文章编号:1001-9081(2007)09-2262-05

一种改进的动态二叉树的自组织神经网络算法

张群洪[1,2],陈崇成[1,2]
(1.福州大学 福建省空间信息工程研究中心,福州 350002;
2.福州大学 数据挖掘与信息共享教育部重点实验室,福州 350002)
(nrisland@yahoo.com.cn)

摘 要:分析了自组织神经网络各种改进算法的优缺点,详细设计和实现了一种基于改进动态二叉树的自组织映射树(DBTSONN)。在改进动态二叉树中神经元节点可以自动生长和剪除,无需在训练前预先确定自组织神经网络结构。DBTSONN1算法采用单路径自组织树中搜索最匹配叶节点(获胜神经元),DBTSONN2算法考虑了获胜神经元节点所在自组织二叉树的层次,采用双向搜索获胜叶节点,提高了搜索效率。实验结果表明,该算法在向量量化器设计方面具有很好的效果。

关键词:自组织神经网络;动态二叉树;双向搜索机制;算法实验
中图分类号:TP391.9 **文献标志码**:A

Improved dynamical binary-tree based self-organizing neural network algorithm

ZHANG Qun-hong[1,2], CHEN Chong-cheng[1,2]
(1. *Spatial Information Research Center of Fujian, Fuzhou University, Fuzhou Fujian* 350002, *China*;
2. *Key Laboratory of Data Mining & Information Sharing, Ministry of Education, Fuzhou University, Fuzhou Fujian* 350002, *China*)

Abstract: The advantages and disadvantages of various improved self-organizing neural network algorithms were discussed in the paper, and an Improved Dynamical Binary-tree Based Self-Organizing Neural Network (DBTSONN) was designed and implemented in detail. In the binary-tree, neuron nodes can be growing and pruning, and the self-organizing mapping structure is flexible, not needed to be determined in advance. DBTSONN1 algorithm uses single path to search the winning leaf nodes, and DBTSONN2 algorithm uses double path search, considering the hierarchical position of the winning node, which can improve the searching efficiency. The experimental results show that DBTSONN algorithm is very useful for vector quantization.

Key words: self-organizing neural network; dynamic binary-tree; double path search; algorithm experiment

0 引言

最近几年,多层次树结构自组织神经网络引起广泛的关注,许多学者提出了多种基于自组织神经网络的层次聚类算法。文献[1]提出一种层次树型细胞生长结构(TreeGCS)该模型首先使用细胞生长结构(GCS)处理输入样本,然后在细胞阵列基础上构建层次树型结构,TreeGCS层次结构是一棵二叉树。文献[2]在SOM和GCS基础上提出了一种无监督的自组织树型(SOTA)算法。SOTA的拓扑结构也是二叉树型结构,其构建的层次结构无法真实表示输入数据的内在关系。而且,SOTA算法中,映射到上层次节点的输入样本只能映射到该节点的孩子节点上,也就是说,早期阶段的误分类,随着层次的增长仍然存在,无法在后期阶段得到调整。文献[3]提出自组织树图(SOTM)算法。在该神经元网络中,当输入样本与获胜神经元权重向量的距离超过层次控制因子,该树结构将生成一个新的神经元节点。该算法能够最小化输入数据空间的矢量量化误差,但不能用于寻找输入数据内在层次结构,而且树中高层次节点和低层次节点没有层次关系。文献[4]提出一种结构自适应自组织神经网络,该算法通过交错检验几个不同结构网络的方法最终可以得到一个好的网络结构。但是这种方法在网络训练的过程中并不能提供很大的灵活性,而且也不是高效的。

树型结构自组织映射图(Tree-Structured Self-Organizing Map,TS-SOM)[5,6]是由几个传统自组织映射图构成,该算法训练过程是逐层进行的,在每层自组织映射图中对神经元的训练和传统的SOM一样,但是搜索获胜神经元的方法不同。在搜索第k层最匹配神经元时,只需要在第$k-1$层的最匹配神经元的孩子节点中寻找即可,这样大大减少了传统SOM在寻找获胜神经元的计算时间,这也是TS-SOM算法的最大优点。但由于该算法,每层中神经元结构、布局和数量都是固定的,所以属于静态构造的自组织神经网络[7],因此该算法的用途不是很广泛,特别是在解决复杂数据缺乏灵活度。综合上述学者对多层次树结构自组织神经网络的各种改进算法的优缺点,本文设计和实现了一种基于动态二叉树的自组织映射树(DBTSONN),树中神经元节点可以自动生长,无需在训练前确定神经网络结构的大小,动态二叉树是一种层次结构,搜索最匹配神经元的效率较高,结构灵活。对于每个输入样本,DBTSONN1算法采用单路径自组织树中搜索最匹配叶节

收稿日期:2007-03-12;修回日期:2007-06-04。
基金项目:国家自然科学基金资助项目(60602052);福建省重点科技项目(2005H086)。
作者简介:张群洪(1977-),男,福建莆田人,博士研究生,主要研究方向:数据挖掘、神经网络、信息系统的分析与设计; 陈崇成(1968-),男,福建闽清人,教授,博士,主要研究方向:资源与环境信息工程、空间信息集成、计算机仿真。

点(获胜神经元),DBTSONN2 算法采用双向搜索获胜叶节点。

1　DBTSONN 算法的基本思想

本文提出的自组织树 DBTSONN 算法是一种结合自组织神经网络和树结构层次分类的算法,构造树型结构的神经元分布,具有自组织神经网络无监督等特性,同时又具有树型层次结构的优点。

DBTSONN 算法把输入样本分成嵌套结构的几个区域,并给位于同一个区域的所有输入样本指定一个权重向量。这些嵌套结构的区域构成一棵树,树中每个神经元节点 j,有一个权重向量 w_j,一个记录该神经元节点获胜次数的计数器 N_j,一个误差值 e_j。

DBTSONN1 采用传统的单方向搜索方式:在每个内部节点,把输入样本向量和该节点的两个孩子节点的权重向量做比较,选择权重向量更接近输入样本向量的孩子节点作为下一步搜索节点,直到找到一个叶节点。接着,DBTSONN 算法对该获胜叶节点进行分裂(splitting)或剪除(pruning)操作,从而调整自组织树的结构:如果这个获胜叶节点误差值太大,则该获胜叶节点将分裂生成操作新节点;同样如果这个获胜叶节点误差值太大,而且自组织树中的节点数目超过预定的值,则进行剪除操作。按照这种树调整方法,不断调整连接根节点与获胜叶节点路径上的神经元的权值,以反映当前输入样本。DBTSONN 通过判断总误差值 C 来确实是否结束算法:当总误差值 C 变化不大时,算法结束,否则对输入样本进行新一轮的学习训练。

在某种意义上,DBTSONN1 算法中用来分裂或剪除操作的叶节点总是最接近当前输入样本的获胜叶节点,这往往形成某些叶节点总是该层次上的获胜叶节点,自组织树结构失去严重平衡,很容易导致误分类或权值的调整没有向聚类中心靠近。所以本文提出了双路径搜索 DBTSONN2 算法,它不是总是选择最接近输入样本的叶节点作为最终获胜叶节点,而且考虑节点在自组织树所在的层次,采用双路径进行比较搜索。

在训练过程中,根据输入样本,搜索自组织树直到找到叶节点,这时,这个输入样本被映射到这个叶节点神经元,从而实现了聚类,并根据输入样本和叶节点神经元的权重向量计算该节点误差值 e_j。

2　DBTSONN 算法的基本操作

2.1　生长

自组织树最初只有一个根节点神经元,根据输入样本不断增加两个子节点,直到达到预定的最大节点数目 U。接着 DBTSONN 算法在增加新的子节点前,必须在自组织树中删除两个子节点。当获胜叶节点 J 的误差值 e_j 超过预定的分裂阈值 E 时,该节点分裂生成两个子节点,这两个新生成的叶节点按下面方式进行初始化。和其他决策树方法相比,DBTSONN 算法并不需要完整搜索所有的叶节点,以确定哪个叶节点即将分裂。

1) 左孩子节点的权重向量初始化为 w_J,右孩子的权重向量初始化为 $(1+\sigma)w_J$,其中 w_J 为获胜叶节点 J 的权重向量,σ 为一个小的正常数;

2) 每个孩子的计数器初始化为 1;

3) 每个孩子的误差值初始化为 $e_{J/2}$。

2.2　剪除

自组织树以一种贪婪的方式生长神经元,即在叶节点的误差值超过预定的分裂阈值 E 时尽可能生长神经元,而当树中的神经元数目已经达到预定的值 U 时,通过剪除机制可以减少错误分裂造成的影响,剪除机制可以当作分裂机制的补充。从自组织树删除具有最小误差值的节点使得树中的神经元数目减少,这样自组织树中误差值很大的获胜叶节点可以继续进行分裂生长新的神经元节点。

对于每个输入样本,当树中的神经元数目已经达到预定的值 U,而且获胜叶节点的误差值超过预定的分裂阈值 E 时,即 $e_J > E$,DBTSONN 算法寻找具有最小误差值 e_m 的叶节点,当 e_m 足够小,而且 $e_m/e_J < \Gamma$,Γ 是剪除阈值,则剪除叶节点 m 以及另外一个相关的节点。同时获胜叶节点 J 按原来方式分裂生成新的节点。

剪除叶节点 m 需要考虑三种情况:

1) 剪除叶节点 m 的兄弟节点不是叶节点(如图 1),删除叶节点 $m=4$ 以及其父节点 2,并把叶节点 m 的兄弟节点 5 调整到其父节点的位置。树中获胜叶节点 6 按原来方式分裂生成新节点 10 和 11,重新计算输入样本新的获胜叶节点 $J=10$。

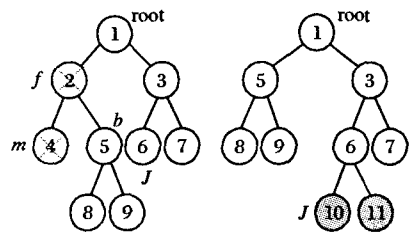


图 1　情况 1: m 的兄弟节点不是叶节点

2) 剪除叶节点 m 的兄弟节点是叶节点,并且该节点是获胜节点 J(如图 2),即剪除叶节点 m 与获胜节点 J 是兄弟节点。此时删除叶节点 $m=8$,以及其兄弟节点 $J=9$,并选择其父节点 5 为分裂节点,生成新节点 10 和 11,并重新计算获胜节点 $J=10$。

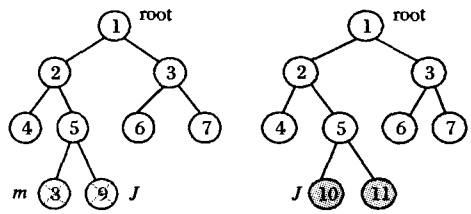


图 2　情况 2: m 和 J 是兄弟节点

3) 剪除叶节点 m 的兄弟节点是叶节点,但不是获胜节点,如图 3,此时删除叶节点 $m=9$,以及其兄弟节点 8。树中获胜叶节点 6 按原来方式分裂生成新节点 10 和 11,重新计算输入样本新的获胜叶节点 $J=10$。同时,剪除叶节点 m 的父节点 5 的误差值需要除 2 减少父节点的误差值以使自组织树如实地反映输入样本,否则自组织树会陷入总是在同一个节点剪除或分裂的局部循环。

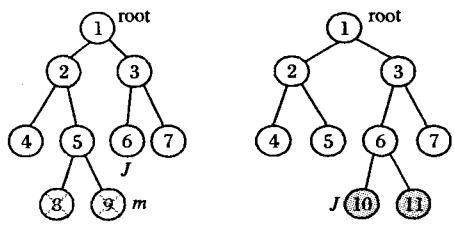


图 3　情况 3: m 的兄弟节点是叶节点但不是 J

2.3 收敛条件

自组织神经网络常用的训练收敛判定条件有两个：

1) $\frac{|C_t - C_{t-1}|}{C_{t-1}} < \varepsilon$，其中 C_t 是第 t 次学习循环中所有训练样本获胜叶节点的总误差，这种方法保证了连续两次循环的总误差波动控制在一个指定的收敛阈值 ε 范围之内。

2) $\frac{|\bar{C}_t - \bar{C}_{t-1}|}{\bar{C}_{t-1}} < \varepsilon$，其中 $\bar{C}_t$ 是第 t 次学习循环中所有训练样本获胜叶节点的平均误差，这种方法保证了连续两次循环的平均总误差波动控制在一个指定的收敛阈值 ε 范围之内。但这两种训练收敛标准经常存在的一个问题是：当连续两次训练循环中出现局部满足收敛条件，则训练结束。本文提出一种所有训练样本获胜叶节点的修正的移动平均误差，采用该方法可以平滑误差值，达到尽可能的整体最优。

$$\bar{C}_t = \begin{cases} \bar{C}_{t-1} + \beta_3 \times (\bar{C}_t - \bar{C}_{t-1}), & t > 1 \\ \bar{C}_t, & t = 1 \end{cases}$$

其中：$\bar{C}_t$ 为 t 次学习循环中所有训练样本获胜叶节点的修正的移动平均误差，$\bar{C}_t$ 为 t 次学习循环中当前训练所有获胜叶节点的平均误差，β_3 为调整参数。

2.4 参数调整

在自组织树形成过程中，对输入样本划分的原则是保证神经元权值的分布尽可能反映输入样本的分布，即树中所有叶节点将以同样的几率成为获胜神经元。本算法采用误差值 e_j 最大者为获胜神经元，并使用计数器 N_j 来记录该神经元节点获胜次数。当误差度量值超过预定的分裂阈值 E 时，该胜神经元将分裂。所以误差度量值的计算非常重要。

DBTSONN 模型的构造与具体的应用有关，但在很多应用场合，通常采用下式来计算总的误差度量，并尽量使 D 最小。

$$D = \sum_j \sum_{i=1}^{N_j} \| A_i - w_j \|^2$$

其中，A_i 是映射到叶节点 j 的输入样本的集合，N_j 为神经元叶节点 j 获胜次数，在计算 N_j 需要注意的是对于每个输入样本向量 A，自组织树每层都有一个获胜神经元节点。

实现总误差度量值 D 最小，通常的做法令 $e_j = \| A - w_j \|^2$，尽量使每个节点的误差最小，以实现整体 D 最小。然而误差 e_j 在训练的早期阶段要远大于收敛阶段，即每个节点的误差主要取决于早期阶段，对初始条件很敏感，所以需要提出一种有效的方法来计算误差值 e_j。本算法采用一种相对值[8]来计算误差值 e_j，即对于获胜叶节点 J：

μ_J 为获胜叶节点的误差值，按式 $\mu_J = \| A - w_J \|^2$ 计算。

$\tilde{\mu}_J$ 为获胜叶节点修正的加速平均误差值，按下式修改 $\tilde{\mu}_J$：

$$\Delta\tilde{\mu}_J = \begin{cases} \beta_2 \times (\mu_J - \tilde{\mu}_J), & N_J > 1 \\ \mu_J, & N_J = 1 \end{cases}$$

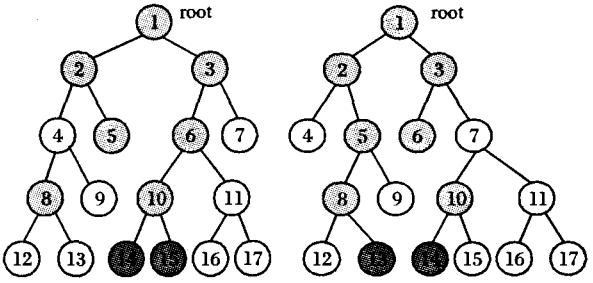
$\hat{\mu}$ 为获胜叶节点的相对误差值，按式 $\hat{\mu} = \mu_J / \tilde{\mu}_J$ 计算，μ 为本算法将采用的相对误差值。即按 $\Delta e_j = \hat{\mu}_J$ 来调整 e_j。在训练早期阶段，连接权值变化比较快时，μ_J 比较大，$\tilde{\mu}_J$ 也比较大；而在训练的接近收敛阶段，μ_J 比较小，$\tilde{\mu}_J$ 也比较小，所以相对误差 $\hat{\mu}$ 能够有效反映各个节点的误差值。

2.5 双路径搜索

DBTSONN 算法的目标是使树结构自身约束导致的分类偏离最小化。DBTSONN2 采用双通道搜索最接近当前输入样本的叶节点。在训练过程中，对每个输入样本向量，首先选择根节点作为最初获胜神经元，如果该节点没有孩子节点，则该节点作为最终获胜神经元，算法结束。否则选择该节点的两个孩子节点作为当前获胜神经元，并把当前输入样本向量和当前两个获胜神经元的四个孩子节点的权重向量进行比较，从中选择最接近当前输入样本的两个孩子节点作为自组织树中下一个层次的两个获胜神经元。这个过程不断进行下去，直到当前两个获胜神经元都为叶节点，而最终获胜神经元的选择与自组织树中节点的数目有关。

如果自组织树中的节点数目达到最大值 U，则不管当前两个获胜神经元位于树中的哪个层次，总是选择接近输入样本向量的节点作为最终获胜神经元。而当自组织树还处于增长阶段，则分两种情况：如果当前两个获胜神经元都位于自组织树中的同一层，则选择接近输入样本向量的节点作为最终获胜神经元；否则，即当前两个获胜神经元位于自组织树中的不同层，则选择深度小的节点作为最终获胜神经元。

图 4 说明了双路径搜索的两个例子，浅灰色表示每个层次上的获胜节点，深灰色表示自组织树的获胜叶节点，图 4(a)表示两个获胜叶节点 14 和 15 在自组织树的同一侧；图 4(b)表示两个获胜叶节点 13 和 14 在自组织树的不同侧。



(a) 两个获胜叶节点在同一侧　(b) 两个获胜叶节点在不同侧

图 4　自组织树中双路径搜索的两种方式

3 DBTSONN 算法的基本流程与实现

3.1 基本流程

DBTSONN 神经网络是一种通过增量学习来调整权重向量的树型结构聚类算法，算法过程如图 5。

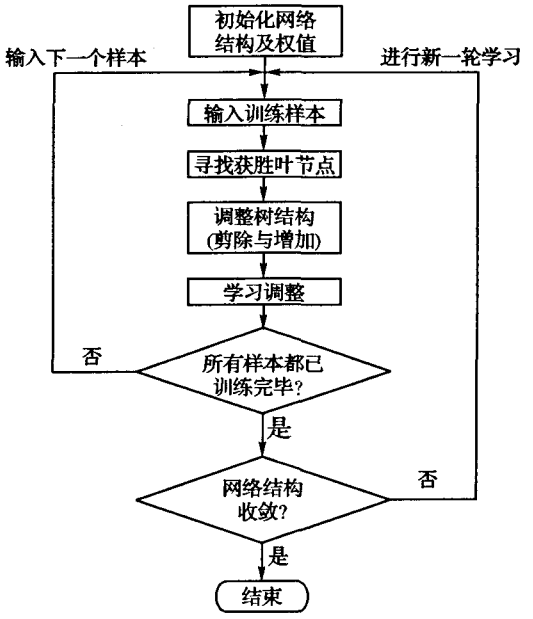


图 5　DBTSONN 算法流程

3.2 实现

对于每个输入样本向量 A，自组织树每层都有一个获胜神经元节点，找到最终获胜叶节点 J，首先判断是否需要调整自组织树的结构，接着修改从获胜叶节点 J 到根节点路径上的各个节点的参数，包括权重向量 w_j、获胜计数器 N_j 以及误

差度量值 e_j,即 DBTSONN 算法中获胜神经元 J 的邻域包括从获胜叶节点 J 到根节点路径上的所有节点,在一个学习循环中所有训练样本都已训练完毕,判断算法学习是否已经收敛。算法中使用到的参数和变量定义如表 1 和表 2。

表 1　DBTSONN 算法中需要用到的参数

参数	描述
E_0	分裂阈值 E 的初始值,取 50
β_1	分裂阈值 E 的调整值,取[0.01,0.02]
β_2	获胜叶节点 J 修正的平均误差的调整值,取 0.075
β_3	所有获胜节点修正的移动平均误差的调整值,取 0.2
γ	修改自组织结构(分裂)后的调整值,取[1.1,1.5]
Γ	剪除阈值,取[0.35,0.4]
ε	收敛阈值,取 0.01
U	自组织树的最大节点数,是个奇数
N	学习训练样本总数

表 2　DBTSONN 算法中需要用到的变量

变量	描述
A	输入向量($x_1 \cdots x_i \cdots x_M$)
w_j	树中第 j 个节点的权重向量($w_{1j} \cdots w_{ij} \cdots w_{Mj}$)
A_j	映射到叶节点 j 的输入样本的集合
e_j	树中第 j 个节点的误差值($cost$)
μ_J	树中获胜叶节点 J 的误差值
$\tilde{\mu}_J$	树中获胜叶节点 J 修正的加速平均误差值
$\hat{\mu}_J$	树中为获胜叶节点 J 的相对误差值
N_j	树中第 j 个节点获胜神经元的次数
P_j	树中第 j 个节点父节点的标号,根节点 $P_j = 0$
S_j	树中第 j 个节点兄弟节点的标号,根节点 $P_j = 0$
θ_j	树中第 j 个节点两个子节点的集合
Ω	树中从根节点到获胜叶节点路径上所有节点的集合
γ	树中所有叶节点的集合
l_j	树中第 j 个节点的深度
E	分裂阈值
u	当前树中节点的数目
t	学习循环的次数,每次学习循环中训练所有样本
C	当前学习循环中所有获胜节点的总误差
$\bar{C}_t$	第 t 次循环中当前训练所有获胜叶节点的平均误差
$\bar{\bar{C}}_t$	第 t 次循环中所有获胜节点修正的移动平均误差

主算法:

1) 初始化:自组织树中只有一个根节点,并设置:
$t = 1$; $u = 1$; $w_j = 0$; $N_j = 0$; $e_j = 0$; $\bar{\mu}_j = 0$; $E = E_0$; $C = 0$; $P_1 = 0$;

2) 输入样本:样本向量 $A = (x_1 \cdots x_i \cdots x_M)$;

3) 寻找叶节点:调用子函数 *single-path-search*( ) 或 *Double-path-search*( ) 寻找获胜叶节点 J;

4) 修改自组织树结构:调用子函数 *Adapt-Structure*( ) 分裂或剪除操作;

5) 修改从获胜叶节点 J 到根节点路径上各个节点的参数:调用子函数 *Modify-Parameter*( );

6) 按下式修改当前学习循环中所有获胜节点总误差 C:
$\Delta C = \| A - w_J \|^2$

7) 返回第 2 步,选取另一个学习模式提供给网络的输入层,直至所有样本全部训练结束;
检验收敛:

8) 按下式计算第 t 次学习循环中所有获胜节点的平均误差:
$\bar{C}_t = C/N$;

9) 按下式计算第 t 次学习循环中所有获胜节点的修正的移动平均误差:

$$\bar{\bar{C}}_t = \begin{cases} \bar{\bar{C}}_{t-1} + \beta_3 \times (\bar{C}_t - \bar{\bar{C}}_{t-1}), & t > 1 \\ \bar{C}_t, & t = 1 \end{cases}$$

10) if($t > 1$) { If $\left(\dfrac{|\bar{\bar{C}}_t - \bar{\bar{C}}_{t-1}|}{\bar{\bar{C}}_{t-1}} < \varepsilon \right)$ 算法结束; }

11) $t = t + 1$;

12) $C = 0$,返回第 2 步,进行新一轮训练学习;

```
// 功能:修改获胜节点到根节点路径上各个节点
// 的各种参数
Modify-Parameter( )
{
```

1) 计算获胜叶节点 J 的误差值 μ_J 以及修正的加速平均误差值

$\tilde{\mu}_J, \mu_J = \| A - w_J \|^2$,

$$\Delta\bar{\mu}_J = \begin{cases} \mu_J, & N_J = 1 \\ \beta_2 \times (\mu_J - \bar{\mu}_J), & \text{其他} \end{cases}$$

2) 计算获胜叶节点的相对误差值:
$\hat{\mu}_J = \mu_J / \bar{\mu}_J$

3) 按下式调整自组织树中节点的分裂阈值:
$\Delta E = \beta_1 \times (e_J - E)$

4) 令 Ω 为树中从根节点到获胜叶节点路径上所有节点的集合;

5) 按下式调整自组织树中节点的误差值 e_j:

$$\Delta e_j = \begin{cases} \hat{\mu}_J, & i \in \Omega \\ 0, & i \notin \Omega \end{cases}$$

6) 按下式调整自组织树中节点的计数器 N_j:

$$\Delta N_j = \begin{cases} 1, & i \in \Omega \\ 0, & i \notin \Omega \end{cases}$$

7) 按下式调整自组织树中节点的连接权重 w_j:

$$\Delta w_j = \begin{cases} (A - w_j)/N_j, & i \in \Omega \\ 0, & i \notin \Omega \end{cases}$$

```
}
// 功能:调整自组织树的结构,包括分裂与剪除
Adapt-Structure( )
{
  if(e_J > E && u < U)                    // 分裂
  {
    调用子函数 Splitting( ) 进行分裂操作,增加两个子节点;
    调整分裂阈值:E = E × γ;
  }
  if (u == U)          // 剪除后需要进行分裂操作
  {
    设 γ 为自组织树中所有叶节点的集合;
    m = min{e_j} j ∈ γ
    // 在自组织树中寻找具有最小平均误差 e
    // 的叶节点,并设为 m
    if(e_m/e_J < Γ)
    {
      调用子函数 Pruning( ) 进行剪除操作;
      调用子函数 Splitting( ) 进行分裂操作,增加两个子节点;
      调整分裂阈值:E = E × γ;
    }
  }
}
// 功能:在自组织树中单路径搜索获胜叶节点,
// 并保存在 J 中
Single-path-search( )
```

```
{
  J = 1;                                   // 根节点
  While(J 不是叶节点)
  {
    设 θ_J 为自组织树中获胜叶节点 J 的两个孩子节点的
    集合;
    k = min{A - w_j}, j ∈ θ_J
    J = k;
  }
}
// 功能:在自组织树中双路径搜索获胜叶节点,
// 并保存在 J 中
Double-path-search()
{
  J = 1;                                   // 根节点
  If(U == 1) return;
  分别设置 J_1 和 J_2 为根节点的左孩子和右孩子节点;
  While(J_1 不是叶节点或者 J_2 不是叶节点)
  {
    如果J_1 没有孩子节点,则令 ψ_1 = {J_1},
       否则 ψ_1 = θ_J1;
                    //θ_J1 表示 J_1 的两个孩子节点;
    如果J_2 没有孩子节点,则令 ψ_2 = {J_2},
       否则 ψ_2 = θ_J2
    //θ_J2 表示 J_2 的两个孩子
        { J_1 = arg min{ ‖A_j - w_j‖ },
          j ∈ {ψ_1 ∪ ψ_2}
    J = { J_2 = arg min{ ‖A_j - w_j‖ },
          j ∈ {ψ_1 ∪ ψ_2} and
          {j ≠ J_1}
  }
  令 l_J1 为叶节点 J_1 在自组织树中的深度;
  令 l_J2 为叶节点 J_2 在自组织树中的深度;
      { J_1,    {l_J1 ≤ l_J2} or {u == U}
  J = { J_2,    其他
  }
  // 获胜叶节点 J 分裂生成两个孩子节点,
  // 并调整网络结构
Splitting()
  { θ_J(1) = u + 1, θ_J(2) = u + 2;
    //θ_J(1) 表示获胜叶节点 J 的第一个孩
    // 子节点的标号
    对于 J 的两个孩子节点,即 j ∈ θ_J,令
      N_j = 1, e_j = e_J/2;
    w_i,θJ(1) = w_i,J;
    w_i,θJ(2) = (1 + δ) × w_i,J
    u = u + 2;
    J = θ_J(1);                    // 重新选择获胜叶节点
  }
Pruning()
  { 设 P_m 和 S_m 分别是 m 节点的父节点和兄弟节点;
      // 情况 1:剪除叶节点 m 的兄弟节点不是叶节点
    if(S_m 不是叶节点)
    {
      Z = P_Pm;
          //Z 为即剪除叶节点 m 的父节点的父节点
      Delete m 和 P_m;
      用 S_m 分支代替 P_m 的位置;
      for(q = 1;q <= 2;q++)
      if(θ_Z(q) == P_m) θ_Z(q) = S_m;
                    //S_m 成为替代其父节点的位置
    }
    // 情况 2:剪除叶节点 m 的兄弟节点是叶
    // 节点,并且该节点是获胜节点 J
    else if (S_m 是叶节点,并且是获胜节点 J)
    { Delete m 和 J;
      θ_Pm(1) = 0, θ_Pm(2) = 0;
      // 删除后,剪除叶节点 m 的父节点
      //P_m 没有孩子节点
      J = P_J                      // 得到新的获胜叶节点
    }
    // 情况 3:剪除叶节点 m 的兄弟节点是
    // 叶节点,但不是获胜节点
    else if(S_m 是叶节点,但不是获胜节点)
    { Delete m 和 S_m;
      e_Pm = e_Pm/2;
      θ_Pm(1) = 0, θ_Pm(2) = 0;
      // 删除后,剪除叶节点 m 的父节点
      //P_m 没有孩子节点
    }
  }
```

4 DBTSONN 算法验证与分析

图像矢量量化技术在图像压缩中有着广泛的应用,在向量量化器是设计中,码书的组织对向量匹配搜索算法的影响极大,一般利用训练集生成码书,通过全搜索算法找出一个与输入向量最接近的匹配向量。但是,其计算量大且无快速算法,为了减少搜索算法的计算量,文献[9,10]提出了树型结构向量量化方法(Tree-structured vector quantization, TSVQ)。但该方法构造的树型结构不够灵活,算法运行浪费大量的资源。本文将 DBTSONN 用于向量量化,可以根据样本数据自动分裂和删除神经元,并且该方法是一种在线向量量化方式,在训练过程中不需要把所有的数据都读入内存。

本文采用的数据来自 Gauss-Markov 数据[10],包括 60 000 输入向量。量化重构效果使用 PSNR[11] 来度量,PSNR = 10 lg (256^2/MSE),其中 MSE 是重构的误差平方的平均值。该算法用于向量量化器设计,其运行性能比较如表 3 所示。从中可以看出,DBTSONN 算法比起 TSVQ 算法,以及静态树结构自组织神经网络 TS-SOM 算法具有明显的优势。

表 3 几种算法的运行性能比较(dB)

码书大小	TSVQ	TS-SOM	DBTSONN1	DBTSONN2
2	17.6	17.6	17.7	17.7
4	21.2	21.2	21.3	21.2
8	22.6	22.6	22.4	22.6
16	23.2	23.4	23.2	23.5
32	23.7	24.7	24.2	24.5
64	24.8	25.5	25.1	25.3
128	25.5	26.2	25.8	26.0

5 结语

本文首先介绍了当前多层次自组织神经网络的主要研究进展,特别是树型自组织神经网络算法,分析这些算法的主要

(下转第 2297 页)

D 代表预测流失,实际流失的客户。

预测命中率:

$$\frac{D}{C+D}\times 100\%$$

预测覆盖率:

$$\frac{D}{B+D}\times 100\%$$

根据电信公司商业成本核算的结构,要求预测覆盖率是达到 75% 以上,预测命中率在 65% 以上。

使用 16211 条测试数据(其中包括 *F0A* 用户 13984 条,*F0K* 用户 310 条,*F0J* 用户 95 条,*F0L* 用户 74 条,*F0X* 用户 1748 条)对客户流失模型进行评估,评估结果如表 2 所示。

表 2　评估结果表

客户流失评估项	预测不流失	预测流失	合计
实际不流失	13642	342	13984
实际流失	224	2003	2227
合计	13866	2345	16211

3.3　结果分析

从模型评估结果可以看出,模型的预测命中率为 85.42%;预测覆盖率为 89.94%。从行业标准来看,模型已经达到了要求,可以投入使用。

基于神经网络的客户流失预测与文献[7]中的其他预测方法比较结果如图 5 所示。

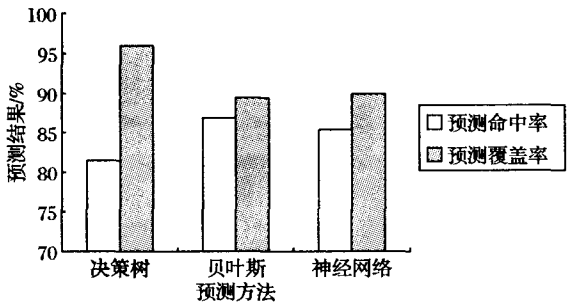


图 5　各种预测方法预测结果对比

从图 5 可以看出,基于决策树的客户流失预测预测覆盖率最高但是预测命中率低,基于贝叶斯网络的客户流失预测和基于神经网络的客户流失预测在预测命中率和预测覆盖率方面效果类似。但是在数据分析过程中,它们的侧重点各有不同。贝叶斯网络方法建模是在普通样本(含已流失和未流失数据)中直接抽取进行建模,而神经网络方法采用的建模数据是电信行业中已流失的客户数据,流失客户的数据量越大,建立的模型越精确。从模型的训练角度来看,数据量越大,对模型的训练效果越好,因此使用大规模流失数据的神经网络方法具有更好的实用性。

4　结语

基于神经网络的客户流失预测模型在应用中还需要及时更新,因为模型的训练是基于一个时间段内的数据进行的。这个模型往往代表了这段时间内的用户的消费习惯和消费结构。因此用模型预测时,其时效性是明显的。当市场环境发生变化,用户的行为发生改变时,模型也就需要及时更新,使用新的数据进行训练,不断进行修正和完善以保证其有效性,随着训练样本的增大,本模型在预测命中率和预测覆盖率方面还将有进一步提高。

参考文献:

[1] HUNG S Y, YEN D C, WANG H Y. Applying data mining to telecom churn management [J]. Expert Systems with Applications, 2006, 31(3): 515 - 524.

[2] MATTERSION R. Telecom churn management[D]. Fuquay-Varina: APDG Publishing, 2001.

[3] WEI C P, CHIU I T. Tuning telecommunications call detail to churn prediction: A data mining approach [J]. Expert Systems with Applications, 2002, 23: 103 - 112.

[4] LEJEUNE M. Measuring the impact of data mining on churn management [J]. Internet Research: Electronic Network Applications and Policy, 2001, 11(5): 375 - 387.

[5] 郭明,郑惠莉,卢毓伟. 基于贝叶斯网络的客户流失分析[J]. 南京邮电大学学报: 自然科学版, 2005, 25(5): 79 - 83.

[6] 叶进,张向利,张润莲. 基于数据挖掘的移动客户流失分析系统[J]. 计算机系统应用, 2005, (2): 61 - 64.

[7] 叶进,林士敏. 基于贝叶斯网络的推理在移动客户流失分析中的应用[J]. 计算机应用, 2005, 25(3): 673 - 675.

(上接第 2266 页)

优缺点。在各种改进算法基础上,本文设计和实现了一种基于动态二叉树自组织神经算法 DBTSONN,并提出双路径搜索 DBTSONN2 算法,它不是总是选择最接近输入样本的叶节点作为最终获胜叶节点,而且考虑节点在自组织树所在的层次,采用双路径进行比较搜索,该算法具有接近线性时间复杂度,较好的效率。实验结果表明,该算法在性能方面取得了较好的结果。

参考文献:

[1] HODGE VJ, AUSTIN J. Hierarchical growing cell structures: TreeGCS [J]. IEEE Transactions on Knowledge and Data Engineering, 2001, 13(2): 207 - 218.

[2] DOPAZO J, CARAZO J M. Phylogenetic Reconstruction Using an Unsupervised Growing Neural Network That Adopts the Topology of a phylogenetic Tree [J]. Journal of Molecular Evolution, 1977, 155: 226 - 233.

[3] KONG H S, GUAN L. Self-organizing tree map for eliminating impulse noise with random intensity distributions[J]. Journal of Electronic Imaging, 1998, 7(1): 36 - 55.

[4] 吴郢, 阎平凡. 结构自适应自组织神经网络的研究[J]. 电子学报, 1999, 27(7): 55 - 58.

[5] KOIKKALAINEN P. Progress with the tree-structured self-organizing map[C]// COHN AG, ed. 11th European Conference on Artificial Intelligence, European Committee for Artificial Intelligence (ECCAI). New York: John Wiley & Sons, 1994.

[6] KOIKKALAINEN P, OJA E. Self - organizing hierarchical feature maps[C]// Proceeding IJCNN-90, International Joint Conference On Neural Networks. [S.l.]: IEEE Press, 1990, 2: 279 - 285.

[7] LI T, TANG Y Y, FANG L Y. A Structure-Parameter-Adaptive (SPA) neural tree for the recognition of large character set[J]. Pattern Recognition, 1996, 28(3): 316 - 329.

[8] Riskin E A, Gray R M. A greedy tree growing algorithm for the design of variable rate vector quantizes[C]// IEEE Transactions on Signal Processing. [S.l.]: IEEE Press, 1991: 2500 - 2507.

[9] BUZO A, GRAY AH, GRAY RM, *et al*. Speech coding based upon vector quantization[C]// IEEE Transactions on Acoustics, Speech, and Signal Processing. [S.l.]: IEEE San Jose, 1980: 562 - 574.

[10] MAKHOUL J, ROUCOS S, GISH H. Vector quantization in speech coding[J]. Proceedings of the IEEE, 1985, 73(11): 1551 - 1585.

[11] LINDE Y, BUZO A, GRAY R M. An algorithm for vector quantizer design [J]. IEEE Transaction Commun, 1980, COM-28: 84 - 95.