第5章 神经网络

5.1 神经元模型

神经元是构成神经网络最基本的部分，“M-P神经元模型”一直沿用至今，如下图5.1。在这个模型中，神经元接收到来自n个其他神经元传递过来的输入信号，这些输入信号通过带权重的连接进行传递，神经元接收到的总输入值将与神经元的阈值进行比较，然后通过 **激活函数** 处理后输出

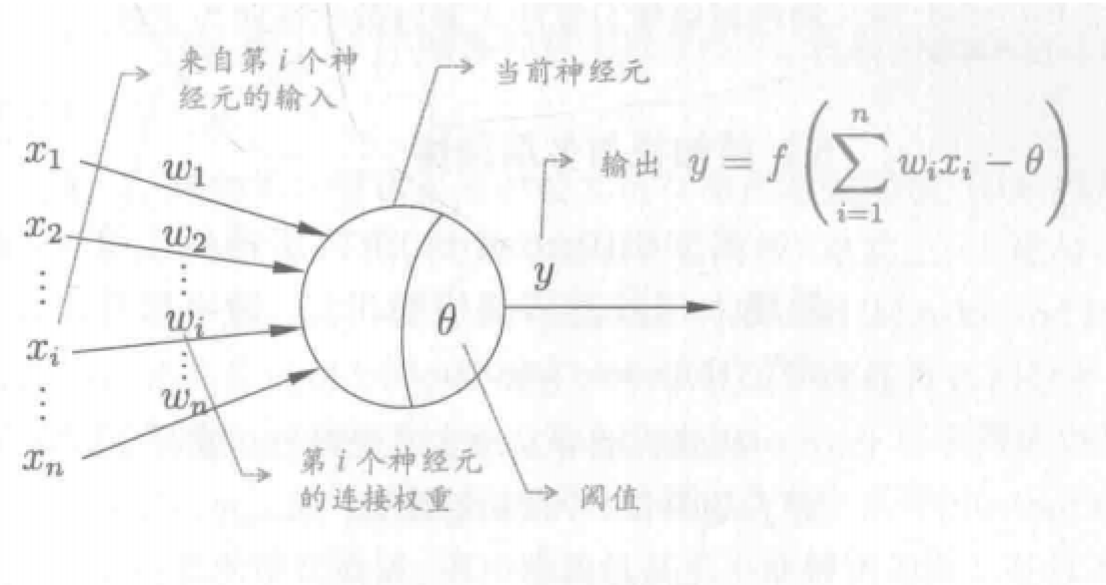


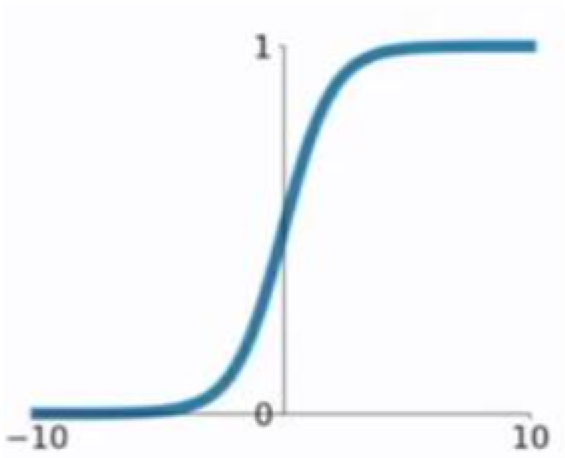
图 5.1 M-P 神经元模型

5.2 激活函数

深度学习中，常用的激活函数主要有：**sigmoid函数、tanh函数、ReLU函数**

1. sigmoid函数

$$g(z) = \frac{1}{1+e^{-z}} \tag{1}$$



对于sigmoid函数的求导为

$$
\begin{aligned}
g\prime(z) &= \left(\frac{1}{1+e^{-z}}\right)' \\
&= g(z)(1-g(z))
\end{aligned} \tag{2}
$$

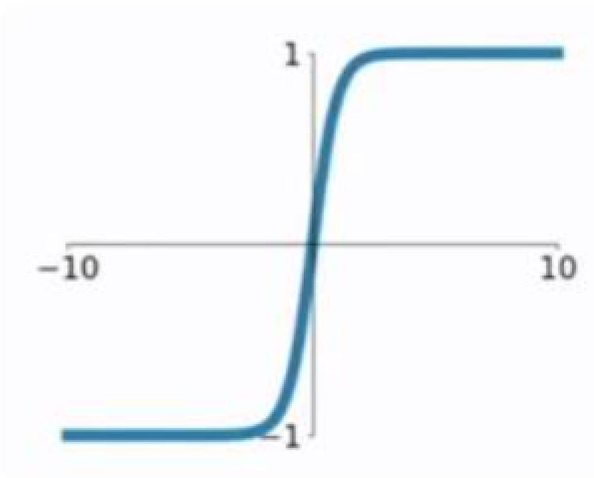
sigmoid函数作为非线性激活函数，其实并不常用，它有以下几个缺点：

- 当z值非常大或者非常小时，sigmoid函数的导数值将接近0。这会导致权重W的梯度将接近0，**使得梯度更新很慢**，即**梯度消失。**
- **函数的输出不是以0作为均值**，不便于下层的计算。sigmoid函数可用在网络最后一层，作为输出层进行二分类。

2. tanh函数

tanh函数相较于sigmoid函数要常见一些，该函数将取值$(-\infty,+\infty)$的数映射到$(-1,1)$之间，其公式与图形为：

$$
g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3}
$$



tanh函数在0附近很短一段区域内可看作线性的。由于tanh函数的均值为0，因此弥补了sigmoid函数均值为0.5的缺点，对于tahn函数的求导为：

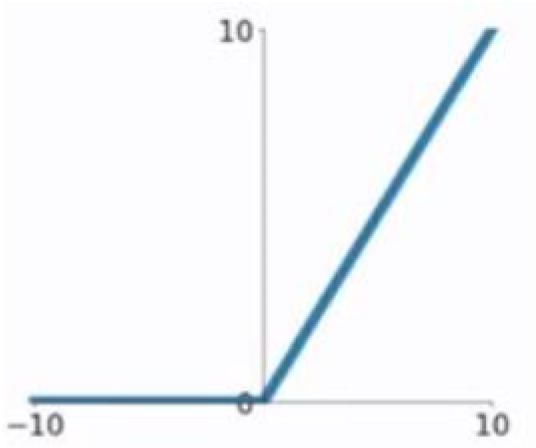
$$
\begin{aligned}
g\prime(z) &= \left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right)' \\
&= 1 - g(z)^2
\end{aligned} \tag{4}
$$

tanh函数的缺点同sigmoid函数的第一个缺点一样，当z **很大或很小**时，会导致梯度很小，权重更新非常缓慢，即梯度消失问题。

3. ReLU函数

ReLU函数又称**修正线性单元**，是一种分段线性函数，其弥补了sigmoid函数以及tanh函数的梯度消失问题。ReLU函数的公式以及图形如下：

$$g(z) = \begin{cases} 0 & (z < 0) \\ z & (z \geq 0) \end{cases} \tag{5}$$
$$or$$
$$g(z) = \max\{0, z\}$$



对ReLU函数的求导可得：

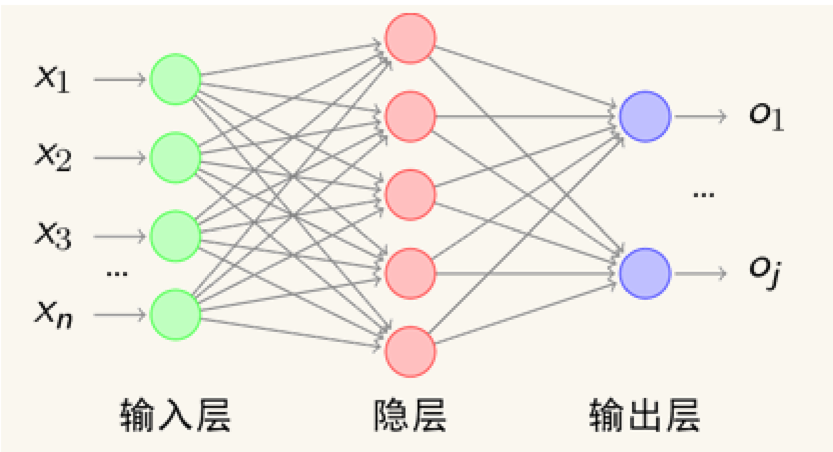
$$g\prime(z) = \begin{cases} 0 & (z < 0) \\ 1 & (z \geq 0) \end{cases} \tag{6}$$

ReLU函数的优点有以下几点

- 当z大于0时，不会出现梯度消失的情况
- 计算速度要快很多

5.3 BP算法

激活函数如果采用sigmoid函数，搭建一个多层神经网络，为简单起见，使用三层网络来分析



首先定义损失函数（0.5方便后面求导后系数为1），由于网络的输出层有多个输出结点，我们需要将输出层每个输出结点的差值平方求和。于是得到每一个训练样例的损失函数为：

$$E = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2 \tag{7}$$

算法如下：

1. 将网络中的所有权值随机初始化
2. 对每一个训练样例，执行如下操作：
    - 根据实例的输入，从前向后依次计算，得到输出层每个单元的输出。然后从输出层开始反向计算每一层的每个单元的误差项
    - 对于输出层的每个单元k，计算它的误差项：

$$\delta_k = o_k(1-o_k)(t_k-o_k)$$

    - 对于网络中每个隐层单元h，计算它的误差项：

$$\delta_h = o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_h$$

    - 更新每个权值：

$$w_{ji} = w_{ji} + \eta\delta_j x_{ji}$$

$\Delta w_{ji} = \eta\delta_j x_{ji}$ 被称为权值更新法则