信陽師範學院
数学与统计学院
SCHOOL OF MATHEMATICS AND STATISTICS

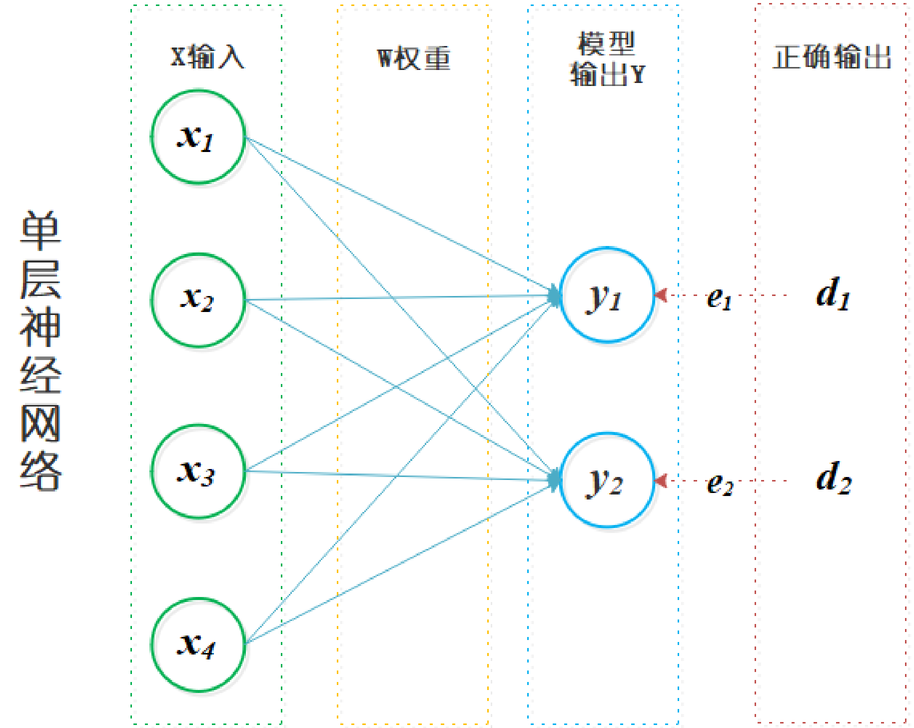
第14章 神经网络与深度学习

讲授人：牛言涛

日期：2020年5月15日

目录
CONTENTS

1. 单层神经网络模型



输入数据向量：$X=(x_0,x_1,\cdots,x_n)$ （矩阵表示: X）

输出数据向量：$Y=(y_0,y_1,\cdots,y_m)$ （矩阵表示: Y）

权重矩阵：

$$W=\begin{pmatrix} w_{00} & w_{02} & \cdots & w_{0n} \\ w_{11} & w_{12} & \cdots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix}$$

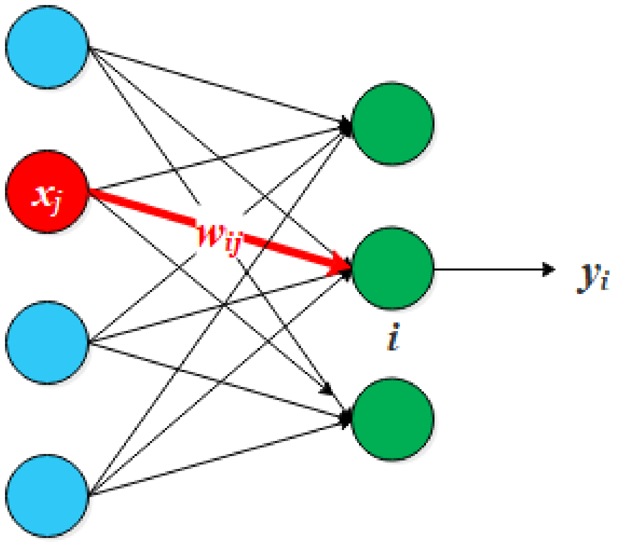
- 单层神经网络的计算输出公式： $net_{out}=WX^T+W_b$
  - X：输入特征数据，使用行向量表示.
  - W_b：表示加权求和的偏置项。
- 如果考虑激活函数，则计算输出公式为：$out_Y=\varphi_{acvivity}(WX^T+W_b)$

1. 单层神经网络模型

- 偏置量的作用是给网络分类增加平移的能力，对神经元激活状态的控制。因为输入X数据不是以0为中心分布的。
- 激活函数的主要作用是提供网络的非线性建模能力。如果没有激活函数，那么该网络仅能够表达线性映射，此时即便有再多的隐藏层，其整个网络跟单层神经网络也是等价的。因此也可以认为，只有加入了激活函数之后，深度神经网络才具备了分层的非线性映射学习能力。
  1. 可微性： 当优化方法是基于梯度的时候，这个性质是必须的。
  2. 单调性： 当激活函数是单调的时候，单层网络能够保证是凸函数。
  3. 输出值的范围： 当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著;当激活函数的输出是无限的时候，模型的训练会更加高效。
- 常见的激活函数：sigmoid函数，tanh函数，Relu函数及其改进型（如Leaky-ReLU、P-ReLU、R-ReLU等），Swish函数(Google大脑团队)，Gated linear units(GLU)函数(facebook提出)。

2. 增量规则

- 神经网络以权重的形式存储信息，为了能用新的信息训练神经网络，权重应做出相应的变化。根据给定的信息来修正权重的系统性方法叫学习规则。
- 增量规则(delta rule)是典型的单层神经网络的学习规则。



$$e_i \overset{def}{=} d_i - y_i$$

- 增量规则按如下运算调整权重：“如果一个输入节点对输出节点的误差有贡献，那么这两个节点间的权重应当以输入值x_j和输出值误差e_i成比例地调整。”公式为：

$$w_{ij} \leftarrow w_{ij} + \alpha e_i x_j$$

- 学习率$\alpha(0 < \alpha \leq 1)$决定了每次权重的改变量。如果该值太大，则输出就会在其解的周围徘徊（震荡）；相反，如果该值太小，则趋近真解的计算过程会非常慢。

3. 单层神经网络训练

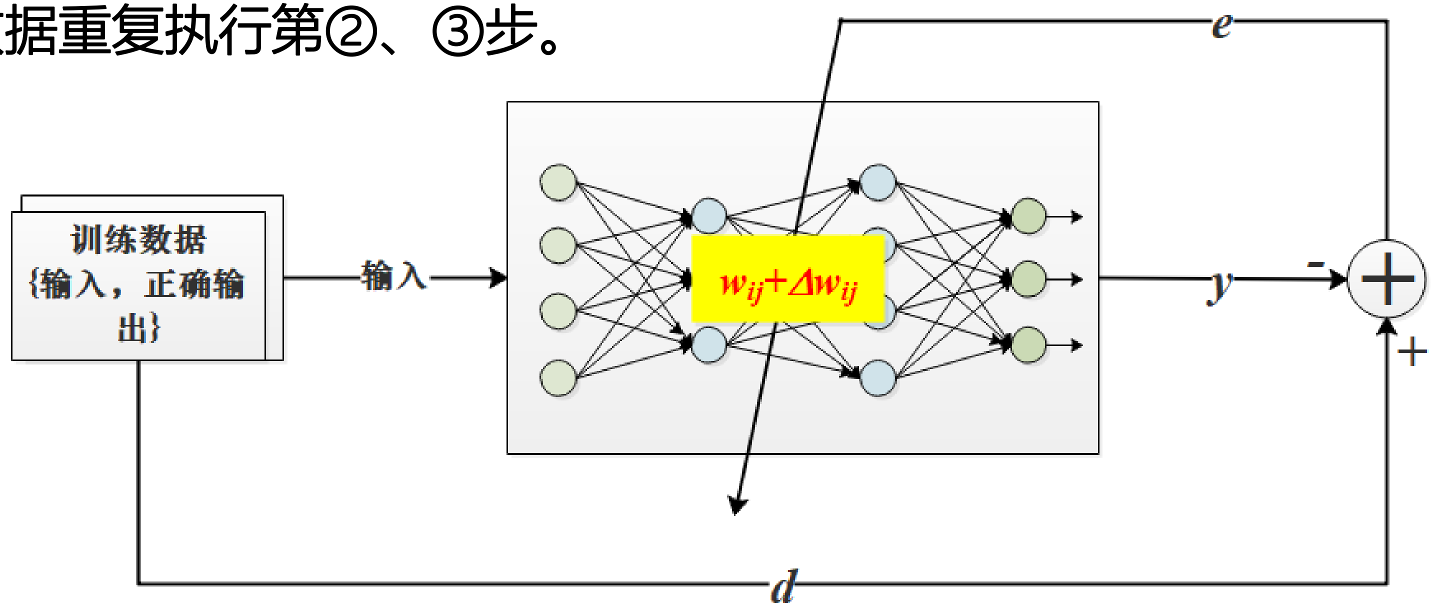
① 用适当的值初始化权重；

② 从训练数据中获得“输入”，将“输入”传递到神经网络模型中，从模型获得输出，并依据“正确输出”计算误差$e_i = d_i - y_i$；

③ 按照增量规则计算权重的更新，即$\Delta w_{ij} = \alpha e_i x_j$；

④ 调整权重，即$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} = w_{ij} + \alpha e_i x_j$；

⑤ 将所有训练数据重复执行第②、③步。



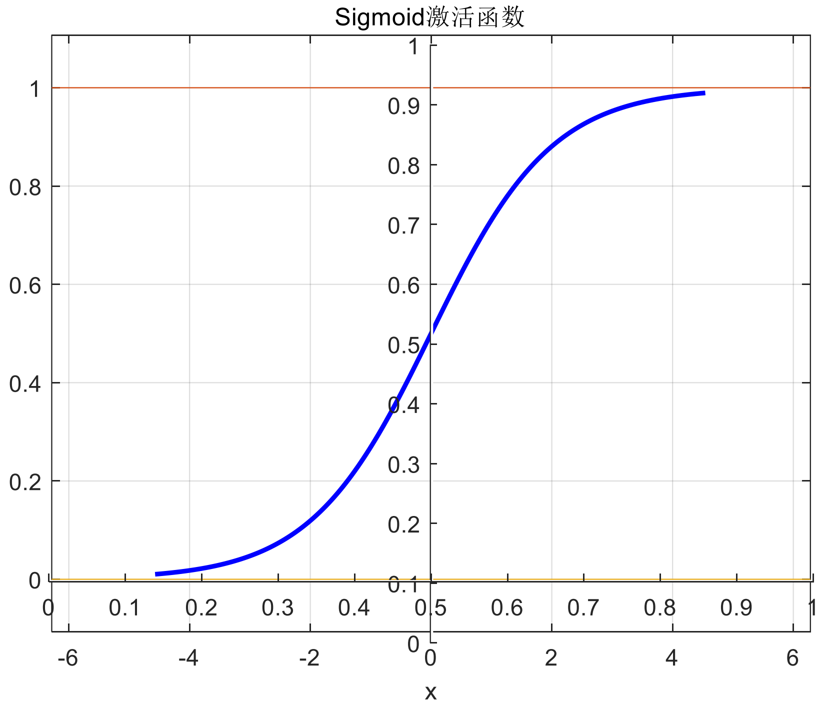
4. 广义增量规则

- 对于任意一个激活函数，都可以用$w_{ij} \leftarrow w_{ij} + \alpha\delta_i x_j$表示增量规则。定义$\delta_i = \varphi'(v_i)e_i$：
  - e_i =输出节点i的误差
  - v_i =输出节点i的加权和
  - φ' =输出节点i的激活函数的导数。
- 线性激活函数$\varphi(x) = x$，则$\varphi'(x) = 1$，带入得$\delta_i = e_i$，针对线性有效。
- Sigmoid激活函数：

$$\varphi(x) = \frac{1}{1+e^{-x}}, \ \varphi'(x) = \varphi(x)[1-\varphi(x)]$$

$$\delta_i = \varphi'(v_i)e_i$$

$$w_{ij} \leftarrow w_{ij} + \alpha\varphi(v_i)[1-\varphi(v_i)]e_i x_j$$

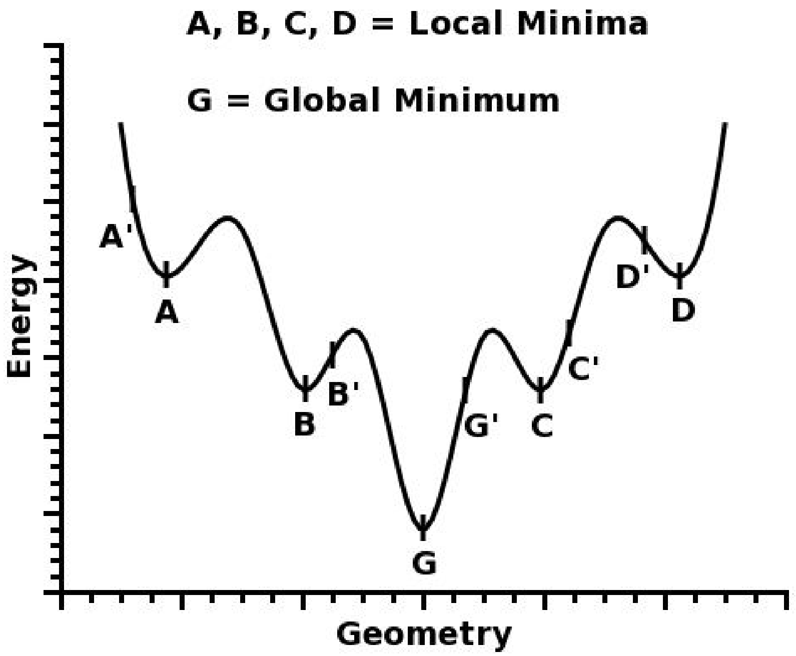


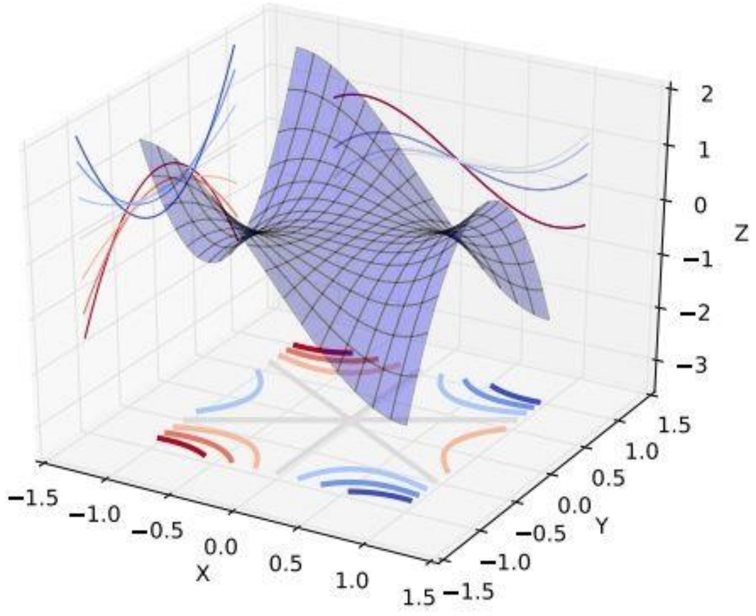
权重与输出节点误差e_i和输入节点值x_j成正比。

5. 神经网络的监督学习过程

信阳师范学院
数学与统计学院
SCHOOL OF MATHEMATICS AND STATISTICS

- 梯度下降法的基本思想：通过梯度下降法，使得网络参数不断收敛到全局（或者局部）最小值，但是由于神经网络层数太多，需要通过反向传播算法，把误差一层一层地从输出传播到输入，逐层地更新网络参数。由于梯度方向是函数值变大的最快的方向，因此负梯度方向则是函数值变小的最快的方向。沿着负梯度方向一步一步迭代，便能快速地收敛到函数最小值。
- 局部最优是神经网络优化的主要难点，只有当函数为凸函数时，才是全局最优。





5. 神经网络的监督学习过程

1、随机梯度下降算法

随机梯度下降(Stochastic Gradient Descent, SGD)法计算每一个训练数据的误差并随机调整权值。假如有100个训练数据点，则SGD算法将进行100次的权重调整。

因为SGD算法对每一个数据点都调整权重，所以在训练过程中，神经网络的性能是变化的。"随机"暗含了训练过程的随机性。用SGD算法计算权重更新值公式为：$\Delta w_{ij} = \alpha \delta_i x_j$.

2、批量算法

在批量(batch)算法中，对于每一个权重，使用全部训练数据分别计算出它的权重更新值，然后用这些权重的平均值来调整权重。批量算法每次都用到所有的训练数据，最后只更新权重一次。

$$\Delta w_{ij} = \frac{1}{N}\sum_{k=1}^{N}\Delta w_{ij}(k)$$

其中：$\Delta w_{ij}(k)$ =第k个训练数据的权重更新值；N =训练数据的总个数。由于是计算平均权重更新值，所以批量算法需要消耗较长的训练时间。

5. 神经网络的监督学习过程

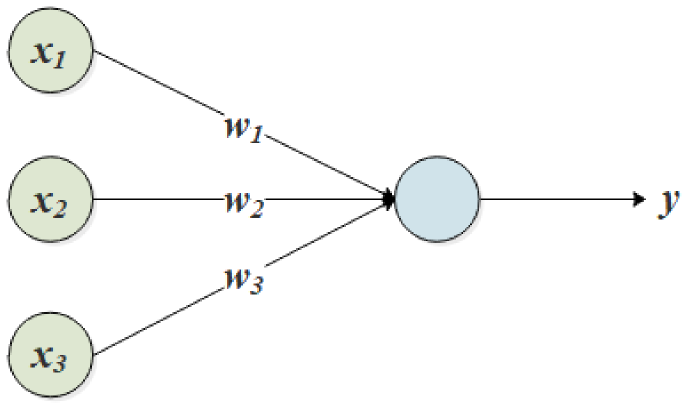
3、小批量算法

小批量(minibatch)算法是SGD算法和批量算法的混合形式。即选出一部分数据集，用批量算法训练这个数据集。如100个训练数据点中任意选取20个数据点，批量算法应用于这20个数据点上即可。如此，为了完成所有数据点的训练，需进行5次权重调整。

当确定合适的小批量数据点的个数时，小批量算法可以兼得这两种算法的优势：SGD算法的高速度和批量算法的稳定性。故此，小批量算法常常被应用于需要处理大量数据的深度学习模型。

6. 案例：增量规则

- 考虑最简单形式：三个输入节点、一个输出节点的单层神经网络



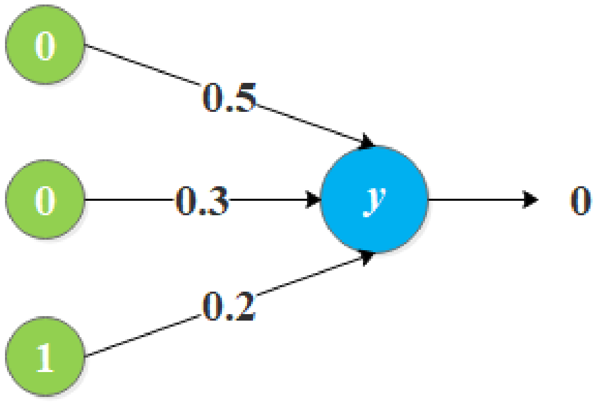
输入节点			输出节点
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1

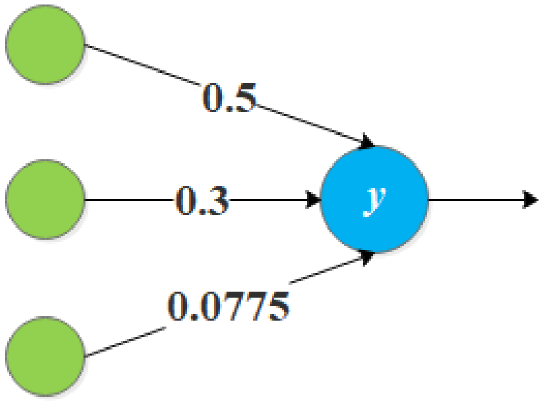
- 采用Sigmoid函数的增量规则作为学习规则。

$$\begin{cases} \delta_i = \varphi(v_i)\left[1-\varphi(v_i)\right]e_i \\ \Delta w_{ij} = \alpha\delta_i x_j \\ w_{ij} = w_{ij} + \Delta w_{ij} \end{cases}$$

6. 案例：增量规则

输入节点			输出节点
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1





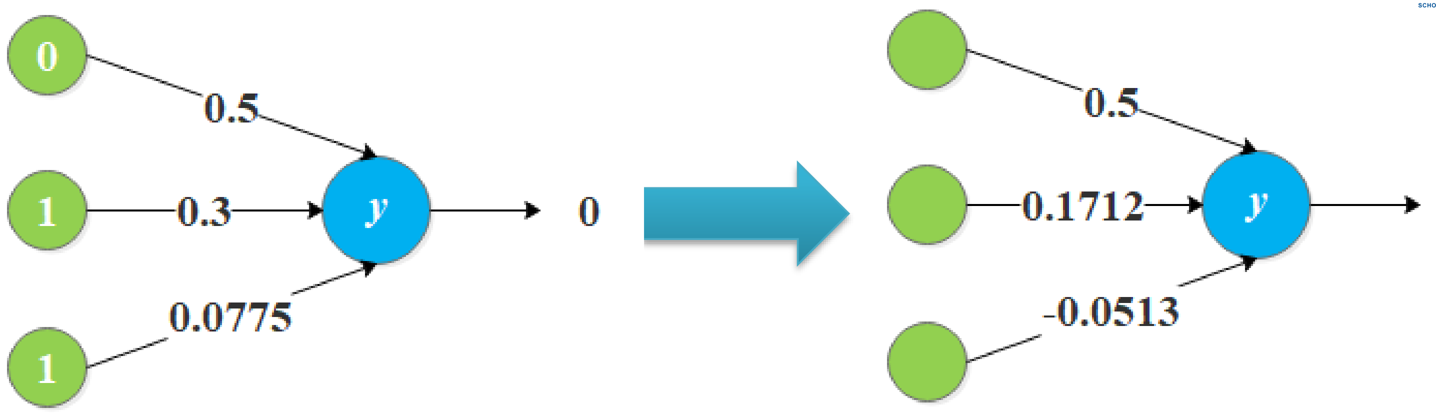
$$\begin{cases} v_1 = 0\times 0.5 + 0\times 0.3 + 1\times 0.2 = 0.2 \\ y_1 = \varphi(v_1) = \dfrac{1}{1+e^{-v_1}} = \dfrac{1}{1+e^{-0.2}} = 0.5498 \\ \delta_1 = \varphi(v_1)\left[1-\varphi(v_1)\right]e_1 = 0.5498(1-0.5498)(0-0.5498) = -0.1361 \\ \Delta w = \alpha\delta_1 x(1) = 0.9\times(-0.1361)\begin{bmatrix}0 & 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 0 & -0.1225\end{bmatrix} \\ w = w + \Delta w = \begin{bmatrix}0.5 & 0.3 & 0.2\end{bmatrix} + \begin{bmatrix}0 & 0 & -0.1225\end{bmatrix} = \begin{bmatrix}0.5 & 0.3 & 0.0775\end{bmatrix} \end{cases}$$

$$\begin{cases} \delta_i = \varphi(v_i)\left[1-\varphi(v_i)\right]e_i \\ \Delta w_{ij} = \alpha\delta_i x_j \\ w_{ij} = w_{ij} + \Delta w_{ij} \end{cases}$$

6. 案例：增量规则

信阳师范学院
数学与统计学院
SCHOOL OF MATHEMATICS AND STATISTICS

输入节点			输出节点
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1

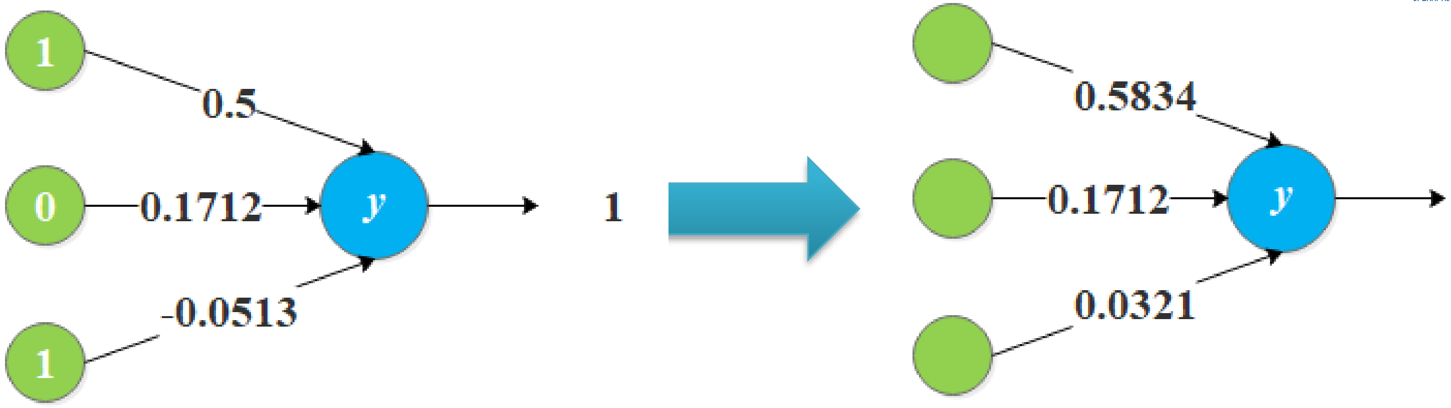


$$\begin{cases} v_2 = 0\times 0.5 + 1\times 0.3 + 1\times 0.0775 = 0.3775 \\ y_2 = \varphi(v_2) = \dfrac{1}{1+e^{-v_2}} = \dfrac{1}{1+e^{-0.3775}} = 0.5933 \\ \delta_2 = \varphi(v_2)\left[1-\varphi(v_2)\right]e_2 = 0.5933(1-0.5933)(0-0.5933) = -0.1432 \\ \Delta w = \alpha\delta_2 x(2) = 0.9\times(-0.1432)\begin{bmatrix}0 & 1 & 1\end{bmatrix} = \begin{bmatrix}0 & -0.1288 & -0.1288\end{bmatrix} \\ w = w + \Delta w = \begin{bmatrix}0.5 & 0.3 & 0.0775\end{bmatrix} + \begin{bmatrix}0 & -0.1288 & -0.1288\end{bmatrix} = \begin{bmatrix}0.5 & 0.1712 & -0.0513\end{bmatrix} \end{cases}$$

$$\begin{cases} \delta_i = \varphi(v_i)\left[1-\varphi(v_i)\right]e_i \\ \Delta w_{ij} = \alpha\delta_i x_j \\ w_{ij} = w_{ij} + \Delta w_{ij} \end{cases}$$

6. 案例：增量规则

输入节点			输出节点
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1

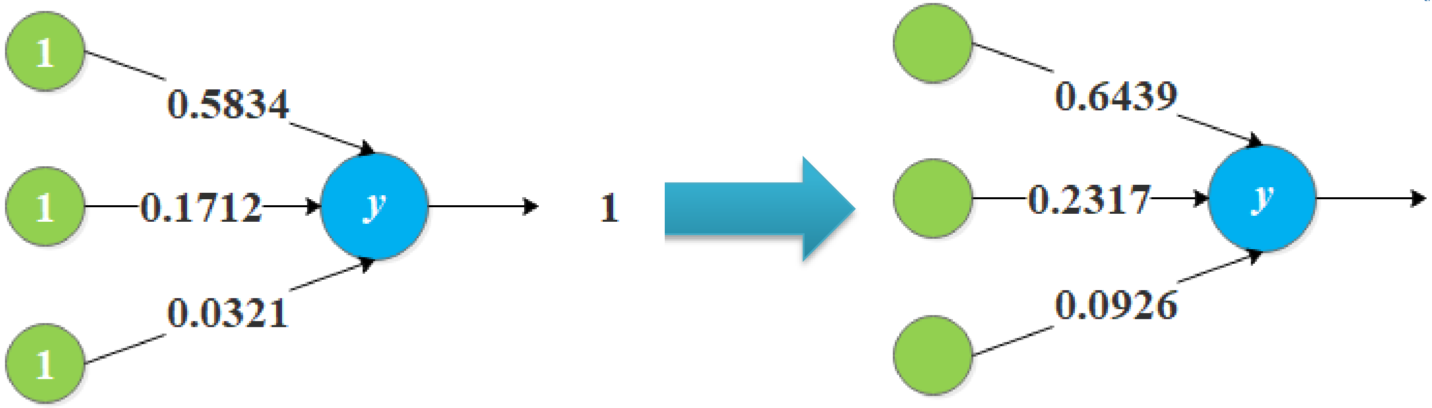


$$\begin{cases} v_3 = 1\times 0.5 + 0\times 0.1712 - 1\times 0.0513 = 0.4487 \\ y_3 = \varphi(v_3) = \dfrac{1}{1+e^{-v_3}} = \dfrac{1}{1+e^{-0.4487}} = 0.6103 \\ \delta_3 = \varphi(v_3)\left[1-\varphi(v_3)\right]e_3 = 0.6103(1-0.6103)(1-0.6103) = 0.0927 \\ \Delta w = \alpha\delta_3 x(3) = 0.9\times 0.0927\times\begin{bmatrix}1 & 0 & 1\end{bmatrix} = \begin{bmatrix}0.0834 & 0 & 0.0834\end{bmatrix} \\ w = w + \Delta w = \begin{bmatrix}0.5 & 0.1712 & 0.0513\end{bmatrix} + \begin{bmatrix}0.0834 & 0 & 0.0834\end{bmatrix} = \begin{bmatrix}0.5834 & 0.1712 & 0.0321\end{bmatrix} \end{cases}$$

$$\begin{cases} \delta_i = \varphi(v_i)\left[1-\varphi(v_i)\right]e_i \\ \Delta w_{ij} = \alpha\delta_i x_j \\ w_{ij} = w_{ij} + \Delta w_{ij} \end{cases}$$

6. 案例：增量规则

输入节点			输出节点
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	1



$$\begin{cases} v_4 = 1\times 0.5834 + 1\times 0.1712 + 1\times 0.0321 = 0.7867 \\ y_4 = \varphi(v_4) = \dfrac{1}{1+e^{-v_4}} = \dfrac{1}{1+e^{-0.7867}} = 0.6871 \\ \delta_4 = \varphi(v_4)\left[1-\varphi(v_4)\right]e_4 = 0.6871(1-0.6871)(1-0.6871) = 0.0673 \\ \Delta w = \alpha\delta_4 x(4) = 0.9\times 0.0673\times\left[1 \quad 1 \quad 1\right] = \left[0.0605 \quad 0.0605 \quad 0.0605\right] \\ w = w+\Delta w = \left[0.5834 \quad 0.1712 \quad 0.0321\right]+\left[0.0605 \quad 0.0605 \quad 0.0605\right] = \left[0.6439 \quad 0.2317 \quad 0.0926\right] \end{cases}$$

$$\begin{cases} \delta_i = \varphi(v_i)\left[1-\varphi(v_i)\right]e_i \\ \Delta w_{ij} = \alpha\delta_i x_j \\ w_{ij} = w_{ij} + \Delta w_{ij} \end{cases}$$

至此一遍训练完成。

7. 增量规则——SGD实现

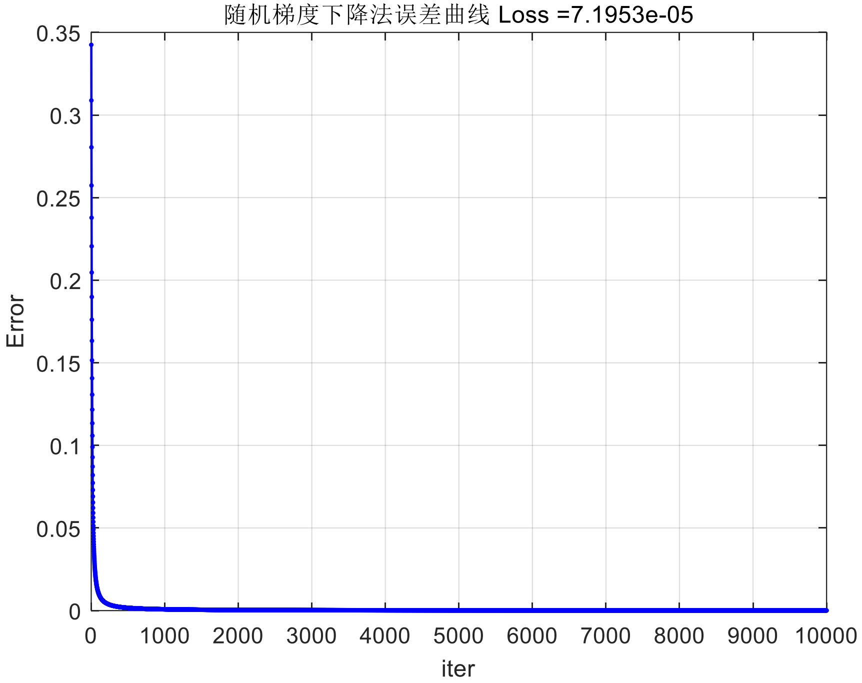
DeltaSGD.m

```
function [W,Y,Error] = DeltaSGD(X,D,alpha,iter)
% DeltaSGD实现单层神经网络按照增量规则的随机梯度下降法
% X是输入数据，一行表示一个观测数据；D是标准正确输出，一行表示对应X的标准输出；
% alpha是学习率0~1，iter是最大训练次数
% 输出W是训练网络的最终权值，Y是最终输出，Error为误差向量

    %% 数据的初始化和预处理
    fn = size(X,2); %X的特征数
    W = 2*rand(1,fn) - 1; %随机生成权重，范围[-1,1]
    n = size(X,1);  %训练数据数

    %% 网络训练
    Error = zeros(1,iter);
    for i = 1:iter  %训练次数
        for k = 1:n
            x = X(k,:)';  %取训练数据的每一行输入
            d = D(k);  %取训练数据的每一行标准输出
            v = W * x; %输出节点的权重和
            y = Sigmoid(v);  %经过Sigmoid函数激活
            e = d - y;    %输出值与真实值的误差
            delta = y*(1-y)*e;
            dW = alpha * delta * x;   % delta rule
            W = W + dW';  %更新权值
        end
        Error(i) = mean((Sigmoid(W * X') - D').^2);
    end

    %% 误差可视化
    plot(Error,'b.-','LineWidth',1);
    title(strcat('随机梯度下降法误差曲线 Loss = ',...
        num2str(Error(end))))
    xlabel('iter'); ylabel('Error');grid on

    %% 最终输出
    Y = zeros(1,n);
    for k = 1:n
        Y(k) = Sigmoid(W * X(k,:)');
    end

    %% 激活函数
    function y = Sigmoid(x)
        y = 1 ./ (1 + exp(-x));
    end
end
```

7. 增量规则——SGD实现

```
>> X = [0 0 1;0 1 1;1 0 1;1 1 1];
>> D = [0 0 1 1]';
>> [W,Y] = DeltaSGD(X,D,0.9,10000)
W =
    9.5648   -0.2092   -4.5746
Y =
    0.0102    0.0083    0.9932    0.9917
```

$$\begin{bmatrix} 0.0102 \\ 0.0083 \\ 0.9932 \\ 0.9917 \end{bmatrix} \Leftrightarrow D = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



- 从结果可以看出，输出非常解决于D的正确输出，说明神经网络得到了正确的训练。
- 误差损失曲线下降收敛。
- 最终权重向量为[9.5648 -0.2092 -4.5746]。
- 学习率采用0.8，迭代次数增加到100000，进行神经网络训练，结果理想。

```
>> [W,Y] = DeltaSGD(X,D,0.8,100000)
W =
   11.7838   -0.2047   -5.6876
Y =
    0.0034    0.0028    0.9978    0.9972
```

7. 增量规则——批量算法实现

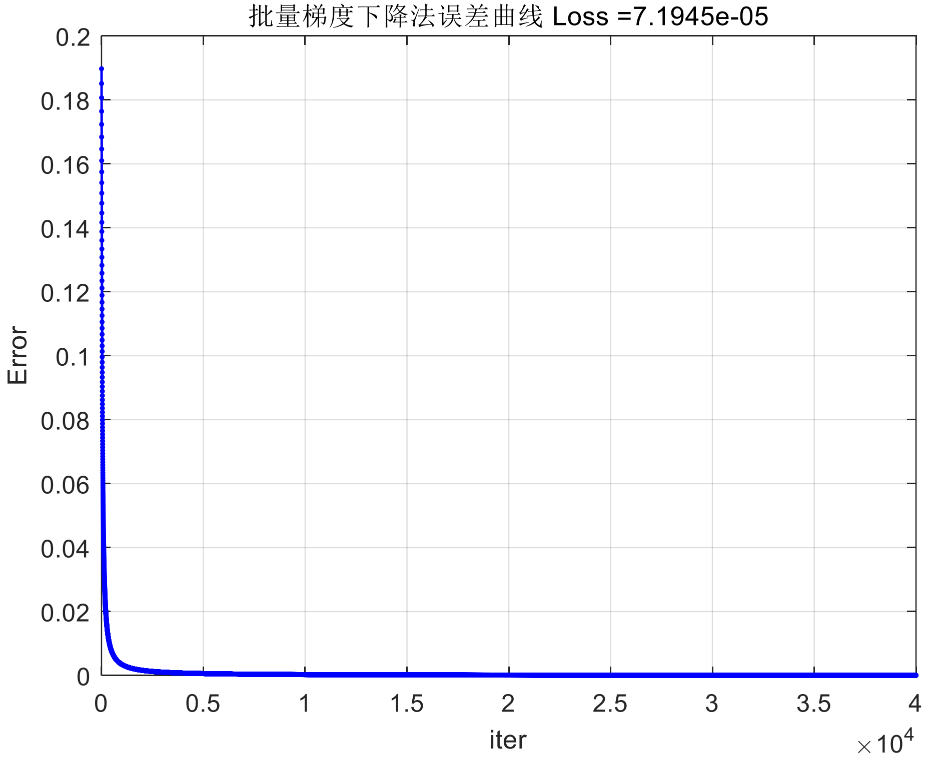
DeltaGDBatch.m

```matlab
function [W,Y,Error] = DeltaGDBatch(X,D,alpha,iter)
% DeltaGDBatch实现单层神经网络按照增量规则的批量梯度下降法
% X是输入数据，一行表示一个观测数据；D是标准正确输出，一行表示对应X的标准输出；
% alpha是学习率0~1，iter是最大训练次数
% 输出W是训练网络的最终权值，Y是最终输出，Error为误差向量

    %% 数据的初始化和预处理
    fn = size(X,2); %X的特征数
    W = 2*rand(1,fn) - 1; %随机生成权重，范围[-1,1]
    n = size(X,1);  %训练数据数

    %% 网络训练
    Error = zeros(1,iter);
    for i = 1:iter  %训练次数
        dWsum = zeros(fn,1);
        for k = 1:n
            x = X(k,:)';  %取训练数据的每一行输入
            d = D(k);  %取训练数据的每一行标准输出
            v = W * x; %输出节点的权重和
            y = Sigmoid(v);  %经过Sigmoid函数激活
            e = d - y;    %输出值与真实值的误差
            delta = y*(1-y)*e;
            dW = alpha * delta * x;   % delta rule
            dWsum = dWsum + dW; %权值求和
        end
        dWavg = dWsum / n; %权重的平均值来调整权重
        W = W + dWavg'; %更新权值
        Error(i) = mean((Sigmoid(W * X') - D').^2);
    end

    %% 误差可视化
    plot(Error,'b.-','LineWidth',1);
    title(strcat('批量梯度下降法误差曲线 Loss = ',...
        num2str(Error(end))))
    xlabel('iter'); ylabel('Error');grid on

    %% 最终输出
    Y = zeros(1,n);
    for k = 1:n
        Y(k) = Sigmoid(W * X(k,:)');
    end

    %% 激活函数
    function y = Sigmoid(x)
        y = 1 ./ (1 + exp(-x));
    end
end
```

7. 增量规则——批量算法实现

```
>> [W,Y] = DeltaGDBatch(X,D,0.9,10000)
W =
    8.1246   -0.2158   -3.8494
Y =
    0.0208    0.0169    0.9863    0.9830
>> [W,Y] = DeltaGDBatch(X,D,0.9,40000)
W =
    9.5647   -0.2086   -4.5749
Y =
    0.0102    0.0083    0.9932    0.9917
```

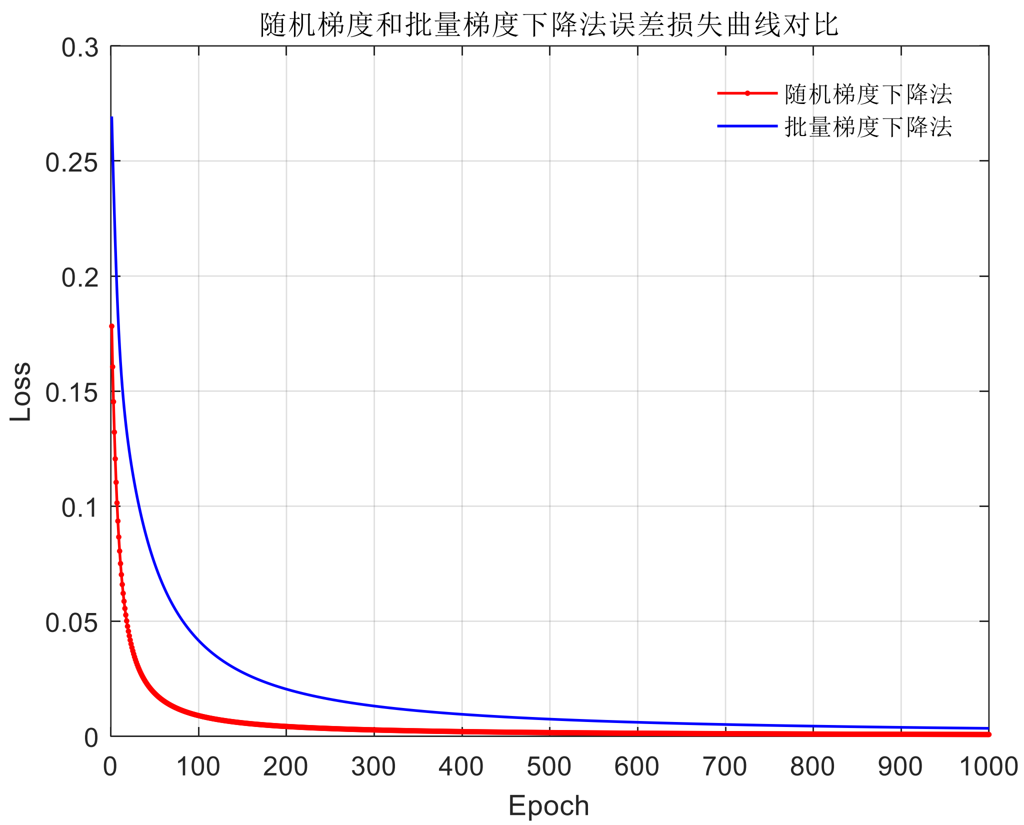
$$\begin{bmatrix} \mathbf{0.0102} \\ \mathbf{0.0083} \\ \mathbf{0.9932} \\ \mathbf{0.9917} \end{bmatrix} \Leftrightarrow D = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{1} \end{bmatrix}$$



从结果可以看出，训练40000次的结果与SGD算法的10000次精度相当，<u>故批量算法的学习速度比较慢。</u>

SGD与批量算法的比较

```
>> [W,Y,ErrSGD] = DeltaSGD(X,D,0.9,1000);
>> [W,Y,ErrGDB] = DeltaGDBatch(X,D,0.9,1000);
>> plot(ErrSGD,'r.-','LineWidth',1)
>> hold on
>> plot(ErrGDB,'b-','LineWidth',1)
>> grid on
>> legend('随机梯度下降法','批量梯度下降法')
>> legend('boxoff')
>> xlabel('Epoch');ylabel('Loss')
>> title('随机梯度和批量梯度下降法误差损失曲线对比')
```



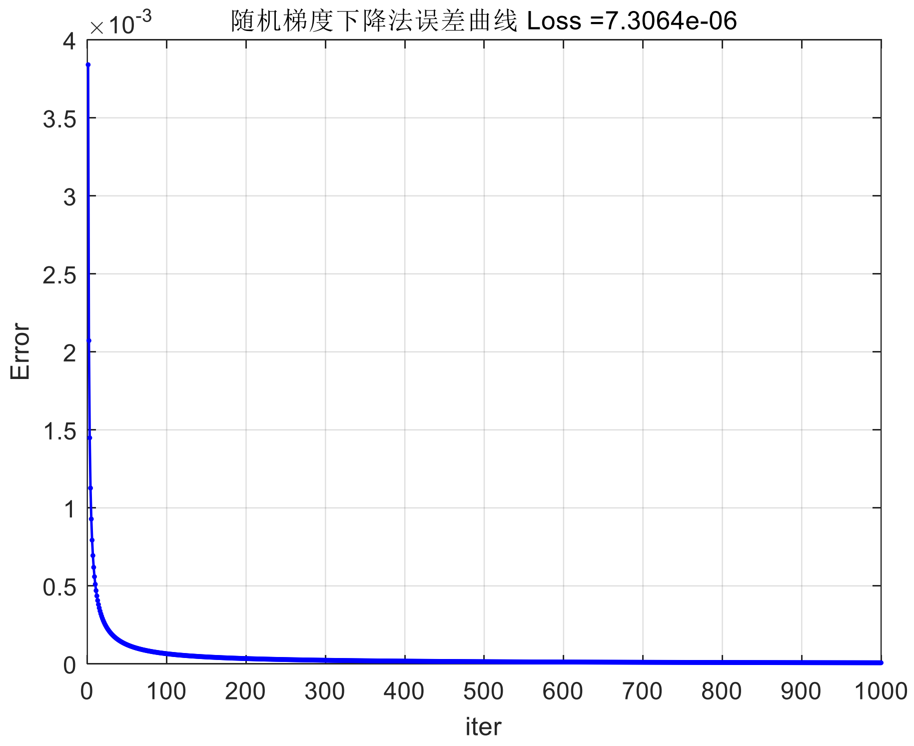
表明SGD算法比批量算法能更快地降低学习误差，也即SGD算法的学习速度更快。

7. 增量规则——线性分类案例

例1：选取鸢尾花前100行两类别数据，样本属性共4个，进行单层神经网络训练并判别。

iris_linearClass.m

```
load fisheriris
X = zscore(meas(1:100,:));  %选取二分类数据
D = [zeros(50,1);ones(50,1)]; %类别标签
[W,Y,~] = DeltaSGD(X,D,0.9,1000); %训练单层神经网络
Yc = zeros(1,length(D)); %初始化预测值
acc = 0; %正确数
for i = 1:length(D) %对每个预测输出进行判别
    if Y(i) < 0.5  %激活函数输出范围0-1，规定小于0.5为1类
        Yc(i) = 0;
    elseif Y(i) > 0.5  %大于0.5为2类
        Yc(i) = 1;
    end
    if Yc(i) == D(i)  %判断与标准输出是否一致
        acc = acc + 1; %累加
    end
end
accrate = acc/length(D)*100  %正确率
```




```
>> iris_linearClass
accrate =
    100
```

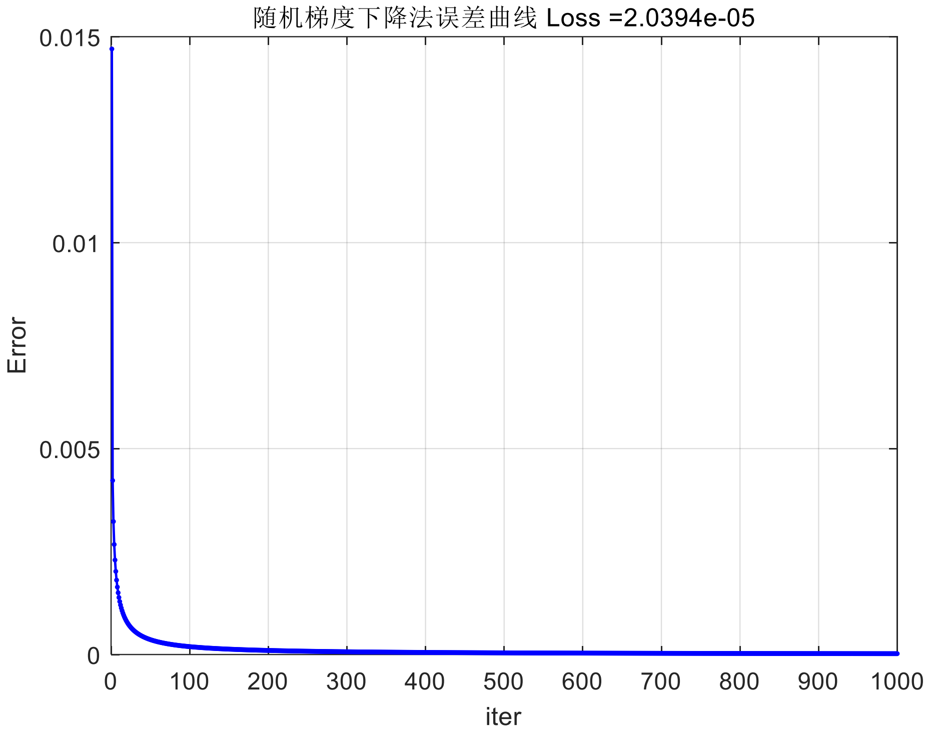
网络训练1000次，损失误差曲线仍然是随着迭代次数的增加而下降收敛，分类正确率为100%，权重W共4个。

7. 增量规则——线性分类案例

例2：选取wind酒前130行两类别数据，样本属性共13个，进行单层神经网络训练并判别。

wine_linearClass.m

```
load wine.data
X = zscore(wine(1:130,2:end)); %选取二分类数据
D = [zeros(59,1);ones(71,1)]; %类别标签
[W,Y,~] = DeltaSGD(X,D,0.9,1000); %训练单层神经网络
Yc = zeros(1,length(D)); %初始化预测值
acc = 0; %正确数
for i = 1:length(D) %对每个预测输出进行判别
    if Y(i) < 0.5  %激活函数输出范围0-1，规定小于0.5为1类
        Yc(i) = 0;
    elseif Y(i) > 0.5  %大于0.5为2类
        Yc(i) = 1;
    end
    if Yc(i) == D(i)  %判断与标准输出是否一致
        acc = acc + 1; %累加
    end
end
accrate = acc/length(D)*100  %正确率
```



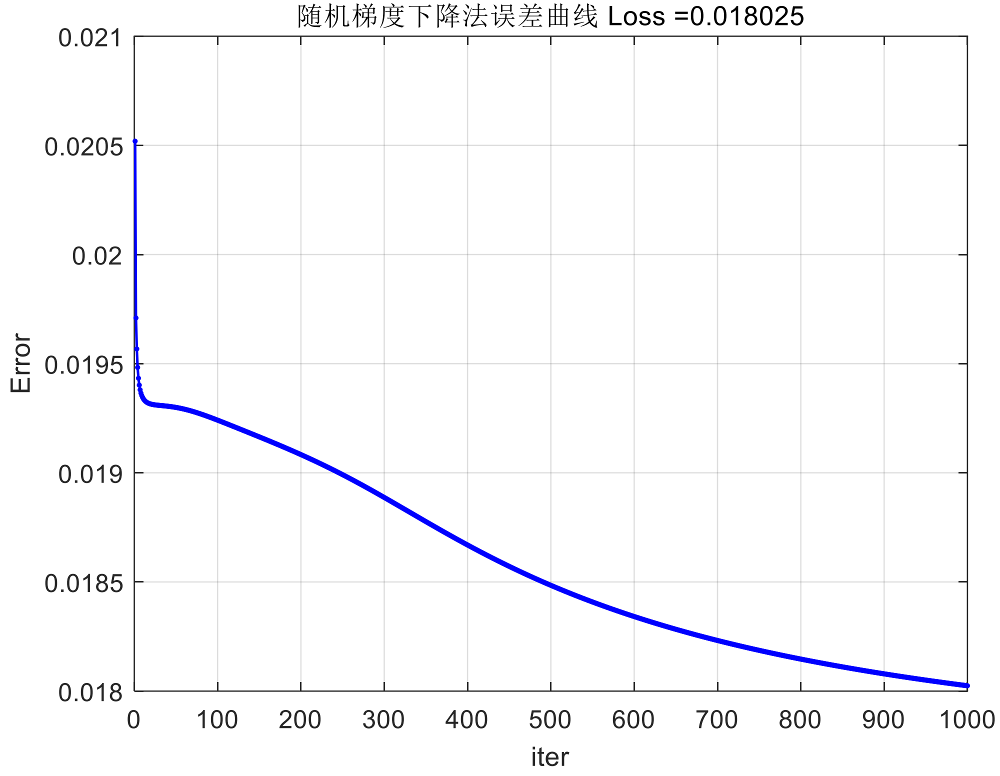

```
>> wine_linearClass
accrate =
   100
```

网络训练1000次，损失误差曲线仍然是随着迭代次数的增加而下降收敛，分类正确率为100%，权重W共13个。

7. 增量规则——线性分类案例

例3：癌症数据分类，样本属性共9个，进行单层神经网络训练并判别。

```
data = xlsread('breatcancerdata.xlsx');
data(:,1) = [];
X = zscore(data(:,1:end-1));  %选取二分类数据
D = zeros(size(X,1),1);
for i = 1:size(X,1) %类别标签编码
    if data(i,end) == 2
        D(i) = 0; %类别标签
    else
        D(i) = 1; %类别标签
    end
end
[W,Y,~] = DeltaSGD(X,D,0.9,1000); %训练单层神经网络
Yc = zeros(1,length(D)); %初始化预测值
acc = 0; %正确数
for i = 1:length(D) %对每个预测输出进行判别
    if Y(i) < 0.5  %激活函数输出范围0-1，规定小于0.5为1类
        Yc(i) = 0;
    elseif Y(i) > 0.5  %大于0.5为2类
        Yc(i) = 1;
    end
    if Yc(i) == D(i)  %判断与标准输出是否一致
        acc = acc + 1; %累加
    end
end
accrate = acc/length(D)*100  %正确率
```



网络训练1000次，损失误差曲线仍然是随着迭代次数的增加而下降收敛，分类正确率为98.24%，权重W共9个。

8. 单层神经网络的局限性

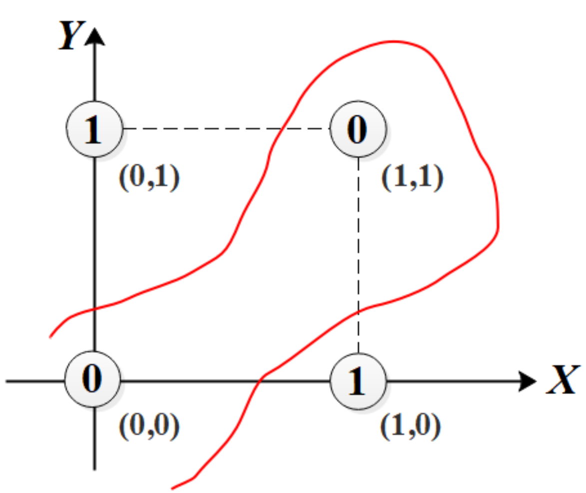
```
>> X = [0 0 1; 0 1 1; 1 0 1; 1 1 1];
>> D = [0 1 1 0]';   %改变输出变量的值
>> [W,Y] = DeltaSGD(X,D,0.9,100000)
W =
   -0.2375   -0.1188    0.1188
Y =
    0.5297    0.5000    0.4703    0.4409
>> [W,Y] = DeltaSGD(X,D,0.8,1000000)
W =
   -0.2099   -0.1050    0.1050
Y =
    0.5262    0.5000    0.4738    0.4477
```

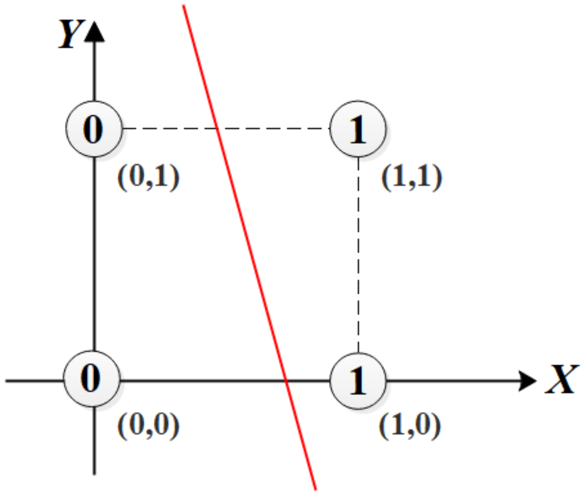
$$\begin{bmatrix} 0.5262 \\ 0.5000 \\ 0.4738 \\ 0.4477 \end{bmatrix} \neq D = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

单层神经网络只能解决线性可分割问题，因为单层神经网络是一种线性分割输入数据空间的模型。为了克服单层神经网络的限制，需要为网络增加更多的节点层。

8. 单层神经网络的局限性

左图，不能用一条直线将“0”和“1”的区域分割，只能用一条复杂的曲线分割区域，故是线性不可分割问题；对于线性可分割问题，如右图，则可以找到一条直线分割。





单层神经网络只能解决线性可分割问题，因为单层神经网络是一种线性分割输入数据空间的模型。

为了克服单层神经网络的限制，需要为网络增加更多的节点层。


01
IDEA
02
LEARNING
03
CREATIVE
INSPIRATION
EDUCATION
BUSINESS
PHOTO BOOK

感谢聆听