二次代价函数（quadratic cost）

$C = \frac{1}{2n}\sum_x ||y(x) - a^L(x)||^2$

- 其中，C表示代价函数，x表示样本，y表示实际值（label），a表示输出值（预测值），n表示样本的总数。
- 例：当样本为1时，即x,n=1：
  $C = \frac{(y-a)^2}{2}$
  其中$a = \sigma(z), z = \sum W_j * X_j + b$；其中$\sigma()$是激活函数；　。
- 使用梯度下降法（Gradient descent）来调整权值参数的大小，权值W和偏置b的梯度推导如下：
  $\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x, \frac{\partial C}{\partial b} = (a-y)\sigma'(z)$
  其中，z表示神经元的输入，σ表示激活函数。w和b的梯度跟激活函数的梯度成正比，激活函数的梯度越大，w和b的大小调制得越快。

交叉熵代价函数（cross-entropy）

不改变激活函数，而改变代价函数，改用交叉熵代价函数：
$C = -\frac{1}{n}\sum_x [ylna + (1-y)ln(1-a)]$

- 其中，C表示代价函数，x表示样本，y表示实际值，a表示输出值，n表示样本的总数。
  其中$a = \sigma(z), z = \sum W_j * X_j + b, \sigma'(z) = \sigma(z)(1-\sigma(z))$
  $\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) - y), \frac{\partial C}{\partial b} = \frac{1}{n}\sum_x(\sigma(z) - y)$
- 权值和偏置值的调整与$\sigma'(z)$（激活函数的导数）无关，另外，梯度公式中的$\sigma(z) - y$表示输出值与实际值的误差。所以当误差越大时，梯度就越大，参数w和b的调整就越快，训练的速度也就越快。
- 如果输出神经元是线性的，那么二次代价函数就是一种合适的选择。如果输出神经元是S型函数，那么比较适用交叉熵代价函数。

对数似然代价函数（log-likelihood cost）

- 对数似然函数常用来作为$softmax$回归的代价函数，如果输出层神经元是$sigmoid$函数，可以采用交叉熵代价函数。而深度学习中更普遍的做法是将$softmax$作为最后一层，此时常用的代价函数是对数似然函数。
- 对数似然代价函数与$softmax$的组合和交叉熵与$sigmoid$函数的组合非常相似。对数似然代价函数在二分类时可以简化为交叉熵代价函数的形式。
- 在$temsorflow$中用：
  - tf.nn.sigmoid_cross_entropy_with_logits()来表示跟$sigmoid$搭配使用的交叉熵。
  - tf.nn.softmax_cross_entropy_with_logits()来表示跟$softmax$搭配使用的交叉熵。

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data   #手写数字相关的数据包
```

```
# 载入数据集
```

```
mnist = input_data.read_data_sets("MNIST_data",one_hot=True)    #载入数据，{数据集包路径，把标签转化为只有0和1的形式}

#定义变量，即每个批次的大小
batch_size = 100    #一次放100章图片进去
n_batch = mnist.train.num_examples // batch_size    #计算一共有多少个批次；训练集数量（整除）一个批次大小

#定义两个placeholder
x = tf.placeholder(tf.float32,[None,784])    #[行不确定，列为784]
y = tf.placeholder(tf.float32,[None,10])    #数字为0-9，则为10

#创建简单的神经网络
W = tf.Variable(tf.zeros([784,10]))    #权重
b = tf.Variable(tf.zeros([10]))     #偏置
prediction = tf.nn.softmax(tf.matmul(x,W)+b)    #预测

#定义二次代价函数
# loss = tf.reduce_mean(tf.square(y-prediction))
#定义交叉熵代价函数
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y,logits=prediction))
#使用梯度下降法
train_step = tf.train.GradientDescentOptimizer(0.2).minimize(loss)

#初始化变量
init = tf.global_variables_initializer()

#准确数，结果存放在一个布尔型列表中
correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(prediction,1))    #比较两个参数大小是否相同，同则返回为true，不同则返回为false；argmax()：返回张量中最大的值所在的位置

#求准确率
accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
#cast()：将布尔型转换为32位的浮点型；（比方说9个T和1个F，则为9个1，1个0，即准确率为90%）

with tf.Session() as sess:
    sess.run(init)
    for epoch in range(21):
        for batch in range(n_batch):
            batch_xs,batch_ys = mnist.train.next_batch(batch_size)
            sess.run(train_step,feed_dict={x:batch_xs,y:batch_ys})

        acc = sess.run(accuracy,feed_dict={x:mnist.test.images,y:mnist.test.labels})
        print("Iter" + str(epoch) + ",Testing Accuracy" + str(acc))

```

```
Extracting MNIST_data\train-images-idx3-ubyte.gz
Extracting MNIST_data\train-labels-idx1-ubyte.gz
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz
Iter0,Testing Accuracy0.8502
Iter1,Testing Accuracy0.8954
Iter2,Testing Accuracy0.9014
```

```
Iter3,Testing Accuracy0.9052
Iter4,Testing Accuracy0.9079
Iter5,Testing Accuracy0.91
Iter6,Testing Accuracy0.9115
Iter7,Testing Accuracy0.9132
Iter8,Testing Accuracy0.9152
Iter9,Testing Accuracy0.9159
Iter10,Testing Accuracy0.9167
Iter11,Testing Accuracy0.9181
Iter12,Testing Accuracy0.9189
Iter13,Testing Accuracy0.9192
Iter14,Testing Accuracy0.9205
Iter15,Testing Accuracy0.9202
Iter16,Testing Accuracy0.921
Iter17,Testing Accuracy0.9209
Iter18,Testing Accuracy0.9213
Iter19,Testing Accuracy0.9216
Iter20,Testing Accuracy0.922
```

```
```