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) = $\sum_{i=1}^{n}(y-y\_)$^2 /n

   ```
   loss = tf.reduce_mean(tf.square(y_ - y))
   ```
4. 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)
   ```

5. 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)
```

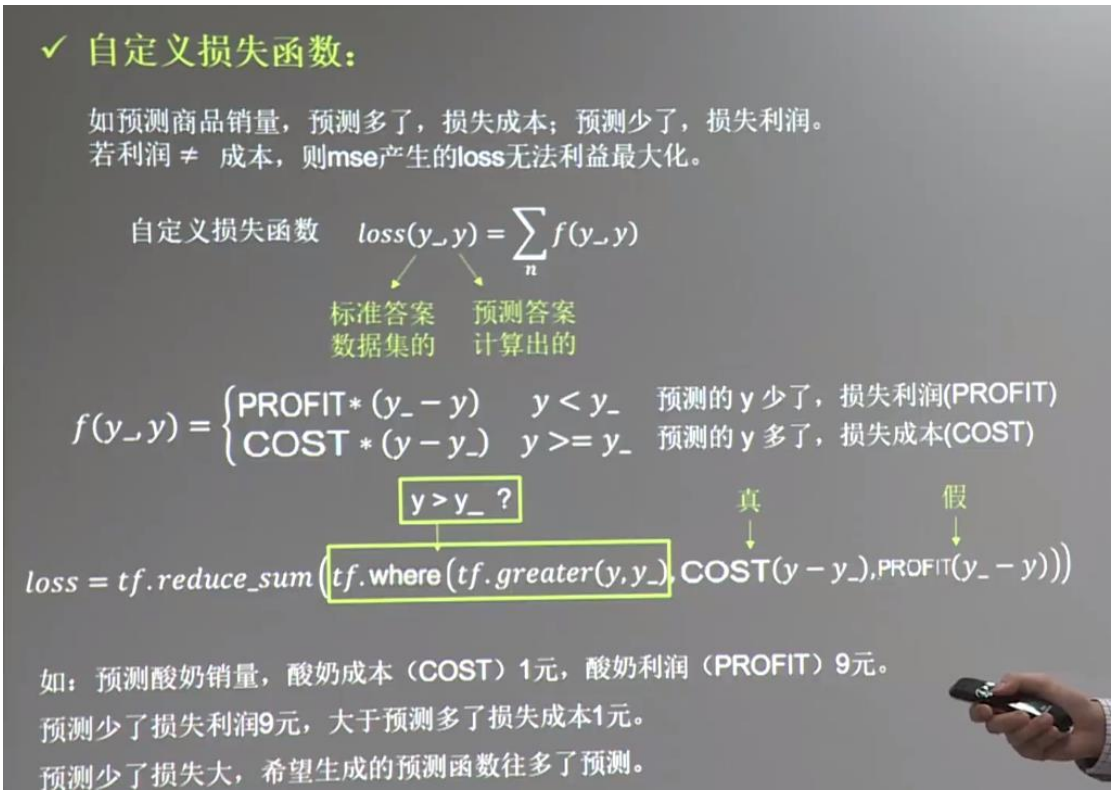
损失函数

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)
2. 均方误差 mse : MSE(y_,y) = $\sum_{i=1}^{n}(y-y\_)$^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) = -$\sum y\_ * logy$

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（$y_1,y_2,\ldots,y_n$）通过 softmax()函数便满足了概率分布要求：

$$\forall x \quad P(X=x) \in [0,1] \text{ 且 } \sum_x P(X=x) = 1$$

$$softmax(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}}$$

ce=tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, labels=tf.argmax(y_, 1))

cem=tf.reduce_mean(ce)

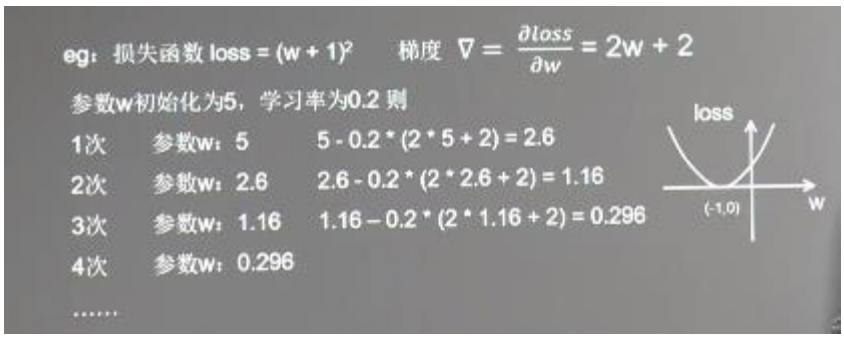
学习率：

learning_rate:每次参数更新的幅度

$w_{n+1} = w_n - learning\_rate\nabla$

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))
```

3. 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 * $\mathrm{LEARNING\_RATE\_DECAY}^{\frac{global\_step}{LEARNING\_RATE\_STPE}}$

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：

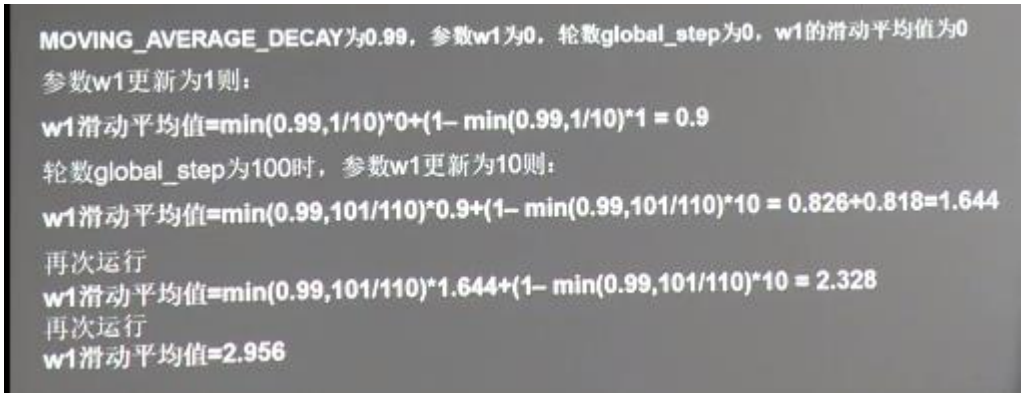
```
#设损失函数 loss=(w+1)^2，令 w 初值是常数 5，反向传播就是求最优 w,即求最小 loss 对应的 w 值
#使用指数衰减的学习率，在迭代初期得到较高的下降速度，可以在较小的训练轮数下取得更有收敛度
 import tensorflow as tf
LEARNING_RATE_BASE=0.1  最初学习率
LEARNING_RATE_DECAY=0.99 学习率衰减率
LEARNING_RATE_STEP=1 #喂入多少轮 BATCH_SIZE 后，更新一次学习率，一般设为：总样本数
```

```
/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：

1. 滑动平均（影子值）：记录了每个参数一段时间内过往值的平均，增加了模型的泛化性。
2. 针对所有参数：w 和 b
3. 像是给参数加了影子，参数变化，影子缓慢追随
4. 影子=衰减率 * 影子+（1 – 衰减率）*参数　影子初值=参数初值
5. 衰减率=min{ MOVING_AVERAGE_DECAY，$\frac{1+轮数}{10+轮数}$}

例子如图：



6. tensorflow 中这样使用：
   a. ema=tf.train.ExponentialMovingAverage(
      衰减率 MOVING_AVERAGE_DECAY,
      当前轮数 global_step)
   b. ema_op = ema.apply([])
      ema_op=ema.apply(tf.trainable_variables())
      每运行此句，所有待优化的参数求滑动平均
   c．with tf.control_dependencies([train_step, ema_op]):
      train_op = tf.no_op(name='train')
   d．ema.average(参数名)
      查看某参数的滑动平均值

例子：

```
import tensorflow as tf
#1 定义变量及滑动平均类
#定义一个 32 位浮点变量，初始值为 0.0 这个代码就是不断更新 w1 参数，优化 w1 参数，滑动平均做了个 w1 的影子
w1=tf.Variable(0, dtype=tf.float32)
#定义 num_updates(NN 迭代轮数)，初始值为 0，不可被优化（训练），这个参数不训练
global_step=tf.Variable(0, trainable =False)
#实例化滑动平均类，给删减率为 0.99，当前轮数 global_step
MOVING_AVERAGE_DECAY =0.99
ema=tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY, global_step)
#ema.apply 后的括号里是更新列表，每次运行 sess.run(ema_op)时，对更新列表中的元素求滑动平均值。
#在实际应用中会使用 tf.trainable_variables()自动将所有待训练的参数汇总为列表
#ema_op =ema.apply([w1])
ema_op=ema.apply(tf.trainable_variables())
#2 查看不同迭代中变量取值的变化.
with tf.Session() as sess:
    #初始化
```

```
    init_op=tf.global_variables_initializer()
    sess.run(init_op)
    #用 ema.average(w1)获取滑动平均值（要运行多个节点，作为列表中的元素列出，写在 sess.run 中）
#打印出当前参数 w1 和 w1 滑动平均值
print(sess.run([w1, ema.average(w1)]))
#参数 w1 的值赋为 1
sess.run(tf.assign(w1,1))
sess.run(ema_op)
print(sess.run([w1, ema.average(w1)]))
#更新 step 和 w1 的值，模拟出 100 轮迭代后，参数 w1 变为 10
sess.run(tf.assign(global_step,100))
sess.run(tf.assign(w1,10))
sess.run(ema_op)
print(sess.run([w1,ema.average(w1)]))
#每次 sess.run 会更新一次 w1 的滑动平均值
sess.run(ema_op)
print(sess.run([w1,ema.average(w1)]))

sess.run(ema_op)
print(sess.run([w1,ema.average(w1)]))

sess.run(ema_op)
print(sess.run([w1,ema.average(w1)]))

sess.run(ema_op)
print(sess.run([w1,ema.average(w1)]))

sess.run(ema_op)
print(sess.run([w1,ema.average(w1)]))

sess.run(ema_op)
print(sess.run([w1,ema.average(w1)]))
```