深度学习入门

基于Python的理论与实现

Stephen CUI

March 14, 2023

Chapter 1

感知机

本章将介绍感知机（perceptron）这一算法。感知机作为神经网络（深度学习）的起源的算法。因此，学习感知机的构造是学习通向神经网络和深度学习的一种重要思想。

1.1 感知机是什么

感知机接收多个输入信号，输出一个信号。像电流流过导线，向前方输送电子一样，感知机的信号也会形成流，向前方输送信息。但是，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”。

假设一个接收两个输入信号的感知机的例子。x_1、x_2是输入信号， y 是输出信号，w_1、w_2 是权重（w 是 weight 的首字母）。输入信号被送往神经元时，会被分别乘以固定的权重(w_1x_1, w_2x_2)，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。

感知机的运行原理只有这些！把上述内容用数学式来表示：

$$y = \begin{cases} 0, & w_1x_1 + w_2x_2 \leq \theta \\ 1, & w_1x_1 + w_2x_2 > \theta \end{cases} \tag{1.1}$$

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。

1.2 简单逻辑电路

1.2.1 与门

与门（AND gate）是有两个输入和一个输出的门电路。与门仅在两个输入均为1时输出1，其他时候则输出0。

1.2.2 与非门和或门

与非门（NAND gate）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。

或门是“只要有一个输入信号是1，输出就为1”的逻辑电路。

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。

与门、与非门、或门的感知机构造是一样的。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，就可以不断变换为与门、与非门、或门。

1.3 感知机的实现

1.3.1 简单的实现

1.3.2 导入权重和偏置

首先把 Equation 1.1 的θ换成$-b$，就可以用 Equation 1.2 来表示感知机的行为。

$$y = \begin{cases} 0, & b + w_1 x_1 + w_2 x_2 \leq 0 \\ 1, & b + w_1 x_1 + w_2 x_2 > 0 \end{cases} \tag{1.2}$$

这里，b称为偏置，w_1和w_2称为权重。感知机会计算输入信号和权重的乘积，然后加上偏置，如果这个值大于0则输出1，否则输出0。请注意，偏置和权重w_1、w_2的作用是不一样的。具体地说，w_1和w_2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。

1.4 感知机的局限性

1.4.1 异或门

异或门也被称为逻辑异或电路。仅当x_1或x_2中的一方为 1时，才会输出1（“异或”是拒绝其他的意思）。实际上，用前面介绍的感知机是无法实现这个异或门的。

可以考虑在二维平面上添加一个直线，但是没有一个直线能够满足将$(0,0),(0,1),(1,0),(1,1)$实现异或门的切分。

1.4.2 线性和非线性

显然，我们可以使用抛物线这种非线性的曲线来实现异或门，这就是用非线性的方式来实现的。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。由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。

1.5 多层感知机

感知机不能表示异或门让人深感遗憾，但也无需悲观。实际上，感知机的绝妙之处在于它可以“叠加层”，这样可以实现异或门。

1.5.1 已有门电路的组合

> 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”。接下来，我们将看到通过组合感知机（叠加层）就可以实现异或门。

异或门可以通过与门、与非门、或门组成的两层感知机来实现，首先是第一层：使用非门和与非门来作为输入，然后第二层使用非门和与非门的输出作为与门的输入，即可实现异或门。

1.5.2 异或门的实现

与门、或门是单层感知机，而异或门是2层感知机。叠加了多层的感知机也称为多层感知机（multi-layered perceptron, MLP）。

1.6 从与非门到计算机

计算机是处理信息的机器。向计算机中输入一些信息后，它会按照某种既定的方法进行处理，然后输出结果。所谓“按照某种既定的方法进行处理” 是指，计算机和感知机一样，也有输入和输出，会按照某个既定的规则进行计算。

Chapter 2

神经网络

关于感知机，既有好消息，也有坏消息。好消息是，即便对于复杂的函数，感知机也隐含着能够表示它的可能性。上一章已经介绍过，即便是计算机进行的复杂处理，感知机（理论上）也可以将其表示出来。坏消息是，设定权重的工作，即确定合适的、能符合预期的输入与输出的权重，现在还是由人工进行的。

2.1 从感知机到神经网络

2.1.1 神经网络的例子

用图来表示神经网络的话，如Figure 2.1所示。我们把最左边的一列称为输入层，最右边的一列称为输出层，中间的一列称为中间层。中间层有时也称为隐藏层。“隐藏”一词的意思是，隐藏层的神经元（和输入层、输出层不同）肉眼看不见。另外，本书中把输入层到输出层依次称为第 0层、第 1 层、第2 层。

2.1.2 复习感知机

引入新函数$h(x)$，将Equation 1.2改写成下面的方程：

$$y = h(b + w_1 x_1 + w_2 x_2) \tag{2.1}$$

$$h(x) = \begin{cases} 0 & ,x \leq 0 \\ 1 & ,x > 0 \end{cases} \tag{2.2}$$

2.1.3 激活函数登场

Equation 2.2中的$h(x)$函数会将输入信号的总和转换为输出信号，这种函数一般称为激活函数（activation function）。

> 本书在使用“感知机”一词时，没有严格统一它所指的算法。一般而言，“朴素感知机”是指单层网络，指的是激活函数使用了阶跃函数A 的模型。“多层感知机”是指神经网络，即使用sigmoid 函数等平滑的激活函数的多层网络。

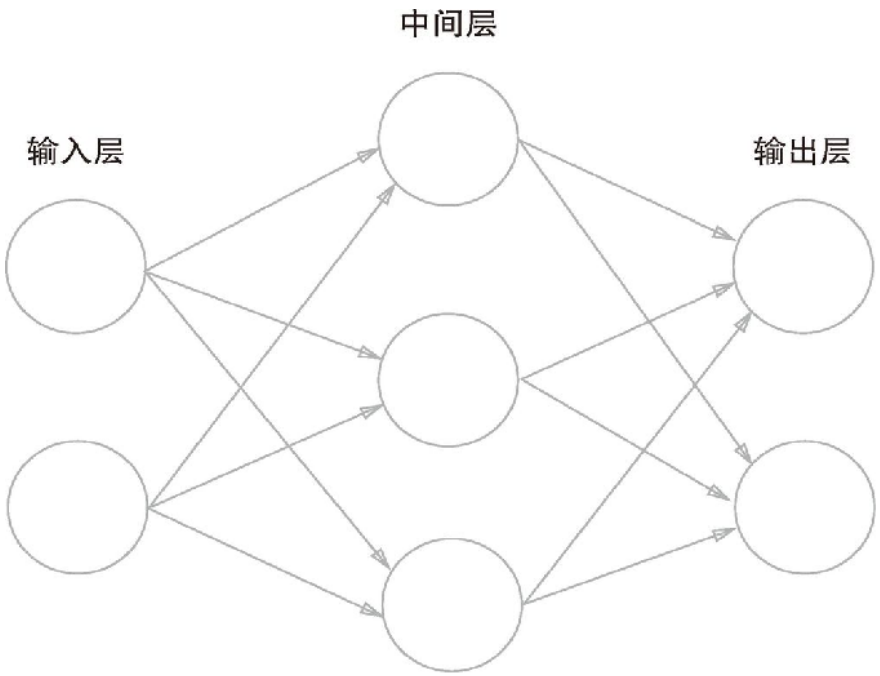

Figure 2.1: Examples of Neural Networks

2.2 激活函数

Equation 2.2表示的激活函数以阈值为界，一旦输入超过阈值，就切换输出。这样的函数称为“阶跃函数[1]”。因此，可以说感知机中使用了阶跃函数作为激活函数。也就是说，在激活函数的众多候选函数中，感知机使用了阶跃函数。

2.2.1 sigmoid函数

神经网络中经常使用的一个激活函数是sigmoid函数（sigmoid function）：

$$h(x) = \frac{1}{1+\exp(-x)} \tag{2.3}$$

2.2.2 sigmoid函数和阶跃函数的比较

首先注意到的是“平滑性”的不同。sigmoid函数是一条平滑的曲线，输出随着输入发生连续性的变化。而阶跃函数以0为界，输出发生急剧性的变化。sigmoid函数的平滑性对神经网络的学习具有重要意义。

另一个不同点是，相对于阶跃函数只能返回0或1，sigmoid函数可以返回0.731 . . .、0.880 . . .等实数（这一点和刚才的平滑性有关）。也就是说，感知机中神经元之间流动的是0或1的二元信号，而神经网络中流动的是连续的实数值信号。

接着说一下阶跃函数和sigmoid函数的共同性质。阶跃函数和sigmoid 函数虽然在平滑性上有差异，可以发现它们具有相似的形状。实际上，两者的结构均是“输入小时，输出接近0（为0）；随着输入增大，输出向1靠近（变成1）”。也就是说，当输入信号为重要信息时，阶跃函数和sigmoid函数都会输出较大的值；当输入信号为不重要的信息时，两者都输出较小的值。还有一个共同点是，不管输入信号有多小，或者有多大，输出信号的值都在0到1之间。

[1]阶跃函数是指一旦输入超过阈值，就切换输出的函数。应该有更严格的数学定义

2.2.3 非线性函数

神经网络的激活函数必须使用非线性函数。换句话说，激活函数不能使用线性函数。为什么不能使用线性函数呢？因为使用线性函数的话，加深神经网络的层数就没有意义了。

线性函数的问题在于，不管如何加深层数，总是存在与之等效的“无隐藏层的神经网络”。为了具体地（稍微直观地）理解这一点，我们来思考下面这个简单的例子。这里我们考虑把线性函数 h(x) = cx 作为激活函数，把 $y(x) = h(h(h(x)))$ 的运算对应 3 层神经网络[2]。这个运算会进行 $y(x) = c \times c \times c \times x$的乘法运算，但是同样的处理可以由$y(x) = ax$（注意， $a = c^3$）这一次乘法运算（即没有隐藏层的神经网络）来表示。如本例所示，使用线性函数时，无法发挥多层网络带来的优势。因此，为了发挥叠加层所带来的优势，激活函数必须使用非线性函数。

2.2.4 ReLU函数

在神经网络发展的历史上，sigmoid函数很早就开始被使用了，而最近则主要使用ReLU（Rectified Linear Unit）函数。ReLU函数可以表示为:

$$h(x) = \begin{cases} x & ,x > 0 \\ 0 & ,x \leq 0 \end{cases} \tag{2.4}$$

2.3 多维数组的运算

`np.dot()`接收两个NumPy数组作为参数，并返回数组的乘积。

2.4 输出层的设计

神经网络可以用在分类问题和回归问题上，不过需要根据情况改变输出层的激活函数。一般而言，回归问题用恒等函数，分类问题用softmax函数。

2.4.1 恒等函数和softmax函数

恒等函数会将输入按原样输出，对于输入的信息，不加以任何改动地直接输出。因此，在输出层使用恒等函数时，输入信号会原封不动地被输出。和前面介绍的隐藏层的激活函数一样，恒等函数进行的转换处理可以用一根箭头来表示。

分类问题中使用的softmax函数表示为：

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} \tag{2.5}$$

softmax函数的分子是输入信号a_k的指数函数，分母是所有输入信号的指数函数的和。

[2]该对应只是一个近似，实际的神经网络运算比这个例子要复杂，但不影响后面的结论成立。