广东海洋大学

《MATLAB 语言及应用》课程论文

误差反向传播神经网络的 Matlab 实现及应用

学生姓名 杨子棵

专　　业 电子信息工程专业

班　　级 电子 1172

学　　号 201711611231

指导老师 徐国保

摘 要

误差反向传播神经网络是一种典型的人工神经网络，在一些数学问题的解决中起到了重要的作用。本文介绍了误差反向传播神经网络的原理和模型，对网络的前向传播和后向传播进行了详细的数学推导，并利用 Matlab 对网络进行了建模，通过分类问题对所得的模型进行训练、测试、分析，测试结果的正确率均在 99% 以上，最后对所构建的模型和测试情况进行了简要说明和分析。

本文的创新点主要有：

- 对误差反向传播神经网络进行了数学推导；
- 用 Matlab 实现误差反向传播神经网络；
- 用误差反向传播神经网络解决分类问题。

关键词：误差反向传播神经网络；训练；测试；Matlab；分类问题

Abstract

The Error Back Propagation Neural Network is a typical artificial neural network and plays an important role in solving some mathematical problems. This paper introduces the principle and model of the Error Back Propagation Neural Network, and makes a detailed mathematical derivation of the forward and backward propagation of the network, and uses Matlab to model the network, through classification problem. The classification problem is to train, test, and analyze the obtained model. The correct rate of the test results is above 99%. Finally, the model and test conditions are briefly described and analyzed.

The main innovations of this paper are:

- Mathematical Derivation of Error Back Propagation Neural Networks；
- Model Error Back Propagation Neural Network in Matlab；
- Using Error Back Propagation Neural Network to Solve Classification Problems。

Key Words: Error Back Propagation Neural Network; Training; Testing; Matlab; Classification problem

目 录

主要符号对照表

$Y^{(n)}$	神经网络第 n 层的输入向量
$Y_{(i)}^{(n)}$	神经网络第 n 层第 i 个神经元的输入
$V^{(n)}$	神经网络第 n 层的输出矩阵
$V_{(i)}^{(n)}$	神经网络第 n 层第 i 个神经元的输出
$W^{(n)}$	第 n 层与第 $n-1$ 层神经元之间的连接权值矩阵
$W_{ij}^{(n)}$	第 n 层的第 j 个神经元与第 $n-1$ 层的第 i 个神经元之间的连接权值
$\theta^{(n)}$	第 n 层神经元的阈值向量
$\theta_{(i)}^{(n)}$	第 n 层的第 i 个神经元的阈值
k	第 n 层神经元的个数
q	第 $n-1$ 层神经元的个数
T	期望值
δ_i	第 i 个神经元与期望值的误差
η	训练步长

第 1 章　误差反向传播神经网络概述

在机器学习和认知科学领域中，人工神经网络是一种模仿生物神经网络 (动物的中枢神经系统，特别是大脑) 的结构和功能的数学模型或计算模型，用于对函数进行估计或近似。神经网络由大量的人工神经元联结进行计算。大多数情况下人工神经网络能在外界信息的基础上改变内部结构，是一种自适应系统。

误差反向传播神经网络[1] 是一种与最优化方法（如梯度下降法）结合使用的，用来训练人工神经网络的常见方法。该方法对网络中所有权重计算损失函数的梯度。这个梯度会反馈给最优化方法，用来更新权值以最小化损失函数。反向传播要求有对每个输入值想得到的已知输出，来计算损失函数梯度。因此，它通常被认为是一种监督式学习方法，虽然它也用在一些无监督网络中。

误差反向传播神经网络的算法思想是：让信号正向传播和误差反向传播这两个过程交替循环进行，信号每正向传播一次之后都要计算一次误差，让误差沿梯度负方向下降一个很小的变化量，将这个误差变化量反向传播到神经网络的各层，然后对各层参数矩阵的值进行调整，之后再进行下一次循环。理论上讲，经过多次循环之后，神经网络的误差会收敛到一个较为稳定的范围之内，这时候就可以认为各层参数矩阵的值达到了理想值，即模型达到了最优状态。这种误差反向传播训练算法是人工神经网络的核心思想，当前比较流行的卷积神经网络、循环神经网络等训练算法都是在误差反向传播神经网络算法基础上经过优化和改进发展而来的。[2]

第 2 章　网络模型与原理

2.1　模型概述

在神经网络信号的正向传播中，神经元接收到来自其他神经元的输入信号，这些信号乘以权重累加到神经元接收的总输入值上，随后与当前神经元的阈值进行比较，然后通过激活函数处理，产生神经元的输出。理想的激活函数是阶跃函数，然而阶跃函数的缺点是不连续，不可导，所以常用 *sigmoid* 函数作为误差反向传播神经网络的激活函数。

误差反向传播神经网络的核心思想就是：通过调整各神经元之间的权值，将误差由隐含层向输入层逐层反传，也就是先实现信号的正向传播到误差的反向传播过程。所以误差反向传播神经网络算法的核心步骤如下：（1）求得在特定输入下实际输出与理想输出的平方误差函数（误差函数或者叫代价函数）；（2）利用误差函数对神经网络中的阈值以及连接权值进行求导；（3）根据梯度下降算法，对极小值进行逼近，当满足条件时，跳出循环。

在本篇论文中，将使用的是有监督的误差反向传播神经网络。故我们需要提供一个包括输入和期望输出的训练集，以在训练的过程中获得它相对于训练集的误差。正向传播：输入样本—输入层—各隐含层—输出层；若输出层的实际输出与期望的输出不符，则误差反传：误差表示—修正各层神经元的权值；直到网络输出的误差减少到可以接受的程度，或者进行到预先设定的学习次数为止。

2.2　模型推导

在反向传播神经网络中，通常选择的是 *sigmoid* 函数作为激活函数，这类函数的导数有一个很好的性质，就是自身相关。*sigmoid* 函数如式 (2-1)。

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2-1}$$

其导数如式 (2-2)

$$sigmoid'(x) = sigmoid(x)(1 - sigmoid(x)) \tag{2-2}$$

假定现有一个四层的神经网络，包括一层输入层、两层隐含层和一层输出层。假定各层的神经元数量分别为：6，4，3，2。接下来对各层的输入和输出变量进行定义，输入至输入层的输入向量如式 (2-3)；对应的第一层隐含层的输入向量如式 (2-3)；

第二层隐含层的输入向量为式 (2-3)；输出层的输入向量为式 (2-3)[3]

$$Y^{(0)} = [x_1, x_2, \cdots, x_6]^T = [a_1(0), a_2(0), a_3(0), a_4(0), a_5(0), a_6(0)]^T \tag{2-3}$$

$$Y^{(1)} = [a_1(1), a_2(1), a_3(1), a_4(1)]^T \tag{2-4}$$

$$Y^{(2)} = [a_1(2), a_2(2), a_3(2)]^T \tag{2-5}$$

$$Y^{(3)} = [a_1(3), a_2(3)]^T \tag{2-6}$$

假定第 n 层（$n >= 2$）神经元的个数为 i 个，第 $n-1$ 层的神经元个数为 j 个，定义第 n 层与第 $n-1$ 层之间的连接权值矩阵如式 (2-7)

$$W^{(n)} = \begin{bmatrix} w_{11}^{(n)} & \cdots & w_{1j}^{(n)} \\ \vdots & \cdots & \vdots \\ w_{i1}^{(n)} & \cdots & w_{ij}^{(n)} \end{bmatrix} \tag{2-7}$$

定义第 n 层神经元的阈值向量如式 (2-8)

$$\theta^{(n)} = [\theta_{(1)}^{(n)}, \theta_{(2)}^{(n)}, ..., \theta_{(n)}^{(n)}]^T \tag{2-8}$$

则第 n 层神经元的输出矩阵如式 (2-9)

$$V^{(n)} = W^{(n)} Y^{(n)} + \theta^n \tag{2-9}$$

第 n 层的输入与第 $n-1$ 层的输出的关系如 (2-10)

$$Y^{(n)} = f(V^{(n-1)}) \tag{2-10}$$

其中，f 为激活函数，常使用 $sigmoid$ 函数作为激活函数，故如式 (2-11)

$$Y^{(n)} = sigmoid(V^{(n-1)}) \tag{2-11}$$

目标是希望通过调整 W 使得输出和目标的误差最小，利用最小二乘思想：

$$E = \frac{1}{k} \sum_{l=1}^{k} (T_l - Y_l^{(n)})^2 = \frac{1}{k} \sum_{l=1}^{k} (\delta_l^{(n)})^2 \tag{2-12}$$

其中，δ_l 如式 (2-13)

$$\delta_l = (T_l - Y_l^{(n)}) \tag{2-13}$$

因为误差反向传播神经网络是反馈式网络，所以在更新权值时是从后向前更新的，首先更新的是最后一层的权值。所以有：

$$E = \frac{1}{k}\sum_{l=1}^{k}(T_l - Y_l^{(n)})^2 = \frac{1}{k}\sum_{l=1}^{k}(T_l - f(V_l^{(n)}))^2 = \frac{1}{k}\sum_{l=1}^{k}(T_l - \sum_{i=1}^{q} W_{li}^{q} Y_i^{(n-1)})^2 \tag{2-14}$$

因此 E 和 W 的求导关系为：

$$\frac{\partial E}{\partial W_{li}^{(n)}} = \frac{\partial E}{\partial Y_l^{(n)}}\frac{\partial Y_l^{(n)}}{\partial V_l^{(n)}}\frac{\partial V_l^{(n)}}{\partial W_{li}^{(n)}} \tag{2-15}$$

其中：

$$\frac{\partial E}{\partial Y_l^{(n)}} = (Y_l - T_l) = -\delta_l \tag{2-16}$$

$$\frac{\partial Y_l^{(n)}}{\partial V_l^{(n)}} = f'^{(n)}(V_l^{(n)}) \tag{2-17}$$

$$\frac{\partial V_l^{(n)}}{\partial W_{li}^{(n)}} = Y_i^{(n-1)} \tag{2-18}$$

根据式 (2-16)、式 (2-17)、式 (2-18) 可得：

$$\frac{\partial E}{\partial W_{li}^{(n)}} = -\delta_l f'^{(n)}(V_l^{(n)})Y_i^{(n-1)} \tag{2-19}$$

令

$$S^{(n)} = \delta_l f'^{(n)}(V_l^{(n)}) \tag{2-20}$$

故可得：

$$\frac{\partial E}{\partial \theta^{(n)}} = \frac{\partial E}{\partial V^{(n)}} = -S^{(n)} \tag{2-21}$$

2.3　算法流程

接下来阐述误差反向传播神经网络的算法流程。第一步随机初始化各层神经元之间的连接权值矩阵；第二步通过式 (2-9) 计算每层神经元的输出矩阵；第三步求各层神经元的误差：

$$\delta = Y^{(n)} - T \tag{2-22}$$

第四步对于输出层求得反向传播误差因子：

$$S^{(n)} = f(V^{(n)})\delta \tag{2-23}$$

对于输出层以前的各层求得反向传播误差因子：

$$S^{(n)} = f'^{(n)}(V^{(n)})W^{(n+1)^T} S^{(n+1)} \tag{2-24}$$

第五步更新各神经元层与上一层的连接权值矩阵：

$$W_{lk}^{(n)} = W_{lk}^{(n-1)} + \eta S_k^{(n)} Y_l^{(n-1)} \tag{2-25}$$

$$\theta_k^{(n)} = \theta_k^{(n-1)} + \eta S_k^{(n)} \tag{2-26}$$

判断是否达到条件，如是否达到期望正确率，如未达到条件则返回第二步。

第 3 章　网络的 Matlab 实现

依据上一章节推导得出的算法原理和流程，我们利用 Matlab 来实现误差方向传播神经网络。[①]

首先，我们需要创建初始化神经元层间的连接权值矩阵的函数：

```
function [ W ] = init_w( levels )
% 初始化连接权值矩阵，返回每一层之间的连接权值矩阵
    n_level = numel(levels) - 1;
    W = cell(n_level,1);
    for i=1:n_level
        W{i} = rand(levels(i+1),levels(i));
    end
end
```

接下来创建初始化每层神经元的阈值矩阵的函数：

```
function [ theta ] = init_theta( levels )
% 初始化阈值矩阵
    n_level = numel(levels) - 1;
    theta = cell(n_level, 1);
    for i=1:n_level
        theta{i} = rands(levels(i+1),1);
    end
end
```

接下来创建激活函数，这里我们使用 *sigmoid* 作为神经网络的激活函数：

```
function [ y ] = sigmoid( x )
% sigmoid 函数
    y = 1./(1+exp(-x));
end
```

接下来创建神经网络前向传播运算函数：

① 本文所涉及的所有源码均已放在 Github 上，地址为：https://github.com/RobertIndie/BPNN

```
function [ output,Y] = predict( input, W, theta)
% 前向传播
% input: 输入向量
% W: 连接权值矩阵元胞数组
% theta: 阈值矩阵
% output: 输出向量
% Y: 各层的输出
    f = @sigmoid;
    n_w = numel(W);
    Y = cell(n_w+1, 1);
    y = input';
    Y{1}=input';
    for i=1:n_w
        V = W{i}*y+theta{i};
        y = f(V);
        Y{i+1} = y;
    end
    output = y;
end
```

下面是训练神经网络的函数：

```
function [ W,theta,record ] = train( X, levels, step, min_err, computeE
% 训练
% X: 训练集
% levels: 神经网络结构
% step: 趋近学习步长
% min_err: 期望错误率
% computeErrorFunc: 错误率计算函数
    r_i = 0;record = zeros(1,10000);% 历史错误率
    n_levels = size(levels, 2) - 1;
    n_input = levels(1);
    W = init_w(levels);
    theta = init_theta(levels);
    [n_data,col] = size(X);
```

```
14      train_data = X(:,1:n_input);
15      label = X(:,n_input+1:end)';
16      f = @sigmoid;
17      while true
18          for k=1:n_data
19              [output, Y] =  predict(train_data(k,:), W, theta);
20              delta = label(:,k) - output;
21              % 从最后一层往前更新W
22              for l=n_levels:-1:1
23                  net = W{l}*Y{l} + theta{l};
24                  if l == n_levels
25                      S = diag(f(net).*(1-f(net)))*delta;
26                  else
27                      S = diag((f(net).*(1-f(net))))*W{l+1}'*S;
28                  end
29                  new_W{l} = W{l} + step*S*Y{l}';
30                  new_theta{l} = theta{l} + step*S;
31              end
32              W = new_W;
33              theta = new_theta;
34          end
35
36          y = predict(train_data, W, theta);
37          error = computeErrorFunc(y, label)
38          r_i = r_i + 1;
39          record(r_i) = error; % 记录历史错误率
40          if error < min_err
41              break;
42          end
43      end
44      record=record(1,1:r_i);
45  end
```

训练的中止条件为达到某一期望的最大错误率，错误率计算函数由函数参数

给出。

第 4 章　神经网络的应用实例

4.1　二维分类问题

首先我们将刚构建的神经网络应用到二维分类问题上，第一个解决的问题是圆形分类问题，即划定一个圆，检测某个点是否在圆内。所以构建的神经网络需要接收两个输入，一个是该点的 X 位置，一个是该点的 Y 位置。为了使得位置数值归一化，点的位置限定在了 0 到 1 之间。如图 4.1，为随机生成的测试数据集，红色实心点代表在半径为 0.5，圆心为 $(0.5, 0.5)$ 的圆内的点，绿色空心点为在该圆外的点。

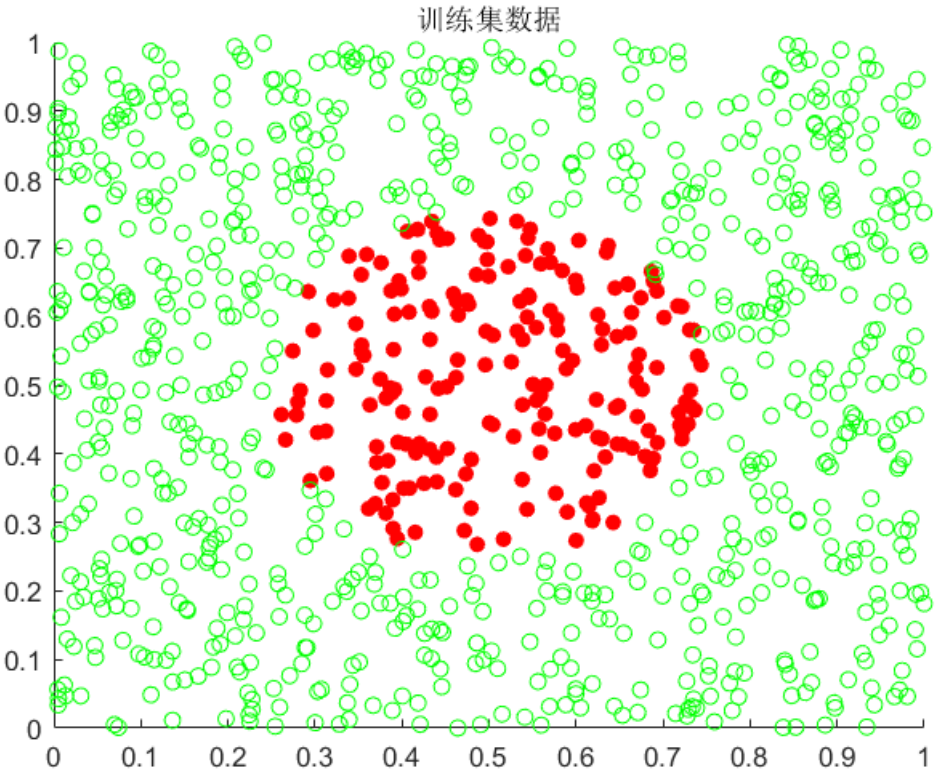

图 4.1　测试集数据

构造数据集的代码如下：

```
%% 构造数据集
data_len = 1000;
data = zeros(data_len,3);
```

```
data(:,1:2) = rand(data_len,2);
for i=1:data_len
    data(i,3) = ((data(i,1)-0.5)^2+(data(i,2)-0.5)^2) <= 0.25^2;
end

gindex = find(data(:,3)==0);
rindex = find(data(:,3)==1);

figure;
scatter(data(rindex,1),data(rindex,2),'filled','r');
hold on;
scatter(data(gindex,1),data(gindex,2),'g');
title('训练集数据');
```

训练的网络结构使用三层神经元结构，输入层有两个神经元，隐含层有七个神经元，输出层有一个神经元。设置学习步长 *step* 为 0.5，期望最高错误率为 1%。训练过程的代码如下：

```
%% 训练
levels = [2,7,1];

[W,theta,record] = BP_tranning(data,levels,0.5,3,@compute_error);
```

其中 *compute_error* 函数如下：

```
function [ error ] = compute_error(output,target)
%COMPUTE_ERROR 此处显示有关此函数的摘要
%    此处显示详细说明
    output(output>0.5)=1;
    output(output<=0.5)=0;
    delta = abs(output - target);
    error = sum(sum(delta))/size(delta,2)*100;
end
```

测试神经网络准确性的代码如下：

```
%% 测试
data_len = 1000;
```

```
data = zeros(data_len,3);
data(:,1:2) = rand(data_len,2);
for i=1:data_len
    data(i,3) = ((data(i,1)-0.5)^2+(data(i,2)-0.5)^2) <= 0.25^2;
end

Y = predict(data(:,1:2),W,theta);

Y(Y>0.5)=1;
Y(Y<=0.5)=0;
T = data(:,3) - Y';
correct_index = find(T == 0);
green_index = find(data(:,3)==0);
error_index = find(T ~= 0);
red_index = find(data(:,3)==1);

figure;
scatter(data(green_index,1),data(green_index,2),'g');
hold on
scatter(data(red_index,1),data(red_index,2),'filled','r');
title('分类测试结果')
figure;
scatter(data(correct_index,1),data(correct_index,2),'g');
hold on
scatter(data(error_index,1),data(error_index,2),'filled','r');
title('正确点和错误点')
figure;
plot(record);
title('历史错误率');
xlabel('次数');
ylabel('错误率');
```

经过分类测试的结果如图 4.2。正确点和错误点图为图 4.3，其中，红色实心点代表错误点，绿色空心点代表正确点。历史错误率如图 4.4。

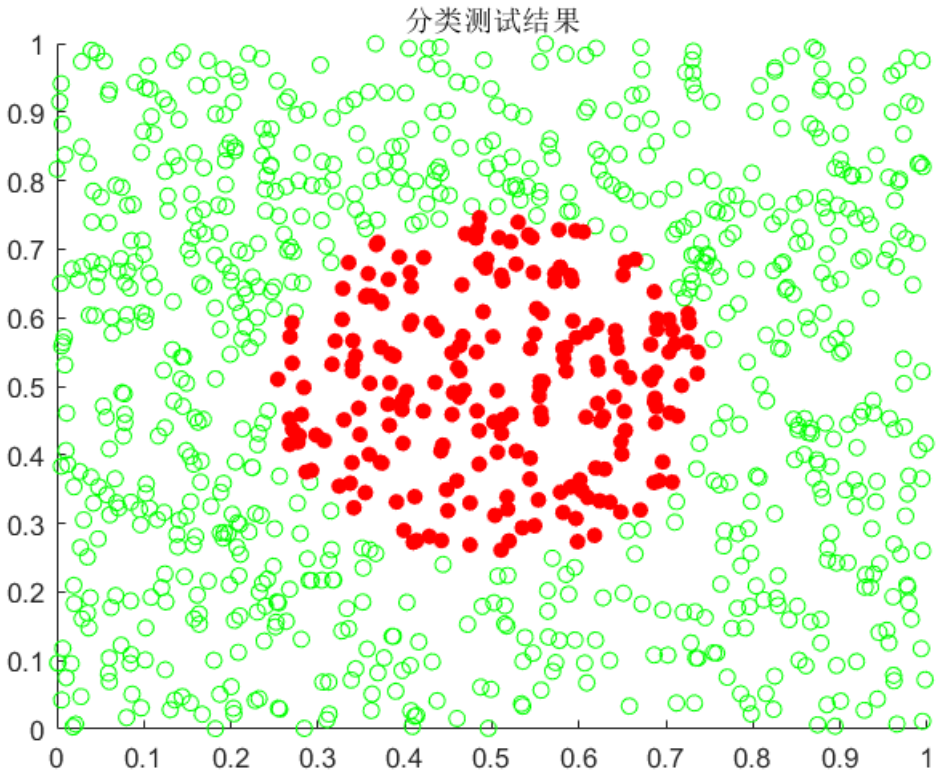

图 4.2　分类测试结果

现将这个神经网络模型运用到双曲线分类问题中，随机生成的数据集如图 4.5 构造数据集的代码如下：

```
%% 构造数据集
data_len = 1000;
data = zeros(data_len,3);
data(:,1:2) = rand(data_len,2);
for i=1:data_len
    data(i,3) = ((data(i,1)-0.5)^2-(data(i,2)-0.5)^2) <= 0.25^2;
end

gindex = find(data(:,3)==0);
rindex = find(data(:,3)==1);

figure;
scatter(data(rindex,1),data(rindex,2),'filled','r');
```

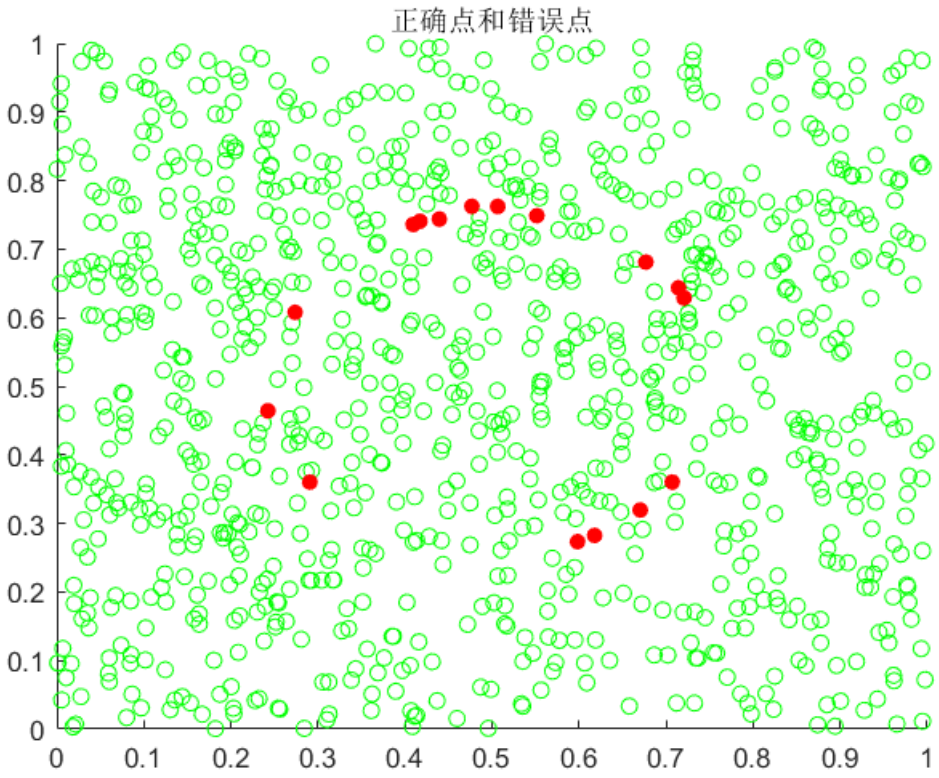

图 4.3　正确点和错误点

```
hold on;
scatter(data(gindex,1),data(gindex,2),'g');
title('训练集数据');
```

训练的网络结构使用三层神经元结构，输入层有两个神经元，隐含层有七个神经元，输出层有一个神经元。设置学习步长 *step* 为 0.5，期望最高错误率为 1%。训练过程的代码如下：

```
%% 训练
levels = [2,7,1];

[W,theta,record] = BP_tranning(data,levels,0.5,3,@compute_error);
```

其中 *compute_error* 函数如下：

```
function [ error ] = compute_error(output,target)
%COMPUTE_ERROR 此处显示有关此函数的摘要
%    此处显示详细说明
    output(output>0.5)=1;
```

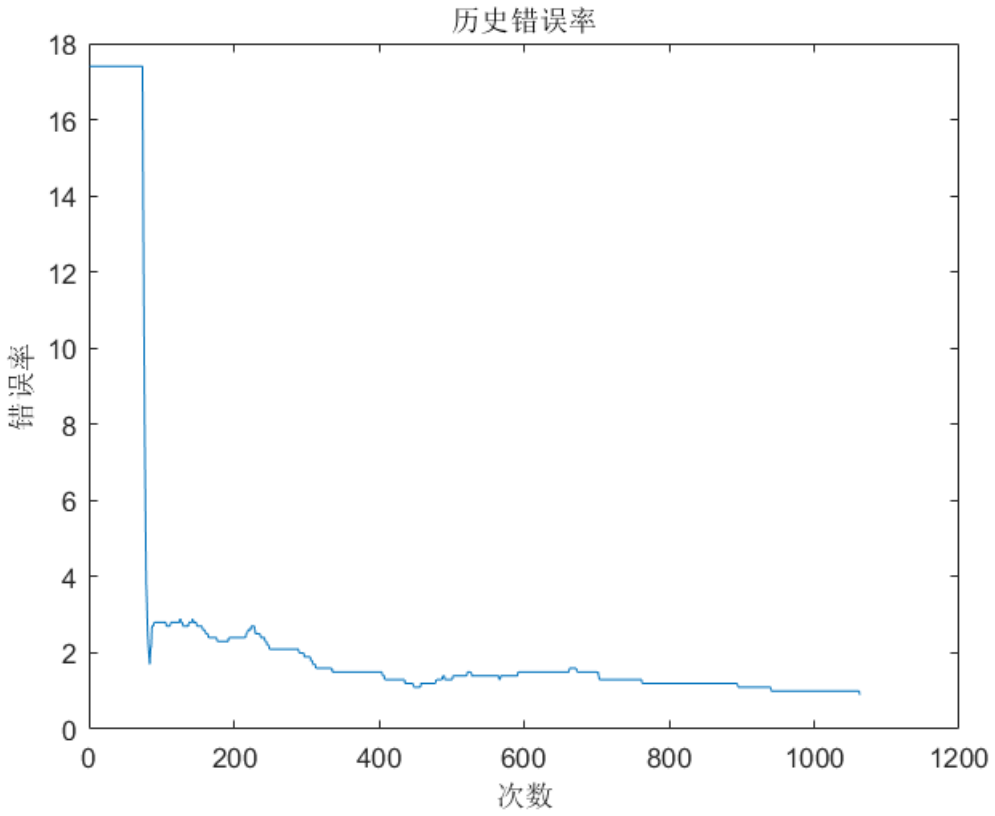

图 4.4　历史错误率

```
    output(output<=0.5)=0;
    delta = abs(output - target);
    error = sum(sum(delta))/size(delta,2)*100;
end
```

测试神经网络准确性的代码如下：

```
%% 测试
data_len = 1000;
data = zeros(data_len,3);
data(:,1:2) = rand(data_len,2);
for i=1:data_len
    data(i,3) = ((data(i,1)-0.5)^2-(data(i,2)-0.5)^2) <= 0.25^2;
end

Y = predict(data(:,1:2),W,theta);

```

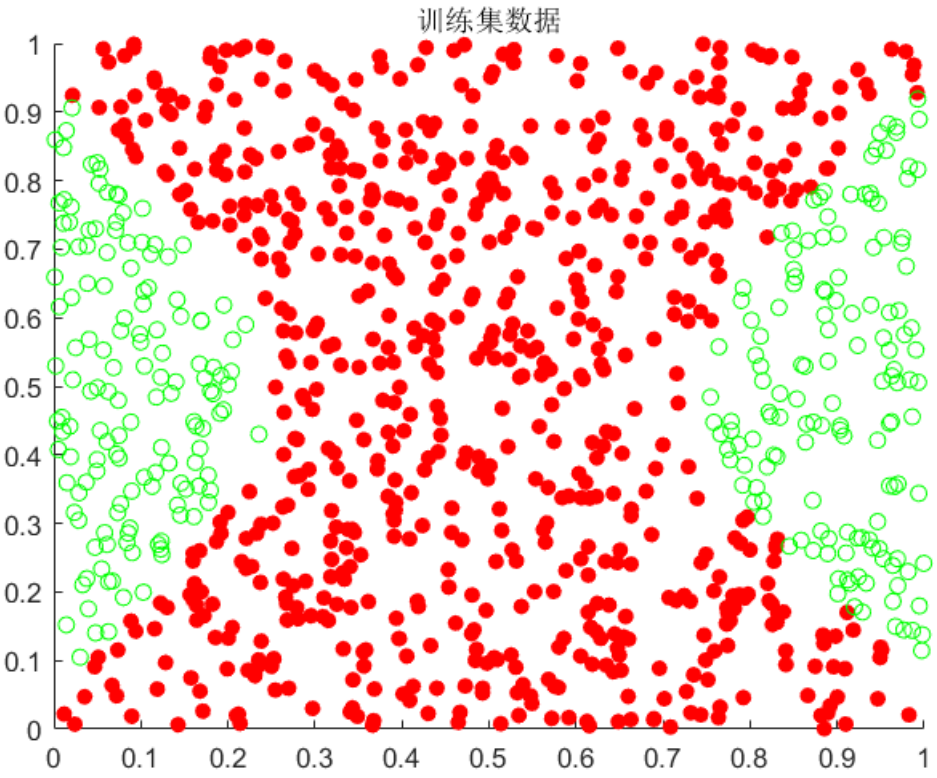

图 4.5　测试集数据

```
Y(Y>0.5)=1;
Y(Y<=0.5)=0;
T = data(:,3) - Y';
correct_index = find(T == 0);
green_index = find(data(:,3)==0);
error_index = find(T ~= 0);
red_index = find(data(:,3)==1);

figure;
scatter(data(green_index,1),data(green_index,2),'g');
hold on
scatter(data(red_index,1),data(red_index,2),'filled','r');
title('分类测试结果')
figure;
scatter(data(correct_index,1),data(correct_index,2),'g');
hold on
```

```
scatter(data(error_index,1),data(error_index,2),'filled','r');
title('正确点和错误点')
figure;
plot(record);
title('历史错误率');
xlabel('次数');
ylabel('错误率');
```

经过分类测试的结果如图 4.6。正确点和错误点图为图 4.7，其中，红色实心点代表错误点，绿色空心点代表正确点。历史错误率如图 4.8。

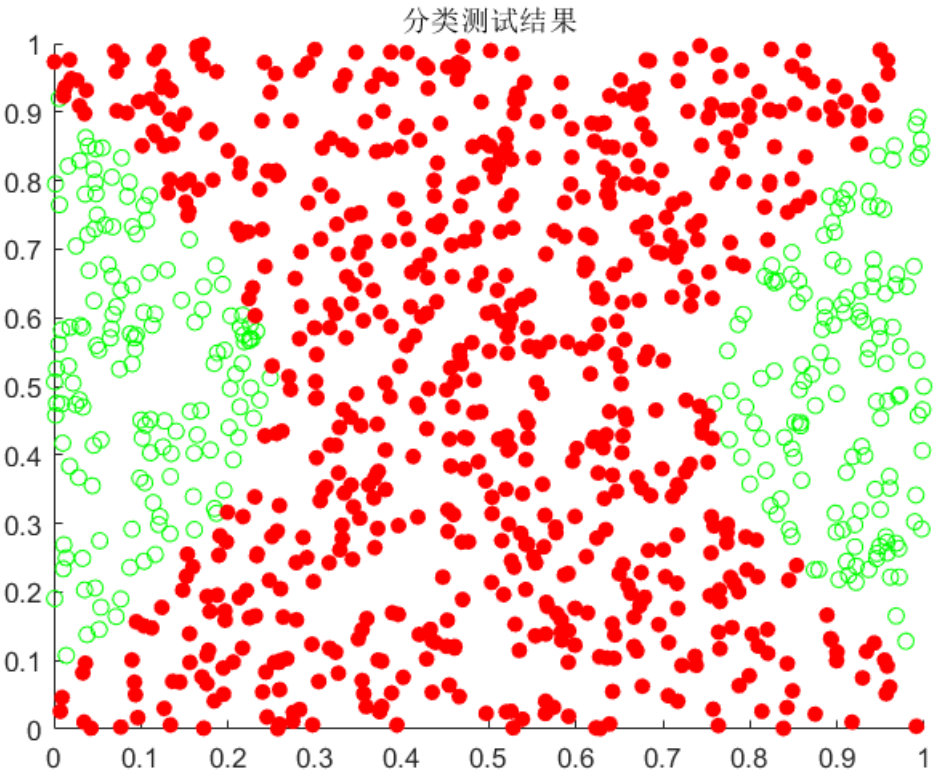

图 4.6　分类测试结果

4.2　三维分类问题

现将这个神经网络模型运用到三维球形分类问题中，随机生成的数据集如图 4.9

构造数据集的代码如下：

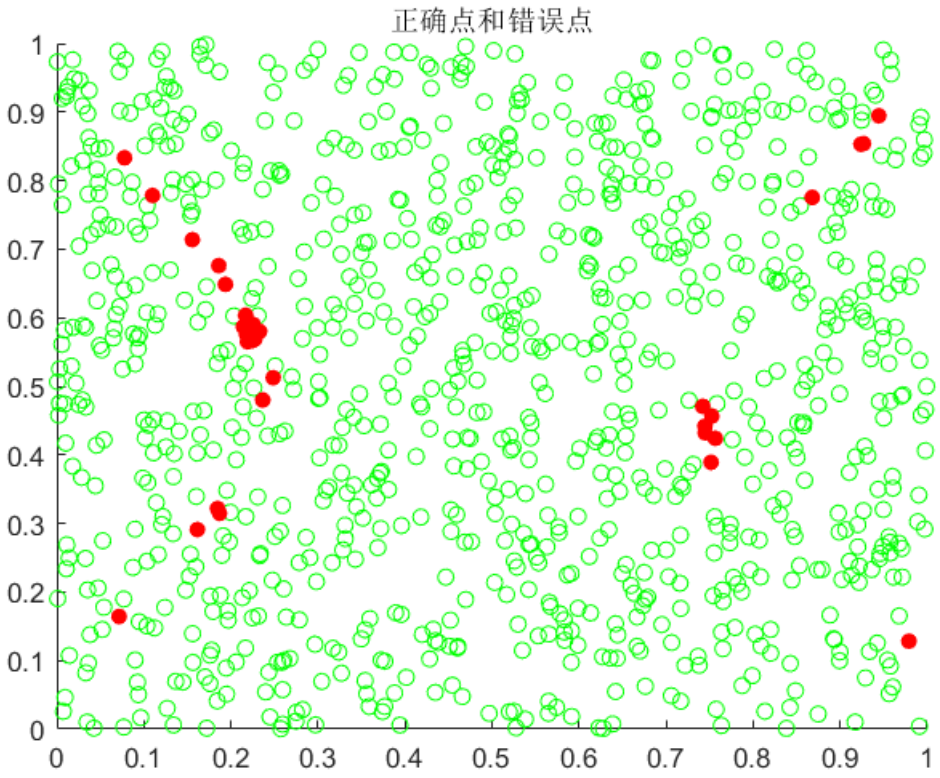

图 4.7　正确点和错误点

```
%% 构造数据集
data_len = 1000;
data = zeros(data_len,4);
data(:,1:3) = rand(data_len,3);
for i=1:data_len
    data(i,4) = ((data(i,1)-0.5)^2+(data(i,2)-0.5)^2+(data(i,3)-0.5)^2)
end

gindex = find(data(:,4)==0);
rindex = find(data(:,4)==1);

figure;
scatter3(data(rindex,1),data(rindex,2),data(rindex,3),'filled','r');
hold on;
scatter3(data(gindex,1),data(gindex,2),data(gindex,3),'g');
title('训练集数据');
```

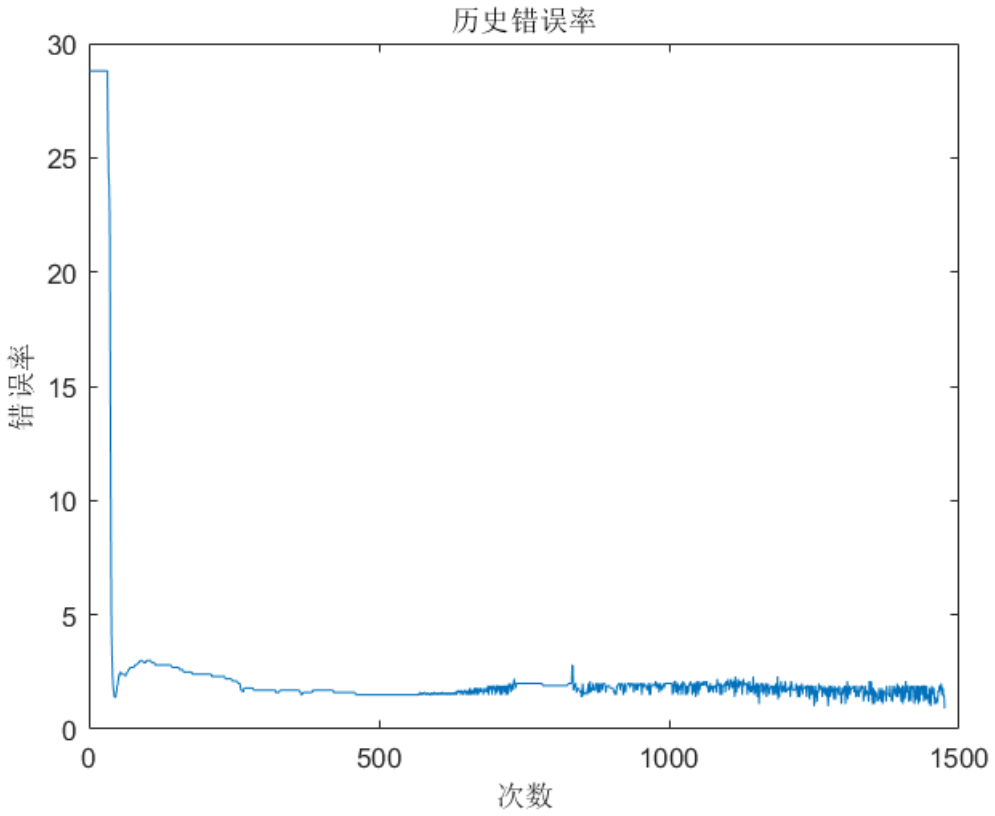

图 4.8　历史错误率

训练的网络结构使用三层神经元结构，输入层有三个神经元，隐含层有七个神经元，输出层有一个神经元。设置学习步长 *step* 为 0.5，期望最高错误率为 1%。训练过程的代码如下：

```
%% 训练
levels = [3,7,1];

[W,theta,record] = BP_tranning(data,levels,0.5,1,@compute_error);
```

其中 *compute_error* 函数如下：

```
function [ error ] = compute_error(output,target)
%COMPUTE_ERROR 此处显示有关此函数的摘要
%    此处显示详细说明
    output(output>0.5)=1;
    output(output<=0.5)=0;
    delta = abs(output - target);
    error = sum(sum(delta))/size(delta,2)*100;
```

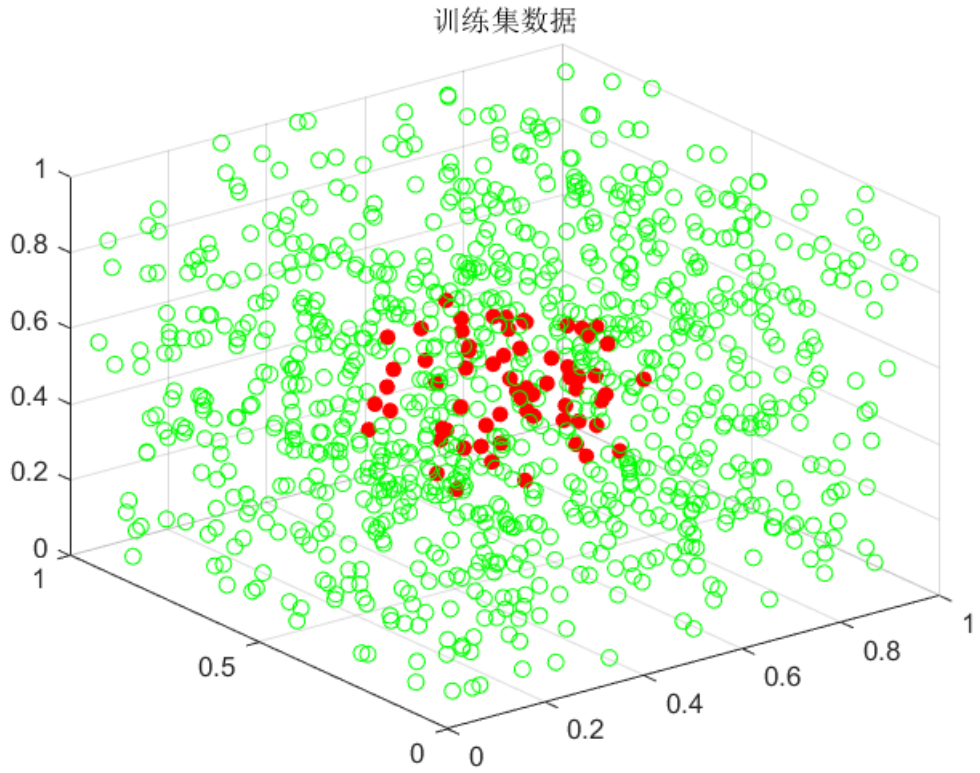

图 4.9　测试集数据

```
end
```

测试神经网络准确性的代码如下：

```
%% 测试
data_len = 1000;
data = zeros(data_len,4);
data(:,1:3) = rand(data_len,3);
for i=1:data_len
    data(i,4) = ((data(i,1)-0.5)^2+(data(i,2)-0.5)^2+(data(i,3)-0.5)^2)
end

Y = predict(data(:,1:3),W,theta);

Y(Y>0.5)=1;
Y(Y<=0.5)=0;
T = data(:,4) - Y';
```

```
correct_index = find(T == 0);
green_index = find(data(:,4)==0);
error_index = find(T ~= 0);
red_index = find(data(:,4)==1);

figure;
scatter3(data(green_index,1),data(green_index,2),data(green_index,3),'g
hold on
scatter3(data(red_index,1),data(red_index,2),data(red_index,3),'filled'
title('分类测试结果')
figure;
scatter3(data(correct_index,1),data(correct_index,2),data(correct_index
hold on
scatter3(data(error_index,1),data(error_index,2),data(error_index,3),'f
title('正确点和错误点')
figure;
plot(record);
title('历史错误率');
xlabel('次数');
ylabel('错误率');
```

经过分类测试的结果如图 4.10。正确点和错误点图为图 4.11，其中，红色实心点代表错误点，绿色空心点代表正确点。历史错误率如图 4.12。

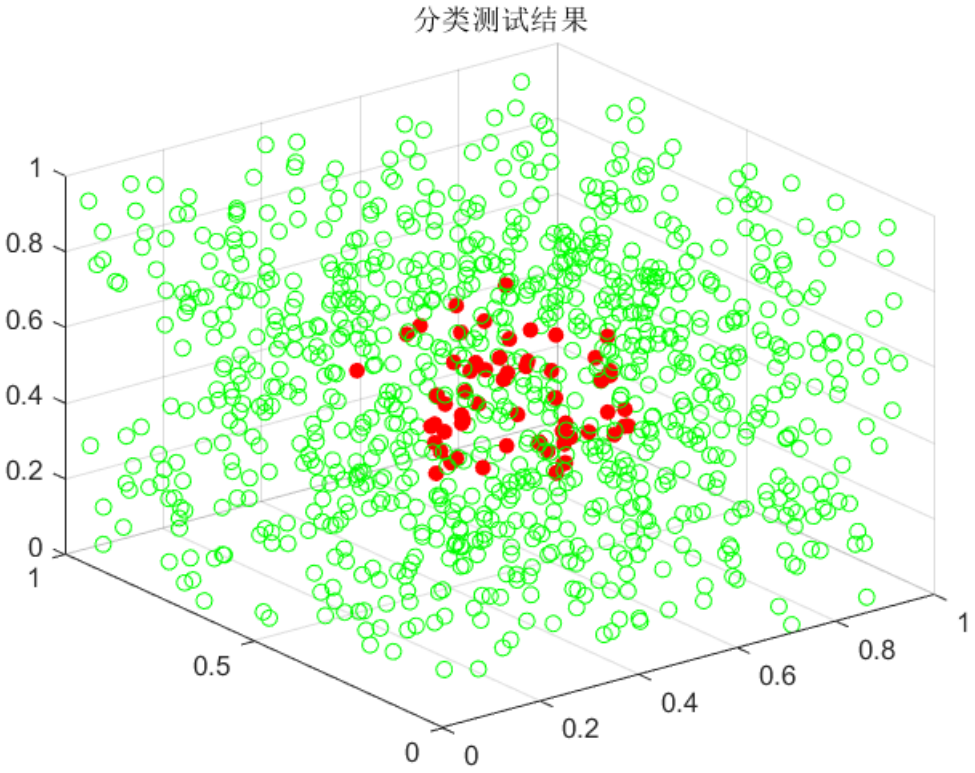

图 4.10　分类测试结果

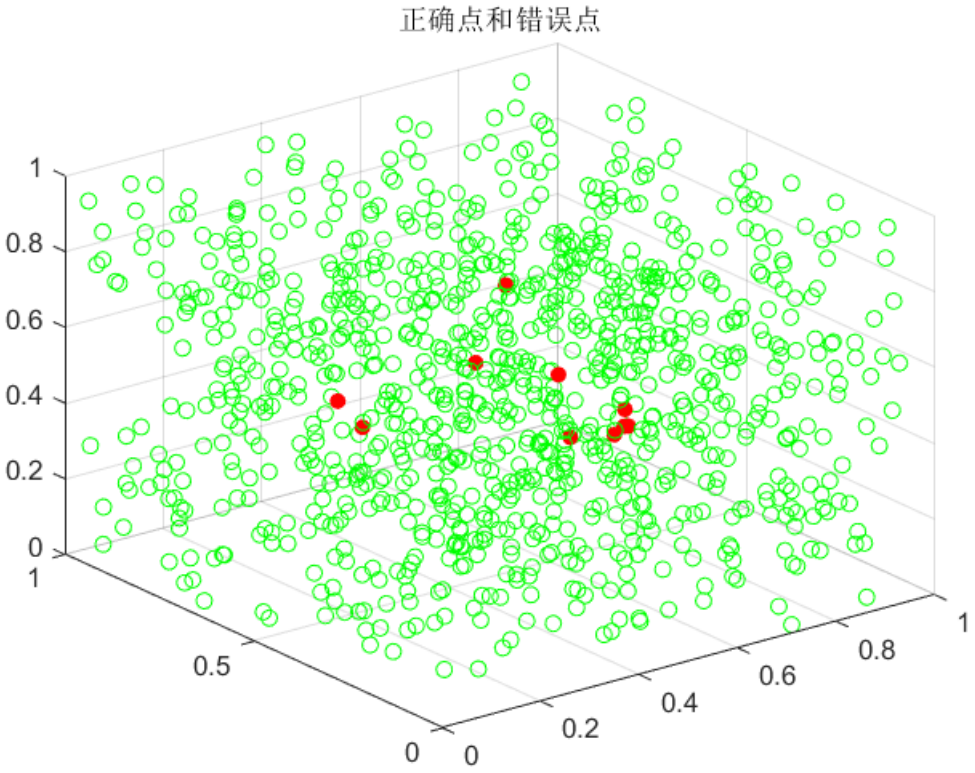

图 4.11　正确点和错误点

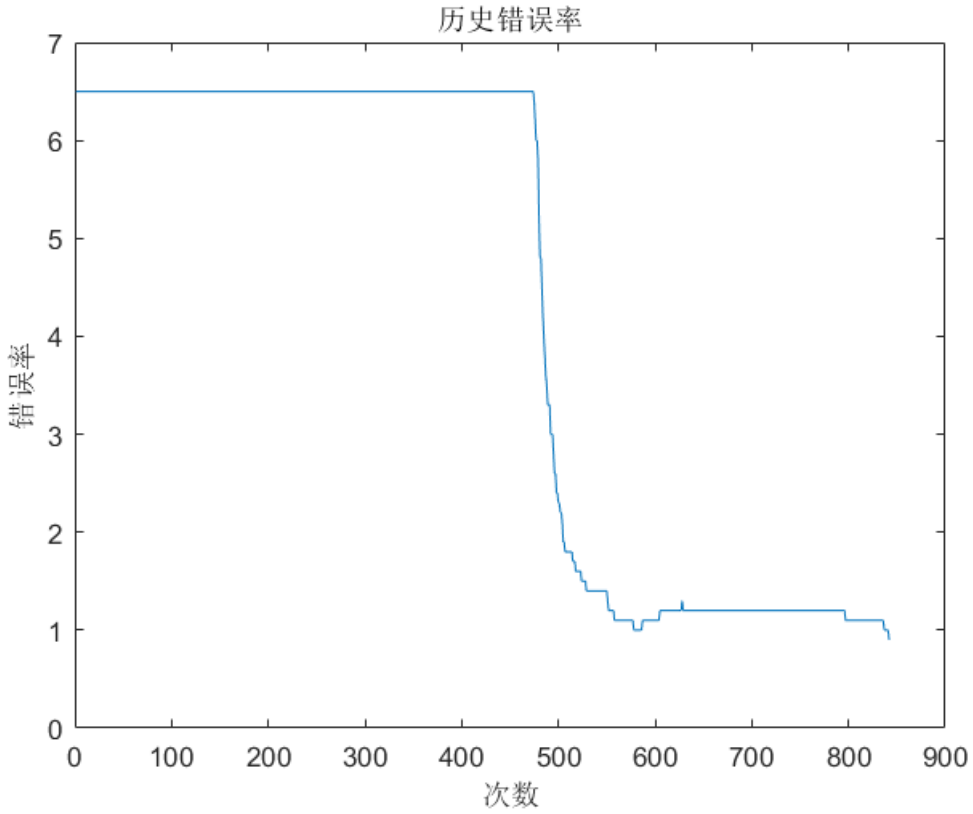

图 4.12　历史错误率

第 5 章　结论

本文构建了误差反向传播神经网络的数学模型，并利用 Matlab 实现了该数学模型，还将此模型运用到二维分类问题和三维分类问题中。采用三层的神经网络模型进行研究，分别为输入层、隐含层和输出层。其中，将点的位置分量作为模型输入层的输入向量，采用 *sigmoid* 为激活函数。

通过不断地训练，两个问题的正确率均能达到 99% 以上，且具有良好的性能，但在分类图形的边界上错误率较高。能够较好地解决这类分类问题。在训练过程中，η 的选值比较关键，在训练后期，错误率容易上下波动，难以收敛。

误差反向传播神经网络摆脱了传统方式的局限，突破了依赖线性模型的限制，但在训练后期，常出现错误率不稳定等问题，本文所构建的模型未能在函数逼近拟合等问题上得到很好的应用，因此，在后续研究中模型仍需改进。

插图索引

公式索引

参考文献

[1] Rumelhart G E W R J, David E.; Hinton. Learning representations by back-propagating errors: number 323[Z]. [S.l.: s.n.], 1986.

[2] 韩普周北望. Bp 神经网络原理研究与实现: 第 10 册[Z]. [出版地不详: 出版者不详], 2018.

[3] 袁冰清郑柳刚. Bp 神经网络基本原理: 第 08 册[Z]. [出版地不详: 出版者不详], 2018.