神经网络
——原理与实现

- 神经网络原理概述
- 神经网络的搭建与训练
- 基于Keras实现神经网络

神经网络原理概述

机器学习与深度学习

人工智能：能够感知、推理、行动和适应的程序，将通常由人类完成的智力任务自动化。

机器学习：通过学习如何组合输入信息来建立模型，进而对未见过的数据做出有用的预测。

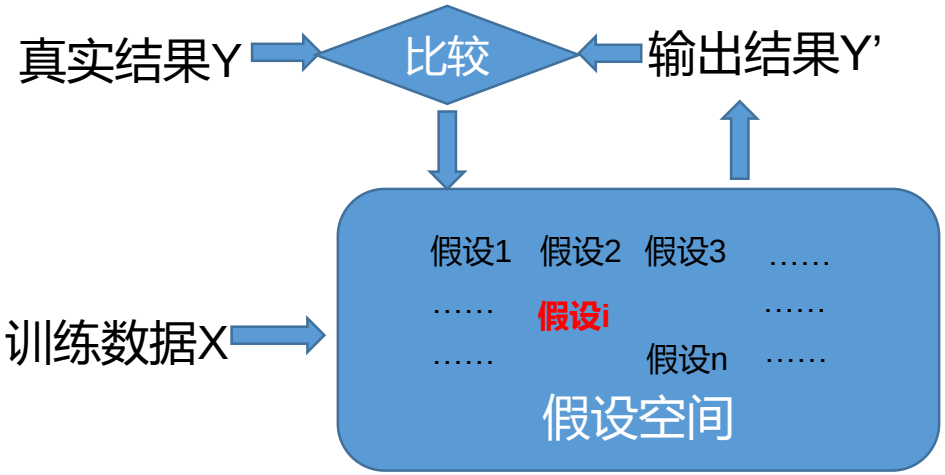
深度学习：是机器学习的一个分支领域。相对于仅仅学习一、两层数据表示的浅层算法而言，强调通过连续的多层（layer）对数据表示进行学习。典型算法是深度神经网络。

机器学习的核心问题

机器学习的核心问题在于有意义地变换数据。

一个输入空间到输出空间的映射变换的集合，就形成了一个假设空间。它由问题类型、模型选择及配置确定。

例如：线性回归的假设空间可表示为：$h_\theta(x)=\sum_{i=0}^{n}\theta_i x_i=\theta^T x$



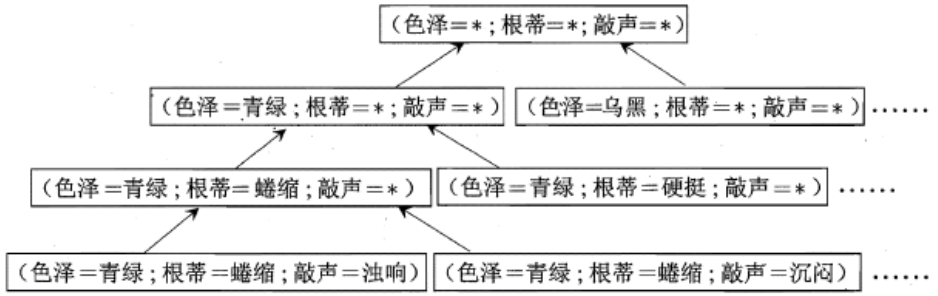
学习过程就是在所有假设组成的空间中根据反馈进行搜索，找到与训练集数据集“匹配”的假设，即得到训练模型。

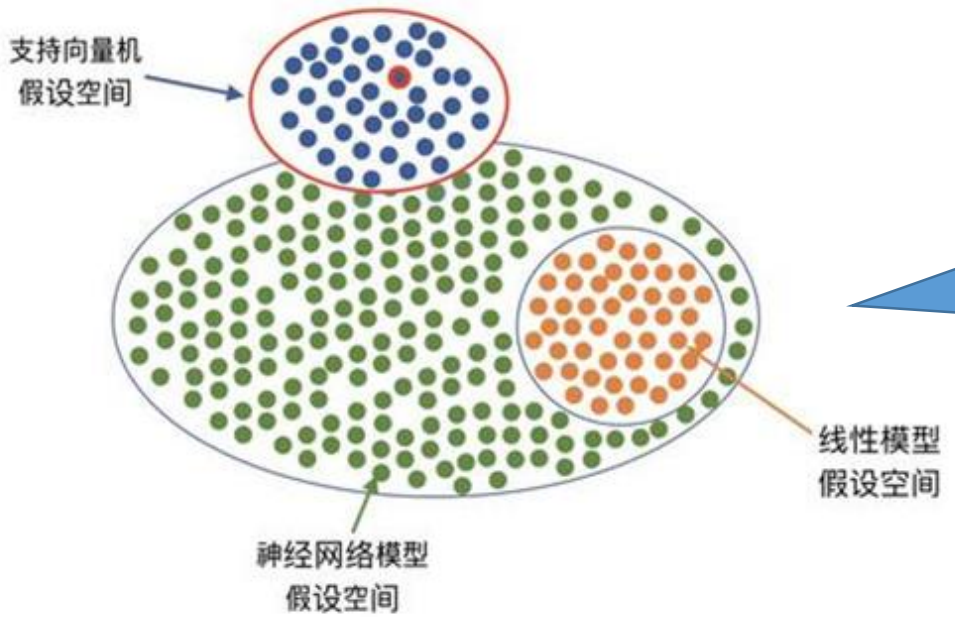
假设的表示一旦确定，假设空间及其规模大小就确定了。

逻辑回归的假设空间

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T \mathbf{x}}}$$

决策树的假设空间(西瓜分类：如果取1-3个特征共有65种组合)

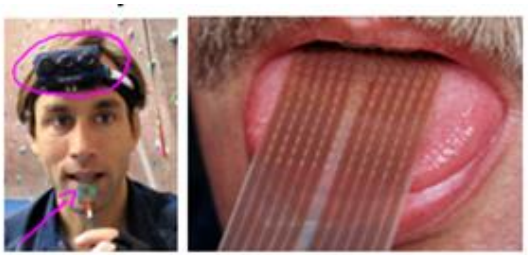




神经网络的容量大，假设空间的表达能力更强，所以可对复杂问题建模。

人脑工作原理猜想

神经重接实验及各种人类行为引起科学家猜想：大脑可以用一种算法处理各种不同问题，只要接入传感信息，大脑就能学会处理它？！



Seeing with your tongue

Human echolocation (sonar)

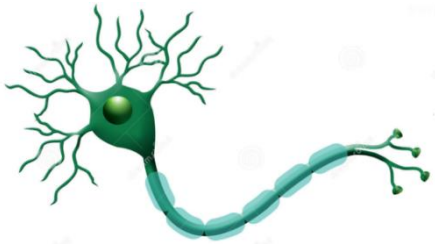
Haptic belt: Direction sense

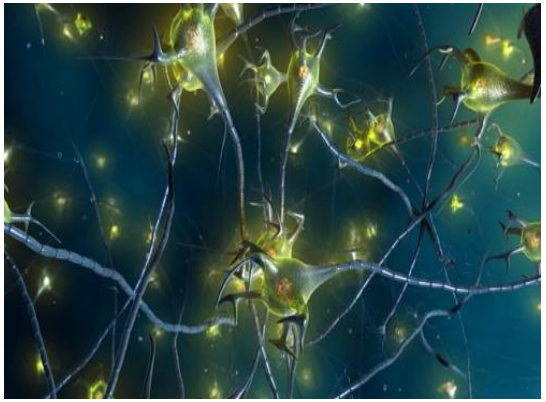
Implanting a 3rd eye

神经网络是人们为了模仿大脑产生的一种算法。

人脑及神经元

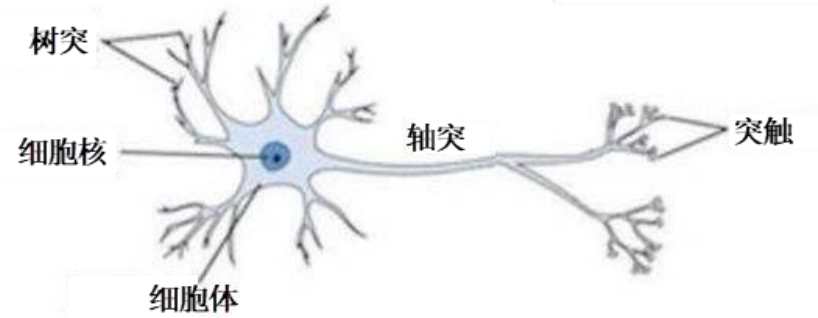
人脑包含800亿个神经元，这些神经元类似一个个小的处理单元，它们按照某种方式连接， 接受外部刺激，做出响应处理。





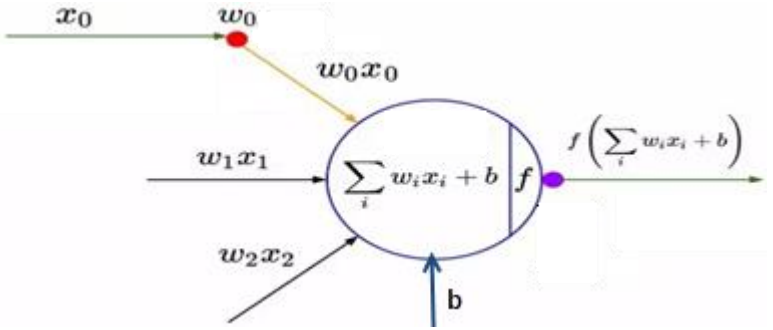
神经元(生物模型)

- **树突**：神经元的输入通道，接收其他神经元电信号
- **细胞体**：处理这些信号，细胞核达到某种“兴奋”状态，就发出信号
- **轴突和突触**：将处理过的信号传递到下一个神经元



人工神经元(数学模型)

- **输入信号**：$x_0, x_1, \ldots x_n$对应树突
- **处理单元**：对应细胞体，对输入信号加权处理$(\omega_0, \omega_1, \ldots \omega_n)$确定其强度；然后求和确定组合效果，再加上截距b；通过激励函数f达到一定的阈值输出
- **输出信号**：$y = f(\sum_{i=0}^{n} \omega_i x_i + b)$对应轴突

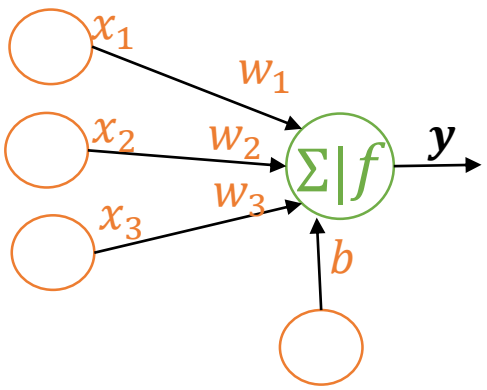


生物神经元的启示

① 神经元是一个**多输入单输出的信号处理单元**

② 神经元具有**阈值特性**

人工神经元的模型描述



$$y = f(\omega_0 x_0 + \omega_1 x_1 \ldots + \omega_n x_n + b)$$

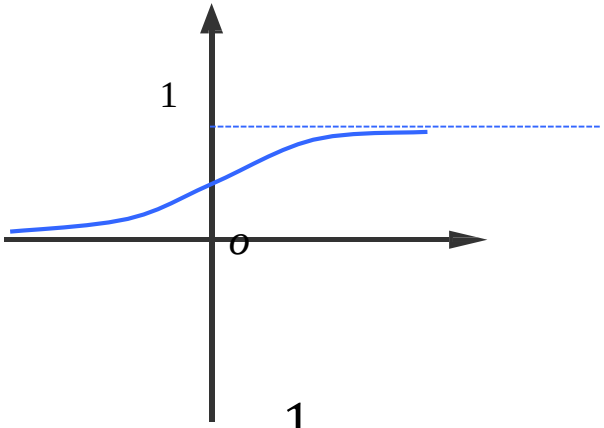
其中：

$x_0, x_1, \ldots x_n$：输入信号

$\omega_0, \omega_1, \ldots \omega_n$：各输入信号的权重

b：是阈值，也称偏置项或截距

$f(z)$：激活函数，是非线性函数



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

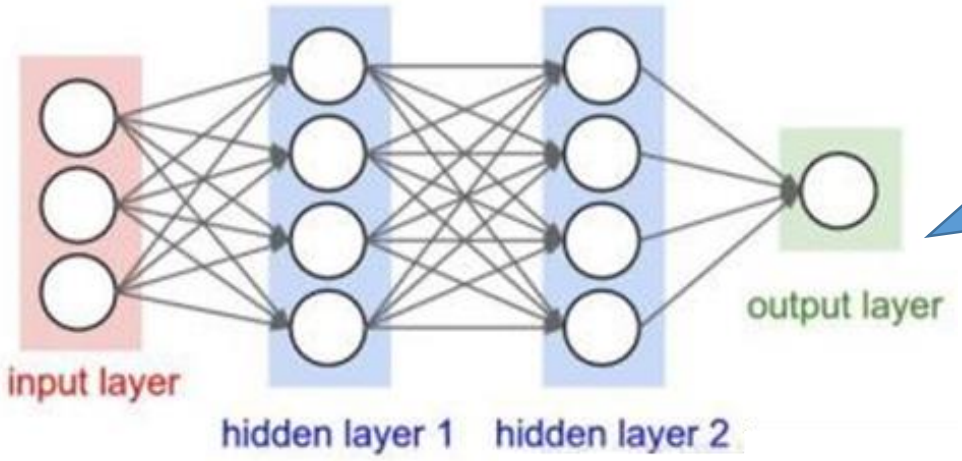
Sigmoid函数

当激活函数选Sigmoid函数时，单个神经元就相当于逻辑回归模型！

$$f_w(x) = \frac{1}{1+e^{-(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b)}}$$

人工神经网络(Artificial Neural Netork,即ANN)

模仿神经元在人脑中的结构连接。将几个甚至几百万个人工神经元排列在一系列的层中，每个层之间彼此相连，就形成人工神经网络。



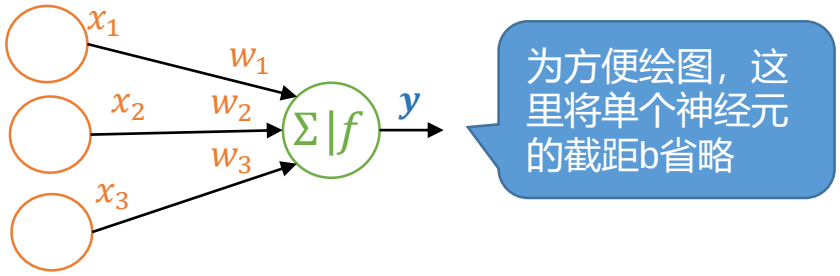
这是一个双隐层神经网络，也称为三层网络(输入层没有处理能力)。共有12个神经元。

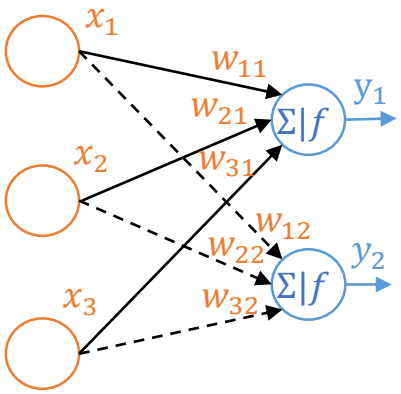
- 一个完整的神经网络由一层输入层、多层隐藏层、一层输出层构成。
- 每层神经元与下一层神经元全互联，同层神经元之间不存在连接。
- 输入层接收外界输入，隐层和输出层神经元对信号进行加工，最终结果由输出层神经元输出。

神经网络中的神经元之间如何连接?

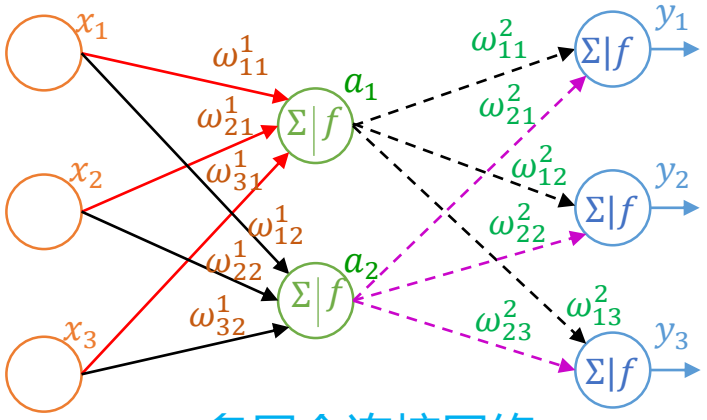
神经元之间是多对多的关系：

- **输入**：来自多个上层的神经元
- **输出**：到多个下层神经元





单层全连接网络



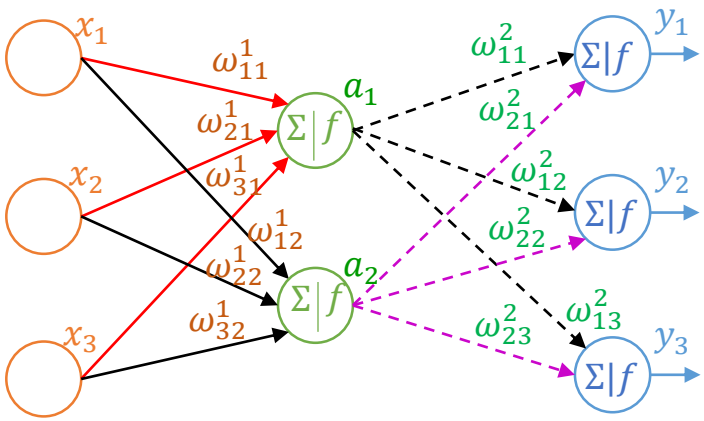
多层全连接网络

也称感知机网络（Perceptron Networks）

- 无隐藏层，只有输入层/输出层
- 可以处理线性问题

- 至少有一个隐藏层
- 可以处理非线性问题

神经网络的数学描述及参数



多层全连接网络

$$a_1 = f(\boldsymbol{w_{11}^1 x_1} + \boldsymbol{w_{21}^1 x_2} + \boldsymbol{w_{31}^1 x_3} + b_{11})$$
$$a_2 = f(\boldsymbol{w_{12}^1 x_1} + \boldsymbol{w_{22}^1 x_2} + \boldsymbol{w_{32}^1 x_3} + b_{12})$$

$$y_1 = f(\boldsymbol{w_{11}^2 a_1} + \boldsymbol{w_{21}^2 a_2} + b_{21})$$
$$y_2 = f(\boldsymbol{w_{12}^2 a_1} + \boldsymbol{w_{22}^2 a_2} + b_{22})$$
$$y_3 = f(\boldsymbol{w_{13}^2 a_1} + \boldsymbol{w_{23}^2 a_2} + b_{23})$$

权重参数w_{mn}^k：k表示第几层，m表示第几个输入，n 表示 第几个神经元，共12个

阈值b_{mn} （也称截距或偏置项）：共有5个

一个网络可以包含几百万个神经元，
多层全连接神经网络参数量巨大!

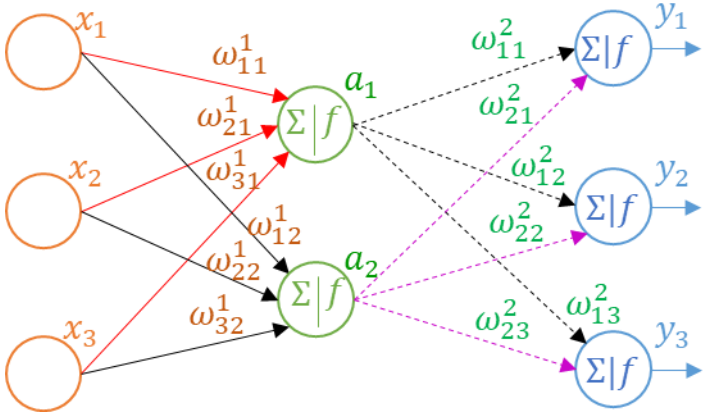
激活函数

- 激活函数的作用

增强网络的表达能力，需要激活函数来将线性函数->非线性函数；否则多层网络连接和单层神经网络本质上就一样了。

- 激活函数的特性

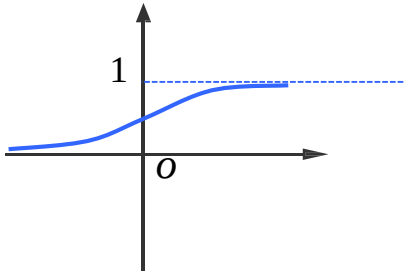
激活函数一定是非线性函数。并且具有连续性、可导性，才能用梯度下降等最优化的方法来求解。



$$a_1 = f(\boldsymbol{w_{11}^1 x_1} + \boldsymbol{w_{21}^1 x_2} + \boldsymbol{w_{31}^1 x_3} + b_{11})$$
$$a_2 = f(\boldsymbol{w_{12}^1 x_1} + \boldsymbol{w_{22}^1 x_2} + \boldsymbol{w_{32}^1 x_3} + b_{12})$$

$$y_1 = f(\boldsymbol{w_{11}^2 a_1} + \boldsymbol{w_{21}^2 a_2} + b_{21})$$
$$y_2 = f(\boldsymbol{w_{12}^2 a_1} + \boldsymbol{w_{22}^2 a_2} + b_{22})$$
$$y_3 = f(\boldsymbol{w_{13}^2 a_1} + \boldsymbol{w_{23}^2 a_2} + b_{23})$$

常用激活函数



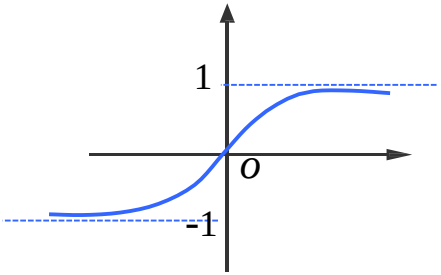
Sigmoid函数

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

$$\acute{\sigma}(z) = \sigma(z)(1-\sigma(z))$$

✓映射到(0,1)区间，二分类常用

✓计算量大，易出现梯度消失



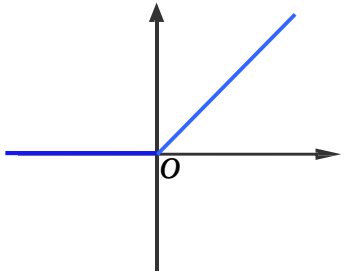
双曲正切(tanh)函数

$$\tanh(z) = 2\sigma(2x) - 1 = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

$$\acute{\tanh}(z) = 1 - \tanh(x)^2$$

✓映射到(-1,1)区间，常用在RNN

✓易出现梯度消失



ReLU函数

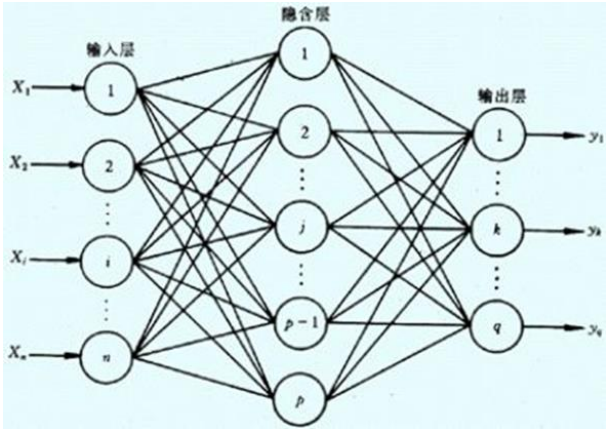
$$relu(x) = \max(0, x)$$

$$\acute{relu}(x) = \begin{cases} x, x > 0 \\ 0, x <= 0 \end{cases}$$

✓效率极高，最常用

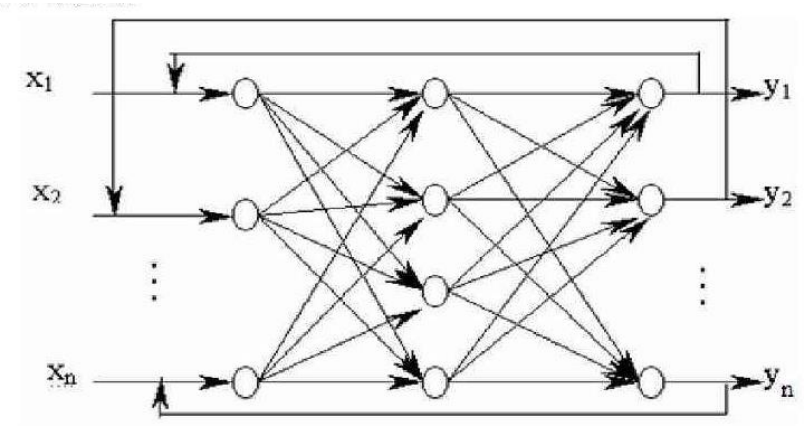
前馈型神经网络
(Feedforward Network)

单向多层结构，即各神经元从输入层开始，只接收上一层的输出并输出到下一层，直至输出层，整个网络拓扑中无反馈回路。常用于图像识别、检测、分割。



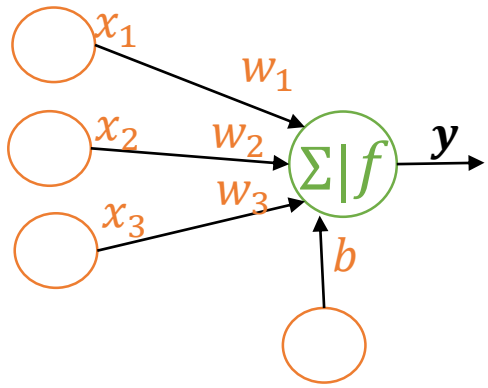
反馈型神经网络
(Recurrent Network)

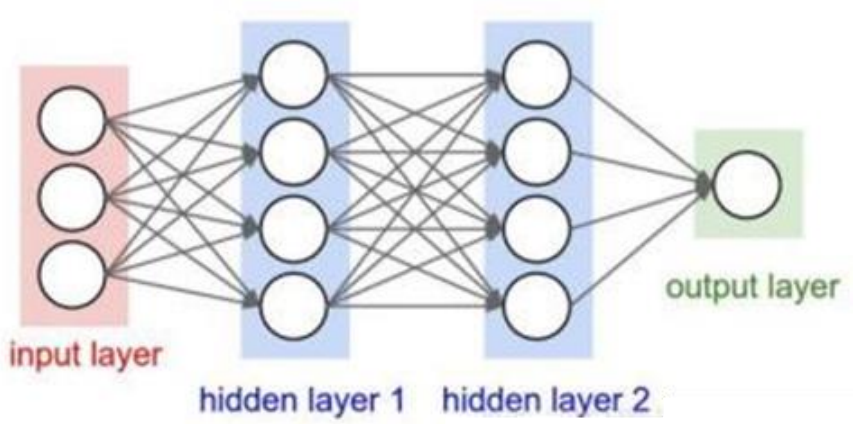
又称自联想记忆网络，是一种从输出到输入具有反馈连接的神经网络，当前的结果受到先前所有的结果的影响，是一种反馈动力学系统。常用于语音、文本处理、问答系统等。



小结

人工神经元排列在一系列的彼此相连接的层中构成神经网络。一个完整的神经网络包含一个输入层、多个隐藏层和一个输出层。

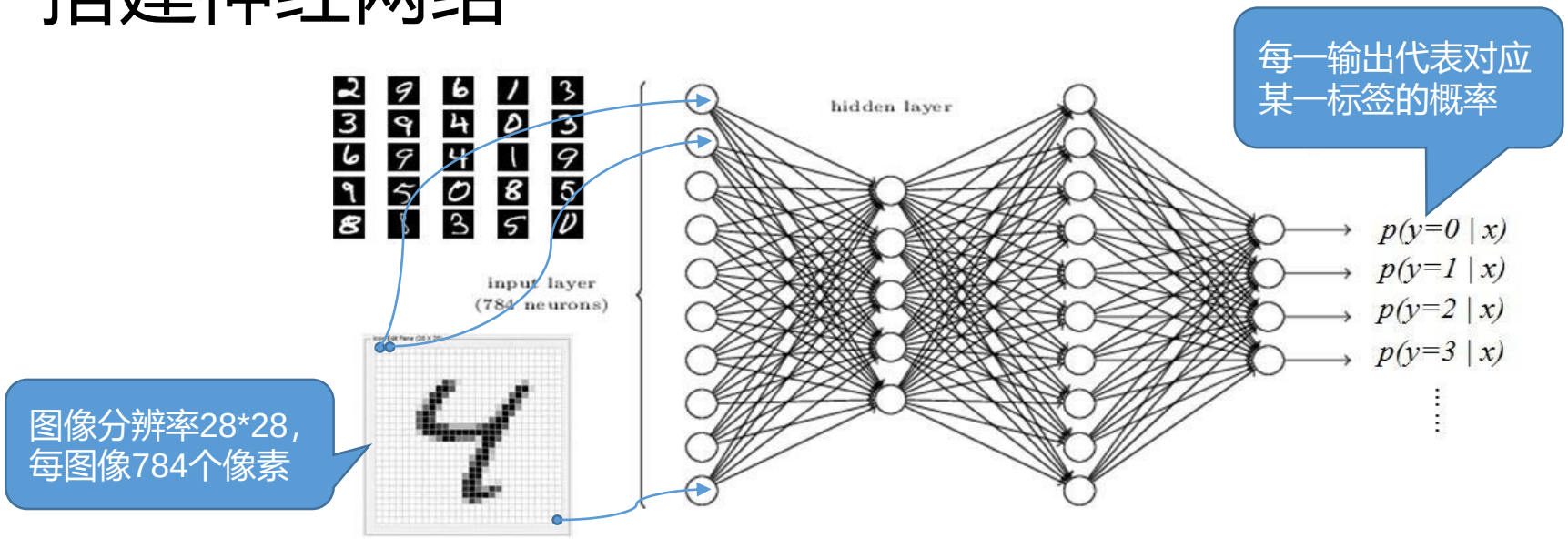




- 神经元的函数描述 $\boldsymbol{y = f(\omega_0 x_0 + \omega_1 x_1 \ldots + \omega_n x_n + b)}$
- 激活函数f非线性、连续、可导。常用Sigmoid、Relu函数。

神经网络的搭建和训练

搭建神经网络



假定要搭建一个神经网络实现手写数字的识别。需定义网络结构：

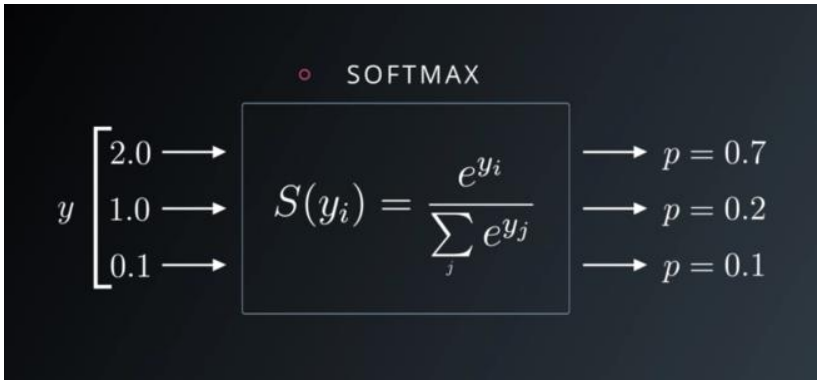
- **输入层神经元个数：**784（每个神经元对应一个像素值）。
- **隐藏层层数**及**每层神经元个数：**根据经验和直觉定义，然后根据欠拟合、过拟合等表现调整。例如隐层1选128，隐层2选256。
- **输出层神经元个数：**分类数，本例选10，因为有10个类别的数字
- **激活函数：**输入层和隐藏层选择Relu函数或Sigmoid函数，输出层因为是多分类，选择softmax函数。

Softmax函数

- softmax函数，又称**归一化指数函数**，是二分类函数sigmoid在多分类上的推广，目的是将多分类的结果以概率的形式展现出来。

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$ y_i表示在第i个分类上的输出结果

概率分布：$1 > S(y_i) > 0,\ \sum_i S(y_i) = 1$



- 分类方式：得分最高的那一类即为输入对应的预测类别。

训练神经网络——参数如何定义？

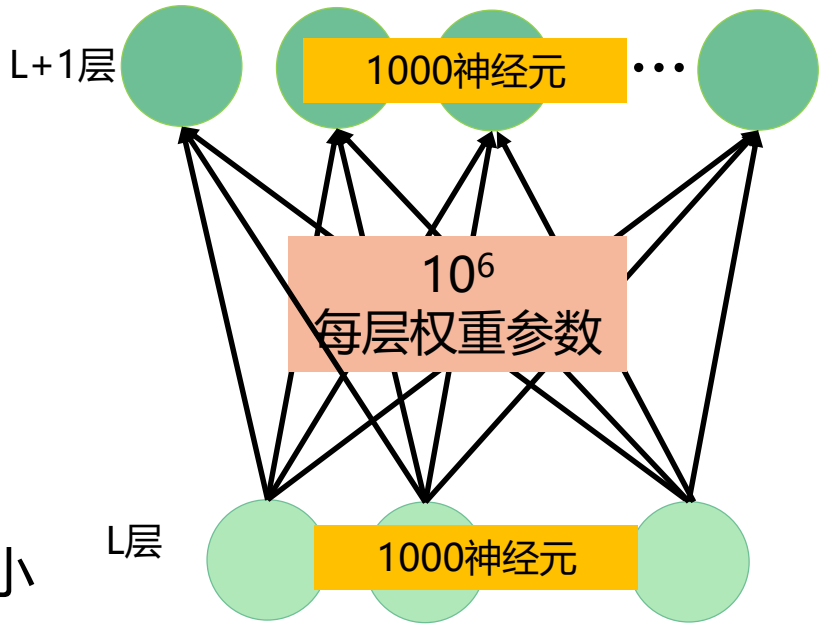
搭建好网络模型，初始的权重和偏置是随机设定的，但不一定是最好的模型，希望模型能够根据数据自己学习。

✗ 枚举所有可能的取值

网络参数$w = \{$权重参数w_{mn}^{k}，阈值参数$b_{mn}\}$，
其中：k表示第几层，m表示第几个输入，n 表示第几个神经元

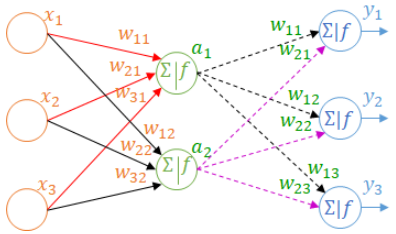
参数个数巨大

✓ 寻找模型最佳参数使损失函数值最小



例如：语音识别模型有 8层，
每层1000神经元

训练神经网络——损失函数定义



损失函数用来估量模型预测值与真实值之间的差距。损失函数给出的差距越小，则模型鲁棒性就越好。**损失函数的设计要根据具体任务的特点。**

◆ 单个样本的损失函数：

✓ **平方损失函数** 衡量两个距离的差异，常用于回归任务

$$Loss(y,\hat{y}) = (y-\hat{y})^2, \qquad \hat{y} = f(x,w)$$

✓ **交叉熵损失函数** 衡量两个概率分布的差异，常用于分类任务

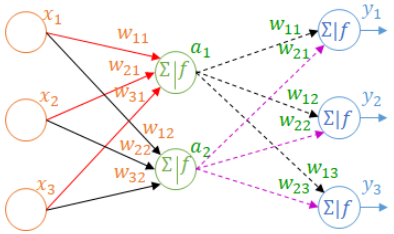
$$Loss(y,\hat{y}) = -\sum_{i=1}^{C} y_i * log\hat{y_i}, \qquad \hat{y_i} = f_i(x,w)$$

◆ 整体样本的损失函数：从数据集的整体来看损失情况。

✓ 优化目标：模型在训练集上的平均差异最小化

$L(w) = \frac{1}{m}\sum_m Loss(f(x,w),\hat{y})$ 其中m为样本个数

训练神经网络——参数学习



采用梯度下降算法求解损失函数的最小值，从而获得最优参数组合。

传统算法流程：

选择一个初始值w

神经网络参数$w = \{w_{mn}^{k}, b_{mn}\}$

Repeat

对w的每一份量计算梯度值 $\partial L/\partial w_j$

$w_j \leftarrow w_j - \eta \partial L/\partial w_j$

Until 直到 ∂L/∂w 非常小（即更新很小）

L

w

$-\eta \partial L/\partial w$

η 是“学习率”

传统算法计算量巨大，一度使神经网络陷入低潮！

训练神经网络——误差反向传播

1986年以后，误差反向传播算法BP(error back propagation, BP)得到关注，解决了对参数逐一求偏导的效率低下问题，为梯度下降算法提供了高效率的实现方法。

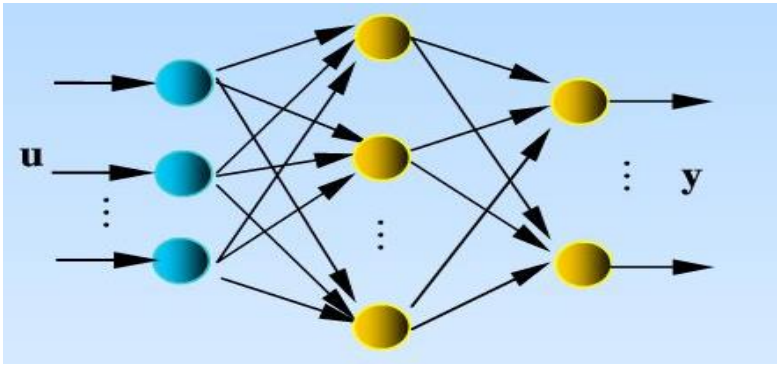
基本思想：由于神经网络涉及多层神经元，将输出层误差逐层反向传播给各隐藏层进行参数更新。

正向传播

计算得到输出值，并计算与实际值间的误差。

误差反向传播

从后向前逐层计算各节点输入参数的梯度（即误差），并根据梯度修改相应参数值。

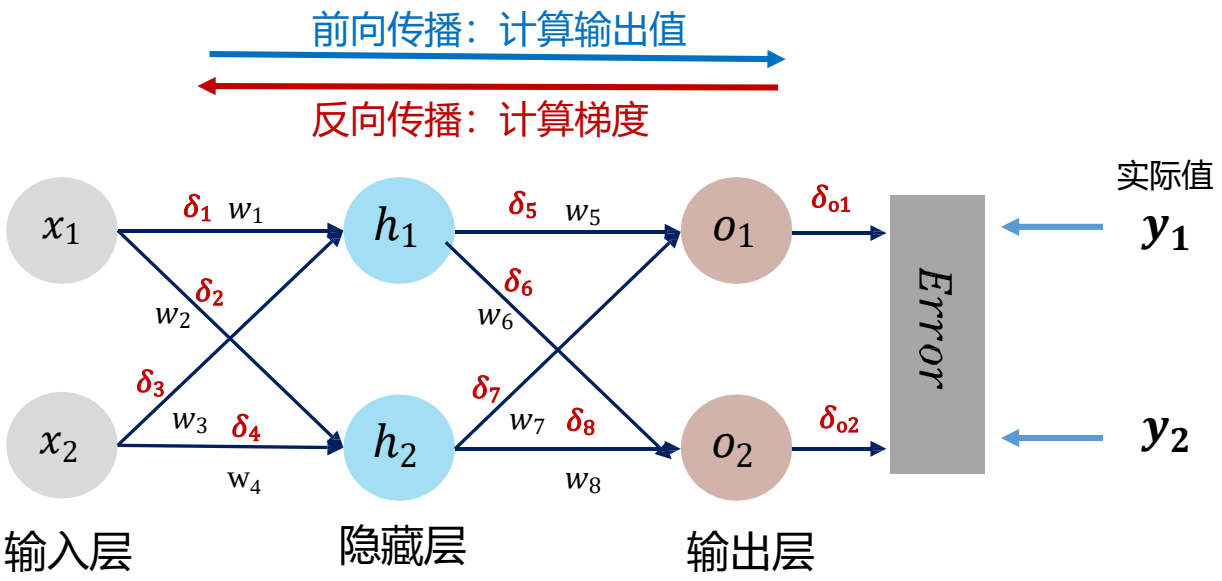


一次前向传播和一次反向传播就可以完成一次梯度下降，实现对所有参数的更新！计算量大大减少。

训练神经网络——误差反向传播

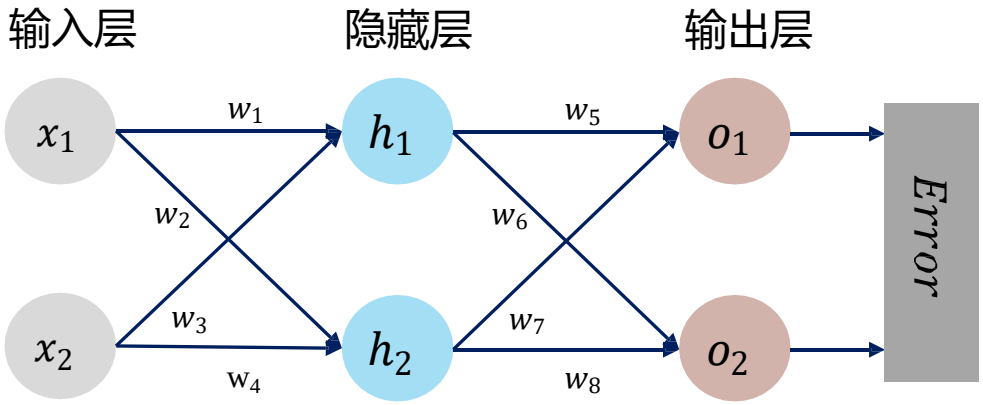
参数的一次更新过程：

1. 参数初始化：为每个连接随机初始化参数值$\{w_1, w_2, \ldots\ldots w_n\}$；
2. 前向传播：计算各隐层和输出层的输出值，并根据标签和输出值计算损失$Error$；
3. 反向传播：计算梯度，即损失$Error$对输出层和各隐层输入参数w_i的偏导数$\delta_i=\frac{\partial Error}{\partial w_i}$；
4. 更新参数：$w_i' = w_i - \eta * \delta_i$，其中$\eta$为学习率。



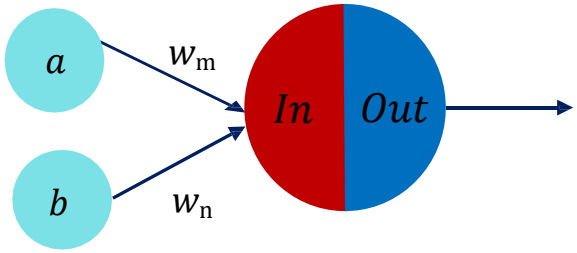
前馈神经网络的误差反向传播

如下结构的2层神经网络，图中w_i为连接权重，Error表示损失。



其中每个神经元的结构如下，包含两部分操作：

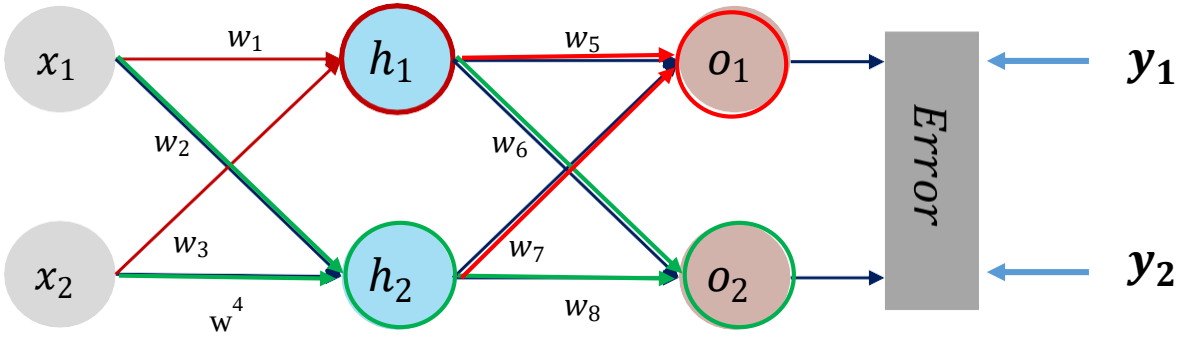
- In操作：对输入加权求和
- Out操作：对In进行激活函数非线性变换



$$In = w_m * a + w_n * b$$

$$Out = Sigmoid(In)$$

前向传播——输出计算



$$In_{h_1} = w_1 * x_1 + w_3 * x_2$$
$$h_1 = Out_{h_1} = Sigmoid(In_{h_1})$$

$$In_{o_1} = w_5 * h_1 + w_7 * h_2$$
$$o_1 = Out_{o_1} = Sigmoid(In_{o_1})$$

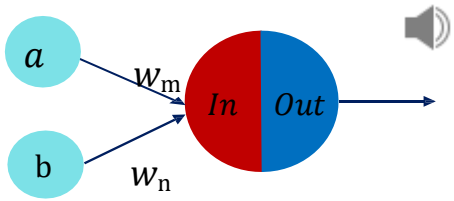
$$In_{h_2} = w_2 * x_1 + w_4 * x_2$$
$$h_2 = Out_{h_2} = Sigmoid(In_{h_2})$$

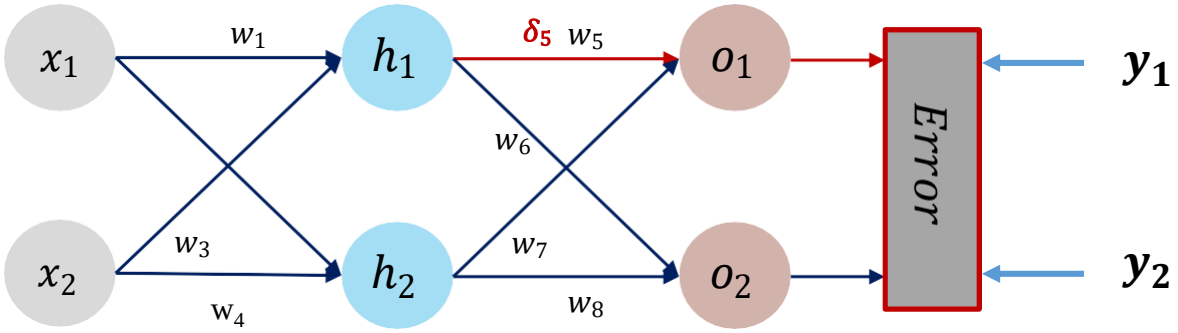
$$In_{o_2} = w_6 * h_1 + w_8 * h_2$$
$$o_2 = Out_{o_2} = Sigmoid(In_{o_2})$$

$$Error = \frac{1}{2}\sum_{i=1}^{2}(o_i - y_i)^2$$

式中1/2是为了后面求导约简方便，不影响网络计算效果。

反向传播——梯度计算





1. 计算w_5~ w_8的梯度(以w_5为例)

$$\partial_5 = \frac{\partial Error}{\partial w_5} = \frac{\partial Error}{\partial O_1} * \frac{\partial O_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial w_5}$$

链式法则： 两个函数组合起来的复合函数，导数等于里函数带入外函数值的导数乘以里函数的导数。即：$y = f(g(x))$，若记$y = f(u), u = g(x)$，则有：$\frac{dy}{dx} = \frac{dy}{du} * \frac{du}{dx}$

where,

$$\frac{\partial Error}{\partial O_1} = o_1 - y_1 \quad \Leftarrow \quad Error = \frac{1}{2}\sum_{i=1}^{2}(o_i - y_i)^2$$

$$\frac{\partial O_1}{\partial In_{o_1}} = o_1 * (1 - o_1) \quad \Leftarrow \quad o_1 = Out_{o_1} = Sigmoid(In_{o_1})$$

$$\frac{\partial In_{o_1}}{\partial w_5} = h_1 \quad \Leftarrow \quad In_{o_1} = w_5 * h_1 + w_7 * h_2$$

$$f(net) = \frac{1}{1+e^{-net}}$$

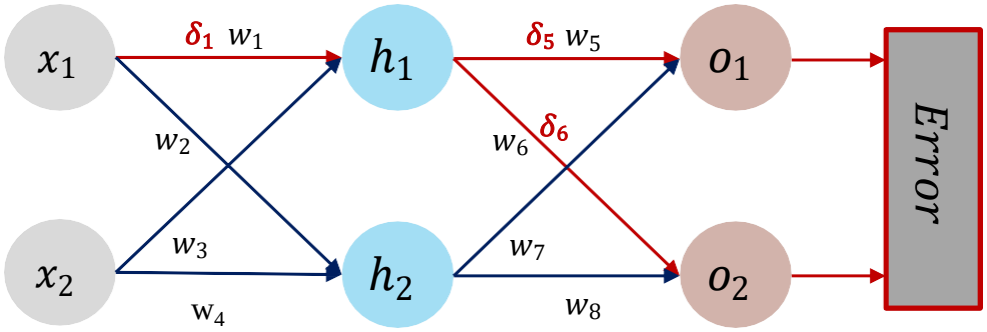
$$f'(net) = \frac{e^{-net}}{(1+e^{-net})^2}$$

$$= \frac{1+e^{-net}-1}{(1+e^{-net})^2}$$

$$= \frac{1}{1+e^{-net}} - \frac{1}{(1+e^{-net})^2}$$

$$= y(1-y)$$

反向传播——梯度计算



1. 计算w_1~ w_4的梯度(以w_1为例)

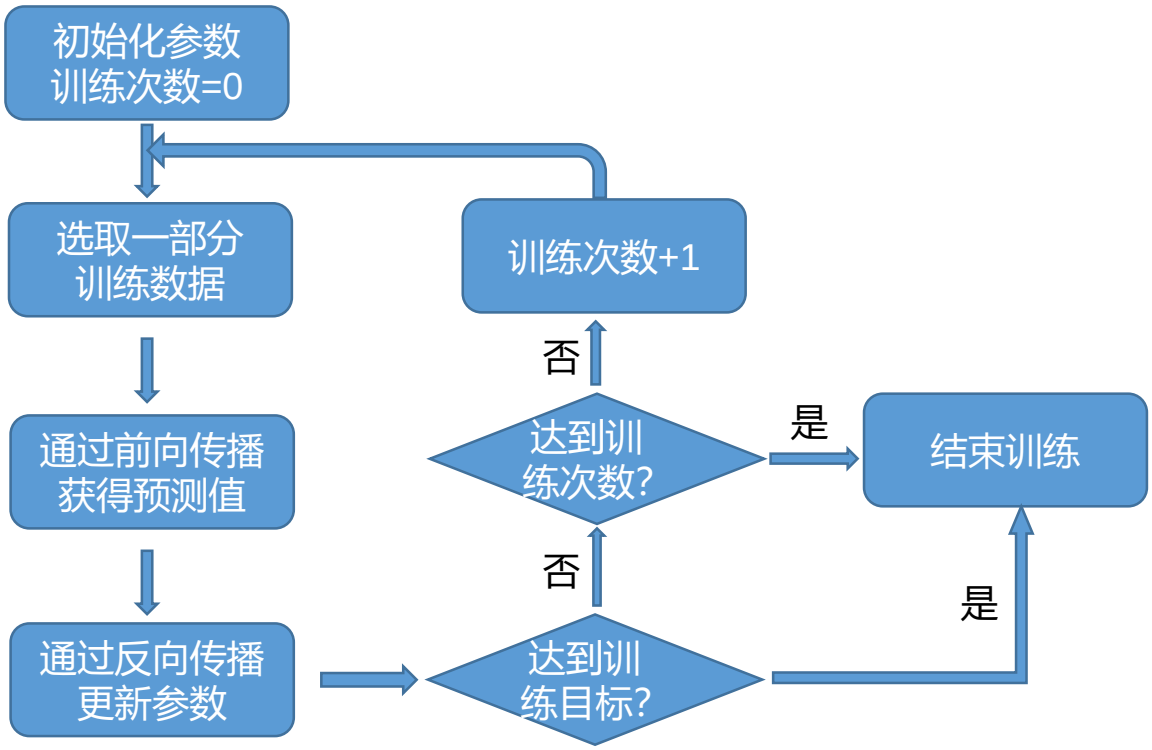
$$\delta_{h_1} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial h_1} + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial In_{o_2}} * \frac{\partial In_{o_2}}{\partial h_1}$$

$$= \delta_{o_1} * [o_1 * (1 - o_1)] * w_5 + \delta_{o_2} * [o_2 * (1 - o_2)] * w_6$$

$$\delta_1 = \frac{\partial Error}{\partial w_1} = \frac{\partial Error}{\partial h_1} * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial_{w_1}}$$

$$= \delta_{h_1} * h_1 * (1 - h_1) * x_1$$

神经网络的训练过程



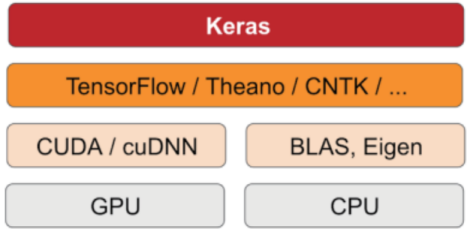
梯度下降很小，参数更新很小

基于Keras实现神经网络

Python的深度学习库Keras

Keras 是深度学习框架，支持用Python语言定义和训练各种深度学习模型，。

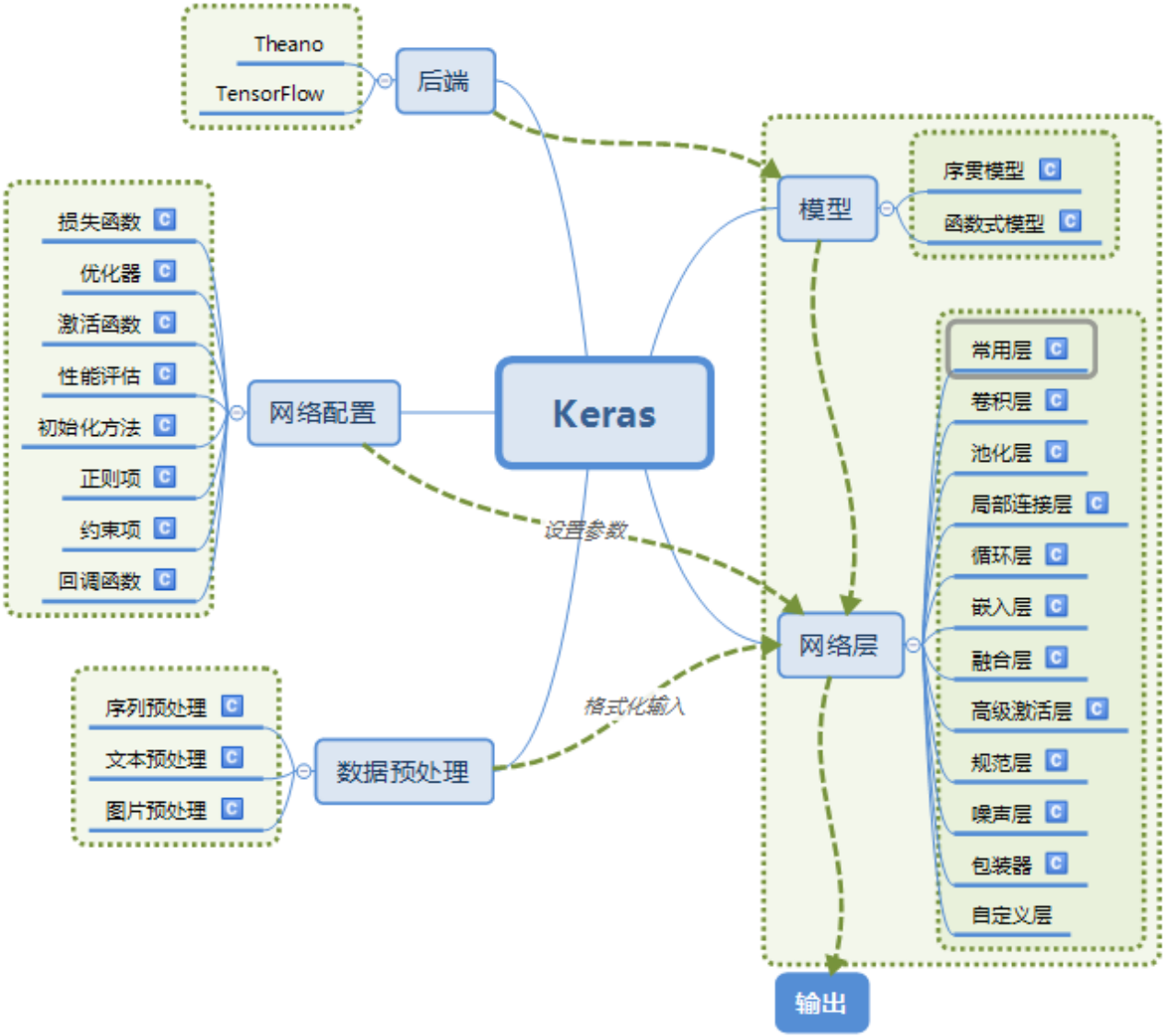
- 高度模块化，支持快速开发。并且支持CPU、GPU无缝切换运行。
- 需要运行在专业的深度学习引擎之上，如Tensorflow、CNTK和Theano等。



注意：在Anaconda集成环境中安装Keras，同时需要安装后端的深度学习引擎如Tensorflow：

```
>>> pip install keras
>>> pip install tensorflow
```

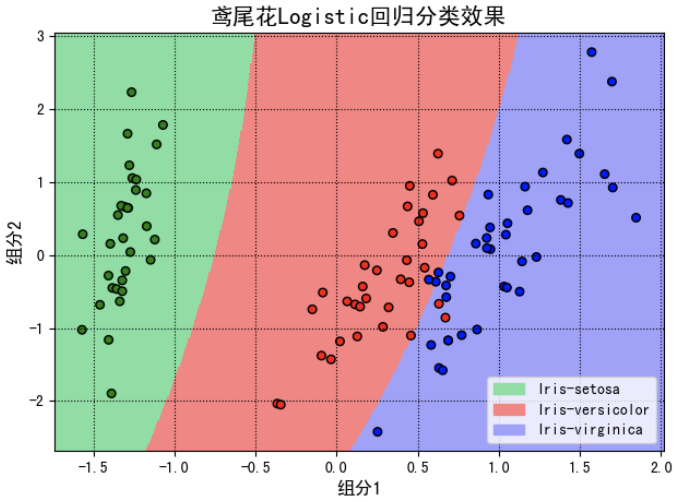
Keras的模块结构


Keras
后端
Theano
TensorFlow
模型
序贯模型
函数式模型
网络配置
损失函数
优化器
激活函数
性能评估
初始化方法
正则项
约束项
回调函数
设置参数
网络层
常用层
卷积层
池化层
局部连接层
循环层
嵌入层
融合层
高级激活层
规范层
噪声层
包装器
自定义层
数据预处理
序列预处理
文本预处理
图片预处理
格式化输入
输出

案例1：用深度神经网络实现鸢尾花分类

- Iris(鸢尾花) 是一个经典数据集，1936年Fisher在模式识别论文中使用。
  - 该数据集共150行，每行1个样本。
  - 每个样本有4个特征： 花萼长度 、花萼宽度、花瓣长度、花瓣宽度
  - 类别(共3类)：山鸢尾Iris Setosa /变色鸢尾Iris Versicolour / 维吉尼亚鸢Iris Virginica

编号	花萼长度(cm)	花萼宽度(cm)	花瓣长度(cm)	花瓣宽度	花的种类
1	5.1	3.5	1.4	0.2	山鸢尾
2	4.9	3.0	1.4	0.2	山鸢尾
3	7.0	3.2	4.7	1.4	杂色鸢尾
4	6.4	3.2	4.5	1.5	杂色鸢尾
5	6.3	3.3	6.0	2.5	维吉尼亚鸢尾
6	5.8	2.7	5.1	1.9	维吉尼亚鸢尾



Scikit-learn自带Iris数据集，可直接引入使用，也可使用下载的数据文件Iris.data。

神经网络分类程序实现方法

第一步导入Keras模型库，创建模型对象

Keras提供了两种“模型”来构建神经网络：

Sequential：顺序式模型或序贯模型，可以通过各层按顺序线性堆叠来构建神经网络层

Functional：函数式模型，在顺序模型的基础上，允许多输出、共享层等结构

```
#导入Keras模型库，定义模型结构
from keras.models import Sequential        # 导入顺序式模型
model = Sequential()                       # 构造一个模型对象model
```

用顺序模型的构建和使用神经网络的基本步骤：

- model.add，添加层；
- model.compile，模型编译，即训练的BP模式设置；
- model.fit，模型训练参数设置 + 训练；
- model.evaluate，模型评估；
- model.predict()，模型预测。

第二步 通过堆叠若干网络层来构建神经网络

model.add(Dense(n, activation, input_shape))

Dense：全连接层，节点与下一层节点完全连接

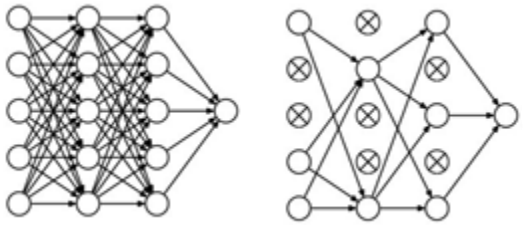
n: 本层节点数

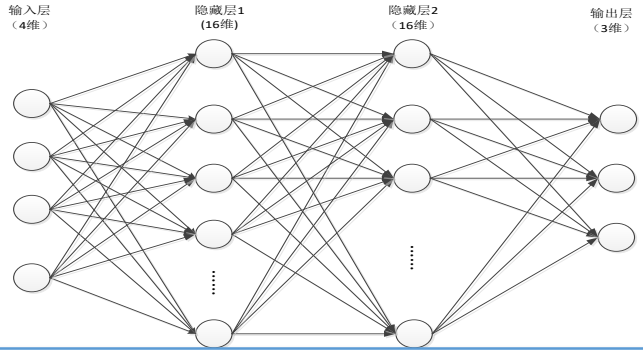
activation：激活函数，如有softmax、relu、tanh、sigmoid等

input_shape：输入数据维度，用元组表示如(6,8)表示6*8两维数据，首层必需说明。

model.add(dropout)

Dropout：在本层添加随机失活比例。为防止过拟合，训练过程中断开一些输入神经元连接。






```
from keras.layers import Dense, Dropout      #导入层次库
# 通过堆叠层次来定义模型结构
model.add(Dense(16,activation='relu',input_shape=(4,)))  #隐层1
model.add(Dense(16,activation='relu'))                   #隐层2
model.add(Dropout(0.25))                    #隐层2随机失活25%
model.add(Dense(3,activation='softmax'))                 #输出层
```

注意：多分类输出层的激活函数要选择'softmax'，即返回一个由多个概率值（总和为1）组成的数组，每个概率值表示输出为某类的概率

另一种写法

```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential([
    Dense(16, input_shape=(4,)), Activation('relu'),
    Dense(16), Activation('relu'),
    Dropout(0.25),
    Dense(3), Activation('softmax'),
])
```

第三步 对网络进行配置，即定义损失函数、优化器、性能评估指标等，并根据这些参数对网络进行编译。编译之后的网络模型即可用于学习训练。

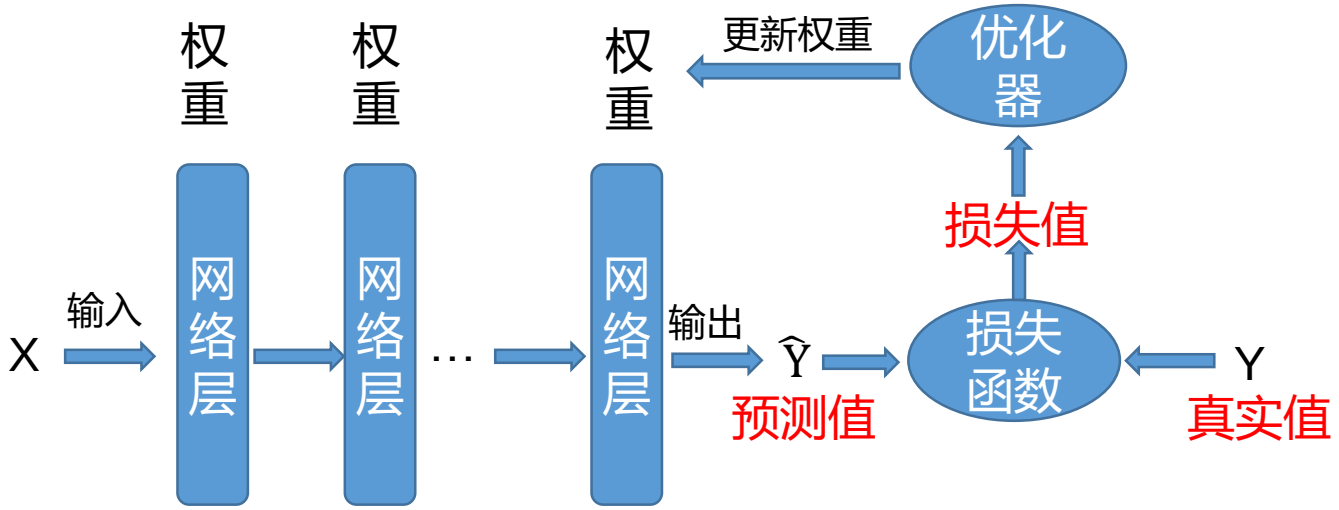
model.compile(loss, optimizer, metrics)

loss：损失函数，如多分类用交叉熵损失函数‘categorical_crossentropy‘, 回归用均方损失函数mean_squared_error 等

optimizer：优化器即参数学习算法，以梯度下降算法为基础的方法‘SGD’ 、’Adam’ 、‘RMSprop’等

metrics: 监控指标列表，包含评估模型在训练和测试时的性能指标，分类一般只关心精度。

```
#编译网络模型
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=["accuracy"])
```



第四步 准备数据

对数据进行预处理，符合网络要求的类型、形状和数据分布。

编号	花萼长度(cm)	花萼宽度(cm)	花瓣长度(cm)	花瓣宽度(cm)	花的种类
1	5.1	3.5	1.4	0.2	山鸢尾
2	4.9	3.0	1.4	0.2	山鸢尾
3	7.0	3.2	4.7	1.4	杂色鸢尾
4	6.4	3.2	4.5	1.5	杂色鸢尾
5	6.3	3.3	6.0	2.5	维吉尼亚鸢尾
6	5.8	2.7	5.1	1.9	维吉尼亚鸢尾

```
#导入库，读入数据文件
import pandas as pd
from sklearn.model_selection import train_test_split
data = pd.read_csv('data\iris.data', header = None)
data.columns = ['sepal length','sepal width','petal length','petal width','class']
print(data.iloc[0:5,:])    #查看前5条数据
```

```
   sepal length  sepal width  petal length  petal width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

1）向量化：神经网络的所有**输入**和**目标**都必须是**浮点数**或**整数**张量(即多维数据)。**本例中，数据为1D张量，形状为(features)。

```
#数据特征X取值于前4列(数据中无编号列)
X = data.iloc[:,0:4].values.astype(float)
#将类名转换为整数
data.loc[ data['class'] == 'Iris-setosa', 'class' ] = 0
data.loc[ data['class'] == 'Iris-versicolor', 'class' ] = 1
data.loc[ data['class'] == 'Iris-virginica', 'class' ] = 2
#标签y取值于第4列
y = data.iloc[:,4].values.astype(int)
#分割数据为训练集和测试集
train_x, test_x, train_y, test_y = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=0)
```

第四步 准备数据

对数据进行预处理，符合网络要求的类型、形状和数据分布。

编号	花萼长度(cm)	花萼宽度(cm)	花瓣长度(cm)	花瓣宽度(cm)	花的种类
1	5.1	3.5	1.4	0.2	山鸢尾
2	4.9	3.0	1.4	0.2	山鸢尾
3	7.0	3.2	4.7	1.4	杂色鸢尾
4	6.4	3.2	4.5	1.5	杂色鸢尾
5	6.3	3.3	6.0	2.5	维吉尼亚鸢尾
6	5.8	2.7	5.1	1.9	维吉尼亚鸢尾

2）标准化：将数据根据自身一定比例进行处理，使之落入一个小的特定区间。因为取值范围差异大容易造成训练不收敛。

标准差标准化z-score。每个特征标准化为均值1、标准差为0的数据分布。

$$x^* = \frac{x - \mu}{\sigma}$$

μ为所有样本数据的均值，σ为标准差

```
#特征数据标准化，转换为均值0、标准差为1的分布
mean=train_x.mean(axis=0)
std=train_x.std(axis=0)
train_x=(train_x-mean)/std
test_x=(test_x-mean)/std
print(train_x[0:5,:])
```

```
[[ 0.61303014  0.10850105  0.94751783  0.73603967]
 [-0.56776627 -0.12400121  0.38491447  0.34808318]
 [-0.80392556  1.03851009 -1.30289562 -1.3330616 ]
 [ 0.25879121 -0.12400121  0.60995581  0.73603967]
 [ 0.61303014 -0.58900572  1.00377816  1.25331499]]
```

注意：测试数据的标准化要根据训练数据的规则计算，否则会造成误差。
例如： 训练集如果100转为1，那么测试集200转为1.5；否则200转为1，但模型会将其看作100.

第四步 准备数据

对数据进行预处理，符合网络要求的类型、形状和数据分布。

编号	花萼长度(cm)	花萼宽度(cm)	花瓣长度(cm)	花瓣宽度(cm)	花的种类
1	5.1	3.5	1.4	0.2	山鸢尾
2	4.9	3.0	1.4	0.2	山鸢尾
3	7.0	3.2	4.7	1.4	杂色鸢尾
4	6.4	3.2	4.5	1.5	杂色鸢尾
5	6.3	3.3	6.0	2.5	维吉尼亚鸢尾
6	5.8	2.7	5.1	1.9	维吉尼亚鸢尾

3）类别标签独热编码：多分类使用损失函数categorical_crossentropy，标签必须为多类模式，即one-hot编码的向量，而不是单个数值。

One-hot**独热码**是一组数，其中只有一个值为1，其余都是0。
如三类花的特征标签分别为0,1,2，采用独热编码，则将每个标签对应一个编码：
0->[1 0 0]
1->[0 1 0]
2->[0 0 1]

np_utils提供函数实现将类别标签向量转换独热矩阵表示：

```
from keras.utils import np_utils
np_utils.to_categorical(类别标签, 总类别数)
```

```
#将标签的结果类型转化为one-hot独热矩阵
from keras.utils import np_utils
train_y_ohe = np_utils.to_categorical(train_y, 3)
test_y_ohe = np_utils.to_categorical(test_y, 3)
print('前5条测试数据标签值：', test_y[0:5])
print('前5条测试数据标签的独热码：\n',test_y_ohe[0:5])
```

```
前5条测试数据标签值： [2 1 0 2 0]
前5条测试数据标签的独热码：
[[ 0.  0.  1.]
 [ 0.  1.  0.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]]
```

第五步 模型训练

model.fit(x, y, batch_size=32, epochs=10, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0**)**

x：**输入数据。**如果模型只有一个输入，那么x的类型是numpy array，如果模型有多个输入，那么x的类型应当为list，list的元素是对应于各个输入的numpy array
y：**标签**，numpy array
batch_size：整数，指定进行梯度下降时每个batch包含的样本数。训练时一个batch的样本会被计算一次梯度下降，更新依次权重，使目标函数优化一步。
epochs：整数，**训练迭代次数**，当未设置initial_epoch时，即是训练的总轮数，否则总轮数为epochs - inital_epoch
verbose：日志显示，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
validation_data：形式为（X，y）的tuple，是指定的验证集。此参数将覆盖validation_spilt。

```
#训练模型
model.fit(train_x, train_y_ohe, epochs=50, batch_size=1, verbose=2, validation_data=(test_x,test_y_ohe))
```

```
Train on 120 samples, validate on 30 samples
Epoch 1/50
 - 1s - loss: 1.4220 - acc: 0.4667 - val_loss: 0.8815 - val_acc: 0.6667
Epoch 2/50
 - 0s - loss: 0.8780 - acc: 0.5500 - val_loss: 0.6716 - val_acc: 0.6000
Epoch 3/50
 - 0s - loss: 0.7186 - acc: 0.6000 - val_loss: 0.5575 - val_acc: 0.5667
Epoch 4/50
 - 0s - loss: 0.5913 - acc: 0.6833 - val_loss: 0.4668 - val_acc: 0.9333
Epoch 5/50
 - 0s - loss: 0.5560 - acc: 0.7083 - val_loss: 0.4418 - val_acc: 0.7667
Epoch 6/50
 - 0s - loss: 0.5093 - acc: 0.7583 - val_loss: 0.4810 - val_acc: 0.6333
Epoch 7/50
 - 0s - loss: 0.4560 - acc: 0.8000 - val_loss: 0.4581 - val_acc: 0.6333
Epoch 8/50
 - 0s - loss: 0.4155 - acc: 0.8417 - val_loss: 0.3384 - val_acc: 1.0000
Epoch 9/50
 - 0s - loss: 0.3977 - acc: 0.8333 - val_loss: 0.3536 - val_acc: 0.8333
Epoch 10/50
 - 0s - loss: 0.4193 - acc: 0.8417 - val_loss: 0.3198 - val_acc: 0.9667
```

各轮训练的情况：时间、训练集损失值、训练集精确率、验证集损失值、验证集精确率

第六步 模型的性能评价和预测应用

- 模型的性能评价

loss,accuracy = **model.evaluate(**X_test,Y_test, verbose**)**

loss：预测标签和目标标签之间的损失值。
accuracy：精确率
X_test：测试集数据
Y_test：测试集标签
verbose：日志显示，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

- 模型的预测应用

model.predict(X_test, batch_size, verbose**)**

X_test：测试集数据
batch_size：整数，指定进行梯度下降时每个batch包含的样本数。
verbose：日志显示，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。

```
#评估模型
loss, accuracy = model.evaluate(test_x, test_y_ohe, verbose=2)
print('loss = {},accuracy = {} '.format(loss,accuracy) )

#查看预测结果，属于各类的概率
classes = model.predict(test_x, batch_size=1, verbose=2)
print('测试样本数：',len(classes))
print("分类概率:\n",classes)
```

loss = 0.024684619158506393,
accuracy = 1.0
注意：每次训练结果可能会不同。

```
测试样本数： 30
分类概率:
[[  1.74706940e-07   8.97341873e-03   9.91026342e-01]
 [  2.18595542e-05   9.99800384e-01   1.77832117e-04]
 [  1.00000000e+00   7.01450498e-09   9.52518725e-17]
 [  1.17023138e-08   1.75700709e-03   9.98242974e-01]
 [  9.99999285e-01   7.22856271e-07   2.99556251e-13]
 [  2.25498220e-09   7.14936992e-04   9.99285042e-01]
 [  9.99999642e-01   3.01418794e-07   1.27348978e-13]
```

Keras的符号计算

Keras的底层库使用Theano或TensorFlow，它们也称为Keras的后端。Theano和TensorFlow都是“符号式”的库。因此使得Keras的编程与传统的Python代码有所差别。Keras采用符号计算。

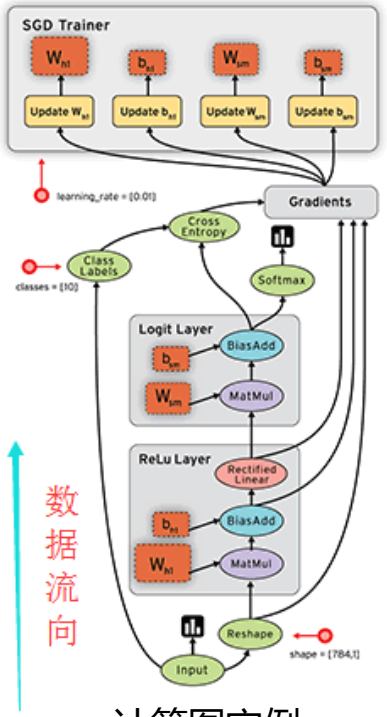
符号计算流程：

1）首先定义各种变量，然后建立一个“计算图”，计算图规定了各个变量之间的计算关系。

2）只有通过编译确定计算图的内部细节，或实际生成可调用的函数（K.function()）后，后面才能使用。但这时仍是一个“空壳子”。

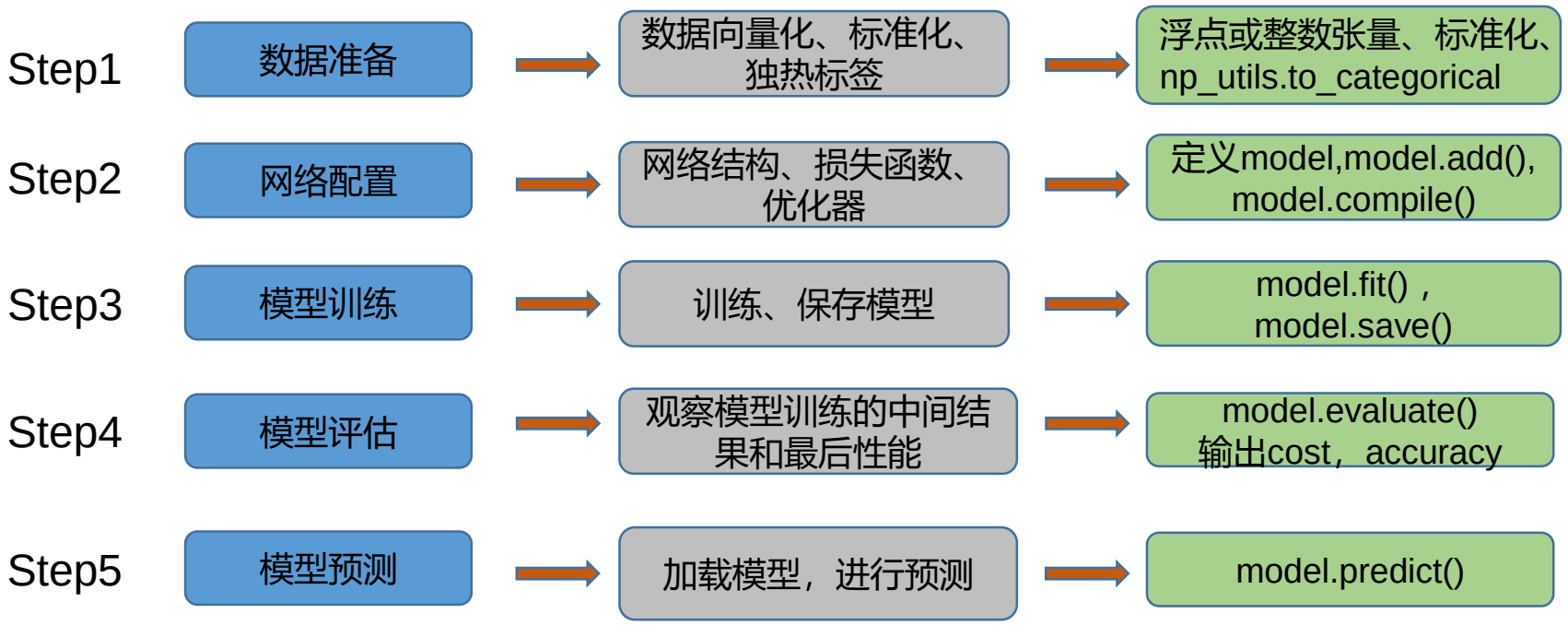
3）只有把需要运算的输入放进去后，才能在整个模型中形成数据流，从而形成输出值。

K.function()是Keras的后端函数，它为符号计算实际生成可调用的函数，它可以接收传入数据，并返回一个numpy数组。



计算图实例

神经网络建模步骤


Step1
数据准备
数据向量化、标准化、独热标签
浮点或整数张量、标准化、np_utils.to_categorical
Step2
网络配置
网络结构、损失函数、优化器
定义model,model.add(), model.compile()
Step3
模型训练
训练、保存模型
model.fit()，model.save()
Step4
模型评估
观察模型训练的中间结果和最后性能
model.evaluate() 输出cost，accuracy
Step5
模型预测
加载模型，进行预测
model.predict()

神经网络算法的特点

优点

- 有很强的非线性拟合能力，可映射任意复杂的非线性关系，而且学习规则简单，便于计算机实现。
- 具有很强的鲁棒性和容错性、以及联想记忆和强大的自学习能力。

缺点

- BP算法学习速度慢

  由于BP算法本质上为梯度下降法，而它所要优化的目标函数又非常复杂。

- 训练失败的可能性大
  - 算法很有可能陷入局部极值，使训练失败。
  - 网络结构的选择尚无一种统一而完整的理论指导，一般只能由经验选定。
  - 容易出现过拟合。

神经网络原理与实现
实验指导

实验

1. 鸢尾花分类识别

1）参照讲义中的案例1 实现程序

2）运行程序，观察模型的分类性能并进行预测应用

1. 程序中注意分析观察数据结构和形状
2. 数据预处理得到了什么样的结构？采用了什么样的数据标准化（归一化）方法？

2. 手写数字的神经网络识别程序

1）模仿案例1并参照教材第2章P21-23 实现程序，运行并查看模型的损失和分类准确率，应用模型到测试集或测试集的某个数据上进行预测。

2）调整参数，比较不同模型分类性能，可以只调整其中1项或几项，观察对分类精度的影响并记录结果。

（1）尝试使用2个或3个隐藏层；

（2）尝试使用更多或更少的隐藏单元，比如 32 个、64 个等；

（3）尝试调整训练轮次epochs, 批次大小batch_size；

(4) 尝试各层采用一定的Dropout失活比例；

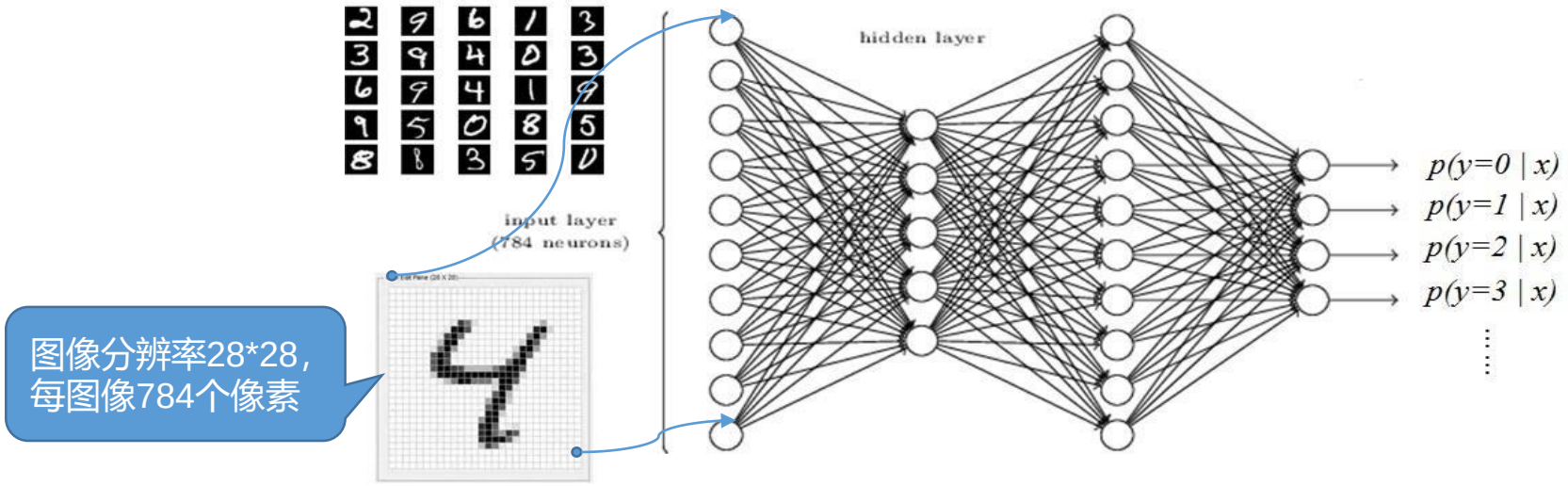
（5）尝试使用 mse 损失函数代替 categorical_crossentropy ；

（6）尝试使用 tanh 激活（这种激活在神经网络早期非常流行）代替 relu；

（7）尝试使用优化器Adam替代rmsprop 。

需提交程序，程序命名“学号_Dense_n.py”，代码对应你最高分类性能。另外，注意为各语句增加注释。

实验2：用神经网络实现手写数字识别



Keras数据集MNIST：

Classes	10类（0~9）
Samples per class	7000左右
Samples total	60000张训练图像 10000张测试图像
Dimensionality	**784(28*28)**
Features	integers 0-255

定义网络结构：

- **输入层神经元shape:((28*28)，)**
- **隐藏层层数及每层神经元个数：1层, 512个**
- **输出层神经元个数：**10
- **激活函数：**隐藏层Relu，输出层softmax

MNIST数据集

- 运行程序时，可以自动下载
- 但由于直接运行程序时下载数据较慢，可以先下载mnist.npz(约11M)放到 ~/.keras/datasets文件夹下，win7/win8下：“~”文件夹一般是C:\Users\Administrator

  下载地址：

  1）https://s3.amazonaws.com/img-datasets/mnist.npz

  2）或其他镜像地址、用户分享等，如分享百度云：
  https://pan.baidu.com/s/1aZRp0uMkNj2QEWYstaNsKQ
  提取码: 3a2u

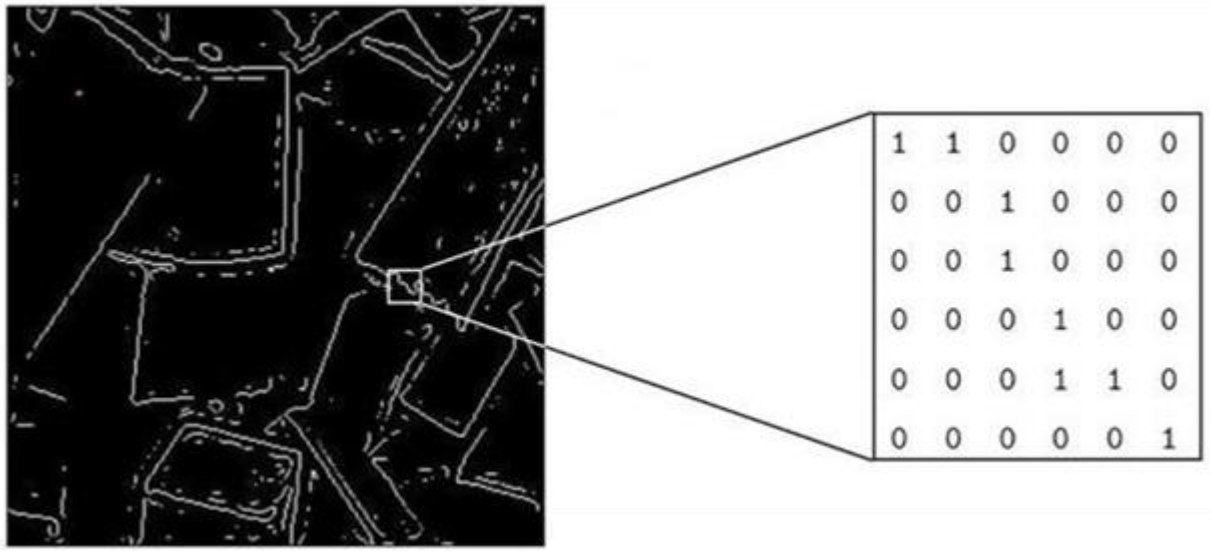
keras框架为我们提供了一些常用的内置数据集这些数据集可直接调用。详见https://keras.io/zh/datasets/ 或“Keras数据集使用说明.doc”。

补充知识1：图像的数字化表示

用给定大小的网格将连续图像离散化，每个小方格是一个像素（Pixel），对应一种颜色值，颜色值矩阵表示数字图像。例如分辨率是640×480，乘积就是像素总数。同样大小的图像，像素越大越清晰。

- **二值图像**

像素矩阵由0、1两个值构成，”0”代表黑色，1”代白色。通常用于文字、线条图的扫描识别（OCR）和掩膜图像的存储。

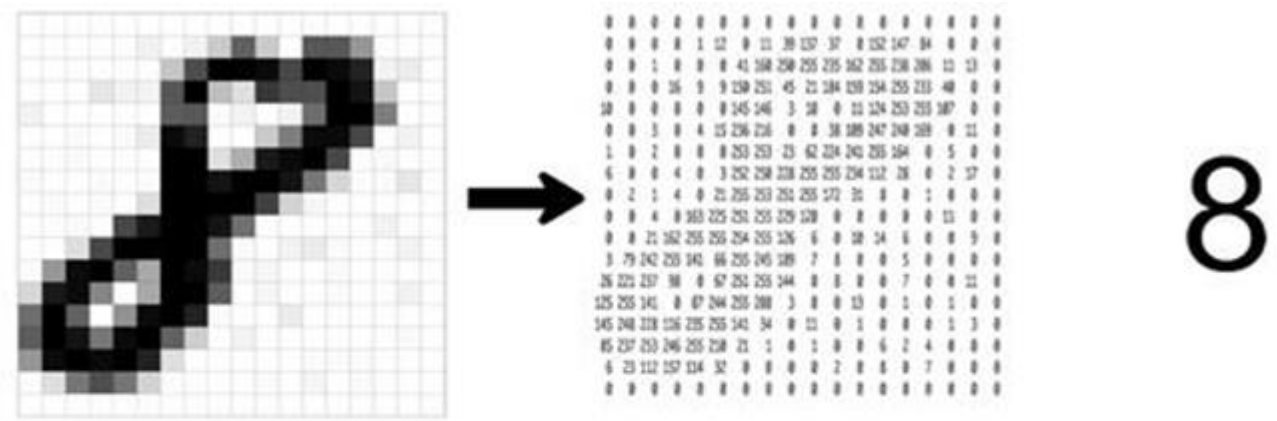


二值图像采用3D 张量表示： (samples, height, width)

➢ 灰度图像

灰度图像矩阵元素的取值范围通常为[0，255]。“0”表示纯黑色，“255”表示纯白色，中间的数字从小到大表示由黑到白的过渡色，每个像素值用8位二进制表示。

二值图像可以看成是灰度图像的特例。



灰度图像采用3D 张量表示： (samples, height, width)

补充知识2：神经网络的数据表示

数据存储在**多维数据**中，也叫**张量(tensor)。**

- 仅包含一个数字的张量也叫标量(0D)
- 一维数据叫向量，对应一维张量(1D)
- 二维数据叫矩阵，对应二维张量(2D)
- 将多个矩阵组合成一个新的数据，得到一个三维张量(3D)
- 由3D张量组合可得到4维张量，以次类推，可得到高维张量。深度学习一般处理0D~4D的张量。处理视频可能会遇到5D

可以通过python的以下属性，观察张量特性，例如：

```
>>>x=np.array([[5,78,2,34,0],[6,79,3,35,1],[7,80,4,36,2]])
>>>x.ndim      #维度，即轴的个数
2
>>>x.shape    #形状，整数元组，表示张量沿每个轴的维度大小（元素个数）
(3,5)
>>>x.dtype     #数据类型，神经网络一般处理的数据可以是float32、uint8、float64
int32
```

补充知识3：MNIST的数据结构及预处理

- MNIST的数据结构：

```
print('训练集形状：',train_images.shape)
print('测试集形状：',test_images.shape)
```

训练集形状： (60000, 28, 28)　　是3D张量（samples,height,width）

测试集形状： (10000, 28, 28)　　是3D张量（samples,height,width）

```
print('训练集标签长度：',len(train_labels),' 训练集标签：',train_labels)
```

训练集标签长度： 10000　测试集标签： [7 2 1 ..., 4 5 6]　是1D张量

```
#因为全连接Dense层只接收1D张量，即input_shape=(n,)所以
train_images = train_images.reshape((60000, 28 * 28))
Print(train_images.shape)                    #得到 (60000, 784)　（samples,features）
```

- 数据标准化（归一化）。因为每个像素用灰度值[0, 255]表示，转换为[0,1]之间。

```
train_images = train_images.astype('float32') / 255
```

补充知识4：模型运行参考结果：

Console 2/A

```
In [4]: runfile('D:/教学/理学院_深度学习/DigitsSort.py', wdir='D:/教学/理学院_深度学习')
训练集形状： (60000, 28, 28)
训练集标签长度： 60000   训练集标签： [5 0 4 ..., 5 6 8]
测试集形状： (10000, 28, 28)
训练集标签长度： 10000   测试集标签： [7 2 1 ..., 4 5 6]
Epoch 1/5
60000/60000 [==============================] - 4s 62us/step - loss: 0.2549 - acc: 0.9270
Epoch 2/5
60000/60000 [==============================] - 4s 62us/step - loss: 0.1041 - acc: 0.9688
Epoch 3/5
60000/60000 [==============================] - 4s 63us/step - loss: 0.0697 - acc: 0.9791
Epoch 4/5
60000/60000 [==============================] - 4s 66us/step - loss: 0.0495 - acc: 0.9852
Epoch 5/5
60000/60000 [==============================] - 4s 72us/step - loss: 0.0379 - acc: 0.9891
loss = 0.07047060898160562,accuracy = 0.9795
测试样本数： 10000
分类概率：
[[  1.32915096e-10   5.11043161e-12   1.44922737e-07 ...,   9.99995708e-01
    1.55983386e-08   1.16970924e-07]
 [  1.10746265e-11   1.50416128e-07   9.99999642e-01 ...,   8.48021097e-18
    8.98889851e-09   3.67933841e-17]
 [  1.26665824e-07   9.98091996e-01   1.05887033e-04 ...,   4.58711293e-04
    1.12818158e-03   8.45541649e-07]
 ...,
 [  7.36297077e-14   6.65802699e-12   7.43826604e-13 ...,   1.18414732e-07
    3.44918192e-08   6.87516035e-07]
 [  1.75850140e-10   1.13931337e-10   7.79363032e-12 ...,   9.93146954e-11
    5.17253784e-06   1.72536326e-11]
 [  1.06310414e-12   3.59812747e-17   1.52263467e-12 ...,   1.97556472e-16
    1.25836485e-14   6.05398006e-16]]
```

注意：神经网络每次运行结果会有所不同！