神经网络：神经元、隐层和神经网络

“人话教育出版社”

yoc - 2023.01.16

1：神经元（Neuron）

何为神经元?

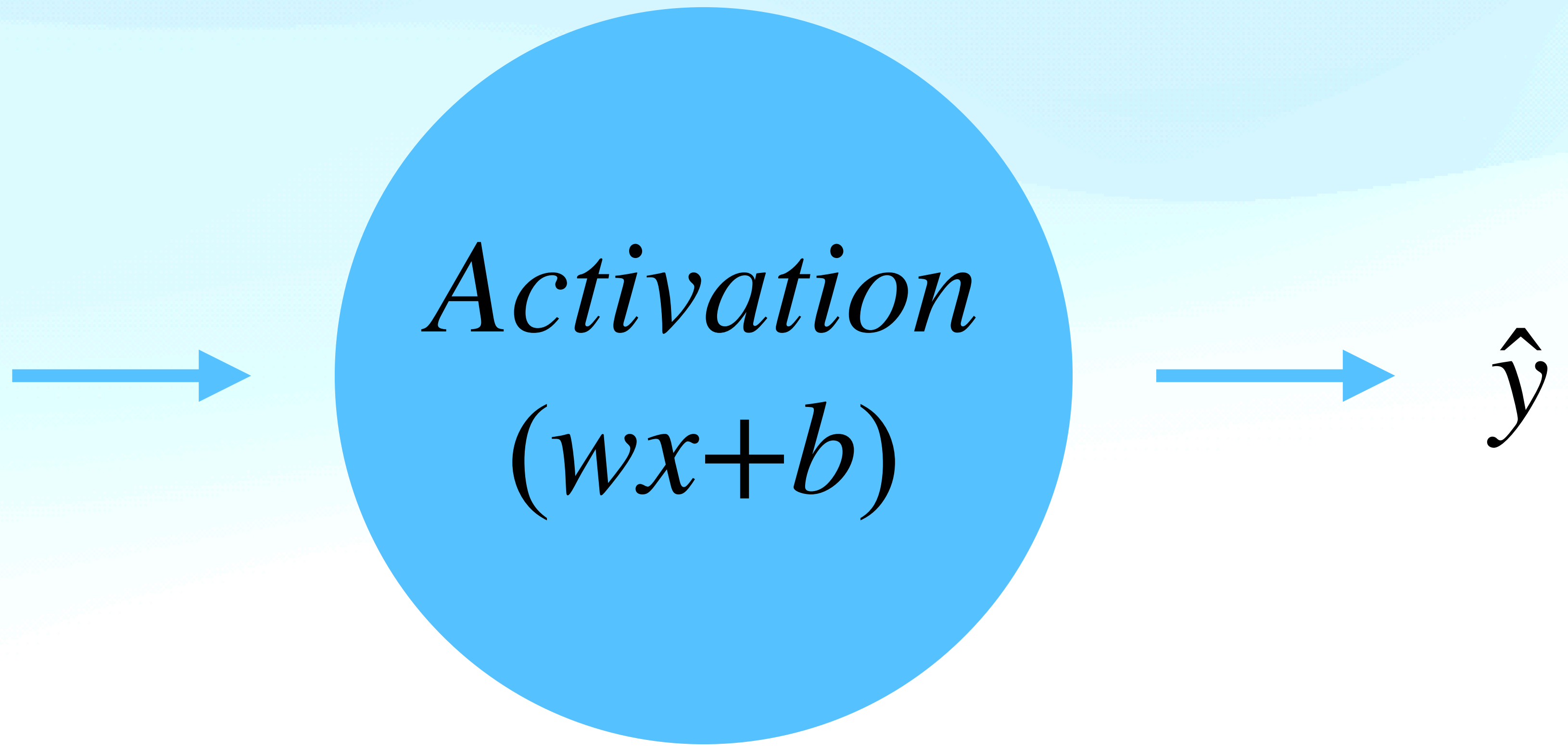
答曰：一元一次函数。

输入：x $\xrightarrow[w]{\text{权重}}$ 偏置 b $\xrightarrow[Activation]{\text{激活函数}}$ 输出：$\hat{y}$

$$\hat{y} = f(x) = Activation(wx + b)$$

何为神经元?

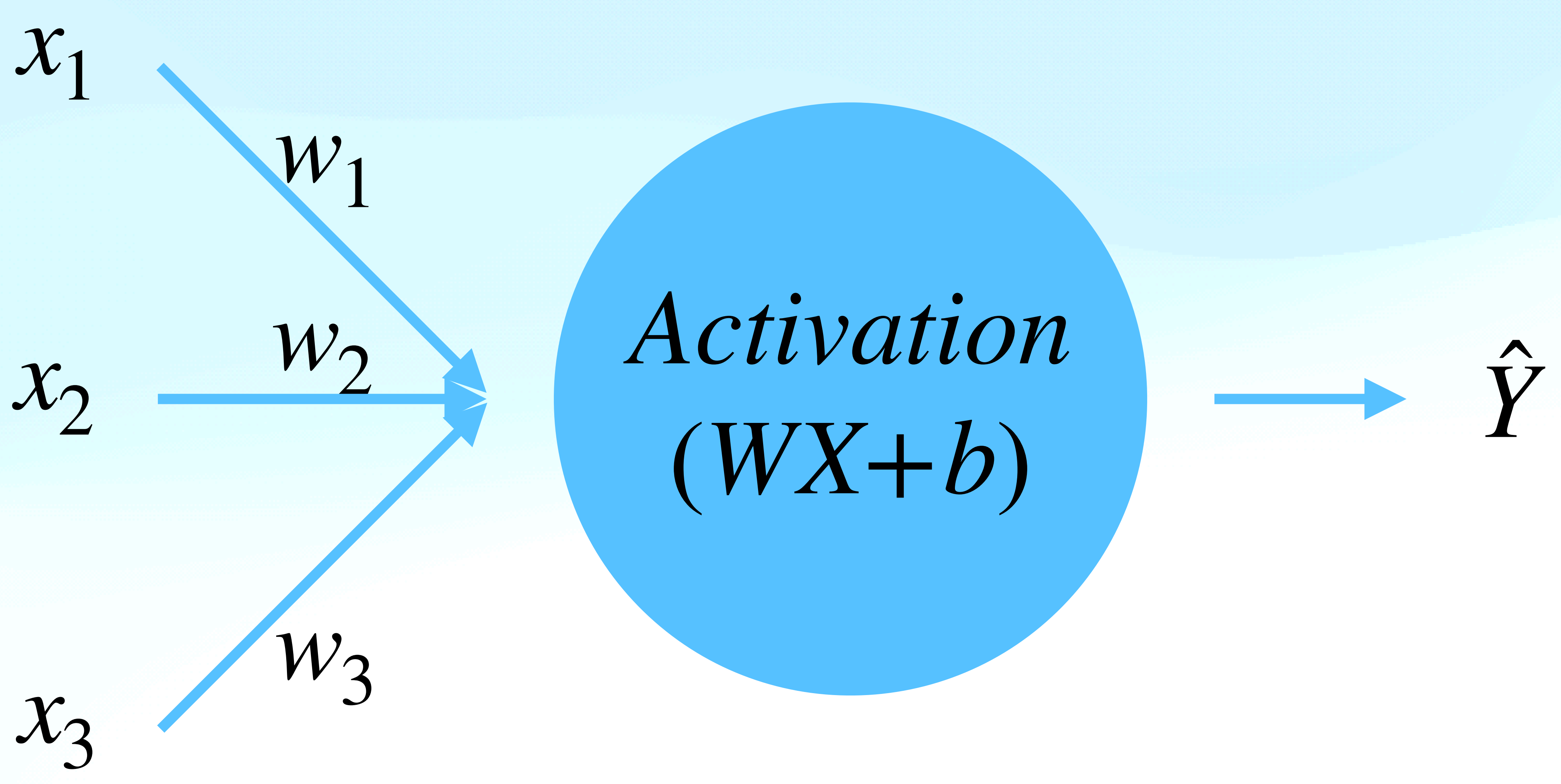
简化



$$\hat{y} = f(x) = Activation(wx + b)$$

何为神经元?

引入矩阵：并行计算

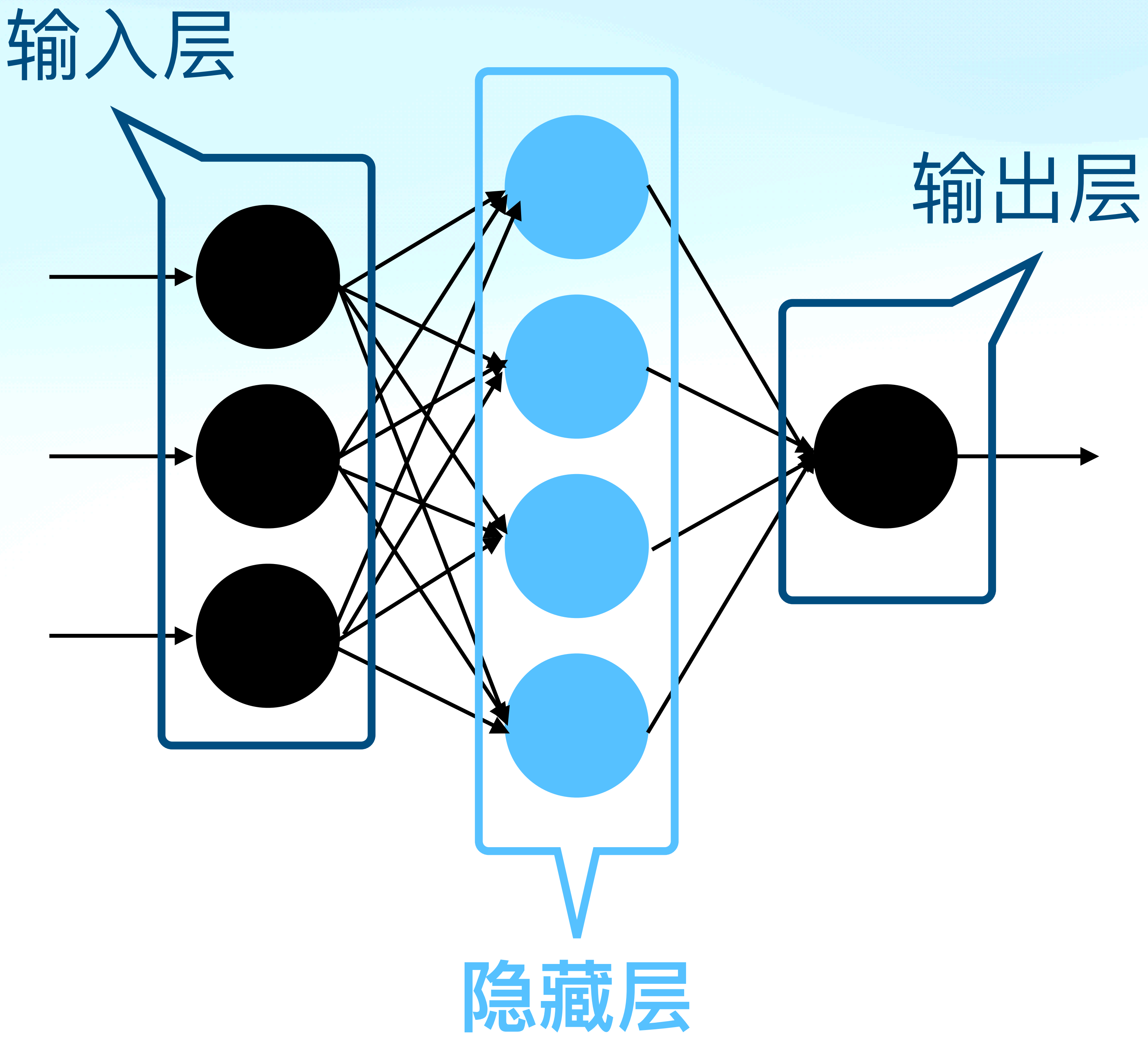


$$\hat{Y} = f(X) = Activation(WX + B)$$

2：隐层（Hidden Layer）

何为隐层?

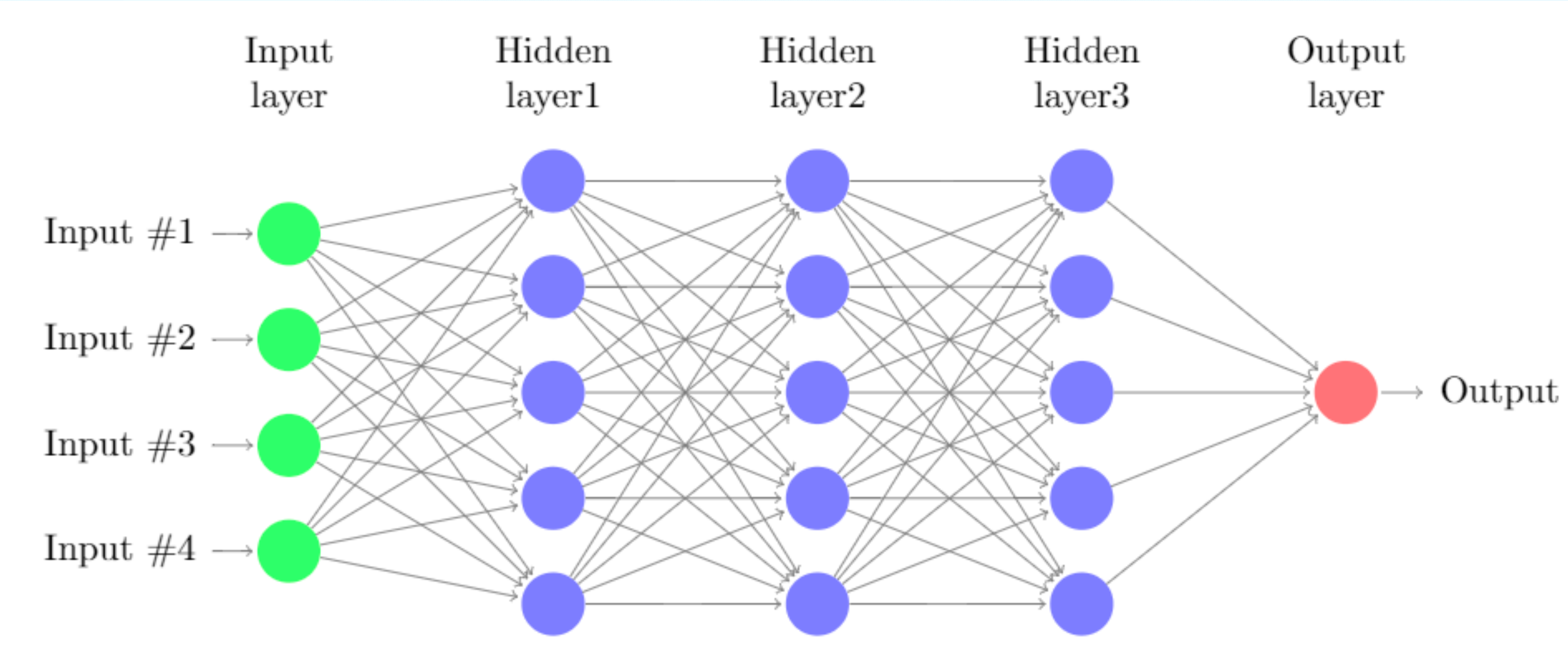
答曰：多个神经元组成的层。



3：神经网络（Neural Network）

何为神经网络?

答曰：多个层的堆砌。

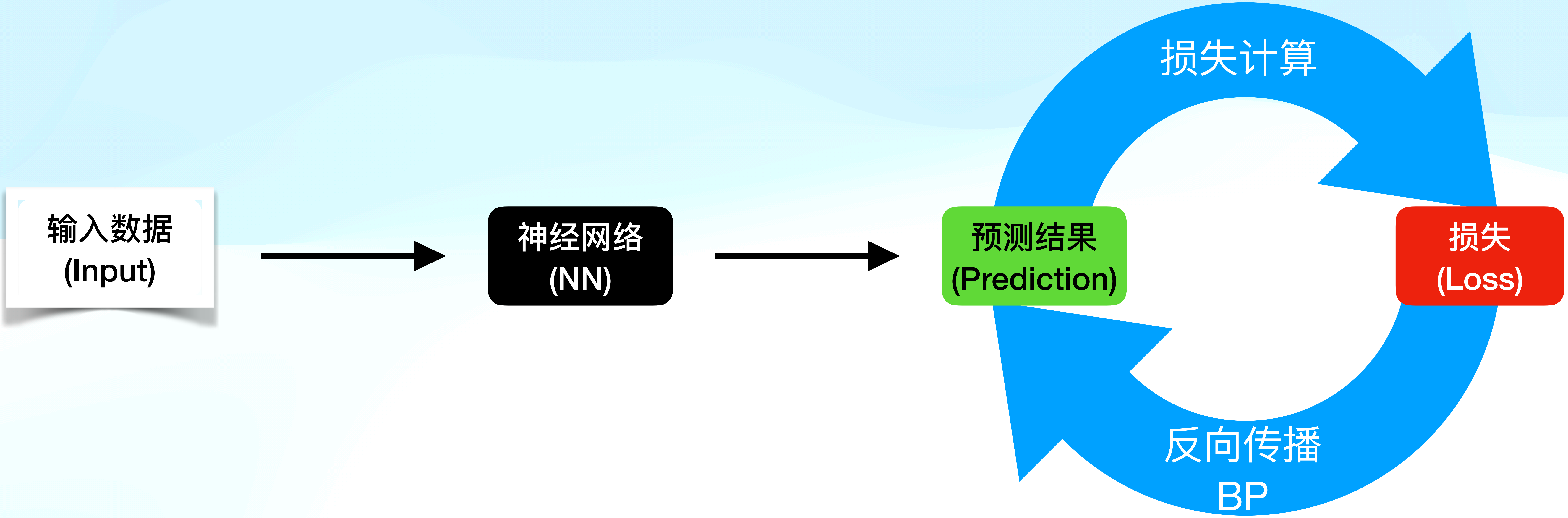


4：损失（Loss）
&
反向传播（Back-Propagation）

注意：只讲逻辑不讲细节，细节我会提供材料以供学习。

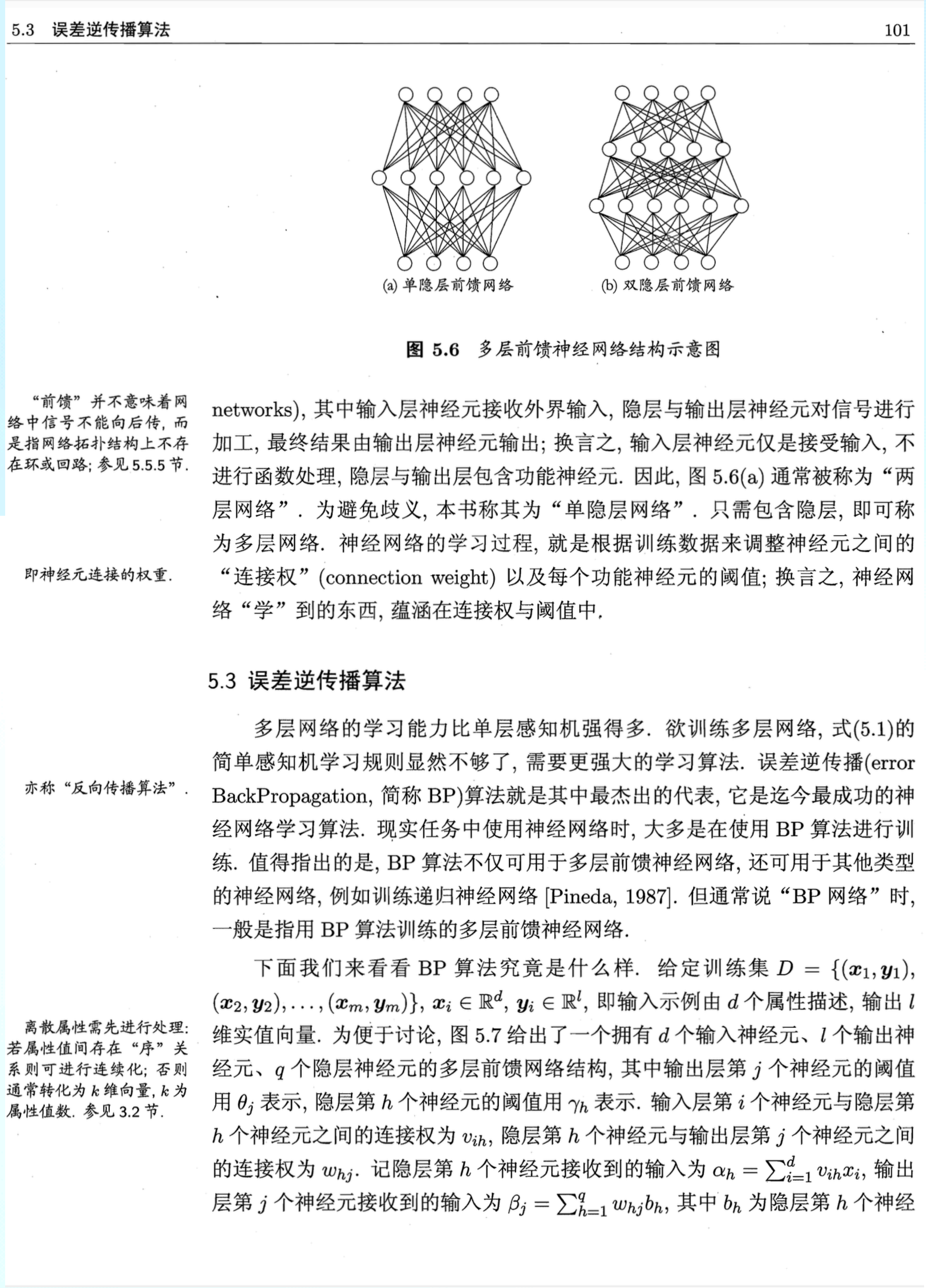
何为损失&反向传播?

答曰：神经网络的答案的得分&根据得分对权重进行更新。



何为损失&反向传播?

详见《机器学习-周志华》P101-106，有极为详尽的链式法则公式推导。



(a) 单隐层前馈网络　(b) 双隐层前馈网络

图 5.6 多层前馈神经网络结构示意图

"前馈"并不意味着网络中信号不能向后传，而是指网络拓扑结构上不存在环或回路；参见 5.5.5 节。

networks)，其中输入层神经元接收外界输入，隐层与输出层神经元对信号进行加工，最终结果由输出层神经元输出；换言之，输入层神经元仅是接受输入，不进行函数处理，隐层与输出层包含功能神经元。因此，图 5.6(a) 通常被称为"两层网络"。为避免歧义，本书称其为"单隐层网络"。只需包含隐层，即可称为多层网络。神经网络的学习过程，就是根据训练数据来调整神经元之间的"连接权"(connection weight) 以及每个功能神经元的阈值；换言之，神经网络"学"到的东西，蕴涵在连接权与阈值中。

即神经元连接的权重。

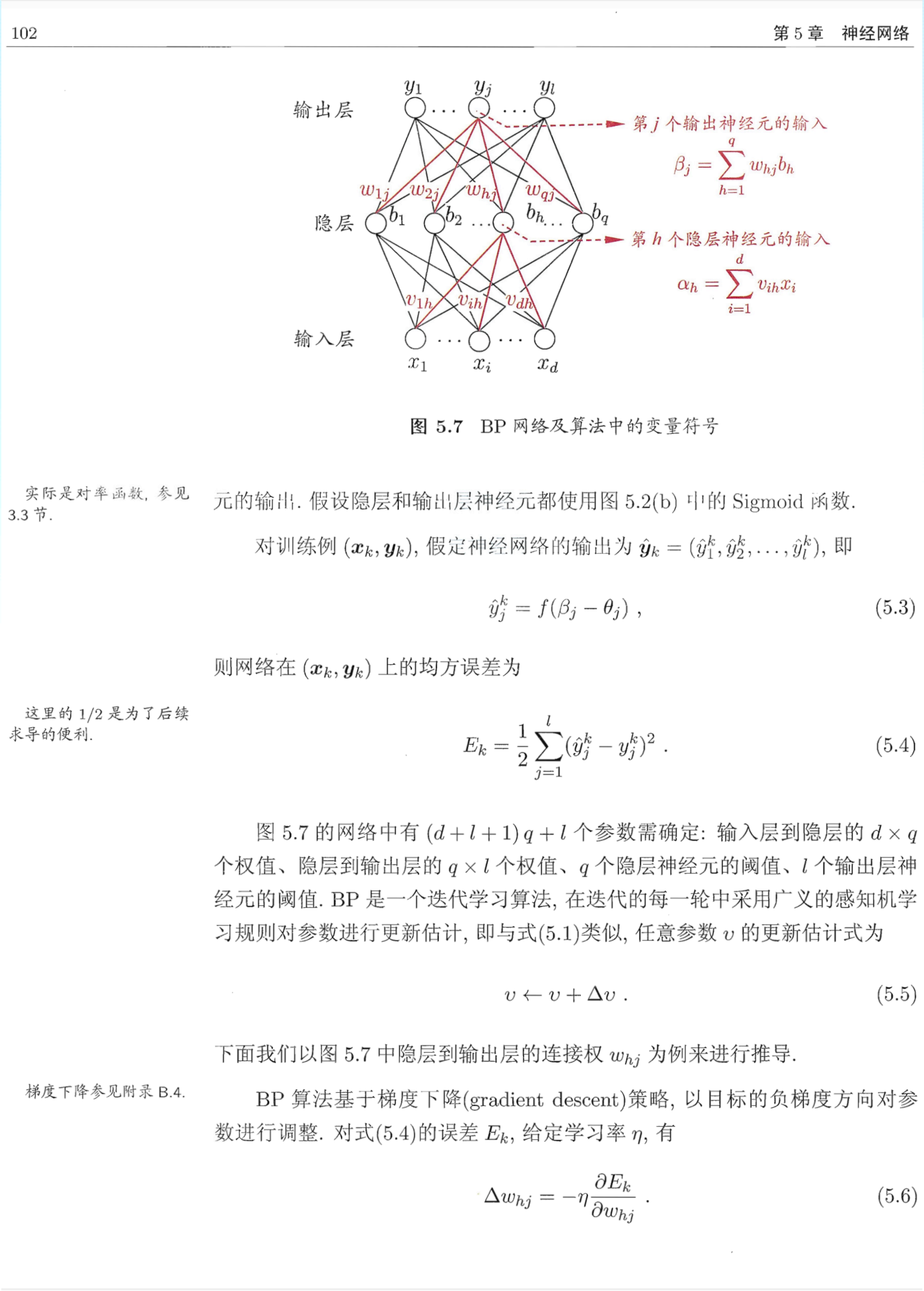
5.3 误差逆传播算法

亦称"反向传播算法"。

多层网络的学习能力比单层感知机强得多。欲训练多层网络，式(5.1)的简单感知机学习规则显然不够了，需要更强大的学习算法。误差逆传播(error BackPropagation，简称 BP)算法就是其中最杰出的代表，它是迄今最成功的神经网络学习算法。现实任务中使用神经网络时，大多是在使用 BP 算法进行训练。值得指出的是，BP 算法不仅可用于多层前馈神经网络，还可用于其他类型的神经网络，例如训练递归神经网络 [Pineda, 1987]。但通常说"BP 网络"时，一般是指用 BP 算法训练的多层前馈神经网络。

离散属性需先进行处理：若属性值间存在"序"关系则可进行连续化；否则通常转化为 k 维向量，k 为属性值数。参见 3.2 节。

下面我们来看看 BP 算法究竟是什么样。给定训练集 $D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$，$\boldsymbol{x}_i \in \mathbb{R}^d$，$\boldsymbol{y}_i \in \mathbb{R}^l$，即输入示例由 d 个属性描述，输出 l 维实值向量。为便于讨论，图 5.7 给出了一个拥有 d 个输入神经元、l 个输出神经元、q 个隐层神经元的多层前馈网络结构，其中输出层第 j 个神经元的阈值用 θ_j 表示，隐层第 h 个神经元的阈值用 γ_h 表示。输入层第 i 个神经元与隐层第 h 个神经元之间的连接权为 v_{ih}，隐层第 h 个神经元与输出层第 j 个神经元之间的连接权为 w_{hj}。记隐层第 h 个神经元接收到的输入为 $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$，输出层第 j 个神经元接收到的输入为 $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$，其中 b_h 为隐层第 h 个神经



输出层　隐层　输入层

第 j 个输出神经元的输入 $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$

第 h 个隐层神经元的输入 $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$

图 5.7 BP 网络及算法中的变量符号

实际是对率函数，参见 3.3 节。

元的输出。假设隐层和输出层神经元都使用图 5.2(b) 中的 Sigmoid 函数。

对训练例 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$，假定神经网络的输出为 $\hat{\boldsymbol{y}}_k = (\hat{y}_1^k, \hat{y}_2^k, \ldots, \hat{y}_l^k)$，即

$$\hat{y}_j^k = f(\beta_j - \theta_j) , \tag{5.3}$$

则网络在 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ 上的均方误差为

这里的 1/2 是为了后续求导的便利。

$$E_k = \frac{1}{2} \sum_{j=1}^{l} (\hat{y}_j^k - y_j^k)^2 . \tag{5.4}$$

图 5.7 的网络中有 $(d + l + 1) q + l$ 个参数需确定：输入层到隐层的 $d \times q$ 个权值、隐层到输出层的 $q \times l$ 个权值、q 个隐层神经元的阈值、l 个输出层神经元的阈值。BP 是一个迭代学习算法，在迭代的每一轮中采用广义的感知机学习规则对参数进行更新估计，即与式(5.1)类似，任意参数 v 的更新估计式为

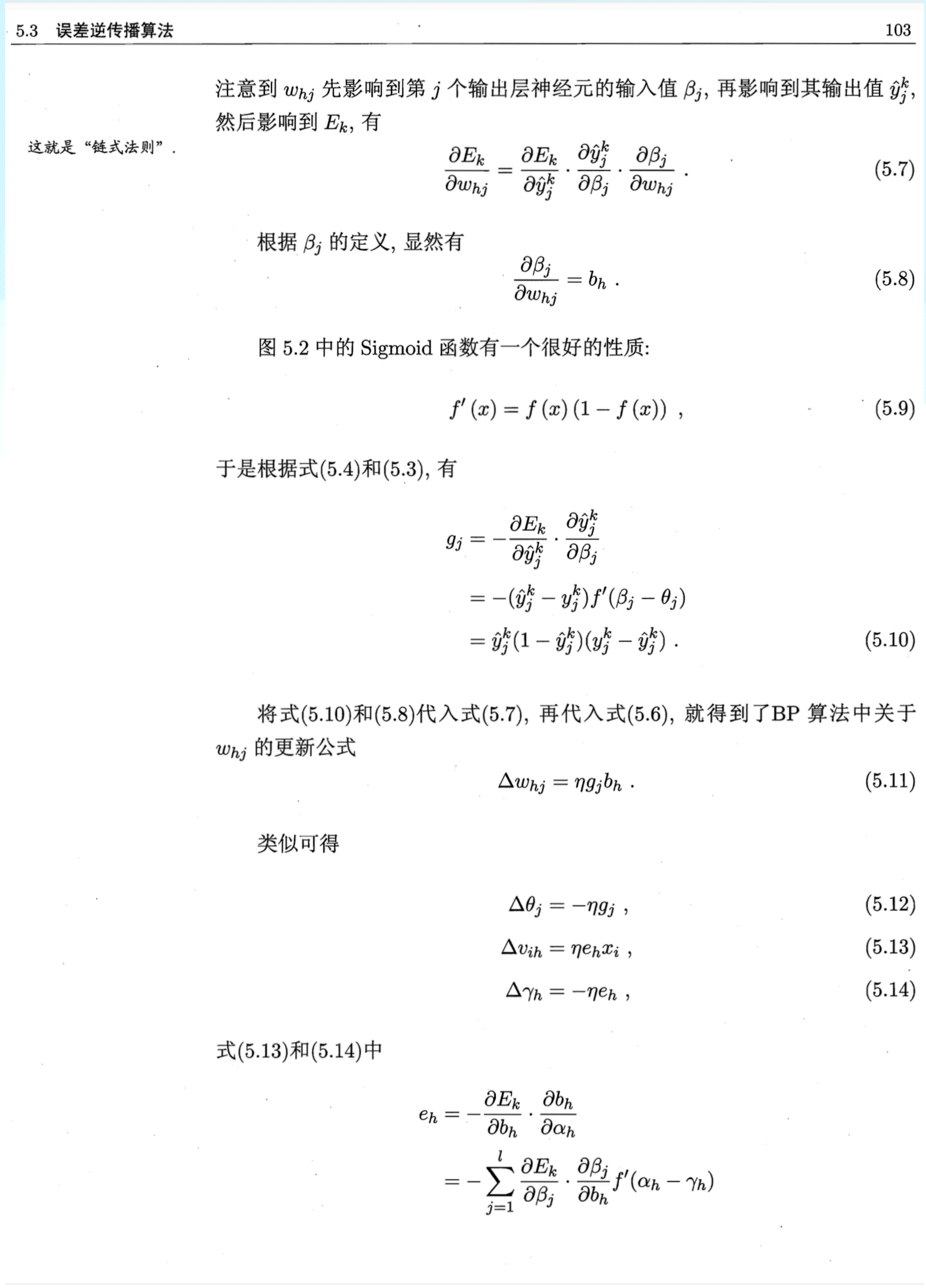
$$v \leftarrow v + \Delta v . \tag{5.5}$$

下面我们以图 5.7 中隐层到输出层的连接权 w_{hj} 为例来进行推导。

梯度下降参见附录 B.4。

BP 算法基于梯度下降(gradient descent)策略，以目标的负梯度方向对参数进行调整。对式(5.4)的误差 E_k，给定学习率 η，有

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} . \tag{5.6}$$



注意到 w_{hj} 先影响到第 j 个输出层神经元的输入值 β_j，再影响到其输出值 $\hat{y}_j^k$，然后影响到 E_k，有

这就是"链式法则"。

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} . \tag{5.7}$$

根据 β_j 的定义，显然有

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h . \tag{5.8}$$

图 5.2 中的 Sigmoid 函数有一个很好的性质：

$$f'(x) = f(x)(1 - f(x)) , \tag{5.9}$$

于是根据式(5.4)和(5.3)，有

$$\begin{aligned} g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) \\ &= \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) . \end{aligned} \tag{5.10}$$

将式(5.10)和(5.8)代入式(5.7)，再代入式(5.6)，就得到了BP 算法中关于 w_{hj} 的更新公式

$$\Delta w_{hj} = \eta g_j b_h . \tag{5.11}$$

类似可得

$$\Delta \theta_j = -\eta g_j , \tag{5.12}$$

$$\Delta v_{ih} = \eta e_h x_i , \tag{5.13}$$

$$\Delta \gamma_h = -\eta e_h , \tag{5.14}$$

式(5.13)和(5.14)中

$$\begin{aligned} e_h &= -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \\ &= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) \end{aligned}$$