文章编号:1001-506X(2010)02-0386-06

基于分布式并行计算的神经网络算法

张代远[1,2]

(1. 南京邮电大学计算机学院, 江苏 南京 210003;
(2. 南京邮电大学计算机技术研究所, 江苏 南京 210003)

摘 要: 为了提高计算性能(速度与可扩展性),提出了一种新颖的神经网络的并行计算体系结构和计算网络权函数的训练算法。权函数是广义 Chebyshev 多项式和线性函数的复合函数,只需要通过代数计算就可以求得,不需要梯度下降计算或者矩阵计算。各个权函数能够独立求解,可以通过并行系统采用并行算法计算。算法可以求得全局最优点,得到反映网络误差的一个有用的表达式。此外,算法在不超过权函数总数的范围内,还具有维持加速比与并行系统中提供的处理器的数量成线性增长的能力。仿真实验结果表明,本文算法的计算性能远远优于传统算法。

关键词: 神经网络; 并行计算; 权函数; Chebyshev 多项式; 可扩展性
中图分类号: TP 183; TP 301 **文献标志码**: A

Training algorithm for neural networks based on distributed parallel calculation

ZHANG Dai-yuan[1,2]

(*1. Coll. of Computer, Nanjing Univ. of Posts and Telecommunications, Nanjing 210003, China*;
(*2. Inst. of Computer Technology, Nanjing Univ. of Posts and Telecommunications, Nanjing 210003, China*)

Abstract: To improve computing performance (speed and scalability), an innovative parallel computation architecture and a training algorithm for neural networks are proposed. Each weight function is a composite function of a generalized Chebyshev polynomial and a linear function, only algebraic calculation is needed, and no requirement is involved for the calculation such as the steepest descent-like algorithms or matrix calculation. The weight functions are found independently, therefore they are calculated by using a parallel algorithm in a parallel system. The algorithm is used to find the global minimum. A useful expression is obtained for the approximate error of the networks. The scalability of the algorithm is described as the ability to maintain linear proportion of speedup to the number of processors available in the parallel system within the total number of weight functions. The results show that the computing performance proposed is much better than that obtained with traditional methods.

Keywords: neural networks; parallel computation; weight functions; Chebyshev polynomials; scalability

0 引 言

传统算法如误差反向传播(error back propagation, BP)算法、径向基函数(radical basis function, RBF)算法、支持向量机(support vector machine, SVM)算法都存在以下共同的缺点:各个权值不能独立求解(至少不能彼此完全独立地求解),因此难以应用并行计算技术;计算复杂,采用最优化方法(如 BP 算法常采用梯度下降类算法),或者要进行矩阵计算(如 RBF 算法和 SVM 算法的计算过程),因此其计算工作量大,速度慢。不仅如此,传统算法在寻求全局最优解以及泛化能力的分析方面也都遇到了很多困难。

一些学者从不同的角度进行了研究。例如分层优化算法[1],这是一类罚函数方法,它从整体上考虑对权值的优化,难以并行化,也没有摆脱梯度下降思想的束缚。二阶算法[2]利用了二阶导数的信息,要计算 Hessian 矩阵,其计算复杂,而且工作量大,同样难以并行化,仍然是一类优化算

收稿日期:2008-12-03; **修回日期**:2009-05-09。

作者简介:张代远(1957-),男,教授,主要研究方向为神经网络、蚁群算法、计算机体系结构和并行计算。E-mail:dyzhang@njupt.edu.cn

法。文献[3]引入了样条函数神经网络的概念，但是需要进行矩阵计算。SVM 算法[4]同样需要求解线性方程组，也要进行优化计算。一般而言，求线性方程组的时间复杂度大约是 $O(N^3)$，要花费较多的计算时间，而梯度下降类算法难以避免局部极小、收敛速度慢等困难。这些传统算法的并行度都比较差，因此需要寻找一种能克服以上缺点的方法，本文的理论和方法正是在这样的背景之下提出的。

本文提出了由广义 Chebyshev 多项式构成的神经网络及其训练算法。与传统算法的很大不同之处在于本文提出的算法训练的是神经网络的权函数，每个权函数是以与它相连的输入结点为自变量的一元函数，而不是传统算法得到的常数，它是广义 Chebyshev 多项式和线性变换函数的复合函数。各个权函数可以并行求解，每个权函数的计算非常简单(无须像传统算法那样，需要计算矩阵或采用梯度下降类的最优化数学方法)，只需要简单的代数计算，其时间复杂度为 $O(N)$，因此本文算法速度很快，远远快于传统算法。另一个很重要的方面是网络的各个权函数彼此独立，可以并行计算，这进一步增加了提高运算速地的潜力，对于大规模问题的求解或实时性要求高的应用场合提供了理论依据。此外，本文提出的理论和方法能够求得全局最优解，并且具有很好的泛化能力。另外，本文提出了分布式并行计算的体系结构，可以实现本文提出的理论与方法，同时具备很好的可扩展性。最后给出了仿真实例，很好地验证了本文理论结果的正确性。

1　能够独立计算权函数的神经网络拓扑结构

首先需要寻找这样一种网络结构，使其能够独立训练网络的权值，以便并行系统能够有效工作。本文提出的理论需要利用被训练问题的 Chebyshev 多项式的 0 点(也称为 Chebyshev 结点)处的目标值。显然，需要选择的样本点就是 Chebyshev 结点和对应的目标值。以下为了讨论方便，将 Chebyshev 结点和对应的目标值称为 Chebyshev 样本点或者简称为 Chebyshev 样本。

神经网络的具体结构如下：输入层直接和神经元相连，假设每一个输入样本向量是 m 维，则输入端有 m 个结点，其第 i 维对应的输入变量记为 $x_i(i=1,2,\cdots,m)$。每个输入端结点通过连接权前馈连接到所有神经元的输入端。不失一般性，首先假设每一个输出样本向量是 1 维，则输出端有 1 个结点，它将神经元的运算结果直接输出，如图 1 所示。神经网络的一般结构如图 2 所示，为了简单，图 2 中省略了 x_i 与 y_i 之间的变换函数。显然，在图 2 所示的网络结构中，其每一个输出结点与图 1 的结构相同，因此只要将图 1 所示的网络研究清楚，其结果就可以直接应用到图 2 所示的网络中来。下面将只研究图 1 所示的网络。

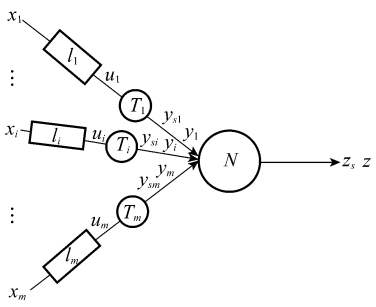

图 1　输入层有 m 个结点、输出层有 1 个结点的神经网络结构

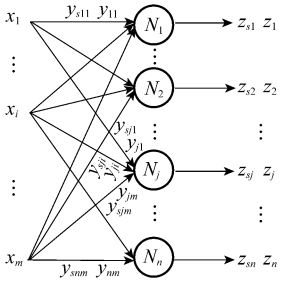

图 2　输入层有 m 个结点、输出层有 n 个结点的神经网络结构

在图 1 中，z 表示输出层神经元所对应的目标值，z_s 表示输出层神经元所对应的网络的实际输出，y_{si} 表示权函数的实际输出，y_i 表示目标输出与一个常数因子的乘积(见后面的讨论)，圆圈中的 N 表示神经元，与传统方法不同，它由加法器构成。各个量之间的关系如下

$$u_i = l_i(x_i) \tag{1}$$

$$y_{si} = T_i(u_i) \tag{2}$$

$$z_s = \sum_{i=1}^{m} y_{si} \tag{3}$$

在式(1)和式(2)中，$l_i(x_i)$ 是线性变换函数，$T_i(u_i)$ 是广义 Chebyshev 多项式。选择广义 Chebyshev 多项式的原因，是为了利用它的优良的逼近能力和计算的简单性，见后面的论述。

为了能将神经元 N 的 m 个输入量求和之后能够得到事先给定的目标样本，这里引入如下的分配系数 η_i，使得

$$y_i = \eta_i z \tag{4}$$

式中

$$\sum_{i=1}^{m} \eta_i = 1,\ 0 \leqslant \eta_i \leqslant 1 \tag{5}$$

因此，根据式(3)～式(5)可知，若每一个 y_{si} 都能够有效逼近 y_i，则 z_s 就能够很好地逼近 z。

对于某一个结点(见图 1)，y_{si} 是 x_i 的一元函数，称为权函数，它是广义 Chebyshev 多项式和一个线性函数的复合函数。显然，本文提出的权函数与传统算法的常数权完全不同。

2 广义 Chebyshev 多项式权函数神经网络的训练算法原理

由于求得每一个权函数的过程是相似的，为了以下数学推导的方便，将图 1 中反映输入结点的下标去掉。

式(1)的线性变换函数(去掉下标)是

$$u = l(x) = \frac{1}{b-a}(2x-a-b) \tag{6}$$

式(6)的意义是将区间 $x\in[a,b]$变换到 $u\in[-1,1]$，以便于使用 Chebyshev 多项式。

选择向量系

$$\begin{cases} \boldsymbol{T}_0 = (T_0(\bar{u}_1) \quad T_0(\bar{u}_2) \quad \cdots \quad T_0(\bar{u}_\beta))^{\mathrm{T}} = (1 \quad 1 \quad \cdots \quad 1)^{\mathrm{T}} \\ \boldsymbol{T}_1 = (T_1(\bar{u}_1) \quad T_1(\bar{u}_2) \quad \cdots \quad T_1(\bar{u}_\beta))^{\mathrm{T}} \\ \cdots \\ \boldsymbol{T}_{\beta-1} = (T_{\beta-1}(\bar{u}_1) \quad T_{\beta-1}(\bar{u}_2) \quad \cdots \quad T_{\beta-1}(\bar{u}_\beta))^{\mathrm{T}} \end{cases} \tag{7}$$

式中，$T_k(u)$为 k 次 Chebyshev 多项式，其定义为

$$\begin{cases} T_k(u) = \cos(k\arccos(u)) \\ T_0(u) = 1 \end{cases}, \quad k = 1,2,\cdots,\beta-1 \tag{8}$$

由式(7)的向量系张成的空间记为

$$\boldsymbol{T}_\beta = \mathrm{span}\{\boldsymbol{T}_0, \boldsymbol{T}_1, \cdots, \boldsymbol{T}_{\beta-1}\} \tag{9}$$

在离散点集上，β 次 Chebyshev 多项式 $T_\beta(u) = \cos(\beta\arccos(u))$，在区间$[-1,1]$上有 β 个 0 点

$$\bar{u}_\alpha = \cos\frac{(2\alpha-1)\pi}{2\beta}, \quad \alpha = 1,2,\cdots,\beta \tag{10}$$

在这 β 个 0 点上，可以证明

$$\langle \boldsymbol{T}_i, \boldsymbol{T}_j \rangle = \sum_{\alpha=1}^{\beta} T_i(\bar{u}_\alpha) T_j(\bar{u}_\alpha) = \begin{cases} 0, & i \neq j \\ \beta/2, & i = j \neq 0 \\ \beta, & i = j = 0 \end{cases} \tag{11}$$

于是，向量系(7)满足

$$\langle \boldsymbol{T}_i, \boldsymbol{T}_j \rangle = 0, \quad i \neq j \tag{12}$$

也就是说，向量系(7)关于点集(见式(10))为一正交函数系。令

$$\bar{\boldsymbol{u}} = \{\bar{u}_\alpha\}_{\alpha=1}^{\beta} \tag{13}$$

假设与式(13)的 $\bar{u}_\alpha$ 相对应的目标值为 $\bar{y}_\alpha(\alpha=1,2,\cdots,\beta)$，即

$$\bar{\boldsymbol{y}} = \{\bar{y}_\alpha\}_{\alpha=1}^{\beta} = \{\eta\bar{z}_\alpha\}_{\alpha=1}^{\beta} \tag{14}$$

由式(13)和式(14)构成的插值点称为 Chebyshev 样本，记为

$$\overline{Ip} = \{(\bar{u}_\alpha, \bar{y}_\alpha)\}_{\alpha=1}^{\beta} = \{(\bar{u}_\alpha, \eta\bar{z}_\alpha)\}_{\alpha=1}^{\beta} \tag{15}$$

此时，法方程组有如下形式

$$\boldsymbol{T}\boldsymbol{a}^* = \boldsymbol{b} \tag{16}$$

式中

$$\boldsymbol{T} = \mathrm{diag}\left(\sum_{p=1}^{\beta} T_0^2(\bar{u}_p) \quad \sum_{p=1}^{\beta} T_1^2(\bar{u}_p) \quad \cdots \quad \sum_{p=1}^{\beta} T_{\beta-1}^2(\bar{u}_p)\right)$$

$$\boldsymbol{a}^* = (a_0^* \quad a_1^* \quad \cdots \quad a_{\beta-1}^*)^{\mathrm{T}}$$

$$\boldsymbol{b} = \left(\sum_{p=1}^{\beta} \bar{y}_p T_0(\bar{u}_p) \quad \sum_{p=1}^{\beta} \bar{y}_p T_1(\bar{u}_p) \quad \cdots \quad \sum_{p=1}^{\beta} \bar{y}_p T_{\beta-1}(\bar{u}_p)\right)^{\mathrm{T}}$$

根据式(11)和式(16)可以求得

$$a_k^* = \frac{\langle \boldsymbol{T}_k, \boldsymbol{T}_k \rangle}{\langle \boldsymbol{T}_k, \boldsymbol{T}_k \rangle} = \begin{cases} a_k^* = \dfrac{2}{\beta}\sum_{p=1}^{\beta} \bar{y}_p T_k(\bar{u}_p) = \dfrac{2}{\beta}\sum_{p=1}^{\beta} \eta\bar{z}_p T_k(\bar{u}_p) \\ a_0^* = \dfrac{1}{\beta}\sum_{p=1}^{\beta} \bar{y}_p T_0(\bar{u}_p) = \dfrac{1}{\beta}\sum_{p=1}^{\beta} \eta\bar{z}_p T_0(\bar{u}_p) \end{cases} \tag{17}$$

于是得到最佳平方逼近意义下的广义 Chebyshev 多项式为

$$\varphi^*(u) = T^*(u) = \sum_{k=0}^{\beta-1} a_k^* T_k(u) \tag{18}$$

可见，本文提出的理论和方法主要是计算式(17)，它是代数计算，其时间复杂度为 $O(\beta)$，因此速度很快。有些传统算法(如 RBF 算法、SVM 算法)需要求解线性方程组，求解线性方程组的时间复杂度为 $O(\beta^\alpha)$，$2<\alpha<3$，不仅如此，通常还需要使用梯度下降类算法进行优化计算，因此其时间复杂度比 $O(\beta^\alpha)$更差。所以，本文算法的运算速度要远远快于传统算法。另一方面，式(17)的每一个 a_k^* 都是独立求得的，因而易于实现并行计算，可以进一步加快运算速度。另外，不存在传统算法(如 BP 等算法)中经常遇到的局部极小、不收敛或收敛速度慢的问题，也不存在线性方程组求解中的矩阵计算问题。

以下定理 1 说明本文算法能求得全局最优点。

定理 1 对于 Chebyshev 样本，以下表达式成立

$$\bar{y}_\alpha = \varphi^*(\bar{u}_\alpha) = T^*(\bar{u}_\alpha) = \sum_{k=0}^{\beta-1} a_k^* T_k(\bar{u}_\alpha)$$

证明 由于 $\bar{\boldsymbol{y}} = \{\bar{y}_\alpha\}_{\alpha=1}^{\beta}$是 β 维向量，则它可以由向量系张成的 β 维空间 $\boldsymbol{T}_\beta = \mathrm{span}\{\boldsymbol{T}_0, \boldsymbol{T}_1, \cdots, \boldsymbol{T}_{\beta-1}\}$的基底线性表示，也就是说 $\bar{\boldsymbol{y}}\in\boldsymbol{T}_\beta$，或者说，向量 $\bar{\boldsymbol{y}}$ 到空间 $\boldsymbol{T}_\beta$ 的距离为 0。因此，对应的最佳平方逼近意义下的最佳逼近值应该等于 0，即

$$\bar{y}_\alpha - \sum_{k=0}^{\beta-1} a_k^* \boldsymbol{T}_k(\bar{u}_\alpha) = 0 \tag{19}$$

3 网络的泛化能力分析

泛化定义如下[3]。

定义 1 (ε-泛化能力) 若一个给定的集合为 D，D_p 是样本集合，D_t 是测试集合，它们之间有如下的关系，$D=D_p\cup D_t$，$D_p\cap D_t=\varnothing$，$\varnothing$ 表示空集。对于一个训练后的网络，假设当网络的输入为 $\boldsymbol{x}$ 时，网络的实际输出为 $\boldsymbol{z}_s$，而此时期望的网络目标输出为 $\boldsymbol{z}$。对给定的 $\varepsilon>0$，若以下表达式成立

$$E = \begin{cases} EP = \|\boldsymbol{z}-\boldsymbol{z}_s\|_2^2 = 0, & \boldsymbol{x}\in D_p \\ ET = \|\boldsymbol{z}-\boldsymbol{z}_s\|_2^2 \leqslant \varepsilon, & \boldsymbol{x}\in D_t \end{cases} \tag{20}$$

则称该网络是关于样本(x,z)在范数 $\|\ \|_2^2$ 意义下，具有误差不超过 ε 的泛化能力，或简称为具有 ε-泛化能力。

为了讨论方便，引入以下函数

$$\delta_i = \delta_i(u_i) = y_i - y_{si} = \eta_i z(u_i) - T^*(u_i) \quad (21)$$

$$\Delta = \Delta(u_1, u_2, \cdots, u_m) = \boldsymbol{z} - \boldsymbol{z}_s = \sum_{i=1}^{m}(\eta_i z(u_i) - T^*(u_i)) = \sum_{i=1}^{m}\delta_i(u_i) \quad (22)$$

引理 1　对于图 1 所示的网络,有

$$ET = \| \Delta \|_2^2 \leqslant m^2 \cdot \max_i \| \delta_i(u_i) \|_2^2 \quad (23)$$

证明　显然有

$$ET = \| \Delta \|_2^2 = \langle \Delta, \Delta \rangle = \sum_{i=1}^{m}\langle \delta_i, \delta_i \rangle + 2\sum_{\substack{i=1 \\ i<k}}^{m}\langle \delta_i, \delta_k \rangle$$

利用 Cauchy-Schwarz 不等式,有

$$ET = \| \Delta \|_2^2 \leqslant \sum_{i=1}^{m} \| \delta_i \|_2^2 + 2\sum_{\substack{i=1 \\ i<k}}^{m}(\| \delta_i \|_2 \cdot \| \delta_k \|_2) = \left(\sum_{i=1}^{m} \| \delta_i \|_2 \right)^2 \leqslant m^2 \cdot \max_i \| \delta_i(u_i) \|_2^2 \quad (24)$$

证毕

假设 $z(u_i) \in C[-1,1]$(符号 $C[-1,1]$ 表示在区间 $u_i \in [-1,1]$ 上所有连续函数的集合),则显然有 $\eta_i z(u_i) \in C[-1,1]$。

定理 2　设 $z(u_i) \in C[-1,1]$,且 $z(u_i)$ 的 β 阶导数在 $u_i \in (-1,1)$ 存在且连续,则对于任意给定的 $\varepsilon > 0$,图 1 所示的网络经过训练后具备 ε-泛化能力,即能够满足式(20)。

证明　由于广义 Chebyshev 多项式(18)是 $\beta-1$ 次多项式,根据定理 1 知道,广义 Chebyshev 多项式(18)精确通过 Chebyshev 样本。由式(10)知道,β 个 Chebyshev 多项式的 0 点是彼此不同的,因此通过这 β 个插值点的多项式是唯一的,其余项为

$$\delta_i(u_i) = \eta_i z(u_i) - T^*(u_i) = \frac{\eta_i}{\beta!} z^{(\beta)}(\xi_i)\omega_\beta(u_i) \quad (25)$$

式中

$$\omega_\beta(u_i) = (u_i - \bar{u}_1)(u_i - \bar{u}_2)\cdots(u_i - \bar{u}_\beta) \quad (26)$$

$$\min\{u_i, \bar{u}_1, \bar{u}_2, \cdots, \bar{u}_\beta\} < \xi_i < \max\{u_i, \bar{u}_1, \bar{u}_2, \cdots, \bar{u}_\beta\} \quad (27)$$

令

$$M_i = \max_{-1 \leqslant u_i \leqslant 1} | z^{(\beta)}(u_i) | \quad (28)$$

将式(26)和式(28)代入式(25),得到

$$\| \eta_i z(u_i) - T^*(u_i) \|_\infty = \max_{-1 \leqslant u_i \leqslant 1} \left| \frac{\eta_i}{\beta!} z^{(\beta)}(\xi_i)\omega_\beta(u_i) \right| \leqslant \frac{\eta_i M_i}{\beta!} \max_{-1 \leqslant u_i \leqslant 1} | \omega_\beta(u_i) | \quad (29)$$

显然,为了获得最好的泛化能力,希望有

$$\min_{\bar{u}_1, \bar{u}_2, \cdots, \bar{u}_\beta} \max_{-1 \leqslant u_i \leqslant 1} | \omega_\beta(u_i) | \quad (30)$$

式(30)正是函数 $\omega_\beta(u_i)$ 在区间 $[-1,1]$ 上的 0 偏差最小问题。根据 Chebyshev 理论知道,当 $\bar{\boldsymbol{u}} = \{\bar{u}_\alpha\}_{\alpha=1}^{\beta}$ 为 β 次 Chebyshev 多项式的 β 个 0 点时,式(30)成立。因此,只要选择函数 $\omega_\beta(u_i)$ 为最高次幂的系数为 1 的 β 次 Chebyshev 多项式就可以了,即

$$\omega_\beta(u_i) = \frac{1}{2^{\beta-1}} T_\beta(u_i) \quad (31)$$

于是,余项的估计值为

$$\| \delta_i(u_i) \|_\infty = \max_{-1 \leqslant u_i \leqslant 1} \left| \frac{\eta_i}{\beta!} z^{(\beta)}(\xi_i)\omega_\beta(u_i) \right| \leqslant \frac{\eta_i M_i}{\beta!} \frac{1}{2^{\beta-1}} \cdot \max_{-1 \leqslant u_i \leqslant 1} | T_\beta(u_i) | = \frac{\eta_i M_i}{2^{\beta-1}\beta!} \quad (32)$$

根据式(23),利用两种范数之间的关系式 $\| \delta_i(u_i) \|_2 \leqslant C_i \cdot \| \delta_i(u_i) \|_\infty$($C_i$ 是一个正常数),可以得到网络测试代价函数的估计式为

$$ET = \| \Delta \|_2^2 \leqslant m^2 \cdot \max_i \| \delta_i(u_i) \|_2^2 \leqslant m^2 \cdot \max_i (C_i^2 \| \delta_i(u_i) \|_\infty^2) \leqslant \max_i \left(\frac{\eta_i m C_i M_i}{2^{\beta-1}\beta!} \right)^2 \quad (33)$$

对于每一个 i,式(33)右端的分子是常数,所以对于任意给定的 $\varepsilon > 0$,必然存在一个正整数 γ,使得当 $\beta > \gamma$ 时,式(33)的右端小于 ε。

定理 1 和定理 2 是非常重要的。这两个定理说明本文提出的神经网络结构和算法不仅可以精确回想起训练(学习)过的样本,而且还具有 ε-泛化能力。这些优点是传统算法所不具备的。

4　分布式并行计算体系结构

显然,能够进行并行计算的前提是除了系统可以提供多个能够彼此完全独立工作的处理器以外,还必须将待求解的问题本身分解成彼此不相关的若干个独立计算部分。传统算法(如 BP、RBF、SVM 等算法)通常不能将各个权独立分开(至少不能完全独立分开),因此很难并行计算,至少不能充分并行计算。根据前面的理论可知,在本文算法中,初始化后(即得到 Chebyshev 样本后),就可以独立计算各个广义 Chebyshev 多项式,因此可以充分并行计算,这正是本文提出分布式并行计算体系结构的前提。

本文算法主要是计算式(17)、式(10)和式(8)(事实上式(10)和式(8)可以事先计算好存储在本地存储器中,故真正要计算的只有式(17)),它是代数计算,因此速度很快。

为了进一步加快计算速度,本文引入了并行计算机制。从图 2(或图 1)可以看出,只要输入输出的样本给定,就可以根据式(17),独立计算各个广义 Chebyshev 多项式,因此各个权函数可以并行计算(以下简称为本文的并行算法)。采用并行计算技术可以进一步节约计算时间,基于以上考虑,本文提出了分布式并行计算体系结构,如图 3 所示。其工作过程如下:P_i 表示第 i 个处理器($i=1,2,\cdots,p$);M_i 表示处理器 P_i 的本地存储器,用于存储原始样本数据。初始化后(即得到 Chebyshev 样本后),每个处理器 P_i 就可以独立计算 Chebyshev 多项式,计算后的结果通过互联网络传送到对应的 T_{ji},神经网络就可以利用得到的结果进行工作。

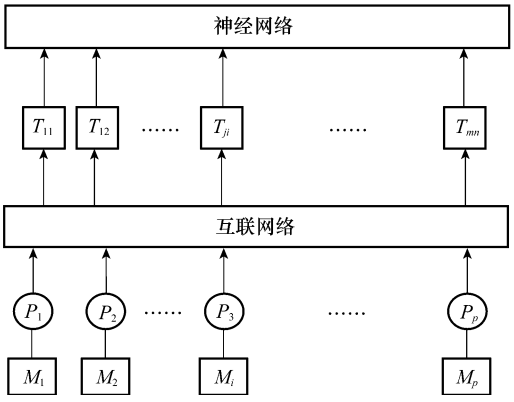

图 3 分布式并行计算体系结构

加速比是描述并行系统的重要指标。加速比定义为串行算法所使用的时间 T_s 除以并行算法所使用的时间 T_p，当不考虑通信开销时，加速比为

$$S=\frac{T_s}{T_p}=\frac{mn}{\lceil mn/p\rceil} \tag{34}$$

当考虑通信开销时，加速比会比式(34)计算的结果略小一些，这取决于本地存储器的访问速度和互联网络的结构和通信质量。

互联网络的拓扑结构一直是国际上的研究热点，因为它直接影响通信开销。人们已提出了多种互联网络拓扑结构，其中超立方体是最流行、使用最广泛的互联网络拓扑结构之一。由于它并不是各方面拓扑性质最好的互联网络，人们开展了交叉立方体网络的研究。文献[5]首次提出了交叉立方体结构，文献[6]给出了一种新的交叉立方体最短路径路由算法。给出适合于本文算法的互联网络的工作将占据较大的篇幅，不属于本文的研究重点，作者将另文专门介绍。

对于并行系统，一个最重要的性能是可扩展性。可扩展性目前没有统一的定义[7]，文献[7]给出了可扩展性的一个定义如下：

定义 2 若一个并行算法在 $1\leqslant p\leqslant p^*$ 内，其时间复杂度为 $O(T(N)/p)$，则称该算法在 $1\leqslant p\leqslant p^*$ 范围内是可扩展的。也就是说，在 $1\leqslant p\leqslant p^*$ 范围内，可以获得线性加速比。这里 N 是问题的规模，p 是处理器的个数，$T(N)$ 是最佳串行算法的时间复杂度。

根据这个定义知道，p^* 越大越好。从式(34)可以看出，至少从理论上讲，本文算法具有很好的可扩展性。事实上，对于本文算法，$p^*=mn$；RBF 算法的可扩展性要差一些，其 $p^*=n$；BP 算法由于其权值不能分开独立计算，其可扩展性最差($p^*=1$)，属于不可扩展算法。因此本文算法要比传统算法优越得多，即在不超过权函数的总数范围内，维持加速比与并行系统中提供的处理器的数量成线性增长的关系。

5 数值仿真实验

以下实验是在 Matlab 下完成，其中传统算法使用了 Matlab 的神经网络工具箱中提供的一些库函数。

实验 1 计算效率实验。假设样本由以下的函数生成

$$\begin{cases}x_1=t\\ x_2=t\\ z=z(t)\end{cases},\ t\in[-1,1] \tag{35}$$

式中

$$z=x_1^2+x_2^2+\cos(x_1+x_2)+\mathrm{e}^{-0.5(x_1+x_2)} \tag{36}$$

首先来看采用本文的串行算法与传统算法的时间效率的比较(结果见表 1)。BP、RBF、SVM 算法的精度取 0.1，隐层节点数取 m。

由于传统算法(例如 BP 算法)，无法实现并行计算，为了使得比较能够说明问题，故首先根据式(17)，串行计算各个广义 Chebyshev 多项式(即串行使用本文的方法去求各个权函数，以下简称为本文的串行算法)，并将其与传统算法进行比较，结果见表 1。由表 1 可以看出，即使采用本文的串行算法，也较传统算法优越得多。

表 1 本文算法和传统算法的时间效率比较 s

时间效率比较	本文算法	BP 算法	RBF 算法	SVM 算法
$m=2$, $n=2$	0.2	21.3	16.3	11.2
$m=2$, $n=4$	0.3	44.1	37.1	23.7
$m=2$, $n=8$	0.4	98.2	77.2	50.0
$m=2$, $n=16$	0.6	221.3	179.3	116.9
$m=2$, $n=32$	1.0	639.0	411.1	279.6

为了看出并行计算的优越性，按照式(34)可以估计出若采用本文的并行算法所获得的加速比，结果见表 2。表 2 中的数值是采用本文的并行算法与采用本文的串行算法相比所能够获得的计算速度的增益的最大倍数。

另外，表 2 中的数据也说明本文的并行算法具有很好的可扩展性。

表 2 本文的并行算法的加速比

加速比 S	$p=2$	$p=4$	$p=6$	$p=8$
$m=2$, $n=2$	2.00	4.00	4.00	4.00
$m=2$, $n=4$	2.00	4.00	4.00	8.00
$m=2$, $n=8$	2.00	4.00	5.33	6.00
$m=2$, $n=16$	2.00	4.00	5.33	6.00
$m=2$, $n=32$	2.00	4.00	5.82	6.00

实验 2 泛化能力实验。为了描述本文算法和 BP、RBF、SVM 算法的泛化能力的比较，表 3 给出了式(35)和式(36)的 Chebyshev 样本误差和测试误差。测试误差是在[−1,1]中随机选 20 个测试点，求出每个点的测试误差，然后再取平均值。在这个例子中，网络的结构参数为 $m=2$，$n=1$，这里假设 $\eta_1=\eta_2=\eta=0.5$。

表 3　本文算法和传统算法的泛化能力比较

泛化能力比较	本文算法	BP 算法	RBF 算法	SVM 算法
Chebyshev 样本误差	1.28×10^{-26}	1.31×10^{-1}	1.01×10^{-1}	5.40×10^{-2}
测试误差	3.35×10^{-5}	2.51×10^{-1}	1.37×10^{-1}	1.02×10^{-1}

实验 1 和实验 2 验证了本文算法的计算效率远远优于传统算法,不仅如此,本文算法能充分利用系统提供的处理器的个数,所以具有很好的可扩展性。另外,本文算法得到的网络测试代价函数值是很小的,这说明本文算法的泛化能力很强,远远优于 BP、RBF、SVM 算法。这些结果都归因于本文所奠定的理论基础。若实验样本不是 Chebyshev 样本,则可以通过文献[8]的方法求得 Chebyshev 样本。

6　结束语

本文提出了一种新颖的神经网络的并行计算体系结构和训练算法,理论和实验结果都表明,此算法的计算性能远远优于传统算法,主要表现在:

(1) 速度快。引入了分布式并行计算机制,各个权函数能够独立求解,因此可以通过并行系统采用并行算法计算。算法简单,主要工作是计算网络的权函数,权函数是广义 Chebyshev 多项式和线性函数的复合函数,仅仅通过代数计算就可以求得,不需要传统算法的梯度下降计算或者矩阵计算,有利于在一些功能简单的计算机或嵌入式系统上实现。

(2) 可扩展性好。即在不超过权函数的总数范围内,维持加速比与并行系统中提供的处理器的数量成线性增长的能力。

(3) 泛化能力强。本文算法具有 ε-泛化能力(见定理 2)。本文得到了反映网络误差的一个有用的表达式。

(4) 可以很容易求得全局最优点,本文提出的神经网络结构和算法可以精确回想起训练(学习)过的样本(见定理 1),这是许多传统算法所不具备的。

参考文献:

[1] Ergezinger S, Tomsen E. An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer[J]. *IEEE Trans. on Neural Networks*, 1995, 6(1): 31 - 42.

[2] Ampazis N, Perantonis S J. Two highly efficient second order algorithms for training feedforward networks[J]. *IEEE Trans. on Neural Networks*, 2002, 13(5): 1064 - 1073.

[3] 张代远. 神经网络新理论与方法[M]. 北京:清华大学出版社, 2006.

[4] Cortes C, Vapnic V. Support vector networks[J]. *Machine Learning*, 1995, 20: 273 - 297.

[5] Efe K. The crossed cub architecture for parallel computing[J]. *IEEE Trans. on Parallel and Distributed Systems*, 1992, 3(5): 513 - 524.

[6] 喻昕, 吴敏, 王国军. 一种新的交叉立方体最短路径路由算法[J]. 计算机学报, 2007, 30(4): 615 - 621.

[7] Li Keqin. Analysis of parallel algorithms for matrix chain product and matrix powers on distributed memory systems[J]. *IEEE Trans. on Parallel and Distributed Systems*, 2007, 18(7): 865 - 878.

[8] 张代远. 基于广义 Чебышев 多项式的新型神经网络算法[J]. 系统工程与电子技术, 2008, 30(11): 2274 - 2279. (Zhang Daiyuan. New algorithm for training neural networks based on generalized Чебышев polynomials[J]. *Systems Engineering and Electronics*, 2008, 30(11): 2274 - 2279.)