人工 神经网络 面向对象 人工智能

计算机科学 1994 Vol. 21 №. 5

24-26

面向对象的人工神经网络

TP18

陆伟民
(同济大学结构工程学院 上海 200092)

摘 要 The artificial neural network models established with object-oriented programming were described in this paper. As a typical sample, the three-layer backpropagation neural network was selected to explain the creation of new classes by using Smalltalk/V。

一、引 言

面向对象编程(OOP)与人类思维和语言在表达模式上的接近而日益受到软件工程界的青睐，人工智能开发上也出现对它明显的倾斜。在诸多面向对象编程工具中，Smalltalk/V的直接编程环境、内设的完整类与方法、弹出式菜单、开发工具包等特色，提供的类封装、继承及多态性这些有力的OOP特性，逐渐成为受用户喜爱的OOP软件。

人工智能的重要分枝：神经网络方法已发展了很多不同的模型，如Adaline、Hopfield、ART、BAM、Hamming、Back-Propagation(B-P)等，其中B-P模型应用较广，成功地用于语言识别、模式分类、数据压缩等方面，为了进一步发挥它的效能，本文探讨与面向对象概念相结合的面向对象的人工神经网络模型建立方法。

陆伟民 教授

二、反向传输网络模型的B-P算法

神经元网络可看作是接收输入产生输出"黑盒"一类的信息处理系统，主要由处理单元(PE)和它们之间加权的联接构成。联接在表示上应包括标记与数值(w_{ij}，表示从PE$_i$联向PE$_j$的权)。权储存信息，其数值通过学习决定，即在学习过程中不断调整。对各个PE执行更新的运算就能使网络回想(recall)信息。

神经网络在组成上包含三个基本方面：如何组织成层和层间联接(拓扑)；信息如何储存(学习)；如何从网络中调出储存的信息(回想)。图1是典型的B-P网络，由一输入层、一输出层和至少一隐层组成，为简单起见，这里取单隐层，每一层与上、下层的所有PE相联，网络的B-P算法包括正向传输和反向传输过程，正向传输即回想时样本输入值从输入层经隐层传

5 结论

使用多重集转换机制可以以很优雅的方式解决许多领域中的问题。GAMMA作为一个并行计算模型，它的特点就是使程序设计者在设计并行程序的时候从无关的细节中解放出来而注重于问题的本质，这个思想和一些其它的并行计算模型如UNITY，Linda都有许多类似之处。特别，Berry和Boudol在GAMMA中引入了"隔膜"的概念产生了所谓的"化学抽象机"(ChAM)。在ChAM中，可以定义代数进程演算的模型、刻画并发λ演算，给出线性逻辑的计算解释。目前，GAMMA语言已成为并行计算模型研究中的一个热点，许多研究者正在探讨在GAMMA上实现函数式语言，研究如何在巨并行计算机上有效地实现GAMMA模型，同时在法国国家并行性研究计划中，GAMMA也占有重要的一席之地。

参考文献

[1]Banatre, J. P. et al., The GAMMA model and its discipline of programming, Sci. Comput. Program. 15 (1990), 55-77

[2]Banatre, J. P. et al., Programming by Multiset Transformation, CACM, Jan. 1993

[3]Berry, G. et al., The Chemical Abstract Machine, TCS, 1992

[4]黄林鹏 孙永强，并发λ演算及在并行抽象机上的实现，《全国第二届系统结构会议论文集》1992，10

至输出层，然后作误差反向传输，通过改变各 PE 间的权值使样本点的实际输出值与期望输出值间误差逐渐减小，网络就是以这样的处理误差方式而得名。

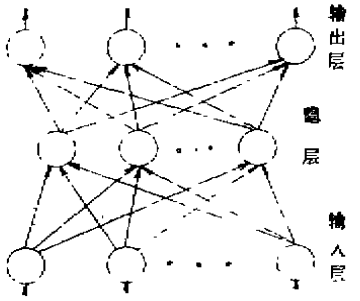

图 1　单隐层 B-P 网络

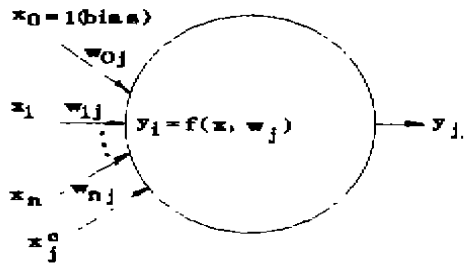

图 2　处理单元(PE)

为说明信息的存取，以图 2 的 PE 为例，其输入的传递为

$$y_i=f(\sum_i(w_{ij}x_i))=f(I_j) \tag{1}$$

其中的 f 常用 Sigmoid 函数：

$$f(x)=(1+e^{-x})^{-1} \tag{2}$$

图 2 中 $x_0=1$ 是偏置输入，x_j^o 是当训练网络时的输入。B-P 算法的实质是进行误差最小化。假设网络的总误差函数：

$$E=\frac{1}{2}\sum_k(x_k^o-o_k)^2 \tag{3}$$

其中 o 为网络在当前一组权值下产生的输出。

则作为关键参数的局部误差为：

$$e_j=-\frac{\partial E}{\partial I_j}=(x_j^o-o_j)f'(I_j) \tag{4}$$

不难推出 $e_j=y_j(1.0-y_j)\sum_k(e_k\cdot w_{jk})$ (5)

误差的减小通过调节权值达到，即

$$\delta w_{ij}=-C_L\cdot(\frac{\partial E}{\partial w_{ij}})=C_L\cdot e_j\cdot x_i \tag{6}$$

式中 C_L 为学习系数。

根据以上式子，若输入层给定向量 x_i，向前传输至输出层得输出向量 o，经与期望输出 x_j^o 比较得出总误差，然后再反向传输，按(5)式计算各层 PE 的局部误差，用(6)式计算修正的 delta 权值。如此通过多次前、后传输迭代使误差减小至允许范围内，这时输出层 PE 的输出即系最终结果。

三、B-P 网络模型的类设计

在 Smalltalk/V 已有的 100 个类中定义五个新的类，如图 3 所示，图中各方框的上方为新设的子类，下方为实例方法。表达神经元网络处理单元功能的子类为 Neuron，它再挂上输入层 PE 的 InputNeuron 与隐层 PE 的 SigmoidNeuron 两个子类以及后者用来处理权值修正的 BackpropagationNeuron 子类；而实现网络反向传输的子类为 BackpropagationNetwork。下面对这些类的主要 OOP 代码分别加以阐述。

四、定义新类方法的 Smalltalk 源代码

1. 类 Neuron

图 2 处理单元通过方法 activation 表示其内部的“活力”状况：activation $=w_{ij}\cdot x_i$，而可被其它 PE 感觉到的该神经元的外部状况由方法 Output 表达：Output $=f(\sum w_{ij}x_i)=f(\sum$activation$)$，所以有：

```
compute
    self ComputeActivation.
    ^ self computeOutput
computeActivation
    activation := o.
```

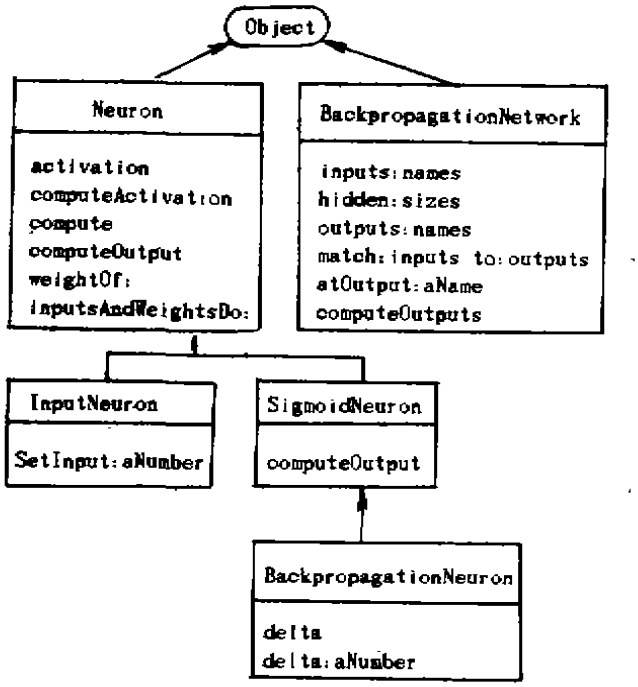

图 3　B-P 神经网络的 OOP 结构

```
    inputs bindingsDo:[ : aNeuron : weight|
    activation:=activation+(aNeuron output * weight)]
computeOutput
    ^ output:=activation
```

其中 bindingsDo: 方法分别在 Collection 类与 Dictionary 类中作为新的实例方法被加入：

```
bindingsDo:aBlock
    ^ (1 to self size)with:self do:aBlock
bindingsDo:aBlock
    ^ self associations Do:[ : anAssociation |aBlock
      Value:anAssociation key
      Value:anAssouciation value]
```

此外，inputsAndWeightsDo: 是一个定序运算方法，允许并行存取 neuron 的输入与权值：

```
inputsAndWeightsDo:aTwoParameterBlock
^ inputs bindingsDo:
aTwoParameterBlock
```

2. 类 InputNeuron

此为输入层的 neuron，其输出即系输入，无需计算，故：

```
setInput:aNumber
    self:output:aNumber
computeOutput
    ^ output
```

3. 类 SigmoidNeuron

对隐层的 neuron，根据(2)式计算其输出(数值在0—1之间)。

```
computeOutput
    ^ output:=1/(1+activation negated exp)
```

4. 类 BackpropagationNeuron

此类用于反向传输时的输出计算，它用到实例变量 delta 和相应的实例方法。

```
    delta
    ^ delta
    delta:aNumber
    delta:=aNumber
```

5. 类 BackpropagationNetwork

它分别对输入层、隐层与输出层设置，即图 3 所示它的前三个方法；然后通过输入与输出的匹配进行学习：

```
match:inputs to:outputs
| out target delta deltaWeight previousLayer|
inputs bindingsDo:[:name :value|
    self atInput:name put:value].
self computeOutputs.
```

并调节输出层中的权值：

```
outputLayer bindingsDo:[:name:outputNeuron|
    out:=outputNeuron output.
    target:=outputs at:name.
    delta:=out*(1.0-out)*(target-out)
    outputNeuron delta:delta.
outputNeuron
    inputsAndWeightsDo:[:inputNeuron:oldWeight|
      deltaWeight:=learningRate * delta * inputNeuron
                   output.
      outputNeuron increaseWeightOf:
        inputNeuron by:deltaWeight] ].
```

调节隐层中权值的代码与此类似。

生成输出：

```
atOutput:aName
 ^ (outputLayer at:aName)output
computeOutputs
|result|
result:=Directionary new.
hiddenLayer do :[:aLayer|
  aLayer do:[:aNeuron |aNeuron compute]].
outputLayer bindingsDo:[:name:neuron|
  result at: names put:neuron compute].
^ result
```

以及实现映射的方法：

```
aBackpropagationNetwork
match:inputs to:outputs.
```

五、讨论

1. 对子类实例的生成与初始化可执行下列代码段：

```
new
    ^ super new initialize
initialize
    inputs:=IdentityDictionary new.
learningRate:=0.5.
inputLayer:=Dictionary new.
hiddenLayers:=OrderedCollection new.
outputLayer:=Dictionary new.
```

2. 本文着重讨论了反向传输三层网络模型的面向对象语言实现，对于多于三层即两个以上隐蔽层的网络可在类 BackpropagationNetwork 中的 hidden:Size 以及相应的方法中作适当修改来完成。

3. 本文所建立的五个新类作为一个模块加入到 Smalltalk/V 的现有类中，需要时通过系统菜单的 browse classes 项调出，在使用上很为方便。

参考文献

[1]John Pugh, Wilf La Londe. Neural network simulation:a network mouse, JOOP. 1992. 2

[2]Edgar 等编. Artificial Neural Network:"Foundation of Neural Networks", IEEE press.

[3]焦李成. 神经网络系统理论，西安电子科技大学出版社，1991

[4]陆伟民. 面向对象程序设计专家系统研究，计算机科学 1990,5