神经网络 程序设计 面向神经元

计算机科学 1996 Vol. 23 №. 5

63 - 66

面向神经元的程序设计*)

TP311

李 涛 伍良富 袁永斌

(成都科技大学计算机系 成都 610065) (成都电子科技大学)

摘 要 A new programming method—neuron-oriented programming has been presented. The concept of neuron-oriented programming unifies and develops the theory of neural networks and object-oriented.

关键词 Big neuron theory, Object-oriented, Neuron-oriented.

在文[1]中,我们提出了大神经元(即神经网络)原理。大神经元原理指出,神经元、神经网络均是一个特殊的对象,也具有对象的所有性质如抽象、封装、继承和多态。为明确起见,我们在这里扩展有关神经元、神经网络的定义。

定义 1 神经类就是一般的大神经元类。其中,一些特殊的运算如学习、回忆、打印结果、输入输出解释等被封装在大神经元类中。这些特殊的运算也称为神经类的行为、属性、性质或成员函数等。

定义 2 一个神经元就是神经类的一个实例化。

从以上定义可以看出,这里神经元的定义发展了经典神经网络、面向对象的有关理论。换句话说,这里的神经元不仅可以完成神经计算,它同时也能完成经典面向对象理论中对象所能完成的事情。这就是本文提出的面向神经元的程序设计。

1 面向神经元的程序设计

面向神经元的程序设计这一术语有着几方面的含义。一方面,对于程序员来讲,神经元只不过是一种新的高级抽象数据结构的代名词。在这样的抽象数据结构中,数据和运算方法(或行为,或属性,如神经计算等)被很好地封装成了一个整体——神经元(或者说模块,或者说包)。每一个神经元都拥有自己私有的存贮器和局部函数,这种封装的结果将导致信息的模块化和信息屏蔽。

另一方面,神经元实例只不过是一种特殊类型的具体形式。换言之,每一个数据(神经元对象)可以看作是一个类型(神经类)的一个元素。复杂的类型可以对一些简单的类型进行一些例如笛卡儿乘积等数学运算来获得。这样,就引出了子类和超类或父子这样的继承关系。

再者,面向神经元的系统可以看成是一种在大型系统中组织和共享信息的方法。大量各自独立的神经元(连同它们的数据和程序)组织成一个层次图。继承的机制提供了类似神经元共享程序代码和数据的可能性。在执行过程中,若某个神经元调用某一属性(函数),则搜索该属性的过程从该神经元所在层由低向上,选取最靠近该神经元的高层神经元层中含有该属性名的属性执行之。于是,低层中的属性可以屏蔽高层中同样名称的属性。换言之,低层中的神经元可以重新定义高层神经元的动作。这就是所谓多态。

神经类不仅是其实例的集合,同时也是创建这些实例的模板。它规定了在整个生存周期里实例中变量的数量及类型。

简言之,面向神经元的程序设计的思想是经典神经网络、面向对象有关理论的综合。换句话说,这里的神经元不仅可以完成神经计算,同时也能完成经典面向对象理论中对象所能完成的事情。

在面向神经元程序设计中,继承包含以下几个方面的意义:

· **类型继承**。儿子神经元与其父亲神经元具有相类似的拓扑结构。这里,儿子神经元包含了所有父亲神经元的局部变量,另外,它也可以拥有其父亲神经元没有的实例变量。

· **外部接口继承**。儿子神经元和其父亲神经元具有类似的外部行为。儿子神经元从其父亲神经元那里获得其父亲所有可见的行为或函数,同时,它还

*)国家教委出国留学回国人员基金及成都科技大学青年教师基金资助项目

可以提供自己独特的一些行为。

·**代码共享**。这里儿子神经元可以使用其父亲神经元所定义的行为。这样就避免了重复代码的生成。于是，复杂的神经元可以由一些相对简单的神经元构成。

·**多态**。一个特定的运算可以从其具有类似性质的祖先运算那里派生[illegible]，例如：大多数学习算法都可从 Hebb 学习规则派生出来。

2 继承

在文[2]中我们定义了一个面向神经元的程序设计语言 ONPL。在 ONPL 中，有三种类型的神经类：即标准简单神经元 NEURON，用户自定义简单神经元以及大神经元(神经网络)。其中 NEURON 是这样一个神经元：

$$\begin{cases} y=f(\sum_{j=1}^{n}\omega_j x_i-\theta) \\ f(x)=1/(1+e^{-x}) \end{cases}$$

f 称为状态转移函数。x_i，ω_i 为神经元的输入和权值，θ 为该神经元的阈值。

倘若你希望定义其它种类的神经元，可以利用神经元定义语句来完成，神经元定义语句的格式如下：

```
NEURON—DEF <名字>:[<父亲名>]
  '{'[<变量声明>];
    BEHAVIOR:{<函数声明>}.
  '}'
```

注意，这里继承是单继承方式，即儿子神经元只能拥有一个父亲神经元。

下面是标准神经元在 ONPL 中的定义。

```
CONCURRENTCLASS NEURON
  {PUBLIC;
    float threshold,state;
    struct input
      {float w;
      int v;
      input * next;
      };
    neuron(…);
    virtual float get_state();
    virtual float get_threshold();
    virtual float get_weight(int v);
    virtual void update_threshold(float t);
    virtual void update_weight(neuron *,float);
    virtual void transfer();
    …
  }
```

CONCURRENTCLASS 是 ConcurrentC++[3]关键字，意在定义一个并发类。它指出在 ONPL 中，各神经元是并发执行的。

如果你想定义一个具有二值输出的神经元，可以这样来实现。

```
NEURON_DEF special_neuron:NEURON
  {BEHAVIOR;
   virtual void transfer()
     {NEURON.transfer();
       if state>=0 state=1
       else state=0;
     }
  }
```

special_neuron 称为 NEURON 的儿子，NEURON 则称为 special_neuron 的父亲。special_neuron 继承了 NEURON 的所有局部变量及所有的函数。special_neuron 重新定义了神经元的状态转移函数 transfer()，从而获得了一个具有二值输出的神经元。将 special_neuron 的状态转移函数定义成虚函数(virtual)的目的是希望它还能继续遗传下去。注意，不是虚函数的成员函数不能够遗传下去，这样的函数被看作是私有的。这种机制保证了有选择性的遗传。

如有需要，你还可以在 special_neuron 中定义另外的局部变量或成员函数。special_neuron 定义好之后，你还可以利用它来定义其它的神经元。

网络可以从它的父亲那里继承下来，并且可以拥有多个父亲，即继承是多继承的。由于继承包含了网络拓扑结构和网络行为的继承，一些神经网络可能具有相同名字的行为，这就导致了行为继承的冲突问题。为此，ONPL 规定，当发生此类问题时，在新的网络中，所有的网络行为都必须明确定义。这里用一个实例加以说明。设有网络 A、B，其拓扑结构分别如图 1(A)及图 1(B)所示。网络 C(如图 1(C)所示)继承了网络 A、B。A、B 称为 C 的父亲，而 C 则是它们的儿子。

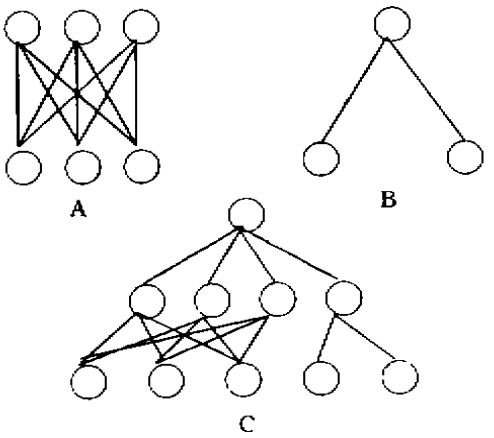

图 1 神经网络的继承

```
NEURALNETWORK A
  (OUTPUT LAYER NEURON s(3);
   INPUT LAYER NEURON t(3);
```

```
    CONNECTION;
      {LINK (s,t);};
    BEHAVIOR;
      {void learning(…){…};
       void print(){…};
              ⋮
      }
  };
NEURALNETWORK B
  {OUTPUT LAYER NEURON s;
   INPUT LAYER NEURON t(2);
   CONNECTION;
     {LINK(s,t);};
   BEHAVIOR;
     {void learning(…){…};
      void print(){…};
             ⋮
     }
  };
NEURALNETWORK C:(A,B)
  {OUTPUT LAYER NEURON p;
   INPUT LAYER A.t,B.t;
   CONNECTION;
     {LINK (p.A.s);
      LINK(p.B.s);
     }
   BEHAVIOR;
     {void learning(…){A.learning(…);};
          /*inherited from A */
      void print()
        {A.print();
         B.print();
                ⋮
        };
     }
}
```

3 基于大神经元的网络系统

大神经元原理还告诉我们网络本身可以作为一个特殊的神经元和其他神经元一起构造更复杂的网络。为保持大神经元的纯洁性，ONPL 规定大神经元中的每一个输入神经元只能接受一个输入，否则，整个网络的训练必将导致大神经元本身的重新训练，这与我们设计大神经元的思想不符。这里用一个例子来加以说明。

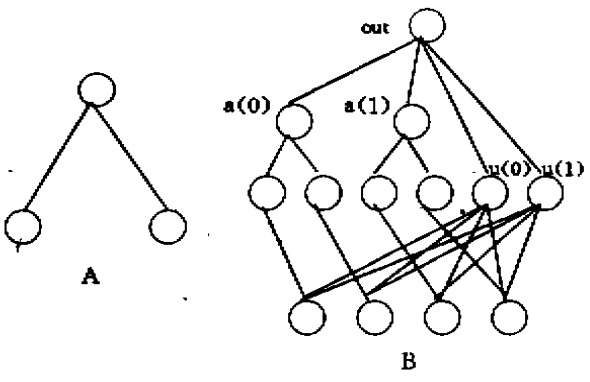

图 2 基于大神经元的网络

设有网络 A，B，其拓扑结构分别由图 2.A 及图 2.B 示出。在 B 中，有两个大神经元 a(0)和 a(1)。

```
NEURALNETWORK A
  {OUTPUT LAYER NEURON s;
   INPUT LAYER NEURON t(2);
   CONNECTION;
     {  LINK(s,t);};
   BEHAVIOR;
     {
        void learn(…){…};
     };
};
NEURALNETWORK B
  {OUTPUT LAYER NEURON out;
   LAYER A a(2),NEURON u(2);
   INPUT LAYER NEURON in(4);
   CONNECTION;
     {LINK(out,u);
      LINK(u,in);
      LINK(out,a(0).s);
      LINK(out,a(1).s);
      LINK(a(0).t(0),in(0));
      LINK(a(0).t(1),in(1));
      LINK (a(1).t(0),in(2);
      LINK (a(1).t(1),in(3));
     };
   BEHAVIOR;
     {void learn(…)
        {⋮
         a(0).learn(…);
         a(1).learn(…);
          ⋮
         };
     };
};
```

在设计包含有大神经元网络的学习算法时必须注意，如果该大神经元还未经训练，那么，整个网络的学习算法应包括以下两个步骤：

步骤 1：首先训练大神经元，其方法是调用大神经元自身的学习算法。

步骤 2：将大神经元考虑成是一个独立的整体（封装），也就是说，在整个网络学习中，大神经元内部的连接权值等不能改变。整个网络学习算法的任务就是调整大神经元和其他神经元之间的权值。

如果该大神经元已经训练完毕，则网络的学习算法设计可直接进入步骤 2。

4 神经网络的动态重构

在 ONPL 中，神经网络的动态重构系指在网络的运行过程中，可以增加或删除节点以及节点之间的连接来达到改变网络的拓扑结构，使其自我进化成一种更高级的网络。这一点极其重要。事实上，人类在不断的学习中，大脑的生物神经细胞构成的网络是不断重构的，以适应对外界的需要。

在 ONPL 中，神经网络的动态重构由◁NEW 语句▷，◁DELETE 语句▷，◁LINK 语句▷以及◁DELETE_LINK 语句▷来实现。这里用实例来说明。设有这样一个网络 A，如图 3.A 所示，其功能为求逻辑与问题。现动态地构造网络 B，如图 3.B 所示，使其能求解更高级的逻辑异或问题。

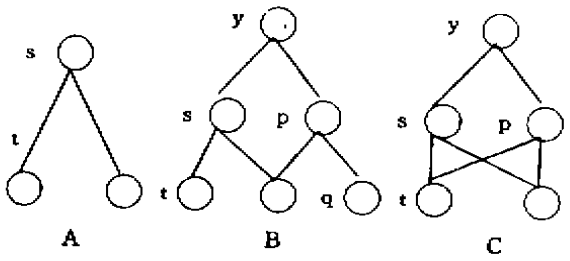

图 3　神经网络的动态重构

```
NEURALNETWORK A
  {OUTPUT LAYER NEURON s;
   INPUT LAYER NEURON t(2);
   CONNECTION;
    {LINK (s,t);};
BEHAVIOR;
  {
   void learning(…)
     {BACKPROPAGATION(…);};
       /* 求解逻辑与的学习算法 */
   NEURON *y,*p,*q;
   void reconfiguration 1()
     {
        p=NEW NEURON;
        q=NEW NEURON;
        y=NEW NEURON;
        OUTPUT LAYER y
        LAYER s;
        INPUT LAYER q
        LAYER p;
        LINK(y,p);
        LINK(p,t(1));
        LINK(p,q);
        BACKPROPAGATION(…);
        /* 重新学习以求解异或问题 */
     };
   void reconfiguration 2()
     {DELETE_LINK(p,q);
      DELETE q;
      LINK(p,t(0));
      BACKPROPAGATION(…);
      /* 重新学习 */
      };
  };
};
```

网络 A 经过 learning(…)学习后，已具备求解逻辑与的功能。此时，reconfiguration 1()引入新节点 y、p、q，并增加必要的连接，从而构成了网络 B。注意，网络的拓扑结构改变后，如果一节点的属性发生变化，如节点 s 由输出节点转化为中间隐含节点，则必须在程序中显示说明。对新增的节点，也要说明其是属于哪一类节点。如 y 则说明成输出节点，p 则说明成隐含节点，q 则为输入节点。新增节点连接好后、经重新学习后即可求解异或逻辑。

reconfiguration2()执行的结果使得网络的拓扑结构演化成网络 C(如图 3.C 所示)。这里删除了多余节点 q。

还应指出，由于网络的封装机制，一个网络的动态重构只能由其自身来完成，因为网络的内部状态对外界来说是不可见的。对外，网络仅表现其应有的行为。

结束语　本文通过神经元的扩展定义统一了一般对象、神经元、神经网络的概念。众多的神经元通过相互联结从而形成更复杂的网络，这就是所谓的面向神经元的系统。这种联结是基于神经元之间的动态的通讯，因此，通讯方向、对象的改变，就意谓着拓扑结构的改变。换句话说，网络的拓扑结构可以动态重构。由于大神经元同时具有面向对象和神经网络的性质，神经网络的继承、遗传等也就成了自然而然的事。

参考文献

〔1〕李涛，The Fundamental Theory of Big-Neuron，Proc. of ICNNSP95，Nanjing，China，Dec. ，1995

〔2〕李涛，ONPL，面向神经元的程序设计语言，《高技术通讯》，1995(6)

〔3〕李涛，ConcurrentC^{++}，面向对象的并发程序设计语言，《电子科学大学学报》，1995(4)

〔4〕Object-Oriented Concurrent Programming，edited by Akinori Yonezawa and Mario Tokoro，The MIT Pressed，1987

(上接第 59 页)

其内部活动机理是不可能的，本文在对表象的已有的研究成果进行分析，提出基于表象演示的直觉思维模型的兰图，试图通过模拟、分析、修改、再模拟的方法逐步使模型贴近人脑的思维表现。本文关于表象的一些讨论以及所设想的演示判断模型还很粗糙，有关意识与非意识的产生、思维过程的串并行配合等问题还未涉及到。

参考文献

〔1〕钱学森主编，《关于思维科学》，上海人民出版社，1986

〔2〕袁安杰，日本计划研制模仿人脑的计算机，国外科技动态，1993 年第十一期

〔3〕B. M. 维里奥科夫斯基，《现代认知心理学》孙晔、张世英等译，社会科学文献出版社，1988

〔4〕尹红风、戴汝为，论思维及模拟智能，中国人工智能，吉林大学出版社，1990. 6

〔5〕潘云鹤、耿卫东、何志均，形象思维中的心象及表征理论，计算机科学，1994 年第一期

〔6〕刘宗田，人工神经系统的心理学方法，计算机科学，1994 年第一期