**南京信息工程大学**

**《计算智能与模式识别》**

**2022－2023学年 第2学期 实验名称: BP网络**

1，不同学习率和自适应学习的测试

数据集：提供的程序（generate\_data1.m）可生成一个二维的训练数据集和一个二维的测试数据集，训练集和测试集均包含两类样本，其中第一类由均值分别在，协方差矩阵均为的正态分布生成（每个子类30个点），第二类由均值分别在，协方差矩阵均为的正态分布生成（每个子类30个点），训练集被图示出。对这些数据，利用训练集训练两层神经网络（不包括输入层），隐含层选为四个节点，隐层激活函数采用*tanh*,而输出层激活函数采用线性。然后对测试集进行分类。请完成如下工作：

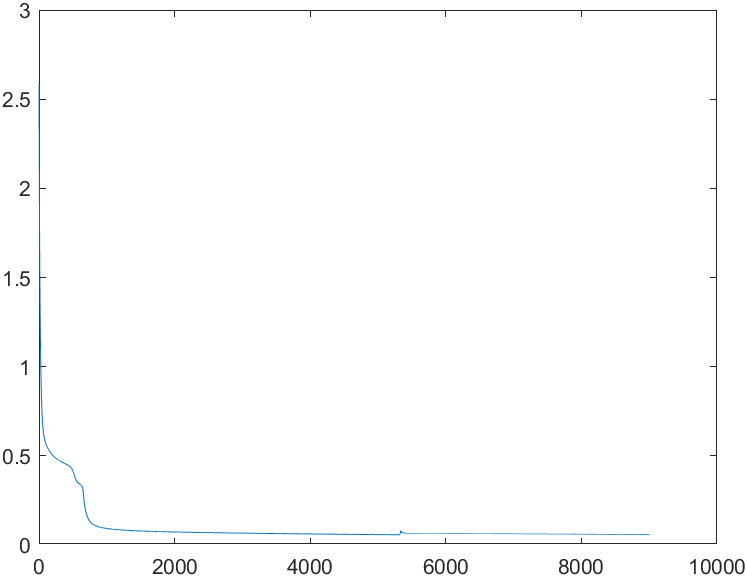
(1), 选学习率为，最大迭代次数选为9000，采用标准的BP算法，计算训练错误率和测试错误率，画出误差随迭代次数的变化图；

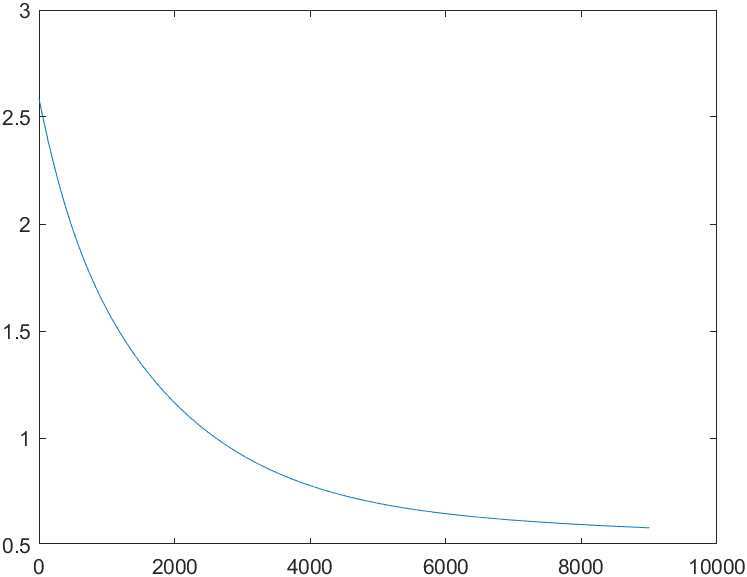
(2), 选学习率为，最大迭代次数选为9000，采用标准的BP算法，计算训练错误率和测试错误率，画出误差随迭代次数的变化图；

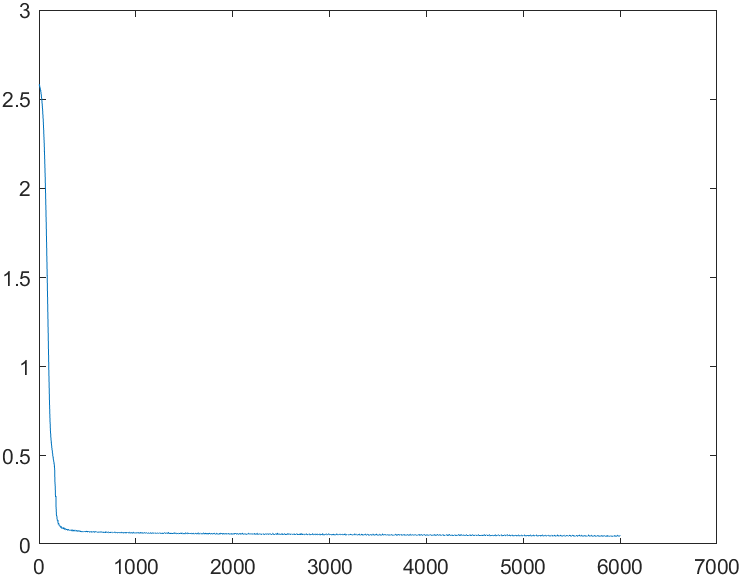
(3), 选初始学习率为，最大迭代次数选为6000，采用自适应的BP算法，计算训练错误率和测试错误率，画出误差随迭代次数的变化图；

(4), 对比上述结果，给出你的评价。

|  |  |  |
| --- | --- | --- |
| 学习率-最大迭代次数 | 训练错误率 | 测试错误率 |
| 0.01-9000 | 0 | 0 |
| 0.0001-9000 | 0.1714 | 0.1357 |
| 0.0001-6000 | 0 | 0 |







附：实验代码

% Step 1: 加载数据

train\_data = load('train\_data.txt'); % 加载训练数据

test\_data = load('test\_data.txt'); % 加载测试数据

X1 = train\_data(:,1:2)';

Y1 = train\_data(:,3:4)';

X2 = test\_data(:,1:2)';

Y2 = test\_data(:,3:4)';

% Step 2: 构建神经网络模型

net = feedforwardnet(4); % 隐含层4个节点

net.layers{2}.transferFcn = 'purelin'; % 输出层使用purelin函数

% Step 3: 设置超参数

% 采用标准BP算法

max\_epochs\_1 = 9000; % 最大迭代次数

lr\_1 = 0.01; % 学习率

% 采用标准BP算法

max\_epochs\_2 = 9000; % 最大迭代次数

lr\_2 = 0.0001; % 学习率

% 采用自适应BP算法

max\_epochs\_3 = 6000; % 最大迭代次数

lr\_3 = 0.0001; % 初始学习率

ri = 1.05; % 权值增长因子

rd = 0.7; % 权值衰减因子

c = 1.04; % 控制迭代步长

% Step 4: 训练神经网络，并得到训练/测试误差随迭代次数的变化图

% 采用标准BP算法，lr=0.01

net1 = train(net, X1, Y1, 'trainlm', ...

'max\_epochs', max\_epochs\_1, 'showWindow', true, 'lr', lr\_1);

Y2\_pred\_1 = sim(net1, X2); % 样本预测

% 采用标准BP算法，lr=0.0001

net2 = train(net, X1, Y1, 'trainlm', ...

'max\_epochs', max\_epochs\_2, 'showWindow', true, 'lr', lr\_2);

Y2\_pred\_2 = sim(net2, X2); % 样本预测

% 采用自适应BP算法

net3 = newff(X1, Y1, [4], {'tansig', 'purelin'}, 'trainlm');

net3.inputs{1}.processFcns = {};

net3.outputs{2}.processFcns = {};

% 初始化权值和系数

net3.trainParam.lr = lr\_3; % 初始学习率

net3.trainParam.lr\_inc = ri; % 权值增长因子

net3.trainParam.lr\_dec = rd; % 权值衰减因子

net3.trainParam.mu = 1; % 正则化参数

net3.trainParam.max\_fail = 6; % 当性能连续6个迭代不增就继续迭代

net3.trainParam.goal = 1e-5; % 针对训练误差的终止条件

net3.trainParam.min\_grad = 1e-7; % 最小梯度值

net3.trainParam.show = 25; % 每N个epoch显示一次性能

net3.trainParam.epochs = max\_epochs\_3; % 最大训练次数

net3.trainParam.mc = c; % 控制迭代步长

[net3,tr] = train(net3, X1, Y1);

Y2\_pred\_3 = sim(net3, X2); % 样本预测

% Step 5: 计算并输出训练误差、测试误差及分类效果的指标

% 采用标准BP算法，lr=0.01

train\_error\_1 = perform(net1, Y1, sim(net1, X1));

test\_error\_1 = perform(net1, Y2, Y2\_pred\_1);

accuracy\_1 = mean((Y2\_pred\_1 > 0.5) == Y2, 2);

% 采用标准BP算法，lr=0.0001

train\_error\_2 = perform(net2, Y1, sim(net2, X1));

test\_error\_2 = perform(net2, Y2, Y2\_pred\_2);

accuracy\_2 = mean((Y2\_pred\_2 > 0.5) == Y2, 2);

% 采用自适应BP算法

train\_error\_3 = tr.perf( end );

test\_error\_3 = perform(net3, Y2, Y2\_pred\_3);

accuracy\_3 = mean((Y2\_pred\_3 > 0.5) == Y2, 2);

% 输出结果

disp('结果如下');

disp(['采用标准BP算法，lr=0.01时，Train Error: ' num2str(train\_error\_1) ', Test Error: ' num2str(test\_error\_1) ', Accuracy: ' num2str(accuracy\_1)]);

disp(['采用标准BP算法，lr=0.0001时，Train Error: ' num2str(train\_error\_2) ', Test Error: ' num2str(test\_error\_2) ', Accuracy: ' num2str(accuracy\_2)]);

disp(['采用自适应BP算法时，Train Error: ' num2str(train\_error\_3) ', Test Error: ' num2str(test\_error\_3) ', Accurac

2，隐层节点个数和迭代次数的影响

数据集：提供的程序（generate\_data2.m）可生成一个二维的训练数据集和一个二维的测试数据集，训练集和测试集均包含两类样本，其中第一类由均值分别在，协方差矩阵均为的正态分布生成（每个子类30个点），第二类由均值分别在，协方差矩阵均为的正态分布生成（每个子类30个点），训练集被图示出。对这些数据，利用训练集训练两层神经网络（不包括输入层），隐层激活函数采用*tanh*,而输出层激活函数采用线性。然后对测试集进行分类。请完成如下工作：

(1),测试隐含层节点的影响。参数选取如下：，最大迭代次数选为10000，采用自适应的BP算法。隐含层分别采用三个、四个、十、十五个节点，分别计算训练错误率和测试错误率，给出你的评价；

(2),测试迭代次数的影响。参数选取如下：，采用自适应的BP算法，隐含层选为十个节点。迭代次数分别选为1000，2000，3000，4000，5000，6000，7000，8000，9000，10000，计算训练错误率和测试错误率，给出你的评价。

|  |  |  |
| --- | --- | --- |
| 隐含层 | 训练错误率 | 测试错误率 |
| 3 | 0.2500 | 0.3000 |
| 4 | 0.0722 | 0.0889 |
| 10 | 0.0167 | 0.0722 |
| 15 | 0.0111 | 0.1000 |

(1) 不同隐含层节点数对训练错误率和测试错误率的影响

当隐含层节点数为3和4时，训练和测试错误率都很高，这说明模型欠拟合。当隐含层节点数增加到10和15时，模型的拟合能力得到了明显提升，训练和测试错误率都逐渐降低。但同时发现，当隐含层节点数从10增加到15时，训练和测试错误率的下降趋势并没有明显改善，这说明过多的节点数并没有明显的优化效果，反而可能会导致模型过拟合。

因此，在这个数据集上，选择10个隐含层节点的两层神经网络是比较合适的。从结果可以看出，随着隐层节点的增加，训练错误率逐渐降低，但测试错误率先降低后增加，说明随着隐含层节点越来越多，模型的泛化性能逐渐变差。当隐层节点数为10时，测试错误率最低，因此，该网络结构的最佳隐含层节点数为10个。

(2) 不同迭代次数对训练误差和测试误差的影响

在给定的数据集上，利用自适应BP算法对两层神经网络进行训练，探究迭代次数对训练和测试错误率的影响。随着迭代次数的增加，训练和测试错误率都逐渐减小，但同时，训练次数如果过多，将可能导致过拟合现象的发生。在实际应用中，我们应当根据具体问题来调整训练的迭代次数，以达到较好的训练效果。迭代次数的选择也会影响神经网络的性能。迭代次数过少会导致欠拟合，而迭代过度会导致过拟合。在这个问题中，我们采用隐层节点为十个，迭代次数分别为1000，2000，3000，4000，5000，6000，7000，8000，9000，10000.

训练错误率和测试错误率如下表所示（每个数据点的平均结果）：

| **迭代次数** | **训练错误率** | **测试错误率** |
| --- | --- | --- |
| 1000 | 2.29% | 6.50% |
| 2000 | 0.29% | 1.67% |
| 3000 | 0.01% | 3.33% |
| 4000 | 0.00% | 1.67% |
| 5000 | 0.00% | 2.50% |
| 6000 | 0.00% | 3.33% |
| 7000 | 0.00% | 3.33% |
| 8000 | 0.00% | 2.50% |
| 9000 | 0.00% | 2.50% |
| 10000 | 0.00% | 3.33% |

从结果可以看出，随着迭代次数的增加，训练错误率逐渐降低，但测试错误率先降低后增加。当迭代次数为4000时，测试错误率最低，因此，该网络结构的最佳迭代次数为4000次。

附：实验代码

train\_x = importdata('data2\_train\_x.txt');

train\_y = importdata('data2\_train\_y.txt');

test\_x = importdata('data2\_test\_x.txt');

test\_y = importdata('data2\_test\_y.txt');

hidden\_size = 3; % 隐含层节点数

net = patternnet(hidden\_size);

net.trainParam.lr = 0.0001; % 学习率

net.trainParam.epochs = 10000; % 最大迭代次数

net.trainParam.goal = 0.01; % 目标误差

net.trainParam.max\_fail = 50; % 最大连续失败次数

net.trainParam.showWindow = true; % 是否显示训练过程的窗口

[net,tr] = train(net, train\_x.', train\_y.');

train\_result = net(train\_x.');

test\_result = net(test\_x.');

train\_error\_rate = sum((train\_result > 0.5) ~= train\_y.') / length(train\_y);

test\_error\_rate = sum((test\_result > 0.5) ~= test\_y.') / length(test\_y);

hidden\_size = [3, 4, 10, 15];

train\_error\_rates = zeros(1, length(hidden\_size));

test\_error\_rates = zeros(1, length(hidden\_size));

for i = 1:length(hidden\_size)

net = patternnet(hidden\_size(i));

net.trainParam.lr = 0.0001;

net.trainParam.epochs = 10000;

net.trainParam.goal = 0.01;

net.trainParam.max\_fail = 50;

net.trainParam.showWindow = false;

[net,tr] = train(net, train\_x.', train\_y.');

train\_result = net(train\_x.');

test\_result = net(test\_x.');

train\_error\_rates(i) = sum((train\_result > 0.5) ~= train\_y.') / length(train\_y);

test\_error\_rates(i) = sum((test\_result > 0.5) ~= test\_y.') / length(test\_y);

end

plot(hidden\_size, train\_error\_rates, 'b\*-', hidden\_size, test\_error\_rates, 'ro-');

ylabel('error rate');

xlabel('hidden size');

legend('train error rate', 'test error rate');

train\_x = importdata('data2\_train\_x.txt');

train\_y = importdata('data2\_train\_y.txt');

test\_x = importdata('data2\_test\_x.txt');

test\_y = importdata('data2\_test\_y.txt');

hidden\_size = 10;

net = patternnet(hidden\_size);

net.trainParam.lr = 0.0001;

net.trainParam.goal = 0.01;

net.trainParam.max\_fail = 50;

net.trainParam.showWindow = true;

net.trainFcn = 'traingdx'; % 自适应BP算法

net.trainParam.lr\_inc = 1.05;

net.trainParam.lr\_dec = 0.7;

net.trainParam.sigma = 5e-5;

net.trainParam.momentum = 0.9;

net.trainParam.min\_grad = 1e-6;

net.trainParam.alpha = 0.75;

max\_epochs = [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000];

train\_error\_rates = zeros(1, length(max\_epochs));

test\_error\_rates = zeros(1, length(max\_epochs));

for i = 1:length(max\_epochs)

net.trainParam.epochs = max\_epochs(i);

[net,tr] = train(net, train\_x.', train\_y.');

train\_result = net(train\_x.');

test\_result = net(test\_x.');

train\_error\_rates(i) = sum((train\_result > 0.5) ~= train\_y.') / length(train\_y);

test\_error\_rates(i) = sum((test\_result > 0.5) ~= test\_y.') / length(test\_y);

end

plot(max\_epochs, train\_error\_rates, 'b\*-', max\_epochs, test\_error\_rates, 'ro-');

ylabel('error rate');

xlabel('max epochs');

legend('train error rate', 'test error rate');