第一章 深度学习基础

本书假设读者有一定的机器学习基础，了解向量、矩阵、优化、梯度、梯度下降等基础知识。不懂深度学习的读者也可以阅读本书，但是会遇到理解上的障碍。本章只是帮助有一定机器学习基础的读者查漏补缺，并且熟悉本书的语言和符号。如果读者完全不熟悉机器学习，应当先从机器学习的基础开始学起，有一定基础后再阅读本书。

1.1 线性模型

线性模型 (Linear Models) 是一类最简单的机器学习模型，常被用于简单的机器学习任务。可以将线性模型视为单层的神经网络。本节用最小二乘回归、逻辑回归、Softmax 分类器这三种模型解决回归、二分类、多分类问题。

1.1.1 线性回归

以房价预测为例讲解回归 (Regression)。一个房屋有 d 个属性（Attributes 或 Features），比如面积、建造年份、离地铁站的距离。把一个房屋的 d 个属性记作向量：

$$\boldsymbol{x} \;=\; \big[x_1,\, x_2,\, \cdots,\, x_d\big]^T.$$

除非特别说明，本书中的向量都表示为列向量，记作粗体小写字母，以区分标量（实数）。房价预测的目标是基于房屋的属性 $\boldsymbol{x} \in \mathbb{R}^d$ 预测其价格。

有多种方法对房价预测问题建模。最简单方法是使用这样一个线性模型：

$$f\big(\boldsymbol{x}; \boldsymbol{w}, b\big) \;\triangleq\; \boldsymbol{x}^T\boldsymbol{w} + b.$$

这里 $\boldsymbol{w} \in \mathbb{R}^d$ 和 $b \in \mathbb{R}$ 是模型的参数 (Parameters)。线性模型 $f(\boldsymbol{x}; \boldsymbol{w}, b)$ 的输出就是对房价的预测；输出既依赖于房屋的特征 $\boldsymbol{x}$，也依赖于参数 $\boldsymbol{w}$ 和 b。很多书和论文将 $\boldsymbol{w}$ 称作权重 (Weights)，将 b 称作偏移量 (Bias 或 Intercept)，原因是这样的：可以将 f 的定义 $\boldsymbol{x}^T\boldsymbol{w} + b$ 展开，得到

$$f\big(\boldsymbol{x}; \boldsymbol{w}, b\big) \;\triangleq\; w_1x_1 \,+\, w_2x_2 \,+\, \cdots \,+\, w_dx_d \,+\, b.$$

如果 x_1 是房屋的面积，那么 w_1 就是房屋面积在房价中的权重。w_1 越大，说明房价与面积的相关性越强；这就是为什么 $\boldsymbol{w}$ 被称为权重。可以把偏移量 b 视作市面上房价的均值或者中位数，它与房屋的具体属性无关。

线性模型 $f(\boldsymbol{x}; \boldsymbol{w}, b)$ 依赖于参数 $\boldsymbol{w}$ 和 b；只有确定了 $\boldsymbol{w}$ 和 b，我们才能利用线性模型做预测。该怎么样获得 $\boldsymbol{w}$ 和 b 呢？可以用历史数据来训练模型，得到参数 $\boldsymbol{w}^\star$ 和 $b^\star$，然后就可以用线性模型做预测：

$$f\big(\boldsymbol{x}; \boldsymbol{w}^\star, b^\star\big) \;\triangleq\; \boldsymbol{x}^T\boldsymbol{w}^\star + b^\star$$

卖家和中介可以用这个训练好的模型 f 给待售房屋定价。对于一个待售的房屋，首先找

到它的面积、建造年份等属性，表示成向量 $\boldsymbol{x}'$，然后把它输入 f，得到

$$\widehat{y}' \ = \ f\big(\boldsymbol{x}'; \boldsymbol{w}^\star, b^\star\big),$$

把它作为对该房屋价格的预测。

下面用最小二乘回归方法 (Least Squares Regression) 为例，讲解如何训练线性模型 $f(\boldsymbol{x}; \boldsymbol{w}, b)$。训练有以下几个要点：

- **第一，准备训练数据**。收集到近期的 n 个房屋的属性和卖价，作为训练数据集。把训练集记作 $(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)$。向量 $\boldsymbol{x}_i \in \mathbb{R}^d$ 表示第 i 个房屋的所有属性，标量 y_i 表示该房屋的成交价格。
- **第二，把训练描述成优化问题**。模型对第 i 个房屋价格的预测是 $\widehat{y}_i = f(\boldsymbol{x}_i; \boldsymbol{w}, b)$，而这个房屋的真实成交价格是 y_i。我们希望 $\widehat{y}_i$ 尽量接近 y_i，所以希望平方误差 $(\widehat{y}_i - y_i)^2$ 越小越好。定义损失函数：

  $$L\big(\boldsymbol{w}, b\big) \ = \ \frac{1}{2n} \sum_{i=1}^{n} \Big[f(\boldsymbol{x}_i; \boldsymbol{w}, b) \ - \ y_i \Big]^2.$$

  我们希望找到 $\boldsymbol{w}$ 和 b 使得损失函数尽量小，也就是让模型的预测尽量准确。定义下面的优化模型：

  $$\min_{\boldsymbol{w}, b} \ L\big(\boldsymbol{w}, b\big) + R(\boldsymbol{w}).$$

  这个优化模型叫做最小二乘回归 (Least Squares Regression)。模型中的参数 $\boldsymbol{w}$ 和 b 在此处叫做优化变量。$L(\boldsymbol{w}, b) + R(\boldsymbol{w})$ 是目标函数。$R(\boldsymbol{w})$ 是正则项 (Regularizer)，比如：

  $$R\big(\boldsymbol{w}\big) \ = \ \lambda \|\boldsymbol{w}\|_2^2 \qquad \text{或} \qquad R\big(\boldsymbol{w}\big) \ = \ \lambda \|\boldsymbol{w}\|_1.$$

  把优化问题的最优解记作：

  $$\big(\boldsymbol{w}^\star, b^\star\big) \ = \ \operatorname*{argmin}_{\boldsymbol{w}, b} \ L\big(\boldsymbol{w}, b\big) + R(\boldsymbol{w}).$$

  请注意 min 与 argmin 的区别。
- **第三，用数值优化算法求解模型**。在建立优化模型之后，需要寻找最优解 $(\boldsymbol{w}^\star, b^\star)$。通常随机初始化（或全零初始化）$\boldsymbol{w}$ 和 b，然后用共轭梯度下降、随机梯度下降等优化算法迭代更新 $\boldsymbol{w}$ 和 b。

1.1.2 逻辑回归

上一小节介绍了回归问题，其中的预测目标 y 是连续变量，比如房价就是连续数值。本小节研究二分类问题 (Binary Classification)，其中的预测目标 y 不是连续变量，而是二元变量，要么等于 0，要么等于 1。本小节用逻辑回归 (Logistic Regression) 解决二元分类问题[1]。

以疾病检测为例讲解二元分类问题。为了初步排查癌症，需要做血检，血检中有 d 项指标，包括白细胞数量、含氧量、以及多种激素含量。一份血液样本的检测报告作为

[1]注意，虽然“逻辑回归”的名字有“回归”，但其通常用于解决二分类问题，而非回归问题

一个 d 维向量：

$$\boldsymbol{x} = [x_1, x_2, \cdots, x_d]^T.$$

医生需要基于 $\boldsymbol{x}$ 来初步判断该血检是否意味着癌症。如果医生的判断为 $y = 1$，则要求病人做进一步检测；如果医生的判断为 $y = 0$，则意味着未患癌症。这就是一个典型的二元分类问题。是否可以让机器学习做这种二元分类呢？

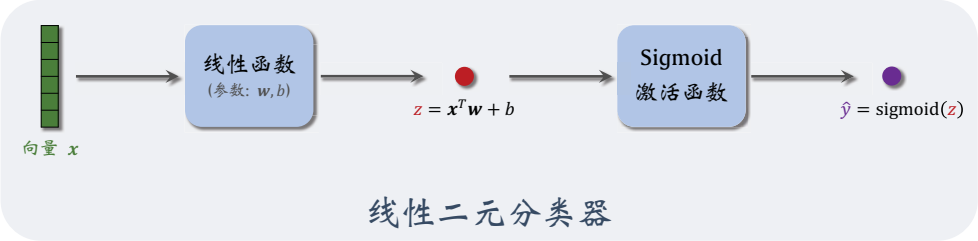

图 1.1: 线性 Sigmoid 分类器的结构。输入是向量 $\boldsymbol{x} \in \mathbb{R}^d$，输出是介于 0 和 1 之间的标量。

常用的是线性 Sigmoid 分类器，结构如图 1.1 所示。基于输入的向量 $\boldsymbol{x}$，线性分类器做出预测：

$$f(\boldsymbol{x}; \boldsymbol{w}, b) \triangleq \text{sigmoid}\left(\boldsymbol{x}^T \boldsymbol{w} + b\right).$$

此处的 Sigmoid 是个激活函数 (Activation Function)，定义为：

$$\text{sigmoid}(z) \triangleq \frac{1}{1 + \exp(-z)}.$$

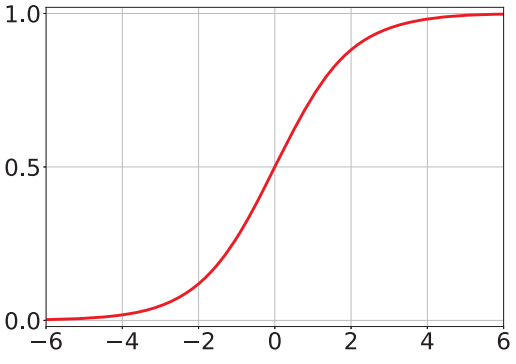

图 1.2: Sigmoid 函数的图像。

如图 1.2 所示，Sigmoid 可以把任何实数映射到 0 到 1 之间。我们希望分类器的输出 $\widehat{y} = f(\boldsymbol{x}; \boldsymbol{w}, b)$ 有这样的性质：如果 $\boldsymbol{x}$ 是癌症患者的血检数据，那么 $\widehat{y}$ 接近 1；如果 $\boldsymbol{x}$ 是健康人的血检数据，那么 $\widehat{y}$ 接近 0。因此 $\widehat{y}$ 叫做 “置信率” (Confidence)，即分类器有多大信心做出阳性的判断。比如 $\widehat{y} = 0.9$ 表示分类器有 0.9 的信心判断血检为阳性；$\widehat{y} = 0.05$ 表示分类器只有 0.05 的信心判断血检为阳性，即 0.95 的信心判断血检为阴性。

在介绍训练 Sigmoid 分类器的算法之前，先介绍交叉熵 (Cross Entropy)，它可以衡量两个概率分布的差别，因此常被用作分类问题的损失函数。用向量

$$\boldsymbol{p} = [p_1, \cdots, p_m] \quad \text{和} \quad \boldsymbol{q} = [q_1, \cdots, q_m]$$

表示两个离散概率分布。向量的元素都非负，而且 $\sum_{j=1}^m p_j = 1$，$\sum_{j=1}^m q_j = 1$。两个概率分布的交叉熵定义为：

$$H(\boldsymbol{p}, \boldsymbol{q}) = -\sum_{j=1}^{m} p_j \cdot \ln q_j.$$

两个概率分布越接近，则交叉熵越大。

我们做以下步骤，从数据中学习模型参数 $\boldsymbol{w}$ 和 b。

- **第一，准备训练数据**。收集 n 份血检报告和最终的诊断，作为训练数据集：$(\boldsymbol{x}_1, y_1)$,

$\cdots, (\boldsymbol{x}_n, y_n)$。向量 $\boldsymbol{x}_i \in \mathbb{R}^d$ 表示第 i 份血检报告中的所有指标；二元标签 $y_i = 1$ 表示患有癌症（阳性），$y_i = 0$ 表示健康（阴性）。

- **第二，把训练描述成优化问题**。分类器对第 i 份血检报告的预测是 $f(\boldsymbol{x}_i; \boldsymbol{w}, b)$，而真实患癌情况是 y_i。想要用交叉熵衡量 y_i 与 $f(\boldsymbol{x}_i; \boldsymbol{w}, b)$ 之间的差别，得把 y_i 与 $f(\boldsymbol{x}_i; \boldsymbol{w}, b)$ 表示成向量：

$$\begin{bmatrix} y_i \\ 1 - y_i \end{bmatrix} \quad \text{和} \quad \begin{bmatrix} f(\boldsymbol{x}_i; \boldsymbol{w}, b) \\ 1 - f(\boldsymbol{x}_i; \boldsymbol{w}, b) \end{bmatrix}.$$

两个向量的第一个元素都对应阳性的置信率，第二个元素都对应阴性的置信率。分类器预测越准确，则两个向量尽量越接近，它们的交叉熵越小。定义损失函数为平均交叉熵：

$$L(\boldsymbol{w}, b) = \frac{1}{n} \sum_{i=1}^{n} H\left(\begin{bmatrix} y_i \\ 1 - y_i \end{bmatrix}, \begin{bmatrix} f(\boldsymbol{x}_i; \boldsymbol{w}, b) \\ 1 - f(\boldsymbol{x}_i; \boldsymbol{w}, b) \end{bmatrix} \right).$$

我们希望找到 $\boldsymbol{w}$ 和 b 使得损失函数尽量小，也就是让分类器的预测尽量准确。定义下面的优化问题：

$$\min_{\boldsymbol{w}, b} L(\boldsymbol{w}, b) + R(\boldsymbol{w}).$$

这个优化问题叫做逻辑回归。公式中的 $R(\boldsymbol{w})$ 是正则项。

- **第三，用数值优化算法求解**。在建立优化模型之后，需要寻找最优解 $(\boldsymbol{w}^\star, b^\star)$。通常随机初始化（或全零初始化）优化变量 $\boldsymbol{w}$ 和 b，然后用梯度下降、随机梯度下降、L-BFGS 等优化算法迭代更新优化变量。

1.1.3 Softmax 分类器

上一小节介绍了二元分类问题，数据只分为两个类别，比如患病和健康。本小节研究多分类问题，数据可以划分为 $k\,(>2)$ 个类别。我们可以用线性 Softmax 分类器解决多分类问题。

本小节用 MNIST 手写数字识别为例讲解多分类问题。如图 1.3 所示，MNIST 数据集有 $n = 60,000$ 个样本，每个样本是 28×28 的图片。数据集有 $k = 10$ 个类别，每个样本有一个类别标签，它是介于 0 到 9 之间的整数，表示图片中的数字。为了训练 Softmax 分类器，我们要对标签做 One-Hot 编码，把每个标签（0 到 9 之间的整数）映射到 $k = 10$ 维的向量：

图 1.3: MNIST 数据集中的图片。

$$0 \quad \Longrightarrow \quad [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],$$

$$
\begin{aligned}
1 \quad &\Longrightarrow \quad [0,1,0,0,0,0,0,0,0,0], \\
&\vdots \\
8 \quad &\Longrightarrow \quad [0,0,0,0,0,0,0,0,1,0], \\
9 \quad &\Longrightarrow \quad [0,0,0,0,0,0,0,0,0,1].
\end{aligned}
$$

把得到的标签记作 $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n \in \mathbb{R}^{10}$。把每张 28×28 的图片拉伸成 $d = 784$ 维的向量，记作 $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathbb{R}^{784}$。

在介绍 Softmax 分类器之前，先介绍 Softmax 激活函数。它的输入和输出都是 k 维向量。设 $\boldsymbol{z} = [z_1, \cdots, z_k]^T$ 是任意 k 维向量，它的元素可正可负。Softmax 函数的输出

$$
\text{softmax}\left(\boldsymbol{z}\right) \triangleq \frac{1}{\sum_{l=1}^{k} \exp(z_l)} \left[\exp\left(z_1\right), \exp\left(z_2\right), \cdots, \exp\left(z_k\right) \right]^T
$$

也是个 k 维向量，它的元素都是非负，而且相加等于 1。如图 1.4 所示，Softmax 函数让最大的元素相对变得更大，让小的元素接近 0。图 1.5 是 Max 函数，它把最大的元素映射到 1，其余所有元素映射到 0。对比一下图 1.4 和图 1.5，不难看出为什么 Softmax 没有让小的元素等于零，这就是为什么它的名字带有 “Soft”。

图 1.4: Softmax 函数把左边红色的 10 个数值映射到右边紫色的 10 个数值。

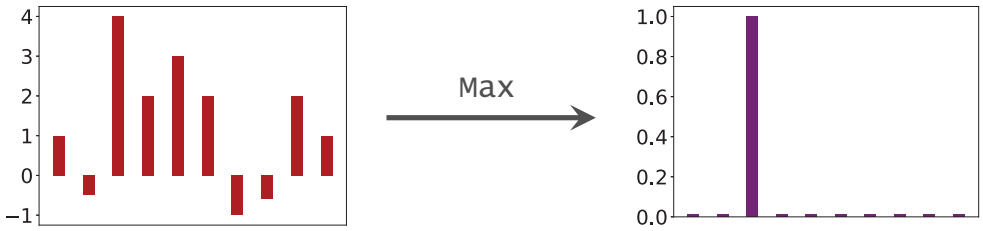

图 1.5: Max 函数把左边红色的 10 个数值映射到右边紫色的 10 个数值。

Softmax 分类器是常用的多元分类器。线性 Softmax 分类器其实是线性函数 + Softmax 激活函数；结构如图 1.6 所示。它的参数是矩阵 $\boldsymbol{W} \in \mathbb{R}^{k \times d}$ 和向量 $\boldsymbol{b} \in \mathbb{R}^k$，这里的 d 是输入向量的维度，k 是类别数量。基于输入的向量 $\boldsymbol{x} \in \mathbb{R}^d$，分类器做出预测：

$$
\boldsymbol{f}\left(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}\right) \triangleq \text{softmax}\left(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}\right).
$$

设 $\widehat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b})$ 是 Softmax 分类器的输出，它的 $k = 10$ 个元素可以视为 $k = 10$ 个类别的置信率。举个例子，设

$$
\widehat{\boldsymbol{y}} = \left[0.1,\, 0.6,\, 0.02,\, 0.01,\, 0.01,\, 0.2,\, 0.01,\, 0.03,\, 0.01,\, 0.01\right]^T.
$$

可以这样理解分类器的输出向量 $\widehat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b})$：

- 第零个（从零计数）元素 0.1 表示分类器以 0.1 的信心判定图片 x 是数字 “0”，
- 第一个元素 0.6 表示分类器以 0.6 的信心判定 $\boldsymbol{x}$ 是数字 “1”，
- 第二个元素 0.02 表示分类器只有 0.02 的信心判定 $\boldsymbol{x}$ 是数字 “2”，

以此类推。由于分类器的输出向量 $\widehat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b})$ 的第 1 个元素 0.6 是最大的，分类器认为图片 $\boldsymbol{x}$ 是数字 “1”。

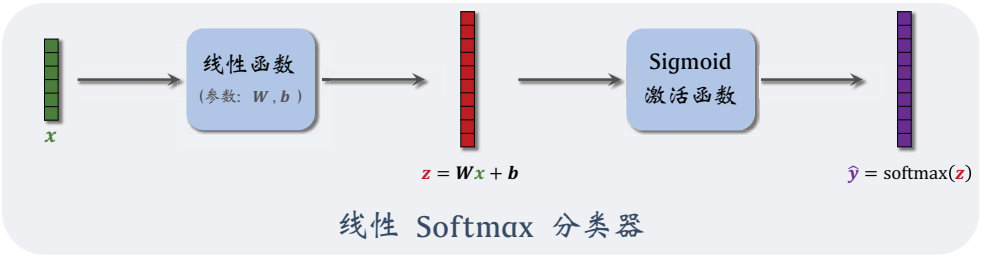

图 1.6: 线性 Softmax 分类器的结构。输入是向量 $\boldsymbol{x} \in \mathbb{R}^d$，输出是 $\widehat{\boldsymbol{y}} \in \mathbb{R}^k$。

我们做以下步骤，从数据中学习模型参数 $\boldsymbol{W} \in \mathbb{R}^{k \times d}$ 和 $\boldsymbol{b} \in \mathbb{R}^k$。

- **第一，准备训练数据**。一共有 $n = 60,000$ 张手写数字图片，每张图片大小为 28×28，需要把图片变成 $d = 784$ 维的向量，记作 $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathbb{R}^d$。每张图片有一个标签，它是 0 到 9 之间的整数，需要把它做 One-Hot 编码，变成 $k = 10$ 维的 One-Hot 向量；把 One-Hot 标签记作 $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n$。
- **第二，把训练描述成优化问题**。分类器对第 i 张图片 $\boldsymbol{x}_i$ 的预测是 $\widehat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \boldsymbol{W}, \boldsymbol{b})$，它是 $k = 10$ 维的向量，可以反映出分类结果。我们希望 $\widehat{\boldsymbol{y}}_i$ 尽量接近真实标签 $\boldsymbol{y}_i$（10 维的 One-Hot 向量），也就是希望交叉熵 $H(\boldsymbol{y}_i, \widehat{\boldsymbol{y}}_i)$ 尽量小。定义损失函数为平均交叉熵：

$$L(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{n} \sum_{i=1}^{n} H(\boldsymbol{y}_i, \widehat{\boldsymbol{y}}_i).$$

  我们希望找到参数矩阵 $\boldsymbol{W}$ 和向量 $\boldsymbol{b}$ 使得损失函数尽量小，也就是让分类器的预测尽量准确。定义下面的优化问题：

$$\min_{\boldsymbol{W}, \boldsymbol{b}} L(\boldsymbol{W}, \boldsymbol{b}) + R(\boldsymbol{W}).$$

- **第三，用数值优化算法求解**。在建立优化模型之后，需要寻找最优解 $(\boldsymbol{W}^\star, \boldsymbol{b}^\star)$。通常随机初始化（或全零初始化）优化变量 $\boldsymbol{W}$ 和 $\boldsymbol{b}$，然后用梯度下降、随机梯度下降等优化算法迭代更新优化变量。

1.2 神经网络

1.2.1 全连接神经网络（多层感知器）

接着上一节的内容，我们继续研究 MNIST 手写识别这个多元分类问题。人类做手写数字识别的准确率接近 100%，然而线性 Softmax 分类器在 MNIST 数据集只有 90% 的准确率，远低于人类的表现。线性分类器的表现差的原因在于模型太小，不能充分利用 $n = 60,000$ 个训练样本。我们可以把“线性函数 + 激活函数”这样的结构作为一个层，累起来，得到一个多层的神经网络，获得更高的预测准确率。

全连接层： 把输入记作向量 $\boldsymbol{x} \in \mathbb{R}^d$，神经网络的一个层把 $\boldsymbol{x}$ 映射到 $\boldsymbol{x}' \in \mathbb{R}^{d'}$。全连接层是这样定义的：

$$\boldsymbol{x}' = \sigma(\boldsymbol{z}), \qquad \boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b},$$

此处的矩阵 $\boldsymbol{W} \in \mathbb{R}^{d' \times d}$ 和向量 $\boldsymbol{b} \in \mathbb{R}^{d'}$ 是这一层的参数，需要从数据中学习；$\sigma(\cdot)$ 是激活函数，比如 Softmax 函数、Sigmoid 函数、ReLU 函数。最常用的激活函数是 ReLU，定义为：

$$\mathrm{ReLU}\left(\boldsymbol{z}\right) = \left[\max\left\{0, z_1\right\}, \max\left\{0, z_2\right\}, \cdots, \max\left\{0, z_{d'}\right\}\right]^T.$$

我们把这样的一个层叫做全连接层（Dense Layer、Linear Layer、或 Fully Connected Layer），结构如图 1.7 所示。只有线性函数、没有激活函数也算一层；但是单独的激活函数不算一层，因为激活函数中没有要学习的参数。

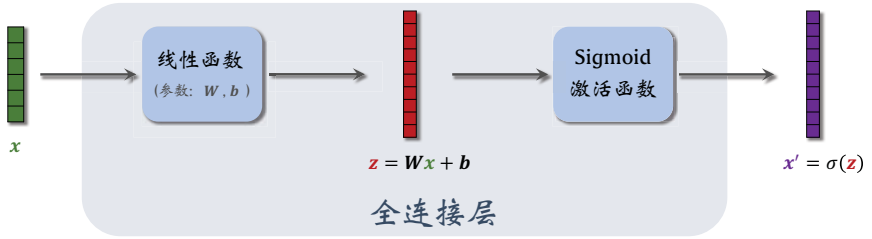

图 1.7: 一个全连接层包括一个线性函数和一个激活函数。

全连接神经网络： 我们可以把全连接层当做基本组件，然后像搭积木一样搭建一个全连接神经网络 (Fully-Connected Neural Network)，也叫多层感知器 (Multi-Layer Perceptron, MLP)。图 1.8 中是一个三层的全连接神经网络，它把输入向量 $\boldsymbol{x}^{(0)}$ 映射到 $\boldsymbol{x}^{(3)}$。一个 l 层的全连接神经网络可以表示为：

$$\begin{aligned}
&\text{第 1 层:} && \boldsymbol{x}^{(1)} = \sigma_1\left(\boldsymbol{W}^{(1)}\boldsymbol{x}^{(0)} + \boldsymbol{b}^{(1)}\right),\\
&\text{第 2 层:} && \boldsymbol{x}^{(2)} = \sigma_2\left(\boldsymbol{W}^{(2)}\boldsymbol{x}^{(1)} + \boldsymbol{b}^{(2)}\right),\\
&\quad\vdots && \quad\vdots\\
&\text{第 } l \text{ 层:} && \boldsymbol{x}^{(l)} = \sigma_l\left(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}\right),
\end{aligned}$$

其中的 $\boldsymbol{W}^{(1)},\cdots,\boldsymbol{W}^{(l)}$, $\boldsymbol{b}^{(1)},\cdots,\boldsymbol{b}^{(l)}$ 是神经网络的参数，需要从训练数据学习；不同层的参数是不同的。$\sigma_1,\cdots,\sigma_l$ 为激活函数；它们可以相同，也可以不同。

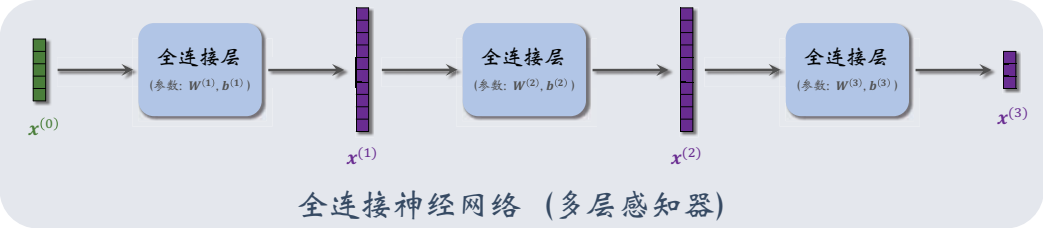

图 1.8: 图中的神经网络由 3 个全连接层组成，每个层有自己的参数。

编程实现： 可以用 TensorFlow、PyTorch、Keras 等深度学习标准库实现全连接神经网络，只需要一两行代码就能添加一个全连接层。添加一个全连接层需要手动指定两个超参数：

- **层的宽度**。如果一个层是隐层（即除了第 l 层之外的所有层），那么需要指定层的宽度（即输出向量的维度）。输出层（即第 l 层）的宽度由问题本身决定。比如 MNIST 数据集有 10 类，那么输出层的宽度必须是 10。而对于二元分类问题，输出层的宽度是 1。
- **激活函数**。用户需要决定每一层的激活函数。对于隐层，通常使用 ReLU 激活函数即可；使用其它激活函数或许能微弱地提高神经网络的表现。对于输出层，激活函数的选择要取决于具体问题。二元分类问题用 Sigmoid，多元分类问题用 Softmax，回归问题通常不用激活函数。

1.2.2 卷积神经网络

卷积神经网络 (Convolutional Neural Network)，缩写 CNN，是主要由卷积层组成的神经网络[2]。卷积神经网络的结构如图 1.9 所示。输入 $\boldsymbol{X}^{(0)}$ 是第三阶张量 (Tensor)[3]。卷积层的输入和输出都是第三阶张量，每个卷积层之后通常有一个 ReLU 激活函数（图 1.9 中没有画出）。可以把几个、甚至几十个卷积层累起来，得到深度卷积神经网络。把最后一个卷积层输出的张量做向量化 (Flatten)，得到一个向量。

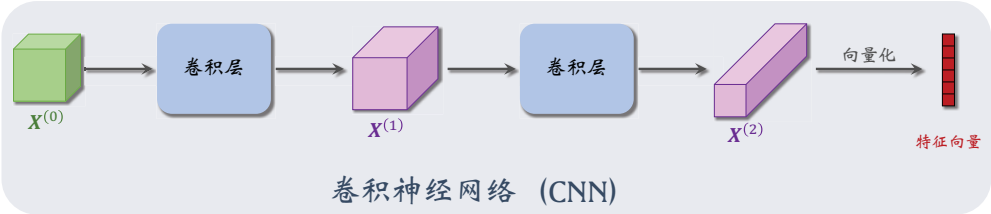

图 1.9: 图中的神经网络由 3 个全连接层组成，每个层有自己的参数。

本书不具体解释 CNN 的原理，本书也不会用到这些原理。读者仅需要记住这个知识

[2]CNN 中也可以有池化层 (Pooling)，本书不做讨论

[3]第零阶张量为标量（实数），第一阶张量为向量，第二阶张量为矩阵，以此类推。

点：卷积神经网络的输入是矩阵或三阶张量；卷积网络从张量提取特征，最终输出提取的特征向量。图片通常是矩阵（灰度图片）和三阶张量（彩色图片），可以用 CNN 从中提取特征，然后用一个或多个全连接层做分类或回归。

图 1.10 是一个由卷积、全连接等层组成的深度神经网络。其中的卷积网络从输入的矩阵（灰度图片）中提取特征，全连接网络把特征向量映射成 10 维的向量，最终的 Softmax 激活函数输出 10 维向量 $\hat{\boldsymbol{y}}$。输出向量 $\hat{\boldsymbol{y}}$ 的 10 个元素是 10 个类别的置信率，可以反映出分类结果。

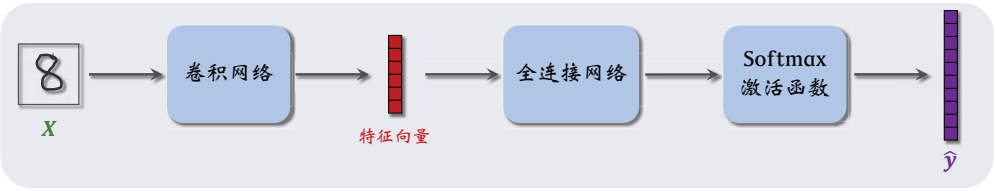

图 1.10: 用于分类 MNIST 手写数字的深度神经网络。

1.3 反向传播和梯度下降

线性模型和神经网络的训练都可以描述成一个优化问题。设 $\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)}$ 为优化变量（可以是向量、矩阵、张量）。我们希望求解这样一个优化问题：

$$\min_{\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)}} L\left(\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)}\right).$$

对于这样一个无约束的最小化问题，最常使用的算法是梯度下降 (Gradient Descent, 缩写 GD) 和随机梯度下降 (Stochastic Gradient Descent, 缩写 SGD) 。本节的内容包括梯度、梯度算法、以及用反向传播计算梯度。

1.3.1 梯度下降

梯度： 几乎所有常用的优化算法都需要计算梯度。目标函数 L 关于一个变量 $\boldsymbol{w}^{(i)}$ 的梯度记作：

$$\underbrace{\nabla_{\boldsymbol{w}^{(i)}} L\left(\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)}\right) \triangleq \frac{\partial L(\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)})}{\partial \boldsymbol{w}^{(i)}}}_{\text{两种符号都表示 } L \text{ 关于 } \boldsymbol{w}^{(l)} \text{ 的梯度}}, \qquad \forall\, i = 1, \cdots, l.$$

目标函数的值是标量（实数），所以梯度 $\nabla_{\boldsymbol{w}^{(i)}} L$ 的形状与 $\boldsymbol{w}^{(i)}$ 完全相同。

- 如果 $\boldsymbol{w}^{(i)}$ 是 $d \times 1$ 的向量，那么 $\nabla_{\boldsymbol{w}^{(i)}} L$ 也是 $d \times 1$ 的向量；
- 如果 $\boldsymbol{w}^{(i)}$ 是 $d_1 \times d_2$ 的矩阵，那么 $\nabla_{\boldsymbol{w}^{(i)}} L$ 也是 $d_1 \times d_2$ 的矩阵；
- 如果 $\boldsymbol{w}^{(i)}$ 是 $d_1 \times d_2 \times d_3$ 的第三阶张量，那么 $\nabla_{\boldsymbol{w}^{(i)}} L$ 也是 $d_1 \times d_2 \times d_3$ 的张量。

不论是自己手动推导梯度，还是用程序自动求梯度，都需要检查梯度的形状与变量的形状是否相同；如果不同，梯度的计算肯定有错。

梯度下降 (GD): 梯度是上升方向，沿着梯度方向对优化变量 $\boldsymbol{w}^{(i)}$ 做一小步更新，可以让目标函数值增加。既然我们的目标是最小化目标函数，就应该沿着梯度的反方向更新优化变量。沿着梯度反方向走就叫做梯度下降 (GD)。设当前的优化变量为 $\boldsymbol{w}_{\text{now}}^{(1)}, \cdots, \boldsymbol{w}_{\text{now}}^{(l)}$，计算目标函数 L 在当前的梯度，然后做 GD 更新优化变量：

$$\boldsymbol{w}_{\text{new}}^{(i)} \leftarrow \boldsymbol{w}_{\text{now}}^{(i)} - \alpha \cdot \nabla_{\boldsymbol{w}^{(i)}} L\left(\boldsymbol{w}_{\text{now}}^{(1)}, \cdots, \boldsymbol{w}_{\text{now}}^{(l)}\right), \qquad \forall\, i = 1, \cdots, l.$$

此处的 α (> 0) 叫做学习率 (Learning Rate) 或者步长 (Step Size)，它的设置既影响 GD 收敛速度，也影响最终神经网络的测试准确率，所以 α 需要用户仔细调整。

随机梯度下降 (SGD): 如果目标函数可以写成连加或者期望的形式，那么可以用随机梯度下降求解最小化问题。假设目标函数可以写成 n 项连加形式：

$$L\left(\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)}\right) = \frac{1}{n} \sum_{j=1}^{n} F_j\left(\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)}\right).$$

函数 F_j 隐含第 j 个训练样本 $(\boldsymbol{x}_j, \boldsymbol{y}_j)$。每次随机从集合 $\{1, 2, \cdots, n\}$ 中抽取一个整数，记作 j。设当前的优化变量为 $\boldsymbol{w}_{\text{now}}^{(1)}, \cdots, \boldsymbol{w}_{\text{now}}^{(l)}$，计算此处的随机梯度，并且做随机梯度下降：

$$\boldsymbol{w}_{\text{new}}^{(i)} \leftarrow \boldsymbol{w}_{\text{now}}^{(i)} - \alpha \cdot \underbrace{\nabla_{\boldsymbol{w}^{(i)}} F_j\left(\boldsymbol{w}_{\text{now}}^{(1)}, \cdots, \boldsymbol{w}_{\text{now}}^{(l)}\right)}_{\text{随机梯度}}, \qquad \forall\, i = 1, \cdots, l.$$

实际训练神经网络的时候，总是用 SGD（及其变体），而不用 GD。主要原因是 GD 用于非凸问题会卡在鞍点 (Saddle Point)，收敛不到局部最优，这会导致测试准确率很低；而 SGD 可以跳出鞍点，趋近局部最优。次要原因是 GD 每一步的计算量都很大，比 SGD 大 n 倍，所以 GD 通常很慢（除非用并行计算）。

SGD 的变体： 理论分析和实践都表明 SGD 的一些变体比简单的 SGD 收敛更快。这些变体都基于随机梯度，只是会对随机梯度做一些变换。常见的变体有 Momentum、AdaGrad、Adam、RMSProp。能用 SGD 的地方就能用这些变体。因此，本书中只用 SGD 讲解强化学习算法，不去具体讨论 SGD 的变体。

1.3.2 反向传播

随机梯度下降需要用到损失关于优化变量（即模型参数）的梯度。对于一个深度神经网络，需要用反向传播 (Backpropagation) 求损失函数关于变量的梯度。如果用 TensorFlow 和 PyTorch 等深度学习平台，你不需要关心梯度是如何求出来的。只要你定义的函数对某个变量可微，TensorFlow 和 PyTorch 就可以自动求该函数关于该变量的梯度。

本节以全连接网络为例，简单介绍反向传播的原理。全连接神经网络（忽略掉偏移量 b）是这样定义的：

$$\begin{aligned}
\text{第 1 层：} \quad & \boldsymbol{x}^{(1)} = \sigma_1\Big(\boldsymbol{W}^{(1)}\boldsymbol{x}^{(0)}\Big), \\
\text{第 2 层：} \quad & \boldsymbol{x}^{(2)} = \sigma_2\Big(\boldsymbol{W}^{(2)}\boldsymbol{x}^{(1)}\Big), \\
\vdots \quad & \quad \vdots \\
\text{第 } l \text{ 层：} \quad & \boldsymbol{x}^{(l)} = \sigma_l\Big(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)}\Big).
\end{aligned}$$

神经网络的输出 $\boldsymbol{x}^{(l)}$ 是神经网络做出的预测。设 h 为损失，比如

$$z = H\big(\boldsymbol{y}, \boldsymbol{x}^{(l)}\big),$$

其中函数 H 表示交叉熵，向量 $\boldsymbol{y}$ 表示真实标签。为了做梯度下降更新参数 $\boldsymbol{W}^{(1)}, \cdots, \boldsymbol{W}^{(l)}$，我们需要计算损失 z 关于每一个变量的梯度：

$$\frac{\partial z}{\partial \boldsymbol{W}^{(1)}}, \quad \frac{\partial z}{\partial \boldsymbol{W}^{(2)}}, \quad \cdots, \quad \frac{\partial z}{\partial \boldsymbol{W}^{(l)}}.$$

损失 z 与参数 $\boldsymbol{W}^{(1)}, \cdots, \boldsymbol{W}^{(l)}$、变量 $\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(l)}$ 的关系如图 1.11 所示。

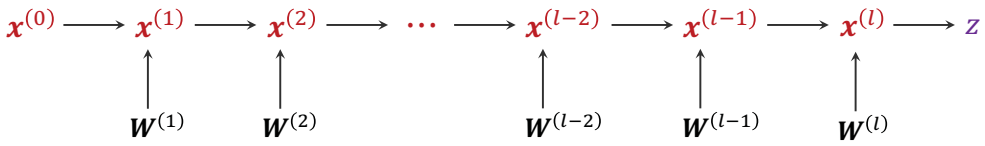

图 1.11: 变量的函数关系。

反向传播的本质是求导的链式法则 (Chain Rule)。设变量有这样的关系：$x \longrightarrow y \longrightarrow z$。那么可以用链式法则求出 z 关于 x 的偏导：

$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x} \cdot \frac{\partial z}{\partial y}.$$

同理，可以用链式法则做反向传播，得到损失关于神经网络参数的梯度。具体这样做。首先求出梯度 $\frac{\partial z}{\partial \boldsymbol{x}^{(l)}}$。然后做循环，从 $i=l,\cdots,1$，依次做如下操作：

- 根据链式法则可得损失 z 关于参数 $\boldsymbol{W}^{(i)}$ 的梯度：

$$\frac{\partial z}{\partial \boldsymbol{W}^{(i)}} = \frac{\partial \boldsymbol{x}^{(i)}}{\partial \boldsymbol{W}^{(i)}} \cdot \frac{\partial z}{\partial \boldsymbol{x}^{(i)}}.$$

  这项梯度被用于更新参数 $\boldsymbol{W}^{(i)}$。

- 根据链式法则可得损失 z 关于参数 $\boldsymbol{x}^{(i-1)}$ 的梯度：

$$\frac{\partial z}{\partial \boldsymbol{x}^{(i-1)}} = \frac{\partial \boldsymbol{x}^{(i)}}{\partial \boldsymbol{x}^{(i-1)}} \cdot \frac{\partial z}{\partial \boldsymbol{x}^{(i)}}.$$

  这项梯度被传播到下面一层（即第 $i-1$ 层），继续循环。

反向传播的路径如图 1.12 所示。只要知道损失 z 关于 $\boldsymbol{x}^{(i)}$ 的梯度，就能求出 z 关于 $\boldsymbol{W}^{(i)}$ 和 $\boldsymbol{x}^{(i-1)}$ 的梯度。

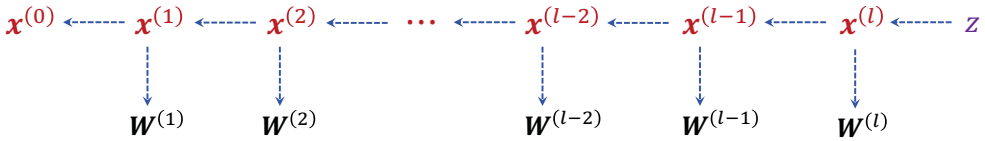

图 1.12: 反向传播的路径。