# 前向传播：

变量初始化，计算图节点运，算都要用绘画（with结构）实现

with tf.Session() as sess:

Sess.run()

变量初始化：在sess.run 函数中使用tf.global\_variables\_initializer()

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)) //2表示有两个特征

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]]})

**例子1：**

#两层简单神经网络(全连接)

Import tensorflow as tf

#定义输入和参数

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

x=tf.constant([[0.7,0.75]])

w1=tf.Variable(tf.random\_normal([2,3],stddev=1,seed=1))

w2=tf.Variable(tf.random\_normal([3,1],stddev=1,seed=1))

#定义前向传播过程

a=tf.matmul(x,w1)

y=tf.matmul(a,w2)

#用会话计算结果

with tf.Session() as sess:

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

sess.run(init\_op)

print(“y is :\n”, sess.run(y))

**例子2：**

#两层简单神经网络(全连接)

Import tensorflow as tf

#定义输入和参数

#用placeholder实现输入定义（sess.run 中喂一组数据）

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

w1=tf.Variable(tf.random\_normal([2,3],stddev=1,seed=1))

w2=tf.Variable(tf.random\_normal([3,1],stddev=1,seed=1))

#定义前向传播过程

a=tf.matmul(x,w1)

y=tf.matmul(a,w2)

#用会话计算结果

with tf.Session() as sess:

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

sess.run(init\_op)

print(“y is :\n”, sess.run(y, feed\_dict={x: [[0.7,0.5]]}))

**例子3：**

#两层简单神经网络(全连接)

Import tensorflow as tf

#定义输入和参数

#用placeholder实现输入定义（sess.run 中喂多组数据）

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

w1=tf.Variable(tf.random\_normal([2,3],stddev=1,seed=1))

w2=tf.Variable(tf.random\_normal([3,1],stddev=1,seed=1))

#定义前向传播过程

a=tf.

matmul(x,w1)

y=tf.matmul(a,w2)

#用会话计算结果

with tf.Session() as sess:

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

sess.run(init\_op)

print(“result is :\n”, sess.run(y, feed\_dict={x: [[0.7,0.5],[0.2,0.3],[0.3,0.4],[0.4,0.5]]}))

print(“w1:\n”, sess.run(w1))

print(“w2:\n”, sess.run(w2))

# 反向传播：

1. 训练模型参数，在所有参数上用梯度下降，使NN模型在训练数据删的损失函数最小
2. 损失函数（loss）：预测值（y）与已知答案（y\_）的差距
3. 均方误差MSE: MSE(y\_,y) = ^2 /n

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

1. 反向传播训练方法：以减小loss值作为优化目标

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)

1. 学习率：决定参数每次更新的幅度

**例子：**

Import tensorflow as tf

Import numpy as np

BATCH\_SIZE=8

Seed=23455

#基于seed产生随机数

rng = np.random.RandomState(seed)

#随机数返回32行2列的矩阵 表示32组 体积和重量 作为输入数据集

X=rng.rand(32,2)

#从X这个32行2列的矩阵中取出一行 判断如果和小于1给Y赋值1 如果和不小于1给Y赋值0

#作为输入数据集的标签（正确答案）

Y=[[int(x0 + x1<1)] for (x0,x1) in X]

Print(“X: \n”, X)

Print(“Y: \n”, Y)

#1定义神经网络的输入，参数和输出，定义前向传播过程

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

y\_ = tf.placeholder(tf.float32,shape=(None,1))

w1 = tf.Variable(tf.random\_normal([2,3],stddev=1,seed=1))

w2= tf.Variable(tf.random\_normal([3,1],stddev=1,seed=1))

a = tf.matmul(x,w1)

y = tf.matmul(a,w2)

#2定义损失函数及反向传播方法

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

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

#train\_step = tf.train.MomentumOptimizer (0.001,0.9).minimize(loss)

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

#3生成会话，训练STEPS轮

with tf.Session() as sess:

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

sess.run(init\_op)

#输出目前（未经训练）的参数取值

print(“w1:\n”, sess.run(w1))

print(“w2:\n”, sess.run(w2))

#训练模型

STEPS=3000

for I in range(STEPS):

start = (i\*BATCH\_SIZE) %32

end = start +BATCH\_SIZE

sess.run(train\_step, feed\_dict={x: X[start:end], y\_: Y[start:end]})

if i % 500 ==0:

total\_loss=sess.run(loss, feed\_dict={x:X, y\_:Y})

print(“After %d training step(s), loss on all data is %g” %(i , total\_loss))

#输出训练后的参数取值

print(“\n”)

print(“w1:\n”,sess.run(w1))

print(“w2:\n”,sess.run(w2))

**总结：**

**搭建神经网络的八股：准备，前传，反传，迭代**

**0 准备 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)**

# 损失函数

1. 损失函数loss

损失函数：（loss）:预测值（y）与已知答案（y\_）的差距

NN优化目标：loss最小：——》

mse(Mean Squared Error)

自定义

ce(Cross Entropy交叉熵)

1. 均方误差mse : MSE(y\_,y) = ^2 /n

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

**例子：**

#预测多或预测少的影响一样

#0导入模块，生成数据集

import tensorflow as tf

import numpy as np

BATCH\_SIZE=8

SEED=23455

rdm=np.random.RandomState(SEED)

X=rdm.rand(32,2)

Y\_=[[x1+x2+(rdm.rand()/10.0-0.05)] for (x1,x2) in X]

#1定义神经网络的输入，参数和输出，定义前向传播过程

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

y\_=tf.placeholder(tf.float32,shape=(None,1))

w1=tf.Variable(tf.random\_normal([2,1],stddev=1,seed=1))

y=tf.matmul(x,w1)

#2定义损失函数及反向传播方法

#定义损失函数为MSE，反向传播方法为梯度下降

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

train\_step=tf.train.GradientDescentOptimizer(0.001).minimize(loss\_mse)

#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=20000

for i in range(STEPS):

start = (i \* BATCH\_SIZE) %32

end = (i \* BATCH\_SIZE) %32 +BATCH\_SIZE

sess.run(train\_step, feed\_dict={x:X[start:end],y\_:Y\_[start:end]})

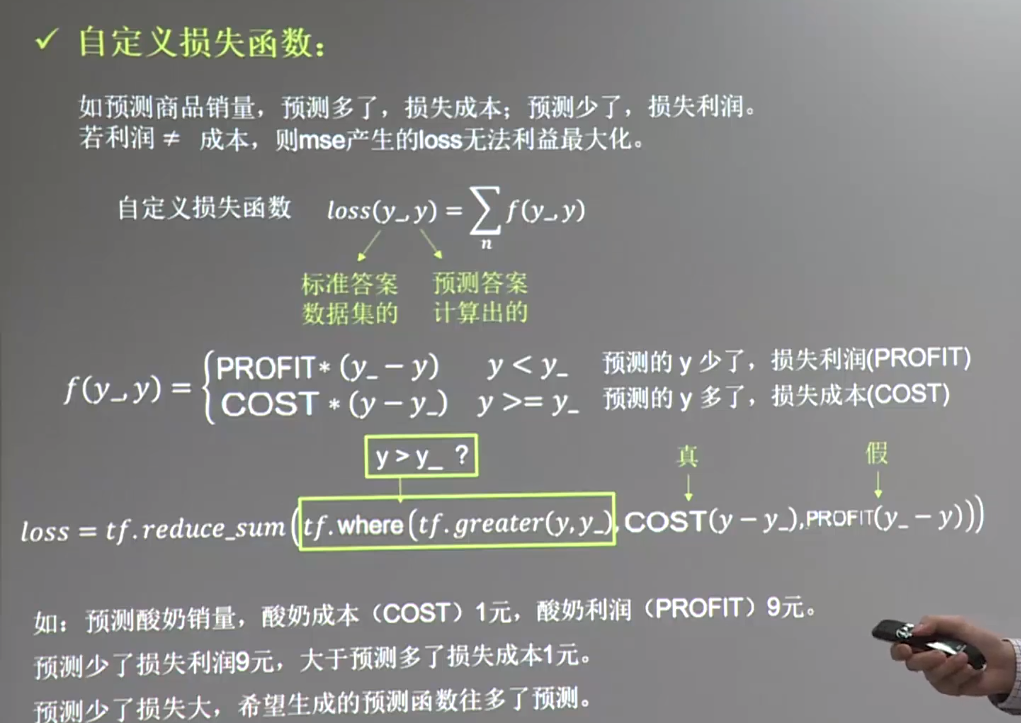
if i %500==0:

print(“After %d training steps, w1 is:” %(i))

print(sess.run(w1) “\n”)

print(“Final w1 is : \n”, sess.run(w1))

## 自定义损失函数：



**例子1：**

#酸奶成本1元，酸奶利润9元

#预测少了损失大，故不要预测少，故生成的模型会多预测一些

#0导入模块，生成数据集

import tensorflow as tf

import numpy as np

BATCH\_SIZE=8

SEED=23455

**COST=1**

**PROFIT=9**

rdm=np.random.RandomState(SEED)

X=rdm.rand(32,2)

Y\_=[[x1+x2+(rdm.rand()/10.0-0.05)] for (x1,x2) in X]

#1定义神经网络的输入，参数和输出，定义前向传播过程

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

y\_=tf.placeholder(tf.float32,shape=(None,1))

w1=tf.Variable(tf.random\_normal([2,1],stddev=1,seed=1))

y=tf.matmul(x,w1)

#2定义损失函数及反向传播方法

**#定义损失函数使得预测少了的损失大，于是模型应该偏向多的方向预测**

**loss=tf.reduce\_sum(tf.where(tf.greater(y,y\_),(y-y\_)\*COST,(y\_ -y)\*PROFIT))**

**train\_step=tf.train.GradientDescentOptimizer(0.001).minimize(loss)**

#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=20000

for i in range(STEPS):

start = (i \* BATCH\_SIZE) %32

end = (i \* BATCH\_SIZE) %32 +BATCH\_SIZE

sess.run(train\_step, feed\_dict={x:X[start:end],y\_:Y\_[start:end]})

if i %500==0:

print(“After %d training steps, w1 is:” %(i))

print(sess.run(w1) “\n”)

print(“Final w1 is : \n”, sess.run(w1))

**例子2：**

#酸奶成本1元，酸奶利润9元

#预测多了损失大，故不要预测多，故生成的模型会少预测一些

#0导入模块，生成数据集

import tensorflow as tf

import numpy as np

BATCH\_SIZE=8

SEED=23455

**COST=9**

**PROFIT=1**

rdm=np.random.RandomState(SEED)

X=rdm.rand(32,2)

Y\_=[[x1+x2+(rdm.rand()/10.0-0.05)] for (x1,x2) in X]

#1定义神经网络的输入，参数和输出，定义前向传播过程

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

y\_=tf.placeholder(tf.float32,shape=(None,1))

w1=tf.Variable(tf.random\_normal([2,1],stddev=1,seed=1))

y=tf.matmul(x,w1)

#2定义损失函数及反向传播方法

**#定义损失函数使得预测少了的损失大，于是模型应该偏向多的方向预测**

**loss=tf.reduce\_sum(tf.where(tf.greater(y,y\_),(y-y\_)\*COST,(y\_ -y)\*PROFIT))**

**train\_step=tf.train.GradientDescentOptimizer(0.001).minimize(loss)**

#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=20000

for i in range(STEPS):

start = (i \* BATCH\_SIZE) %32

end = (i \* BATCH\_SIZE) %32 +BATCH\_SIZE

sess.run(train\_step, feed\_dict={x:X[start:end],y\_:Y\_[start:end]})

if i %500==0:

print(“After %d training steps, w1 is:” %(i))

print(sess.run(w1) “\n”)

print(“Final w1 is : \n”, sess.run(w1))

## 交叉熵ce(Cross Entropy)；表征两个概率分布之间的距离

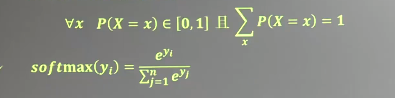
H(y\_ ,y) = -

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

**说明：y小于1e-12为1e-12**

**大于1.0为1.0**

当n分类的n个输出（y1,y2,…,yn）通过softmax()函数便满足了概率分布要求：



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

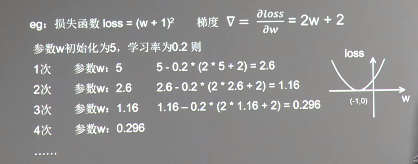
cem=tf.reduce\_mean(ce)

# 学习率：

learning\_rate:每次参数更新的幅度

wn+1 = wn – learning\_rate

更新后的参数=当前更参数 – 学习率 乘以 损失函数的梯度（导数）

例子：

**代码：**

#设损失函数loss=(w+1)^2, 令w初值是常数5，反向传播就是求最优w,即求最小loss对应的w值

import tensorflow as tf

#定义待优化参数w初值赋5

w=tf.Variable(tf.constant(5, dtype=tf.float32))

#定义损失函数loss

loss=tf.square(w+1)

#定义反向传播方法

train\_step=tf.train.GradentDescentOptimizer(**0.2**).minimize(loss)

#生成会话，训练40轮

with tf.Session() as sess:

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

sess.run(init\_op)

w\_val=sess.run(w)

loss\_val=sess.run(loss)

print(“After %s steps: w is %f, loss is %f.” % (i, w\_val, loss\_val))

1. 学习率大了振荡不收敛，学习率小了收敛速度满

**指数衰减学习率：**

learning\_rate = LEARNING\_RATE\_BASE \*

学习率基数，学习率初始值， 学习率衰减率（0，1） ，global\_step:运行了几轮BATCH\_SIZE

LEARNING\_RATE\_STPE:多少率更新一次学习率 =总样本数/BATCH\_SIZE

global\_step=tf.Variable(0, trainable=False)**#非训练，所以未false**

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

LEARNING\_RATE\_BASE,

global\_step,

LEARNING\_RATE\_STEP,

LEARNING\_RATE\_DECAY,

staircase = True)

**注：staircase=True,学习率阶梯型衰减 false：学习率平滑下降的曲线**

**代码：**

#设损失函数loss=(w+1)^2, 令w初值是常数5，反向传播就是求最优w,即求最小loss对应的w值

#使用指数衰减的学习率，在迭代初期得到较高的下降速度，可以在较小的训练轮数下取得更有收敛度

import tensorflow as tf

LEARNING\_RATE\_BASE=0.1 最初学习率

LEARNING\_RATE\_DECAY=0.99学习率衰减率

LEARNING\_RATE\_STEP=1 #喂入多少轮BATCH\_SIZE后，更新一次学习率，一般设为：总样本数/BATCH\_SIZE

#运行了几轮BATCH\_SIZE的计数器，初值给0，设为不被训练

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

#定义指数下降学习率

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

#定义待优化参数，初值给5

w=tf.Variable(tf.constant(5, dtype=tf.float32))

#定义损失函数loss

loss=tf.square(w+1)

#定义反向传播方法

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

#生成会话，训练40轮

with tf.Session() as sess:

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

sess.run(init\_op)

for i in range(40):

sess.run(train\_step)

learning\_rate\_val=sess.run(learning\_rate)

global\_step\_val=sess.run(global\_step)

w\_val=sess.run(w)

loss\_val=sess.run(loss)

print(“After %s steps: global\_step is %f ,w is %f , learning\_rate is %f , loss is %f ” %(i,global\_step\_val,w\_val,learning\_rate\_val,loss\_val))

# 滑动平均：