第二章　感知器

感知器是由美国计算机科学家罗森布拉特于 1957 年提出的。感知器可谓是最早的人工神经网络。单层感知器是一个具有一层神经元、采用阈值激活函数的前向网络。通过对网络权值的训练，可以使感知器对一组输入矢量的响应达到元素为 0 或 1 的目标输出。从而实现对输入矢量分类的目的。图 2.1 给出了单层感知器神经元模型图。

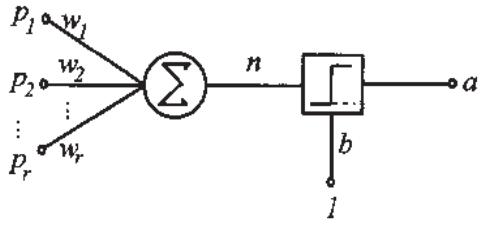


图 2.1　感知器神经元模型

其中，每一个输入分量 $p_j(j=1,2,...,r)$ 通过一个权值分量 w_j 进行加权求和，并作为阈值函数的输入。偏差 b 的加入使得网络多了一个可调参数，为使网络输出达到期望的目标矢量提供了方便。感知器特别适合解决简单的模式分类问题。

感知器实际上是在 MP 模型的基础上加上学习功能，使其权值可以调节的产物。罗森布拉特研究了单层的以及具有一个隐含层的感知器。但在当时他只能证明单层感知器可以将线性可分输入矢量进行正确划分，所以本书中所说的感知器是指单层的感知器。多层网络因为要用到后面将要介绍的反向传播法进行权值修正，所以把它们均归类为反向传播网络之中。

2.1　感知器的网络结构

感知器的网络是由单层的 s 个感知神经元，通过一组权值 $\{w_{ij}\}(i=1,2,...,s;j=1,2,...,r)$ 与 r 个输入相连组成。对于具有输入矢量 $P_{r\times q}$ 和目标矢量 $T_{s\times q}$ 的感知器网络的简化结构如图 2.2 所示。

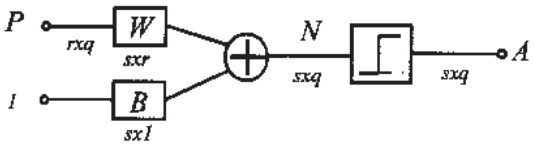


图 2.2　感知器简化结构图

根据网络结构，可以写出第 i 个输出神经元（$i=1,2,...,s$）的加权输入和 n_i 及其输出 a_i 为：

$$n_i = \sum_{j=1}^{r} w_{ij} p_j \tag{2.1}$$

$$a_i = f(n_i + b_i) \tag{2.2}$$

感知器的输出值是通过测试加权输入和值落在阀值函数的左右来进行分类的，即有：

$$a_i = \begin{cases} 1 & n_i + b_i \geqslant 0 \\ 0 & n_i + b_i < 0 \end{cases} \tag{2.3}$$

阀值激活函数如图 2.3 所示。

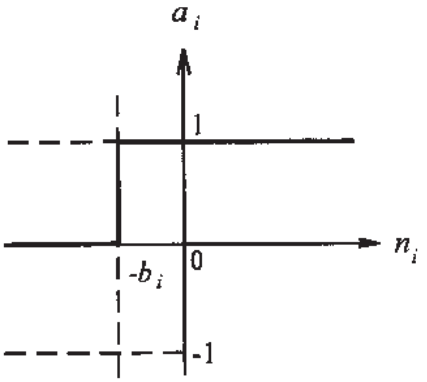


图 2.3　阀值激活函数

由图 2.3 可知：当输入 $n_i + b_i$ 大于等于 0，即有 $n_i \geqslant -b_i$ 时，感知器的输出为 1，否则输出 a_i 为 0。利用偏差 b_i 的使用，使其函数可以左右移动，从而增加了一个自由调整变量和实现网络特性的可能性。

2.2　感知器的图形解释

由感知器的网络结构，我们可以看出感知器的基本功能是将输入矢量转化成 0 或 1 的输出。这一功能可以通过在输入矢量空间里的作图来加以解释。为了简单起见，以下取 $s = 1$，即输出为一个节点的网络的情况来进行作图解释。

由感知器的输入/输出的关系式(2.3)可知，感知器的输出只有 1 或 0 两个状态，其他值由 $W^*P + b$ 的值大于、等于或小于零来确定。当网络的权值 W 和 b 确定后，在由各输入矢量 p_j ($j = 1, 2, ..., r$)为坐标轴所组成的输入矢量空间里，可以画出 $W^*P + b = 0$ 的轨迹，对于任意给定的一组输入矢量 P，当通过感知器网络的权值 W 和 b 的作用，或落在输入空间 $W^*P + b = 0$ 的轨迹上，或落在 $W^*P + b = 0$ 轨迹的上部或下部，而整个输入矢量空间是以 $W^*P + b = 0$ 为分割界，即不落在 $W^*P + b = 0$ 轨迹上的输入矢量，不是属于 $W^*P + b > 0$，就是使 $W^*P + b < 0$。因而感知器权值参数的设计目的，就是根据学习法则设计一条 $W^*P + b = 0$ 的轨迹，使其对输入矢量能够达到期望位置的划分。

以输入矢量 $r = 2$ 为例，对于选定的权值 w_1、w_2 和 b，可以在以 p_1 和 p_2 分别作为横、纵坐标的输入平面内画出 $W^*P + b = w_1p_1 + w_2p_2 + b = 0$ 的轨迹，它是一条直线，此直线上的及其线以上部分的所有 p_1、p_2 值均使 $w_1p_1 + w_2p_2 + b \geqslant 0$，这些点若通过由 w_1、w_2 和 b 构成的感知器则使其输出为 1；该直线以下部分的点则使感知器的输出为 0。所以当采用感知器对不同的输入矢量进行期望输出为 0 或 1 的分类时，其问题可转化为：对于已知输入矢量在输入空间形成的不同点的位置，设计感知器的权值 W 和 b，将由 $W^*P + b = 0$ 的直线放置在适当的位置上使输入矢量按期望输出值进行上下分类。

阀值函数通过将输入矢量的 r 维空间分成若干区域而使感知器具有将输入矢量分类的能力。输出矢量的 0 或 1，取决于对输入的分类。

图 2.4 给出了感知器在输入平面中的图形，从中可以清楚看出：由直线 $W*P+b=0$ 将由输入矢量 p_1 和 p_2 组成的平面分为两个区域，此线与权重矢量 W 正交可根据偏差 b 进行左右平移。直线上部的输入矢量使阈值函数的输入大于 0，所以使感知器神经元的输出为 1 。直线下部的输入矢量使感知器神经元的输出为 0 。分割线可以按照所选的权值和偏差上下左右移动到期望划分输入平面的地方。

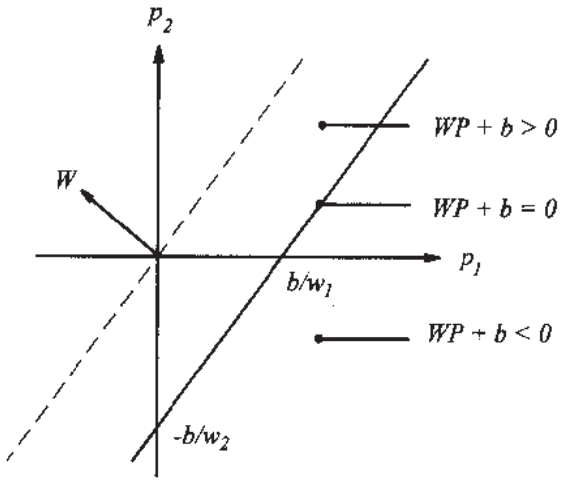


图 2.4　输入矢量平面图

感知器神经元不带偏差时，得到的是通过原点的分类线。有些可以用带偏差解决的问题，不带偏差的网络则解决不了。

熟悉图形解释有助于我们理解和掌握感知器的工作原理。当然在实际应用时，权值的求解全都是由计算机来完成的。

2.3　感知器的学习规则

学习规则是用来计算新的权值矩阵 W 及新的偏差 B 的算法。感知器利用其学习规则来调整网络的权值，以便使该网络对输入矢量的响应达到数值为 0 或 1 的目标输出。

对于输入矢量 P，输出矢量 A，目标矢量为 T 的感知器网络，感知器的学习规则是根据以下输出矢量可能出现的几种情况来进行参数调整的。

1 ）如果第 i 个神经元的输出是正确的，即有：$a_i=t_i$，那么与第 i 个神经元联接的权值 w_{ij} 和偏差值 b_i 保持不变；

2 ）如果第 i 个神经元的输出是 0，但期望输出为 1，即有 $a_i=0$，而 $t_i=1$，此时权值修正算法为：新的权值 w_{ij} 为旧的权值 w_{ij} 加上输入矢量 p_j；类似的，新的偏差 b_i 为旧偏差 b_i 加上它的输入 1；

3 ）如果第 i 个神经元的输出为 1，但期望输出为 0，即有 $a_i=1$，而 $t_i=0$，此时权值修正算法为：新的权值 w_{ij} 等于旧的权值 w_{ij} 减去输入矢量 p_j；类似的，新的偏差 b_i 为旧偏差 b_i 减去 1 。

由上面分析可以看出感知器学习规则的实质为：权值的变化量等于正负输入矢量。具体算法总结如下。

对于所有的 i 和 $j, i = 1, 2, \ldots, s$； $j = 1, 2, \ldots, r$，感知器修正权值公式为：

$$\begin{aligned} \Delta w_{ij} &= (t_i - y_i) \times p_j \\ \Delta b_i &= (t_i - y_i) \times 1 \end{aligned} \tag{2.4}$$

用矢量矩阵来表示为：

$$\begin{aligned} W &= W + EP^T \\ B &= B + E \end{aligned} \tag{2.5}$$

此处，E 为误差矢量，有 $E = T - A$ 。

感知器的学习规则属于梯度下降法，该法则已被证明：如果解存在，则算法在有限次的循环迭代后可以收敛到正确的目标矢量。

上述用来修正感知器权值的学习算法在 MATLAB 神经网络工具箱中已编成了子程序，成为一个名为 **learnp.m** 的函数。只要直接调用此函数，即可立即获得权值的修正量。此函数所需要的输入变量为：输入、输出矢量和目标矢量：P 、A 和 T 。调用命令为：

```
[ dW, dB ] = learnp ( P, A, T );
```

2.4 网络的训练

要使前向神经网络模型实现某种功能，必须对它进行训练，让它逐步学会要做的事情，并把所学到的知识记忆在网络的权值中。人工神经网络权值的确定不是通过计算，而是通过网络的自身训练来完成的。这也是人工神经网络在解决问题的方式上与其他方法的最大不同点。借助于计算机的帮助，几百次甚至上千次的网络权值的训练与调整过程能够在很短的时间内完成。

感知器的训练过程如下：

在输入矢量 P 的作用下，计算网络的实际输出 A ，并与相应的目标矢量 T 进行比较，检查 A 是否等于 T，然后用比较后的误差 E，根据学习规则进行权值和偏差的调整；重新计算网络在新权值作用下的输入，重复权值调整过程，直到网络的输出 A 等于目标矢量 T 或训练次数达到事先设置的最大值时训练结束。

若网络训练成功，那么训练后的网络在网络权值的作用下，对于被训练的每一组输入矢量都能够产生一组对应的期望输出；若在设置的最大训练次数内，网络未能够完成在给定的输入矢量 P 的作用下，使 $A = T$ 的目标，则可以通过改用新的初始权值与偏差，并采用更长训练次数进行训练，或分析一下所要解决的问题是否属于那种由于感知器本身的限制而无法解决的一类。

感知器设计训练的步骤可总结如下：

1 ）对于所要解决的问题，确定输入矢量 P ，目标矢量 T ，并由此确定各矢量的维数以及确定网络结构大小的神经元数目： r ， s 和 q ；

2 ）参数初始化：

a) 赋给权矢量 W 在(-1, 1)的随机非零初始值;

b) 给出最大训练循环次数 max_epoch ;

3) 网络表达式: 根据输入矢量 P 以及最新权矢量 W , 计算网络输出矢量 A ;

4) 检查: 检查输出矢量 A 与目标矢量 T 是否相同, 如果是, 或已达最大循环次数, 训练结束, 否则转入 5) ;

5) 学习: 根据(2.5)式感知器的学习规则调整权矢量, 并返回 3)。

下面给出例题来进一步了解感知器解决问题的方式, 掌握设计训练感知器的过程。

【例 2. 1 】考虑一个简单的分类问题。

设计一个感知器, 将二维的四组输入矢量分成两类。

输入矢量为: P = [-0.5 -0.5 0.3 0;
-0.5 0.5 -0.5 1];

目标矢量为: T = [1.0 1.0 0 0],

解:

通过前面对感知器图解的分析可知, 感知器对输入矢量的分类实质是在输入矢量空间用 $W*P + B = 0$ 的分割界对输入矢量进行切割而达到分类的目的。根据这个原理, 对此例中二维四组输入矢量的分类问题, 可以用下述不等式组来等价表示出:

$$\begin{cases} -0.5w_1 - 0.5w_2 + w_3 \geqslant 0 & (\text{使 } t_1 = 1 \text{ 成立}) \\ -0.5w_1 + 0.5w_2 + w_3 \geqslant 0 & (\text{使 } t_2 = 1 \text{ 成立}) \\ 0.3w_1 - 0.5w_2 + w_3 < 0 & (\text{使 } t_3 = 0 \text{ 成立}) \\ w_2 + w_3 < 0 & (\text{使 } t_4 = 0 \text{ 成立}) \end{cases}$$

实际上可以用代数求解法来求出上面不等式中的参数 w_1 、 w_2 和 w_3。 经过迭代和约简, 可得到解的范围为:

$$\begin{cases} w_1 < 0 \\ 0.8w_1 < w_2 < -w_1 \\ w_1/3 < w_3 < -w_1 \\ w_3 < -w_2 \end{cases}$$

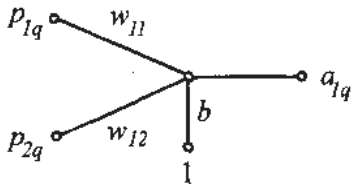
一组可能解为:

$$\begin{cases} w_1 = -1 \\ w_2 = 0 \\ w_3 = -0.1 \end{cases}$$

而当采用感知器神经网络来对此题进行求解时, 意味着采用具有阀值激活函数的神经网络, 按照问题的要求设计网络的模型结构, 通过训练网络权值 $W = [w_{11}\ w_{12}]$和 b , 并根据学习算法和训练过程进行程序编程, 然后运行程序, 让网络自行训练其权矢量, 直至达到不等式组的要求。

鉴于输入和输出目标矢量已由问题本身确定, 所以所需实现其分类功能的感知器网络结构的输入节点 r , 以及输出节点数 s 已被问题所确定而不能任意设置。

根据题意, 网络结构图如图 2.5 所示。



$q = 1, 2, 3, 4$

图 2.5　网络结构图

由此可见，对于单层网络，网络的输入神经元数 r 和输出神经元数 s 分别由输入矢量 P 和目标矢量 T 唯一确定。网络的权矩阵的维数为：$W_{s\times r}$，$B_{s\times 1}$ 权值总数为 $s \times r$ 个，偏差个数为 s 个。

在确定了网络结构并设置了最大循环次数和赋予权值初始值后，设计者可方便地利用 MATLAB，根据题意以及感知器的学习、训练过程来编写自己的程序。下面是对【例 2.1】所编写的网络权值训练用的 MATLAB 程序：

```
% percep1.m
%
P = [ -0.5 -0.5 0.3 0; -0.5 0.5 -0.5 1 ];          T = [ 1 1 0 0 ];
% 初始化
[ R, Q ] = size (P);          [S, Q ] =size (T);
W = rands ( S, R );          B = rands ( S, 1 );
max_epoch = 20;
% 表达式
A = hardlim ( W*P, B );                    % 求网络输出
for epoch = 1 : max_epoch                  % 开始循环训练、修正权值过程
% 检查
  if all ( A == T )                        % 当 A = T 时结束
    epoch = epoch - 1;
    break
  end
% 学习
[ dW, dB ] = learnp ( P, A, T );           % 感知器学习公式
W = W + dW;
B = B + dB;
A = hardlim ( W*P, B );                    % 计算权值修正后的网络输出
end                                        % 程序结束
```

以上就是根据前面所阐述的感知器训练的三个步骤：表达式、检查和学习而编写的 MATLAB 网络设计的程序。

对于本例也可以在二维平面坐标中给出求解过程的图形表示。图 2.6 给出了横轴为 p_1，纵轴为 p_2 的输入矢量平面，以及输入矢量 P 所处的位置。根据目标矢量将期望为 1 输出的输入分量用“+”表示出，而目标为 0 输出的输入分量用“o”表示出。

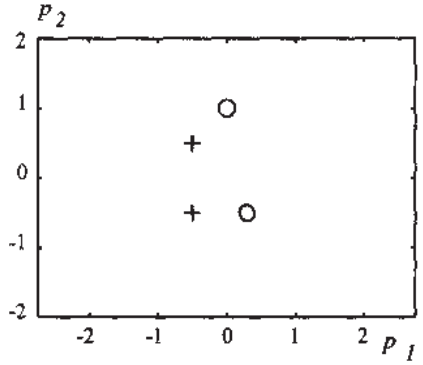


图 2.6 输入矢量位置图

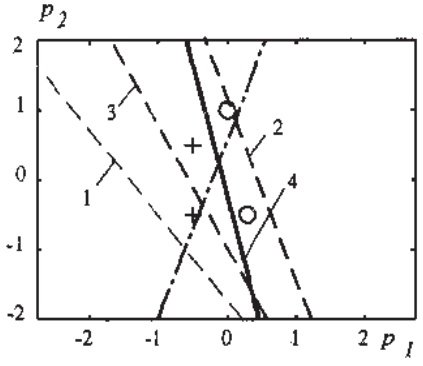


图 2.7 感知器训练过程记录

图 2.7 给出 4 次训练过程中由权值组成的 $W*P + b = 0$ 直线和最终的训练达到目标后的直线，其中，点划线为初始权值形成的直线；虚线为前三次权值修正后划分的结果；显然，“o”和“+”部分没有被合适的分割开。实线是第 4 次权值修正后达到目标的结果。

本例题中网络初始随机权值为：

W0 = [-0.8161　0.3078];　　B0 = [-0.1680];

4 次训练次后的权值为：

W = [-2.6161 -0.6922];　B = -0.1680 ;

只要用下列命令即可得到对网络结果的验证：

```
》 A = hardlim ( W*P, B )
A =
    1 1 0 0
```

图 2.8 给出网络训练过程中的权值和偏差变化记录，其中，横轴变量为训练次数 epoch 。

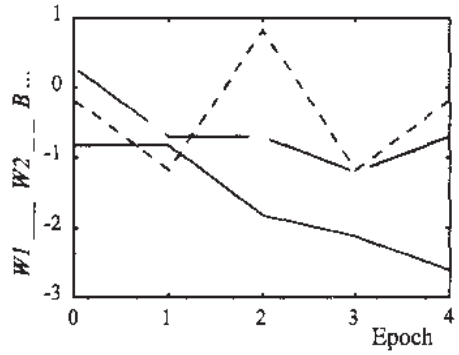


图 2.8　权值和偏差变化记录

感知器对于不同的初始条件可能得到不同的结果，对【例 2.1 】用不同的初始条件重新运行程序进行训练，同样可以解决问题，但得到不同于前一次的权值，例如对于下列权矢量和偏差值：

W = [-2.1642 -0.0744]

B = -0.6433

也是网络的一个解，但在输入平面中对输入矢量进行划分的直线 $W*P+B=0$ 的位置与前面的是不同的。

【例 2.2】多个神经元分类，又称模式联想。

若将【例 2.1】的输入矢量和目标矢量增加为 10 组的二元矩阵，即输入矢量为:

```
P = [ 0.1   0.7   0.8   0.8   1.0   0.3   0.0   -0.3   -0.5   -1.5;
      1.2   1.8   1.6   0.6   0.8   0.5   0.2   0.8    -1.5   -1.3 ];
```

所对应的 10 组二元目标矢量为:

```
T = [ 1   1   1   0   0   1   1   1   0   0;
      0   0   0   0   0   1   1   1   1   1 ];
```

由于增加了矢量数组，必然增加了解决问题的复杂程度。这个问题要是用一般的方法来解决是相当困难和费时的，它需要解一个具有 6 个变量的 20 个约束不等式。而通过感知器来解此问题就显示出它的优越性。完全与【例 2.1】程序相同，只要输入新的 P 和 T，重新运行程序，即可得出训练的结果。

根据输入矢量和目标矢量可得: $r=2, s=2$, 由此可以画出网络结构如图 2.9 所示。

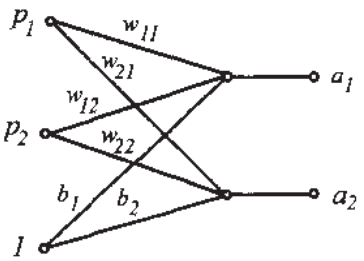


图 2.9　所设计网络结构图

本题用图解法解释为: 求感知器网络权值 W 和 B，使由 $W*P+B=0$ 构成的两条直线将输入矢量所在平面分割成四个区域。图 2.10 给出了输入矢量与不同的目标矢量所对应的不同关系，不同的目标矢量分别用四种符号表示: 00: o; 01: *; 10: +和 11: ×。

虽然【例 2.2】的求解过程复杂了许多，但用 MATLAB 来对其进行训练和设计与【例 2.1】没有什么区别。若我们再利用工具箱的特长，则显得更加简单。在神经网络工具箱中有一个集感知器训练过程中的表达式、检查和权值学习为一体的函数: **trainp.m**。通过此函数，可直接给出最终的训练结果 W 和 B 值，以及所花费的循环次数 epochs。这就允许我们腾出精力来增加一些实时监视程序或作图程序，从而使得训练过程更加生动。【例 2.2】的训练程序可编写如下:

```
% percep2.m
%
% 初始化、赋值
P = [0.1 0.7 0.8 0.8 1.0 0.3 0.0 -0.3 -0.5 -1.5; 1.2 1.8 1.6 0.6 0.8 0.5 0.2 0.8 -1.5 -1.3];
T = [ 1 1 1 0 0 1 1 1 0 0 ; 0 0 0 0 0 1 1 1 1 1 ];
[ R, Q ] = size (P);  [ S, Q ] = size (T);  [ W0, B0 ] = rands ( S, R );
disp_freq = 1;                    % 每训练一次，显示一次
max_epoch = 20;                   % 设置最大循环次数
TP = [ disp_freq   max_epoch ];   % 给 TP 赋值
% 训练网络、修正权值
[ W, B, epochs ] = trainp ( W0, B0, P, T, TP )
```

```
% 绘制训练后的分类结果
V = [ -2 2 -2 2 ];          % 取一数组限制坐标数值大小
plotv ( P, T, V );          % 在输入矢量空间绘画输入矢量和目标矢量的位置
axis ('equal'),             % 令横坐标和纵坐标等距离长度
title ('Input Vector Graph'), % 写图标题
xlabel ('p1'),              % 写横轴标题
ylabel ('p2'),              % 写纵轴标题
hold on                     % 当前图形保护模式开
plotpc ( W0, B0,'-' );      % 绘制由 W0 和 B0 在输入平面中形成的初始分类线
plotpc ( W, B );            % 绘制由 W 和 B 在输入平面中形成的最终分类线
hold off                    % 当前图形保护模式关闭
end
```

当此程序运行后，每修正一次权值后，在工作界面中显示一次已循环的次数，并在训练完成后显示出最终权值 W 和 B 以及所花费的循环总数。另外还绘制出输入矢量在输入平面中的位置以及由训练后的 W 和 B 值组成的 $W*P+B=0$ 直线在输入平面中的分类结果。

经过 10 次训练调整神经元的权矢量，网络将输入矢量分成期望的四类如图 2.11 所示。其中虚线为初始值，实线为最后的结果．网络最终权矢量为:

```
W = [ -4.7926 5.9048;
      -3.2567 -2.6339];
B = [-0.9311; 2.9970]
```

所对应的网络初始权值为:

W0 = [-0.6926 0.6048; 0.1433 -0.9339];　　B0 = [0.0689; 0.0030];

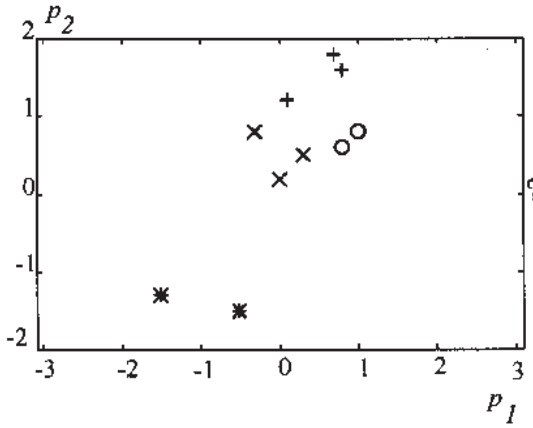


图 2.10　输入矢量目标矢量对应的关系

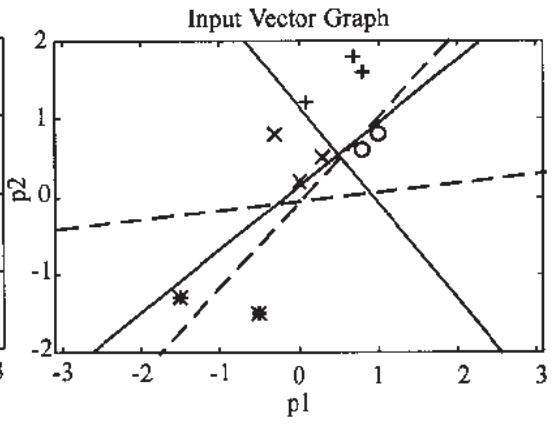


图 2.11　网络训练结果图

对于由不同的输入神经元 r 和输出神经元 s 组成的感知器，当采用输入矢量空间的作

图法来解释网络功能时，其一般情况可以总结如下：

1) 当输入为单个元素，输出也为单个神经元，即 $r=1$，$s=1$ 时，感知器是以点为分割界；

2) 当输入为二元素，即 $r=2$ 时，感知器是以线为分割界；

其中：　当 $s=1$，分割线为一条线；

　　$s=2$，分割线为两条线。

依此类推。在这里输出神经元数 s 决定分割线数，可分成的种类数为 2^s；

3) 当输入为三元素，即 $r=3$ 时，感知器是以面为分割界，而且输出神经元数 s 为分割面数。

2.5　感知器的局限性

由于感知器自身结构的限制，使其应用被限制在一定的范围内。所以在采用感知器解决具体问题时，必须时刻考虑到其特点。一般来说，感知器有以下局限性：

1) 由于感知器的激活函数采用的是阀值函数，输出矢量只能取 0 或 1，所以只能用它来解决简单的分类问题；

2) 感知器仅能够线性地将输入矢量进行分类。如果用一条直线或一个平面把一组输入矢量正确地划分为期望的类别，则称该输入/输出矢量是对线性可分的，否则为线性不可分。那么，利用感知器将永远也达不到期望输出的网络权矩阵。所以用软件设计感知器对权值进行训练时，需要设置一个最大循环次数。如果在达到该最大循环次数后，还没有达到期望的目标，训练则停止，以便不使不可分的矢量占用无限循环的训练时间。不过应当提醒的是，理论上已经证明，只要输入矢量是线性可分的，感知器在有限的时间内总能达到目标矢量；

3) 感知器还有另外一个问题，当输入矢量中有一个数比其他数都大或小得很多时，可能导致较慢的收敛速度。比如当输入/输出矢量分别为：

```
P = [ -0.5 -0.5 +0.3 -0.1 -80;
      -0.5 +0.5 -0.5 +1.0 100];
T = [ 1    1    0    0    1];
```

由于输入第五组数远远大于其他输入数组，这必然导致训练的困难。

下面给出感知器面对线性不可分输入/输出模式时的情况。

【例 2.3】线性不可分输入矢量。

由于感知器对输入矢量空间只能线性地进行输出 0 或 1 的分类，它们只能适当地划分具有线性可分的输入矢量组。如果在输入矢量组和它所对应的 0，1 目标之间不能用直线进行划分，那么感知器就不能正确地分类出输入矢量。

在【例 2.1】中，加入一个新的输入矢量，使之成为：

```
输入矢量为：P = [ -0.5   -0.5   0.3   0   -0.8;
                  0.5    0.5   -0.5   1    0 ];
```

目标矢量为: T = [1.0　1.0　0　0　0];

可以重复使用前面的程序。为了能够让程序自己判断出所做训练是否有效，可以在程序中增加判断显示程序。这只需要在初始化阶段加入函数 **flops.m** ，然后在程序最后加上:

```
fprintf ('\n Final Network Values : \n' )
W
B
fprintf ('Trained for %.0f epochs. \n', epoch )
fprintf ('Training took %.0f flops. \n', flops )
fprintf ('Nwtwork classifies :' );
if all ( hardlim ( W*P, B ) == T )
   disp ('Correctly.' )
else
   disp ('Incorrectly.' )
end
```

图 2.12 给出了要分类的数据组的初始值和训练 40 后的结果。从中可以看出，网络没有能够解决问题。图中实线的结果为:

W = [-2.1912　-1.2975], B = -1.4214

对应网络训练的初始权值为:

W0 = [0.9088 0.7025];　　B0 = [-0.4214]

检验这组由训练权矢量所构成的感知器的输出分类功能，用命令:

```
》 A = hardlim ( W*P, B )
A =
   1 0 0 0 1
```

而 T = [1 1 0 0 0]。很显然可以看到在第二和第五个输入矢量上为不正确的分类。

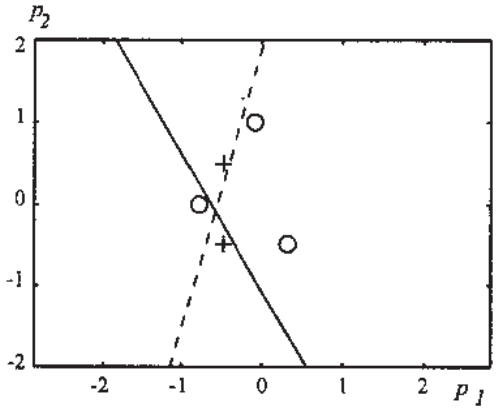


图 2.12　线性不可分模式训练结果

在感知器的应用中，这类线性不可分问题都可归结为“异或”问题。下面就来具体讨论一下这个问题。

2.6 “异或”问题

明斯基等人通过对单层感知器的研究得出结论：单层感知器存在着局限性。他的主要论据之一就是感知器不能实现简单的“异或”逻辑功能。

逻辑运算中的“异或”功能，就是当输入两个二进制数（0或1）均为0或1时，输出为0，只有当两个输入中有一个为1时，输出为1。若希望用感知器来实现此功能的操作，其输入应为两个二进制数的四种组合，目标输出 T 为与输入对应的上述四种输出的组合，即

$$P = [0\ 0\ 1\ 1;\ 0\ 1\ 0\ 1];$$

$$T = [0\ 1\ 1\ 0];$$

所要求解的是：设计一个单层感知器对输入矢量按期望的输出矢量进行分类。

我们已经知道，当输入为二元素时，其分割界为直线，可以用来进行划分的直线是数目与感知器输出神经元的数相等。“异或”问题则是要求用一条直线将平面上的四个点分成两类。而此四点在输入矢量平面上的位置如图2.13所示，其中，“×”表示希望将其点分为1类；“o”表示目标输出为0类。

很显然，对于这样的四点位置，想用一条直线把同类期望输入区分开是不可能的。实际上，如果我们把工作做得再详细点，从逻辑运算入手，即列出感知器对所有的四组二元素输入的可能输出的情况，一共可得16种情况，可以分别代表“与”、“或”、“非”、“与非”、“或非”、“异或”、“异或非”等逻辑功能。用一条直线，将平面分成两部分，分别对16种情况进行划分。结果是，16种功能中只有“异或”和“异或非”是线性不可分的，其他14种逻辑功能均为线性可分，它们都可以用单层感知器来实现。

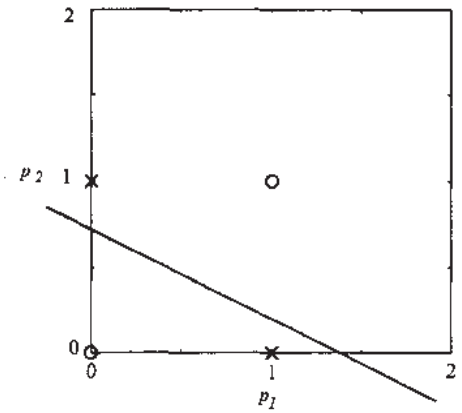


图2.13 “异或”问题的图形表示

输入矢量:

$$P=[0\ 0\ 1\ 1;$$
$$0\ 1\ 0\ 1];$$

16 种目标矢量:

$$T1=[0\ 0\ 0\ 0]$$
$$T2=[0\ 0\ 0\ 1]$$
$$T3=[0\ 0\ 1\ 0]$$
$$\cdots$$
$$T6=[0\ 1\ 0\ 1]$$
$$T7=[0\ 1\ 1\ 0] \quad \text{— 异或}$$
$$\cdots$$
$$T9=[1\ 0\ 0\ 0]$$
$$T10=[1\ 0\ 0\ 1] \quad \text{— 异或非}$$
$$\cdots$$
$$T16=[1\ 1\ 1\ 1]$$

我们不禁要问: 在有限的逻辑运算中，哪些是线性可分的呢?

我们再来考察一下最简单的单个元素输入感知器时可能有的所有逻辑功能。此时，输入有两种可能，输出有四种情况，即

输入矢量:

$$P=[0\quad 1];$$

4 种目标矢量:

$$T1=[0\ 0];$$
$$T2=[0\ 1];$$
$$T3=[1\ 0];$$
$$T4=[1\ 1];$$

因为只有一个输入，其分割界为以输入 P 为坐标线上的点，四种功能可表示为坐标轴上在 0 和 1 点上的不同功能的组合如图 2.14 所示。我们用 “ o ” 表示 0， “ x ” 表示 1 。则四种功能可表示为:

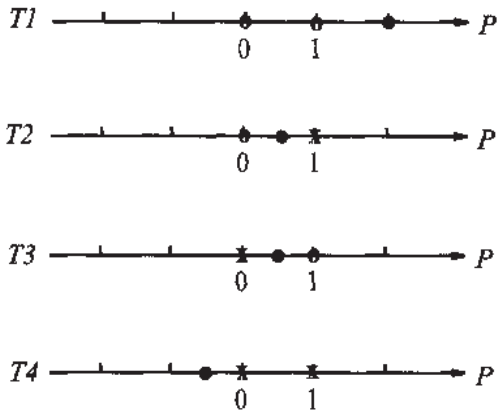


图 2.14　单输入变量时的四种分类方案

所要做的是，在 P 坐标上选一点，图中用黑点表示，将每一功能分成两类。这实际上是要分别解四组不同的分类问题。每一次分类中有一个 w 和一个 b，通过两个约束不等式来进行调节。所以这是个完全有解的线性代数问题，实际上，从图示中很明显地可以看出，只要调整 w 和 b 就很容易找到满足目标矢量的分界点。

同理，当输入具有 3 个元素时，可构成 8 种不同组合，此时的期望目标 T 可有 2 5 6 种功能。研究表明，其中只有 1 0 4 种是线性可分的。

从以上的例子可以得出结论，当网络具有 r 个二进制输入分量时，最大不重复的输入矢量有 2^r 组， 其输出矢量所能代表的逻辑功能总数为 2^{2^r} ，表 2.1 给出了不同输入 r 时，线性可分的功能数。

表 2.1 不同输入 r 时线性可分的功能数

r	2^{2^r}	线性可分功能数
1	4	4
2	16	14
3	256	104
4	65536	1882
5	4.3×10^9	94572
6	1.8×10^{19}	5028134

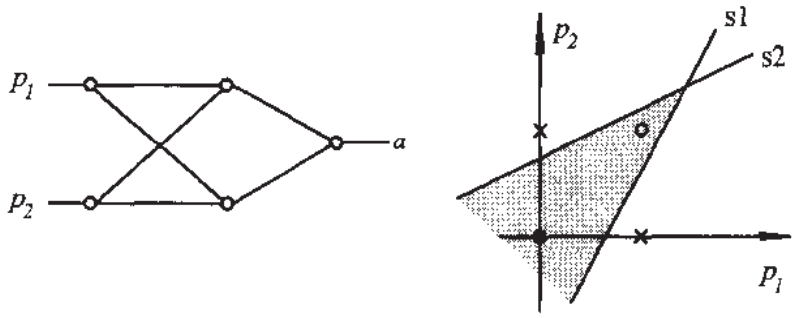
由表 2.1 可知，对于给定输入矢量所设计出的单层感知器，只对一部分输出功能是线性可分的。随着 r 的增加，线性不可分的功能数急剧增加。因此，当给定一个输入/输出矢量对时，首先必须判别该功能是否是线性可分的。遗憾的是，至今为止并没有办法来判定这种线性可分性。尤其当输入矢量增多时，更是难以确定。一般只有通过用一定的循环次数对网络进行训练而判定它是否能被线性可分。所以，单层神经元感知器只能用于简单的分类问题。

2.7 解决线性可分性限制的办法

感知器的线性可分性限制是个严重的问题。6 0 年代末，人们曾致力于该问题的研究，并找到了解决问题的办法，即变单层网络结构为多层网络结构。这实际上是把感知器的概念扩展化了。这样对于“异或”问题，我们可以用两层网络结构，并在隐含层中采用两个神经元，即用两条判决直线 s1 和 s2 来解决，如图 2.15 所示。

使 s1 线下部分为 1，线上部分为 0，而 s2 线的上部为 1，下部为 0，而在输出层中使图中阴影部分为 0，即可使“异或”功能得以实现，而且其实现的可能有许多种。研究表明，两层的阈值网络可以实现任意的二值逻辑函数，且输入值不仅限于二进制数，可以是连续数值。

对于多层感知器的权值训练与学习则需要用到误差的反向传播法，即后面将要学习的 BP 算法。本书中将具有隐含层的网络归为反向传播网络类而不再称为多层感知器。



(a) 网络结构图　　(b) 输入矢量平面图形

图 2.15 “异或”问题的一种解决方案

2.8 本章小结

1）感知器在分类问题上是有效的。它可以很好的划分线性可分的输入矢量，感知器网络的设计完全由需要解决的问题所限制。具有单层的阈值神经网络，其输入节点和神经元数是由输入矢量和输出矢量所确定的;

2）训练时间对突变矢量是敏感的。但突变输入矢量不妨碍网络达到目标。

感知器在解决实际问题时，必须在输入矢量是线性可分时才有效，这是很难得到的情形。虽然感知器有上述局限性，但它在神经网络研究中有着重要的意义和地位。它提出了自组织自学习的思想。对能够解决的问题有一个收敛的算法，并从数学上给出了严格的证明。对这种算法性质的研究仍是至今存在的多种算法中最清楚的算法之一。因此它不仅引起了众多学者对人工神经网络研究的兴趣，推动了人工神经网络研究的发展，而且后来的许多种网络模型都是在这种指导思想下建立起来并改进推广的。

习 题

〖2.1〗利用 MATLAB 对下列具有突变矢量进行编程以观察其对收敛速度的影响。

$$P = [\,-0.5\ -0.5\ +0.3\ -0.1\ -80;$$
$$-0.5\ +0.5\ -0.5\ +1.0\ 100];$$
$$T = [\,1\quad 1\quad 0\quad 0\quad 1];$$

〖2.2〗采用单层感知器的权值训练公式，通过固定隐含层的目标输出为线性可分矢量，设计一个两层感知器来解决“异或”问题，隐含层用两个神经元。