南京航空航天大学
Nanjing University of Aeronautics and Astronautics

软件可靠性课程实验报告

题　　目： BP 神经网络模型

院　　系： 计算机科学与技术学院

专　　业： 软件工程

学生姓名： 朱婷

学　　号： 161730305

二零一九 年 十一月 十五日

目录

1.引言

1．1编写目的

随着软件规模越来越大，结构日趋复杂，应用日趋广泛。软件危机依然是我们难以逾越的障碍， 加强软件工程管理，势在必行，势在必然！改进和提高软件可靠性，为部队提供可靠顶用的装备是我们的义务和责任!

此次试验以BP神经网络模型为核心，帮助我们理解神经网络的基本原理，掌握和应用BP神经网络模型。

2.模型理论

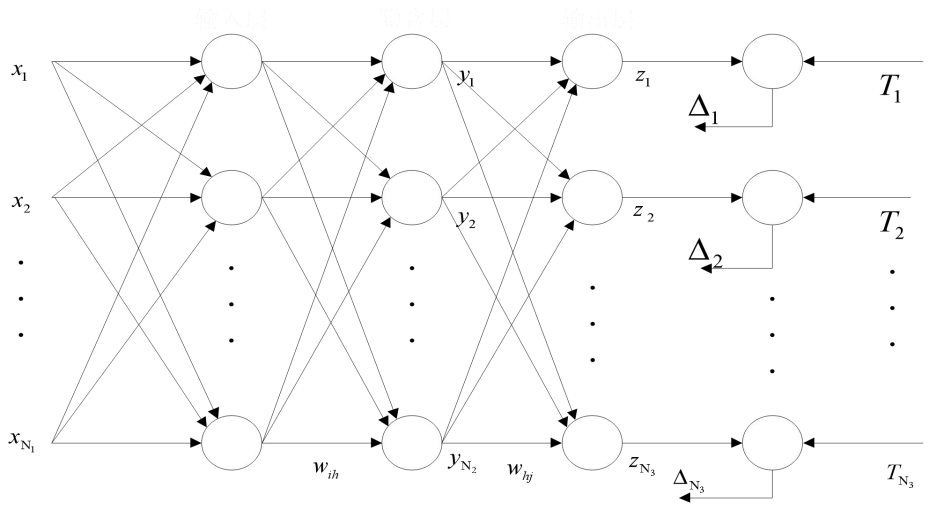
2.1 模型背景

20世纪80年代中期，David Runelhart。Geoffrey Hinton和Ronald W-llians、DavidParker等人分别独立发现了误差反向传播算法(Error Back Propagation Training)，简称BP，系统解决了多层神经网络隐含层连接权学习问题，并在数学上给出了完整推导。人们把采用这种算法进行误差校正的多层前馈网络称为BP网。

BP神经网络具有任意复杂的模式分类能力和优良的多维函数映射能力，解决了简单感知器不能解决的异或(Exclusive OR，XOR)和一些其他问题。从结构上讲，BP网络具有输入层、隐藏层和输出层；从本质上讲，BP算法就是以网络误差平方为目标函数、采用梯度下降法来计算目标函数的最小值。目前，在人工神经网络的实际应用中，绝大部分的神经网络模型都采用BP网络及其变化形式。它也是前向网络的核心部分，体现了人工神经网络的精华。

2.2 模型假设

- 假设模型一共有三层，输入层，中间（隐含层），输出层；
- 假设输入层有N1个节点，隐含层有N2个节点，输出层有N3个节点；
- 给定输入数据置于数组X[N1]中；
- 给定预期输出数据置于数组Do[N3]中；
- Wih[i][j]表示输入层第i个节点到隐含层第j个节点的连接权值；
- Who[j][k]表示隐含层第j个节点到输出层第k个节点的连接权值；
- 使用S型激活函数；



2.3 模型原理

基本原理：利用输出后的误差来估计输出层的直接前导层的误差，再用这个误差估计更前一层的误差，如此一层一层的反传下去，就获得了所有其他各层的误差估计。

学习的过程：神经网络先直接对外界输入样本进行处理，将这个原始输出与之前设置的原始输出进行误差比较，来估计输出层的直接前导层的误差，并通过误差更改网络的连接权值，如此往复，直到网络的输出不断地接近期望的输出。

2.4 模型推导

（1）前向传播

激活函数使用 S 型激活函数：

-输入

net=x1w1+x2w2+……+xnwn

-输出

$$y=f(net)=\frac{1}{1+e^{-net}}$$

·输入层到隐含层：

计算隐含层各神经元的输入值

$$net_{hj}=\sum_{i=0}^{N1} X_i W ih_{ij}+b_1$$

神经元 hj 的输出 outhj:

$$\mathrm{out}_{hj}=\frac{1}{1+e^{-net_{hj}}}$$;

• 隐含层到输出层：

计算输出层各神经元的输入值：

$$\mathrm{net}_{oj}=\sum_{i=0}^{N2} out_{\mathrm{hj}} * Who_{ij}+\mathrm{b}_2$$

神经元 0j 的输出 outoj:

$$\mathrm{out}_{oj}=\frac{1}{1+e^{-net_{oj}}}$$

（2）反向传播

• 先计算总误差

$$E\mathrm{total}=\sum_{k=0}^{N3}\frac{1}{2}(Do[k]-out_{ok}[k])^2$$

• 隐含层到中间层的权值更新

$$\frac{\partial E\mathrm{to}tal}{\partial who_{jk}}=\frac{\partial Etotal}{\partial out_{ok}} * \frac{\partial out_{ok}}{\partial net_{ok}} * \frac{\partial net_{ok}}{\partial who_{jk}}$$

$$E\mathrm{total}=\sum_{k=0}^{N3}\ \frac{1}{2}(Do[k]-out_{ok})^2$$

$$\frac{\partial E\mathrm{total}}{\partial out_{ok}}=-2 * \frac{1}{2}(Do[k]-out_{ok})$$

$$\frac{\partial \mathrm{out}_{\mathrm{ok}}}{\partial net_{ok}}=out_{ok}(1-out_{ok})$$

$$net_{ok}=\sum_{j=0}^{N2}\ Who_{jk} * out_{hj}+b2$$

$$\frac{\partial \mathrm{net}_{ok}}{\partial who_{jk}}=out_{hj}$$

用 So[k]表示输出层的误差；

$$So[k]=\frac{\partial Etotal}{\partial net_{ok}}=(out_{ok}-Do[k]) * out_{ok}(1-out_{ok})$$

$$\frac{\partial E\mathrm{total}}{\partial who_{jk}}=So[k] * out_{hj}$$

因此更新后的权值为

$$Who_{jk} = Who_{jk} - learnrate * \frac{\partial Etotal}{\partial Who_{jk}}$$

• 输入层到隐含层的权值更新

方法和上面相似

$$\frac{\partial E\text{total}}{\partial Wih_{ij}} = \frac{\partial Etotal}{\partial out_{hj}} * \frac{\partial out_{hj}}{\partial net_{hj}} * \frac{\partial net_{hj}}{\partial Wih_{ij}}$$

其中$\frac{\partial Etotal}{\partial out_{hj}} = \sum_{k=0}^{N3} \frac{\partial E_{ok}}{\partial out_{hj}}$

$$\frac{\partial E\text{total}}{\partial Wih_{\text{ij}}} = Sh[j]X[i] = (\sum_{k=0}^{N3} So[k] * Who_{jk}) * out_{hj}(1 - out_{hj}) * X[i]$$

因此更新权值后为

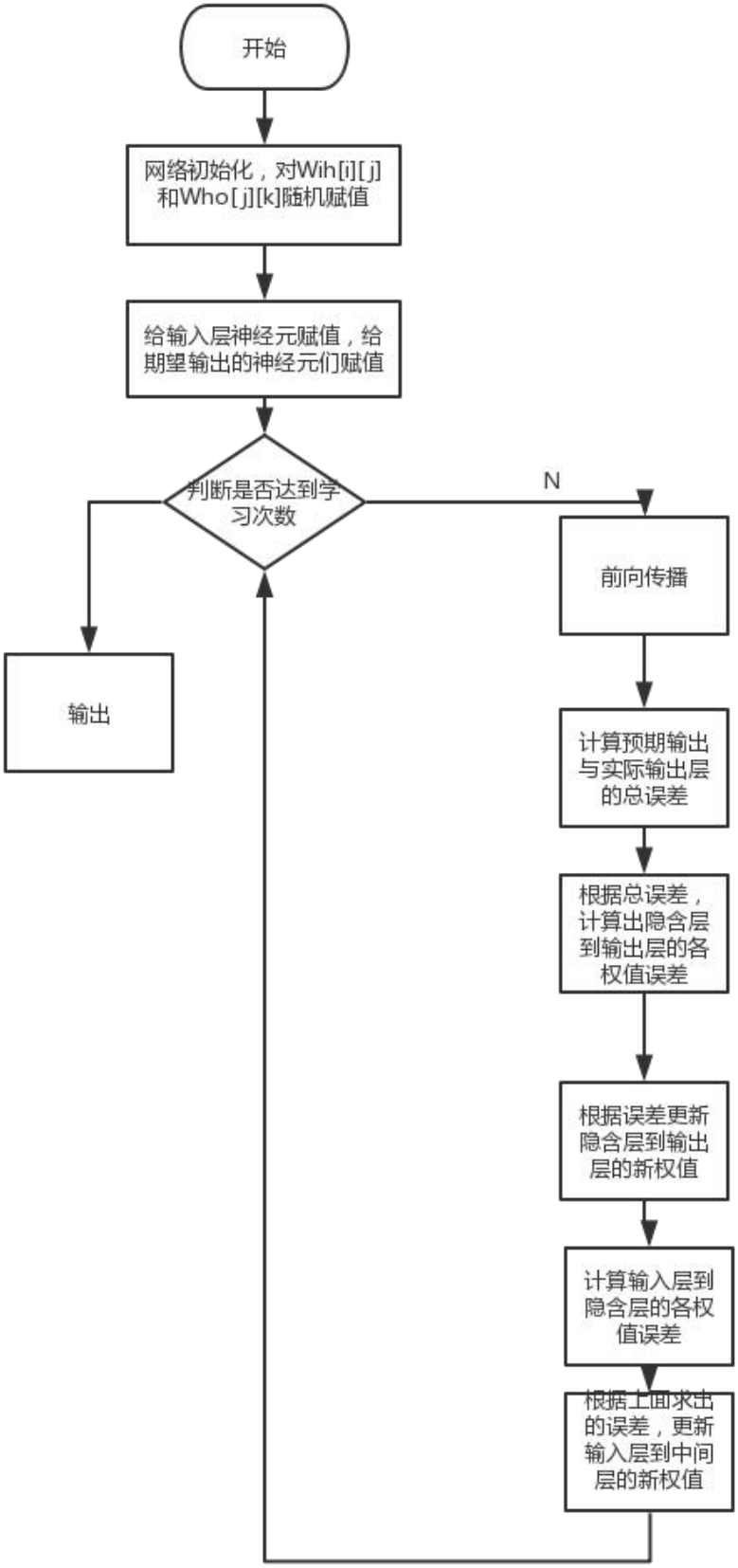
$$Wih_{ij} = Wih_{ij} - learnrate * \frac{\partial Etotal}{\partial Wih_{ij}}$$

然后继续正向传播；

如此循环往复，直到到达学习次数或者误差减小到一定值；

3. 算法实现

3.1 流程图


开始
网络初始化，对Wih[i][j]和Who[j][k]随机赋值
给输入层神经元赋值，给期望输出的神经元们赋值
判断是否达到学习次数
N
输出
前向传播
计算预期输出与实际输出层的总误差
根据总误差，计算出隐含层到输出层的各权值误差
根据误差更新隐含层到输出层的新权值
计算输入层到隐含层的各权值误差
根据上面求出的误差，更新输入层到中间层的新权值

3.2 算法伪代码

```
Void main(){
Bpinit();//网络权值随机化
choseSample();//选择输入层神经元值和预期输出神经元值
Train();
}
Void Bpinit(){
For(i=0~N1){
      For(j=0~N2)
      Wih[i][j]=1-random();
}
For(j=0~N2)
      For(k=0~N3){
      Who[j][k]=1-random;}
}
Void choseSample(){
X[0]=……;
X[1]=……;
……
X[N1]=……; //赋值
Do[0]=……;
Do[1]=……;
……
Do[N3]=……;
}
Void Train(){
For(int i=0;i<M;i++)//最大学习次数
{
       up_calculate();
      calculateE_total();
      update_who();
      update_wih();
}
Printf("……")//输出
}
Void up_calculate(){
      For(j=0;j<N2;j++)
      {hi[j]=Wih[0][j]X[0]+Wih[1][j]X[1]+Wih[2][j]X[2]+……+Wih[N1-1][j]X[N1-1];
      ho[j]=f(hi[j]);
      }

      For(k=0;k<N3;k++)
```

```
        {yi[k]=Who[0][k]ho[0]+Who[1][k]ho[1]+Who[2][k]ho[2]+……+Who[N2-1][k]ho[N2-1];
        yo[j]=f(yi[j]);
        }
        }
calculateE_total(){
For(every Eo[i] )
{Eo[i]=0.5(Do[k]-yo[k])^2;
}
E_total=Eo[0]+Eo[1]+……+Eo[N3-1];
}
        update_who(){
        For(k=0 to N3-1) Calculate So[k];
        For(j=0 to N2-1)
          For(k=0 to N3-1)
        {Who[j][k]=who[j][k]-So[K]*ho[j];
        }
        }
        update_wih(){
        For(j=0 to N2-1) Calculate Sh[j];
        For(i=0 to N1-1)
          For(j=0 to N2-1)
        {Wih[i][j]=wih[i][j]-Sh[j]*X[i];
        }

        }
```

3.3 算法实现

本次实验采用 Java 语言编写，运行的 IDE 是 IDEA。

用到的数据结构：

```
private static final int number_intie=5;//输入层有n个神经元
private static final int number_hidtie=3;//隐含层有p个神经元
private static final int number_outie=4;//输出层有q个神经元
private double E_total=0;//总误差误差
private double E_o[]=new double[number_outie];//输出层各个神经元的分误差
private int M=100000;//最大学习次数
private double epsl=0.1;//计算精度值
private double learnrate=0.5;//学习系数0-1，控制学习的快慢
private double Wih[][]=new double[number_intie][number_hidtie];//输入层与中间层的连接权值
private double Who[][]=new double[number_hidtie][number_outie];//隐含层与输出层的连接权值
private double hidden_threshold=0.35;//隐含层阈值
private double output_threshold=0.35;//输出层神经元阈值
private double X[]=new double[number_intie];//输入向量
private double Do[]=new double[number_outie];//期望输出向量
private double hi[]=new double[number_hidtie];//隐含层输入向量
private double ho[]=new double[number_hidtie];//隐含层输出向量
private double yi[]=new double[number_outie];//输出层输入向量
private double yo[]=new double[number_outie];//输出层输出向量
private double So[]=new double[number_outie];//误差函数对输出层各神经元的偏导数
private double PiandaoE_who[][]=new double[number_hidtie][number_outie];//隐含层到输出层的权值修改值
private double PiandaoE_wih[][]=new double[number_intie][number_hidtie];//输入层到隐含层的权值修改值
```

数据的初始化：

```
public void bpNetinit(){
    //初始化权值和清零
 for(int i=0;i<number_intie;i++)
     for(int j=0;j<number_hidtie;j++){
        Wih[i][j]=1-Math.random();
     }
 for(int j=0;j<number_hidtie;j++)
     for(int k=0;k<number_outie;k++) {
         Who[j][k] = 1 - Math.random();
     }
     for(int k=0;k<number_outie;k++){
         E_o[k]=0;
     }

}
```

主函数中代码如下：

```
public static void main(String []arg)
{
    BpNet bpNet= new BpNet();
    bpNet.train();

}
```

实现的关键代码如图所示：

train()函数:

```
public void train()
{
        bpNetinit();
        choose_sample();
        //反向
 for(int i=0;i<M;i++)
    {
        up_calculate();
        calculateE_total();
        update_who();
        update_wih();
    }
    System.out.println("训练代数："+M);
    System.out.println("误差值："+E_total);
    System.out.println("输入层-隐含层权值：");
    for(int i=0;i<number_intie;i++)
    {for (int j=0;j<number_hidtie;j++)
    {System.out.print(Wih[i][j]+";");

    }
        System.out.println();
    }
    System.out.println();
    System.out.println("隐含层-输出层权值：");
```

```
        System.out.println();
        System.out.println("隐含层-输出层权值：");
        for(int i=0;i<number_hidtie;i++) {
            for (int j = 0; j < number_outie; j++) {
                System.out.print(Who[i][j] + ';');
            }
            System.out.println();
        }
            System.out.println("实际输出层数据：");
            for(int i=0;i<number_outie;i++)
                System.out.print(yo[i]+';');
        System.out.println();
             System.out.println("预期输出层数据：");
        for(int i=0;i<number_outie;i++)
            System.out.print(Do[i]+';');
    }
```

bpNetinit()函数：

```
public void bpNetinit(){
    //初始化权值和清零
 for(int i=0;i<number_intie;i++)
     for(int j=0;j<number_hidtie;j++){
       Wih[i][j]=1-Math.random();
     }
 for(int j=0;j<number_hidtie;j++)
     for(int k=0;k<number_outie;k++) {
         Who[j][k] = 1 - Math.random();
     }
    for(int k=0;k<number_outie;k++){
        E_o[k]=0;
    }

}
```

Choose_sample()函数：

```
}
//第二步,随机选取第k个输入样本及对应期望输出
public void choose_sample(){

  X[0]=0.05;
  X[1]=0.1;
  X[2]=0.3;
  X[3]=0.24;
  X[4]=0.189;
  Do[0]=0.01;
  Do[1]=0.99;
  Do[2]=0.3;
  Do[3]=0.23;

}
```

```
public void up_calculate(){
    for(int h=0;h<number_hidtie;h++)
    {
        double result_wihxi=0;
        for(int j=0;j<number_intie;j++){
            result_wihxi+=Wih[j][h]*X[j];
        }
        hi[h]=result_wihxi+hidden_threshold;
    }//隐含层S激活输出
    for(int h=0;h<number_hidtie;h++){
        ho[h]=1/(1+Math.exp(-hi[h]));
    }
    //System.out.println("ho[]:"+ho[0]+','+ho[1]);
    for(int o=0;o<number_outie;o++){

        double result_whoho=0;
        for(int h=0;h<number_hidtie;h++){
            result_whoho+=Who[h][o]*ho[h];
        }
        yi[o]=result_whoho+output_threshold;//对于这个阈值不是特别理解！！！！！！！！！！
    }
    for(int o=0;o<number_outie;o++)
    {
        yo[o]=1/(1+Math.exp(-yi[o]));
    }
}
//————————————————————以上是正向传播————————————————————
```

反向传播：

```
//————————————————————计算总误差数E_total————————————————————
public void calculateE_total()//计算误差函数
{
    double result_Eoi;
    for(int i=0;i<number_outie;i++){
        result_Eoi=0;
        result_Eoi=Do[i]-yo[i];
        result_Eoi=Math.pow(result_Eoi,2);
        result_Eoi=0.5*result_Eoi;
        E_o[i]=result_Eoi;
    }
    E_total=0;
    for(int j=0;j<number_outie;j++){
        E_total+=E_o[j];
    }
//ystem.out.println("E_total="+E_total);
}
```

```
    public void calculate_So(){
        //计算误差函数对输出层的各神经元的偏导数，这里就直接用公式了推导啥的略过哈哈哈
        //So[i]表示输出层误差
        for(int i=0;i<number_outie;i++){
            So[i]=0;
            So[i]=-(Do[i]-yo[i])*yo[i]*(1-yo[i]);
            //PiandaoE_who[i]=0;
           // PiandaoE_who[i]=Math.abs(So[i]*yo[i]);
        }
        for(int i=0;i<number_hidtie;i++)
            for(int j=0;j<number_outie;j++){
                PiandaoE_who[i][j]=0;
               // PiandaoE_who[i][j]=Math.abs(So[j]*ho[i]);
                PiandaoE_who[i][j]=So[j]*ho[i];
            }

    }



    }

//-------------------------------------------------------------------------
    public void update_who(){//更新中间层到输出层的权值
       // calculateE_total();
        calculate_So();
        for(int i=0;i<number_hidtie;i++)
            for(int j=0;j<number_outie;j++){
                Who[i][j]=Who[i][j]-learnrate*PiandaoE_who[i][j];
            }
//System.out.println("更新后Who[]: "+Who[0][0]+','+Who[0][1]+','+Who[1][0]+','+Who[1][1]);
    }

//-------------------------------------------------------------------------

188     public void calculate_Sh(){
189         //计算偏导数Sh
190         //求PiandaoE_h[];
191         double piandaoEtotal_ho[]=new double[number_hidtie];
192         double piandaoho_hi[]=new double[number_hidtie];
193         double piandaohi_wih[][]=new double[number_intie][number_hidtie];
194         double helpPianEo_yo[]=new double[number_outie];
195         double helpPianyo_yi[]=new double[number_outie];
196         double result=0;
197     for(int i=0;i<number_hidtie;i++){
198         result=0;
199         for(int j=0;j<number_outie;j++){
200          helpPianEo_yo[j]=yo[j]-Do[j];
201          helpPianyo_yi[j]=yo[j]*(1-yo[j]);
202          result=helpPianEo_yo[j]*helpPianyo_yi[j];
203         }
204         piandaoEtotal_ho[i]=result;
205     }
206     for(int i=0;i<number_hidtie;i++){
207         piandaoho_hi[i]=ho[i]*(1-ho[i]);
208     }
209     for(int i=0;i<number_intie;i++){
210         for(int j=0;j<number_hidtie;j++) {
211             piandaohi_wih[i][j] = X[i];
212         }
213     }

    //Sh[i]=piandaoEtotal_ho[j]*piandaoho_hi[j];
    for(int i=0;i<number_intie;i++)
        for(int j=0;j<number_hidtie;j++){
            PiandaoE_wih[i][j]=piandaoEtotal_ho[j]*piandaoho_hi[j]*piandaohi_wih[i][j];
        }
    }
//-----------------------------------------------------------
```

```
//-------------------------------------------------------
    private void update_wih(){
        calculate_Sh();
        for(int i=0;i<number_intie;i++)
            for(int j=0;j<number_hidtie;j++){
               Wih[i][j]=Wih[i][j]-learnrate*PiandaoE_wih[i][j];
            }
        //System.out.println("更新后Wih[]: "+Wih[0][0]+','+Wih[0][1]+','+Wih[1][0]+','+Wih[1][1]);
    //System.out.println("--------------------");
    }
//-------------------------------------------------------


}
```

3.5 数据来源

自己输的数据：

X[0]=0.05;

X[1]=0.1;

X[2]=0.3;

X[3]=0.24;

X[4]=0.189;

Do[0]=0.01;

Do[1]=0.99;

Do[2]=0.3;

Do[3]=0.23;

3.6 结果展示

```
E:\Java\jdk1.8.0_191\bin\java.exe ...
训练代数：100000
误差值：1.8348556004296757E-11
输入层-隐含层权值：
0.7164252999677998;0.8467245557386143;0.7973692663040884;
0.2692748896874349;0.49005532940922847;0.8908414204693685;
0.4407476167753489;0.16631104824146062;0.14307820006221084;
0.5821082074556669;0.29891807552636396;0.7703439519724473;
0.5348678898033777;0.23090741054040714;0.782116549722346;

隐含层-输出层权值：
56.5747957487404360.77098058578652558.2117986669049258.802940067138955
56.93931997556397661.6023692203610558.5451746920640857.91223607454658
56.2272579460924760.9158074919208558.48467221849042457.97512156443527

实际输出层数据：
59.0100042887750259.9899957217385859.359.23
预期输出层数据：
59.0159.9959.359.23
Process finished with exit code 0
```

4.总结

由输出结果查看最后实际输出层与预期输出的误差非常小，接近于 0，证明这个模型的实现效果还是不错的。但是我做的这个模型只有三层，当然做多层完全没问题，但是原理都一样，我就直接做了三层了。
BP 网络看着唬人，做起来还蛮有意思的。

参考文献

[1] (美)[M.R.柳]Michael R.Lyu 主编; 刘喜成,钟婉懿等译.软件可靠性工程手册（Handbook Of Software Reliabilty Engineering）[M].电子工业出版社，1997