中 国 矿 业 大 学

Matlab 结课　论文

姓　　名：丁旭行　　学　　号：14184501

学　　院：计算机科学与技术学院

专　　业：计算机科学与技术专业

论文题目：Matlab 构建 BP 神经网络实现二分类

指导老师：王瑞瑞

2020 年 4 月 30 日　　徐州

摘　要

随着信息技术的发展,人们生产数据和采集数据的能力愈来愈高,但是,我们在数据分析和知识获取方面的能力还相对滞后。因此,从收集数据、创建数据库、管理数据到数据分析、数据挖掘等技术逐渐产生和发展。数据挖掘是一门跨学科的课题,涉及许多领域,包括统计学、数据库、机器学习和人工智能等，其本质就是从“海洋般”的大量数据中获取新颖的、有用的、有效的、可理解的模式的非平凡过程,也就是从大量数据里提取知识。分类问题是数据挖掘技术中非常重要的研究课题,利用分类技术,可以从数据集中提取出描述相同数据类型的模型或函数,并且能够顺利把数据集中每一个未知类别的数据划归到某个已知的类别中去。目前,常用的数据挖掘分类算法主要有：统计分类法、决策树、人工神经网络方法等。不同的算法会产生不同的分类器,而不同的分类器又会影响数据挖掘的准确率和数据挖掘的效率。人工神经网络是数据挖掘常用的方法之一,该方法通过模拟人脑生物神经网络,将若干个具有处理功能的神经元节点,按照一定的网络结构连接起来,使它能够处理不精确数据、模糊数据或者复杂的非线性映射问题。本文利用 MATLAB 语言构建 BP 神经网络，对模拟的二维训练数据分别按照逐个样本修正的 BP 算法和成批样本修正的 BP 算法进行二分类训练，得到二分类器并根据收敛曲线比较其算法性能，最后使用训练后的神经网络来测试模拟的二维测试数据检测二分类器性能。

关键词：数据挖掘；分类算法；分类器；BP 神经网络；MATLAB；二分类

ABSTRACT

With the development of information technology, people's ability to produce and collect data is getting higher and higher. However, our ability in data analysis and knowledge acquisition is still relatively backward. Therefore, from data collection, database creation, data management to data analysis, data mining and other technologies are gradually produced and developing. Data mining is an interdisciplinary subject, involving many fields, including statistics, database, machine learning and artificial intelligence. Its essence is a non trivial process of obtaining novel, useful, effective and understandable patterns from a large number of "ocean like" data, that is, extracting knowledge from a large number of data. Classification is a very important research topic in data mining technology. Using classification technology, models or functions describing the same data types can be extracted from the data set, and each unknown class of data in the data set can be smoothly classified into a known class. At present, the commonly used data mining classification algorithms are: statistical classification, decision tree, artificial neural network methods. Different algorithms will produce different classifiers, and different classifiers will affect the accuracy and efficiency of data mining. Artificial neural network is one of the commonly used methods of data mining. By simulating the biological neural network of human brain, several neural nodes with processing function are connected according to a certain network structure, so that it can deal with imprecise data, fuzzy data or complex nonlinear mapping problems. In this paper, I use MATLAB language to build BP Neural Network to train the simulated two-dimensional training data according to the BP algorithm modified by one sample and the BP algorithm modified by a batch of samples to get binary classifiers, and compare the algorithm performance according to the convergence curve. Finally, we use the trained neural network to test the simulated two-dimensional test data to detect the performance of binary classifiers.

Keyword： Data Mining; Classification; Classifier; BP Neural Network; MATLAB; Binary Classification

目　　录

1 数据分类问题描述

数据分类就是把具有某种共同属性或特征的数据归并在一起，通过其类别的属性或特征来对数据进行区别。为了实现数据共享和提高处理效率，必须遵循约定的分类原则和方法，按照信息的内涵、性质及管理的要求，将系统内所有信息按一定的结构体系分为不同的集合，从而使得每个信息在相应的分类体系中都有一个对应位置。换句话说，就是相同内容、相同性质的信息以及要求统一管理的信息集合在一起，而把相异的和需要分别管理的信息区分开来，然后确定各个集合之间的关系，形成一个有条理的分类系统。本文主要讨论的是分类问题中较为基础和常见的二分类问题，即将数据分为正反两例。使用自己模拟生成的两类二维非线性样本数据进行网络训练和测试，根据测试分类结果，检验算法的正确性和有效性。

2 BP 神经网络算法介绍

BP(Back Propagation)神经网络是 1986 年由 Rumelhart 和 McCelland 为首的科学家小组提出，是一种按误差逆传播算法训练的多层前馈网络，是目前应用最广泛的神经网络模型之一。BP 网络能学习和存贮大量的输入-输出模式映射关系，而无需事前揭示描述这种映射关系的数学方程。

2.1 人工神经元

从人类生物神经元的基本功能出发，科学家们提出了人工神经元的模型用于模拟生物神经元的功能。而若干人工神经元组成人工神经网络，是一种模仿动物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。人工神经网络具有自学习和自适应的能力，可以通过预先提供的一批相互对应的输入-输出数据，分析掌握两者之间潜在的规律，最终根据这些规律，用新的输入数据来推算输出结果，这种学习分析的过程被称为“训练”。人工神经元是人工神经网络的基本单元，其数学模型如图 2-1 所示。

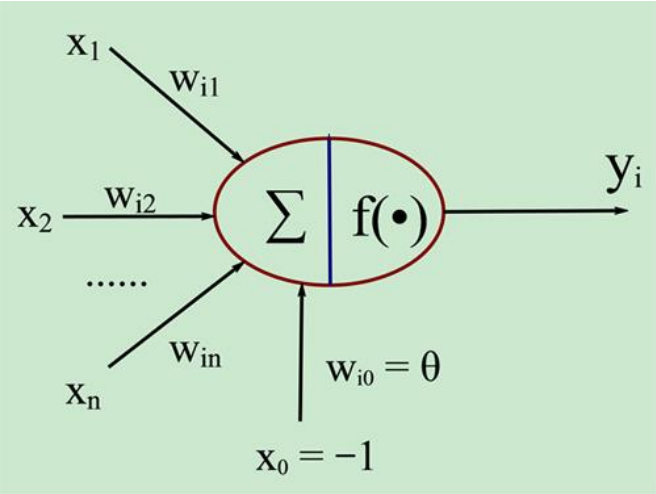

图 2-1 人工神经元模型

图中x_i~x_n是从其他神经元传来的输入信号，w_{ij}表示从神经元j到神经元i的连接权值，θ表示一个阈值或称为偏置。则神经元i的输出与输入的关系表示为：

$$net_i = \sum_{j=1}^{n} w_{ij}x_j - \theta, \tag{2-1}$$

$$y_i = f(net_i), \tag{2-2}$$

公式(2-2)中y_i表示神经元i的输出，函数f称为激活函数(Activation Function)或转移函数(Transfer Function)，net称为净激活(net activation)。若将阈值看成是神经元i的一个输入x_0的权重w_{i0}，则上面的公式(2-1)可以简化为：

$$net_i = \sum_{j=0}^{n} w_{ij}x_j, \tag{2-3}$$

则将其表述为矩阵形式如下：

$$net_i = XW, \tag{2-4}$$

$$y_i = f(net_i) = f(XW), \tag{2-5}$$

其中$X = (x_0, x_1, \ldots\ldots, x_n)$，$W = (w_{i0}, w_{i1}, \ldots\ldots, w_{in})^T$，$x_0 = -1$，$w_{i0} = \theta$。

选择不同的激活函数f，y_i的取值范围也不同，而函数f通常要求选择可微的函数，此时通常选择 Sigmoid 函数，如式(2-6)所示。该函数具有如下特性：非线性、单调性、无限次可微，并且当权值很大时可近似阈值函数。当权值很小时可近似线性函数。

$$f(x) = \frac{1}{1+e^{-x}} \tag{2-6}$$

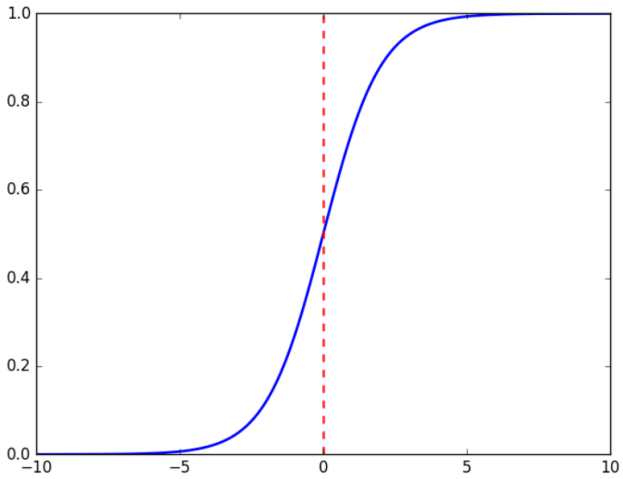

图 2-2 Sigmoid 函数

2.2 前馈神经网络

构成前馈神经网络的神经元接受前一级输入，并输出到下一级，无反馈，如图 2-2。其节点分为两类：输入节点与计算单元。每个计算单元可有任意个输入，但只有一个输出，而输出可藕合到任意多个其他节点的输入。前馈网络通常分为不同的层，第i层的输入只与第$i-1$层的输出相联。输入节点为第一层。输入和输出节点由于可与外界相连，称为可见层，而其他的中间层则称为隐藏层。而其中三层前馈网络又是其特例，三层前馈网络由输入层、中间层和输出层构成。有两个计算层，也称三层感知器，能够求解非线性问题。三层或三层以上的前馈网络通常又被叫做多层感知器(Multi-Layer Perceptron.简称 MLP)。由三部分组成：一组感知单元组成输入层；一层或多层计算节点的隐藏层；一层计算节点的输出层。多层感知器的表示：输入节点数-第 1 隐层节点数-第 2 隐层节点数-…-输出节点数。如图 2-2 可表示为：2-3-3 网络。

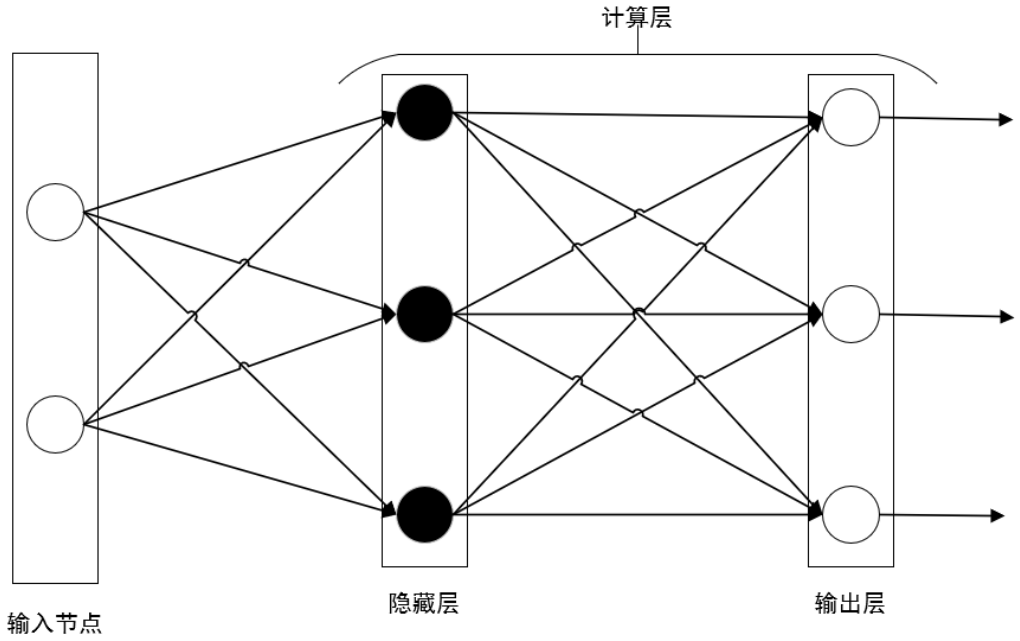

图 2-3 前馈神经网络模型

当使用阈值函数作为神经元输出函数时，任何逻辑函数都可以使用一个三层的前馈网络实现。当神经元的输出函数为 Sigmoid 函数时，上述结论可以推广到连续的非线性函数。在很宽松的条件下，三层前馈网络可以逼近任意的多元非线性函数，突破了二层前馈网络线性可分的限制。

要使神经元合理的组合输出，需要先让其学习以模拟生物的学习过程。神经元的学习算法通常采用 Hebb 学习规则，如果神经元u_i接收来自另一神经元u_j的输出，则当这两个神经元同时兴奋时，从u_j到u_i的权值w_{ij}就得到加强。具体到前述的神经元模型，可以将 Hebb 规则表现为如下的算法$\Delta w_i = \eta y x_i$。式中Δw_i是对第i个权值的修正量，η是控制学习速度的系数。太大会影响训练的稳定性，太小则使训练的收敛速度变慢，一般取$0 < \eta \leq 1$，人工神经网络首先要以一定的学习准则进行学习，然后才能工作。

2.3 BP 学习算法

三层前馈网络的适用范围大大超过二层前馈网络，但学习算法较为复杂，主要困难是中间的隐藏层不直接与外界连接，无法直接计算其误差。 而 BP 算法可解决这一问题。其主要思想是从后向前(反向)逐层传播输出层的误差，以间接算出隐藏层误差。因此算法分为两个阶段：

第一阶段(正向传导过程)：输入信息从输入层经隐藏层逐层计算各单元的输出值；

第二阶段(反向传播过程)：输出误差逐层向前算出隐藏层各单元的误差，并用此误差修正前层权值。

在反向传播算法中通常采用梯度法修正权值，为此要求输出函数可微，通常采用 Sigmoid 函数作为输出函数。下面考虑一个如图 2-3 所示的三层前馈网络，其中i表示前层第i个单元，k表示后层第k个单元，O_j表示本层的输出，w_{ij}表示前层到本层的权值，w_{jk}表示本层到后层的权值。其中的输出函数选择 Sigmoid 函数。

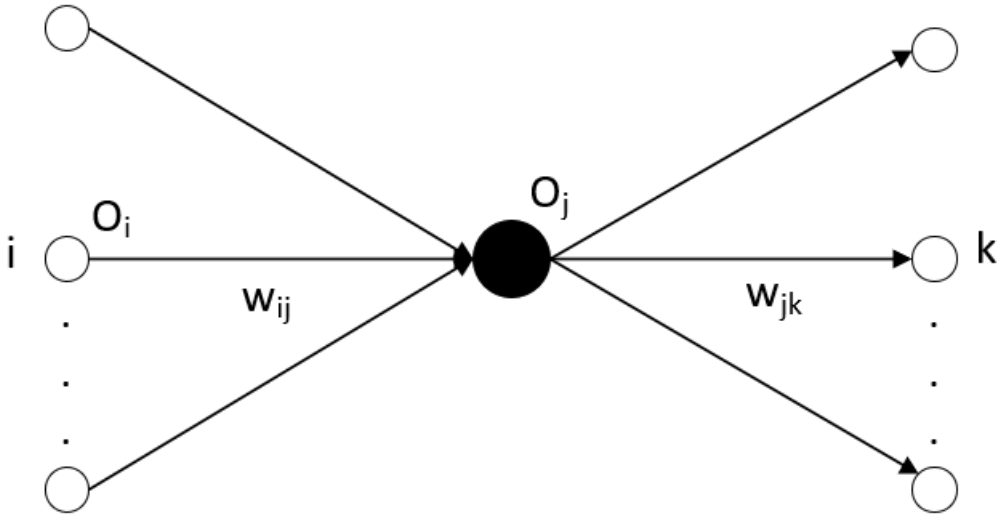

图 2-4 三层前馈网络简化模型

在第一阶段的正向传导过程中，某层的输出为公式(2-8)所示，第一阶段从输入层逐层往后计算输出，直到最后一层。

$$net_j = \sum_{i=1}^{n} w_{ij} O_i, \tag{2-7}$$

$$O_j = f(net_j) = \frac{1}{1+e^{-net_j}} \quad 。 \tag{2-8}$$

在第二阶段的反向传播过程中，对于输出层，O_k是实际输出，与理想输出y_k的总误差可以通过公式(2-9)计算，其中N表示训练样本个数。

$$E = \frac{1}{N}\sum_k (y_k - O_k)^2, \tag{2-9}$$

输出层的局部梯度为公式(2-10)：

$$\delta_j = \frac{\partial E}{\partial net_j} = -(y_j - O_j)O_j(1 - O_j), \tag{2-10}$$

因此可以计算权值对误差的影响如公式(2-11)：

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \delta_j O_j, \tag{2-11}$$

采用负梯度修正权值，但是实际计算时，为了加快收敛速度，一般要加上前次权值修正量α，称为惯性量，因此实际的权值修正量如公式(2-12)：

$$\Delta w_{ij} = -\eta\delta_j O_i + \alpha\Delta w_{ij}(k-1), \tag{2-12}$$

修正算法可以写为公式(2-13)：

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k), \tag{2-13}$$

若节点j不是输出单元，则其局部梯度计算采用公式(2-14)：

$$\delta_j = \frac{\partial E}{\partial net_j} = \sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial O_j}\frac{\partial O_j}{\partial net_j} = \sum_k \delta_k w_{jk} f'(net_j)$$

$$= O_j(1 - O_j)\sum_k \delta_k w_{jk} \quad 。 \tag{2-14}$$

当采用逐个样本修正时，每次输入一个样本就修正一次权值，程序流程参考算法 1。

算法 1 – 逐个样本修正的 BP 算法	
S1:	选定(0,1)之间均匀分布的权系数w_{ij}，步长η，惯性系数α； 准备训练样本 Xin 及其理想输出 Yd；
S2:	根据公式(2-7)和(2-8)从前向后计算各层各单元输出O_j；
S3:	根据公式(2-9)计算输出层误差E； 当E小于一个设定的常数时，跳转到S7，否则继续下一步；

S4: 根据公式(2-10)对输出层计算局部梯度δ_k；

S5: 根据式(2-14)逐层向前计算各层的局部梯度δ_j；

S6: 根据公式(2-12)和(2-13)修正权值；然后跳转到S2；

S7: 结束修正，返回权值w_{ij}。

当采用成批样本修正时，可以在全部训练样本输入完成后统一修正权值，程序流程参考算法 2。其中成批样本修正时对于N个样本，全部输入完毕后统一计算权值修正如公式(2-15)，式中p表示N个样本中的第p个样本。

$$\Delta w_{ij} = -\left(\sum_{p=1}^{N} \eta \delta_{jp} O_{ip}\right) + \alpha \Delta w_{ij}(k-1) \tag{2-15}$$

算法 2– 成批样本修正的 BP 算法

S1: 选定(0,1)之间均匀分布的权系数w_{ij}，步长η，惯性系数α；
准备训练样本Xin及其理想输出Yd；

S2: 对$p \in \{1，2，....，N\}$执行S3到S6步骤，完成后跳转到S7；

S3: 根据公式(2-7)和(2-8)从前向后计算各层各单元输出O_{jp}；

S4: 根据公式(2-9)计算输出层误差E；
当E小于一个设定的常数时，跳转到S8，否则继续下一步；

S5: 根据公式(2-10)对输出层计算局部梯度δ_{kp}；

S6: 根据式(2-14)逐层向前计算各层的局部梯度δ_{jp}；

S7: 根据公式(2-15)和(2-12)修正权值；然后跳转到S2。

S8: 结束修正，返回权值w_{ij}。

3 BP 神经网络实现二分类的步骤

1）生成样本数据

编写模拟二维样本生成函数 sample_create，根据输入的样本数量产生两个类别的样本 1 和样本 2，如图 3-1 所示：

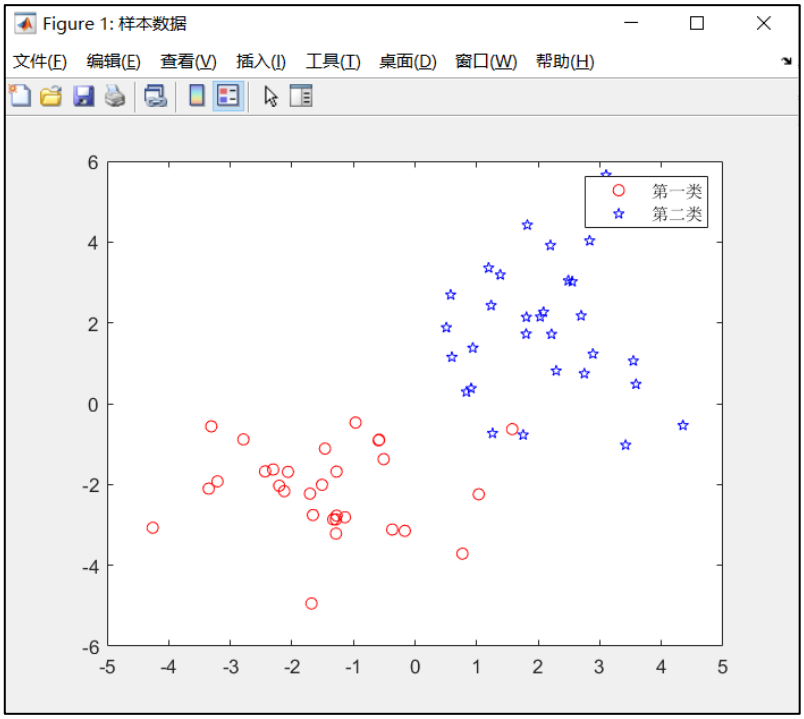

图 3-1 样本数据

2）生成神经网络

编写神经网络的生成函数 create_net，根据输入的层数、各层神经元节点数、误差界限、最大迭代次数、步长、惯性系数生成神经网络结构体，并完成权值w矩阵的初始化操作，将其初始化权值为 0 到 1 之间均匀分布的随机数。

3）训练神经网络

编写神经网络的训练函数 net_train_per 和 net_train_mass，分别采用逐个样本修正和成批样本修正实现神经网络的 BP 学习算法，根据输入的神经网络结构体和训练样本迭代按照算法 1 和算法 2 的步骤进行训练，最终输出训练结束后的结构体。

4）测试神经网络

编写网络分类函数 net_test 实现对输入样本的分类，输出参数就是分类结果。

为了便于使用和显示结果，我将上述过程分为两个阶段，即训练阶段和测试阶段，并且分别编写了相应的脚本文件 training 和 testing。详细见附录内容。

4 代码运行结果

执行脚本 training，生成模拟训练样本数据：

表格 4-1 训练样本部分数据

第一类		第二类	
0.908008030729362	-1.47993989854454	0.638305529129246	1.24762229558018
-1.17478110577151	-2.02002785164254	2.45502955644433	4.37885371688920
-0.621028022083386	-2.03477108602848	1.15129062006634	0.761531681692381
-3.05818025798736	-2.79816358456414	1.66511306103595	1.31578219445405
-2.46861558110062	-0.981314717871425	2.55278334594455	0.993073510294080
-2.27246940925019	-2.13321747950773	3.03909065350496	0.339412995748038
-0.901575382111377	-2.71453016378716	0.882361316734792	1.72813226707190
-2.27787193278764	-0.648614231573343	3.26065870912090	1.61240616378660
-1.29845854183672	-2.22477105605258	2.66014314104698	4.16384930131286
-4.05181629991115	-2.58902903072080	1.93213444645731	1.64782583177064
-2.35384999777443	-2.29375359773542	1.80477880210125	0.494974958232556
-2.82358652515685	-2.84792624363793	1.78239364985681	4.26763105768749
-3.57705702279920	-3.12012830124373	1.69689237864826	3.74609999471218
-1.49202534909405	0.525999692118309	2.02304562442511	1.67525915876827
-1.71801593632944	-0.344502407112654	2.05129035584877	-0.130031480467965
-1.96652011775555	-1.69246484076175	2.82606279021160	1.37120131177234
-3.33367794342811	-3.25711835935205	3.52697668673337	1.77946607234752
-0.872507721658410	-2.86546803055480	2.46691443568470	2.39041946885064
-1.64982058939669	-2.17653411423145	1.79028666161126	1.63065883014410
-2.29906603033298	-1.20858393837137	2.62519035708763	2.62709328308239
-1.97711020724837	-3.33200442131525	2.18322726300144	2.55422210599485
-2.26199543496609	-4.32986715580508	0.970232456433379	0.231272927792138

构建三层 BP 神经网络，各层分别为 2，5，1 个节点，使用上述数据分别使用逐个样本修正和成批样本修正算法进行训练，得到训练后网络以及误差曲线，结果显示逐个样本修正算法迭代了 15000 次，最后终止误差为 1.14%；成批样本修正算法迭代了 15000 次，最后终止误差为 1.12%。

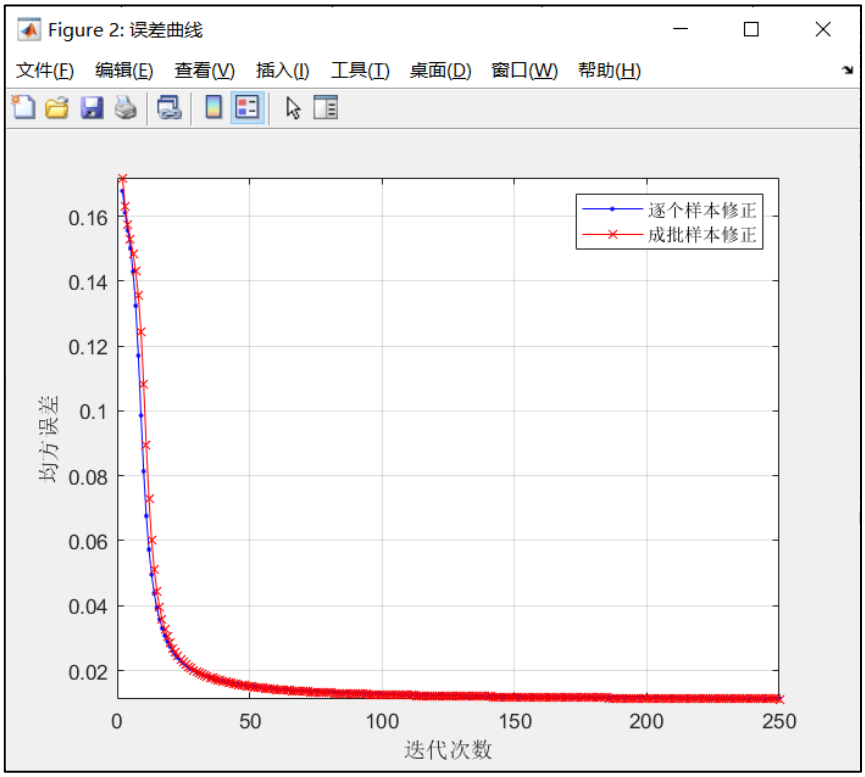

图 4-1 完整的误差曲线

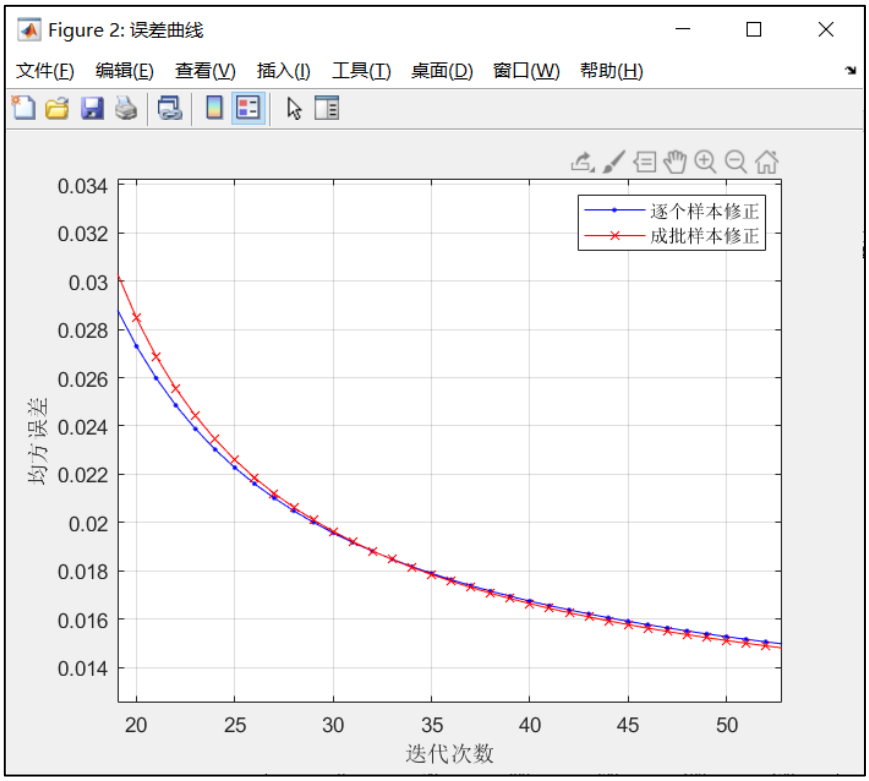

图 4-2 误差曲线局部放大

执行 testing 脚本，使用训练好的网络对模拟的测试样本数据进行分类，得到分类错误样本个数，结果表明训练后的网络的二分类效果很好，说明 BP 网络构建使用成功。

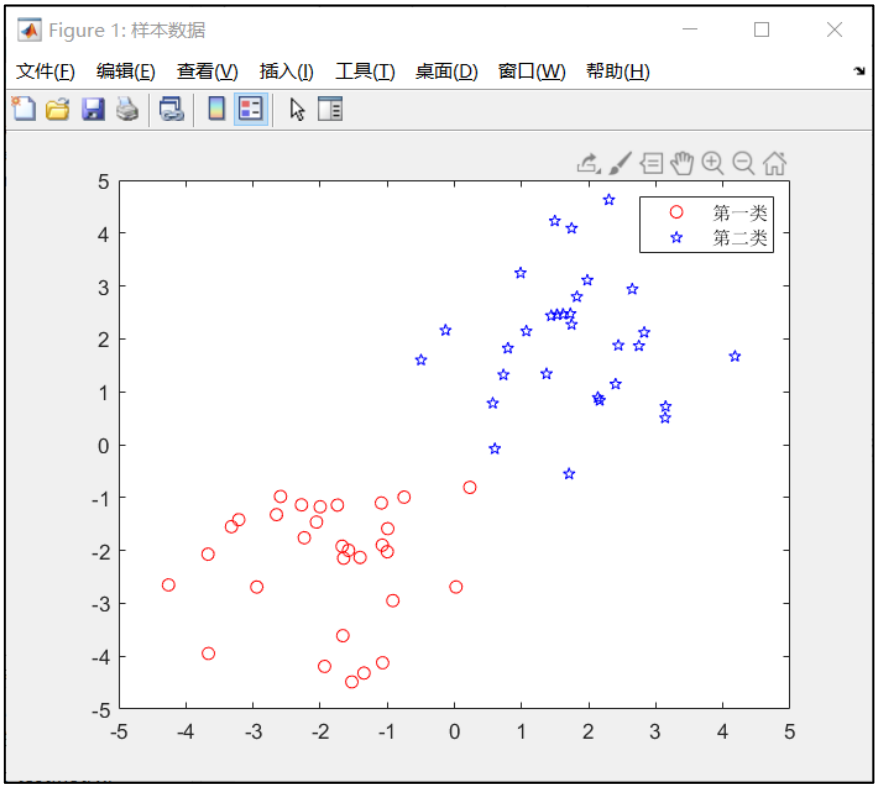

图 4-3 测试样本数据分布

```
>> testing

ans =

     0     0
```

图 4-4 分类错误样本个数

结论

本次使用 MATLAB 语言构建 BP 神经网络进行二维数据二分类操作，并对逐个样本修正算法和成批样本修正算法进行对比，根据结果可以得出如下结论：

(1) 无论逐个样本修正还是成批样本修正的 BP 算法，在当前参数和样本下 BP 算法随着迭代次数的增加,均方误差总是不断减小的，但是减小的幅度也是不断减小的，可以推测最终逼近一个阈值，该阈值和样本本身的分布有关。

(2) 成批样本修正较逐个样本修正而言在当前情况下收敛速度稍微快一些，原因是因为成批样本修正每次修正权值时综合考虑了所有样本的局部梯度，因此收敛速度较快。

参考文献

[1] 常凯. 基于神经网络的数据挖掘分类算法比较和分析研究[D].安徽大学,2014.
[2] 莫礼平,樊晓平.BP 神经网络在数据挖掘分类中的应用[J].吉首大学学报(自然科学版),2006(01):59-62.
[3] Stormy Attaway. MATLAB: A Practical Introduction to Programming and Problem Solving Third Edition.
[4] BP 神经网络原理及其 matlab 实现.
https://blog.csdn.net/bxk88/article/details/46120809

附录：程序代码

```
//模拟二维样本生成函数 sample_create：
function [Xin,Yd] = sample_create(N)
% 产生二维的测试样本和理想输出

% N;%样本数
%% 产生绘制样本
%第一类
u1 = [-2 -2]';
sigma1 = [1 0;0 1];
r1 = mvnrnd(u1,sigma1,N);
%第二类
u2 = [2 2]';
sigma2 = [1 0;0 2];
r2 = mvnrnd(u2,sigma2,N);

figure('Name','样本数据')
plot(r1(:,1),r1(:,2),'ro');
hold on;
plot(r2(:,1),r2(:,2),'bp');
legend('第一类','第二类')

Xin = [r1;r2];%前面 R1 类，后面 R2 类
Yd = [zeros(N,1);ones(N,1)];%前面 R1 类 0，后面 R2 类 1

end

//神经网络的生成函数 create_net：
function net = create_net(lNum,pNum,varargin)
%% 产生神经网络 net，参数：
% lNum 表示层数，数值
% pNum 表示各层的节点数目，一维向量
% net 输出的 net 结构体
% 其它输入：
% minErr 误差界
% maxIter 最多迭代次数
% enta 步长，通常 0-1 之间
```

```
% alpha 惯性系数，0-1 之间


net.lNum = lNum;%层数
net.pNum = pNum;%各层的节点数目
net.ErrIter = inf;%误差
if length(varargin) == 4
    net.minErr = varargin{1};%误差界
    net.maxIter = varargin{2};%最多迭代次数
    net.enta = varargin{3};%步长，通常 0-1 之间
    net.alpha  = varargin{4};%惯性系数，0-1 之间
else
    % 默认参数
    net.minErr = 0.1;%误差界
    net.maxIter = 300;%最多迭代次数
    net.enta = 0.5;%步长，通常 0-1 之间
    net.alpha  = 0.5;%惯性系数，0-1 之间
end

%初始化各个节点权值
for k=1:(lNum-1)
    w{k} = rand(pNum(k+1),pNum(k));%各层权值，本层节点数 x 上一层节点数，这样安排有利于输出相乘
    deltaw{k} = zeros(pNum(k+1),pNum(k)); %初始化该变量用于后面的计算需要
end
net.w=w;%输出神经网络的权向量
net.deltaw = deltaw;%缓存的上一步 w 变换量
end

//神经网络的训练函数逐个样本修正 net_train_per：
function net = net_train_per(net,Xin,Yd)
%% 根据输入训练样本，修正网络节点权值（逐个样本修正），采用 BP 算法，参数：
% net 神经网络，权值修正前后
% Xin 输入样本，[样本数 x 特征维度] 的矩阵
% Yd 理想输出分类

%% 参数初始化
% 迭代参数
minErr = net.minErr;%误差界
maxIter = net.maxIter;%最多迭代次数
```

```
% 计算所需参数
lNum = net.lNum;%层数
pNum = net.pNum;%各层的节点数目

% 迭代参数
ErrIter = [];%误差记录
sampleN = size(Xin,1);%输入样本数量

%% BP 算法（逐个样本修正）
Errtmp = inf;%初始误差
for k = 1:maxIter%*sampleN%可以保证循环完成
    p = mod(k-1,sampleN)+1;%当前轮样本标号
    %% 从前向后计算各个单元输出直到最后一个
    xin = Xin(p,:)';%本次输入变量
    yd = Yd(p,:)';%本次理想输出
    [yo,layer_out] = net_test(net, xin);%计算各个输入样本的各层输出

    %% 终止判断
    yerr = yd - yo; %最终分类误差
    if p == 1 %本轮刚开始
        Errtmp = Errtmp/sampleN;%误差均值
        ErrIter = [ErrIter,Errtmp];%保存上一轮轮误差
        if Errtmp<minErr
            break;%终止迭代求解
        end
        Errtmp = sum(yerr.^2);
    else%本轮样本正在循环
        Errtmp = Errtmp + sum(yerr.^2);%累计误差
    end

    %% 从后向前各层依次计算误差并修正权值，逐个修正
    net.w = net_back_adjust(net,layer_out,yd);

end
net.ErrIter = ErrIter;%迭代误差曲线，用于绘制收敛速度对比

fprintf('终止误差=%1.4e,  迭代步数=%d\n',ErrIter(end),k);
end
```

```
function [y,layer_out] = net_test(net,x)
%% 根据 net 计算向量 x 的输出 y 以及各层的输出 layer_out
sigmoid = @(x) (1./(1+exp(-x)));%函数声明

layer_out{1} = x;%X 必须是一个一维向量
for k=2:net.lNum
    %依次计算各层的输出
    w = net.w{k-1};%前一层->当前层的权值向量
    layer_out{k} = sigmoid(w*layer_out{k-1});%当前层输出
end
y = layer_out{net.lNum};
end



function wadj = net_back_adjust(net,layer_out,yd)
%% 反向利用误差校正权值
enta = net.enta;%步长，通常 0-1 之间
alpha  = net.alpha;%惯性系数，0-1 之间
N = length(layer_out);%层数

%% 计算各层的输出误差
err_out{N} = -layer_out{N}.*(1-layer_out{N}).*(yd - layer_out{N});%输出层
for k=(N-1):-1:1
    errTmp = net.w{k}'*err_out{k+1} ;%后一层误差和映射到当前层各个节点
    err_out{k} = layer_out{k}.*(1-layer_out{k}).*errTmp;%书上公式
    %权值修正
    net.deltaw{k} = -enta*err_out{k+1}*layer_out{k}' + net.deltaw{k}*alpha;%上
一步的惯性也加上了
    wadj{k} = net.w{k} + net.deltaw{k};
end

end

//神经网络的训练函数成批样本修正 net_train_mass：
function net = net_train_mass(net,Xin,Yd)
%% 根据输入训练样本，修正网络节点权值（成批样本修正），采用 BP 算法，参数：
% net 神经网络，权值修正前后
% Xin 输入样本，[样本数 x 特征维度] 的矩阵
% Yd 理想输出分类
```

```
%% 参数初始化
% 迭代参数
minErr = net.minErr;%误差界
maxIter = net.maxIter;%最多迭代次数

% 计算所需参数
lNum = net.lNum;%层数
pNum = net.pNum;%各层的节点数目

% 迭代参数
ErrIter = [];%误差记录
sampleN = size(Xin,1);%输入样本数量

%% BP 算法（成批样本修正）
Errtmp = inf;%初始误差
for k = 1:maxIter%*sampleN%可以保证循环完成
    p = mod(k-1,sampleN)+1;%当前轮样本标号
    %% 从前向后计算各个单元输出直到最后一个
    xin = Xin(p,:)';%本次输入变量
    yd = Yd(p,:)';%本次理想输出
    [yo,layer_out{p}] = net_test(net, xin);%计算各个输入样本的各层输出

    %% 终止判断
    yerr = yd - yo; %最终分类误差
    if p == 1 %本轮刚开始
        Errtmp = Errtmp/sampleN;%误差均值
        ErrIter = [ErrIter,Errtmp];%保存上一轮轮误差
        if Errtmp<minErr
            break;%终止迭代求解
        end
        Errtmp = sum(yerr.^2);
    else%本轮样本正在循环
        Errtmp = Errtmp + sum(yerr.^2);%累计误差
    end

    %% 从后向前各层依次计算误差，并最后用于成批修正权值
    err_out{p} = net_err(net,layer_out{p},yd);
    if p == sampleN %完成一轮，批量修正权值
        net.w = net_back_adjust(net,layer_out,err_out);
    end
```

```
end
net.ErrIter = ErrIter;%迭代误差曲线，用于绘制收敛速度对比

fprintf('终止误差=%1.4e,  迭代步数=%d\n',ErrIter(end),k);
end

function [y,layer_out] = net_test(net,x)
%% 根据 net 计算向量 x 的输出 y 以及各层的输出 layer_out
sigmoid = @(x) (1./(1+exp(-x)));%函数声明

layer_out{1} = x;%X 必须是一个一维向量
for k=2:net.lNum
    %依次计算各层的输出
    w = net.w{k-1};%前一层->当前层的权值向量
    layer_out{k} = sigmoid(w*layer_out{k-1});%当前层输出
end
y = layer_out{net.lNum};
end


function err_out = net_err(net,layer_out,yd)
%% 计算各层误差
N = length(layer_out);%层数

%% 计算各层的输出误差
err_out{N} = -layer_out{N}.*(1-layer_out{N}).*(yd - layer_out{N});%输出层
for k=(N-1):-1:1
    errTmp = net.w{k}'*err_out{k+1} ;%后一层误差和映射到当前层各个节点
    err_out{k} = layer_out{k}.*(1-layer_out{k}).*errTmp;%书上公式
end

end


function wadj = net_back_adjust(net,layer_out,err_out)
%% 反向利用误差校正权值
enta = net.enta;%步长，通常 0-1 之间
alpha  = net.alpha;%惯性系数，0-1 之间
```

```
sampleN = length(layer_out);%采样点数
N=length(layer_out{1});%层数

%% 权值修正
for m=(N-1):-1:1%对各层
    deltaw = 0;%初始化
    for n=1:sampleN%对各个样本求和
        deltaw = deltaw -enta*err_out{n}{m+1}*layer_out{n}{m}';
    end
    net.deltaw{m} = deltaw + net.deltaw{m}*alpha;%上一步的惯性也加上了
    wadj{m} = net.w{m} + net.deltaw{m};%权值修正
end

end
```

//网络分类函数 net_test：

```
function y = net_test(net,x)
%% 根据 net 计算向量 x 的输出 y 以及各层的输出 layer_out
sigmoid = @(x) (1./(1+exp(-x)));%函数声明

y = [];
for p =1:size(x,1)
    layer_out{1} = x(p,:)';%X 必须是一个一维向量
    for k=2:net.lNum
        %依次计算各层的输出
        w = net.w{k-1};%前一层->当前层的权值向量
        layer_out{k} = sigmoid(w*layer_out{k-1});%当前层输出
    end
    y = [y;layer_out{k}'];
end

end
```

//训练神经网络脚本 training：

```
clear; clc; close all;
%% 训练数据初始化
vecLen=2;%特征维度
sampleN=30;%各类样本数据个数
[Xin,Yd] = sample_create(sampleN);
```

```
%% 产生神经网络
net = create_net(3,[vecLen,5,1],...%产生和初始化神经网络，3 层，各层分别为 2、5、
1 个节点
    1e-3,500*sampleN,0.1,0.0);%设定各个迭代控制参数
%此处可以直接修改 net 的属性用于迭代

%% 训练网络
net1 = net_train_per(net,Xin,Yd);%逐个样本训练
net2 = net_train_mass(net,Xin,Yd);%成批样本训练


figure('Name','误差曲线');
plot(net1.ErrIter,'b.-'),hold on; grid on
plot(net2.ErrIter,'rx-');
ylim([min([net1.ErrIter(end),net2.ErrIter(end)]),max([net1.ErrIter(1),net2.Er
rIter(1)])]);
legend('逐个样本修正','成批样本修正');xlabel('迭代次数');ylabel('均方误差');

//测试神经网络脚本 testing：
%% 模拟测试样本
sampleN=30;%各类样本数据个数
[Xin,Yd] = sample_create(sampleN);
%% 识别样本
yo1 = net_test(net1,Xin);%逐个样本修正
yo1 = yo1>=0.5;
per1 = sum(yo1 ~=Yd)/length(Yd);
yo2 = net_test(net2,Xin);%成批样本修正
yo2 = yo2>=0.5;
per2 = sum(yo2 ~=Yd)/length(Yd);
[per1,per2]
```