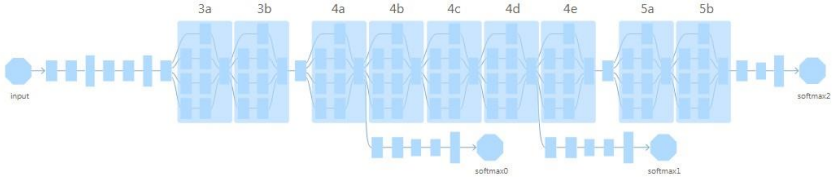
Google TensorFlow™

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE



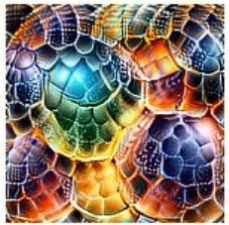
深度学习应用开发
基于TensorFlow的实践

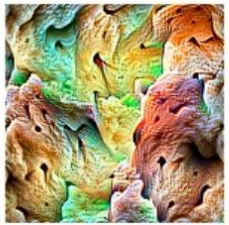
吴明晖 李卓蓉 金苍宏

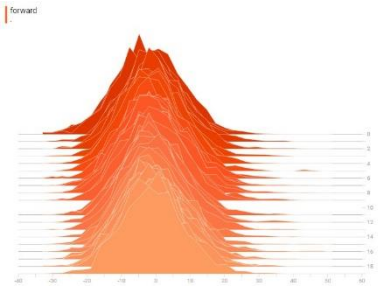
浙江大学城市学院

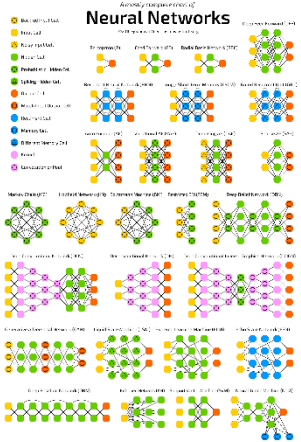
计算机与计算科学学院

Dept. of Computer Science
Zhejiang University City College

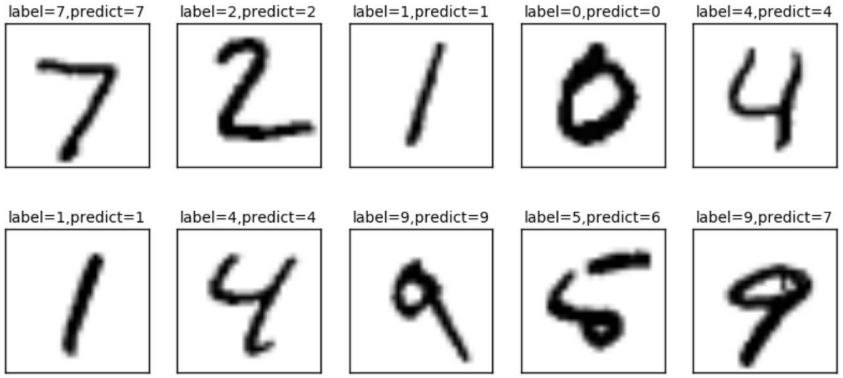








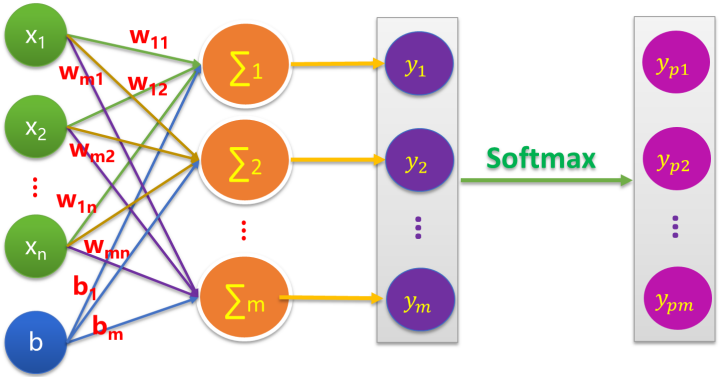
MNIST手写数字识别：分类应用入门

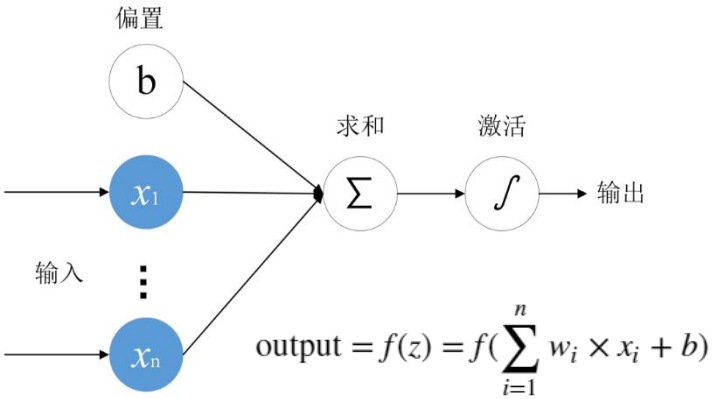



```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
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
```

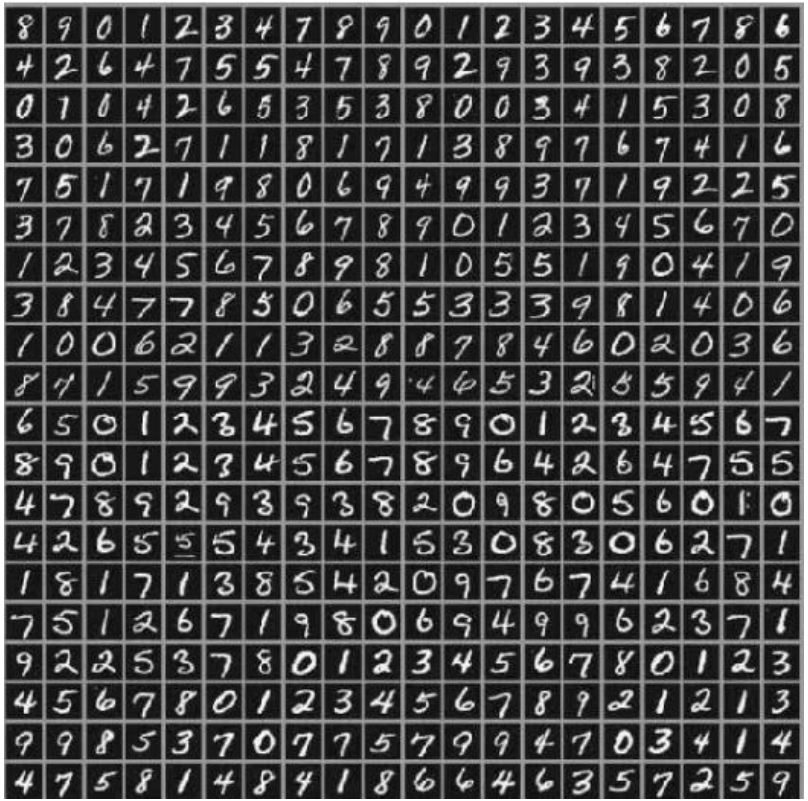
用神经元处理分类问题

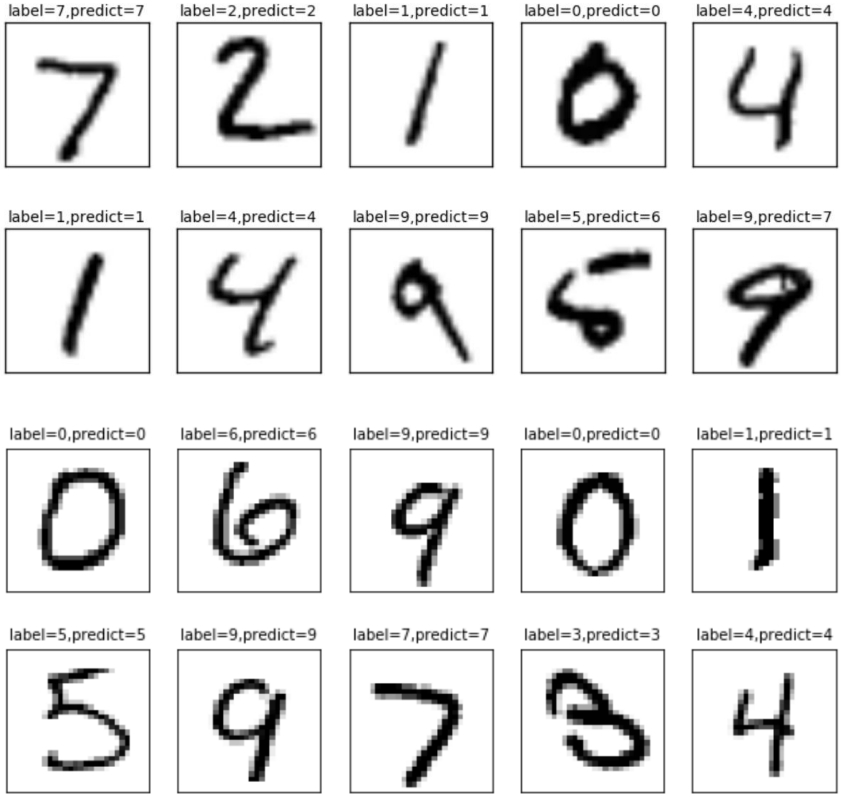




MNIST手写数字识别问题

MNIST手写数字识别：分类问题





MNIST手写数字识别数据集

MNIST 数据集来自美国国家标准与技术研究所，National Institute of Standards and Technology（NIST）.

数据集由来自 250 个不同人手写的数字构成，其中 50% 是高中学生，50% 来自人口普查局（the Census Bureau）的工作人员

训练集 60000　测试集 10000

MNIST手写数字识别数据集

MNIST 数据集可在 http://yann.lecun.com/exdb/mnist/ 获取

TensorFlow提供了数据集读取方法(1.x和2.0版本提供的方法不同)

```
import tensorflow as tf    # 导入Tensorflow
import numpy as np      # 导入numpy
import matplotlib.pyplot as plt # 导入matplotlib

# 在Jupyter中，使用matplotlib显示图像需要设置为 inline 模式，否则不会在网页里显示图像
%matplotlib inline

print("Tensorflow版本是：",tf.__version__) #显示当前TensorFlow版本
```

```
Tensorflow版本是： 2.0.0
```

MNIST手写数字识别数据集


```
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

MNIST数据集文件在读取时如果指定目录下不存在，则会自动去下载，需等待一定时间；如果已经存在了，则直接读取。

提示：

如果运行时出现网络连接错误，可以从 https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz 或 https://s3.amazonaws.com/img-datasets/mnist.npz 下载 MNIST 数据集 mnist.npz 文件，并放置于用户目录的 .keras/dataset 目录下（Windows 下用户目录为 C:\Users\用户名，Linux 下用户目录为 /home/用户名）

了解MNIST手写数字识别数据集

```
print("Train image shape:",train_images.shape,"Train label shape:",train_labels.shape)
print("Test image shape:",test_images.shape,"Test label shape:",test_labels.shape)
```

```
Train image shape: (60000, 28, 28) Train label shape: (60000,)
Test image shape: (10000, 28, 28) Test label shape: (10000,)
```

图像的大小是多少 ？

784

标签是怎样的？

具体看一幅image的数据

```
print("image data:",train_images[1])
```

```
image data: [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  51 159 253
  159  50   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  48 238 252 252
  252 237   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  54 227 253 252 239
  233 252  57   6   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  10  60 224 252 253 252 202
   84 252 253 122   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 163 252 252 252 253 252 252
   96 189 253 167   0   0   0   0   0   0]
```

具体看一幅image的数据

```
print("label data:",train_labels[1])
```

```
label data: 0
```

可视化 image

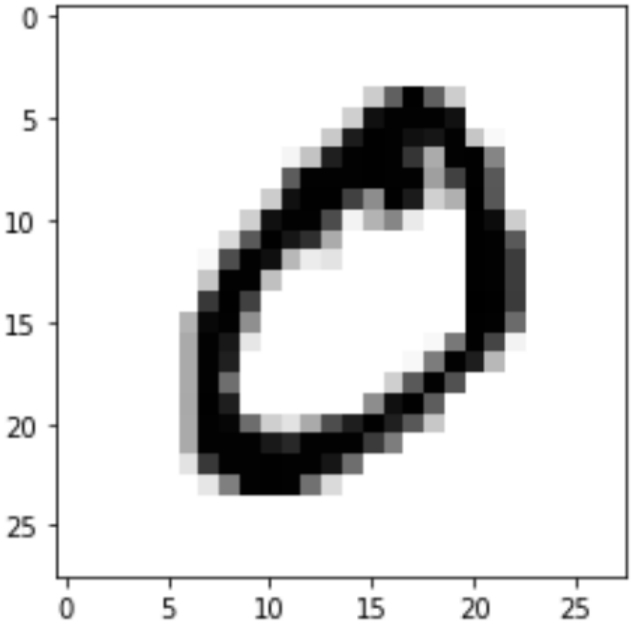
```
import matplotlib.pyplot as plt

def plot_image(image):
    plt.imshow(image.reshape(28,28),cmap='binary')
    plt.show()
```

plt.imshow()第二个参数是这个图像的模式参数，“binary”表示以灰度模式显示。
plt.imshow()函数中的图像数据参数支持一下数据形状：

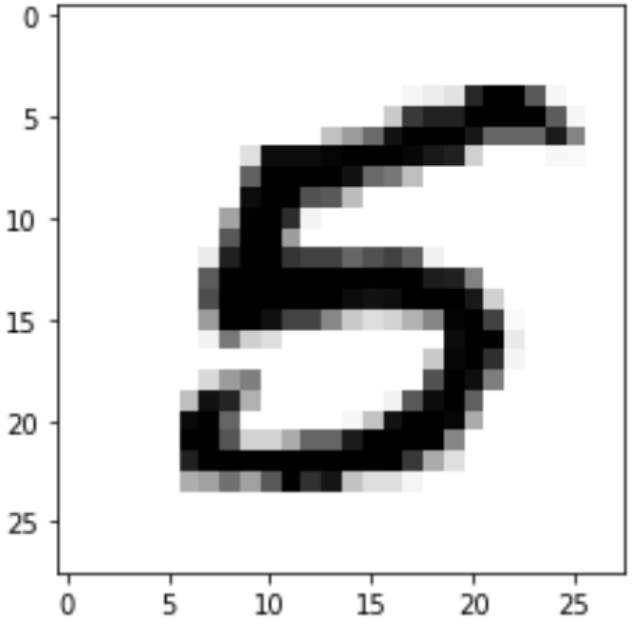
- （M，N）：二维数值，代表图像大小为M行N列，值为每个像素点的取值。
- （M，N，3）：三维度数值，代表图像大小为M行N列（即图片的高和宽），每个像素点的取值具有RGB三个通道的值（float或uint8）。
- 参数cmap缺省值为none，将把图像数据映射为彩色图显示

可视化 image

```
plot_image(train_images[1])
```




```
plot_image(train_images[20000])
```



进一步了解 reshape()

```
import numpy as np
int_array = np.array([i for i in range(64)])
print (int_array)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
 50 51 52 53 54 55 56 57 58 59 60 61 62 63]
```

```
int_array.reshape(8,8)
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20
       [24, 25, 26, 27, 28
       [32, 33, 34, 35, 36
       [40, 41, 42, 43, 44
       [48, 49, 50, 51, 52
       [56, 57, 58, 59, 60
```

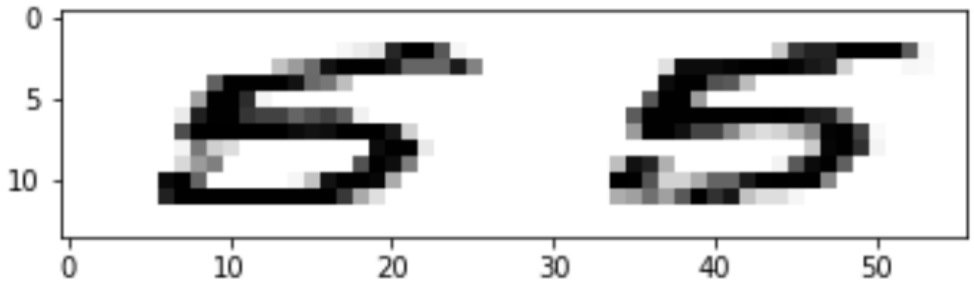
行优先，逐行排列

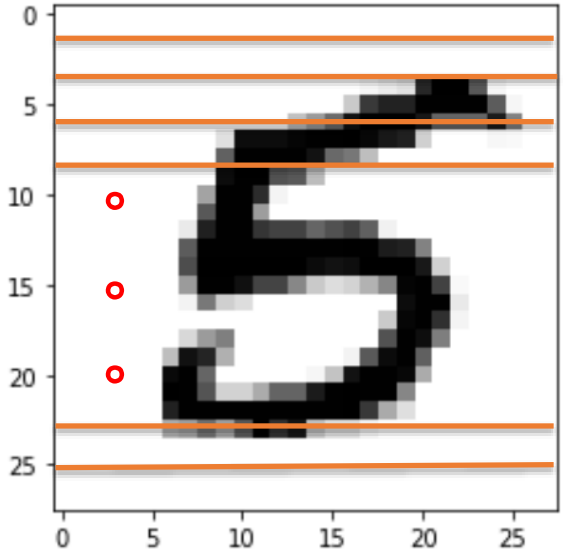
```
int_array.reshape(4,16)
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31],
       [32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47],
       [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63]])
```

思考：以下代码会输出什么图像？

```
plt.imshow(mnist.train.images[20000].reshape(14,56),cmap='binary')
plt.show()
```





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：

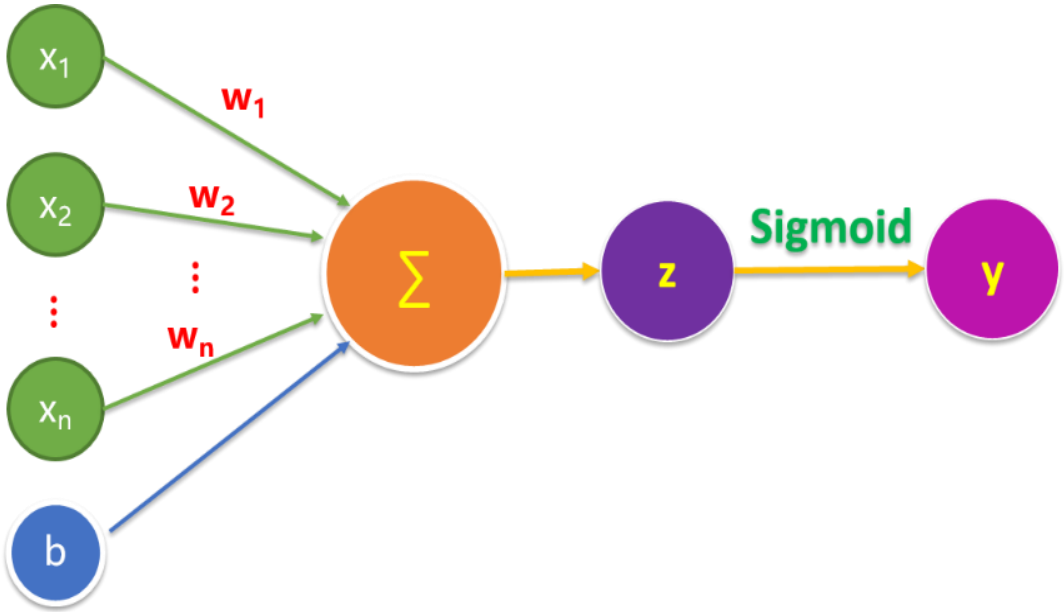
- 根据邮件内容判断是垃圾邮件的可能性
- 根据医学影像判断肿瘤是恶性的可能性
- 手写数字分别是 0、1、2、3、4、5、6、7、8、9的可能性（概率）

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

逻辑回归基本模型

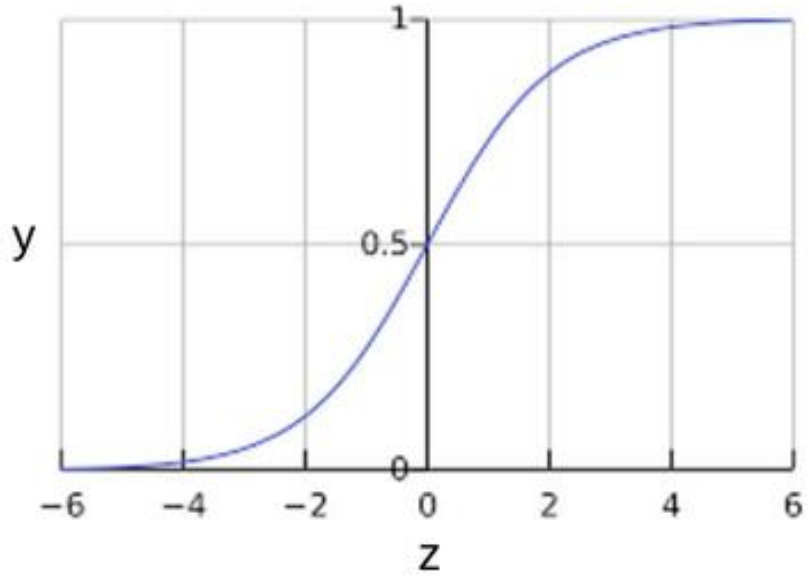


Sigmoid函数

逻辑回归模型如何确保输出值始终落在 0 和 1 之间。

Sigmoid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

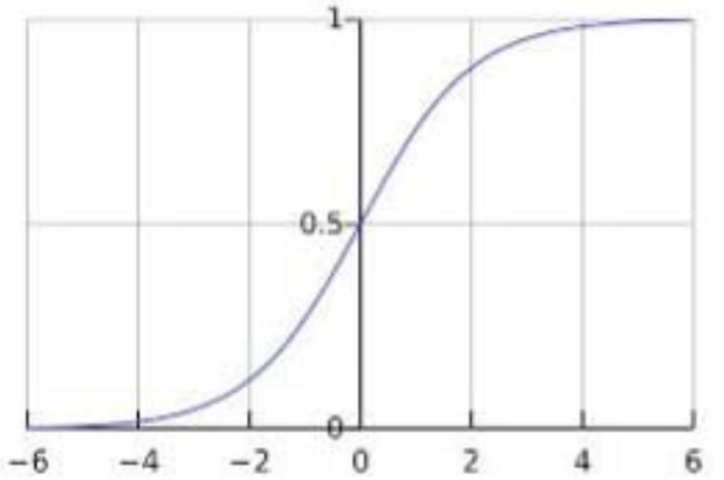
sigmoid函数连续可微分

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

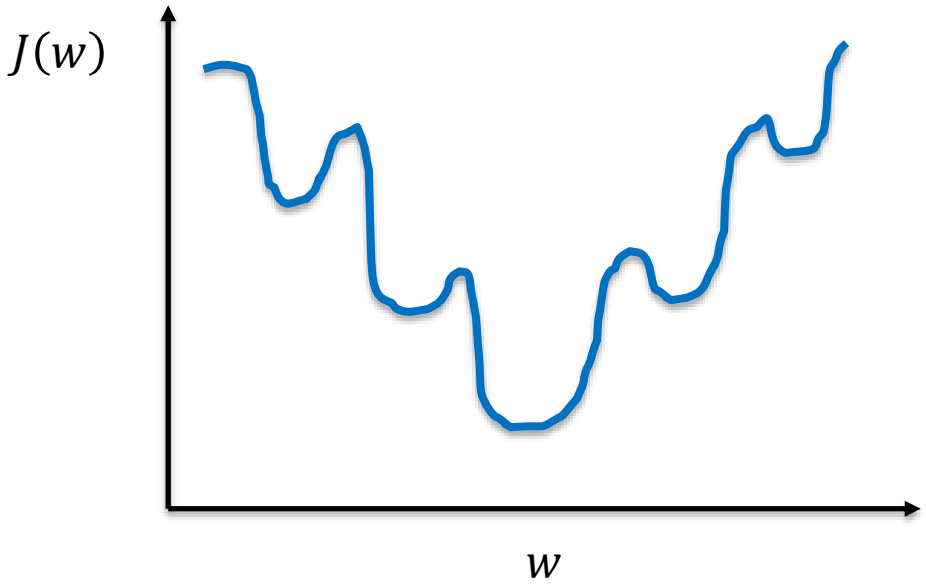
$$J(w) = \sum_{i=1}^{n} \frac{1}{n} (\varphi(z_i) - y_i)^2$$

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

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：

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

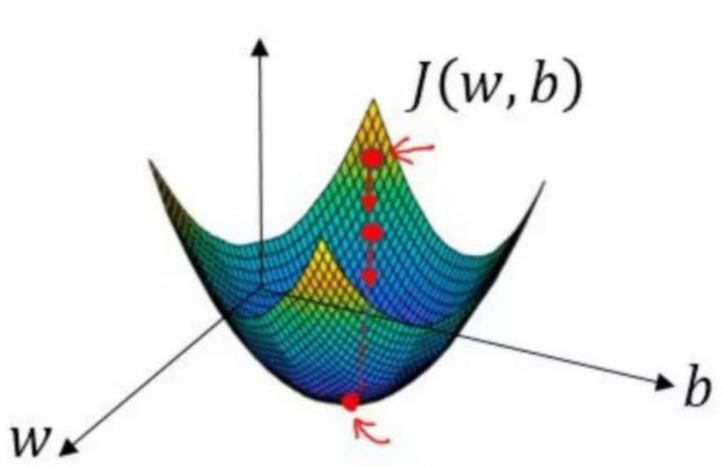
凸函数

其中：

$(x,y) \in D$是有标签样本 (x,y) 的数据集

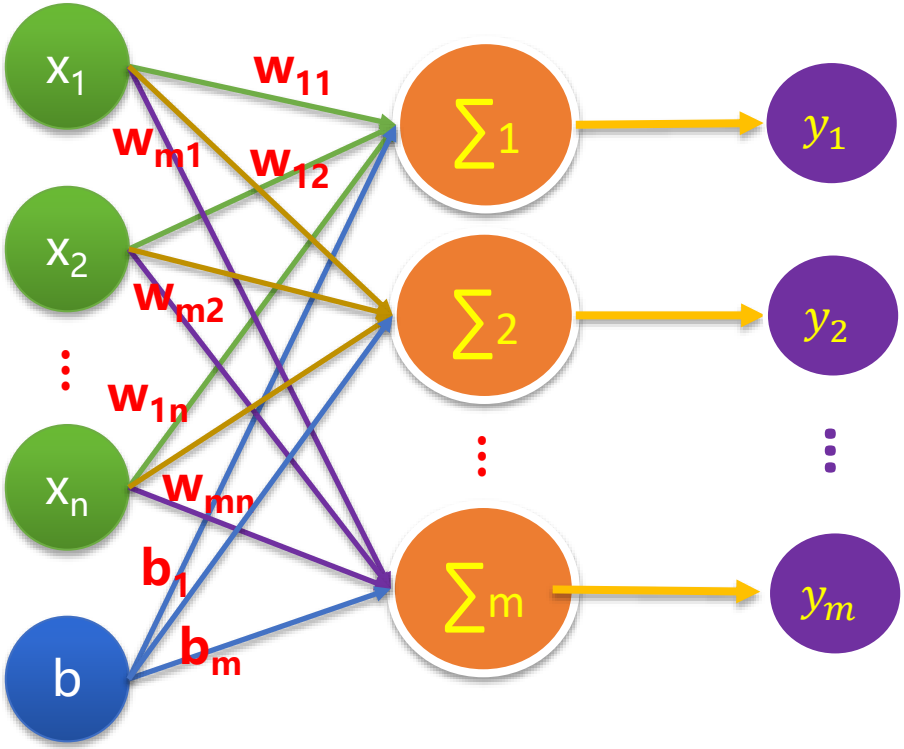
y是有标签样本中的标签，取值必须是 0 或 1

y'是对于特征集x的预测值（介于 0 和 1 之间）

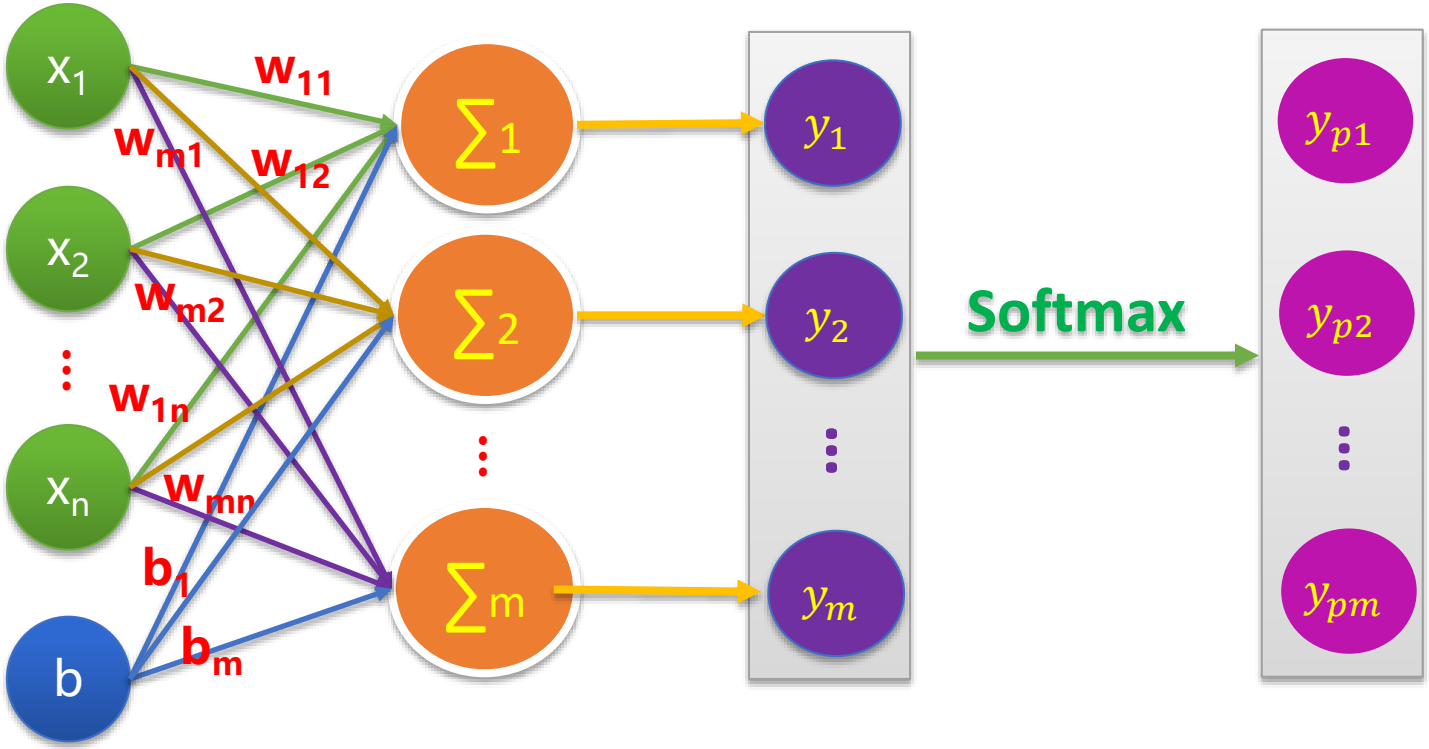


多元分类

多元分类基本模型



多元分类基本模型


x1
x2
xn
b
w11
wm1
w12
wm2
w1n
wmn
b1
bm
∑1
∑2
∑m
y1
y2
ym
Softmax
yp1
yp2
ypm

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。

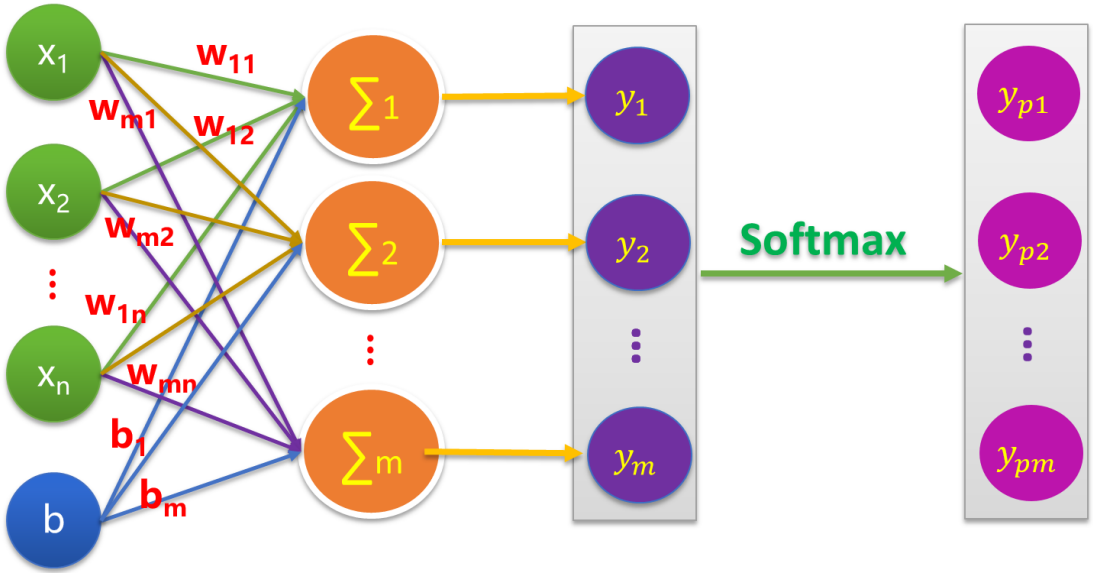
Softmax示例

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

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

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

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

多分类问题中的标签数据与独热编码

直接采用数字做标签的问题

机器学习算法中，特征之间距离的计算或相似度的常用计算方法都是基于欧式空间的

能说 1 比 8 更相似于 3 吗?

数字	0	1	2	3	4	5	6	7	8	9
编码	0	1	2	3	4	5	6	7	8	9

为什么要采用one hot编码

1 将离散特征的取值扩展到了欧式空间，离散特征的某个取值就对应欧式空间的某个点

2 机器学习算法中，特征之间距离的计算或相似度的常用计算方法都是基于欧式空间的

3 将离散型特征使用one-hot编码，会让特征之间的距离计算更加合理

数字	0	1	2	3	4	5	6	7	8	9
编码	0	1	2	3	4	5	6	7	8	9

```
array([ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.])
```

独热编码（one hot encoding）

一种稀疏向量，其中：

- 一个元素设为 1
- 所有其他元素均设为 0

独热编码常用于表示拥有有限个可能值的字符串或标识符

例如：假设某个植物学数据集记录了 15000 个不同的物种，其中每个物种都用独一无二的字符串标识符来表示。在特征工程过程中，可能需要将这些字符串标识符编码为独热向量，向量的大小为 15000

```
mnist.train.labels[1]
```

```
array([ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.])
```

独热编码示例

```
x=[3,4]

tf.one_hot(x, depth=10)
```

```
]: <tf.Tensor: id=10, shape=(2, 10), dtype=float32, numpy=
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]], dtype=float32)>
```

独热编码如何取值？

对于一维的数据，argmax（）返回最大值的下标

argmax返回的是最大数的索引

```
A=tf.constant([3,20,60,7,6])

print(tf.argmax(A).numpy())
```

```
2
```

对于二维的数据，如果指定轴的参数axis=0，则按第0轴（行）中的元素取值，查找同列中最大值并输出其位置（行号），因此返回向量中元素个数和列数相同。如果指定轴的参数axis=1，则按第1轴（列）中的元素取值，查找同行中最大值并输出其位置（列号），因此返回向量中元素个数和行数相同。

```
B=tf.constant([[3,20,60,7,6],
               [2,11,8,1,87],
               [14,57,33,5,21]])

print(tf.math.argmax(B,axis=0).numpy())
print(tf.math.argmax(B,axis=1).numpy())
```

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

独热编码如何取值？

在Numpy中也提供了argmax()方法，针对np.array数据可以直接应用

```
C=np.array([[3,20,60,7,6],[2,11,8,1,87],[14,57,33,5,21]])
print(np.argmax(C,axis=1))
```

```
[2 4 1]
```

交叉熵损失

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

MNIST手写数字识别案例
TensorFlow 2.0 实践

数据获取

```
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

数据集划分

划分验证集

In [14]:
```
total_num = len(train_images)
valid_split = 0.2    # 验证集的比例占20%
train_num = int(total_num*(1-valid_split))    #训练集的数目

train_x = train_images[:train_num]    # 前部分给训练集
train_y = train_labels[:train_num]

valid_x = train_images[train_num:]    # 后20%给验证集
valid_y = train_labels[train_num:]

test_x = test_images
test_y = test_labels
```

In [15]:
```
valid_x.shape
```

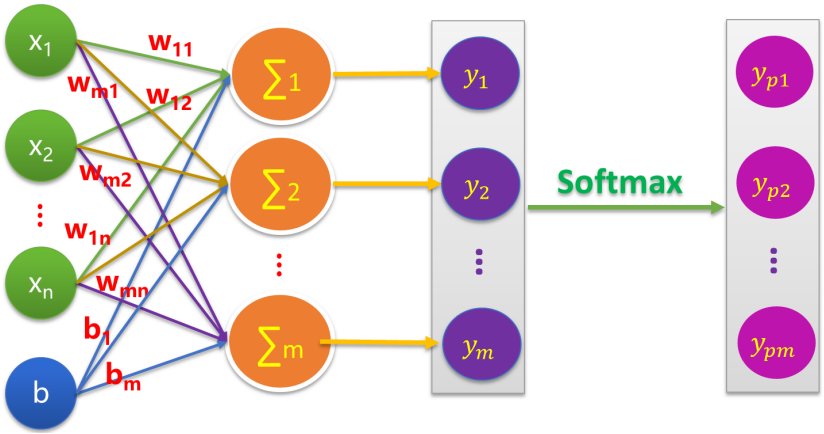
Out[15]: (12000, 28, 28)

数据塑形

[16]:

```
# 把（28 28）的结构拉直为一行 784
train_x = train_x.reshape(-1,784)
valid_x = valid_x.reshape(-1,784)
test_x = test_x.reshape(-1,784)
```

模型的输入数据形状



特征数据归一化

```
train_x = tf.cast(train_x/255.0, tf.float32)
valid_x = tf.cast(valid_x/255.0, tf.float32)
test_x = tf.cast(test_x/255.0, tf.float32)
```

```
train_x[1]
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.21176471, 0.8901961 , 0.99215686, 0.9882353 ,
0.9372549 , 0.9137255 , 0.9882353 , 0.22352941, 0.02352941,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.03921569, 0.23529412, 0.8784314 ,
0.9882353 , 0.99215686, 0.9882353 , 0.7921569 , 0.32941177,
0.9882353 , 0.99215686, 0.47843137, 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

标签数据独热编码

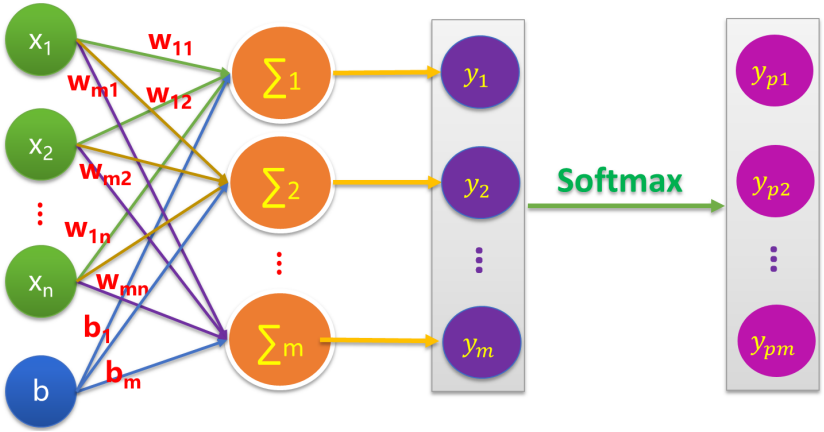
```
# 对标签数据进行独热编码
train_y = tf.one_hot(train_y, depth=10)
valid_y = tf.one_hot(valid_y, depth=10)
test_y = tf.one_hot(test_y, depth=10)
```

```
train_y
```

```
]: <tf.Tensor: id=19, shape=(48000, 10), dtype=float32, numpy=
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.]], dtype=float32)>
```

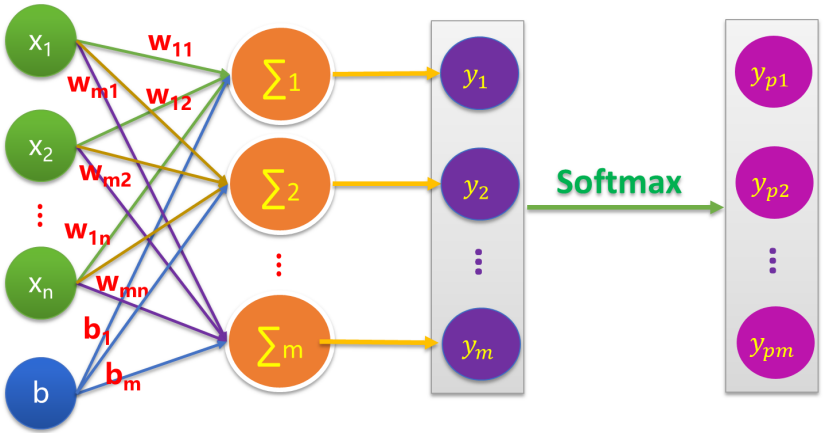
构建模型

```
def model(x, w, b):
    pred = tf.matmul(x, w) + b
    return tf.nn.softmax(pred)
```



定义模型变量

```
# 定义变量
W = tf.Variable(tf.random.normal([784, 10],mean=0.0, stddev=1.0, dtype=tf.float32))
B = tf.Variable(tf.zeros([10]),dtype = tf.float32)
```



定义损失函数

定义交叉熵损失函数

```python
# 定义交叉熵损失函数

def loss(x, y, w, b):
    pred = model(x, w, b) #  计算模型预测值和标签值的差异
    loss_ = tf.keras.losses.categorical_crossentropy(y_true=y, y_pred=pred)
    return tf.reduce_mean(loss_)  # 求均值，得出均方差.
```

在自定义的损失函数loss中直接调用了TensorFlow提供的交叉熵函数。

交叉熵函数接口

```
tf.keras.losses.categorical_crossentropy(y_true,
                                         y_pred,
                                         from_logits=False)
```

其中：y_true参数为独热编码后的真实标签；
y_pred参数为模型计算出的预测值；
from_logits参数缺省值为False，当取值为False时，y_pred参数应该传入经过Softmax函数计算后的输出，当from_logits参数设置为True时，y_pred应该代入没有经过Softmax函数计算的值

在计算交叉熵时，会因为计算log(0)出现数值溢出的问题,在Softmax计算中也会出现类似数据溢出的问题，导致计算结果不稳定

定义训练超参数

```
training_epochs = 20 # 训练轮数
batch_size = 50  # 单次训练样本数（批次大小）
learning_rate= 0.001  # 学习率
```

定义梯度计算函数

```
# 计算样本数据[x,y]在参数[w,b]点上的梯度
def grad(x, y, w, b):
    with tf.GradientTape() as tape:
        loss_ = loss(x, y, w, b)
    return tape.gradient(loss_, [w, b])# 返回梯度向量
```

选择优化器

```
#Adam优化器

optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
```

常用优化器：

SGD、Adagrad、Adadelta、RMSprop、Adam

定义准确率

```
def accuracy(x, y, w, b):
    pred = model(x, w, b) #  计算模型预测值和标签值的差异
    # 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
    return tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

训练模型

```
total_step= int(train_num/batch_size)  # 一轮训练有多少批次

loss_list_train = [] # 用于保存训练集loss值的列表
loss_list_valid = [] # 用于保存验证集loss值的列表
acc_list_train = [] # 用于保存训练集Acc值的列表
acc_list_valid = [] # 用于保存验证集Acc值的列表

for epoch in range (training_epochs):
    for step in range(total_step):
        xs = train_x[step*batch_size:(step+1)*batch_size]
        ys = train_y[step*batch_size:(step+1)*batch_size]

        grads = grad(xs, ys, W, B)# 计算梯度
        optimizer.apply_gradients(zip(grads, [W,B])) # 优化器根据梯度自动调整变量w和b

    loss_train = loss(train_x, train_y, W, B).numpy()    # 计算当前轮训练损失
    loss_valid = loss(valid_x, valid_y, W, B).numpy()    # 计算当前轮验证损失
    acc_train = accuracy(train_x, train_y, W, B).numpy()
    acc_valid = accuracy(valid_x, valid_y, W, B).numpy()
    loss_list_train.append(loss_train)
    loss_list_valid.append(loss_valid)
    acc_list_train.append(acc_train)
    acc_list_valid.append(acc_valid)
    print("epoch={:3d},train_loss={:.4f},train_acc={:.4f},val_loss={:.4f},val_acc={:.4f}".format(
        epoch+1, loss_train, acc_train,loss_valid, acc_valid))
```

训练模型

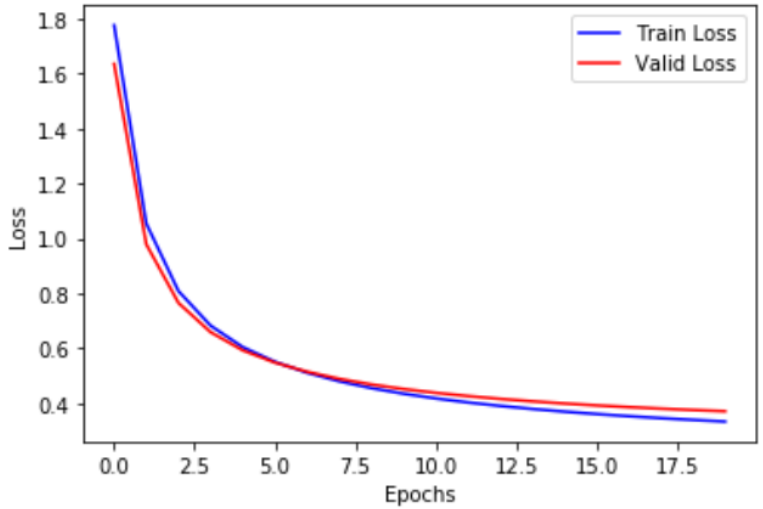
```
epoch=  1, train_loss=1.7756, train_acc=0.6651, val_loss=1.6352, val_acc=0.6852
epoch=  2, train_loss=1.0546, train_acc=0.7787, val_loss=0.9789, val_acc=0.7940
epoch=  3, train_loss=0.8089, train_acc=0.8242, val_loss=0.7659, val_acc=0.8367
epoch=  4, train_loss=0.6832, train_acc=0.8488, val_loss=0.6593, val_acc=0.8565
epoch=  5, train_loss=0.6050, train_acc=0.8636, val_loss=0.5932, val_acc=0.8681
epoch=  6, train_loss=0.5516, train_acc=0.8737, val_loss=0.5487, val_acc=0.8760
epoch=  7, train_loss=0.5118, train_acc=0.8811, val_loss=0.5154, val_acc=0.8813
epoch=  8, train_loss=0.4807, train_acc=0.8864, val_loss=0.4896, val_acc=0.8855
epoch=  9, train_loss=0.4558, train_acc=0.8909, val_loss=0.4692, val_acc=0.8889
epoch= 10, train_loss=0.4354, train_acc=0.8945, val_loss=0.4525, val_acc=0.8916
epoch= 11, train_loss=0.4184, train_acc=0.8981, val_loss=0.4387, val_acc=0.8942
epoch= 12, train_loss=0.4038, train_acc=0.9007, val_loss=0.4269, val_acc=0.8963
epoch= 13, train_loss=0.3912, train_acc=0.9030, val_loss=0.4167, val_acc=0.8982
epoch= 14, train_loss=0.3801, train_acc=0.9053, val_loss=0.4078, val_acc=0.9003
epoch= 15, train_loss=0.3703, train_acc=0.9073, val_loss=0.3999, val_acc=0.9017
epoch= 16, train_loss=0.3616, train_acc=0.9091, val_loss=0.3930, val_acc=0.9029
epoch= 17, train_loss=0.3537, train_acc=0.9100, val_loss=0.3868, val_acc=0.9038
epoch= 18, train_loss=0.3466, train_acc=0.9114, val_loss=0.3812, val_acc=0.9053
epoch= 19, train_loss=0.3402, train_acc=0.9124, val_loss=0.3762, val_acc=0.9072
epoch= 20, train_loss=0.3343, train_acc=0.9129, val_loss=0.3717, val_acc=0.9076
```

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。

显示训练过程数据

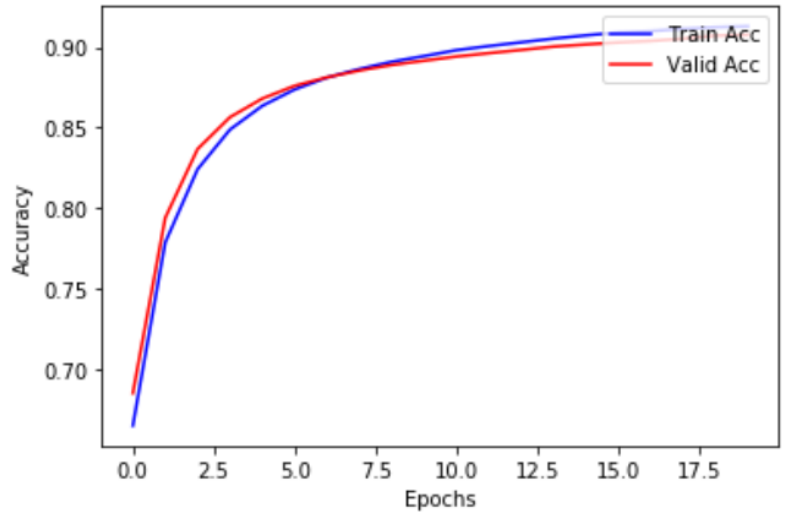
```
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.plot(loss_list_train,'blue',label="Train Loss")
plt.plot(loss_list_valid,'red',label="Valid Loss")
plt.legend(loc=1)# 通过参数loc指定图例位置
```

]: <matplotlib.legend.Legend at 0x170a275ee08>




```
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.plot(acc_list_train,'blue',label="Train Acc")
plt.plot(acc_list_valid,'red',label="Valid Acc")
plt.legend(loc=1)# 通过参数loc指定图例位置
```

]: <matplotlib.legend.Legend at 0x170a27e6108>



评估模型

完成训练后，在测试集上评估模型的准确率

```
acc_test = accuracy(test_x, test_y, W, B).numpy()
print("Test accuracy:",acc_test)
```

```
Test accuracy: 0.9076
```

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

```
# 定义预测函数
def predict(x,  w, b):
    pred = model(x, w, b) #  计算模型预测值
    result = tf.argmax(pred, 1).numpy()
    return result
```

```
pred_test=predict(test_x, W, B)
```

```
pred_test[0]
```

35]: 7

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
                                  preds,   # 预测值列表
                                  index=0,        # 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, 4)  # 1英寸等于 2.54 cm
    if num > 10:
        num = 10            # 最多显示10个子图
    for i in range(0, num):
        ax = plt.subplot(2,5, i+1) # 获取当前要处理的子图

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

        title = "label=" + str(labels[index])  # 构建该图上要显示的title信息
        if len(preds)>0:
            title += ",predict=" + str(preds[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 = index + 1
    plt.show()
```

可视化预测结果

```
plot_images_labels_prediction(test_images, test_labels,pred_test,10,10)
```