BP 神经网络的比较与设计

张镕麒 2019081301026 计算机科学与工程学院

摘　要：神经网络是一门高度综合的交叉学科。它的研究和发展涉及神经生理学、数理科学、信息科学和计算机科学等众多领域。以函数逼近功能为例，通过对不同的 BP 神经网络进行仿真。比较不同的 BP 神经网络的性能。以模数转换问题为例，给出了能量函数设计的一般方法。

关键词：BP 神经网络；函数逼近；能量函数；优化计算

0. 引言

人工神经网络（Artificial Neural Networks）被越来越多地应用在工程技术领域，而 BP 神经网络是目前应用最多的神经网络，占整个神经网络应用的 80%左右。BP 神经网络又称反向传播（Back Propagation）网络，这是因为该神经网络的连接权的调整采用的是反向传播的学习算法。BP 神经网络是一种前馈网络，采用最小均方差学习方式。BP 前向网络的主要作用是函数映射，用于模式识别和函数逼近，具体可用于语言综合、识别和自动控制等用途。其主要缺点是收敛速度慢，局部极小。针对这些缺点，人们又提出了 BP 算法的一些改进算法，如加入动量项、学习率自适应、LM 等算法。

优化问题是求解满足一定条件下的目标函数的极小值问题。有关优化的传统算法很多，如梯度法、单纯形法等。由于在某些情况下，约束条件过于复杂。加上变量维数较多等诸多原因，使得采用传统算法进行的优化工作耗时过多，有的甚至达不到预期的优化结果。 Hopfield 能量函数是一个反映了多维神经元整体状态的标量函数。而且可以用简单的电路形成人工神经网络。他们的互联形成了并联计算的机制。当各参数设计合理时，由电路组成的系统状态，可以随时间的变化，最终收敛到渐近稳定点上，并在这些稳定点上使能量函数达到极小值，其系统状态满足了约束条件下的目标函数的极小值，在此方式下。利用人工神经网络来解决优化问题。由于人工神经网络是并行计算的。其计算量不随维数的增加而发生指数性质的爆炸，因而特别适用于解决有此问题的优化问题。

1. BP 神经网络算法分析

由多层感知器模型，输入层中以神经元的输出为输入模式分量的加权和。其余各层中，设某一层中任一神经元 j 输入为 net_j，输出为 y_i，与这一层相邻的低一层中任一神经元 i 的输出为 y_i，则有

$$net_j = \sum_i w_{ji} y_i \qquad y_i = f\left(net_j\right)$$

式中 w_{ji} 为神经元 j 与神经元 i 之间的连接权； $f(\bullet)$ 为神经元的输出函数，我们取其为 sigmoid 函数，即

$$y_j = f\left(net_j\right) = \frac{1}{1+e^{-\left(net_j + h_j\right)/\theta_0}} \tag{1}$$

式中 h_j 为神经元 j 的阈值，它影响输出函数的水平方向位置；θ_0 用来修改输出函数的形状。

设输出层第 k 个神经元的实际输出为 y_k，输入为 net_k，与输出相邻的隐层中第 j 个神经元的输出为 y_j。此时有

$$net_k = \sum_j w_{kj} y_j \tag{2}$$

$$y_k = f\left(net_k\right) \tag{3}$$

对于一个输入模式 X_p，若输出层中第 k 个神经元的期望输出为 O_{pk}，实际输出为 y_{pk}，则输出层的输出方差为

$$E_p = \frac{1}{2}\sum_k \left(O_{pk} - y_{pk}\right)^2 \tag{4}$$

若输入 N 个模式，则网络的系统均方差为

$$E = \frac{1}{2N}\sum_p \left(\sum_k \left(O_{pk} - y_{pk}\right)^2 \right) = \frac{1}{N}\sum_p E_p$$

权值 w_{ji} 的修正应使 E_p 最小。因此 w_{ki} 应沿 E_p 的负梯度方向变化。也就是说，当输入 X_p 时， w_{kj} 的修正量 $\Delta_p w_{kj}$ 应与 $\left(-\partial E_p / \partial w_{kj}\right)$ 成正比，即

$$\Delta_p w_{kj} = -\eta \frac{\partial E_p}{\partial w_{kj}} = -\eta \frac{\partial E_p}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}}$$

由(2)式得

$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}}\sum_j w_{kj} y_{pj}$$

令 $\delta_{pk} = -\partial E_p / \partial net_k$，由式(3)和式(4)得

$$\delta_{pk} = -\frac{\partial E_p}{\partial y_{pk}}\frac{\partial y_{pk}}{\partial net_k} = \left(O_{pk} - y_{pk}\right) f'\left(net_k\right)$$

由式(1)和式(2)得

$$f'(net_k)=\frac{\partial}{\partial net_k}\left(\frac{1}{1+e^{-(net_j+h_j)/\theta_0}}\right)=y_{pk}(1-y_{pk}) \tag{5}$$

$$\Delta_p w_{kj}=\eta\delta_{pk}y_{pj}$$

对于与输出层相邻的隐层中神经元 j 和比改隐层低一层的神经元 i，权值 w_{ij} 的修正量仍应为

$$\Delta_p w_{kj}=-\eta\frac{\partial E_p}{\partial w_{kj}}=-\frac{\partial E_p}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}}=-\frac{\partial E_p}{\partial net_k}=\eta\delta_{pk}y_{pj}$$

$$\delta_{pj}=-\frac{\partial E_p}{\partial net_j}=-\frac{\partial E_p}{\partial y_{pj}}\frac{\partial y_{pj}}{\partial net_j}=-\frac{\partial E_p}{\partial y_{pj}}f'(net_k)=-\frac{\partial E_p}{\partial y_{pj}}y_{pk}(1-y_{pk})$$

式中 $-\partial E_p/\partial y_{ij}$ 不能直接计算，可以根据其他已知量计算。具体算法为

$$-\frac{\partial E_p}{\partial y_{ij}}=-\sum_k\frac{\partial E_p}{\partial net_k}\frac{\partial net_k}{\partial y_{ij}}=\sum_k\left(-\frac{\partial E_p}{\partial net_k}\right)\frac{\partial}{\partial y_{ij}}\sum_m w_{km}y_{pm}=\sum_k\left(-\frac{\partial E_p}{\partial net_k}\right)w_{kj}=\sum_k\delta_{pk}w_{kj}$$

因此得到

$$\delta_{pj}=y_{pj}(1-y_{pj})\sum_k\delta_{pk}w_{kj} \tag{6}$$

如式(5)和式(6)所示，输出层中神经元的输出误差反向传播到前面各层，以各层之间的权值进行修正。

2.MATLAB 中 BP 神经网络的对比

2.1 MATLAB 中典型的 BP 神经网络算法

(1)trainrp：弹性 BP 算法。它只利用导数的符号表示权更新的方向，而不考虑导数的大小。因此可以消除偏导数的大小对权值的不利影响。这种算法具有收敛速度优势，且其占用内存小的优点。

(2)traingdx：自适应学习速率法。自适应学习速率法检查权重的修正值是否真正降低了误差函数。如果确实如此，则说明选取的学习速率有上升的空间，可以对其增加一个量。如若不是，那么就应减小学习速率的值。因此学习速率可以根据误差性能函数进行调节，能够解决标准 BP 算法中学习速率选择不当的问题。

(3)traincgf：共轭梯度法。这是采用 Fletcher-Reeves 算法的共轭梯度算法。这种方法收敛速度比普通的梯度下降法要快很多。这种方法需要线性搜索，存储量的要求很大，对收敛速度来讲，不同的搜索对象速度不同。共轭梯度算法的计算代价比较低，在较大规模问题中十分有用。

(4)trainbfg：拟牛顿算法。权值根据公式 $X=X+a*dX$ 来修改，dX 是搜索的方向，a 用来沿着搜索方向最小性能函数。最初的搜索方向沿着梯度的负方向，在之后的迭代中按照 $dX=-H-1*gX$ 来修改，H 为近似 Hessian 矩阵。

Trainbfg 算法需要的迭代次数比较少，但由于要每步都要存储 Hessian 矩阵，所以单步计算量和存储量都很大，比较适合小型网络。

(5)trainlm:Levenberg-Marquardt 优化算法，权值根据 $dX=-\left(jX^T*jX+I*\mu\right)^{-1}jX^T*E$ 进行修正，其中 jX 为误差对权值微分的 Jacobian 矩阵，E 为误差向量，μ 为调整量。该方法学习速度很快，但占用内存很大，对于中等规模的网络来说，是最好的一种训练算法。对于大型网络，可以通过置参数 mem-reduct 将 Jacobian 矩阵分为几个子矩阵。这样可以减少内存的消耗，但学习时间会增大。

2.2 BP 神经网络拟合函数的比较

以拟合函数为例，选取 $y=\sin(\frac{1}{2}\pi x)+\sin(\pi x)$ 为函数对象，来测试不同 BP 神经网络的性能表现。通过对几种典型的 BP 神经网络进行训练和仿真比较得到表 1 所示数据（代码请见附录 6.1）。从中我们可以看出 LM 神经网络能达到预定误差，且比较其他神经网络其训练速度相当的快。其仿真结果、训练曲线和原始曲线如图 1 至图 5 所示。通过比较可得，前三种神经网络存在一定的误差，仿真曲线与原曲线有一定的失真。而拟牛顿神经网络和 LM 神经网络都有很好的逼近效果。其训练误差能够达到期望误差，训练后曲线与原始曲线十分相似，几乎重合。

表 1 几种典型 BP 神经网络算法仿真结果比较

训练函数	BP 算法	训练次数	均方误差
Trainrp	弹性 BP 算法	2000	0.0427956
Traingdx	自适应学习速率算法	2000	0.0267222
Traincgf	共轭梯度法	625	0.00270386
Trainbfg	拟牛顿算法	527	0.00000997647
Trainlm	Levenberg-Marquardt 算法	67	0.00000951896

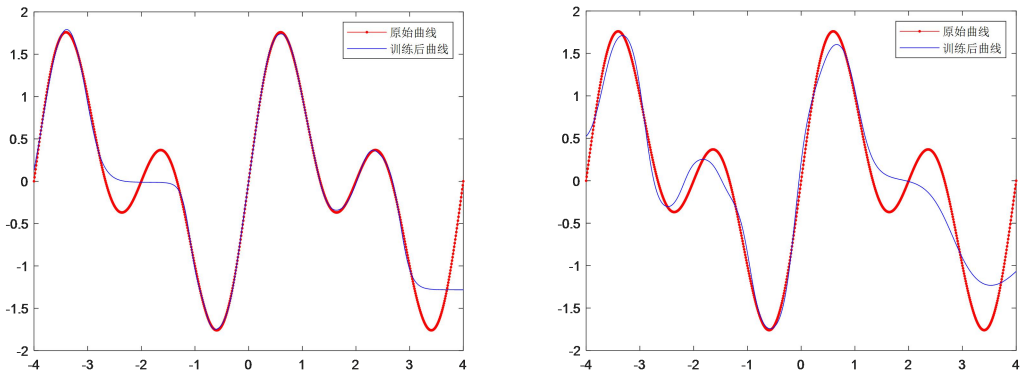
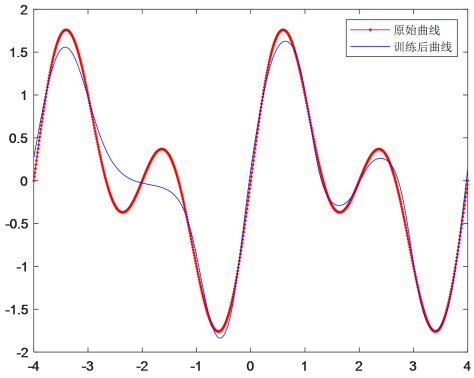


图 1、2 弹性 BP 神经网络(trainrp)和自适应学习速率神经网络(traingdx)仿真效果图



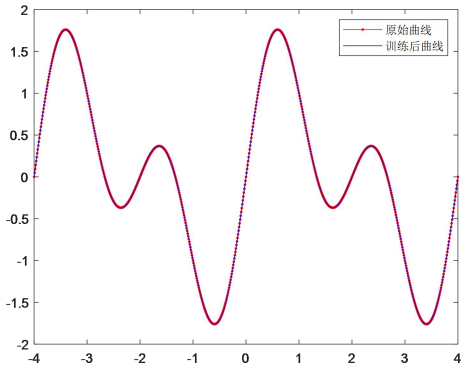
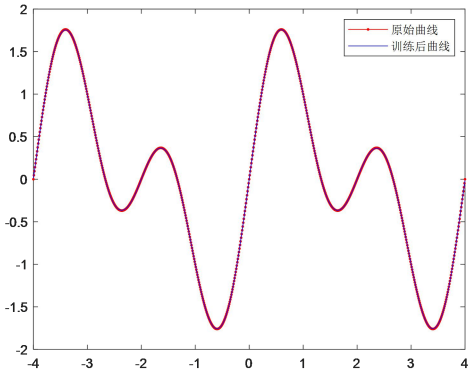


图 3、4 共轭梯度神经网络(traincgf)和拟牛顿法神经网络(trainbfg)仿真效果图



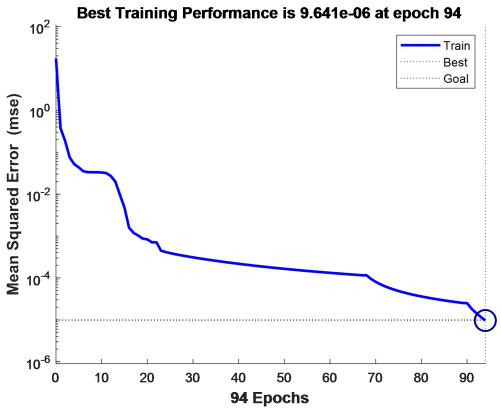


图 5、6 LM 神经网络(trainlm)仿真效果图和 LM 神经网络均方误差曲线

由表 1 的仿真结果可以看出，当误差要求比较高时，弹性 BP 算法和自适应学习速率法需要很长的训练时间，在设定的训练步骤范围内没能达到期望误差。从仿真图可以看出仿真曲线有一定的失真。然而弹性 BP 算法不需要进行搜索，需要内存比较小，因此在一些大型网络问题中比较适用。而在一些需要较慢收敛速度的情况下可以选择使用自适应速率法。对于中型网络，拟牛顿算法收敛效果仅次于 LM 算法，且其需要的内存相对比较小，在实际应用中可以尝试选择。从图 6、图 7 可以看出，对于简单的小型对象，LM 算法是最好、最快的算法。仿真效果相当的好，是做仿真的首要选择。当然公轭剃度算法和拟牛顿算法也都有很好的仿真效果。但是，LM 算法对于复杂对象需要很大的内存，对于内存不充分大的设备，LM 算法并不是最佳的选择。拟牛顿算法需要存储 Hessian 矩阵，对于复杂的大型网络对象也不适用。因此。在时间充足情况下，对于复杂网络公轭梯度法和弹性 BP 是较好的选择。

3 能量函数设计

3.1 能量函数设计的一般方法

设优化目标函数为 $f(u)$，$u \in R^n$ 为人工神经网络的状态，也是目标函数中的变量。优化中的约束条件为：$g(u)=0$。优化问题归结为：在满足约束的条件下，

使目标函数最优。由于可以设计出等价最小的能量函数 E 为：

$$E = f(u) + \sum |g_i(u)|$$

这里 $|g_i(u)|$ 也称为惩罚函数，因为在约束条件 $g_i(u)=0$ 不能满足时，$\sum |g_i(u)|$ 的值总是大于 0，造成 E 不是最小。

对于目标函数 $f(u)$，一般总是取一个期望值与实际值之间之差的平方或绝对值的标量函数，这样能够保证 $f(u)$ 总是大于零。根据 Hopfield 能量函数和梯度下降法的要求，只有在 E 在负的方向上有界，即 $|E| < E_{\max}$，同时 $\frac{dE}{dt} \le 0$，则系统最后总能达到 E 的最小值且 $\frac{dE}{dt}=0$ 的点，此点同时又是系统的稳定点，即 $\frac{du_i}{dt}=0$ 的点。由于求解优化问题的 E 往往是状态 u 的函数，所以，为了求解方便，常常将 $\frac{dE}{dt} \le 0$ 的条件转化为对状态求导的条件如下：

$$\frac{\partial E(u_i, v_i)}{\partial u_i} = -\frac{du_i}{dt}$$

这是因为，当 $\frac{du_i}{dt} = -\frac{\partial E}{\partial u_i}$，

$$\frac{dE}{dt} = \sum_i \frac{\partial E}{\partial u_i}\frac{du_i}{dt} = -\sum_i \left(\frac{du_i}{dt}\right)^2 \le 0$$

所以梯度下降法可以确保 E 总是下降，直到抵达某一局部最小。

同理可得另一个等价的条件为：

$$\frac{\partial E(u_i, v_i)}{\partial v_i} = -\frac{dv_i}{dt}$$

所以在用 Hopfield 能量函数求解优化问题时，首先应把问题转化为目标函数和约束条件，然后构造出能量函数，并利用条件式求出能量函数中的参数，由此得到人工神经网络的连接权值。

3.2 具体设计步骤

(1) 根据要求的目标函数，写出能量函数的第一项 $f(u)$；

(2) 根据约束条件 $g(u)=0$，写出惩罚函数，使其在满足约束条件时为最小，作为能量函数的第二项；

加上一项；$\sum_i \frac{1}{R}\int_0^{v_i} F^{-1}(\eta)d\eta$，此项是人为加上的，因为在神经元状态方程中，

存在一项$-\frac{u_i}{R}$，它是在人工神经网络的电路实现中产生的。为了使设计出的优化结果能够在电路中得以实现而加上此项。它是一个正值函数，在运行放大增益足够大时，此项可以忽略。

(3) 根据能量函数 E 求出状态方程，并使下式成立：

$$\frac{\partial E(u_i, v_i)}{du_i} = -\frac{du_i}{dt}$$

(4) 根据条件与参数之间的关系，求出W_{ij}和b_i，$(i = 1,2,...n)$；

(5) 求出对应的电路参数，并进行模拟电路的实现。

3.3 实例应用

用人工神经网络来设计一个 4 位 A/D 变换器，要求将一个连续的从 0 到 15 的模拟量 u 变化为输出为 0 或 1 的二进制数字量，即$v_i \in \{0,1\}$，v_i代表第i个神经元的输出：$v_i = F(u)$，F为单调上升有限量函数。

转换器的实质是对于给定的模拟量输入，寻找一个二进制数字量输出，使输出值与输入模拟量之间的差最小。传统的转换方式，只要对模拟输入量用展转相除，记录余数，就可以得到二进制的变换值。采用人工神经网络进行转换，首先需定义能量函数。一个四位 A / D 转换器可以用具有四个输出结点的人工神经网络来实现。

假定神经元的输出电压可在 0 与 1 之间连续变化，当网络达到稳态时，各节点的输出为 0 或 1，若此时的输出状态所表示的二进制与模拟输入量相等，则表明此人工神经网络达到了变换器的功能。输入与输出之间的关系则满足下式：

$$\sum_{i=0}^{3} v_i 2^i = u$$

由此可见，要是 A/D 转换器结果$v_3 v_2 v_1 v_0$为输入u的最佳数字表示，必须满足两点：

每个输入v_i必须趋于 0 值或 1 值，至少比较接近这两个数，$v_3 v_2 v_1 v_0$的值应尽量接近值u，为此，利用最小方差的概念，对输出v_i按下列指标来选取$f(u)$：

$$f(u) = \frac{1}{2}\left(u - \sum_{i=0}^{3} v_i 2^i\right)^2 > 0$$

上式中，u为输入的模拟量，v_i为数字量，此目标函数大于零，所以$f(u)$存在极小值，且当$f(u) = f_{\max}(u)$时，v_i为u的正确转换。

将$f(u)$展开，并整理后得：

$$f(u)=\frac{1}{2}\sum_{i=0}^{3}\sum_{j=0}^{3}\left(2^{i+j}v_iv_j\right)-\sum_{i=0}^{3}2^iuv_i+\frac{1}{2}u$$

仅用一项$f(u)$并不能保证v_i的值充分接近逻辑值 0 或 1，因为可能存在其他v_i值使$f(u)$为最小，为此增加一个约束条件：

$$g(u)=-\frac{1}{2}\sum_{i=0}^{3}\left(2^i\right)^2(v_i-1)v_i$$

$g(u)$保证了输出只有在{0,1}时取最小值零，而对于v_i为 0 到 1 之间的实数时，$g(u)\neq 0$，因此约束条件保证了输出的数字量 0 或 1.

综上所述，我们得出了一般步骤：

(1)写出能量函数E

$$E=f(u)+g(u)+\sum_{i=0}^{3}\frac{1}{R_i}\int_0^{v_i}F^{-1}(\eta)d\eta$$

$$=\frac{1}{2}\left(u-\sum_{i=0}^{3}v_i2^i\right)^2-\frac{1}{2}\sum_{i=0}^{3}\left(2^i\right)^2(v_i-1)v_i+\sum_{i=0}^{3}\frac{1}{R_i}\int_0^{v_i}F^{-1}(\eta)d\eta$$

(2) 计算$\frac{du_i}{dt}$

因为

$$\frac{du_i}{dt}=F(u_i)\frac{du_i}{dt}$$

而$F(u_i)>0$，在放大区内可以近似为一个常数C，即

$$F(u_i)=C$$

所以有

$$\frac{\partial E}{\partial v_i}=-\frac{dv_i}{dt}=-C\frac{du_i}{dt}$$

而

$$\frac{\partial E}{\partial v_i}=\sum_{\substack{j=0\\j\neq i}}^{3}2^{i+j}v_i-2^iu+2^{2-1}+\frac{u_i}{R}=-C\frac{du_i}{dt}$$

重写上式得

$$C\frac{du_i}{dt}=-\frac{u_i}{R}+\sum_{\substack{j=0\\j\neq i}}^{3}\left(-2^{i+j}\right)v_j+\left(-2^{2i-1}+2^iu\right)$$

(3)将上式与实现电路的状态方程组相比较可得：

$$W_{ij}=\begin{cases}-2^{i+j} & i\neq j\\ 0 & i=j\end{cases};\quad I_i=-2^{2i-1}+2^i u$$

根据上面的设计，可以设计出完成优化求解的模拟电路人工神经网络系统。权的负值是通过倒相运算放大器来完成的，所得到的负输出再用一次倒相变正。从这个例子可以看出，优化问题的求解是设法将问题转化为能量函数的构造。一旦对应的能量函数构造成功，人工神经网络也就设计出来，最优解就可以随之解出来。

4 结束语

本文通过对 matlab 中几种不同神经网络学习算法的比较，以函数逼近功能为例，通过对不同的 BP 神经网络进行仿真，找到了各种算法的优劣。

基于 Hopfield 神经网络的 A/D 转换器实质上是一种非线性优化映射器，与传统的 A/D 转换器比较，它具有转换速度高，电路结构简单、可具智能优点。虽然在它的学习算法中，可能出现多个局部极小点，但随着神经网络学习理论的进一步研究，相信基于神经网络的 A/D 转换器必将得到推广和应用。

5 参考文献

[1]李孝安.神经网络与神经计算机导论[M].西安：西北工业大学出版社,1994.
[2]程相军.神经网络原理及其应用[M].北京：国防工业出版社,1995.
[3]薄春,孙政顺,赵世敏.MATLAB 神经网络工具箱 BP 算法比较[J].计算机仿真,2006,23(5):1006-9348;(2006)05-0142-03.
[4]姬巧玲,漆为民,蔡维由,程远楚.MATLAB7.0 中改进 BP 神经网络的实现[J].电脑开发与应用,1003-5850(2005),07-0021-03.
[5]陈伟,马如雄,郝艳红.基于 MATLAB 的人工神经网络设计[J].电脑学习,2005,(2).
[6]沈世镒.神经网络系统理论及其应用[M].北京:科学出版社,2000.
[7]阎平凡，张长水．人工神经网络与模拟进化计算[M]．北京:清华大学出版社，2002.
[8]王士同,陈剑夫．问题求解的人工智能神经网络方法[M]．北京：气象出版社．1999.
[9]袁曾任.人工神经网络及其应用[M].北京:清华大学出版社,1999.
[10]徐俊波等.反馈神经网络进展[J].化工自动化及仪表,2003,30(I):6-10.
[11]飞思科技产品研发中心.MATLAB6.5 辅助神经网络分析与设计[M].北京:电子工业出版社,2004.

6 附件

6.1 几种典型 BP 神经网络算法 MATLAB 实现（仅给出 trainlm 一例）

```
x=[-4:0.01:4];
y=sin(0.5*pi*x)+sin(pi*x);
net=newff(minmax(x),[1,15,1],{'tansig','tansig','purelin'},'trainlm');
net.trainparam.epochs=2000;
net.trainparam.goal=1e-5;
```

```
net=train(net,x,y);
y2=sim(net,x);
plot(x,y,'r.-',x,y2,'b');
legend('原始曲线','训练后曲线');
```