MATLAB 程序代码--BP 神经网络的设计实例

例 1 采用动量梯度下降算法训练 BP 网络。

训练样本定义如下：

输入矢量为

p =[-1 -2 3 1

-1 1 5 -3]

目标矢量为 t = [-1 -1 1 1]

解：本例的 MATLAB 程序如下：

```
close all
clear
echo on
clc
% NEWFF——生成一个新的前向神经网络
% TRAIN——对 BP 神经网络进行训练
% SIM——对 BP 神经网络进行仿真
pause
%    敲任意键开始
clc
%    定义训练样本
% P 为输入矢量
P=[-1,    -2,    3,    1;        -1,    1,    5,    -3];
% T 为目标矢量
T=[-1, -1, 1, 1];
pause;
clc
%    创建一个新的前向神经网络
net=newff(minmax(P),[3,1],{'tansig','purelin'},'traingdm')
%    当前输入层权值和阈值
inputWeights=net.IW{1,1}
inputbias=net.b{1}
%    当前网络层权值和阈值
layerWeights=net.LW{2,1}
layerbias=net.b{2}
pause
clc
%    设置训练参数
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
```

```
bpnet=newff(pr,[12 4],{'logsig', 'logsig'}, 'traingdx', 'learngdm');
%建立 BP 神经网络， 12 个隐层神经元，4 个输出神经元
%tranferFcn 属性 'logsig' 隐层采用 Sigmoid 传输函数
%tranferFcn 属性 'logsig' 输出层采用 Sigmoid 传输函数
%trainFcn 属性 'traingdx' 自适应调整学习速率附加动量因子梯度下降反向传播算法训练函数
%learn 属性 'learngdm' 附加动量因子的梯度下降学习函数
net.trainParam.epochs=1000;%允许最大训练步数 2000 步
net.trainParam.goal=0.001; %训练目标最小误差 0.001
net.trainParam.show=10; %每间隔 100 步显示一次训练结果
net.trainParam.lr=0.05; %学习速率 0.05
```

```
net.trainParam.epochs = 1000;
net.trainParam.goal = 1e-3;
pause
clc
%    调用 TRAINGDM 算法训练 BP 网络
[net,tr]=train(net,P,T);
pause
clc
%    对 BP 网络进行仿真
A = sim(net,P)
%    计算仿真误差
E = T - A
MSE=mse(E)
pause
clc
echo off
```

例 2 采用贝叶斯正则化算法提高 BP 网络的推广能力。在本例中，我们采用两种训练方法，即 L-M 优化算法（trainlm）和贝叶斯正则化算法（trainbr），用以训练 BP 网络，使其能够拟合某一附加有白噪声的正弦样本数据。其中，样本数据可以采用如下 MATLAB 语句生成：

输入矢量：P = [-1:0.05:1];

目标矢量：randn(’seed’,78341223);

T = sin(2*pi*P)+0.1*randn(size(P));

解：本例的 MATLAB 程序如下：

```
close all
clear
```

```
echo on
clc
% NEWFF——生成一个新的前向神经网络
% TRAIN——对 BP 神经网络进行训练
% SIM——对 BP 神经网络进行仿真
pause
%    敲任意键开始
clc
%    定义训练样本矢量
% P 为输入矢量
P = [-1:0.05:1];
% T 为目标矢量
randn('seed',78341223); T = sin(2*pi*P)+0.1*randn(size(P));
%    绘制样本数据点
plot(P,T,'+');
echo off
hold on;
plot(P,sin(2*pi*P),':');
%    绘制不含噪声的正弦曲线
echo on
clc
pause
clc
%    创建一个新的前向神经网络
net=newff(minmax(P),[20,1],{'tansig','purelin'});
pause
clc
echo off
clc
disp('1.    L-M 优化算法 TRAINLM'); disp('2.    贝叶斯正则化算法 TRAINBR');
choice=input('请选择训练算法(1,2):');
figure(gcf);
if(choice==1)
    echo on
    clc
    %    采用 L-M 优化算法 TRAINLM
    net.trainFcn='trainlm';
    pause
    clc
    %    设置训练参数
    net.trainParam.epochs = 500;
    net.trainParam.goal = 1e-6;
    net=init(net);
    %    重新初始化
```

```
        pause
        clc
elseif(choice==2)
        echo on
        clc
        %    采用贝叶斯正则化算法 TRAINBR
        net.trainFcn='trainbr';
        pause
        clc
        %    设置训练参数
        net.trainParam.epochs = 500;
        randn('seed',192736547);
        net = init(net);
        %    重新初始化
        pause
        clc
end
%    调用相应算法训练 BP 网络
[net,tr]=train(net,P,T);
pause
clc
%    对 BP 网络进行仿真
A = sim(net,P);
%    计算仿真误差
E = T - A;
MSE=mse(E)
pause
clc
%    绘制匹配结果曲线
close all;
plot(P,A,P,T,'+',P,sin(2*pi*P),':');
pause;
clc
echo off
```

通过采用两种不同的训练算法，我们可以得到如图 1 和图 2 所示的两种拟合结果。图中的实线表示拟合曲线，虚线代表不含白噪声的正弦曲线，“＋”点为含有白噪声的正弦样本数据点。显然，经 trainlm 函数训练后的神经网络对样本数据点实现了“过度匹配”，而经 trainbr 函数训练的神经网络对噪声不敏感，具有较好的推广能力。

值得指出的是，在利用 trainbr 函数训练 BP 网络时，若训练结果收敛，通常会给出提示信息“Maximum MU reached”。此外，用户还可以根据 SSE 和 SSW 的大小变化情况来判断训练是否收敛：当 SSE 和 SSW 的值在经过若干步迭代后处于恒值时，则通常说明

网络训练收敛，此时可以停止训练。观察 trainbr 函数训练 BP 网络的误差变化曲线,可见，当训练迭代至 320 步时，网络训练收敛，此时 SSE 和 SSW 均为恒值，当前有效网络的参数（有效权值和阈值）个数为 11.7973。

例 3 采用“提前停止”方法提高 BP 网络的推广能力。对于和例 2 相同的问题，在本例中我们将采用训练函数 traingdx 和“提前停止”相结合的方法来训练 BP 网络，以提高 BP 网络的推广能力。

解：在利用“提前停止”方法时，首先应分别定义训练样本、验证样本或测试样本，其中，验证样本是必不可少的。在本例中，我们只定义并使用验证样本，即有
验证样本输入矢量：val.P = [-0.975:.05:0.975]
验证样本目标矢量：val.T = sin(2*pi*val.P)+0.1*randn(size(val.P))
值得注意的是，尽管“提前停止”方法可以和任何一种 BP 网络训练函数一起使用，但是不适合同训练速度过快的算法联合使用，比如 trainlm 函数，所以本例中我们采用训练速度相对较慢的变学习速率算法 traingdx 函数作为训练函数。
本例的 MATLAB 程序如下：

```
close all
clear
echo on
clc
% NEWFF——生成一个新的前向神经网络
% TRAIN——对 BP 神经网络进行训练
% SIM——对 BP 神经网络进行仿真
pause
%    敲任意键开始
clc
%    定义训练样本矢量
% P 为输入矢量
P = [-1:0.05:1];
% T 为目标矢量
randn('seed',78341223);
T = sin(2*pi*P)+0.1*randn(size(P));
%    绘制训练样本数据点
plot(P,T,'+');
echo off
hold on;
plot(P,sin(2*pi*P),':');              %    绘制不含噪声的正弦曲线
echo on
clc
pause
clc
%    定义验证样本
val.P = [-0.975:0.05:0.975];          %    验证样本的输入矢量
val.T = sin(2*pi*val.P)+0.1*randn(size(val.P));          %    验证样本的目标矢量
```

```
pause
clc
%    创建一个新的前向神经网络
net=newff(minmax(P),[5,1],{'tansig','purelin'},'traingdx');
pause
clc
%    设置训练参数
net.trainParam.epochs = 500;
net = init(net);
pause
clc
%    训练 BP 网络
[net,tr]=train(net,P,T,[],[],val);
pause
clc
%    对 BP 网络进行仿真
A = sim(net,P);
%    计算仿真误差
E = T - A;
MSE=mse(E)
pause
clc
%    绘制仿真拟合结果曲线
close all;
plot(P,A,P,T,'+',P,sin(2*pi*P),':');
pause;
clc
echo off
```

下面给出了网络的某次训练结果，可见，当训练至第 136 步时，训练提前停止，此时的网络误差为 0.0102565。给出了训练后的仿真数据拟合曲线，效果是相当满意的。

```
[net,tr]=train(net,P,T,[],[],val);
TRAINGDX, Epoch 0/500, MSE 0.504647/0, Gradient 2.1201/1e-006
TRAINGDX, Epoch 25/500, MSE 0.163593/0, Gradient 0.384793/1e-006
TRAINGDX, Epoch 50/500, MSE 0.130259/0, Gradient 0.158209/1e-006
TRAINGDX, Epoch 75/500, MSE 0.086869/0, Gradient 0.0883479/1e-006
TRAINGDX, Epoch 100/500, MSE 0.0492511/0, Gradient 0.0387894/1e-006
TRAINGDX, Epoch 125/500, MSE 0.0110016/0, Gradient 0.017242/1e-006
TRAINGDX, Epoch 136/500, MSE 0.0102565/0, Gradient 0.01203/1e-006
TRAINGDX, Validation stop.
```

例 3 用 BP 网络估计胆固醇含量

这是一个将神经网络用于医疗应用的例子。我们设计一个器械，用于从血样的光谱组成的测量中得到血清的胆固醇含量级别，我们有 261 个病人的血样值，包括 21 种波长的谱线的数据，对于这些病人，我们得到了基于 光谱分类的胆固醇含量级别 hdl,ldl,vldl。

(1) 样本数据的定义与预处理。

choles_all.mat 文件中存储了网络训练所需要的全部样本数据。

利用 load 函数可以在工作空间中自动载入网络训练所需的输入数据 p 和目标数据 t，即

```
load choles_all
sizeofp = size (p)
sizeofp = 21    264
sizeoft = size (t)
sizeoft = 3     264
```

可见，样本集的大小为 264。为了提高神经网络的训练效率，通常要对样本数据作适当的预处理。首先，利用 prestd 函数对样本数据作归一化处理，使得归一化后的输入和目标数据均服从正态分布，即 [pn,meanp,stdp,tn,meant,stdt] = prestd(p,t);

然后，利用 prepca 函数对归一化后的样本数据进行主元分析，从而消除样本数据中的冗余成份，起到数据降维的目的。

```
[ptrans,transMat] = prepca(pn,0.001);
[R,Q] = size(ptrans)
R = 4 Q = 264
```

可见，主元分析之后的样本数据维数被大大降低，输入数据的维数由 21 变为 4。

(2) 对训练样本、验证样本和测试样本进行划分。

为了提高网络的推广能力和识别能力，训练中采用“提前停止”的方法，因此，在训练之前，需要将上面处理后的样本数据适当划分为训练样本集、验证样本集和测试样本集。

(3) 网络生成与训练。 选用两层 BP 网络，其中网络输入维数为 4，输出维数为 3，输出值即为血清胆固醇的三个指标值大小。网络中间层神经元数目预选为 5，传递函数类型选为 tansig 函数，输出层传递函数选为线性函数 purelin，训练函数设为 trainlm。网络的生成语句如下：

```
net = newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm');
```

利用 train 函数对所生成的神经网络进行训练，训练结果如下：

```
[net,tr]=train(net,ptr,ttr,[],[],val,test);
```

见，网络训练迭代至第 20 步时提前停止，这是由于验证误差已经开始变大。利用下面语句可以绘制出训练误差、验证误差和测试误差的变化曲线，如图 4.50 所示。由图可见，验证误差和测试误差的变化趋势基本一致，说明样本集的划分基本合理。由训练误差曲线可见，训练误差结果也是比较满意的。

(4) 网络仿真。 为了进一步检验训练后网络的性能，下面对训练结果作进一步仿真分析。利用 postreg 函数可以对网络仿真的输出结果和目标输出作线性回归分析，并得到两者的相关系数，从而可以作为网络训练结果优劣的判别依据。仿真与线性回归分析如下：

```
an = sim(net,ptrans);
a = poststd(an,meant,stdt);
for i=1:3
    figure(i)
    [m(i),b(i),r(i)] = postreg(a(i,:),t(i,:));
end
```

```
%导入原始测量数据
load choles_all;
%对原始数据进行规范化处理,prestd 是对输入数据和输出数据进行规范化处理，
%prepca 可以删除一些数据，适当地保留了变化不小于 0.01 的数据
[pn,meanp,stdp,tn,meant,stdt]=prestd(p,t);
[ptrans,transMat]=prepca(pn,0.001);
[R,Q]=size(ptrans)
%将原始数据分成几个部分作为不同用途四分已用于确证，四分一用于测试，二分一用于训练网络
iitst=2:4:Q;
iival=4:4:Q;
iitr=[1:4:Q 3:4:Q];
%vv 是确证向量，.P 是输入，.T 是输出，vt 是测试向量
vv.P=ptrans(:,iival);
vv.T=tn(:,iival);
vt.P=ptrans(:,iitst);
vt.T=tn(:,iitst);
ptr=ptrans(:,iitr);
ttr=tn(:,iitr);
%建立网络，隐层中设计 5 个神经元，由于需要得到的是 3 个目标，所以网络需要有 3 个输出
net=newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm');
%训练网络
net.trainParam.show=5;
[net,tr]=train(net,ptr,ttr,[],[],vv,vt);
%绘出训练过程中各误差的变化曲线
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',tr.epoch,tr.tperf,'-.b');
legend('训练','确证','测试',-1);
ylabel('平方误差');
xlabel('时间');
pause;
%将所有数据通过网络（包括训练，确证，测试），然后得到网络输出和相应目标进行线性回归，
%对网络输出进行反规范化变换，并绘出个各级别的线性回归结果曲线
an=sim(net,ptrans);
a=poststd(an,meant,stdt);
%得到 3 组输出，所以进行 3 次线性回归
for i=1:3
figure(i)
[m(i),b(i),r(i)] = postreg(a(i,:),t(i,:));
end
```

网络输出数据和目标数据作线性回归后，前面两个输出对目标的跟踪比较好，相应的 R 值

接近 0.9。而第三个输出却并不理想，我们很可能需要在这点上做更多工作。可能需要使用其它的网络结构（使用更多的隐层神经元），或者是在训练技术上使用贝页斯规范华而不实使用早停的方法。
把隐层数目改为 20 个时，网络训练的 3 种误差非常接近，得到的结果 R 也相应提高。但不代表神经元越多就越精确。
多层神经网络能够对任意的线性或者非线性函数进行逼近，其精度也是任意的。但是 BP 网络不一定能找到解。训练时，学习速率太快可能引起不稳定，太慢则要花费太多时间，不同的训练算法也对网络的性能有很大影响。BP 网络对隐层的神经元数目也是很敏感的，太少则很难适应，太多则可能设计出超适应网络。

注：例子均来自于互联网，本人的工作只是将多个例子整合在一起。