[鱼书] 第2章 感知机

感知机是由美国学者Frank Rosenblatt在1957年提出来的。为何我们现在还要学习这一很久以前就有的算法呢？因为感知机也是作为神经网络（深度学习）的起源的算法。

感知机是什么

感知机**接收多个输入信号，输出一个信号**。和实际的电流不同的是，感知机的信号只有“流/不流”（1/0）两种取值。在本书中，0对应“不传递信号”，1对应“传递信号”。

图2-1是一个接收两个输入信号的感知机的例子。x1、x2是输入信号，y是输出信号，w1、w2是权重（w是weight的首字母）。图中的○称为“神经元”或者“节点”。输入信号被送往神经元时，会被分别乘以固定的权重（w1x1、w2x2）。神经元会计算传送过来的信号的总和，只有当这个总和超过了某个界限值时，才会输出1。这也称为“神经元被激活”。这里将这个界限值称为**阈值**，用符号θ表示。

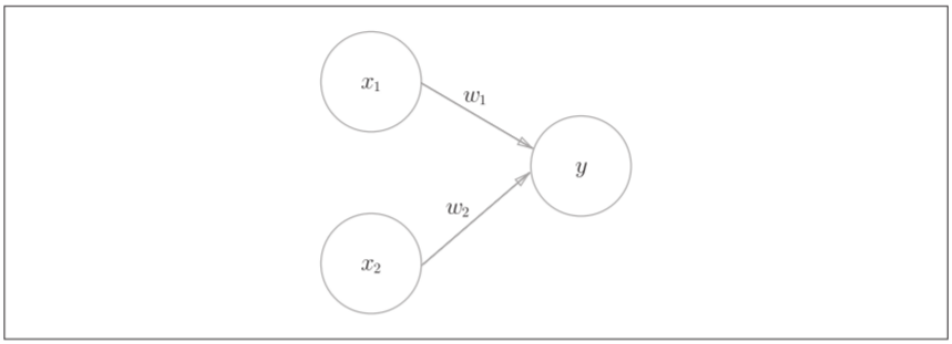

图2-1　有两个输入的感知机

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leqslant \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

感知机的多个输入信号都有各自固有的权重，这些权重发挥着控制各个信号的重要性的作用。也就是说，权重越大，对应该权重的信号的重要性就越高。

简单逻辑电路

与门

与门（AND gate）：与门是有两个输入和一个输出的门电路。

图2-2这种输入信号和输出信号的对应表称为“真值表”。

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

图2-2　与门的真值表

用感知机来表示这个与门，需要做的就是确定能满足图2-2的真值表的w1、w2、θ的值。实际上，满足图2-2的条件的参数的选择方法有无数多个。比如，当(w1,w2, θ) = (0.5, 0.5, 0.7)时，设定这样的参数后，仅当x1和x2同时为1时，信号的加权总和才会超过给定的阈值θ。

与非门和或门

NAND是Not AND的意思，与非门就是颠倒了与门的输出。仅当x1和x2同时为1时输出0，其他时候则输出1。

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

图2-3　与非门的真值表

要表示与非门，可以用(w1,w2, θ) = (－0.5, －0.5, －0.7)这样的组合（其他的组合也是无限存在的）。实际上，只要把实现与门的参数值的符号取反，就可以实现与非门。

或门是“只要有一个输入信号是1，输出就为1”的逻辑电路。

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

图2-4　或门的真值表

这里决定感知机参数的并不是计算机，而是我们人。我们看着真值表这种“训练数据”，人工考虑（想到）了参数的值。而机器学习的课题就是将这个决定参数值的工作交由计算机自动进行。学习是确定合适的参数的过程，而人要做的是思考感知机的构造（模型），并把训练数据交给计算机。

实际上，3个门电路**只有参数的值（权重和阈值）不同**。也就是说，相同构造的感知机，只需通过适当地调整参数的值，就可以像“变色龙演员”表演不同的角色一样，变身为与门、与非门、或门。

感知机的实现

简单实现

```
def AND(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    y = w1 * x1 + w2 * x2
    if y <= theta:
        return 0
    else tmp> theta:
        return 1
```

导入权重和偏置

把式（2.1）的θ换成－b，于是就可以用式（2.2）来表示感知机的行为。

$$y = \begin{cases} 0 & (b + w_1 x_1 + w_2 x_2 \leqslant 0) \\ 1 & (b + w_1 x_1 + w_2 x_2 > 0) \end{cases}$$

此处，**b称为偏置，w1和w2称为权重**。

```
>>> import numpy as np
>>> x = np.array([0, 1])      # 输入
>>> w = np.array([0.5, 0.5]) # 权重
>>> b = -0.7                  # 偏置
>>> w*x
array([ 0. ,  0.5])
>>> np.sum(w*x)
0.5
>>> np.sum(w*x) + b
-0.19999999999999996   # 大约为-0.2(由浮点小数造成的运算误差)
```

```
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

偏置和权重w1、w2的作用是不一样的。具体地说，w1和w2是控制输入信号的**重要性**的参数，而偏置是调整神经元**被激活的容易程度**（输出信号为1的程度）的参数。但是根据上下文，有时也会将b、w1、w2这些参数统称为权重。

```
def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5]) # 仅权重和偏置与AND不同!
    b = 0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

感知机的局限性

异或门

异或门也被称为**逻辑异或**电路。仅当x1或x2中的一方为1时，才会输出1（“异或”是拒绝其他的意思）。

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

图2-5　异或门的真值表

用前面介绍的感知机是**无法实现**这个异或门的。

我们试着将或门的动作形象化。或门的情况下，当权重参数(b, w1,w2) = (－0.5, 1.0, 1.0)时，可满足图2-4的真值表条件。

$$y = \begin{cases} 0 & (-0.5 + x_1 + x_2 \leqslant 0) \\ 1 & (-0.5 + x_1 + x_2 > 0) \end{cases}$$

式（2.3）表示的感知机会生成由直线－0.5 + x1*x2 = 0分割开的两个空间。其中一个空间输出1，另一个空间输出0，如图2-6所示。

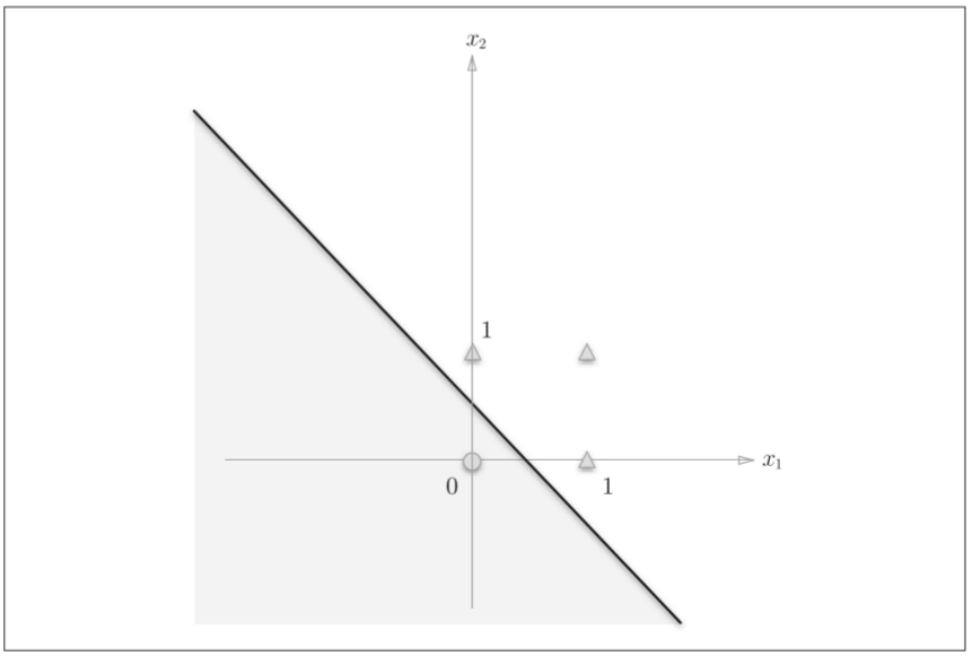

图2-6　感知机的可视化：灰色区域是感知机输出0的区域，这个区域与或门的性质一致

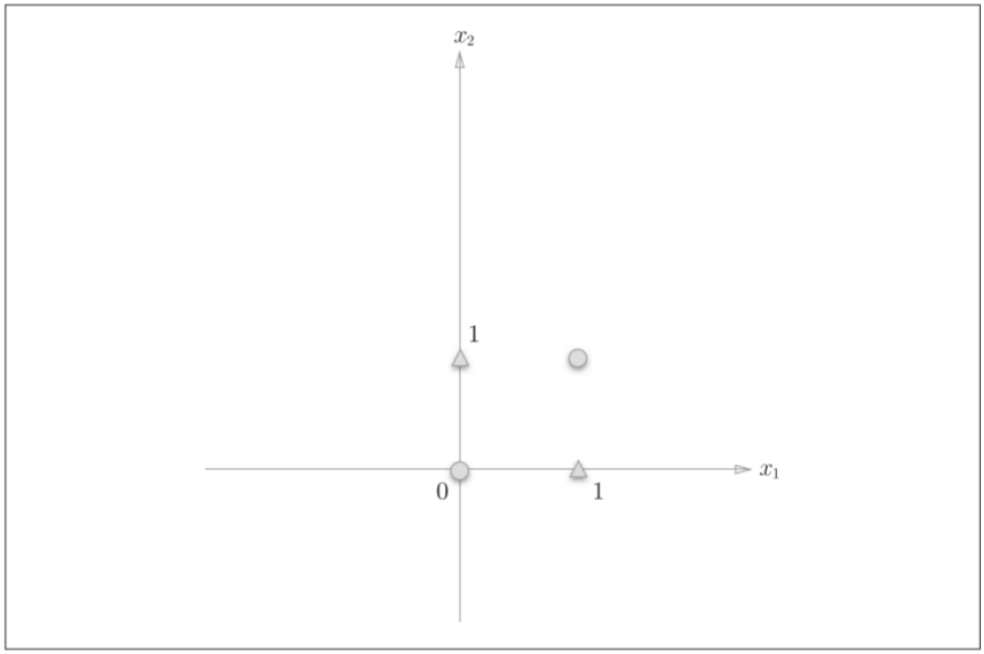

图2-7　○和△表示异或门的输出。可否通过一条直线作出分割○和△的空间呢?

想要用一条直线将图2-7中的○和△分开，无论如何都做不到。事实上，用一条直线是无法将○和△分开的。

线性与非线性

图2-7中的○和△无法用一条直线分开，但是如果将“**直线**”这个限制条件去掉，就可以实现了。

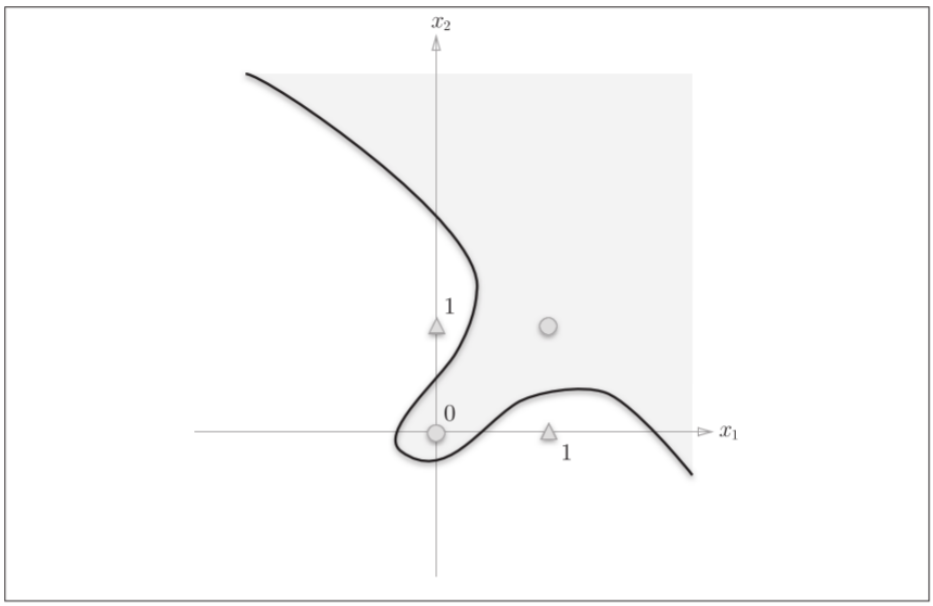

图2-8　使用曲线可以分开○和△

感知机的局限性就在于它只能表示由一条**直线**分割的空间。图2-8这样弯曲的曲线无法用感知机表示。另外，由图2-8这样的曲线分割而成的空间称为**非线性空间**，由直线分割而成的空间称为**线性空间**。

感知机的局限性，严格地讲，应该是“单层感知机无法表示异或门”或者“**单层感知机**无法分离非线性空间”

多层感知机

感知机的绝妙之处在于它可以“叠加层”（通过叠加层来表示异或门是本节的要点）。

已有电路门的组合

异或门的制作方法有很多，其中之一就是组合我们前面做好的与门、与非门、或门进行配置。图2-9中与非门前端的○表示反转输出的意思。

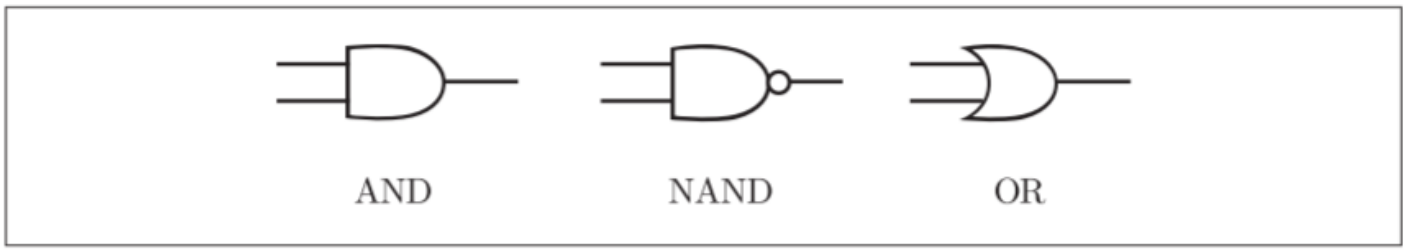

图2-9　与门、与非门、或门的符号

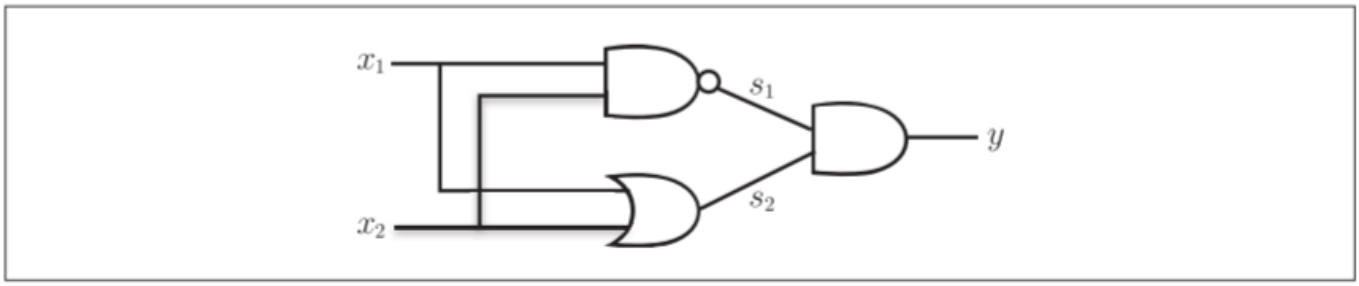

图2-11　通过组合与门、与非门、或门实现异或门

异或门的实现

```
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y
```

异或门是一种**多层结构的神经网络**。这里，将最左边的一列称为第0层，中间的一列称为第1层，最右边的一列称为第2层。

叠加了多层的感知机也称为多层感知机（multi-layered perceptron）。

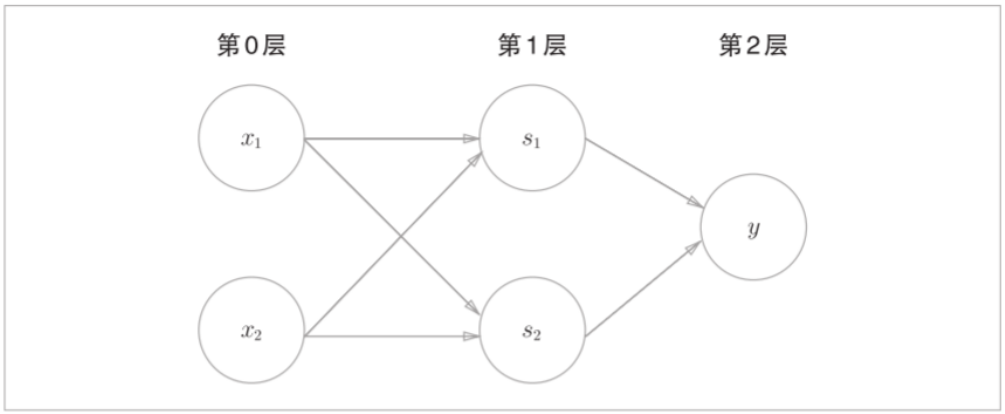

图2-13　用感知机表示异或门

图2-13中的感知机总共由3层构成，但是**因为拥有权重的层实质上只有2层（第0层和第1层之间，第1层和第2层之间）**，所以称为“**2层感知机**”。不过，有的文献认为图2-13的感知机是由3层构成的，因而将其称为“3层感知机”。

在图2-13所示的2层感知机中，先在第0层和第1层的神经元之间进行信号的传送和接收，然后在第1层和第2层之间进行信号的传送和接收。

从与非门到计算机

人们一般会认为计算机内部进行的处理非常复杂，而令人惊讶的是，实际上只需要通过**与非门的组合，就能再现计算机进行的处理**。也就是说，如果通过组合与非门可以实现计算机的话，那么通过组合感知机也可以表示计算机（感知机的组合可以通过叠加了多层的单层感知机来表示）。

《计算机系统要素：从零开始构建现代计算机》这本书以深入理解计算机为主题，论述了通过NAND构建可运行俄罗斯方块的计算机的过程。此书能让读者真实体会到，通过简单的NAND元件就可以实现计算机这样复杂的系统。

理论上可以说2层感知机就能构建计算机。这是因为，已有研究证明，2层感知机（严格地说是**激活函数使用了非线性的sigmoid函数的感知机**，具体请参照下一章）可以表示**任意函数**。但是，使用2层感知机的构造，通过设定合适的权重来构建计算机是一件非常累人的事情。

实际上，在用与非门等低层的元件构建计算机的情况下，分阶段地制作所需的零件（模块）会比较自然，即先实现与门和或门，然后实现半加器和全加器，接着实现算数逻辑单元（ALU），然后实现CPU。因此，通过感知机表示计算机时，使用叠加了多层的构造来实现是比较自然的流程。

感知机通过叠加层能够进行**非线性的表示**，理论上还可以表示计算机进行的处理。