模型构建

定义待输入数据的占位符

```
# mnist 中每张图片共有28*28=784个像素点
x = tf.placeholder(tf.float32, [None, 784], name="X")

# 0-9 一共10个数字=> 10 个类别
y = tf.placeholder(tf.float32, [None, 10], name="Y")
```

定义模型变量

在本案例中，以正态分布的随机数初始化权重W，以常数0初始化偏置b

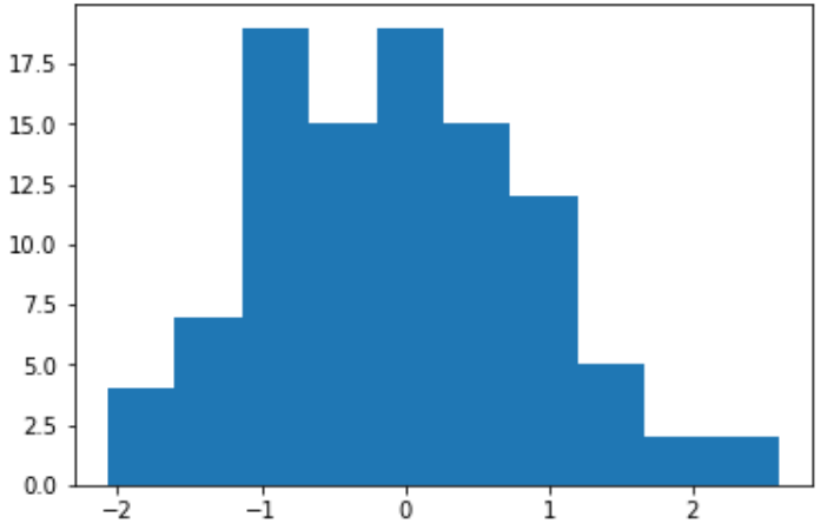
```
# 定义变量
W = tf.Variable(tf.random_normal([784, 10]),name="W")
b = tf.Variable(tf.zeros([10]),name="b")
```

了解一下 tf.random_normal()

```
norm = tf.random_normal([100]) #生成100个随机数
with tf.Session() as sess:
    norm_data=norm.eval()
print(norm_data[:10])          #打印前10个随机数
```

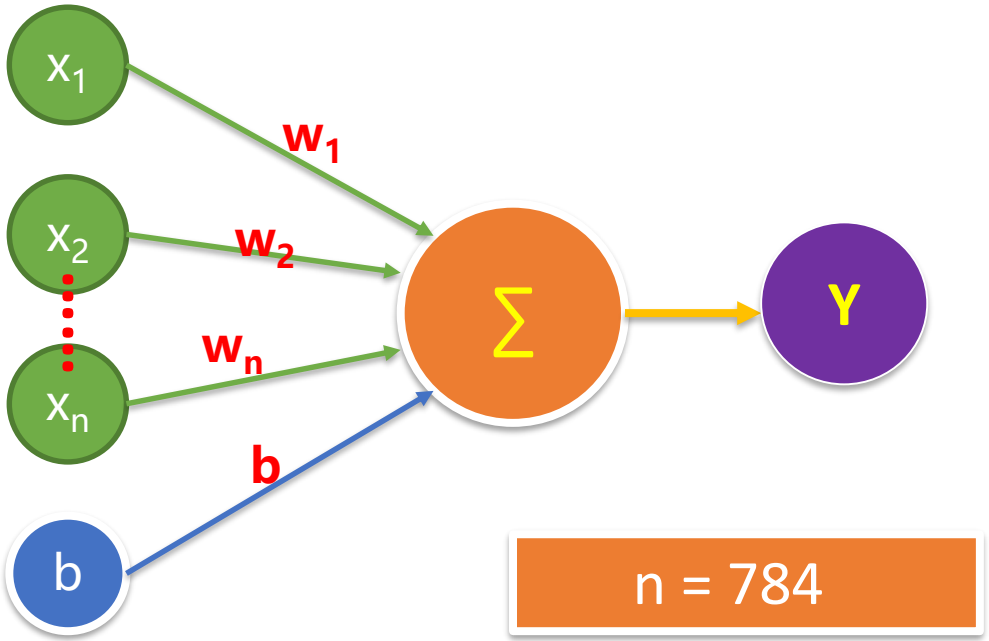
```
[-1.60550928  1.65648937 -0.05990671 -0.25936738 -1.23932779 -0.21585168
 -0.9228276  -2.07075715  0.27118909 -1.28056562]
```

```
import matplotlib.pyplot as plt
plt.hist(norm_data)
plt.show()
```



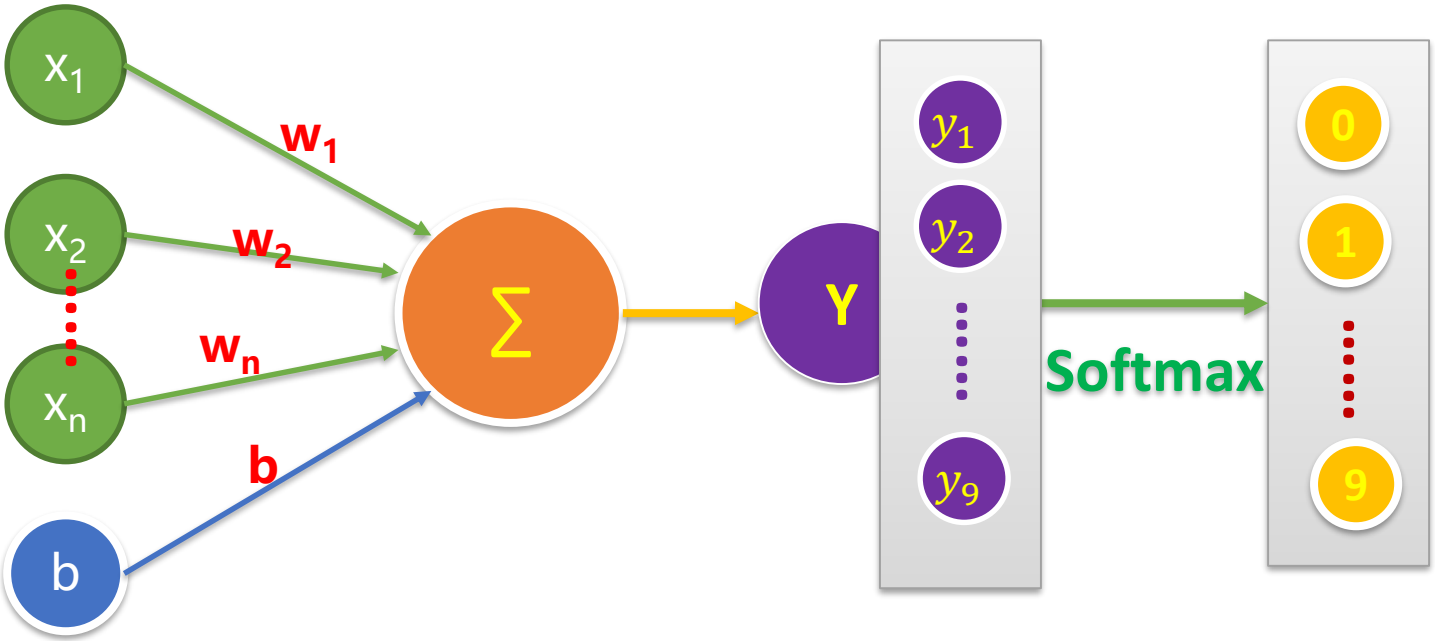
定义前向计算

```
forward=tf.matmul(x, W) + b # 前向计算
```



结果分类

```
pred = tf.nn.softmax(forward) # Softmax分类
```



从预测问题到分类问题

从线性回归到逻辑回归

逻辑回归

逻辑回归

许多问题的预测结果是一个在连续空间的数值，比如房价预测问题，可以用线性模型来描述：

$$Y = x_1 * w_1 + x_2 * w_2 + \cdots + x_n * w_n + b$$

但也有很多场景需要输出的是概率估算值，例如：

- 根据邮件内容判断是垃圾邮件的可能性
- 根据医学影像判断肿瘤是恶性的可能性
- 手写数字分别是 0、1、2、3、4、5、6、7、8、9的可能性（概率）

这时需要将预测输出值控制在 [0，1]区间内

二元分类问题的目标是正确预测两个可能的标签中的一个

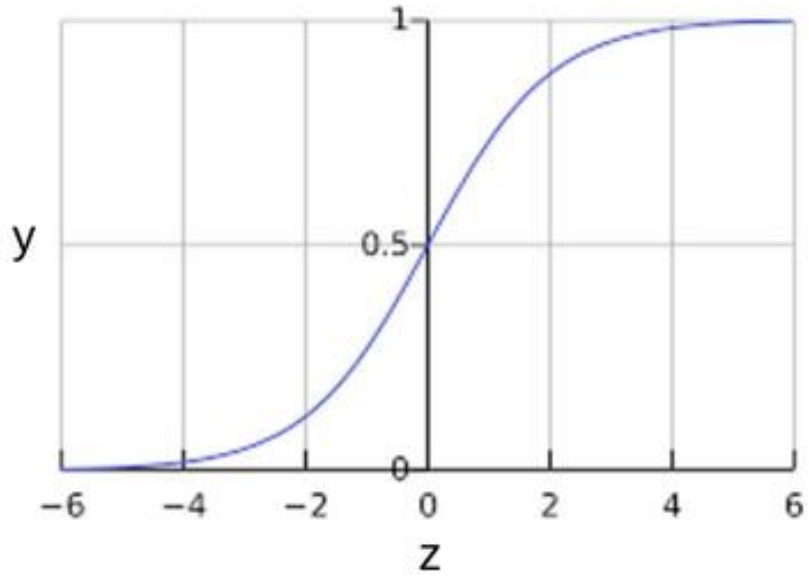
逻辑回归（Logistic Regression）可以用于处理这类问题

Sigmod函数

逻辑回归模型如何确保输出值始终落在 0 和 1 之间。

Sigmod函数（S型函数）生成的输出值正好具有这些特性，其定义如下：

$$y = \frac{1}{1 + e^{-(z)}}$$



定义域为全体实数，值域在[0, 1]之间

Z值在0点对应的结果为0. 5

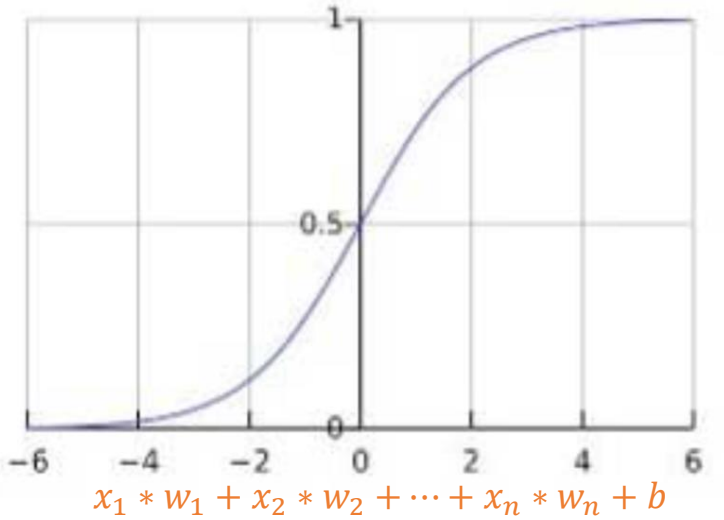
sigmoid函数连续可微分

特定样本的逻辑回归模型的输出

$$z = x_1 * w_1 + x_2 * w_2 + \cdots + x_n * w_n + b$$

$$y = \frac{1}{1 + e^{-(z)}}$$

概率输出



逻辑回归中的损失函数

前面**线性回归**的损失函数是**平方损失**，如果**逻辑回归**的损失函数也定义为**平方损失**，那么：

$$J(w) = \frac{1}{n}\sum_{i=1}^{n}(\varphi(z_i) - y_i)^2$$

其中：

i表示第i个样本点

$z_i = x_i * w + b$

$\varphi(z_i)$表示对i个样本的预测值

y_i表示第i个样本的标签值

逻辑回归中的损失函数

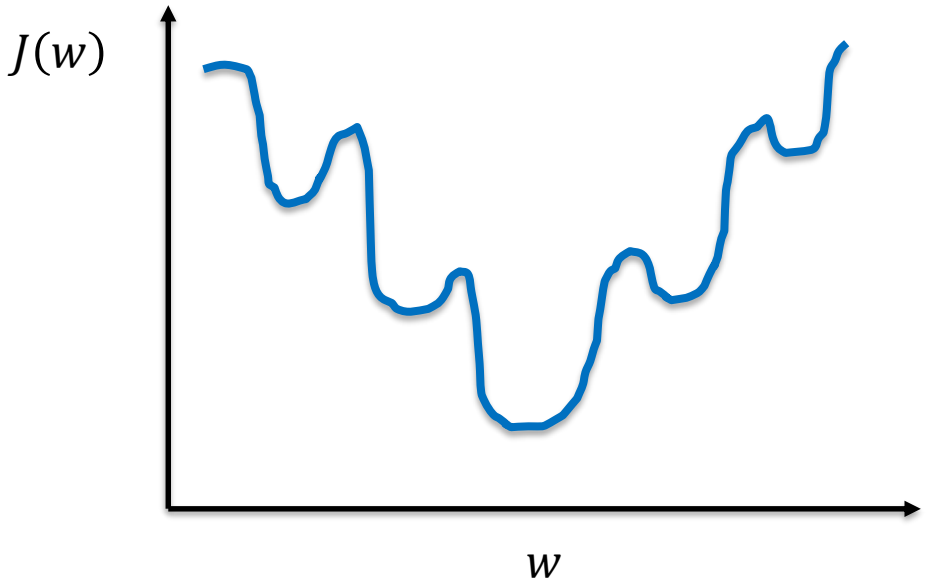
$$J(w) = \sum_{i=1}^{n} \frac{1}{n} (\varphi(z_i) - y_i)^2$$

$$\varphi = \frac{1}{1 + e^{-(z)}}$$

将Sigmoid函数带入上述函数

非凸函数，有多个极小值

如果采用梯度下降法，会容易导致陷入局部最优解中



逻辑回归中的损失函数

二元逻辑回归的损失函数一般采用**对数损失函数**，定义如下：

$$J(W,b) = \sum_{(x,y)\in D} -ylog(y') - (1-y)\log(1-y')$$

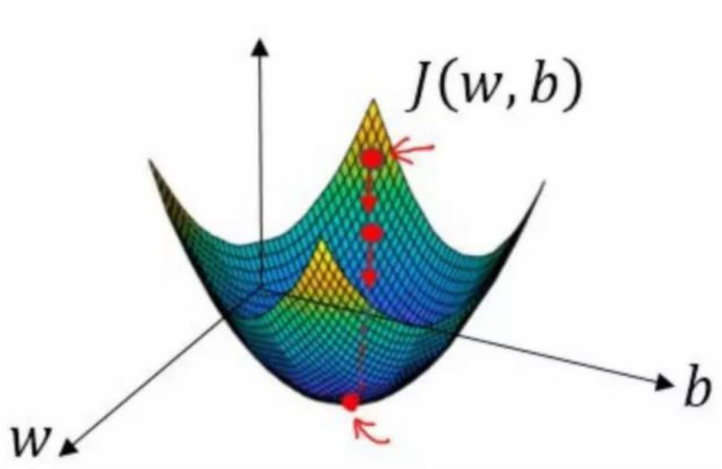
凸函数

其中：

$(x,y) \in D$是有标签样本 (x,y) 的数据集

y是有标签样本中的标签，取值必须是 0 或 1

y'是对于特征集x的预测值（介于 0 和 1 之间）



https://blog.csdn.net/t46414704152abc/article/details/79574003

多元分类

Softmax思想

逻辑回归可生成介于 0 和 1.0 之间的小数。

例如，某电子邮件分类器的逻辑回归输出值为 0.8，表明电子邮件是垃圾邮件的概率为 80%，不是垃圾邮件的概率为 20%。很明显，一封电子邮件是垃圾邮件或非垃圾邮件的概率之和为 1.0。

Softmax 将这一想法延伸到多类别领域。

在多类别问题中，Softmax 会为每个类别分配一个用小数表示的概率。这些用小数表示的概率相加之和必须是 1.0。

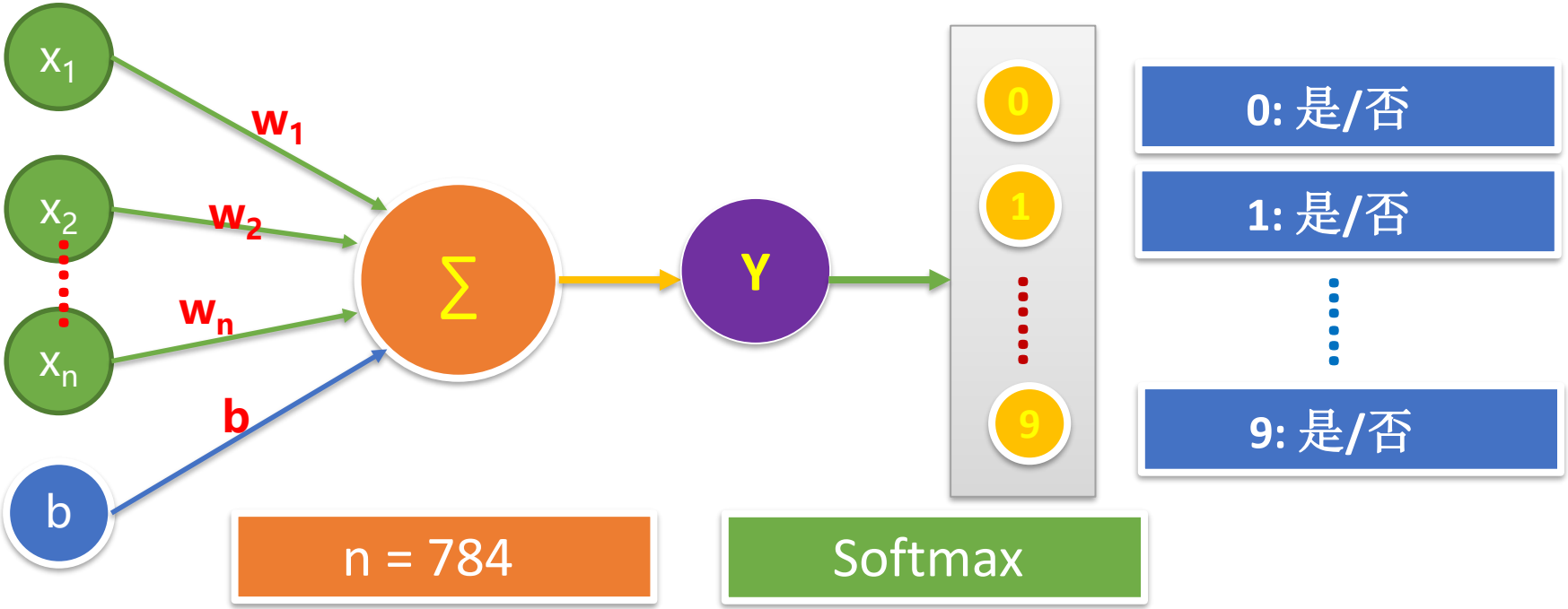
Softmax示例

类别	概率	类别	概率
0	0. 052	5	0. 035
1	0. 001	6	0. 006
2	0. 023	7	0. 002
3	0. 721	8	0. 104
4	0. 003	9	0. 053

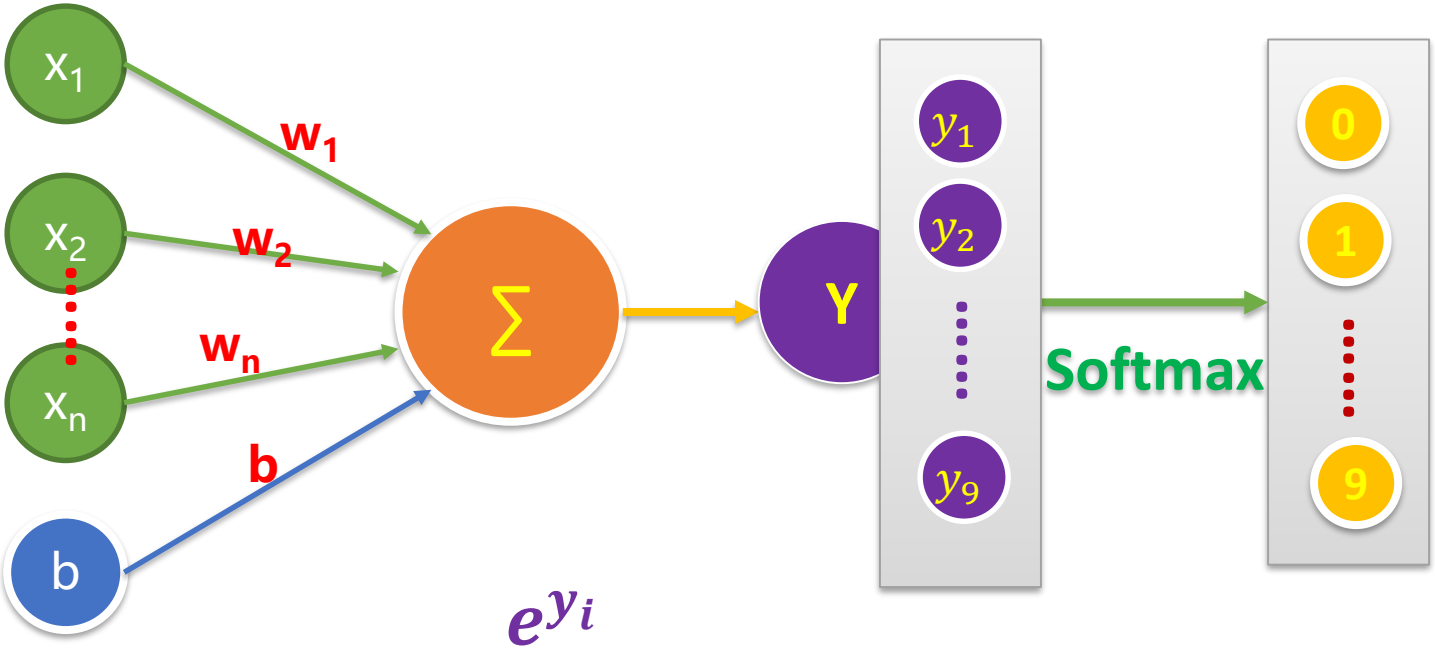
概率相加之和是 1. 0

最有可能的类别是 3

神经网络中的 Softmax 层



Softmax 方程式



$$p_i = \frac{e^{y_i}}{\sum_{k=1}^{C} e^{y_k}}$$

此公式本质上是将逻辑回归公式延伸到了多类别

Softmax 举例

$$p_i = \frac{e^{y_i}}{\sum_{k=1}^{C} e^{y_k}}$$

$$\mathrm{Y} = \begin{bmatrix} -3.1 \\ 1.8 \\ 9.7 \\ -2.5 \end{bmatrix} \qquad Softmax(Y) = \begin{bmatrix} 2.75972792e-6 \\ 3.70603254e-4 \\ 9.99621608e-1 \\ 5.02855213e-6 \end{bmatrix}$$

交叉熵损失函数

交叉熵是一个信息论中的概念，它原来是用来估算平均编码长度的。给定两个概率分布p和q，通过q来表示p的交叉熵为

$$H(p,q) = -\sum_{x} p(x)\log q(x)$$

交叉熵刻画的是**两个概率分布之间的距离**，p代表正确答案，q代表的是预测值，交叉熵越小，两个概率的分布约接近

交叉熵损失函数计算案例

假设有一个3分类问题，某个样例的正确答案是（1，0，0）

甲模型经过softmax回归之后的预测答案是（0.5，0.2，0.3）

乙模型经过softmax回归之后的预测答案是（0.7，0.1，0.2）

$$H(p,q) = -\sum_{x} p(x)\log q(x)$$

$$H\big((1,0,0),(0.5,0.2,0.3)\big) = -\log 0.5 \approx 0.301$$

$$H\big((1,0,0),(0.7,0.1,0.2)\big) = -\log 0.7 \approx 0.155$$

定义交叉熵损失函数

交叉熵损失函数定义为：

$$Loss = -\sum_{i=1}^{n} y_i \log y_i'$$

其中： y_i为标签值， y_i'为预测值

定义损失函数

```
loss_function = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred),
                                              reduction_indices=1)) # 交叉熵
```

浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

分类模型构建与训练实践

选择优化器

选择优化器

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss_function) #梯度下降
```

定义准确率

定义准确率

```
# 检查预测类别tf.argmax(pred, 1)与实际类别tf.argmax(y, 1)的匹配情况
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
```

```
# 准确率，将布尔值转化为浮点数，并计算平均值
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

argmax详解

```
arr1 = np.array([1, 3, 2, 5, 7, 0])
arr2 = np.array([[1, 2, 3], [3, 2, 1], [4, 7, 2], [8, 3, 2]])
print("arr1=", arr1)
print("arr2=\n", arr2)
```

```
arr1= [1 3 2 5 7 0]
arr2=
 [[1 2 3]
 [3 2 1]
 [4 7 2]
 [8 3 2]]
```

```
argmax_1 = tf.argmax(arr1)
argmax_20 = tf.argmax(arr2, 0)
argmax_21 = tf.argmax(arr2, 1)

with tf.Session() as sess:
    print(argmax_1.eval())
    print(argmax_20.eval())
    print(argmax_21.eval())
```

```
4
[3 2 0]
[2 0 1 0]
```

声明会话，初始化变量

```
sess = tf.Session() #声明会话
init = tf.global_variables_initializer() # 变量初始化
sess.run(init)
```

训练模型

```
# 开始训练
for epoch in range(train_epochs ):
    for batch in range(total_batch):
        xs, ys = mnist.train.next_batch(batch_size)# 读取批次数据
        sess.run(optimizer,feed_dict={x: xs,y: ys}) # 执行批次训练

    #total_batch个批次训练完成后，使用验证数据计算误差与准确率；验证集没有分批
    loss,acc = sess.run([loss_function,accuracy],
                        feed_dict={x: mnist.validation.images, y: mnist.validation.labels})
    # 打印训练过程中的详细信息
    if (epoch+1) % display_step == 0:
        print("Train Epoch:", '%02d' % (epoch+1), "Loss=", "{:.9f}".format(loss),\
              " Accuracy=","{:.4f}".format(acc))

print("Train Finished!")
```

```
Train Epoch: 01 Loss= 5.626945972  Accuracy= 0.2986
Train Epoch: 02 Loss= 3.634179592  Accuracy= 0.4492
Train Epoch: 03 Loss= 2.735285044  Accuracy= 0.5382
```

训练模型

```
Train Epoch: 39 Loss= 0.701643348  Accuracy= 0.8510
Train Epoch: 40 Loss= 0.695050657  Accuracy= 0.8518
Train Epoch: 41 Loss= 0.689112484  Accuracy= 0.8540
Train Epoch: 42 Loss= 0.682926178  Accuracy= 0.8556
Train Epoch: 43 Loss= 0.677166224  Accuracy= 0.8560
Train Epoch: 44 Loss= 0.671819627  Accuracy= 0.8568
Train Epoch: 45 Loss= 0.666360080  Accuracy= 0.8578
Train Epoch: 46 Loss= 0.661095500  Accuracy= 0.8574
Train Epoch: 47 Loss= 0.656263292  Accuracy= 0.8580
Train Epoch: 48 Loss= 0.651501536  Accuracy= 0.8598
Train Epoch: 49 Loss= 0.647264063  Accuracy= 0.8606
Train Epoch: 50 Loss= 0.642509282  Accuracy= 0.8606
Train Finished!
```

从上述打印结果可以看出损失值 **Loss** 是趋于更小的，同时，准确率 **Accuracy** 越来越高。

评估模型

完成训练后，在测试集上评估模型的准确率

```
accu_test =  sess.run(accuracy,
                      feed_dict={x: mnist.test.images, y: mnist.test.labels})

print("Test Accuracy:",accu_test)
```

```
Test Accuracy: 0.8646
```

评估模型

完成训练后，在验证集上评估模型的准确率

```
accu_validation =  sess.run(accuracy,
                    feed_dict={x: mnist.validation.images, y: mnist.validation.labels})

print("Test Accuracy:",accu_validation)
```

```
Test Accuracy: 0.8606
```

完成训练后，在训练集上评估模型的准确率

```
accu_train =  sess.run(accuracy,
                    feed_dict={x: mnist.train.images, y: mnist.train.labels})

print("Test Accuracy:",accu_train)
```

```
Test Accuracy: 0.855364
```

模型应用与可视化

应用模型

在建立模型并进行训练后，若认为准确率可以接受，则可以使用此模型进行预测

```
# 由于pred预测结果是one-hot编码格式，所以需要转换为0~9数字

prediction_result=sess.run(tf.argmax(pred,1),
                           feed_dict={x: mnist.test.images })
```

查看预测结果

```
#查看预测结果中的前10项

prediction_result[0:10]
```

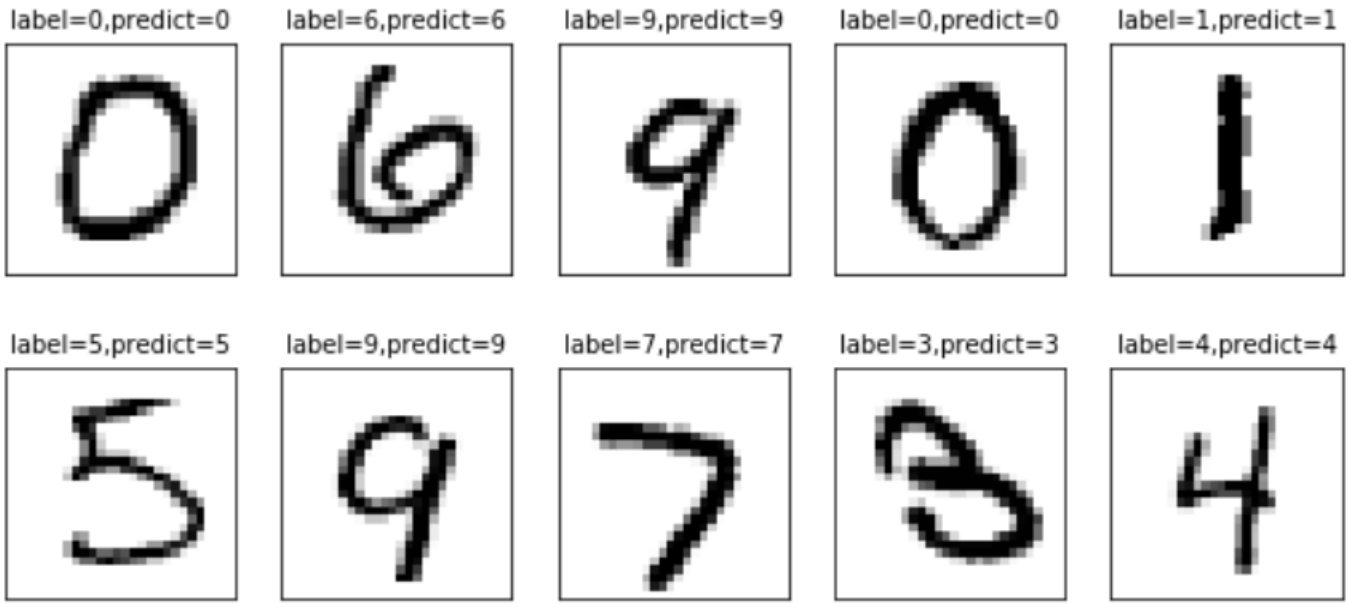
```
array([7, 2, 1, 0, 4, 1, 4, 5, 4, 9], dtype=int64)
```

定义可视化函数

```
import matplotlib.pyplot as plt
import numpy as np
def plot_images_labels_prediction(images,      # 图像列表
                                  labels,      # 标签列表
                                  prediction,  # 预测值列表
                                  index,       # 从第index个开始显示
                                  num=10 ):    # 缺省一次显示 10 幅
    fig = plt.gcf() # 获取当前图表，Get Current Figure
    fig.set_size_inches(10, 12)  # 1英寸等于 2.54 cm
    if num > 25:
        num = 25             # 最多显示25个子图
    for i in range(0, num):
        ax = plt.subplot(5,5, i+1) # 获取当前要处理的子图

        ax.imshow(np.reshape(images[index],(28, 28)),  # 显示第index个图像
                  cmap='binary')

        title = "label=" + str(np.argmax(labels[index]))  # 构建该图上要显示的title信息
        if len(prediction)>0:
            title += ",predict=" + str(prediction[index])

        ax.set_title(title,fontsize=10)    # 显示图上的title信息
        ax.set_xticks([]);  # 不显示坐标轴
        ax.set_yticks([])
        index += 1
    plt.show()
```

可视化预测结果

```
plot_images_labels_prediction(mnist.test.images,
                              mnist.test.labels,
                              prediction_result,10,10)
```



可视化预测结果

```
plot_images_labels_prediction(mnist.test.images,
                              mnist.test.labels,
                              prediction_result,10,25)
```