1、Tensorflow的张量

定义常量

A = tf.constant([1.0, 2.0], name = “a”)

产生新的计算图

g1 = tf.Graph()

with g1.as\_default():

…

张量所属的计算图

a.graph as tf.get\_default\_graph()

可为计算图指定CPU或GPU

g = tf.Graph()

with g.device(‘/gpu:0’):

…

集合列表

tf.GraphKeys.VARIABLES 所有变量

tf.GraphKeys.TRAINABLE\_VARIABLES 可学习的变量

tf.GraphKeys.SUMMARIES 日志生成相关的张量

tf.GraphKeys.QUEUE\_RUNNERS 处理输入的QueueRunner

tf.GraphKeys.MOVING\_AVERAGE\_VARIABLES 所有计算了滑动平均值的变量

2、会话

创建会话

sess = tf.Session()

sess.run(a)

sess.close()

或者上下文管理器来管理

with tf.Session() as sess:

sess.run(…)

创新交互式会话

sess = tf.InteractiveSession()

print(result.eval())

sess.close()

会话参数配置

config = tf.ConfigProto(allow\_soft\_placement=True,

log\_device\_placement=True)

sess1 = tf.InteractiveSession(config=config)

初始化所有变量

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

3、变量

w = tf.Variable(tf.random\_normal([2,3], stddev=2))

随机函数

tf.random\_normal 正太分布

tf.truncated\_normal 正太分布，如随机生成的值偏离平均值2个标准差，则这个值重新随机

tf.random\_uniform 平均分布

tf.random\_gamma Gamma分布

常量生成函数

tf.zeros 全0数组 tf.zeros([2,3], int32)

tf.ones 全1数组 tf.ones([2,3], int32)

tf.fill 填充指定数字

tf.constant 指定数字的常量

输入处理数据的占位符

tf.placeholder(tf.float32, shape=(1,2), name=’input’)

4、激活函数

ReLU激活函数 f(x) = max(x, 0)

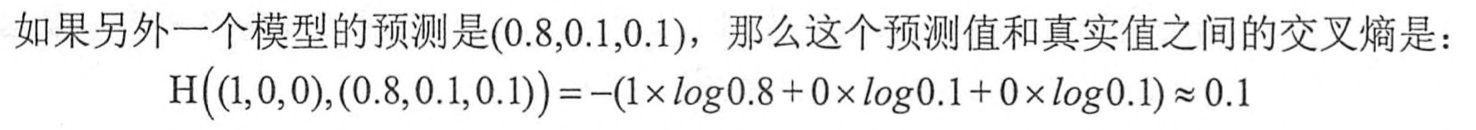
sigmoid激活函数 f(x) = 1/(1+e-x)

tanh激活函数 f(x) = (1-e-2x)/(1+e-2x)

5、经典损失函数

（1）分类问题使用交叉熵

交叉熵，刻画的是两个概率分布p,q之间的距离。

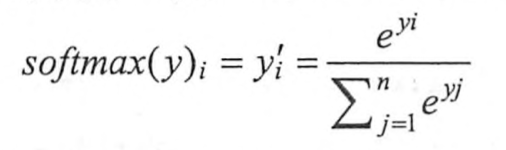


交叉熵实现代码：(y\_: 正确答案 y: 预测结果)

cross\_entropy = -tf.reduce\_mean(y\_ \* tf.log(tf.clip\_by\_value(y, 1e-10, 1.0)))

将神经网络前向传播结果转化为概率分布

softmax回归处理：



一般交叉熵与softmax回归一起使用

cross\_entropy = tf.nn.softmax\_cross\_entropy\_with\_logits(y, y\_)

在只一个正确答案的分类问题，使用

tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits来加快运算

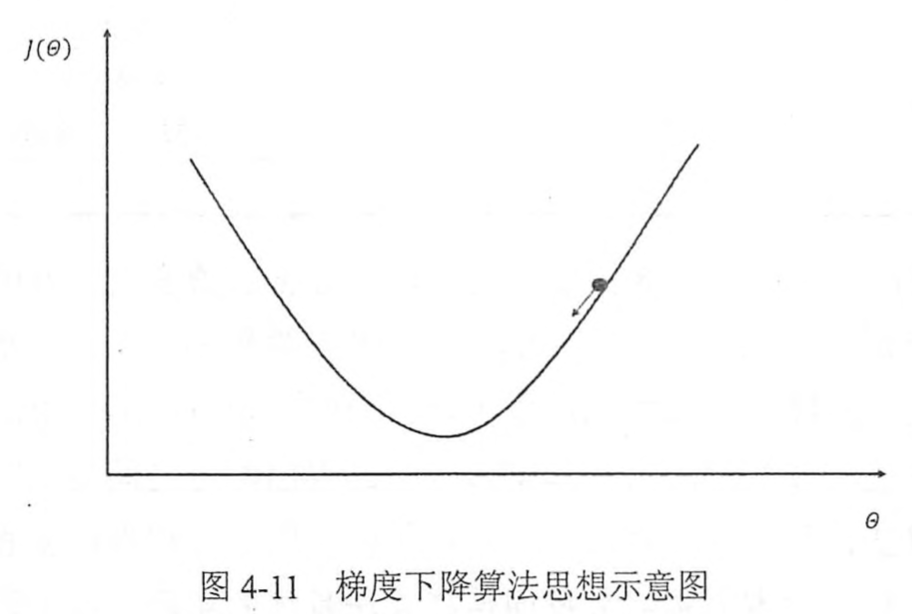
（2）回归问题解决对具体数值的预测

最常用的损失函数是均方误差MSE（mean squared error）

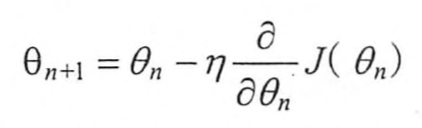
yi是批次中第i个数据正确答案，yi’是神经网络的预测结果。

mean=tf.reduce\_mean(tf.square(y\_ - y))

6、梯度下降算法



参数、损失函数J()，对于参数，其梯度为，定义一个学习率



定义了学习率的梯度下降算法：

train\_step = tf.train.AdamOptimizer(0.001).minimize(loss)

学习率的指数衰减算法：tf.train.exponential\_decay

开始较大学习率快速得到较优的解，迭代逐步减小学习率使得结果更加稳定。

decayed\_learning\_rate = learning\_rate \* decay\_rate ^ (global\_step / decay\_steps)

decayed\_learning\_rate: 每轮优化使用的学习率

learning\_rate: 初始学习率

decay\_rate: 衰减率

decay\_steps: 衰减速度

global\_step = tf.Variable(0, trainable=False)  
starter\_learning\_rate = 0.1  
learning\_rate = tf.train.exponential\_decay(starter\_learning\_rate, global\_step,100000, 0.96, staircase=True)  
# 将 global\_step 传入 minimize() will 将会在每一个step中增加它的值.  
learning\_step = (  
    tf.train.GradientDescentOptimizer(learning\_rate)  
    .minimize(...my loss..., global\_step=global\_step)  
)

7、过拟合问题

常用避免过拟合的方法是正则化，正则化的思想是在损失函数中加入刻画模型复杂程序的指标。刻画训练数据的损失函数是，优化时去优化，R(w)表示模型的复杂程序，表示复杂损失在总损失中的比例，表示所有参数，包括权重w和偏置b。

函数R(w)一般有L1正则化和L2正则化

L1正则化：

L2正则化：

L1正则化使得参数更加稀疏，使用得更多参数为零，达到特征选取的作用。L2正则化则使得部分参数变得更小，可以忽略不计。

实践中L1正则化和L2正则化同时使用

例：

loss = tf.reduce\_mean(tf.square(y\_ - y)) + tf.contrib.layers.l2\_regularizer(lambda)(w)

8、滑动平均模型

tf.train.ExponentialMovingAverage实现滑动平均模型

维护一个影子变量shadow\_variable，初始值和初始参数variable一样。decay是衰减率，一般是接近1的小数：如0.999。

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

ExponentialMovingAverage还提供num\_updates来动态调整decay的大小，衰减率是：

使用方法：

#定义num\_updates

num\_updates = tf.Variable(0, trainable=False)

ema = tf.train.ExponentialMovingAverage(0.999, num\_updates)

avg\_op = ema.apply([…需滑动平均的变量列表…])

#在trainstep之后运行avg\_op

sess.run(avg\_op)

9、变量管理

(1)以下两个语句等价：

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

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

(2)初始化函数：

tf.constant\_initializer：初始化为给定的常量

tf.random\_normal\_initializer: 初始化为正态分布的随机数

tf.truncated\_normal\_initializer：初始化为正态分布的随机数，若值偏离平均值2个标准差，则重新随机。

tf.random\_uniform\_initializer：初始化为平均分布的随机值

tf.uniform\_unit\_scaling\_initializer：初始化为平均分布的随机值

tf.zeros\_initializer：初始化为全0

tf.ones\_initializer：设置为全1

(3)命名空间tf.variable\_scope

with tf.variable\_scope(“foo”):

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

以下代码重用之前已经定义的变量

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

v1 = tf.get\_variable(“v”, [1])

即reuse=False或reuse=None，get\_variable将创建变量，否则将获取已创建的变量。

10、模型持久化

（1）保存

定义Saver类用于保存模型

saver = tf.train.Saver()

在会话中保存模型

saver.save(sess, ‘/path/to/model/model.ckpt’)

（2）恢复

saver = tf.train.Saver()

在会话中恢复

saver.restore(sess, ‘/path/to/model/model.ckpt’)

sess.run(…)

（3）恢复中直接加载持久化的运算图

saver=tf.train.import\_meta\_graph(‘/path/to/model/model.ckpt.meta’)

with tf.Session() as sess:

saver.restore(sess, ‘/path/to/model/model.ckpt’)

#获取张量

Sess.run(tf.get\_default\_graph().get\_tensor\_by\_name(‘add:0’)

（4）选择性保存张量

saver = tf.train.Saver([v1, v2]) #指定保存的张量列表

加载重命名

saver=tf.train.Saver({“v1”:v1, “v2”:v2})

11、卷积神经网络

（1）卷积神结网络组成结构

输入层：三维图像矩阵维度包括国像长、宽，深度表示色彩的通道数

卷积层：每一节点对应上一层的一小块，通常这个小块选择3x3或5x5。卷积层实际上是对上一层进行特征抽象。卷积层输出的深度会增加。

池化层：将上一层的矩阵池化，缩小矩阵大小，对于图像来说相当于降低分辨率，深度不变。

全连接层：多轮卷积层和池化层处理后，一般最后由1~2个全连接层给出结果。

softmax层：最后一层用于分类问题输出。

（2）卷积层

过滤器(filter)或内核(kernel)常用尺寸3x3、5x5，深度常见有32、64。

卷积乘法是点乘，即输入层与过滤器之间对应元素相乘，不是矩阵乘法。

卷积层参数个数＝输入层深度x卷积长x卷积宽x卷积深度 + 偏置深度个数

卷积层全0填充后输出的矩阵大小：

长度＝输入层长度 / 步长

宽度＝输入层宽度 / 步宽

不填充输出矩阵大小：

长度＝(输入层长度 – 过滤器长度 + 1) / 步长

宽度＝(输入层宽度 – 过滤器宽度 + 1) / 步宽

tf.nn.conv2d(input, filter\_weight, strides, padding)

input: 输入，有4个维度：第一维batch大小，第2~4输入的节点矩阵

filter\_weight: 权重：4个维度，长、宽、输入层深度、输出层深度

strides: 步数：第1、4个维度是固定1，第2、3定义长、宽方向的步数。

padding:SAME表示全0填充，VALID表示不填充