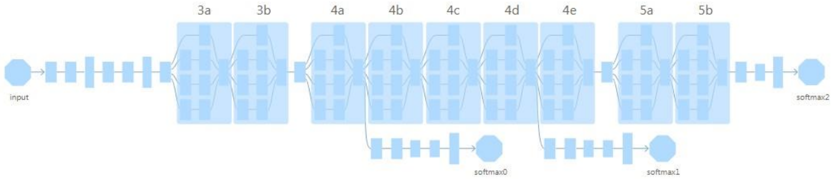


深度学习应用开发

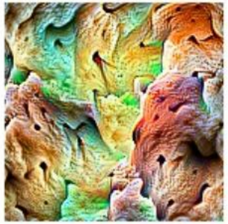
基于TensorFlow的实践

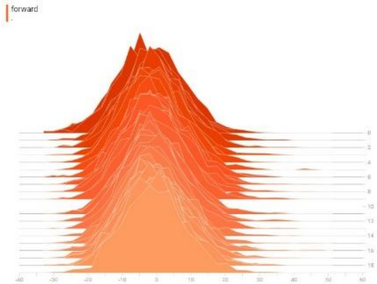
吴明晖 李卓蓉 金苍宏

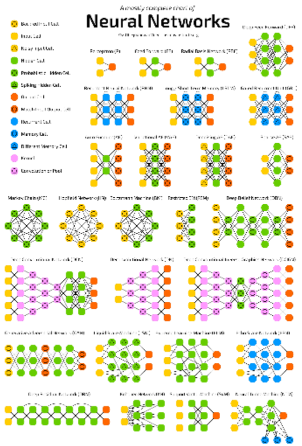
浙江大学城市学院

计算机与计算科学学院

Dept. of Computer Science
Zhejiang University City College



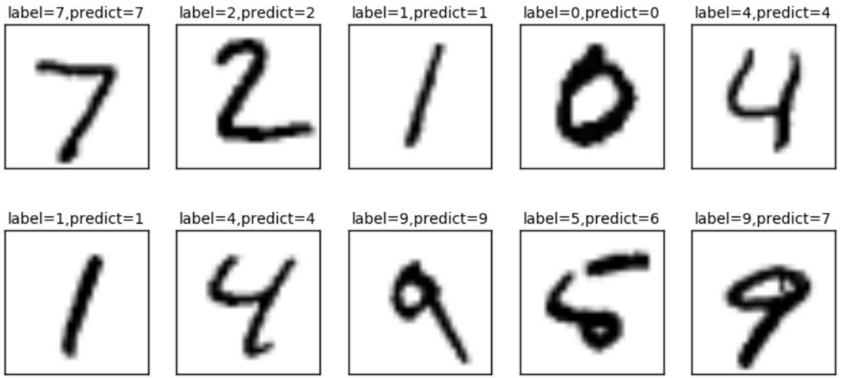
浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

MNIST手写数字识别进阶

多层神经网络与应用

MNIST手写数字识别：分类应用入门

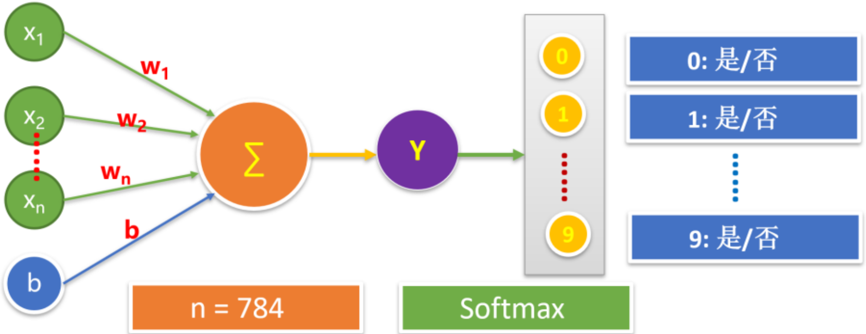
浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

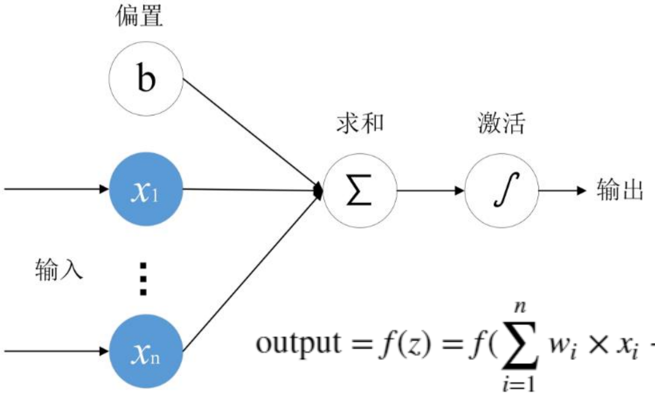



```python
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
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
```

一个神经元处理分类问题



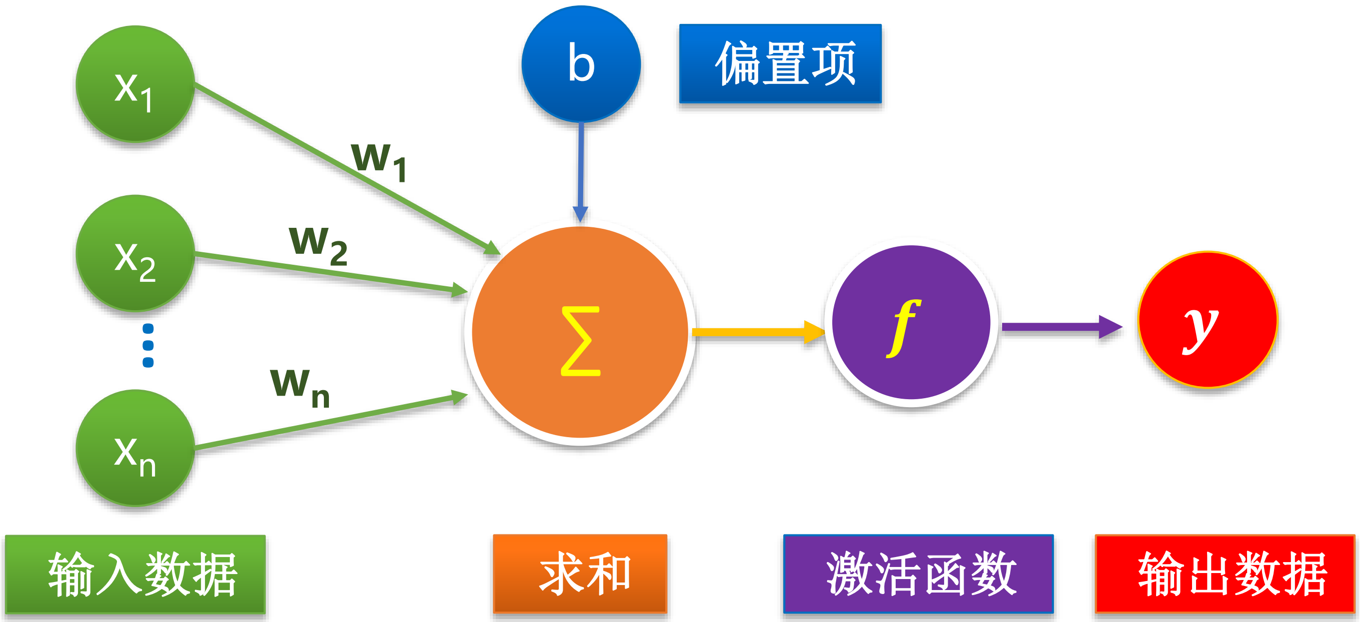


$$output = f(z) = f(\sum_{i=1}^{n} w_i \times x_i + b)$$

单个神经元模型

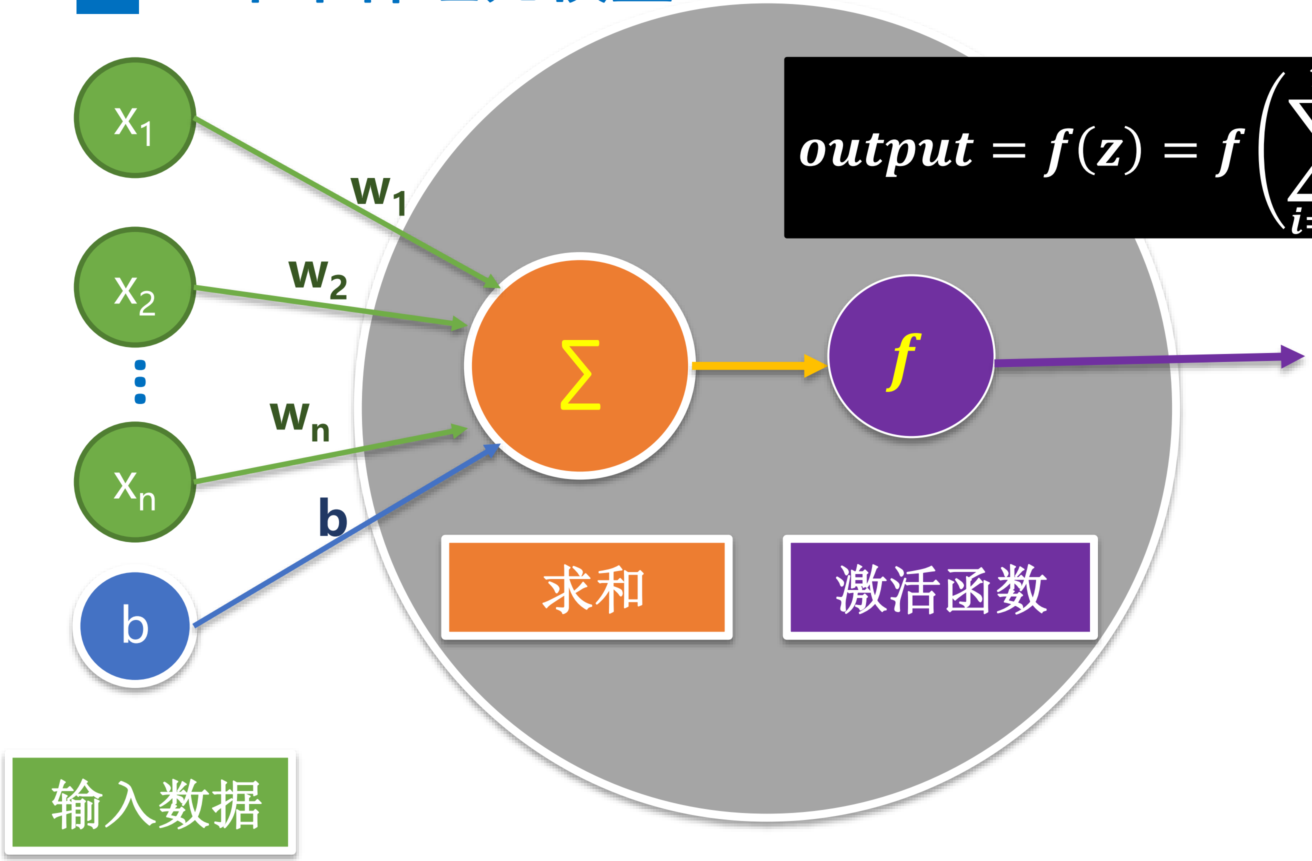
浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

$$output = f(z) = f\left(\sum_{i=1}^{n}(x_i * w_i) + b\right)$$

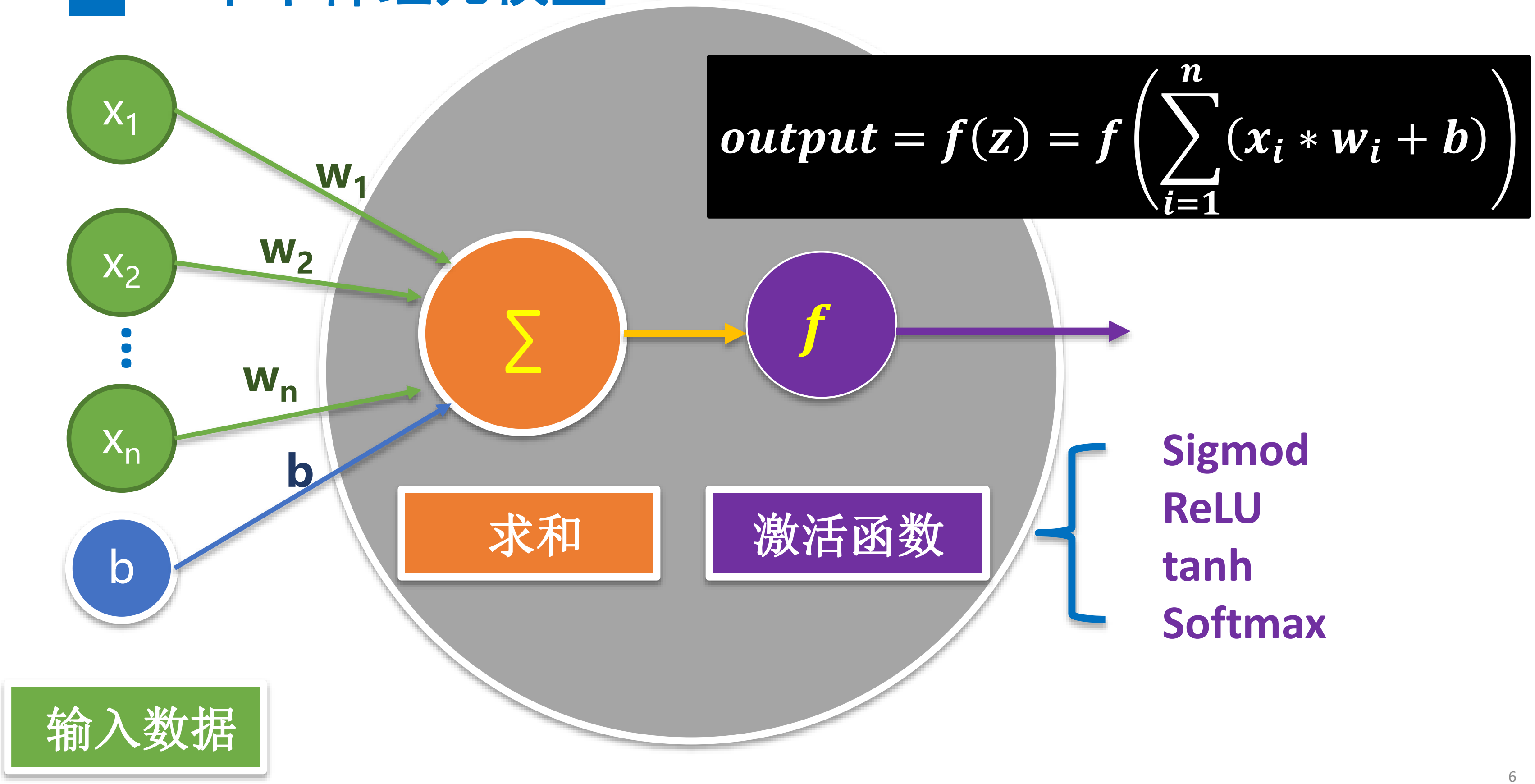


单个神经元模型

$$output = f(z) = f\left(\sum_{i=1}^{n}(x_i * w_i + b)\right)$$



单个神经元模型

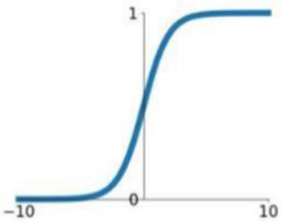


$$output = f(z) = f\left(\sum_{i=1}^{n}(x_i * w_i + b)\right)$$

常见激活函数

Sigmoid

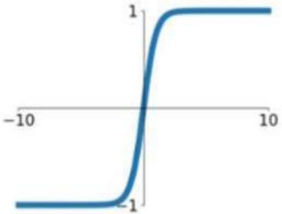
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



S型函数

tanh

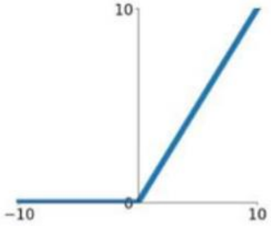
$$\tanh(x)$$



双曲正切函数

ReLU

$$\max(0, x)$$



修正线性单元函数

MNIST手写数字识别：单神经元模型效果

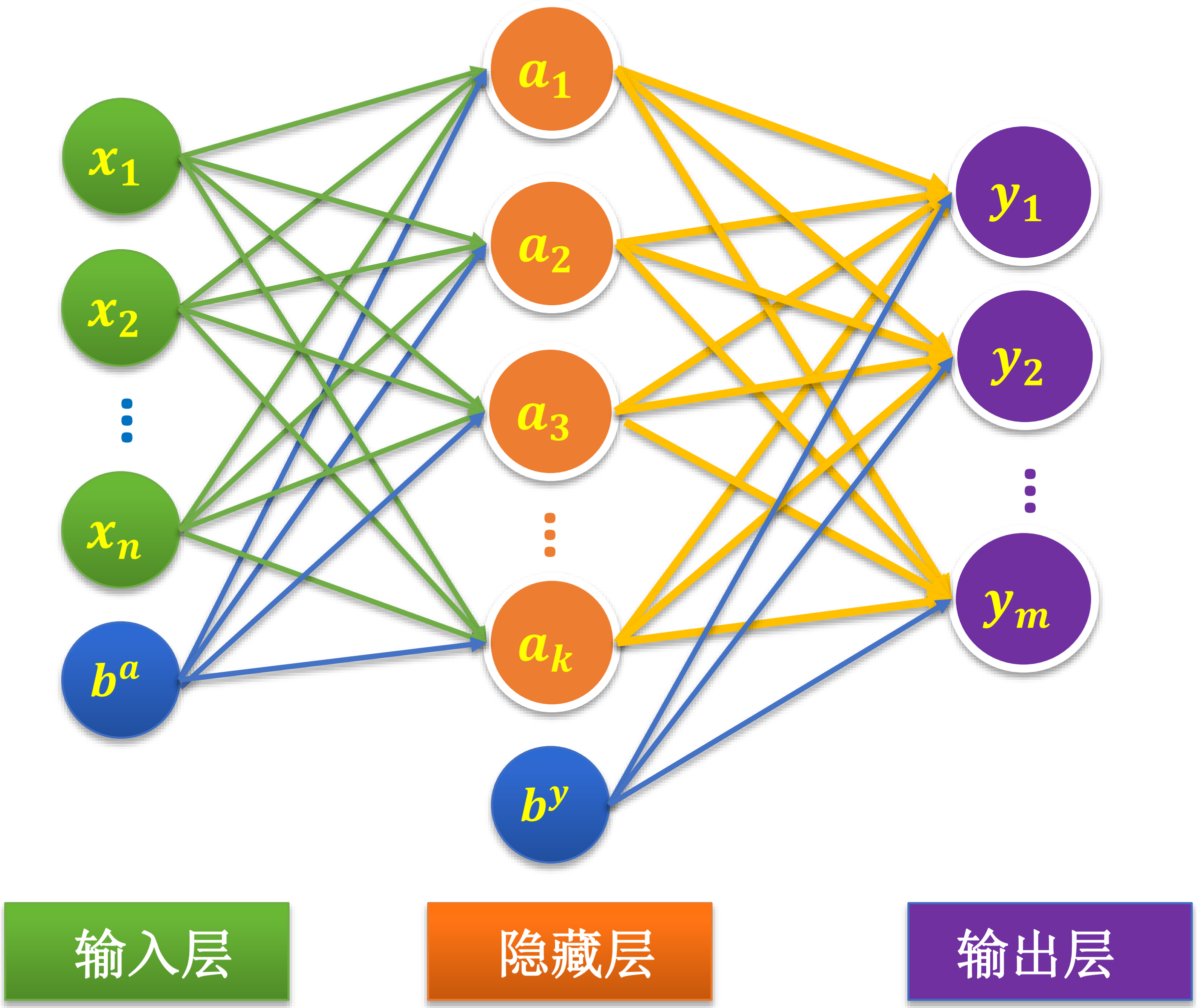
```
training_epochs = 20 # 训练轮数
batch_size = 50   # 单次训练样本数（批次大小）
learning_rate= 0.001   # 学习率
```

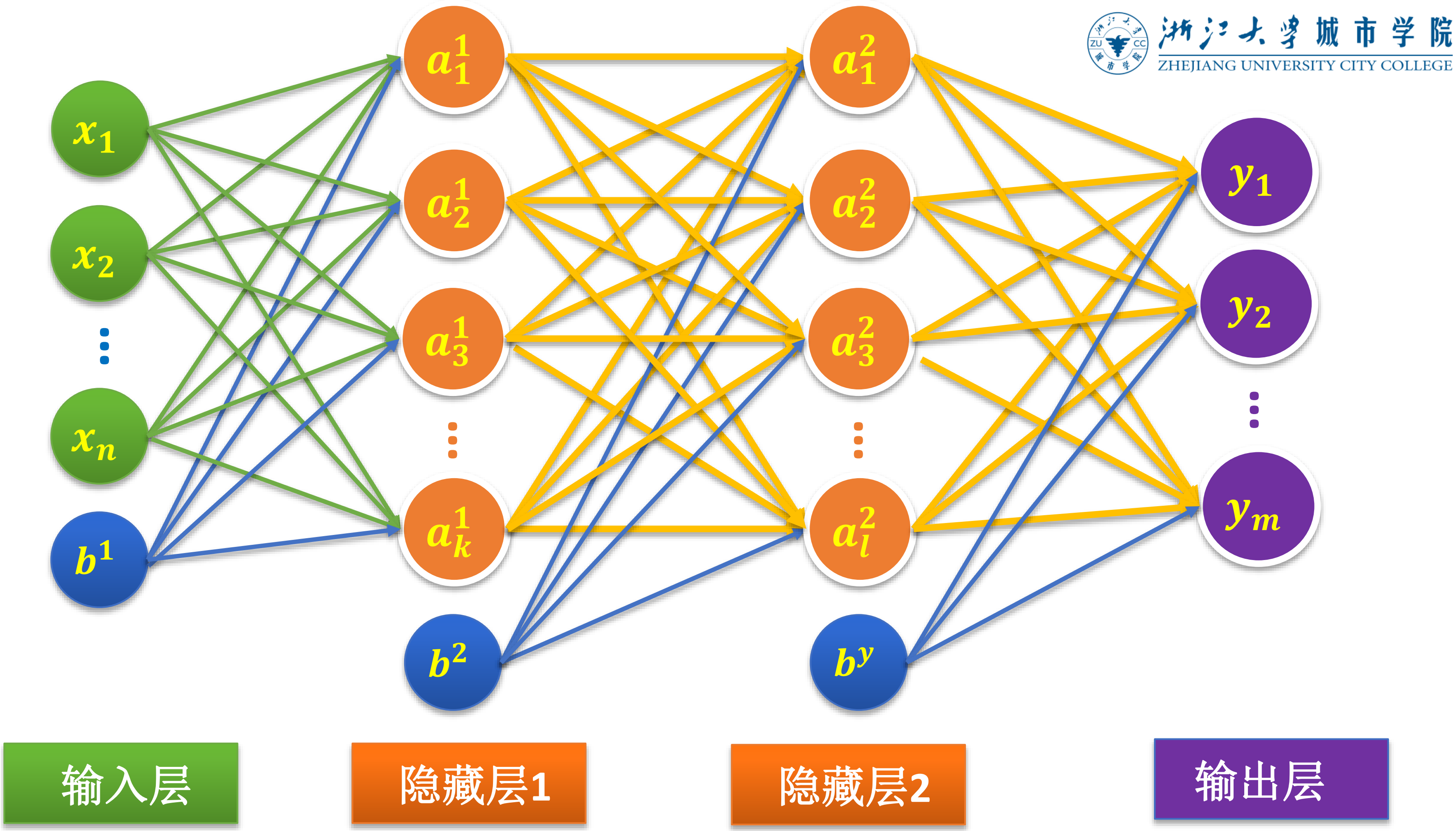
```
epoch= 10, train_loss=0.4354, train_acc=0.8945, val_loss=0.4525, val_acc=0.8916
epoch= 11, train_loss=0.4184, train_acc=0.8981, val_loss=0.4387, val_acc=0.8942
epoch= 12, train_loss=0.4038, train_acc=0.9007, val_loss=0.4269, val_acc=0.8963
epoch= 13, train_loss=0.3912, train_acc=0.9030, val_loss=0.4167, 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
```

想要更加准确？多一点神经元

浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

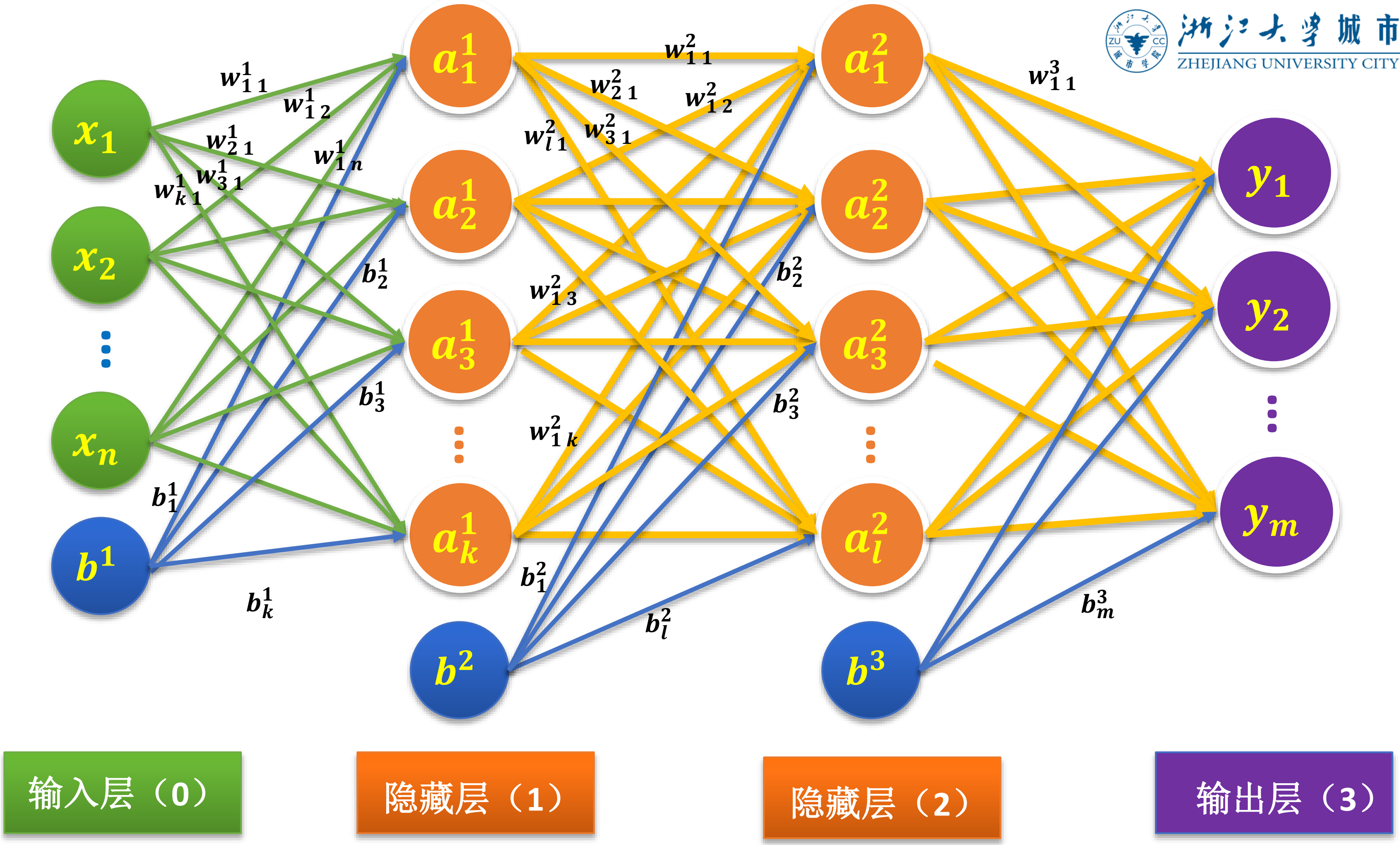
全连接单隐含层神经网络





浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE
x_1
x_2
x_n
b^1
a_1^1
a_2^1
a_3^1
a_k^1
b^2
a_1^2
a_2^2
a_3^2
a_l^2
b^y
y_1
y_2
y_m
输入层
隐藏层1
隐藏层2
输出层

多层全连接神经网络的计算表达

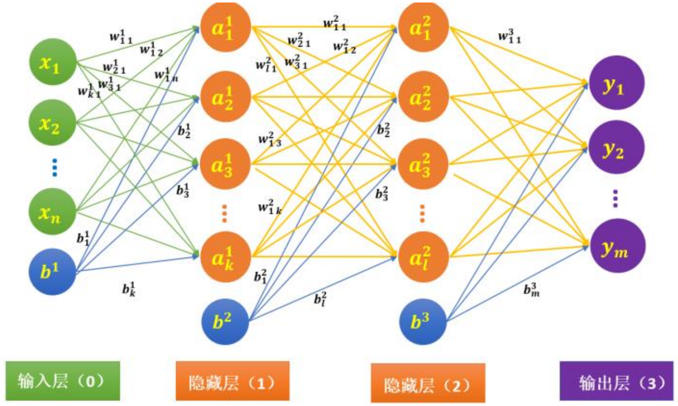

x_1
x_2
x_n
b^1
$w_{1\,1}^1$
$w_{1\,2}^1$
$w_{2\,1}^1$
$w_{1\,n}^1$
$w_{3\,1}^1$
$w_{k\,1}^1$
b_1^1
b_2^1
b_3^1
b_k^1
a_1^1
a_2^1
a_3^1
a_k^1
b^2
$w_{1\,1}^2$
$w_{2\,1}^2$
$w_{1\,2}^2$
$w_{l\,1}^2$
$w_{3\,1}^2$
$w_{1\,3}^2$
$w_{1\,k}^2$
b_1^2
b_2^2
b_3^2
b_l^2
a_1^2
a_2^2
a_3^2
a_l^2
b^3
$w_{1\,1}^3$
b_m^3
y_1
y_2
y_m
输入层（0）
隐藏层（1）
隐藏层（2）
输出层（3）

多层全连接神经网络的计算表达

第一层的第i个节点a_i^1的值可以这样实现：

$$a_i^1 = f\left(x_1 * w_{i\,1}^1 + x_2 * w_{i\,2}^1 + \cdots + x_n * w_{i\,n}^1 + b_i^1\right)$$

$$a_i^1 = f\left(\begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} * \begin{bmatrix} w_{i\,1}^1 \\ w_{i\,2}^1 \\ \vdots \\ w_{i\,n}^1 \end{bmatrix} + b_i^1\right)$$



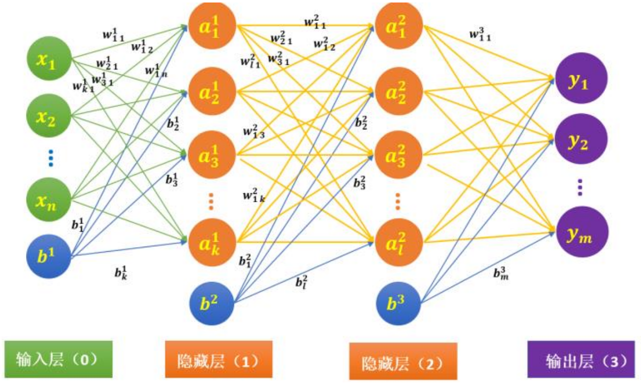
多层全连接神经网络的计算表达

第一层的每个节点的值计算都整合到一起形成整层的计算：

$$A^1 = \begin{bmatrix} a_1^1 & a_2^1 & \cdots & a_k^i \end{bmatrix}$$

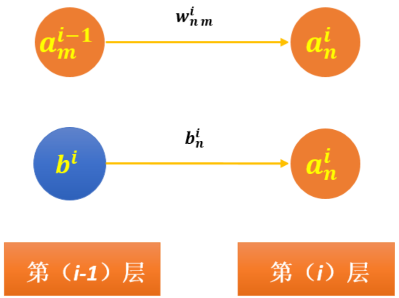
$$= f\left(\begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} * \begin{bmatrix} w_{1\,1}^1 & w_{2\,1}^1 & \cdots & w_{k\,1}^1 \\ w_{1\,2}^1 & w_{2\,2}^1 & \cdots & w_{k\,2}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{1\,n}^1 & w_{2\,n}^1 & \cdots & w_{k\,n}^1 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ \vdots \\ b_k^1 \end{bmatrix} \right)$$

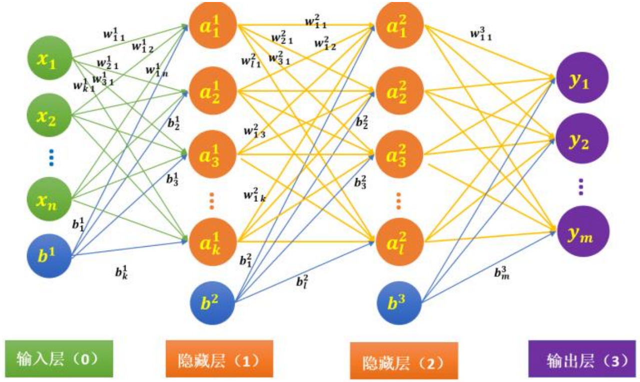
$$= f(X * W^1 + B^1)$$



多层全连接神经网络的计算表达

任意第i层整层的计算：





$$A^i = \begin{bmatrix} a_1^i & a_2^i & \cdots & a_m^i \end{bmatrix}$$

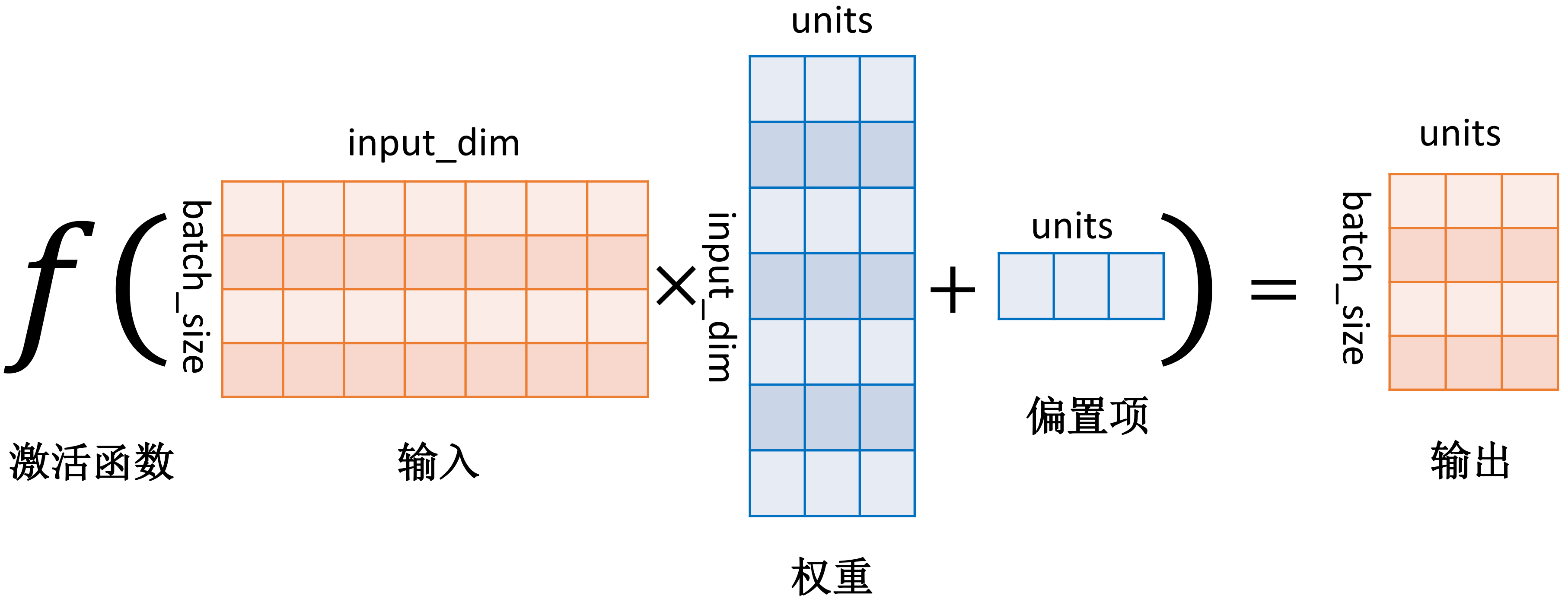
$$= f\left(\begin{bmatrix} a_1^{i-1} & a_2^{i-1} & \cdots & a_k^{i-1} \end{bmatrix} * \begin{bmatrix} w_{1\,1}^{1} & w_{2\,1}^{1} & \cdots & w_{m\,1}^{i} \\ w_{1\,2}^{1} & w_{2\,2}^{1} & \cdots & w_{m\,2}^{i} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1\,k}^{i} & w_{2\,k}^{i} & \cdots & w_{m\,k}^{i} \end{bmatrix} + \begin{bmatrix} b_1^i \\ b_2^i \\ \vdots \\ b_k^i \end{bmatrix}\right)$$

$$= f\left(A^{i-1} * W^i + B^i\right)$$

第i层有m个节点

第i-1层有k个节点

网络层的计算示意图



units
input_dim
f
batch_size
input_dim
units
batch_size
units
激活函数
输入
偏置项
输出
权重

全连接单隐藏层网络建模实现

载入数据

载入数据

```
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 = 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

```
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
```

数据塑形

```
# 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)
```

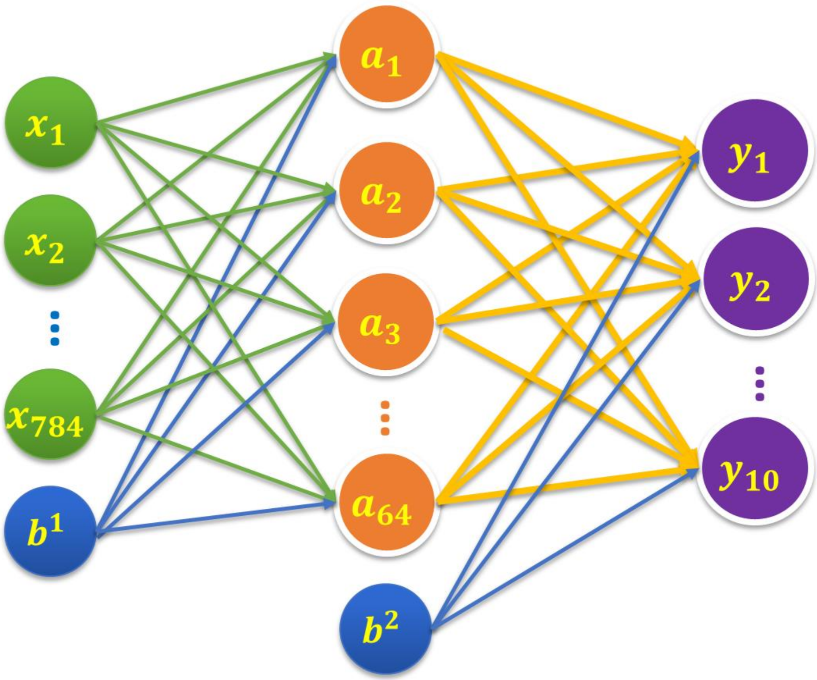
特征数据归一化

```
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)
```

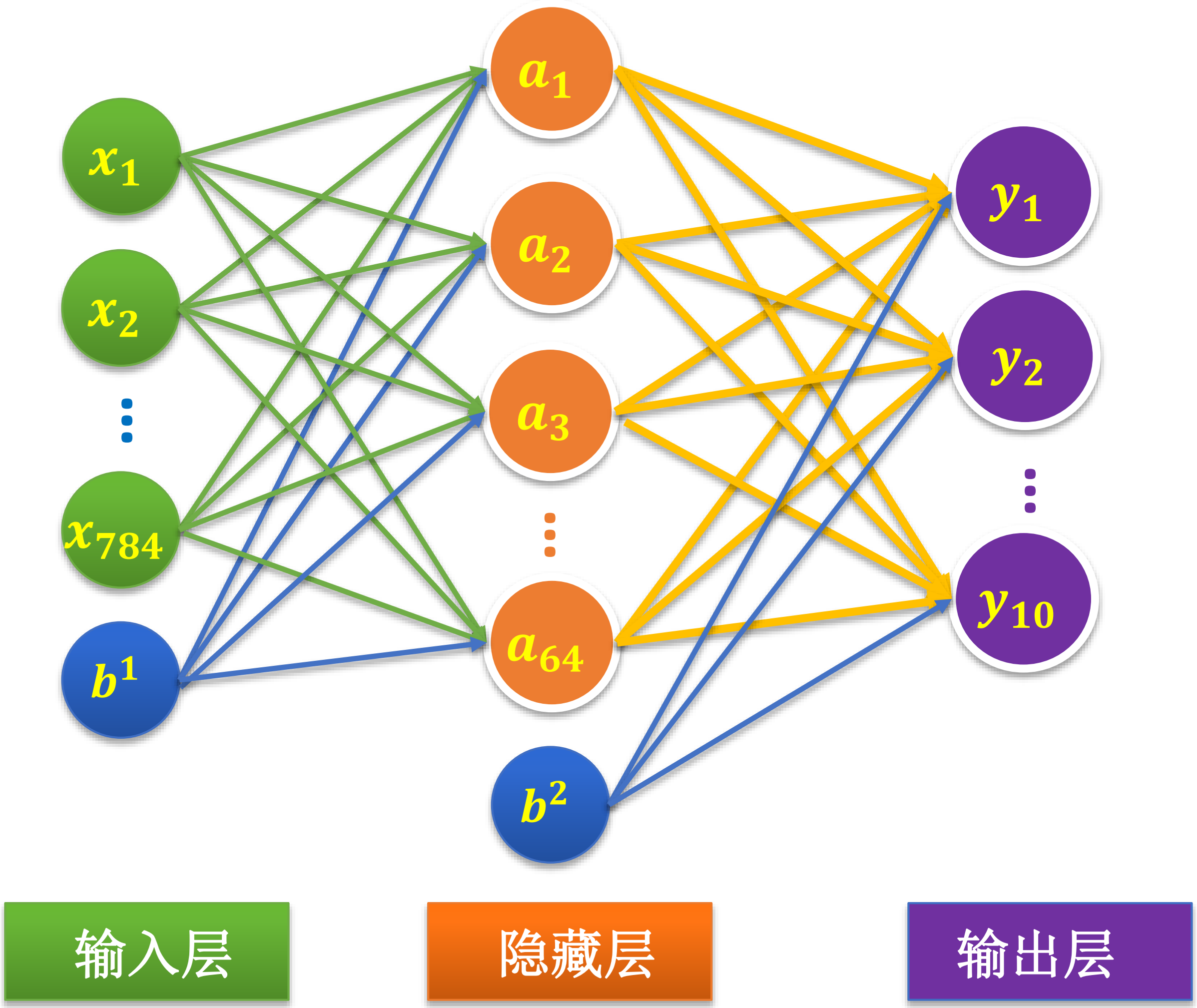
标签数据独热编码

```
# 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)
```

构建模型



Zhejiang University City College logo and wordmark



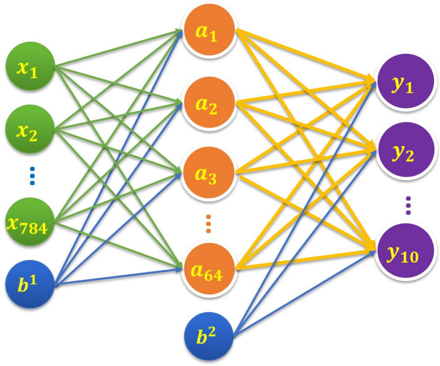
全连接单隐含层神经网络

创建待优化变量

```python
# 定义第一层隐藏层权重和偏置项变量
Input_Dim = 784
H1_NN = 64
W1 = tf.Variable(tf.random.normal([Input_Dim, H1_NN],mean=0.0, stddev=1.0, dtype=tf.float32))
B1 = tf.Variable(tf.zeros([H1_NN]),dtype = tf.float32)
```

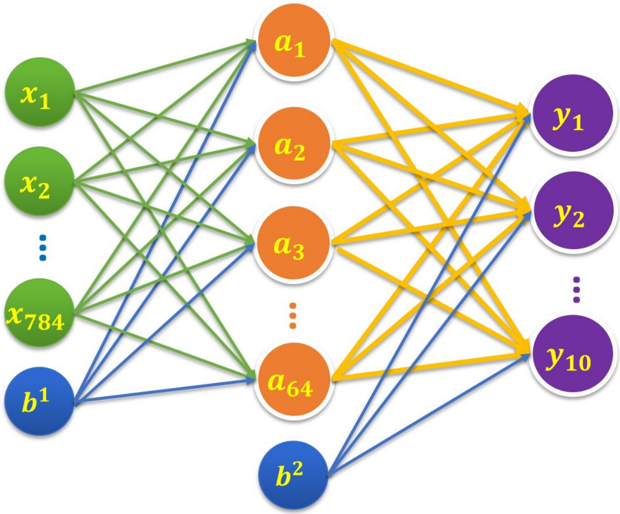
```python
# 定义输出层权重和偏置项变量
Output_Dim = 10
W2 = tf.Variable(tf.random.normal([H1_NN, Output_Dim],mean=0.0, stddev=1.0, dtype=tf.float32))
B2 = tf.Variable(tf.zeros([Output_Dim]),dtype = tf.float32)
```

```python
# 建立待优化变量列表
W = [W1, W2]
B = [B1, B2]
```



定义模型前向计算

```
def model(x, w, b):
    x = tf.matmul(x, w[0]) + b[0]
    x = tf.nn.relu(x)
    x = tf.matmul(x, w[1]) + b[1]
    pred = tf.nn.softmax(x)
    return pred
```



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。

设置训练超参数

```
training_epochs = 20 # 训练轮数
batch_size = 50   # 单次训练样本数（批次大小）
learning_rate= 0.01   # 学习率
```

定义梯度计算函数

```python
# 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)
```

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

```
steps= 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(steps):
        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
```

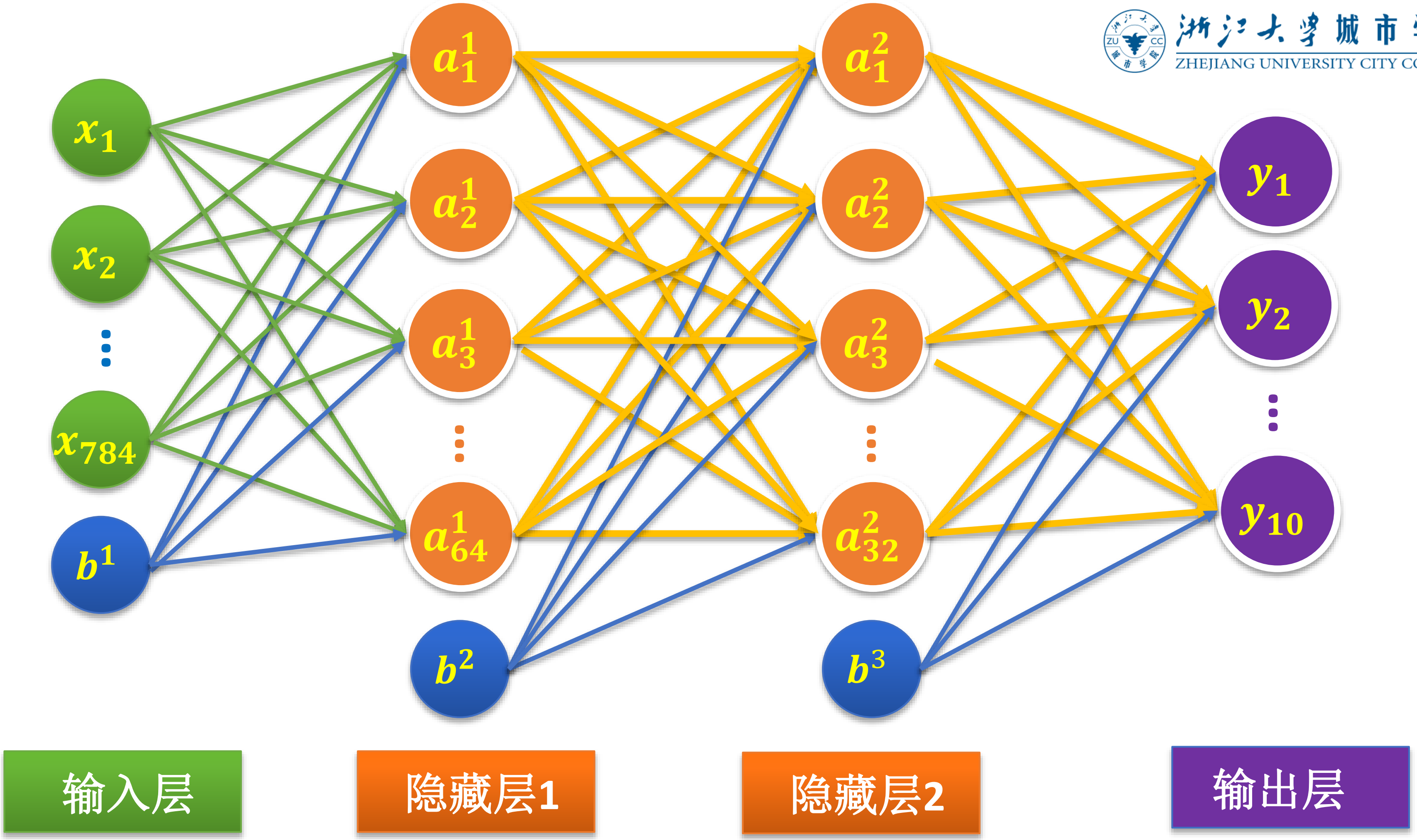
训练模型

```
epoch=  1, train_loss=8.3710, train_acc=0.4753, val_loss=8.4180, val_acc=0.4723
epoch=  2, train_loss=6.9728, train_acc=0.5606, val_loss=6.9890, val_acc=0.5596
epoch=  3, train_loss=3.8019, train_acc=0.7561, val_loss=3.7573, val_acc=0.7590
epoch=  4, train_loss=3.6183, train_acc=0.7684, val_loss=3.5944, val_acc=0.7697
epoch=  5, train_loss=2.2311, train_acc=0.8498, val_loss=2.1868, val_acc=0.8524
epoch=  6, train_loss=1.9697, train_acc=0.8709, val_loss=1.9611, val_acc=0.8702
epoch=  7, train_loss=1.9733, train_acc=0.8704, val_loss=1.9752, val_acc=0.8685
epoch=  8, train_loss=1.9700, train_acc=0.8699, val_loss=2.0001, val_acc=0.8655
epoch=  9, train_loss=0.6384, train_acc=0.9499, val_loss=0.7351, val_acc=0.9410
epoch= 10, train_loss=0.5541, train_acc=0.9575, val_loss=0.6890, val_acc=0.9459
epoch= 11, train_loss=0.4643, train_acc=0.9637, val_loss=0.5572, val_acc=0.9558
epoch= 12, train_loss=0.4470, train_acc=0.9650, val_loss=0.5746, val_acc=0.9537
epoch= 13, train_loss=0.3801, train_acc=0.9698, val_loss=0.5190, val_acc=0.9605
epoch= 14, train_loss=0.3585, train_acc=0.9715, val_loss=0.5445, val_acc=0.9572
epoch= 15, train_loss=0.3796, train_acc=0.9691, val_loss=0.5452, val_acc=0.9548
epoch= 16, train_loss=0.4196, train_acc=0.9648, val_loss=0.5963, val_acc=0.9506
epoch= 17, train_loss=0.3472, train_acc=0.9712, val_loss=0.5177, val_acc=0.9596
epoch= 18, train_loss=0.3295, train_acc=0.9726, val_loss=0.5325, val_acc=0.9578
epoch= 19, train_loss=0.3210, train_acc=0.9744, val_loss=0.5420, val_acc=0.9592
epoch= 20, train_loss=0.3396, train_acc=0.9728, val_loss=0.5407, val_acc=0.9576
```

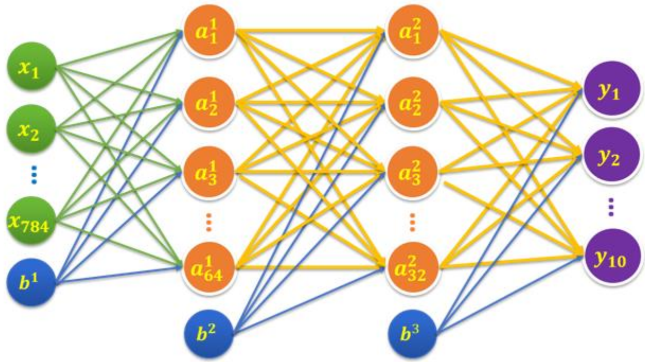
从上述打印结果可以看出损失值** Loss **是趋于更小的，同时，准确率** Accuracy **越来越高。

更多层网络模型实现

浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE



需要改变的地方



创建变量

```
# 定义第1层隐藏层权重和偏置项变量
Input_Dim = 784
H1_NN = 64
W1 = tf.Variable(tf.random.normal([Input_Dim, H1_NN],mean=0.0, stddev=1.0, dtype=tf.float32))
B1 = tf.Variable(tf.zeros([H1_NN]),dtype = tf.float32)
```

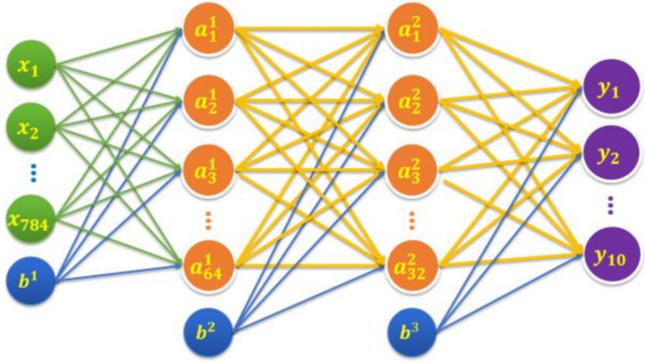
```
# 定义第2层隐藏层权重和偏置项变量
H2_NN = 32
W2 = tf.Variable(tf.random.normal([H1_NN, H2_NN],mean=0.0, stddev=1.0, dtype=tf.float32))
B2 = tf.Variable(tf.zeros([H2_NN]),dtype = tf.float32)
```

```
# 定义输出层权重和偏置项变量
Output_Dim = 10
W3 = tf.Variable(tf.random.normal([H2_NN, Output_Dim],mean=0.0, stddev=1.0, dtype=tf.float32))
B3 = tf.Variable(tf.zeros([Output_Dim]),dtype = tf.float32)
```

```
# 建立待优化变量列表
W = [W1, W2, W3]
B = [B1, B2, B3]
```

定义模型前向计算

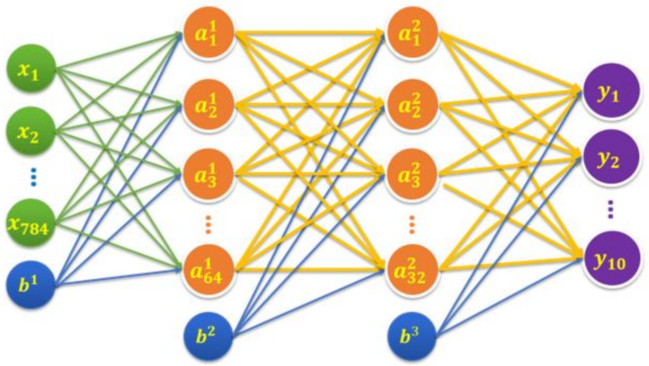
```python
def model(x, w, b):
    x = tf.matmul(x, w[0]) + b[0]
    x = tf.nn.relu(x)
    x = tf.matmul(x, w[1]) + b[1]
    x = tf.nn.relu(x)
    x = tf.matmul(x, w[2]) + b[2]
    pred = tf.nn.softmax(x)
    return pred
```



超参数调整

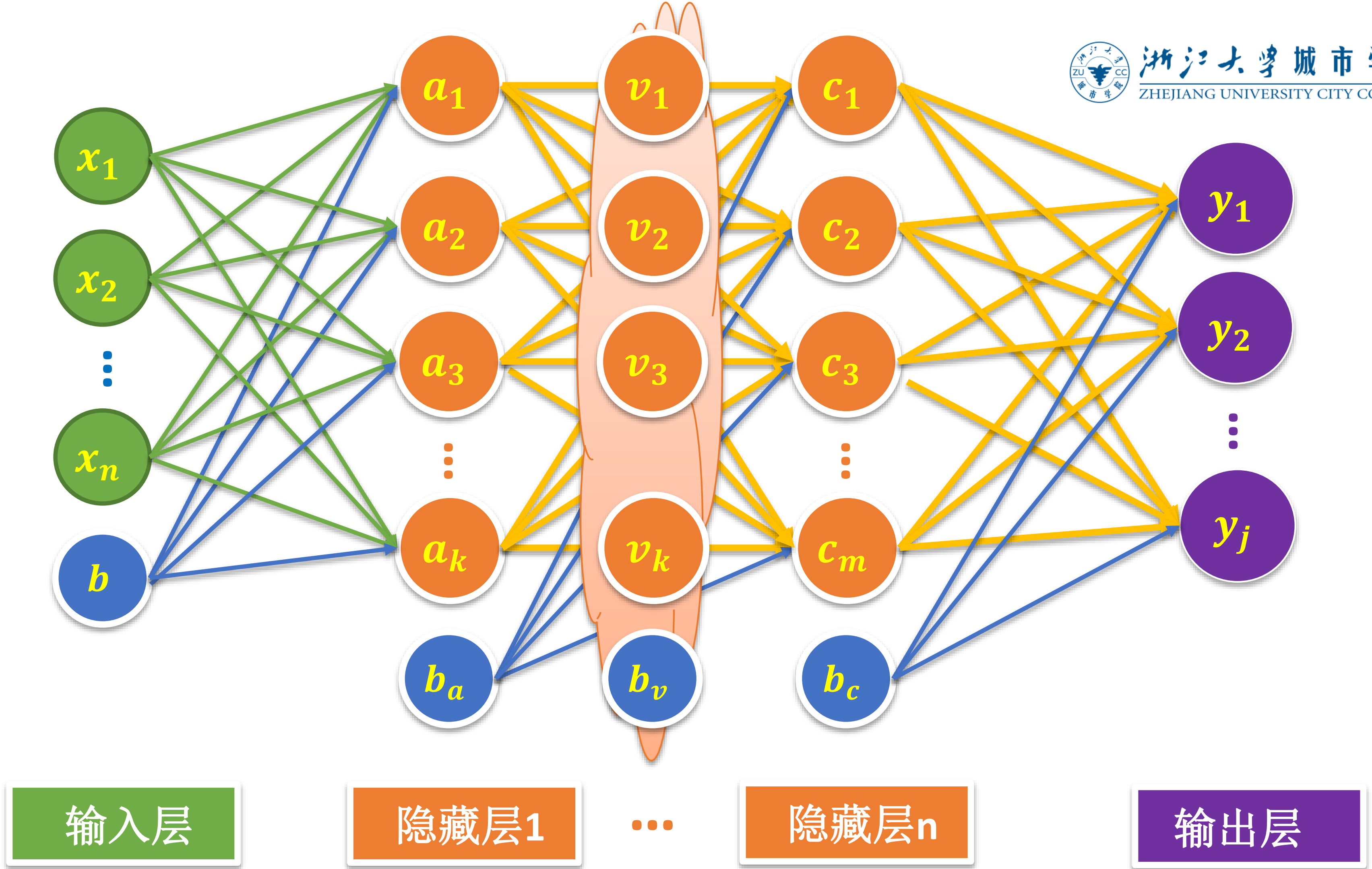
模型也未必是越复杂效果越好，还需要配合超参数的调整。

请同学们试一试学习率设为0.01和0.001的训练结果区别。



再多一点，多层网络建模实现

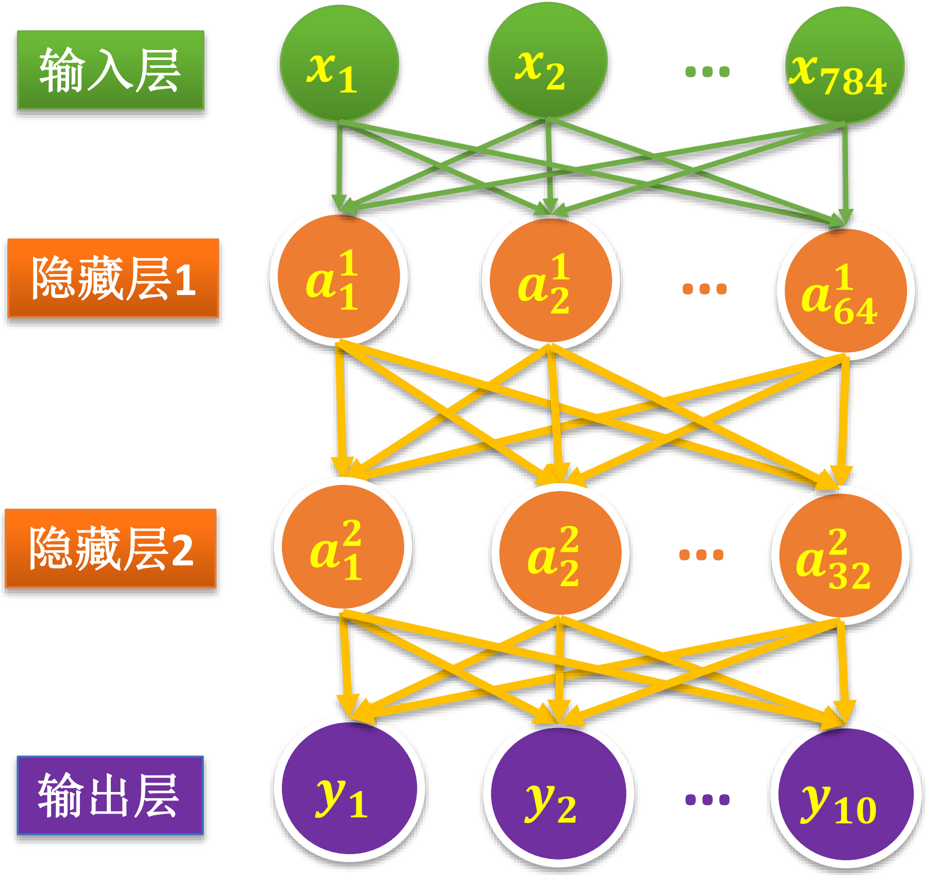
什么，还要再多一点？



x_1
x_2
x_n
b
a_1
a_2
a_3
a_k
b_a
v_1
v_2
v_3
v_k
b_v
c_1
c_2
c_3
c_m
b_c
y_1
y_2
y_j
输入层
隐藏层1
…
隐藏层n
输出层

使用Keras序列模型建模

Keras序列模型建模



Keras序列模型建模的一般步骤

采用Keras序列模型进行建模与训练过程一般分为六个步骤：

（1）创建一个Sequential模型；

（2）根据需要，通过“add()”方法在模型中添加所需要的神经网络层，完成模型构建；

（3）编译模型，通过“compile()”定义模型的训练模式；

（4）训练模型，通过“fit()”方法进行训练模型；

（5）评估模型，通过“evaluate()”进行模型评估；

（6）应用模型，通过“predict()”进行模型预测。

载入数据

载入数据

```python
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

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

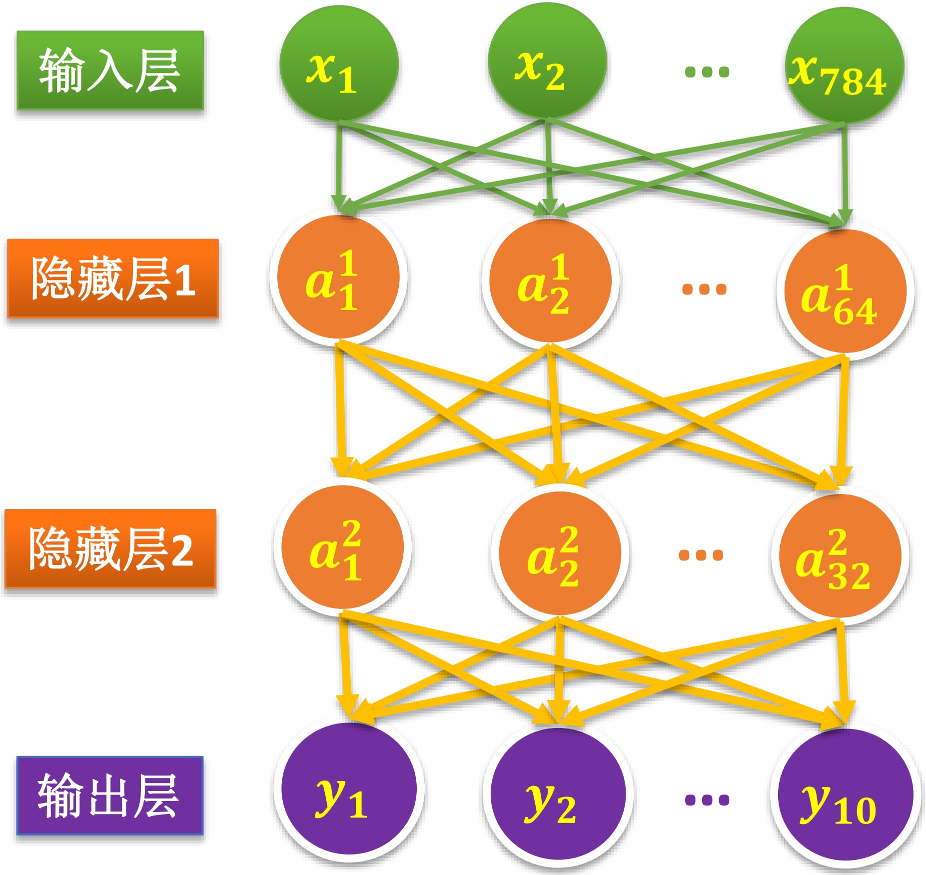
特征数据归一化

```python
# 对图像images进行数字标准化
train_images = train_images / 255.0
test_images = test_images / 255.0
```

标签数据独热编码

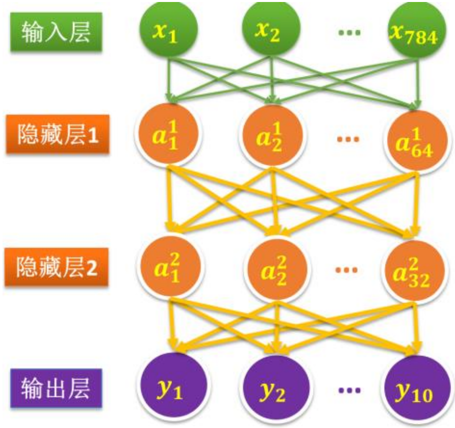
```
# 对标签labels进行One-Hot Encoding
train_labels_ohe = tf.one_hot(train_labels, depth = 10).numpy()
test_labels_ohe = tf.one_hot(test_labels, depth = 10).numpy()
```

目标模型



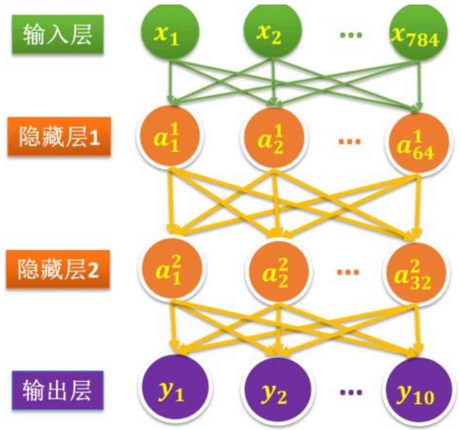
新建一个序列模型

```
# 建立Sequential线性堆叠模型
model = tf.keras.models.Sequential()
```



添加输入层 （平坦层，Flatten）

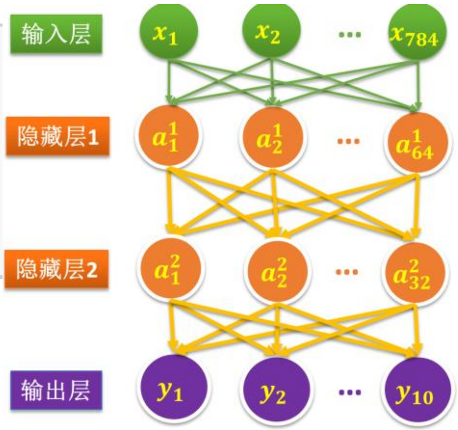
```
# 添加平坦层
model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
```



添加隐藏层（密集层，Dense）

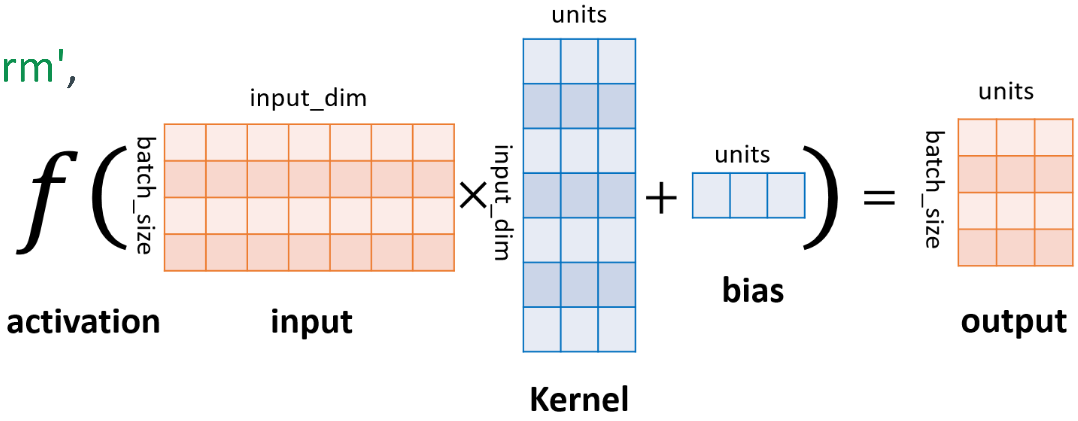
```
# 添加全连接层1
model.add(tf.keras.layers.Dense(units = 64,
                                #   input_dim = 784,    # 输入的shape或者维度都可以不填
                                kernel_initializer = 'normal',
                                activation = 'relu'))
```

```
# 添加全连接层2
model.add(tf.keras.layers.Dense(units = 32,
                                #       input_dim = 256,
                                kernel_initializer = 'normal',
                                activation = 'relu'))
```



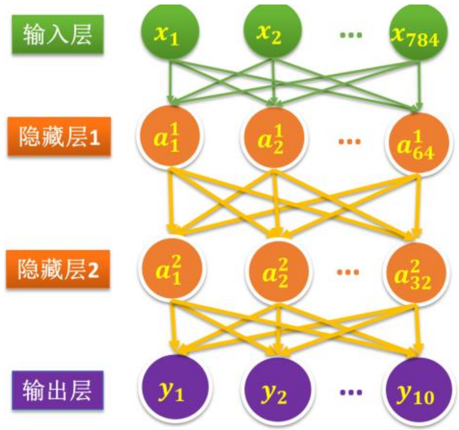
Kease的密集层

```
tf.keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```



添加输出层（还是密集层）

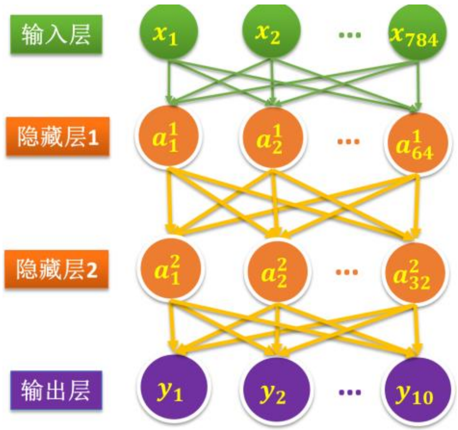
```python
# 添加输出层
model.add(tf.keras.layers.Dense(10, activation = 'softmax'))
```



模型摘要

```
# 输出模型摘要
model.summary()
```

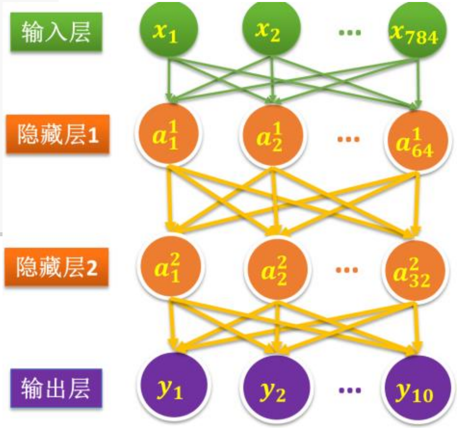
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 52,650
Trainable params: 52,650
Non-trainable params: 0
```



一次性建模

以上建模也可以一次性完成

```
# 一次性建立Sequential线性堆叠模型
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(64,activation=tf.nn.relu),
    tf.keras.layers.Dense(32,activation=tf.nn.relu),
    tf.keras.layers.Dense(10,activation=tf.nn.softmax)
])
```



定义训练模式

```
# 定义训练模式
model.compile(optimizer = 'adam', # 优化器
              loss = 'categorical_crossentropy', # 损失函数
              metrics = ['accuracy']) # 评估模型的方式
```

tf.keras.Model.compile 接受 3 个重要的参数：

optimizer ：优化器，可从 tf.keras.optimizers 中选择；

loss ：损失函数，可从 tf.keras.losses 中选择；

metrics ：评估指标，可从 tf.keras.metrics 中选择。

设置训练参数

```python
# 设置训练参数
train_epochs = 10 # 训练轮数
batch_size = 30   # 单次训练样本数（批次大小）
```

模型训练

```
# 训练模型
train_history=model.fit(train_images, train_labels_ohe,
                        validation_split = 0.2,
                        epochs = train_epochs,
                        batch_size = batch_size,
                        verbose = 2)
```

tf.keras.Model.fit()常见参数：

x ： 训练数据；
y ： 目标数据（数据标签）；
epochs ： 将训练数据迭代多少遍；
batch_size ： 批次的大小；
validation_data ： 验证数据，可用于在训练过程中监控模型的性能。
verbose：训练过程的日志信息显示，0为不在标准输出流输出日志信息，1为输出进度条记录，2为每个epoch输出一行记录。

模型训练

```
# 训练模型
train_history=model.fit(train_images, train_labels_ohe,
                        validation_split = 0.2,
                        epochs = train_epochs,
                        batch_size = batch_size,
                        verbose = 2)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 - 5s - loss: 0.3584 - accuracy: 0.8971 - val_loss: 0.1930 - val_accuracy: 0.9465
Epoch 2/10
48000/48000 - 2s - loss: 0.1610 - accuracy: 0.9519 - val_loss: 0.1481 - val_accuracy: 0.9574
Epoch 3/10
48000/48000 - 2s - loss: 0.1154 - accuracy: 0.9653 - val_loss: 0.1168 - val_accuracy: 0.9668
Epoch 4/10
48000/48000 - 2s - loss: 0.0894 - accuracy: 0.9731 - val_loss: 0.1108 - val_accuracy: 0.9681
Epoch 5/10
48000/48000 - 2s - loss: 0.0732 - accuracy: 0.9779 - val_loss: 0.1066 - val_accuracy: 0.9702
Epoch 6/10
48000/48000 - 3s - loss: 0.0627 - accuracy: 0.9807 - val_loss: 0.1095 - val_accuracy: 0.9696
Epoch 7/10
48000/48000 - 3s - loss: 0.0517 - accuracy: 0.9837 - val_loss: 0.1201 - val_accuracy: 0.9665
Epoch 8/10
48000/48000 - 3s - loss: 0.0428 - accuracy: 0.9864 - val_loss: 0.1147 - val_accuracy: 0.9697
Epoch 9/10
48000/48000 - 2s - loss: 0.0379 - accuracy: 0.9881 - val_loss: 0.1221 - val_accuracy: 0.9678
Epoch 10/10
48000/48000 - 3s - loss: 0.0324 - accuracy: 0.9894 - val_loss: 0.1107 - val_accuracy: 0.9726
```

训练过程指标数据

In [16]:
```
train_history.history
```

Out[16]:
```
{'loss': [0.3583986661117414,
  0.16095229309779824,
  0.1153935849564732,
  0.08942422725362122,
  0.07315575623746554,
  0.06270683312141045,
  0.05168723869837777,
  0.04284361907812126,
  0.03789665313222030,
  0.0324425651088040],
 'accuracy': [0.89710414,
  0.9519375,
  0.9652917,
  0.97314584,
  0.9779375,
  0.98070836,
  0.98366666,
  0.986375,
  0.9880625,
  0.98941666],
 'val_loss': [0.192979598979,
  0.14810317863477393,
```

history是一个字典类型数据，包含了4个Key：loss、accuracy、val_loss和val_accuracy，分别表示训练集上的损失、准确率和验证集上的损失和准确率。

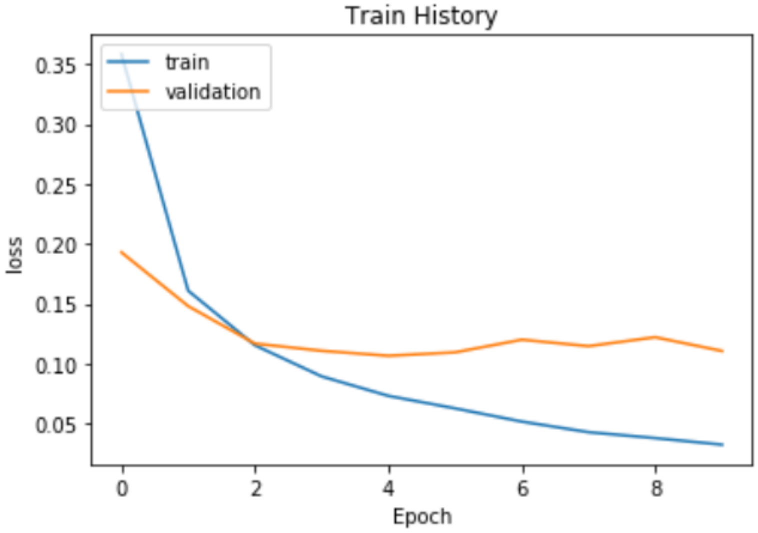
它们的值都是一个列表，记录了每个周期该指标的具体数值。

训练过程指标可视化

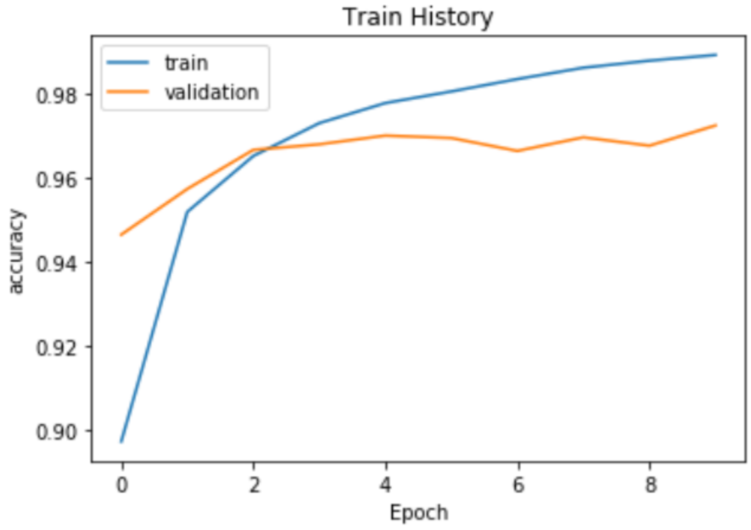

```python
import matplotlib.pyplot as plt
def show_train_history(train_history,train_metric,val_metric):
    plt.plot(train_history.history[train_metric])
    plt.plot(train_history.history[val_metric])
    plt.title('Train History')
    plt.ylabel(train_metric)
    plt.xlabel('Epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
```

训练过程指标可视化

```
show_train_history(train_history,'loss','val_loss')
```




```
show_train_history(train_history,'accuracy','val_accuracy')
```



评估模型

```
# 评估模型
test_loss, test_acc = model.evaluate(test_images, test_labels_ohe, verbose = 2)
```

```
10000/1 - 0s - loss: 0.0502 - accuracy: 0.9735
```

模型的度量指标

```
yy= model.evaluate(test_images, test_labels_ohe, verbose = 2)
```

```
10000/1 - 0s - loss: 0.0502 - accuracy: 0.9735
```

```
yy
```

```
]: [0.10032350613603194, 0.9735]
```

```
model.metrics_names
```

```
]: ['loss', 'accuracy']
```

模型评估evaluate()的返回值是一个损失值的标量（如果没有指定其他度量指标），或者是一个列表（如果指定了其他度量指标）。

应用模型

```python
# 进行预测
test_pred = model.predict(test_images)
```

```python
test_pred.shape
```

]: (10000, 10)

```python
# 预测值
np.argmax(test_pred[0])
```

]: 7

应用模型

```
# 直接进行分类预测
test_pred = model.predict_classes(test_images)
```

```
test_pred[0]
```

2]: 7

```
#标签值
test_labels[0]
```

3]: 7

面向整数标签的Keras序列模型构建与训练

面向整数标签的序列模型构建与训练

针对采用整数类型的标签类别数据，Keras提供了更为简便的方法，无需针对这些标签数据先进行独热编码就能直接应用

采用“sparse_categorical_crossentropy”损失函数来替换“categorical_crossentropy”损失函数

```
loss = tf.keras.losses.sparse_categorical_crossentropy(y_true=y, y_pred=y_pred)
```

作用相同

```
loss = tf.keras.losses.categorical_crossentropy(
    y_true=tf.one_hot(y, depth=tf.shape(y_pred)[-1]),
    y_pred=y_pred
)
```

结束