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)

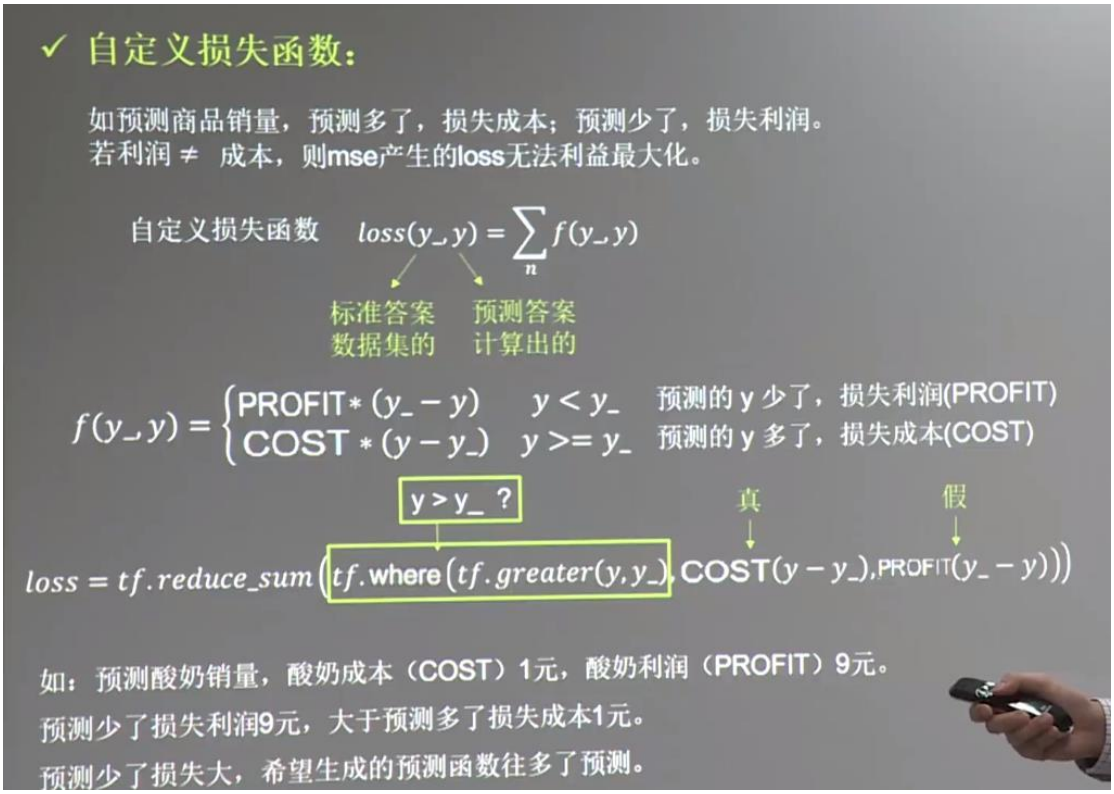
损失函数

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)

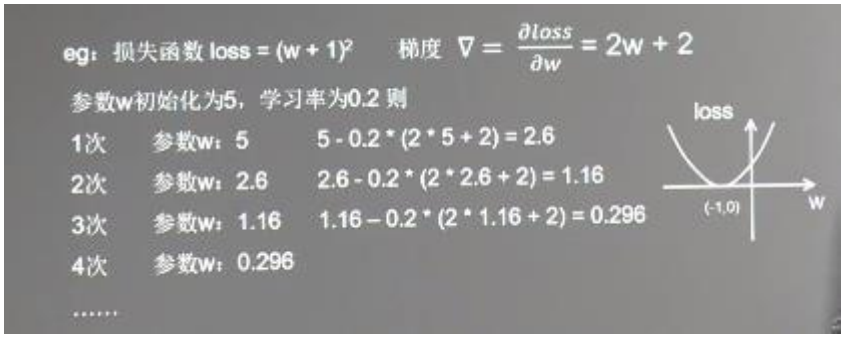
学习率：

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)
    for I in range(40):
        sess.run(train_step)
        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：

$learning\_rate = LEARNING\_RATE\_BASE * LEARNING\_RATE\_DECAY^{\frac{global\_step}{LEARNING\_RATE\_STPE}}$

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，所以为 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：

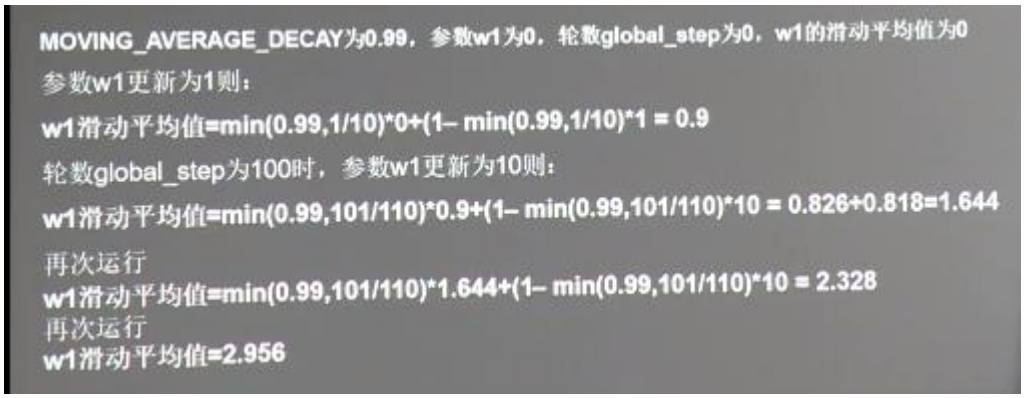
```
#设损失函数 loss=(w+1)^2, 令 w 初值是常数 5，反向传播就是求最优 w,即求最小 loss 对应的 w 值
#使用指数衰减的学习率，在迭代初期得到较高的下降速度，可以在较小的训练轮数下取得更有收敛度
 import tensorflow as tf
LEARNING_RATE_BASE=0.1 最初学习率
LEARNING_RATE_DECAY=0.99 学习率衰减率
LEARNING_RATE_STEP=1 #喂入多少轮 BATCH_SIZE 后，更新一次学习率，一般设为：总样本数/BATCH_SIZE
#运行了几轮 BATCH_SIZE 的计数器，初值给 0，设为不被训练
global_step=tf.Variable(0, trainable=False)
#定义指数下降学习率
learning_rate=tf.train.exponential_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)]))
```

正则化：

正则化缓解过拟合
正则化在损失函数中引入模型复杂度指标，利用给 w 加权值，弱化了训练数据的噪声（一般不正则化 b）
loss=loss(y 与 y_) +REGULARIZER * loss(w)

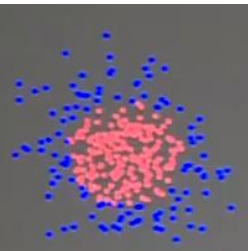
注：
loss：模型中所有参数的损失函数 如 交叉熵，均方误差
用超参数 REGULARIZEER 给出参数 w 在总 loss 中的比例，即正则化的权重
w：需要正则化的参数
loss(w) =tf.contrib.layers.l1_regularizer(REGULARIZER)(w)

$$loss_{L1}(w) = \sum_i |w_i|$$

loss(w) = tf.contrib.layers.l2_regularizer(REGULARIZER)(w)

$$loss_{L2}(w) = \sum_i |w_i^2|$$

tf.add_to_collection('losses', tf.contrib.layers.l2_regularizer(regularizer)(w))
把内容加到集合对应位置做加法（加到 losses 里面）
loss=cem +tf.add_n(tf.get_collection('losses'))



数据X $[x_0, x_1]$为正态分布随机点

标注Y_当$x_0^2 + x_1^2 < 2$ 时 y_=1（红），其余y_=0（蓝）

```
import matplotlib.pyplot as plt
```

sudo pip install 待安装模块名

✓
```
plt.scatter (x坐标, y坐标, c="颜色")
plt.show()
```

✓
```
xx, yy = np.mgrid[起:止:步长, 起:止:步长]
```

```
grid = np.c_[xx.ravel(), yy.ravel()]
```

组成矩阵　　拉直

```
probs = sess.run(y, feed_dict={x:grid})
probs = probs.reshape(xx.shape)
```

✓
```
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()
```

神经网络搭建八股：

搭建模块化的神经网络八股：

A. 前向传播就是搭建网络，设计网络结构（forward.py）

设计三个函数：

```
def forward(x, regularizer):
    w=
    b=
    y=
    return y
```

说明：给出输入到输出的数据通路

输入 x,正则化权重 regularizer

forward 函数定义了前向传播过程:w, b, y

```
def get_weight(shape, regularizer):
    w=tf.Variable( ) // 赋初值
    #把 w 的每个损失加入到总损失 losses 中
    tf.add_to_collection('losses', tf.contrib.layers.l2_regularizer(regularizer)(w))
    return w
```

说明：两个参数 w 的形状 shape；正则化权重 regularizer

```
def get_bias(shape):
    b=tf.Variable() //赋初值
    return b
```

说明:参数 shape 为 b 的 shape，某层中 b 的个数

B. 反向传播就是训练网络，优化网络参数（backward.py）

```
def backward():
    #给输入 x，y_占位
      x=tf.placeholder(          )
      y_ = tf.placeholder(       )
#利用 forward 模块复现前向传播的网络结构
      y=forward.forward(x, REGULARIZER)
#轮数计数器
      global_step = tf.Variable(0, trainable=False)
      loss =
```

正则化：

loss 可以是：

y 与 y_ 的差距（loss_mse） = tf.reduce_mean(tf.square(y – y_))

也可以是：

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)

加入正则化后：

loss = y 与 y_ 的差距 +tf.add_n(tf.get_collection("losses"))

指数衰减学习率：

```
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)
```

滑动平均：

```
  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')
  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：
                    print
#判断 python 运行的文件是否为主文件
#如果是主文件，则执行 backward 函数
 if __name__ == '__main__':
       backward()
```

例子中：生成数据集： generateds.py 前向传播:forward.py 反向传播;backward.py

MNIST 数据集

- ✓ MNIST 数据集：

  提供 6w 张 28*28 像素点的 0-9 手写数字图片和标签，用于训练

  提供 1w 张 28*28 像素点的 0-9 手写数字图片和标签，用于测试

  每张图片的 784 个像素点（28*28=784）组成长度为 784 的一维数组，作为输入特征。

  图片的标签以一维数组形式给出，每个元素表示对应分类出现的概率

  eg:[0,0,0,0,0,1,0,0,0]　　6

- ✓ from tensorflow.examples.tutorials.mnist import input_data　#自动加载
  mnist=input_data.read_data_sets('./data/',one_hot=True)#路径，读热码的形式
- ✓ 返回各子集样本数

  print("train data size:",mnist.train.num_examples)

  print("validation data size:",mnist.train.num_examples)

  print("test data size:",mnist.test.num_examples)
- ✓ 返回标签和数据

  mnist.train.labels[0]　**#查看指定编号的标签/图片**

  mnist.train.images[0]　　**#第 0 个像素的 784 个像素点**
- ✓ 取一小撮数据，准备喂入神经网络训练

  BATCH_SIZE=200　#定义一小撮是多少

  xs，ys = mnist.train.net_batch(BATCH_SIZE)

  print("xs shape:",xs.shape)

  print("ys shape:",ys.shape)
- ✓ 认识几个函数：

  tf.get_collection(" ") 从集合中取全部变量，生成一个列表

  tf.add_n([ ])　列表内对应元素相加

  tf.cast(x, dtype) 把 x 转为 dtype 类型

  tf.argmax(x, axis) 返回最大值所在索引号，x 为列表;如：tf.argmax([1,0,0] ， 1) 返回 0

  os.path.join("home","name")　返回 home/name

  字符串.spliit()　　按指定拆分符对字符串切片，返回分割后的列表

  with tf.Graph().as_default() as g:　　其内定义的节点在计算图 g 中
- ✓ 保存模型

  saver = tf.train.Saver()　实例化 saver 对象

  with tf.Session() as sess:　在 with 结构 for 循环中一定轮数时，保存模型当前会话

  ```
  for i in range(STEPS):
      if i%轮数==0：　拼接./ MODEL_SAVE_PATH/ MODEL_NAME- global_step
          saver.save(sess,
          os.path.join(MODEL_SAVE_PATH,MODEL_NAME),global_step=global_step)
  ```

- ✓ 加载模型

```
with tf.Session() as sess:
    ckpt=tf.train.get_checkpoint_state(存储路径)
    if ckpt and ckpt.model_checkpoint_path:    #模型参数加载到当前会话中
        saver.restore(sess, ckpt.model_checkpoint_path)
```

- ✓ 实例化可还原滑动平均值的 saver

```
ema=tf.train.ExponentialMovingAverage(滑动平均值)
ema_restore=ema.variables_to_restore()
saver=tf.train.Saver(ema_restore)
```

- ✓ 准确率计算方法

```
correct_prediction =tf.equal(tf.argmax(y,1),tf.argmax(y_, 1))
accuracy=tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
```

- ✓ 回顾模块化搭建神经网络八股

```
------前向传播-----
def forward(x, regularizer):
    w=
    b=
    y=
    return y
def get_weight(shape, regularizer):
def get_bias(shape):
----反向传播--------
def backward(mnist):
    x=
    y_ =
    y =
    global_step=
    loss=
    <正则化，指数衰减学习率，滑动平均>
    train_step=
    实例化 saver
    with tf.Session() as sess:
        初始化
        for i in age(STEPS):
            sess.run(train_step,feed_dict={x: , y_: })
            if i%轮数==0：
                print
                saver.save(    )
```

- ✓ 损失函数 loss 含正则化 regularization

backward.py 中加入

```
ce=tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y,labels=tf.argmax(y_,1))
cem=tf.reduce_mean(ce)
loss=cem+tf.add_n(tf.get_collection('losses'))
```

forward.py 中加入

```
if regularizer !=None: tf.add_to_collection('losses',tf.contrib.layers.l2_regularizer(regularizer)(w))
```

✓ 学习率 learning_rate

backward.py 中加入

```
learning_rate=tf.train.exponential_decay(
  LEARNING_RATE_BASE,
  global_step,
  LEARNING_RATE_STEP,
  LEARNING_RATE_DECAY,
  staircase = True)
```

✓ 滑动平均 ema

backward.py 中加入

```
ema=tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY,global_step)
ema_op=ema.apply(tf.trainable_variaables())
with tf.control_dependencies([train_step,ema_op]):
    train_op=tf.no_op(name='train')
```

test.py

```
def test(mnist):
    with tf.Graph().as_default() as g:
        定义 x y_ y
        实例化可还原滑动平均值的 saver
        计算正确率
        while True:
            with tf.Session() as sess:
                加载 ckpt 模型
                ckpt=tf.train.get_checkpoint_state(存储路径)
                如果已有 ckpt 模型则恢复
```

```
                if ckpt and ckpt.model_checkpoint_path:
                    恢复会话
                    saver.restore(sess,ckpt.model_checkpoint_path)
                    恢复轮数
                    global_step=ckpt.model_checkpoint_path.split('/')[-
1].split('-')[-1]
                    计算准确率
                    accuracy_score=sess.run(accuracy,feed_dict={
                       x:mnist.test.images,y_:mnist.test.labels})
                    打印提示
                    print("after %s training steps, test
accuracy=%g" %(global_step,accuracy_score))
                    如果没有模型 else:
                                    给出提示
                                    print("No checkpoint file found")
                                    return
def main():
    mnist=input_data.read_data_sets("./data/",one_hot=True)
    test(mnist)
if __name__=='__main__':
    main()
```

手写数字识别准确率输出

输入手写数字图片输出识别结果

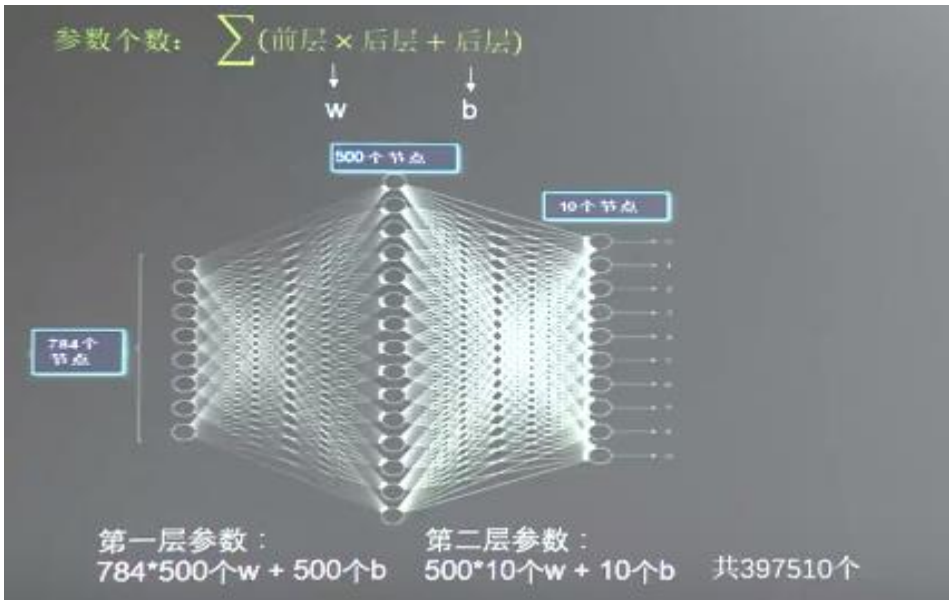
- 如何对输入的真实图片，输出预测结果？
  越接近 0 越黑，越接近 1 越白
  输入图片输出预测值：
  前向传播：mnist_forward.py
  反向传播：mnist_backward.py
  测试程序：mnist_test.py
  应用程序：mnist_app.py
- 如何制作数据集，实现特定应用？
  - tfrecords 文件

  tfrecords 是一种二进制文件，可先将图片和标签制作成该格式的文件。
  使用 tfrecords 进行数据读取，会提高内存利用率。
  用 tf.train.Example 的协议存储训练数据。训练数据的特征用键值对的形式表示：
  如 ‘img_raw’ :值 ‘label’ :值 值是 Byteslis/FloatList/Int64List
  用 SerializeToString()把数据序列化成字符串存储。
  - 生成 tfrecords 文件

    writer = tf.python_io.TFRecordWriter(tfRecordName) 新建一个 writer
    for 循环遍历每张图和标签：
    example=tf.train.Example(features=i)

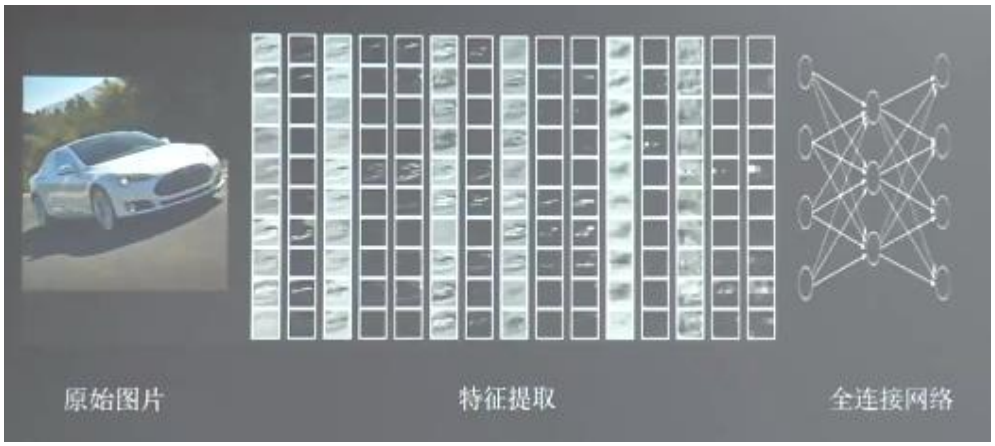
卷积神经网络

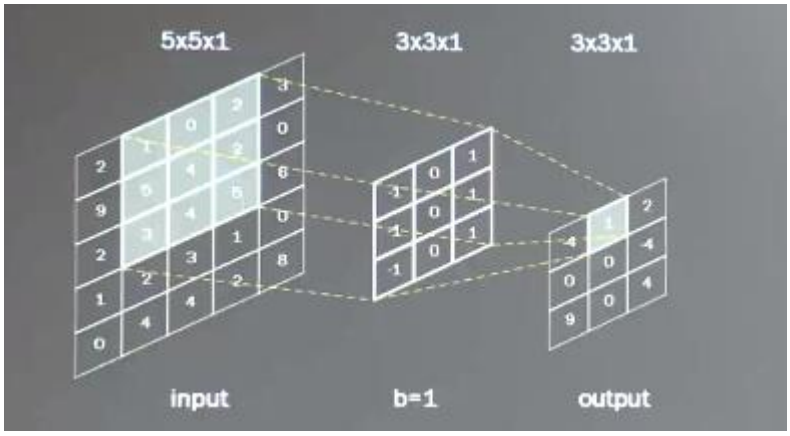
- ✓ 全连接 NN：每个神经元与前后相邻层的每一个神经元都有连接关系，输入是特征，输出为预测的结果。



待优化的参数过多容易导致模型过拟合

实际应用中会先对原始图像进行特征提取，再把提取到的特征喂给全连接网络



- ✓ **卷积 Convolutional**
- 卷积可认为是一种有效提取图像特征的方法。
- 一般会用一个正方形卷积核，遍历图片上的每个点。图片区域内，相对应的每一个像素值，乘以卷积核内相对应点的权重，求和，再加上偏置。
- 

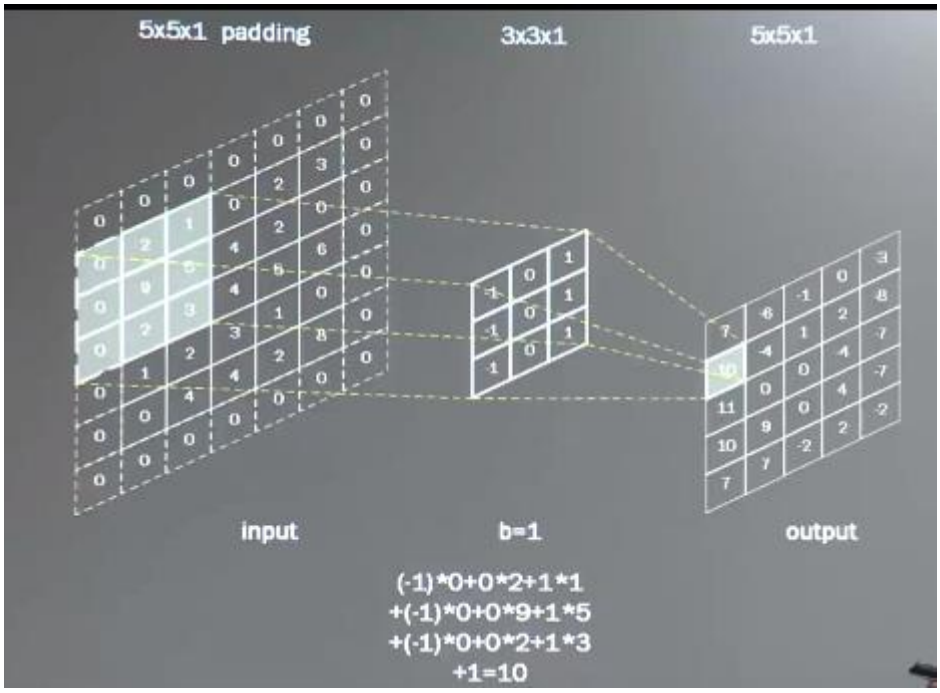
1*(-1)+0*0+2*1+5*(-1)+4*0+2*1+3*(-1)+4*0+5*1+1=1

说明： **5*5*1；1 代表单通道，5*5 代表分辨率**

输出图片边长=（输入图片边长-卷积核长+1）/步长

此图：（5-3+1）/1=3

✓ Padding

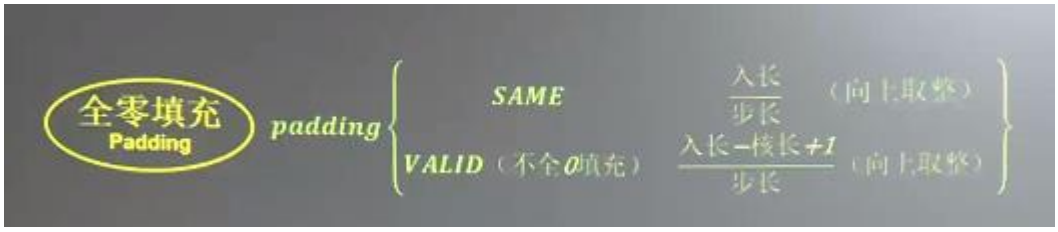
- 

说明：在输入图片周围进行全 0 填充，保证输出图片和输入图片尺寸一致；

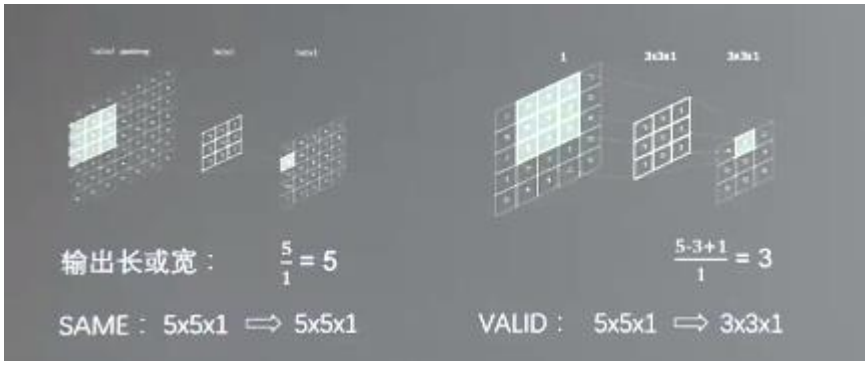
---这个过程叫 padding

输出图片边长=输入图片边长/步长；此图：5/1=5

✓ 全零填充 padding



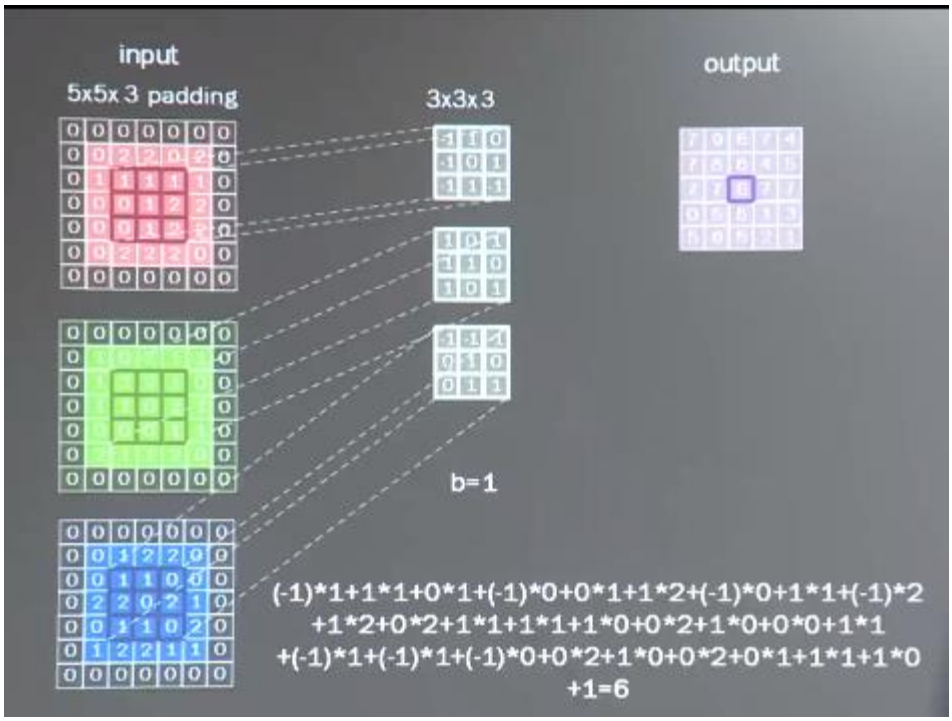
- 在 Tensorflow 框架中，
  用参数 padding='SAME' 或 padding='VALID'表示
  如图例子：



✓ **Tensorflow 计算卷积：**

- tf.nn.conv2d(输入描述，eg.[batch,5,5,1]----batch 一次喂入多少张图片；5,5 分辨率；1:通道数（灰度图为 1，彩色图，如红绿蓝，为 3）
  卷积核描述，eg.[3,3,1,16]----3,3 行列分别率；1 通道数；16 核个数（卷积操作后输出图片的深度为 16，即输出是 16 通道）
  核滑动步长，eg.[1,**1**,**1**,1]---**1** 行步长，**1** 列步长(前后两个 1 固定)
  padding='VALID')
- **卷积核的通道数=输入图片的通道数**

- 多通道：（红绿蓝 3 通道如图）

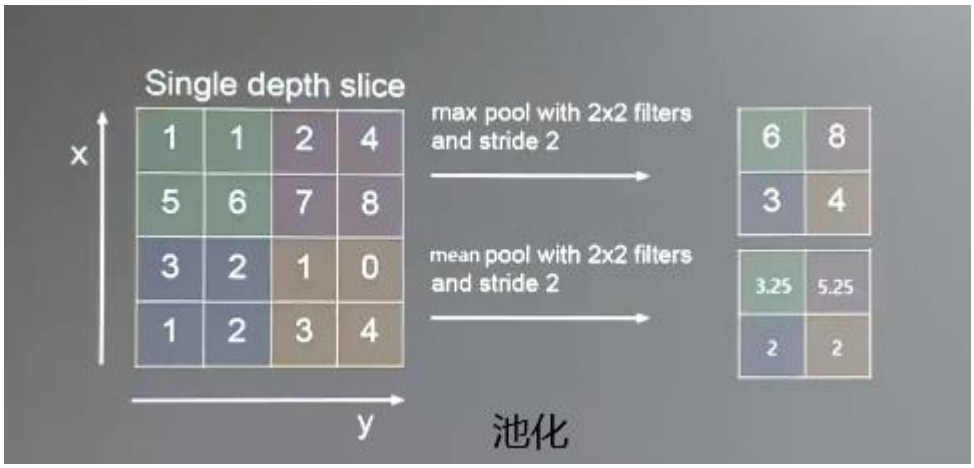


上图在 tensorflow 中：

- tf.nn.conv2d(输入描述，eg.[batch,5,5,3]
  卷积核描述，eg.[3,3,3,16]
  核滑动步长，eg.[1,**1**,**1**,1]
  padding='SAME)

✓ 池化 Pooling

- **池化用于减少特征数量**
- **最大值池化可提取图片纹理，均值池化可保留背景特征**

- 

- **Tensorflow 计算池化：**

  pool=tf.nn.avg_pool

  pool=tf.nn.max_pool(输入描述，eg.[batch,28,28,6]---28,,28 行列分辨率;6 通道数

  池化核描述(仅大小)，eg.[1,2,2,1]---2,2 行列分辨率(前后两个 1 固定)

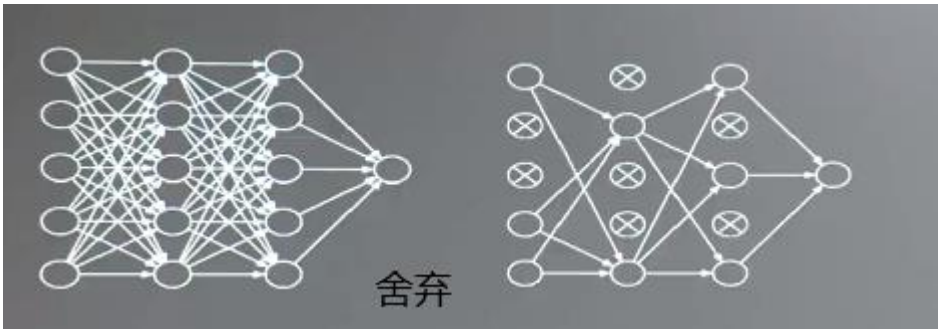
  池化核滑动步长，eg.[1,2,2,1]---2 行步长，2 列步长(前后两个 1 固定)

  padding='SAME')

✓ 舍弃 Dropout（训练时舍弃）

- 在神经网络的训练过程中，将一部分神经元按照一定概率从神经网络中暂时舍弃。使用时被舍弃的神经元恢复链接。

  如图：



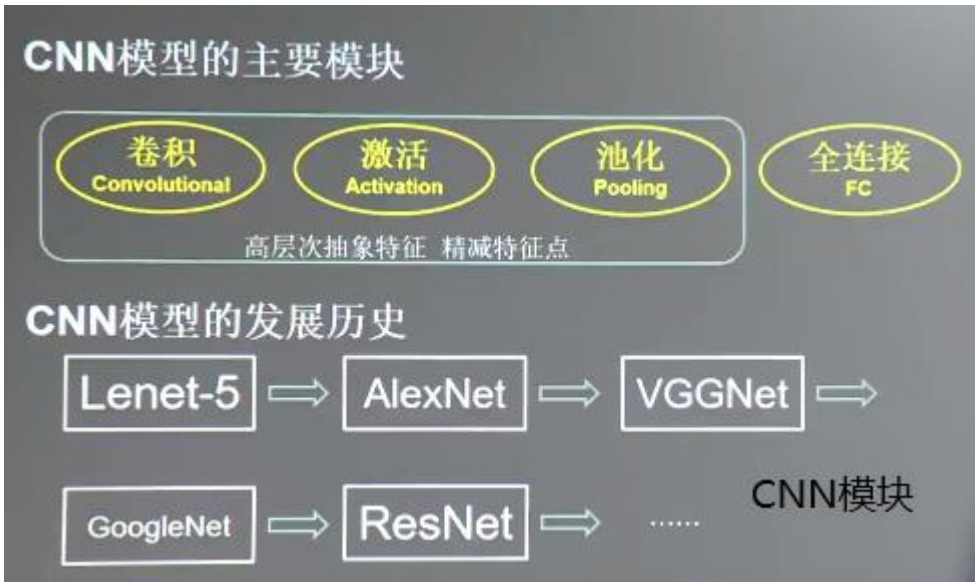
  tf.nn.dropout(上层输出，暂时舍弃的概率)

  if train: 输出=tf.nn.dropout(上层输出，暂时舍弃的概率)

  注：是 0 的神经元不参与当前轮的参数优化

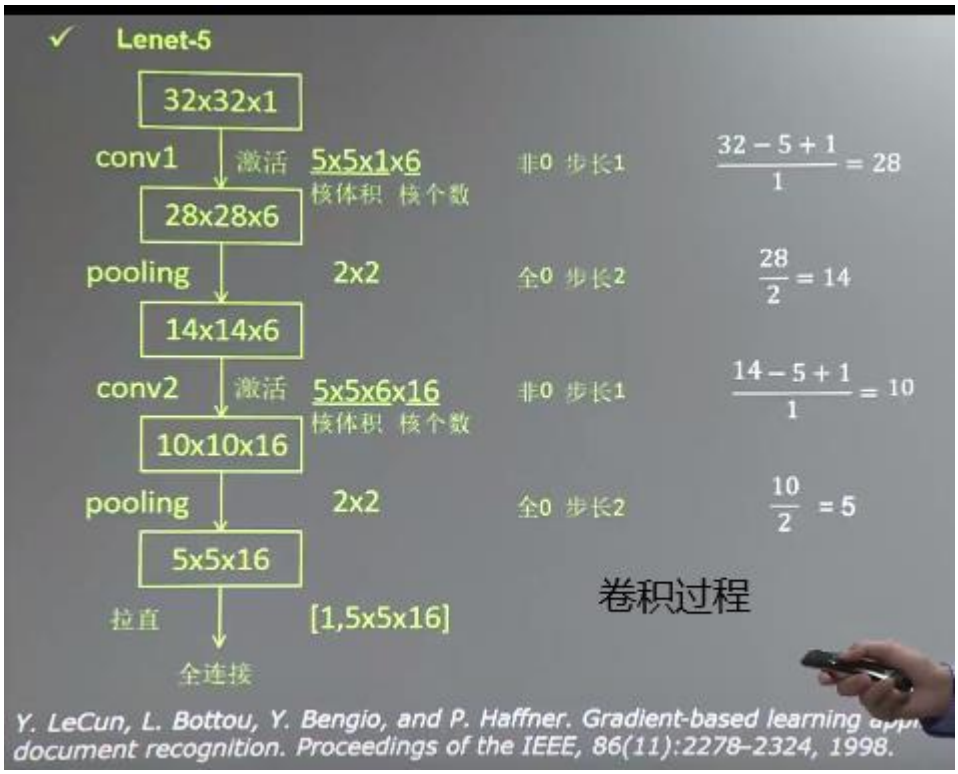
✓ 卷积 NN：借助卷积核(kernel)提取特征后，送入全连接网络

✓ CNN 模型的主要模块及发展历史：如图


CNN模型的主要模块
卷积
Convolutional
激活
Activation
池化
Pooling
全连接
FC
高层次抽象特征 精减特征点
CNN模型的发展历史
Lenet-5
AlexNet
VGGNet
GoogleNet
ResNet
.....
CNN模块

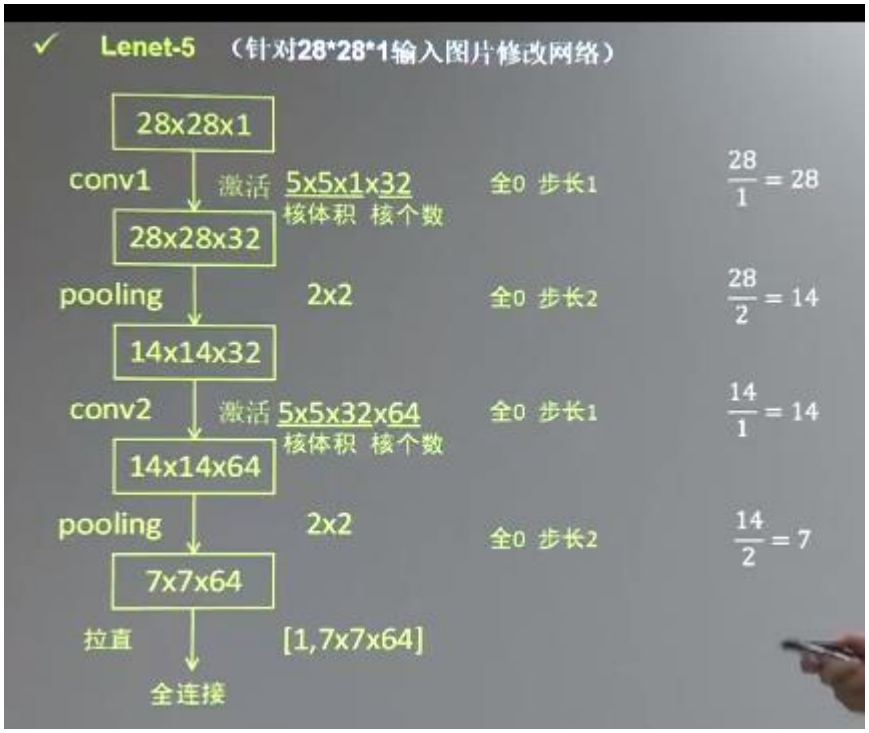
lenet5 代码讲解：

卷积过程：如图 1



说明：[1,5*5*16] 转化为一维数组送入全连接网络

图 2：



复现已有的卷积神经网络

- ✓ x=tf.placeholder(tf.float32,shape=[BATCH_SIZE,IMAGE_PIXELS])
  [1,244,244,3]
  1: 表一次喂入一张图片，244，244，3 是分辨率和通道数
  - tf.placeholder 用于传入真实训练样本/测试/真实特征/待处理特征，仅占位，
  - 不必给初值，用 sess.run 的 feed_dict 参数以字典形式喂入 x:
  - sess.run(求分类评估值的节点，feed_dict{x: })
- ✓ np.load/save 将数组以二进制格式读出/写入磁盘，扩展名为.npy
- np.save("名.npy",某数组)
- 某变量=np.load("名.npy",encoding="").item()
  -----encoding 可不写，'latin1' , 'ASCII', 'bytes' 默认为'ASCII'
- ✓ .item() 遍历(键值对)
  如：data_dict=np.load(vgg16.npy,encoding='latin1').item()
  --读 vgg16.npy 文件，遍历其内键值对，导出模型参数赋给 data_dict.
- ✓ tf.shape(a) 返回 a 的维度
  ----a 可为 tensor, list, array
  eg. x=tf.constant([[1,2,3],[4,5,6]]) tensor
  y=[[1,2,3],[4,5,6]] list
  z=np.arange(24).reshape([2,3,4]) array
  sess.run(tf.shape(x)) [2 3]
  sess.run(tf.shape(y)) [2 3]
  sess.run(tf.shape(z)) [2 3 4]
- ✓ tf.nn.bias_add(乘加和, bias) 把 bias 加到乘加和上
- ✓ tf.reshape(tensor , $\begin{matrix}[n行,m列]\\[-1,m列]\end{matrix}$) → -1 表示行跟随 m 列自动计算
- ✓ np.argsort(列表) 对列表从小到大排序，返回索引值
- ✓ os.getcwd() 返回当前工作目录
- ✓ os.path.join(, ,…) 拼出整个路径，可引导到特定文件
- ✓ eg. vgg16_path=os.path.join(os.getcwd(), "vgg16.npy")
  ----当前目录/vgg16.npy 索引到 vgg16.npy 文件