多层感知器的训练算法

齐平
辽宁工程技术大学土木建筑工程学院，辽宁阜新 (123000)
E-mail: qipingws@163.com

摘　要：本文着重介绍的在人工智能中的多层感知器(MLP)是什么，是怎样构建的。多层感知器和单层感知器一样，是一种人工神经网络。单层感知器只能处理线形问题,对复杂的问题只能粗略进行近似表示。多层感知器是建立在单层感知器的基础上的，它的结构基本类似于一套级联的感知器，对输入层和输出层之间的关系进行研究。
本文侧重描述多层感知器（MLP）的逆向传递和训练过程，并给出了相应的公式和训练算法。以便了解在人工智能中，它的训练学习过程。为了简单起见用，本文中的算法是以伪代码的方式加以描述，这样，就可以用几乎任何一门语言实现它们。

关键词：神经网络，多层感知器，算法
中图分类号：tp18

1. 引 言

多层感知器(MLP)是一种人工神经网络,它使用输入与输出之间的多层加权连接.MLP 的结构基本类似于一套级联的感知器,其中每一格处理单元都有一格相对复杂的输出函数,从而增强网络的性能.[1]

多层感知器是建立在单层感知器的基础上的.
单层感知器只能处理线形问题,而对复杂的问题只能粗略进行近似表示.多层感知器与单层感知器有两个主要的区别:

1. 明确区别:多层感知器存在中间层,它们增加了感知器近似表示的能力.
2. 不明确区别:对于中间层在系统中发挥的作用是必不可少的,这涉及到使用更加复杂的激励函数.

2. 多层感知器（MLP）

2.1 拓扑结构

拓扑就是神经网络中处理单元的拓扑,以及它们之间如何连接在一起。一个 MLP 的拓扑被称为前馈（如图 1），由于不存在后向的连接——也叫做回归连接。通常信息直接从输入流向输出，而 MLP 的重要结构就是改善中间层。

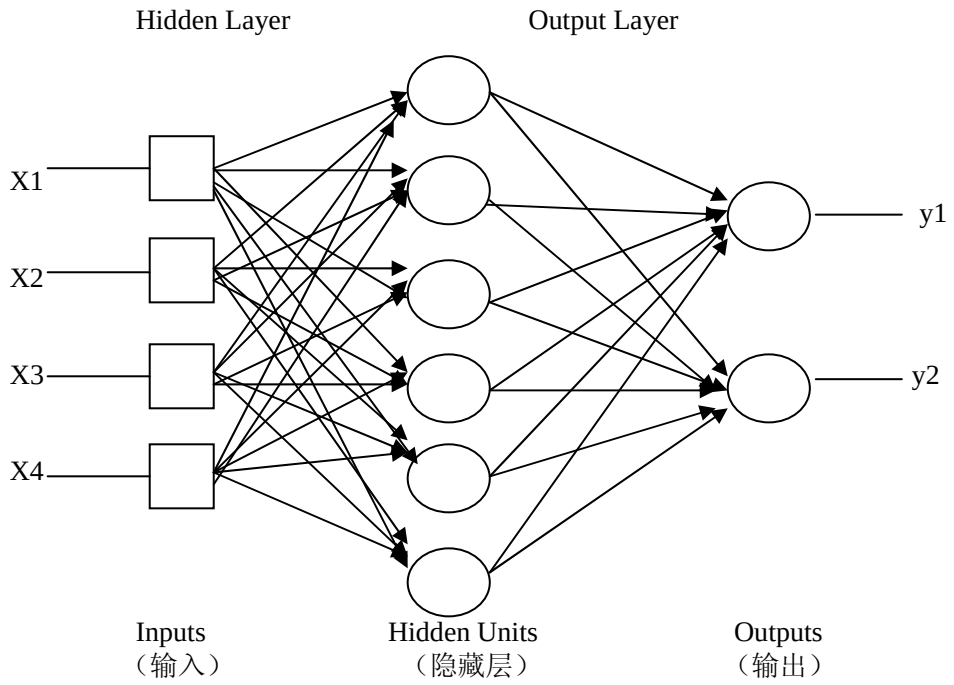

图 1 一个 MLP 的拓扑结构

2.2 中间层

虽然普遍情况下中间层只有一层，但理论上可以存在任意数量的中间层。这些中间层有时是被隐含的，并不直接连接到输出。

在某些情况下，在输入变量之间存在许多相互依赖的关系，并且问题的复杂度很高。这时，一个附加的层可以有助于减少进行适当的近似表示所需要的总的权数。在实践中，在任何拓扑中都很少用到 2 个以上的隐含层。具有 2 个中间层的感知器几乎可以表示任何函数——甚至是非连续的。[1]

对于只有 2 个输入的感知器，决策层面是一个二维线条。对于 MLP，决策层面比直线更加精确，可以被理解为一条将输入模式彼此分隔开的曲线。

随着维数的增长（更多的输入），所需要的决策层面的复杂度也在增长。此外，这也要求了隐含神经元的数目以指数形式增长，从而导致了一个被称为维度发难的困境。这是神经网络并不大规模扩展，以及它们并不适合于处大规模问题的解释之一。

2.3 激励函数

激励函数基于每个单元的净总和来计算输出,不同类型的函数都是可用的.然而,中间层中的线性激励函数几乎一点用也没有,MLP 可能具有同一般感知器同样的功能,因而将 2 个线性函数组合起来会得到另一个线性函数。为了使隐含层对 MLP 的计算能力起作用，一个非线性的激励函数是必需的。

3. 训练算法

MLP 的数学背景具备一个优点：它们可以被训练。训练 MLP 的任务基本上就是数学最优化。必须找到每一个权的最佳值以最大限度地减少输出错误。

3.1 逆向传递

逆传是一个从输出层开始贯穿前面所有层的过滤错误的过程。由于最初的感知器训练算法中的问题而使这项技术得到了发展。它们不能训练隐含层，BP 是大多数 MLP 学习算法的基础，因为它允许在每一个权值中找到梯度，因而可以被优化。

对于最后一层，输出中的错误立刻就可以得到，做为实际输出和理想输出之间的差别加以计算。对于感知器使用同样的规则，可以找到每一个权值的错误梯度方向。沿梯度下降算法可以使用错误的坡度来调整最后一层的权值。[2]

对于隐含层，不能立即发现错误，因为没有输出可以参考比较。可是能够找到与输出中发生错误的神经元相连接的隐含神经元。通过在神经网络中反向传递错误，就可以将错误分布到前面的神经元上。

反向传递的过程，如图 2 表示，传递了错误梯度。隐含单元的错误梯度就是输出单元中错误梯度的加权和。连接的权值决定了来自于已知处理单元的隐含的错误梯度。

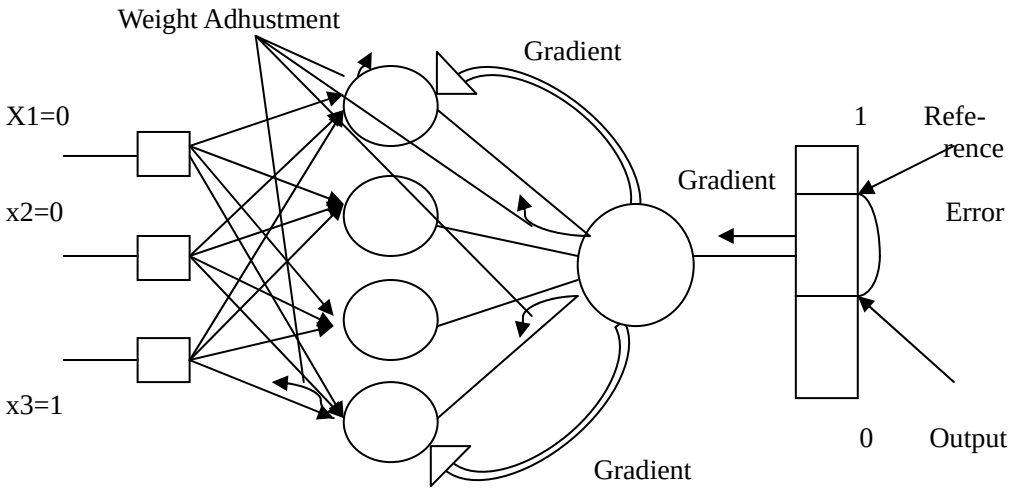

图 2 各层之间的错误反向传递

这可以被理解为错误梯度的递归定义过程。不管有多少曾，都可以从输出开始反向处理。并计算出处理单元中所有的错误梯度。然后对感知器使用同样的技术，就可以计算出每一个权值上的梯度。最速下降——或者其他任何基于梯度的优化——旧可以应用到权值中对它们进行相应的调整。

3.2 正式验证

对于潜在的训练细节，我们将尝试找出对于连接单元 i 和连接单元 j 的权值所衍生出的错误。记为 $\partial E / \partial w_{ij}$。在这里可以被写成：

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \zeta}\frac{\partial \zeta}{\partial W_{ij}}$$

$\partial \zeta / \partial w_{ij}$ 是单元 j 关于权值的净总和。它是前面一个单元 i 的输出 y_i：

$$\frac{\partial \zeta}{\partial W_{ij}} = y_i$$

$\partial y_j / \partial \zeta$ 表示了单元 j 关于净总和的输出梯度。这由激励函数定义，所以需要对它进行考虑：

$$\frac{\partial y_j}{\partial \zeta} = \sigma'(\zeta_j)$$

$$\partial E / \partial y_i = -(t_i - y_i)$$

对于那些没有直接连接到 MLP 的输出上的单元 J，这一项根据那些连接到输出上的单元 K 进行递归的运算，与输出 $\partial E / \partial y_k$ 相对的梯度错误，乘以与净总和 $\partial y_k / \partial \zeta_k$ 相对的梯度输出，在乘以 j 和 k 之间的权值 w_{jk}，然后再求和就得到这一项[2]：

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial \zeta_k} w_{jk}$$

通常，是用比较简单的符号对执行算法很有帮助，如果我们做如下定义：

$$\delta_j = -\frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \zeta_j}$$

那么与净总和 δ_j 相对应的错误梯度就可以根据它在网络中的位置进行计算了：

$$\delta_j = \begin{cases} \sigma'(\zeta_j)(t_j - y_j) & (1) \\ \sigma'(\zeta_j) \sum_k \delta_k w_{jk} & (2) \end{cases}$$

此式的条件为：j 是一个输出单元。式的条件为：j 是一个隐含单元。

其重点在于需要逆向传递过程，每一层中每一个单元的 δ_j 根据前一个单元的 K 进行计算。

4. 逆向传递算法

使用以上公式可以得到一个简单的训练算法，并对一组实例使用递增的训练。[3]这种方法称为逆向传递算法。

以下是两段代码。一个是用来计算每一个单元中错误梯度的逆向传递算法，一个是每一个权值在错误梯度方向上的最快下降。

4.1 计算每一个单元中错误梯度的逆向传递算法

```
#     compute the gradient in the units of the first layer
For each unit j in the last layer
Delta[j]=derive_activate(net_sum)
*(desired[j]-output[j])
End for
# process the layer backwards and propagate the error gradient
For each layer from last-1 down to first
For each unit j in layer
Total=0
# add up the weighted error gradient from next layer
Foe each unit k in layer+1
Total +=delta[k]*weights[j][k]
```

```
End for
Detal[j]=derive_activat(net_sum)*total
End for
End for
```

4.2 每一个权值在错误梯度方向上的最快下降

```
For each unit j
For each input i
#adjust the weights using the error gradient compute
Weight[j][i]+=learning_rate*detal[j]*output[i]
End for
End for
```

为了简单起见用伪代码的方式加以描述，这样，就可以用几乎任何一门语言实现它们。

5. 结论

感知器有一个必然的数学基础。最近的技术在寻找存在的解决方案上非常有效。对于定义良好的问题，感知器看起来经常是最好的选择之一。

感知器的另一个优点在于可用的训练算法的广泛形式。这是能够在许多不同情况，不同限制条件下开发ＭＬＰ。这提供了另人惊奇的灵活性。

开发一个ＭＬＰ要做大量的试验。处理单元的甚至是层数都是需要开发的参数。输入和输出的设计也需要特别的注意，因为它们对问题有巨大的影响。

参考文献

[1] Alex J.Champandard 《人工智能游戏开发》[M] ，陈贵敏 冯兰胜 李萌萌译.北京：北京希望电子出版社.2004.12

[2] 李鸿吉．《模糊数学基础及实用算法》 [M]，北京：科学出版社，2005。2。

[3] Steve Rabin《人工智能游戏编程真言》[M]，庄越挺 吴飞译，北京：清华大学出版社。2005.1.

Multi-sensor training algorithm

Qi Ping

institute of civil Engineering ,Liaoning Technical University, fuxin, Liaoning (123000)

Abstract

In this paper，it 'll show that the Multi-sensor (MLP) is what is how to build it. Like the Single-layer, multi-sensor is an artificial neural network. Perception can only handle single-linear problems , but for complex problems , it can only give a rough approximation. Multi-sensor is built on a single its basic structure similar to a cascade of perception, it study for how to show the relationship between the input and output layer .

This paper describes the reverse transfer and training process of multi-sensor (MLP), and gives the corresponding formula and training algorithm. In artificial intelligence to understand, it's training and learning process. To simple, the algorithm is in this pseudo-code, so that you can use almost any one language to achieve them.

Keywords: neural networks, multi-sensor, the algorithm