第3章 感 知 机

CHAPTER 3

在人工智能这个大的范畴内，其学术思想基本可以分为两大派别——一类是主张模仿人类（或最起码是一些高等动物）的行为模式进行研究，例如专家系统、蚁群算法等；而另一类则是从数学的逻辑和符号系统进行严格推证，以保证方法的数学严密性，例如统计学习方法等。毫无疑问，神经网络方法的基本思想是属于仿生智能的。神经网络方法的先行者们意识到仿生智能的优越性，受到人类神经元工作模式的启发首先提出了 M-P 模型。之所以称为 M-P 模型是用来纪念两位神经网络方法的开拓者 Warren Sturgis McCulloch 和 Walter Harry Pitts，Jr.，取他们姓氏的首个英文字母组成的。在第 1 章中介绍了 M-P 模型的基本结构，在 M-P 模型中，其激活（活化）函数取为符号函数，也称为开关特性函数，即

$$y=\mathrm{sgn}\left(\sum_{i=1}^{n}w_i x_i-\theta\right) \tag{3-1}$$

式中，x_i——神经元的输入激励；

w_i——与输入激励相对应的权值；

θ——神经元的阈值；

sgn(·)——符号函数。

从这个模型中可以看出，这种模型具有以下特点：

（1）每个神经元都是一个多输入单输出的系统。这表明了单个神经元可以接收多种信息的输入（激励），在一定程度上保留了生物神经元的特点。

（2）神经元对于多个输入激励并不是等量齐观的，而是有所侧重的，这主要表现在对于各输入的权值 w_i 的不同上。对于神经元来讲，比较“重要的”的输入信息作为兴奋性输入，其权值设置得较大；而不太重要的输入信息作为抑制性输入，将其权值设置得较小，甚至为 0。在 M-P 模型中，这些权值一般都预先由人指定，而且在运行过程中不会对这些权值进行更改。

（3）神经元的输出状态有两种（根据符号函数的特点可以得出）：一类是符号函数的正向性输出，此时表明神经元被激活（兴奋），另一类是符号函数的负向性输出，表明神经元被抑制。这也可以看作是激活函数的名称由来。

(4) 神经元的激活运算中具有整合和阈值特性。在各种输入信息经过一定的加权运算后,首先要经过求和整合,然后将这些经过整合的数据信息与阈值进行比较,再交由激活函数(符号函数)进行运算,得到输出的状态。

在基本 M-P 模型的基础上,还可以衍生出一些改进型的 M-P 模型,例如带有延时特性的 M-P 模型等。M-P 模型是神经网络方法的一个萌芽性的模型,虽然它能够解决的问题非常有限(连简单非线性分类的"异或"问题都无能为力),但毕竟开启了仿生智能算法的时代。究其原因,是由于 M-P 模型中的各输入激励权值 w_i 为人为指定,而且一旦指定就不能进行调整,这大大限制了其应用的范围。为了解决这个问题,就必须让输入激励的权值能够进行调整,这样就诞生了具有一定学习功能的神经网络——感知机。

3.1 感知机的基本结构与算法基础

感知机神经网络的基本架构与 M-P 模型基本类似。但是正如前面所说,感知机进一步改进了 M-P 模型,使得新的模型结构可以调整输入激励信息的权值大小,也就是说有这样一个概念,即

M-P 模型+可调整的权值=感知机

权值调整的过程就被称为神经网络的学习过程。在学习的过程(也就是权值的调整过程)中需要遵守一定的规则进行。这种学习的规则同样根据仿生的原则确定。在生物学的细胞理论中,很多动物具有条件反射。例如在给动物喂食时先响铃,动物的条件反射就会将铃声和食物联系起来。受到这种模式的启发,感知机的学习方式就是将在同一时间被激活的神经元之间的联系强化;而如果两个神经元总是保持抑制状态,则将其之间的联系逐渐削弱。这种学习规则的思想是借鉴加拿大著名生理心理学家唐纳德·赫布(Donald Olding Hebb,1904—1985)的神经科学研究成果,因此被称为神经网络的 Hebb 学习规则。在这种学习规则下,神经网络的权值调整可以用以下的公式来表示

$$\Delta \boldsymbol{W}_i = K \times f(\boldsymbol{W}_i^{\mathrm{T}} \boldsymbol{X}) \boldsymbol{X} \tag{3-2}$$

式中,$\boldsymbol{W}_i$——各权值向量;

K——比例系数;

$f(\cdot)$——激活函数;

$\boldsymbol{X}$——外界输入向量;

$f(\boldsymbol{W}_i^{\mathrm{T}} \boldsymbol{X})$——神经元的输出(阈值为 0 的情况)。

这个公式说明各权值的调整量是与神经元的输入、输出的乘积成正比关系的。如果输入、输出都比较大,那么就说明这种输入/输出模式将在神经元中占有比较大的比重,需要将其不断固化,作为神经网络的强化行为模式;而如果输入或输出的量只要出现比较小的情况,就认为这是一种被"抑制"的模式,逐渐将其移除。另一方面,从式(3-2)也可以看出,如果输入、输出一直被强化,则会导致权值的修正量不停地增长,因此在神经网络的 Hebb 学习规则中需要设定权值的饱和值。

可以看出，Hebb 学习规则与人（或某些高等动物）的认知和行为方式基本一致。在生物界，生物体就是不断地根据实践情况对自身的行为进行调整的，在这个过程中包含有一定的统计特征意义。Hebb 学习规则根据神经元之间连接的激活水平进行权值调整，这种方法被称为相关学习或联合学习。此外，从智能体的学习方式来看，Hebb 学习属于比较典型的无监督学习模式：根据神经网络自身的输入、输出情况进行学习而不是根据外在的评价标准进行学习。

3.1.1　单层感知机的基本结构

下面针对在智能研究领域中非常典型的分类问题，介绍单层感知机的基本工作原理。以二维线性分类问题为例，需要使用单层感知机将外部输入的数据分为两类。如图 3-1 所示，在二维平面上有两类点，对这两类点进行分类只需要找到图中所示的分类线就可以了。而外界输入新的数据也只要和这条分类线进行对比就可以得出相应的分类。这个分类线也称为分类判别线（对于高维的情况则称为分类判别面）。

由于只有二维数据，因此设置两个输入项即可构建一个单层感知机，如图 3-2 所示。这样，输入的数据经加权后进行融合（求和），即 $w_1x_1+w_2x_2$，然后和阈值 θ 进行比较，则得

$$X=w_1x_1+w_2x_2-\theta \tag{3-3}$$

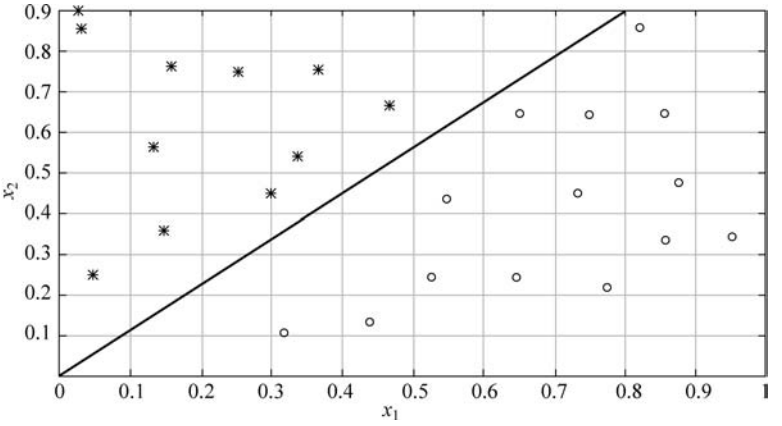


图 3-1　二维线性分类问题

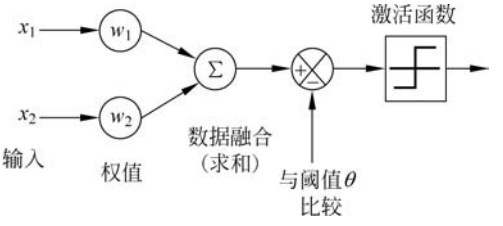


图 3-2　二维单层感知机结构

最后，代入式(3-1)的激活函数进行处理，就可以得到对外界输入数据的分类结果。可以看出，感知机在学习过程中对权值进行调整，实际上就是在对这条分类线的斜率进行调整。在调整过程中，分类线不断调整斜率以适应分类的需要，最终稳定下来，把斜率固定为某个数，从而完成对感知机的训练。权值的调整过程实际上就是学习的过程。但是从分类的任务角度来讲，这种学习实际上是一个有监督的学习过程，因此并不需要利用 Hebb 学习规则进行学习。

需要指出的是，在神经网络的调整过程中有“学习”和“训练”两种表达方法。实际上这两种表达并无本质上的区别，而且经常会有交叉使用的情况。但一般来讲，将有监督的学习过程称为“训练”，表示其外部有训练的主体；而将无监督的学习过程称为“学习”，以突出其自主进行权值调整的特点。对于上述单层感知机实现分类的算法，其训练的过程如下：

(1) 首先对权值、阈值赋初值 $w_1(0)$，$w_2(0)$，$\theta(0)$。可以任意赋值，但一般都先赋较小的正值。

(2) 输入样本对 $\{x_1, x_2; R\}$，R 为希望的分类结果。

(3) 根据激活函数计算实际的输出结果。

(4) 对比实际输出结果和希望的分类结果，对权值和阈值进行调整。

(5) 返回步骤(2)，输入新样本进行训练，直至所有的样本都分类正确为止。

在进行调整的过程中，不能进行漫无目的的试凑，而是要遵循一定的规则。例如，什么是好的分类结果呢？应该有一个标准。这就是在训练和学习中的目标函数。目标函数的选取带有某种主观性，并没有一种固定的程式和法则。但目标函数选取得适当与否又是很客观的，因为它直接关系到神经网络最后的运算结果以及在此过程中算法的复杂性。

感知机训练的目标函数通常使用损失函数。所谓损失函数是指被错误分类的数据点到分类判别线(面)的“距离”。设在分类的空间中有一点 x_0，则根据空间解析几何关系，该点到分类判别线的距离为

$$d_0 = \frac{|wx_0 + b|}{\|w\|} \tag{3-4}$$

式中，w——分类判别线(面)的斜率，即感知机的输入权值；

b——分类判别线(面)的截距，即感知机的阈值；

$\|\cdot\|$——欧几里得范数。

假设有一数据 (x_i, y_i) 被错误分类，则有

$$-y_i(wx_i + b) > 0 \tag{3-5}$$

则该错分数据点到分类判别线(面)的距离为

$$d_i = -\frac{y_i(wx_i + b)}{\|w\|} \tag{3-6}$$

这样可以得出，所有错误分类点到分类判别线(面)的距离总和为

$$D=-\frac{1}{\|w\|}\sum_{x_i\in C}y_i(wx_i+b) \tag{3-7}$$

式中，C 为所有错误分类数据组成的集合。由此可以得出感知机学习的损失函数为

$$L(\boldsymbol{W},b)=-\sum_{x_i\in C}y_i(wx_i+b) \tag{3-8}$$

式中的 $L(\cdot)$ 为损失函数，$\boldsymbol{W}$ 表示权重向量，w 表示权重的具体数值，考虑到式(3-7)中的范数对于运算没有太大关系，因此可以简化为式(3-8)。

感知机的学习过程就是要寻找合适的 $\boldsymbol{W},b$，能够使错误分类的损失函数达到最小。于是有

$$\min_{W,b}L(\boldsymbol{W},b)=\min\left\{-\sum_{x_i\in C}y_i(wx_i+b)\right\} \tag{3-9}$$

则

$$\begin{cases}\left.\dfrac{\partial L(\boldsymbol{W},b)}{\partial \boldsymbol{W}}\right|_{W=W^*}=-\displaystyle\sum_{x_i\in C}y_ix_i\\ \left.\dfrac{\partial L(\boldsymbol{W},b)}{\partial \boldsymbol{b}}\right|_{b=b^*}=-\displaystyle\sum_{x_i\in C}y_i\end{cases} \tag{3-10}$$

从理论上讲应该让式(3-10)都等于零才能达到损失最小，但在实际的运算过程中很难达到这样的要求。因此在实际的运算过程中，通常使用迭代算法使损失函数渐次达到最小值。迭代公式的一般形式为

$$h(k+1)=h(k)-\eta\nabla(h) \tag{3-11}$$

式中，$\nabla(h)$——函数 h 的梯度；

η——梯度的修正系数，称为学习率。

式(3-11)表明了在最小化损失函数的计算过程中采用迭代计算的形式，而且每次迭代均与当前状态的梯度有关。学习率的大小决定了计算寻优过程的速度和效率。η 不能够过大，这样可以为输入向量提供一个比较稳定的权值、阈值估计，否则修正的估计值将会很大，不利于网络进入稳态。当然，η 也不能太小，否则每一步修正的作用体现得不明显。

在调整学习率时可以首先进行粗调，先将权值、阈值调整到一个合适的范围，然后再进行精调，最终确定合适的权值、阈值。

将式(3-10)中权值、阈值的表达式代入式(3-11)中，就得到了权值、阈值的计算形式，即

$$\begin{cases}\boldsymbol{W}(k+1)=\boldsymbol{W}(k)+\eta y_ix_i\\ b(k+1)=b(k)+\eta y_i\end{cases} \tag{3-12}$$

显然，式(3-10)是损失函数梯度的表达形式。

这种权值、阈值学习修正方法的基本思路是要将数据的误分类情况减小到最小。除此之外，还可以使用常见的最小二乘法的思想来进行权值、阈值的学习修正，会得到相同的结果。

下面以例 3.1 来说明单层感知机在线性分类问题中的应用。

【例 3.1】 如图 3-3 所示，在二维平面上给出了 4 个点，这 4 个点分为两种类型，在图中分别以“*”和“○”表示，对这些数据点进行分类构造单层感知机，并最终确定感知机的相关参数。

解：现根据这 4 个点的坐标及分类情况建立以下模型，如表 3-1 所示。

表 3-1 例 3.1 数据集分类对应情况表

输入数据	分类情况
$\boldsymbol{x}_1=(-1,-1)^{\mathrm{T}}$	* 为 $y_1=-1$
$\boldsymbol{x}_2=(0.5,-1)^{\mathrm{T}}$	* 为 $y_2=-1$
$\boldsymbol{x}_3=(1,2)^{\mathrm{T}}$	○ 为 $y_3=1$
$\boldsymbol{x}_4=(2,1)^{\mathrm{T}}$	○ 为 $y_4=1$

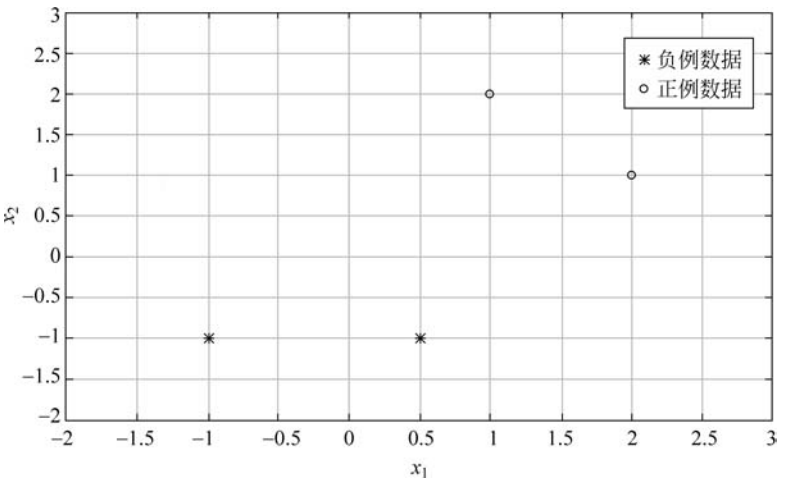


图 3-3 简单的二维线性分类问题

(1) 首先设定权值、阈值的初始值，不妨设

$$\boldsymbol{W}(0)=[1\quad 1],\quad b=[1]$$

(2) 此时可计算这 4 个输入点所对应的分类值，利用分类评价的函数进行分类。如果分类评价函数大于 0 则表示分类正确，如果小于 0 则表示分类错误。为了表达方便起见，$\boldsymbol{W}(0)$用 $\boldsymbol{W}_0$ 表示。

$$y_1(\boldsymbol{W}_0\boldsymbol{x}_1+b)=-1\left([1\quad 1]\begin{bmatrix}-1\\-1\end{bmatrix}+1\right)=1>0\rightarrow \text{正确分类}$$

$$y_2(\boldsymbol{W}_0\boldsymbol{x}_2+b)=-1\left([1\quad 1]\begin{bmatrix}0.5\\-1\end{bmatrix}+1\right)=-0.5<0\rightarrow \text{错误分类}$$

$$y_3(\boldsymbol{W}_0\boldsymbol{x}_3+b)=1\left([1\quad 1]\begin{bmatrix}1\\2\end{bmatrix}+1\right)=4>0\rightarrow \text{正确分类}$$

$$y_4(\boldsymbol{W}_0\boldsymbol{x}_4+b)=1\left([1\quad 1]\begin{bmatrix}2\\1\end{bmatrix}+1\right)=4>0\rightarrow \text{正确分类}$$

在这一步中，得到的分类线为

$$\boldsymbol{x}_1+\boldsymbol{x}_2+1=0 \tag{3-13}$$

从图 3-4 中可以明显地看到有一个错误分类，即将负例数据 $\boldsymbol{x}_2$ 错分为正例数据。于是进行权值、阈值的调整。调整方式根据式(3-12)进行，有

$$\begin{cases}\boldsymbol{W}(1)=\boldsymbol{W}(0)+1\times(-1)\times[0.5 \quad -1]=[0.5 \quad 2]\\ b(1)=b(0)+1\times(-1)=0\end{cases}$$

在此迭代计算中，学习率取 $\eta=1$。

(3) 将更新后的权值、阈值进行检验，有

$$y_1(\boldsymbol{W}_0\boldsymbol{x}_1+b)=-1\left([0.5 \quad 2]\begin{bmatrix}-1\\-1\end{bmatrix}+0\right)=2.5>0\rightarrow \text{正确分类}$$

$$y_2(\boldsymbol{W}_0\boldsymbol{x}_2+b)=-1\left([0.5 \quad 2]\begin{bmatrix}0.5\\-1\end{bmatrix}+0\right)=1.75>0\rightarrow \text{正确分类}$$

$$y_3(\boldsymbol{W}_0\boldsymbol{x}_3+b)=1\left([0.5 \quad 2]\begin{bmatrix}1\\2\end{bmatrix}+0\right)=4.5>0\rightarrow \text{正确分类}$$

$$y_4(\boldsymbol{W}_0\boldsymbol{x}_4+b)=1\left([0.5 \quad 2]\begin{bmatrix}2\\1\end{bmatrix}+0\right)=3>0\rightarrow \text{正确分类}$$

至此，经过一次迭代分类全部正确，计算终止。得到的分类直线为

$$0.25\boldsymbol{x}_1+2\boldsymbol{x}_2=0 \tag{3-14}$$

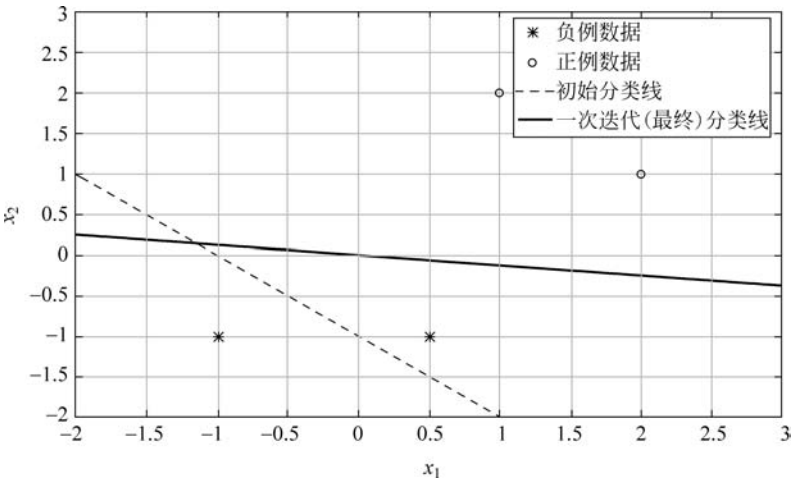


图 3-4　利用单层感知机进行简单的二维线性分类

从图 3-4 中可以看出分类线应该不止式(3-14)这一条，但是这却是一条最“好”的分类线。这是因为在整个迭代过程中，权值、阈值的调整过程遵循了式(3-10)的原则，使错误分类的损失函数达到最小。

单层感知机对于线性可分类问题处理得非常有效，不仅对于二维的情况是这样，对于

更高维的情况也是如此，只不过此时的线性分类判别线扩展为线性分类判别面（或超平面）了。一个单层感知机将空间分为两个区域，则 N 个单层感知机就可以将某模式空间分为 $2N$ 个区域。根据凸优化理论，这些区域都各自为一个凸区域。此外，这些凸区域可能会有一些共同的区域，形成交集。图 3-5 给出了 3 个单层感知机将二维空间（平面）划分区域的情况。

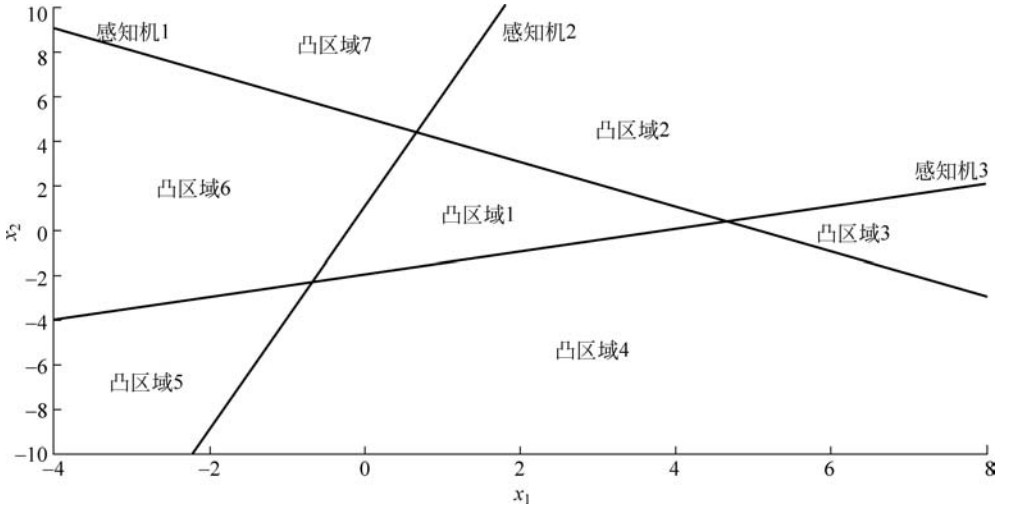


图 3-5　3 个单层感知机将二维（平面）空间划分区域的情况

从图 3-5 中可以看出，3 个单层感知机将二维空间（平面）划分为 7 个凸区域。虽然这些区域都是凸区域，但这些区域并不能全部由 3 个感知机所识别。这是因为每个单层感知机只能做有限的二分类，而不能做“且”运算。例如在凸区域 1 中的数据点，应该是小于感知机 1 的分类线，小于感知机 2 且大于感知机 3 的分类线，然而，在单层感知机的输出集合里却无法这样表达。单层感知机的输出集合一般以向量形式给出，如本例的情况，输出应该是一个三元组：(y_1　y_2　y_3)。通过判别在三元组中各个元素的 0、1 情况进行分类。因此，像凸区域 1 这种情况是无法进行有效识别的。由此可以看到，单层感知机只能进行线性可分类问题的处理。然而，在实际的应用中，线性不可分类问题也需要处理，这时就不能使用单层感知机进行了。

3.1.2　多层感知机的基本结构与算法基础

由前述可知，单层感知机对于线性不可分类问题无法进行正确的分类，需要进一步的研究。在线性不可分类问题中，最著名的例子就是“异或”问题。“异或”是一种数字逻辑运算，其基本输入/输出关系（真值表）在表 3-2 中给出。其示意图如图 3-6 所示。

从图 3-6 中看出，“异或”问题实际上是一个线性不可分类问题。也就是使用一条直线无法对这类问题进行正确分类：仅使用单层感知机，只有两个权值、一个阈值是不能对“异

或”问题进行分类的。从图 3-6 可以看出，要对“异或”问题进行正确处理需使用两条直线才行。但从图中可以看出，这两条直线所构成的分类区域并不全是凸区域，使用两个单层感知机也是不行的。在神经网络发展的初始时期，这个问题曾引起了争论。

表 3-2 “异或”问题输入/输出关系（真值表）

输入数据		输出数据
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

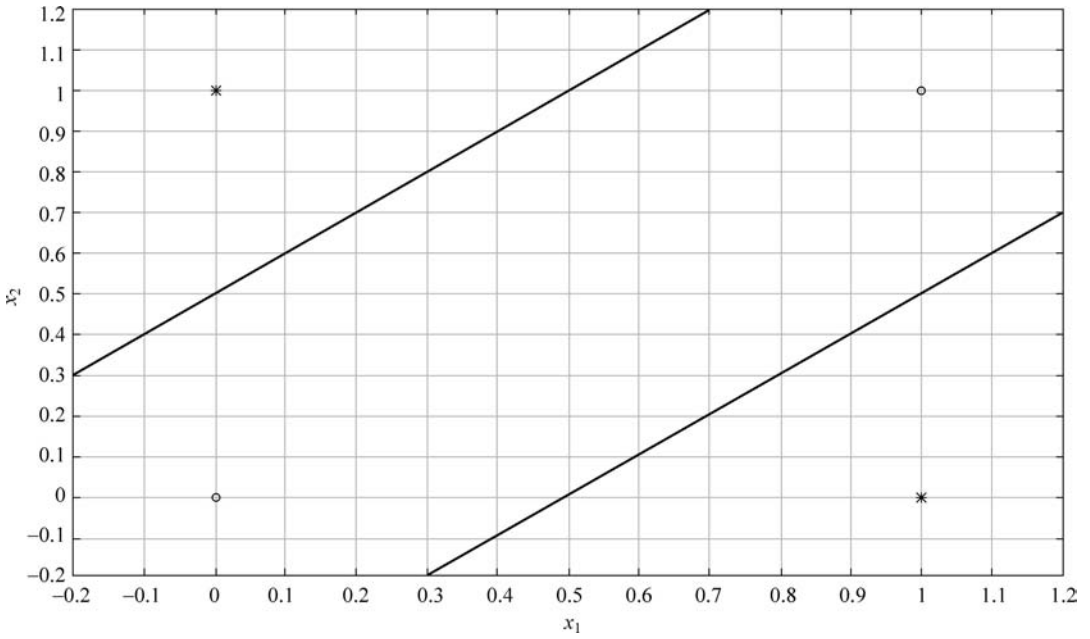


图 3-6 “异或”问题示意图

对于“异或”问题需要对原来的单层感知机进行修正，将其层数加大，变成多层感知机。不妨先修正为两层感知机，其结构如图 3-7 所示。很明显，这种两层结构的类型可以生成两条直线。在第一层，也就是输入层有两个神经元，分别输入两维数据；在第二层，也就是输出层有一个神经元。在输入层中的神经元首先绘制一条直线，随着权值、阈值的调整，这条直线可以任意变动，先分离出其中的一类点：如在图 3-7 中，可以将分类后剩下的结果再次输入另一个输入层的神经元，然后调整该神经元的权值、阈值，使另外一条直线进行调整，从而将图 3-6 中第一次分类左上部分的“ * ”类也分离出来。最后综合这两条直线的分类情

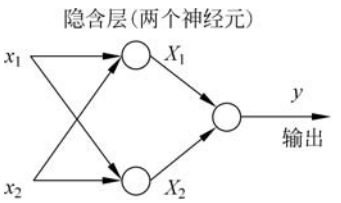


图 3-7　两层感知机结构示意图

况，达到对“异或”问题的求解。其具体步骤如下。

从图 3-7 中可以看出在隐含层有两个神经元，设第一个神经元对于两个输入的权值为(1，−1)，阈值为 0.5，则其生成分类线为

$$x_1 - x_2 + 0.5 \tag{3-15}$$

经过隐含层第一个神经元运算后这 4 个点的分类情况为

$$x_1 - x_2 + 0.5 = 1 \times 0 - 1 \times 0 + 0.5 = 0.5 \rightarrow \text{正例 } X_1 = 1$$

$$x_1 - x_2 + 0.5 = 1 \times 0 - 1 \times 1 + 0.5 = -0.5 \rightarrow \text{负例 } X_1 = 0$$

$$x_1 - x_2 + 0.5 = 1 \times 1 - 1 \times 0 + 0.5 = 1.5 \rightarrow \text{正例 } X_1 = 1$$

$$x_1 - x_2 + 0.5 = 1 \times 1 - 1 \times 1 + 0.5 = 0.5 \rightarrow \text{正例 } X_1 = 1$$

很明显这里有个错误分类，可以不去管它，因为毕竟还有另外两个神经元还没有运行。接下来，考察隐含层第二个神经元的情况。对于隐含层中第二个神经元可设其权值为(1，−1)，阈值为−0.5，则其生成分类线为

$$x_1 - x_2 - 0.5$$

经隐含层第二个神经元运算后这 4 个点的分类情况为

$$x_1 - x_2 - 0.5 = 1 \times 0 - 1 \times 0 - 0.5 = -0.5 \rightarrow \text{负例 } X_2 = 0$$

$$x_1 - x_2 - 0.5 = 1 \times 0 - 1 \times 1 - 0.5 = -1.5 \rightarrow \text{负例 } X_2 = 0$$

$$x_1 - x_2 - 0.5 = 1 \times 1 - 1 \times 0 - 0.5 = 0.5 \rightarrow \text{正例 } X_2 = 1$$

$$x_1 - x_2 - 0.5 = 1 \times 1 - 1 \times 1 - 0.5 = -0.5 \rightarrow \text{负例 } X_2 = 0$$

这样来看，隐含层中第二个神经元的分类也有一个不正确。但输出层的神经元还没有运行。将隐含层输出的数据再重新组成二元组(X_1　X_2)，将这个二元组作为输出层的输入，进入输出层的神经元。输出层的神经元权值设定为(−1，1)，阈值为 0.5，则其生成分类线为

$$-X_1 + X_2 + 0.5$$

将隐含层中神经元的输出作为输出层的输入，经上述的输出层运算后，输出层神经元的输出应为

$$-X_1 + X_2 + 0.5 = -1 \times 1 + 1 \times 0 + 0.5 = -0.5 \rightarrow \text{负例 } y = 0$$

$$-X_1 + X_2 + 0.5 = -1 \times 0 + 1 \times 0 + 0.5 = 0.5 \rightarrow \text{正例 } y = 1$$

$$-X_1 + X_2 + 0.5 = -1 \times 1 + 1 \times 1 + 0.5 = 0.5 \rightarrow \text{正例 } y = 1$$

$$-X_1 + X_2 + 0.5 = -1 \times 1 + 1 \times 0 + 0.5 = -0.5 \rightarrow \text{负例 } y = 0$$

将这个过程列成表 3-3，可以清楚地看到各神经元的输入、输出情况。至此，“异或”这一典型的线性不可分问题得以解决。在此过程中，并没有演示各神经元的调整过程，而是直接给出了权值、阈值的最终结果。各神经元权值、阈值的具体调整过程可以参见单层感知机的情况，此处略去。

从二维空间将这种模式推广，可以得出：如果增加神经元的数量和层数，就可以得出更为多样的分类区域，甚至可以拟合曲线形式的非线性分类线。此外，在数据维度上将其进行推广，就可以得到有限维空间上的曲面非线性分类器。图 3-8 给出了多层感知机的基本结构。

表 3-3 两个神经元解决“异或”问题

输入		隐含层的输出(输出层的输入)		输出	分类情况
x_1	x_2	第一个神经元输出 X_1	第二个神经元输出 X_2	y	
0	0	1	0	0	负例
0	1	0	0	1	正例
1	0	1	1	1	正例
1	1	1	0	0	负例

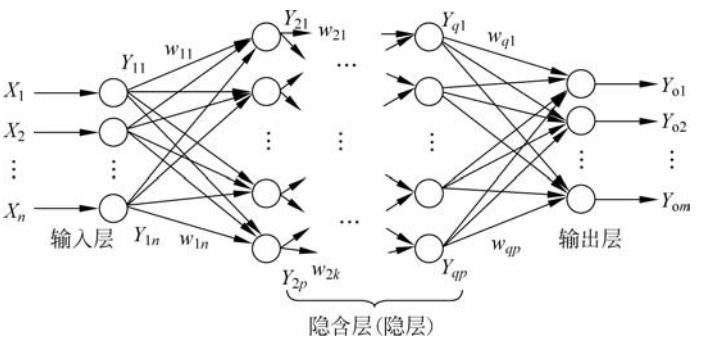


图 3-8 多层感知机的基本结构

在多层感知机中，所有的“层”被划分为三类：输入层、输出层以及隐含层(隐层)。输入层和输出层一般只有一个，隐含层的数目可以根据需要设置若干个。图中的圆圈代表各个神经元。输入层的神经元并不是严格意义上的神经元，只起信息的传递作用并无权值、阈值的连接，也没有激活函数。隐含层、输出层的各神经元与前述的单层感知机神经元的基本结构类似，但是其激活函数有了更多的选择范围，只要是非线性的函数都可以选择，例如第 2 章提到的在 MATLAB 软件中常用的激活函数，当然读者也可以自行定义。

前面已提到过，对于单层感知机权值、阈值调整需要进行学习(或训练)，对于多层感知机同样需要进行学习(或训练)。多层感知机的学习规则与单层感知机有相似之处。在分类这个大范畴内，不论感知机层数的多寡，一般都采用有监督的学习(或训练)模式。在此过程中，对于输入感知机的数据样本都有相应的期望输出。在训练的初始阶段，可能感知机的输出与期望的输出之间差异会比较大，按照训练规则经过一定的调整后最终会达到期望的目标。对于简单的线性分类问题来讲，分类的误差最终会彻底消除。但是对于非线性分类问题来讲就可能存在一些问题，那就是虽然经过很多次调整可能不会使分类误差最终彻底消除，这时就需要事先设定好一个能够容忍的误差范围，使得网络训练的误差不超过某个值，

这个值称为误差容限。一旦网络的训练误差进入误差容限范围内，就停止训练，固定当前的权值、阈值。在某些情况下也会出现经过大量的训练和调整权值、阈值，网络的误差仍然不能满足要求，出现这种情况的原因可能是感知机的架构存在问题：例如网络的层数和神经元的数目不太合适；或是在训练的初期权值、阈值设置不当。这时就需要重新对网络的结构和训练运算初值进行选择。

需要说明的是，对于神经网络的构建和训练并不是针对一些特定的数据样本的，而是要使用普通数据样本对神经网络进行训练，使得该神经网络对特征类型的数据很“敏感”。输入的外部数据样本虽然具体数据不同，但是其数据特征与样本数据相同时，神经网络能够很好地完成赋予它的任务。例如，对处理“异或”问题的神经网络进行训练后，它可以处理相应的线性不可分问题、非凸集数据分类问题，而不是仅解决“异或”问题。因此在一个神经网络经过学习或训练成功后，还需要对其进行一定的测试，使该神经网络能够很好地处理在数据集中的各种数据，这个过程称为神经网络的泛化。如果某个神经网络，在进行学习和训练之后只对典型类型数据起作用，而对于数据集中的非典型数据并不适用，这就是所谓的“过拟合”情况，说明神经网络对于进行训练的数据拟合（训练和学习）的程度太“过”了。

感知机对于数据分类处理非常有效，但是其本身结构也有一些问题，例如感知机的激活函数采用开关型的符号函数，致使其输出只有 0、1 两种状态，也限制了其应用。

感知机作为第一个从算法上完整描述的神经网络，开辟了人工神经网络的新型智能算法。它不仅简单易懂、概念清晰，而且包含了学习（训练）的思想和方法、损失函数求解以及优化方法。这些思想和基本方法为在其后神经网络及机器学习的发展奠定了基础。虽然现在感知机网络的实际应用逐渐在减少（被功能更为强大的神经网络所替代），但它是神经网络和智能算法的一个基础。

3.2 感知机的 MATLAB 实现

在了解感知机的基本结构和工作方式后，就可以利用 MATLAB 软件进行感知机网络的仿真了。在 MATLAB 软件中进行仿真，虽然距离神经网络的真正使用还有一段距离，但对于在更深程度上了解神经网络的运行方式是非常有帮助的。首先来看单层感知机的 MATLAB 仿真实现。

3.2.1 单层感知机的 MATLAB 仿真实现

在第 2 章中，曾经以一个例子说明在二维平面上进行线性分类的情况，如果能够将其进行推广，就可以得到在三维空间中的数据分类。

【例 3.2】 在三维平面上给出数据 $\boldsymbol{x}=[0.8\ -0.3\ -0.4;\ 0.7\ 0.3\ 0.5;\ 0.8\ -0.2\ 0.3]$。这些数据分属于两个数据类别，其分类情况用 $\boldsymbol{y}=[0\ 1\ 1]$ 表示。试使用单层感知机

对这些数据点进行分类并对网络进行仿真、绘制分类结果图形。

解：可按照例 2.1 中的模式，仅将数据维数扩充至三维即可。其代码如下(可与例 2.1 对比)：

```
x = [0.8 -0.3 -0.4; 0.7 0.3 0.5 ;0.8 -0.2 0.3];
y = [0 1 1];
P = [-1 1; -1 1; -1 1]
net = newp(P,1);
net = train(net,x,y);
A = sim(net,x);
plotpv(x,A)
plotpc(net.iw{1},net.b{1})
grid on
```

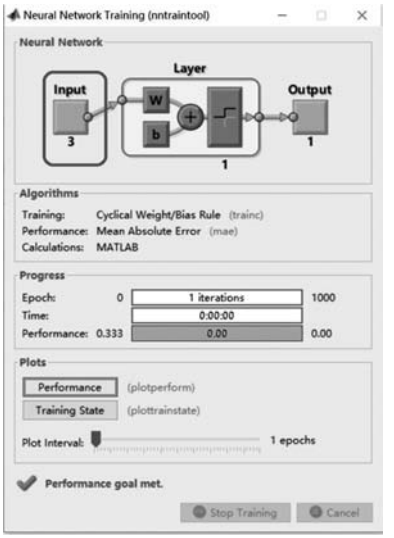


图 3-9　三维单层感知机

该感知机网络的基本情况如图 3-9 所示，从图中可以看到输入的维数已经改成了三维，输出的维数仍为 1，毕竟这还是属于分类情况。其运行结果如图 3-10 所示。

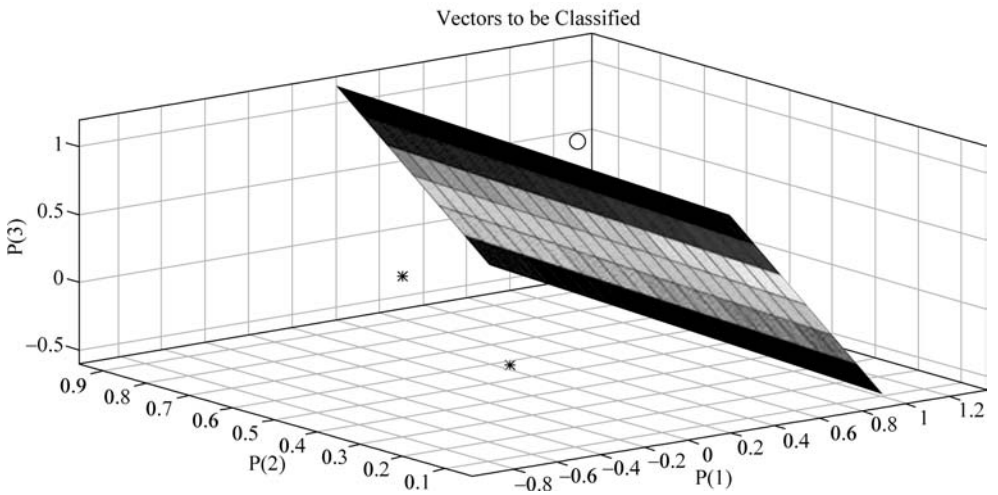


图 3-10　三维单层感知机对于三维线性可分类问题数据的分类情况

从这个图上可以清楚地看出，对于三维线性可分类问题，单层感知机仍然可以胜任，只不过分类线变为了分类平面。由此可以推知，即使对于高维线性可分类问题，单层感知机可以生成一个超平面将样本数据成功进行分类。这也是单层感知机的应用范围。

下面将对在感知机中用到的函数进行较详细的介绍。

1）newp 函数

函数名：newp。

基本格式：newp(PR，S,TF，LF)。

参数说明：

PR——输入数据向量的取值区间,一般为 R×2 矩阵,限定输入数据的最大值、最小值；

S——神经元的数目；

TF——神经网络的激活函数,例如可以是 hardlims 函数等；

LF——神经网络的学习函数,例如可以是 learnp 等。

根据例 3.2,可以观察所生成的感知机网络情况,这时需要输入指令：

```
>> whos
```

则得到以下的运行结果：

```
Name      Size      Bytes Class      Attributes
P         3×2       48               double
net       1×1       23337            network
x         3×3       72               double
y         1×3       24               double
```

从这个结果,可以了解到各个数据的详细属性。如果需要对所建立的感知机再做深入了解,还可以查询各变量的信息。在 MATLAB 软件中,很多函数是以结构体的形式出现的。因此可以查看结构体的情况。例如,在例 3.2 中利用函数 newp(PR，S,TF，LF)建立的感知机 net 就是类别(Class)为 network 的结构体,因此可用以下方法了解感知机 net 的各项相关信息。

(1) 输入：>> inputweights＝net. inputweights{1} ——了解该感知机网络的权值情况。

可得到的属性列表：

```
inputweights =
    Neural Network Weight
delays: 0
initFcn: 'initzero'
initSettings: (none)
learn: true
learnFcn: 'learnp'
learnParam: (none)
size: [1 3]
weightFcn: 'dotprod'
weightParam: (none)
userdata: (your custom info)
```

(2) 输入：>> biases＝net. biases{1} ——了解该感知机网络的阈值情况。

可得到的属性列表：

```
biases =
```

```
    Neural Network Bias
initFcn: 'initzero'
learn: true
learnFcn: 'learnp'
learnParam: (none)
size: 1
userdata: (your custom info)
```

(3) 输入：>> trainFcn=net.trainFcn ——了解该感知机网络的训练函数情况。

可得到的属性列表：

```
trainFcn =
    trainc
```

由第 2 章的内容可知，这是循环训练函数，其基本的情况在第 2 章中有说明。对神经网络进行仿真，会弹出如图 3-9 所示的网络训练界面。除了前述的内容外，从图中还可以看出神经网络的训练状态以及性能情况。

2) sim 函数

网络仿真函数，其基本格式在第 2 章已有说明，此处不再赘述。

在这个例子中，网络的很多属性都使用了默认属性，例如延迟时间、初始化函数等，因此需要给出详细说明的地方并不多。

3) plotpv 绘图函数

这个函数用来绘制感知机的输入向量和目标向量。其基本格式为：

```
plotpv(X,T)
```

其中，X 为输入向量，T 为目标向量。输入向量和输出向量的列数相等。

4) plotpc 绘图函数

这个函数是用来绘制感知机的分类线(面)的。其基本格式为：

```
plotpc(W,b)
```

其中，W 为分类线(面)的权值，b 为分类线(面)的阈值。在此函数中，分类线的权值为 S×R 阶矩阵。一般来讲，R 应小于或等于 3(如果大于 3，则为超平面，不可能以图形的形式给出)。而分类线的阈值 b 为 S×1 阶向量。

在第 2 章中提到过，除了使用 MATLAB 软件编写 m 文件实现神经网络的分析以外，还可以使用基于图形界面的 MATLAB 神经网络工具箱分析。在该工具箱中可以看到，对于单层感知机，MATLAB 软件提供了两种激活函数，分别是单极性开关特性激活函数和双极性开关特性激活函数。而学习函数也提供了两种，分别是 learnp 和 learnpn。learnp 函数在第 2 章中已介绍，learnpn 函数为标准化的学习函数。与 learnp 函数相比，learnpn 函数对于输入量大小变化不敏感。当输入的数据向量变化比较大时，使用 learnpn 函数，可以加快感知机的运算速度。

单层感知机网络结构简单，权值、阈值调节起来相对比较快，对于线性分类问题有着非常好的适应性，以上所举的例子比较简单，主要用来说明单层感知机的运行方式。下面将以对英文字母和阿拉伯数字的识别为例说明感知机的实际应用。

【例 3.3】 试举例说明使用单层感知机如何识别基本英文字母。

解：为了能够实现对基本英文字母的识别，首先必须让 MATLAB 软件能够读懂输入的信息。因为当下计算机均为数字化计算机，因此需要将字母进行数字化处理。使用一种 7×4 的方格对英文字母进行数字化，如图 3-11 所示。可以规定字母笔画占据方格的记为 1，而没有占据方格的记为 0。这样，任何一个字母均可以用 7×4 的矩阵表示（如在小方格内为空白记 0，如在小方格内有笔画占用，则记为 1）。考虑到感知机的泛化问题，这里使用 3 种不同的字体对同一个字母进行训练。例如对于字母 E 的 Times New Roman 字体，其数字化表达为：

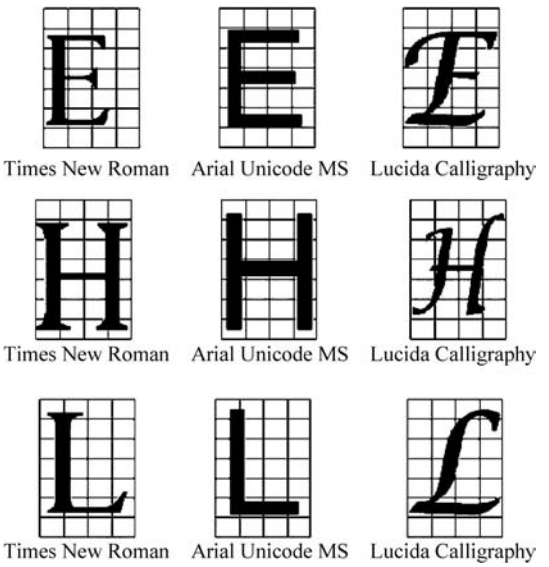


图 3-11 使用方格对英文字母进行数字化处理

$$\boldsymbol{E}_1=\begin{bmatrix}0&0&0&0\\1&1&1&0\\1&0&0&0\\1&1&1&0\\1&0&0&0\\1&1&1&0\\0&0&0&0\end{bmatrix}$$

则字母 E 相应的 Arial Unicode MS 字体及 Lucida Calligraphy 字体数字化表达为：

$$\boldsymbol{E}_2=\begin{bmatrix}0&0&0&0\\1&1&1&1\\1&0&0&0\\1&1&1&1\\1&0&0&0\\1&1&1&1\\0&0&0&0\end{bmatrix},\quad \boldsymbol{E}_3=\begin{bmatrix}0&0&0&0\\1&1&1&1\\1&1&0&0\\0&1&1&1\\0&1&0&0\\1&1&1&1\\0&0&0&0\end{bmatrix}$$

对于字母 H 有：

$$\boldsymbol{H}_1=\begin{bmatrix}0&0&0&0\\1&1&1&1\\1&0&0&1\\1&1&1&1\\1&0&0&1\\1&0&0&1\\1&1&1&1\end{bmatrix},\quad \boldsymbol{H}_2=\begin{bmatrix}1&0&0&1\\1&0&0&1\\1&0&0&1\\1&1&1&1\\1&0&0&1\\1&0&0&1\\1&0&0&1\end{bmatrix},\quad \boldsymbol{H}_3=\begin{bmatrix}0&0&0&1\\1&1&1&0\\1&1&1&0\\0&1&1&0\\1&1&0&0\\1&0&1&0\\0&0&0&0\end{bmatrix}$$

对于字母 L 有：

$$\boldsymbol{L}_1=\begin{bmatrix}1&1&0&0\\1&0&0&0\\1&0&0&0\\1&0&0&0\\1&0&0&1\\1&1&1&1\\0&0&0&0\end{bmatrix},\quad \boldsymbol{L}_2=\begin{bmatrix}1&0&0&0\\1&0&0&0\\1&0&0&0\\1&0&0&0\\1&0&0&0\\1&1&1&1\\0&0&0&0\end{bmatrix},\quad \boldsymbol{L}_3=\begin{bmatrix}0&0&1&1\\0&1&1&1\\0&1&0&0\\0&1&0&0\\0&1&0&0\\1&1&1&1\\0&0&0&0\end{bmatrix}$$

下面开始编写 MATLAB 代码：

```
% 首先输入训练样本数据
% 字母 E 的 3 种字体的数字化样本
% (按照上述数字化表达以 MATLAB 数据输入的标准格式输入,此处考虑到篇幅问题不再展开)
E1 = [ … ];
E2 = [ … ];
E3 = [ … ];
% 字母 H 的 3 种字体的数字化样本
H1 = [ … ];
H2 = [ … ];
H3 = [ … ];
% 字母 L 的 3 种字体的数字化样本
L1 = [ … ];
L2 = [ … ];
```

```
L3 = [ … ];
% 选取字母的各种字体的数字化样本进行训练
p = [E1(1:end);E2(1:end);E3(1:end);H1(1:end);H2(1:end);H3(1:end);L1(1:end);L2(1:end);L3
(1:end)]';

% t 为目标输出,每个列向量对应一个样本的目标输出.其中向量[0 0 1]'代表字母 E,向量[0 1 0]'代
% 表字母 H,其中向量[1 0 0]'代表字母 L
t = [0 0 0 0 0 0 1 1 1;
  0 0 0 1 1 1 0 0 0;
  1 1 1 0 0 0 0 0 0];

% pr 初始化为 28 行,2 列的零矩阵
pr = zeros(28,2);

% 设定输入向量每个维度的最小值和最大值
pr(:,2) = 1;
% 感知机网络参数设置函数、激活函数、学习(训练)函数均为默认
net = newp(pr,3);

% 设置最大迭代次数为 20
net.trainParam.epochs = 20;

% 将训练集 p 和目标输出 t(分类结果)载入 net
net = train(net,p,t);
```

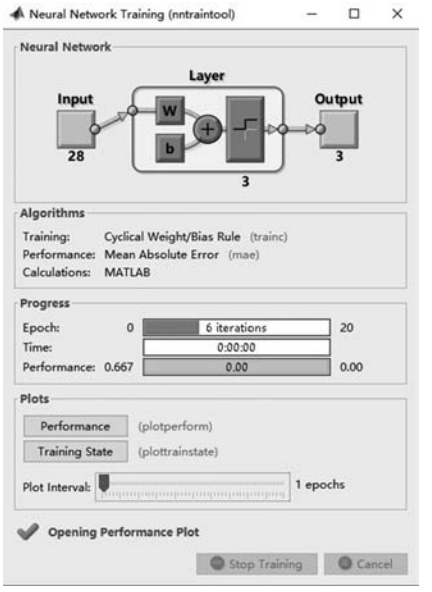


图 3-12　例 3.3 建立的单层感知机网络运行界面

至此一个名为 net 的单层感知机就建立并训练完成了。在对上述代码运行后,会弹出如图 3-12 所示的界面。从图中可以看出,输入的数据量为 7×4=28 个,单层神经元的数目为 3 个,输出为 3 个(需要识别的字母),激活函数为单极性开关型激活函数。原来预计需要进行 20 步迭代,结果进行了 6 步迭代就进入稳态。如果想进一步了解迭代学习(训练)的过程,可单击 Performance 按钮,就可以看出各步迭代学习(或训练)的情况,如图 3-13 所示。

前面提及神经网络需要有一定的泛化能力,就是说神经网络仅能识别曾经训练过的数据样本是不够的,数据样本发生一些非特征性的变化也应该能够识别。因此可以选择一些不是已经训练过的,但是又具有基本特征的数据样本进行测试。

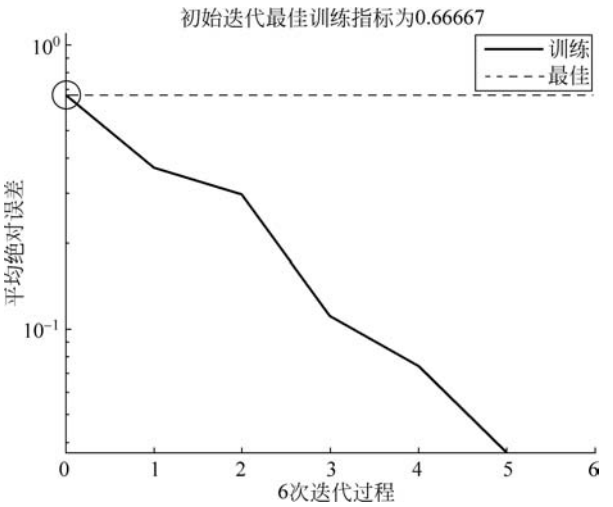
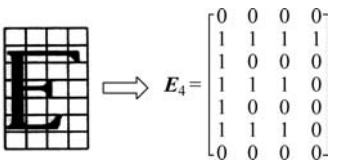
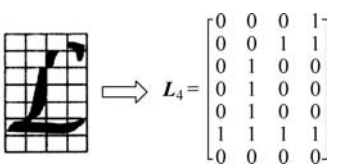


图 3-13　例 3.3 建立的单层感知机网络迭代情况

例如字母 E 的 Times New Roman 字体有些错印，变成：



而字母 L 的 Lucida Calligraphy 字体也错印，变成：



下面来考察单层感知机的泛化能力。在 MATLAB 软件中输入如下代码：

```
% 使用 sim 将错印样本进行测试,返回到 A
A1 = sim(net,E4(1:end)');
A2 = sim(net,L4(1:end)');
```

然后在命令行窗口，分别输入 A1 和 A2，得到以下结果：

```
>> A1
A1 =
     0
     0
     1
>> A2
```

```
A2 =
    1
    0
    0
```

由前述注释“向量[0 0 1]代表字母 E,向量[1 0 0]代表字母 L”,可以看出该感知机成功识别了这两个非样本集中的字母,说明这个单层感知机有一定的泛化能力。

上例给出了单层感知机对于英文字母的识别情况,下面再举例说明单层感知机对于数字的识别。

对于数字的奇偶识别有很多种方法,此处使用感知机进行分类识别主要是为了再次展示其基本特点,同时也能从中发现问题所在。

【例 3.4】 试举例说明使用单层感知机如何对小于 10 的阿拉伯数字进行奇偶识别分类。

解:与识别基本英文字母的相同,使用一定的平面方格划分对阿拉伯数字进行数字化,如图 3-14 所示。对于阿拉伯数字采用了 Times New Roman 字体。从上面的例子来看,其实采用哪种字体对识别结果的影响并不大。

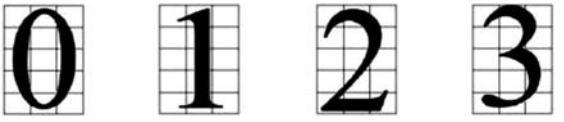


图 3-14 阿拉伯数字的数字化

因为要进行奇偶识别,输出的结果只有两个:“奇”或“偶”,因此对于阿拉伯数字的数字化方法可以进一步简化,即不再使用二维矩阵,而只使用一维向量即可。例如,0 数字化为:

$$[1\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1]$$

这样可以在计算上提高一点效率。由此可得 10 以内阿拉伯数字的数字化结果如表 3-4 所示。

在有了上述准备后,可编写 MATLAB 代码如下:

```
%输入训练样本
p=[1 1 1 1 0 1 1 0 1 1 0 1 1 1 1;        %数字 0
   1 1 0 0 1 0 0 1 0 0 1 0 1 1 1;        %数字 1
   1 1 0 1 1 1 1 0 1 0 0 1 0 1 1 1;      %数字 2
   1 1 1 0 1 1 0 1 1 0 0 1 1 1 1;        %数字 3
   …;
   1 1 1 1 0 1 1 1 1 0 1 1 1 1 0]';      %数字 9
```

表 3-4 10 以内阿拉伯数字的数字化结果

阿拉伯数字	图像数字化结果
0	1 1 1 1 0 1 1 0 1 1 0 1 1 1 1
1	1 1 0 0 1 0 0 1 0 0 1 0 1 1 1
2	1 1 0 1 1 1 0 1 0 0 1 0 1 1 1
3	1 1 1 0 1 1 0 1 1 0 0 1 1 1 1
⋮	⋮
9	1 1 1 1 0 1 1 1 1 0 1 1 1 1 0

```
%输入目标向量,奇数为1,偶数为0
t=[0 1 0 1 0 1 0 1 0 1];

%设定输入向量每个维度的最小值和最大值
pr=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];

%感知机网络参数设置函数、激活函数、学习(训练)函数均为默认
net=newp(pr,1);

%设置最大迭代次数为20
net.trainParam.epochs=20;

%将训练集p和目标输出t(分类结果)载入网络net
[net,Tr]=train(net,p,t);
```

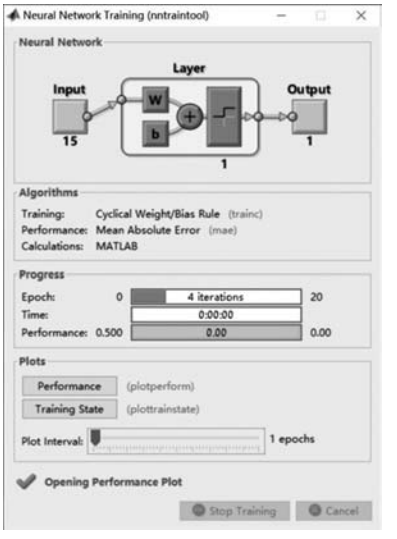


图 3-15 例 3.4 中单层感知机的情况

运行以上程序代码,可以得到如图 3-15 所示的结果。从图中可以看出由于输出的结果只是二分类:非奇即偶,因此网络的结果简单了一些。但是即使这个简单的感知机也有一定的泛化能力。例如由于一定的原因,1 数字化为:

"1"→ [0 1 0 0 1 0 0 1 0 0 1 0 0 1 0]'

此时来考察该感知机网络的泛化能力,输入:

```
>> a=[0 1 0 0 1 0 0 1 0 0 1 0 0 1 0]';
>> A=sim(net,a)
```

得到:

```
A =
   1
```

判断为奇数。证明这个感知机有一定的泛化能力。输入以下指令,还可以得到分类超平面的权值、阈值:

```
%给出分类超平面的权值、阈值
```

```
Weight = net.iw{1}
Bias = net.b{1}
```

得到：

```
Weight =
    1 0 3 -5 1 0 -3 3 2 -5 0 1 0 0 -2
Bias =
    0
```

由于是分类超平面，超过了三维，因此没有办法绘制分类图像。

单击图 3-15 界面中的 Performance 按钮，可以看到各步迭代学习（或训练）的情况，如图 3-16 所示。可以看到，在程序代码中预计需要 20 步的训练步数，实际上 4 步就完成了训练过程。如果在 MATLAB 界面中输入：

```
% 给出每一步训练的误差值
perf = Tr.perf
```

则可以清晰地看到每一步训练的误差情况，输出为：

```
perf =
         0.5000 0.5000 0.5000 0.1000 0
```

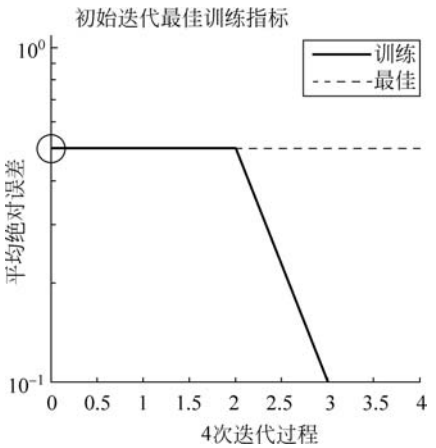


图 3-16　例 3.4 中单层感知机网络迭代情况

从以上例子可以看出，单层感知机的结构简单，学习、训练调整较为便捷。在能够进行线性分类的场合，单层感知机的表现非常良好，与维数问题无关。即使是看起来有些复杂的问题，但只要其是线性可分的，使用单层感知机就可以迎刃而解。

3.2.2　多层感知机的 MATLAB 仿真实现

单层感知机对于线性分类问题的处理得心应手，但是在很多场合，分类问题并不仅仅是线性可分的。面对线性不可分类问题，单层感知机就无能为力了。前面所说的“异或”问题

就是一个典型的例子。在 3.1.2 节讨论了多层感知机的基本结构与算法基础，下面利用 MATLAB 实现多层感知机分类问题。

【例 3.5】 试使用 MATLAB 构建多层感知机，并利用该多层感知机解决“异或”这一线性不可分问题。

解：先来看下面的例子。其代码如下：

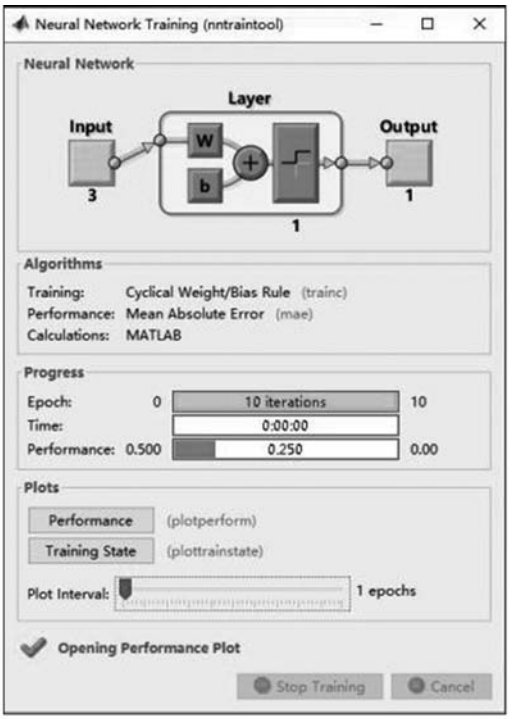
```
clear all          %清空内存——在编写行数较多的代码时最好添加此代码,以避免变量混淆等问题
clc                %清屏——可使 MATLAB 编辑页面整洁

%建立隐含层感知机并对其进行初始化
pr1 = [0 1; 0 1];                  %设置输入变量的值域
net1 = newp(pr1,3);                %建立隐含层感知机
%对隐含层的权值、阈值进行初始化,均为随机函数
net1.inputweights{1}.initFcn = 'rands';
net1.biases{1}.initFcn = 'rands';
%对隐含层进行初始化
net1 = init(net1);
iw1 = net1.iw{1};
b1 = net1.b{1};

%对隐含层进行仿真,产生输出
p1 = [0 0; 0 1; 1 0; 1 1]';        %输入被识别的向量
[a1, pf] = sim(net1,p1);           %仿真输出

%对输出层进行初始化
pr2 = [0 1; 0 1; 0 1];             %设置输出层输入变量的值域
net2 = newp(pr2,1);                %建立输出层感知机

%对输出层进行训练
net2.trainParam.epochs = 10;       %设置训练步数为 10
net2.trainParam.show = 1;          %在每一步迭代后都显示迭代的结果,以监控每一步训练的情况
p2 = ones(3,4);                    %产生一个全为 1 的 3×4 的矩阵,以便与隐含层输出结果进行运算
p2 = p2. * a1;                     %将隐含层输出结果作为输出层的输入向量
t2 = [0 1 1 0];                    %设置输出层的输出目标向量
[net2,tr2] = train(net2,p2,t2);    %对输出层进行训练
epoch2 = tr2.epoch                 %输出训练每一步的标号
perf2 = tr2.perf                   %输出每一步的训练误差

%给出输出层最终的权值、阈值
iw2 = net2.iw{1};
b2 = net2.b{1};
```

然后，在 MATLAB 软件中运行此程序，得到如下结果：

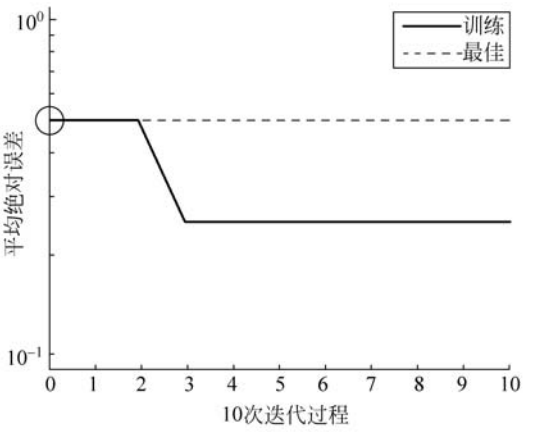
```
epoch2 =
     0     1     2     3     4     5     6     7     8     9    10
```

```
perf2 =
    0.5000   0.5000   0.5000   0.2500   0.2500   0.2500   0.2500   0.2500   0.2500   0.2500
0.2500
```

从以上结果可以看出，这个两层感知机经过 10 步训练后并不渐近收敛，仿真结果如图 3-17 所示。



(a) 例3.5建立的多层感知机的情况



(b) 例3.5多层感知机网络迭代情况

图 3-17　例 3.5 建立的多层感知机在 MATLAB 中第一次仿真结果

那么，这样能达到对“异或”问题的正确分类吗？可以对其输出进行考察，输入：

```
>> a2 = sim(net2,p2)
```

得到以下结果：

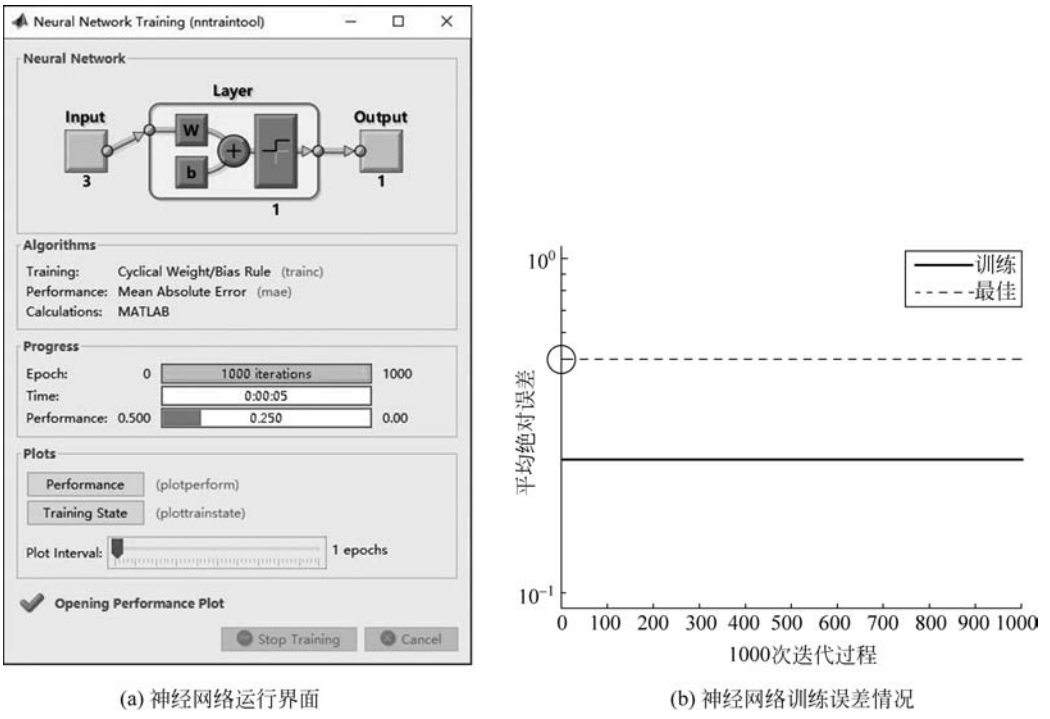
```
a2 =
     0     1     0     0
```

从程序的运行结果来看并没有实现对“异或”问题的正确分类，这是什么原因导致的呢？是否因为在例 3.5 的代码中设置的训练步数（为 10）太少了呢？不妨把步数增加为 1000 次，再次运行以上代码得到如图 3-18 所示的结果。从这个结果中可以看到，虽然将训练的步数增加为 1000 次，但是得到的结果还是最小误差为 0.25。再次考察其运行结果，得到：

```
a2 =
     1     1     1     0
```

这次的运行不但没有实现对“异或”问题的正确分类，而且和上次的运行结果也不一样。

那么问题究竟在什么地方呢？分析例 3.5 所给出的代码可以发现，在对隐含层的权值、阈值进行初始化时，均采用了随机函数，这使得在隐含层感知机的输出变为随机的，因此整个网络的运行很难人为控制，很有可能达不到训练的要求；而且每次的运行结果也不尽相同。在实际的仿真运算过程中需要不断地运行以上代码，才能逐渐进入稳态，达到训练要求，如图 3-19 所示。



(a) 神经网络运行界面　　(b) 神经网络训练误差情况

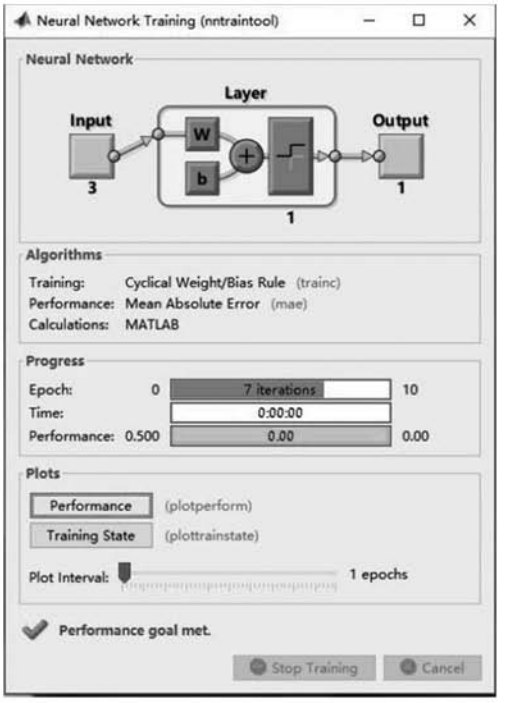
图 3-18　例 3.5 中多层感知机训练 1000 次的仿真结果

图 3-19 给出了运行例 3.5 代码 5 次、6 次和 9 次后的结果界面和误差性能。

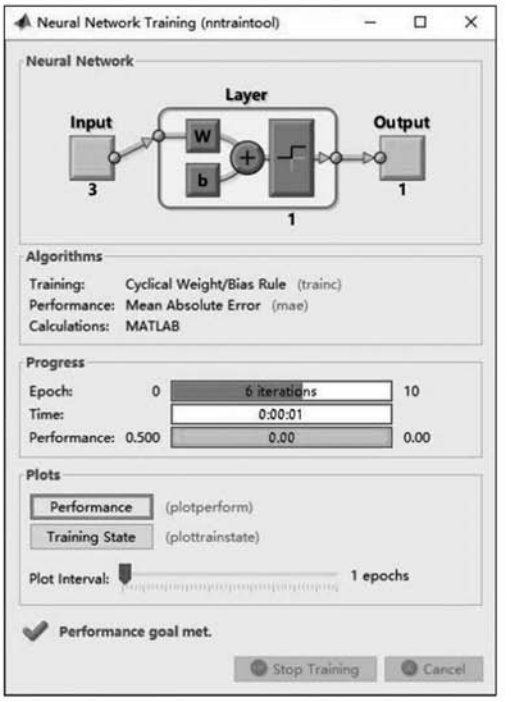
从图上可以看出，在运行多次以后整个多层感知机的性能越来越好，误差性能均在 10 步之内进入稳态，其误差达到了 0。在第 9 次运行时竟然可以在 3 步达到要求。在此基础上可以检验这个多层感知机的输出。这 3 次运行的结果均为：

```
a2 =
     0     1     1     0
```

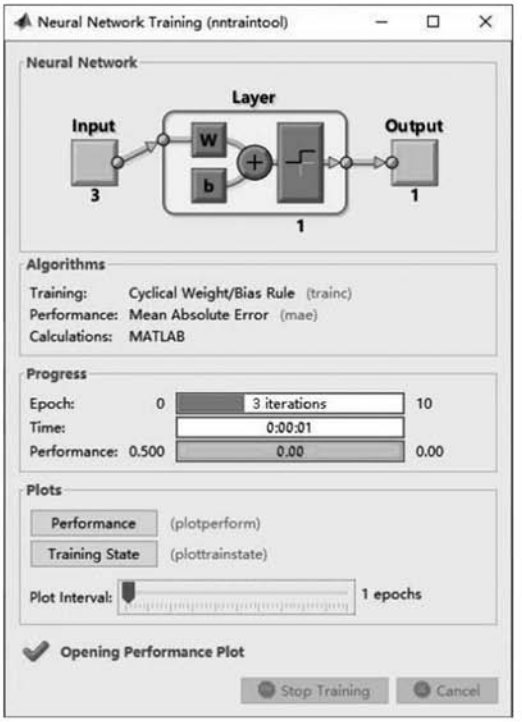
至此，该神经网络已完全实现“异或”功能，达到了训练的要求。根据运行结果，将这 3 次运行的权值、阈值结果列于表 3-5 中。



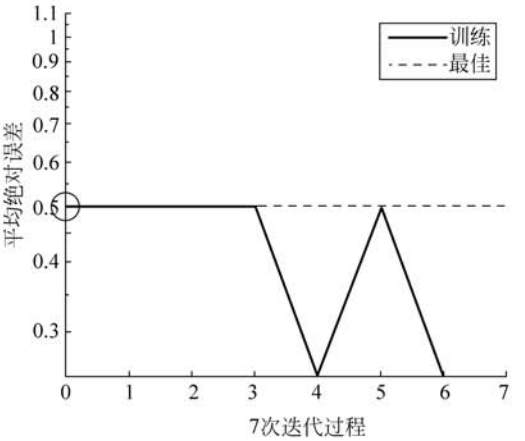
(a) 运行代码5次的界面



(b) 运行代码6次的界面

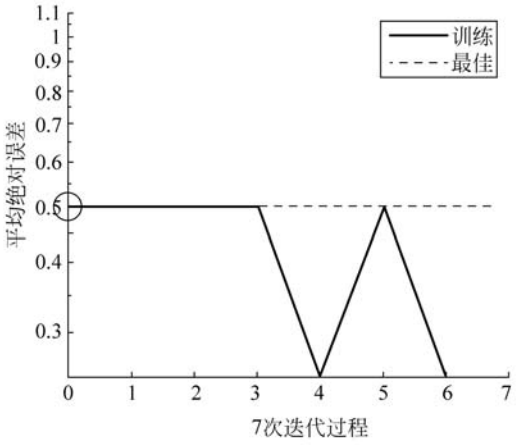


(c) 运行代码9次的界面

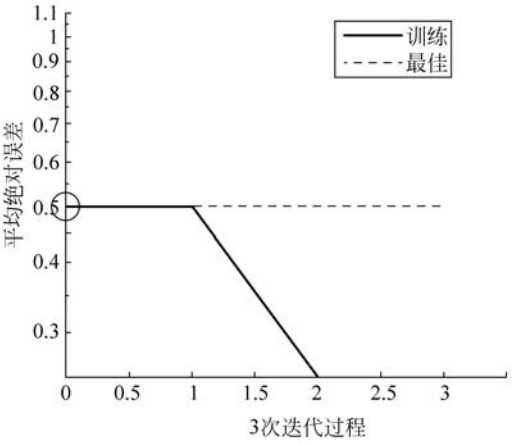


(d) 运行代码5次后的误差性能

图 3-19　多次运行例 3.5 中代码的结果界面



(e) 运行代码6次后的误差性能



(f) 运行代码9次后的误差性能

图 3-19 （续）

表 3-5 例 3.5 代码运行多次的权值、阈值

项 目		第 5 次运行	第 6 次运行	第 9 次运行
权值	隐含层 iw1	−0.8324 −0.6952 −0.5420 0.6516 0.8267 0.0767	−0.3246 −0.7776 0.8001 0.5605 −0.2615 −0.2205	−0.9140 0.4634 −0.6620 0.2955 0.2982 −0.0982
	输出层 iw2	−2 −1 −2	2 1 0	2 −1 0
阈值	隐含层 b1	0.3784 0.4963 −0.0989	0.8896 −0.0183 −0.0215	−0.2937 0.6424 −0.9692
	输出层 b2	2	−3	0

从表 3-5 中可以再一次看出，虽然这 3 次的运行结果都达到了神经网络的训练目的，但是每次运行结果的权值、阈值都不同，说明对于“异或”问题的分类线有很多种，而且都可以达到分类要求。这不禁会让人产生疑问和困惑，关于这一点将会在随后的内容中进行讨论。

通过编写代码(.m)构建多层感知机解决“异或”问题之后，再来使用基于图形界面的 MATLAB 神经网络工具箱的方法构建多层感知机解决“异或”问题。在命令行窗口下输入指令：

```
>> nntool
```

会出现神经网络工具箱的图形化界面。选择创建感知机，出现神经网络各参数界面。从这个界面可以看出并不能构建两层及以上的感知机！这是因为在基于图形界面的 MATLAB 神经网络工具箱中默认“感知机”就是单层感知机，并不支持多层感知机的构建！

在 MATLAB 中另一种构建神经网络的方法是使用神经网络工具箱中的定制网络方法，原则上说这种方法几乎可以构建任何一种成熟和经典的神经网络。针对例 3.5 的“异或”问题，可以利用该方法构建两层感知机网络，代码如下：

```
clear all
clc
%设置网络的基本结构
net = network;
net.numInputs = 1;
net.numLayers = 2;                          %构建两层感知机网络

%设置隐含层的连接
net.biasConnect = [1;1];                    %设置网络阈值
net.inputConnect = [1;0];                   %设置网络的输入权值
net.LayerConnect = [0 0 ;1 0];              %各层的权值
net.outputConnect = [0 1];                  %设置输出连接情况

%隐含层的相关设置
net.layers{1}.size = 2;                     %设置隐含层规模
net.layers{1}.transferFcn = 'hardlim';  %隐含层激活函数为单极性开关特性激活函数(硬限函数)
net.layers{1}.initFcn = 'initnw';           %初始化隐含层函数

%输出层的相关设置
net.layers{2}.size = 1;                     %设置输出层规模
net.layers{2}.transferFcn = 'hardlim';      %输出层激活函数为单极性开关特性激活函数
net.layers{2}.initFcn = 'initnw';           %初始化输出层函数

%网络相关参数设置
net.adaptFcn = 'trains';                    %使用自适应函数在线学习
net.performFcn = 'mse';                     %两层感知机网络的性能指标函数选用均方误差
net.trainFcn = 'trainlm';                   %训练算法选为 Levenberg - Marquardt 算法
net.initFcn = 'initlay';                    %逐层进行初始化

%对网络总体进行初始化
net = init(net);

%进行网络输入和目标输出的设置
P = [0 0; 0 1; 1 0; 1 1]';
T = [0 1 1 0];

%设置训练步数并开始对网络进行训练
net.trainParam.epochs = 1000
net = train(net, P, T);
```

运行以上程序代码会得到关于该两层感知机网络的相关信息如下。

```
net =
    Neural Network

    name: 'Custom Neural Network'                    % 网络名称
    userdata: (your custom info)                     % 应用数据

    dimensions:                                      % 输入/输出的各种参数
         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
numFeedbackDelays: 0
numWeightElements: 5
        sampleTime: 1

    connections:                                     % 感知机各层的连接情况
        biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:                                      % 感知机各层的数据情况
              input: Equivalent to inputs{1}
             output: Equivalent to outputs{2}

             inputs: {1x1 cell array of 1 input}
             layers: {2x1 cell array of 2 layers}
            outputs: {1x2 cell array of 1 output}
             biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:                                       % 感知机各层使用到的各种函数情况
           adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: (none)
       divideParam: (none)
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {}
        plotParams: {1x0 cell array of 0 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
```

```
                .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                .mu_inc, .mu_max

weight and bias values:                                  %感知机各层权值、阈值情况
            IW: {2x1 cell} containing 1 input weight matrix
            LW: {2x2 cell} containing 1 layer weight matrix
             b: {2x1 cell} containing 2 bias vectors

methods:                                                 %感知机相关函数的应用情况
          adapt: Learn while in continuous use
      configure: Configure inputs & outputs
         gensim: Generate Simulink model
           init: Initialize weights & biases
        perform: Calculate performance
            sim: Evaluate network outputs given inputs
          train: Train network with examples
           view: View diagram
    unconfigure: Unconfigure inputs & outputs

evaluate: outputs = net(inputs)                          %输出评价情况
```

同时，也会弹出如图 3-20 所示的窗口。从图中可以清晰地看到两层感知机的网络结构、连接情况、激活函数以及训练函数、性能指标函数等，比编写代码构建感知机形象得多。在有了上述准备后，可以对网络进行检验。输入指令：

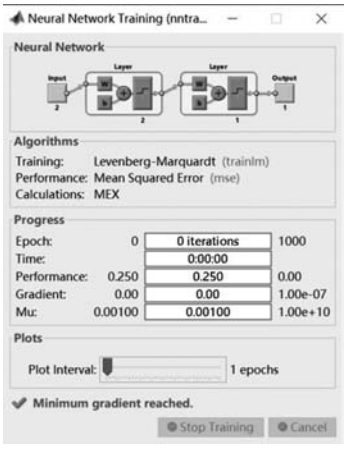


图 3-20　网络构建方法所得到的两层感知器运行界面

```
>> a = sim(net,P)
```

可以得到结果：

```
a =
   0   0   1   0
```

很明显，该方法并没有能够完全解决“异或”问题，这一点与编写代码构建的两层感知机神经网络的情况是一致的，同样也需要进行多次运行才能达到对“异或”问题的完全解决。

从以上两种方法对于“异或”问题的解决，可以引发我们思考一些问题。对于线性可分类问题，单层感知机的表现非常良好，训练速度快，而且还具有一定的泛化能力。从例 3.2～例 3.4 就可以看出，只要是线性可分类问题，不论其维度有多高，都可以使用单层感知机进行处理。但是一旦所面临的是线性不可分类问题，单层感知机就无法解决。这时可以将感知机的层数增大，由单层感知机变为两层感知机或多层感知机是可以解决问题的。然而感知机的训练过程过于漫长，而

且每次得到的结果(分类线)几乎都不相同。那么这些分类线之间究竟是什么关系呢？哪种分类线又是“最佳”的呢？神经网络(感知机)方法对于简单的“异或”问题都尚且如此(如例3.5),更别提其他更为复杂的非线性分类问题了。这个问题一出现就引起了人们对于人工智能方法的争议。

本章开篇就提到,人工智能领域存在两个学术派别。对神经网络方法有争议的学派提出了很多质疑,他们认为“神经网络算法不能被证伪”“神经网络设计者用高超的工程技巧弥补了数学上的缺陷”“神经网络不像一门科学,更像一门工程技巧”。学术争鸣往往会在很多方面促进学术发展。这两个学派别分别在不同的方向上进行深入研究,各自都取得了很多学术成果。统计学习理论学派从数学的逻辑和符号严格推证方法出发,逐渐形成了统计学习方法,支持向量机方法就是这一学派的优秀成果之一。而受到质疑的神经网络方法则在不断地改进,形成了具有很多独特风格的神经网络,在各种类型的工程实践上都有广泛的应用。