第 2 章 数学和 Python 基础知识

深度学习是一类结构复杂的数学模型，我们需要掌握一定的数学知识，特别是线性代数、微积分和概率论的相关知识，才能很好地学习和理解它。幸运的是，如果只需要理解深度学习的基本原理和应用，我们不需要精通这 3 门课程的所有知识，而只需要掌握这 3 门课程的一些核心概念和方法。这 3 门课程都是理工科学生的必修基础课，相信你在大学一二年级时都曾经学习过。在这里，为了方便随时复习和查阅相关知识，我们将介绍深度学习常用到的一些关键知识点。在本章中，我们也将介绍 Python 编程相关的基础知识，包括 Anaconda、Jupyter Notebook 和 Python 基础知识。

2.1 线性代数

2.1.1 数、向量、矩阵和张量

1. 数 （Scalar）：数是一个数字，通常指一个实数。例如，通常用自然数n表示数据的观测点个数。

2. 向量 （Vector）：把多个数字有序地放在一起称为向量。例如，写成如下形式的$\boldsymbol{x}$表示一个向量。

$$\boldsymbol{x} = (x_1 \quad x_2 \quad \cdots \quad x_n), \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

上面左边的向量叫做行向量；右边的向量叫做列向量。我们用**黑体小写字母**表示一个向量。向量里面的数字叫做向量的元素。在上面例子中，$x_1, x_2, \cdots, x_n$是向量$\boldsymbol{x}$的n个元素。向量的元素可以用对应的位置序号表示，例如，$\boldsymbol{x}$的第一个元素是x_1，第二个元素是x_2，等等。

3. 矩阵 （Matrix）：把多个数字写成二维的形式称为矩阵。例如，6 个数字写成如下形式表示一个3×2的矩阵。

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{pmatrix}$$

我们用**黑体大写字母**表示一个矩阵。矩阵行和列的数量称为矩阵的维度。在上面例子中，矩阵$\boldsymbol{X}$有 3 行，2 列，因此，$\boldsymbol{X}$的维度为3×2。当我们要强调矩阵的维度时，也会把矩阵$\boldsymbol{X}$写成$\boldsymbol{X}_{3\times2}$。矩阵元素可以用对应的行号和列

号表示，例如，$\boldsymbol{X}$的第 2 行，第 1 列的元素可以写成$\boldsymbol{X}_{21}$。矩阵第i行可以用$\boldsymbol{X}_{i:}$表示，矩阵第i列可以用$\boldsymbol{X}_{:i}$表示。有一类特殊的矩阵在运算中经常会用到，那就是单位矩阵（Identity Matrix）。首先，单位矩阵是一个方阵（方阵指行数和列数相等）。其次，单位矩阵左上角到右下角的主对角线上的元素都是 1，其他位置的元素都是 0。单位矩阵通常用$\boldsymbol{I}_n$（n表示方阵的维度）表示。例如，$\boldsymbol{I}_3$表示3×3的单位矩阵。

$$\boldsymbol{I}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4. 张量 (Tensor)：多个数字写成大于二维的形式称为张量。例如，24 个数字写成如图 2-1 的形式表示一个 3 个维度的张量。我们用**黑体大写字母**表示一个张量。在图 2-1 的例子中，$\boldsymbol{X}$的维度为$3 \times 4 \times 2$。有时候，我们也会把$\boldsymbol{X}$写成$\boldsymbol{X}_{3\times4\times2}$。张量的元素也可以用对应的维度表示，例如，$\boldsymbol{X}$的(2,2,2)元素是 14。特别的，矩阵是一个只有两个维度的张量。在图 2-1 的例子中，可以认为张量$\boldsymbol{X}$由两个3×4的矩阵组成。有时候，深度学习的数据需要用张量来表示。由谷歌开发的深度学习框架叫做 TensorFlow，直译就是“张量流”。

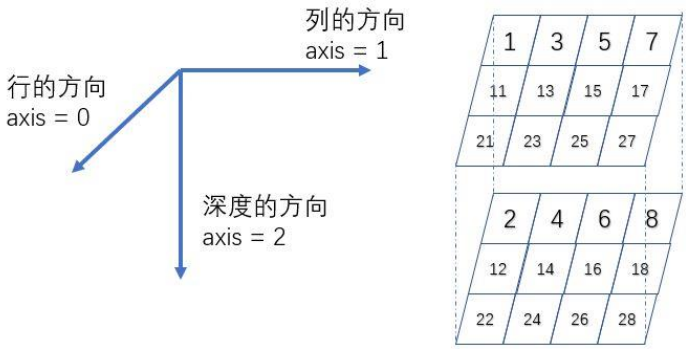

图 2-1 张量

2.1.2 矩阵的转置

矩阵最常用到的运算是矩阵的转置（Transpose）。矩阵$\boldsymbol{X}$的转置记为$\boldsymbol{X}^{\mathsf{T}}$（符号$^{\mathsf{T}}$是英文 Transpose 的首字母）。直观来看，矩阵的转置是矩阵的翻转，即$\boldsymbol{X}$的第一列变为$\boldsymbol{X}^{\mathsf{T}}$的第一行，$\boldsymbol{X}$的第二列变为$\boldsymbol{X}^{\mathsf{T}}$的第二行，等等。例如

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{pmatrix} \Rightarrow \boldsymbol{X}^{\mathsf{T}} = \begin{pmatrix} x_{11} & x_{21} & x_{31} \\ x_{12} & x_{22} & x_{32} \end{pmatrix}$$

因此，我们也可以得到

$$(\boldsymbol{X}^{\mathrm{T}})_{ij} = \boldsymbol{X}_{ji}$$

2.1.3 矩阵的基本运算

1. 矩阵加法：矩阵相加要求两个矩阵维度相同，矩阵对应元素相加。假设矩阵$\boldsymbol{A}$和$\boldsymbol{B}$的维度都是$n \times p$，$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$意味着$\boldsymbol{C}_{ij} = \boldsymbol{A}_{ij} + \boldsymbol{B}_{ij}$，$1 \le i \le n$且$1 \le j \le p$。例如

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix} + \begin{pmatrix} 3 & 1 \\ 2 & 5 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 3 \\ 6 & 10 \\ 8 & 9 \end{pmatrix}$$

   在深度学习中，还会用到一些在传统线性代数课程中不常用的加法运算。例如，矩阵$\boldsymbol{A}$和行向量$\boldsymbol{b}$的加法运算（要求矩阵$\boldsymbol{A}$的列数等于向量$\boldsymbol{b}$的元素个数）$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{b}$定义为：$\boldsymbol{C}_{ij} = \boldsymbol{A}_{ij} + \boldsymbol{b}_{j}$，$1 \le i \le n$且$1 \le j \le p$。例如

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix} + (2 \quad 3) = \begin{pmatrix} 3 & 5 \\ 6 & 8 \\ 9 & 11 \end{pmatrix}$$

2. 矩阵减法：矩阵相减要求两个矩阵维度相同，矩阵对应元素相减。假设矩阵$\boldsymbol{A}$和$\boldsymbol{B}$的维度都是$n \times p$，$\boldsymbol{C} = \boldsymbol{A} - \boldsymbol{B}$意味着$\boldsymbol{C}_{ij} = \boldsymbol{A}_{ij} - \boldsymbol{B}_{ij}$，$1 \le i \le n$且$1 \le j \le p$。例如

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix} - \begin{pmatrix} 3 & 1 \\ 2 & 5 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ 2 & 0 \\ 6 & 7 \end{pmatrix}$$

   矩阵$\boldsymbol{A}$减行向量$\boldsymbol{b}$（要求矩阵$\boldsymbol{A}$的列数等于向量$\boldsymbol{b}$的元素个数）$\boldsymbol{C} = \boldsymbol{A} - \boldsymbol{b}$定义为：$\boldsymbol{C}_{ij} = \boldsymbol{A}_{ij} - \boldsymbol{b}_{j}$，$1 \le i \le n$且$1 \le j \le p$。例如

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix} - (2 \quad 3) = \begin{pmatrix} -1 & -1 \\ 2 & 2 \\ 5 & 5 \end{pmatrix}$$

3. 矩阵乘法（一）：我们介绍的第一种矩阵乘法称为逐点相乘（英文称为Element-wise product 或者 Hadamard product），记为$\boldsymbol{C} = \boldsymbol{A} \circ \boldsymbol{B}$。该乘法要求两个矩阵维度相同，矩阵对应元素相乘。假设矩阵$\boldsymbol{A}$和$\boldsymbol{B}$的维度都是$n \times p$，$\boldsymbol{C} = \boldsymbol{A} \circ \boldsymbol{B}$意味着$\boldsymbol{C}_{ij} = \boldsymbol{A}_{ij} \times \boldsymbol{B}_{ij}$，$1 \le i \le n$且$1 \le j \le p$。例如

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix} \circ \begin{pmatrix} 3 & 1 \\ 2 & 5 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 8 & 25 \\ 7 & 8 \end{pmatrix}$$

4. 矩阵乘法（二）：该矩阵乘法是通常线性代数课程中介绍的矩阵乘法，记为$\boldsymbol{C}=\boldsymbol{AB}$。要求矩阵$\boldsymbol{A}$的列数等于矩阵$\boldsymbol{B}$的行数。如果矩阵$\boldsymbol{A}$的维度为$m\times n$, 矩阵$\boldsymbol{B}$的维度为$n\times p$，那么$\boldsymbol{A}$和$\boldsymbol{B}$相乘的结果$\boldsymbol{C}$的维度为$m\times p$。$\boldsymbol{C}$的$(i,j)$元素是$\boldsymbol{A}$的第$i$行与$\boldsymbol{B}$的第$j$列对应元素相乘，然后求和的结果，即$\boldsymbol{C}_{ij}=\sum_k A_{ik}B_{kj}$。例如

$$\begin{pmatrix}1&2\\4&5\\7&8\end{pmatrix}\begin{pmatrix}3&2&1\\1&5&1\end{pmatrix}=\begin{pmatrix}5&12&3\\17&33&9\\29&54&15\end{pmatrix}$$

5. 点乘（dot product）：点乘是两个元素个数相同的向量对应元素相乘，然后再相加的结果，记为$\boldsymbol{x}\cdot\boldsymbol{y}$。即$\boldsymbol{x}\cdot\boldsymbol{y}=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$，这里，$\boldsymbol{x}$和$\boldsymbol{y}$都为列向量。例如，$\boldsymbol{x}=(1\quad 2\quad 3)^{\mathrm{T}}$，$\boldsymbol{y}=(4\quad 5\quad 6)^{\mathrm{T}}$，那么$\boldsymbol{x}\cdot\boldsymbol{y}=1\times4+2\times5+3\times6=32$。矩阵的乘法（二），$\boldsymbol{C}=\boldsymbol{AB}$，也可以理解为，$\boldsymbol{C}$的$(i,j)$元素是$\boldsymbol{A}$的第$i$行与$\boldsymbol{B}$的第$j$列点乘的结果，即$\boldsymbol{C}_{ij}=\boldsymbol{A}_{i:}\cdot\boldsymbol{B}_{:j}$，如图 2-2 所示。

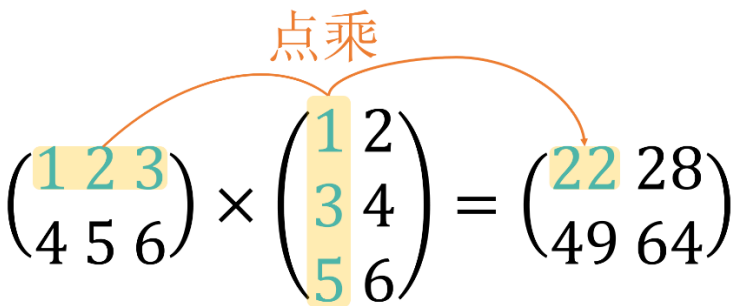

图 2-2 矩阵乘法与点乘

2.1.4 向量和矩阵的范数

我们可以通过范数（norm）衡量向量或者矩阵的大小。向量$\boldsymbol{x}$的L^p范数定义为

$$||\boldsymbol{x}||_p=\left(\sum_i|x_i|^p\right)^{\frac{1}{p}}$$

这里要求$p\geq1$。特别的，在深度学习中，我们常用到L^2范数$||\boldsymbol{x}||_2=\sqrt{\sum_i x_i{}^2}$。对于矩阵，在深度学习中，最常用的范数是 Frobenius 范数。矩阵$\boldsymbol{A}$的 Frobenius 范数定义为矩阵的所有元素的平方和，再求平方根。

$$||\boldsymbol{A}||_F=\sqrt{\sum_{ij}\boldsymbol{A}_{ij}^2}$$

2.2 微积分

微积分在深度学习中应用最多的概念是导数。因此，在这里，我们将主要介绍导数和求导法则。

2.2.1 导数的概念

函数$y = f(x)$在x_0处的导数记为$\frac{dy}{dx}$或者$f'(x)$，定义为

$$\frac{dy}{dx} = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

$f'(x)$为函数$f(x)$在x_0处切线的斜率，表示函数$f(x)$在x_0处的变化率。在图 2-3 中，实线表示$f(x) = x^2$的曲线，虚线表示$f(x)$在$x = -1,0,1.5$处的切线。

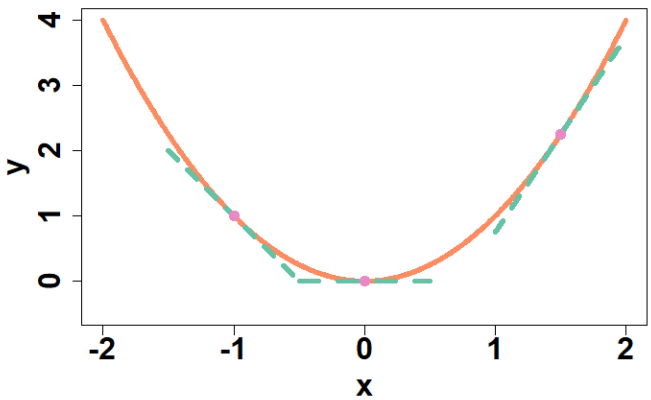

图 2-3 函数$f(x) = x^2$及其在$x = -1,0,1.5$处的导数

如果函数$z = f(x,y)$具有两个变量，可以把y看成固定常数，对x求导数，称为z关于x的偏导数，记为$\frac{\partial z}{\partial x}$; 也可以把$x$看成固定常数，对$y$求导数，称为$z$关于$y$的偏导数，记为$\frac{\partial z}{\partial y}$。

根据同样的方式，可以把偏导数的定义推广到有多个变量的函数情形。$f(x_1, x_2, \cdots, x_n)$关于x_i的偏导数为，固定$x_1, \cdots, x_{i-1}$和$x_{i+1}, \cdots, x_n$，函数对x_i求导，记为$\frac{\partial f(x_1, x_2, \cdots, x_n)}{\partial x_i}$。

2.2.2 求导法则

1. 函数加减法：如果$f(x) = u(x) \pm v(x)$，则$f'(x) = u'(x) \pm v'(x)$。
2. 函数乘法：如果$f(x) = u(x)v(x)$，则$f'(x) = u'(x)v(x) + u(x)v'(x)$。该计算方法也可以推广到任意有限个函数相乘的情形。例如，当$f(x) = u(x)v(x)w(x)$，则

$$f'(x) = u'(x)v(x)w(x) + u(x)v'(x)w(x) + u(x)v(x)w'(x)$$

3. 函数除法：如果$f(x) = \frac{u(x)}{v(x)}$，则

$$f'(x) = \frac{u'(x)v(x) - u(x)v'(x)}{v(x)^2}$$

4. 复合函数的求导（链式法则）：如果函数$y = f(u)$，$u = g(x)$，则复合函数$y = f(g(x))$关于x的导数为

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

2.3 概率论

解决现实问题时，深度学习需要处理带有不确定性特征的数据。例如，建立一个天气预报的模型。明天的天气情况本身是很多因素在未知机制作用下的结果，这些影响因素有些是已知并且可以测量的（但是，测量过程可能具有不确定性），有些却是未知的或者不能够测量的。深度学习在建模过程中需要考虑这些不确定的影响才能更好地做出预测。在这里，我们将简要介绍一些深度学习常用的概率论的概念。

2.3.1 随机变量

直观理解，随机变量（Random Variable）是一个取值不确定的变量。该变量的结果可能有多种，而现实中，我们不能确定哪个结果会出现。通常，随机变量记为大写字母X，而观测到的随机变量的值记为小写字母x。例如，考虑明天某只股票的价格。在今天的这个时点，明天某只股票的价格可能上涨，可能下跌或者不涨不跌。在今天，我们可以把明天的股价看成是随机变量，记为X。到了明天，我们观测到该股票的收盘价。这时，收盘价是随机变量X观测到的值，记为x。

随机变量可以分成两类，离散型随机变量和连续型随机变量。离散型随机变量可能的结果的数量是有限的，或者可数的。例如，掷一个骰子，可能的结果只有 6 种，1，2，3，4，5，6。连续型随机变量可能取到实数轴中某些区间的所有值。

2.3.2 随机变量的分布

随机变量的分布用来描述随机变量出现某种结果的可能性。随机变量的累积分布函数（Cumulative Distribution Function, CDF）记为$F(x)$，定义如下，

$$F(x) = \mathrm{P}(X \leq x)$$

离散型随机变量也可以用分布律（Probability Mass Function, PMF）表示，定义为

$$f(x) = \mathrm{P}(X = x)$$

连续型随机变量也可以用概率密度函数（Probability Density Function, PDF）表示，记为$f(x)$，且$f(x)$满足

$$F(x) = \int_{-\infty}^{x} f\,(t)dt, \quad x \in (-\infty, \infty)$$

对于离散型随机变量，$F(x)$与$f(x)$（PMF）等价地表示随机变量X的分布信息。如果已知随机变量X的 CDF，$F(x)$，可以得到随机变量X的 PMF，$f(x) = F(x) - F(x - \epsilon)$，这里$\epsilon$是一个很小的正数；反之，如果已知随机变量$X$的 PMF，$f(x)$，可以得到随机变量$X$的 CDF，$F(x) = \sum_{i:x_i \le x} f\,(x_i)$。

对于连续型随机变量，$F(x)$与$f(x)$（PDF）等价地表示随机变量X的分布信息。如果已知随机变量X的 CDF，$F(x)$，可以得到随机变量X的 PDF，$f(x) = F'(x)$；反之，如果已知随机变量X的 PDF，$f(x)$，可以得到随机变量X的 CDF，$F(x) = \int_{-\infty}^{x} f\,(t)dt$。累积分布函数和分布律或者概率密度函数的等价关系可以总结如表 2-1 所示。

表 2-1 累积分布函数和分布律或者概率密度函数的相互变换

	$F(x) \Rightarrow f(x)$	$f(x) \Rightarrow F(x)$
离散随机变量	$f(x) = F(x) - F(x - \epsilon)$	$F(x) = \sum_{i:x_i \le x} f\,(x_i)$
连续随机变量	$f(x) = F'(x)$	$F(x) = \int_{-\infty}^{x} f\,(t)dt$

2.3.3 常见的概率分布

1. 伯努利分布（Bernoulli Distribution），伯努利实验只有两个可能的取值，1 和 0。1 和 0 发生概率分别是p和$1 - p$，$0 \le p \le 1$。$X = 1$通常表示“成功”，p表示成功的概率；$X = 0$通常表示“失败”，$1 - p$表示失败的概率。伯努利分布的 PMF 为$f(1) = \mathrm{P}(X = 1) = p$，$f(0) = \mathrm{P}(X = 0) = 1 - p$，可以用表 2-2 表示。

表 2-2 伯努利分布的分布律

X	0	1
$f(x)$	$1 - p$	p

2. 多项分布（Multinomial Distribution），多项分布有n个不同的取值，{1,2,⋯,n}。第i个取值发生的可能性为p_i，$i = 1, \cdots, n$，且要求$\sum_{i=1}^{n} p_i = 1$。多项分布的 PMF 可以用表 2-3 表示。

表 2-3 多项分布的分布律

X	1	2	⋯	n
$f(x)$	p_1	p_2	⋯	p_n

3. 正态分布（Gaussian Distribution），也叫做高斯分布，是应用最为广泛的概率分布。正态分布记为$X \sim N(\mu, \sigma^2)$。正态分布的概率密度函数为

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

正态分布的两个参数μ和σ^2分别决定了正态分布的位置和分散程度，$\mu = \mathrm{E}(X)$和$\sigma^2 = \mathrm{Var}(X)$。图 2-4 的两条曲线分别表示均值为 0，标准差为 1 和 2 的正态分布的概率密度函数。函数$f(x)$最高点在μ处，$f(x)$在最高点处的值由σ^2决定，σ^2越大，$f(x)$在最高点处的值越小。随着$|x|$增大，$f(x)$变小。σ^2越大，$f(x)$变小的速度越慢，意味着随机变量X分散程度越高。

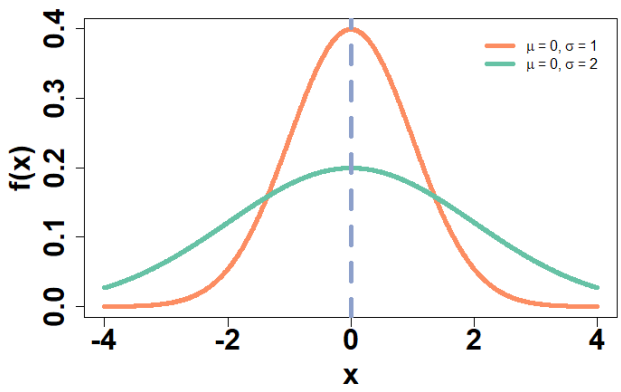

图 2-4 正态分布的概率密度函数

2.3.4 条件概率

很多情况下，我们会关心当 A 事件发生时，B 事件发生的概率。例如，在概率论这门课程中，同学们可能会关心，如果每周学习概率论 5 个小时（A 事件），期末成绩是A⁺（B 事件）的概率。这就是条件概率。条件概率记为$\mathrm{P}(Y = y|X = x)$，定义为

$$\mathrm{P}(Y = y|X = x) = \frac{\mathrm{P}(Y = y, X = x)}{\mathrm{P}(X = x)}$$

$P(Y=y|X=x)$表示随机变量X等于x时，Y等于y的概率，$P(Y=y,X=x)$表示$X=x$和$Y=y$同时发生的概率，$P(X=x)$表示$X=x$发生的概率。

2.4 Anaconda

当我们写 Python 代码或者使用其他人写的代码时，通常会使用特定的 Python 版本，例如，Python 2 或者 Python 3。我们也有可能使用某些特定版本的 Python 模块或者包。例如，在一个项目中使用 Python 2，Numpy（1.13.0）；而在另一个项目中使用 Python 3，Numpy（1.16.1）。但是，在同一个电脑中，同时安装不同版本的 Python 和不同版本的包不是一件容易的事情。Anaconda 可以帮助我们实现这一点。Anaconda 是一个免费开源的包和环境的管理器，可以实现在同一台计算机上安装不同版本的 Python 和包，并能够在不同的版本之间切换。Anaconda 的主要作用是，

- 虚拟环境管理：在 Anaconda 中可以建立多个虚拟环境，用于隔离不同项目所需的不同版本的软件或者工具包，以防止版本上的冲突。
- 包管理： 可以使用 Anaconda 安装、更新 和卸载包。

2.4.1 安装 Anaconda

Anaconda 支持多个系统平台，包括 Windows、Mac OS X 和 Linux。你可以从 Anaconda 官网下载适合自己系统的安装程序。我们建议下载 Python 3 的安装程序。虽然现在同时存在 Python 2 和 Python 3，但是我们相信 Python 3 会吸引越来越多的用户，而 Python 2 可能会慢慢的失去维护和更新。而且，本书使用 Python 3。如果计算机操作系统是 64 位的，最好选择 64 位的 Anaconda 安装程序。

在 Windowns 中，只需要双击扩展名为 exe 的文件（例如，Anaconda3-2018.12-Windows-x86_64.exe），然后一直点下一步，完成安装就可以了。完成安装后，打开 Anaconda Prompt，可以看到如图 2-5 所示的界面。

图 2-5 Anaconda Prompt 打开界面

在符号>后面输入

```
conda list
```

可以看到 Anaconda 自带的 Python 和所有包的版本号。安装文件自带的有些包可能不是最新版本的，我们可以在符号>后输入以下代码更新所有包。

```
conda upgrade --all
```

在运行上述命令过程中，可能需要输入 y 然后点击回车键，Anaconda 才会更新所有不是最新版本的包。

2.4.2 包的管理

1. 包的安装：conda install package_name。例如，安装 numpy 包：conda install numpy。
   - 同时安装多个包，例如，conda install numpy pandas。
   - 安装指定版本的包，例如，conda install pandas=0.46.4。
2. 包的删除：conda remove package_name。例如，删除 numpy 包：conda remove numpy。
3. 包的更新：conda update package_name。例如，更新 numpy 包：conda update numpy。
4. 列出当前环境中所有的包：conda list。

2.4.3 环境管理

Anaconda 通过环境实现在同一台计算机中安装不同版本的 Python 和包。基本原理是，Anaconda 首先创立环境，然后在该环境中安装特定版本的 Python 和包。不同环境是完全隔离的。刚打开 Anaconda Prompt 时，可以看到光标所在行的最左边括号里写着 base，表示处于 Anaconda 自带的 base 环境中。

1. 创建环境：conda create -n env_name package_names。例如，创建一个叫做 py37 的环境，安装最新版的 Python 3，同时还安装 pandas 和 numpy，命令如下：conda create -n py37 python=3 pandas numpy。这里-n 后面的 py37 是新环境的名字，同时我们还在该环境中安装 python=3，pandas，numpy。python=3 表示安装最新版的 Python 3。如果要在新环境中安装 Python 2，则可以用命令：conda create -n py37 python=2。我们还可以指定更加具体的 Python 版本，例如：conda create -n py37 python=3.5。
2. 列出已创建的所有环境：conda env list。
3. 进入或者退出环境。进入或者退出环境在不同系统平台略有不同。在 Windows 中，用 conda activate env_name 进入环境，用 conda deactivate 退出环境。例如，进入环境 py37， conda activate py37。这时可以看到光标所在行的最左边括号里写着 py37，表示现在处于环境 py37 中。

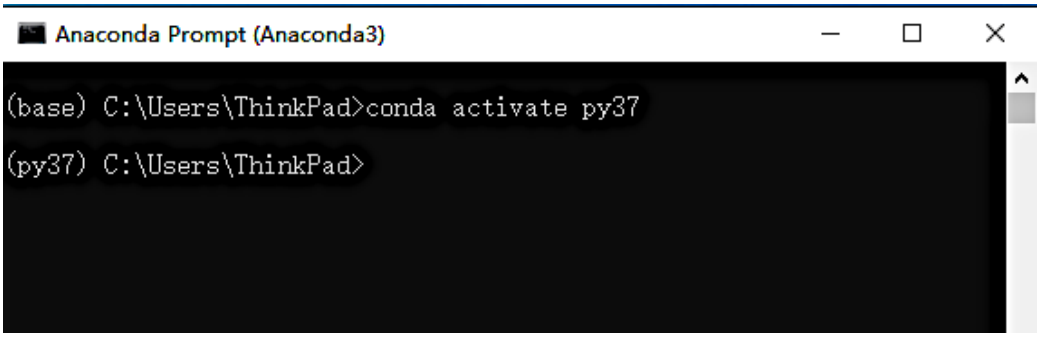

图 2-6 进入环境

4. 在环境中安装 Python 包和上述介绍的包安装方法一样。例如，在 py37 中安装 numpy，只需要进入 py37，然后输入 conda install numpy。
5. 删除环境：conda env remove -n env_name。例如，conda env remove -n py37 将删除刚刚创建的环境 py37。

2.5 Jupyter Notebook

Jupyter Notebook 是一个 Web 应用程序，便于创建和共享数据分析文档。我们可以使用 Jupyter Notebook 编辑代码，运行代码，查看结果，可视化数据，编辑说明文字、公式等。这些特性使其成为一款数据分析的便捷工具，可以用于数据清理、统计建模、构建和训练机器学习模型、可视化数据、以及大数据分析等。

2.5.1 安装 Jupyter Notebook

安装 Jupyter Notebook 最简单方式是使用 Anaconda 的 conda 命令，具体步骤如下，

1. 打开 Anaconda Prompt。
2. 使用 conda activate env_name 命令进入一个已经创建的环境（例如，进入环境 py37，可以在符号>后输入 conda activate py37）。
3. 输入 conda install jupyter notebook，Anaconda 会自动在环境中安装 Jupyter Notebook。

2.5.2 打开和关闭 Jupyter Notebook

在 Anaconda prompt 中输入 jupyter notebook，按回车便可以打开 Jupyter Notebook，如图 2-7 所示。

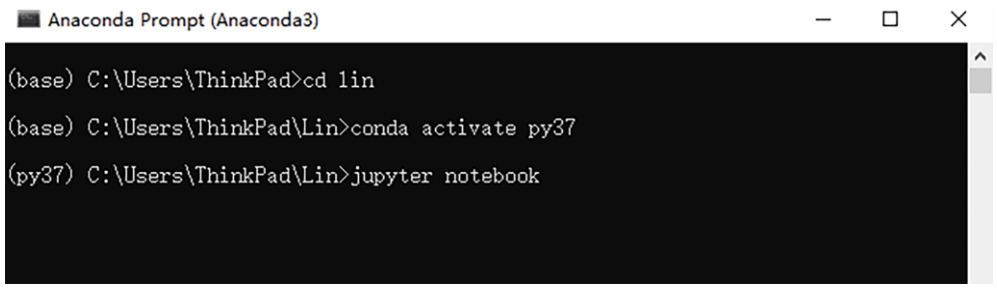

图 2-7 在 Anaconda 中打开 Jupyter Notebook

浏览器将打开如图 2-8 所示的网页。

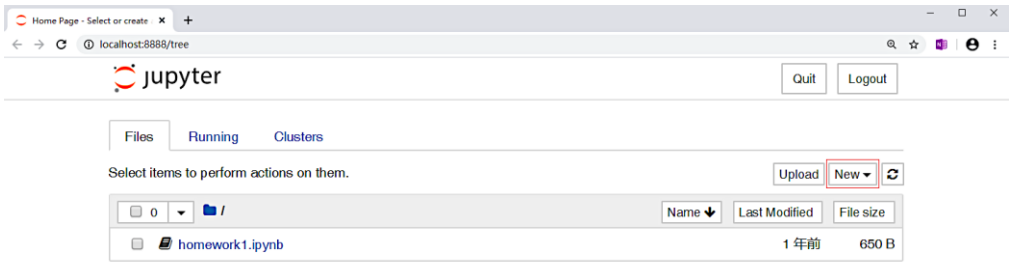

图 2-8 Jupyter Notebook 在浏览器中打开

现在可以打开现有的 Jupyter Notebook 文档（图 2-8 所示的电脑当前文件夹中，有一个 Jupyter Notebook 文档：howmework1.ipynb。Jupyter Notebook 文档的后缀是“.ipynb”），或者新建一个 Jupyter Notebook 文档。点击网页右上部分的 new（图 2-8 红色矩形框内），可以看到新建文件的类型，包括 Python 3, Text File 等。在 New 的菜单中，点击 Python 3 将创建一个运行 Python 3 的 Jupyter Notebook 文档，如图 2-9 所示。

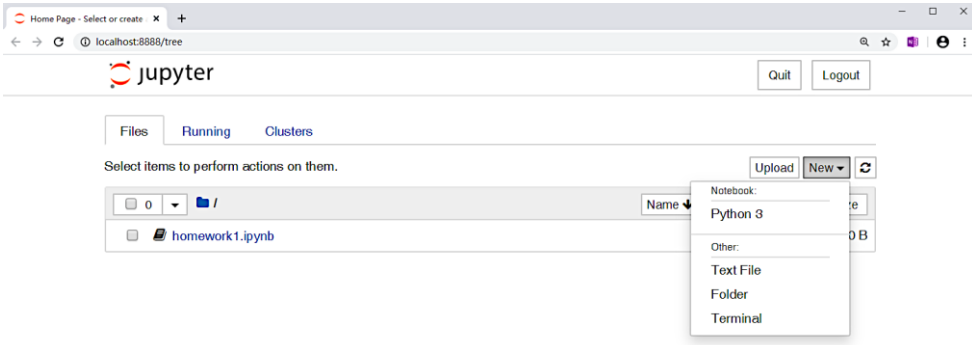

图 2-9 新建 Jupyter Notebook 文档

现在，一个新的 Jupyter Notebook 文件已经创建，显示为如图 2-10 所示的网页。

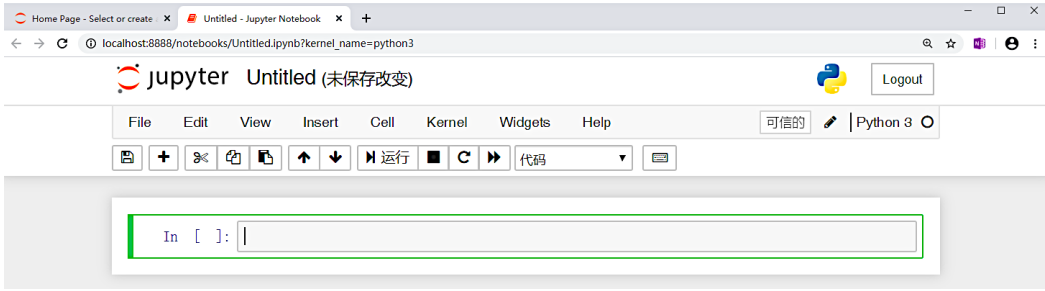

图 2-10 新建的 Jupyter Notebook 文档

新建的 Jupyter Notebook 文档的文件名都会以 Untitled 开头，第一个新建文档的文件名是“Untitled.ipynb”，再新建一个 Jupyter Notebook 文档，则文件名是“Untitled1.ipynb”，等等。可以点击 File->rename 更改文件名，如图 2-11 所示。

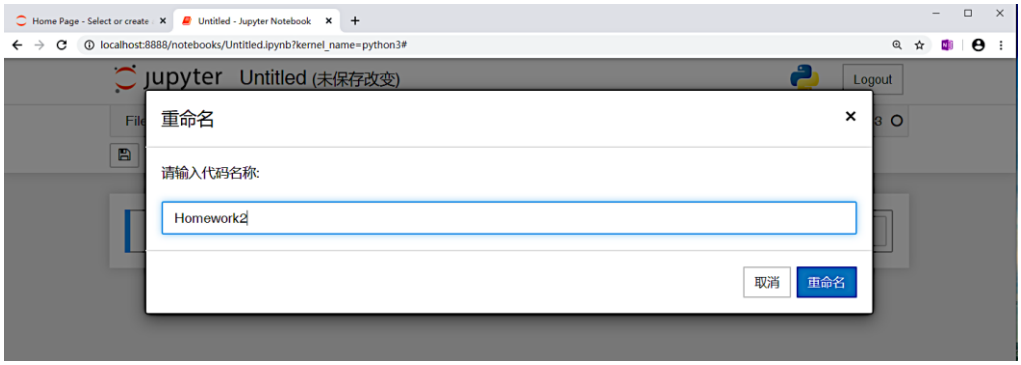

图 2-11 重命名 Jupyter Notebook 文档

当我们完成 Jupyter Notebook 文件的编辑后，可以点击网页左上角的保存按钮保存文档（位于左上菜单栏 File 下面）。然后，可以直接关闭网页，在 Anaconda 中连按两次 Ctrl+C 完全关闭 Jupyter Notebook。

2.5.3 代码框

代码框（Code Cell）是 Jupyter Notebook 用于写代码的地方。在代码框中，可以写任何 Python 代码，包括赋值、函数、类、画图等。在图 2-13 中，左边有蓝色竖线的框就是代码框，在红色矩形框处可以看到“代码”两个字。我们在代码框中载入 numpy 模块，运行 `2+3`，并赋值给变量 `a`，最后用函数 `print()` 输出结果。

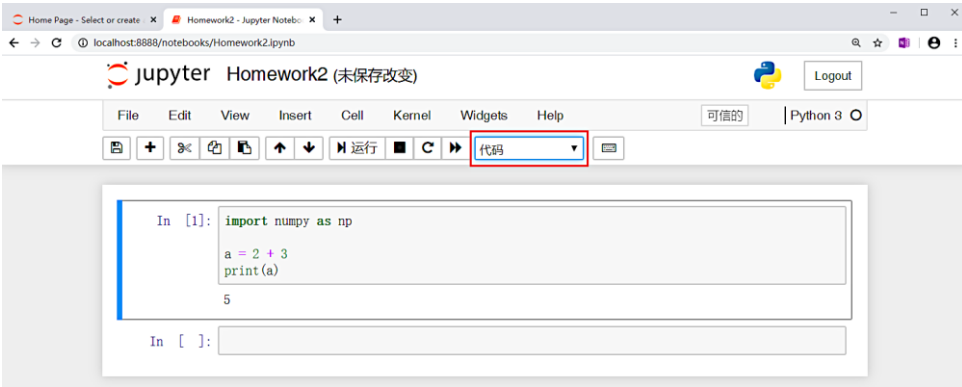

图 2-12 Jupyter Notebook 的代码框（命令模式）

在 Jupyter Notebook 中，无论是代码框还是将要介绍的标记框，都有两种模式。一种是命令模式，一种是编辑模式。命令模型左边有蓝色竖线，编辑模式左边有绿色竖线。图 2-12 的代码框处于命令模式。当一个框处于命令模式，双击鼠标或者点击回车键可以进入编辑模式，左边竖线变成绿色，如图 2-13 所示。

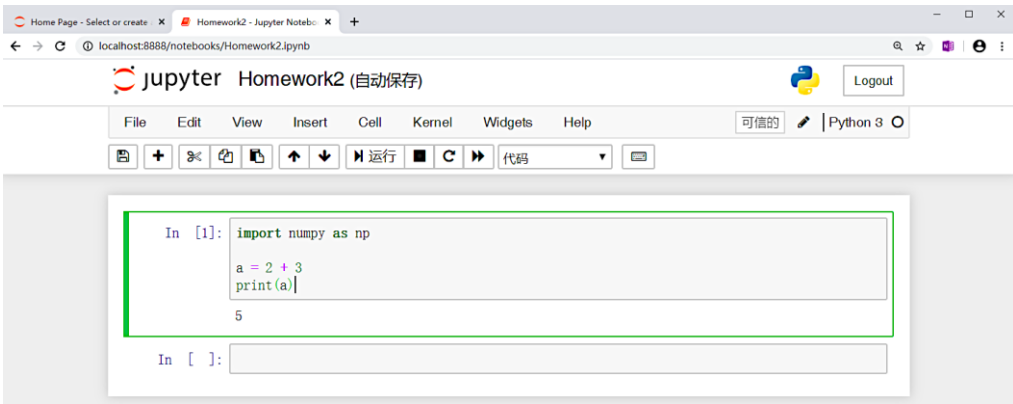

图 2-13 Jupyter Notebook 的编辑模式

这时可以在框中编辑代码。处于编辑模式时，点击键盘上的 Esc 键可以进入命令模式。点击 Shift+Enter 或者 Ctrl+Enter 可以运行代码框的代码或者编译标记框的文字等内容。Shift+Enter 和 Ctrl+Enter 的区别在于，Shift+Enter 运行代码或者编译标记框，然后自动进入下面一个代码框或者标记框；而 Ctrl+Enter 只是运行代码或者编译标记框。

2.5.4 标记框

标记框（Markdown Cell）可以用于编辑文字说明，包括普通中英文、标题、数学公式、图片等。Jupyter Notebook 采用的是 Markdown 的语法。下面将介绍常用的一些语法，包括标题、强调、列表、链接、图片、表格和数学公式。

1. 标题：在标记框中，标题以#开始，Nupyter Notebook 支持 6 级标题。

```
# Header 1
## Header 2
### Header 3
#### Header 4
##### Header 5
###### Header 6
```

上面的输入编译后的结果如下：

Header 1
Header 2
Header 3
Header 4
Header 5
Header 6

2. 强调：可以给文字加粗，变为斜体或者中间加横线 。

```
Jupyter Notebook 是一个很强大的工具，可以用单个下划线`_`或者`*`把文字变成_斜体_或者 *斜体*
也可以用两个下划线`__`或者`**`把文字变成__粗体__或者**粗体**
还可以用`~~`在文字中间加~~横线~~
```

上面的输入编译后的结果如下：

Jupyter Notebook 是一个很强大的工具，可以用单个下划线_或者*把文字变成*斜体*或者*斜体*
也可以用两个下划线__或者**把文字变成**粗体**或者**粗体**
还可以用~~在文字中间加~~横线~~

3. 无序列表：可以添加无序的项目符号。

```
- 无序号的列表，可以用减号-
* 或者星号*
+ 或者加号+开头
+ 注意：符号-，*或者+之后需要有一个空格
```

上面的输入编译后的结果如下：

- 无序号的列表，可以用减号-
- 或者星号*
- 或者加号+开头
- 注意：符号-，*或者+之后需要有一个空格

4. 有序列表：添加有序的项目符号。

```
1. 有序号的列表，可以以数字加.表示
1. 具体的数字是什么并不重要
1. Jupyter Notebook 会自动按顺序变成 1,2,3...
```

上面的输入编译后的结果如下：

```
1. 有序号的列表，可以以数字加.表示
2. 具体的数字是什么并不重要
3. Jupyter Notebook 会自动按顺序变成 1,2,3...
```

5. 链接：在 Jupyter Notebook 中，创建链接只需要把文字放在中括号内，把网址放在紧接着的圆括号内。例如

```
[异步社区]( https://www.epubit.com/)
```

呈现的结果如下

异步社区

6. 图片：在标记框内插入图片可以用如下两种方式。例如，我们要插入的图片是“insert_figs.png”，该图片保存在文件夹“figs”中。第一种方式是感叹号+中括号（中括号内可以写图片的标题）+小括号（小括号内写图片的路径和名称）；第二种方式是用尖括号，尖括号内部写 img+src（src 等于图片的路径和名称）+图片的宽度。

```
![fig_title](figs/insert_figs.png)
<img src="figs/insert_figs.png" width=400>
```

7. 表格：表格用竖线|分割每一列，用多个减号---分割列名和其他行。例如

```
| 第一列   | 第二列   | 第三列 |
| ------ |--------| -----|
| 第一行      | 1    |   2 |
| 第二行      | 3    |   4 |
| 第三行      | 5    |   6 |
```

呈现的结果如表 2-4 所示。

表 2-4 Jupyter Notebook 生成的表格

第一列	第二列	第三列
第一行	1	2
第二行	3	4
第三行	5	6

8. 数学公式：文字中的数学公式使用单个符号$，例如，$\beta$编译为$\beta$。单独一行且居中的公式使用符号$$，例如

```
$$
\mathbf{y}=\mathbf{X}\boldsymbol{\beta}+\boldsymbol{\epsilon}
$$
```

编译为

$$\boldsymbol{y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon}$$

2.6 Python

下面将简要介绍本书用到的 Python 功能，包括基本数据结构、控制结构、函数、画图和两个数据分析常用的包（Numpy 和 Pandas）。

2.6.1 Python 基础

Python 常用的数据类型有：整型（int）、浮点型（float）、 字符串（string）和布尔型（bool）。函数 type()可以返回数据类型。

```
# 查看数据类型
type(3)        # returns 'int'
type(3.0)      # returns 'float'
type('three') # returns 'str'
type(True)     # returns 'bool'
type(None)     # returns 'NoneType'

# 转换数据类型
float(3)
int(3.8)
str(33)
```

常用的 Python 基本运算有加（+）、减（-）、乘（*）、除（/）、乘方（**）、求余（%）。

```
12 + 5   # 加 (returns 17)
12 - 5   # 减 (returns 7)
12 * 5   # 乘 (returns 60)
12 / 5   # 除 (returns 2.4)
12 ** 5  # 乘方 (returns 248832)
12 % 5  # 求余 (returns 2)
```

Python 的比较运算包括大于（>）、大于等于（>=）、小于（<）、小于等于（<=）、不等于（!=）、相等（==）。 Python 的布尔运算包括与（and）、或（or）、非（not）。

```
#比较
6 < 4    # returns False
6 <= 4   # returns False
6 > 4    # returns True
6 >= 4   # returns True
6 != 4   # returns True
4 == 4   # returns True

# 布尔运算
6 > 4 and 2 > 3  # returns False
6 > 4 or 6 < 4   # returns  True
```

```
not False;        # returns True
```

Python 加载模块（module）或者包（package）。每次打开 Python 时，Python 只会载入少量必需的模块，这时 Python 可以实现一些基本功能。如果要拓展 Python 的功能，就需要载入额外的模块或者包。这样做的好处是，安装和打开 Python 时不会占用太多的计算资源和内存空间，而我们可以随时安装和加载所需要的模块或者包，使得 Python 可以很好地满足我们数据分析的需求。

Python 模块和包是两个不同的概念，其区别在于，模块是以.py 结尾的一个 Python 文件，包是包含多个模块的文件夹（在这个文件中还有一些其他的文件，如__init__.py，使得文件夹中的模块成为一个整体）。Python 加载模块或者包有如下方式。

```
import math                              # 加载 math 模块
math.sqrt(100)                           # 使用方式是：模块名.函数名

from math import sqrt                    # 从 math 模块加载函数 sqrt
sqrt(100)                                # 这时不需要写模块名

from math import cos, floor              # 从 math 模块加载函数 cos 和 floor
cos(10)
floor(2.8)

# 从 os 模块中载入所有的函数(不推荐，这样容易发生函数混淆)
from os import *

import numpy as np                       # 加载 numpy 模块，并命名为“np”
np.sqrt(100)
```

2.6.2 Python 基本数据结构

1. 列表

列表（List）是 Python 最常用的数据结构，可以容纳任何数据类型，如浮点型、字符串、布尔型等。列表的数据项不需要具有相同的类型。创建一个列表，只要用逗号分隔不同元素，并放在方括号中，如下所示。

```
num = [3, 2, 1, 4, 2015]
string = ["lin", "yi"]
mixed_num_string = ["lin", 2015, "yi", 8]
```

列表是有序的，其索引从 0 开始，依次递增 1。例如，列表 mixed_num_string 各个元素的索引如表 2-5 所示。

表 2-5 列表的索引

列表元素	“lin”	2015	“yi”	8
索引	0	1	2	3

可以如下用列表索引访问列表元素。

```
mixed_num_string[2]    # 返回索引为 2 的元素
mixed_num_string[0:2] # 返回索引为 0,1 的元素，注：不包括索引为 2 的元素
mixed_num_string[-1]   # 返回列表最后 1 个元素
mixed_num_string[-2:] # 返回列表最后 2 个元素
```

函数 append()，函数 extend()，函数 pop()是 3 个常用的添加和删除列表元素的函数。

- 函数 append()：在列表后面增加一个元素。
- 函数 extend()：合并两个列表。
- 函数 pop()：移除列表最后一个元素。

```
num[1] = 200                # 把索引为 1 的元素赋值为 200
num
[3, 200, 1, 4, 2015]

num.append(1024)
num
[3, 200, 1, 4, 2015, 1024]

num.append(string)          # 把列表 string 当成一个元素添加到列表 num 中
num
[3, 200, 1, 4, 2015, 1024, ['lin', 'yi']]

num.pop()                   # 从列表 num 中，移除最后一个元素
num
[3, 200, 1, 4, 2015, 1024]

num.extend(string)          # 合并列表 num 和列表 string
num
[3, 200, 1, 4, 2015, 1024, 'lin', 'yi']
```

函数 len()可以返回 list 的元素个数。在 list 中，需要注意，赋值只是给 list 起了一个新的名字。例如

```
len(num)                # 返回 num 的元素个数
8

string2 = string        # 相当于给列表 string 起了一个新的名字
string2[1] = "hh"       # 更改 string2 的元素也会更改 string 的元素
string
['lin', 'hh']

string3 = string[:]     # 复制列表 string，并把复制的列表赋值给 string3
string3[1] = "hello"    # 更改 string3 的元素不会更改 string 的元素
string
['lin', 'hh']
```

2. 元组

Python 的元组（Tuple）与列表类似，不同之处在于：元组的元素不能修改，元组使用小括号，列表使用方括号。元组创建很简单，只需要在小括号中添加元素，并使用逗号隔开（括号也可以省略）。

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5 )
tup3 = "a", "b", "c", "d"       # 省略括号
```

元组使用索引访问元素的方式和列表相同。

```
tup1[2]                         # returns 1997
tup1[0:2]                       # returns ('physics', 'chemistry')
# 注意
# 以下修改元组元素操作是非法的。
tup1[0] = 100                   # 非法的操作
```

3. 字典

字典（Dictionary）的元素由键（key）和值（value）成对组成。字典也被称作关联数组或哈希表，基本语法如下。

```
dict = {'yi': 2015, 'lin': 1984, 'luo': 1985}
```

键与值用冒号隔开，每对键和值用逗号分割，整体放在花括号{}中。键必须独一无二，但值则不必。字典是无序的数据结构，因此不能通过索引号访问字典元素。在字典里，可以通过键访问值。

```
dict = {'yi': 2015, 'lin': 1984, 'luo': 1985}
dict['yi']     # returns 2015
len(dict)      # returns 3
dict.keys()    # returns ['yi', 'lin', 'luo']
dict.values()  # returns [2015, 1984, 1985]

# returns [('yi', 2015), ('lin', 1984),('luo', 1985)]
dict.items()
```

4. 集合

Python 集合（set）与数学中集合的概念类似，是指由不同元素组成的合集。下面代码可以从列表中创建一个集合。

```
s_list = [3,1,1,4,5,5,6,6]
s = set(s_list)
s
{1, 3, 4, 5, 6}
```

函数 `add()` 可以给集合添加新的元素。

```
s.add(7)
s
{1, 3, 4, 5, 6, 7}
```

2.6.3 控制结构和函数

1. 控制结构

Python 的控制结构可以控制代码的运行顺序。控制结构中，缩进是非常重要的，标识了代码的结构。缩进可以通过 4 个空格或者 1 个制表符（tab）实现。

（1）for 循环和 while 循环。for 循环的结构如下。

```
for each_item in list:
    do something to
    each_item
```

需要注意的是，在 for each_item in list:这行中，最后一定有一个冒号“:”；冒号后面属于 for 循环的行都要有 4 个空格的缩进。在 for 循环中，each_item 首先等于 list 的第一个元素，执行冒号后面的代码；接着 each_item 等于 list 的第二个元素，执行冒号后面的代码；直到 each_item 等于 list 的最后一个元素，执行冒号后面的代码；结束 for 循环。以下代码实现列表中所有元素求和。

```
a = [1,2,3,4,5,6]
sum_a = 0
for each in a:
    sum_a += each
    print(each)
sum_a
1
2
3
4
5
6

21
```

while 循环的结构如下。

```
while statement:
    do something
    do more
```

在 while 循环中，statement 是一个判断，结果为 True 或者 False，如果 statement 的结果为 True，则执行冒号后面的代码，如果 statement 的结果为 False，则结束循环。在语句 while statement:后面有冒号，属于 while 循环的代码需要 4 个空格的缩进。以下代码使用 while 循环实现列表中所有元素求和。

```
a = [1,2,3,4,5,6]
sum_a = 0
i = 0
while i<len(a):
```

```
    sum_a += a[i]
    print(a[i])
    i += 1
sum_a
1
2
3
4
5
6

21
```

（2）if 语句：if 语句的结构如下。

```
if statement1:
    do first_job
elif statement2:
    do second_job
else:
    do third_job
```

如果 statement1 的结果为 True，则执行 statement1:后面缩进 4 个空格的代码；如果 statement1 的结果为 False，则运行 statement2，如果 statement2 的结果为 True，则执行 statement2:后面缩进 4 个空格的代码；如果 statement2 的结果为 False, 则执行 else:后面缩进 4 个空格的代码。在 Python 中，elif 表示 else if。以下代码可以把列表 a 的奇数和偶数分别保存在不同列表中。

```
a = [1,2,3,4,5,6]
odd_num = []
even_num = []
for each in a:
    if each % 2 != 0 :
        odd_num.append(each)
    else:
        even_num.append(each)

print("Odd numbers in list a: " + str(odd_num))
print("Even numbers in list a: " + str(even_num))
Odd numbers in list a: [1, 3, 5]
Even numbers in list a: [2, 4, 6]
```

2. 自定义函数

函数能提高代码的模块性和重复利用率。Python 提供了许多内建函数，如 len()，print()等。我们也可以自定义函数。函数定义的规则： 函数代码块以关键词 def 开头，后接函数名称和圆括号()，传入函数的参数放在圆括号内，函数内容以冒号起始，并且缩进，若有返回值，使用 return 返回结果，结束函数。函数的结构如下：

```
def fun_name(arg1, arg2):
    do something
```

```
    return result1, result2
```

下面创建一个函数，函数名为“odd_even”。函数的任务是输入一个列表，输出列表的奇数和偶数。

```
def odd_even(ls):
    odd_num = []
    even_num = []
    for each in ls:
        if each % 2 != 0 :
            odd_num.append(each)
        else:
            even_num.append(each)
    return odd_num, even_num
a1 = [3,4,5,6,10,11,2,3]
odd_num_a1, even_num_a1 = odd_even(a1)
print("Odd numbers in list a1: " + str(odd_num_a1))
print("Even numbers in list a1: " + str(even_num_a1))

Odd numbers in list a1: [3, 5, 11, 3]
Even numbers in list a1: [4, 6, 10, 2]

a2 = [1,1,1,2,2,2]
odd_num_a2, even_num_a2 = odd_even(a2)
print("Odd numbers in list a2: " + str(odd_num_a2))
print("Even numbers in list a2: " + str(even_num_a2))

Odd numbers in list a2: [1, 1, 1]
Even numbers in list a2: [2, 2, 2]
```

2.6.4 Numpy

Python 是非常强大的计算机语言，可以完成很多复杂的任务。不过，Python 本身对数学运算的支持并不是很好。在使用标准的 Python 时，进行矩阵和向量的运算都需要使用循环语句，实现过程比较复杂，计算速度也较慢。Numpy 可以实现快速的数学运算，特别是矩阵运算。Numpy 提供了大量矩阵运算函数。而且 Numpy 的内部运算通过 C 语言而不是 Python 实现，使得它具有快速运算能力。Numpy 包含了两种基本数据类型：数组（array）和矩阵（matrix）。数组和矩阵的运算稍有不同，在这里，我们将着重介绍数组的使用。使用 Numpy 前，需要先加载 Numpy 包。

```
import numpy as np
```

1. 创建数组

可以使用函数 np.array()创建一个数组。

```
a = np.array([2,3,4])
a
array([2, 3, 4])
```

上面创建的数组 a 相当于一个向量。我们可以使用函数 a.shape()看 a 的维度。a 的维度为(3,0)，表示 a 是一个长度为 3 的数组。

```
a.shape
(3,)

b = np.array([[1,2,3],[4,5,6]])
b
array([[1, 2, 3],
       [4, 5, 6]])

b.shape
(2, 3)
```

上面创建的数组 b 的维度是(2,3)，表示 b 是一个 2 行，3 列的数组。数组的维度可以使用函数 reshape()变换，例如，

```
# 返回一个数组, array([0,1,2,3,4,5,6,7,8,9,10,11])
c = np.arange(12)
d = c.reshape(3,4)   # 返回一个维度为(3,4)的数组
d.reshape(2,6)       # 返回一个维度为(2,6)的数组
```

通过输入列表，利用函数 np.array()可以创建数组。Numpy 也可以产生经常用到的特殊数组。例如，函数 np.arange()可以产生在某个区间内步长相等的数组，函数 np.zeros()可以产生元素全是 0 的数组，函数 np.ones()可以产生元素全是 1 的数组，函数 np.eye()可以产生对角线是 1，其他元素都是 0 的二维数组。例如，

```
np.arange(4)        # 返回元素是 0,1,2,3 的数组
np.arange(4,8)      # 返回元素是 4,5,6,7 的数组

np.zeros(3)         # 返回元素是 0,0,0 的数组
np.zeros((2,3))     # 返回一个维度是(2,3),元素都是 0 的数组

np.ones(2)          # 返回元素是 1,1 的数组
np.ones((2,3))      # 返回一个维度是(2,3),元素都是 1 的数组

np.eye(3)           # 返回一个对角线是 1，其他元素都是 0, 维度(3,3)的数组
```

2. 数组的运算

两个维度相同的 Numpy 数组的加（+）、减（-）、乘（*）、 除（/）表示两个数组相同位置元素的加（+）、减（-）、乘（*）、 除（/）。

```
a = np.array([[1,2,3],[4,5,6]])
b = np.array([[4,5,6],[1,2,3]])
a + b
a - b
a * b
a / b
```

当数组维度不同时，有时候也可以计算。例如，a + 2 表示 a 的所有元素加 2， a + np.array([[3,2,1]]) 表示 a 的每一行加上维度(1,3)的数组

`np.array([[3,2,1]])`， `a + np.array([[3],[2]])`表示 `a` 的每一列加上维度(2,1)的数组 `np.array([[3],[2])`。但是，`a + np.array([[3,2]])`则会报错。

```
a + 2
a + np.array([[3,2,1]])
a + np.array([[3],[2]])
```

有的运算对数组每个元素分别计算，例如，函数 `np.sqrt()`对数组的每个元素求开平方根，函数 `np.log()`对数组的每个元素求自然对数。

```
np.sqrt(a)
np.log(a)
```

在 Numpy 中，矩阵相乘（二）有 3 种实现方式，函数 `np.dot()`、函数 `np.matmul()`和二元运算符`@`。三种方式得到同样的结果。

```
a = np.arange(1,7).reshape(2,3)
b = np.arange(7,13).reshape(3,2)

a @ b
np.dot(a, b)
np.matmul(a, b)
```

3. 数组的元素访问

我们可以很容易访问数组的行、列或者单独的元素。冒号“:”表示某一行或者某一列的所有元素。

```
a = np.arange(12).reshape(3,4)   # 创建维度为(3, 4)的数组
a[0,:]                           # 数组的第一行
a[0]                             # 数组的第一行
a[0:2]                           # 数组的前两行
a[:,0]                           # 数组的第一列
a[0:2, 0:2]                      # 数组前两行,前两列
a[1,1]                           # 数组第 2 行，第 2 列
```

2.6.5 Pandas

Pandas 是基于 NumPy 的一个开源 Python 包，它被广泛用于数据分析，数据清洗和准备等工作。它的名字由“ Panel data”两个单词拼成。Pandas 数据结构主要有两种：序列（Series）和数据表（DataFrame）。

```
import pandas as pd
```

1. 创建 Series 和 DataFrame

Pandas 的序列（Series）是一维数组，和 NumPy 的数组相似。和 NumPy 数组不同的是，Series 能为数据自定义标签，也就是索引（index），然后通过索引访问数组中的数据。例如，可以用一个序列表示一个班级同学的数学成绩，index 是同学的名字。Pandas 的数据表（DataFrame）是二维数据表，数据以表格的形式存储，分成若干行和列。通过 DataFrame，能够方便地处理数据。例如，

可以用一个数据表表示一个班级同学的多维信息，第一列表示数学成绩，第二列表示性别。数据表的每一列数据类型是一样的，不同列的数据类型可以不一样。

```
# 当代码太长，我们可以使用反斜杠"\"把代码写在两行或者多行
math_score = pd.Series([80, 20, 50, 60, 70], \
                       index = ["lin", "luo", "pan", "li", "wang"])
math_score["lin"]                # 返回 lin 同学的数学成绩
math_score[["lin", "luo"]];      # 返回 lin 和 luo 两位同学的数学成绩
class_info = pd.DataFrame({   \
             "math_score" : [80, 90, 50, 60, 70], \
             "gender" : ["male", "female", "female", "female",
             "male"]}, \
              index = ["lin", "luo", "pan", "li", "wang"])
class_info
```

表 2-5 创建数据表

	math_score	gender
lin	80	male
luo	90	female
pan	50	female
li	60	female
wang	70	male

2. 数据表元素的访问

上面列表中`['lin', 'luo', 'pan', 'li', 'wang']`可以看成是行的名字，`['math_score', 'gender']`可以看成是列的名字。在数据表中，我们可以通过行或者列名字，或者数字索引来访问数据表的元素。

```
class_info["math_score"]    # 返回数据表中"math_score"对应的列
class_info[0:2]            # 返回前两行
```

使用`.loc` 可以方便地通过行或者列名字访问数据表。使用`.iloc` 可以方便地通过数字索引访问数据表。例如，

```
class_info.loc[["lin", "luo"]]              # 返回数据表的前两行
# 返回数据表前两行，"gender"列的元素
class_info.loc[["lin", "luo"], "gender"]
class_info.iloc[2:4]                        # 返回数据表的 3,4 行
class_info.iloc[:,1];                       # 返回数据表的第 2 行
```

3. 数据表的基本探索

我们可以通过观察数据表，以及一些统计量了解数据表数据的特征。例如，

```
class_info.head()         # 返回前 6 行
class_info.tail()         # 返回后 6 行
class_info.describe()    # 返回均值及一些分位数
```

2.6.6 画图

在 Python 中，Matplotlib 包是最流行的画图工具。Matplotlib 能让使用者很轻松地将数据图形化，兼具灵活性和易操作性。需要载入 Matplotlib 包的 pyplot。

```
# 在 jupyter notebook 中，百分号"%"开始的命令叫做魔法命令
# 下面的魔法命令可以使图片分辨率更高
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt        # 载入 Matplotlib 的 pyplot
```

函数 `plt.plot()` 可以容易画出散点图和线图。下面代码画出 0 到 7 之间的 sin 函数的曲线，函数 `plt.plot()` 可以自动识别出第一个参数是 x 轴坐标，第二个参数是 y 轴坐标。函数 `plt.show()` 画出图形。

```
x = np.arange(0, 7, 0.1)
y = np.sin(x)
plt.plot(x, y, color="r", linestyle="-", linewidth=1, \
         marker = "*")
plt.show()
```

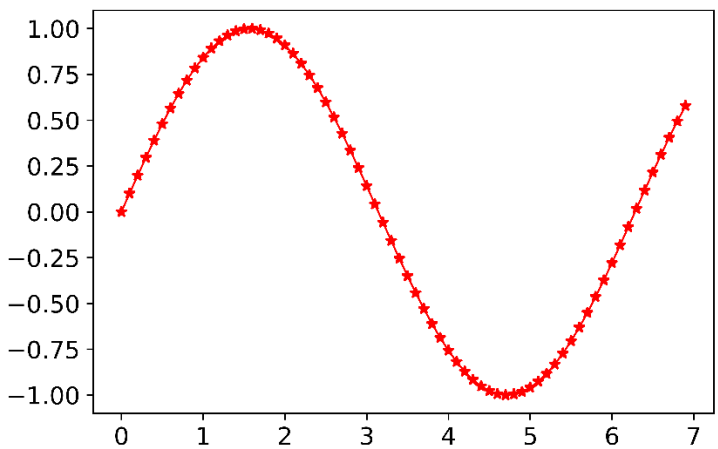

图 2-14 Matplotlib 画正弦函数曲线

函数 `plt.plot()` 还可以设置很多参数，这些参数可以改变线图或者散点图的呈现结果。最常用的有，color（颜色）、linestyle（线条类型）、marker（点类型）、linewidth（线条宽度）。linewidth 只需要设置数字就可以了，数字越大，线条越宽。常用的 color（颜色）、linestyle（线条类型）和 marker（点类型）可以总结如下。

Matplotlib 常用的颜色如表 2-6 所示。

表 2-6 Matplotlib 常用颜色

字符	颜色	字符	颜色	字符	颜色	字符	颜色

b	蓝色	g	绿色	r	红色	y	黄色
c	青色	k	黑色	m	洋红色	w	白色

Matplotlib 常用的线条类型如表 2-7 所示。

表 2-7 Matplotlib 常用线条类型

字符	线条类型	字符	线条类型	字符	线条类型	字符	线条类型
-	实线	:	虚线	–	破折线	-.	点划线

Matplotlib 常用的点类型如表 2-8 所示。

表 2-8 Matplotlib 常用点类型

字符	点类型	字符	点类型	字符	点类型	字符	点类型
o	圆圈	.	点	D	菱形	s	正方形
h	六边形	*	星号	_	水平线	8	八边形

我们还可以把多条线或者不同的散点图画在同一幅图中，并且用函数 `plt.legend()`在图上加上图例，这时需要在函数 `plt.plot()`中加入参数 `label`，`label` 会显示在图例中，如图 2-15。

```
x = np.arange(0, 7, 0.1);
y = np.sin(x)
z = np.cos(x)
plt.plot(x, y, color="r", linestyle="-", linewidth=1,\
         marker = "*", label="sin")
plt.plot(x, z, color="b", linestyle="--", linewidth=2, \
         marker = "D", label="cos")
plt.legend()
plt.show()
```

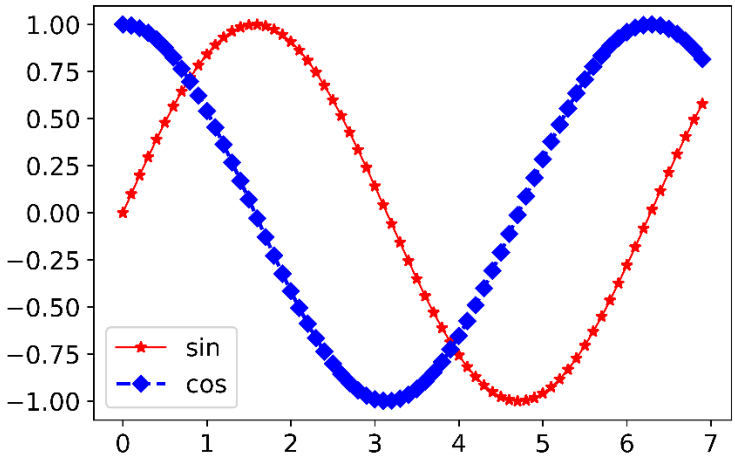

图 2-15 Matplotlib 添加图例

我们也可以使用函数 `plt.xlabel()`和 `plt.ylabel()`添加横坐标和纵坐标的名字，如图 2-16。

```
plt.plot(x, y, color="r", linestyle="-", linewidth=1,\
         marker = "*", label="sin")
plt.plot(x, z, color="b", linestyle="--", linewidth=2,
         marker = "D", label="cos")
plt.legend()
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

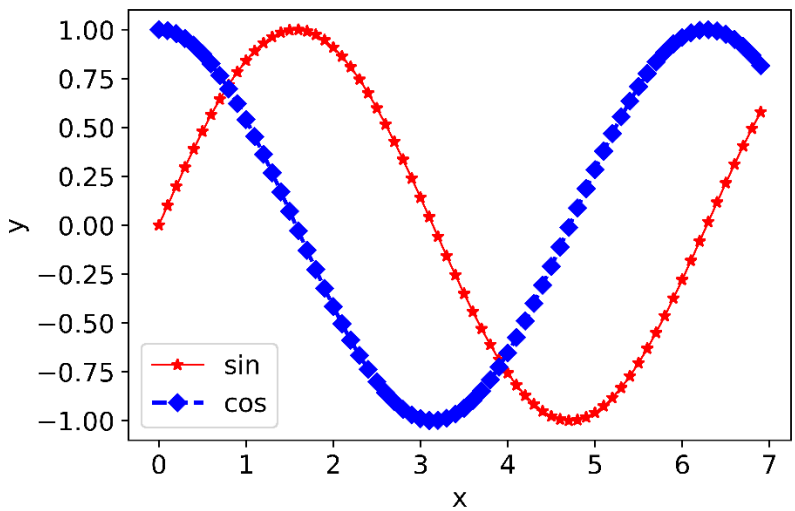

图 2-16 函数 `plt.xlabel()`和 `plt.ylabel()`添加坐标轴名称

函数 `plt.scatter()`可以画出散点图，如图 2-17。

```
x = np.random.normal(size=100)  # 产生 100 个服从标准正态分布的随机数
y = 2*x + np.random.normal(size=100)
plt.scatter(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

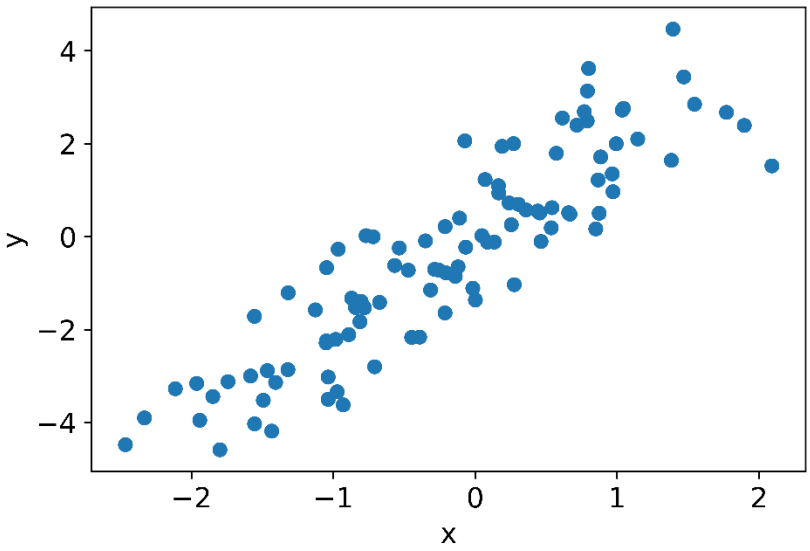

图 2-17 函数 `plt.scatter()` 画散点图

2.7 本章小结

本章介绍的线性代数、微积分、概率论和 Python 相关知识可以帮助你更容易理解和应用深度学习，同时可以作为参考，当你有所遗忘时，可以返回本章复习相关知识。

学习 Anaconda、Jupyter Notebook 或者 Python 时，你一定要自己输入命令或者代码，思考并且观察输出结果。这样，相信你的编程技巧一定可以快速进步！