。

下面来点简单的小例子说明把。

>>> x = set('spam')>>> y = set(['h','a','m'])>>> x, y

(set(['a', 'p', 's', 'm']), set(['a', 'h', 'm']))

再来些小应用。

>>> x & y # 交集

set(['a', 'm'])

>>> x | y # 并集

set(['a', 'p', 's', 'h', 'm'])

>>> x - y # 差集

set(['p', 's'])

## 20.list的index方法

index() 函数用于从列表中找出某个值第一个匹配项的索引位置。

语法

index()方法语法：

list.index(obj)

参数

obj -- 查找的对象。

返回值

该方法返回查找对象的索引位置，如果没有找到对象则抛出异常。

实例

以下实例展示了 index()函数的使用方法：

#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc'];

print "Index for xyz : ", aList.index( 'xyz' ) ;print "Index for zara : ", aList.index( 'zara' ) ;

以上实例输出结果如下：

Index for xyz : 1Index for zara : 2

## 21.sort函数

描述

sort()函数是Python的列表函数，用于对原列表进行排序。

语法

list.sort(key=function, reverse=boolean)

名称

说明

备注

list

列表名称

key=function

列表元素权值参数

可省略的参数。省略时列表排序不考虑元素的权值

reverse=boolean

排序规则参数

可省略的参数。省略时reverse默认值False

返回值：函数在原有列表上排序并返回排序后的列表

使用示例

1. 所有参数都省略

当sort函数的所有参数都省略的情况下，对原列表进行升序排序：

1. 如果原列表是数字，则按照数字的大小排序；

2. 如果原列表是字符串，则会按照字母表顺序排列.

>>> \_list\_a = [1, 34, 21, 4]>>> \_list\_a.sort() #对元素是数字的列表进行排序>>> \_list\_a[1, 4, 21, 34] >>> \_list\_b = ['Huawei', 'Chinasoft', 'CASC', 'Alibaba', 'SoftStone', 'Tencent']>>> \_list\_b.sort() #对元素是字符串的列表进行排序>>> \_list\_b['Alibaba', 'CASC', 'Chinasoft', 'Huawei', 'SoftStone', 'Tencent']

2. 指定key参数

key参数指的是为列表的元素一一标记权值，并按照元素所对应的权值来排序元素。key=函数名，该函数的返回值作为元素的权值大小。

例如，使用字符串长度作为元素的权值，用该权值排序元素是字符串的列表：

下面的例子中，分别不指定权值和指定权值来排序两个一模一样的列表。

>>> \_list\_c = ['Beijing', 'Xi\'an', 'Taiyuan', 'Taipei', 'Hongkang']>>> \_list\_d = ['Beijing', 'Xi\'an', 'Taiyuan', 'Taipei', 'Hongkang']>>> \_list\_c.sort() #不指定权值>>> \_list\_d.sort(key=len) #指定权值>>> \_list\_c['Beijing', 'Hongkang', 'Taipei', 'Taiyuan', "Xi'an"]>>> \_list\_d["Xi'an", 'Taipei', 'Beijing', 'Taiyuan', 'Hongkang'] #按照元素的长度作为权值的排序结果

很明显，\_list\_d按照元素长度排序，没有按照元素的字母表顺序排序。

3. 指定reverse参数

reverse参数指的是列表排序是升序还是降序排序。当reverse参数省略或reverse=False时，列表升序排序。当reverse=True时，列表降序排序：

>>> \_list\_a = [13, 45, 23, 66]>>> \_list\_b = [13, 45, 23, 66]>>> \_list\_a.sort() #默认升序排序>>> \_list\_b.sort(reverse=True) #降序排序>>> \_list\_a[13, 23, 45, 66]>>> \_list\_b[66, 45, 23, 13]

💡提示：当key参数和reverse参数同时指定时，sort函数按照元素的权值进行升序（reverse=False）或降序（reverse=True）进行排序。

注意事项

1. sort函数会改变原列表顺序

使用sort函数时要注意的是，sort函数在调用后会改变列表本身的顺序，而不是复制列表后排序，并返回复制后的列表。

>>> \_list\_a = [1, 34, 21, 4]>>> \_list\_a.sort()>>> \_list\_a[1, 4, 21, 34]

从上面的代码可以看出，\_list\_a的列表本身顺序发生了变化。

2. 列表元素类型不一致

当列表中的所有元素都是同一种类型时，sort()函数会正常工作。有些时候甚至多种类型也可----例如整型和浮点型----只要它们能够自动地互相转换：

>>> list\_a = [23, 21.2, 22, -67.6]>>> list\_a.sort()>>> list\_a[-67.6, 21.2, 22, 23]

但是当元素之间的类型不能够互相转换的时候，Python就会报错，例如整数和字符串类型：

>>> a\_list = [1, 3, '5', 2]>>> a\_list.sort()Traceback (most recent call last): File "<stdin>", line 1, in <module>TypeError: '<' not supported between instances of 'str' and 'int'

## 22.匿名函数

[python的匿名函数lambda解释及用法](https://www.cnblogs.com/xisheng/p/7301245.html)

lambda函数的语法只包含一个语句，如下：    lambda arg1,arg2,.....argn:expression（主要是看下面的例子）  
代码示例：

[IMG_256](https://www.cnblogs.com/xisheng/p/javascript:void(0);)

#-\*- coding:utf-8 -\*-

#\_\_author\_\_ = "www.iplaypy.com"

# 普通python函数def func(a,b,c):

return a+b+c

print func(1,2,3)# 返回值为6

# lambda匿名函数

f = lambda a,b,c:a+b+c

print f(1,2,3)# 返回结果为6

[IMG_257](https://www.cnblogs.com/xisheng/p/javascript:void(0);)

# 大家注意观察上面的Python示例代码，f = lambda a,b,c:a+b+c 中的关键字lambda表示匿名函数，  
# 冒号:之前的a,b,c表示它们是这个函数的参数。  
# 匿名函数不需要[return](http://www.iplaypy.com/jinjie/return.html" \o "python return 函数返回值" \t "https://www.cnblogs.com/xisheng/p/_blank)来返回值，表达式本身结果就是返回值。

（以下是我复制过来的一些代码，正是通过这些代码让我对匿名函数有了一个比较深入的了解，所以建议大家认真看看，）

无参匿名函数:  
------  
>>> t = lambda : True #分号前无任何参数  
>>> t()  
True  
  
等价于下面的def定义的函数  
>>> def func(): return True  
>>> func()  
True  
  
------  
  
>>> s = "this is\na\ttest" #建此字符串按照正常情形输出  
>>> s  
'this is\na\ttest'  
>>> print s.split() #split函数默认分割:空格，换行符，TAB  
['this', 'is', 'a', 'test']  
>>> ' '.join(s.split()) #用join函数转一个列表为字符串  
'this is a test'  
  
等价于  
  
>>> (lambda s:' '.join(s.split()))("this is\na\ttest")  
  
  
  
  
带参数匿名函数  
  
>>> lambda x: x\*\*3 #一个参数  
>>> lambda x,y,z:x+y+z #多个参数  
  
>>> lambda x,y=3: x\*y #允许参数存在默认值  
  
  
  
匿名函数调用  
  
#直接赋值给一个变量,然后再像一般函数调用  
  
------  
  
>>> c = lambda x,y,z: x\*y\*z  
>>> c(2,3,4)  
24  
  
------  
  
>>> c = lambda x,y=2: x+y #使用了默认值  
>>> c(10) #不输的话，使用默认值2  
12  
  
------  
  
>>> a = lambda \*z:z #\*z返回的是一个元祖  
>>> a('Testing1','Testing2')  
('Testing1', 'Testing2')  
  
------  
  
>>> c = lambda \*\*Arg: Arg #arg返回的是一个字典  
>>> c()  
{}  
  
#直接后面传递实参  
  
------  
  
>>> (lambda x,y: x if x> y else y)(101,102)  
102   
  
------  
  
>>> (lambda x:x\*\*2)(3)  
9  
  
#lambda返回的值，结合map,filter,reduce使用  
  
>>> filter(lambda x:x%3==0,[1,2,3,4,5,6])  
[3, 6]  
  
等价于下面的列表推导式  
  
>>> l = [x for x in [1,2,3,4,5,6] if x%3==0]  
>>> l  
[3, 6]

Join

  join()：    连接字符串数组。将字符串、元组、列表中的元素以指定的字符(分隔符)连接生成一个新的字符串

#对序列进行操作（分别使用' '与':'作为分隔符）

>>> seq1 = ['hello','good','boy','doiido']

>>> print ' '.join(seq1)

hello good boy doiido

>>> print ':'.join(seq1)

hello:good:boy:doiido

## 23.input和raw\_input

**Python2:**

使用input和raw\_input都可以读取控制台的输入，但是input和raw\_input在处理数字时是有区别的  
纯数字输入

当输入为纯数字时

    input返回的是数值类型，如int,float  
    raw\_inpout返回的是字符串类型，string类型

输入字符串为表达式

input会计算在字符串中的数字表达式，而raw\_input不会。

如输入 “57 + 3”:

    input会得到整数60  
    raw\_input会得到字符串”57 + 3”

**Python3:**

python3将raw\_input和input进行了整合，只有input

## 24 三元运算符

python中只有类似的替代办法，如果a>b的结果为真，h="变量1",如果为假，h="变量2"

[IMG_256](https://www.cnblogs.com/mywood/p/javascript:void(0);)

a = 1

b = 2

h = ""

h = "变量1" if a>b else "变量2"

print(h)

[IMG_257](https://www.cnblogs.com/mywood/p/javascript:void(0);)

也可以用简单的公式，如下，

[IMG_258](https://www.cnblogs.com/mywood/p/javascript:void(0);)

a = 1

b = 2

h = ""

h = a-b if a>b else a+b

print(h)

## 25 枚举

enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。

Python 2.3. 以上版本可用，2.6 添加 start 参数。

**语法**

以下是 enumerate() 方法的语法:

enumerate(sequence, [start=0])

**参数**

* sequence -- 一个序列、迭代器或其他支持迭代对象。
* start -- 下标起始位置。

**返回值**

返回 enumerate(枚举) 对象。

[回到顶部](https://www.cnblogs.com/wxj1129549016/p/9516610.html" \l "_labelTop)

**实例**

以下展示了使用 enumerate() 方法的实例：

>>>seasons = ['Spring', 'Summer', 'Fall', 'Winter']>>> list(enumerate(seasons))

[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]>>> list(enumerate(seasons, start=1)) # 小标从 1 开始

[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]

[IMG_256](https://www.cnblogs.com/wxj1129549016/p/javascript:void(0);)

普通的 for 循环>>>i = 0>>> seq = ['one', 'two', 'three']>>> for element in seq:

... print i, seq[i]

... i +=1

...

0 one1 two2 three

for 循环使用 enumerate

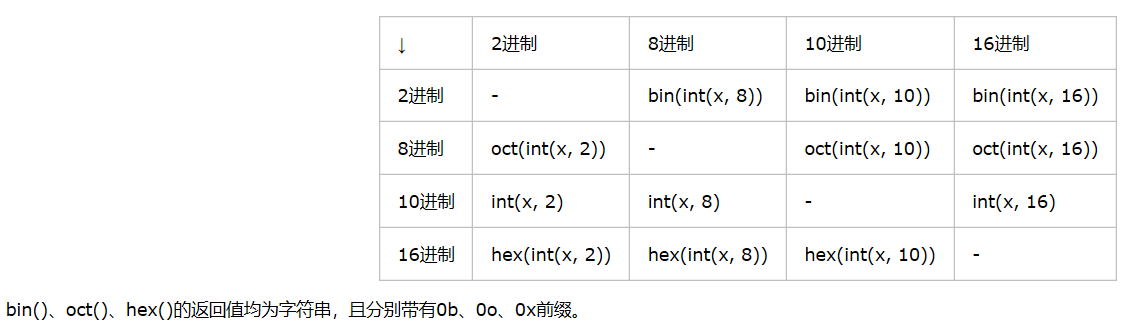
>>>seq = ['one', 'two', 'three']>>> for i, element in enumerate(seq):

... print i, element

...

0 one1 two2 three

## 26 进制转换



## 27 排序

### 27.1 快排

快速排序

快速排序是由东尼·霍尔所发展的一种排序算法。在平均状况下，排序 n 个项目要 Ο(nlogn) 次比较。在最坏状况下则需要 Ο(n2) 次比较，但这种状况并不常见。事实上，快速排序通常明显比其他 Ο(nlogn) 算法更快，因为它的内部循环（inner loop）可以在大部分的架构上很有效率地被实现出来。

快速排序使用分治法（Divide and conquer）策略来把一个串行（list）分为两个子串行（sub-lists）。

快速排序又是一种分而治之思想在排序算法上的典型应用。本质上来看，快速排序应该算是在冒泡排序基础上的递归分治法。

快速排序的名字起的是简单粗暴，因为一听到这个名字你就知道它存在的意义，就是快，而且效率高！它是处理大数据最快的排序算法之一了。虽然 Worst Case 的时间复杂度达到了 O(n²)，但是人家就是优秀，在大多数情况下都比平均时间复杂度为 O(n logn) 的排序算法表现要更好，可是这是为什么呢，我也不知道。好在我的强迫症又犯了，查了 N 多资料终于在《算法艺术与信息学竞赛》上找到了满意的答案：

快速排序的最坏运行情况是 O(n²)，比如说顺序数列的快排。但它的平摊期望时间是 O(nlogn)，且 O(nlogn) 记号中隐含的常数因子很小，比复杂度稳定等于 O(nlogn) 的归并排序要小很多。所以，对绝大多数顺序性较弱的随机数列而言，快速排序总是优于归并排序。

1. 算法步骤

从数列中挑出一个元素，称为 “基准”（pivot）;

重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作；

递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序；

递归的最底部情形，是数列的大小是零或一，也就是永远都已经被排序好了。虽然一直递归下去，但是这个算法总会退出，因为在每次的迭代（iteration）中，它至少会把一个元素摆到它最后的位置去。

def quickSort(arr, left=None, right=None):  
   left = 0 if not isinstance(left,(int, float)) else left  
   right = len(arr)-1 if not isinstance(right,(int, float)) else right  
   if left < right:  
       partitionIndex = partition(arr, left, right)  
       quickSort(arr, left, partitionIndex-1)  
       quickSort(arr, partitionIndex+1, right)  
   return arr  
  
def partition(arr, left, right):  
   pivot = left  
   index = pivot+1  
   i = index  
   while  i <= right:  
       if arr[i] < arr[pivot]:  
           swap(arr, i, index)  
           index+=1  
       i+=1  
   swap(arr,pivot,index-1)  
   return index-1  
  
def swap(arr, i, j):  
   arr[i], arr[j] = arr[j], arr[i]

## 27.2 堆排

**堆排序**

堆排序（Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆积是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。堆排序可以说是一种利用堆的概念来排序的选择排序。

def buildMaxHeap(arr):  
   import math  
   for i in range(math.floor(len(arr)/2),-1,-1):  
       heapify(arr,i)  
  
def heapify(arr, i):  
   left = 2\*i+1  
   right = 2\*i+2  
   largest = i  
   if left < arrLen and arr[left] > arr[largest]:  
       largest = left  
   if right < arrLen and arr[right] > arr[largest]:  
       largest = right  
  
   if largest != i:  
       swap(arr, i, largest)  
       heapify(arr, largest)  
  
def swap(arr, i, j):  
   arr[i], arr[j] = arr[j], arr[i]  
  
def heapSort(arr):  
   global arrLen  
   arrLen = len(arr)  
   buildMaxHeap(arr)  
   for i in range(len(arr)-1,0,-1):  
       swap(arr,0,i)  
       arrLen -=1  
       heapify(arr, 0)  
   return arr

## 28 isinstance函数用法

**语法：isinstance（object，type）**

**作用：来判断一个对象是否是一个已知的类型。**

**其第一个参数（object）为对象，第二个参数（type）为类型名(int...)或类型名的一个列表((int,list,float)是一个列表)。其返回值为布尔型（True or flase）。**

**若对象的类型与参数二的类型相同则返回True。若参数二为一个元组，则若对象类型与元组中类型名之一相同即返回True。**

**下面是两个例子：**

**例一**

>>> a = 4  
>>> isinstance (a,int)  
True  
>>> isinstance (a,str)  
False  
>>> isinstance (a,(str,int,list))  
True