神经网络作业报告

姓名：王丹
学号：2120151036

一、实验目的

1.掌握神经网络的基本原理和基本的设计步骤
2.了解神经网络在实际中的应用
3.针对简单的实际问题，能够建立神经网络控制模型

二、实验内容

1、用监督型的人工神经网络画一幅画
2、用非监督型的人工神经网络画一幅画

三、实验环境

Netbeans8.0；Ubuntu14.04

四、实验原理

1.神经网络简介

在神经网络中，对外部环境提供的模式样本进行学习训练，并能存储这种模式，则称为感知器；对外部环境有适应能力，能自动提取外部环境变化特征，则称为认知器。

神经网络在学习中，一般分为有教师和无教师学习两种。感知器采用有教师信号进行学习，而认知器则采用无教师信号学习的。在主要神经网络如 BP 网 络，Hopfield 网络，ART 网络和 Kohonen 网络中；BP 网络和 Hopfield 网络是需要教师信号才能进行学习的；而 ART 网络和 Kohonen 网络则无需教师信号就可以学习。所谓教师信号，就是在神经网络学习中由外部提供的模式样本信号。

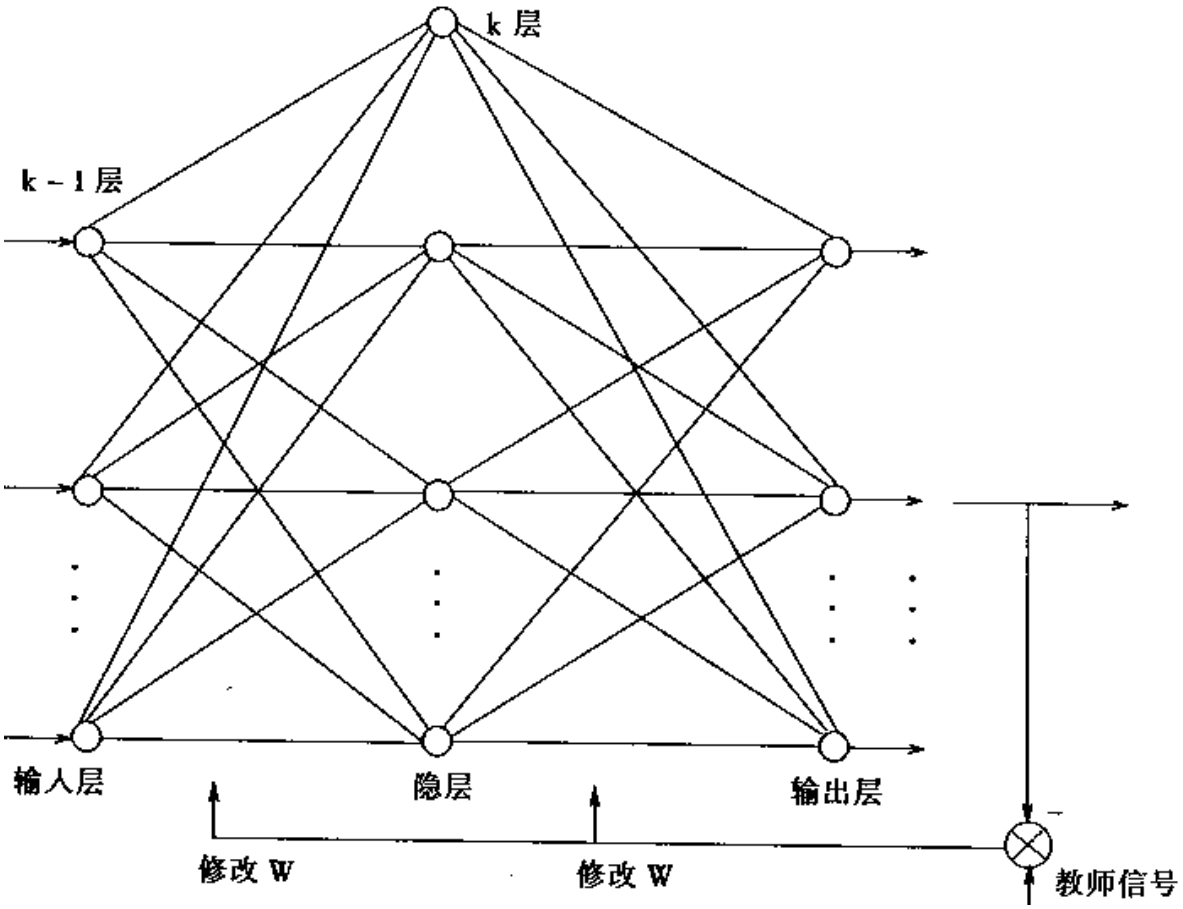
2.BP 神经网络

反向传播算法也称 BP 算法。由于这种算法在本质上是一种神经网络学习的数学模型，所以，有时也称为 BP 模型。

BP 算法是为了解决多层前向神经网络的权系数优化而提出来的；所以，BP 算法也通常暗示着神经网络的拓扑结构是一种无反馈的多层前向网络。故而．有时也称无反馈多层前向网络为 BP 模型。

在这里，并不要求过于严格去争论和区分算法和模型两者的有关异同。感知机学习算法是一种单层网络的学习算法。在多层网络中．它只能改变最后权系数。 因此， 感知机学习算法不能用于多层神经网络的学习。1986 年，Rumelhart 提出了反向传播学习算法，即 BP(backpropagation)算法。这 种算法可以对网络中各层的权系数进行修正，故适用于多层网络的学习。BP 算法是目前最广泛用的神经网络学习算法之一，在自动控制中是最有用的学习算法。

BP 算法是用于前馈多层网络的学习算法，前馈多层网络的结构一般如图下图所示：



它含有输人层、输出层以及处于输入输出层之间的中间层。中间层有单层或多层，由于它们和外界没有直接的联系，故也称为隐层。在隐层中的神经元也称隐 单元。 隐层虽然和外界不连接．但是，它们的状态则影响输入输出之间的关系。这也是说，改变隐层的权系数，可以改变整个多层神经网络的性能。

设有一个 m 层的神经网络，并在输入层加有样本 X；设第 k 层的 i 神经元的输入总和表示为 U_i^k，输出 X_i^k；从第 k—1 层的第 j 个神经元到第 k 层的第 i 个神经元的权系数为 W_{ij} 各个神经元的激发函数为 f，则各个变量的关系可用下面有关数学式表示：

$$X_i^k=f(U_i^k)$$

$$U_i^k=\sum_j W_{ij}X_j^{k-1}$$

反向传播算法分二步进行，即正向传播和反向传播。这两个过程的工作简述如下。

1)正向传播

输入的样本从输入层经过隐单元一层一层进行处理，通过所有的隐层之后，则传向输出层；在逐层处理的过程中，每一层神经元的状态只对下一层神经元的状态产生影响。在输出层把现行输出和期望输出进行比较，如果现行输出不等于期望输出，则进入反向传播过程。

2)反向传播

反向传播时，把误差信号按原来正向传播的通路反向传回，并对每个隐层的各个神经元

的权系数进行修改，以望误差信号趋向最小。

五、实验步骤

1.非监督神经网络

(1)使用 public BufferedImage createWeight() {……}生成由杂乱像素点构成的权重图。图像大小为 200*200。

(2)使用 public BufferedImage setSamples() {……}生成样本图。图像大小为 10*10。

(3)对于样本图中的每一个像素，在权重图中进行迭代。每一次迭代找出图像中距离每个样本点最接近的像素点，将他的颜色朝着样本像素点的颜色方向改变，同时将图片中该点邻域内的点也朝着该样本颜色的方向改变。最终达到颜色汇聚的效果。

```
public void update() {
    System.out.println("开始：");
    int nearestX, nearestY;
    Point a = new Point();
    int rgb;
    for (int i = 0; i < sampleSize; i++) {
        for (int j = 0; j < sampleSize; j++) {
            rgb = sampleImage.getRGB(i, j);
            //寻找 image0 中的最近点
            a = findNearestPoint(rgb);
            nearestX = a.x;
            nearestY = a.y;
            //更新邻居点颜色
            updateColor(i, j, nearestX, nearestY, rgb);
        }
    }
}
```

(4)像素点的更新公式如下：

```
newR = (int) (R1 + (R - R1) * rate) % 255;
newG = (int) (G1 + (G - G1) * rate) % 255;
newB = (int) (B1 + (B - B1) * rate) % 255;
rgb0 = ((newR * 256) + newG) * 256 + newB;
```

R1、G1、B1 是当前邻居点的颜色，R、G、B 是样本点的颜色，rate 是学习速率，定义如下：

```
rate = setRate(i, j, nearestX, nearestY);
public double setRate(double x, double y, double u1, double u2) {
    double res = 0.0;
    res = 1 - (x + y) / (u1 + u2);
    return res;
}
```

(u1,u2)是与样本点最接近的像素点，称之为获胜点，(x,y)是获胜点的邻居(由半径为 140 的园确定)。

以下方法见附录 UnsuperNN 代码：

```
public BufferedImage createWeight() ；
public BufferedImage setSamples()
public Point findNearestPoint(int rgb) ；
public void updateColor(int a, int b, int x, int y, int rgb) ；
```

2.BP 神经网络

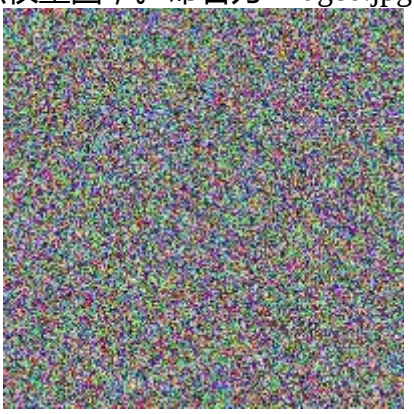
实验通过 java 的神经网络工具包 neuroph，建立了一个 4 层的前馈神经网络，其中一个输入层，节点为 2 个表示点的坐标 x,y，隐含层两层每层 5 个节点，输出层 1 个节点表示输出的颜色。

六、实验结果

1.非监督神经网络

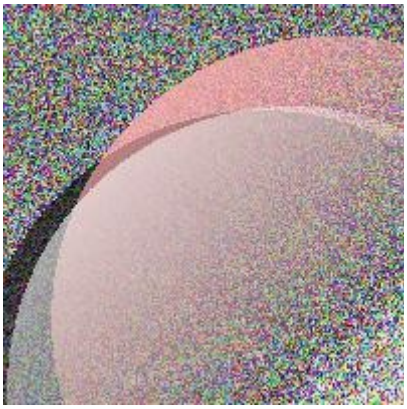
图像保存在 unnresult 文件夹下。

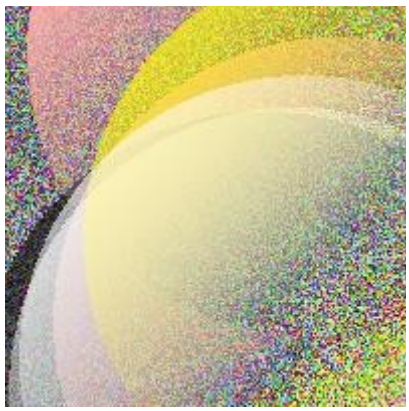
(1)随机生成的杂乱的像素点权重图，。命名为 image0.jpg，大小为 200*200。

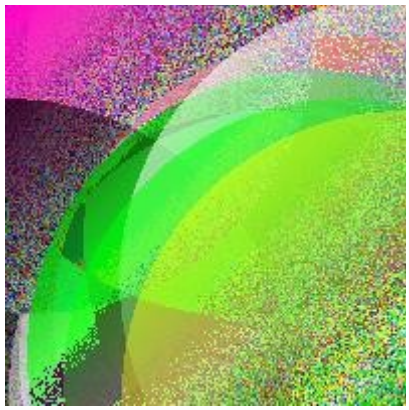


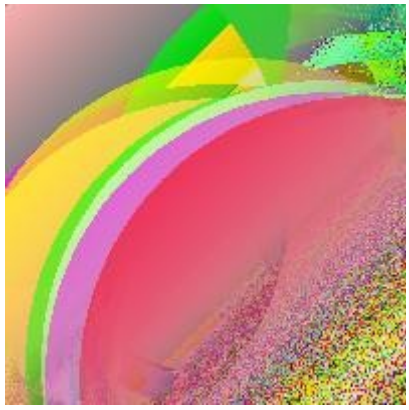
(2)样本颜色图，由 Color.WHITE,Color.BLACK,Color.BLUE,Color.MAGENTA,Color.PINK,Color.ORANGE,Color.YELLOW,Color.PINK,Color.white,Color.RED 十种颜色构成。命名为 Data.jpg，大小为 10*10。

(3) 每一次迭代找出图像中距离每个样本点最接近的像素点，将他的颜色朝着样本像素点的颜色方向改变，同时将图片中该点邻域内的点也朝着该样本颜色的方向改变。最终达到颜色汇聚的效果。中间图像如下：









2.BP 神经网络

图像保存在 bpresult 文件夹下。

通过 BP 算法对输入的图像进行预测，得到如下中间图像：

七、附录

UnsuperNN.java

```
public class UnsuperNN {

    private Images image;
    private BufferedImage sampleImage;
    private BufferedImage image0;
    private int imageSize;
    private int sampleSize;

    public static void main(String[] args) {
        UnsuperNN unn = new UnsuperNN();
        unn.InitImages();
        unn.update();

    }

    public void InitImages() {
        image = new Images();
        image0 = image.createWeight();
        sampleImage = image.setSamples();
        imageSize = image.getImageSize();
        sampleSize = image.getSampleSize();
        System.out.println("准备完毕");
    }

    //更新
    public void update() {
        System.out.println("开始：");
        int nearestX, nearestY;
        Point a = new Point();
        int rgb;
        for (int i = 0; i < sampleSize; i++) {
            for (int j = 0; j < sampleSize; j++) {
                rgb = sampleImage.getRGB(i, j);
                //寻找 image0 中的最近点
                a = findNearestPoint(rgb);
                nearestX = a.x;
                nearestY = a.y;
                //处理像素
                updateColor(i, j, nearestX, nearestY, rgb);
            }
        }
    }

    public Point findNearestPoint(int rgb) {
        System.out.println("开始查找最近点");
        Point a = new Point();
        int R, G, B, R1, G1, B1, absR, absG, absB, distance;
        int D = Integer.MAX_VALUE;
        R = (rgb & 0xff0000) >> 16;
        G = (rgb & 0xff00) >> 8;
        B = (rgb & 0xff);
        for (int i = 0; i < imageSize; i++) {
            for (int j = 0; j < imageSize; j++) {
                rgb = image0.getRGB(i, j);
                //颜色相似度计算找到最相似的点
                R1 = (rgb & 0xff0000) >> 16;
                G1 = (rgb & 0xff00) >> 8;
                B1 = (rgb & 0xff);
```

```
                absR = R - R1;
                absG = G - G1;
                absB = B - B1;
                distance = (int) Math.sqrt(absR * absR + absG * absG + absB * absB);
                if (distance < D) {
                    D = distance;
                    a.x = i;
                    a.y = j;
                }
            }
        }
        return a;
    }

    public void updateColor(int a, int b, int x, int y, int rgb) {
        System.out.println("开始更新像素点");
        File file = new File(UnsuperNN.class.getResource("/").getFile().toString() + "\\result\\Data" + a + b + ".jpg");
        int R, G, B, nearestX, nearestY, newR, newG, newB, R1, G1, B1;
        int rgb0;
        double rate;
        R = (rgb & 0xff0000) >> 16;
        G = (rgb & 0xff00) >> 8;
        B = (rgb & 0xff);
        nearestX = x;
        nearestY = y;
        for (int i = 0; i < imageSize; i++) {
            for (int j = 0; j < imageSize; j++) {
                //判断是否是邻居点
//              if (Math.abs(i - nearestX) + Math.abs(j - nearestY) > 80) {
//                  continue;
//              }
                if ((i - nearestX) * (i - nearestX) + (j - nearestY) * (j - nearestY) > 20000) {
                    continue;
                }
                rate = setRate(i, j, nearestX, nearestY);
                //找到邻居点 ij，更新颜色
                rgb0 = image0.getRGB(i, j);
                R1 = (rgb0 & 0xff0000) >> 16;
                G1 = (rgb0 & 0xff00) >> 8;
                B1 = (rgb0 & 0xff);
                newR = (int) (R1 + (R - R1) * rate) % 255;
                newG = (int) (G1 + (G - G1) * rate) % 255;
                newB = (int) (B1 + (B - B1) * rate) % 255;
                rgb0 = ((newR * 256) + newG) * 256 + newB;
                image0.setRGB(i, j, rgb0);
            }
        }
        try {
            ImageIO.write(image0, "jpg", file);
            System.out.println("新图更新完毕");
        } catch (Exception e) {

        }

    }

    public double setRate(double x, double y, double u1, double u2) {
        double res = 0.0;
        res = 1 - (x + y) / (u1 + u2);
        return res;
    }
```

```
class Images {

    private int red;
    private int blue;
    private int green;
    private int imageSize = 200;
    private int sampleSize = 10;
    BufferedImage image;
    private int rgb;
    Random a = new Random();

    //随机生成权重
    public BufferedImage createWeight() {
        Random a = new Random();
        File file = new File(UnsuperNN.class.getResource("/").getFile().toString() + "\\result\\image0.jpg");
        image = new BufferedImage(imageSize, imageSize, BufferedImage.TYPE_3BYTE_BGR);
        for (int i = 0; i < imageSize; i++) {
            for (int j = 0; j < imageSize; j++) {
                red = a.nextInt(255);
                blue = a.nextInt(255);
                green = a.nextInt(255);
                rgb = ((red * 256) + green) * 256 + blue;
                image.setRGB(i, j, rgb);
            }
        }
        try {
            ImageIO.write(image, "jpg", file);
            System.out.println("权重图片已生成");
        } catch (Exception e) {

        }
        return image;
    }

    //设置样本点
    public BufferedImage setSamples() {
        Color[] colors = new Color[10];
        colors[0] = Color.WHITE;
        colors[1] = Color.BLACK;
        colors[2] = Color.BLUE;
        colors[3] = Color.MAGENTA;
        colors[4] = Color.PINK;
        colors[5] = Color.ORANGE;
        colors[6] = Color.YELLOW;
        colors[7] = Color.PINK;
        colors[8] = Color.white;
        colors[9] = Color.RED;
        File file = new File(UnsuperNN.class.getResource("/").getFile().toString() + "\\result\\Data.jpg");
        image = new BufferedImage(sampleSize, sampleSize, BufferedImage.TYPE_3BYTE_BGR);
        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                int randomc = a.nextInt(10);
                red = (int) colors[randomc].getRed();
                blue = (int) colors[randomc].getGreen();
                green = (int) colors[randomc].getBlue();
                rgb = ((red * 256) + green) * 256 + blue;
                image.setRGB(i, j, rgb);
            }
        }
        try {
            ImageIO.write(image, "jpg", file);
```

```
                System.out.println("样本图片已生成");
            } catch (Exception e) {

            }
            return image;
        }

        public int getImageSize() {
            return imageSize;
        }

        public int getSampleSize() {
            return sampleSize;
        }
    }

}
```

BP.java

```
package bp;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.Vector;

import javax.imageio.ImageIO;

import org.encog.neural.data.NeuralData;
import org.encog.neural.data.NeuralDataPair;
import org.neuroph.core.NeuralNetwork;
import org.neuroph.core.learning.SupervisedTrainingElement;
import org.neuroph.core.learning.TrainingElement;
import org.neuroph.core.learning.TrainingSet;
import org.neuroph.nnet.MultiLayerPerceptron;
import org.neuroph.nnet.learning.DynamicBackPropagation;
import org.neuroph.nnet.learning.MomentumBackpropagation;
import org.neuroph.util.TransferFunctionType;



public class BP {
    public static void main(String[] args) throws Exception {
        gendata();
        ArrayList<String> content=readFile("data.dat");
        TrainingSet trainingSet = new TrainingSet(2, 1);
        for(int i=0;i<content.size();i++)
        {
                String[] t=content.get(i).split(",");
                double di=Double.parseDouble(t[0]);
                double dj=Double.parseDouble(t[1]);
                double r=Double.parseDouble(t[2]);
        trainingSet.addElement(new SupervisedTrainingElement(new double[]{di, dj}, new double[]{r}));
        }
```

```
    MultiLayerPerceptron network = new MultiLayerPerceptron(TransferFunctionType.TANH, 2,5,5,1);

    DynamicBackPropagation train = new DynamicBackPropagation();
    train.setNeuralNetwork(network);
    network.setLearningRule(train);
    drawpic(trainingSet,0,network);
    int epoch = 1;
    do
    {
            train.doOneLearningIteration(trainingSet);

            System.out.println("Epoch " + epoch + ", error=" + train.getTotalNetworkError());
            epoch++;
            if(epoch%100==0)
                    drawpic(trainingSet,epoch,network);

    } while(train.getTotalNetworkError()>0.1);
    drawpic(trainingSet,epoch,network);

    System.out.println("Neural Network Results:");


    for(TrainingElement element : trainingSet.trainingElements()) {
            network.setInput(element.getInput());
           network.calculate();
           Vector<Double> output = network.getOutput();
           SupervisedTrainingElement ste = (SupervisedTrainingElement)element;

                    System.out.println(element.getInput().get(0) + "," + element.getInput().get(1)
                                    + ", actual=" + output.get(0) + ",ideal=" + ste.getDesiredOutput().get(0));
            }


}

public static void drawpic(TrainingSet trainset,int m,MultiLayerPerceptron network) throws Exception
    {
             int width = 400;
            int height = 600;
              File file = new File("pic\\3image"+m+".jpg");
            Font font = new Font("Serif", Font.BOLD, 0);
            BufferedImage bi = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
            Graphics2D g2 = (Graphics2D)bi.getGraphics();
            g2.setBackground(Color.WHITE);
            g2.clearRect(0, 0, width, height);
            MultiLayerPerceptron network1 =network ;
            for(TrainingElement element : trainset.trainingElements()) {
               network1.setInput(element.getInput());
                 network1.calculate();
                   Vector<Double> output = network1.getOutput();
               SupervisedTrainingElement ste = (SupervisedTrainingElement)element;
               if(Math.abs( network1.getOutput().get(0)-1)<Math.abs(network1.getOutput().get(0)-0))
               {
                    Color c=Color.black;
                    g2.setPaint(c);
                    double i=(element.getInput().get(0));
                    int ii=(int)(i);
                    double j=(element.getInput().get(1));
                    int jj=(int)(j);
                    if(m==0)
                    System.out.println(ii+","+jj);
                          g2.fillRect(jj*100+50, ii*100+50, 100, 100);
               }
                    }
            ImageIO.write(bi, "jpg", file);
```

```
    }

    public static ArrayList<String> readFile (String path) throws Exception
    {
        ArrayList<String> result = new ArrayList<String>();

        File f = new File(path);
        FileInputStream fis = new FileInputStream(f);
        InputStreamReader isr = new InputStreamReader(fis,"UTF-8");
        BufferedReader br = new BufferedReader(isr);
        String line = "";
        while((line=br.readLine())!=null&&line.trim().length()>0)
        {
            line = line.trim();
            result.add(line);
        }

        br.close();
        isr.close();
        fis.close();
        return result;
    }

    public static void gendata() throws Exception
    {
        double[][] data=new double[][]{{1.0,1.0,1.0},{0.0,0.0,1.0},{1.0,1.0,1.0},{0.0,0.0,1.0},{1.0,1.0,1.0}};
        StringBuffer buf=new StringBuffer();
        for(int i=0;i<data.length;i++)
            for(int j=0;j<data[0].length;j++)
                buf.append(i+","+j+","+data[i][j]+"\n");
        writeFile("data.dat", buf.toString());

    }

    public static void writeFile(String path, String content) throws Exception
    {
        File f = new File(path);
        FileOutputStream fos = new FileOutputStream(f);
        OutputStreamWriter osw = new OutputStreamWriter(fos,"UTF-8");
        BufferedWriter bw = new BufferedWriter(osw);
        bw.write(content);
        bw.close();
        osw.close();
        fos.close();
    }
}
```