6.1 多层网络搞定“异或”问题

现在我们都知道，深度学习是一个包括很多隐含层的复杂网络结构。感知机之所以当年搞不定“非线性可分”问题，也是因为相比于深度学习这个“老江湖”，是因为它“too young，too simple（太年轻，太简单）”。当时，“感知机”刚刚诞生不久，如初生之婴儿，你也很难期望这么一个襁褓之娃复杂起来。

如前文所述，想解决“异或”问题，就需要让网络复杂起来。这是因为，复杂的网络，表征能力就比较强[1]。按照这个思路，我们在输入层和输出层之间，添加一层神经元，将其称之为隐含层（hidden layer，亦有简称为“隐层”）。这样一来，隐含层和输出层中的神经元都拥有激活函数。假设各个神经元的阈值均为0.5，权值如图6-1所示，就实现了“异或”功能。

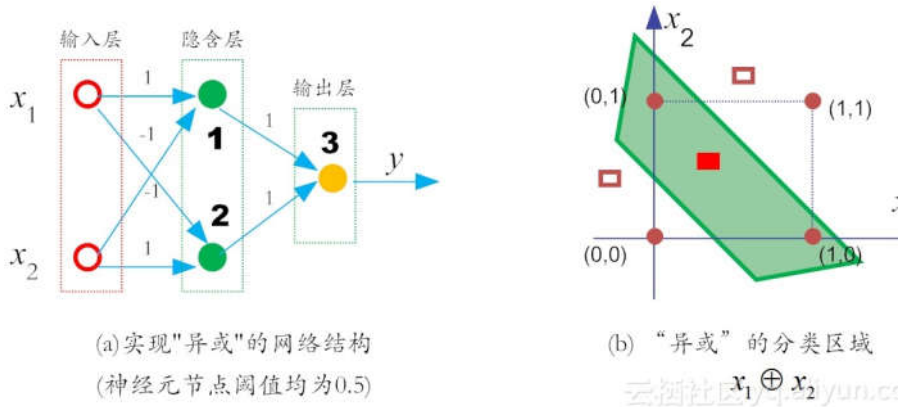

图6-1 可解决“异或”问题的两层感知机

下面我们来详述这个实现流程。假设在如图6-1-a所示的神经元（即实心圆）的激活函数，依然是阶跃函数（即sgn函数），那么它的输出规则很非常简单：当x>0时，f(x)输出为1，否则输出0。

那么，对于x_1和x_2相同（即均为1）时，对于在隐含层的神经元1有：

$$\begin{aligned} f_1 &= \mathrm{sgn}(x_1 w_1 + x_2 w_2 - \theta) \\ &= \mathrm{sgn}(1\times 1 + 1\times(-1) - 0.5) \\ &= \mathrm{sgn}(-0.5) \\ &= 0 \end{aligned}$$

类似地，对于在隐含层的神经元2有：

$$\begin{aligned} f_2 &= \mathrm{sgn}(x_1 w_1 + x_2 w_2 - \theta) \\ &= \mathrm{sgn}(1\times 1 + 1\times(-1) - 0.5) \\ &= \mathrm{sgn}(-0.5) \\ &= 0 \end{aligned}$$

然后，对于输出层的输出神经元3而言，这时f_1和f_2是它的输入，于是有：

$$\begin{aligned} y = f_3 &= \mathrm{sgn}(f_1 w_1 + f_2 w_2 - \theta) \\ &= \mathrm{sgn}(0\times 1 + 0\times(1) - 0.5) \\ &= \mathrm{sgn}(-0.5) \\ &= 0 \end{aligned}$$

也就是说，x_1和x_2同为1时，输出为0。

那么对于x_1和x_2不相同（比如说，x_1为1时， x_2为0）时，对于在隐含层的神经元1有：

$$
\begin{aligned}
f_1 &= \mathrm{sgn}(x_1 w_1 + x_2 w_2 - \theta) \\
&= \mathrm{sgn}(1\times 1 + 0\times(-1) - 0.5) \\
&= \mathrm{sgn}(0.5) \\
&= 1
\end{aligned}
$$

类似地，对于在隐含层的神经元2有：

$$
\begin{aligned}
f_2 &= \mathrm{sgn}(x_1 w_1 + x_2 w_2 - \theta) \\
&= \mathrm{sgn}(1\times 1 + 0\times(-1) - 0.5) \\
&= \mathrm{sgn}(0.5) \\
&= 1
\end{aligned}
$$

然后，对于输出层的神经元3而言，f_1和f_2是它的输入，于是有：

$$
\begin{aligned}
y = f_3 &= \mathrm{sgn}(f_1 w_1 + f_2 w_2 - \theta) \\
&= \mathrm{sgn}(1\times 1 + 1\times 1 - 0.5) \\
&= \mathrm{sgn}(1.5) \\
&= 1
\end{aligned}
$$

不失一般性，由于x_1和x_2的地位是可以互换的，因此从上面分析可知，如图6-1所示的两层感知机，就可以实现“异或”功能。在这里，网络中的权值和阈值是我们事先给定的，而实际上，它们是需要神经网络自己通过反复地“试错”学习而来。

让我们再简单回顾一下神经网络的发展历史。1958年，弗兰克·罗森布拉特（Frank Rosenblatt）提出“感知机”（Perceptron）的概念。

1965年，A. G.伊瓦赫年科（Alexey Grigorevich Ivakhnenko）就提出了多层人工神经网络的设想。而这种基于多层神经网络的机器学习模型，后来被人们称为“深度学习”。

简单来说，所谓深度学习，就是包括很多隐含层的神经网络学习。这里的“深”即意味着“层深”（’Deep’ means many hidden layers），“网络深深深几许”呢？至少要大于3吧，多则不限，可以成百甚至上千。

所以，你看到了吧，如果追根溯源的话，伊瓦赫年科才是“深度学习”之父[2]，而不是现在的“深度学习”大牛杰弗里•辛顿（Geoffrey Hinton），但鉴于辛顿的杰出贡献——是他让“深度学习”重见天日、大放异彩。因此，称呼辛顿为深度学习教父（Godfather of deep learning），似乎也更为合适[3]。

更需要我们玩味的是，在多层神经网络概念提出4年之后的1969年，明斯基才写出来他的那本“毒苹果”之作《感知机》。也就是说，多层神经网络在提出之后，并没有受到应有的重视。

直到1975年（此时，距离伊瓦赫年科提出多层神经网络概念已达10年之久了），感知机的“异或难题”才被理论界彻底解决。由此，见微知著可以看到，科学技术的发展，从而都不是线性地一蹴而就，而是呈现螺旋上升的！

我们现在学习这个“异或”解决方案，可能仅需要数分钟，但也是“书上一分钟，书后十年功”啊！

6.2多层前馈神经网络

更一般地，常见的多层神经网络如图6-2所示。在这种结构中，每一层神经元仅仅与下一层的神经元全连接。而在同一层，神经元彼此不连接，而且跨层的神经元，彼此间也不相连。这种被简化的神经网络结构，被称之为“多层前馈神经网络（multi-layer feedforward neural networks）”。

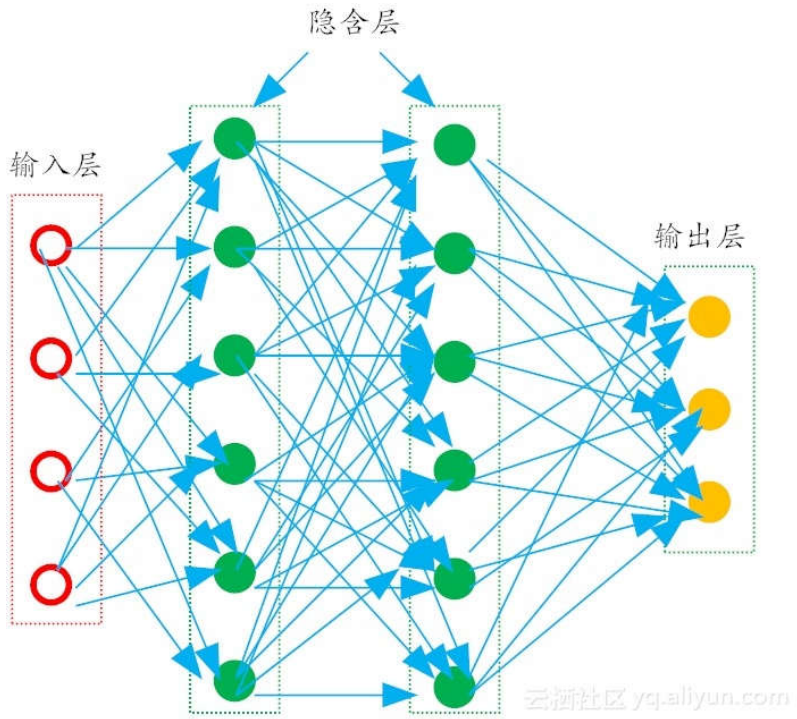

图6-2 多层前馈神经网络结构示意图

在多层前馈神经网络中，输入层神经元主要用于接收外加的输入信息，在隐含层和输出层中，都有内置的激活函数，可对输入信号进行加工处理，最终的结果，由输出层“呈现”出来。

这里需要说明的是，神经元中的激活函数，并不限于我们前面提到的阶跃函数、Sigmod函数，还可以是现在深度学习常用的ReLU（Rectified Linear Unit）和sofmax等。

简单来说，神经网络的学习过程，就是通过根据训练数据，来调整神经元之间的连接权值（connection weight）以及每个功能神经元的输出阈值。换言之，神经网络需要学习的东西，就蕴含在连接权值和阈值之中。

拟人化来说，对于识别某个对象来说，神经网络中的连接权值和阈值，就是它关于这个对象的“记忆（memory）”啊！

我们知道，大脑对于事物和概念的记忆，不是存储在某个单一的地点，而是像分布式地存在于一个巨大的神经元网络之中。

硅谷投资人王川先生认为，分布式表征（Distributed Representation），是人工神经网络研究的一个核心思想。那什么是分布式表征呢？简单来说，就是当我们表达一个概念时，神经元和概念之间不是一对一对应映射（map）存储的，它们之间的关系是多对多。具体而言，就是一个概念可以用多个神经元共同定义表达，同时一个神经元也可以参与多个不同概念的表达，只不过所占的权重不同罢了。

举例来说，对于“小红汽车”这个概念，如果用分布式特征地表达，那么就可能是一个神经元代表大小（形状：小），一个神经元代表颜色（颜色：红），还有一个神经元代表车的类别（类别：汽车）。只有当这三个神经元同时被激活时，就可以比较准确地描述我们要表达的物体。

分布式表征表示有很多优点。其中最重要的一点，莫过于当部分神经元发生故障时，信息的表达不会出现覆灭性的破坏。比如，我们常在影视作品中看到这样的场景，仇人相见分外眼红，一人（A）发狠地说，“你化成灰，我都认识你（B）！”这里并不是说B真的“化成灰”了，而是说，虽然时过境迁，物是人非，当事人B外表也变了很多（对于识别人A来说，B在其大脑中的信息存储是残缺的），但没有关系，只要B的部分核心特征还在，那A还是能够把B认得清清楚楚、真真切切！人类的大脑还是真的厉害啊！（现在这句话，也被好端端地被玩坏了，见图6-3）

图6-3 核心特征不丢失:化成灰我都认识你

前文提到，对于相对复杂的前馈神经网络，其各个神经元之间的链接权值和其内部的阈值，是整个神经网络的灵魂所在，它需要通过反复训练，方可得到合适的值。而训练的抓手，就是实际输出值和预期输出值之间存在着“落差”（你可以称之为“误差”）。

下面我们就先用一个小故事，来说明如何利用“落差”来反向调节网络参数的。

6.3现实很丰满，理想很骨感

说到理想和现实的“落差”时，人们常用这么一句话来表达：“理想很丰满，现实很骨感。”

事实上，有时候，这句话反着说，也是成立的：“现实很丰满，理想很骨感”。你可能猜到了，我说的是“减肥”这件事。

有人开玩笑说，男人有两大烦恼：一是把别人的肚子搞大了，二是把自己的肚子搞大了。还记得六七年前我在美国读书时，由于美帝的物质条件太好，肉食非常便宜，所以一不小心，糟了！我把自己的肚子搞大了。

等到临近回国的前三个月，一次无意站在体重秤上，我“惊喜”地发现，我有了——体重居然飙到可怕的200磅！

这次真的把自己都吓到了，这该如何是好？有个古诗改编的段子，很能体现我当时的“窘境”：

瘦小离家老胖回，乡音无改肥肉堆。

儿童相见不相识，笑问胖纸你是谁？

痛定思痛，我决定减肥（说好听点，是健身！）。

那该如何减（健）呢？其实就一个六字秘诀：“迈开腿，管住嘴！”

从那天起，我每天从住处到巴哈伊教神庙 (Baha'i House of Worship)两个来回，大概10公里的有氧长跑，几乎雷打不动。你看那，蓝蓝的天空，青青的草坪，幽静的小路上，总会有个胖纸正在挥汗如雨，汗流浃背，用不停止的脚步，来为过去的贪吃“赎罪”。

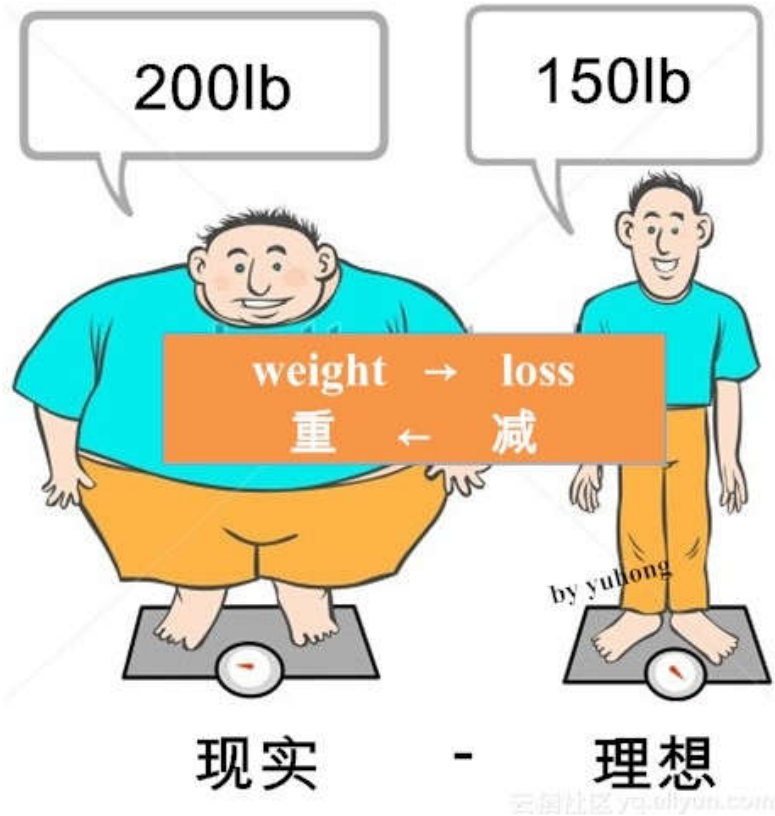

图 6-4 损失函数？减重功能！

现在回想起来，我挺佩服自己那会的毅力。三个月下来，我减下来近50磅！

在踏上返回北京航班的前一个小时，我静静地坐在租住屋里，模仿那首《再别康桥》，在自己的博客上，用两句话总结了我的美国之旅：

“轻轻的我走了，

正如我轻轻的来；

我挥一挥手，

不带走一两肥肉。”

或许你会疑惑，咦，我们正在学习神经网络咧，你吹这段过往的牛逼，是准备改行给我们推销减肥药吗？

哈哈，当然不是！

因为这段往事，让我想起了今天的主题“误差逆传播算法”！

这又哪跟哪啊？

别急，且听我慢慢道来。

我们知道，在机器学习中的“有监督学习”算法里，在假设空间F中，构造一个决策函数f，对于给定的输入X，由f(X)给出相应的输出Y，这个实际输出值Y和原先预期值Y’可能不一致。于是，我们需要定义一个损失函数（loss function），也有人称之为代价函数（cost function）来度量这二者之间的“落差”程度。这个损失函数通常记作L(Y,Y‘)= L(Y, f(X))，为了方便起见，这个函数的值为非负数（请注意：在这里我们大写Y和Y’，分别表示的是一个输出值向量和期望值向量，它们分别包括多个不同对象的实际输出值和期望值）。

常见的损失函数有如下3类：

（1）0-1损失函数（0-1 loss function）：

$$L(Y,f(X))=\begin{cases}1, & Y\neq f(X)\\ 0, & Y=f(X)\end{cases}$$

（2）绝对损失函数（absolute loss function）

$$L(Y,f(X))=|Y-f(X)|$$

（3）平方损失函数（quadratic loss function）

$$L(Y,f(X))=(Y-f(X))^2$$

损失函数值越小，说明实际输出和预期输出的差值就越小，也就说明我们构建的模型越好。

对于第一类损失函数，用我自身减肥的例子很容易解释。就是减肥目标达到没？达到了，输出为0（没有落差嘛），没有达到输出1（减肥尚未成功，胖纸还需努力！）

对于第二类损失函数就更具体了。当前体重秤上读数和减肥目标的差值，这个差值有可能为正，但还有可能为负值，比如说，减肥目标为150磅，但一不小心减肥过猛，减到140磅，这是值就是“-10”磅，为了避免这样的正负值干扰，干脆就取一个绝对值好了。

对于第三类损失函数，类似于第二类。同样达到了避免正负值干扰，但是为了计算方便（主要是为了求导），通常还会在前面加一个“1/2”，这样一求导，指数上的“2”和“1/2”就可以相乘为“1”了：

$$L(Y,f(X))=\frac{1}{2}(Y-f(X))^2$$

当然，为了计算方面，还可以用对数损失函数（logarithmic loss function）。这样做的目的，主要是便于使用最大似然估计的方法来求极值。一句话，咋样方便咋样来！

或许你会问，有了这些损失函数？有啥子用呢？当然有用了！因为可以用它反向配置网络中的权值（weight），让损失（loss）最小啊。

我们都知道，神经网络学习的本质，其实就是利用“损失函数（loss function）”，来调节网络中的权重（weight）。而“减肥”的英文是“weight loss”，所以你看，我用自身减肥的案例来讲这个“损失函数”，是不是很应景啊？

或许你又会说，就算应景，那神经网络的权值，到底该咋个调法咧？

总体来讲，有两大类方法比较好使。第一种方法就是“误差反向传播（Error Back propagation，简称BP）”。简单说来，就是首先随机设定初值，然后计算当前网络的输出，然后根据网络输出与预期输出之间的差值，采用迭代的算法，反方向地去改变前面各层的参数，直至网络收敛稳定。

这个例子说起来很抽象，我们还是用减肥的例子感性认识一下。比如说，影响减肥的两个主要因素是“运动”和“饮食”，但它们在减肥历程中的权值，并不了然。然后，如果我减肥目标是150磅，而体重秤上给出实际值是180磅，这个30磅的落差，反过来调整我“运动”和“饮食”在减肥过程中的权值（是多运动呢，还是多吃点低卡食物呢？）。

这个BP算法，的确大名鼎鼎。它最早是由Geoffrey Hinton 和 David Rumelhart等人1986年在《Nature》（自然）杂志上发表的论文：“Learning Representations by Back-propagating errors”中提出来的。该论文首次系统而简洁地阐述了反向传播算法在神经网络模型上的应用。

BP反向传播算法非常好使，它直接把纠错的运算量，降低到只和神经元数目本身成正比的程度。现在，我们可以回答上一讲中提出的问答了，是哪位“王子”把人工智能这位“白雪公主”吻醒的了。是的，没错，就是当前的这位“深度学习”教父杰弗里•辛顿（Geoffrey Hinton）！

图 6-5 “人工智能”白马王子：杰弗里•辛顿

BP算法非常经典，在很多领域都有着经典的应用，当时它的火爆程度在绝不输给现在的“深度学习”。但后来大家发现，实际用起来它还是有些问题。比如说，在一个层数较多网络中，当它的残差反向传播到最前面的层（即输入层），其影响已经变得非常之小，甚至出现梯度扩散（gradient-diffusion），严重影响训练精度。

其实，这也是容易理解的。因为在“信息论”中有个信息逐层缺失的说法，就是说信息在逐层处理时，信息量是不断减少的。例如，处理A信息而得到B，那么B所带的信息量一定是小于A的。这个说法，再往深层次的探寻，那就是信息熵的概念了。推荐读者阅读一部影响我世界观的著作《熵：一种新的世界观》[5]。

根据热力学第二定律我们知道，能量虽然可以转化，但是无法100%利用。在转化过程中，必然会有一部分能量会被浪费掉。这部分无效的能量，就是“熵”。把“熵”的概念，迁移到信息理论，它就表示“无序的程度”。

按照阮一峰先生的解读，当一种形式的"有序化（即信息）"，转化为另一种形式的"有序化"，必然伴随产生某种程度上的"无序化（即熵）"[6]。依据这个理论，当神经网络层数较多时（比如说大于7层），BP反向传播算法中“误差信息”，就会慢慢“消磨殆尽”，渐渐全部变成无序的“熵”，因此就无法指导神经网络的参数调整了。

再后来，第二类改进方法就孕育而生了。它就是当前主流的方法，也就是“深度学习”常用的“逐层初始化”(layer-wise pre-training)训练机制[7]，不同于BP的“从后至前”的训练参数方法，“深度学习”采取的是一种从“从前至后”的逐层训练方法。这是后话，我们回头再表。

6.4 小结

下面我们小结一下本章的主要知识点。首先，我们讲解了如何利用多层神经网络搞定“异或”问题的方法，然后我们用“减肥”的案例，讲解了多层前馈神经网络和损失函数的概念。

经典是永恒的，永远值得品味。在下一讲中，我们将用图文并茂的方式，详细讲解BP算法和梯度递减的概念，特别是这个“梯度递减”概念，它还会深深影响“深度学习”算法。请你关注。

6.5 请你思考

通过本讲的学习，请你思考如下问题：

（1）著名科技哲学家詹姆斯·卡斯在其著作《有限与无限的游戏》[8]中指出，世界上有两种类型的“游戏”：“有限的游戏”和“无限的游戏”。有限游戏的目的在于，赢得胜利；而无限的游戏，却旨在让游戏永远进行下去。从神经网络发展的起起落落中，你觉得“人工智能”的发展，是有限的游戏呢？还是无限的？为啥？

（2）杰弗里•辛顿先后提出的BP算法和深度学习算法，它们之间除了训练参数的方法有不同之外，还有什么本质上的不同呢？