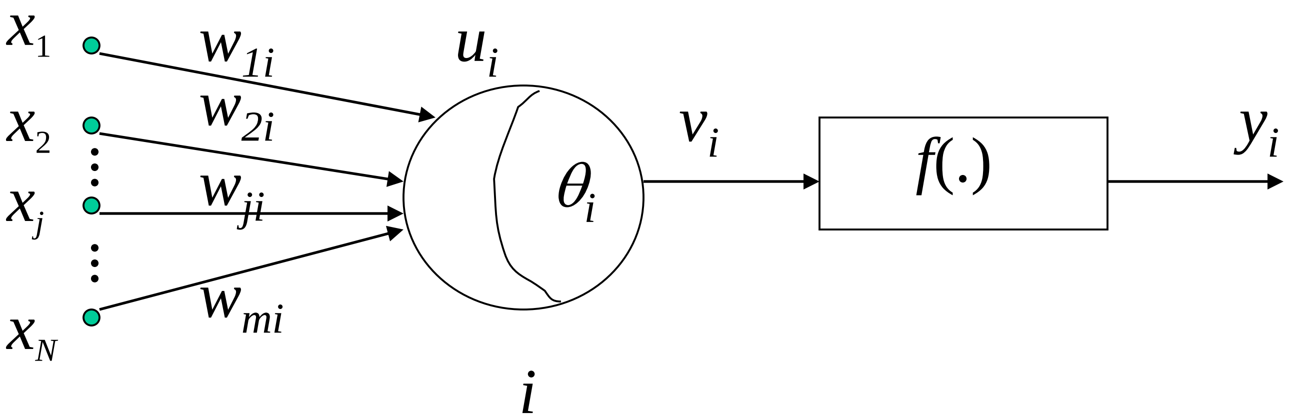
第三章 多层前向网络及其BP算法

内容

- 多层前向网络模型
- 逼近定理
- BP算法及其推导过程
- BP算法步骤
- BP算法分析
- BP算法改进
- 仿真实验

单层感知机模型（一个神经元）

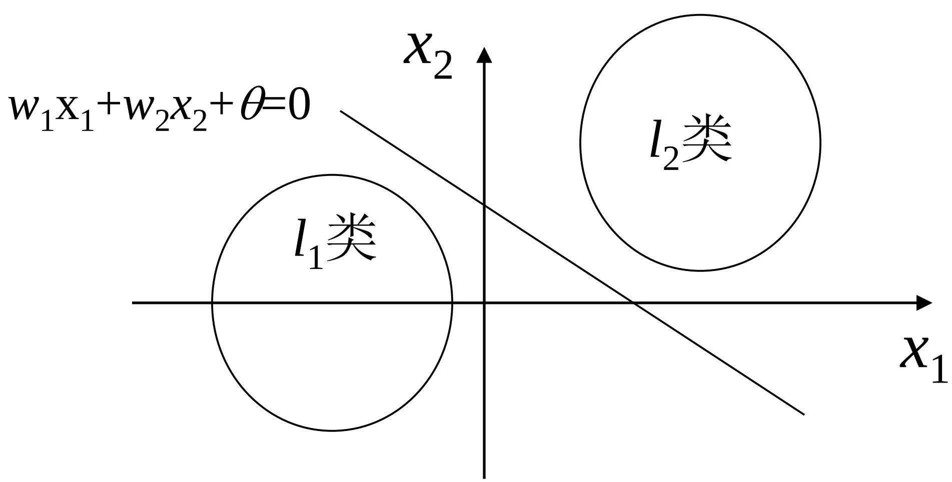


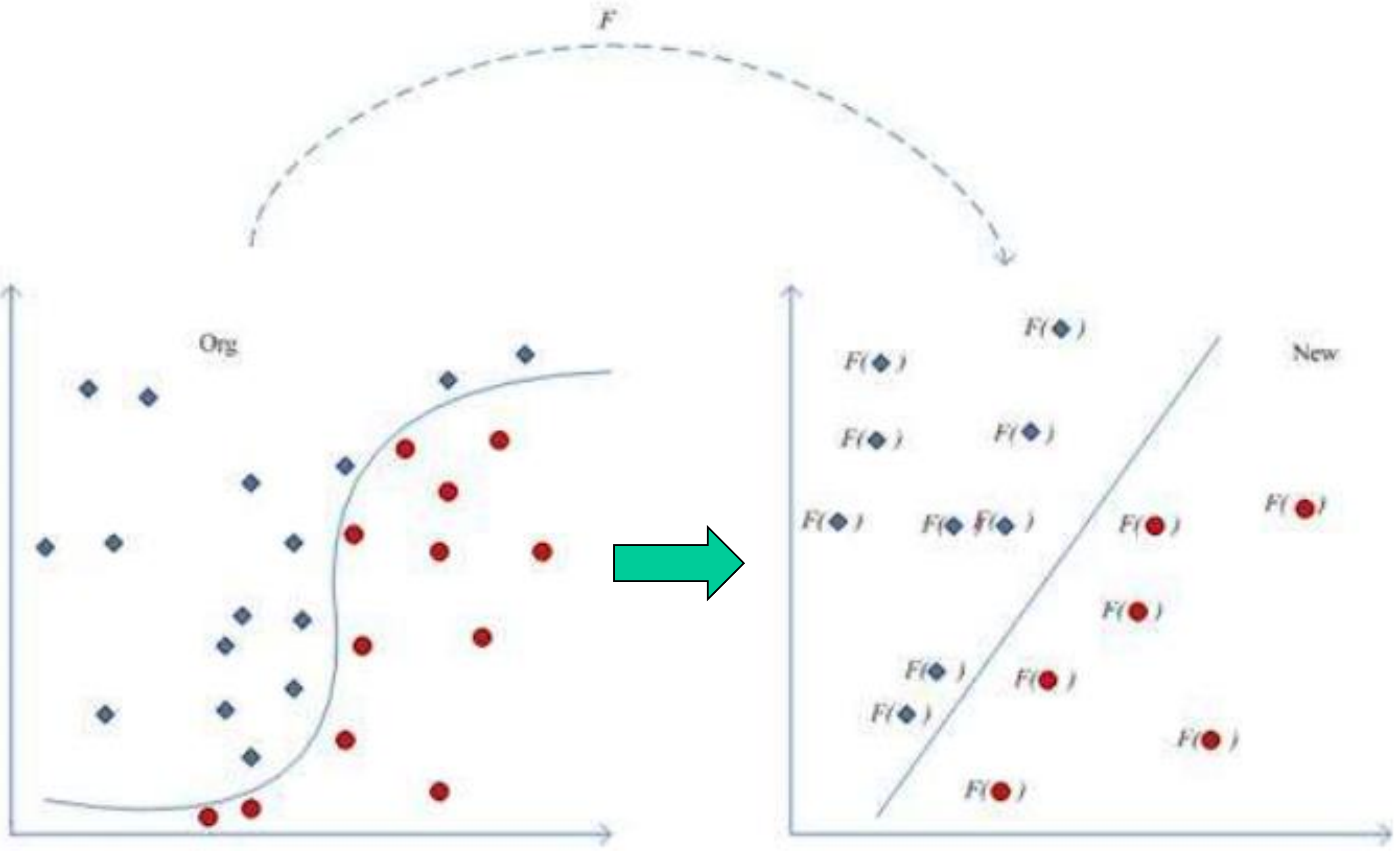
$$y_i = \mathrm{sgn}\left(\sum_{j=1}^{m} w_{ji} x_j + \theta_i\right)$$

$$= \begin{cases} +1\,(or\ 1), if & \sum_{j=1}^{m} w_{ji} x_j + \theta_i \geq 0 \\ -1\,(or\ 0), if & \sum_{j=1}^{m} w_{ji} x_j + \theta_i < 0 \end{cases}$$

单层感知机的作用

- 单层感知机(一个神经元)的作用就是对输入模式进行分类。若模型输出为+1，则将输入归为l_1类；否则，若模型输出为-1(或0)，则将输入归为l_2类。
- 单层感知机(一个神经元)进行模式分类的判据为$\Sigma w_{ji}x_j+\theta_i=0$.
- 下图给出2维输入的一种超平面判决，其判决边界为直线$w_1x_1+w_2x_2+\theta=0$，其中w_1、 w_2 、θ 为参数，可通过学习算法确定。




F
Org
New
F(◆)
F(●)

多层前向网络模型

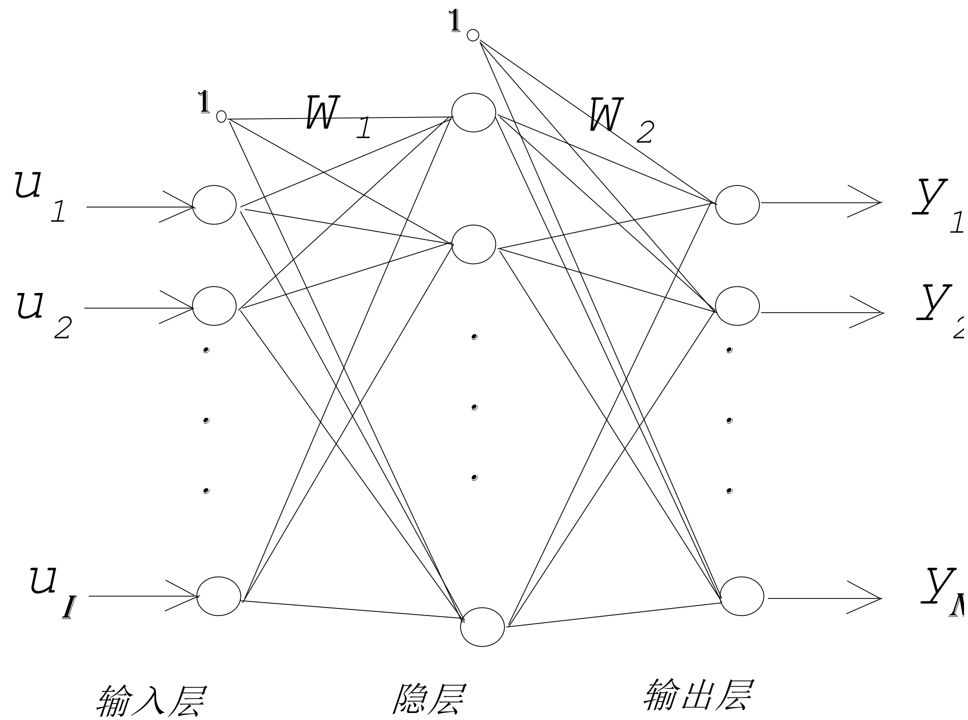
$$x_j^l = \sum_{i=0}^{N_{l-1}} w_{ij}^{l-1,l} y_i^{l-1}, l > 0, j > 0$$

$$f_l(x) = [1 + \exp(-x)]^{-1}, l > 0$$

$$\log sig(n) = \frac{1}{1+e^{-n}}$$

也可以是

$$\tan sig(n) = \frac{2}{1+e^{-2n}} - 1$$



$$E = \sum_{p=1}^{P} E_p = \frac{1}{2} \sum_{p=1}^{P} \sum_{j=1}^{N_{M-1}} (y_{p,j}^{M-1} - t_{p,j})^2$$

逼近定理

- 令$\varphi(.)$为有界、非常量的单调增连续函数，I_p表示p维单位超立方体$[0,\ 1]^p$，$C(I_p)$表示定义在I_p上的连续函数的集合，则给定任何函数$f\in C(I_p)$和$\varepsilon>0$，存在整数M和一组实常量α_i，θ_i，w_{ij}，其中$i=1,\ldots,M$，$j=1,\ldots,p$使得网络的输出 $F(x_1,\cdots,x_p)=\sum_{i=1}^{M}\alpha_i\varphi(\sum_{j=1}^{p}w_{ij}x_j-\theta_i)$ 可任意逼近f，即：

$$\forall(x_1,\cdots,x_p)\in I_p,\quad \left|F(x_1,\cdots,x_p)-f(x_1,\cdots,x_p)\right|<\varepsilon$$

注：

该定理说明只含一个隐层的前向网络是一个通用的逼近器，但没有说明如何构造网络，也没有说明怎样的网络效果最好。另外，前向网络的逼近定义也可以推广到连续函数。

BP算法

$$w_{i,j}^{l-1,l}(k+1) = w_{i,j}^{l-1,l}(k) - \alpha \cdot \partial E / \partial w_{i,j}^{l-1,l}(k)$$

$$= w_{i,j}^{l-1,l}(k) + \alpha \cdot \sum_{p=1}^{P} \delta_{p,j}^{l}(k) \cdot y_{p,i}^{l-1}(k)$$

α 为学习率

$$\delta_{p,j}^{l}(k) = \begin{cases} [t_{p,j} - y_{p,j}^{l}(k)] \cdot f'[x_{p,j}^{l}(k)] & l = M-1 \\ f'[x_{p,j}^{l}(k)] \sum_{n=1}^{N_{l+1}} \delta_{p,n}^{l+1}(k) \cdot w_{j,n}^{l+1}(k) & l = M-2, \ldots, 1 \end{cases}$$

推导过程：定义

- 令输入层为第0层，其激励函数为线性函数$y=x$；
- 隐层为第1层至第M-1层，其中第M-1层为输出层，隐层均采用Sigmoid激励函数，即$y=1/[1+e^{-x}]$。
- 记第l层的神经元数目为N_l，其中第0号神经元为阈值单元，其输出恒为1。另外，输出层不含阈值神经元。
- 记第l-1层中第i个神经元到第l层中第j个神经元的连接权为$w^{l-1,l}_{i,j}$。
- 从而，第l(l>0)层第j(j>0)个神经元的输入为 $x_j^l = \sum_{i=0}^{N_{l-1}} w_{ij}^{l-1,l} y_i^{l-1}$。
- 记样本总数为P，NN对应第p个样本的第j个输出分量为$t_{j,p}$，则目标函数为

$$E = \sum_{p=1}^{P} E_p = \frac{1}{2} \sum_{p=1}^{P} \sum_{j=1}^{N_{M-1}} (y_{j,p}^{M-1} - t_{j,p})^2$$

推导过程：前推

- 第0层： $\begin{cases} y_{p,j}^{0}=x_{p,j}^{0}=u_{p,j},\ j=1,\cdots,N_0,\ p=1,\cdots,P; \\ y_0^0\equiv 1 \end{cases}$
- 第1层： $\begin{cases} y_{p,j}^{1}=f(x_{p,j}^{1})=f(\sum_{i=0}^{N_0} w_{i,j}^{0,1}y_{p,i}^{0}),\ j=1,\cdots,N_1,\ p=1,\cdots,P; \\ y_0^1\equiv 1 \end{cases}$
- 第l层：类似于第1层

$$\begin{cases} y_{p,j}^{l}=f(x_{p,j}^{l})=f(\sum_{i=0}^{N_{l-1}} w_{i,j}^{l-1,l}y_{p,i}^{l-1}),\ j=1,\cdots,N_l,\ p=1,\cdots,P; \\ y_0^l\equiv 1 \end{cases}$$

- …
- 第M-1层(输出层)：

$$y_{p,j}^{M-1}=f(x_{p,j}^{M-1})=f(\sum_{i=0}^{N_{M-2}} w_{i,j}^{M-2,M-1}y_{p,i}^{M-2}),\ j=1,\cdots,N_{M-1},\ p=1,\cdots,P$$

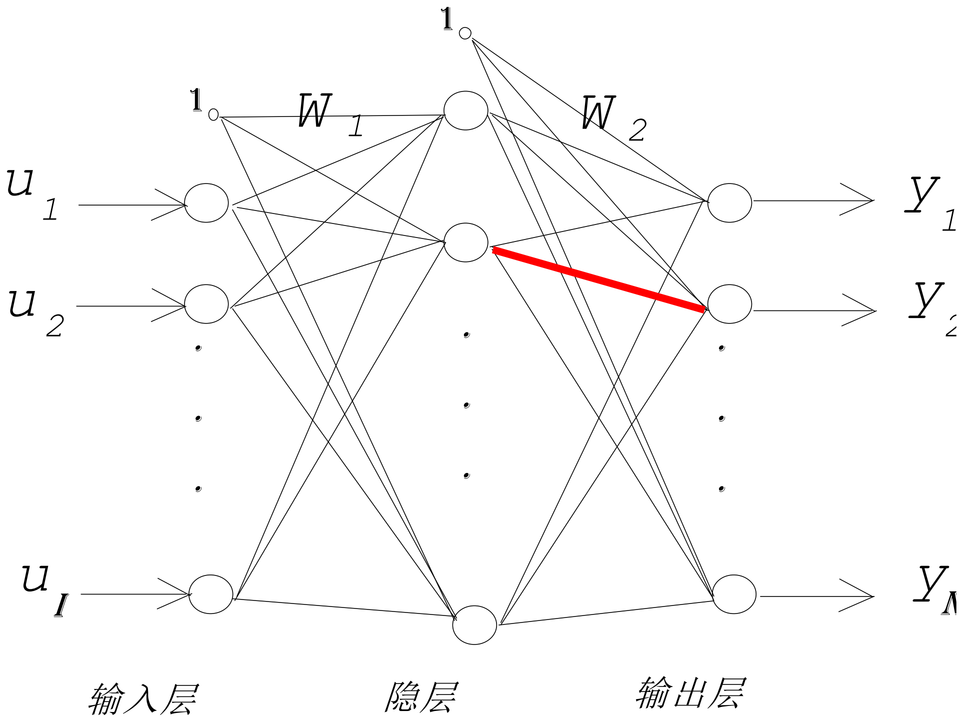
推导过程：梯度下降

- 目标函数：

$$E=\sum_{p=1}^{P}E_p=\frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{N_{M-1}}(y_{j,p}^{M-1}-t_{j,p})^2$$

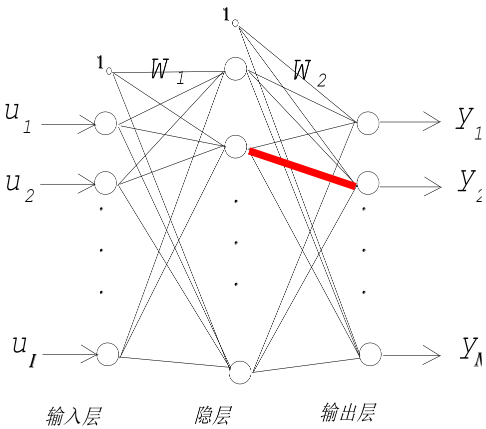
- 梯度下降：

$$w_{i,j}^{l-1,l}(k+1)=w_{i,j}^{l-1,l}(k)-\alpha\cdot\frac{\partial E}{\partial w_{i,j}^{l-1,l}}\Big|_{W=W(k)}$$


1
1
W 1
W 2
u 1
u 2
u I
y 1
y 2
y M
输入层
隐层
输出层

推导过程：反向梯度计算

- 输出层：



$$\partial E \Big/ \partial w_{i,j}^{M-2,M-1} = \sum_{p=1}^{P} \partial E_p \Big/ \partial w_{i,j}^{M-2,M-1}$$

$$= \sum_{p=1}^{P} \partial E_p \Big/ \partial x_{p,j}^{M-1} \quad \partial x_{p,j}^{M-1} \Big/ \partial w_{i,j}^{M-2,M-1}$$

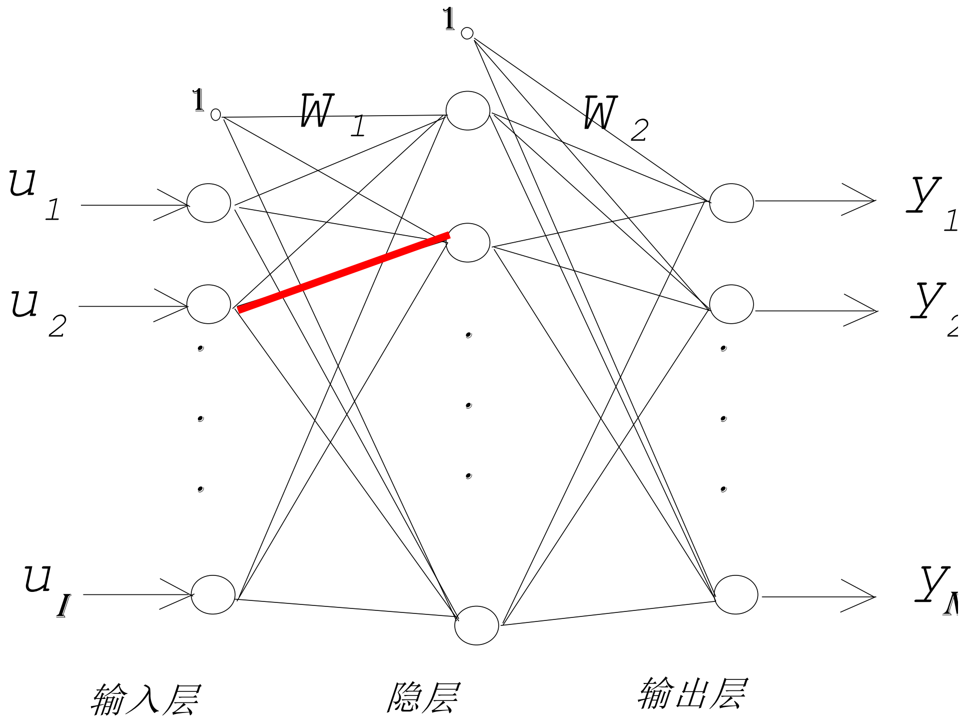
$$= \sum_{p=1}^{P} (-\delta_{p,j}^{M-1} \cdot y_{p,i}^{M-2})$$

$$\delta_{p,j}^{M-1} = -\partial E_p \Big/ \partial x_{p,j}^{M-1} = -\partial E_p \Big/ \partial y_{p,j}^{M-1} \quad \partial y_{p,j}^{M-1} \Big/ \partial x_{p,j}^{M-1}$$

$$= (t_{p,j} - y_{p,j}^{M-1}) \, \partial y_{p,j}^{M-1} \Big/ \partial x_{p,j}^{M-1}$$

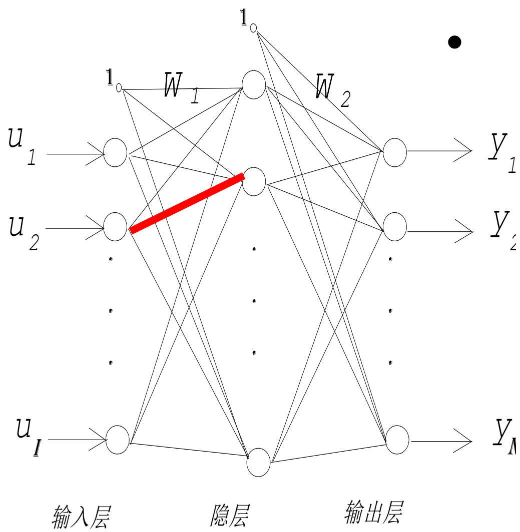
$$= (t_{p,j} - y_{p,j}^{M-1}) \cdot y_{p,j}^{M-1} \cdot (1 - y_{p,j}^{M-1}), \; j = 1, \cdots, N_{M-1}$$

$$y = 1/[1 + e^{-x}] \Rightarrow \partial y / \partial x = y(1-y)$$



1
1
W 1
W 2
u 1
u 2
u I
y 1
y 2
输入层
隐层
输出层

推导过程：反向梯度计算



- 隐层(第M-2层)：

$$\partial E \big/ \partial w_{i,j}^{M-3,M-2} = \sum_{p=1}^{P} \partial E_p \big/ \partial w_{i,j}^{M-3,M-2}$$

$$= \sum_{p=1}^{P} \partial E_p \big/ \partial x_{p,j}^{M-2} \quad \partial x_{p,j}^{M-2} \big/ \partial w_{i,j}^{M-3,M-2}$$

$$= \sum_{p=1}^{P} (-\delta_{p,j}^{M-2} \cdot y_{p,i}^{M-3})$$

$$\delta_{p,j}^{M-2} = -\partial E_p \big/ \partial x_{p,j}^{M-2} = -\partial E_p \big/ \partial y_{p,j}^{M-2} \quad \partial y_{p,j}^{M-2} \big/ \partial x_{p,j}^{M-2}$$

$$= -\partial E_p \big/ \partial y_{p,j}^{M-2} \cdot y_{p,j}^{M-2} \cdot (1 - y_{p,j}^{M-2}),\ j = 1, \cdots, N_{M-2}$$

$$\partial E_p \big/ \partial y_{p,j}^{M-2} = \sum_{n=1}^{N_{M-1}} \partial E_p \big/ \partial x_{p,n}^{M-1} \quad \partial x_{p,n}^{M-1} \big/ \partial y_{p,j}^{M-2}$$

$$= \sum_{n=1}^{N_{M-1}} (-\delta_{p,n}^{M-1} \cdot w_{j,n}^{M-2,M-1})$$

$$\therefore \quad \delta_{p,j}^{M-2} = \sum_{n=1}^{N_{M-1}} \delta_{p,n}^{M-1} \cdot w_{j,n}^{M-2,M-1} \cdot y_{p,j}^{M-2} \cdot (1 - y_{p,j}^{M-2})$$

注：隐层第M-3层至第1层的权值更新公式同上。

BP算法

$$w_{i,j}^{l-1,l}(k+1) = w_{i,j}^{l-1,l}(k) - \alpha \cdot \partial E / \partial w_{i,j}^{l-1,l}(k)$$
$$= w_{i,j}^{l-1,l}(k) + \alpha \cdot \sum_{p=1}^{P} \delta_{p,j}^{l}(k) \cdot y_{p,i}^{l-1}(k)$$

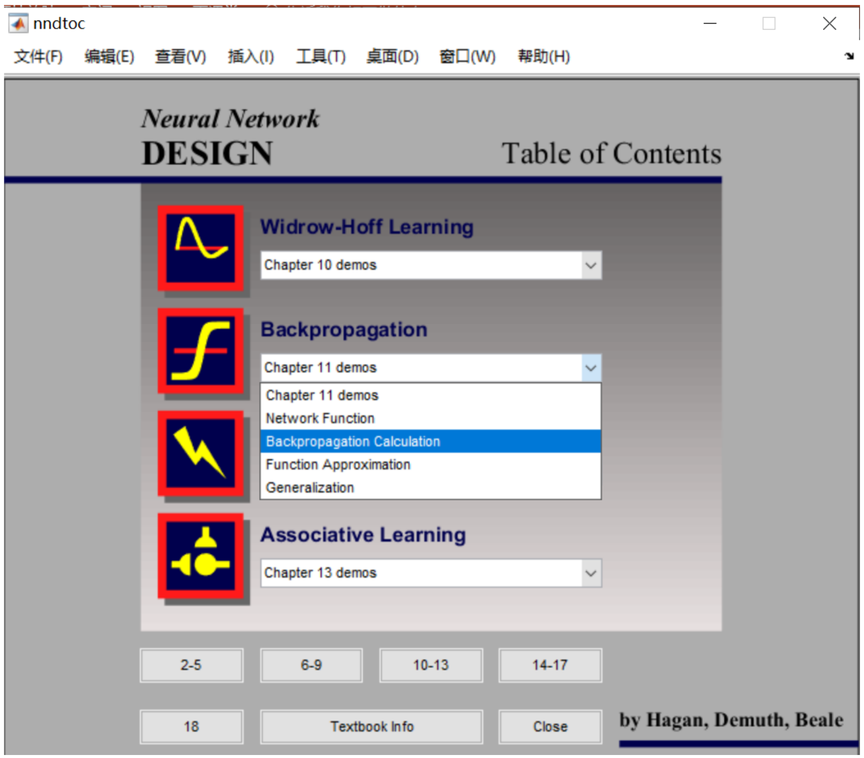
α 为学习率

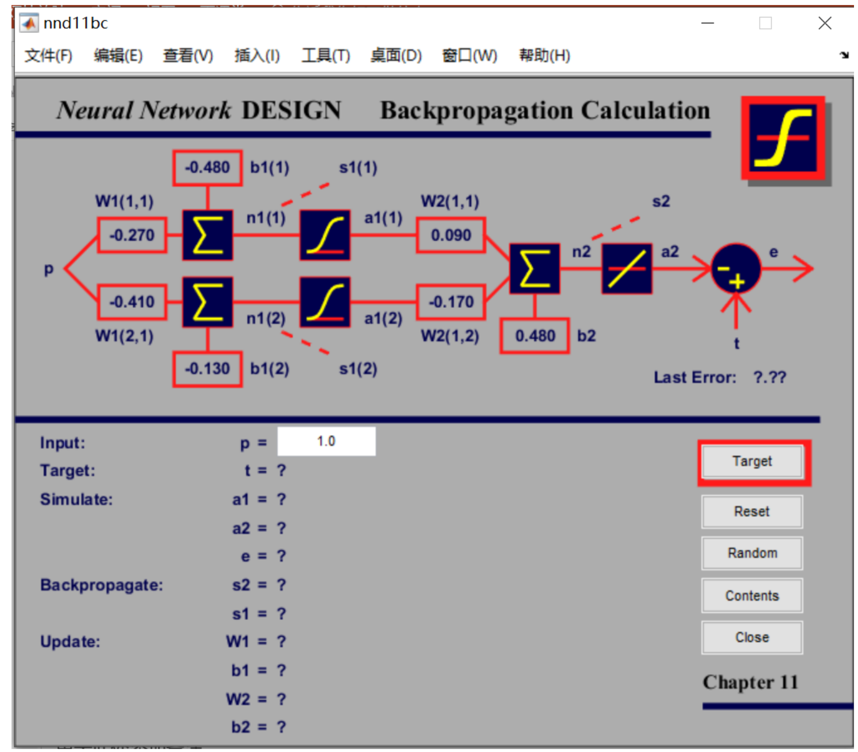
$$\delta_{p,j}^{l}(k) = \begin{cases} [t_{p,j} - y_{p,j}^{l}(k)] \cdot f'[x_{p,j}^{l}(k)] & l = M-1 \\ f'[x_{p,j}^{l}(k)] \sum_{n=1}^{N_{l+1}} \delta_{p,n}^{l+1}(k) \cdot w_{j,n}^{l+1}(k) & l = M-2,\ldots,1 \end{cases}$$

BP算法步骤

- 第1步：设置变量和参量，譬如学习步长等；
- 第2步：初始化权值矩阵(较小的非0阵)；
- 第3步：对输入样本在当前权值下前向计算FNN各神经元的输出；
- 第4步：判断算法终止条件是否满足？若是则输出权值，否则转步骤5；
- 第5步：反向计算各权值的变化量，更新权值，然后返回步骤3。

BP-FNN计算演示

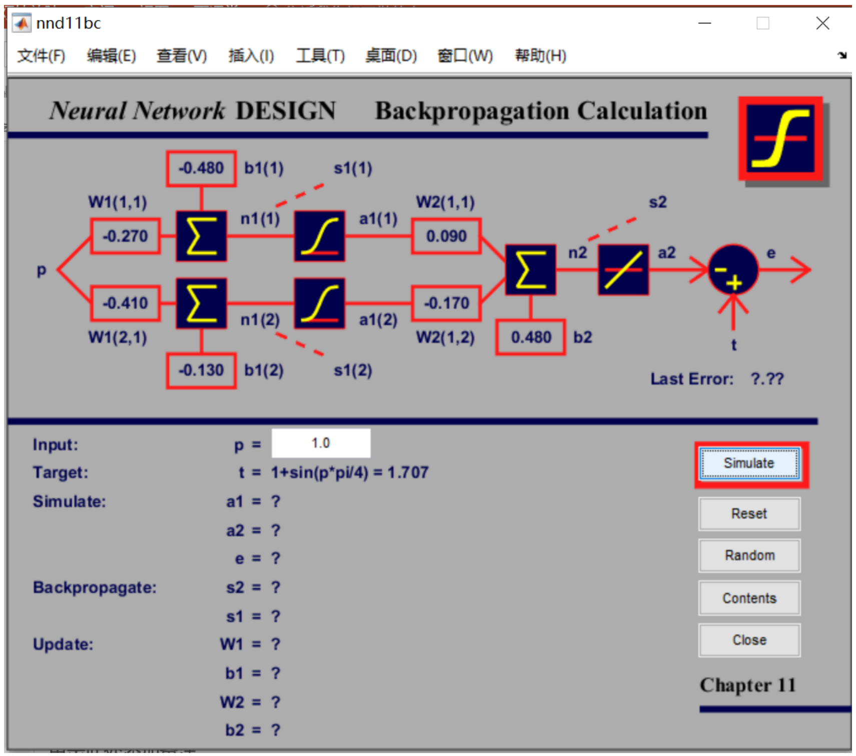


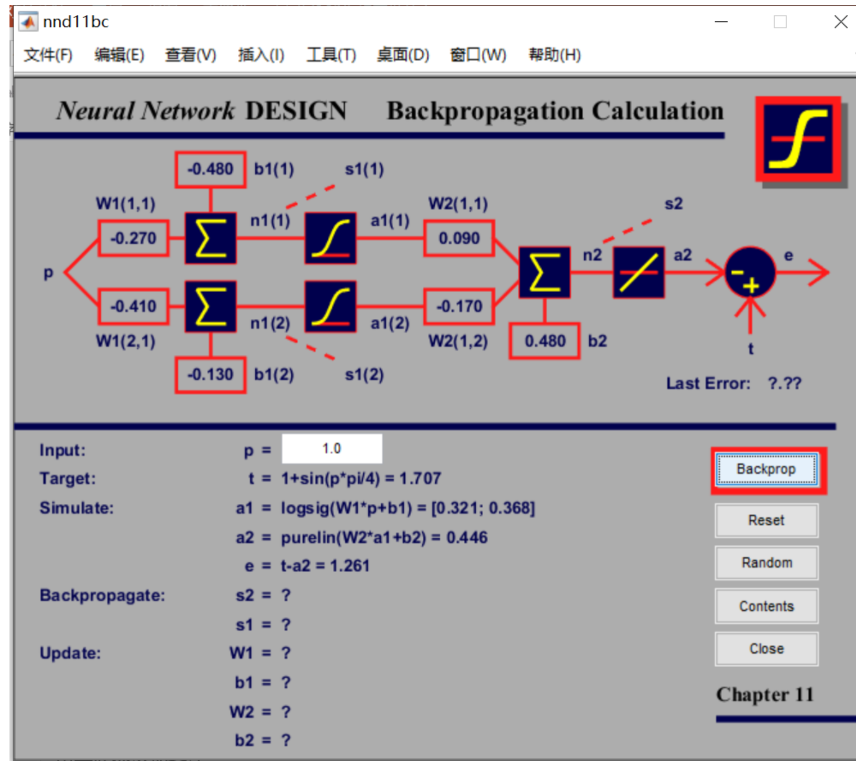


启动演示模块

网络结构与初始参数

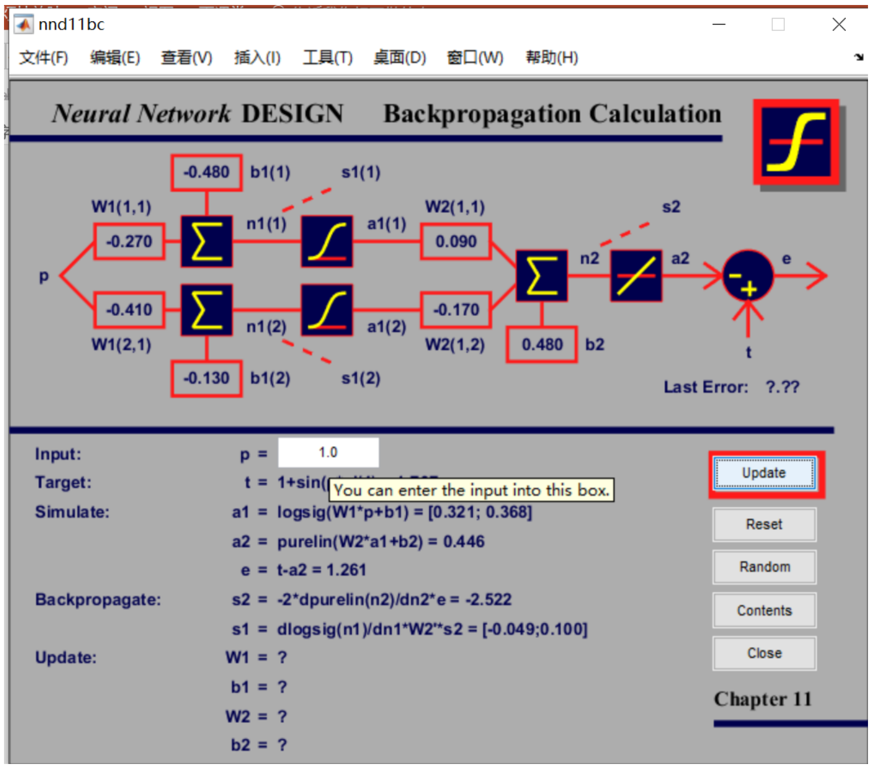
BP-FNN计算演示



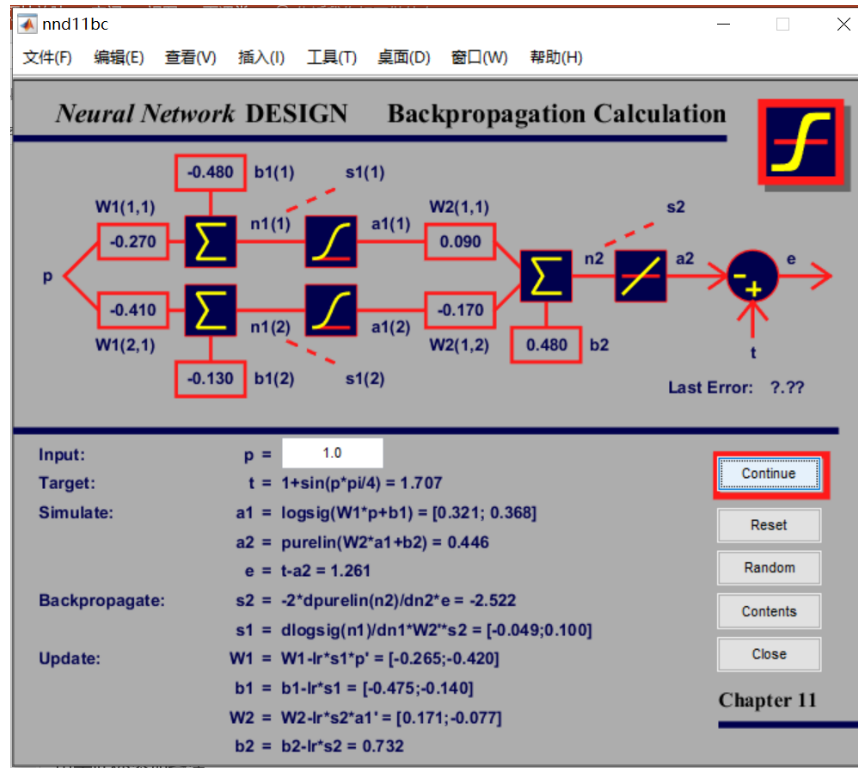


给出样本的期望输出　　　　前向计算各层神经元的输入输出

BP-FNN计算演示

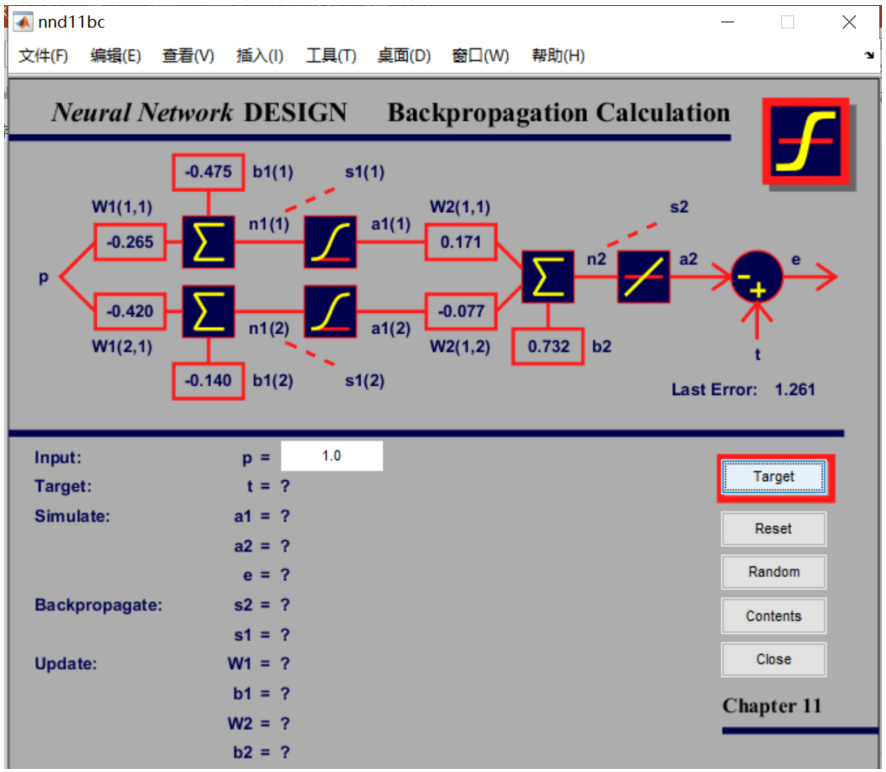


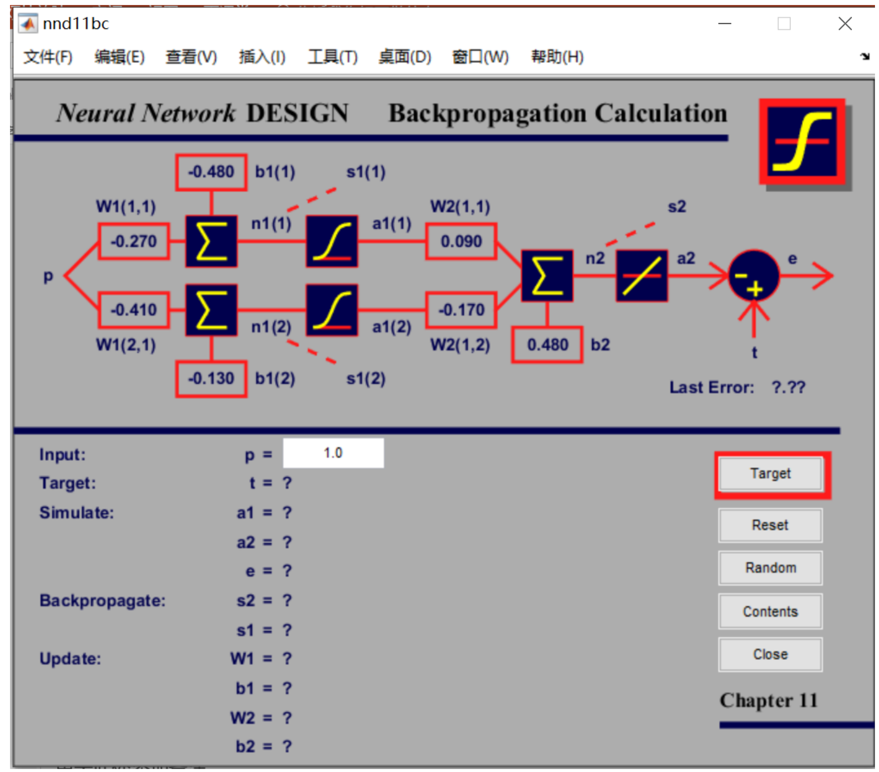
反向传播计算更新量



确定更新量

BP-FNN计算演示



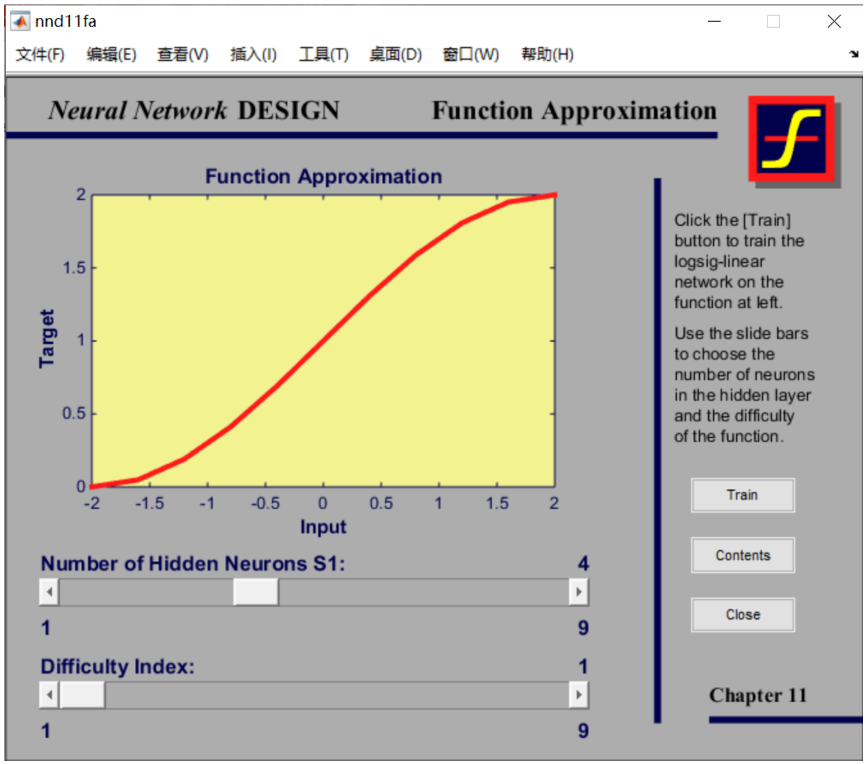


获得更新后的FNN

更新前的FNN

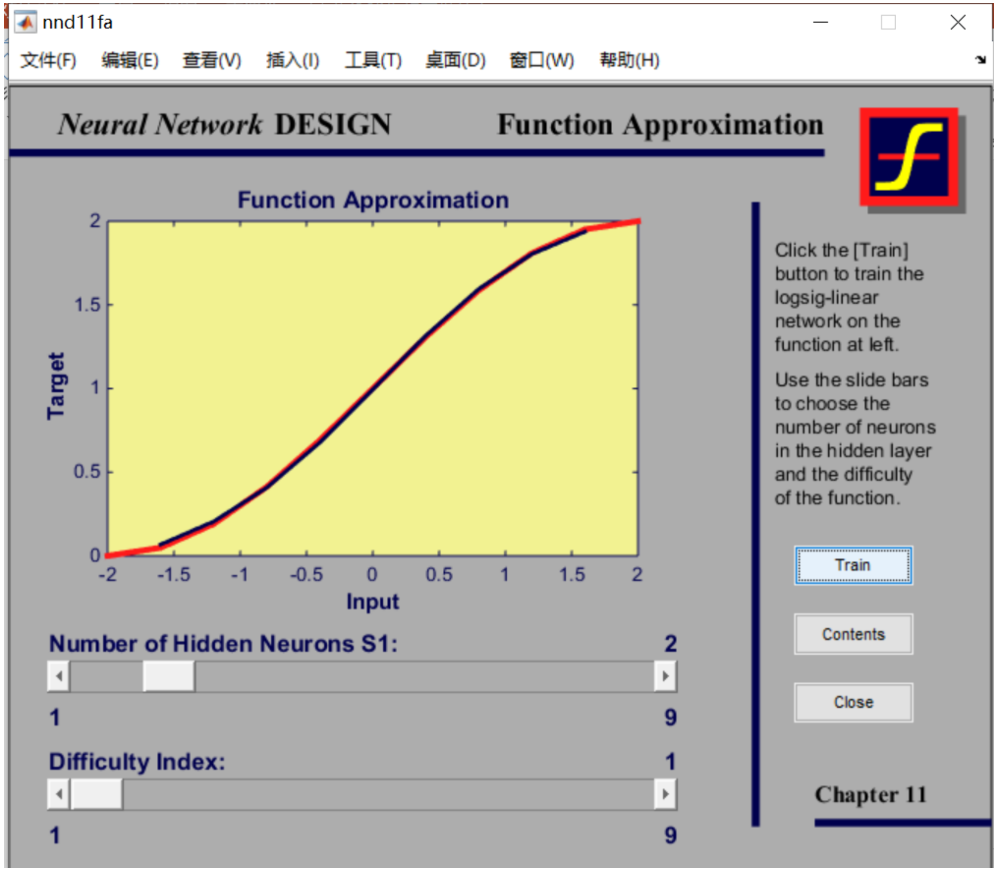
仅仅权值发生变化

BP-FNN函数逼近演示



启动演示模块　　　　函数逼近界面说明

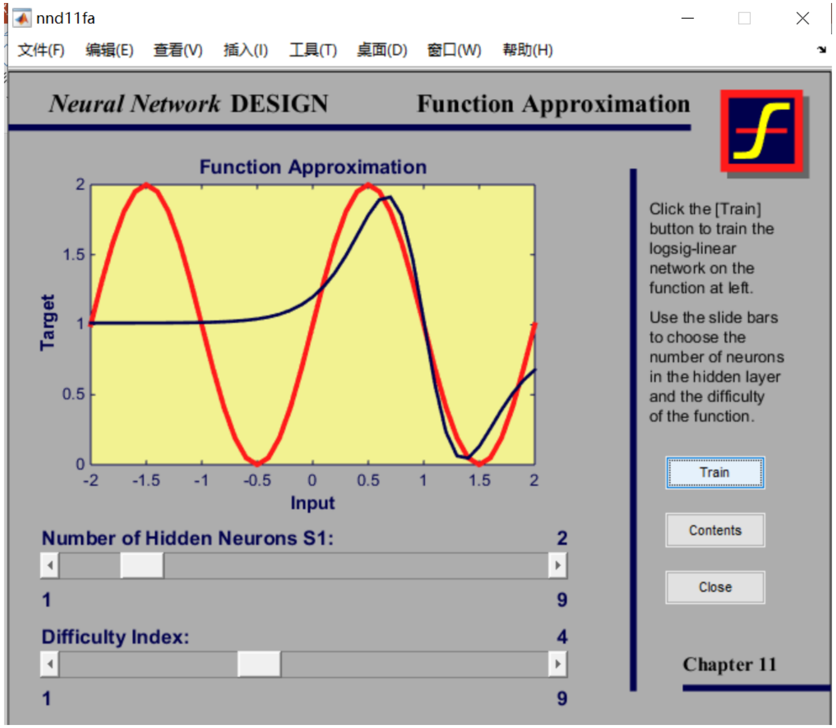
BP-FNN函数逼近演示

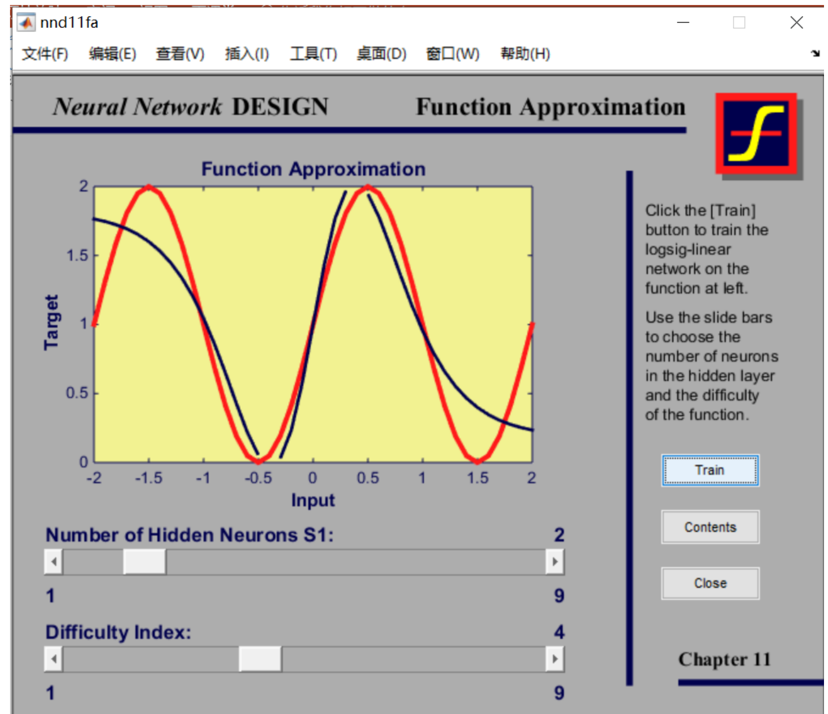


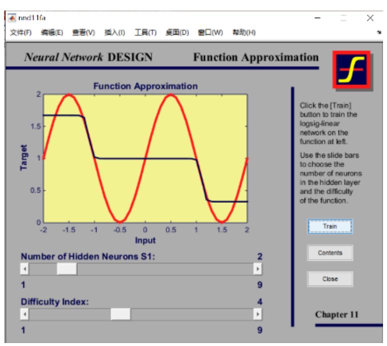
运行结果，基本一致

2个隐节点，难度系数1

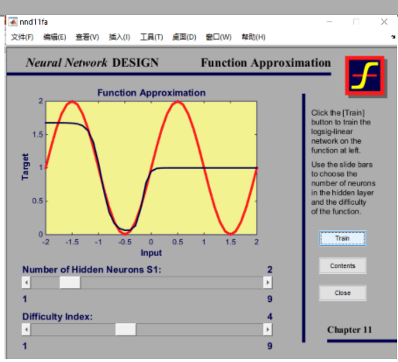
BP-FNN函数逼近演示



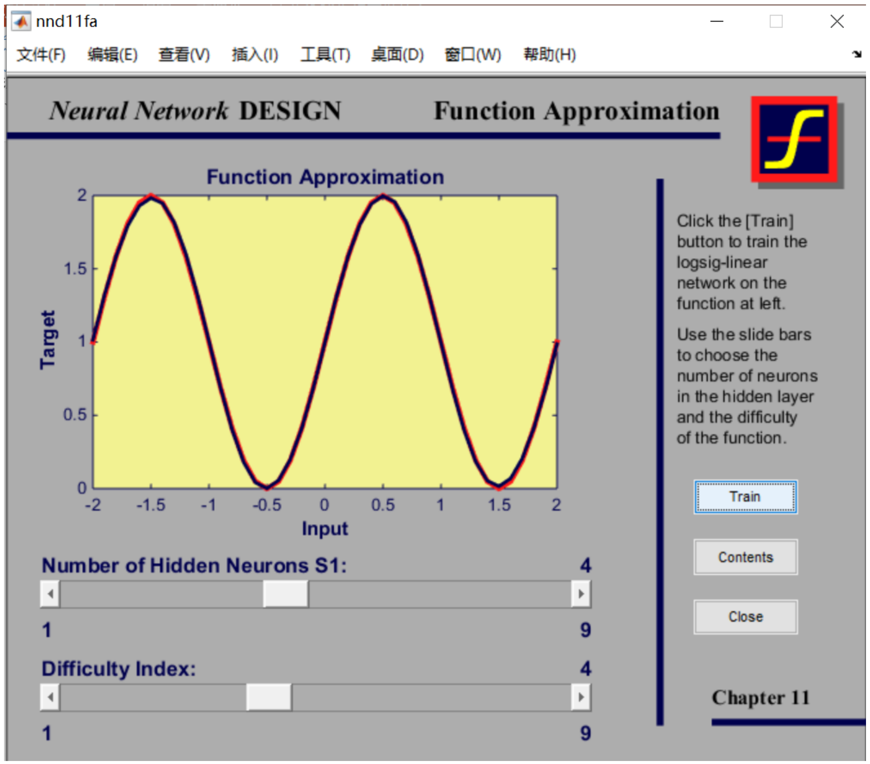


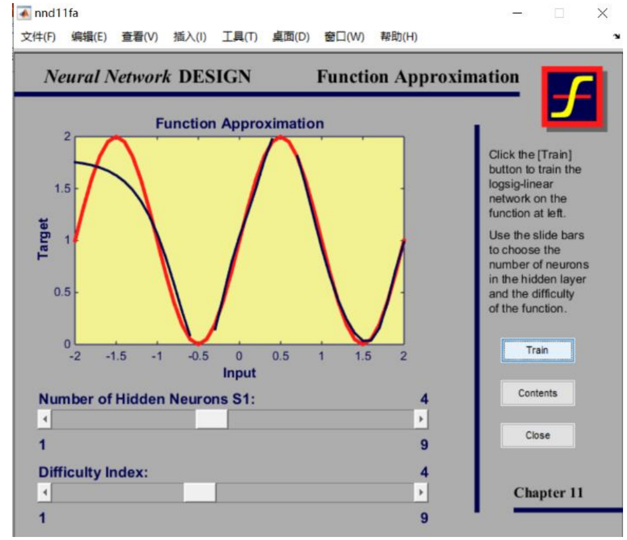


2个隐节点，难度系数4

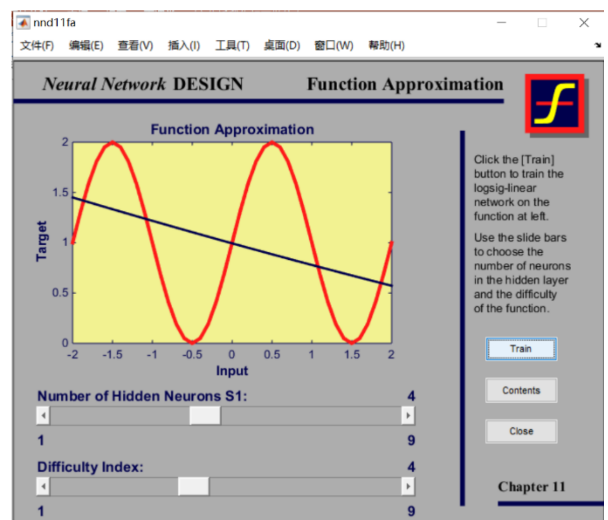


BP-FNN函数逼近演示





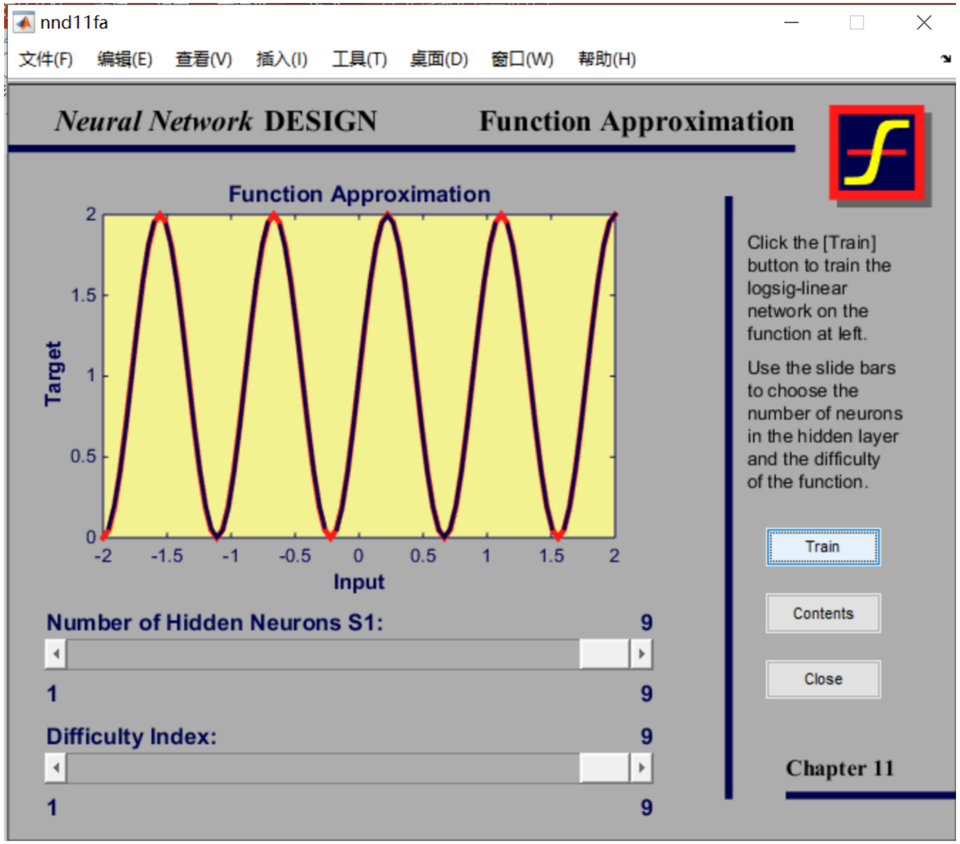
少量结果



大多数结果

4个隐节点，难度系数4

BP-FNN函数逼近演示



运行结果，基本一致，但运行时间有差异

9个隐节点，难度系数9

思考

- 从上述演示中发现了哪些现象？

BP算法分析

- 实质上，BP算法是一种梯度下降法，算法性能依赖于初始条件，学习过程易于陷入局部极小。
- 数值仿真研究表明，BP算法的学习速度、精度、初值鲁棒性和网络推广性能都较差。

BP算法收敛缓慢的主要原因

- BP算法利用梯度信息来调整权值，在误差曲面平坦处，导数值较小使得权值调整幅度较小，从而误差下降很慢；在曲面曲率大处，导数值较大使得权值调整幅度较大，会出现跃冲极小点现象，从而引起振荡。
- 神经元的总输入偏离阈值太远时，总输入就进入激励函数非线性特性的饱和区。此时若实际输出与期望输出不一致，激励函数较小的导数值将导致算法难以摆脱“平台”区。
- 由于网络结构的复杂性，不同权值和阈值对同一样本的收敛速度不同，使得整体学习速度缓慢。

克服BP算法训练缓慢的常用方法

- 改变学习步长。等效于对权值的改变，从而改变误差曲面的形状，缩短到达极小点的路径而加速收敛。
- 加动量项和改变动量因子。即，$\Delta w_{ij}(n)=\eta\times\Delta w_{ij}(n\text{-}1)\text{-}\alpha\times\partial E/\partial w_{ij}$。使权值变化更平滑而有利于加速收敛，但动量因子需适当选择或自适应改变。
- 合适选择神经元激励函数和初始权值、阈值，并对输入样本的归一化处理。目的是避免“平台”现象的出现。
- 采用合适的训练模式(如逐一式、批处理、跳跃式等)。避免权值收敛速度的不平衡现象。
- 采用高阶导数信息、最优滤波法和启发式算法。如二阶导数法、共轭梯度法、准牛顿法、扩展Kalman算法和delta-bar-delta算法等。

BP算法易陷入局部极小的原因和改进措施

- 由于不能保证目标函数在权空间中的正定性，而误差曲面往往复杂且无规则，存在多个分布无规则的局部极小点，因而基于梯度下降的BP算法易于陷入局部极小。
- 改进措施主要有：
  - 引入全局优化技术；
  - 平坦化优化曲面以消除局部极小；
  - 设计合适网络使其满足不产生局部极小条件。

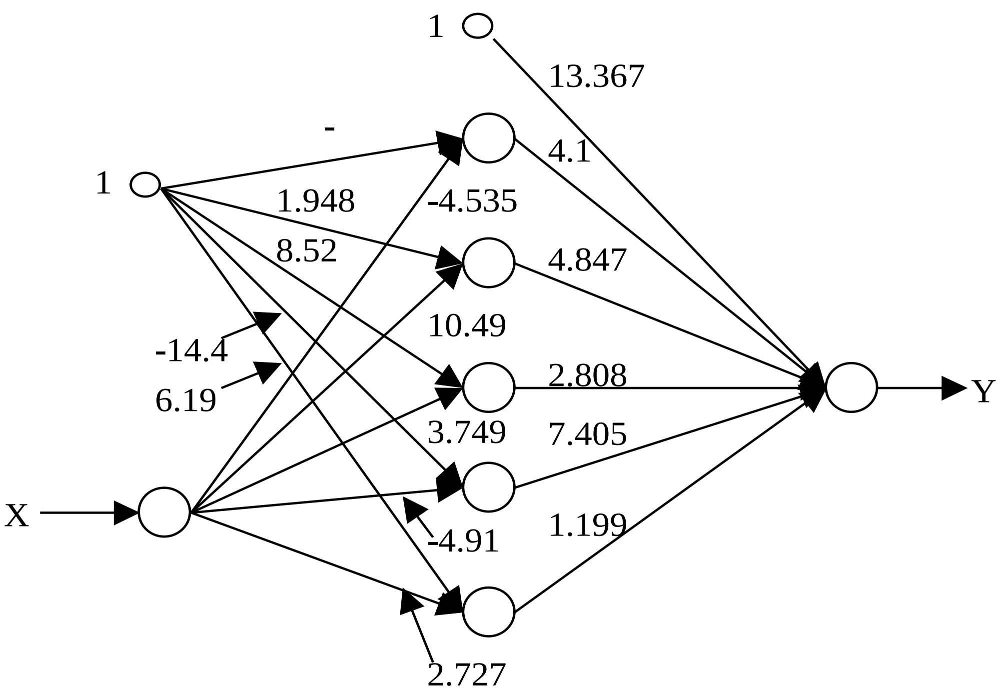
BP算法的推广性能及其改进措施

- 网络的推广性能差主要表现为，网络能够很好地实现训练样本的输入输出映射，但不能保证对未训练的样本输入得到理想的输出。
- 改进方法主要有：
  - 引入与问题相关的先验知识对权值加以限制；
  - 产生虚拟“瓶颈层”，以便对权矩阵的秩施加限制；
  - 对目标函数附加惩罚项以强制无用权值趋于零；
  - 动态修改网络结构，对推广函数与目标函数进行多目标优化等。

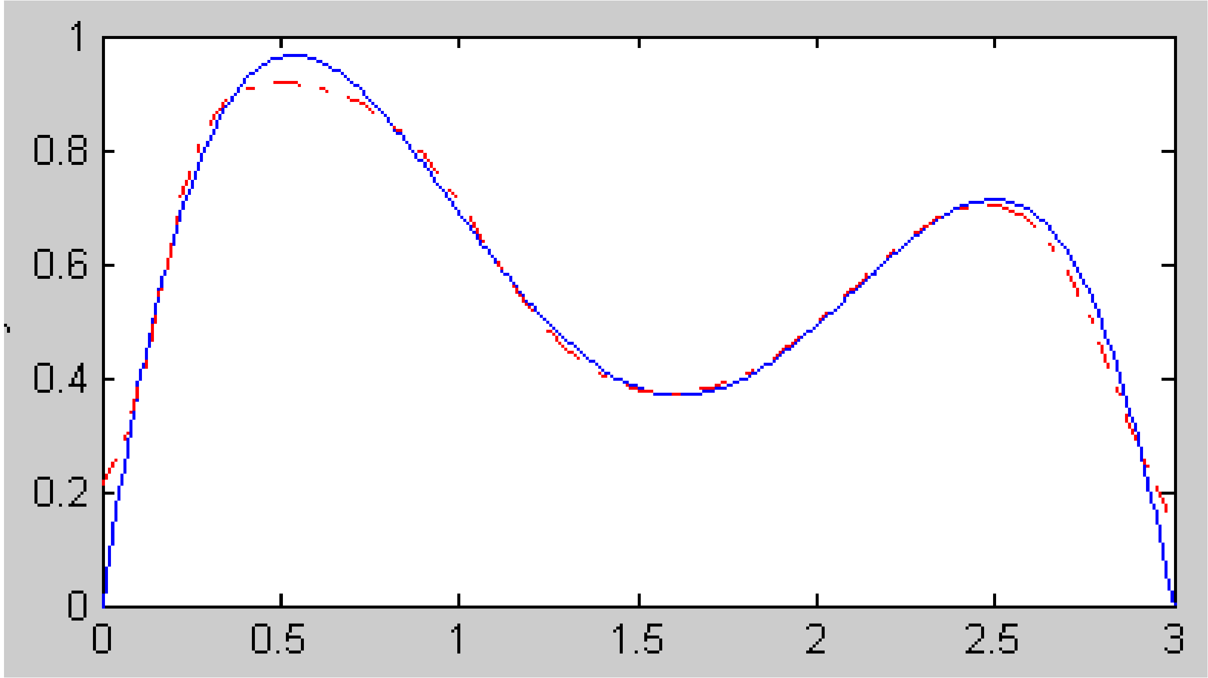
函数逼近仿真实验1

$$f = -x(x^2 - 3.2x + 1.7^2)(x-3)/2 \qquad x \in [0,3]$$

- 学习速率0.2，动量因子0.4，隐节点5个，训练步数10000步



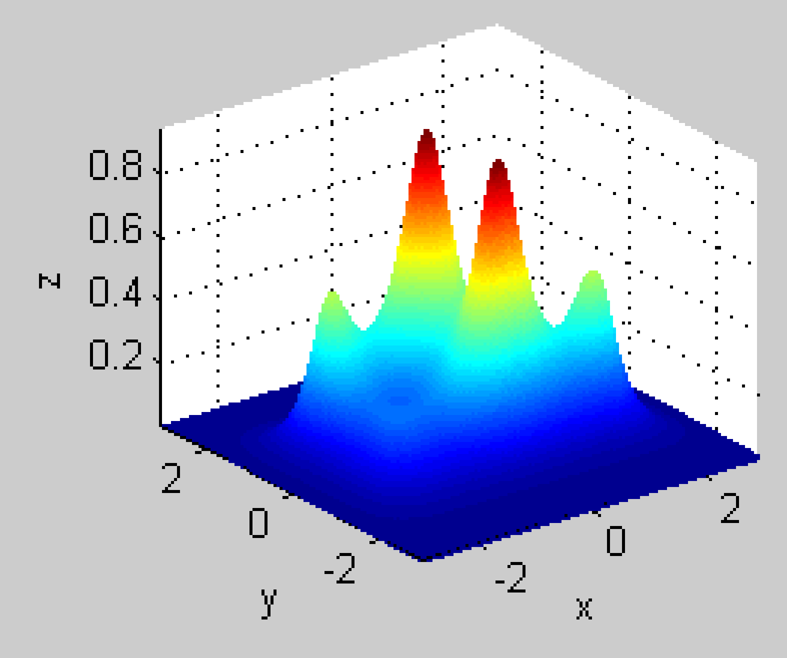
实验结果

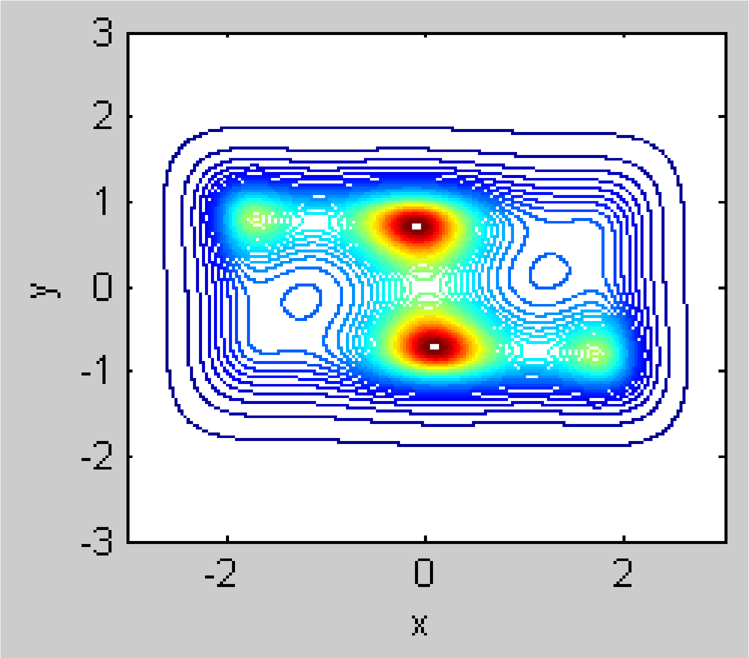


(实线：真实模型曲线　　点划线：神经网络输出曲线)

函数逼近仿真实验2

$$f=\frac{10}{(4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4+2)*11} \qquad -3\le x_i\le 3$$





仿真结果

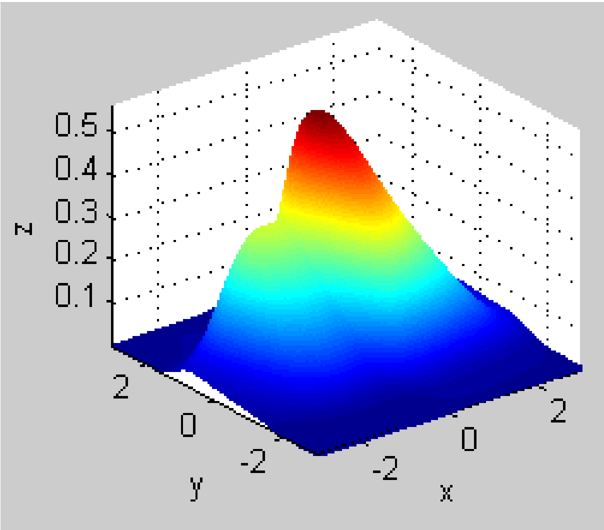
- 采用150个样本，学习速率0.1，动量因子0.4，隐节点7个，训练步数15000步

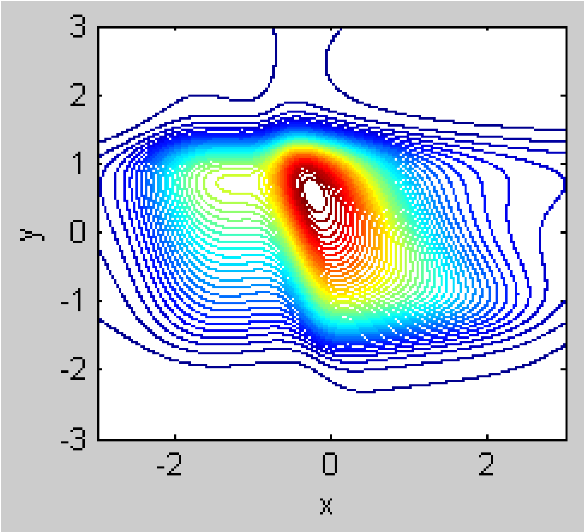
输入-隐层权值矩阵：$\begin{bmatrix} -6.695 & -2.753 & -0.305 & 0.722 & 0.357 & -0.096 & 1.78 \\ -1.032 & -0.152 & -0.593 & -0.583 & -3.826 & 4.936 & 1.997 \end{bmatrix}$

隐层-输出层权矩阵：$[-1.862 \quad -3.590 \quad -1.866 \quad -3.451 \quad -3.302 \quad -3.986 \quad -2.581]^T$

输入层阈值权值矩阵：$[3.207 \quad 6.596 \quad 0.062 \quad -0.211 \quad 6.825 \quad 7.257 \quad 2.365]^T$

隐层阈值权值：-3.179





仿真结果

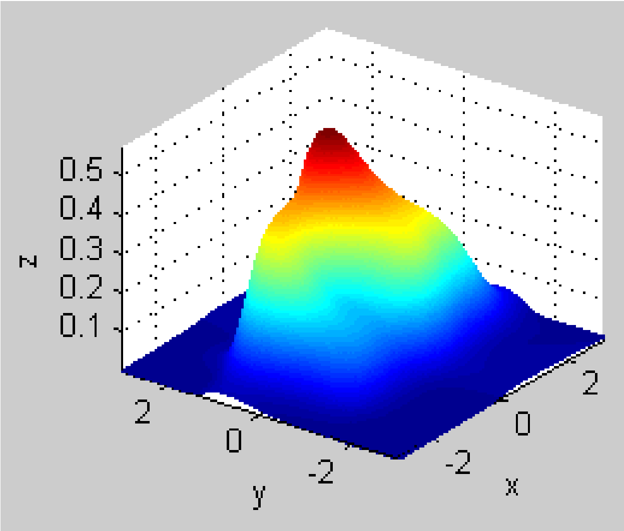
- 200个样本，其他参数同前

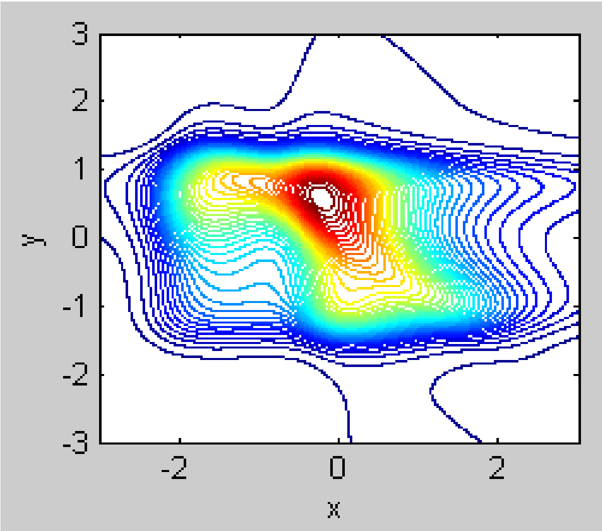
输入-隐层权值矩阵：$\begin{bmatrix} -5.38 & -3.83 & 0.333 & 0.252 & 0.265 & 1.058 & -3.866 \\ -0.53 & 0.373 & -2.557 & -4.846 & 5.854 & 1.293 & -4.788 \end{bmatrix}$

隐层一输出权值矩阵：$[-1.851 \quad -4.417 \quad -2.695 \quad -4.298 \quad -3.335 \quad -5.360 \quad -1.217]^T$

输入层阈值矩阵：$[2.433 \quad 9.114 \quad -1.060 \quad 7.646 \quad 7.529 \quad 0.576 \quad 0.950]^T$

隐层阈值：-4.328





仿真结果

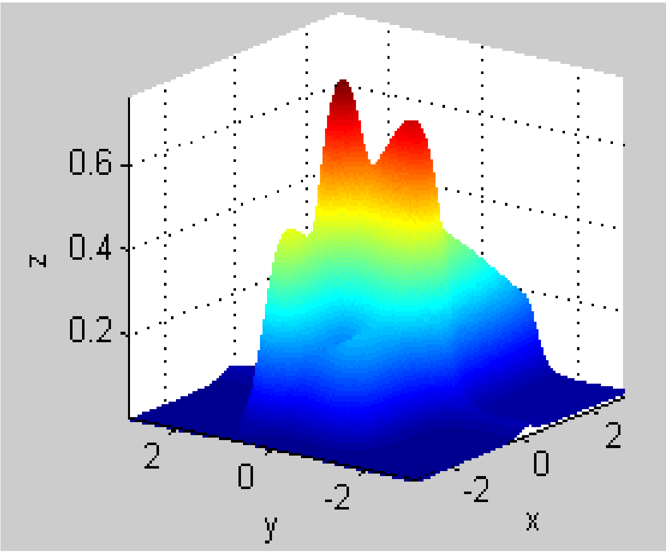
- 200个带信息的样本：一半随机取，一半来自特征区域 $x\in[-2,2],\quad y\in[-1,1]$

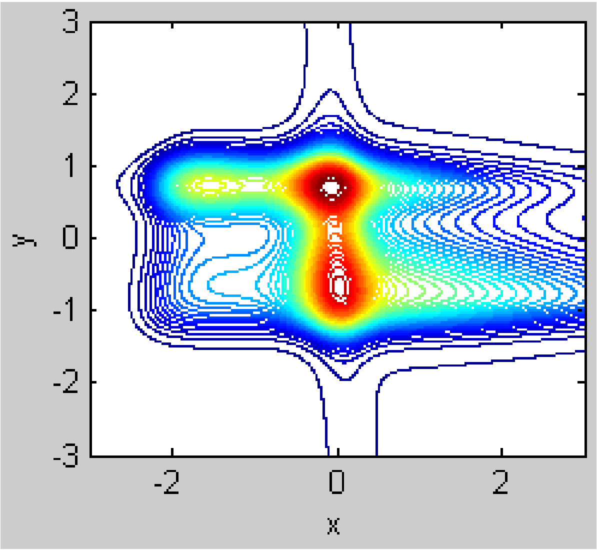
输入-隐层权值矩阵：$\begin{bmatrix} -4.452 & 0.507 & 0.197 & 6.846 & 0.49 & -4.489 & -0.117 \\ 0.567 & 1.792 & -5.203 & 0.186 & 4.692 & -0.302 & 5.472 \end{bmatrix}$

隐层-输出权值矩阵：$\begin{bmatrix} -5.925 & -4.921 & -4.762 & -2.180 & -4.400 & -3.579 & 3.921 \end{bmatrix}^T$

输入层阈值矩阵：$\begin{bmatrix} 10.826 & 0.859 & 6.982 & 1.426 & 5.359 & 1.341 & 2.809 \end{bmatrix}^T$

隐层阈值：-2.767





思考

- FNN可用于挖掘样本输入与输出之间的关系，即数据建模。那么，基于一组样本，是否可能找到某一输入量使得输出近似最大或最小呢？
- **数据驱动的优化**