Neural Networks in Plain English

- 用平常语言介绍神经网络

(Neural Networks in Plain English)

因为我们没有很好了解大脑，我们经常试图用最新的技术作为一种模型来解释它。在我童年的时候，我们都坚信大脑是一部电话交换机。**(**否则它还能是什么呢？**)**我当时还看到英国著名神经学家谢林顿把大脑的工作挺有趣地比作一部电报机。更早些时候，弗罗伊德经常把大脑比作一部水力发电机，而莱布尼茨则把它比作了一台磨粉机。我还听人说，古希腊人把大脑功能想象为一付弹弓。显然，目前要来比喻大脑的话，那只可能是一台数字电子计算机了。 **—John R.Searle [**注 **1]**

神经网络介绍（**Introduction to Neural Networks**）

曾有很长一个时期，人工神经网络对我来说是完全神秘的东西。当然，有关它们我在文献中已经读过了，我也能描述它们的结构和工作机理，但我始终没有能"啊哈！"一声，如同你头脑中一个难于理解的概念有幸突然得到理解时的感觉那样。我的头上好象一直有个榔头在敲着，或者像电影 **Animal House(**中文片名为"动物屋"**)**中那个在痛苦地尖叫"先生，谢谢您，再给我一个啊！"的可怜家伙那样。我无法把数学概念转换成实际的应用。有时我甚至想把我读过的所有神经网络的书的作者都抓起来，把他们缚到一棵树上，大声地向他们吼叫："不要再给我数学了，快给我一点实际东西吧！"。但无需说，这是永远不可能发生的事情。我不得不自己来填补这个空隙**...**由此我做了在那种条件下唯一可以做的事情。我开始干起来了。

这样几个星期后，在一个美丽的日子里，当时我在苏格兰海边度假，当我越过一层薄雾凝视着狭长的海湾时，我的头脑突然受到一个冲击。一下子悟到了人工神经网络是怎样工作的。我得到"啊哈！"的感觉了！但我此时身边只有一个帐篷和一个睡袋，还有半盒子的脆玉米片，没有电脑可以让我迅速写出一些代码来证实我的直觉。**Arghhhhh**！这时我才想到我应该买一台手提电脑。不管怎样，几天后我回到家了，我立刻让我的手指在键盘上飞舞起来。几个小时后我的第一人工神经网络程序终于编成和运行了，并且工作得挺好！自然，代码写的有点乱，需要进行整理，但它确实已能工作了，并且，更重要的是，我还知道它为什么能工作！我可以告诉你，那天我是一位非常得意的人。

我希望本书传递给你的就是这种"啊哈！"感觉。当我们学完遗传算法时，你可能已尝到了一点感觉，但你希望这种感觉是美妙的话，那就要等把神经网络部分整个学完。

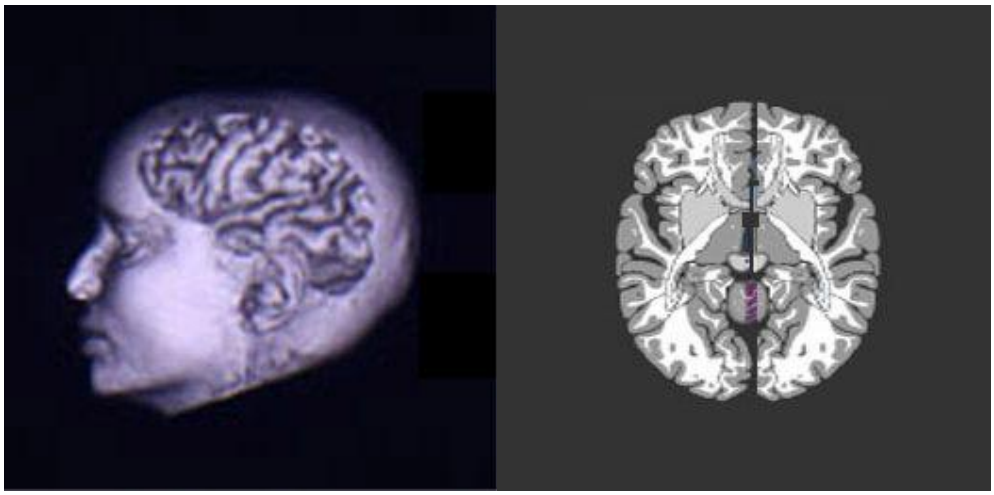
生物学的神经网络**-**大脑

(A Biological Neural Network–The Brain)

....你的大脑是一块灰色的、像奶冻一样的东西。它并不像电脑中的 **CPU** 那样，利用单个的处理单元来进行工作。如果你有一具新鲜地保存到福尔马林中的尸体，用一把锯子小心地将它的头骨锯开，搬掉头盖骨后，你就能看到熟悉的脑组织皱纹。大脑的外层象一个大核桃那样，全部都是起皱的**[**图 **0** 左**]**，这一层组织就称皮层**(Cortex)**。如果你再小心地用手指把整个大脑从头颅中端出来，再去拿一把外科医生用的手术刀，将大脑切成片，那么你将看到大脑有两层**[**图 **0** 右**]:** 灰色的外层**(**这就是"灰质"一词的来源，但没有经过福尔马林固定的新鲜大脑实际是粉红色的。**)** 和白色的内层。灰

色层只有几毫米厚，其中紧密地压缩着几十亿个被称作 **neuron**（神经细胞、神经元）的微小细胞。白色层在皮层灰质的下面，占据了皮层的大部分空间，是由神经细胞相互之间的无数连接组成。皮层象核桃一样起皱，这可以把一个很大的表面区域塞进到一个较小的空间里。这与光滑的皮层相比能容纳更多的神经细胞。人的大脑大约含有 **1OG**（即 **100** 亿）个这样的微小处理单元**;**一只蚂蚁的大脑大约也有 **250,OOO** 个。

以下表 **l** 显示了人和几种动物的神经细胞的数目。



- 图 **0-1** 大脑半球像核桃 图 **0-2** 大脑皮层由灰质和白质组成

图 **0** 大脑的外形和切片形状

表 **l** 人和几种动物的神经细胞的数目

动 物 神经细胞的数目（数量级）

蜗 牛 **10,000** （**=10^4**）

蜜 蜂 **100,000** （**=10^5**）

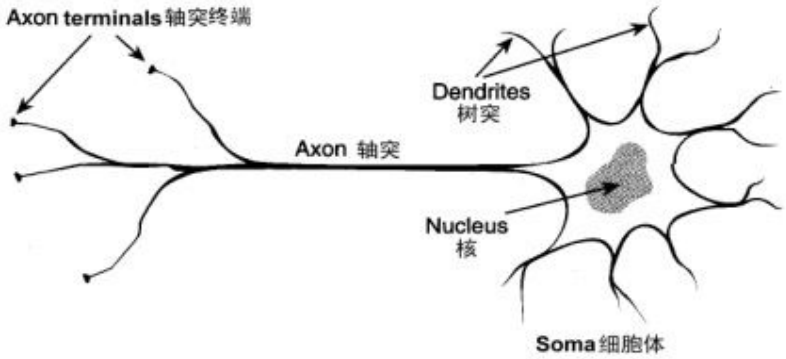
蜂 雀 **10,000,000** （**=10^7**）

老 鼠 **100,000,000** （**=10^8**）

人 类 **10,000,000,000** （**=10^10**）

大 象 **100,000,000,000** （**=10^11**）

图 **1** 神经细胞的结构



- 在人的生命的最初 **9** 个月内，这些细胞以每分钟 **25,000** 个的惊人速度被创建出来。神经细胞和人身上任何其他类型细胞十分不同，每个神经细胞都长着一根像电线一样的称为轴突（**axon**）的东西，它的长度有时伸展到几厘米［译注］，用来将信号传递给其他的神经细胞。神经细胞的结构如图 **1** 所示。它由一个细胞体**(soma)**、一些树突**(dendrite)** 、和一根可以很长的轴突组成。神经细胞体是一颗星状球形物，里面有一个核**(nucleus)**。树突由细胞体向各个方向长出，本身可有分支，是用来接收信号的。轴突也有许多的分支。轴突通过分支的末梢**(terminal)**和其他神经细胞的树突相接触**,**形成所谓的突触（**Synapse**，图中未画出），一个神经细胞通过轴突和突触把产生的信号送到其他的神经细胞。

每个神经细胞通过它的树突和大约 **10,000** 个其他的神经细胞相连。这就使得你的头脑中所有神经细胞之间连接总计可能有 **l,000,000,000,000,000** 个。这比 **100** 兆个现代电话交换机的连线数目还多。所以毫不奇怪为什么我们有时会产生头疼毛病！

有趣的事实

曾经有人估算过，如果将一个人的大脑中所有神经细胞的轴突和树突依次连接起来，并拉成一根直线，可从地球连到月亮，再从月亮返回地球。如果把地球上所有人脑的轴突和树突连接起来，则可以伸展到离开们最近的星系！

神经细胞利用电**-**化学过程交换信号。输入信号来自另一些神经细胞。这些神经细胞的轴突末梢（也就是终端）和本神经细胞的树突相遇形成突触（**synapse**），信号就从树突上的突触进入本细胞。信号在大脑中实际怎样传输是一个相当复杂的过程，但就我们而言，重要的是把它看成和现代的计算机一样，利用一系列的 **0** 和 **1** 来进行操作。就是说，大脑的神经细胞也只有两种状态：兴奋（**fire**）和不兴奋（即抑制）。发射信号的强度不变，变化的仅仅是频率。神经细胞利用一种我们还不知道的方法**,**把所有从树突上突触进来的信号进行相加，如果全部信号的总和超过某个阀值，就会激发神经细胞进入兴奋（**fire**）状态，这时就会有一个电信号通过轴突发送出去给其他神经细胞。如果信号总和没有达到阀值，神经细胞就不会兴奋起来。这样的解释有点过分简单化，但已能满足我们的目的。

神经细胞利用电-化学过程交换信号。输入信号来自另一些神经细胞。这些神经细胞的轴突末梢（也就是终端）和本神经细胞的树突相遇形成突触（**synapse**），信号就从树突上的突触进入本细胞。信号在大脑中实际怎样传输是一个相当复杂的过程，但就我们而言，重要的是把它看成和现代的计算机一样，利用一系列的 **0** 和 **1** 来进行操作。就是说，大脑的神经细胞也只有两种状态：兴奋（**fire**）和不兴奋（即抑制）。发射信号的强度不变，变化的仅仅是频率。神经细胞利用一种我们还不知道的方法,把所有从树突上突触进来的信号进行相加，如果全部信号的总和超过某个阀值，就会激发神经细胞进入兴奋（**fire**）状态，这时就会有一个电信号通过轴突发送出去给其他神经细胞。如果信号总和没有达到阀值，神经细胞就不会兴奋起来。这样的解释有点过分简单化，但已能满足我们的目的。

正是由于数量巨大的连接，使得大脑具备难以置信的能力。尽管每一个神经细胞仅仅工作于大约 **100Hz** 的频率，但因各个神经细胞都以独立处理单元的形式并行工作着，使人类的大脑具有下面这些非常明显的特点：

能实现无监督的学习。 有关我们的大脑的难以置信的事实之一，就是它们能够自己进行学习，而不需要导师的监督教导。如果一个神经细胞在一段时间内受到高频率的刺激，则它和输入信号的神经细胞之间的连接强度就会按某种过程改变，使得该神经细胞下一次受到激励时更容易兴奋。这一机制是 **50** 多年以前由 **Donard Hebb** 在他写的 **Organination of Behavior** 一书中阐述的。他写道**:**

"当神经细胞 **A** 的一个轴突重复地或持久地激励另一个神经细胞 **B** 后，则其中的一个或同时两个神经细胞就会发生一种生长过程或新陈代谢式的变化，使得励 **B** 细胞之一的 **A** 细胞，它的效能会增加"

与此相反的就是，如果一个神经细胞在一段时间内不受到激励，那么它的连接的有效性就会慢慢地衰减。这一现象就称可塑性（**plasticity**）。

对损伤有冗余性**(tolerance)**。大脑即使有很大一部分受到了损伤，它仍然能够执行复杂的工作。一个著名的试验就是训练老鼠在一个迷宫中行走。然后，科学家们将其大脑一部分一部分地、越来越大地加以切除。他们发现，即使老鼠的很大的一部大脑被切除后，它们仍然能在迷宫中找到行走路径。这一事实证明了，在大脑中，知识并不是保存在一个局部地方。另外所作的一些试验则表明，如果大脑的一小部分受到损伤，则神经细胞能把损伤的连接重新生长出来。

处理信息的效率极高。神经细胞之间电-化学信号的传递，与一台数字计算机中 **CPU** 的数据传输相比，速度是非常慢的，但因神经细胞采用了并行的工作方式，使得大脑能够同时处理大量的数据。例如，大脑视觉皮层在处理通过我们的视网膜输入的一幅图象信号时，大约只要 **100ms** 的时间就能完成。考虑到你的神经细胞的平均工作频率只有 **100Hz**，**100ms** 的时间就意味只能完成 **10** 个计算步骤！想一想通过我们眼睛的数据量有多大，你就可以看到这真是一个难以置信的伟大工程了。

善于归纳推广。大脑和数字计算机不同，它极擅长的事情之一就是模式识别，并能根据已熟悉信息进行归纳推广**(generlize)**。例如，我们能够阅读他人所写的手稿上的文字，即使我们以前从来没见过他所写的东西。

它是有意识的。意识（**consciousness**）是神经学家和人工智能的研究者广泛而又热烈地在辩论的一个话题。有关这一论题已有大量的文献出版了，但对于意识实际究竟是什么，至今尚未取得实

质性的统一看法。我们甚至不能同意只有人类才有意识，或者包括动物王国中人类的近亲在内才有意识。一头猩猩有意识吗？你的猫有意识吗？上星期晚餐中被你吃掉的那条鱼有意识吗？

因此，一个人工神经网络**(Artificial neural network,**简称 **ANN)**就是要在当代数字计算机现有规模的约束下，来模拟这种大量的并行性**,**并在实现这一工作时，使它能显示许多和生物学大脑相类似的特性。下面就让我们瞧瞧它们的表演吧！

- **3** 数字版的神经网络 **(The Digital Version)**

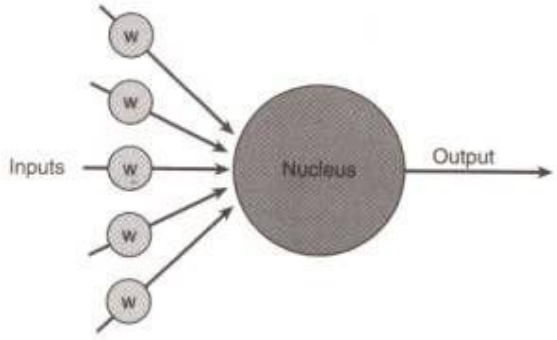
上面我们看到了生物的大脑是由许多神经细胞组成，同样，模拟大脑的人工神经网络 **ANN** 是由许多叫做人工神经细胞（**Artificial neuron**，也称人工神经原，或人工神经元）的细小结构模块组成。人工神经细胞就像真实神经细胞的一个简化版，但采用了电子方式来模拟实现。一个人工神经网络中需要使用多少个数的人工神经细胞，差别可以非常大。有的神经网络只需要使用 **10** 个以内的人工神经细胞，而有的神经网络可能需要使用几千个人工神经细胞。这完全取决于这些人工神经网络准备实际用来做什么。

有趣的事实

有一个叫 **Hugo de Garis** 的同行，曾在一个雄心勃勃的工程中创建并训练了一个包含 **1000,000,000** 个人工神经细胞的网络。这个人工神经网络被他非常巧妙地建立起来了，它采用蜂房式自动机结构，目的就是为一机器客户定制一个叫做 **CAM BrainMachine(**"**CAM** 大脑机器"**)** 的机器（**CAM** 就是 **Cellular Automata Machine** 的缩写）。此人曾自夸地宣称这一人工网络机器将会有一只猫的智能。许多神经网络研究人员认为他是在"登星"了，但不幸的是，雇用他的公司在他的梦想尚未实现之前就破产了。此人现在犹他州，是犹他州大脑工程（**Utah Brain Project**）的领导。时间将会告诉我们他的思想最终是否能变成实际有意义的东西。**[**译注**]**

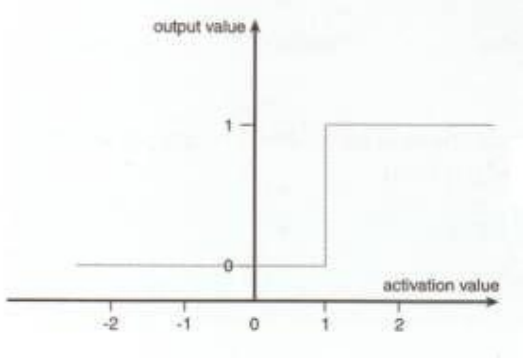
我想你现在可能很想知道，一个人工神经细胞究竟是一个什么样的东西？但是，它实际上什么东西也不像**;** 它只是一种抽象。还是让我们来察看一下图 **2** 吧，这是表示一个人工神经细胞的一种形式。

[译注**]Hugo de Garis** 现在为犹他州立大学教授，有关他和他的 **CAM** 机器，可在该校网站的一个网页上看到报道，其上有真实的照片，见 **http://www.cs.usu.edu/~degaris**



- 图 **2** 一个人工神经细胞

图中，左边几个灰底圆中所标字母 **w** 代表浮点数，称为权重（**weight**，或权值，权数）。进入人工神经细胞的每一个 **input(**输入**)**都与一个权重 **w** 相联系，正是这些权重将决定神经网络的整体活跃性。你现在暂时可以设想所有这些权重都被设置到了**-** 1 和 1 之间的一个随机小数。因为权重可正可负，故能对与它关联的输入施加不同的影响，如果权重为正，就会有激发（**excitory**）作用，权重为负，则会有抑制（**inhibitory**）作用。当输入信号进入神经细胞时，它们的值将与它们对应的权重相乘，作为图中大圆的输入。大圆的'核'是一个函数，叫激励函数**(activation function)**，它把所有这些新的、经过权重调整后的输入全部加起来，形成单个的激励值**(activation value)**。激励值也是一浮点数，且同样可正可负。然后，再根据激励值来产生函数的输出也即神经细胞的输出：如果激励值超过某个阀值（作为例子我们假设阀值为 **1.0**），就会产生一个值为 **1** 的信号输出；如果激励值小于阀值 **1.0**，则输出一个 **0**。这是人工神经细胞激励函数的一种最简单的类型。在这里，从激励值产生输出值是一个阶跃函数**[**译注**]**。看一看图 **3** 后你就能猜到为什么有这样的名称。



- 图 **3** 阶跃激励函数

[译注**]** 由图可知阶跃函数是一元的，而激励函数既然能把多个输入相加应为多元，故需加以区别。

如果到目前为止你对这些还没有获得很多感觉，那也不必担心。窍门就是**:** 不要企图去感觉它，暂时就随波逐流地跟我一起向前走吧。在经历本章的若干处后，你最终就会开始弄清楚它们的意义。而现在，就放松一点继续读下去吧。

3.1 现在需要一些数学了（**Now for Some Math**）

今后讨论中，我将尽量把数学降低到绝对少量，但学习一些数学记号对下面还是很有用的。我将把数学一点一点地喂给你，在到达有关章节时向你介绍一些新概念。我希望采用这样的方式能使你的头脑能更舒适地吸收所有的概念，并使你在开发神经网络的每个阶段都能看到怎样把数学应用到工作中。现在首先让我们来看一看，怎样把我在此之前告诉你的所有知识用数学方式表达出来。

一个人工神经细胞**(**从现在开始，我将把"人工神经细胞"简称它为"神经细胞"**)** 可以有任意 **n** 个输入，**n** 代表总数。可以用下面的数学表达式来代表所有 **n** 个输入：

x1, x2, x3, x4, x5, ..., xn

同样 **n** 个权重可表达为**:**

w1, w2, w3, w4, w5 ..., wn

请记住，激励值就是所有输入与它们对应权重的之乘积之总和，因此，现在就可以写为**:**

a = w1x1 + w2x2 + w3x3 + w4x4 + w5x5 +...+ wnxn

以这种方式写下的求和式，我在第 **5** 章"建立一个更好的遗传算法"中已提到，可以用希腊字母 Σ 来简化：

$$a = \sum_{j=0}^{n} w_j x_j$$

- 注：

神经网络的各个输入，以及为各个神经细胞的权重设置，都可以看作一个 **n** 维的向量。你在许多技术文献中常常可以看到是以这样的方式来引用的。

下面我们来考察在程序中应该怎样实现？假设输入数组和权重数组均已初始化为 **x[n]**和 **w[n]**，则求和的代码如下**:**

double activation = 0;

for(int i=0; i 激活阀值 1，所以这个神经细胞将输出 **1**。

在进一步读下去之前，请你一定要确切弄懂激励函数怎样计算。

表 **2** 神经细胞激励值的计算

输 入 权 重 输入*****权重的乘积 运行后总和

1 0.5 0.5 0.5

0 -0.2 0 0.5

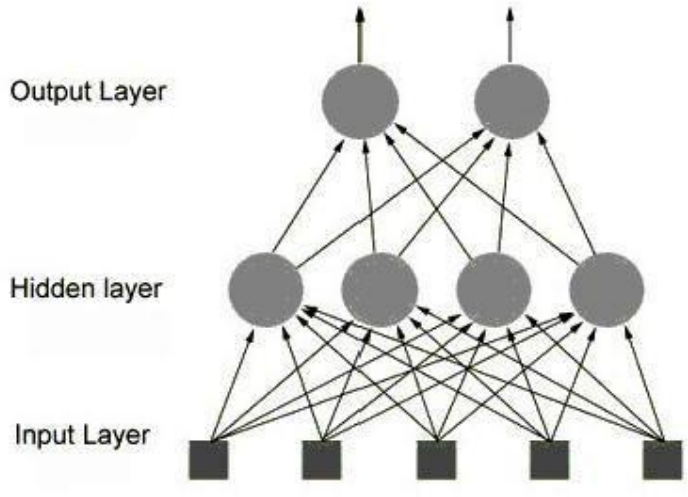
1 -0.3 -0.3 0.2

1 0.9 0.9 1.1

0 0.1 0 1.1

- **3.2** 行，我知道什么是神经细胞了，但用它来干什么呢？

大脑里的生物神经细胞和其他的神经细胞是相互连接在一起的。为了创建一个人工神经网络，人工神经细胞也要以同样方式相互连接在一起。为此可以有许多不同的连接方式，其中最容易理解并且也是最广泛地使用的，就是如图 **5** 所示那样，把神经细胞一层一层地连结在一起。这一种类型的神经网络就叫前馈网络（**feedforword network**）。这一名称的由来，就是因为网络的每一层神经细胞的输出都向前馈送（**feed**）到了它们的下一层（在图中是画在它的上面的那一层**)**，直到获得整个网络的输出为止。



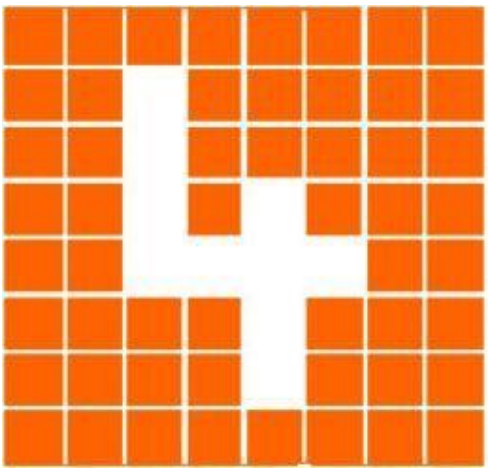
- 图 **5** 一个前馈网络

由图可知，网络共有三层（译注：输入层不是神经细胞，神经细胞只有两层）。输入层中的每个输入都馈送到了隐藏层，作为该层每一个神经细胞的输入；然后，从隐藏层的每个神经细胞的输出都连到了它下一层（即输出层）的每一个神经细胞。图中仅仅画了一个隐藏层，作为前馈网络，一般地可以有任意多个隐藏层。但在对付你将处理的大多数问题时一层通常是足够的。事实上，有一些问题甚至根本不需要任何隐藏单元，你只要把那些输入直接连结到输出神经细胞就行了。另外，我为图 **5** 选择的神经细胞的个数也是完全任意的。每一层实际都可以有任何数目的神经细胞，这完全取决于要解决的问题的复杂性。但神经细胞数目愈多，网络的工作速度也就愈低，由于这一缘故，以及为了其他的几种原因（我将在第 **9** 章作出解释），网络的规模总是要求保持尽可能的小。

到此我能想象你或许已对所有这些信息感到有些茫然了。我认为，在这种情况下，我能做的最好的事情，就是向你介绍一个神经网络在现实世界中的实际应用例子，它有望使你自己的大脑神经细胞得到兴奋！不错吧？好的，下面就来了**...**

你可能已听到或读到过神经网络常常用来作模式识别。这是因为它们善于把一种输入状态（它所企图识别的模式）映射到一种输出状态（它曾被训练用来识别的模式）。

下面我们来看它是怎么完成的。我们以字符识别作为例子。设想有一个由 **8x8** 个格子组成的一块面板。每一个格子里放了一个小灯，每个小灯都可独立地被打开（格子变亮）或关闭（格子变黑），这样面板就可以用来显示十个数字符号。图 **6** 显示了数字**"4"**。



- 图 **6** 用于字符显示的矩阵格点

要解决这一问题，我们必需设计一个神经网络，它接收面板的状态作为输入，然后输出一个 **1** 或 **0**；输出 **1** 代表 **ANN** 确认已显示了数字**"4"**，而输出 **0** 表示没有显示**"4"**。因此，神经网络需要有 **64** 个输入**(**每一个输入代表面板的一个具体格点**)** 和由许多神经细胞组成的一个隐藏层，还有仅有一个神经细胞的输出层，隐藏层的所有输出都馈送到它。我真希望你能在你的头脑中画出这个图来，因为要我为你把所有这些小圆和连线统统画出来确实不是一桩愉快的事。

一旦神经网络体系创建成功后，它必须接受训练来认出数字**"4"**。为此可用这样一种方法来完成：先把神经网的所有权重初始化为任意值。然后给它一系列的输入，在本例中，就是代表面板不同配置的输入。对每一种输入配置，我们检查它的输出是什么，并调整相应的权重。如果我们送给网络的输入模式不是**"4"**， 则我们知道网络应该输出一个 **0**。因此每个非**"4"**字符时的网络权重应进行调节，使得它的输出趋向于 **0**。当代表**"4"**的模式输送给网络时，则应把权重调整到使输出趋向于 **1**。

如果你考虑一下这个网络，你就会知道要把输出增加到 **10** 是很容易的。然后通过训练，就可以使网络能识别 **0** 到 **9** 的所有数字。但为什么我们到此停止呢？我们还可以进一步增加输出，使网络能识别字母表中的全部字符。这本质上就是手写体识别的工作原理。对每个字符，网络都需要接受许多训练，使它认识此文字的各种不同的版本。到最后，网络不单能认识已经训练的笔迹，还显示了它有显著的归纳和推广能力。也就是说，如果所写文字换了一种笔迹，它和训练集中所有字迹都略有不同，网络仍然有很大几率来认出它。正是这种归纳推广能力，使得神经网络已经成为能够用于无数应用的一种无价的工具，从人脸识别、医学诊断，直到跑马赛的预测，另外还有电脑游戏中的 **bot**（作为游戏角色的机器人）的导航，或者硬件的 **robot**（真正的机器人）的导航。

这种类型的训练称作有监督的学习（**supervised learnig**），用来训练的数据称为训练集（**training set**）。调整权重可以采用许多不同的方法。对本类问题最常用的方法就是反向传播（**backpropagation**，简称 **backprop** 或 **BP**）方法。有关反向传播问题，我将会在本书的后面，当你已能训练神经网络来识别鼠标走势时，再来进行讨论。在本章剩余部分我将集中注意力来考察另外的一种训练方式，即根本不需要任何导师来监督的训练，或称无监督学习（**unsupervised learnig**）。

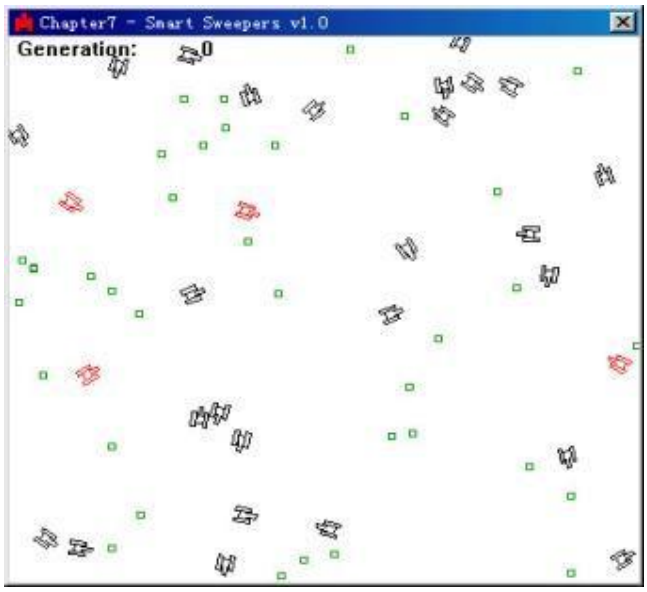
这样我已向你介绍了一些基本的知识，现在让我们来考察一些有趣的东西，并向你介绍第一个代码工程。

游戏编程中的人工智能技术

(连载之三**)**

4. 聪明的扫雷机工程（**Smart Minesweeper Project**）

我要向你介绍的第一个完整例子，是怎么使用神经网络来控制具有人工智能的扫雷机的行为。扫雷机工作在一个很简单的环境中，那里只有扫雷机以及随机散布的许多地雷



- 图 **7** 运行中的演示程序。

尽管书上图形画成了黑白色，但当你运行程序时性能最好的扫雷机将显现为红色。地雷，你可能已经猜到，就是那些小方形。工程的目标是创建一个网络，它不需要从我们这里得到任何帮助，就能自己进行演化（**evolve**）去寻找地雷。为了实现这一功能，网络的权重将被编码到基因组中，并用一个遗传算法来演化它们。

怎么样，很酷吧？

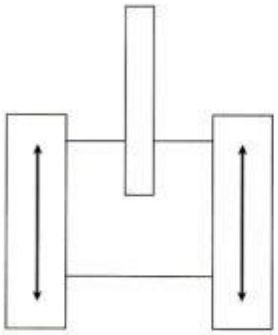
提示（重要）

如果你跳过前面的一些章节来到这里，而你又不了解怎样使用遗 传算法，则在进一步阅读下面的内容之前，你应回到前面去补读一下有关遗传算法的内容。

首先让我解释人工神经网络**(ANN)**的体系结构。我们需要决定输入的数目、输出的数目、还有隐藏层和每个隐藏层中隐藏单元的数目。

4.1 选择输出（**Choosing the Outputs**）

那么，人工神经网络怎样控制扫雷机的行动呢？很好！我们把扫雷机想象成和坦克车一样，通过左右 **2** 个能转动的履带式轮轨（**track**）来行动的。见图案 **9.8**。



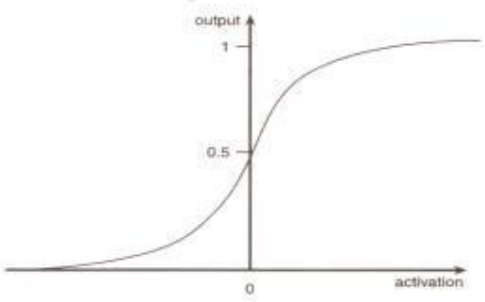
- 图 **8** 扫雷机的控制

扫雷机向前行进的速度，以及向左、向右转弯的角度，都是通过改变 **2** 个履带轮的相对速度来实现的。因此，神经网络需要 **2** 个输入，**1** 个是左侧履带轮的速度，另一个是右侧履带轮的速度。

啊，但是**...,**我听见你在嘀咕了。如果网络只能输出一个 1 或一个 0，我们怎么能控制车轨移动的快慢呢？ 你是对的；如果利用以前描述的阶跃函数来决定输出，我们就根本无法控制扫雷机实际移动。幸好，我有一套戏法，让我卷起袖子来，把激励函数的输出由阶跃式改变成为在 0 － 1 之间连续变化的形式，这样就可以供扫雷机神经细胞使用了。为此，有几种函数都能做到这样，我们使用的是一个被称为逻辑斯蒂 **S** 形函数（**logistic sigmoid function**）**[**译注 **1]**。该函数所实现的功能，本质上说，就是把神经细胞原有的阶跃式输出曲线钝化为一光滑曲线，后者绕 **y** 轴 **0.5** 处点对称**[**译注 **2]**，如图 **9** 所示。

[译注 **1] logistic** 有'计算的'或'符号逻辑的'等意思在内，和'逻辑的**(logic)**'意义不同。

[译注 **2]** 点对称图形绕对称点转 **180** 度后能与原图重合。若 **f(x)**以原点为点对称**,**则有 **f(-x)=-f(x)**



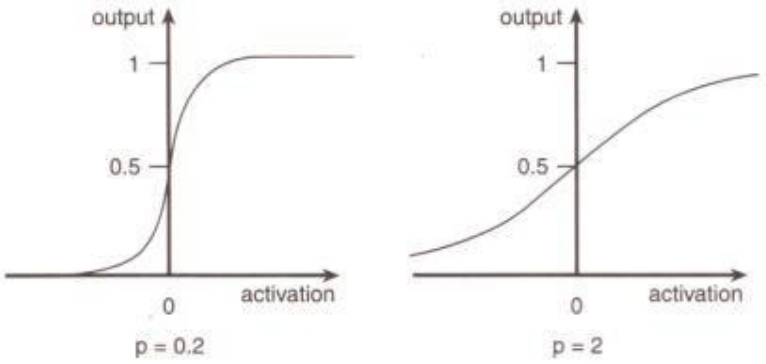
- 图 **9** **S** 形曲线。

当神经细胞的激励值趋于正、负无穷时，**S** 形函数分别趋于 1 或 0 。负的激励值对应的函数值都 **0.5**。**S** 形函数用数学表达式写出来则为**:**

$$\mathbf{output} = \frac{1}{1+\mathbf{e}^{-a/p}}$$

- 这个方程看上去可能会吓唬一些人，但其实很简单。**e** 是数学常数，近似等于 **2.7183**，**a** 是神经细胞的激励值，它是函数的自变量，而 **p** 是一个用来控制曲线形状变化快慢或陡峭性的参数。**p** 通常设定为 **1**。当 **p** 赋以较大值时，曲线就显得平坦，反之，就会使曲线变为陡峭。见图 **1O**。很低的 **p** 值所生成的函数就和阶跃函数近似。**P** 值的大小用来控制何时使神经网络由低变高开始翻转有很大作用，但是在本例子中我们将它保持为 1 。

注：**"S** 型**"**的英文原名 **Sigmoid** 或 **Sigmoidal** 原来是根据希腊字**"Sigma"**得来的，但非常巧它也可以说成是曲线的一种形状。



- 图 **7**。**10** 不同的 **S** 形响应曲线。

4.2 选择输入（**Choosing the Inputs**）

上面我们已经把输出安排好了，现在我们来考虑输入，确定网络需要什么样的输入？为此，我们必须想象一下扫雷机的具体细节：需要什么样的信息才能使它朝地雷前进？你可能想到的第一个输入信息清单是：

扫雷机的位置**(x1,y1)**

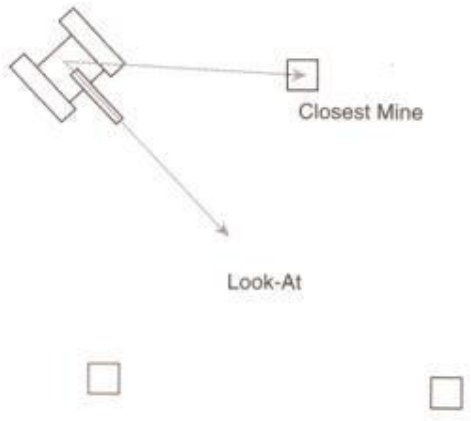
与扫雷机最靠近的地雷的位置**(x2,y2)**

代表扫雷机前进方向的向量**(x3,y3)**

这样一共得到 **6** 个输入。但是，要网络使用这些输入，工作起来就非常困难，因为，网络在像我们希望的那样执行工作之前，必须寻找所有 **6** 个输入之间的数学关系，而这有相当工作量。可以把此作为一个练习倒是很理想的：去试试如何给出最少数量的输入而仍能为网络传达解决问题所需要的全部信息。 你的网络使用的输入愈少，网络所要求的神经细胞数目也愈少。而较少的神经细胞就意味更快速的训练和更少的计算，有利于网络更高速度的工作。

只要作少量的额外考虑，就能够把输入的个数减少为 **4**，这就是图 **11** 中所画出的两个向量的 4 个参数。

把神经网络的所有输入进行规范化是一种好想法。这里的意思并不是说每个输入都要改变大小使它们都在 **0**～**1** 间，而是说每一个输入应该受到同等重视。例如，拿我们已经讨论过的扫雷机输入为例。瞄准向量或视线向量（**look-at vector**）总是一个规范化向量，即长度等于 **1**，分量 **x** 和 **y** 都在 **0**～**1** 间。但从扫雷机到达其最近地雷的向量就可能很大，其中的一个分量甚至有可能和窗体的宽度或高度一样大。如果这个数据以它的原始状态输入到网络，网络对有较大值的输入将显得更灵敏，由此就会使网络性能变差。因此，在信息输入到神经网络中去之前，数据应预先定比（**scaled**）和标准化（**standardized**），使它们大小相似（**similar**）。在本特例中，由扫雷机引到与其最接近地雷的向量需要进行规范化（**normalized**）。这样可以使扫雷机的性能得到改良。



- 图 **11** 选择输入。

小技巧：

有时，你把输入数据重新换算（**rescale**）一下，使它以 **0** 点为中心，就能从你的神经网络获得最好的性能。这一小窍门在你设计网络时永远值得一试。但我在扫雷机工程中没有采用这一方法，这是因为我想使用一种更直觉的方法。

- **4.3** 隐藏的神经细胞要多少？（**How many Hidden Neurons?**）

到此我们已把输入、输出神经细胞的数目和种类确定下来了，下一步是确定隐藏层的数目，并确定每个隐藏层中神经细胞必须有多少？但遗憾的是，还没有一种确切的规则可用来计算这些。它们的开发又需要凭个人的"感觉"了。某些书上和文章中确实给过一些提纲性的东西,告诉你如何去决定隐藏神经细胞个数，但业内专家们的一致看法是：你只能把任何建议当作不可全信的东西，主要还要靠自己的不断尝试和失败中获得经验。但你通常会发现，你所遇到的大多数问题都只要用一个隐藏层就能解决。所以，本领的高低就在于如何为这一隐藏层确定最合适的神经细胞数目了。显然，个数是愈少愈好，因为我前面已经提及，数目少的神经细胞能够造就快速的网络。通常，为了确定出一个最优总数，我总是在隐藏层中采用不同数目的神经细胞来进行试验。我在本章所编写的神经网络工程的.

第一版本中一共使用了 **10** 个隐藏神经细胞（当然，我的这个数字也不一定是最好的）。你应围绕这个数字的附近来做游戏，并观察隐藏层神经细胞的数目对扫雷机的演化会产生什么样的影响。不管怎样，理论已经够了，让我们拿一个具体程序来看看吧！你可以在本书所附光盘的 **Chapter7/Smart Sweepers v1.0** 文件夹中找到本章下面几页即将描述的所有程序的源码。

- **4.4 CNeuralNet.h**（神经网络类的头文件）

在 **CNeuralNet.h** 文件中，我们定义了人工神经细胞的结构、定义了人工神经细胞的层的结构、以及人工神经网络本身的结构。首先我们来考察人工神经细胞的结构。

4.4.1 SNeuron（神经细胞的结构）

这是很简单的结构。人工神经细胞的结构中必须有一个正整数来纪录它有多少个输入，还需要有一个向量 **std:vector** 来表示它的权重。请记住，神经细胞的每一个输入都要有一个对应的权重。

```
Struct SNeuron
{
// 进入神经细胞的输入个数

int m_NumInputs;

// 为每一输入提供的权重
vector m_vecWeight;

//构造函数
SNeuron(int NumInputs);
};
```

以下就是 **SNeuron** 结构体的构造函数形式:

```
SNeuron::SNeuron(int NumInputs): m_NumInputs(NumInputs+1)
(
```

```
// 我们要为偏移值也附加一个权重，因此输入数目上要 +1
for (int i=0; i<NumInputs+1; ++i)
{

// 把权重初始化为任意的值
m_vecWeight.push_back(RandomClamped());
}
}
```

由上可以看出，构造函数把送进神经细胞的输入数目 **NumInputs** 作为一个变元，并为每个输入创建一个随机的权重。所有权重值在**-1** 和 **1** 之间。

- 这是什么？ 我听见你在说。这里多出了一个权重！ 不错，我很高兴看到你能注意到这一点，因为这一个附加的权重十分重要。但要解释它为什么在那里，我必须更多地介绍一些数学知识。回忆一下你就能记得，激励值是所有输入*权重的乘积的总和，而神经细胞的输出值取决于这个激励值是否超过某个阀值(t)。这可以用如下的方程来表示:

w1x1 + w2x2 + w3x3 +...+ wnxn >= t

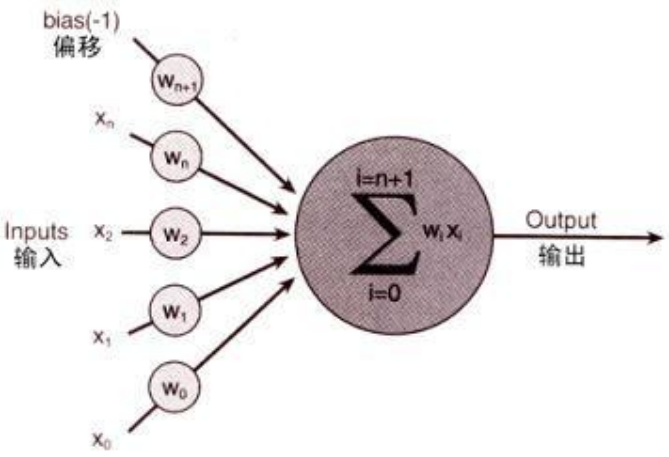
上式是使细胞输出为 1 的条件。因为网络的所有权重需要不断演化（进化），如果阀值的数据也能一起演化，那将是非常重要的。要实现这一点不难，你使用一个简单的诡计就可以让阀值变成权重的形式。从上面的方程两边各减去 t，得:

w1x1 + w2x2 + w3x3 +...+ wnxn –t >= 0

这个方程可以再换用一种形式写出来，如下:

w1x1 + w2x2 + w3x3 +...+ wnxn + t *(–1) >= 0

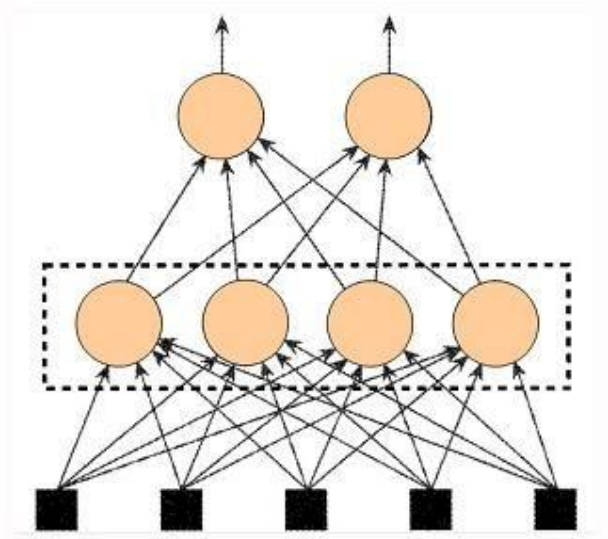
到此，我希望你已能看出，阀值 t 为什么可以想像成为始终乘以输入为 - 1 的权重了。这个特殊的权重通常叫偏移（bias），这就是为什么每个神经细胞初始化时都要增加一个权重的理由。现在，当你演化一个网络时，你就不必再考虑阀值问题，因为它已被内建在权重向量中了。怎么样，想法不错吧？为了让你心中绝对敲定你所学到的新的人工神经细胞是什么样子，请再参看一下图 12。



- 图 12 带偏移的人工神经细胞。

4.4.2 SNeuronLayer（神经细胞层的结构）

神经细胞层 SNeuronLayer 的结构很简单；它定义了一个如图 13 中所示的由虚线包围的神经细胞 SNeuron 所组成的层。



- 以下就是层的定义的源代码，它应该不再需要任何进一步的解释:

```
struct SNeuronLayer
{
  // 本层使用的神经细胞数目
   int             m_NumNeurons;

// 神经细胞的层
   vector<SNeuron> m_vecNeurons;


   SNeuronLayer(int NumNeurons, int NumInputsPerNeuron);
};
```

4.4.3 CNeuralNet（神经网络类）

这是创建神经网络对象的类。让我们来通读一下这一个类的定义:

```
class CNeuralNet
{

private:
```

```
int m_NumInputs;
int m_NumOutputs;
int m_NumHiddenLayers;
int         m_NeuronsPerHiddenLyr;
// 为每一层（包括输出层）存放所有神经细胞的存储器
vector<SNeuronLayer> m_vecLayers;
```

所有 private 成员由其名称容易得到理解。需要由本类定义的就是输入的个数、输出的个数、隐藏层的数目、以及每个隐藏层中神经细胞的个数等几个参数。

```
public:
CNeuralNet();
```

该构造函数利用 ini 文件来初始化所有的 Private 成员变量，然后再调用 CreateNet 来创建网络。

```
// 由 SNeurons 创建网络
void CreateNet();
```

我过一会儿马上就会告诉你这个函数的代码。

```
// 从神经网络得到（读出）权重
vector<double> GetWeights()const;
```

由于网络的权重需要演化，所以必须创建一个方法来返回所有的权重。这些权重在网络中是以实数型向量形式表示的，我们将把这些实数表示的权重编码到一个基因组中。当我开始谈论本工程的遗传算法时，我将为您确切说明权重如何进行编码。

```
// 返回网络的权重的总数
int GetNumberOfWeights()const;
// 用新的权重代替原有的权重
void PutWeights(vector<double> &weights);
```

这一函数所做的工作与函数 GetWeights 所做的正好相反。当遗传算法执行完一代时，新一代的权重必须重新插入神经网络。为我们完成这一任务的是 PutWeight 方法。

```
// S 形响应曲线
inline double      Sigmoid(double activation, double response);
```

当已知一个神经细胞的所有输入*重量的乘积之和时，这一方法将它送入到 S 形的激励函数。

// 根据一组输入，来计算输出

vector<double> Update(vector<double> &inputs);

对此 Update 函数函数我马上就会来进行注释的。

}; // 类定义结束

4.4.3.1 CNeuralNet::CreateNet（创建神经网络的方法）

我在前面没有对 CNeuralNet 的 2 个方法加以注释，这是因为我要为你显示它们的更完整的代码。这 2 个方法的第一个是网络创建方法 CreateNet。它的工作就是把由细胞层 SNeuronLayers 所收集的神经细胞 SNeurons 聚在一起来组成整个神经网络，代码为:

```
void CNeuralNet::CreateNet()
{
// 创建网络的各个层
if (m_NumHiddenLayers > 0)
{

//创建第一个隐藏层［译注］
m_vecLayers.push_back(SNeuronLayer(m_NeuronsPerHiddenLyr,
m_NumInputs));

for( int i=O; i<m_NumHiddenLayers-l; ++i)
{
m_vecLayers.push_back(SNeuronLayer(m_NeuronsPerHiddenLyr，

m_NeuronsPerHiddenLyr));
}
```

［译注］如果允许有多个隐藏层，则由接着 for 循环即能创建其余的隐藏层。

```
// 创建输出层
m_vecLayers.push_back(SNeuronLayer(m_NumOutput,m_NeuronsPerHiddenLyr));
```

```
}

else //无隐藏层时，只需创建输出层
{
// 创建输出层
m_vecLayers.push_back(SNeuronLayer(m_NumOutputs, m_NumInputs));
}
}
```

4.4.3.2 CNeuralNet::Update（神经网络的更新方法）

Update 函数(更新函数)称得上是神经网络的“主要劳动力”了。这里，输入网络的数据 input 是以双精度向量 std::vector 的数据格式传递进来的。Update 函数通过对每个层的循环来处理输入*权重的相乘与求和，再以所得的和数作为激励值，通过 S 形函数来计算出每个神经细胞的输出，正如我们前面最后几页中所讨论的那样。Update 函数返回的也是一个双精度向量 std::vector，它对应的就是人工神经网络的所有输出。

请你自己花两分钟或差不多的时间来熟悉一下如下的 Update 函数的代码，这能使你正确理解我们继续要讲的其他内容:

```
vector<double> CNeuralNet::Update(vector<double> &inputs)
{
// 保存从每一层产生的输出

vector<double> outputs;
int cWeight = 0;
// 首先检查输入的个数是否正确
if (inputs.size() != m_NumInputs)
{
// 如果不正确，就返回一个空向量

return outputs;
}

// 对每一层,...
for (int i=0; i<m_NumHiddenLayers+1; ++i)
{
```

```
if (i>O)

{
inputs = outputs;
}
outputs.clear();

cWeight = 0;

// 对每个神经细胞,求输入*对应权重乘积之总和。并将总和抛给 S 形函数,以计算输出

for (int j=0; j<m_vecLayers[i].m_NumNeurons; ++j)
{
double netinput = 0;

int NumInputs = m_vecLayers[i].m_vecNeurons[j].m_NumInputs;

// 对每一个权重
for (int k=O; k<NumInputs-l; ++k)

{
// 计算权重*输入的乘积的总和。
netinput += m_vecLayers[i].m_vecNeurons[j].m_vecWeight[k] *
                    inputs[cWeight++];
}

// 加入偏移值

netinput += m_vecLayers[i].m_vecNeurons[j].m_vecWeight[NumInputs-1] *
CParams::dBias;
```

别忘记每个神经细胞的权重向量的最后一个权重实际是偏移值，这我们已经说明过了，我们总是将它设置成为 –1 的。我已经在 ini 文件中包含了偏移值，你可以围绕它来做文章，考察它对你创建的网络的功能有什么影响。不过，这个值通常是不应该改变的。

```
// 每一层的输出，当我们产生了它们后，我们就要将它们保存起来。但用 Σ 累加在一起的
```

// 激励总值首先要通过 S 形函数的过滤，才能得到输出

outputs.push_back(Sigmoid(netinput,CParams::dActivationResponse)); cWeight = 0:

}

}

return outputs;

}

游戏编程中的人工智能技术

.

<神经网络入门>

.

(连载之五)

4.5 神经网络的编码（Encoding the Network）

在本书的开始几章中，你已经看到过怎样用各种各样的方法为遗传算法编码。但当时我并没有向你介绍过

一个用实数编码的具体例子，因为我知道我要留在这里向你介绍。我曾经讲到，为了设计一个前馈型神经网络，

编码是很容易的。我们从左到右读每一层神经细胞的权重，读完第一个隐藏层，再向上读它的下一层，把所读

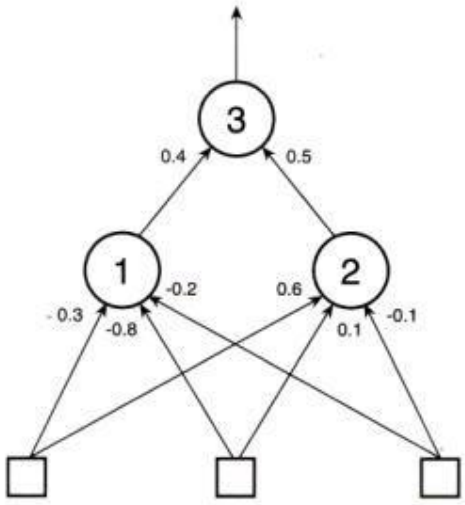
到的数据依次保存到一个向量中，这样就实现了网络的编码。因此，如果我们有图 14 所示的网络，则它的权重

编码向量将为:

0.3, -O.8， -O.2， 0.6， O.1， -0.l， 0.4， 0.5

在这一网络中，为了简单，我没有把偏移值的权重包括进去。但在实际实现编码时，你必须包含偏移值这

个权重，否则你肯定无法获得你所需要的结果。



- 图 14 为权重编码。

在此之前讲的事情你都懂了吗？好极了，那下面就让我们转来考虑，怎样用遗传算法来操纵已编码的基因吧。

4.6 遗传算法（The Genetic Algorithm）

到此，所有的权重已经象二进制编码的基因组那样，形成了一个串，我们就可以象本书早先讨论过的那样

来应用遗传算法了。遗传算法（GA）是在扫雷机已被允许按照用户指定的帧数（为了某种缘故，我下面更喜欢

将帧数称作滴答数，英文是 ticks）运转后执行的。你可以在 ini 文件中找到这个滴答数（iNumTicks）的设置。

下面是基因组结构体的代码。这些对于你应该是十分面熟的东西了。

```
Struct SGenome

{

vector <double>      vecWeights;


double               dFitness;


SGenome():dFitness(0) {}
```

```
SGenome(vector <double> w, double f):vecWeights(w),dFitness(f){}

//重载'<'的排序方法
friend bool operator<(const SGenome& lhs, const SGenome& rhs)
{
    return (lhs.dFitness < rhs.dFitness);

}
};
```

从上面的代码你可看出，这一 SGenome 结构和我们在本书所有其他地方见到的 SGenome 结构几乎完全一致，唯一的差别就是这里的染色体是一个双精度向量 std::vector。因此，可以和通常一样来应用杂交操作和选择

操作。但突变操作则稍微有些不同，这里的权重值是用一个最大值为 dMaxPerturbation 的随机数来搔扰的。这一

参数 dMaxPerturbation 在 ini 文件中已作了声明。另外，作为浮点数遗传算法，突变率也被设定得更高些。在本工

程中，它被设成为 0.1。

下面就是扫雷机工程遗传算法类中所见到的突变函数的形式:

```
void CGenAlg::Mutate(vector<double> &chromo)
{
// 遍历权重向量，按突变率将每一个权重进行突变
for (int i=0; i<chromo.size(); ++i)
{

// 我们要骚扰这个权重吗?
if (RandFloat() < m_dMutationRate)
{
// 为权重增加或减小一个小的数量
chromo[i] += (RandomClamped() * CParams::dMaxPerturbatlon);
}

}
}
```

如同以前的工程那样，我已为 v1.0 版本的 Smart Minesweepers 工程保留了一个非常简单的遗传算法。这样

就能给你留下许多余地，可让你利用以前学到的技术来改进它。就象大多数别的工程一样，v1.O 版只用轮盘赌

方式选精英，并采用单点式杂交。

注意:

当程序运行时，权重可以被演化成为任意的大小，它们不受任何形式的限制。

4.7 扫雷机类（The CMinesweeper Class）

这一个类用来定义一个扫雷机。就象上一章描述的登月艇类一样，扫雷机类中有一个包含了扫雷机位置、

速度、以及如何转换方向等数据的纪录。类中还包含扫雷机的视线向量（look-at vector）；它的 2 个分量被用

来作为神经网络的 2 个输入。这是一个规范化的向量，它是在每一帧中根据扫雷机本身的转动角度计算出来的，

它指示了扫雷机当前是朝着哪一个方向，如图 11 所示。

下面就是扫雷机类 CMinesweeper 的声明:

```
class CMinesweeper
{
private:
// 扫雷机的神经网络
CNeuralNet m_ItsBrain;

// 它在世界坐标里的位置
SVector2D m_vPosition;

// 扫雷机面对的方向
SVector2D m_vLookAt;
// 它的旋转(surprise surprise)
double m_dRotation;

double m_dSpeed;
// 根据 ANN 保存输出
```

double m_lTrack,

m_rTrack;

m_lTrack 和 m_rTrack 根据网络保存当前帧的输出。

这些就是用来决定扫雷机的移动速率和转动角度的数值。

// 用于度量扫雷机适应性的分数

double m_dFitness;

每当扫雷机找到一个地雷，它的适应性分数就要增加。

// 扫雷机画出来时的大小比例

double m_dScale;

// 扫雷机最邻近地雷的下标位置

int m_iClosestMine;

在控制器类 CControl1er 中，有一个属于所有地雷的成员向量 std::vector。

而 m_iClosestMine 就是代表最靠近扫雷机的那个地雷在该向量中的位置的下标。

public:

CMinesweeper();

// 利用从扫雷机环境得到的信息来更新人工神经网

bool Update(vector<SVector2D> &mines);

// 用来对扫雷机各个顶点进行变换，以便接着可以画它出来

void WorldTransform(vector<SPoint> &sweeper);

// 返回一个向量到最邻近的地雷

5Vector2D GetClosestMine(vector<SVector2D> &objects);

// 检查扫雷机看它是否已经发现地雷

int CheckForMine(vector<SVector2D> &mines, double size);

void Reset();

// ---------- 定义各种供访问用的函数

SVector2D Position()const { return m_vPosition; }

void IncrementFitness(double val) { m_dFitness += val; }

double Fitness()const { return m_dFitness; }

```
void PutWeights(vector<double> &w) { m_ItsBrain.PutWeights(w); }
int GetNumberOfWeights()const
{ return m_ItsBrain.GetNumberOfWeights(); }
};
```

4.7.1 The CMinesweeper::Update Function（扫雷机更新函数）

需要更详细地向你说明的 CMinesweeper 类的方法只有一个，这就是 Update 更新函数。该函数在每一帧中

都要被调用，以更新扫雷机神经网络。让我们考察这函数的肚子里有些什么货色:

```
bool CMinesweeper::Update(vector<SVector2D> &mines)
{
//这一向量用来存放神经网络所有的输入
vector<double> inputs;
//计算从扫雷机到与其最接近的地雷（2 个点）之间的向量

SVector2D vClosestMine = GetClosestMine(mines);
//将该向量规范化
Vec2DNormalize(vClosestMine);
```

首先，该函数计算了扫雷机到与其最靠近的地雷之间的向量，然后使它规范化。（记住，向量规范化后它

的长度等于 1。）但扫雷机的视线向量（look-at vector）这时不需要再作规范化，因为它的长度已经等于 1 了。

由于两个向量都有效地化成了同样的大小范围，我们就可以认为输入已经是标准化了，这我前面已讲过了。

```
//加入扫雷机->最近地雷之间的向量
Inputs.push_back(vClosestMine.x);
Inputs.push_back(vCIosestMine.y);
//加入扫雷机的视线向量
Inputs.push_back(m_vLookAt.x);
Inputs.push_back(m_vLookAt.y);
```

//更新大脑，并从网络得到输出

vector<double> output = m_ItsBrain.Update(inputs);

然后把视线向量，以及扫雷机与它最接近的地雷之间的向量，都输入到神经网络。函数 CNeuralNet::Update 利

用这些信息来更新扫雷机网络，并返回一个 std::vector 向量作为输出。

//保证在输出的计算中没有发生错误

if (output.size() < CParams::iNumOutputs)

{

return false;

}

// 把输出赋值到扫雷机的左、右轮轨

m_lTrack = output[0];

m_rTrack = output[1];

在更新神经网络时，当检测到确实没有错误时，程序把输出赋给 m_lTrack 和 m_rTrack。这些值代表施加

到扫雷机左、右履带轮轨上的力。

// 计算驾驶的力

double RotForce = m_lTrack - m_rTrack;

// 进行左转或右转

Clamp(RotForce, -CParams::dMaxTurnRate, CParams::dMaxTurnRate);

m_dSpeed = (m_lTrack + m_rTrack);

扫雷机车的转动力是利用施加到它左、右轮轨上的力之差来计算的。并规定，施加到左轨道上的力减去施

加到右轨道上的力，就得到扫雷机车辆的转动力。然后就把此力施加给扫雷机车，使它实行不超过 ini 文件所规

定的最大转动率的转动。而扫雷机车的行进速度不过就是它的左侧轮轨速度与它的右侧轮轨速度的和。既然我

们知道了扫雷机的转动力和速度，它的位置和偏转角度也就都能更新了。

//更新扫雷机左右转向的角度

m_dRotation += RotForce;

// 更新视线角度

m_vLookAt.x = -sin(m_dRotation);

m_vLookAt.y = cos(m_dRotation);

// 更新它的位置

m_vPosition += (m_vLookAt* m_dSpeed);

// 如果扫雷机到达窗体四周，则让它实行环绕，使它不至于离开窗体而消失

If (m_vPosition.x > CParams::WindowWidth) m_vPosition.x = 0;

If (m_vPosition.x < 0) m_vPosition.x = CParams::WindowWidth;

If (m_vPosition.y > CParams::WindowHeight) m_vPosition.y = 0;

If (m_vPosition.y < D) m_vPosition.y = CParams::WindowHeight;

为了使事情尽可能简单，我已让扫雷机在碰到窗体边框时就环绕折回(wrap)。采用这种方法程序就不再需

要做任何碰撞-响应方面的工作。环绕一块空地打转对我们人来说是一桩非常不可思议的动作，但对扫雷机，这

就像池塘中的鸭子。

Returen true;

}

4.8 CController Class（控制器类）

CController 类是和一切都有联系的类。图 15 指出了其他的各个类和 CController 类的关系。

下面就是这个类的定义:

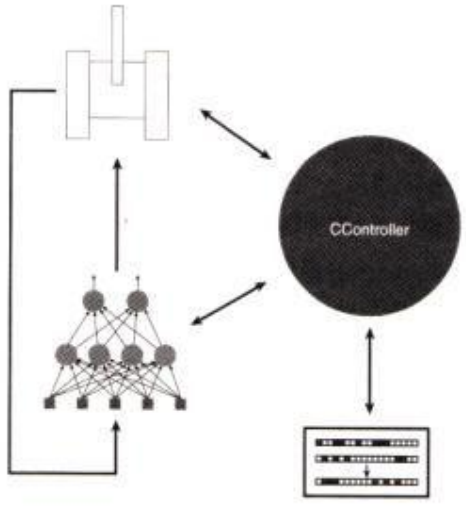
class CController

{

private:

// 基因组群体的动态存储器（一个向量）

vector<SGenome> m_vecThePopulation;



- 图 15 minesweeper 工程的程序流程图

```
// 保存扫雷机的向量
vector<CMinesweeper> m_vecSweepers;

// 保存地雷的向量
vector<SVector2D> m_vecMines;
// 指向遗传算法对象的指针

CGenAIg* m_pGA;

int m_NumSweepers;

int m_NumMines;
// 神经网络中使用的权重值的总数
int m_NumWeightsInNN;
// 存放扫雷机形状各顶点的缓冲区

vector<SPoint> m_SweeperVB;
// 存放地雷形状各顶点的缓冲区
vector<SPoint> m_MineVB;
// 存放每一代的平均适应性分数，供绘图用
vector<double> m_vecAvFitness;
```

```
// 存放每一代的最高适应性分

vector<double> m_vecBestFitness;
// 我们使用的各种不同类型的画笔
HPEN m_RedPen;
HPEN m_BluePen;
HPEN m_GreenPen;
HPEN m_OldPen;

// 应用程序窗口的句柄
HWND m_hwndMain;
// 切换扫雷机程序运行的速度
bool m_bFastRender;

// 每一代的帧数（滴答数）
int m_iTicks;

// 代的计数
int m_iGenerations;
// 窗体客户区的大小
int cxClient，cyClient;
// 本函数在运行过程中画出具有平均-，和最优适应性值的图
void PlotStats(HDC surface);

public:
CController(HWND hwndMain);
~CController();
void Render(HDC surface);
void WorldTransform(vector<SPoint> &VBuffer,
SVector2D vPos);

bool Update();
// 几个公用的访问方法
```

```
bool FastRender() { return m_bFastRender; }
void FastRender(bool arg){ m_bFastRender = arg; }
void FastRenderToggle() { m_bFastRender = !m_bFastRender; }
};
```

当创建 CController 类的某个实例时，会有一系列的事情发生：

创建 CMinesweeper 对象。

统计神经网络中所使用的权重的总数，然后此数字即被利用来初始化遗传算法类的一个实例。

从遗传算法对象中随机提取染色体(权重)并(利用细心的脑外科手术)插入到扫雷机的经网络中。

创建了大量的地雷并被随机地散播到各地。

为绘图函数创建了所有需要用到的 GDI 画笔。

为扫雷机和地雷的形状创建了顶点缓冲区。

所有的一切现都已完成初始化，由此 Update 方法就能在每一帧中被调用来对扫雷机进行演化。

4.8.1 CController::Update Method（控制器的更新方法）

控制器更新方法 CController::Update 方法（或函数）在每一帧中都要被调用。当调用 update 函数时，函数

的前一半通过对所有扫雷机进行循环，如发现某一扫雷机找到了地雷，就 update 该扫雷机的适应性分数。由于 m_vecThePopulation 包含了所有基因组的拷贝，相关的适应性分数也要在这时进行调整。如果为完成一个代（generation）所需要的帧数均已通过，本方法就执行一个遗传算法的时代(epoch)来产生新一代的权重。这些

权重被用来代替扫雷机神经网络中原有的旧的权重，使扫雷机的每一个参数被重新设置，从而为进入新一 generation 做好准备。

```
bool CController::Update()
{
// 扫雷机运行总数为 CParams::iNumTicks 次的循环。在此循环周期中，扫雷机的神经网络
// 不断利用它周围特有的环境信息进行更新。而从神经网络得到的输出，使扫雷机实现所需的
// 动作。如果扫雷机遇见了一个地雷，则它的适应性将相应地被更新，且同样地更新了它对应
// 基因组的适应性。
```

```
if (m_iTicks++ < CParams::iNumTicks)
{
for (int i=O; i<m_NumSweepers; ++i)
{
//更新神经网络和位置
if (!m_vecSweepers[i].Update(m_vecMines))

{
//处理神经网络时出现了错误，显示错误后退出
MessageBox(m_hwndMain, 'Wrong amount of NN inputs!",
"Error", MB_OK);

return false;
}

// 检查这一扫雷机是否已经发现地雷
int GrabHit = m_vecSweepers[i].CheckForMine(m_vecMines，
CParams::dMineScale);
if (GrabHit >= 0)
{
// 扫雷机已找到了地雷，所以要增加它的适应性分数

m_vecSweepers[i].IncrementFitness();
// 去掉被扫雷机找到的地雷，用在随机位置放置的一个新地雷来代替
m_vecMines[GrabHit] = SVector2D(RandFloat() * cxClient，
RandFloat() * cyClient);
}
// 更新基因组的适应性值

m-vecThePopulation[i].dFitness = m_vecSweepers[i].Fitness();
}
}
// 一个代已被完成了。
// 进入运行遗传算法并用新的神经网络更新扫雷机的时期
```

```
else

{
// 更新用在我们状态窗口中状态
m_vecAvFitness.push_back(m_pGA->AverageFitness());
m_vecBestFitness.push_back(m_pGA->BestFitness());
// 增加代计数器的值
++m_iGenerations;


// 将帧计数器复位
m_iTicks = 0;
// 运行 GA 创建一个新的群体
m-vecThePopulation = m_pGA->Epoch(m_vecThePopulation);
// 在各扫雷机中从新插入新的（有希望）被改进的大脑
// 并将它们的位置进行复位，等

for(int i=O; i<m_NumSweepers; ++i)
{m_vecSweepers[i].m_ItsBrain.PutWeights(m_vecThePopulation[i].vecWeights);
m_vecSweepers[i].Reset();
}
}
returen true;

}
```

概括起来，程序为每一世代做的工作是:

l．为所有扫雷机和为 iNumTicks 个帧组织循环，调用 Update 函数
并根据情况增加扫雷机适应值的得分。

2．从扫雷机神经网络提取权重向量。

3．用遗传算法去演化出一个新的网络权重群体。

4．把新的权重插入到扫雷机神经网络。

5．转到第 1 步进行重复，直到获得理想性能时为止。

最后，表 3 列出了 Smart Sweepers 工程 v1.0 版所有缺省参数的设置值。

表 3 Smart Sweepers v1.0 工程的缺省设置

神经网络

参 数 设 置 值

输入数目 4

输出数目 2

隐藏层数目 1

隐藏层神经元数目 10

激励响应 1

遗 传 算 法

参 数 设 置 值

群体大小 30

选择类型 旋转轮

杂交类型 单点

杂交率 0.7

突变率 0.1

精英设置（on/off） On

精英数目（N/copies） 4/1

总 体 特 性

参 数 设 置 值

每时代的帧数 2000

4.9 运行此程序 (Running the Program)

当你运行程序时，“F”键用来切换 2 种不同的显示状态，一种是显示扫雷机怎样学习寻找地雷，一种是

示在运行期中产生的最优的与平均的适当性分数的统计图表。 当显示图表时，程序将会加速运行。

游戏编程中的人工智能技术

.

<神经网络入门>

.

(连载之六)

4.10 功能的两个改进 (A Couple of Improvements)

尽管扫雷机学习寻找地雷的本领十分不错，这里我仍有两件事情要告诉你，它们能进一步改进扫雷机的性能。

4.10.1 改进一（Improvement Number One）

首先，单点 crossover 算子留下了许多可改进的余地。按照它的规定，算子是沿着基因组长度任意地方切开的，这样常有可能使个别神经细胞的基因组在权重的中间被一刀两段地分开。

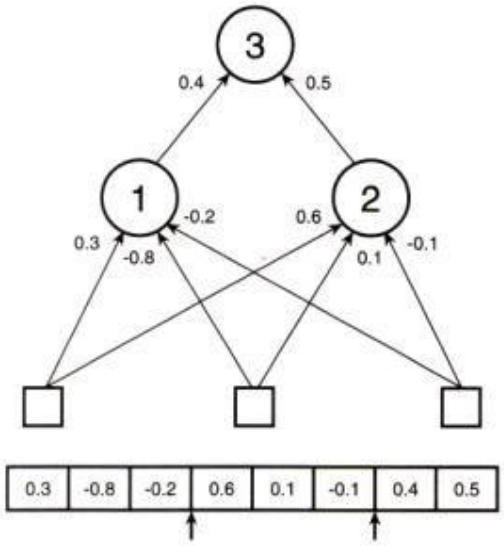
为清楚起见，我们来考察图 16 的权重。这是我们以前在说明基因组如何编码时看过的一个简单网络。 在这

里，杂交算子可以沿向量长度的任意一处切开，这样，就会有极大几率在某个神经细胞（如第二个）的权重中

间断开，也就是说，在权重 0.6 和-0.1 之间某处切开。这可能不会是我们想要的，因为，如果我们把神经细胞作

为一个完整的单元来看待，则它在此以前所获得的任何改良就要被骚扰了。事实上，这样的杂交操作有可能非

常非常象断裂性突变（disruptive mutation）操作所起的作用



- 图 16 简单的网络

与此针锋相对，我已创建了另一种类型的杂交运算，它只在神经细胞的边界上进行切开。在图 16 的例子中，

就是在第 3、4 或第 6、7 的两个基因之间切开，如小箭头所示。 为了实现这一算法，我已在 CNeuralNet 类中补

充了另一个切割方法: CalculateSplitPoints。这一方法创建了一个用于保存所有网络权重边界的矢量，它的代

码如下:

```
vector<int> CNeuralNet::CalculateSplitPoints() const
{

vector<int> SplitPoints;

int WeightCounter = 0;

// 对每一层
for (int i=O; i<m_NumHiddenLayers + 1; ++i)
{
// 对每一个神经细胞

for (int j=O; j<m_vecLayers[i].m_NumNeurons; ++j)
    {
```

```
// 对每一个权重
for (int k=O; k<m_vecLayers[i].m_vecNeurons[j].m_NumInputs; ++k)
{
++WeightCounter;

}
SplitPoints.push_back(WeightCounter - 1);
}
}
return SplitPoints;
}
```

这一方法是 CController 类构造函数在创建扫雷机并把断裂点向量传递给遗传算法类时调用的。它们被存储

在一个名叫 m_vecSplitPoints 的 std::vector 向量中。然后遗传算法就利用这些断裂点来实现两点杂交操作，其代

码如下:

```
void CGenAlg::CrossoverAtSplits(const vector<double> &mum,
const vector<double> &dad,
vector<double> &babyl,

vector<double> &baby2)
{
// 如果超过了杂交率，就不再进行杂交，把 2 个上代作为 2 个子代输出
// 如果 2 个上辈相同，也把它们作为 2 个下辈输出
if ( (RandFloat() > m_dCrossoverRate) || (mum == dad))
{

baby1 = mum;
baby2 = dad;
return;
}
// 确定杂交的 2 个断裂点
int index1 = RandInt(0, m_vecSplitPoints.size()-2);
```

```
int index2 = RandInt(Index1, m_vecSplitPoints.size()-1);
int cp1 = m_vecSplitPoints[Index1];
int cp2 = m_vecSplitPoints[Index2];
// 创建子代

for (int i=0; i<mum.size(); ++i)
{

if ( (i<cp1) || (i>=cp2) )
{
// 如果在杂交点外，保持原来的基因
babyl.push_back(mum[i]);
baby2.push_back(dad[i]);
}

else
{
// 把中间段进行交换
baby1.push_back(dad[1]);
baby2.push_back(mum[1]);
}

}
return;
}
```

根据我的经验，我已发现，在进行杂交时，把神经细胞当作一个不可分割的单位，比在染色体长度上任意

一点分裂基因组，能得到更好的结果。

4.10.2 改进二（Improvement Number Two）

我想和你讨论的另一个性能改进，是用另一种方式来观察网络的那些输入。在你已看到的例子中，我们为

网络使用了 4 个输入参数: 2 个用于表示扫雷机视线方向的向量,另外 2 个用来指示扫雷机与其最靠近的地雷的方

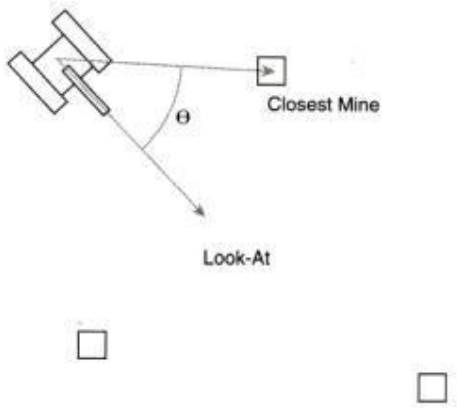
向的向量。然而，有一种办法，可以把这些参数的个数减少到只剩下一个。

其实你想一想就可知道，扫雷机为了确定地雷的位置，只要知道从它当前的位置和朝向出发，需要向左或

向右转动多大的一个角度这一简单的信息就够了(如果你已经考虑到了这一点，那我在这里要顺便向您道贺了)。

由于我们已经计算了扫雷机的视线向量和从它到最邻近地雷的向量，再来计算它们之间的角度(θ)应是一件极为

简单的事情 – 这就是这两个向量的点积，这我们在第 6 章“使登陆月球容易一点”中已讨论过。见图 17。



- 图 17 计算到最邻近地雷的转动角度。

不幸的是，点积仅仅给出角度的大小; 它不能指示这一角度是在扫雷机的那一侧。因此，我已写了另一个向

量函数返回一个向量相对于另一个向量的正负号。该函数的原型如下所示:

```
inline int Vec2DSign(SVector2D &v1,SVector2D &v2);
```

如果你对它的机理感兴趣，你可以在文件 SVector2D.h 中找到它的源码。但它的基本点就是: 如果 v1 至 v2 是

按顺时针方向转的，则函数返回 +1；如果 v1 至 v2 是按逆时针方向转，则函数返回 -1。

把点积和 Vec2Dsign 二者联合起来，就能把输入的精华提纯出来，使网络只需接受一个输入就行了。下面

就是新的 CMinesweeper::Update 函数有关段落的代码形式:

```
// 计算到最邻近地雷的向量
```

```
SVector2D vClosestMine = GetClosestMine(mines);


// 将它规范化
  Vec2DNormalize(vClosestMine);


   // 计算扫雷机视线向量和它到最邻近地雷的向量的点积。它给出了我们要面对
   // 最邻近地雷所需转动的角度
double dot = Vec2DDot(m_vLookAt, vClosestMine);
// 计算正负号


int sign = Vec2DSign(m_vLookAt, vClosestMine);
Inputs.push_back(dot*sign);
```

运行一下光盘 Chapter7/Smart Sweepers v1.1 目录下的可执行程序 executable，你就知道经过以上 2 个改

进，能为演化过程提速多少。

需要注意的一桩重要事情是，带有 4 个输入的网络要花很长时间进行演化，因为它必须在各输入数据之间找

出更多的关系才能确定它应如何行动。事实上，网络实际就是在学习怎么做点积并确定它的正负极性。因此，当

你设计自己的网络时，你应仔细权衡一下，是由你自己预先来计算许多输入数据好呢(它将使 CPU 负担增加，但

导致进化时间加快)还是让网络来找输入数据之间的复杂关系好(它将使演化时间变长，但能使 CPU 减少紧张)？

5 结束语（last words）

我希望你已享受到了你第一次攻入神经网络这一奇妙世界的快乐。我打赌你一定在为如此简单就能使用它

们而感到惊讶吧，对吗？我想我是猜对了。

在下面几章里我将要向你介绍更多的知识，告诉你一些新的训练手段和演绎神经网络结构的更多的方法。

但首先请你利用本章下面的提示去玩一下游戏是有意义的。

6 练习题 (Stuff to Try)

1。 在 v1.0 中，不用 look-at 向量作为输入，而改用旋转角度 θ 作为输入，由此就可以使网络的输入个数减少

成为 1 个。请问这对神经网络的演化有什么影响？你对此的看法怎样？

2。 试以扫雷机的位置（x1,y1）、和扫雷机最接近的地雷的位置（x2,y2）、以及扫雷机前进方向的向量

（x3,y3）等 6 个参数作为输入，来设计一个神经网络，使它仍然能够演化去寻找地雷。

3。 改变激励函数的响应。试用 O.1 - O.3 之间的低端值，它将产生和阶跃函数非常相像的一种激励函数。

然后再试用高端值，它将给出较为平坦的响应曲线。考察这些改变对演化进程具有什么影响？

4。 改变神经网络的适应性函数，使得扫雷机不是去扫除地雷，而是要演化它，使它能避开地雷。

5。 理一理清楚有关遗传算法的各种不同设置和运算中使你感到模糊的东西！

6。 加入其他的对象类型，比如人。给出一个新环境来演化扫雷机，使它能避开人，但照样能扫除地雷。

（这可能没有你想象那么容易！)