百度不行就谷歌，然后是看文档http://devdocs.io/tensorflow~python/

@author 梁浩赞

1. x=tf.placeholder("float",[None,784])

placeholder占位符，可以先创建可以任意维度的矩阵，待以后输入；None在这里表示第一维可以是任意大小的，float是数据类型

还可以是input=tf.placeholder(tf.float32)

1. a=tf.constant([1.0,2.0],name="a")

constant生成tf的常量

constant （

value ，

dtype = None ，

shape = None ，

name = 'Const' ，

verify\_shape = False

）

value：输出类型 dtype 的常量值 (或列表)。

dtype：所得张量的元素类型。

shape：所得张量的可选维度。

name：张量的可选名称。

verify\_shape：布尔值，可以验证数值的形状。

1. with tf.device('/gpu:0'):

指定设备，cpu:0,gpu:0等

1. v=tf.get\_variable("v",shape=[1],initializer=tf.zeros\_initializer())

创建或者获取变量，指定名称、维度、初始化函数

初始化函数如下：

tf.constant\_initializer：常量初始化函数

tf.random\_normal\_initializer：正态分布

tf.truncated\_normal\_initializer：截取的正态分布

tf.random\_uniform\_initializer：均匀分布

tf.zeros\_initializer：全部是0

tf.ones\_initializer：全是1

tf.uniform\_unit\_scaling\_initializer：满足均匀分布，但不影响输出数量级的随机值

1. tf.global\_variables\_initializer().run()

运行初始化函数，初始化变量

1. tf.Variable

tf.Variable(initial\_value, trainable=True, collections=None, validate\_shape=True, name=None)



生成变量，支持不同参数，可以是整数，浮点数，矩阵

实例：

1. a1 = tf.Variable(tf.random\_normal(shape=[2,3], mean=0, stddev=1), name='a1')
2. a2 = tf.Variable(tf.constant(1), name='a2')
3. a3 = tf.Variable(tf.ones(shape=[2,3]), name='a3')
4. new\_value = tf.add(state, one)

加法运算器

1. update = tf.assign(state, new\_value)

用new\_value替换state，类似还有assign\_add(state,value):state+=value;assign\_sub()

1. sess=tf.Session()

创建session对象，来计算图

1. sess.run(state)

到这一步计算才被执行

1. output=tf.multiply(input1,input2)

乘法，旧版本写法tf.mul()

1. product=tf.matmul(matrix1,matrix2)

矩阵相乘，1\*2

1. y=tf.nn.softmax(tf.matmul(x,W)+b)

softmax分类器，使用logistic函数求y值，参数为预测的y的表达式

1. cross\_entropy=-tf.reduce\_sum(y\_\*tf.log(y))

reduce\_sum求和，这里用交叉熵来表示损失y\_\*tf.log(y)

1. train\_step=tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)

梯度下降算法，0.01是学习率，minimize表示最小化损失，第二个参数是损失的表达式；还可以使用不同的优化算法

1. batch\_xs,batch\_ys=mnist.train.next\_batch(100)

MNIST自带的数据读取方式，每次随机读取100个数据

1. tf.argmax | tf.argmin

tf.argmax(input=tensor,dimention=axis)

axis表示方向，0表示每列的最大值，1表示每行的最大值，返回一个矩阵

PS：tensorflow的其他有关矩阵方向求值的都是0表示列，1表示行

1. tf.equal(a，b)  
   判断两个tensor是否每个元素都相等。返回一个格式为bool的tensor
2. b = tf.cast(a,dtype=tf.bool)

将x的数据格式转化成dtype.例如，原来x的数据格式是bool，那么将其转化成float以后，就能够将其转化成0和1的序列。反之也可以

1. tf.reduce\_sum(),tf.reduce\_max(),tf.reduce\_min(),tf.reduce\_mean()

分别是求总和、最大值、最小值、平均值

函数声明：

reduce\_mean( input\_tensor, axis=None, keep\_dims=False, name=None, reduction\_indices=None )

压缩哪个维度，即是往哪个方向压缩张量

参数：

input\_tensor：要减少的张量。应该有数字类型。

axis：要减小的尺寸。如果为None（默认），则减少所有维度。必须在[-rank(input\_tensor), rank(input\_tensor))范围内。二维矩阵情况下，0表示按列，1表示按行

keep\_dims：如果为true，则保留长度为1的缩小尺寸。

name：操作的名称（可选）。

reduction\_indices：axis的不支持使用的名称。

1. TensorBoard可视化一般框架
2. tf.summary.scalar(“name”，value)
3. merged\_summary\_op = tf.summary.merge\_all()
4. summary\_writer = tf.summary.FileWriter('/tmp/mnist\_logs', sess.graph)
5. total\_step = 0
6. **while** training:
7. total\_step += 1
8. session.run(training\_op)
9. **if** total\_step % 100 == 0:
10. summary\_str = session.run(merged\_summary\_op)
11. summary\_writer.add\_summary(summary\_str, total\_step)

cmd命令：tensorboard --logdir=filewriter的dir

ps：如果遇到提示没有scalars，那么先检查路径下是否有日志，如果有，那么先cd到日志文件的上一层目录，再激活adaconda3环境，再运行tensorboard --logdir=

1. tf.gfile.FastGFile(path,decodestyle) ：读取图像或模型
2. image\_jpg = tf.gfile.FastGFile('dog.jpg','rb').read()#’path’，’rb’是非utf8编码，’r’是utf8编码
3. image\_jpg = tf.image.decode\_jpeg(image\_jpg) #图像解码
4. **print**(sess.run(image\_jpg))#打印解码后的图像（即为一个三维矩阵[w,h,3]）
5. image\_jpg = tf.image.convert\_image\_dtype(image\_jpg,dtype=tf.uint8) #改变图像数据类型
6. plt.figure(1) #图像显示
7. plt.imshow(image\_jpg.eval())

利用tf.gfile.FastGfile()保存和加载pb格式模型：

<https://blog.csdn.net/wc781708249/article/details/78043099>

实例：

1. **import** tensorflow as tf
2. **from** tensorflow.python.framework **import** graph\_util
3. **import** numpy as np
4. output\_graph\_path = r"./expert-graph.pb"
5. with tf.Session() as sess:
6. # with tf.gfile.FastGFile(output\_graph\_path, 'rb') as f:
7. # graph\_def = tf.GraphDef()
8. # graph\_def.ParseFromString(f.read())
9. # sess.graph.as\_default()
10. # tf.import\_graph\_def(graph\_def, name='')
11. tf.global\_variables\_initializer().run()
12. output\_graph\_def = tf.GraphDef()
13. with open(output\_graph\_path, "rb") as f:
14. output\_graph\_def.ParseFromString(f.read())
15. \_ = tf.import\_graph\_def(output\_graph\_def, name="")
16. input\_x = sess.graph.get\_tensor\_by\_name("input:0")
17. **print**(input\_x)
18. y=input\_x
19. out=sess.run(y,{input\_x:np.random.random([1,28\*28])})
20. **print**(out[:10])
21. tf.truncated\_normal([100,100],stddev=0.01)

声明：

tf.truncated\_normal(

shape,

mean=0.0,

stddev=1.0,

dtype=tf.float32,

seed=None,

name=None)

Args:

shape: 张量维度

mean:均值

stddev:标准差

dtype: 类型

seed: A Python integer. Used to create a random seed for the distribution. See [tf.set\_random\_seed](http://devdocs.io/tensorflow~python/tf/set_random_seed) for behavior.

name: A name for the operation (optional).

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

第一个参数input：指需要做卷积的输入图像，它要求是一个Tensor，具有[batch, in\_height, in\_width, in\_channels]这样的shape，具体含义是[训练时一个batch的图片数量, 图片高度, 图片宽度, 图像通道数]，注意这是一个4维的Tensor，要求类型为float32和float64其中之一

第二个参数filter：相当于CNN中的卷积核，它要求是一个Tensor，具有[filter\_height, filter\_width, in\_channels, out\_channels]这样的shape，具体含义是[卷积核的高度，卷积核的宽度，图像通道数，卷积核个数]，要求类型与参数input相同，有一个地方需要注意，第三维in\_channels，就是参数input的第四维

第三个参数strides：卷积时在图像每一维的步长，这是一个一维的向量，长度4，一般取strides=[1,1,1,1]

第四个参数padding：string类型的量，只能是"SAME","VALID"其中之一，这个值决定了不同的卷积方式。当其为‘SAME’时，对应padding=1，表示卷积核可以停留在图像边缘，5\*5矩阵，3\*3卷积核，得到5\*5 feature map；’VALID’时，对应padding=0，得到3\*3

第五个参数：use\_cudnn\_on\_gpu:bool类型，是否使用cudnn加速，默认为true

结果返回一个Tensor，这个输出，就是我们常说的feature map，shape仍然是[batch, height, width, channels]这种形式。

参考：<https://www.cnblogs.com/qggg/p/6832342.html>

1. tf.nn.max\_pool(value, ksize, strides, padding, name=None)

第一个参数value：需要池化的输入，一般池化层接在卷积层后面，所以输入通常是feature map，依然是[batch, height, width, channels]这样的shape

第二个参数ksize：池化窗口的大小，取一个四维向量，一般是[1, height, width, 1]，因为我们不想在batch和channels上做池化，所以这两个维度设为了1

第三个参数strides：和卷积类似，窗口在每一个维度上滑动的步长，一般也是[1, stride,stride, 1]

第四个参数padding：和卷积类似，可以取'VALID' 或者'SAME'

返回一个Tensor，类型不变，shape仍然是[batch, height, width, channels]这种形式

参考：<https://blog.csdn.net/mao_xiao_feng/article/details/53453926>

PS：参数最后一维都是通道数，同一个通道的矩阵一起计算，不同通道分开计算，最终合并在同一行里。例如，RGB，先池化R，再G，再B，在把这三个值放在同一行不同列里

1. tf.reshape(tensor, shape, name=None)

函数的作用是将tensor变换为参数shape的形式

reshape（t, shape） => reshape(t, [-1]) => reshape(t, shape)

首先将矩阵t变为一维矩阵，然后再对矩阵的形式更改就可以了。

-1代表的含义是不用我们自己指定这一维的大小，函数会自动计算，但列表中只能存在一个-1。

# tensor 't' is [1, 2, 3, 4, 5, 6, 7, 8, 9] # tensor 't' has shape [9] reshape(t, [3, 3]) ==> [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

参考：<https://blog.csdn.net/mao_xiao_feng/article/details/53453926>

1. tf.nn.dropout(x, keep\_prob, noise\_shape=None, seed=None, name=None)

对全连接层得到的矩阵进行随机丢弃一些值，置为0，防止过拟合

x                 :  输入tensor  
keep\_prob    :  float类型，每个元素被保留下来的概率  
noise\_shape  : 一个1维的int32张量，代表了随机产生“保留/丢弃”标志的shape。  
seed             : 整型变量，随机数种子。  
name            : 名字

1. tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

Adam算法优化器

1. tf.clip\_by\_value(value,minvalue,maxvalue)

将张量value的数值限制在minvalue和maxvalue之间，可以防止避免一些运算错误

1. tf.nn.softmax\_cross\_entropy\_with\_logtis(logtis,labels)

将softmax和交叉熵一起合并成同一个函数，使用softmax激活函数之后算交叉熵

PS:分类多使用交叉熵（cross\_entropy）作为损失函数，回归预测多使用均方差（MSE）

1. tf.train.exponential\_decay(learning\_rate, global\_step, decay\_steps, decay\_rate, staircase=False, name=None)

指数衰减的学习率：learning\_rate=learning\_rate\*(decay\_rate^(global\_step/decay\_steps)

参数：

learning\_rate: float32或者float64张量；初始学习率

global\_step: int32或者int64张量；增长，用来除以decay\_steps

decay\_steps: int32或者int64；每多少步学习率衰减一次

decay\_rate:衰减率

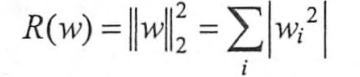
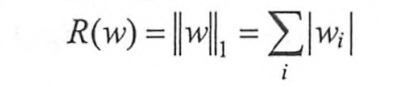
staircase：True时，global\_step/decay\_steps取整数，阶梯式下降；False时，取小数

name: string. Optional name of the operation. Defaults to 'ExponentialDecay'

使用框架：

1. global\_step = tf.Variable(0, trainable=False)
2. starter\_learning\_rate = 0.1
3. learning\_rate = tf.exponential\_decay(starter\_learning\_rate, global\_step,
4. 100000, 0.96, staircase=True)
5. optimizer = tf.GradientDescent(learning\_rate)
6. # Passing global\_step to minimize() will increment it at each step.
7. optimizer.minimize(...my loss..., global\_step=global\_step)
8. tf.contrib.layers.l1\_regularizer(lamda)(W)和tf.contrib.layers.l2\_regularizer(lamda)(W)

L1正则和L2正则，解决过拟合问题



定义损失函数为



使用框架：

W=tf.Variable(tf.random\_normal([2,2],stedev=1,seed=1))

y=tf.matmul(x,W)

l2\_loss=tf.reduce\_mean(tf.square(y\_-y))+tf.contrib.l2\_regularizer(lambda)(W)

1. tf.train.ExponentialMovingAverage(decay, steps)

滑动平均模型，decay为衰减速率，steps为迭代次数

tf.train.ExponentialMovingAverage这个函数用于更新参数，就是采用滑动平均的方法更新参数。这个函数初始化需要提供一个衰减速率（decay），用于控制模型的更新速度。这个函数还会维护一个影子变量（也就是更新参数后的参数值），这个影子变量的初始值就是这个变量的初始值，影子变量值的更新方式如下：

shadow\_variable = decay \* shadow\_variable + (1-decay) \* variable

tf.train.ExponentialMovingAverage这个函数还提供了自动更新decay的计算方式：

decay= min（decay，（1+steps）/（10+steps））

使用框架：

1. v1 = tf.Variable(0, dtype=tf.float32)
2. step = tf.Variable(tf.constant(0))
3. ema = tf.train.ExponentialMovingAverage(0.99, step)
4. maintain\_average = ema.apply([v1])
5. sess.sun(maintain\_average)
6. tf.variable\_scope(name\_or\_scope,default\_name=None,values=None,initializer=None,regularizer=None,caching\_device=None,partitioner=None,custom\_getter=None,reuse=None,dtype=None)

常用：with tf.variable\_scope(“name”,reuse=True)

参数：

name\_or\_scope：string或VariableScope：要打开的范围。

default\_name：如果name\_or\_scope参数为None，则将使用默认名称，此名称将被唯一。 如果提供了name\_or\_scope，它将不会被使用，因此它不是必需的，可以是None。

值：传递给op函数的Tensor参数列表。

初始化器：此范围内的变量的默认初始化程序。

regularizer：此范围内的变量的默认正则符。

caching\_device：此范围内的变量的默认缓存设备。

partitioner：此范围内变量的默认分区。

custom\_getter：此范围内变量的默认定制getter。

reuse：True或None 如果是，我们进入该范围以及所有子范围的重用模式，，默认获取已经创建的变量; 如果没有，我们只是继承父范围重用。

dtype：在此范围中创建的变量类型（默认为传递范围中的类型，或从父范围继承）

1. tf.train.Saver类

tf.train.Saver(max\_to\_keep=None,keep\_checkpoint\_every\_n\_hour=No)

max\_to\_keep:  表明保存的最大checkpoint 文件数。当一个新文件创建的时候，旧文件就会被删掉。如果值为None或0，表示保存所有的checkpoint 文件。默认值为5（也就是说，保存最近的5个checkpoint 文件）。

keep\_checkpoint\_every\_n\_hour:  除了保存最近的max\_to\_keep checkpoint 文件，你还可能想每训练N小时保存一个checkpoint 文件。这将是非常有用的，如果你想分析一个模型在很长的一段训练时间内是怎么改变的。例如，设置 keep\_checkpoint\_every\_n\_hour=2 确保没训练2个小时保存一个checkpoint 文件。默认值10000小时无法看到特征。

tf.train.Saver.save(sess, save\_path, global\_step=None, latest\_filename=None, meta\_graph\_suffix='meta', write\_meta\_graph=True)

保存变量（可以持久化训练好的模型）

这个方法运行通过构造器添加的操作。它需要启动图的session。被保存的变量必须经过了初始化。

方法返回新建的checkpoint 文件的路径。路径可以直接传给restore() 进行调用。

参数：

sess:  用于保存变量的Session

save\_path:  checkpoint 文件的路径。如果saver 是共享的，这是共享checkpoint 文件名的前缀。

global\_step:  如果提供了global step number，将会追加到 save\_path 后面去创建checkpoint 的文件名。可选参数可以是一个Tensor，一个name Tensor或integer Tensor.

tf.train.Saver.restore(sess, save\_path)

恢复之前保存的变量

这个方法运行构造器为恢复变量所添加的操作。它需要启动图的Session。恢复的变量不需要经过初始化，恢复作为初始化的一种方法。

save\_path 参数是之前调用save() 的返回值，或调用 latest\_checkpoint() 的返回值。

参数：

sess:  用于恢复参数的Session

save\_path:  参数之前保存的路径

使用框架：

1. saver=tf.train.Saver() #生成Saver对象
2. saver.save(sess,path/model.ckpt) #保存模型
3. saver.restore(sess,path/model.ckpt) #重新载入模型

说明：

这段代码中，通过saver.save函数将TensorFlow模型保存到了save/model.ckpt文件中，这里代码中指定路径为"save/model.ckpt"，也就是保存到了当前程序所在文件夹里面的save文件夹中。

TensorFlow模型会保存在后缀为.ckpt的文件中。保存后在save这个文件夹中会出现3个文件，因为TensorFlow会将计算图的结构和图上参数取值分开保存。

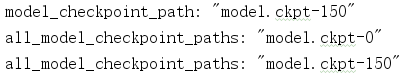
·checkpoint文件保存了一个目录下所有的模型文件列表，这个文件是tf.train.Saver类自动生成且自动维护的。在 checkpoint文件中维护了由一个tf.train.Saver类持久化的所有TensorFlow模型文件的文件名。当某个保存的TensorFlow模型文件被删除时，这个模型所对应的文件名也会从checkpoint文件中删除。checkpoint中内容的格式为CheckpointState Protocol Buffer.

·model.ckpt.meta文件保存了TensorFlow计算图的结构，可以理解为神经网络的网络结构   
TensorFlow通过元图（MetaGraph）来记录计算图中节点的信息以及运行计算图中节点所需要的元数据。TensorFlow中元图是由MetaGraphDef Protocol Buffer定义的。MetaGraphDef 中的内容构成了TensorFlow持久化时的第一个文件。保存MetaGraphDef 信息的文件默认以.meta为后缀名，文件model.ckpt.meta中存储的就是元图数据。

·model.ckpt文件保存了TensorFlow程序中每一个变量的取值，这个文件是通过SSTable格式存储的，可以大致理解为就是一个（key，value）列表。model.ckpt文件中列表的第一行描述了文件的元信息，比如在这个文件中存储的变量列表。列表剩下的每一行保存了一个变量的片段，变量片段的信息是通过SavedSlice Protocol Buffer定义的。SavedSlice类型中保存了变量的名称、当前片段的信息以及变量取值。TensorFlow提供了tf.train.NewCheckpointReader类来查看model.ckpt文件中保存的变量信息。

附：Tnesorflow加载模型汇总

1. checkpoint文件会记录保存信息，通过它可以定位最新保存的模型：
2. tf.train.get\_checkpoint\_state(‘./model/’)
3. **print**(ckpt.model\_checkpoint\_path)



.meta文件保存了当前图结构

.index文件保存了当前参数名

.data文件保存了当前参数值

（2）不加载图结构，只加载参数

由于实际上我们参数保存的都是Variable变量的值，所以其他的参数值（例如batch\_size）等，我们在restore时可能希望修改，但是图结构在train时一般就已经确定了，所以我们可以使用tf.Graph().as\_default()新建一个默认图（建议使用上下文环境），利用这个新图修改和变量无关的参值大小，从而达到目的。

1. # 只加载数据，不加载图结构，可以在新图中改变batch\_size等的值
2. # 不过需要注意，Saver对象实例化之前需要定义好新的图结构，否则会报错
3. saver = tf.train.Saver()
4. with tf.Session() as sess:
5. ckpt = tf.train.get\_checkpoint\_state('./model/')
6. saver.restore(sess,ckpt.model\_checkpoint\_path)

（3）加载图结构和参数

1. # 连同图结构一同加载
2. ckpt = tf.train.get\_checkpoint\_state('./model/')
3. saver = tf.train.import\_meta\_graph(ckpt.model\_checkpoint\_path +'.meta')
4. with tf.Session() as sess:
5. saver.restore(sess,ckpt.model\_checkpoint\_path)
6. tf.train.latest\_checkpoint(checkpoint\_dir)

checkpoint\_dir存放模型文件的文件夹路径

返回最新的模型文件的名称；例如，model-0

1. tf.train.import\_meta\_graph(meta\_graph\_file\_path)

meta\_graph\_file\_path：.meta文件的路径

返回一个Saver对象，用来加载保存好的模型

使用框架：

1. #加载元图和变量
2. saver=tf.train.import\_meta\_graph('{}.meta'.format(checkpoint\_file))
3. saver.restore(sess,checkpoint\_file)
4. graph.get\_tensor\_by\_name(tensorname)

tensorname：张量名称

说明：从图中通过名称获取张量

1. graph.get\_operation\_by\_name(openration\_name)

openration\_name：一个操作对应的张量名称

说明：从图中通过操作名称获取张量

使用框架：

1. #通过名称从图中获得输入占位符
2. input\_x=graph.get\_operation\_by\_name('BottleneckInputPlaceholder').outputs[0]