中国计量学院
本科毕业设计（论文）

神经网络中的分类问题

Classification Probelm in Neural Network

学生姓名 钟威 学号 0800903133

学生专业 信息与计算科学 班级 09 信算(2)班

二级学院 理学院 指导教师 张永全

中国计量学院

2013 年 5 月

郑 重 声 明

本人呈交的毕业设计论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本学位论文的知识产权归属于培养单位。

学生签名：_____　　日期：_____

致谢

我的本科毕业论文将给我大学的学习画上了一个句号。回想本科四年，的确通过看似枯燥的理论基础学习让自己的思维方式得到了锻炼，也在接触各种算法的过程中受益。

我最应该感谢的是我的家庭，他们给余我充分自由的空间让我转到了这个专业。除此以外，家庭的支持也是我大学学习最不能缺少的。

也由衷感谢我们专业课上的很多老师，正是他（她）们耐心的教学和真心的教育，启发、帮助了我的学习。而学习到的这些知识或许就是这篇学位论文的基础、或许也将在将来的某个时刻给与我启发或帮助。

神经网络中的分类问题

摘要：本文主要回顾总结了分类问题在机器学习领域的两种主流解决途径及其理论，包括了神经网络的 BP 模型以及支持向量理论背景下的 C-SVC 算法。除对各自理论的核心推导进行复述以外，还根据一个分类问题实例（异或问题）分别就 BP 网络算法和 C-SVC 算法进行学习和求解，并对两种机器学习方法进行对比和讨论。

关键词：神经网络；BP；支持向量机；C-SVC；分类问题。

Classification Problem in Neural Network

Abstract: This article mainly covers two mainstream ways of dealing classification problem in the area of machine learning, including BP module of neural network and the C-SVC of support vector machine. Apart from the formula presentation of basises of these modules, the article also gives an example on the XOR problem using the two methods as an illustration, and makes an analysis on the differences between the two learning methods.

Keywords: neural network; BP; C-SVC; classification problem.

目 录

1 引言

机器学习（machine learning）算法是一类从数据中自动分析获得规律，并利用规律对未知数据进行预测的算法[1]，在分类问题的应用中凸现优势（分类问题或许是机器学习领域被应用最为广泛的方向）。而在机器学习的众多理论中，支持向量机理论和神经网络理论又是处理分类问题的主流算法理论，在现实生活中得到了广泛的应用[2]5（比如垃圾邮件过滤、文件分类、图像识别和手写字识别等）。

本文对支持向量机理论和神经网络理论中处理分类问题的两个典型算法（C-SVC 和 BP 算法）进行了介绍和主要推导过程的重述，并用这两个算法解决一个分类问题的实例，旨在对两个理论提供一个直观的阐明和对比。由于 C-SVC 和 BP 算法在现实中的成功应用，了解和对比这两种算法是十分有益的。

2 分类问题

分类问题是本文所要探讨的中心问题。分类问题又进一步分为两类分类问题和多类分类问题。而多类分类机通常的算法是构造一系列两类分类机[3]215，每一个分类机都把其中的一类同余下的各类划分开来，据此判断某个输入的归属。其他多类分类的算法（比如成对分类法 [3]217）也是把多类分类问题转化为两类分类问题。所以解决分类问题的基础在于解决两类分类问题。

首先，对本文论述的两类分类问题给出以下定义：

给定训练集合

$$T=\{(x_1,y_1),\ldots,(x_i,y_i)\}\in(R^n\times\mathcal{Y})^l \tag{2.1}$$

其中 $x_i\in R^n,\quad y_i\in\mathcal{Y}=\{1,-1\},\quad i=1,2\ldots l$. 寻找 R^n 空间上的一个实值函数 $g(x)$，以便用决策函数

$$f(x)=sgn(g(x)) \tag{2.2}$$

以此判断任意输入 x 对应的输出 y。

3 C-SVC 模型

在用于分类的支持向量理论中，C-SVC（C-支持向量分类机）是一个经典、容错的非线性可分算法。C-SVC 输入非线性软间隔分类机，是综合了线性软间隔分类机（线性支持向量分类机）和非线性硬间隔分类机（可分支持向量分类机）的方法[3]190。

它的基本思想是，将分类问题通过最大间隔法[4]190 转换为优化问题，再通过求解这个优化问题的对偶问题（是一个凸二次规划问题）来解决原问题。并进一步地，运用了核函数技巧（kernel trick）在处理线性不可分问题[4]46 时不用关心如何对变量进行不同维数间的映射；使用松弛变量（regularization parameter）[2]502 来提供允许错误的样本。

之所以要转换为对偶问题，主要因为这里对偶问题的形式有以下几点优势：

(1) 输入变量在对偶问题中形式只以内积出现，从而能够运用核函数[4]81；

(2) 有利于利用问题的稀疏性，从而适合一些大规模算法的实现。比如针对 C-SVC 中对偶问题的序列最小最优化（sequential minimal optimization-SMO）算法[4]85。虽然 QP (Quadratic Programming) 的优化包算法[5]也可以求解模型中的原问题，但通过对偶问题求解这里的原问题比直接使用通用的 QP 优化包进行优化要高效得多[6]；

3.1 问题的转换：最大间隔法

为了将原始分类问题转换为优化问题，考虑最大间隔法：

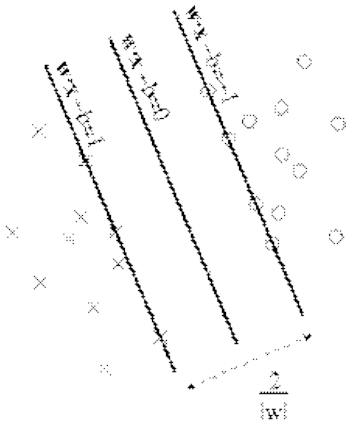

图 1. 最大间隔法

如上图所示，对于一个线性可分的训练集合 T，设所有能划分训练集合的超平面 $(w \cdot x) - b = 0$中，能经过训练集合的两个极端平面为：

$$\begin{aligned}(\tilde{w}\cdot x)-\tilde{b}=k\\(\tilde{w}\cdot x)-\tilde{b}=-k\end{aligned} \tag{3.1.2}$$

令$w=\frac{\tilde{w}}{k}\quad b=\frac{\tilde{b}}{k}$，用以下平面表示最大间隔平面，满足到(3.1.2)的两个平面间隔最大：

$$(w\cdot x)-b=0 \tag{3.1.3}$$

设最大间隔为d，可由以下方程联立求得

$$\begin{cases}(w\cdot x_0)-b=1\\ d=\frac{(w\cdot x_0)-b+1}{||x_0||}\end{cases} \tag{3.1.4}$$

可解得最大间隔并把分类问题的目标函数转换为

$$\max_{w,\,b}\qquad d=\frac{2}{||w||} \tag{3.1.5}$$

写成习惯上的最小化形式就得到了分类问题的等价优化问题：

$$\begin{aligned}&\min_{\omega,\,b}\qquad \frac{1}{2}\|\omega\|^2\\&s.t.\qquad y_i((\omega\cdot x_i)+b)\geq 1,\qquad i=1,\ldots,l\end{aligned} \tag{3.1.6}$$

3. 2 问题的转换：对偶问题

问题(3.1.6)带有约束条件，可以通过引入 Lagrange 因子把(3.1.6)的约束条件以 Lagrange 函数的形式融入到目标函数中：

$$\mathbf{L}(\omega,b,\alpha)=\tfrac{1}{2}\|\omega\|^2-\sum_{i=1}^{l}\alpha_i(y_i((\omega\cdot x_i)+b)-1) \tag{3.2.1}$$

使得问题(3.1.6)等价于

$$\min_{\omega,b}\ \max_{\alpha_i\geq 0}\mathbf{L}(\omega,b,\alpha) \tag{3.2.2}$$

交换(3.2.2)中取最大值和最小值的顺序，便得到原始问题(3.2.2)的对偶问题：

$$\max_{\alpha_i\geq 0}\ \min_{\omega,b}\mathbf{L}(\omega,b,\alpha) \tag{3.2.3}$$

注意到使得(3.2.1) 式中使$(y_i((\omega\cdot x_i)+b)-1)$等于 0 的训练样本处于(3.1.2)式所表示的极端平面上，称作支持向量。从这个意义可以推断，要使(3.2.3)式达到关于 Lagrange 因子的最大，非支持向量所对应的 Lagrange 因子必须等于零（但是支持向量所对应的 Lagrange 因子也可以为零）。

我们对(3.2.3)式做一定的变形。由于$\mathbf{L}(\omega,b,\alpha)$是关于$\omega$的严格凸二次函数，它取最小值时

$$\nabla_\omega\mathbf{L}(\omega,b,\alpha)=\omega-\sum_{i=1}^{l}y_ix_i\alpha_i=0 \tag{3.2.4}$$

即

$$\omega=\sum_{i=1}^{l}\alpha_iy_ix_i \tag{3.2.5}$$

将上式代入(3.2.1)式

$$\begin{aligned}\mathbf{L}(\omega,b,\alpha)&=\frac{1}{2}\sum_{i=1}^{l}\sum_{i=j}^{l}y_iy_j\alpha_i\alpha_j(x_i\cdot x_j)-\sum_{i=1}^{l}\alpha_i(y_i(\sum_{j=1}^{l}\alpha_jy_j(x_i\cdot x_j)+b)-1)\\&=\frac{1}{2}\sum_{i=1}^{l}\sum_{i=j}^{l}y_iy_j\alpha_i\alpha_j(x_i\cdot x_j)-\sum_{i=1}^{l}\sum_{i=j}^{l}y_iy_j\alpha_i\alpha_j(x_i\cdot x_j)-b\sum_{i=1}^{l}\alpha_iy_i+\sum_{i=1}^{l}\alpha_i\\&=-\frac{1}{2}\sum_{i=1}^{l}\sum_{i=j}^{l}y_iy_j\alpha_i\alpha_j(x_i\cdot x_j)-b\sum_{i=1}^{l}\alpha_iy_i+\sum_{i=1}^{l}\alpha_i\end{aligned}$$

通过上式可以推出以下两种情况

$$\min_{\omega,b}\mathbf{L}(\omega,b,\alpha)=\begin{cases}-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}y_iy_j\alpha_i\alpha_j(x_i\cdot x_j)+\sum_{j=1}^{l}\alpha_j & when\sum_{i=1}^{l}y_i\alpha_i=0\\ -\infty & else\end{cases}$$

由于(3.2.3)式需要最大化上式，所以(3.2.3)等价于上式第一种情，此时$\mathbf{L}(\omega,b,\alpha)$不再和$\omega,\ b$有关系，所以(3.2.3)可以直接写成

$$\begin{aligned}\max_{\alpha}\quad & -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}y_iy_j(x_i\cdot x_j)\alpha_i\alpha_j+\sum_{j=1}^{l}\alpha_j\\ s.t.\quad & \sum_{i=1}^{l}y_i\alpha_i=0\\ & \alpha_i\geq 0,\quad i=1,\ldots,l\end{aligned}$$

写成习惯上的最小化形式：

$$\begin{aligned}\min_{\alpha}\quad & \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}y_iy_j(x_i\cdot x_j)\alpha_i\alpha_j-\sum_{j=1}^{l}\alpha_j\\ s.t.\quad & \sum_{i=1}^{l}y_i\alpha_i=0\\ & \alpha_i\geq 0,\quad i=1,\ldots,l\end{aligned}\tag{3.2.6}$$

该问题的矩阵形式可以表示为：

$$\begin{aligned}\min_{\alpha}\quad & W(\alpha)=\frac{1}{2}\alpha^TH\alpha-e^T\alpha\\ s.t.\quad & \alpha^Ty=0\\ & \alpha\geq 0\end{aligned}\tag{3.2.7}$$

其中

$$\begin{aligned}H&=(y_iy_j(x_i\cdot x_j))_{l\times l}\\ e&=(1,\ldots,1)^T\\ \alpha&=(\alpha_1,\ldots,\alpha_l)^T\\ y&=(y_1,\ldots,y_l)^T\end{aligned}\tag{3.2.8}$$

容易证明[4]9，该对偶问题是如下所定义的凸规划问题：

$$\begin{aligned} \min \quad & f_0(x) && x \in R^n \\ s.t. \quad & f_i(x) \le 0 && i = 1, \dots, m \\ & h_i(x) = a_i^T x - b_i = 0 && i = 1, \dots, p. \end{aligned} \tag{3.2.9}$$

其中 $f_0(x)$ 和 $f_i(x)$ 是连续可微的凸函数[4]6， $h_i(x)$ 是线性（仿射）函数。

需要知道的是，凸规划问题一般具备一些解的优良性质，比如局部解的全局性和解的唯一性[4]9,10。对于凸规划问题可以简单的说“问题的解”，因为它的整体解和局部解并无区别。

3. 3 对偶问题和原问题的关系

在给出了对偶问题的最终形式(3.2.8)以后，再来探讨对偶问题和原始问题(3.2.2)的关系。

令原始问题和对偶问题的最优解分别为：

$$\begin{aligned} &\min_{\omega,\, b} \ \max_{\alpha_i \ge 0} \mathbf{L}(\omega, b, \alpha) = p^* \\ &\max_{\alpha_i \ge 0} \ \min_{\omega,\, b} \mathbf{L}(\omega, b, \alpha) = d^* \end{aligned} \tag{3.2.9}$$

由弱对偶定理[3]34（弱对偶定理是“最小-最大”对偶问题普遍满足的定理，与问题是否是凸规划问题无关）可以得出

$$p^* \ge d^* \tag{3.2.10}$$

从上面的结论可以直接推出，如果能找到原始问题(3.2.2)和对偶问题(3.2.3)的一个可行解 $(\tilde{\omega},\ \tilde{b})$ 和 $\tilde{\alpha}$，使得(3.2.9)式中的

$$f(\tilde{\omega}, \tilde{b}) = \max_{\alpha_i \ge 0} \mathbf{L}(\tilde{\omega}, \tilde{b}, \alpha) = g(\tilde{\alpha}) = \min_{\omega,\, b} \mathbf{L}(\omega, b, \tilde{\alpha}) \tag{3.2.11}$$

成立，则(3.2.10)中的不等式取等号，我们就可以断定 $(\tilde{\omega},\ \tilde{b})$ 和 $\tilde{\alpha}$ 就是所对应的原始问题和对偶问题的整体最优解。

假设已经通过求解对偶问题(3.2.7)得到了解 $\tilde{\alpha}$，我们将在后面的 3.5 节中通过 KKT 条件作为纽带，用 $\tilde{\alpha}$ 表示满足(3.2.11)式的原问题的可行解 $(\tilde{\omega},\ \tilde{b})$，进而由上面的讨论可知 $(\tilde{\omega},\ \tilde{b})$ 就是我们要求的原问题整体最优解。

3.4 Karush–Kuhn–Tucker 条件

根据强对偶定理[4]14（证明参见文献[7]234），使得(3.2.10)式取等号的条件（也可以使其他条件）是原问题（必须是凸规划问题）满足 Slater 条件：

称凸规划问题(3.1.1)满足 Slater 条件，如果存在可行点 $\hat{x}$ 满足

$$\begin{cases} f_i(\hat{x}) & < 0 \qquad i = 1,\dots,m \\ a_i^T\hat{x} - b_i & = 0 \qquad i = 1,\dots,p \end{cases} \tag{3.4.1}$$

即，存在可行解使得原问题中不等式约束的“小于等于号”能取“严格小于号”。

需要注意的是，在凸规划问题(3.1.1)中的不等式约束为线性约束时，(3.2.10)式取可以直接取等号而不用满足 Slater 条件[4]13。显然，我们的对偶问题(3.2.7)就是属于这种情况。

在(3.2.10)式取等号的条件下我们可以证明，如果 x^* 是凸规划问题(3.1.1)的解（也是最优解），则 x^* 满足以下定义的 KKT 条件：

考虑凸规划问题(3.1.1)，称 x^* 满足 KKT 条件，如果存在分别与约束条件相对应的 Lagrange 乘子向量 $\lambda^* = (\lambda_1^*,\dots,\lambda_m^*)$ 和 $\nu^* = (\nu_1^*,\dots,\nu_m^*)$ 使得 Lagrange 函数

$$\mathbf{L}(x,\lambda,\nu) = f_0(x) + \sum_{i=1}^{m}\lambda_i f_i(x) + \sum_{i=1}^{p}\nu_i h_i(x) \tag{3.4.2}$$

满足

$$\begin{aligned} f_i(x^*) &\le 0, \qquad i = 1,\dots,m \\ h_i(x^*) &= 0, \qquad i = 1,\dots,p \\ \lambda_i^* &\ge 0, \qquad i = 1,\dots,m \\ \lambda_i^* f_i(x^*) &= 0, \qquad i = 1,\dots,m \\ \nabla_x\mathbf{L}(x^*,\lambda^*,\nu^*) &= \nabla f_0(x^*) + \sum_{i=1}^{m}\lambda_i^*\nabla f_i(x^*) + \sum_{i=1}^{p}\nu_i^*\nabla h_i(x^*) = 0 \end{aligned} \tag{3.4.3}$$

实际上，对于凸规划问题，KKT 条件不仅是最优解的必要条件，也是充分条件。由于我们将仅需要用到必要条件，下面只给出上述 KKT 条件是最优解的必要条件的证明：

首先，引入 Lagrange 乘子以后把原问题写成

$$\min_{x}\max_{\lambda\geq 0,\nu}\mathbf{L}(x,\lambda,\nu)=\min_{x}f_0(x) \tag{3.4.4}$$

则其对偶形式为

$$\max_{\lambda\geq 0,\nu}\min_{x}\mathbf{L}(x,\lambda,\nu)=\max_{\lambda\geq 0,\nu}g(\lambda,\nu) \tag{3.4.5}$$

若(3.2.10)式能取等号，根据强对偶定理[4]14 可以得出原问题和对偶问题的最优值相等。故有以下不等式关系成立

$$\begin{aligned} f_0(x^*) &= g(\lambda^*,\nu^*) \\ &= \min_{x}\mathbf{L}(x,\lambda^*,\nu^*) \\ &\leq \mathbf{L}(x^*,\lambda^*,\nu^*) \\ &\leq f_0(x^*) \end{aligned} \tag{3.4.6}$$

由于(3.4.6)式的首尾两项相同，所以中间的不等号可以加强为等号。这样根据(3.4.6) 式的最后两项可以得到

$$\sum_{i=1}^{m}\lambda_i^* f_i(x^*)+\sum_{i=1}^{p}\nu_i^* h_i(x^*)=0 \tag{3.4.7}$$

现在证明原问题的最优解 x^* 满足 KKT 条件。(3.4.3)中的前三个式子显然成立；注意到(3.4.7)式等号左边的 $h_i(x^*)$ 等于零，而剩余其它项为非负，故(3.4.3)中的第四个式子也证；再根据(3.4.6)式， x^* 是函数 $\mathbf{L}(x,\lambda^*,\nu^*)$ 的极小点，所以有(3.4.3)中的第五个式子成立，证毕。

3.5 原问题解的推导

在 3.2 节中的最后已经给出对偶问题(3.2.7)的解是凸规划问题，所以根据 3.4 节的结论，如果 α^*是问题(3.2.7)的解（也是最优解），则 α^*满足 KKT 条件。据此，存在 Lagrange 乘子向量s^*和 $\tilde{b}^*$ 使得对应的 Lagrange 函数

$$\mathbf{L}(\alpha, s, \tilde{b}) = W(\alpha) + s^T(-\alpha) + \tilde{b}(\alpha^T y)$$

满足：

$$\begin{aligned} \alpha^* &\geq 0 \\ \alpha^{*T} y &= 0 \\ s^* &\geq 0 \\ s^{*T}\alpha^* &= 0 \\ H\alpha^* - e + \tilde{b}^* y - s^* &= 0 \end{aligned} \tag{3.5.1}$$

由(3.5.1)第三式和第五式可知

$$H\alpha^* - e + \tilde{b}^* y \geq 0 \tag{3.5.2}$$

令

$$\omega^* = \sum_{i=1}^{l} \alpha_i^* y_i x_i \tag{3.5.3}$$

则(3.5.2)式等价于

$$y_i((\omega^* \cdot x_i) + \tilde{b}^*) \geq 1, \qquad i = 1, \ldots, l \tag{3.5.4}$$

这表明$(\omega^*, \tilde{b}^*)$是原问题(3.1.6)的可行解。进一步，由(3.5.3)和(3.5.1)可以推出原问题(3.1.6)的可行解(ω^*, b^*)和对偶问题的解(3.2.7) α^*可以使得(3.2.10)中的不等式取等号（原问题和对偶问题的最优值相同）：

$$\begin{aligned}\frac{1}{2}\|\omega^*\|^2 &= \frac{1}{2}\alpha^{*T}H\alpha^* \\ &= \frac{1}{2}\alpha^{*T}H\alpha^* - \alpha^{*T}(H\alpha^* + \tilde{b^*}y - e - s^*) \\ &= -\frac{1}{2}\alpha^{*T}H\alpha^* - \tilde{b^*}\alpha^{*T}y + e^T\alpha^* + s^{*T}\alpha^* \\ &= -\frac{1}{2}\alpha^{*T}H\alpha^* - \tilde{b^*}\mathbf{0} + e^T\alpha^* + \mathbf{0} \\ &= -\frac{1}{2}\alpha^{*T}H\alpha^* + e^T\alpha^*\end{aligned}$$

根据 3.3 节中的结论，$(\omega^*, \tilde{b^*})$就是原问题(3.1.6)的整体最优解。

在(3.5.3)式中已给出通过对偶问题的解 α^*求 ω^*的关系式，下面再次利用 KKT 条件来用 α^*表示另外一个待求解的变量 $\tilde{b^*}$。

若 $\exists j \quad s.t. \quad \alpha_j^* > 0$，由(3.5.1)中的式子可以看出 $s_j^* = 0$，所以 $H\alpha^* - e + \tilde{b^*}y$ 的第 j 个分量等于零，有

$$\begin{aligned}\begin{bmatrix} y_1y_j(x_1 \cdot x_j) & \cdots & y_ly_j(x_l \cdot x_j) \end{bmatrix}\begin{bmatrix}\alpha_1^* \\ \vdots \\ \alpha_l^*\end{bmatrix} - 1 + \tilde{b^*}y_j &= 0 \\ 1 - y_j\sum_{i=1}^{l}\alpha_i^* y_i(x_i \cdot x_j) &= \tilde{b^*}y_j \\ \frac{1}{y_j} - \sum_{i=1}^{l}\alpha_i^* y_i(x_i \cdot x_j) &= \tilde{b^*} \\ y_j - \sum_{i=1}^{l}\alpha_i^* y_i(x_i \cdot x_j) &= \tilde{b^*}\end{aligned}$$

综上，最终的决策函数可以表示为

$$f(x) = sgn(g(x)) \tag{3.5.5}$$

其中划分超平面为

$$g(x) = (\omega^* \cdot x) + \tilde{b^*} = \sum_{i=1}^{l} y_i\alpha_i^*(x_i \cdot x) + \tilde{b^*} \tag{3.5.6}$$

3.6 理论的扩展

根据上面几节的讨论我们已经得出一个多维空间的线性分类算法（被称为线性可分支持向量机[4]54），下面介绍对算法进行的两个扩展，赋予算法容错能力和提高算法处理非线性分类问题的能力。

引入松弛变量

$$\xi_i \geq 0, \quad i = 1, 2 \ldots l \tag{3.6.1}$$

得到对应于原始优化问题(3.1.6)软化了的约束条件

$$y_i \left((\omega \cdot x_i) + b \right) \geq 1 - \xi_i \ , \quad i = 1, 2 \ldots l \tag{3.6.2}$$

即使存在错误的（希望被忽略的）训练样本，松弛变量的引入可以让样本仍旧满足约束。但是另一方面，我们不希望松弛变量太大，所以自然想到在相应问题(3.1.6)的目标函数中加入对松弛变量的最小化：

$$\min_{\omega, b, \xi} \quad \frac{1}{2} \left|| \omega \right||^2 + C \sum_{i=1}^{l} \xi_i \tag{3.6.3}$$

其中 C 被称为惩罚参数，其意义是我们对错误允许大小的惩罚程度。在文献[2]504 中讨论了有关惩罚参数选择的话题。

经过上面几节相似的推导过程[4]60 可得出具有容错能力的算法：

算法 1 （线性支持向量机）

(1) 给定训练集合

$$T=\{(x_1,y_1),\ldots,(x_l,y_l)\}\in(R^n\times\mathcal{Y})^l \quad where \quad x_i\in R^n, y\in\mathcal{Y}=\{1,-1\}, i=1,2\ldots l.$$

(2) 构造并求解凸二次规划问题

$$\begin{aligned}\min_{\alpha}\quad & \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}y_iy_j(x_i\cdot x_j)\alpha_i\alpha_j-\sum_{j=1}^{l}\alpha_j\\ s.t.\quad & \sum_{i=1}^{l}y_i\alpha_i=0\\ & C\geq\alpha_i\geq 0,\quad i=1,\ldots,l\end{aligned}$$

得到解

$$\alpha^*=(\alpha_1^*,\ldots,\alpha_l^*)^T$$

(3) 选取 α^*的一个正分量的下标号 j，据此计算

$$\tilde{b^*}=y_j-\sum_{i=1}^{l}\alpha_i^*y_i(x_i\cdot x_j)$$

(4) 得到最终决策函数

$$f(x)=sgn(\sum_{i=1}^{l}y_i\alpha_i^*(x_i\cdot x)+\tilde{b^*})$$

在上述算法的基础上引入核函数以后就是 C-SVC 算法。对于核函数，考虑某个线性不可分样本的输入变量，我们通常通过对把输入变量做一个到其他维数空间的映射

$$x'=\phi(x) \tag{3.6.4}$$

例如，对于二维输入变量作为输入点的样本平面，考虑类似下图的样本数据图像（不同颜色区分了样本的正负类别）：

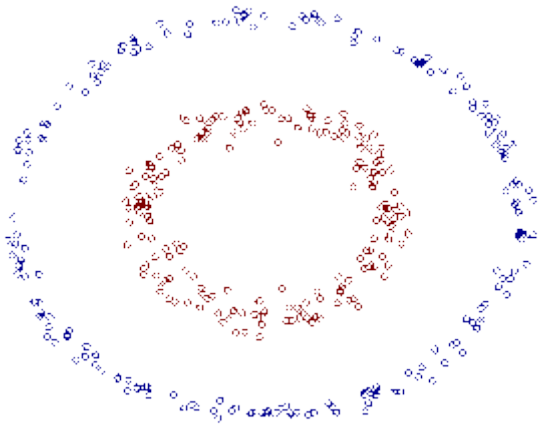

图 2. 二维样本示例

容易想到这样一个变换 ϕ_1，它可以把此例转的输入样本转化为三维空间上的线性可分问题：

$$\phi_1\left(\left[x_{[1]},x_{[2]}\right]^T\right)=\left[x_{[1]},x_{[2]},\sqrt{x_{[1]}^2+x_{[2]}^2}\right]^T$$

事实上，还可以选择更为容易的 ϕ_2 将输入变量转换到一维空间仍然可以使得问题在一维空间线性可分：

$$\phi_2\left(\left[x_{[1]},x_{[2]}\right]^T\right)=\left[x_{[1]}^2+x_{[2]}^2\right]^T$$

可见变换 ϕ 的选取可以是多样的。

我们也可以不用选取变换 ϕ 而完成问题的求解。观察到算法 1 的公式中输入变量都是以内积的形式出现的，如果能选取一个函数，使得存在变换 ϕ 满足

$$K(x_1,x_2)=(\phi(x_1)\cdot\phi(x_2)) \tag{3.6.5}$$

则称这样的函数为核函数。那么我们可以用输入变量的核函数值来代替输入变量内积的形式，从而并不需要计算变换 ϕ 之后再做向量内积运算，某些情况

下可以减少计算量。可以用作核函数的函数有很多，这里并不具体深入讨论，相关细节可以参看文献[4]95。

引入了核函数的算法 1 被就是广泛被应用的 C-SVC 算法。

4 BP 神经网络模型

在人工神经网络发展的前期，单层感知器作为最简单的分类网络，被感知器收敛定理（perceptron convergence theorem）[9]159 证明只能处理线性可分问题。而含有隐藏层的多层感知器网络的典型算法 BP 算法（反向传播算法，Back-Propagation algorithm）可以处理线性不可分问题。这里我们将主要从 BP 神经网络的正向传播和反向传播（权值修正）两个内容来推导 BP 算法。

4.1 激发函数

在多层感知器网络中激发函数应选择非线性函数，否则网络可能退化为单层感知器，从而不能处理线性不可分问题[9]179。

本文的 BP 算法选用 Log-Sigmoid 函数：

$$f(n)=\frac{1}{1+e^{-n}} \tag{4.1.1}$$

该函数的导数可以写成如下的形式，这在算法程序实现的时候提供了方便：

$$f'(n)=f(n)(1-f(n)) \tag{4.1.2}$$

4.2 网络的正向传播

网络的传播规则需要从多层感知器网络的网络拓扑结构理解，如下图所示的多层感知器网络是一个含有两个隐藏层，每个隐藏层有 3 个感知器的典型多层感知器网络：

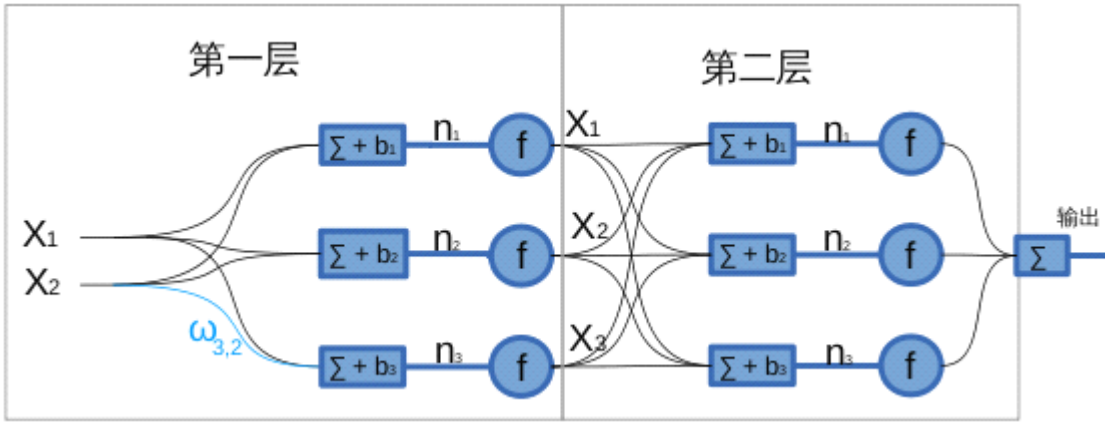

图 3. 多层感知器网络示例

对于网络的每一层，接受向量 $\mathbf{X}^k$ 作为输入，通过该层的网络权矩阵 $\mathbf{W}^k$ 传递给这一层的各个感受器。感受器加和所有输入以后经过激发函数作为本节点的输出，并将这个输出作为下一层神经元的一个输入 $\mathbf{X}^{k+1}$（用上标表示变量所属的网络层次）。

比如，图 3 的第一层的网络可以表示为：

$$\begin{aligned}\mathbf{n}^1 &= \mathbf{W}^1\mathbf{X}^1 + \mathbf{b}^1 \\ \mathbf{X}^2 &= f(\mathbf{n}^1)\end{aligned} \tag{4.2.1}$$

其中

$$\mathbf{W}^1 = \begin{bmatrix} \omega_{1,1}^1 & \omega_{1,2}^1 \\ \omega_{2,1}^1 & \omega_{2,2}^1 \\ \omega_{3,1}^1 & \omega_{3,2}^1 \end{bmatrix} \qquad \mathbf{n}^1 = \begin{bmatrix} n_1^1 \\ n_2^1 \\ n_3^1 \end{bmatrix}$$

$$\mathbf{x}^1 = \begin{bmatrix} x_1^1 \\ x_2^1 \end{bmatrix} \qquad \mathbf{x}^2 = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \end{bmatrix} \qquad \mathbf{b}^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

由第一层的正向传递过程可以很容易类推到后面一层以及一般情况下网络的正向传递过程。给定学习样本 (X^1, Y^*)，将 X^1 作为第一层网络的输入，网络正向传播的结果是，经过所有层网络的计算以后给出一个输出 Y 。

4.3 网络的反向传播（权值修正）

我们的目的是要根据 Y 和样本目标值 Y^* 的网络误差

$$\hat{F} = \sum_i (Y_i^* - Y_i)^2 \tag{4.3.2}$$

来调整各个神经连接的连接权值 $\omega_{i,j}^m$ 以及偏置值 b_i^m，并不断迭代，期望使得误差逐渐变小。之后的推导将会说明，我们需要从最后一层网络开始，反向地计算上一层网络的连接权值修正量，以此方向来进行权值修正。

考虑网络中的某一个权值 $\omega_{i,j}^m$，我们希望调整它，使得下一次迭代误差更小。根据最速下降法[10]的思想，可以按照以下式子迭代修正（ k 为迭代序号）：

$$\omega_{i,j}^{m}(k+1)=\omega_{i,j}^{m}(k)-\alpha\frac{\partial\hat{F}}{\partial\omega_{i,j}^{m}} \tag{4.3.3}$$

其中常数α是学习速度。(4.3.3)式是很好理解的，如下图所示，在某点的网络误差关于权值的偏导数大于零的时候我们希望$\omega_{i,j}^{m}$再减少一些以使得网络误差$\hat{F}$更小：

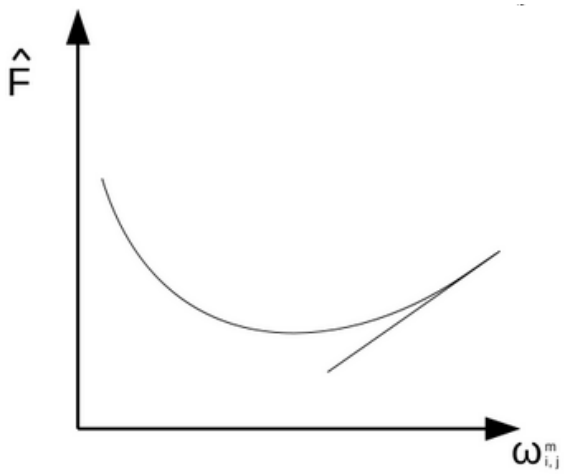

图 4. 网络误差关于权值的修正示意

同理地偏置值b_i^m：

$$b_i^m(k+1)=b_i^m(k)-\alpha\frac{\partial\hat{F}}{\partial b_i^m} \tag{4.3.4}$$

现在，问题转换为求(4.3.3)式和(4.3.4)式中的偏导数$\frac{\partial\hat{F}}{\partial\omega_{i,j}^{m}}$和$\frac{\partial\hat{F}}{\partial b_i^m}$，我们接下来将根据偏导数的链式法则来求解。

首先，考虑某一层（第m层）网络，把网络误差$\hat{F}$看作是以$\omega_{i,j}^{m}$为自变量，完全依赖于这一层所有

$$n_i^m \qquad (i=1,2\ldots l_m) \tag{4.3.5}$$

为中间变量的复合函数，其中l_m为第m层网络感知器的个数。

根据偏导数的链式法则，有：

$$\begin{aligned}\frac{\partial \hat{F}}{\partial \omega_{i,j}^m} &= \frac{\partial \hat{F}}{\partial n_1^m}\frac{\partial n_1^m}{\partial \omega_{i,j}^m} + \frac{\partial \hat{F}}{\partial n_2^m}\frac{\partial n_2^m}{\partial \omega_{i,j}^m} + \ldots + \frac{\partial \hat{F}}{\partial n_{l_m}^m}\frac{\partial n_{l_m}^m}{\partial \omega_{i,j}^m} \\ &= 0 + \ldots + 0 + \frac{\partial \hat{F}}{\partial n_i^m}\frac{\partial n_i^m}{\partial \omega_{i,j}^m} + 0 + \ldots \\ &= \frac{\partial \hat{F}}{\partial n_i^m}\frac{\partial n_i^m}{\partial \omega_{i,j}^m}\end{aligned} \tag{4.3.6}$$

对于偏置值变量b_i^m，同理有

$$\frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m}\frac{\partial n_i^m}{\partial b_i^m} \tag{4.3.7}$$

而由于

$$n_i^m = \sum_{j=1}^{l_{m-1}} \omega_{i,j}^m x_j^m + b_i^m \tag{4.3.8}$$

可以将(4.3.6)和(4.3.7)式各自最后的因子进一步写为

$$\begin{aligned}\frac{\partial n_i^m}{\partial \omega_{i,j}^m} &= x_j^m \\ \frac{\partial n_i^m}{\partial b_i^m} &= 1\end{aligned} \tag{4.3.9}$$

目前(4.3.6)和(4.3.7)两式都还剩余一个相同的因子没有进一步表示，我们把定义它为

$$s_i^m = \frac{\partial \hat{F}}{\partial n_i^m} \tag{4.3.10}$$

这个定义式了可以通过一个递推关系求得。具体地，把网络误差$\hat{F}$看作是以$m+1$层所有

$$n_i^{m+1} \qquad (i = 1, 2 \ldots l_m) \tag{4.3.11}$$

为中间变量的复合函数，再次根据偏导数的链式法则：

$$
\begin{aligned}
s_i^m = \frac{\partial \hat{F}}{\partial n_i^m} &= \sum_{j=1}^{l_{m+1}} \left(\frac{\partial \hat{F}}{\partial n_j^{m+1}} \frac{\partial n_j^{m+1}}{\partial n_i^m} \right) \\
&= \sum_{j=1}^{l_{m+1}} \left(s_j^{m+1} \frac{\partial n_j^{m+1}}{\partial n_i^m} \right) \\
&= \sum_{j=1}^{l_{m+1}} \left(s_j^{m+1} \frac{\partial \left(\sum_{p=1}^{l_m} \left(\omega_{j,p}^{m+1} x_p^{m+1} + b_j^{m+1} \right) \right)}{\partial n_i^m} \right) \\
&= \sum_{j=1}^{l_{m+1}} \left(s_j^{m+1} \omega_{j,i}^{m+1} \frac{\partial x_i^{m+1}}{\partial n_i^m} \right) \\
&= \sum_{j=1}^{l_{m+1}} \left(s_j^{m+1} \omega_{j,i}^{m+1} \frac{\partial f(n_i^m)}{\partial n_i^m} \right) \\
&= \sum_{j=1}^{l_{m+1}} \left(s_j^{m+1} \omega_{j,i}^{m+1} f(n_i^m)(1 - f(n_i^m)) \right) \\
&= f(n_i^m)(1 - f(n_i^m) \sum_{j=1}^{l_{m+1}} \left(s_j^{m+1} \omega_{j,i}^{m+1} \right)
\end{aligned}
\tag{4.3.12}
$$

可见前一层的某一个 s_i^m 可以通过后一层所有的

$$
s_j^{m+1} \quad (j = 1, 2 \ldots l_{m+1}) \tag{4.3.13}
$$

计算出来。故实际计算时需要从最后一层（假设为第 M 层）来往前逐层推导，这就是 BP（反向传播）算法得名的原因。

递推关系式的初始点自然是最高层的 s_i^M，它可以通过下式计算：

$$
\begin{aligned}
s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} &= \frac{\partial \left(\sum_{j=1}^{l_M} (Y_j^* - Y_j)^2 \right)}{\partial n_i^M} \\
&= -2(Y_i^* - Y_i) \frac{\partial Y_i}{\partial n_i^M} \\
&= -2(Y_i^* - Y_i) \frac{\partial f(n_i^M)}{\partial n_i^M}
\end{aligned}
\tag{4.3.14}
$$

为了化简计算，最后一层的激活函数可以取 $f(n)=n$，此时(4.3.14)式等于

$$s_i^M = -2(Y_i^* - Y_i) \tag{4.3.15}$$

至此 BP 神经网络的反向传播权值修正的公式推导就完成了。

回顾以上推导，我们可以归纳出 BP 算法：

算法 2 （BP 算法）

(1) 根据输入样本 (X^1, Y^*) 的输入 X^1 经过网络正向传播的输出 Y，求出(4.3.15)式作为递推表达式(4.3.12)的初始点：

$$s_i^M = -2(Y_i^* - Y_i)$$

(2) 对于这一层（第 m 层）网络，通过以下式子修正当前层的网络连接权值和偏置值（选择一个常数 α 作为学习速度）：

$$\omega_{i,j}^m(k+1) = \omega_{i,j}^m(k) - \alpha s_i^m x_j^m$$
$$b_i^m(k+1) = b_i^m(k) - \alpha s_i^m$$

(3) 由(4.3.12)式对应的递推表达式得出上一层网络的 s_i^m：

$$s_i^m = f(n_i^m)(1-f(n_i^m))\sum_{j=1}^{l_{m+1}}(s_j^{m+1}\omega_{j,i}^{m+1})$$
$$= x_i^{m+1}(1-x_i^{m+1})\sum_{j=1}^{l_{m+1}}(s_j^{m+1}\omega_{j,i}^{m+1})$$

(4) 回到步骤(2)进行对上一层网络的迭代，直到当前网络层为第一个隐含层时结束算法。

需要提及的是，在解决很多实际问题的时候，有必要运用一些技巧(比如输入变量的单位化)让算法 2 表现得更好（比如修正后的参数能让网络误差收敛更快），具体请参考文献[9]200。

5 一个分类问题的实例：异或问题

异或问题可以看作一个线性不可分的分类问题，其训练集合如下：

$$T=\{(\begin{bmatrix}1\\-1\end{bmatrix},1),(\begin{bmatrix}-1\\1\end{bmatrix},1),(\begin{bmatrix}-1\\-1\end{bmatrix},-1),(\begin{bmatrix}1\\1\end{bmatrix},-1)\} \tag{5.1}$$

我们将分别用算法 1 和算法 2 对该问题进行求解。

5.1 C-SVC 对问题的求解

对于训练样本集合(5.1)，选取二阶非齐次多项式核函数[4]95：

$$\begin{aligned}&K(x_1,x_2)\\&=(1+(x_i\cdot x_j))^2\\&=(1+x_{i1}x_{j1}+x_{i2}x_{j2})^2\\&=1+x_{i1}^2x_{j1}^2+2x_{i1}x_{j1}x_{i2}x_{j2}+x_{i2}^2x_{j2}^2+2x_{i1}x_{j1}+2x_{i2}x_{j2}\end{aligned}$$

可以看出，对于某个样本输入 x_i，该核函数所对应的变换 ϕ 为：

$$\phi(x_i)=[1,\quad x_{i1}^2,\quad \sqrt{2}x_{i1}x_{i2},\quad x_{i2}^2,\quad \sqrt{2}x_{i1},\quad \sqrt{2}x_{i2}]$$

则变换 ϕ 对本例的四个样本作用后的结果为：

$$\begin{aligned}&\phi(x_1)=[1,1,-\sqrt{2},1,\sqrt{2},-\sqrt{2}]\\&\phi(x_2)=[1,1,-\sqrt{2},1,-\sqrt{2},\sqrt{2}]\\&\phi(x_3)=[1,1,\sqrt{2},1,-\sqrt{2},-\sqrt{2}]\\&\phi(x_4)=[1,1,\sqrt{2},1,\sqrt{2},\sqrt{2}]\end{aligned}$$

由于本例重在演示，所以我们算出了每个样本对应的 ϕ 变换，用样本变量变换之后的内积计算，而事实上我们可以直接运用核函数而减少很多计算量。

我们选取惩罚参数 C 为正无穷，根据算法 1，求解凸二次规划问题：

$$\begin{aligned}\min_{\alpha} \quad & \frac{9}{2}\left(\alpha_1^2+\alpha_2^2+\alpha_3^2+\alpha_4^2\right)+\alpha_1\alpha_2-\alpha_1\alpha_3-\alpha_1\alpha_4 \\ & -\alpha_2\alpha_3-\alpha_2\alpha_4+\alpha_3\alpha_4-\alpha_1-\alpha_2-\alpha_3-\alpha_4 \\ s.t. \quad & \alpha_1+\alpha_2-\alpha_3-\alpha_4=0\end{aligned}$$

求出解

$$\alpha_1=\alpha_2=\alpha_3=\alpha_4=\frac{1}{8}$$

然后求出分割平面的参数：

$$\begin{aligned}\omega^* &= \sum_{i=1}^{l}\alpha_i y_i \phi(x_i) \\ &= \frac{1}{8}\left(\phi(x_1)+\phi(x_2)-\phi(x_3)-\phi(x_4)\right) \\ &= \begin{bmatrix}0 & 0 & -\frac{\sqrt{2}}{2} & 0 & 0 & 0\end{bmatrix}\end{aligned}$$

$$\begin{aligned}\tilde{b^*} &= y_j-\sum_{i=1}^{l}\alpha_i^* y_i\left(\phi(x_i)\cdot\phi(x_j)\right), \quad j=1 \\ &= 1-\frac{1}{8}\left(9+1-1-1\right) \\ &= 0\end{aligned}$$

继而计算最后的决策函数：

$$\begin{aligned}f(x) &= sgn(\omega^*\phi(x)+\tilde{b^*}) \\ &= sgn(-\frac{\sqrt{2}}{2}\sqrt{2}x_{[1]}x_{[2]}) \\ &= sgn(-x_{[1]}x_{[2]})\end{aligned}$$

5.2 BP 神经网络对问题的求解

考虑到 BP 神经网络的训练需要反复执行算法 2 来使得网络收敛，有比较大的计算量，本文使用 C 语言程序来实现 BP 神经网络对(5.1)式所表示训练集合的学习，用计算机来完成算法的计算。

程序使用一个含有两个隐藏层，每个隐藏层三个感知器的多层网络对训练样本集合(5.1)进行学习（实际上，解决异或问题的 BP 神经网络仅仅需要一个含有两个感知器的隐藏层组成的网络[9]198）。

程序执行的结果是，给定一个固定的随机种子(467)以后分别生成在[0,1]范围内的随机数赋值给每一个网络权值和偏置值，网络会在 1200 左右次迭代后网络误差(4.3.2)收敛到零。下图给出了网络输出与样本期望输出的差值关于训练次数的变化，反映了网络误差的收敛过程：

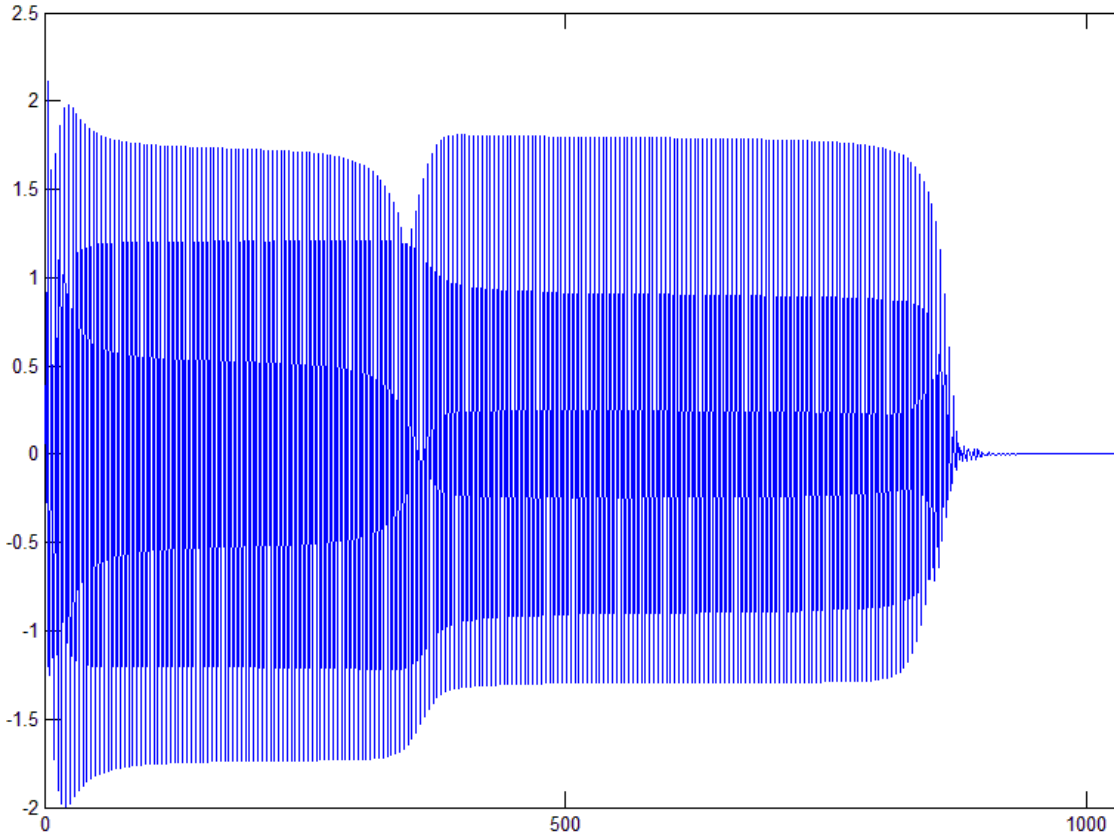

图 5. 网络误差收敛图

具体程序见附录。

6 对比和总结

从之前的理论推导和例子的对比可以看出，C-SVC 和 BP 算法下的多层感知器神经网络在解决异或问题的方式上各有特点。BP 神经网络在解决异或问题上计算量明显大于 C-SVC，有些时候甚至会陷入局部最有优（长时间网络误差无法收敛到零），需要重新给网络分配随机权值和偏置值来再次执行算法。除此以外，神经网络隐含层数量和感知器数量的选择也缺乏理论根据，学习速度常数的选取也需要调试，太小会使得网络收敛时间不能接受，太大会使算法发散。而支持向量机在选定了核函数以后就可以固定求出凸规划问题的最优解，不但保证了解的全局最优性，而且不需要考虑参数的选取，避免了 BP 神经网络中的一些经验调试成分。但是，BP 神经网络算法易于实现，涉及到的理论知识较少。另外，C-SVC 只给出了两类分类问题的求解，对于多类分类问题还需要构造一系列两类分类机；但是 BP 神经网络的输出并不局限于两个值，可以直接应用于处理多类分类问题。

7 附录

7.1 BP 神经网络算法实现

以下 C 语言程序是算法 2 在多层感知器网络上的实现：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#ifdef _MSC_VER
#include <windows.h>
#define SLEEP Sleep(1000)
#else
#define SLEEP sleep(1)
#endif

typedef unsigned int uint;
typedef double n_var;

struct neuro {
    n_var s;
    n_var x;
    n_var b;
};

#define LAYER 3
#define MAX_NEURO_PER_LAYER 3
#define LEARNING_RATE 0.3

uint          lm[LAYER + 1] = {2, 3, 3, 1};
struct neuro  neu[MAX_NEURO_PER_LAYER][LAYER + 1];
n_var        w[MAX_NEURO_PER_LAYER][MAX_NEURO_PER_LAYER][LAYER];

n_var
sigmoid(n_var n)
{
    return 1.0 / (1.0 + exp(-n));
}

n_var
n_rand()
{
    return ((n_var)rand())/((n_var)RAND_MAX);
}

void
bpn_init()
{
    uint i, j, l;
    srand(467);

    for (l = 0; l < LAYER; l++)
```

```
        for (i = 0; i < lm[l + 1]; i++) {
            neu[i][l + 1].b = n_rand();
            for (j = 0; j < lm[l]; j++)
                w[i][j][l] = n_rand();
        }
}

void
bpn_layer_out(uint layer)
{
    uint i, j;
    for (j = 0; j < lm[layer + 1]; j ++) {
        neu[j][layer + 1].x = neu[j][layer + 1].b;
        for (i = 0; i < lm[layer]; i++)
            neu[j][layer + 1].x += neu[i][layer].x * w[j][i][layer];

        if (layer != LAYER - 1)
            neu[j][layer + 1].x = sigmoid(neu[j][layer + 1].x);
    }
}

void
bpn_layer_bp(uint layer)
{
    uint i, j;
    n_var sum;

    for (j = 0; j < lm[layer + 1]; j ++) {
        neu[j][layer + 1].b -= LEARNING_RATE * neu[j][layer + 1].s;
    }

    for (i = 0; i < lm[layer]; i++) {
        sum = 0;
        for (j = 0; j < lm[layer + 1]; j ++) {
            sum += neu[j][layer + 1].s * w[j][i][layer];
            w[j][i][layer]  -=
                LEARNING_RATE * neu[j][layer + 1].s * neu[i][layer].x;
        }
        neu[i][layer].s = sum * neu[i][layer].x * (1.0 - neu[i][layer].x);
    }
}

n_var train_data[][3] =
{
    {1, -1, 1},
    {-1, 1, 1},
    {-1, -1, -1},
    {1, 1, -1}
};

#define TRAIN_NUM (sizeof(train_data)/(3*sizeof(n_var)))

int
main()
{
    uint l, i = 0, it = 0;
    n_var y, delta;
    bpn_init();
```

```
    while (1) {
        neu[0][0].x = train_data[i][0];
        neu[1][0].x = train_data[i][1];
        y = train_data[i][2];
        i = (++i)%TRAIN_NUM;

        for (l = 0; l < LAYER; l++)
            bpn_layer_out(l);

        delta = neu[0][LAYER].x - y;
        neu[0][LAYER].s = 2.0 * delta;

        for (l = 0; l < LAYER; l++)
            bpn_layer_bp(LAYER - l - 1);

        printf("[%lf %lf] -> [%lf], y = %lf, delta = %lf \n",
                neu[0][0].x, neu[1][0].x, neu[0][LAYER].x, y, delta);

        if (++it % 500 == 0) {
            printf("training %d ...\n", it);
            SLEEP;
        }
    }

    return 0;
}
```

7.2 Matlab 命令生成脚本

对于 7.1 节给出的程序，可以通过以下 Bash 脚本从程序标准输出的 log 文件里得出用于输出图 5 的 Matlab 命令：

```
#!/bin/bash
x=`awk '{x += 1; printf("%d ", x);}' bpn.log`
y=`awk '{printf("%s ", $NF);}' bpn.log`
echo "y=[$y];x=[$x];plot(x,y);"
```

参考文献

[1] 维基百科，机器学习[EB/OL]. https://zh.wikipedia.org/wiki/机器学习,2013-03-07/2013-04-25.

[2] Kevin P. Murphy. Machine Learning: A Probabilistic Perspective[M]. Cambridge, Massachusetts: The MIT Press, 2012.

[3] 邓乃杨，田英杰. 数据挖掘中的新方法：支持向量机[M].北京:科学出版社，2004.

[4] 邓乃杨,田英杰.支持向量机:理论、算法与拓展[M].北京:科学出版社,2009.

[5] Nocedal Jorge, Wright Stephen J. Numerical Optimization (2nd ed.)[M]. Berlin, New York: Springer-Verlag, 2006.

[6] pluskid. 支持向量机: Support Vector [EB/OL].

http://blog.pluskid.org/?p=682, 2010-09-10/2013-04-25.

[7] Stephen Boyd，Lieven Vandenberghe. Convex Optimization [M]. Cambridge, New York: Cambridge University Press, 2004.

[8] 蒋宗礼.人工神经网络导论[M].北京:高等教育出版社，2001.

[9] Simon Haykin. Neural Networks - A Comprehensive Foundation, Second Edition[M]. Person Education, 2005.

[10] Martin T. Hagan, Howard B. Demuth, Mark H. Beale. 神经网络设计[M]. 戴葵等译. 北京:机械工业出版社,2002,9. 201-205.