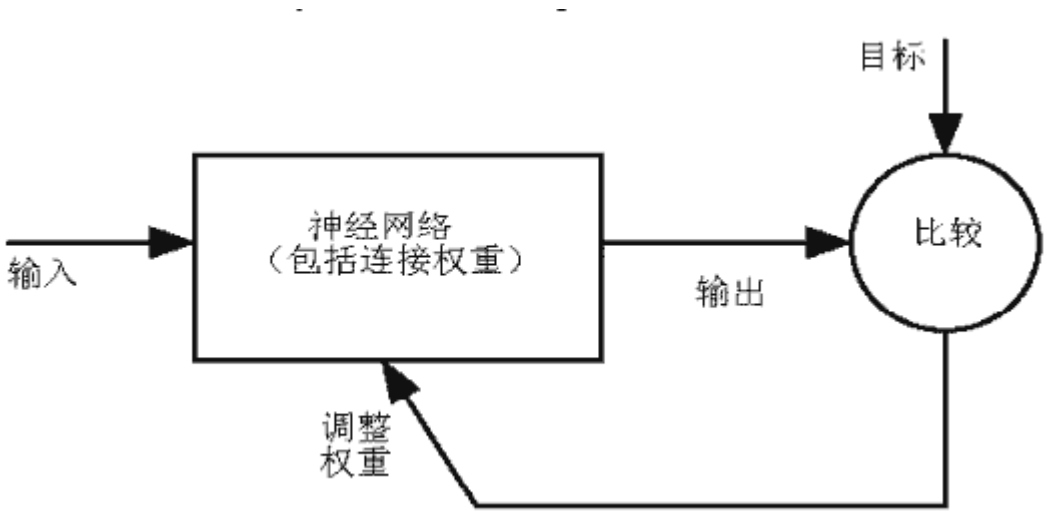
Matlab神经网络工具箱应用简介

第一章 介绍

1. 神经网络

神经网络是单个并行处理元素的集合，我们从生物学神经系统得到启发。在自然界，网络功能主要由神经节决定，我们可以通过改变连接点的权重来训练神经网络完成特定的功能。

一般的神经网络都是可调节的，或者说可训练的，这样一个特定的输入便可得到要求的输出。如下图所示。这里，网络根据输出和目标的比较而调整，直到网络输出和目标匹配。作为典型，许多输入/目标对应的方法已被用在有监督模式中来训练神经网络。



神经网络已经在各个领域中应用，以实现各种复杂的功能。这些领域包括：模式识别、鉴定、分类、语音、翻译和控制系统。

如今神经网络能够用来解决常规计算机和人难以解决的问题。我们主要通过这个工具箱来建立示范的神经网络系统，并应用到工程、金融和其他实际项目中去。

一般普遍使用有监督训练方法，但是也能够通过无监督的训练方法或者直接设计得到其他的神经网络。无监督网络可以被应用在数据组的辨别上。一些线形网络和Hopfield网络是直接设计的。总的来说，有各种各样的设计和学习方法来增强用户的选择。

神经网络领域已经有 50 年的历史了，但是实际的应用却是在最近 15 年里，如今神经网络仍快速发展着。因此，它显然不同与控制系统和最优化系统领域，它们的术语、数学理论和设计过程都已牢固的建立和应用了好多年。我们没有把神经网络工具箱仅看作一个能正常运行的建好的处理轮廓。我们宁愿希望它能成为一个有用的工业、教育和研究工具，一个能够帮助用户找到什么能够做什么不能做的工具，一个能够帮助发展和拓宽神经网络领域的工具。因为这个领域和它的材料是如此新，这个工具箱将给我们解释处理过程，讲述怎样运用它们，并且举例说明它们的成功和失败。我们相信要成功和满意的使用这个工具箱，对范例和它们的应用的理解是很重要的，并且如果没有这些说明那么用户的埋怨和质询就会把我们淹没。所以如果我们包括了大量的说明性材料，请保持耐心。我们希望这些材料能对你有帮

助。

这个章节在开始使用神经网络工具箱时包括了一些注释，它也描述了新的图形用户接口和新的运算法则和体系结构，并且它解释了工具箱为了使用模块化网络对象描述而增强的机动性。最后这一章给出了一个神经网络实际应用的列表并增加了一个新的文本--神经网络设计。这本书介绍了神经网络的理论和它们的设计和应用，并给出了相当可观的MATLAB和神经网络工具箱的使用。

2. 准备工作

基本章节

第一章是神经网络的基本介绍，第二章包括了由工具箱指定的有关网络结构和符号的基本材料以及建立神经网络的一些基本函数，例如new、init、adapt和train。第三章以反向传播网络为例讲解了反向传播网络的原理和应用的基本过程。

帮助和安装

神经网络工具箱包含在nnet目录中，键入help nnet可得到帮助主题。

工具箱包含了许多示例。每一个例子讲述了一个问题，展示了用来解决问题的网络并给出了最后的结果。显示向导要讨论的神经网络例子和应用代码可以通过键入help nndemos找到。

安装神经网络工具箱的指令可以在下列两份MATLAB文档中找到：the Installation Guide for MS-Windows and Macintosh 或者the Installation Guide for UNIX。

第二章 神经元模型和网络结构

1. 符号

数学符号

下面给出等式和数字中用到的基本符号：

标量--小写的斜体字.....*a, b, c*

向量--小写加粗的非斜体字.....**a, b, c**

矩阵 - 大写加粗的非斜体字.....**A, B, C**

向量表示一组数字

数学符号和字符的等价

从数学符号到字符的转换或者反过来可以遵循一些规则，为了便于今后引用我们将这些规则列出。为了从数学符号变为MATLAB符号用户需要：

变上标为细胞数组标号

例如 $p^1 \rightarrow p\{1\}$

变下标为圆括号标号

例如 $p_2 \rightarrow p(2)$ 和 $p_2^1 \rightarrow p\{1\}(2)$

变圆括号标号为二维数组标号

例如 $p^1(k-1) \rightarrow p\{1,k-1\}$

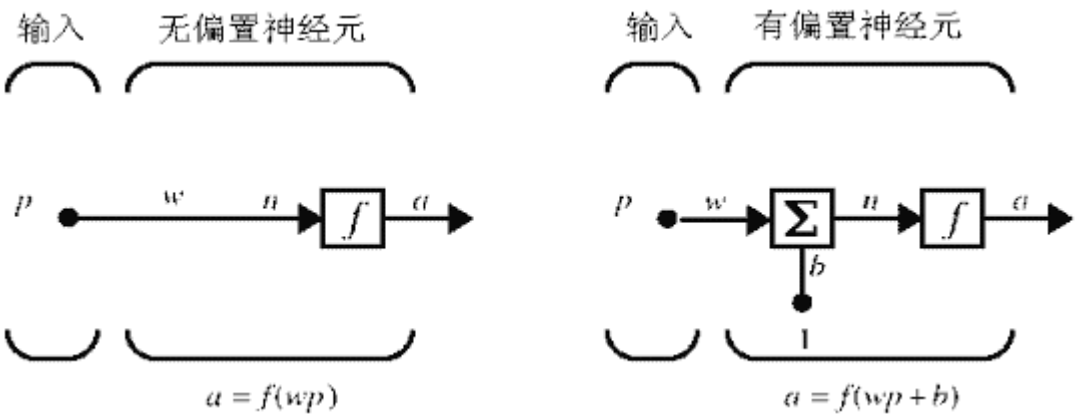
变数学运算符为MATLAB 运算符和工具箱函数

例如 $ab \rightarrow a*b$

2．神经元模型

单神经元

下图所示为一个单标量输入且无偏置的神经元。



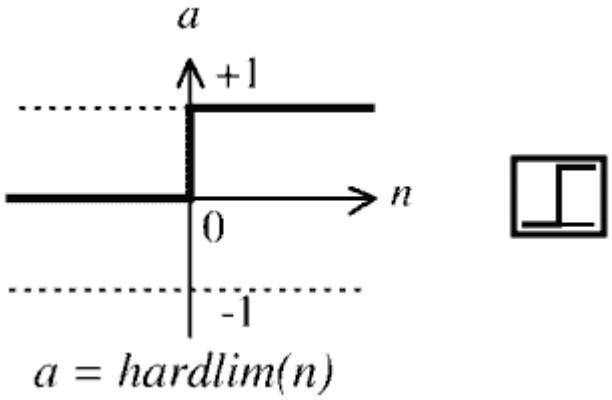
这个输入标量通过乘以权重为标量w的连结点得到结果wp，这仍是一个标量。这里，加权的输入wp仅仅是转移函数f的参数，函数的输入是标量a。右边的神经元有一个标量偏置b，你既可以认为它仅仅是通过求和节点加在结果 wp上，也可以认为它把函数f左移了b个单位，偏置除了有一个固定不变的输入值 1 以外，其他的很像权重。标量n是加权输入wp和偏置b的和，它作为转移函数f的参数。函数f是转移函数，它可以为阶跃函数或者曲线函数，它接收参数n给出输出a，下一节将给出各种不同的转移函数。注意神经元中的w和b都是可调整的标量参数。神经网络的中心思想就是参数的可调整使得网络展示需要和令人感兴趣的行为。这样，我们就可以通过调整权重和偏置参量训练神经网络做一定的工作。或者神经网络自己调整参数以得到想要的结果。

在这个工具箱里所有的神经元都提供偏置，我们的许多例子中都用到了偏置并且假定它在这个工具箱的大多数情况下都要用到。可是，如果你愿意的话，你也可以在一个神经元中省略偏置。

正如上面所提到的，在神经元中，标量b是个可调整的参数。它不是一个输入。可是驱动偏置的常量 1 却是一个输入而且当考虑线性输入向量时一定要这样认为。

转移函数

在这个工具箱里包括了许多转移函数。你能在"Transfer Function Graphs"中找到它们的完全列表。下面列出了三个最常用的函数。



上图所示的阶跃转移函数限制了输出，使得输入参数小于 0 时输出为 0，大于或等于 0 时输出为 1，在第三章中我们将用它来进行分类。

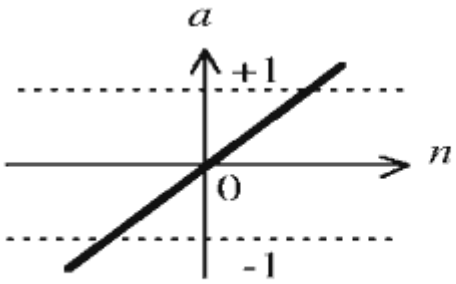
工具箱中有一个函数hardlim来数学上的阶跃，如上图所示。我们可以输入以下代码

```
n = -5:0.1:5;
plot(n,hardlim(n),'c+:');
```

它产生一张在-5 到 5 之间的阶跃函数图。

所有在工具箱中的数学转移函数都能够用同名的函数实现。

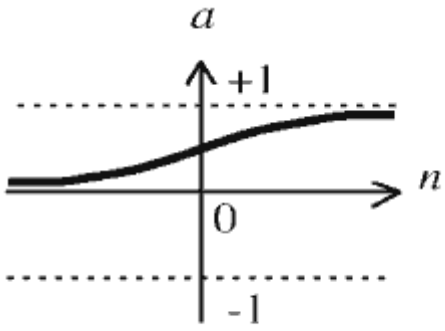
线性转移函数如下图所示



$a = purelin(n)$

这种类型的神经元将在第四章的自适应线性滤波中用作线性拟合。

下图显示的曲线转移函数的输入参数是正负区间的任意值，而将输出值限定于 0 到 1 之间。



$a = logsig(n)$

这种传递函数通常用于反向传播（BP）网络，这得益于函数的可微性。

在上面所示的每一个转移函数图的右边方框中的符号代表了对应的函数，这些图表将替换网络图的方框中的f来表示所使用的特定的转移函数。

第 13 章列出了所有的转移函数和图标。你能够定义自己的传递函数，你可以不限于使用第 13 章所列的转移函数。你能够通过运行示例程序nn2n1 来试验一个神经元和各种转移函数。

带向量输入的神经元

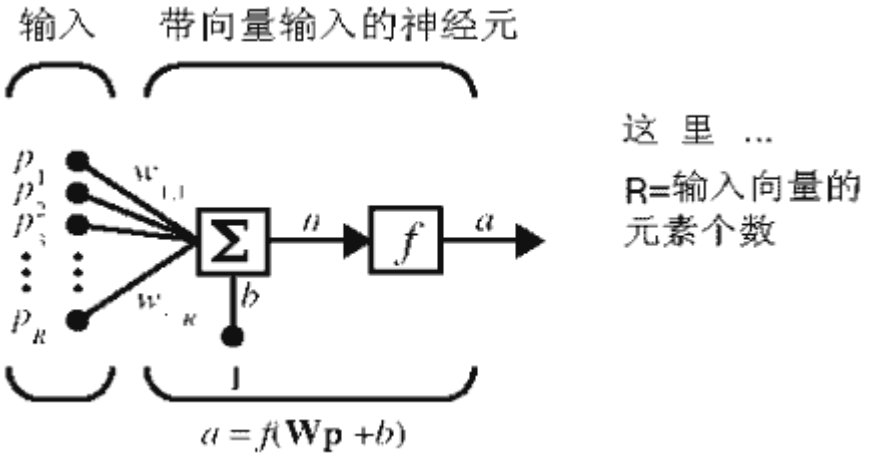
一个有R个元素输入向量的神经元如下图所示。这里单个输入元素

$$p_1, p_2 \cdots p_R$$

乘上权重

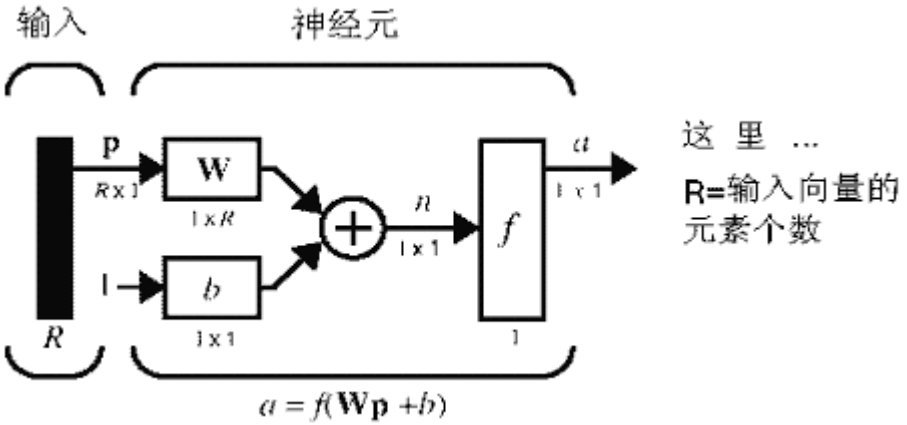
$$w_{1,1}, w_{1,2} \cdots w_{1,R}$$

得到加权值输入求和节点。它们的和是Wp，单行矩阵W和向量p的点乘。



这个神经元有一个偏置b，它加在加权的输入上得到网络输入n，和值n是转移函数f的参数。表达式自然可用MATLAB代码表示为： n =W*p + b

可是，用户很少要写如此底层的代码，因为这些代码已经被建立到函数中来定义和模拟整个网络。上面所示的图包括了许多细节。当我们考虑有许多神经元和可能是许多神经元组成的多层网络时，我们可能会漏掉许多细节。因此，作者设计了一个简洁的符号代表单个神经元。这个符号如下图中所示，它将会在以后的多重神经元电路中用到。

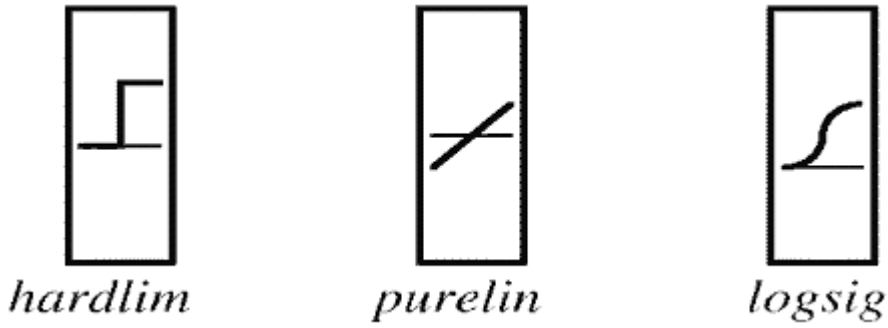


这里输入向量p用左边的黑色实心竖条代表，p的维数写在符号p下面，在图中是Rx1。（注意我们用的是大写字母，正如在以前句子里R用来表示向量大小时一样。）因此，p是一个有R个输入元素的向量。这个输入列向量乘上R列单行矩阵W。和以前一样，常量 1 作为一个输入乘上偏置标量b，给转移函数的网络输入是n，它是偏置与乘积Wp的和。这个和值传给转移函数f得到网络输出a，在这个例子中它是一个标量。注意如果我们有超过一个神经元，网络输出就有可能是一个向量。

上面图中定义了神经网络的一层。一层包括权重的组合，乘法和加法操作（这里就是向量乘积Wp），偏置b和转移函数f。输入数组，即向量p不包括在一层中。

这个简洁的网络符号每一次都会被用到，向量的大小会显示在矩阵变量名字的下面。我们希望这个符号会让你理解神经网络的结构以及与之相关的矩阵数学。

正如前面所讨论的，当特定的转移函数在一张图中被使用时，转移函数将用上面所示的符号代替。下面是几个例子：



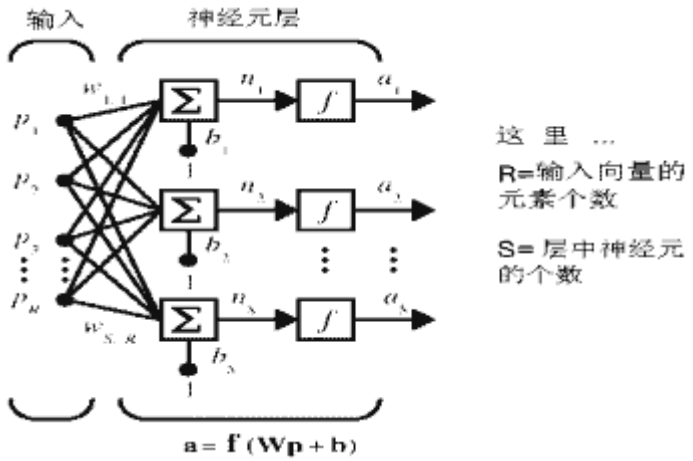
你可以通过运行示例程序nnd2n2 来试验有 2 个元素的神经元。

3．网络结构

两个或更多的上面所示的神经元可以组合成一层，一个典型的网络可包括一层或者多层。我们首先来研究神经元层。

单层神经元网络

有R输入元素和S个神经元组成的单层网络如下图所示：



在一个单层网络中，输入向量p的每一个元素都通过权重矩阵W和每一个神经元连接起来。第I个神经元通过把所有加权的输入和偏置加起来得到它自己的标量输出n(i)。不同的n(i)合起来形成了有S个元素的网络输入向量n。最后，网络层输出一个列向量a，我们在图的底部显示了a的表达式。

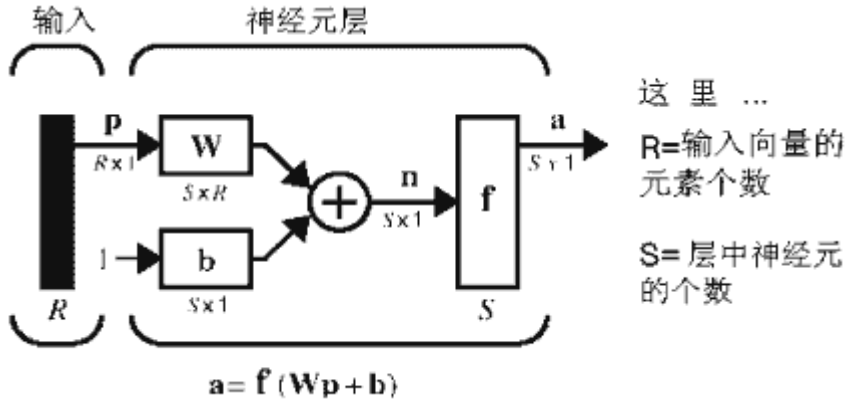
注意输入元素个数R和神经元个数S通常是不等的，我们也并不需要这两者相等。你也可以建立一个简单的复合神经元层，它将上面所示的网络并行的合在一起，使用不同的转移函数。所有的网络都有相同的输入，而每一个网络都会产生输出。

输入向量元素经加权矩阵W作用输入网络。

$$W=\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

注意加权矩阵W的行标标记权重的目的神经元，列标标记待加权的输入标号。因此， 的标号表示从输入信号的第二个元素到第一个神经元的权重是 。有S个神经元和R个输入元素

的神经网络也能够简化成以下符号：



这里，p是一个有R个元素的输入向量，W是一个SxR的矩阵，a和b是有S个元素的向量。如前面所定义的，神经元层包括权重矩阵，乘法运算，偏置向量b，求和符和转移函数框。

输入和层

我们将要讨论多层网络，所以我们需要拓展我们的符号来描述这样的网络。特别是我们要弄清连接输入的权重矩阵和连接层的权重矩阵之间的区别。我们也要分清权重矩阵的目的和源。

我们将把连接输入的权重矩阵成为输入权重，把来自层输出的权重矩阵称为层矩阵。进一步说，我们在各个权重和其他网络元素中将用上标区分源（第二个标号）和目的（第一个标号）。作为示例，我们用简化的形式重画了上面所画的单层多输入网络。

你可以看到，我们把连接输入向量p的权重矩阵标记为输入权重矩阵(IW1,1)，第二个标号 1 是源，第二个标号 1 是目的。同样，第一层的元素，比如偏置、网络输入和输出都有上标 1 来表示它们属于第一层。

在下一章节，我们将用LW表示层权重矩阵，用IW表示输入权重矩阵。

你可以复习以下这一章开始的符号那一节，它把特定的网络net中用数学符号表示的层权重矩阵转换成代码，如下所示：

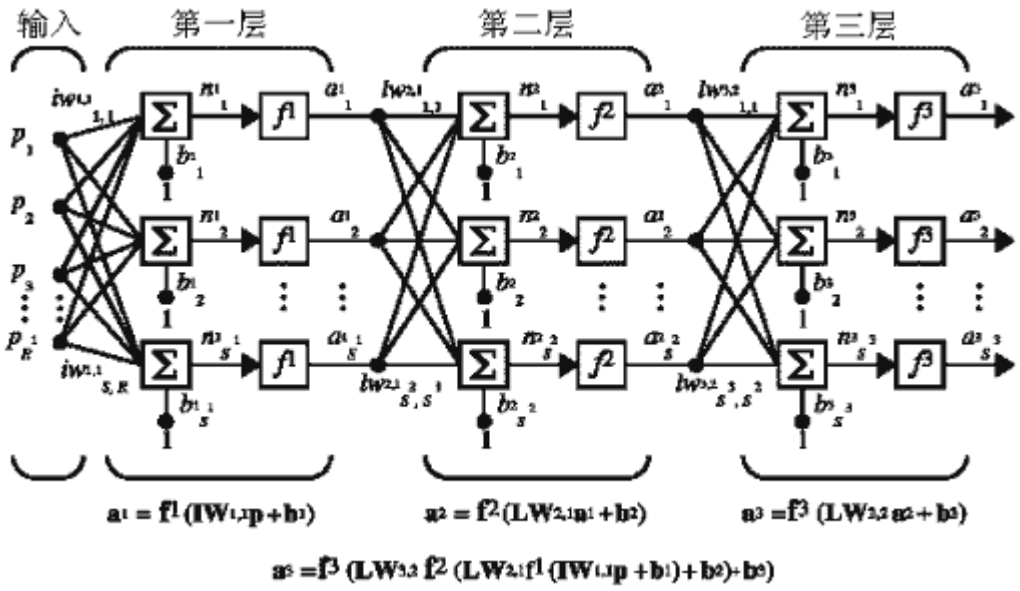
IW1,1 net. IW{1,1}

这样,你就可以写代码来得到对转移函数的网络输入了：

n{1}=net.IW{1,1}*p+net.b{1}

多层神经元网络

一个网络可以有几层，每一层都有权重矩阵 W，偏置向量 b 和输出向量 a。为了区分这些权重矩阵、输出矩阵等等，在图中的每一层，我们都为感兴趣的变量以上标的形式增加了层数。你能够看到在下面所示的三层网络图和等式中使用层符号。

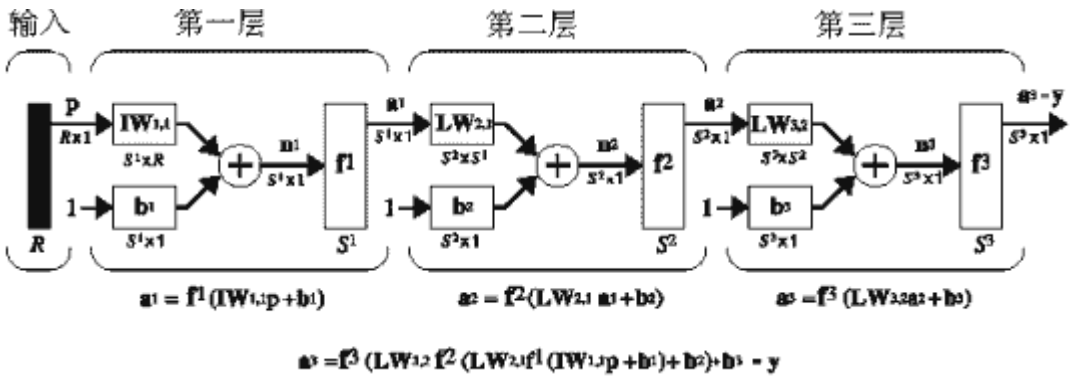


上面所示的网络有 R1 个输入，第一层有 S1 个神经元，第二层有 S2 个神经元，以次类推。一般不同层有不同数量的神经元。每一个神经元的偏置输入是常量 1。

注意中间层的输出就是下一层的输入。第二层可看作有 S1 个输入，S2 个神经元和 S1xS2 阶权重矩阵 W2 的单层网络。第二层的输入是 a1，输出是 a2，现在我们已经确定了第二层的所有向量和矩阵，我们就能把它看成一个单层网络了。其他层也可以照此步骤处理。

多层网络中的层扮演着不同的角色。给出网络输出的层叫做输出层。所有其他的层叫做隐层。上图所示的三层网络有一个输出层（第三层）和两个隐层（第一和第二层）。有些作者把输入作为第四层，这里不用这种指定。

上面所示的三层网络的简洁画法如下图所示：



多层网络的功能非常强大。举个例子，一个两层的网络，第一层的转移函数是曲线函数，第二层的转移函数是线性函数，通过训练，它能够很好的模拟任何有有限断点的函数。这种两层网络集中应用于"反向传播网络"。

注意我们把第三层的输出 a3 标记为 y。我们将使用这种符号来定义这种网络的输出。

4．数据结构

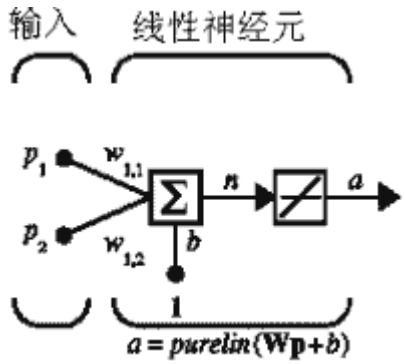
这一节将讨论影响网络仿真的输入数据结构的格式。我们首先讨论静态网络，在讨论动态网络。

我们将关心两种基本的输入向量类型：同步（同时或者无时序）向量和异步向量。对异步向量来说，向量的顺序是非常重要的。对同步向量来说，顺序是不重要的，并且如果我们已经有一定数量的并行网络我们就能把一个输入向量输入到其中的任意网络。

静态网络中的同步输入仿真

仿真静态网络（没有反馈或者延迟）是网络仿真最简单的一种。在这种情况中，我们不需要关心向量输入的时间顺序，所以我们可以认为它是同时发生的。另外，为了是问题更简单，我们假定开始网络仿真仅有一个输入向量。我们用下面的网络作为例子。

为了建立这个网络我们可以用以下命令：



net = newlin([-1 1;-1 1],1);

简单起见我们假定权重矩阵和偏置为：

W=[1，2]，b=[0]

其命令行是：

net.IW{1,1} = [1 2];

net.b{1} = 0;

假定模拟的网络有四个无序向量，即 Q=4：

$$p_1=\begin{bmatrix}1\\2\end{bmatrix}, p_2=\begin{bmatrix}2\\1\end{bmatrix}, p_3=\begin{bmatrix}2\\3\end{bmatrix}, p_4=\begin{bmatrix}3\\1\end{bmatrix}$$

这些同步向量可以用一个矩阵来表示：

P = [1 2 2 3; 2 1 3 1];

现在我们就可以模拟这个网络了：

A = sim(net,P)

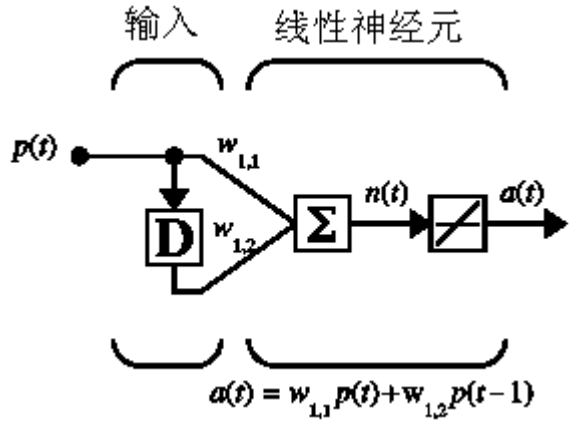
A =

5 4 8 5

我们向网络输入一个简单的同步向量矩阵，得到了一个简单的同步向量输出矩阵。结果不论是由一个网络串行输出还是由四个网络并行输出得到的都是一样的。由于输入并无关联，输入向量的顺序并不重要。

动态网络中的异步输入仿真

当网络中存在延迟时，顺序发生的输入向量就要按一定的序列输入网络。为了演示这种情况，我们用了一个有延迟的简单网络。



为了建立这个网络我们可以用以下命令：

net = newlin([-1 1],1,[0 1]);

net.biasConnect = 0;

假定权重矩阵为：

W=[1，2]

命令行为：

net.IW{1,1} = [1 2];

假定输入顺序为：

p(1)=[1], **p**(2)=[2],**p**(3)=[3], p(4)=[4]

输入序列可以用一个细胞数组来表示：

P = {1 2 3 4};

这样我们就能模拟这个网络了：

A = sim(net,P)

A =

[1] [4] [7] [10]

我们输入一个包含输入序列的细胞数组，网络产生一个包含输出序列的细胞数组。注意异步输入中的输入顺序是很重要的。在这个例子中，当前输出等于当前输入乘 1 加上前一个输入乘 2。如果我们改变输入顺序，那么输出结果也回随之改变。

动态网络中的同步输入仿真

如果我们在上一个例子中把输入作为同步而不是异步应用，我们就会得到完全不同的响应。（虽然我们不清楚为什么要在动态网络中使用这种方式。）这就好象每一个输入都同时加到一个单独的并行网络中。在前一个例子中，如果我们用一组同步输入，我们有：

p1=[1], **p**2=[2],**p**3=[3], **p**4=[4]

这可用下列代码创建：

P =[1 2 3 4];

模拟这个网络，我们得到：

A = sim(net,P)

A =

1 2 3 4

这个结果和我们同时把每一个输入应用到单独的网络中并计算单独的输出没什么两样。注意如果我们没有初始化延迟时间，那么缺省值就是 0。在这个例子中，由于当前输入的权重是 1，输出就是输入乘 1。在某些特定的情况下，我们可能想要在同一时间模拟一些不同序列的网络响应。这种情况我们就要给网络输入一组同步序列。比如说，我们要把下面两个

序列输入网络：

p(1)=[1], **p**(2)=[2],**p**(3)=[3], **p**(4)=[4]

p(1)=[4], **p**(2)=[3],**p**(3)=[2], **p**(4)=[1]

输入 P 应该是一个细胞数组，每一个数组元素都包含了两个同时发生的序列的元素。

P = {[1 4] [2 3] [3 2] [4 1]};

现在我们就可以模拟这个网络了：

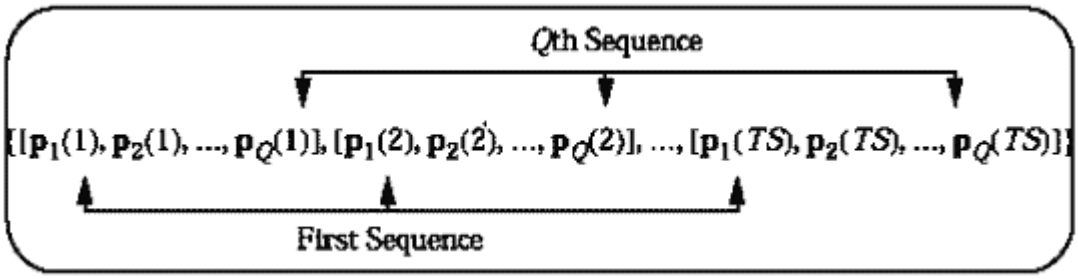
A = sim(net,P);

网络输出结果将是：

A = {[1 4] [4 11] [7 8] [10 5]}

你可以看到，每个矩阵的第一列是由第一组输入序列产生的输出序列，每个矩阵的第二列是由第二组输入序列产生的输出序列。这两组序列之间没有关联，好象他们是同时应用在单个的并行网络上的。

下面的图表显示了当我们有 Q 个 TS 长度的序列时，在函数 sim 中输入 P 的一般格式。它函盖了单输入向量的所有的情况。每一个细胞数组的元素都是一个同步向量矩阵，它对应于每一个序列的同一时间点。如果有多输入向量，那么在细胞数组中的矩阵里就有多行。



这一节我们我们把同步和异步输入应用到了动态网络中。在以前的章节中我们把同步输入应用到了静态网络中。我们也能把异步序列应用到静态网络中。这不会改变网络的输出响应，但是这会影响训练过的网络的形式。在下一节你会更清楚的了解这一点。

5. 训练方式

在这一节中，我们将描述两种不同的训练方式。在增加方式中，每提交一次输入数据，网络权重和偏置都更新一次。在批处理方式中，仅仅当所有的输入数据都被提交以后，网络权重和偏置才被更新。

增加方式（应用与自适应网络和其他网络）

虽然增加方式更普遍的应用于动态网络，比如自适应滤波，但是在静态和动态网络中都可以应用它。

在这一节中我们将示范怎样把增加方式应用到这两种网络中去。

静态网络中的增加方式

继续考虑前面用过的第一个静态网络的例子，我们用增加方式来训练它，这样每提交一次输入数据，网络权重和偏置都更新一次。在这个例子里我们用函数 adapt，并给出输入和目标序列：

假定我们要训练网络建立以下线性函数：

t=2p1+p2

我们以前用的输入是：

$$p_1=\begin{bmatrix}1\\2\end{bmatrix},p_2=\begin{bmatrix}2\\1\end{bmatrix},p_3=\begin{bmatrix}2\\3\end{bmatrix},p_4=\begin{bmatrix}3\\1\end{bmatrix}$$

目标输出是：

t1=[4],t2=[5] ,t3=[7] ,t4=[7]

我们首先用 0 初始化权重和偏置。为了显示增加方式的效果，我们把学习速度也设为 0。

```
net = newlin([-1 1;-1 1],1,0,0);
net.IW{1,1} = [0 0];
net.b{1} = 0;
```

为了用增加方式，我们把输入和目标输出表示为以下序列：

```
P = {[1;2] [2;1] [2;3] [3;1]};
T = {4 5 7 7};
```

前面的讨论中，不论是作为一个同步向量矩阵输入还是作为一个异步向量细胞数组输入，模拟的输出值是一样的。而在训练网络时，这是不对的。当我们使用 adapt 函数时，如果输入是异步向量细胞数组，那么权重将在每一组输入提交的时候更新（就是增加方式），我们将在下一节看到，如果输入是同步向量矩阵，那么权重将只在所有输入提交的时候更新（就是批处理方式）。

我们现在开始用增加方式训练网络：

```
[net,a,e,pf] = adapt(net,P,T);
```

由于学习速度为 0，网络输出仍然为 0，并且权重没有被更新。错误和目标输出相等。

```
a = [0] [0] [0] [0]
e = [4] [5] [7] [7]
```

如果我们设置学习速度为 0.1，我们就能够看到当每一组输入提交时，网络是怎么调整的了。

```
net.inputWeights{1,1}.learnParam.lr=0.1;
net.biases{1,1}.learnParam.lr=0.1;
[net,a,e,pf] = adapt(net,P,T);
a = [0] [2] [6.0] [5.8]
e = [4] [3] [1.0] [1.2]
```

由于在第一个输入数据提交前还没有更新，第一个输出和学习速率为 0 时一样。由于权重已更新，第二个输出就不一样了。每计算一次错误，权重都不断的修改。如果网络可行并且学习速率设置得当，错误将不断的趋向于 0。

动态网络中的增加方式

我们同样也能用增加方式训练动态网络。实际上，这是最普遍的情况。让我们用前面用过的那个有输入延迟的线性网络作为例子，我们将初始化权重为 0，并把学习速率设为 0.1。

```
net = newlin([-1 1],1,[0 1],0.1);
net.IW{1,1} = [0 0];
net.biasConnect = 0;
```

为了用增加方式，我们把输入和目标输出表示为细胞数组的元素：

```
Pi = {1};
P = {2 3 4};
T = {3 5 7};
```

这里我们尝试训练网络把当前输入和前一次输入加起来作为当前输出。输入序列和我们

以前使用 sim 的例子中用过的一样，除了我们指定了输入序列的第一组作为延迟的初始状态。现在我们可以用 adapt 来训练网络了：

[net,a,e,pf] = adapt(net,P,T,Pi);

a = [0] [2.4] [7.98]

e = [3] [2.6] [-1.98]

由于权重没有更新，第一个输出是 0。每一个序列步进，权重都改变一次。

批处理方式

在批处理方式中，仅仅当所有的输入数据都被提交以后，网络权重和偏置才被更新，它也可以应用于静态和动态网络。我们将在这一节讨论这两种类型。

静态网络中的批处理方式

批处理方式可以用adapt或train函数来实现，虽然由于由于采用了更高效的学习算法，train通常是最好的选择。增加方式只能用adapt来实现，train函数只能用于批处理方式。

让我们用前面用过的静态网络的例子开始，学习速率设置为 0.1。

net = newlin([-1 1;-1 1],1,0,0.1);

net.IW{1,1} = [0 0];

net.b{1} = 0;

用adapt函数实现静态网络的批处理方式，输入向量必须用同步向量矩阵的方式放置：

P = [1 2 2 3; 2 1 3 1];

T = [4 5 7 7];

当我们调用adapt时将触发adaptwb函数，这是缺省的线性网络调整函数。learnwh是缺省的权重和偏置学习函数。因此，Widrow-Hoff学习法将会被使用：

[net,a,e,pf] = adapt(net,P,T);

a = 0 0 0 0

e = 4 5 7 7

注意网络的输出全部为 0，因为在所有要训练的数据提交前权重没有被更新，如果我们显示权重，我们就会发现：

>>net.IW{1,1}

ans = 4.9000 4.1000

>>net.b{1}

ans =

2.3000

经过了用adapt函数的批处理方式调整，这就和原来不一样了。

现在用train函数来实现批处理方式。由于Widrow-Hoff规则能够在增加方式和批处理方式中应用，它可以通过adapt和train触发。我们有好几种算法只能用于批处理方式（特别是Levenberg-Marquardt算法），所以这些算法只能用train触发。

网络用相同的方法建立：

net = newlin([-1 1;-1 1],1,0,0.1);

net.IW{1,1} = [0 0];

net.b{1} = 0;

在这种情况下输入向量即能用同步向量矩阵表示也能用异步向量细胞数组表示。用train函数，任何异步向量细胞数组都会转换成同步向量矩阵。这是因为网络是静态的，并且因为train总是在批处理方式中使用。因为MATLAB实现同步模式效率更高，所以只要可能总是采用同步模式处理。

P = [1 2 2 3; 2 1 3 1];

T = [4 5 7 7];

现在我们开始训练网络。由于我们只用了一次adapt，我们这里训练它一次。缺省的线性网络训练函数是trainwb。learnwh是缺省的权重和偏置学习函数。因此，我们应该和前面缺省调整函数是adaptwb的例子得到同样的结果。

net.inputWeights{1,1}.learnParam.lr = 0.1;

net.biases{1}.learnParam.lr = 0.1;

net.trainParam.epochs = 1;

net = train(net,P,T);

经过一次训练后，我们显示权重发现：

>>net.IW{1,1}

ans = 4.9000 4.1000

>>net.b{1}

ans =

2.3000

这和用adapt训练出来的结果是一样的。在静态网络中，adapt函数能够根据输入数据格式的不同应用于增加方式和批处理方式。如果数据用同步向量矩阵方式输入就用批处理方式训练；如果数据用异步方式输入就用增加方式。但这对于train函数行不通，无论输入格式如何，它总是采用批处理方式。

动态网络中的增加方式

训练静态网络相对要简单一些。如果我们用train训练网络，即使输入是异步向量细胞数组，它也是转变成同步向量矩阵而采用批处理方式。如果我们用adapt。输入格式决定着网络训练方式。如果传递的是序列，网络用增加方式，如果传递的是同步向量就采用批处理方式。

在动态网络中，批处理方式只能用train完成，特别是当仅有一个训练序列存在时。为了说明清楚，让我们重新考虑那个带延迟的线性网络。我们把学习速率设为 0.02（当我们采用梯度下降算法时，我们要用比增加方式更小的学习速率，应为所有的分立的梯度都要在决定权重改变步进之前求和）

net = newlin([-1 1],1,[0 1],0.02);

net.IW{1,1}=[0 0];

net.biasConnect=0;

net.trainParam.epochs = 1;

Pi = {1};

P = {2 3 4};

T = {3 5 6};

我们用以前增加方式训练过的那组数据训练，但是这一次我们希望只有在所有数据都提交后才更新权重（批处理方式）。因为输入是一个序列，网络将用异步模式模拟。但是权重将用批处理方式更新。

net=train(net,P,T,Pi);

经过一次训练后，权重值为：

>>net.IW{1,1}

ans = 0.9000 0.6200

这里的权重值和我们用增加方式得到的不同。在增加方式中，通过训练设置，一次训练可以更新权重三次。在批处理方式中，每次训练只能更新一次。

第三章 反向传播网络（BP网络）

1．概述

前面介绍了神经网络的结构和模型，在实际应用中，我们用的最广泛的是反向传播网络（BP网络）。下面就介绍一下BP网络的结构和应用。

BP网络是采用Widrow-Hoff学习算法和非线性可微转移函数的多层网络。一个典型的BP网络采用的是梯度下降算法，也就是Widrow-Hoff算法所规定的。backpropagation就是指的为非线性多层网络计算梯度的方法。现在有许多基本的优化算法，例如变尺度算法和牛顿算法。神经网络工具箱提供了许多这样的算法。这一章我们将讨论使用这些规则和这些算法的优缺点。

一个经过训练的BP网络能够根据输入给出合适的结果，虽然这个输入并没有被训练过。这个特性使得BP网络很适合采用输入/目标对进行训练，而且并不需要把所有可能的输入/目标对都训练过。为了提高网络的适用性，神经网络工具箱提供了两个特性--规则化和早期停止。这两个特性和用途我们将在这一章的后面讨论。这一章还将讨论网络的预处理和后处理技术以提高网络训练效率。

2．基础

网络结构

神经网络的结构前一章已详细讨论过，前馈型BP网络的结构结构和它基本相同，这里就不再详细论述了，这里着重说明以下几点：

1． 常用的前馈型BP网络的转移函数有logsig，tansig，有时也会用到线性函数purelin。当网络的最后一层采用曲线函数时，输出被限制在一个很小的范围内，如果采用线性函数则输出可为任意值。以上三个函数是BP网络中最常用到的函数，但是如果需要的话你也可以创建其他可微的转移函数。

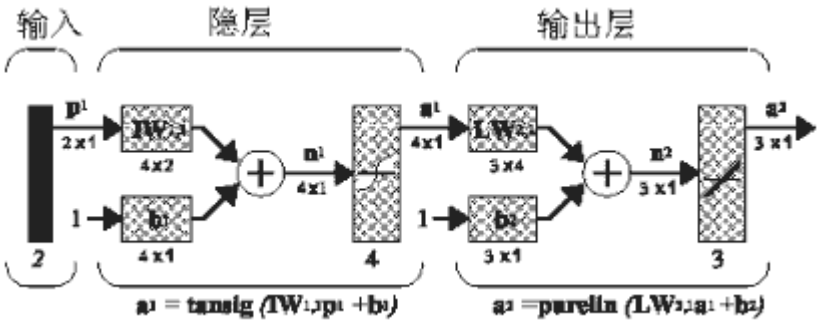
2． 在BP网络中，转移函数可求导是非常重要的，tansig、logsig和purelin都有对应的导函数dtansig、dlogsig和dpurelin。为了得到更多转移函数的导函数，你可以带字符"deriv"的转移函数：

```
tansig('deriv')
ans = dtansig
```

网络构建和初始化

训练前馈网络的第一步是建立网络对象。函数newff建立一个可训练的前馈网络。这需要 4 个输入参数。第一个参数是一个Rx2 的矩阵以定义R个输入向量的最小值和最大值。第二个参数是一个颠顸每层神经元个数的数组。第三个参数是包含每层用到的转移函数名称的细胞数组。最后一个参数是用到的训练函数的名称。

举个例子，下面命令将创建一个二层网络,其网络模型如下图所示。



它的输入是两个元素的向量，第一层有三个神经元，第二层有一个神经元。第一层的转移函数是tan-sigmoid，输出层的转移函数是linear。输入向量的第一个元素的范围是-1 到 2，输入向量的第二个元素的范围是 0 到 5，训练函数是traingd。

net=newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');

这个命令建立了网络对象并且初始化了网络权重和偏置，因此网络就可以进行训练了。我们可能要多次重新初始化权重或者进行自定义的初始化。下面就是初始化的详细步骤。

在训练前馈网络之前，权重和偏置必须被初始化。初始化权重和偏置的工作用命令init来实现。这个函数接收网络对象并初始化权重和偏置后返回网络对象。下面就是网络如何初始化的：

net = init(net);

我们可以通过设定网络参数net.initFcn和net.layer{i}.initFcn这一技巧来初始化一个给定的网络。net. initFcn用来决定整个网络的初始化函数。前馈网络的缺省值为initlay，它允许每一层用单独的初始化函数。设定了net.initFcn ，那么参数net.layer{i}.initFcn 也要设定用来决定每一层的初始化函数。

对前馈网络来说，有两种不同的初始化方式经常被用到：initwb和initnw。initwb函数根据每一层自己的初始化参数(net.inputWeights{i,j}.initFcn)初始化权重矩阵和偏置。前馈网络的初始化权重通常设为rands，它使权重在-1 到 1 之间随机取值。这种方式经常用在转换函数是线性函数时。initnw通常用于转换函数是曲线函数。它根据Nguyen和Widrow[NgWi90]为层产生初始权重和偏置值，使得每层神经元的活动区域能大致平坦的分布在输入空间。它比起单纯的给权重和偏置随机赋值有以下优点：（1）减少神经元的浪费（因为所有神经元的活动区域都在输入空间内）。（2）有更快的训练速度（因为输入空间的每个区域都在活动的神经元范围中）。

初始化函数被newff所调用。因此当网络创建时，它根据缺省的参数自动初始化。init不需要单独的调用。可是我们可能要重新初始化权重和偏置或者进行自定义的初始化。例如，我们用newff创建的网络，它缺省用initnw来初始化第一层。如果我们想要用rands重新初始化第一层的权重和偏置，我们用以下命令：

net.layers{1}.initFcn = 'initwb';

net.inputWeights{1,1}.initFcn = 'rands';

net.biases{1,1}.initFcn = 'rands';

net.biases{2,1}.initFcn = 'rands';

net = init(net);

网络模拟(SIM)

函数sim 模拟一个网络。sim 接收网络输入p，网络对象net，返回网络输出a，这里是simuff用来模拟上面建立的带一个输入向量的网络。

p = [1;2];

a = sim(net,p)

a =

-0.1011

(用这段代码得到的输出是不一样的，这是因为网络初始化是随机的。)

下面调用sim来计算一个同步输入 3 向量网络的输出：

p = [1 3 2;2 4 1];

a=sim(net,p)

a =

-0.1011 -0.2308 0.4955

网络训练

一旦网络加权和偏差被初始化，网络就可以开始训练了。我们能够训练网络来做函数近似（非线性 后退），模式结合，或者模式分类。训练处理需要一套适当的网络操作的例子--网络输入p和目标输出t。在训练期间网络的加权和偏差不断的把网络性能函数net.performFcn减少到最小。前馈网络的缺省性能函数是均方误差mse--网络输出和目标输出t之间的均方误差。这章的余项将描述几个对前馈网络来说不同的训练算法。所有这些算法都用性能函数的梯度来决定怎样把权重调整到最佳。梯度由叫做反向传播的技术决定，它要通过网络实现反向计算。反向传播计算源自使用微积分的链规则。基本的反向传播算法的权重沿着梯度的负方向移动，这将在下一节讲述。以后的章节将讲述更复杂的算法以提高收敛速度。

反向传播算法

反向传播算法中有许多变量，这一章将讨论其中的一些。反向传播学习算法最简单的应用是沿着性能函数最速增加的方向--梯度的负方向更新权重和偏置。这种递归算法可以写成：

xk+1 = xk- a k g k

这里xk是当前权重和偏置向量，g k是当前梯度，a k是学习速率。有两种不同的办法实现梯度下降算法：增加模式和批处理模式。在增加模式中，网络输入每提交一次，梯度计算一次并更新权重。在批处理模式中，当所有的输入都被提交后网络才被更新。下面两节将讨论增加模式和批处理模式。

增加模式训练法（ADAPT）

函数adapt用来训练增加模式的网络，它从训练设置中接受网络对象、网络输入和目标输入，返回训练过的网络对象、用最后的权重和偏置得到的输出和误差。

这里有几个网络参数必须被设置，第一个是net.adaptFcn，它决定使用哪一种增加模式函数，缺省值为adaptwb，这个值允许每一个权重和偏置都指定它自己的函数，这些单个的学习函数由参数net.biases{i,j}.learnFcn 、 net.inputWeights{i,j}.learnFcn 、net.layerWeights{i,j}.learnFcn和Gradient Descent (LEARDGD)来决定。对于基本的梯度最速下降算法，权重和偏置沿着性能函数的梯度的负方向移动。在这种算法中，单个的权重和偏置的学习函数设定为"learngd"。下面的命令演示了怎样设置前面建立的前馈函数参数：

```
net.biases{1,1}.learnFcn = 'learngd';
net.biases{2,1}.learnFcn = 'learngd';
net.layerWeights{2,1}.learnFcn = 'learngd';
net.inputWeights{1,1}.learnFcn = 'learngd';
```

函数learngd有一个相关的参数--学习速率lr。权重和偏置的变化通过梯度的负数乘上学习速率倍数得到。学习速率越大，步进越大。如果学习速率太大算法就会变得不稳定。如果学习速率太小，算法就需要很长的时间才能收敛。当learnFcn设置为learngd时，就为每一个权重和偏置设置了学习速率参数的缺省值，如上面的代码所示，当然你也可以自己按照意愿改变它。下面的代码演示了把层权重的学习速率设置为 0.2。我们也可以为权重和偏置单独的设置学习速率。

```
net.layerWeights{2,1}.learnParam.lr= 0.2;
```

为有序训练设置的最后一个参数是net.adaptParam.passes，它决定在训练过程中训练值重复的次数。这里设置重复次数为 200

```
net.adaptParam.passes = 200;
```

现在我们就可以开始训练网络了。当然我们要指定输入值和目标值如下所示：

```
p = [-1 -1 2 2;0 5 0 5];
```

t = [-1 -1 1 1];

如果我们要在每一次提交输入后都更新权重，那么我们需要将输入矩阵和目标矩阵转变为细胞数组。每一个细胞都是一个输入或者目标向量。

p = num2cell(p,1);

t = num2cell(t,1);

现在就可以用adapt来实现增加方式训练了:

[net,a,e]=adapt(net,p,t);

训练结束以后，我们就可以模拟网络输出来检验训练质量了。

a = sim(net,p)

a =

[-0.9995] [-1.0000] [1.0001] [1.0000]

带动力的梯度下降法(LEARDGDM)

除了learngd以外，还有一种增加方式算法常被用到，它能提供更快的收敛速度--learngdm,带动量的最速下降法。动力允许网络不但根据当前梯度而且还能根据误差曲面最近的趋势响应。就像一个低通滤波器一样，动量允许网络忽略误差曲面的小特性。没有动量，网络又可能在一个局部最小中被卡住。有了动量网络就能够平滑这样的最小。动量能够通过把权重变得与上次权重变化的部分和由算法规则得到的新变化的和相同而加入到网络学习中去。上一次权重变化对动量的影响由一个动量常数来决定，它能够设为 0 到 1 之间的任意值。当动量常数为 0 时，权重变化之根据梯度得到。当动量常数为 1 时新的权重变化等于上次的权重变化，梯度值被忽略了。

Learngdm函数有上面所示的learngd函数触发，除非mc和lr学习参数都被设置了。由于每一个权重和偏置有它自己的学习参数，每一个权重和偏置都可以用不同的参数。

下面的命令将用lerangdm为前面建立的用增加方式训练的网络设置缺省的学习参数：

net.biases{1,1}.learnFcn = 'learngdm';

net.biases{2,1}.learnFcn = 'learngdm';

net.layerWeights{2,1}.learnFcn = 'learngdm';

net.inputWeights{1,1}.learnFcn = 'learngdm';

[net,a,e]=adapt(net,p,t);

批处理训练方式

训练的另一种方式是批处理方式，它由函数train触发。在批处理方式中，当整个训练设置被应用到网络后权重和偏置才被更新。在每一个训练例子中的计算的梯度加在一起来决定权重和偏置的变化。

批处理梯度下降法(TRAINGD)

与增加方式的学习函数learngd等价的函数是traingd，它是批处理形式中标准的最速下降学习函数。权重和偏置沿着性能函数的梯度的负方向更新。如果你希望用批处理最速下降法训练函数，你要设置网络的trainFcn为traingd，并调用train函数。不像以前章节的学习函数，它们要单独设置权重矩阵和偏置向量，这一次给定的网络只有一个学习函数。

Traingd有几个训练参数：epochs,show,goal,time,min_grad,max_fail和lr。这里的学习速率和lerangd的意义是一样的。训练状态将每隔show次显示一次。其他参数决定训练什么时候结束。如果训练次数超过epochs,性能函数低于goal，梯度值低于mingrad或者训练时间超过time，训练就会结束。

下面的代码将重建我们以前的网络，然后用批处理最速下降法训练网络。（注意用批处理方式训练的话所有的输入要设置为矩阵方式）

net=newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');

```
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=train(net,p,t);
TRAINGD, Epoch 0/300, MSE 1.59423/1e-05, Gradient 2.76799/
1e-10
TRAINGD, Epoch 50/300, MSE 0.00236382/1e-05, Gradient
0.0495292/1e-10
TRAINGD, Epoch 100/300, MSE 0.000435947/1e-05, Gradient
0.0161202/1e-10
TRAINGD, Epoch 150/300, MSE 8.68462e-05/1e-05, Gradient
0.00769588/1e-10
TRAINGD, Epoch 200/300, MSE 1.45042e-05/1e-05, Gradient
0.00325667/1e-10
TRAINGD, Epoch 211/300, MSE 9.64816e-06/1e-05, Gradient
0.00266775/1e-10
TRAINGD, Performance goal met.
a = sim(net,p)
a =
-1.0010 -0.9989 1.0018 0.9985
```

用nnd12sd1 来演示批处理最速下降法的性能。

带动量的批处理梯度下降法（TRAINGDM）

带动量的批处理梯度下降法用训练函数traingdm触发。这种算法除了两个例外和learmgdm是一致的。第一．梯度是每一个训练例子中计算的梯度的总和，并且权重和偏置仅仅在训练例子全部提交以后才更新。第二．如果在给定重复次数中新的性能函数超过了以前重复次数中的性能函数的预定义速率max_perf_inc(典型的是 1.04)倍，那么新的权重和偏置就被丢弃，并且动量系数mc就被设为 0。

在下面的代码重，我们重建了以前的网络并用带动量的梯度下降算法重新训练。Traingdm的训练参数和traingd的一样，动量系数mc和性能最大增量max_perf_inc也是如此。（无论什么时候，只要net.trainFcn倍设为traingdm,训练参数就被设为缺省值。）

```
net=newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingdm');
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=train(net,p,t);
TRAINGDM, Epoch 0/300, MSE 3.6913/1e-05, Gradient 4.54729/
1e-10
```

TRAINGDM, Epoch 50/300, MSE 0.00532188/1e-05, Gradient
0.213222/1e-10
TRAINGDM, Epoch 100/300, MSE 6.34868e-05/1e-05, Gradient
0.0409749/1e-10
TRAINGDM, Epoch 114/300, MSE 9.06235e-06/1e-05, Gradient
0.00908756/1e-10
TRAINGDM, Performance goal met.
a = sim(net,p)
a =
-1.0026 -1.0044 0.9969 0.9992

注意，既然我们在训练前重新初始化了权重和偏置，我们就得到了一个和使用traingd不同的均方误差。如果我们想用traingdm重新初始化并且重新训练,我们仍将得到不同的军方误差。初始化权重和偏置的随机选择将影响算法的性能。如果我们希望比较不同算法的性能，我们应该测试每一个使用着的不同的权重和偏值的设置。

用nnd12mo来演示批处理最速下降法的性能。