搭建神经网络

3.1

一、基本概念

√基于Tensorflow的NN：用张量表示数据，用计算图搭建神经网络，用会话执行计算图，优化线上的权重（参数），得到模型。

√张量：张量就是多维数组（列表），用“阶”表示张量的维度。

0阶张量称作标量，表示一个单独的数；

举例S=123

1阶张量称作向量，表示一个一维数组；

举例V=[1,2,3]

2阶张量称作矩阵，表示一个二维数组，它可以有i行j列个元素，每个元素可以用行号和列号共同索引到；

举例m=[[1, 2, 3], [4, 5, 6], [7, 8,9]]

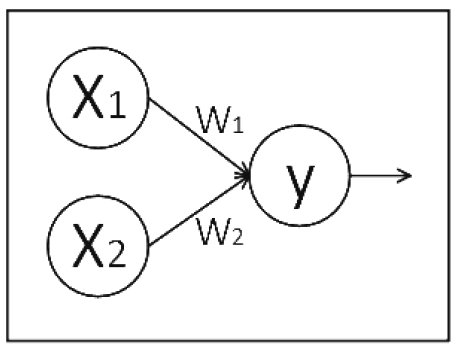
判断张量是几阶的，就通过张量右边的方括号数，0个是0阶，n个是n阶，张量可以表示0阶到n阶数组（列表）；

举例t=[ [ […] ] ]为3阶。

√数据类型：Tensorflow的数据类型有tf.float32、tf.int32等。

√计算图（Graph）：搭建神经网络的计算过程，是承载一个或多个计算节点的一张图，只搭建网络，不运算。

举例



在第一讲中我们曾提到过，神经网络的基本模型是神经元，神经元的基本模型其实就是数学中的乘、加运算。我们搭建如下的计算图：

x1、x2表示输入，w1、w2分别是x1到y和x2到y的权重，y=x1\*w1+x2\*w2。

3.2

一、神经网络的参数

√神经网络的参数：是指神经元线上的权重w，用变量表示，一般会先随机生成这些参数。生成参数的方法是让w等于tf.Variable，把生成的方式写在括号里。

神经网络中常用的生成随机数/数组的函数有：

tf.random\_normal() 生成正态分布随机数

tf.truncated\_normal() 生成去掉过大偏离点的正态分布随机数

tf.random\_uniform() 生成均匀分布随机数

tf.zeros 表示生成全0数组

tf.ones 表示生成全1数组

tf.fill 表示生成全定值数组

tf.constant 表示生成直接给定值的数组

二、神经网络的搭建

当我们知道张量、计算图、会话和参数后，我们可以讨论神经网络的实现过程了。

√神经网络的实现过程：

1、准备数据集，提取特征，作为输入喂给神经网络（Neural Network，NN）

2、搭建NN结构，从输入到输出（先搭建计算图，再用会话执行）

（NN前向传播算法计算输出）

3、大量特征数据喂给NN，迭代优化NN参数

（NN反向传播算法优化参数训练模型）

4、使用训练好的模型预测和分类

由此可见，基于神经网络的机器学习主要分为两个过程，即训练过程和使用过程。训练过程是第一步、第二步、第三步的循环迭代，使用过程是第四步，一旦参数优化完成就可以固定这些参数，实现特定应用了。

很多实际应用中，我们会先使用现有的成熟网络结构，喂入新的数据，训练相应模型，判断是否能对喂入的从未见过的新数据作出正确响应，再适当更改网络结构，反复迭代，让机器自动训练参数找出最优结构和参数，以固定专用模型。

三、前向传播

√前向传播就是搭建模型的计算过程，让模型具有推理能力，可以针对一组输入给出相应的输出。

√前向传播过程的tensorflow描述：

√变量初始化、计算图节点运算都要用会话（with结构）实现

with tf.Session() as sess:

sess.run()

√变量初始化：在sess.run函数中用tf.global\_variables\_initializer()汇总所有待优化变量。

init\_op = tf.global\_variables\_initializer()

sess.run(init\_op)

√计算图节点运算：在sess.run函数中写入待运算的节点

sess.run(y)

√用tf.placeholder 占位，在sess.run 函数中用feed\_dict喂数据

喂一组数据：

x = tf.placeholder(tf.float32, shape=(1, 2))

sess.run(y, feed\_dict={x: [[0.5,0.6]]})

喂多组数据：

x = tf.placeholder(tf.float32, shape=(None, 2))

sess.run(y, feed\_dict={x: [[0.1,0.2],[0.2,0.3],[0.3,0.4],[0.4,0.5]]})

3.3

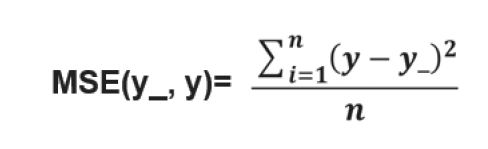
一、反向传播

√反向传播：训练模型参数，在所有参数上用梯度下降，使NN模型在训练数据上的损失函数最小。

√损失函数（loss）：计算得到的预测值y与已知答案y\_的差距。

损失函数的计算有很多方法，均方误差MSE是比较常用的方法之一。

√均方误差MSE：求前向传播计算结果与已知答案之差的平方再求平均。



用tensorflow函数表示为：

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

√反向传播训练方法：以减小loss值为优化目标，有梯度下降、momentum优化器、adam优化器等优化方法。

这三种优化方法用tensorflow的函数可以表示为：

train\_step=tf.train.GradientDescentOptimizer(learning\_rate).minimize(loss)

train\_step=tf.train.MomentumOptimizer(learning\_rate, momentum).minimize(loss)

train\_step=tf.train.AdamOptimizer(learning\_rate).minimize(loss)

三种优化方法区别如下：

1. tf.train.GradientDescentOptimizer()使用随机梯度下降算法，使参数沿着梯度的反方向，即总损失减小的方向移动，实现更新参数。

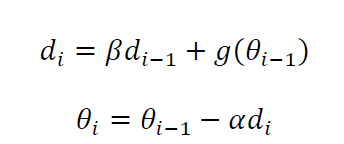


参数更新公式是



其中，𝐽(𝜃)为损失函数，𝜃为参数，𝛼为学习率。

1. tf.train.MomentumOptimizer()在更新参数时，利用了超参数，参数更新公式是



其中，𝛼为学习率，超参数为𝛽，𝜃为参数，𝑔(𝜃𝑖−1)为损失函数的梯度。

③tf.train.AdamOptimizer()是利用自适应学习率的优化算法，Adam算法和随机梯度下降算法不同。随机梯度下降算法保持单一的学习率更新所有的参数，学习率在训练过程中并不会改变。而Adam 算法通过计算梯度的一阶矩估计和二阶矩估计而为不同的参数设计独立的自适应性学习率。

√学习率：决定每次参数更新的幅度。

优化器中都需要一个叫做学习率的参数，使用时，如果学习率选择过大会出现震荡不收敛的情况，如果学习率选择过小，会出现收敛速度慢的情况。我们可以选个比较小的值填入，比如0.01、0.001。

二、搭建神经网络的八股

我们最后梳理出神经网络搭建的八股，神经网络的搭建课分四步完成：准备工作、前向传播、反向传播和循环迭代。

√0.导入模块，生成模拟数据集；

import

常量定义

生成数据集

√1.前向传播：定义输入、参数和输出

x= y\_=

w1= w2=

a= y=

√2.反向传播：定义损失函数、反向传播方法

loss=

train\_step=

√3.生成会话，训练STEPS轮

with tf.session() as sess

Init\_op=tf. global\_variables\_initializer()

sess\_run(init\_op)

STEPS=3000

for i in range(STEPS):

start=

end=

sess.run(train\_step, feed\_dict:)

4.1

√神经元模型：用数学公式表示为：𝐟(Σ𝒊𝒙𝒊𝒘𝒊+𝐛)，f为激活函数。神经网络是以神经元为基本单元构成的。

√激活函数：引入非线性激活因素，提高模型的表达力。

常用的激活函数有relu、sigmoid、tanh等。

1. 激活函数relu:在Tensorflow中，用tf.nn.relu()表示



relu()数学表达式 relu()数学图形

1. 激活函数sigmoid：在Tensorflow中，用tf.nn.sigmoid()表示



sigmoid()数学表达式 sigmoid()数学图形

1. 激活函数tanh：在Tensorflow中，用tf.nn.tanh()表示



tanh()数学表达式 tanh()数学图形

√神经网络的复杂度：可用神经网络的层数和神经网络中待优化参数个数表示

√神经网路的层数：一般不计入输入层，层数=n个隐藏层+ 1个输出层

√神经网路待优化的参数：神经网络中所有参数w的个数+ 所有参数b的个数

例如：

输入层 隐藏层 输出层



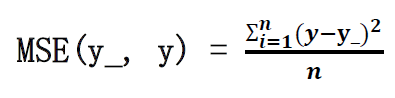
在该神经网络中，包含1个输入层、1个隐藏层和1个输出层，该神经网络的层数为2层。

在该神经网络中，参数的个数是所有参数w的个数加上所有参数b的总数，第一层参数用三行四列的二阶张量表示（即12个线上的权重w）再加上4个偏置b；第二层参数是四行两列的二阶张量（）即8个线上的权重w）再加上2个偏置b。总参数=3\*4+4 + 4\*2+2 = 26。

√**损失函数（**loss**）：**用来表示**预测值（**y**）与已知答案（**y\_**）的差距。**在训练神经网络时，通过不断改变神经网络中所有参数，使损失函数不断减小，从而训练出更高准确率的神经网络模型**。**

√常用**的**损失函数**有均方误差、自定义和交叉熵**等。

√均方误差mse：n个样本**的**预测值y**与已知**答案y\_**之差的**平方和，再求平均值。



在Tensorflow中用loss\_mse = tf.reduce\_mean(tf.square(y\_ -y))

√自定义损失函数：**根据**问题的**实际情况，定制**合理的**损失函数**。

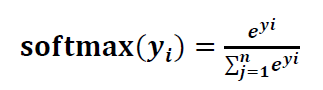
√交叉熵**(Cross Entropy)**：表示两个概率分布之间的距离。交叉熵越大，两个概率分布距离越远，两个概率分布越相异；交叉熵越小，两个概率分布距离越近，两个概率分布越相似。

交叉熵计算公式：𝐇(𝐲\_ ,𝐲)=−Σ𝐲\_∗𝒍𝒐𝒈 𝒚

用Tensorflow函数表示为

ce= -tf.reduce\_mean(y\_\* tf.log(tf.clip\_by\_value(y, 1e-12, 1.0)))

√**softmax**函数：将**n**分类的**n**个输出（**y1,y2…yn**）变为满足以下概率分布要求的函数。∀𝐱 𝐏(𝐗=𝐱)∈[𝟎,𝟏] 且Σ𝑷𝒙(𝑿=𝒙)=𝟏

softmax函数表示为：

softmax函数应用：在n分类中，模型会有n个输出，即y1,y2…yn，其中yi表示第i种情况出现的可能性大小。将n个输出经过softmax函数，可得到符合概率分布的分类结果。

√在**Tensorflow**中，一般让模型的输出经过**sofemax**函数，以获得输出分类的概率分布，再与标准答案对比，求出交叉熵，得到损失函数，用如下函数实现：

**ce = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=y, labels=tf.argmax(y\_, 1))**

**cem = tf.reduce\_mean(ce)**

4.2

√学习率learning\_rate：表示了每次参数更新的幅度大小。学习率过大，会导致待优化的参数在最小值附近波动，不收敛；学习率过小，会导致待优化的参数收敛缓慢。

在训练过程中，参数的更新向着损失函数梯度下降的方向。

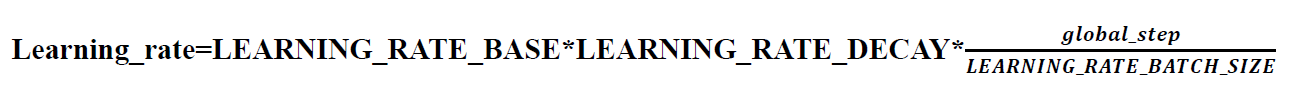
参数的更新公式为：

𝒘𝒏+𝟏= 𝒘𝒏−𝒍𝒆𝒂𝒓𝒏𝒊𝒏𝒈\_𝒓𝒂𝒕𝒆𝛁

√学习率的设置

学习率过大，会导致待优化的参数在最小值附近波动，不收敛；学习率过小，会导致待优化的参数收敛缓慢。

√指数衰减学习率：学习率随着训练轮数变化而动态更新

学习率计算公式如下：用**Tensorflow**的函数表示为：

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

**learning\_rate = tf.train.exponential\_decay( LEARNING\_RATE\_BASE, global\_step, LEARNING\_RATE\_STEP, LEARNING\_RATE\_DECAY, staircase=True/False)**

其中，LEARNING\_RATE\_BASE为学习率初始值，LEARNING\_RATE\_DECAY为学习率衰减率,global\_step记录了当前训练轮数，为不可训练型参数。学习率learning\_rate更新频率为输入数据集总样本数除以每次喂入样本数。若staircase设置为True时，表示global\_step/learning rate step取整数，学习率阶梯型衰减；若staircase设置为false时，学习率会是一条平滑下降的曲线。

4.3

√滑动平均：记录了一段时间内模型中所有参数w和b各自的平均值。利用滑动平均值可以增强模型的泛化能力。

√滑动平均值（影子）计算公式：

影子=衰减率\* 影子+（1-衰减率）\* 参数

其中，衰减率=𝐦𝐢𝐧{𝑴𝑶𝑽𝑰𝑵𝑮𝑨𝑽𝑬𝑹𝑨𝑮𝑬𝑫𝑬𝑪𝑨𝒀,𝟏+轮数𝟏𝟎+轮数}，影子初值=参数初值

√用Tesnsorflow函数表示为：

√ema = tf.train.ExponentialMovingAverage(MOVING\_AVERAGE\_DECAY，global\_step)

其中，MOVING\_AVERAGE\_DECAY表示滑动平均衰减率，一般会赋接近1的值，global\_step表示当前训练了多少轮。

√ema\_op = ema.apply(tf.trainable\_variables())

其中，ema.apply()函数实现对括号内参数求滑动平均，tf.trainable\_variables()函数实现把所有待训练参数汇总为列表。

√with tf.control\_dependencies([train\_step, ema\_op]):

train\_op = tf.no\_op(name='train')

其中，该函数实现将滑动平均和训练过程同步运行。

查看模型中参数的平均值，可以用ema.average()函数。

例如：

在神经网络模型中，将MOVING\_AVERAGE\_DECAY设置为0.99，参数w1设置为0，w1的滑动平均值设置为0。

①开始时，轮数global\_step设置为0，参数w1更新为1，则w1的滑动平均值为：

w1滑动平均值=min(0.99,1/10)\*0+(1–min(0.99,1/10)\*1 = 0.9

③ 当轮数global\_step设置为100时，参数w1更新为10，以下代码global\_step保持为100，每次执行滑动平均操作影子值更新，则滑动平均值变为：

w1滑动平均值=min(0.99,101/110)\*0.9+(1–min(0.99,101/110)\*10 = 0.826+0.818=1.644

③再次运行，参数w1更新为1.644，则滑动平均值变为：

w1滑动平均值=min(0.99,101/110)\*1.644+(1–min(0.99,101/110)\*10 = 2.328

④再次运行，参数w1更新为2.328，则滑动平均值：

w1滑动平均值=2.956

4.4

√过拟合：神经网络模型在训练数据集上的准确率较高，在新的数据进行预测或分类时准确率较低，说明模型的泛化能力差。

√正则化：在损失函数中给每个参数w加上权重，引入模型复杂度指标，从而抑制模型噪声，减小过拟合。

使用正则化后，损失函数loss变为两项之和：

loss = loss(y与y\_)+REGULARIZER\*loss(w)

其中，第一项是预测结果与标准答案之间的差距，如之前讲过的交叉熵、均方误差等；第二项是正则化计算结果。

√正则化计算方法：

1. L1正则化：𝒍𝒐𝒔𝒔𝑳𝟏=Σ𝒊|𝒘𝒊|

用Tesnsorflow函数表示:

loss(w) =tf.contrib.layers.l1\_regularizer(REGULARIZER)(w)

1. L2正则化：𝒍𝒐𝒔𝒔𝑳𝟐=Σ𝒊|𝒘𝒊|𝟐

用Tesnsorflow函数示:

loss(w)=tf.contrib.layers.l2\_regularizer(REGULARIZER)(w)

√用Tesnsorflow函数实现正则化：

tf.add\_to\_collection('losses', tf.contrib.layers.l2\_regularizer(regularizer)(w)

loss = cem + tf.add\_n(tf.get\_collection('losses'))

cem的计算已在4.1节中给出。

例如：

用300个符合正态分布的点X[x0, x1]作为数据集，根据点X[x0, x1]计算生成标注Y\_，将数据集标注为红色点和蓝色点。

标注规则为：当x02+ x12< 2 时，y\_=1，标注为红色；当x02+ x12≥2 时，y\_=0，标注为蓝色。

我们分别用无正则化和有正则化两种方法，拟合曲线，把红色点和蓝色点分开。在实际分类时，如果前向传播输出的预测值y接近1则为红色点概率越大，接近0则为蓝色点概率越大，输出的预测值y为0.5是红蓝点概率分界线。

在本例子中，我们使用了之前未用过的模块与函数：

√matplotlib模块：Python中的可视化工具模块，实现函数可视化

终端安装指令：sudo pip install matplotlib

√函数plt.scatter（）：利用指定颜色实现点(x,y)的可视化

plt.scatter (x坐标, y坐标, c=”颜色”)

plt.show()

√收集规定区域内所有的网格坐标点：

xx, yy = np.mgrid[起:止:步长, 起:止:步长]#找到规定区域以步长为分辨率的行列网格坐标点

grid = np.c\_[xx.ravel(), yy.ravel()] #收集规定区域内所有的网格坐标点

√plt.contour()函数：告知x、y坐标和各点高度，用levels指定高度的点描上颜色

plt.contour (x轴坐标值, y轴坐标值, 该点的高度, levels=[等高线的高度])

plt.show()

4.5搭建模块化神经网络八股

√前向传播：由输入到输出，搭建完整的网络结构

描述前向传播的过程需要定义三个函数：

√**def forward(x, regularizer):**

**w=**

**b=**

**y=**

**return y**

第一个函数**forward()**完成网络结构的设计，从输入到输出搭建完整的网络结构，实现前向传播过程。该函数中，参数x为输入，regularizer为正则化权重，返回值为预测或分类结果y。

√**def get\_weight(shape, regularizer):**

w = tf.Variable( )

tf.add\_to\_collection('losses', tf.contrib.layers.l2\_regularizer(regularizer)(w))

return w

第二个函数get\_weight()对参数w设定。该函数中，参数shape表示参数w的形状，regularizer表示正则化权重，返回值为参数w。其中，tf.variable()给w赋初值，tf.add\_to\_collection()表示将参数w正则化损失加到总损失losses中。

√def get\_bias(shape):

b = tf.Variable( )

return b

第三个函数get\_bias()对参数b进行设定。该函数中，参数shape表示参数b的形状,返回值为参数b。其中，tf.variable()表示给b赋初值。

√反向传播：训练网络，优化网络参数，提高模型准确性。

√**def backward( ):**

**x = tf.placeholder( )**

**y\_ = tf.placeholder( )**

**y = forward.forward(x, REGULARIZER)**

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

**loss =**

函数backward()中，placeholder()实现对数据集x和标准答案y\_占位，forward.forward()实现前向传播的网络结构，参数global\_step表示训练轮数，设置为不可训练型参数。

在训练网络模型时，常将正则化、指数衰减学习率和滑动平均这三个方法作为模型优化方法。

√在Tensorflow中，正则化表示为：

首先，计算预测结果与标准答案的损失值

①MSE：**y**与**y\_**的差距**(loss\_mse) = tf.reduce\_mean(tf.square(y-y\_))**

②交叉熵：**ce = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=y, labels=tf.argmax(y\_, 1))**

**y**与**y\_**的差距**(cem) = tf.reduce\_mean(ce)**

③自定义：**y**与**y\_**的差距

其次，总损失值为预测结果与标准答案的损失值加上正则化项

**loss = y**与**y\_**的差距**+ tf.add\_n(tf.get\_collection('losses'))**

√在Tensorflow中，指数衰减学习率表示为：

learning\_rate = tf.train.exponential\_decay(

LEARNING\_RATE\_BASE,

global\_step,

数据集总样本数/ BATCH\_SIZE,

LEARNING\_RATE\_DECAY,

staircase=True)

train\_step=tf.train.GradientDescentOptimizer(learning\_rate).minimize(loss, global\_step=global\_step)

√在Tensorflow中，滑动平均表示为：

ema = tf.train.ExponentialMovingAverage(MOVING\_AVERAGE\_DECAY, global\_step)

ema\_op = ema.apply(tf.trainable\_variables())

with tf.control\_dependencies([train\_step, ema\_op]):

train\_op = tf.no\_op(name='train')

其中，滑动平均和指数衰减学习率中的global\_step为同一个参数。

√用with结构初始化所有参数

with tf.Session() as sess:

init\_op = tf.global\_variables\_initializer()

sess.run(init\_op)

for i in range(STEPS):

sess.run(train\_step, feed\_dict={x: , y\_: })

if i % 轮数== 0:21

print

其中，with结构用于初始化所有参数信息以及实现调用训练过程，并打印出loss值。

√判断python运行文件是否为主文件

if \_\_name\_\_=='\_\_main\_\_':

backward()

该部分用来判断python运行的文件是否为主文件。若是主文件，则执行backword()函数。