第5章 体系架构

深度网络是神经网络的自然进化，它们本身都是起源于由 Rosenblatt 在 1957 年提出的感知机的概念[165]。从历史来看①，Minsky 和 Papert 在 1969 年批评感知机不能对**非线性可分域（nonlinearly separable domain）**进行分类[130]②，而也正是由于这一批评，推动了另一种基于符号表示和推理的人工智能方法的发展。

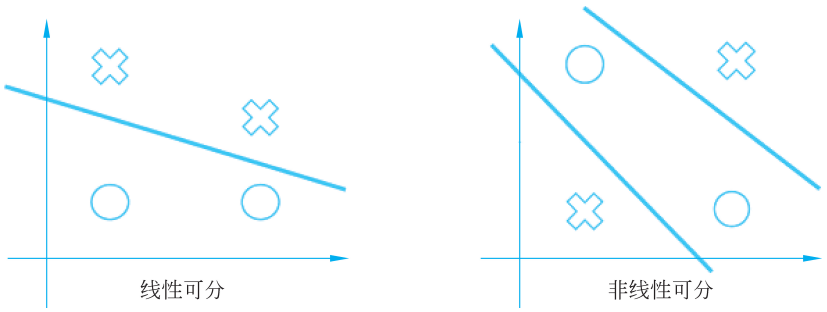

图 5-1 线性可分性的例子（左图）和非线性可分的例子（右图）

神经网络在 20 世纪 80 年代再次出现，它利用**隐藏层（hidden layer）**与非线性单元联合的思想，解决了对初始的线性可分性的限制，并通过**反向传播（Back Propagation，BP）**算法，训练了这类多层神经网络[166]。

20 世纪 90 年代，由于神经网络难以有效地训练多层③，以及其竞争者**支持向量机（Support Vector Machines，SVM）**具有强大的理论支撑且 SVM 的设计理念是使得**分离间**

① 请参阅文献[62]1.2 节，以获得深度学习历史中有关的关键趋势的更详细分析。

② 如图 5-1 所示是一个简单例子和一个线性可分性的反例（在二维空间中属于绿叉或红圆类的 4 个点的集合）。如果至少有一条直线将这两类元素分开，那么它们就是线性可分的。请注意，反例的离散数据版本对应于异或（XOR）逻辑运算符的情况，这是由 Minsky 和 Papert 在文献[130]中提出的。

③ 与此同时，卷积网络开始引起人们的兴趣，特别是在手写数字识别应用中[111]。正如 Goodfellow 等人在文献[62]的 9.11 节中所说的那样："在许多方面，他们为其余的深度学习的传播带来了希望，并为神经网络的广泛应用铺平了道路。"

隔(Separation Margin,SM)最大化[196],因此,人们对神经网络的兴趣开始下降①。

2006 年,Hinton 等人提出的**预训练技术(pre-training)**②[79],解决了这一局限性。2012 年,一种名为 AlexNet 的深度神经网络算法③以显著优势④超越其他基于手工特性的算法并因此赢得了图像识别竞赛(ImageNet Large Scale Visual Recognition Challenge[167])。这一惊人的胜利,终结了那些认为具有许多隐含层的神经网络无法被有效地训练⑤的观点。

5.1 神经网络简介

本节将介绍并回顾**人工神经网络(artificial neural networks)**的基本原理,目的是明确我们在分析各种音乐生成系统时将要使用的关键**概念(concepts)**和**术语(terminology)**。然后,介绍各种被用于音乐应用的衍生架构的概念和基本原理,如自动编码器、递归神经网络、RBM 等。本书不会详细描述神经网络和深度学习的技术,若需要了解有关技术细节可参考文献[62]。

5.1.1 线性回归

虽然受生物神经元的启发,但神经网络和深度学习的基础仍然是**线性回归(linear regression)**。在统计学中,线性回归是一种对假定线性关系进行建模的方式,它用来表示标量 $y \in \mathrm{IR}$ 和一个⑥或多个**解释变量(explanatory variable(s))** $x_1 \cdots x_n$ 之间的关系,其中,$x_i \in \mathrm{IR}$,共同记为向量 $\boldsymbol{x}$。一个简单的例子是预测房价,这需要考虑其中的某些因素(例如,房屋大小、高度、位置……)。

公式(5-1)给出了(多元)线性回归的一般模型,其中:

---

① 另一个相关的限制,是在非常长的序列上有效地训练它们的困难,虽然这是针对于循环网络的情况。Hochreiter 和 Schmidhuber 在 1997 年用**长短期记忆神经网络(Long Short-Term Memory,LSTM)**体系架构解决了这个问题,参见文献[82]5.8.3 节。

② 预训练是指对每个隐含层进行**级联(cascade)**,一次一层,也称为**贪婪分层无监督训练(greedy layer-wise unsupervised training)**的先验训练[79][62]。结果表明,它对于具有多层[46]的神经网络的精确训练,有显著改进。也就是说,预训练现在很少使用,它已经被其他更新的技术,如**批量归一化(batch normalization)**和**深度残差学习(deep residual learning)**所取代。它的底层技术对于解决一些新的问题(比如**迁移学习(transfer learning)**是有用的,它解决了**可重用性(reusability)**的问题(请参阅 8.3 节)。

③ AlexNet 是由 Hinton 领导的 SuperVision 团队设计的,该团队由 Alex Krizhevsky、Ilya Sutskever 和 Geoffrey E. Hinton 组成[103]。AlexNet 是基于深度卷积神经网络构建的,有 6000 万个参数和 65 万个神经元,由 5 个卷积层、最大池化层以及 3 个全局连接层组成。

④ 在第一个任务中,AlexNet 以 15%的错误率赢得了比赛,而其他团队的错误率较高,超过 26%。

⑤ 有趣的是,Hinton 等人关于预训练的文章[79]的标题是关于“深度信念网”,并没有提到“神经网络”这个术语,因为正如 Hinton 在文献[105]中描述的那样:“当时,有一种强烈的信念,即认为深度神经网络不好,它**不可能(never)**被训练,而 ICML(机器学习国际会议)不应该接受有关神经网络的论文。”

⑥ 只有一个解释变量的情况称为**简单线性回归(simple linear regression)**,否则称为**多元线性回归(multiple linear regression)**。

$$h(x)=b+\theta_1 x_1+\cdots+\theta_n x_n=b+\sum_{i=1}^{n}\theta_i x_i \tag{5-1}$$

(1) h 是**模型(model)**,也叫**假说(hypothesis)**,因为它是假设的将要发现(即学习到)的最好模型。

(2) b 是**偏差(bias)**[①],代表**偏移(offset)**。

(3) $\theta_1,\cdots,\theta_n$ 是模型的**参数(parameters)**,也叫**权重(weights)**,它们是相对于不同的解释变量 $x_1,\cdots,x_n$ 的。

5.1.2 符号

本书简单地做以下符号约定。

(1) **常量(constant)**用 Roman(straight)字体表示,例如,整数 1 和音符 C_4。

(2) 模型**变量(variable)**用 Roman 字体表示,例如,输入变量 x 和输出变量 y(可能是向量)。

(3) 模型**参数(parameter)**用斜体表示,如偏置 b、权重参数 θ_1、模型函数 h、解释变量数 n、变量 x_i 的索引 i。

(4) 用斜体和大写表示**概率(probability)**和**概率分布(probability distribution)**,如概率 $P(\text{note}=A_4)$ 表示音符变量的值为 A_4,概率分布 $P(\text{note})$ 表示所有可能音符(结果)的概率分布。

5.1.3 模型训练

训练线性回归模型的目的是找出最适合的针对实际训练数据/示例的每个权重 θ_i 和偏差 b 的值,即各种值对(x, y)。换句话说,要根据某些度量值(**成本(cost)**),找到相应的参数和偏差值,使得对于所有的 x 值,$h(x)$ **尽可能地接近于(as close as possible)** y[②]。在**所有**样本中,这个度量值表示 $h(x)$(即预测值,也记为 $\hat{y}$)和 y(即实际的真实值)之间的**差异(distance)**。

成本,又称**损失(loss)**,记为 $J_\theta(h)$[③],是可以度量的。例如,用均方误差(MSE)度量平均平方差,如式(5-2)所示,其中,m 是样本的数量,$(x^{(i)},y^{(i)})$是第 i 个样本对。

$$J_\theta(h)=1/m\sum_{i=1}^{m}(y^{(i)}-h(x^{(i)}))^2=1/m\sum_{i=1}^{m}(y^{(i)}-\hat{y}^{(i)})^2 \tag{5-2}$$

图 5-2 是简单的线性回归的一个例子,它只有一个解释变量 x。训练数据用蓝色实点表示。一旦模型被训练好,参数值被调节,就可以用蓝色实线拟合最合适的样本。然后将该模型用于**预测(prediction)**。例如,通过计算 $h(x)$(绿色),可以很好地估计出在给定

① 也可记为 θ_0,参见 5.1.5 节。

② 实际上,比线性回归更复杂的神经网络(非线性模型),将在 5.5 节中介绍。这是出于“对训练数据的完美拟合不一定是最好的拟合”这一假设,因为这种情况下可能带来**低泛化(generalization)**的问题,即**预测不可见数据(yet unseen data)**的能力较低。这个问题叫作**过拟合(overfitting)**,它将在 5.5.9 节介绍。

③ 或者 $J(\theta)$,J_θ 或 $J\theta$。

值 x 下，对应于实际值 y 的估计值$\hat{y}$。

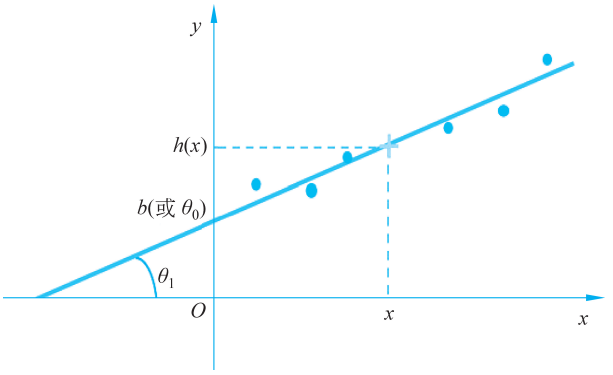

图 5-2　简单的线性回归示例

5.1.4　梯度下降训练算法

使用简单的**梯度下降(gradient descent)**法来训练线性回归模型的基本算法，是非常简单的①。

(1) 将每个参数 θ_i 和偏差 b 初始化为一个随机值或某个启发式值②。

(2) 计算对于所有样本的模型的值 h③。

(3) 计算**成本(cost)** $J_\theta(h)$，如式(5-2)所示。

(4) 计算**梯度(gradients)** $\frac{\partial J_\theta(h)}{\partial \theta_i}$，即代价函数 $J_\theta(h)$ 对每个 θ_i 和偏差 b 的**偏导数(partial derivatives)**。

(5) 按更新规则，**同时更新(update simultaneously)**④所有参数 θ_i 和偏差，如式(5-3)所示⑤，其中，α 为**学习率(learning rate)**。

$$\theta_i := \theta_i - \alpha \frac{\partial J_\theta(h)}{\partial \theta_i} \tag{5-3}$$

如图 5-3 所示，为了降低损失函数 $J_\theta(h)$，进行梯度反向更新。

(6) **迭代(iterate)**执行，直到错误达到**最小值(minimum)**⑥，或者达到一定次数的迭

① 详情见文献[141]。

② 预训练带来了显著的进步，因为它通过使用实际训练数据，通过对连续层[46]进行顺序训练，改善了参数的初始化情况。

③ 计算所有数据例的代价是最好的方法，但也需要很高的计算成本。有许多启发式的选择来最小化计算成本，例如，**随机梯度下降** SGD(**stochastic gradient descent**，其中的数据例是随机选择的)和**小批量梯度下降(minibatch gradient descent**，其中一个数据例的子集是随机选择的)。更多解释可参见文献[62]5.9 节和 8.1.3 节。

④ 同步更新是算法正确运行的必要条件。

⑤ 更新规则也可以记为 $\theta := \theta - \alpha \nabla_\theta J_\theta(h)$，其中，$\nabla_\theta J_\theta(h)$ 是梯度$\frac{\partial J_\theta(h)}{\partial \theta_i}$的向量。

⑥ 如果成本函数是**凸函数(convex)**，对于线性回归的情况，则只有一个**全局最小值(global minimum)**，因此能找到最优的模型(译者注：即能找到全局最优)。

代后终止。

5.1.5 从模型到体系架构

图 5-4 中是作为神经网络前身的线性回归模型的图形表示，所表示的**体系架构(architecture)**实际上是该模型的计算表示[①]。

加权和被表示为一个**计算单元(computational unit)**[②]，即图 5-4 中带有 $\sum$ 的方框，它从画成圆的 x_i 结点获取输入。

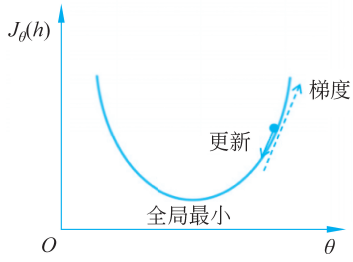

图 5-3 梯度下降

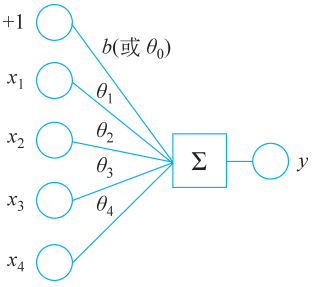

图 5-4 线性回归的架构模型

在图中的示例中有四个输入型解释变量：x_1、x_2、x_3 和 x_4。注意，通常情况下，将偏差 b 视为权重的一种特殊情况(因此记为 θ_0)，它也有一个相应的输入结点(称为**偏差结点(bias node)**)，这是**隐式的(implicit)**[③]，表示为一个常数值(记作+1)。这实际上相当于考虑一个隐含的附加解释变量 x_0，其值为常数值+1(译者注：其权重为偏差 b)，如式(5-4)所示，该公式是最初定义的线性回归公式(5-1)中的一个变形：

$$h(x)=\theta_0+\theta_1x_1+\cdots+\theta_nx_n=\sum_{i=1}^{n}\theta_ix_i \tag{5-4}$$

5.1.6 从模型到线性代数表示

初始线性回归方程式(5-1)中引入式(5-5)的线性代数符号后变得更紧凑。

$$h(x)=b+\theta x \tag{5-5}$$

(1) b 和 $h(x)$是标量。

(2) $\boldsymbol{\theta}$ 是一个由一行 n 个元素$[\theta_1,\theta_2,\cdots,\theta_n]$组成的行向量[④]。

---

① 在这本书中，主要使用**体系架构(architecture)**这个术语，因为我们关注的是实现和计算给定模型的方法以及体系架构和表示之间的关系。

② 我们将结点(node)这个术语用于神经网络的任何组成部分，无论是一个接口(interface)(例如，一个输入结点)还是一个计算单元(computational unit)(例如，一个加权和或一个函数)。我们只在计算结点的情况下使用单元(unit)这个术语。神经元(neuron)这个词也经常用来代替单元(unit)，以强调这是来自生物神经网络的灵感。

③ 然而，稍后将在 5.5 节中解释，偏置结点很少出现在非玩具神经网络的插图中。

④ 这是一个只有一行的矩阵，即它是一个维数为 $1\times n$ 的矩阵。

(3) $\boldsymbol{x}$ 是一个由一列 n 个元素 $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ 组成的列向量[①]。

5.1.7 从简单模型到多元模型

当存在多个待预测的变量 $y_1, \cdots, y_p$ 时，线性回归可泛化为**多元线性回归(multivariate linear regression)**，如图 5-5 所示，有三个预测变量 y_1, y_2, y_3，每个子网用不同的颜色表示。

对应的线性代数方程为式(5-6)，其中：

$$h(\boldsymbol{x}) = \boldsymbol{b} + \boldsymbol{w}_x \tag{5-6}$$

(1) $\boldsymbol{b}$：偏置向量，是维度为 $p \times 1$ 的列向量，其中，b_j 表示偏置输入结点与第 j 个输出结点对应的第 j 个求和运算的关联权重。

(2) $\boldsymbol{W}$：权重矩阵，是维数为 $p \times n$ 的矩阵，有 p 行 n 列，其中，$W_{i,j}$ 代表第 j 个输入结点与第 i 个输出结点对应的第 i 个求和运算对应的权重。

(3) n：输入结点数(不考虑偏置结点)。

(4) p：输出结点的数量。

对于如图 5-5 所示的架构，$n=4$（指 $\boldsymbol{W}$ 的输入结点数和列数），$p=3$(指 $\boldsymbol{W}$ 的输出结点数和行数)。对应的 $\boldsymbol{b}$ 和 $\boldsymbol{W}$ 如式(5-7)和式(5-8)[②]以及图 5-6 所示[③]。

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{5-7}$$

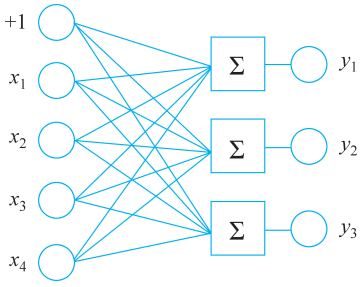

图 5-5 多元线性回归的体系架构

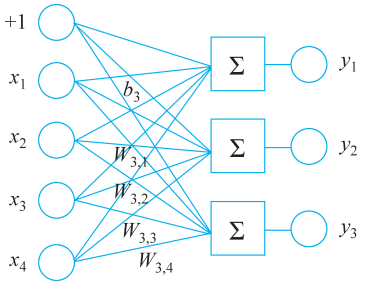

图 5-6 显示与第三层级输出的连接对应的偏置和权重的多元线性回归的体系架构

① 这是一个只有一列的矩阵，即，它是一个维数为 $n \times 1$ 的矩阵。

② 实际上，$\boldsymbol{b}$ 和 $\boldsymbol{W}$ 是 $\boldsymbol{b}$ 和 $\boldsymbol{\theta}$ 从单变量线性回归情况(见 5.1.6 节)到多行多变量线性回归情况的推广，每一行对应一个输出结点。

③ 通常只显示到一个输出结点的连接点，以保持其可读性。

$$\boldsymbol{W}=\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} & W_{1,4} \\ W_{2,1} & W_{2,2} & W_{2,3} & W_{2,4} \\ W_{3,1} & W_{3,2} & W_{3,3} & W_{3,4} \end{bmatrix} \tag{5-8}$$

5.1.8 激活函数

在图 5-7 中，对每个加权求和单元应用了**激活函数(activation function，AF)**。允许使用任意的**非线性函数(nonlinear functions)**来作为这种激活函数。

(1) 从**工程(engineering)**的角度来看，需要一个非线性函数来克服单层感知机的线性可分性限制(见 5.5 节)。

(2) 从**生物灵感(biological inspiration)**的角度来看，非线性函数可以通过传入信号(通过树突)来捕捉神经元激活的**阈值(threshold)**效应，并决定是否沿着输出信号(轴突)来输出信号。

(3) 从**统计学(statistical)**的角度看，当激活函数为 sigmoid 函数时，模型为 **logistic 回归(logistic regression)**模型，它对某一类或某一事件的概率进行建模，并进行二值分类①。

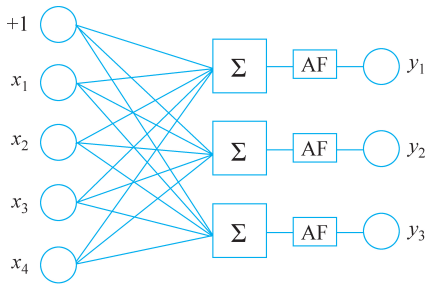

图 5-7 带有激活函数的多元线性回归的系统架构模型

从历史上看，**logistic 回归(logistic regression)**模型的激活函数中，sigmoid 函数是最常见的。sigmoid 函数(通常写成 σ)定义在式(5-9)中，其图形如图 5-8 所示。在 5.5.3 节中将对激活函数做进一步的分析。

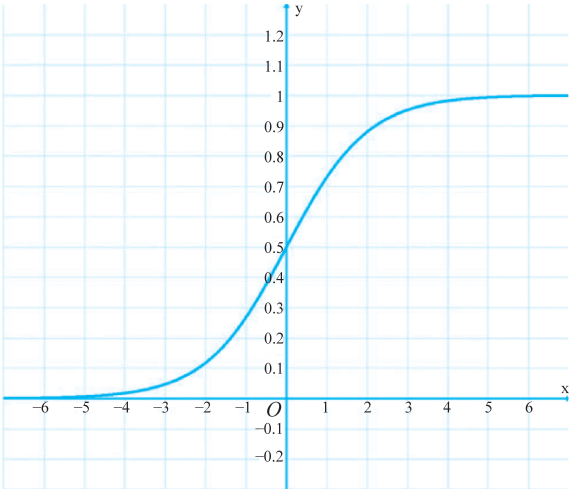

图 5-8 sigmoid 函数

① 对于每个输出结点/变量。更多细节请参见 5.5.3 节。

另一个可选用的激活函数是类似于 sigmoid 的通常被称为 tanh()的双曲正切函数，其值域范围为[−1,+1](而 sigmoid 的值域范围为[0,1])。tanh()的定义如式(5-10)所示，函数图像如图 5-9 所示。

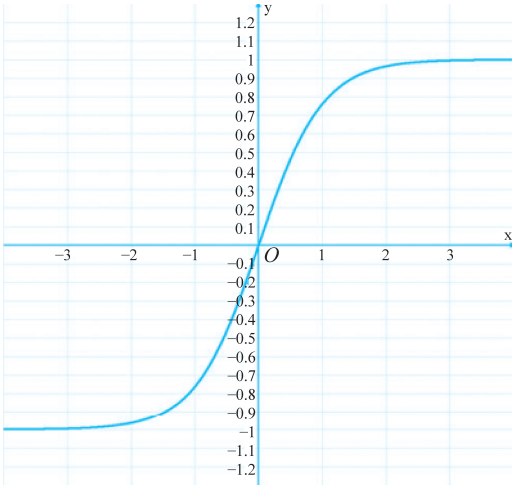

图 5-9 tanh()函数

如今，ReLU 激活函数因其简单和高效而被广泛使用。ReLU 表示**线性整流单元(rectified linear unit)**，其定义见式(5-11)，图像如图 5-10 所示。请注意，通常使用 z 作为激活函数的变量名，因为 x 通常用于输入变量。

$$\text{sigmoid}(z)=\sigma(z)=\frac{1}{1+e^{-z}} \tag{5-9}$$

$$\tanh(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \tag{5-10}$$

$$\text{ReLU}(z)=\max(0,z) \tag{5-11}$$

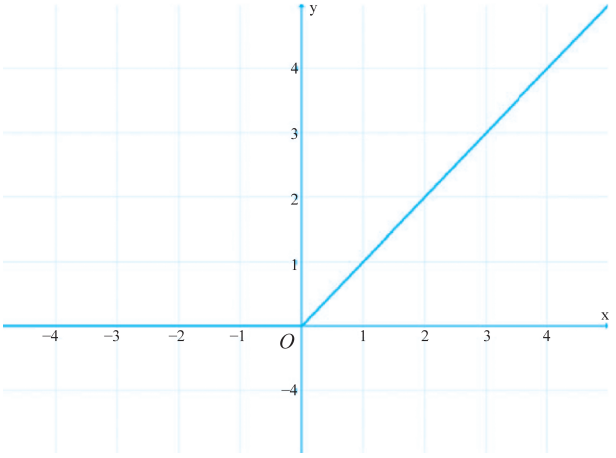

图 5-10 ReLU 函数

5.2 基本组件

如图 5-7 所示的带有激活函数的多元线性回归的体系架构，是一个基于神经网络和深度学习框架的**基本组件(basic building block)**的一个实例(它具有 4 个输入结点和 3 个输出结点)。虽然它的图形简单，但这个基本组件确实是一个工作中的神经网络架构组件。它有两层[①]：

(1) 图中左侧为**输入层(input layer)**，由**输入结点(input nodes)** x_i 和**偏置结点(bias node)**组成。偏置结点是一个**隐式的(implicit)**特定输入结点，常值为 1，因此它通常记为+1。

(2) 图右侧为**输出层(output layer)**，由输出结点 y_j 组成。

训练这样一个基本模块，本质上与训练一个线性回归模型是相同的(它已经在 5.1.3 节中描述了)。

5.2.1 前馈计算

经过训练后，我们可以使用这个基于基本组件的神经网络进行预测。因此，只需对网络进行**前馈(feedforward)**，即向网络“馈进”输入数据并计算对应的输出值，如式(5-12)所示。

$$\hat{y} = \boldsymbol{h}(\boldsymbol{x}) = \mathrm{AF}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}) \tag{5-12}$$

对于如图 5-5 所示的体系架构，用于预测的前馈计算方法如式(5-13)所示，其中，$h_j(\boldsymbol{x})$(即$\hat{y}_j$)为第 j 个变量 y_j 的预测值。

$$\begin{aligned}
\hat{y} = \boldsymbol{h}(\boldsymbol{x}) &= \boldsymbol{h}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \mathrm{AF}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{x}) \\
&= \mathrm{AF}\left(\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} & W_{1,4} \\ W_{2,1} & W_{2,2} & W_{2,3} & W_{2,4} \\ W_{3,1} & W_{3,2} & W_{3,3} & W_{3,4} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}\right) \\
&= \mathrm{AF}\left(\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} W_{1,1} & x_1 + W_{1,2} & x_2 + W_{1,3} & x_3 + W_{1,4} & x_4 \\ W_{2,1} & x_1 + W_{2,2} & x_2 + W_{2,3} & x_3 + W_{2,4} & x_4 \\ W_{3,1} & x_1 + W_{3,2} & x_2 + W_{3,3} & x_3 + W_{3,4} & x_4 \end{bmatrix}\right)
\end{aligned}$$

① 正如将在 5.5.2 节中看到的，虽然它将被视为一个单层神经网络体系架构，但由于其没有隐藏层，因此它仍然受到感知机对线性可分性的限制。

$$
\begin{aligned}
&= \mathrm{AF}\left(\begin{bmatrix} b_1 + W_{1,1} & x_1 + W_{1,2} & x_2 + W_{1,3} & x_3 + W_{1,4} & x_4 \\ b_2 + W_{2,1} & x_1 + W_{2,2} & x_2 + W_{2,3} & x_3 + W_{2,4} & x_4 \\ b_3 + W_{3,1} & x_1 + W_{3,2} & x_2 + W_{3,3} & x_3 + W_{3,4} & x_4 \end{bmatrix}\right) \\
&= \begin{bmatrix} h_1(x) \\ h_2(x) \\ h_3(x) \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}
\end{aligned} \tag{5-13}
$$

5.2.2 同时计算多个输入数据

同时进行前馈的几组输入，很容易通过矩阵乘法表示为一个矩阵。将式(5-12)中的单向量 $\boldsymbol{x}$ 用向量矩阵(通常记为 $\boldsymbol{X}$)代替，可得到式(5-14)。

$\boldsymbol{X}$ 矩阵的连续列对应于不同的输入。我们用一个上标记号 $\boldsymbol{X}^{(k)}$ 来表示第 k 组输入(即 $\boldsymbol{X}$ 矩阵的第 k 列)，以避免其与下标符号 x_i(它表示第 i 个输入变量)混淆。因此，$X_i^{(k)}$ 表示第 k 组输入的第 i 个输入值。几组输入的前馈计算如式(5-15)所示，输出预测值 $\boldsymbol{h}(\boldsymbol{X}^{(k)})$ 是结果输出矩阵的连续列。

$$
\boldsymbol{h}(\boldsymbol{X}) = \mathrm{AF}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{X}) \tag{5-14}
$$

$$
\begin{aligned}
\boldsymbol{h}(\boldsymbol{X}) &= \boldsymbol{h}\begin{bmatrix} X_1^{(1)} & X_1^{(2)} & \cdots & X_1^{(m)} \\ X_2^{(1)} & X_2^{(2)} & \cdots & X_2^{(m)} \\ X_3^{(1)} & X_3^{(2)} & \cdots & X_3^{(m)} \\ X_4^{(1)} & X_4^{(2)} & \cdots & X_4^{(m)} \end{bmatrix} = \mathrm{AF}(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{X}) \\
&= \mathrm{AF}\left(\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} & W_{1,4} \\ W_{2,1} & W_{2,2} & W_{2,3} & W_{2,4} \\ W_{3,1} & W_{3,2} & W_{3,3} & W_{3,4} \end{bmatrix} \times \begin{bmatrix} X_1^{(1)} & X_1^{(2)} & \cdots & X_1^{(m)} \\ X_2^{(1)} & X_2^{(2)} & \cdots & X_2^{(m)} \\ X_3^{(1)} & X_3^{(2)} & \cdots & X_3^{(m)} \\ X_4^{(1)} & X_4^{(2)} & \cdots & X_4^{(m)} \end{bmatrix}\right) \\
&= \begin{bmatrix} h_1(X^{(1)}) & h_1(X^{(2)}) & \cdots & h_1(X^{(m)}) \\ h_2(X^{(1)}) & h_2(X^{(2)}) & \cdots & h_2(X^{(m)}) \\ h_3(X^{(1)}) & h_3(X^{(2)}) & \cdots & h_3(X^{(m)}) \end{bmatrix} = [\boldsymbol{h}(\boldsymbol{X}^{(1)})\boldsymbol{h}(\boldsymbol{X}^{(2)})\cdots\boldsymbol{h}(\boldsymbol{X}^{(m)})]
\end{aligned} \tag{5-15}
$$

请注意，正在进行的主要计算[①]是矩阵的乘积，通过使用已有的线性代数向量化库以及图形处理单元(GPU)等专门的软硬件，可以使计算更高效。

5.3 机器学习

5.3.1 定义

现在，结合线性回归模型(在 5.1.1 节讲述)以及 5.2 节中介绍的基本构件体系架构，

① 除了使用 AF 激活函数的计算。在使用 ReLU 激活函数的情况下，这是很快的。