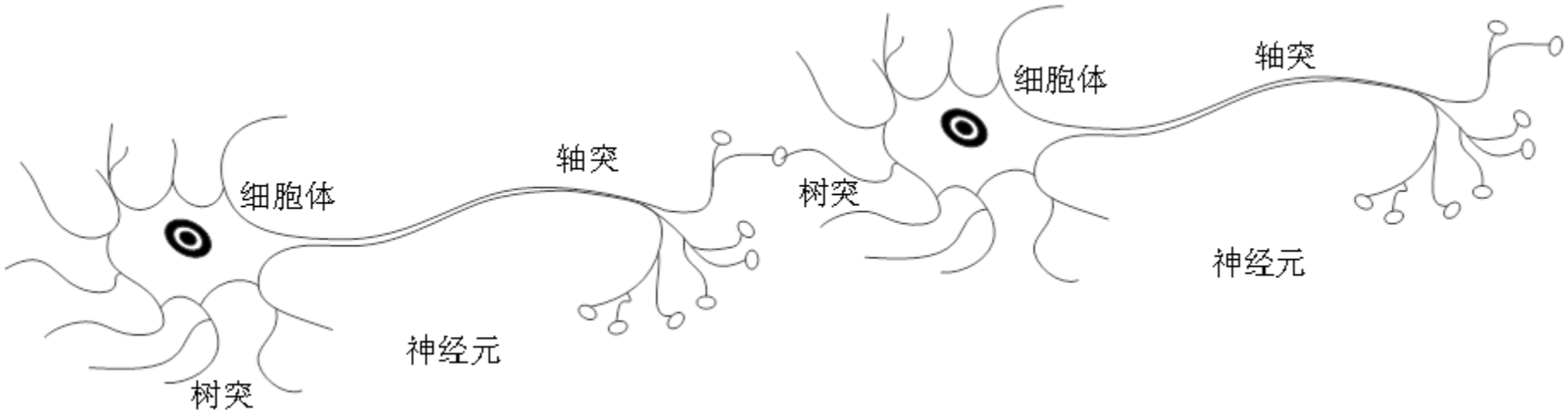
神经元

生物神经元

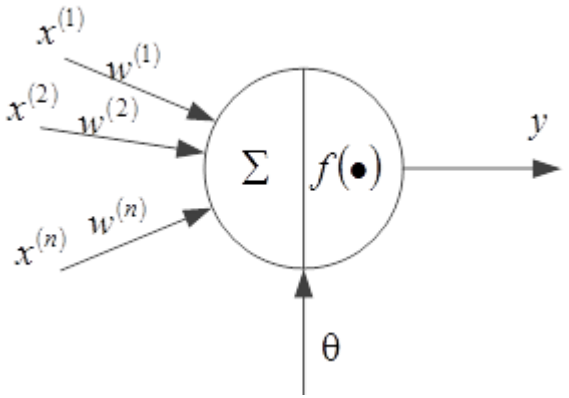
人工神经元（简称神经网络的基本组成单元，它是对生物神经元的模拟、抽象和简化。现代神经生物学的研究表明，生物神经元是由细胞体、树突和轴突组成的。通常一个神经元包含一个细胞体和一条轴突，但有一个至多个树突。



神经元

M-P模型

受生物神经元对信息处理过程的启迪，人们提出了很多人工神经元模型，其中影响最大的是1943年心理学家McCulloch和数学家W. Pitts提出的M-P模型（**感知机模型**）。



$x^{(i)}$表示来自其它神经元的输入信息，$i = 1,2,\ldots,n$。$w^{(i)}$表示输入信息对应的连接系数值。∑表示对输入信息进行加权求和。θ是一个阈值，模拟生物神经元的兴奋“限度”。输入信息经过加权求和后，与阈值进行比对。

神经元

人工神经元模型

信息加工过程可描述如下式：

$$I = \sum_{i=1}^{n} w^{(i)} x^{(i)} + \theta$$

$f(\cdot)$称为激励函数或转移函数，它对前期加工后的信息进行一个映射，得到输出y，如下式：

$$y = f(I)$$

激励函数一般采用非线性函数。

对M-P模型而言，神经元只有兴奋和抑制两种状态，因此，它的激励函数$f(\cdot)$定义为单位阶跃函数，输出y只有0和1两种信号。

神经元

感知机模型

感知机模型是二分类的线性分类模型，它能对空间中线性可分的二分类样本点进行划分。

$$y = u\left(\sum_{i=1}^{n} w^{(i)}x^{(i)} + \theta\right) = u\left(\boldsymbol{W}\cdot\boldsymbol{x}^{\boldsymbol{T}} + \theta\right)$$

式中，$u(\cdot)$为单位阶跃函数，$\boldsymbol{W} = \left(w^{(1)}, w^{(2)}, \dots, w^{(n)}\right)$，$\boldsymbol{x} = \left(x^{(1)}, x^{(2)}, \dots, x^{(n)}\right)$。

$\boldsymbol{W}\cdot\boldsymbol{x}^{\boldsymbol{T}} + \theta = 0$表示空间中的一个超平面，该超平面将空间中的点划分为两类。对空间中的某点$\boldsymbol{x_i}$，如果被正确划分，易知$y_i\left(\boldsymbol{W}\cdot\boldsymbol{x}^{\boldsymbol{T}} + \theta\right) > 0$。如果被错误划分，即为误分类点，则$y_i\left(\boldsymbol{W}\cdot\boldsymbol{x}^{\boldsymbol{T}} + \theta\right) \leq 0$。

神经元

感知机模型

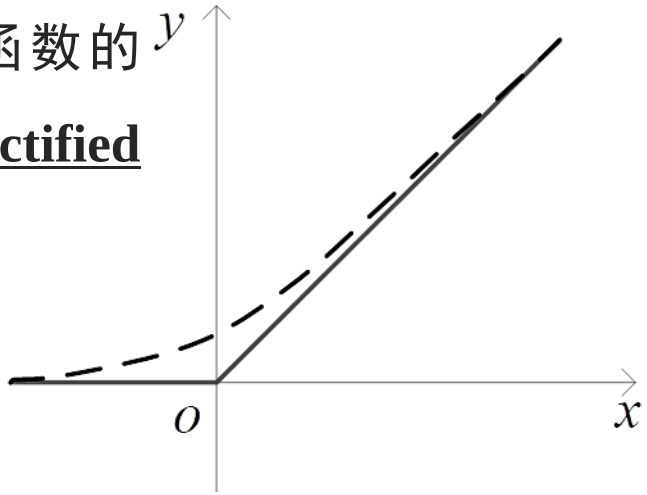
设感知机的样本集$\boldsymbol{S}=\{\boldsymbol{s}_1,\boldsymbol{s}_2,\ldots,\boldsymbol{s}_m\}$包含个$m$样本，每个样本$\boldsymbol{s}_i=(\boldsymbol{x}_i,y_i)$包括一个实例$\boldsymbol{x}_i$和一个标签$y_i$，$y_i\in\{1，0\}$。

步数	操作
1	设定步长η，随机选取$\boldsymbol{W}$和θ初值
2	从样本集S中选取一个样本$s_i=(\boldsymbol{x}_i,y_i)$，计算$\hat{y}_i=u(\boldsymbol{W}\cdot\boldsymbol{x}_i^{\boldsymbol{T}}+\theta)$，调整系数：$\boldsymbol{W}\leftarrow\boldsymbol{W}+\eta(y_i-\hat{y}_i)\boldsymbol{x}_i$，$\theta\leftarrow\theta+\eta(y_i-\hat{y}_i)_i$
3	重复第2步，直到没有误分类点

神经元

激励函数

除了阶跃函数、符号函数和近似阈值函数的Sigmoid函数外，常用的还有ReLU函数(**Rectified Linear Unit**)、Softplus函数和tanh函数。



ReLU函数的定义为：

$$f(x) = max(0, x)$$

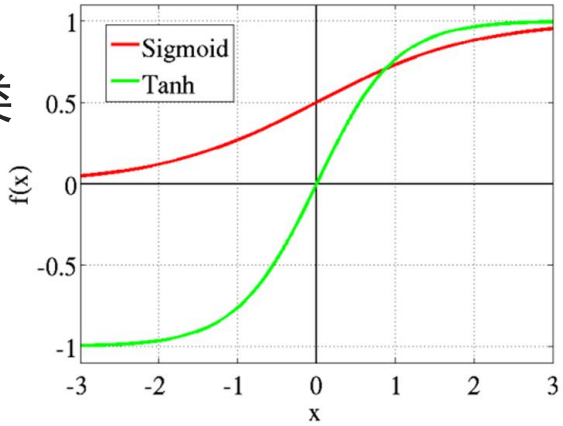
Softplus函数（**ReLU的平滑函数**）的定义为：

$$f(x) = ln(1 + e^x)$$

tanh函数的图像类似于Sigmoid函数，作用也类似于Sigmoid函数。它的定义为：

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

***tanh*(*x*)=2*Sigmoid*(2*x*)-1**

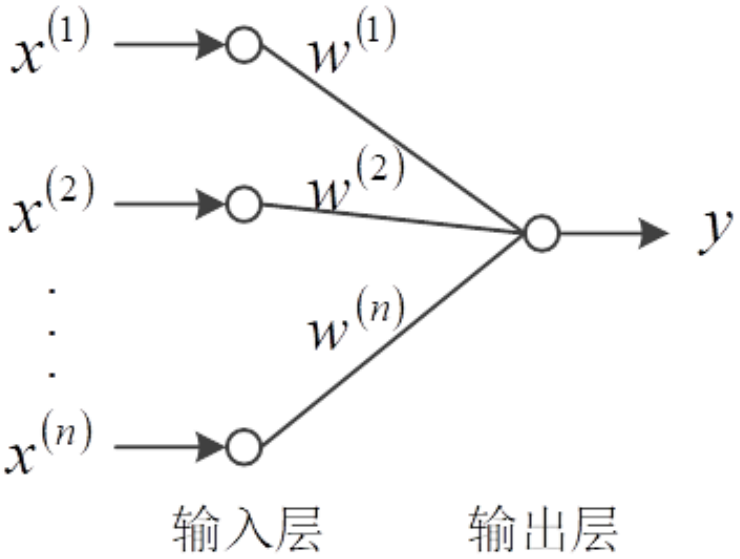


思考：CNN中一般用ReLU取代Sigmoid。为什么?

神经网络

神经网络

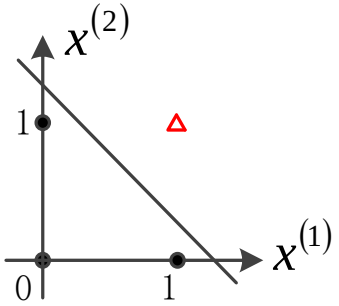
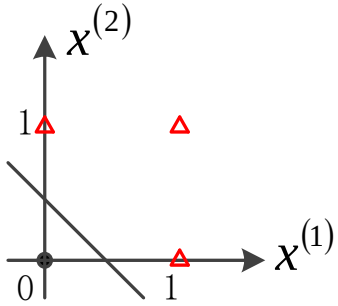
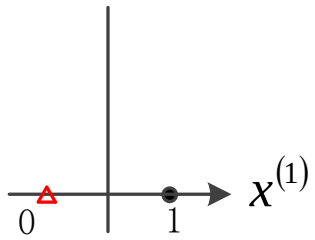
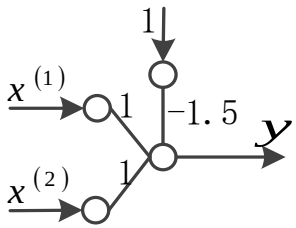
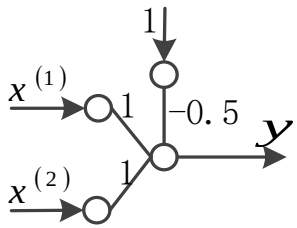
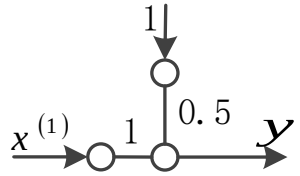
神经元模型在神经网络中的画法



单个神经元只能划分线性可分的二分类点。如果将神经元连接成神经网络，则处理能力会大为增强，这也是神经网络得到广泛应用的原因。

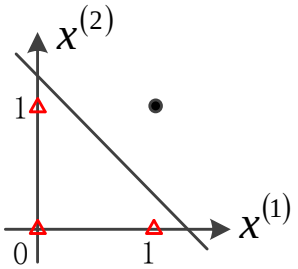
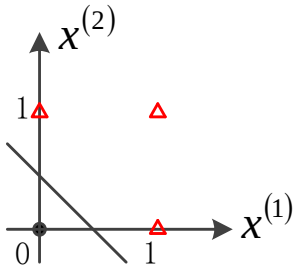
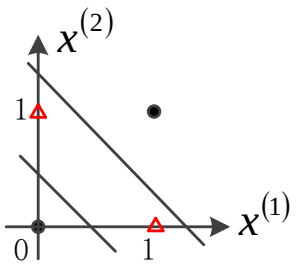
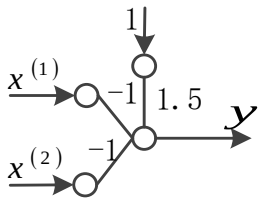
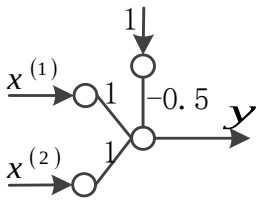
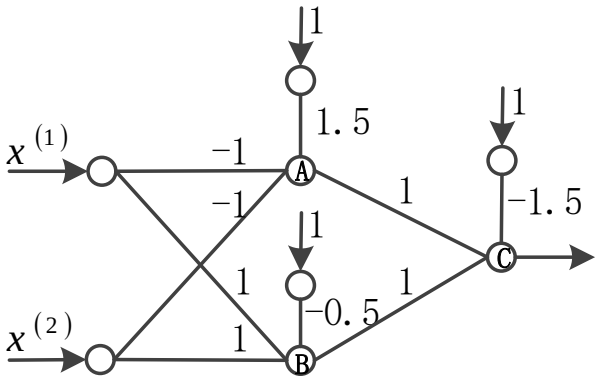
神经网络

用神经网络来模拟逻辑代数中的异或运算

	与运算	或运算	非运算
真值表	$x^{(1)}$ $x^{(2)}$ y 0 0 0 0 1 0 1 0 0 1 1 1	$x^{(1)}$ $x^{(2)}$ y 0 0 0 0 1 1 1 0 1 1 1 1	$x^{(1)}$ y 0 1 0 1 1 0 1 0
图示			
感知机			

神经网络

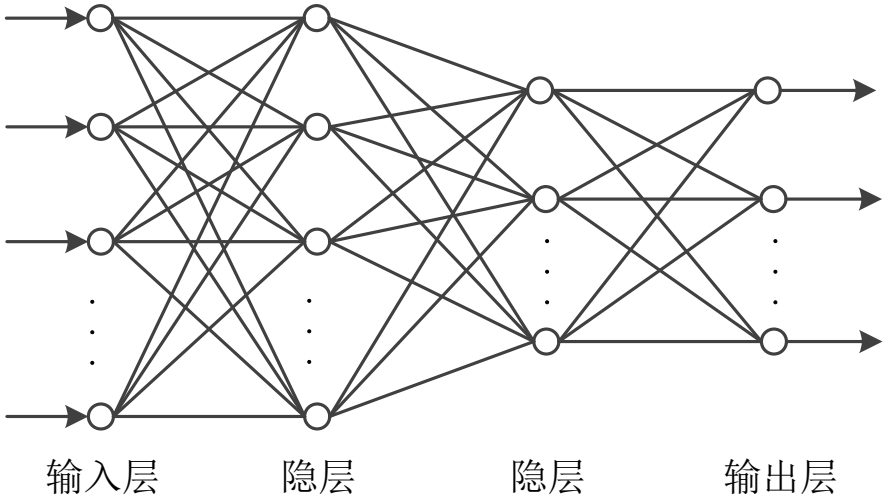
用神经网络来模拟逻辑代数中的异或运算

	与非运算	或运算	异或运算
真值表	$x^{(1)}$ $x^{(2)}$ y_1 0 0 1 0 1 1 1 0 1 1 1 0	$x^{(1)}$ $x^{(2)}$ y_2 0 0 0 0 1 1 1 0 1 1 1 1	$x^{(1)}$ $x^{(2)}$ y 0 0 0 0 1 1 1 0 1 1 1 0
图示			
感知机或神经网络			

神经网络

层状神经网络

理论上，可以通过将神经元的输出连接到另外神经元的输入而**形成任意结构**的神经网络，但目前应用较多的是**层状结构**。



层状结构由**输入层**、**隐层**和**输出层**构成，其中可以有多个隐层。

从信息处理方向来看，神经网络分为**前馈型**和**反馈型**两类：

前馈型网络的信息处理方向是从输入层到输出层逐层前向传递；

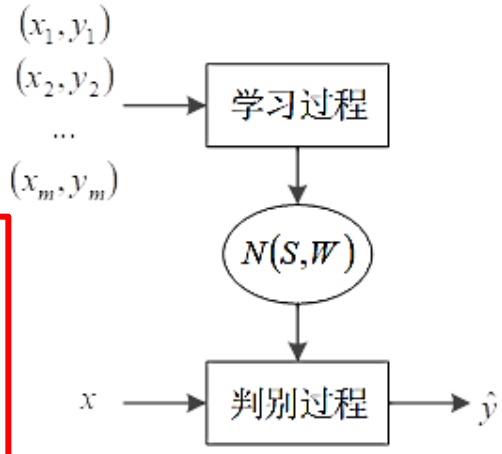
反馈型网络存在信息反向传递，有“闭环”。

神经网络应用模型

S—有向图，提前已定，无需学习

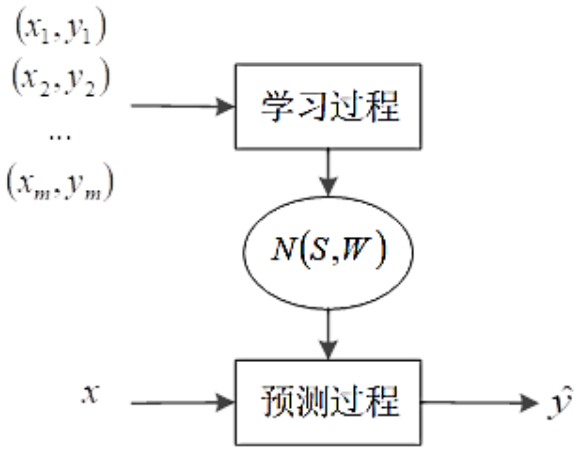
W—权值向量，学习的内容

N—神经网络



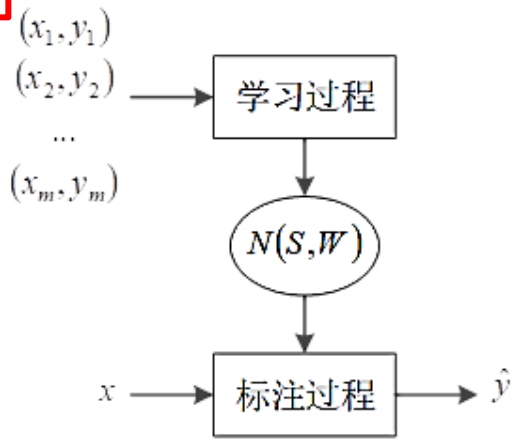
分类任务

BP-NN



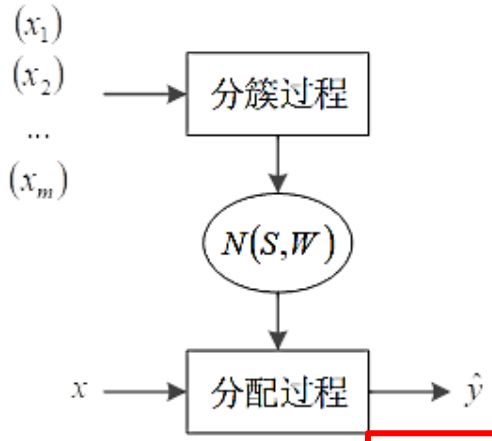
回归任务

BP-NN、*RBF-NN*



标注任务

RNN



聚类任务

自组织特征映射NN

误差反向传播

在研究早期，没有适合多层神经网络的**有效学习方法**是长期困扰该领域研究者的关键问题，以至于人们对人工神经网络的前途产生了怀疑，导致该领域的研究进入了低谷期。直到**1986**年，以Rumelhart和McCelland为首的小组发表了**误差反向传播**(Error Back Propagation，BP)算法，该问题才得以解决，多层神经网络从此得到快速发展采用BP算法来学习的、无反馈的、同层节点无连接的、多层结构的前馈神经网络称为BP神经网络。

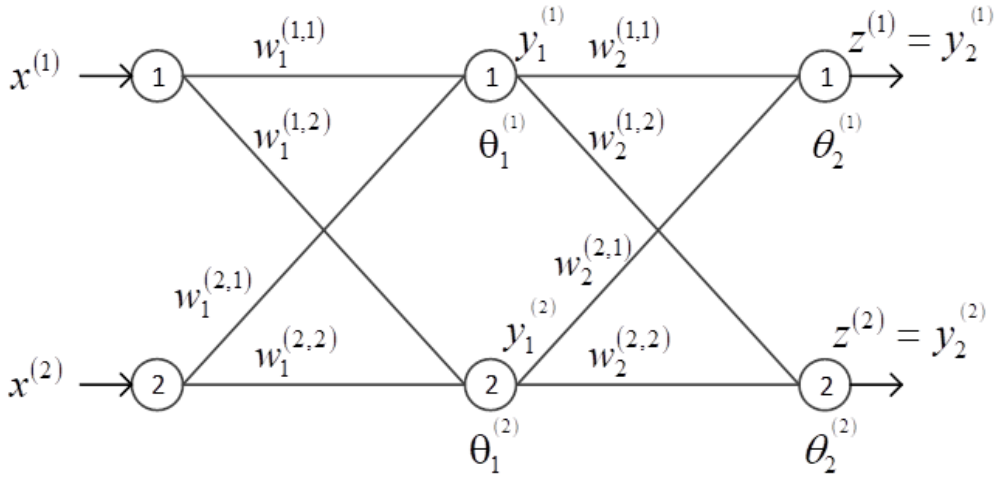
误差反向传播

示例说明

为了便于说明**BP算法**的过程而将真值和假值的分类用**独热编码**来表示

	$x^{(1)}$	$x^{(2)}$	$l^{(1)}$	$l^{(2)}$
1	0	0	0	1
2	0	1	1	0
3	1	0	1	0
4	1	1	0	1

三层感知机结构网络结构



$$\boldsymbol{W}_1 = \begin{bmatrix} w_1^{(1,1)} & w_1^{(1,2)} \\ w_1^{(2,1)} & w_1^{(2,2)} \end{bmatrix}$$

$$\boldsymbol{\theta_1} = \begin{bmatrix} \theta_1^{(1)} & \theta_1^{(2)} \end{bmatrix}$$

误差反向传播

BP学习算法

隐层和输出层的激励函数采用Sigmoid函数，它的导数为：

$$g'(z) = \frac{-1}{(1+e^{-z})^2} e^{-z}(-1) = \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}}$$

$$= g(z)\big(1 - g(z)\big)$$

BP学习算法可分为**前向传播预测**与**反向传播学习**两个过程。要学习的各参数值一般先作**随机初始化**。取训练**样本输入**网络，**逐层前向**计算输出，在输出层得到预测值，此为前向传播预测过程。根据预测值与实际值的误差再从输出层开始逐层反向调节各层的参数，此为反向传播学习过程。经过多样本的**多次前向传播预测和反向传播学习**过程，最终得到网络各参数的最终值。

误差反向传播

前向传播预测过程

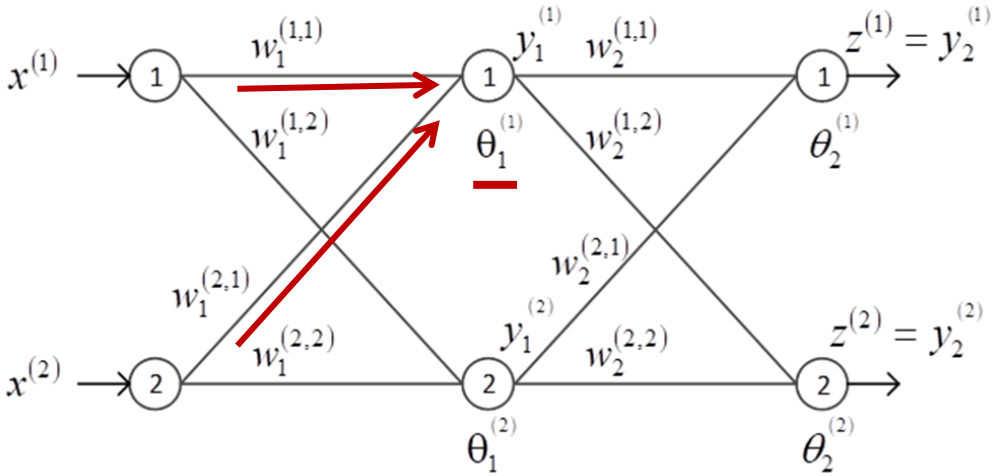
前向传播预测的过程是一个逐层计算的过程。设网络各参数初值为：

$\boldsymbol{W}_1 = \begin{bmatrix} 0.1 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}$，$\boldsymbol{\theta_1} = [0.3 \quad 0.3]$，$\boldsymbol{W}_2 = \begin{bmatrix} 0.4 & 0.5 \\ 0.4 & 0.5 \end{bmatrix}$，$\boldsymbol{\theta_2} = [0.6 \quad 0.6]$。

取第一个训练样本(0,0)，第1隐层的输出：$y_1^{(1)} = g\left(w_1^{(1,1)}x^{(1)} + w_1^{(2,1)}x^{(2)} + \theta_1^{(1)}\right)$

$$= \frac{1}{1 + e^{-\left(w_1^{(1,1)}x^{(1)} + w_1^{(2,1)}x^{(2)} + \theta_1^{(1)}\right)}}$$

$$= \frac{1}{1+e^{-0.3}} = 0.574,$$

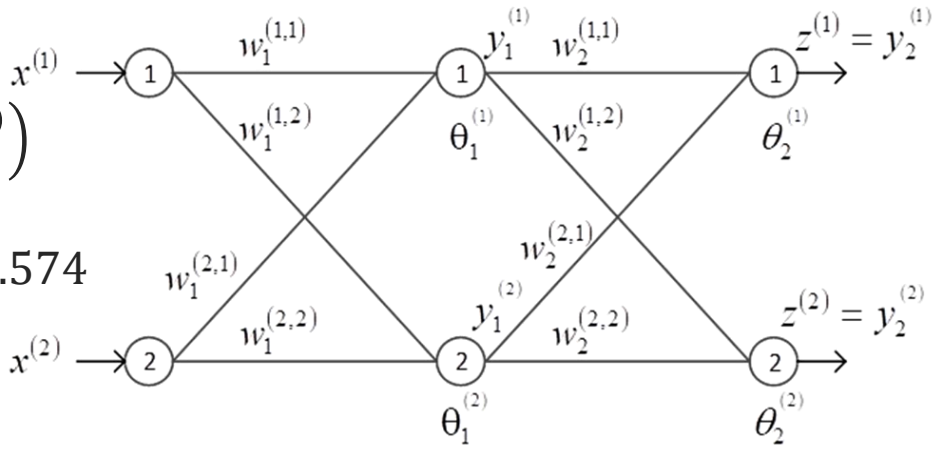


误差反向传播

前向传播预测过程

$$y_1^{(2)} = g\left(w_1^{(1,2)}x^{(1)} + w_1^{(2,2)}x^{(2)} + \theta_1^{(2)}\right)$$

$$= \frac{1}{1 + e^{-\left(w_1^{(1,2)}x^{(1)} + w_1^{(2,2)}x^{(2)} + \theta_1^{(2)}\right)}} = 0.574$$



同样计算第2隐层，也就是输出层的输出：

$$z^{(1)} = y_2^{(1)} = g\left(w_2^{(1,1)}y_1^{(1)} + w_2^{(2,1)}y_1^{(2)} + \theta_2^{(1)}\right)$$

$$= \frac{1}{1 + e^{-\left(w_2^{(1,1)}y_1^{(1)} + w_2^{(2,1)}y_1^{(2)} + \theta_2^{(1)}\right)}} = 0.743$$

$$\boldsymbol{W}_1 = \begin{bmatrix} 0.1 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}, \quad \boldsymbol{\theta_1} = [0.3 \quad 0.3]$$

$$\boldsymbol{W}_2 = \begin{bmatrix} 0.4 & 0.5 \\ 0.4 & 0.5 \end{bmatrix}, \quad \boldsymbol{\theta_2} = [0.6 \quad 0.6]$$

$$z^{(2)} = y_2^{(2)} = g\left(w_2^{(1,2)}y_1^{(1)} + w_2^{(2,2)}y_1^{(2)} + \theta_2^{(2)}\right)$$

$$= \frac{1}{1 + e^{-\left(w_2^{(1,2)}y_1^{(1)} + w_2^{(2,2)}y_1^{(2)} + \theta_2^{(2)}\right)}} = 0.764$$

误差反向传播

反向传播学习过程

用$l^{(1)}$和$l^{(2)}$表示标签值，采用各标签值的误差平方和均值作为总误差，并将总误差依次展开至输入层：

$$E=\frac{1}{2}\sum_{i=1}^{2}\left(z^{(i)}-l^{(i)}\right)^2=\frac{1}{2}\sum_{i=1}^{2}\left(g\left(w_2^{(1,i)}y_1^{(1)}+w_2^{(2,i)}y_1^{(2)}+\theta_2^{(i)}\right)-l^{(i)}\right)^2$$

$$=\frac{1}{2}\sum_{i=1}^{2}\left(g\left(\begin{array}{c}w_2^{(1,i)}g\left(w_1^{(1,1)}x^{(1)}+w_1^{(2,1)}x^{(2)}+\theta_1^{(1)}\right)+\\ w_2^{(2,i)}g\left(w_1^{(1,2)}x^{(1)}+w_1^{(2,2)}x^{(2)}+\theta_1^{(2)}\right)+\theta_2^{(i)}\end{array}\right)-l^{(i)}\right)^2$$

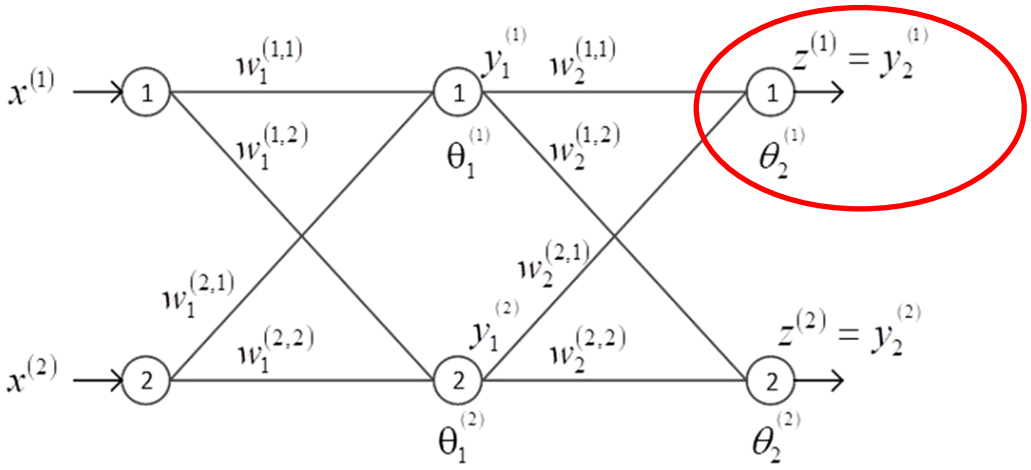
可见，总误差E是各层参数变量的函数，因此学习的**目的就是通过调整各参数变量的值，使E最小。**可采用**梯度下降**法求解。

误差反向传播

反向传播学习过程

第一个训练样本的标签值为(0,1)，因此，输出产生的误差分别为0.743和−0.236。总误差：$E=\frac{1}{2}\sum_{i=1}^{2}\left(z^{(i)}-l^{(i)}\right)^2=0.304$

输出层节点的参数更新，以节点1的$w_2^{(1,1)}$和$\theta_2^{(1)}$为例详细讨论。

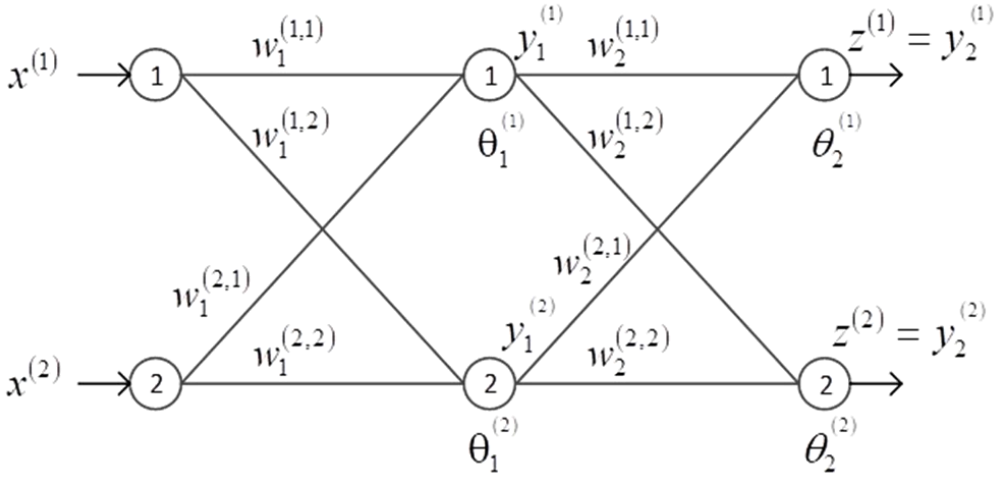


误差反向传播

反向传播学习过程

先求偏导 $\frac{\partial E}{\partial w_2^{(1,1)}}$：$\frac{\partial E}{\partial w_2^{(1,1)}} = \frac{\partial E}{\partial y_2^{(1)}} \cdot \frac{\partial y_2^{(1)}}{\partial w_2^{(1,1)}} = \frac{\partial \left[\frac{1}{2}\sum_{i=1}^{2}\left(y_2^{(i)} - l^{(i)}\right)^2\right]}{\partial y_2^{(1)}} \cdot \frac{\partial y_2^{(1)}}{\partial w_2^{(1,1)}} = \left(y_2^{(1)} - l^{(1)}\right) \cdot \frac{\partial y_2^{(1)}}{\partial w_2^{(1,1)}}$

式中括号$\left(y_2^{(1)} - l^{(1)}\right)$是输出层节点1的**校对误差，记为$E_2^1$**，即$E_2^1 = y_2^{(1)} - l^{(1)} = 0.743$。因此$\frac{\partial E}{\partial w_2^{(1,1)}}$可视为该节点的校对误差乘以该节点输出对待更新参数变量的偏导：$\boldsymbol{\frac{\partial E}{\partial w_2^{(1,1)}} = E_2^1 \cdot \frac{\partial y_2^{(1)}}{\partial w_2^{(1,1)}}}$



误差反向传播

反向传播学习过程

设梯度下降法中的步长α为0.5，$w_2^{(1,1)}$更新为：

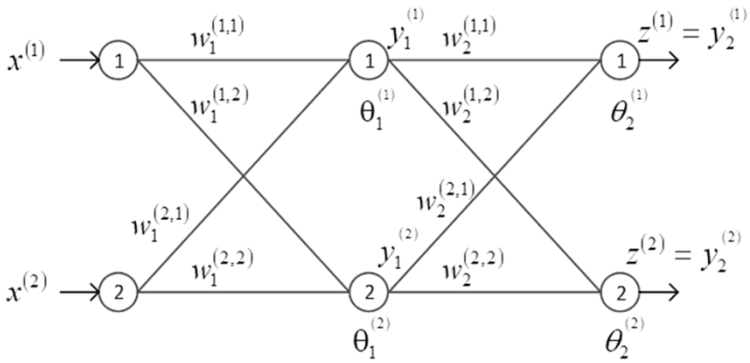
$$w_2^{(1,1)} \leftarrow w_2^{(1,1)} - \alpha E_2^1 \cdot \frac{\partial y_2^{(1)}}{\partial w_2^{(1,1)}}$$

式中偏导数$\frac{\partial y_2^{(1)}}{\partial w_2^{(1,1)}}$的计算为：

$$\frac{\partial y_2^{(1)}}{\partial w_2^{(1,1)}} = y_2^{(1)} \cdot \left(1 - y_2^{(1)}\right) \cdot \frac{\partial \left(w_2^{(1,1)} y_1^{(1)} + w_2^{(2,1)} y_1^{(2)} + \theta_2^{(1)}\right)}{\partial w_2^{(1,1)}}$$

$$= y_2^{(1)} \cdot \left(1 - y_2^{(1)}\right) \cdot y_1^{(1)}$$

Sigmoid函数求导



误差反向传播

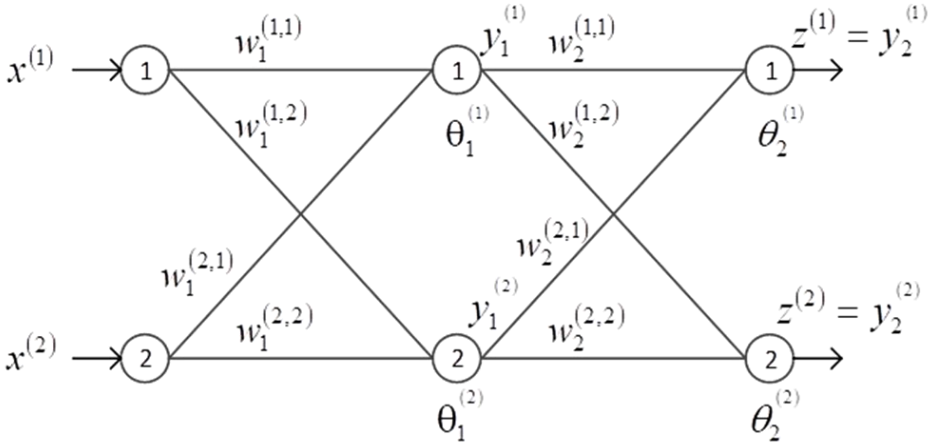
反向传播学习过程

因此：

$$w_2^{(1,1)} \leftarrow w_2^{(1,1)} - \alpha E_2^1 \cdot \frac{\partial y_2^{(1)}}{\partial w_2^{(1,1)}} = w_2^{(1,1)} - \alpha E_2^1 \cdot y_2^{(1)} \cdot \left(1 - y_2^{(1)}\right) \cdot y_1^{(1)}$$

$$= 0.4 - 0.5 \times 0.743 \times 0.743 \times (1 - 0.743) \times 0.574 = 0.359$$

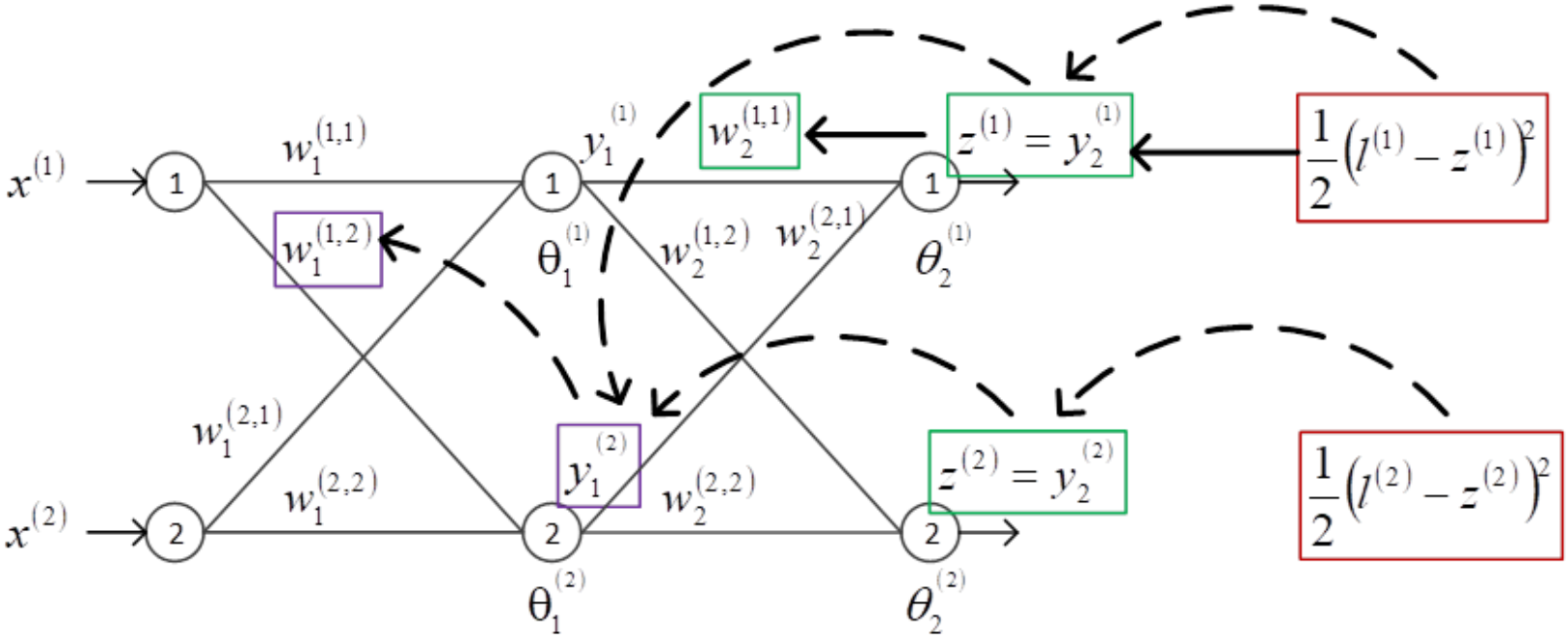
同样，可得$w_2^{(1,2)}$、$w_2^{(2,1)}$和$w_2^{(2,1)}$的更新值分别为：0.512、0.359和0.512。



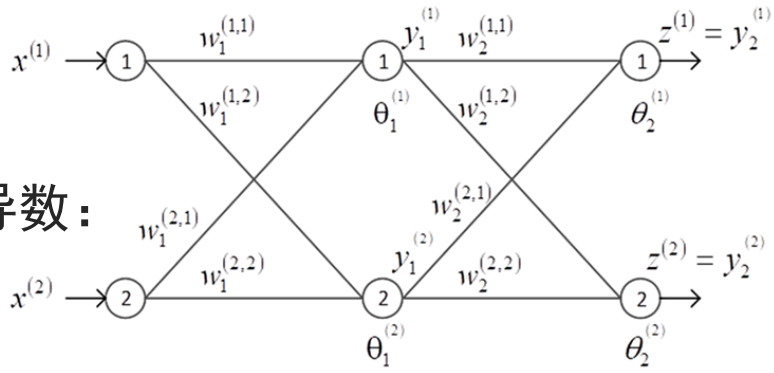
误差反向传播

反向传播学习过程

$\frac{\partial E}{\partial w_2^{(1,1)}}$的求导路径如图中**粗实线**所示。



误差反向传播



反向传播学习过程

对于$\theta_2^{(1)}$的更新，先求总误差对它的偏导数：

$$\frac{\partial E}{\partial \theta_2^{(1)}} = \frac{\partial E}{\partial y_2^{(1)}} \cdot \frac{\partial y_2^{(1)}}{\partial \theta_2^{(1)}} = E_2^1 \cdot \frac{\partial y_2^{(1)}}{\partial \theta_2^{(1)}}$$

$$= E_2^1 \cdot y_2^{(1)} \cdot \left(1 - y_2^{(1)}\right) \cdot \frac{\partial \left(w_2^{(1,1)} y_1^{(1)} + w_2^{(2,1)} y_1^{(2)} + \theta_2^{(1)}\right)}{\partial \theta_2^{(1)}}$$

$$= E_2^1 \cdot y_2^{(1)} \cdot \left(1 - y_2^{(1)}\right)$$

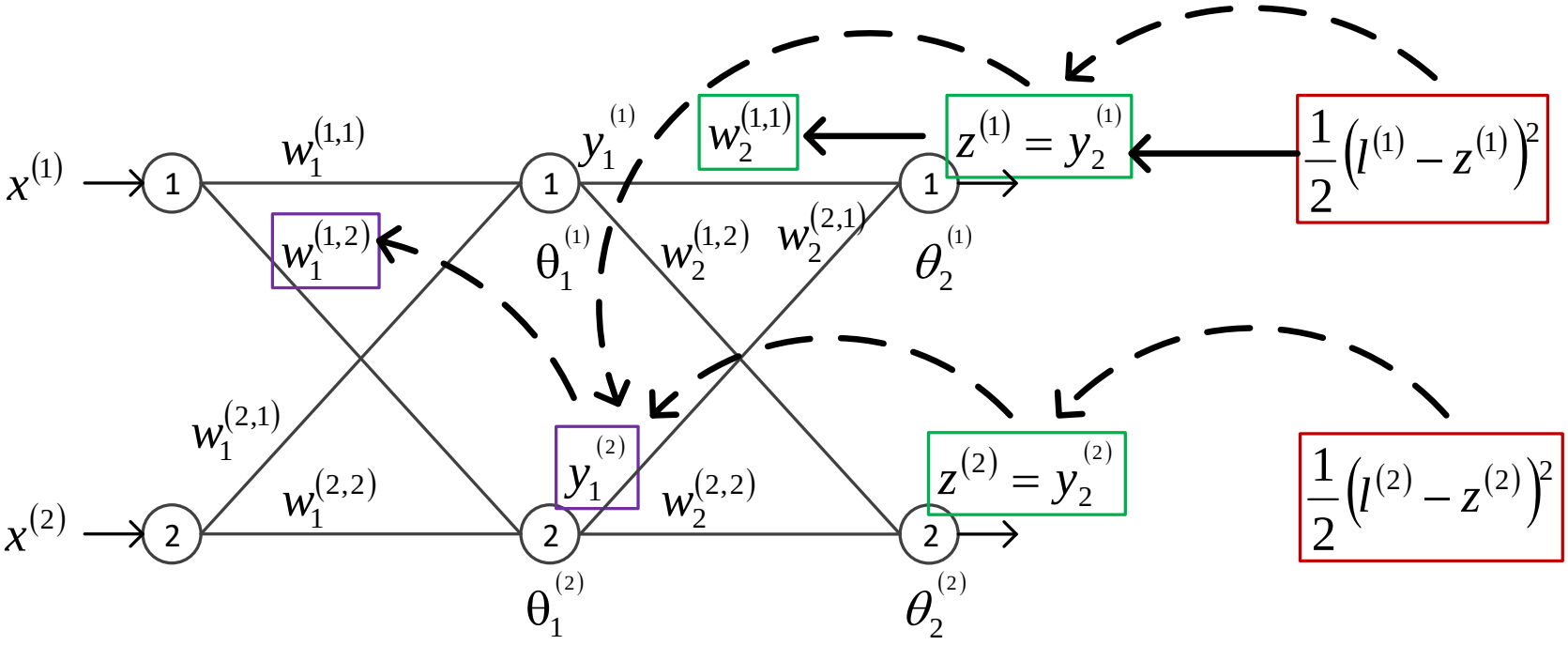
因此$\frac{\partial E}{\partial \theta_2^{(1)}}$可视为该节点的校对误差乘以该节点输出对待更新阈值变量的偏导。$\theta_2^{(1)}$的更新为：

$$\theta_2^{(1)} \leftarrow \theta_2^{(1)} - \alpha \frac{\partial E}{\partial \theta_2^{(1)}} = 0.529$$

误差反向传播

反向传播学习过程

第1隐层的参数更新，以节点2的$w_1^{(1,2)}$为例详细讨论。对$w_1^{(1,2)}$的求导有两条路径，粗虚线所示。



误差反向传播

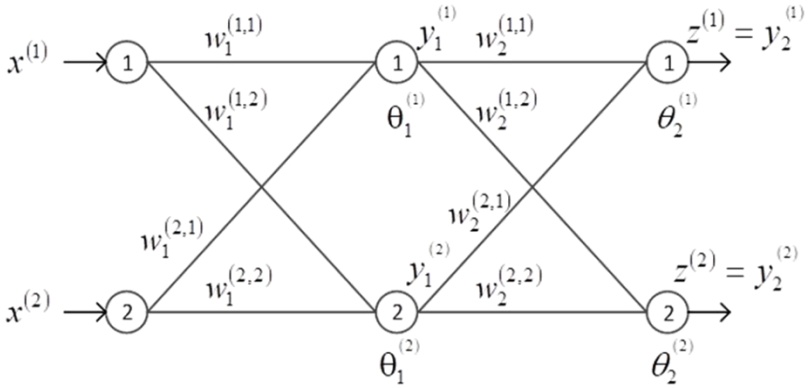
反向传播学习过程

$$\frac{\partial E}{\partial w_1^{(1,2)}} = \frac{\partial E}{\partial y_2^{(1)}} \cdot \frac{\partial y_2^{(1)}}{\partial w_1^{(1,2)}} + \frac{\partial E}{\partial y_2^{(2)}} \cdot \frac{\partial y_2^{(2)}}{\partial w_1^{(1,2)}}$$

$$= \frac{\partial E}{\partial y_2^{(1)}} \cdot \frac{\partial y_2^{(1)}}{\partial y_1^{(2)}} \cdot \frac{\partial y_1^{(2)}}{\partial w_1^{(1,2)}} + \frac{\partial E}{\partial y_2^{(2)}} \cdot \frac{\partial y_2^{(2)}}{\partial y_1^{(2)}} \cdot \frac{\partial y_1^{(2)}}{\partial w_1^{(1,2)}}$$

$$= \left(\frac{\partial E}{\partial y_2^{(1)}} \cdot \frac{\partial y_2^{(1)}}{\partial y_1^{(2)}} + \frac{\partial E}{\partial y_2^{(2)}} \cdot \frac{\partial y_2^{(2)}}{\partial y_1^{(2)}} \right) \cdot \frac{\partial y_1^{(2)}}{\partial w_1^{(1,2)}}$$

$$= \left(E_2^1 \cdot \frac{\partial y_2^{(1)}}{\partial y_1^{(2)}} + E_2^2 \cdot \frac{\partial y_2^{(2)}}{\partial y_1^{(2)}} \right) \cdot \frac{\partial y_1^{(2)}}{\partial w_1^{(1,2)}}$$

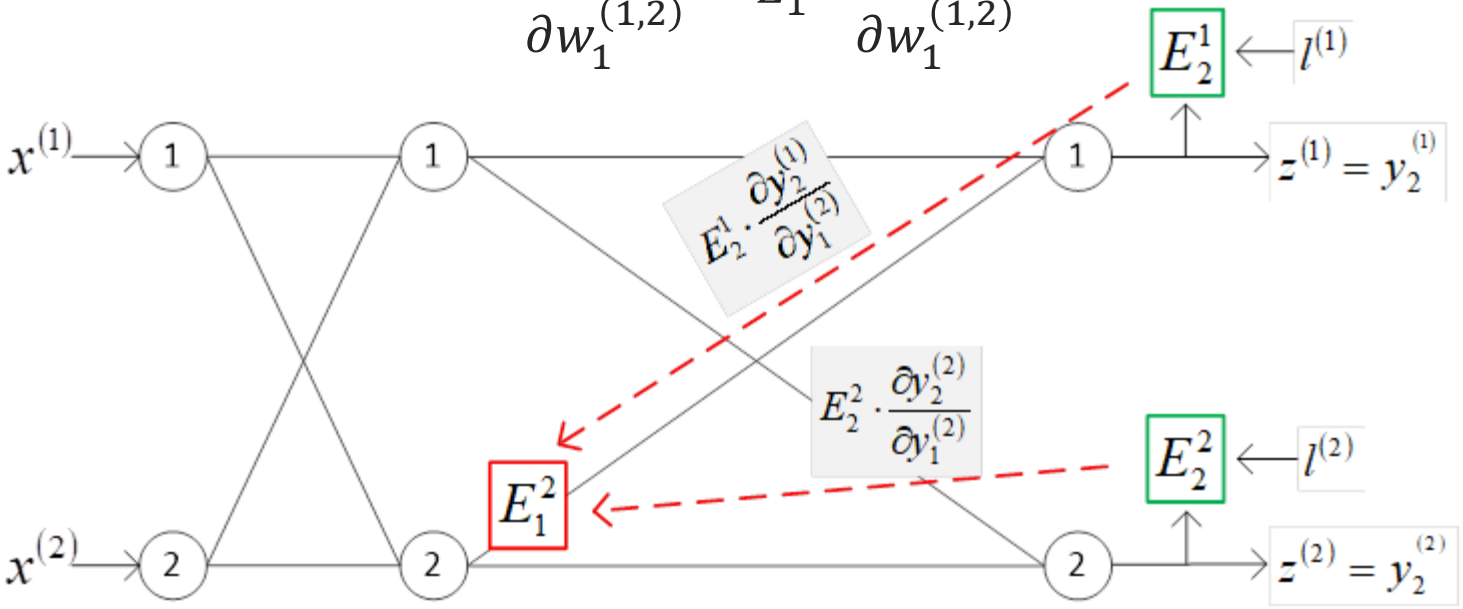


误差反向传播

反向传播学习过程

$E_2^2 = \left(y_2^{(2)} - l^{(2)}\right)$，是输出层节点2的校对误差。可将$\left(E_2^1 \cdot \frac{\partial y_2^{(1)}}{\partial y_1^{(2)}} + E_2^2 \cdot \frac{\partial y_2^{(2)}}{\partial y_1^{(2)}}\right)$视为校对误差$E_2^1$和$E_2^2$沿求导路径反向传播到第1隐层节点2的校对误差，将该校对误差记为E_1^2。

$$\frac{\partial E}{\partial w_1^{(1,2)}} = E_1^2 \cdot \frac{\partial y_1^{(2)}}{\partial w_1^{(1,2)}}$$



误差反向传播

反向传播学习过程

$$\frac{\partial E}{\partial w_1^{(1,2)}} = E_1^2 \cdot \frac{\partial y_1^{(2)}}{\partial w_1^{(1,2)}}$$

$$E_1^2 = E_2^1 \cdot \frac{\partial y_2^{(1)}}{\partial y_1^{(2)}} + E_2^2 \cdot \frac{\partial y_2^{(2)}}{\partial y_1^{(2)}}$$

$$= E_2^1 \cdot y_2^{(1)}\left(1 - y_2^{(1)}\right) w_2^{(2,1)} + E_2^2 \cdot y_2^{(2)}\left(1 - y_2^{(2)}\right) w_2^{(2,2)}$$

$$\frac{\partial y_1^{(2)}}{\partial w_1^{(1,2)}} = y_1^{(2)}\left(1 - y_1^{(2)}\right) \frac{\partial\left(w_1^{(1,2)} x^{(1)} + w_1^{(2,2)} x^{(2)} + \theta_1^{(2)}\right)}{\partial w_1^{(1,2)}}$$

$$= y_1^{(2)}\left(1 - y_1^{(2)}\right) x^{(1)} = 0$$

因此，$\frac{\partial E}{\partial w_1^{(1,2)}} = 0$。$w_1^{(1,2)}$不变：

$$w_1^{(1,2)} \leftarrow w_1^{(1,2)} - \alpha \frac{\partial E}{\partial w_1^{(1,2)}} = w_1^{(1,2)} = 0.1$$

误差反向传播

示例代码

```
for j in range(2000): # 训练轮数
    E = 0.0
    for i in range(4):
        print(i)
        print(XX[i])
        print(L[i])
        ### 前向传播预测
        # 计算第 1 隐层的输出
        Y1[0] = y_1_1(W1, theta1, XX[i])
        Y1[1] = y_1_2(W1, theta1, XX[i])

        # 计算第 2 隐层的输出
        Y2[0] = y_2_1(W2, theta2, Y1)
        Y2[1] = y_2_2(W2, theta2, Y1)
        print(Y2)

        ### 后向传播误差
        # 计算第 2 隐层的误差
        E2[0] = Y2[0] - L[i][0]
        E2[1] = Y2[1] - L[i][1]
        E += 0.5*(E2[0]*E2[0]+E2[1]*E2[1])
```

```
 # 计算第 1 隐层的误差
        E1[0] = E2[0]*Y2[0]*(1 - Y2[0])*W2[0,0] + E2[1]*Y2[1]*(1 - Y2[1])*W2[0,1]
        E1[1] = E2[0]*Y2[0]*(1 - Y2[0])*W2[1,0] + E2[1]*Y2[1]*(1 - Y2[1])*W2[1,1]

        ### 更新系数
        # 更新第 2 隐层的参数
        W2[0,0] = W2[0,0] - a*E2[0]*Y2[0]*(1 - Y2[0])*Y1[0]
        W2[1,0] = W2[1,0] - a*E2[0]*Y2[0]*(1 - Y2[0])*Y1[1]
        theta2[0] = theta2[0] - a*E2[0]*Y2[0]*(1 - Y2[0])
        W2[0,1] = W2[0,1] - a*E2[1]*Y2[1]*(1 - Y2[1])*Y1[0]
        W2[1,1] = W2[1,1] - a*E2[1]*Y2[1]*(1 - Y2[1])*Y1[1]
        theta2[1] = theta2[1] - a*E2[1]*Y2[1]*(1 - Y2[1])

        # 更新第 1 隐层的参数
        W1[0,0] = W1[0,0] - a*E1[0]*Y1[0]*(1 - Y1[0])*XX[i][0]
        W1[1,0] = W1[1,0] - a*E1[0]*Y1[0]*(1 - Y1[0])*XX[i][1]
        theta1[0] = theta1[0] - a*E1[0]*Y1[0]*(1 - Y1[0])
        W1[0,1] = W1[0,1] - a*E1[1]*Y1[1]*(1 - Y1[1])*XX[i][0]
        W1[1,1] = W1[1,1] - a*E1[1]*Y1[1]*(1 - Y1[1])*XX[i][1]
        theta1[1] = theta1[1] - a*E1[1]*Y1[1]*(1 - Y1[1])
    print("平均总误差" + str(E/4.0))
```

BP算法

前向传播预测

设BP神经网络共有M + 1层，包括输入层和M个隐层（第M个隐层为输出层）。网络输入分量个数为U，输出分量个数为V。设神经元采用的激励函数为$f(x)$。设训练样本为$(\boldsymbol{x}, \boldsymbol{l})$，实例向量$\boldsymbol{x} = \left(x^{(1)}, x^{(2)}, \dots, x^{(U)}\right)$，标签向量$\boldsymbol{l} = \left(l^{(1)}, l^{(2)}, \dots, l^{(V)}\right)$。

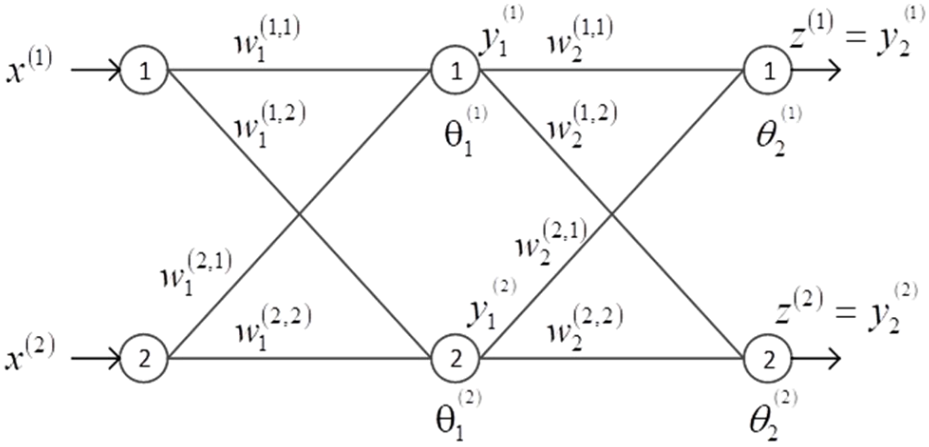
设第1隐层共有n_1个节点，它们的输出记为$\boldsymbol{y_1} = \left[y_1^{(1)}, y_1^{(2)}, \dots, y_1^{(n_1)}\right]$，它们的阈值系数记为$\boldsymbol{\theta_1} = \left[\theta_1^{(1)}, \theta_1^{(2)}, \dots, \theta_1^{(n_1)}\right]$，从输入层到该隐层的连接系数记为$\boldsymbol{W}_1 = \begin{bmatrix} w_1^{(1,1)} & \cdots & w_1^{(1,n_1)} \\ \vdots & \ddots & \vdots \\ w_1^{(U,1)} & \cdots & w_1^{(U,n_1)} \end{bmatrix}$。可得：$\boldsymbol{y_1} = f(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{\theta_1})$

BP算法

前向传播预测

设第2隐层共有n_2个节点，它们的输出记为$\boldsymbol{y_2} = \left[y_2^{(1)}, y_2^{(2)}, \ldots, y_2^{(n_2)}\right]$，它们的阈值系数记为$\boldsymbol{\theta_2} = \left[\theta_2^{(1)}, \theta_2^{(2)}, \ldots, \theta_2^{(n_2)}\right]$，从第1隐层到该隐层的连接系数记为$\boldsymbol{W}_2 = \begin{bmatrix} w_2^{(1,1)} & \cdots & w_2^{(1,n_2)} \\ \vdots & \ddots & \vdots \\ w_2^{(n_1,1)} & \cdots & w_2^{(n_1,n_2)} \end{bmatrix}$。可得：$\boldsymbol{y_2} = f(\boldsymbol{y_1}\boldsymbol{W}_2 + \boldsymbol{\theta_2})$

依次可前向计算各层输出，直到输出层。输出为$\boldsymbol{z} = \left(z^{(1)}, z^{(2)}, \ldots, z^{(V)}\right)$。



BP算法

反向传播学习

输出层的校对误差记为$\boldsymbol{E}_M$：

$$\boldsymbol{E}_M = \left(E_M^1, E_M^2, \ldots, E_M^V\right) = \boldsymbol{z} - \boldsymbol{l}$$

第$M-1$层的校对误差记为$\boldsymbol{E}_{M-1}$：

$$\boldsymbol{E}_{M-1} = \boldsymbol{E}_M \begin{bmatrix} \dfrac{\partial y_M^{(1)}}{\partial y_{M-1}^{(1)}} & \cdots & \dfrac{\partial y_M^{(1)}}{\partial y_{M-1}^{(n_{M-1})}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_M^{(V)}}{\partial y_{M-1}^{(1)}} & \cdots & \dfrac{\partial y_M^{(V)}}{\partial y_{M-1}^{(n_{M-1})}} \end{bmatrix}$$

式中右侧的矩阵是第M层输出对第$M-1$层输出的偏导数排列的矩阵，称为雅可比矩阵，其中的n_{M-1}是第$M-1$层的节点数。

依次可反向计算各层的校对误差，直到第1隐层。

BP算法

反向传播学习

接下来，根据校对误差更新连接系数和阈值系数。对第i隐层的第j节点的第k个连接系数$w_i^{(k,j)}$：

$$w_i^{(k,j)} \leftarrow w_i^{(k,j)} - \alpha \cdot E_i^j \cdot \frac{\partial y_i^{(j)}}{\partial w_i^{(k,j)}}$$

其中$\frac{\partial y_i^{(j)}}{\partial w_i^{(k,j)}}$的计算为：

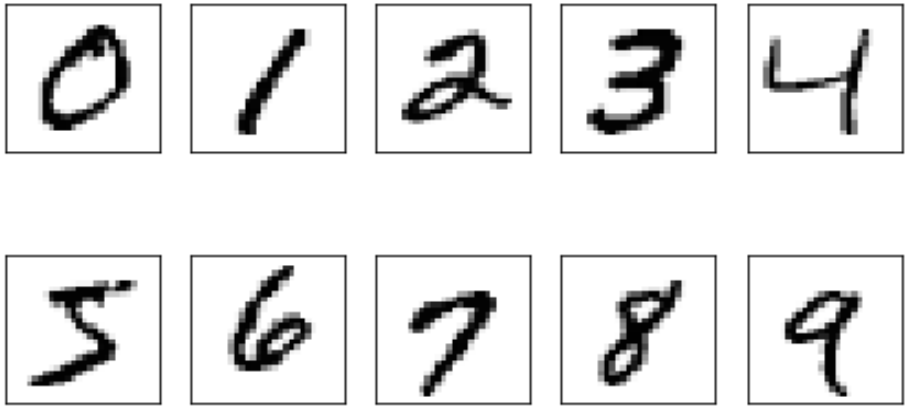
$$\frac{\partial y_i^{(j)}}{\partial w_i^{(k,j)}} = \frac{\partial y_i^{(j)}}{\partial \left(\boldsymbol{y_{i-1}} \times \boldsymbol{W}_i|_j + \theta_i^{(j)} \right)} \cdot \frac{\partial \left(\boldsymbol{y_{i-1}} \times \boldsymbol{W}_i|_j + \theta_i^{(j)} \right)}{\partial w_i^{(k,j)}}$$

$$= f'(x)\Big|_{x=\boldsymbol{y_{i-1}} \times \boldsymbol{W}_i|_j + \theta_i^{(j)}} \cdot y_{i-1}^{(k)}$$

BP算法

MNIST数据集

MNIST数据集是一个手写体的数字图片集，来自美国国家标准与技术研究所。它包含训练集和测试集，由250个人手写的数字构成。训练集包含60000个样本，训练集包含10000个样本。每个样本包括一张图片和一个标签。每张图片由28×28个像素点构成,每个像素点用1个灰度值表示。标签是与图片对应的0到9的数字。



BP算法

手写体数字识别示例

```
(X_train, y_train), (X_test, y_test) = ka.datasets.mnist.load_data()

num_pixels = X_train.shape[1] * X_train.shape[2] # 784

# 将二维的数组拉成一维的向量
X_train = X_train.reshape(X_train.shape[0],
num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')

X_train = X_train / 255
X_test = X_test / 255

y_train = ka.utils.to_categorical(y_train) # 转化为独热编码
y_test = ka.utils.to_categorical(y_test)
num_classes = y_test.shape[1] # 10
```

BP算法

手写体数字识别示例

```
# 多层全连接神经网络模型
model = ka.Sequential([
    ka.layers.Dense(num_pixels, input_shape=(num_pixels,),
                    kernel_initializer='normal', activation='relu'),
    ka.layers.Dense(784, kernel_initializer='normal',
activation='relu'),
    ka.layers.Dense(num_classes, kernel_initializer='normal',
                    activation='softmax')
])
model.summary()

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

startdate = datetime.datetime.now() # 获取当前时间
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20,
batch_size=200, verbose=2)
enddate = datetime.datetime.now()
```

BP算法

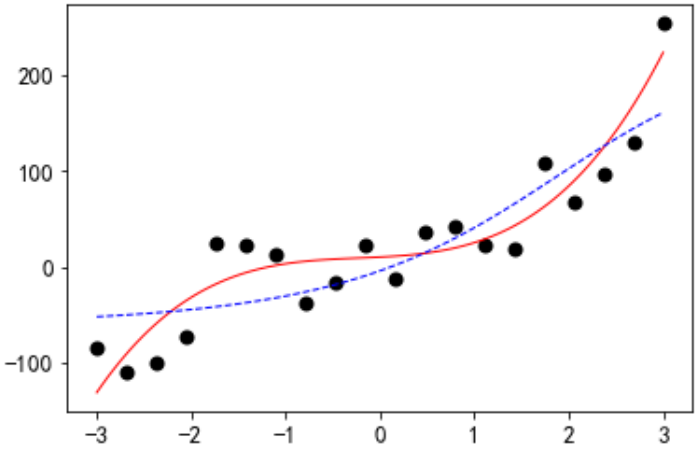
拟合三次多项式示例

用神经网络作模型来拟合三次多项式。采用Tensorflow2.0框架来实现。

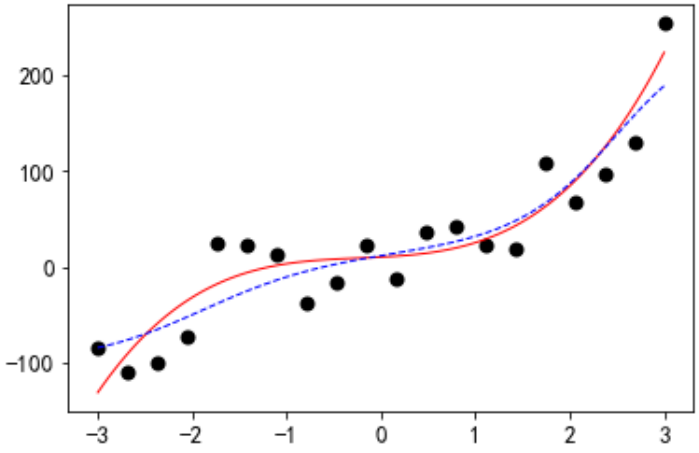
```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation='sigmoid', input_shape=(1,),
                          kernel_initializer='random_uniform',
bias_initializer='zeros'),
    tf.keras.layers.Dense(5, activation='sigmoid',
                          kernel_initializer='random_uniform',
bias_initializer='zeros'),
    tf.keras.layers.Dense(1, activation='sigmoid',
                          kernel_initializer='random_uniform',
bias_initializer='zeros')
])
```

BP算法

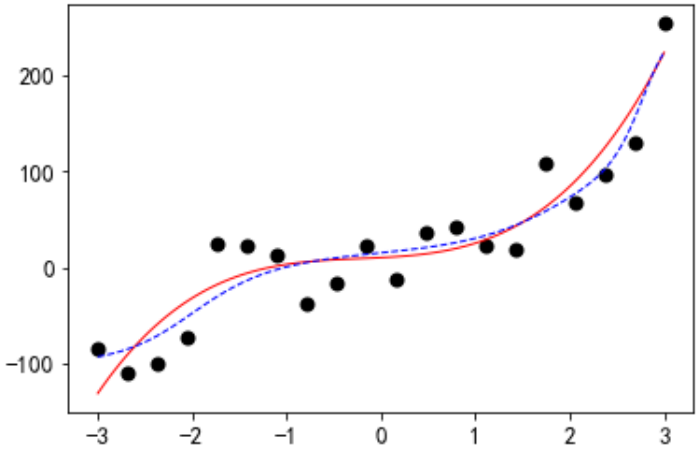
不同网络结构拟合结果



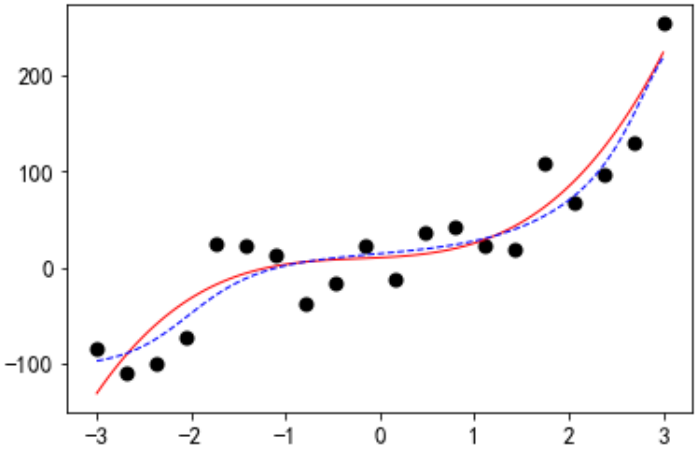
（1）三层（1，1，1）结构拟合结果



（2）三层（1，2，1）结构拟合结果



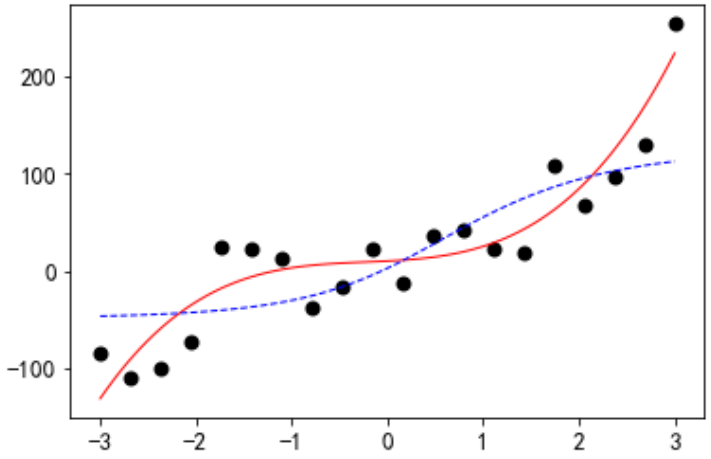
（3）四层（1，5，5，1）结构拟合结果



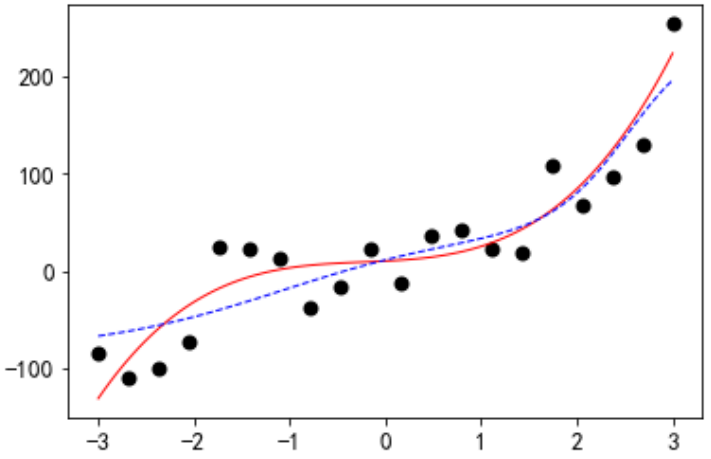
（4）四层（1，10，15，1）结构拟合结果

BP算法

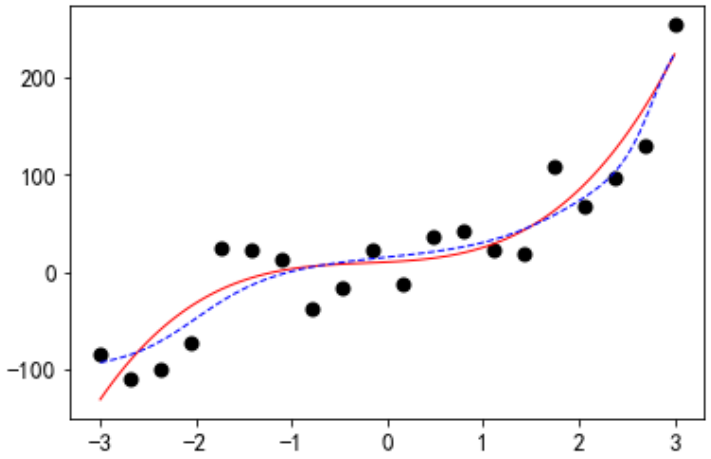
不同训练轮数拟合结果



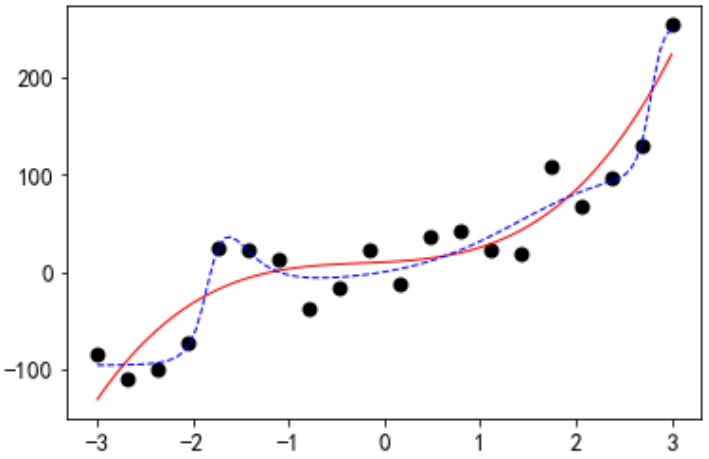
（1）训练 1000 轮的拟合结果



（2）训练 3000 轮的拟合结果



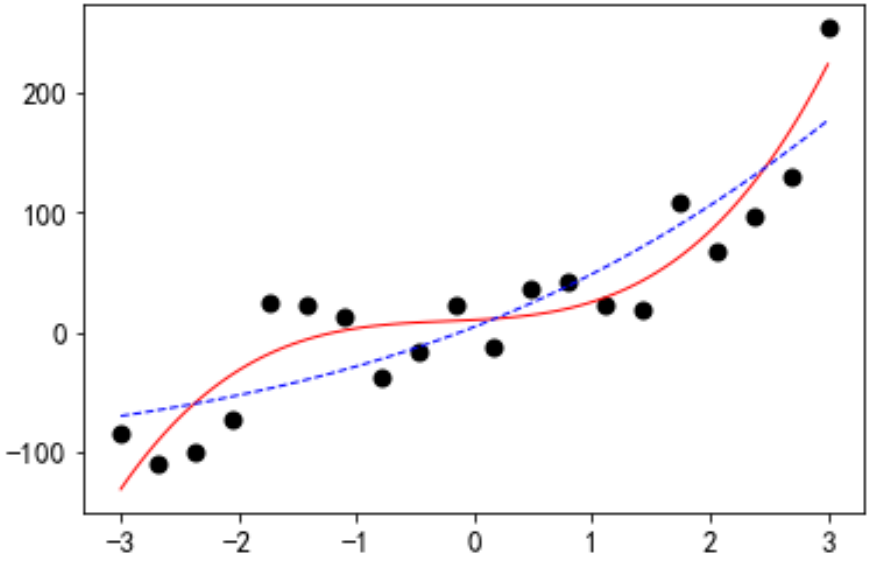
（3）训练 5000 轮的拟合结果



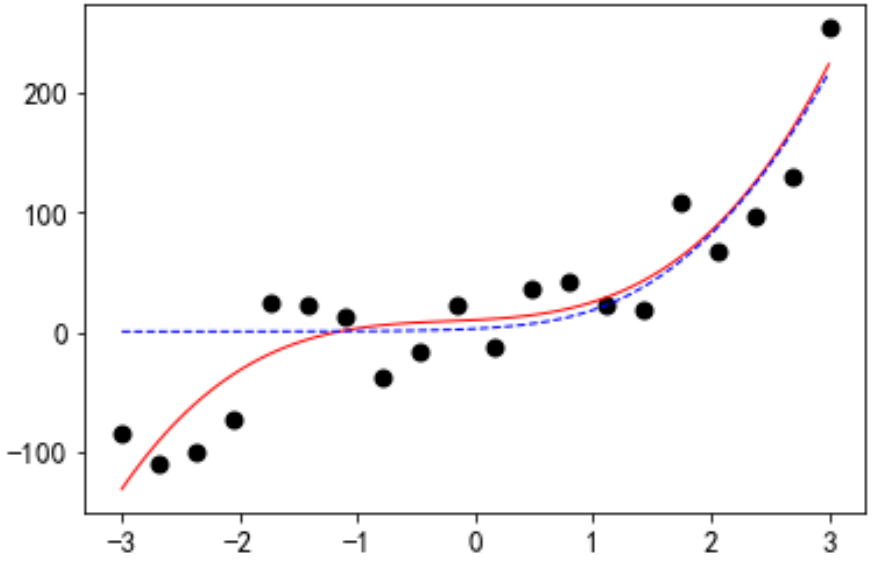
（4）训练 10000 轮的拟合结果

BP算法

采用softplus激活函数的拟合结果



（1）训练样本归一化处理



（2）训练样本不归一化处理

多层神经网络常用损失函数

1. 相对熵损失函数和交叉熵损失函数

交叉熵可以用来衡量两个分布之间的差距。

a）[0.07158904 0.92822515] -> [0. 1.]

b）[0.9138734   0.08633152] -> [1. 0.]

c）[0.91375259 0.08644981] -> [1. 0.]

d）[0.11774177 0.88200493] -> [0. 1.]

问题：a和d哪个更准确呢？

信息熵的定义：$\mathrm{H}(X)=-\sum_{i=1}^{n} p_i \log p_i$。用$p_i$表示第$i$个输出的标签值，即真实值，用$q_i$表示第$i$个输出值，即预测值。将$p_i$与$q_i$之间的对数差在$p_i$上的期望值称为相对熵：

$$D_{KL}(p\|q)=E_{p_i}(\log p_i-\log q_i)=\sum_{\mathrm{i}=1}^{n} p_i(\log p_i-\log q_i)=\sum_{\mathrm{i}=1}^{n} p_i \log\frac{p_i}{q_i}$$

多层神经网络常用损失函数

1. 相对熵损失函数和交叉熵损失函数

计算a和d两项输出的相对熵：

$$D_a = 0 \times \log\frac{0}{0.07158904} + 1 \times \log\frac{1}{0.92822515} = 0.07447962$$

$$D_d = 0 \times \log\frac{0}{0.11774177} + 1 \times \log\frac{1}{0.88200493} = 0.12555622$$

相对熵越大，输出与标签差距越大

将相对熵的定义式进一步展开：

$$D_{KL}(p\|q) = \sum_{i=1}^{n} p_i(\log p_i - \log q_i)$$

$$= \sum_{i=1}^{n} p_i \log p_i + \left[-\sum_{i=1}^{n} p_i \log q_i\right] = -H(p_i) + \left[-\sum_{i=1}^{n} p_i \log q_i\right]$$

前一项保持不变，因此一般用后一项作为两个分布之间差异的度量，称为交叉熵：$H(p,q) = -\sum_{i=1}^{n} p_i \log q_i$

多层神经网络常用损失函数

1. 相对熵损失函数和交叉熵损失函数

如果只有正负两个分类，记标签为正类的概率为y，记预测为正类的概率为p，那么上式为：

$$H(y,p) = -[y\log p + (1-y)\log(1-p)]$$

在讨论逻辑回归算法时，实际上已经应用了二分类的交叉熵损失函数。

交叉熵损失函数在梯度下降法中可以避免MSE学习速率降低的问题，得到了广泛的应用。

多层神经网络常用损失函数

2. 余弦相似度损失函数

将标签和预测看作值向量，可用

$$\cos\theta = \frac{\boldsymbol{x}_i \cdot \boldsymbol{x}_j}{\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\|} = \frac{\sum_{l=1}^{n} {x_i}^{(l)} {x_j}^{(l)}}{\sqrt{\sum_{l=1}^{n} ({x_i}^{(l)})^2} \sqrt{\sum_{l=1}^{n} \left({x_j}^{(l)}\right)^2}}$$

计算得到余弦相似度作为损失函数。

3. 双曲余弦对数损失函数

双曲余弦对数的计算方法为：

$$\operatorname{logcosh}(p, q) = \sum_{i=1}^{n} \log \frac{e^{q_i - p_i} + e^{-(q_i - p_i)}}{2}$$

双曲余弦对数损失函数相似于MSE，但比MSE相对稳定。

多层神经网络常用优化算法

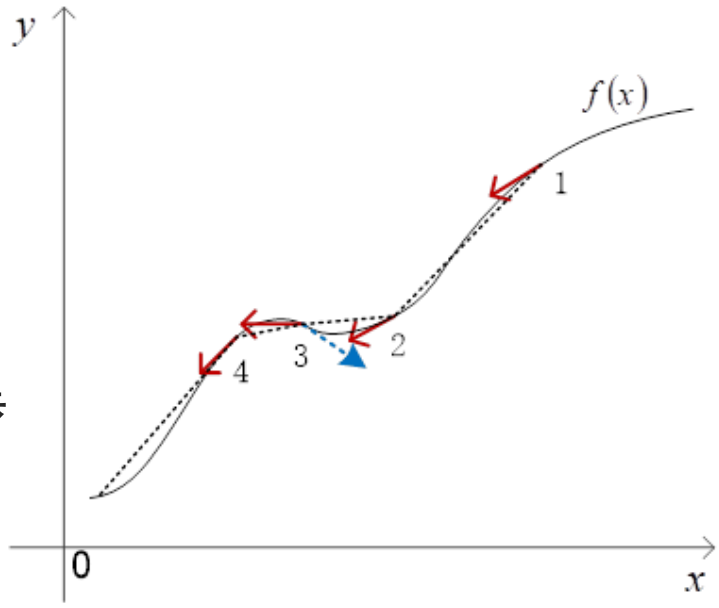
1. 动量优化算法

在经典力学中，动量体现为物体运动的惯性。

加入动量的梯度下降法迭代关系式为：

$$\theta_{i+1} = \beta\theta_i - \alpha \cdot \frac{df(x)}{dx}\Big|_{x=x_i},$$

$$x_{i+1} = x_i + \theta_{i+1}$$



可见，加入动量之后的前进量θ_{i+1}是由上一步的前进量θ_i和新梯度值$\frac{df(x)}{dx}\Big|_{x=x_i}$的加权和，其中$\beta$决定了保留上一步前进量的大小，称为动量系数。初始$\theta_0 = 0$。

多层神经网络常用优化算法

1. 动量优化算法

加入动量的梯度下降的迭代关系式还有一种改进方法，称为NAG（Nesterov accelerated gradient）：

$$\theta_{i+1} = \beta\theta_i - \alpha \cdot \frac{df(\boldsymbol{x})}{d\boldsymbol{x}}\Big|_{\boldsymbol{x}=\boldsymbol{x}_i+\beta\theta_i}$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \theta_{i+1}$$

该方法计算梯度的点发生了变化，即在$\boldsymbol{x}_i + \beta\theta_i$处计算梯度，而不是原$\boldsymbol{x}_i$点处。它可以理解为先按“惯性”前进一小步$\beta\theta_i$，再计算梯度。这种方法在每一步都往前多走了一小步，有时可以加快收敛速度。

多层神经网络常用优化算法

2. 步长优化算法

Adagrad（Adaptive Gradient）算法的梯度迭代关系式为：

$$r_i = r_{i-1} + \left(\left. \frac{df(\boldsymbol{x})}{d\boldsymbol{x}} \right|_{x=x_i} \right)^2$$

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} - \frac{lr}{\sqrt{r_i} + \epsilon} \cdot \left. \frac{df(\boldsymbol{x})}{d\boldsymbol{x}} \right|_{x=x_i}$$

其中，r_i称为累积平方梯度，它是到当前为止所有梯度的平方和，初值为0。这里的$\left(\left. \frac{df(x)}{dx} \right|_{x=x_i} \right)^2$表示向量元素的平方向量：$\left. \frac{df(x)}{dx} \right|_{x=x_i} \odot \left. \frac{df(x)}{dx} \right|_{x=x_i}$，下同。超参数$lr$为事先设置的步长，$\epsilon$为很小的常数。初始$r_0 = 0$。

可见，随着迭代次数的增加，r_i越来越大，实际步长$\frac{lr}{\sqrt{r_{i+1}}+\epsilon}$越来越小，可以防止在极小值附近来回振荡。

多层神经网络常用优化算法

2. 步长优化算法

RMSProp（Root Mean Square Prop）算法的原始形式对Adagrad算法进行了简单改进，增加了一个系数rho来控制历史信息与当前梯度的比例：

$$r_i = rho \cdot r_{i-1} + (1 - rho)\left(\left.\frac{df(\boldsymbol{x})}{d\boldsymbol{x}}\right|_{x=x_i}\right)^2$$

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} - \frac{lr}{\sqrt{r_i} + \epsilon} \cdot \left.\frac{df(\boldsymbol{x})}{d\boldsymbol{x}}\right|_{x=x_i}$$

多层神经网络常用优化算法

2. 步长优化算法

对RMSProp的原始形式增加动量因子如下：

$$r_i = rho \cdot r_{i-1} + (1 - rho)\left(\left.\frac{df(\boldsymbol{x})}{d\boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_i}\right)^2$$

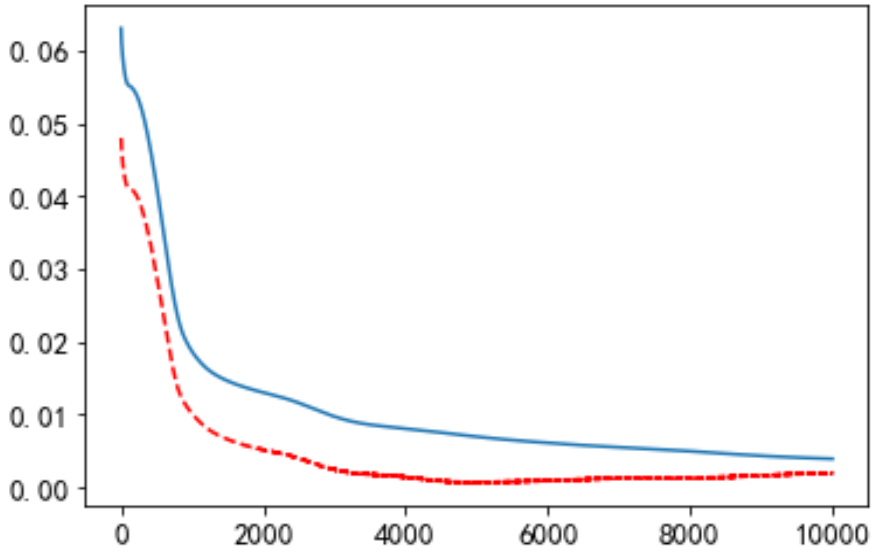
$$\theta_i = \beta\theta_{i-1} + \frac{lr}{\sqrt{r_i + \epsilon}} \cdot \left.\frac{df(\boldsymbol{x})}{d\boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_i}$$

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} - \theta_i$$

其中，β为动量系数。

多层神经网络中过拟合的抑制

多层神经网络中的过拟合



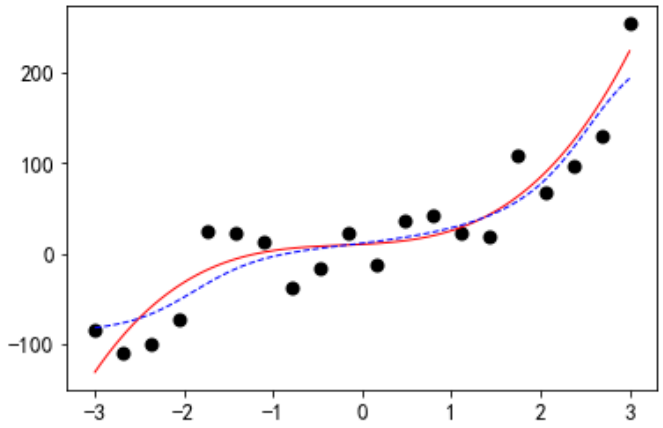
减少模型规模（减少多层神经网络的层数和节点数）和增加训练样本数量是防止过拟合的重要方法。

本小节讨论抑制过拟合的正则化、早停和Dropout等方法在多层神经网络中的应用。

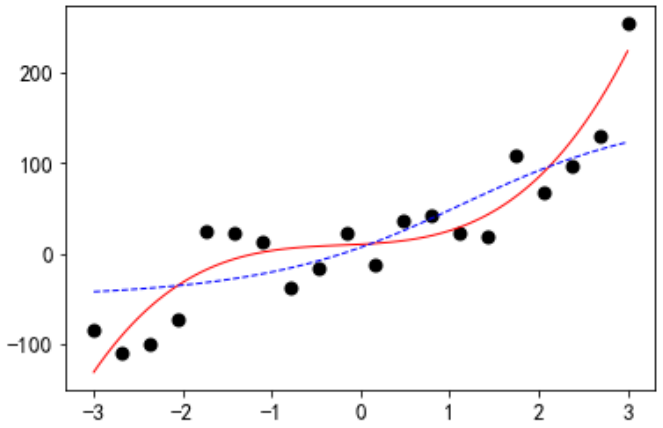
多层神经网络中过拟合的抑制

1. 正则化方法的应用

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation='sigmoid', input_shape=(1,),
                          kernel_initializer='random_uniform',
bias_initializer='zeros'),
    tf.keras.layers.Dense(5, activation='sigmoid',
kernel_regularizer=regularizers.l2(0.001),
                          kernel_initializer='random_uniform',
bias_initializer='zeros'),
    tf.keras.layers.Dense(1, activation='sigmoid',
                          kernel_initializer='random_uniform',
bias_initializer='zeros')
```



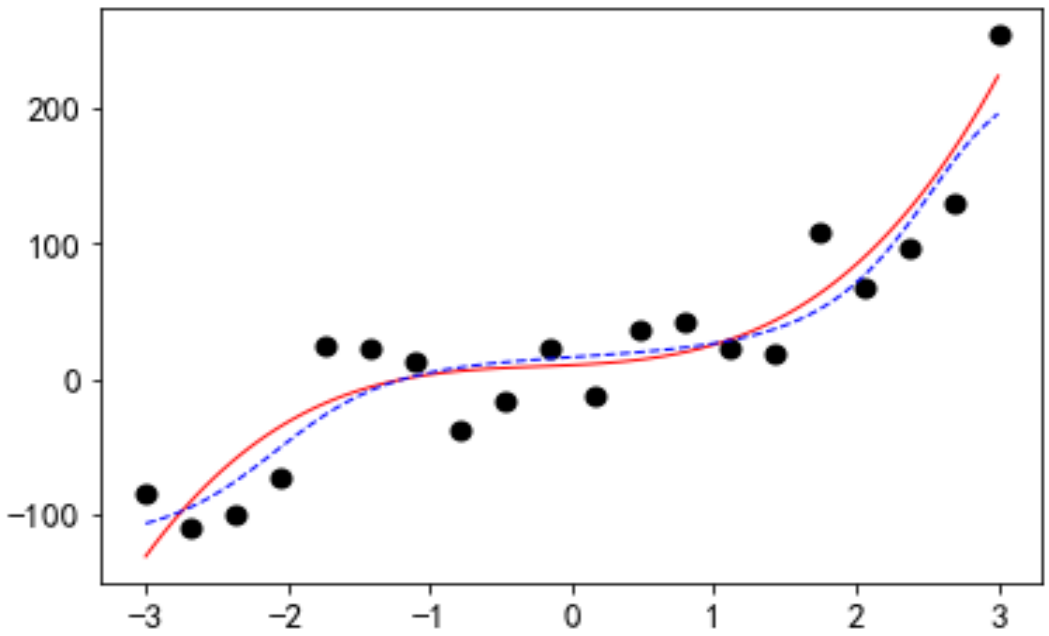
（1）第三层节点增加 L2 正则化的拟合结果



（2）所有隐层节点增加 L2 正则化的拟合结果

多层神经网络中过拟合的抑制

2. 早停法的应用

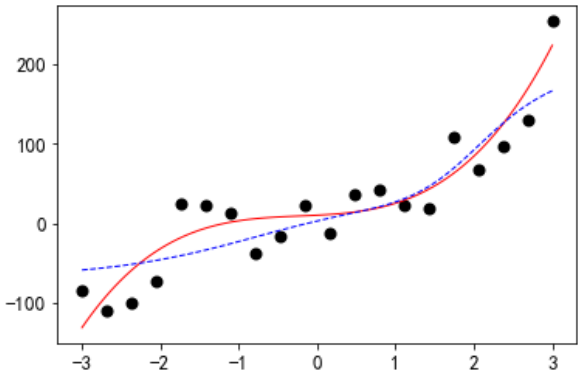



```
# 验证集
x1 = np.linspace(-3, 3, 100)
y0 = myfun(x1)
y00 = y0.copy()
standard(y0, -131.0, 223.0)

earlyStopping=tf.keras.callbacks.EarlyStopping(monitor='val_loss',
min_delta=0.000001, patience=5, verbose=1, mode='min')

model.fit(x, y, batch_size=20, epochs=10000, verbose=1,
callbacks=[earlyStopping],
    validation_data=(x1, y0))
```

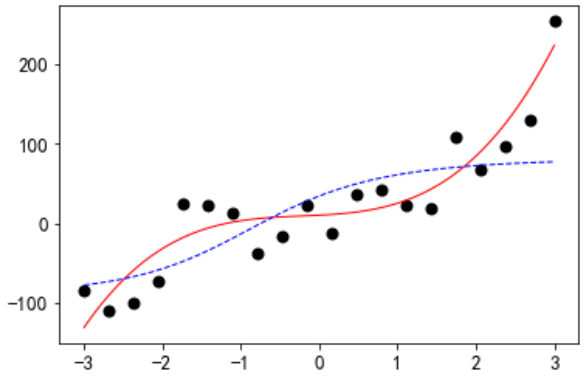
多层神经网络中过拟合的抑制

3. Dropout法的应用

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation='sigmoid', input_shape=(1,),
                          kernel_initializer='random_uniform',
bias_initializer='zeros'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(5, activation='sigmoid',
                          kernel_initializer='random_uniform',
bias_initializer='zeros'),
    tf.keras.layers.Dense(1, activation='sigmoid',
                          kernel_initializer='random_uniform',
bias_initializer='zeros')
])
```



（1）第二层后增加 Dropout（0.1）层的拟合结果



（2）第三层后增加 Dropout（0.5）层的拟合结果

The End