

# 深度学习之美

## AI时代的数据处理与最佳实践

张玉宏 著



中国工信出版集团



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
<http://www.phei.com.cn>



### 版权相关注意事项：

- 1、书籍版权归著者和出版社所有
- 2、本PDF来自于各个广泛的信息平台，经过整理而成
- 3、本PDF仅限用于非商业用途或者个人交流研究学习使用
- 4、本PDF获得者不得在互联网上以任何目的进行传播
- 5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
- 6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
- 7、请于下载PDF后24小时内研究使用并删掉本PDF

## 作者简介

张玉宏



2012年在电子科技大学取得博士学位，攻读博士期间担任美国西北大学访问学者，现执教于河南工业大学。

中国计算机协会（CCF）会员，CCF YOCSEF郑州2018—2019年度副主席，ACM/IEEE会员。《品味大数据》一书作者。主要研究方向为大数据、人工智能、技术哲学。发表学术论文20余篇，国内外学术作品7部。阿里云云栖社区专栏作家，博文累计阅读逾百万次。



博文视点AI系列

# 深度学习之美

## AI时代的数据处理与最佳实践

张玉宏 著

D e e p L e a r n i n g

电子工业出版社  
Publishing House of Electronics Industry  
北京•BEIJING



## 内 容 简 介

深度学习是人工智能的前沿技术。本书深入浅出地介绍了深度学习的相关理论和实践，全书共分 16 章，采用理论和实践双主线写作方式。第 1 章给出深度学习的大图。第 2 章和第 3 章，讲解了机器学习的相关基础理论。第 4 章和第 5 章，讲解了 Python 基础和基于 Python 的机器学习实战。第 6 至 10 章，先后讲解了 M-P 模型、感知机、多层次神经网络、BP 神经网络等知识。第 11 章讲解了被广泛认可的深度学习框架 TensorFlow。第 12 章和第 13 章详细讲解了卷积神经网络，并给出了相关的实战项目。第 14 章和第 15 章，分别讲解了循环递归网络和长短期记忆（LSTM）网络。第 16 章讲解了神经胶囊网络，并给出了神经胶囊网络设计的详细论述和实践案例分析。

本书结构完整、行文流畅，是一本难得的零基础入门、图文并茂、通俗易懂、理论结合实战的深度学习书籍。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

### 图书在版编目（CIP）数据

深度学习之美：AI 时代的数据处理与最佳实践/张玉宏著. —北京：电子工业出版社，2018.7  
(博文视点 AI 系列)

ISBN 978-7-121-34246-2

I. ①深… II. ①张… III. ①机器学习 IV. ①TP181

中国版本图书馆 CIP 数据核字（2018）第 106119 号

策划编辑：孙奇俏

责任编辑：刘 航

印 刷：三河市双峰印刷装订有限公司

装 订：三河市双峰印刷装订有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱

邮编：100036

开 本：787×980 1/16 印张：42.5

字数：892 千字

版 次：2018 年 7 月第 1 版

印 次：2018 年 7 月第 1 次印刷

定 价：128.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，  
联系及邮购电话：(010) 88254888, 88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819, faq@phei.com.cn。





# 推荐序一

## 通俗也是一种美德

这是一个大数据时代。

这也是一个人工智能时代。

如果说大数据技术是还有待人们去研究、去挖掘、去洞察的问题，那么人工智能无疑是这个问题的解决方案，至少是方案之一。

人的智能，无疑是学习的产物。那么机器的智能呢，它何尝不是学习的产物？只不过在当下，它被深深地打上了“深度学习”的烙印。通过深度学习，我们可以把大数据挖掘的技术问题转换为可计算的问题。

有人说，深度学习不仅是一种算法的升级，更是一种思维模式的升级。其带来的颠覆性在于，它把人类过去痴迷的算法问题演变成数据和计算问题。吴军博士更是断言，未来只有 2% 的人有能力在智能时代独领风骚，成为时代的弄潮儿。所以，拥抱人工智能，携手深度学习，不仅是一种时代的召唤，而且顺应了当前科学技术对人才的紧迫需求。

深度学习矗立于人工智能的前沿。我们远眺它容易，但近爱它却不易。在信息过剩的时代，我们可能会悲哀地发现，知识鸿沟横在我们面前。的确，大量有关深度学习的书籍占据着我们的书架，数不尽的博客充斥着我们的屏幕。然而，很多时候，我们依然对深度学习敬而远之。

这个“敬”是真实的，这个“远”通常是被迫的。因为找到一本通俗易懂的有关深度学习的读物，并非易事。

张玉宏博士所著的《深度学习之美：AI 时代的数据处理与最佳实践》，正是在这种背景下



## IV | 深度学习之美：AI时代的数据处理与最佳实践

以通俗易懂的姿态面世的书籍。如果用一句话来形容这本书，那就是“通俗易懂也是一种美德”。

在任何时代，能把复杂的事情解释清楚，都是一项非常有用的本领。张玉宏博士是一名高校教师，加之他还是一名科技作家，因此，通俗易懂、行文流畅、幽默风趣，便成了这本书的特征。

这里有一个比喻：如果你爱好明史，且古文基础深厚，去阅读《明史》可能是你的不二选择；但倘若你不太喜欢学究气的著作，那就推荐你去读读当年明月所写的《明朝那些事儿》。本书仿佛就是《明朝那些事儿》版的“深度学习”。巧妙的比喻、合理的推断、趣味的故事，时不时地散落在书里，妙趣横生。

当然，这本书也不是十全十美的，但瑕不掩瑜，如果你想零基础入门深度学习，那么相信这本书一定能够给你提供很多帮助。

黄文坚  
墨宽投资创始人、《TensorFlow 实战》作者  
2018年5月于北京



## 推荐序二

# 技术，也可以“美”到极致

和张玉宏博士认识是在 2015 年，那个时候我还在 CSDN&《程序员》杂志做编辑。结缘的过程颇为巧合，仅仅是因为一次无意浏览，看到他在 CSDN 论坛义务回答网友的问题，被其认真、高水平和深入浅出的回答所吸引，倾慕之下，我策划了一次采访。而这次采访奠定了我们的友谊，文章在 CSDN 作为头条发布并获得 3 万多的阅读量后，我们在论文翻译、《程序员》杂志供稿上开始了合作。

仅仅牛刀小试，张博士翻译的文章就篇篇阅读量破万。对于晦涩难懂的技术文章而言，阅读量破万是很高的数字，无异于如今朋友圈的 10 万阅读量。其中最值得一提的是《PayPal 高级工程总监：读完这 100 篇论文就能成大数据高手》这篇文章，在 2015 年阅读量近 6 万后，2017 年又被某大数据公众号转载，在朋友圈刷屏。为此，我在微博上有感而发：“有价值的内容，是经得起时间冲刷的。”然而，文章被追捧的背后，则是张博士呕心沥血的付出，也许只有我知道——为了保障质量，那 100 篇论文张博士全部下载并看了一遍，独立注解超过 50%，前后花了整整一周时间，通宵达旦，聚沙成塔。

2016 年我来到阿里巴巴做云栖社区的内容运营，和张博士的合作也来到了这里。云栖社区是阿里云运营、阿里技术协会和集团技术团队支持的开放技术社区，是云计算、大数据和人工智能顶级社区之一。在这个阶段，张博士在技术文章翻译上又有了另外一种风格：局部放大翻译，就某个点深度解读，并加入自己的认识，这种类似“书评”式的写作方式，让原本枯燥的技术文章显得很有趣，在阅读量上也是立竿见影——他的文章的阅读量比一般技术文章的阅读量多好几倍，这在彼时的社区是极为少见的。



## VI | 深度学习之美：AI时代的数据处理与最佳实践

后来，张博士和我说，他想发表系列文章，细致地讲一讲深度学习，我非常认同和支持他做这件事。

于是接下来的几个月，张博士不断受到以下赞誉：

“文章形象生动，耐人寻味，重燃深度学习的欲望。”

“这个系列写得太棒了！！！ 谢谢您愿意分享。”

“大神，更新的频率可以稍微快点吗？万分感谢！”

有人甚至用追剧来形容自己的感受，并评论：“期待好久了，终于更新了，作者辛苦了，感谢作者提供该优秀文章以供学习。”

“容易理解，害怕大神不更新。”

.....

这些赞誉，说的是在社区篇篇 10 多万阅读量的文章：

一入侯门“深”似海，深度学习深几许（深度学习入门系列之一）

人工“碳”索意犹尽，智能“硅”来未可知（深度学习入门系列之二）

神经网络不胜语，M-P 模型似可寻（深度学习入门系列之三）

“机器学习”三重门，“中庸之道”趋若人（深度学习入门系列之四）

Hello World 感知机，懂你我心才安息（深度学习入门系列之五）

损失函数减肥用，神经网络调权重（深度学习入门系列之六）

山重水复疑无路，最快下降问梯度（深度学习入门系列之七）

BP 算法双向传，链式求导最缠绵（深度学习入门系列之八）

全面连接困何处，卷积网络见解深（深度学习入门系列之九）

卷地风来忽吹散，积得飘零美如画（深度学习入门系列之十）

局部连接来减参，权值共享肩并肩（深度学习入门系列之十一）

激活引入非线性，池化预防过拟合（深度学习入门系列之十二）



循环递归 RNN，序列建模套路深（深度学习入门系列之十三）

LSTM 长短记，长序依赖可追忆（深度学习入门系列之十四）

上面这些文章，光看标题就非同凡响，是的，能把技术文章的标题写得这么文艺和生动，一看文学功力就十分深厚。内容更不用说，通俗易懂、图文并茂、形象生动。“用这么幽默的语言，将那么高大上的技术讲得一个过路人都能够听懂，这是真的好文章。”身边的一位朋友如此述说。而更值得一提的是，在每篇博文后面，除了小结，还有“请你思考”，考察读者对知识的掌握情况，锻炼读者的思辨能力，让读者能够进一步主动学习，触类旁通。

这样出色的文章，也许只有张博士才能写出来。为什么呢？我想，一方面得益于其读博士时，在美国西北大学有过两年访学经历，他在中美教育差异上有过深刻的思考；另一方面，也源自他丰富的教学经验——是的，他在河南工业大学执教多年，懂得“教”与“授”的拿捏。对于张博士的教学，学生袁虎是这么谈论自己的感受的：“他的课跟美国高校的课堂比较接近——开放、平等、互动性强，鼓励学生去思考。上课的时候，他并不死守课本知识，而是特别注重教授给我们学习方法。”袁虎还特别指出，他们专业出来的几个技术大神多多少少都算是张博士的门徒。

在云栖社区的连载，从 2017 年 5 月 17 日开始，到 8 月 17 日结束，一共 14 篇文章，很多读者“追剧”至此，仍意犹未尽。有读者说：“作为机器学习小白，楼主的文章真是赞。楼主，出书吧！！！看博客真担心哪天突然就没有了。”因为一些原因，社区的博客篇幅有限，内容浅尝辄止……他觉得自己可以做得更多、更好。

因此，《深度学习之美：AI 时代的数据处理与最佳实践》在这里和大家见面了。可以说，拿到这本书的读者是非常幸运的，因为你们不需要每天刷博客“追剧”，也不需要苦苦等待。你们可以边捧书边喝咖啡，在橘黄色的台灯下、在安静的深夜里看个尽兴。

人工智能在当下非常火爆。不可否认，也许你可以从汗牛充栋的网上获得深度学习的一些知识点或技巧，但网络中的知识是碎片化的。尤其对于初学者，如果想走得更远，需要一本书系统地进行指导，并从底层思考这些知识的来龙去脉，以及知识之间的关联，本书正是这样一本本书。

结集成册的《深度学习之美：AI 时代的数据处理与最佳实践》除了继承之前博文趣味性、通俗易懂等诸多优点之外，篇幅更宏大（此前的连载只是一个起步），内容上还增加了实战环节，让大家能够学以致用，在实践中与理论印证。另外，相比此前连载的博文，书籍中增加了许多



## VIII | 深度学习之美：AI时代的数据处理与最佳实践

张老师亲自绘制的趣图，诙谐地说明了不同知识点或概念间的区别。在理论上，张博士也对公式的前因后果给出了详细的推导过程，只有知道它是怎么来的，才能更好地运用它。学习知识不正是这样吗？

社会变化非常快，因此人们总爱反复核算事物的价值，喜欢性价比高的东西。如何衡量一本书的价值，除了看它是否能帮到你之外（技能价值），还要看它的社会价值。本书是张博士深度学习的思想随笔，兴之所至的内容，往往也是精彩至极、深度思考的结晶。

我非常佩服张博士，他不仅博览群书，还能够将不同类型的书籍内化，并结合生活案例，以一种非常有趣的形式将深奥的知识表达出来，比如用“求婚”“耳光”等例子讲解“激活函数”和“卷积函数”。尤其是“中庸之道”的例子，让大家在悟透一个很难弄懂的知识点的同时，自己的思想也从富有哲理的故事中变得不一样。

这种技术领域的人文情怀，绝非一般高手能做到的。上述认识，相信手握此书的您，也会很快感受到。

——@我是主题曲哥哥，网易高级编辑  
前阿里云资深内容运营、CSDN&《程序员》杂志编辑

2018年5月



## 自序

### 深度学习的浅度梦想

这是一本有关“深度学习”的图书！

这是一本有关“深度学习”通俗易懂的图书！

这是一本有关“深度学习”的、有些人文情怀的图书！

我希望，我的读者在读这本书时，能给它这三种不同境界的渐进式的评价。第一个评价，说明它“有料”。第二个评价，说明它“有用”。第三个评价，说明它“有趣”。“有料、有用且有趣”是我对本书的定位，也是写作本书的浅度梦想，不是有大咖说过吗，“梦想还是要有的，万一实现了呢？”

写一本好书，真的很难！

但并非不能达成。窃以为，写成一本好书，通常有两条途径。第一条我称之为“自上而下大家传道法”。也就是说，有些学术大家已在领域内功成名就，名声斐然，他们俯下身段，抽出时间，高屋建瓴，精耕细作，必出精品。比如，卡耐基梅隆大学的 Tom Mitchell 教授编写的《机器学习》、南京大学周志华老师编写的《机器学习》，都是业内口碑极好的畅销常青树，实为我辈楷模。

但“大家写好书”并不是充分条件，因为大家通常都非常忙，他们可能非常“有料、有钱（有经费）”，但却未必“有闲”。要知道，写作不仅仅是一项脑力活，它还是一项极花费时间的体力活。

好在还有写成好书的第二条途径，我且称之为“自下而上小兵探道法”。也就是说，写书的作者本身并非领域专家，而是来自科研实战一线，他们的眼前也时常迷茫一片，不得不肉搏每一个理论困惑，手刃每一个技术难题，一路走来，且泣且歌，终于爬上一个小山丘。松了口气，渴了口水，嗯，我要把自己趟过的河，踩过的坑，写出来总结一下，除了自勉，也能让寻路而来的同门或同道中人，不再这么辛苦。

很显然，我把自己定位为第二类（至少梦想是）。

我是一个科技写作爱好者，我在网络上写过很多有关于大数据主题的（主要发表在 CSDN）文章，也有关于深度学习的（主要发表于阿里云-云栖社区）。出于爱好写作的原因，有时我也关注写作的技巧。直到有一天，一位知名人士的一席话，一下子“电着”我了。他说，“写作的终极技巧，就是看你写的东西对读者有没有用。”拿这个标准来衡量一下，什么辞藻华丽、什么文笔优美，都可能是绿叶与浮云。在这一瞬间，我也明白了，为什么我所在的城市，地铁时刻表的变更通知，寥寥几百字，短短没几天，阅读量也可以轻易达到 10 万。嗯，这样的写作，有干货，对读者有用。好作品的要素，它都有！

于是，“对读者有用”，就成为指导我写作这本书的宗旨。以用户的思维度量，就可以比较清晰地知道，什么对读者有用。

当前，人工智能非常火爆。自从 AlphaGo 点燃世人对人工智能的极大热情后，学术界和产业界都积极投身于此，试图分得一杯羹。而当前（至少是当前）人工智能的当红主角就是“深度学习”，它不仅仅表现在 AlphaGo 一战成名的技术上，还表现在图像识别、语音识别、自然语言处理性能提升上，总总而生，林林而群。

当然，想投身于此并非易事，因为深度学习的门槛比较高。为了搞懂深度学习，我把国内市面上大部分与深度学习相关的书籍都买来拜读了（在后记中，我会感谢支持的各种基金），受益匪浅，但至少于我而言，它们大部分的学习曲线都是陡峭的，或者说它们大多高估了初学者的接受程度，为了读懂它们，读者真的需要“深度学习”。

在深度学习领域，的确也有一批高水平的读者，但他们可能并不需要通过相对滞后的书籍来提高自己的知识水平，新鲜出炉的 arXiv 论文，才是他们的“菜”。但高手毕竟有限，懵懵懂懂的初学者，数量还是相当庞大的。

于是，我想，写一本零基础入门的、通俗易懂的、图文并茂的、理论结合实战的深度学习

书籍，对广大的深度学习初学者来说，应该是有用的。

本书的写作风格，也紧扣前面的四个修饰词，章节的安排也是按照循序渐进的节奏展开的。为了降低门槛和强调实践性，本书采用了双主线写作方式，一条主线是理论脉络，从基础的机器学习概念，到感知机、M-P 模型、全连接网络，再到深度学习网络，深入浅出地讲解相关的理论。另外一条主线是实战脉络，从 Python 零基础入门说起，直到 TensorFlow 的高级应用。

全书共分 16 章，具体来说，第 1 章给出深度学习的大图（Big Picture），让读者对其有一个宏观认知。第 2 章和第 3 章，给出了机器学习的相关基础理论。仅仅懂理论是不够的，还需要动手实践，用什么实践呢？最热门的机器学习语言非 Python 莫属了。于是我们在第 4 章添加了 Python 基础，以边学边用边提高为基调，并在第 5 章讲解了基于 Python 的机器学习实战。

有了部分 Python 基础，也有了部分机器学习基础，接下来，我们该学习与神经网络相关的理论了。于是在第 6 章至第 10 章，我们先后讲解了 M-P 模型、感知机、多层神经网络、BP 神经网络等知识。其中大部分的理论都配有 Python 实战讲解，就是让读者有“顶天（上接理论）立地（下接实战）”的感觉。接下来的问题就是，如果所有神经网络学习的项目都是 Python 手工编写的，是不是效率太低了呢？

是的，是该考虑用高效率框架的时候了，于是在第 11 章，我们讲解了被广泛认可的深度学习框架 TensorFlow。有了这个基础，后面的深度学习理论就以此做实战基础。第 12 章详细讲解了卷积神经网络。随后，在第 13 章，我们站在实战的基础上，对卷积神经网络的 TensorFlow 实践进行了详细介绍。

任何一项技术都有其不足。在第 14 章，我们讲解了循环递归网络（RNN）。在第 15 章，我们讲解了长短期记忆（LSTM）网络。以上两章内容，并非都是高冷的理论，除了给出理论背后有意思的小故事，还结合 TensorFlow 进行了实战演练。在第 16 章，我们顺便“惊鸿一瞥”解读了 Hinton 教授的新作“神经网络胶囊（CapsNet）”，点出卷积神经网络的不足，并给出了神经胶囊的详细论述和实践案例分析。

本书中的部分内容（共计 14 篇），先后发表在技术达人云集的云栖社区（<https://yq.aliyun.com/topic/111>），然后被很多热心的网友转载到 CSDN、知乎、微信公众号、百度百家等自媒体中，受到了很多读者的认可。于吾心，有乐陶然。

当然，从我对自己的定位——“小兵探道”可知，我对深度学习的认知，仍处于一种探索

阶段，我仍是一个深度学习的学习者。在图书中、在网络中，我学习并参考了很多有价值的资料。这里，我对这些有价值的资料的提供者、生产者，表示深深的敬意和谢意。

有时候，我甚至把自己定位为一个“知识的搬运工”、深度学习知识的梳理者。即使如此，由于学术水平尚浅，我对一些理论或技术的理解，可能是肤浅的，甚至是错误的，所以，如果本书有误，且如果读者“有闲”，不妨给出您的宝贵建议和意见，我在此表示深深的感谢。同时，由于时间和精力有限，很多有用的深度学习理论和技术还没有涉及，只待日后补上。

我的联系信箱为：[zhangyuhong001@gmail.com](mailto:zhangyuhong001@gmail.com)。

张玉宏

2018年3月

## 读者服务

轻松注册成为博文视点社区用户（[www.broadview.com.cn](http://www.broadview.com.cn)），扫码直达本书页面。

- **下载资源：**本书如提供示例代码及资源文件，均可在下载资源处下载。
- **提交勘误：**您对书中内容的修改意见可在提交勘误处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动：**在页面下方读者评论处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：<http://www.broadview.com.cn/34246>



# 目录

第1章 一入侯门“深”似海，深度学习深几许 .....	1
1.1 深度学习的巨大影响 .....	2
1.2 什么是学习 .....	4
1.3 什么是机器学习 .....	4
1.4 机器学习的4个象限 .....	5
1.5 什么是深度学习 .....	6
1.6 “恋爱”中的深度学习 .....	7
1.7 深度学习的方法论 .....	9
1.8 有没有浅层学习 .....	13
1.9 本章小结 .....	14
1.10 请你思考 .....	14
参考资料 .....	14
第2章 人工“碳”索意犹尽，智能“硅”来未可知 .....	16
2.1 信数据者得永生吗 .....	17
2.2 人工智能的“江湖定位” .....	18
2.3 深度学习的归属 .....	19
2.4 机器学习的形式化定义 .....	21
2.5 为什么要用神经网络 .....	24
2.6 人工神经网络的特点 .....	26

2.7 什么是通用近似定理 .....	27
2.8 本章小结 .....	31
2.9 请你思考 .....	31
参考资料 .....	31
第 3 章 “机器学习”三重门，“中庸之道”趋若人 .....	33
3.1 监督学习 .....	34
3.1.1 感性认知监督学习 .....	34
3.1.2 监督学习的形式化描述 .....	35
3.1.3 $k$ -近邻算法 .....	37
3.2 非监督学习 .....	39
3.2.1 感性认识非监督学习 .....	39
3.2.2 非监督学习的代表—— $K$ 均值聚类 .....	41
3.3 半监督学习 .....	45
3.4 从“中庸之道”看机器学习 .....	47
3.5 强化学习 .....	49
3.6 本章小结 .....	52
3.7 请你思考 .....	53
参考资料 .....	53
第 4 章 人生苦短对酒歌，我用 Python 乐趣多 .....	55
4.1 Python 概要 .....	56
4.1.1 为什么要用 Python .....	56
4.1.2 Python 中常用的库 .....	58
4.2 Python 的版本之争 .....	61
4.3 Python 环境配置 .....	65
4.3.1 Windows 下的安装与配置 .....	65
4.3.2 Mac 下的安装与配置 .....	72
4.4 Python 编程基础 .....	76
4.4.1 如何运行 Python 代码 .....	77

4.4.2 代码缩进 .....	79
4.4.3 注释 .....	80
4.4.4 Python 中的数据结构 .....	81
4.4.5 函数的设计 .....	93
4.4.6 模块的导入与使用 .....	101
4.4.7 面向对象程序设计 .....	102
4.5 本章小结 .....	112
4.6 请你思考 .....	112
参考资料 .....	113
 第 5 章 机器学习终觉浅，Python 带我来实践 .....	114
5.1 线性回归 .....	115
5.1.1 线性回归的概念 .....	115
5.1.2 简易线性回归的 Python 实现详解 .....	119
5.2 k-近邻算法 .....	139
5.2.1 k-近邻算法的三个要素 .....	140
5.2.2 k-近邻算法实战 .....	143
5.2.3 使用 scikit-learn 实现 k-近邻算法 .....	155
5.3 本章小结 .....	162
5.4 请你思考 .....	162
参考资料 .....	162
 第 6 章 神经网络不胜语，M-P 模型似可寻 .....	164
6.1 M-P 神经元模型是什么 .....	165
6.2 模型背后的那些人和事 .....	167
6.3 激活函数是怎样的一种存在 .....	175
6.4 什么是卷积函数 .....	176
6.5 本章小结 .....	177
6.6 请你思考 .....	178
参考资料 .....	178

第 7 章 Hello World 感知机，懂你我心才安息 .....	179
7.1 网之初，感知机 .....	180
7.2 感知机名称的由来 .....	180
7.3 感性认识“感知机” .....	183
7.4 感知机是如何学习的 .....	185
7.5 感知机训练法则 .....	187
7.6 感知机的几何意义 .....	190
7.7 基于 Python 的感知机实战 .....	191
7.8 感知机的表征能力 .....	196
7.9 本章小结 .....	199
7.10 请你思考 .....	199
参考资料 .....	199
第 8 章 损失函数减肥用，神经网络调权重 .....	201
8.1 多层网络解决“异或”问题 .....	202
8.2 感性认识多层前馈神经网络 .....	205
8.3 是浅而“胖”好，还是深而“瘦”佳 .....	209
8.4 分布式特征表达 .....	210
8.5 丢弃学习与集成学习 .....	211
8.6 现实很丰满，理想很骨感 .....	212
8.7 损失函数的定义 .....	213
8.8 热力学定律与梯度弥散 .....	215
8.9 本章小结 .....	216
8.10 请你思考 .....	216
参考资料 .....	217
第 9 章 山重水复疑无路，最快下降问梯度 .....	219
9.1 “鸟飞派”还飞不 .....	220
9.2 1986 年的那篇神作 .....	221
9.3 多层感知机网络遇到的大问题 .....	222

9.4 神经网络结构的设计 .....	225
9.5 再议损失函数 .....	227
9.6 什么是梯度 .....	229
9.7 什么是梯度递减 .....	231
9.8 梯度递减的线性回归实战 .....	235
9.9 什么是随机梯度递减 .....	238
9.10 利用 SGD 解决线性回归实战 .....	240
9.11 本章小结 .....	247
9.12 请你思考 .....	248
参考资料 .....	248
<b>第 10 章 BP 算法双向传，链式求导最缠绵 .....</b>	<b>249</b>
10.1 BP 算法极简史 .....	250
10.2 正向传播信息 .....	251
10.3 求导中的链式法则 .....	255
10.4 误差反向传播 .....	264
10.4.1 基于随机梯度下降的 BP 算法 .....	265
10.4.2 输出层神经元的权值训练 .....	267
10.4.3 隐含层神经元的权值训练 .....	270
10.4.4 BP 算法的感性认知 .....	273
10.4.5 关于 BP 算法的补充说明 .....	278
10.5 BP 算法实战详细解释 .....	280
10.5.1 初始化网络 .....	280
10.5.2 信息前向传播 .....	282
10.5.3 误差反向传播 .....	285
10.5.4 训练网络（解决异或问题） .....	288
10.5.5 利用 BP 算法预测小麦品种的分类 .....	293
10.6 本章小结 .....	301
10.7 请你思考 .....	302
参考资料 .....	304

第 11 章 一骑红尘江湖笑，TensorFlow 谷歌造.....	305
11.1 TensorFlow 概述 .....	306
11.2 深度学习框架比较.....	309
11.2.1 Theano.....	309
11.2.2 Keras .....	310
11.2.3 Caffe.....	311
11.2.4 PyTorch .....	312
11.3 TensorFlow 的安装.....	313
11.3.1 Anaconda 的安装.....	313
11.3.2 TensorFlow 的 CPU 版本安装 .....	315
11.3.3 TensorFlow 的源码编译 .....	323
11.4 Jupyter Notebook 的使用 .....	331
11.4.1 Jupyter Notebook 的由来 .....	331
11.4.2 Jupyter Notebook 的安装 .....	333
11.5 TensorFlow 中的基础语法.....	337
11.5.1 什么是数据流图 .....	338
11.5.2 构建第一个 TensorFlow 数据流图 .....	339
11.5.3 可视化展现的 TensorBoard .....	342
11.5.4 TensorFlow 的张量思维 .....	346
11.5.5 TensorFlow 中的数据类型 .....	348
11.5.6 TensorFlow 中的操作类型 .....	353
11.5.7 TensorFlow 中的 Graph 对象 .....	356
11.5.8 TensorFlow 中的 Session.....	358
11.5.9 TensorFlow 中的 placeholder .....	361
11.5.10 TensorFlow 中的 Variable 对象.....	363
11.5.11 TensorFlow 中的名称作用域 .....	365
11.5.12 张量的 Reduce 方向 .....	367
11.6 手写数字识别 MNIST .....	372
11.6.1 MNIST 数据集简介 .....	373
11.6.2 MNIST 数据的获取与预处理.....	375
11.6.3 分类模型的构建——Softmax Regression.....	378

11.7 TensorFlow 中的 Eager 执行模式 .....	394
11.7.1 Eager 执行模式的背景 .....	394
11.7.2 Eager 执行模式的安装 .....	395
11.7.3 Eager 执行模式的案例 .....	395
11.7.4 Eager 执行模式的 MNIST 模型构建 .....	398
11.8 本章小结 .....	401
11.9 请你思考 .....	402
参考资料 .....	403
 第 12 章 全面连接困何处，卷积网络显神威 .....	404
12.1 卷积神经网络的历史 .....	405
12.1.1 眼在何方？路在何方？ .....	405
12.1.2 卷积神经网络的历史脉络 .....	406
12.1.3 那场著名的学术赌局 .....	410
12.2 卷积神经网络的概念 .....	412
12.2.1 卷积的数学定义 .....	412
12.2.2 生活中的卷积 .....	413
12.3 图像处理中的卷积 .....	414
12.3.1 计算机“视界”中的图像 .....	414
12.3.2 什么是卷积核 .....	415
12.3.3 卷积在图像处理中的应用 .....	418
12.4 卷积神经网络的结构 .....	420
12.5 卷积层要义 .....	422
12.5.1 卷积层的设计动机 .....	422
12.5.2 卷积层的局部连接 .....	427
12.5.3 卷积层的 3 个核心概念 .....	428
12.6 细说激活层 .....	434
12.6.1 两个看似闲扯的问题 .....	434
12.6.2 追寻问题的本质 .....	435
12.6.3 ReLU 的理论基础 .....	437
12.6.4 ReLU 的不足之处 .....	441

12.7 详解池化层 .....	442
12.8 勿忘全连接层 .....	445
12.9 本章小结 .....	446
12.10 请你思考 .....	447
参考资料 .....	448
第 13 章 纸上谈兵终觉浅，绝知卷积要编程 .....	450
13.1 TensorFlow 的 CNN 架构 .....	451
13.2 卷积层的实现 .....	452
13.2.1 TensorFlow 中的卷积函数 .....	452
13.2.2 图像处理中的常用卷积核 .....	456
13.3 激活函数的使用 .....	460
13.3.1 Sigmoid 函数 .....	460
13.3.2 Tanh 函数 .....	461
13.3.3 修正线性单元——ReLU .....	462
13.3.4 Dropout 函数 .....	462
13.4 池化层的实现 .....	466
13.5 规范化层 .....	470
13.5.1 为什么需要规范化 .....	470
13.5.2 局部响应规范化 .....	472
13.5.3 批规范化 .....	475
13.6 卷积神经网络在 MNIST 分类器中的应用 .....	480
13.6.1 数据读取 .....	480
13.6.2 初始化权值和偏置 .....	480
13.6.3 卷积和池化 .....	482
13.6.4 构建第一个卷积层 .....	482
13.6.5 构建第二个卷积层 .....	483
13.6.6 实现全连接层 .....	484
13.6.7 实现 Dropout 层 .....	485
13.6.8 实现 Readout 层 .....	485
13.6.9 参数训练与模型评估 .....	485

13.7 经典神经网络——AlexNet 的实现 .....	488
13.7.1 AlexNet 的网络架构 .....	488
13.7.2 数据读取 .....	490
13.7.3 初始化权值和偏置 .....	491
13.7.4 卷积和池化 .....	491
13.7.5 局部响应归一化层 .....	492
13.7.6 构建卷积层 .....	492
13.7.7 实现全连接层和 Dropout 层 .....	493
13.7.8 实现 Readout 层 .....	494
13.7.9 参数训练与模型评估 .....	494
13.8 本章小结 .....	495
13.9 请你思考 .....	496
参考资料 .....	496

## 第 14 章 循环递归 RNN，序列建模套路深 ..... 498

14.1 你可能不具备的一种思维 .....	499
14.2 标准神经网络的缺陷所在 .....	501
14.3 RNN 简史 .....	502
14.3.1 Hopfield 网络 .....	503
14.3.2 Jordan 递归神经网络 .....	504
14.3.3 Elman 递归神经网络 .....	505
14.3.4 RNN 的应用领域 .....	506
14.4 RNN 的理论基础 .....	506
14.4.1 Elman 递归神经网络 .....	506
14.4.2 循环神经网络的生物学机理 .....	508
14.5 RNN 的结构 .....	509
14.6 循环神经网络的训练 .....	512
14.6.1 问题建模 .....	512
14.6.2 确定优化目标函数 .....	513
14.6.3 参数求解 .....	513
14.7 基于 RNN 的 TensorFlow 实战——正弦序列预测 .....	514

14.7.1 生成数据.....	516
14.7.2 定义权值和偏置.....	517
14.7.3 前向传播.....	519
14.7.4 定义损失函数.....	522
14.7.5 参数训练与模型评估.....	522
14.8 本章小结 .....	524
14.9 请你思考 .....	524
参考资料 .....	525
 第 15 章 LSTM 长短记，长序依赖可追忆 .....	526
15.1 遗忘是好事还是坏事 .....	527
15.2 施密德胡伯是何人 .....	527
15.3 为什么需要 LSTM .....	529
15.4 拆解 LSTM .....	530
15.4.1 传统 RNN 的问题所在 .....	530
15.4.2 改造的神经元 .....	531
15.5 LSTM 的前向计算 .....	533
15.5.1 遗忘门 .....	534
15.5.2 输入门 .....	535
15.5.3 候选门 .....	536
15.5.4 输出门 .....	537
15.6 LSTM 的训练流程 .....	539
15.7 自然语言处理的一个假设 .....	540
15.8 词向量表示方法 .....	542
15.8.1 独热编码表示 .....	543
15.8.2 分布式表示 .....	545
15.8.3 词嵌入表示 .....	547
15.9 自然语言处理的统计模型 .....	549
15.9.1 NGram 模型 .....	549
15.9.2 基于神经网络的语言模型 .....	550
15.9.3 基于循环神经网络的语言模型 .....	553

15.9.4 LSTM 语言模型的正则化 .....	556
15.10 基于 Penn Tree Bank 的自然语言处理实战 .....	560
15.10.1 下载及准备 PTB 数据集 .....	561
15.10.2 导入基本包 .....	562
15.10.3 定义相关的参数 .....	562
15.10.4 语言模型的实现 .....	563
15.10.5 训练并返回 perplexity 值 .....	573
15.10.6 定义主函数并运行 .....	575
15.10.7 运行结果 .....	578
15.11 本章小结 .....	579
15.12 请你思考 .....	580
参考资料 .....	580
 第 16 章 卷积网络虽动人，胶囊网络更传“神” .....	583
16.1 从神经元到神经胶囊 .....	584
16.2 卷积神经网络面临的挑战 .....	584
16.3 神经胶囊的提出 .....	588
16.4 神经胶囊理论初探 .....	591
16.4.1 神经胶囊的生物学基础 .....	591
16.4.2 神经胶囊网络的哲学基础 .....	592
16.5 神经胶囊的实例化参数 .....	594
16.6 神经胶囊的工作流程 .....	598
16.6.1 神经胶囊向量的计算 .....	598
16.6.2 动态路由的工作机理 .....	600
16.6.3 判断多数字存在性的边缘损失函数 .....	606
16.6.4 胶囊神经网络的结构 .....	607
16.7 CapsNet 的验证与实验 .....	614
16.7.1 重构和预测效果 .....	614
16.7.2 胶囊输出向量的维度表征意义 .....	616
16.7.3 重叠图像的分割 .....	617
16.8 神经胶囊网络的 TensorFlow 实现 .....	618

16.8.1 导入基本包及读取数据集 .....	619
16.8.2 图像输入 .....	619
16.8.3 卷积层 Conv1 的实现 .....	619
16.8.4 PrimaryCaps 层的实现 .....	620
16.8.5 全连接层 .....	622
16.8.6 路由协议算法 .....	628
16.8.7 估计实体出现的概率 .....	630
16.8.8 损失函数的实现 .....	631
16.8.9 额外设置 .....	639
16.8.10 训练和评估 .....	640
16.8.11 运行结果 .....	643
16.9 本章小结 .....	644
16.10 请你思考 .....	645
16.11 深度学习美在何处 .....	646
参考资料 .....	647
后记 .....	648
索引 .....	651

## 深度学习入门与实践

Chapter one

# 第1章 一入侯门“深”似海， 深度学习深几许

当你和恋人在路边手拉手约会的时候，你可曾想，你们之间早已碰撞出了一种神秘的智慧——深度学习。恋爱容易，相处不易，不断磨合，打造你们的默契，最终才能决定你们是否能在一起。深度学习也一样，输入各种不同的参数，进行训练拟合，最后输出拟合结果。恋爱本不易，且学且珍惜！

## 1.1 深度学习的巨大影响

近年来，作为人工智能领域最重要的进展——深度学习（Deep Learning），在诸多领域都有很多惊人的表现。例如，它在棋类博弈、计算机视觉、语音识别及自动驾驶等领域，表现得与人类一样好，甚至更好。早在 2013 年，深度学习就被麻省理工学院的《MIT 科技评论》（*MIT Technology Review*）评为世界 10 大突破性技术之一，如图 1-1 所示。

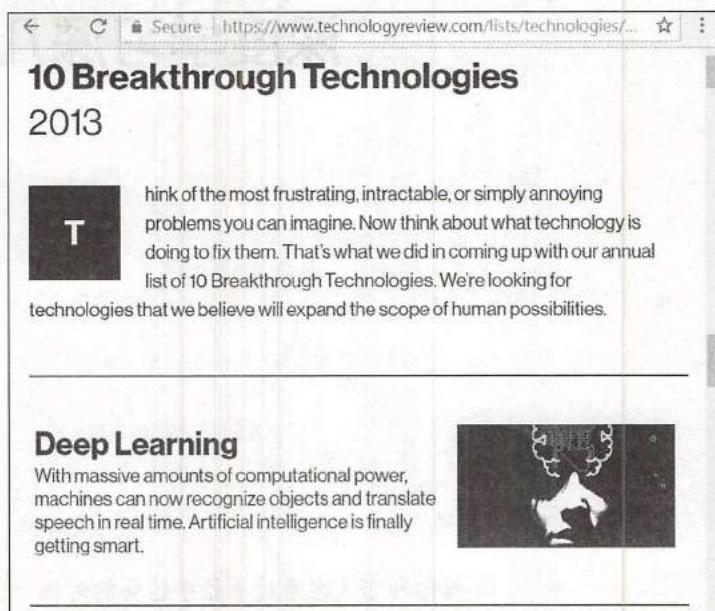


图 1-1 深度学习入围 2013 年 MIT 10 大突破性技术

另一个更具有划时代意义的案例是，2016 年 3 月，围棋世界顶级棋手李世石九段，以 1 : 4 不敌谷歌公司研发的阿尔法围棋（AlphaGo，亦称阿尔法狗），这标志着人工智能在围棋领域已经开始“碾压”人类。在 2016 年年末至 2017 年年初，AlphaGo 的升级版 Master（大师）又在围棋快棋对决中，以 60 场连胜横扫中日韩顶尖职业高手，一时震惊四野。

但有人并不服气，或者想替人类争口气。1997 年出生的柯洁，就是这样的一个人。他是围棋史上最年轻的四冠王，围棋等级世界排名第一。2017 年 5 月 23 日至 27 日，在与 AlphaGo 2.0 进行的人机大战中，柯洁虽殚精竭虑，无奈还是以 0 : 3 战败，再次令世人瞠目结舌（参见图 1-2）。而背后支撑 AlphaGo 具备如此强悍智能的“股肱之臣”之一，正是深度学习。

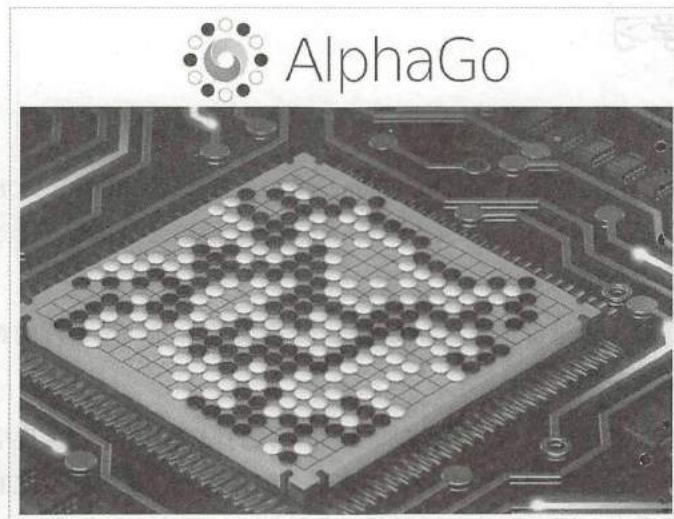


图 1-2 AlphaGo

一时间，深度学习，这个本专属于计算机科学的术语，成为包括学术界、工业界甚至风险投资界等众多领域的热词。的确，它已对我们的工作、生活甚至思维都产生了深远的影响。

比如，有人<sup>①</sup>认为，深度学习不仅是一种算法的升级，还是一种全新的思维方式。我们完全可以利用深度学习，通过对海量数据的快速处理，消除信息的不确定性，从而帮助我们认知世界。它带来的颠覆性在于，将人类过去痴迷的算法问题，演变成数据和计算问题。在以前，“算法为核心竞争力”正在转变为“数据为核心竞争力”。这个观点，多少有点暗合阿里巴巴集团董事局主席马云的观点，我们正进入一个DT（Data Technology，数据技术）时代。

在人工智能领域，深度学习之所以备受瞩目，是因为从原始的输入层开始，到中间每一个隐含层的数据抽取变换，到最终的输出层的判断，所有特征的提取，全程是一个没有人工干预的训练过程。这个自主特性，在机器学习领域，是革命性的。

世界知名深度学习专家吴恩达（Andrew Ng）曾表示：“我们没有像通常（机器学习）做的那样，自己来框定边界，而是直接把海量数据投放到算法中，让数据自己说话，系统会自动从数据中学习。”谷歌大脑项目（Google Brain Project）的计算机科学家杰夫·迪恩（Jeff Dean）则说：“在训练的时候，我们从来不会告诉机器说：‘这是一只猫’。实际上，是系统自己发明或者领悟了‘猫’的概念。”

<sup>①</sup> 傅盛认为深度学习是一种新的思维方式，要了解更多信息请参阅 <http://36kr.com/p/5057846.html>。

## 1.2 什么是学习

说到“深度学习”，追根溯源，我们需要先知道什么是“学习”。

著名学者赫伯特·西蒙教授（Herbert Simon，1975 年图灵奖获得者、1978 年诺贝尔经济学奖获得者）曾对“学习”下过一个定义：“如果一个系统，能够通过执行某个过程，就此改进了它的性能，那么这个过程就是学习”。

大师果然名不虚传，永远都是那么言简意赅，一针见血。从西蒙教授的观点可以看出，学习的核心目的就是改善性能。

其实对于人而言，这个定义也是适用的。比如，我们现在正在学习深度学习的知识，其本质目的就是为了提升自己在机器学习上的认知水平。如果我们仅仅是低层次的重复性学习，而没有达到认知升级的目的，那么即使表面看起来非常勤奋，其实也仅仅是一个“伪学习者”，因为我们没有改善性能。

按照这个解释，那句著名的口号“好好学习，天天向上”，就会焕发新的含义：如果没有性能上的“向上”，即使非常辛苦地“好好”，即使长时间地“天天”，都无法算作“学习”。

## 1.3 什么是机器学习

遵循西蒙教授的观点，对于计算机系统而言，通过运用数据及某种特定的方法（比如统计方法或推理方法）来提升机器系统的性能，就是机器学习（Machine Learning，简称 ML）。

英雄所见略同。卡耐基梅隆大学的机器学习和人工智能教授汤姆·米切尔（Tom Mitchell），在他的经典教材《机器学习》<sup>[1]</sup>中，也给出了更为具体（其实也很抽象）的定义：

对于某类任务（Task，简称 T）和某项性能评价准则（Performance，简称 P），如果一个计算机程序在 T 上，以 P 作为性能的度量，随着经验（Experience，简称 E）的积累，不断自我完善，那么我们称这个计算机程序从经验 E 中进行了学习。

比如，学习围棋的程序 AlphaGo，它可以通过和自己下棋获取经验，那么，它的任务 T 就是“参与围棋对弈”，它的性能 P 就是用“赢得比赛的百分比”来度量的。类似的，学生的任务 T 就是“上课看书写作业”，它的性能 P 就用“考试成绩”来度量。

因此，Mitchell 教授认为，对于一个学习问题，我们需要明确三个特征：任务的类型、衡

量任务性能提升的标准以及获取经验的来源。

事实上，看待问题的角度不同，机器学习的定义也略有不同。比如，支持向量机（SVM）的主要提出者弗拉基米尔·万普尼克（Vladimir Vapnik），在其著作《统计学习理论的本质》<sup>[2]</sup>中就提出，“机器学习就是一个基于经验数据的函数估计问题”。

而在另一本由斯坦福大学统计系的特雷弗·哈斯蒂（Trevor Hastie）等人编写的经典著作《统计学习基础》<sup>[3]</sup>则认为，机器学习就是“抽取重要的模式和趋势，理解数据的内涵表达，即从数据中学习（to extract important patterns and trends, and understand “what the data says. We call this learning from data”）”。

这三个有关机器学习的定义，各有侧重，各有千秋。Mitchell 的定义强调学习的效果；Vapnik 的定义侧重机器学习的可操作性；而 Hastie 等人的定义则突出了学习任务的分类。但其共同的特点在于，都强调了经验和数据的重要性，都认可机器学习提供了从数据中提取知识的方法<sup>[4]</sup>。

当下，我们正处于大数据时代。众所周知，大数据时代的一个显著特征就是，“数据泛滥成灾，信息超量过载，然而知识依然匮乏不堪”。因此，能自动从大数据中获取知识的机器学习，必然会在大数据时代的舞台上扮演重要角色。

## 1.4 机器学习的 4 个象限

一般来说，知识在两个维度上可分成四类，如图 1-3 所示。即从可统计与否上来看，可分为可统计的知识和不可统计的知识这两个维度。从能否推理上看，可分为可推理的知识和不可推理的知识这两个维度<sup>[5]</sup>。

在横向，对于可推理的，可以通过机器学习的方法，最终完成这个推理。传统的机器学习方法，就是试图找到可举一反三的方法，向可推理但不可统计的象限进发（象限Ⅱ）。目前看来，这个象限的研究工作（即基于推理的机器学习）陷入了不温不火的境地，能不能峰回路转，还有待时间的检验。

而在纵向，对于可统计的、但不可推理的（即象限Ⅲ），可通过神经网络这种特定的机器学习方法，达到性能提升的目的。目前，基于深度学习的棋类博弈（阿尔法狗）、计算机视觉（猫狗识别）、自动驾驶等，其实都是在这个象限做出了耀眼的成就。

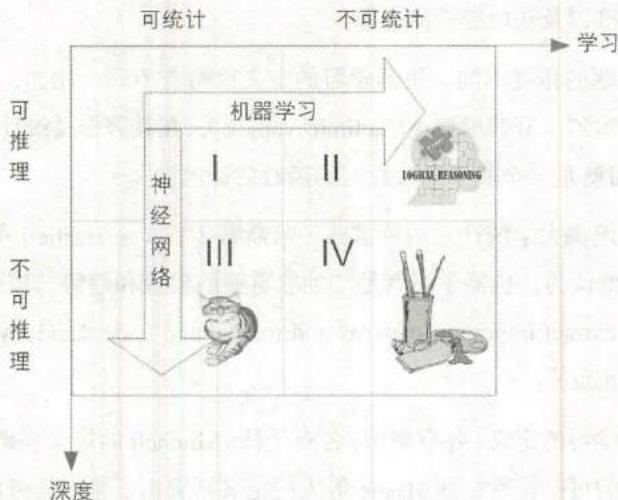


图 1-3 知识的 4 个象限

从图 1-3 可知，深度学习属于统计学习的范畴。用李航博士的话来说<sup>[6]</sup>，统计机器学习的对象，其实就是数据。这是因为，对于计算机系统而言，所有的“经验”都是以数据的形式存在的。作为学习的对象，数据的类型是多样的，可以是数字、文字、图像、音频、视频，也可以是它们的各种组合。

统计机器学习，就是从数据出发，提取数据的特征（由谁来提取，是一个大是大非的问题，下面将进行介绍），抽象出数据的模型，发现数据中的知识，最后再回到数据的分析与预测中去。

经典机器学习（位于第Ⅱ象限），通常是用人类的先验知识，把原始数据预处理成各种特征（Feature），然后对特征进行分类。然而，这种分类的效果，高度取决于特征选取的好坏。传统的机器学习专家们，把大部分时间都花在如何寻找更加合适的特征上。因此，早期的机器学习专家非常辛苦。传统的机器学习，其实可以有一个更合适的称呼——特征工程（Feature Engineering）。

但功不唐捐。这种痛，也有其好的一面。这是因为，特征是由人辛辛苦苦找出来的，自然也就为人所能理解，性能好坏，机器学习专家可以“冷暖自知”，灵活调整。

## 1.5 什么是深度学习

后来，机器学习的专家们发现，可以让神经网络自己学习如何抓取数据的特征，这种学习

方式的效果似乎更佳。于是兴起了特征表示学习（Feature Representation Learning）的风潮。这种方式，对数据的拟合也更加灵活好用。于是，人们终于从自寻特征的痛苦生活中解脱了出来。

但这种解脱也需要付出代价，那就是机器自己学习出来的特征，它们存在于机器空间，完全超越了人类理解的范畴，对人而言，这就是一个黑盒世界。为了让神经网络的学习性能表现得更好，人们只能依据经验，不断尝试性地进行大量重复的网络参数调整，同样是苦不堪言。于是，人工智能领域就有了这样的调侃：“有多少人工，就有多少智能”。

因此，你可以看到，在这个世界上，存在着一个“麻烦守恒定律”：麻烦不会减少，只会转移。

再后来，网络进一步加深，出现了多层次的“表示学习”，它把学习的性能提升到另一个高度。这种学习的层次多了，其实也就是套路深了。于是，人们就给它取了一个特别的名称——Deep Learning（深度学习）。

简单来说，深度学习就是一种包括多个隐含层（越多即为越深）的多层感知机。它通过组合低层特征，形成更为抽象的高层表示，用以描述被识别对象的高级属性类别或特征。能自生成数据的中间表示（虽然这个表示并不能被人类理解），是深度学习区别于其他机器学习算法的独门绝技。

深度学习的学习对象同样是数据。与传统机器学习不同的是，它需要大量的数据，也就是“大数据（Big Data）”。有一个观点在工业界一度很流行，那就是在大数据条件下，简单的学习模型会比复杂模型更加有效。而简单的模型，最后会趋向于无模型，也就是无理论。

例如，早在 2008 年，美国《连线》（Wired）杂志主编克里斯·安德森（Chris Anderson）就曾发出“理论的终结（The End of Theory）”的惊人断言<sup>[7]</sup>：“海量数据已经让科学方法成为过去时（The data deluge makes the scientific method obsolete）”。

但地平线机器人创始人（前百度深度学习研究院副院长）余凯先生认为<sup>[8]</sup>，深度学习的惊人进展，是时候促使我们重新思考这个观点了。也就是说，他认为“大数据+复杂（大）模型”或许能更好地提升学习系统的性能。

## 1.6 “恋爱”中的深度学习

法国科技哲学家伯纳德·斯蒂格勒（Bernard Stiegler）认为，人们总以自己的技术和各种物

化的工具，作为自己“额外”的器官，不断地成就自己。按照这个观点，其实，在很多场景下，计算机都是人类思维的一种物化形式。换句话说，计算机的思维（比如各种电子算法）中总能找到人类生活实践的影子。

比如，现在火热的深度学习，与人们的恋爱过程也有相通之处。在知乎社区上，就有人（如 jacky yang）以恋爱为例来说明深度学习的思想，倒也非常传神，如图 1-4 所示。我们知道，男女恋爱大致可分为以下三个阶段。

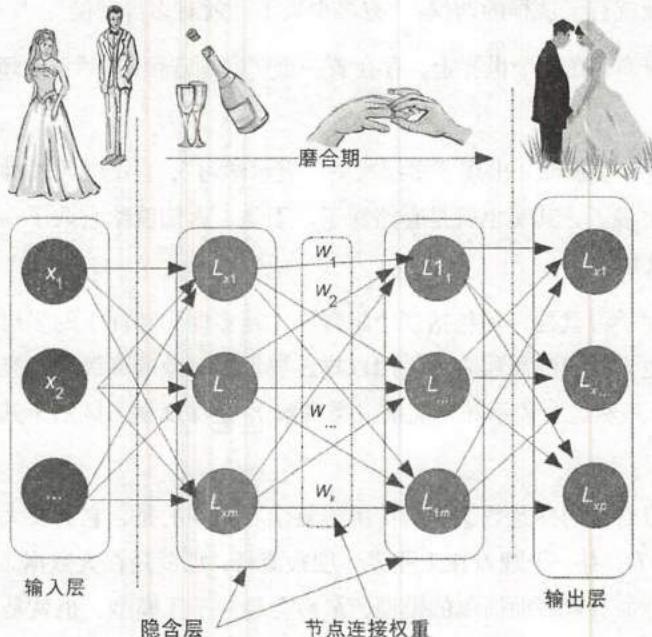


图 1-4 恋爱中的深度学习

第一阶段是初恋期，相当于深度学习的输入层。女孩吸引你，肯定是因为有很多因素的，比如外貌、身高、身材、性格、学历等，这些都是输入层的参数。对于喜好不同的人，他们对输出结果的期望是不同的，自然他们对这些参数设置的权重也是不一样的。比如，有些人是奔着结婚去的，那么他们对女孩的性格可能给予更高的权重。否则，外貌的权重可能会更高。

第二阶段是热恋期，对应于深度学习的隐含层。在这期间，恋爱双方都要经历各种历练和磨合。清朝湖南湘潭人张灿写了一首七绝：

书画琴棋诗酒花，当年件件不离他。

而今七事都更变，柴米油盐酱醋茶。

这首诗说的就是，在过日子的洗礼中，各种生活琐事的变迁。恋爱是过日子的一部分，确实也是如此，需要双方不断磨合。磨合中的权重取舍，就相当于深度学习中隐含层的参数调整，需要不断地训练和修正这些参数！恋爱双方相处，磨合是非常重要的。要怎么磨合呢？光说“我爱你”是苍白的。这就给我们提了个醒，爱她（他）就要多陪陪她（他）。陪陪她（他），就增加了参数调整的机会。参数调整得好，输出的结果才能是你想要的。

第三阶段是稳定期，自然相当于深度学习的输出层。输出结果是否合适，是否达到预期，高度取决于隐含层中的参数“磨合”得怎么样。

## 1.7 深度学习的方法论

在深度学习中，经常有“end-to-end（端到端）”学习的提法，与之相对应的传统机器学习是“Divide and Conquer（分而治之）”。这些都是什么意思呢？

“end-to-end”（端到端）说的是，输入的是原始数据（始端），然后输出的直接就是最终目标（末端），中间过程不可知，也难以知。比如，基于深度学习的图像识别系统，输入端是图片的像素数据，而输出端直接就是或猫或狗的判定。这个端到端就是，像素→判定。

再比如，“end-to-end”的自动驾驶系统<sup>[9]</sup>，输入的是前置摄像头的视频信号（其实也就是像素），而输出的直接就是控制车辆行驶的指令（方向盘的旋转角度）。这个端到端就是，像素→指令。

就此，有人批评深度学习就是一个黑箱（Black Box）系统，其性能很好，却不知道为何好，也就是说，缺乏解释性。其实，这是由深度学习所处的知识象限决定的。从图 1-3 可以看出，深度学习在本质上属于可统计不可推理的范畴。“可统计”是很容易理解的，就是说，对于同类数据，它具有一定的统计规律，这是一切统计学习的基本假设。那“不可推理”又是什么意思？其实就是“剪不断、理还乱”的非线性状态。

从哲学上讲，这种非线性状态是具备了整体性的“复杂系统”，属于复杂性科学范畴。复杂性科学认为，构成复杂系统的各个要素自成体系，但阡陌纵横，其内部结构难以分割。简单来说，对于复杂系统， $1+1 \neq 2$ ，也就是说，一个简单系统加上另外一个简单系统，其效果绝不是两个系统的简单累加效应，参见图 1-5 所示的漫画。因此，我们必须从整体上认识这样的复杂系统。于是，在认知上，就有了从一个系统或状态（end）直接整体迁移到另外一个系统或状态（end）的形态。这就是深度学习背后的方法论。



图 1-5 1 个人+1 个人=?

与之对应的是“Divide and Conquer（分而治之）”，其理念正好相反，在哲学中它属于“还原主义（Reductionism，或称还原论）”。在这种方法论中，有一种“追本溯源”的蕴意包含于其内，即一个系统（或理论）无论多复杂，都可以分解、分解、再分解，直到能够还原到逻辑原点。

在还原主义中就是“ $1+1=2$ ”，也就是说，一个复杂的系统，都可以由简单的系统叠加而成（可以理解为线性系统），如果各个简单系统的问题解决了，那么整体的问题也就得以解决。比如，很多的经典力学问题，不论形式有多复杂，通过不断分解和还原，最后都可以通过牛顿的三大定律得以解决。

从传统的“还原论”出发，单纯的线性组合思维，势必会导致在人工智能系统的设计上功能过于简单。如果我们希望模拟的是一个“类人”的复杂系统（即人工智能系统），自然就无法有效达到目的，具体来说，有如下两个方面的原因：

（1）这个世界（特别是有关人的世界）本身是一个纷繁复杂的系统，问题之间互相影响，

形成复杂的网络，这样的复杂系统很难用一个或几个简单的公式、定理来描述和界定。

(2) 在很多场景下，受现有测量和认知工具的局限，很多问题在认识上根本不具有完备性。因此，难以从一个“残缺”的认知中，提取适用于全局视角的公式和定理。

柏拉图在《理想国》<sup>[10]</sup>中讲到了一个经典比喻——“洞穴之喻 (Allegory of the Cave )”，如图 1-6 所示。设想有一个很深的洞穴，洞穴里有一些囚徒，他们生来就被锁链束缚在洞穴之中，他们背向洞口，头不能转动，眼睛只能看着洞壁。

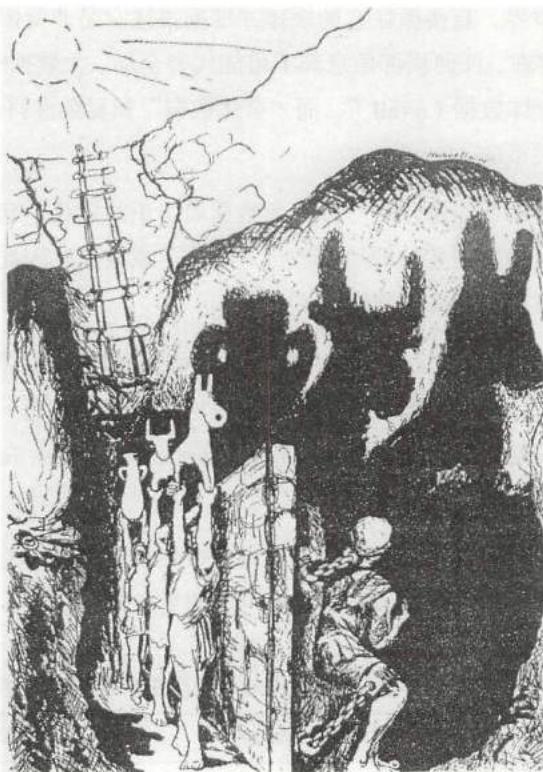


图 1-6 柏拉图的洞穴之喻（图片来源<sup>①</sup>）

在他们后面砌有一道矮墙，墙和洞口之间燃烧着一团火，一些人举着各种器物沿着墙往来走动，如同木偶戏的屏风。当人们扛着各种器具走过墙后的小道时，火光便把那些器物的影像投射到面前的洞壁上。由于这些影像是洞中囚徒们唯一能见到的事物，所以他们即以为这些影像就是这个世界上最真实的事物。

<sup>①</sup> 维基百科：[https://en.wikipedia.org/wiki/Allegory\\_of\\_the\\_Cave](https://en.wikipedia.org/wiki/Allegory_of_the_Cave)。

洞穴人会误把其所能感知到的投影于洞壁的影像（二维世界），当作真实的世界（三维世界），他们怎能基于一个二维世界观测的现象归纳出一个适用于三维世界的规律呢？

但幸运的是，我们已进入大数据时代，它为我们提供了一种认知纷繁复杂世界的无比珍贵的资源——多样而全面的数据。有学者就认为<sup>[11]</sup>，大数据时代之所以具有颠覆性，就是因为目前一切事物的属性和规律，只要通过适当的编码（即数字介质），都可以传递到另外一个同构的事物上，得以无损全息表达。

但对于这个复杂的世界，直接抓住它的规律并准确描述它是非常困难的。在一个复杂系统中，由于非线性因素的存在，任何局部信息都不可能代表全局。大数据时代有一个典型的特征，“不是随机样本，而是全体数据（ $n=all$ ）”，而“全体数据”和复杂性科学中的“整体性”，在一定程度上是有逻辑对应关系的。

深度学习所表现出来的智能也正是“食”大数据而“茁壮成长”起来的，其智能所依赖的人工神经网络模型，还可随数据量的增加而进行“进化”或改良。因此，它可被视为在大数据时代遵循让“数据自己发声”的典范之作。

已有学者论证<sup>[12]</sup>，大数据与复杂性科学在世界观、认识论和方法论等诸多方面都是互通的。复杂性是大数据技术的科学基础，而大数据是复杂性科学的技术实现。深度学习是一种数据饥渴型（data-hungry）的数据分析系统，天生就和大数据捆绑在一起。在某种程度上，大数据是问题，而深度学习就是其中的一种解决方案。

表 1-1 所示的是几个流行的深度学习项目中的参数细节。从表 1-1 可以看到，深度学习网络本身就是一个训练数据量巨大、调节参数数量巨多的复杂网络。

表 1-1 深度学习项目中的数据规模与网络节点参数调整数量

项目名称	VGGNet	DeepVideo	GNMT
项目用途	识别图像并分类	识别视频并分类	翻译
输入数据类型	图像	视频	英语文本
输出数据类型	100 种类别	47 种类别	法语文本
调节参数数量	1.4 亿个	约 1 亿个	3.8 亿个
数据规模	120 万张已分类图片	110 万个已分类视频	600 万语句对，3.4 亿个单词
数据集合	ILSVRC-2012	Sports-1M	WMT'14

在复杂系统中，各要素之间紧密相连，构成了一个巨大的关联网络，存在着各种各样的复

杂联系，各种要素组合起来会带来新结构、新功能的涌现，也就是说，整体往往会大于部分之和。从上面的分析可知，深度学习具备的特征抽取自主性、网络节点的多关联性（难以找到一个线性结构描述上亿级别的参数），“智能”提升的涌现性，这些都表明它是复杂性科学中的一种技术实现。

## 1.8 有没有浅层学习

有了“深度学习”，读者很容易想到，那有没有相应的“浅层学习”呢？答案是，有。传统意义上的人工神经网络，主要由输入层、隐含层、输出层构成，其中隐含层也叫多层感知机（Multi Layer Perceptron）。

正如其名称所示，多层感知机的确也可以是多层的网络，但是层与层之间特征的选择，需要人手动实现，算法的训练难度非常大，故此感知机的层数通常并不多，这些机器学习算法，通常被称为浅层感知机。例如，支持向量机（Support Vector Machine，SVM）、Boosting、最大熵方法（如 Logistic Regression，逻辑回归，简称 LR），这些机器学习模型，隐含层只有一层，甚至连一层都没有（如 LR 算法）。

相比而言，区别于传统的浅层学习，深度学习强调模型结构的深度，隐含层远远不止一层。通常来说，层数更多的网络，通常具有更强的抽象能力（即数据表征能力），也就能够产生更好的分类识别的结果。

2012 年，加拿大多伦多大学的资深机器学习教授杰弗里·辛顿（Geoffery Hinton）团队在 ImageNet 中首次使用深度学习完胜其他团队，那时网络层深度只有个位数。2014 年，谷歌团队把网络做了 22 层，问鼎当时的 ImageNet 冠军。到了 2015 年，微软研究院团队设计的基于深度学习的图像识别算法 ResNet，把网络层做到了 152 层。很快，在 2016 年，商汤科技更是叹为观止地把网络层做到了 1207 层<sup>[13]</sup>，这可能是当前在 ImageNet 上最深的一个网络。我们不禁要问，这深度学习，到底“深”几许啊？

如果深度神经网络的层数再往“深处”做，也可能会达到 2000 层、3000 层，但我们会发现，任何时候，都可能存在“过犹不及”的情况。因为这种极深的架构叠加，带来的通信开销会淹没性能的提升。

因此，我们需要清醒地认识到，对于构建出来的深度模型，“深度”仅仅是手段，“表示学习”才是目的。深度学习通过自动完成逐层特征变换，将样本在原空间的特征表示变换到一个

新特征空间，从而使分类或预测更加准确。与原来的“浅层网络”的人工提取特征的方法相比，深度学习利用了大数据来自动获得事物特征，让“数据自己说话”，因此，更能够刻画数据丰富的内在信息。

## 1.9 本章小结

在本章，我们学习了机器学习的核心要素，那就是通过运用数据，依据统计或推理的方法，让计算机系统的性能得到提升。而深度学习，则是把由人工选取对象特征，变为通过神经网络自己选取特征，为了提升学习的性能，神经网络表示学习的层次较多（较深）。

以上仅仅给出机器学习和深度学习的概念性描述，在下一章中，我们将介绍机器学习的形式化表示，以及传统机器学习和深度学习的不同之处等。

## 1.10 请你思考

通过本章的学习，请你思考如下问题：

- (1) 在大数据时代，你是赞同科技编辑出身的克里斯·安德森的观点呢（仅仅需要小模型），还是更认可工业界大神余凯先生的观点呢（还是需要复杂模型）？为什么？
- (2) 你认为用恋爱的例子比拟深度学习贴切吗？为什么？
- (3) 为什么非要用“深度”学习，“浅度”不行吗？

## 参考资料

- [1] Tom Mitchell. 曾华军等译. 机器学习[M]. 北京：机械工业出版社, 2002.
- [2] Vladimir N. Vapnik. 张学工译. 统计学习理论的本质[M]. 北京：清华大学出版社, 2000.
- [3] Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning[M]. 北京：世界图书出版公司, 2015.
- [4] 于剑. 机器学习：从公理到算法[M]. 北京：清华大学出版社, 2017.
- [5] 张玉宏. 云栖社区. AI 不可怕，就怕 AI 会画画——这里有一种你还不知道的“图”灵

测试. <https://yq.aliyun.com/articles/74383>.

[6] 李航. 统计学习方法[M]. 北京: 清华大学出版社, 2012.

[7] Anderson C. The end of theory: The data deluge makes the scientific method obsolete[J]. Wired magazine, 2008, 16(7): 16-07.

[8] 余凯, 贾磊, 陈雨强, 等. 深度学习的昨天、今天和明天[J]. 计算机研究与发展, 2013, 50(9):1799-1804.

[9] Bojarski M, Del Testa D, Dworakowski D, et al. End to end learning for self-driving cars[J]. arXiv preprint arXiv:1604.07316, 2016.

[10] 柏拉图. 黄颖译. 理想国[M]. 北京: 中国华侨出版社, 2012.

[11] 李德伟等. 大数据改变世界[M]. 北京: 电子工业出版社, 2013.

[12] 黄欣荣. 从复杂性科学到大数据技术[J]. 长沙理工大学学报(社会科学版), 2014, 29(2): 5-9.

[13] 胡祥杰, 零夏. 1200 层神经网络夺冠 ImageNet, 深度学习越深越好? <https://www.sypopo.com/post/6KQLelDer4/>.

Chapter two

## 第2章 人工“碳”索意犹尽， 智能“硅”来未可知

现在的人工智能，大致就是用“硅基大脑”模拟或重现“碳基大脑”的过程。那么，在未来会不会出现“碳硅合一”的大脑或者全面超越人脑的“硅基大脑”呢？专家们的回答是，会的。而由深度学习引领的人工智能，正在开启这样的时代。



在第1章中，我们仅从概念上描述了机器学习、深度学习等，在本章中，我们将给出更加准确的形式化描述。经常听到别人说人工智能如何、深度学习怎样，那么它们之间有什么关系呢？在本章中，我们首先从宏观上谈谈人工智能的“江湖定位”和深度学习的归属。然后从微观上聊聊机器学习的数学本质是什么，以及我们为什么要用神经网络。

## 2.1 信数据者得永生吗

以凯文·凯利（Kevin Kelly）为代表的数据主义认为，宇宙是由数据流构成的，任何现象或实体的价值，都体现在对数据处理的贡献度上。

在《未来简史》<sup>[1]</sup>一书里，新锐历史学家尤瓦尔·赫拉利（Yuval Harari）说，根据数据主义的观点，可把整个人类族群视为一个分布式的数据处理系统。在这个系统中，每个单独的个体都是其中的一个芯片。为了改善这个系统的性能，需要在如下4个方面不断改进：

（1）增加处理器的数量。人多力量大，就是这个观点的通俗版本。

（2）增加处理器的种类。其实就是要精细化社会分工，各司其职，各负其责。农民干不了祭司的活，祭司也吃不了农民的苦。

（3）增加处理器自己的连接。这说的是，孤木难以成林，人类只有互通有无，方能做大做强。目前我国主导的“一带一路”倡议，究其本质，就是达到了添加连接的目的。

（4）增加现有连接的流通程度。如果数据无法自由流动，光有连接也是无济于事的。

可能你会困惑，说来说去，这和“人工智能”“深度学习”到底有什么关系呢？莫急，它们之间还真有点关系，至少在逻辑上是有的，且听我慢慢分解。

目前，我们正处于一个大数据时代，不管你认为这是炒作，还是信它为事实，有一点可以肯定，因为数据流动量过大，人类已经无法将数据转化为信息，更不用说从庞大的数据中提炼出知识和智能。

这是一个多么大的缺陷啊！

科技哲学家伯纳德·斯蒂格勒（Bernard Stiegler）认为，人，天然就是一种缺陷存在，但恰恰因为这种本质的缺陷存在，技术才有其存在的根本意义。那什么是技术呢？技术就是生命用来进化的支架。

想一想，人类为什么发明马车、汽车、飞机，这是因为人跑得不够快啊（缺陷！）。那人类



为什么要发明望远镜、显微镜，这是因为人看得不够远、不够真切（缺陷！）。而人类为什么要发明计算机，这是因为记不牢、算不快、扒拉算盘好烦啊（还是缺陷！）。也正是因为这个缺陷的底层逻辑，在条件成熟时，人们就有一种强烈的欲望，即热切地想用外部的“电子算法”，替换自己大脑中的“生物算法”。

而这个强烈欲望的结果，就诞生了今天的“人工智能”（Artificial Intelligence, AI）。

## 2.2 人工智能的“江湖定位”

从宏观上来看，人类科学和技术的发展，大致都遵循着这样的规律：现象观察、理论提取和人工模拟（或重现）。人类“观察大脑”的历史由来已久，但由于对大脑缺乏“深入认识”，常常“绞尽脑汁”，也难以“重现大脑”。

直到20世纪40年代以后，脑科学、神经科学、心理学及计算机科学等众多学科取得了一系列重要进展，使得人们对大脑的认识相对深入，从而为科研人员从“观察大脑”到“重现大脑”搭起了桥梁，哪怕这个桥梁到现在还仅仅是一个并不坚固的浮桥（参见图2-1）。

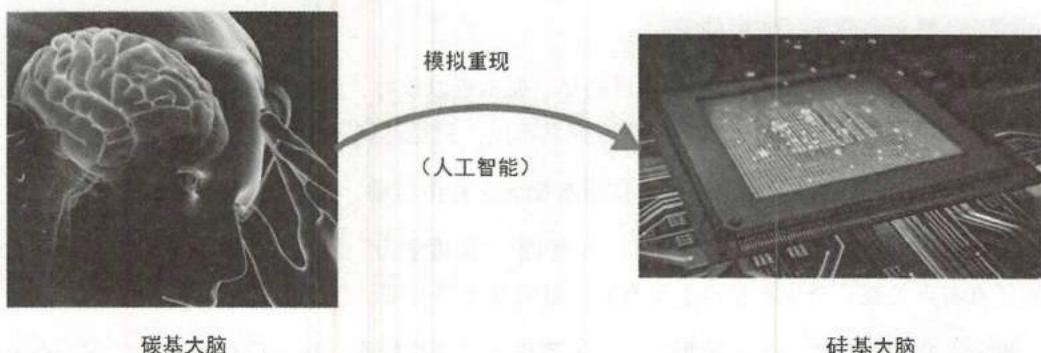


图2-1 人工智能的本质

而所谓的“重现大脑”，在某种程度上，就是目前的研究热点——人工智能。简单来讲，人工智能就是为机器赋予与人类类似的智能。由于目前机器的核心部件是由晶体硅构成的，所以可归属为“硅基大脑”。而人类的大脑主要由碳水化合物构成，因此可称之为“碳基大脑”。

那么，现在的人工智能，简单来讲，大致就是用“硅基大脑”模拟或重现“碳基大脑”。那么，在未来会不会出现“碳硅合一”的大脑或者全面超越人脑的“硅基大脑”呢？

有人认为，在很大程度上，这个答案可能是“会的”！比如，未来预言大师雷·库兹韦尔（Ray



Kurzweil) 预测，到 2045 年，人类的“奇点 (Singularity)”<sup>[2]</sup>时刻就会临近。这里的“奇点”是指，人类与其他物种（物体）的相互融合。确切地说，是指硅基智能与碳基智能兼容的那个奇妙时刻。届时，严格意义上的人类将不复存在！

## 2.3 深度学习的归属

在当下，虽然深度学习领跑人工智能。但事实上，人工智能的研究领域很广，包括机器学习、计算机视觉、专家系统、规划与推理、语音识别、自然语音处理和机器人等。而机器学习又包括深度学习、监督学习、无监督学习等。简单来讲，机器学习是实现人工智能的一种方法，而深度学习仅仅是实现机器学习的一种技术而已（参见图 2-2）。

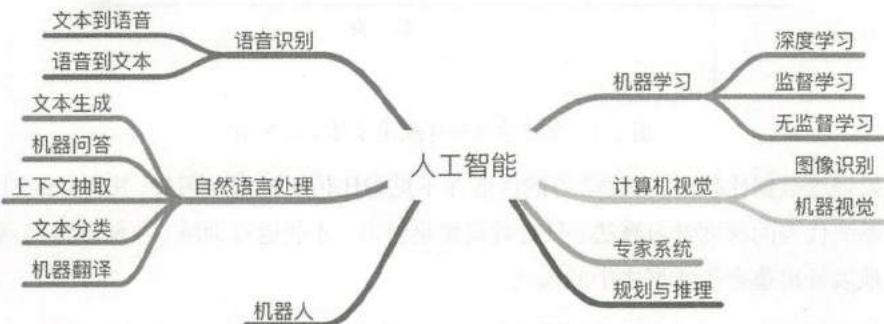


图 2-2 深度学习的“江湖地位”

需要说明的是，对人工智能做任何形式的划分，都可能是有缺陷的。在图 2-2 中，人工智能的各类技术分支，彼此泾渭分明。但实际上，它们之间却可能阡陌纵横，比如，深度学习可以是无监督的，语音识别也可以用深度学习来完成。再比如，图像识别、机器视觉更是当前深度学习的拿手好戏。

一言以蔽之，人工智能并不是一个有序的树，而是一个彼此缠绕的灌木丛。有时候，一个分藤蔓比另一个分藤蔓生长得快，并且处于显要地位，那么它就是当时的研究热点。深度学习的前身——神经网络的发展，就经历了这样的几起几落。当下，深度学习如日中天，但会不会也有“虎落平阳”的一天呢？从事物的发展规律来看，这一天肯定会到来！

在图 2-2 中，既然我们把深度学习与传统的监督学习和无监督学习单列出来，自然是有一定道理的。这是因为，深度学习是高度数据依赖型的算法。它的性能，通常是随着数据量的增加而不断增强的，也就是说，它的可扩展性（Scalability）显著优于传统的机器学习算法



(参见图 2-3 所示)。

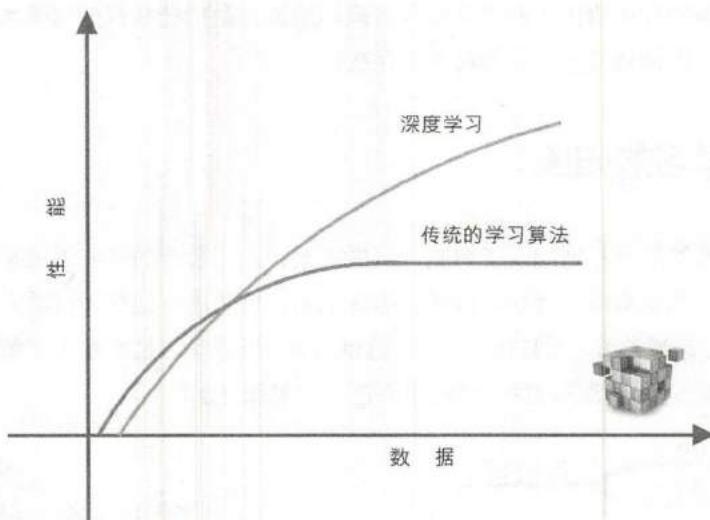


图 2-3 深度学习和传统学习算法的区别

但如果训练数据比较少，深度学习的性能并不见得比传统机器学习好。其潜在的原因在于，作为复杂系统代表的深度学习算法，只有数据量足够多，才能通过训练，在深度神经网络中“恰如其分”地表征出蕴含于数据之中的模式。

不论是机器学习，还是它的特例深度学习，大致都存在两个层面的分析（参见图 2-4）：

(1) 面向过去（对收集到的历史数据进行训练），发现潜藏在数据之下的模式，我们称之为描述性分析（Descriptive Analysis）。

(2) 面向未来，基于已经构建的模型，对于新输入的数据对象实施预测，我们称之为预测性分析（Predictive Analysis）。

前者主要使用了“归纳”方法，而后者侧重于“演绎”。对历史对象的归纳，可以让人们获得新洞察、新知识，而对新对象实施演绎和预测，可以使机器更加智能，或者说让机器的某些性能得以提高。二者相辅相成，缺一不可。

在前面的部分，我们给出了机器学习的概念性描述，下面我们将给出机器学习的形式化定义。



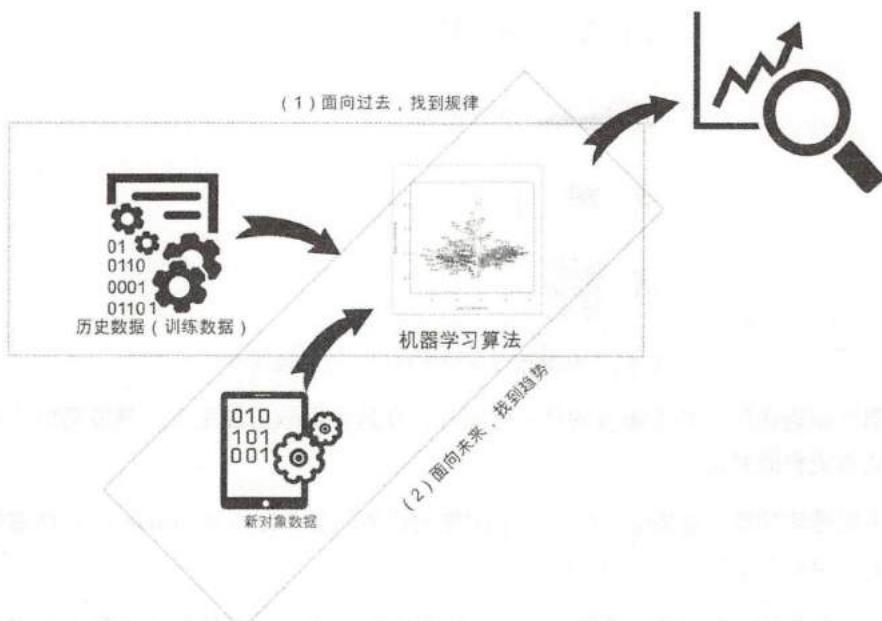


图 2-4 机器学习的两层作用

## 2.4 机器学习的形式化定义

在《未来简史》一书中，赫拉利还说，根据数据主义的观点，人工智能实际上就是找到一种高效的“电子算法”，用以代替或在某项指标上超越人类的“生物算法”。那么，任何一个“电子算法”都要实现一定的功能（Function），才有意义。

在计算机术语中，将“Function”翻译成“函数”，这多少有点“词不达意”，因为它并没有达到“信、达、雅”的标准，除了给我们留下一个抽象的概念之外，几乎什么也没有剩下。但这一称呼已被广泛接受，我们也只能约定俗成地把“功能”叫作“函数”。

根据台湾大学李宏毅博士的通俗说法，所谓机器学习，在形式上可近似等同于，在数据对象中通过统计或推理的方法，寻找一个有关特定输入和预期输出的功能函数  $f$ （参见图 2-5）。通常，我们把输入变量（特征）空间记作大写的  $X$ ，而把输出变量空间记为大写的  $Y$ 。那么所谓的机器学习，在形式就是完成如下变换： $Y=f(X)$ 。

在这样的函数中，针对语音识别功能，如果输入一个音频信号，那么这个函数  $f$  就能输出诸如“你好”“How are you?”等这类识别信息。



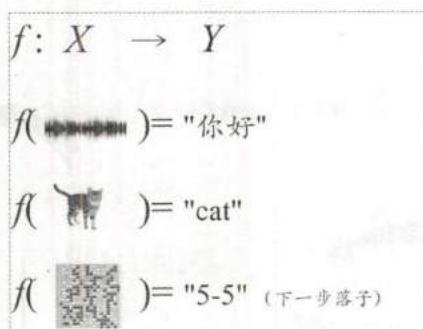


图 2-5 机器学习近似于找一个好用的函数

针对图片识别功能，如果输入的是一张图片，在这个函数的加工下，就能输出（或称识别出）一个或猫或狗的判定。

针对下棋博弈功能，如果输入的是一个围棋的棋谱局势（比如 AlphaGo），它能输出这盘围棋下一步的“最佳”走法。

而对于具备智能交互功能的系统（比如微软的小冰），当我们给这个函数输入诸如“*How are you?*”，它就能输出诸如“*I am fine, thank you, and you?*”等智能的回应。

每个具体的输入都是一个实例（Instance），它通常由特征向量构成。在这里，将所有特征向量存在的空间，称为特征空间（Feature Space），特征空间的每一个维度，对应于实例的一个特征。

但问题来了，这样“好用的”函数并不那么好找。当输入一只猫的图像后，这个  $f$  函数并不一定就能输出一只猫，可能它会错误地输出为一条狗或一条蛇。

这样一来，我们就需要构建一个评估体系，来辨别函数的好坏。当然，这中间自然需要训练数据（Training Data）来“培养”函数的好品质（参见图 2-6）。

在第 1 章中我们提到，学习的核心就是改善性能，在图 2-6 中，通过训练数据，我们把  $f_1$  改善为  $f_2$  的样子，性能（判定的准确度）提高了，这就是学习。（需要注意的是， $f_2$  的性能并非就必须打满分，比如它依然把“狗”误判为“猫”，但这无损于“学习”的定义，因为  $f_2$  的性能高过  $f_1$ ）。很自然，如果这个学习过程是在机器上完成的，那就是“机器学习”了。

具体来说，机器学习要想做得好，需要走好三大步：

- (1) 如何找一系列的函数来实现预期的功能，这是建模问题。
- (2) 如何找出一系列评价标准来评估函数的好坏，这是评估问题。



## 第2章 人工“碳”索意犹尽，智能“硅”来未可知 | 23

(3) 如何快速找到性能最佳的函数，这是优化问题（比如，机器学习中随机梯度下降法，干的就是这个活）。

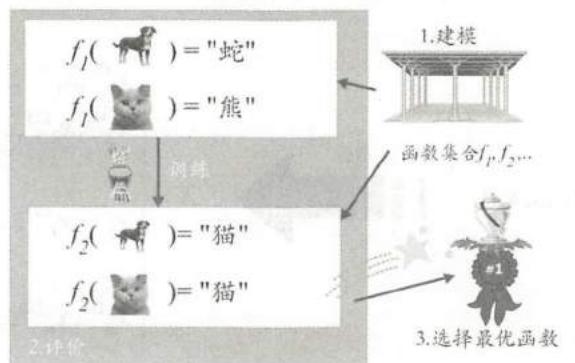


图 2-6 机器学习的三步走

习惯上，我们把具体的输入、输出变量（而非空间）用小写的  $x$  和  $y$  表示。变量既可以是标量 (scalar)，也可以是向量 (vector)。标量是指只有大小，没有方向的量。而向量则是既有大小又有方向的量（在物理学或工学中，向量通常也被称为矢量）。通常一个标量是向量空间的一个域值元素。比如，在后面的章节中我们经常提及的梯度，就是一个向量，它既有大小，也有方向。为了表明向量有方向，有时候我们会在向量符号上方加上一个箭头 ( $\rightarrow$ )，如  $\vec{x}$ 、 $\vec{e}$  等。

除了特殊说明外，本书所言向量均为列向量。例如输入实例  $x$  的特征向量可记为如图 2-7 所示的形式。标准的写法自然是如图 2-7 (a) 所示，但这种写法比较占空间，因此我们通常采用转置 (Transpose) 的写法，如图 2-7 (b) 所示，图中的上标 “T” 就是转置的英文首字母。

该图展示了输入特性的两种表示方法：

(a) 列向量表示：  

$$x = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \cdots \\ x^{(i)} \\ \cdots \\ x^{(n)} \end{bmatrix}$$
 其中，上方标注“某一列”，并有一个指向第二行的箭头。

(b) 转置表示：  
 中间是一个大箭头，上方标注“转置”。右侧展示了转置后的向量形式：  

$$x = (x^{(1)}, x^{(2)}, \dots, x^{(i)}, \dots, x^{(n)})^T$$

图 2-7 输入的特性向量



这里的  $x^{(i)}$  表示的是输入变量  $x$  的第  $i$  个特征。需要特别注意的是，当输入变量有多个时，我们用  $x_i$  表示。如此一来， $x_j^{(i)}$  就表示第  $j$  个变量的第  $i$  个特征，如图 2-8 所示。

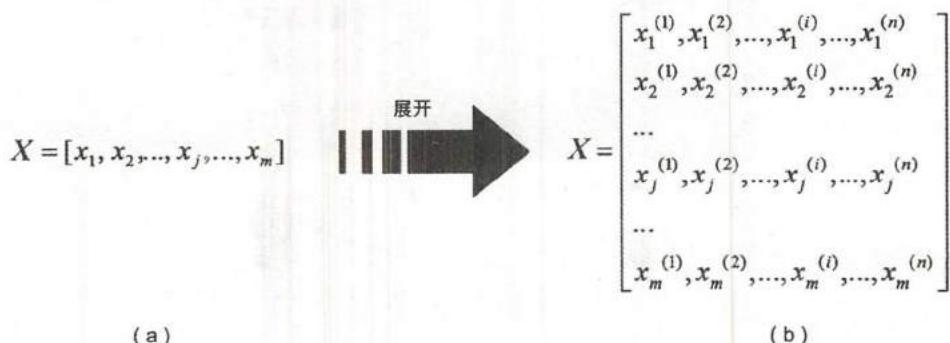


图 2-8 输入变量矩阵

对于有监督学习来说，所构建的模型通常在训练数据（Training Data）集合中学习，调整模型参数，在测试数据（Test Data）集合中进行预测验证。训练数据通常是输入与输出成对出现的（这里，输出信号有时也被称为“教师信号”，它通过损失函数来“调教”模型中的参数）。因此，训练集合通常用如公式（2-1）所示的方式进行描述。

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_j, y_j), \dots, (x_m, y_m)\} \quad (2-1)$$

输入变量空间  $X$  和输出变量空间  $Y$  可以有不同的类型，它们既可以是连续的，也可以是离散的。通常，人们根据输入和输出变量的类型不同，给预测任务赋予不同的名称。比如，如果输入变量和输出变量均为连续变量，那么这样的预测任务就称为回归（Regression）。如果输出变量为有限的几个离散值，那么这样的预测任务就称为分类（Classification）。如果输入变量和输出变量均为变量序列，那么这样的预测任务就称为标注（Tagging），我们可以认为标注问题是分类问题的一个推广<sup>[3]</sup>。

## 2.5 为什么要用神经网络

我们知道，深度学习的概念源于人工神经网络的研究。包含多个隐含层的多层感知机就是一种深度学习结构。所以说到底，就不能不提神经网络。



那么什么是神经网络呢？有关神经网络的定义有很多。这里我们给出芬兰计算机科学家托伊沃·科霍宁（Teuvo Kohonen）的定义（他以提出“自组织神经网络”而名扬人工智能领域）：“神经网络是一种由具有自适应性的简单单元构成的广泛并行互联的网络，它的组织结构能够模拟生物神经系统对真实世界所做出的交互反应。”

在生物神经网络中，人类大脑通过增强或者弱化突触进行学习的方式，最终会形成一个复杂的网络，形成一个分布式特征表示（Distributed Representation）。

在人工智能领域，正是受到生物神经网络的启发，自20世纪80年代起，人工神经网络（Artificial Neural Network, ANN）开始兴起，而且在很长一段时间内都是人工智能领域的研究热点。

作为处理数据的一种新模式，人工神经网络的强大之处在于，它拥有很强的学习能力。在得到一个训练集合之后，通过学习，提取到所观察事物的各个部分的特征，特征之间用不同网络节点链接，通过训练链接的网络权重，改变每一个链接的强度，直到顶层的输出得到正确的答案（参见图2-9）。

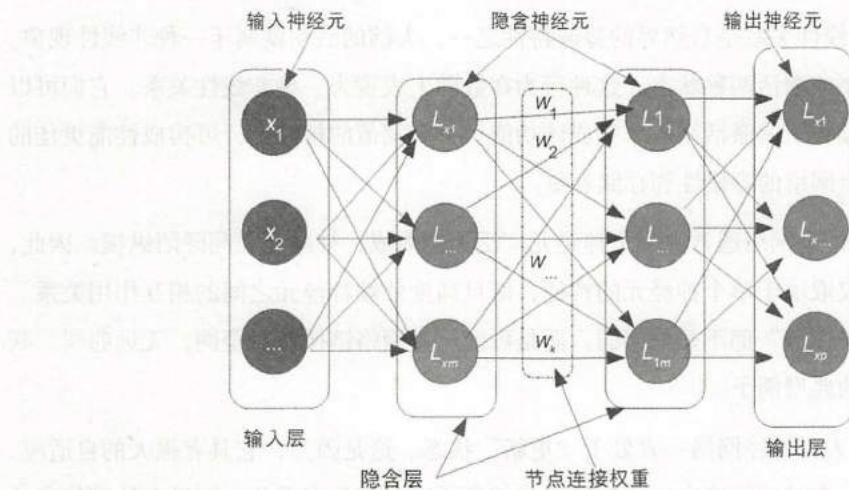


图2-9 人工神经网络

在机器学习中，我们常常提到神经网络，实际上是指神经网络学习。学习是大事，切不可忘！

那为什么我们要用神经网络学习呢？这个原因说起来有点“情非得已”。

我们知道，在人工智能领域，有两大主流门派。第一个门派是符号主义。符号主义认为，



知识是信息的一种表达形式，人工智能的核心任务就是处理好知识表示、知识推理和知识运用。这个门派的核心方法论是，自顶向下设计规则，然后通过各种推理，逐步解决问题。很多人工智能的先驱（比如 CMU 的赫伯特·西蒙）和逻辑学家们，很喜欢这种方法。但这个门派的发展，目前看来并不太好。未来会不会“峰回路转”，现在还不好说。

还有一个门派试图编写一个通用模型，然后通过数据训练，不断改善模型中的参数，直到输出的结果符合预期，这个门派就是连接主义。连接主义认为，人的思维就是某些神经元的组合。因此，可以在网络层次上模拟人的认知功能，用大脑的并行处理模式，来表征认知过程。这种受神经科学启发的网络，就是前面提到的人工神经网络（ANN）。这种方法的升级版就是目前非常流行的深度学习。

## 2.6 人工神经网络的特点

从上面的描述可知，人工神经网络是一种非线性、自适应的信息处理系统，该系统由大量彼此相连但功能简单的处理单元构成。一般说来，人工神经网络具有四“非”特征<sup>[4]</sup>。

**(1) 非线性。**非线性关系是自然界的普遍特性之一，大脑的活动就属于一种非线性现象。人工神经元可处于抑制或激活两种状态，这种行为在数学上表现为一种非线性关系。它们可以通过具有阈值（或称偏置）的激活函数来完成该功能。具有阈值的神经元，可构成性能更佳的神经网络，可提高整个网络的容错性和存储容量。

**(2) 非局限性。**神经网络通常由多个神经元广泛连接而成，神经元之间阡陌纵横。因此，系统的整体行为，不仅取决于单个神经元的特征，而且高度依赖神经元之间的相互作用关系。任何一个神经元的“作用域”都不是局部的，而是可能通过网络链接波及全网，无远弗届。联想记忆就是非局限性的典型例子。

**(3) 非常定性。**人工神经网络一直处于“更新”状态。这是因为，它具有强大的自适应、自组织、自学习能力。在神经网络中，不但处理的信息可以是变化多端的，而且在处理信息的同时，非线性动力系统本身可能也在演化（比如网络连续权值的迭代更新）。

**(4) 非凸性。**一个系统的演化方向，在一定条件下取决于某个特定的状态函数，如目标函数和激活函数。当前的神经网络，基本都放弃了线性激活函数，通常采用诸如 Sigmoid、Tanh、ReLU 等非线性激活函数，这就导致神经网络的目标函数具有非凸性。所谓非凸性，是指函数



可能有多个极值。极值通常对应于系统比较稳定的状态，多极值表明系统具备多个较稳定的平衡态，而多个平衡态将导致系统演化出多样性。

## 2.7 什么是通用近似定理

前面我们提到，机器学习在本质上就是找到一个好用的函数。而人工神经网络最牛的地方可能就在于，它可以在理论上证明：“一个包含足够多隐含层神经元的多层前馈网络，能以任意精度逼近任意预定的连续函数<sup>[5]</sup>”。

这个定理也被称为通用近似定理（Universal Approximation Theorem）。这里的“Universal”，也有人将其翻译成“万能的”，由此可以看出，这个定理的能量有多大。

通用近似定理告诉我们，不管函数  $f(x)$  在形式上有多复杂，我们总能确保找到一个神经网络，对任何可能的输入  $x$ ，以任意高的精度近似输出  $f(x)$ （参见图 2-10）。即使函数有多个输入和输出，即  $f = f(x_1, x_2, x_3, \dots, x_m)$ ，通用近似定理的结论也是成立的。换句话说，神经网络在理论上可近似解决任何问题，这就厉害了！有关神经网络可以计算任何函数的可视化证明，感兴趣的读者可以参阅迈克尔·尼尔森（Michael Nielsen）的博客文章<sup>[6]</sup>。

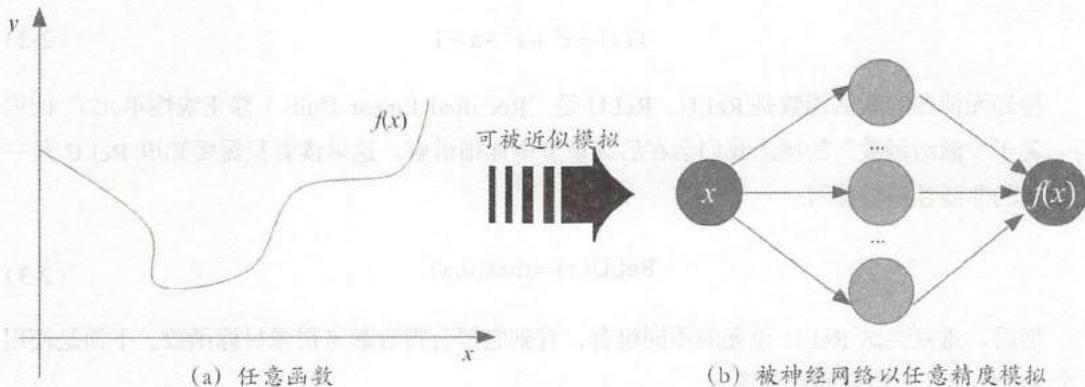


图 2-10 通用近似定理

使用这个定理时，需要注意如下两点<sup>[7]</sup>：

（1）定理说的是，可以设计一个神经网络尽可能好地去“近似”某个特定函数，而不是说“准确”计算这个函数。我们通过增加隐含层神经元的个数来提升近似的精度。

（2）被近似的函数，必须是连续函数。如果函数是非连续的，也就是说有极陡跳跃的函数，



那神经网络就“爱莫能助”了。

即使函数是连续的，有关神经网络能不能解决所有问题，也是有争议的。原因很简单，就如同那句玩笑话“理想很丰满，现实很骨感”，通用近似定理在理论上是一回事，而在实际操作中又是另外一回事。

比如，深度学习新秀、生成对抗网络（GAN）的提出者伊恩·古德费洛（Ian Goodfellow）就曾说过：“仅含有一层的前馈网络，的确足以有效地表示任何函数，但是，这样的网络结构可能会格外庞大，进而无法正确地学习和泛化（A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly）。”

Goodfellow 的言外之意是说，“广而浅薄”的神经网络在理论上是万能的，但在实践中却不是那么回事。因此，网络往“深”的方向去做才是正途。事实上，“通用近似定理”1989 年就被提出了，到 2006 年深度学习开始厚积薄发，这期间神经网络并没有因为这个理论而得到蓬勃发展。因此，从某种程度上验证了 Goodfellow 的判断。

深度学习工程师布伦丹·福蒂内（Brendan Fortuner）在其博客<sup>[8]</sup>中比较形象地说明了“通用近似定理”的长处和短处，他把目标函数设为：

$$f(x) = x^3 + x^2 - x - 1 \quad (2-2)$$

神经元使用的激活函数是 ReLU。ReLU 是“Rectified Linear Units（修正线性单元）”的缩写。关于“激活函数”的概念我们会在后续章节中详细讲解。这里读者只需要知道 ReLU 是一个简单的非线性函数即可：

$$\text{ReLU}(x) = \max(0, x) \quad (2-3)$$

然后，重复尝试 ReLU 单元的不同组合，直到它拟合得看起来很像目标函数。下面是使用 6 个 ReLU 神经元组合的示意图。

这 6 个神经元，分别用公式（2-4）~公式（2-9）所示。

$$n_1(x) = \text{ReLU}(-5x - 7.7) \quad (2-4)$$

$$n_2(x) = \text{ReLU}(-1.2x - 1.3) \quad (2-5)$$

$$n_3(x) = \text{ReLU}(1.2x + 1) \quad (2-6)$$

$$n_4(x) = \text{ReLU}(1.2x - 0.2) \quad (2-7)$$

$$n_5(x) = \text{ReLU}(2x - 1.1) \quad (2-8)$$

$$n_6(x) = \text{ReLU}(5x - 5) \quad (2-9)$$

在上述公式中，诸如“-5”“-1.2”等数值，是输入神经元与隐含层神经元之间的连接权值，而诸如“-7.7”“-1.3”等数值，是隐含层神经元的偏置（bias）。本来这些参数都应该通过不断地训练学习而得来，这里为了说明问题，就直接给出了。

然后把这些神经元组合起来，就形成了输出函数  $f(x)$ ，如公式（2-10）所示。

$$f(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x) \quad (2-10)$$

事实上，神经元前面的系数“-1”“-1”“-1”“1”“1”“1”分别是隐含层神经元和输出神经元之间的连接权重值，它们也是需要学习才能得到的，这里也直接给出了。

通过上述工作，我们来看看如图 2-11 所示的神经网络拟合的效果如何。拟合的效果参见图 2-12，从该图中可以看出，神经网络的确可以用少量神经元和单个隐含层来为非平凡函数（Non-trivial Functions）建模，如果我们把神经网络的参数调整得更加“细腻”，那么这个“近似”将会更加逼真。

细心的读者可能发现了图 2-12 中的端倪，那就是这个函数“近似”的区间在[-2,2]之间。也就是说，在这个区间中，神经网络“近似”的效果，看起来还不错，但经不起区间范围的放大。

一旦区间放大，上述神经网络的近似效果就会“大相径庭”。换句话说，如果不重新训练且不想增加网络隐含层神经元，那么就别指望它可以“泛化”支撑其他输入。

这里的泛化（Generalization）是指，训练好的机器学习模型对新样本的适应能力。如果所设计的模拟，仅仅对训练数据集合有效，而对训练集合之外的元素效率较低，那么就称这样的模型泛化能力差，或陷入了过拟合（Overfitting）状态。

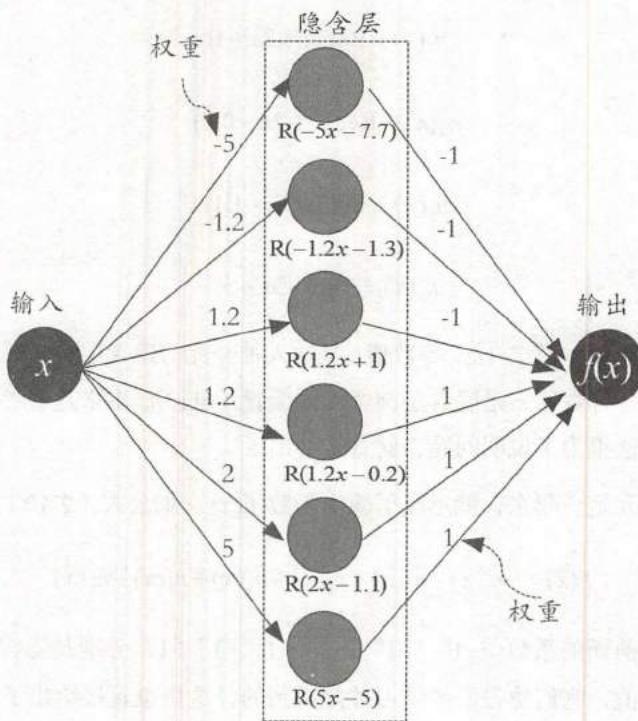


图 2-11 神经网络拟合示意图

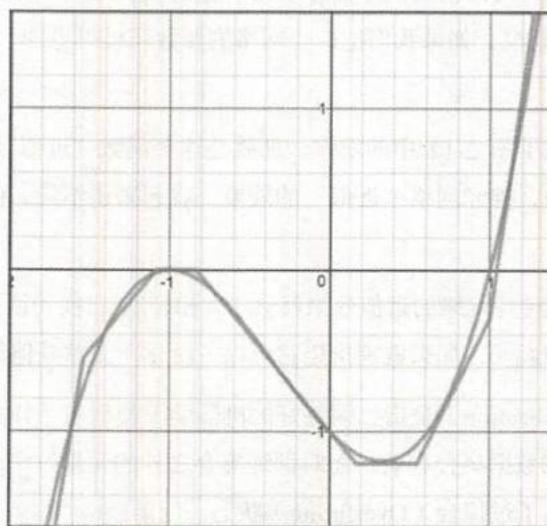


图 2-12 神经网络模拟函数示意图（来自参考资料[9]）

## 2.8 本章小结

在本章中，我们首先谈了人工智能的“江湖定位”，然后指出深度学习仅是人工智能研究领域中很小的一个分支。接着我们给出了机器学习的形式化定义，回答了为什么人工神经网络能“风起云涌”。简单来说，在理论上可以证明，它能以任意精度逼近任意形式的连续函数，而机器学习的本质，不就是要找到一个好用的函数吗？最后，我们介绍了人工神经网络的四“非”特征：非线性、非局限性、非常定性和非凸性。

在下一章中，我们将轻松解读机器学习的一些重要基本原理。

## 2.9 请你思考

学完前面的知识，请你思考如下问题（掌握思辨能力，或许比知识本身更重要）：

（1）库兹韦尔把“奇点”当作一个极佳的比喻，其用意在于说明，当智能机器的能力跨越某一个临界点后，人类的知识单元、链接数目以及思考能力，都会产生质的飞跃，我们目前所熟知的人类的社会、艺术和生活模式，都将不复存在。你认可库兹韦尔的“到2045年人类的奇点时刻就会临近”的观点吗？为什么？库兹韦尔的预测属于科学的范畴吗？（提示：可以从哲学家波普尔的科学评判的标准——是否具备可证伪性来分析。）

（2）深度学习的性能，高度依赖于训练数据量的多少？这个特性是好还是坏？

（3）伊隆·马斯克经常用“第一性原理”来解释他的创业思路。比如，他认为目前传统发射火箭的大部分成本，实际上都是与人有关的成本（比例高达98%）。而通过“第一性原理”进行剥离分析，发射火箭的成本不过是千万美元的级别。那么，你认为“深度学习”的“第一性原理”是什么？未来深度学习应该依据“第一性原理”向哪个方向发展？

## 参考资料

- [1] 尤瓦尔·赫拉利. 林俊宏译. 未来简史[M]. 北京: 中信出版社, 2017.
- [2] 雷·库兹韦尔. 李庆诚等译. 奇点临近[M]. 北京: 机械工业出版社, 2012.
- [3] 李航. 统计学习方法[M]. 北京: 清华大学出版社, 2012.

- [4] 于剑. 机器学习——从公理到算法[M]. 北京：清华大学出版社, 2017.
- [5] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators[J]. Neural Networks, 1989, 2(5):359-366.
- [6] Michael A. Nielsen. A visual proof that neural nets can compute any function. <http://neuralnetworksanddeeplearning.com/chap4.html>.
- [7] Michael A. Nielsen. Neural Networks and Deep Learning[M]. Determination Press, 2015.
- [8] Brendan Fortuner. Can neural networks solve any problem? Medium. <https://medium.com/towards-data-science/can-neural-networks-really-learn-any-function-65e106617fc6>
- [9] <https://www.desmos.com/calculator/cfvfjusqmq>.

## Chapter three

## 第3章 “机器学习”三重门，“中庸之道”趋若人

监督学习

半监督学习

王国维先生在《人间词话》里提到，人生有三重境界。第一重境界是“立”、第二重境界是“守”、第三重境界是“得”。对应的，“机器学习”也有三大类算法：监督学习、非监督学习和半监督学习。而有着“中庸之道”的半监督学习很有可能成为未来机器学习的大趋势。

仅供非商业用途或交流学习使用

在第2章中，我们讨论了机器学习的形式化描述和神经网络的基本概念。下笔之处，尽显“神经”。当然这里所谓的“神经”，是指我们把不同领域的知识，以天马行空的方式糅和在一起，协同提升认知水平。其实，这不也正是深度学习的前沿方向之一——多任务迁移学习（Multi-Task and Transfer Learning）要干的事情吗？

下面，让我们继续“神经”下去，首先聊聊机器学习的几大流派，然后以“中庸之道”来看看机器学习的发展方向。小时候，我们都学过《三字经》，其中有一句“性相近，习相远。”说的就是，“人们生下来的时候，性情都差不多，但由于后天的学习环境不一样，性情也就有了千差万别。”

其实，这句话用在机器学习领域也是适用的。机器学习的学习对象是数据，数据是否有标签，就是机器学习所处的环境，环境不一样，其表现出来的“性情”也有所不同，机器学习大致可分为3大类：监督学习、非监督学习、半监督学习。下面分别进行介绍。

## 3.1 监督学习

### 3.1.1 感性认知监督学习

用数据挖掘领域大家韩家炜教授的观点<sup>[1]</sup>来说，所有的监督学习（Supervised Learning），基本上都是“分类（Classification）”的代名词。它从有标签的训练数据中学习模型，然后给定某个新数据，利用模型预测它的标签。这里的标签，其实就是某个事物的分类。

比如，小时候父母告诉我们某个动物是猫、是狗或是猪，然后在我们的大脑里就会形成或猫或狗或猪的印象（相当于模型构建），然后面前来了一条“新”小狗，如果你能叫出来“这是一只小狗”，那么恭喜你，标签分类成功！但如果你回答说“这是一头小猪”。这时你的监护人就会纠正你的偏差，“乖，不对，这是一只小狗”，这样一来二去地进行训练，不断更新你大脑的认知体系，聪明如你，下次再遇到这类新的“猫、狗、猪”等，你就会天才般地给出正确的“预测”分类（示意图如图3-1所示）。

事实上，整个机器学习的过程就是在干一件事，即通过训练，学习得到某个模型，然后期望这个模型也能很好地适用于“新样本”。这种模型适用于新样本的能力，也称为“泛化能力”，它是机器学习算法非常重要的性质。

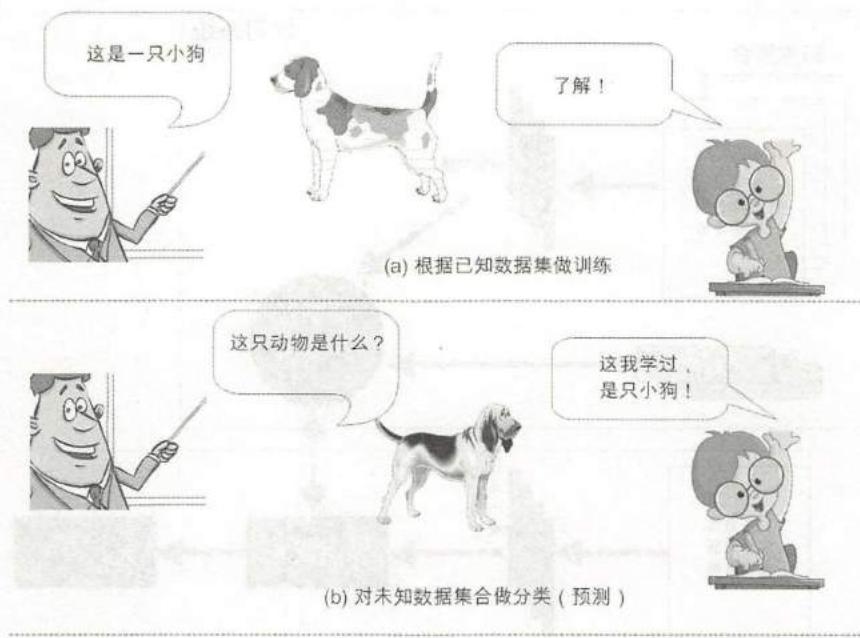


图 3-1 监督学习

### 3.1.2 监督学习的形式化描述

下面我们给出监督学习更加形式化（或者说更正式）的描述。所谓监督学习，就是先用训练数据集合学习得到一个模型，然后再使用这个模型对新样本进行预测（Prediction）。

在学习过程中，需要使用训练数据，而训练数据往往是人工给出的。在这个训练集合中，系统的预期输出（即标签信息）已经给出，如果模型的实际输出与预期不符（二者有差距），那么预期输出就有责任“监督”学习系统，重新调整模型参数，直至二者的误差在可容忍的范围之内。因此，预期输出（标签信息）也被称为“教师信号”。

监督学习的流程框架大致如图 3-2 所示。首先，准备输入数据，这些数据可以是文本、图片，也可以是音频、视频等；然后，再从数据中抽取所需的特征，形成特征向量（Feature Vector）；接下来，把这些特征向量和输入数据的标签信息送入学习模型（具体来说是某个学习算法），经过反复训练，“打磨”出一个可用的预测模型；再采用同样的特征抽取方法作用于新样本，得到新样本的特征向量；最后，把这些新样本的特征向量作为输入，使用预测模型实施预测，并给出新样本的预测标签信息（Expected Label）。

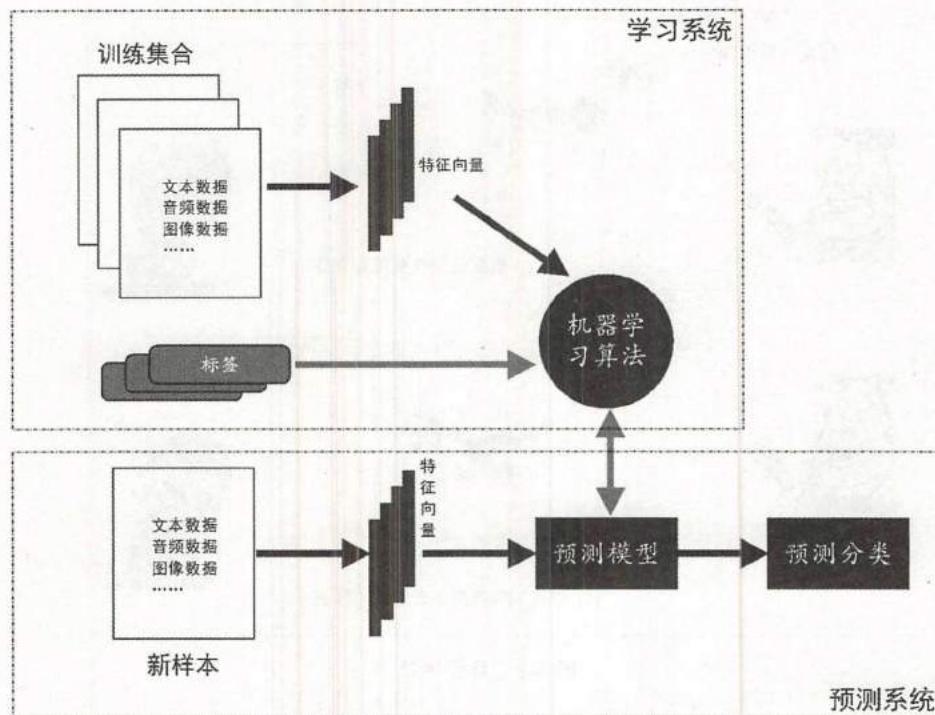


图 3-2 监督学习的基本流程

在监督学习里，根据目标预测变量的类型不同，监督学习大体可分为回归分析和分类学习。回归分析主要包括线性回归（Linear Regression）和逻辑回归（Logistic Regression）。这里回归的主要功能在于，预测输入变量  $X$ （自变量，即特征向量）和输出变量  $Y$ （因变量，即标签）之间的关系。这个关系的表现形式通常是一个函数解析式。回归问题的学习，在某种程度上，等价于函数的拟合，即选择一条函数曲线，使其能很好地拟合已知数据，并较好地预测未知数据<sup>[2]</sup>。

如前文所述，作为有监督学习的代表之一，回归问题也分为学习和预测两大部分。首先给定一个训练数据集  $T$ ：

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\} \quad (3-1)$$

在这里， $x_i \in R^n$  表示输入， $y_i \in \mathbf{R}$  对应输出， $i = 1, 2, \dots, N$ 。 $\mathbf{R}$  表示实数集， $R^n$  表示  $n$  维实数向量空间。学习系统基于训练数据构建出一个模型，即函数  $Y$ ：

$$Y \approx f(X, \beta) \quad (3-2)$$

这里， $\beta$  表示未知参数，它可以是一个标量，也可以是一个向量。这样一来，回归模型就把  $Y$  和一个  $X$  和  $\beta$  的函数关联起来了。然后，给定某个新的输入  $x_{N+1}$ ，预测系统就根据所学的模型（公式 3-2）给出相应的输出  $y_{N+1}$ ，示意图如图 3-3 所示。

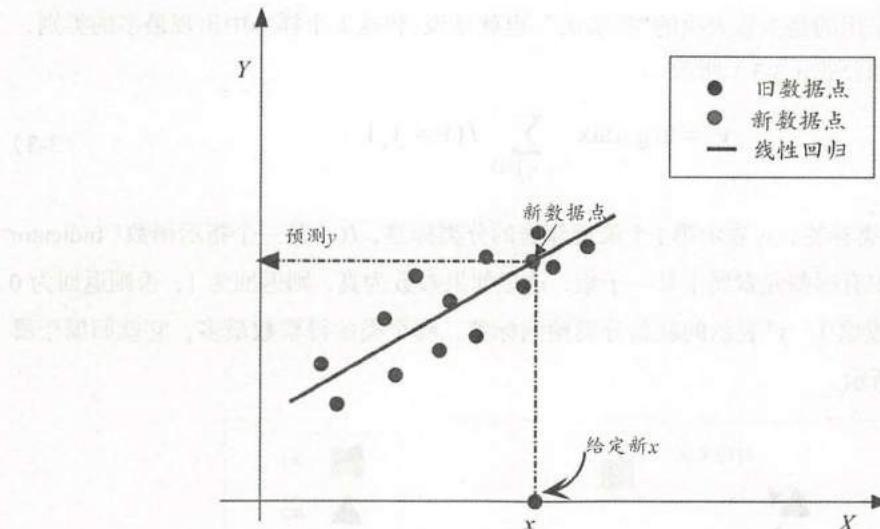


图 3-3 一元线性回归示意图

如果按照输入变量的个数来分，回归问题可分为一元回归和多元回归。如果按照输入变量和输出变量之间的关系类型来分，又可分为线性回归和非线性回归。回归学习常用的“损失函数”是平方损失函数，在这种情况下，回归问题通常用最小二乘法（Least Squares Method, LSM）来求解。

分类学习算法比较多，比较著名的有  $k$ -近邻（ $k$ -Nearest Neighbor,  $k$ NN）、支持向量机（Support Vector Machine, SVM）、朴素贝叶斯分类器（Naive Bayes）、决策树（Decision Tree）、BP 反向传播算法等。其中 BP 算法将会在后续的章节中详细进行讲解，因为它是很多神经网络算法的基础。

为了便于在后续章节开展基于 Python 的机器学习实战的讲述，这里，我们选择监督学习中最具有代表性的算法—— $k$ -近邻，进行简单介绍。

### 3.1.3 $k$ -近邻算法

$k$ -近邻算法是经典的监督学习算法，位居 10 大数据挖掘算法之列<sup>[3]</sup>。 $k$ -近邻算法的工作机

制并不复杂，简单描述如下：给定某个待分类的测试样本，基于某种距离（如欧几里得距离）度量，找到训练集合中与其最近的  $k$  个训练样本。然后基于这  $k$  个最近的“邻居”（ $k$  为正整数，通常较小），进行预测分类。

预测策略通常采用的是多数表决的“投票法”。也就是说，将这  $k$  个样本中出现最多的类别，标记为预测结果，如公式（3-3）所示。

$$y' = \arg \max_v \sum_{(x_i, y_i) \in D_z} I(v = y_i) \quad (3-3)$$

这里， $v$  表示分类标签， $y_i$  表示第  $i$  个最近邻居的分类标签， $I(\cdot)$  是一个指示函数（Indicator Function），表示其中有哪些元素属于某一子集，这里如果参数为真，则返回为 1，否则返回为 0（实际上就是分类投票）。 $y'$  表示的就是分类预测标签，哪个类的得票数最多，它就归属于哪一个类，如图 3-4 所示。

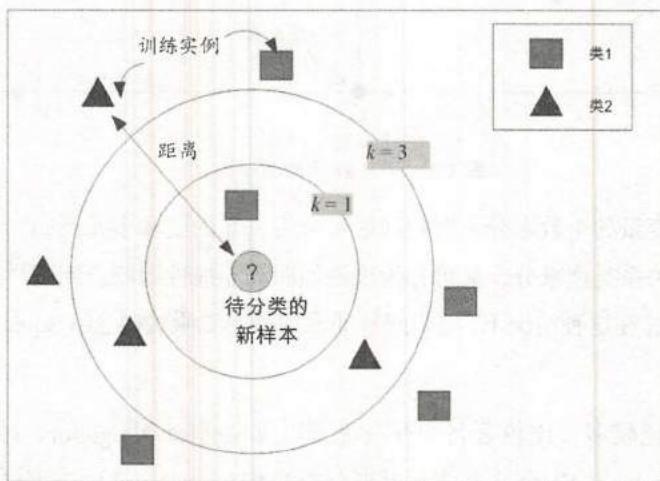


图 3-4  $k$ -近邻分类示意图

在回归任务中， $k$ -近邻算法多采用“平均值”法，即将这  $k$  个样本标记的平均值用于预测结果。若  $k=1$ ，则该对象的类别直接由最近的一个节点赋予。

$KNN$  是一种基于实例的学习，也是惰性学习的典型代表。所谓惰性学习（Lazy Learning）是指，它没有显式的训练过程。此类学习方法，在训练阶段仅仅将样本保存起来，所以训练时间开销为零。待收到测试样本时，才开始处理。与之相反的是，在训练阶段就“火急火燎”地从训练样本中建模型、调参数的学习方法，称为“急切学习（Eager Learning）”。

$k$ NN 算法虽然简单易用，但也有其不足之处。首先，“多数表决”分类会在类别分布偏斜时浮现缺陷。也就是说， $k$  的选取非常重要，因为出现频率较多的样本将会主导测试点的预测结果。从图 3-4 中可见， $k$  取值不同，分类的结果明显不同。当  $k=1$  时，待分类的样本自然属于第 1 类（即方块类）；而当  $k=3$  时，遵循“少数服从多数”原则，待分类的样本又归属为第 2 类（1 个方块，2 个三角形，1 : 2，三角形类获胜）。自然，当继续扩大  $k$  值时，待分类的样本又可能发生变化。

其次，“少数服从多数”原则也容易产生“多数人的暴政”问题。通过学习历史知识，我们知道，如果某个君王刚愎自用，听不进他人的谏言，不察民情，导致民不聊生的现象，可谓“寡人暴政”。那什么又是“多数人的暴政”呢？最早提出“多数人的暴政”概念的是法国历史学家托克维尔（Tocqueville），他将这种以多数人名义行使无限权力的情况，称为“多数人的暴政”。

“多数人的意见虽然代表了大多数人的利益，但‘多数’可能恰恰就是平庸的多数，精英永远是少数。大众民主，并不能保证人类社会向正确的方向发展”。“多数人的暴政”的历史渊源，最早可以追溯到古希腊时代的“苏格拉底之死”，如此智慧之人的死刑判决，竟然是由雅典人一人一票表决出来的。

类似的， $k$ NN 算法如果简单地实施“众（数据）点平等”的“少数服从多数”原则，那么也可能将新数据的类别归属误判。那么，怎样才能缓解这一不利的趋势呢？俗话说得好，“远亲不如近邻”。事实上，我们需要给不同的点赋以不同的权重，轻重有别，越靠近数据点的投票权重越高，这样才能在投票原则下更为准确地预测其类别。

最后，距离计算的方式不同，也会显著影响谁是它的“最近邻”，从而也会显著影响分类结果。常用的距离计算方式有欧氏距离（Euclidean Distance）、马氏距离（Mahalanobis Distance）及海明距离（Hamming Distance）等。

## 3.2 非监督学习

### 3.2.1 感性认识非监督学习

与监督学习相反的是，非监督学习（Unsupervised Learning）所处的学习环境，都是非标签的数据。韩家炜教授接着说<sup>[1]</sup>，“非监督学习，本质上就是‘聚类（Cluster）’的近义词。”

话说聚类的思想起源非常早，在中国，可追溯到《周易·系辞上》中的“方以类聚，物以群分，吉凶生矣”。但真正意义上的聚类算法，却是 20 世纪 50 年代前后才被提出的。为何会如

此滞后呢？原因在于，聚类算法的成功与否，高度依赖于数据。数据量小了，聚类意义不大。数据量大了，人脑就不灵光了，只能交由计算机解决，而计算机 1946 年才开始出现。

如果说分类是指，根据数据的特征或属性，划分到已有的类别当中。那么，聚类一开始并不知道数据会分为几类，而是通过聚类分析将数据聚成几个群。

简单来说，给定数据，聚类从数据中学习，能学到什么，就看数据本身具备什么特性了(given data, learn about that data)。对此，北京航空航天大学的于剑教授，对聚类有 12 字的精彩总结<sup>[4]</sup>，“归哪类，像哪类。像哪类，归哪类。”展开来说，给定  $N$  个对象，将其分成  $K$  个子集，使得每个子集内的对象相似，不同子集之间的对象不相似。

但这里的“类”也好，“群”也罢，事先我们是并不知情的。一旦归纳出一系列“类”或“群”的特征，如果再来一个新数据，我们就根据它距离哪个“类”或“群”较近，就预测它属于哪个“类”或“群”，从而完成新数据的“分类”或“分群”功能(参见图 3-5)。



图 3-5 非监督学习

比较有名的非监督学习算法有  $K$  均值聚类(K-Means Clustering)、关联规则分析(Association Rule, 如 Apriori 算法等)、主成分分析(Principal Components Analysis, PCA)、随机森林(Random

Forests)、受限玻尔兹曼机 (Restricted Boltzmann Machine, RBM) 等。

目前用在深度学习里，最有前景的无监督学习算法是 Ian Goodfellow 提出来的“生成对抗网络 (Generative Adversarial Networks)”。下面我们简单介绍一下非监督学习中的代表—— $K$  均值聚类，目的是为后续章节中讲解基于 Python 的机器学习实战做铺垫。

### 3.2.2 非监督学习的代表—— $K$ 均值聚类

聚类分析在模式识别、机器学习以及图像分割领域有着重要作用。 $K$  均值聚类是一种重要的聚类算法。由于该算法时间复杂度较低， $K$  均值聚类被广泛应用在各类数据信息挖掘业务中。 $K$  均值聚类是 James MacQueen 于 1967 年提出来的，时至今日，它仍然是很多改进版聚类模型的基础。聚类算法的最终目的之一，是将集合划分为若干个簇。那么，到底什么是聚类呢？

#### 3.2.2.1 聚类的基本概念

至今，聚类并没有一个公认的严格定义。宽泛来说，聚类指的是，将物理或抽象对象的集合分成由类似的对象组成的多个类的过程。从这个简单的描述中可以看出，聚类的关键是如何度量对象间的相似性。

在实际操作中，对于“相似”的定义，会引出诸多问题。比如，给定三个三角形，红、绿、蓝各一个，再给定三个方形，红、绿、蓝各一。对于这六个对象，分别按照形状相似和颜色相似，可以得到两种划分方法。如果按形状作为“相似”的度量，那么可以得到两个聚类：三角形类和方形类（这些类名都是“聚”之后取的，下同）。如果按照颜色来度量，可以得到三个聚类（即红、绿、蓝三类）。两种方法都对，不同的是对“相似”的定义。所以，“主观”是相似性最大的问题之一<sup>[4]</sup>。

较为常见的用于度量对象的相似度的方法有距离、密度等。由聚类所生成的簇 (Cluster) 是一组数据对象的集合，这些对象的特性是，同一个簇中的对象彼此相似，而与其他簇中的对象相异，且没有预先定义的类（即属于非监督学习的范畴），如图 3-6 所示。

对于聚类分析而言，它通常要分四步走，即数据表示、聚类判据、聚类算法和聚类评估。数据表示是设计聚类算法的第一步。同一个聚类算法只能用一种数据表示，否则相似性无法度量。数据表示可分为外显和内在两部分。图像、语音、文本等都是数据外显表示的常见形式，而内在表示则显得有点玄妙。此处于剑教授给出了一个非常经典的例子：高山流水（最早见于《列子·汤问》）。在这个典故里，琴师伯牙与樵夫钟子期互为知音，子期接受到外在的琴声（外在数据表示），而琴声的内在表示却是“高山和流水”。通常，机器善于感知数据的外显表示，

而不长于其内在的部分。数据内在的部分，还需要算法学习和抽象。

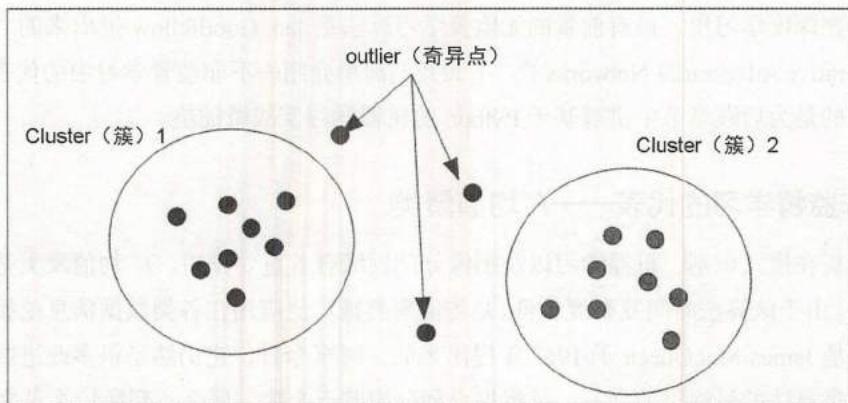


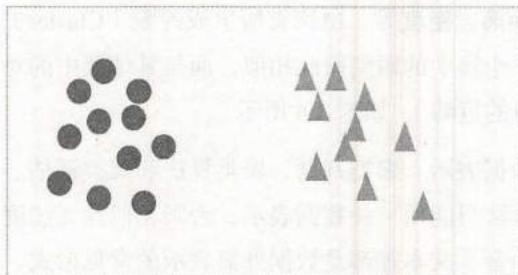
图 3-6 聚类示意图

聚类的第二步是判据，算法通过判据来确定聚类搜索的方向。第三步才是聚类算法设计。有了数据表示和聚类方向之后，我们就可以在战术上施展拳脚，在聚类算法的速度和准确度上，一比高低。

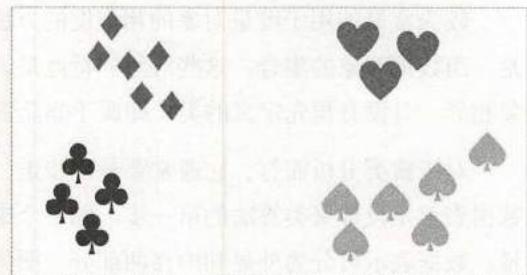
最后一步是聚类的评估。评估环节是聚类和分类最大的差异之处，分类有明确的外界标准，是猫是狗，一目了然，而聚类的评估，则显得相对主观。

### 3.2.2.2 簇的划分

在聚类过程中，我们规定同一簇特征相似，有别于其他簇。那么，簇的划分一定是直观可见的吗？不见得。在图 3-7 的左图中，我们可以很直观地看出有 2 个簇，在图 3-7 的右图中，可以明显看出有 4 个簇。



2个Cluster (簇)



4个Cluster (簇)

图 3-7 分类示意图

但是，当簇的特征不是那么明显时，就无法显而易见地看出来了，如图 3-8 所示。这时，就需要一个算法来帮助我们划分簇， $K$  均值聚类就是其中最常用的一种算法。



图 3-8 簇的个数未知

### 3.2.2.3 $K$ 均值聚类算法的核心

$K$  均值聚类的目的在于，给定一个期望的聚类个数  $K$ ，和包括  $N$  个数据对象的数据集合，将其划分为满足距离方差最小的  $K$  个类。

该算法的基本流程为：首先选取  $K$  个点作为初始  $K$  个簇的中心，然后将其余的数据对象按照距离簇中心最近的原则，分配到不同的簇。当所有的点均被划分到一个簇后，再对簇中心进行更新。更新的依据是根据每个聚类对象的均值，计算每个对象到簇中心对象距离的最小值，直到满足一定的条件才停止计算。满足的条件一般为函数收敛（比如前后两次迭代的簇中心足够接近）或计算达到一定的迭代次数自动停止。 $K$  均值聚类算法的核心如图 3-9 所示。

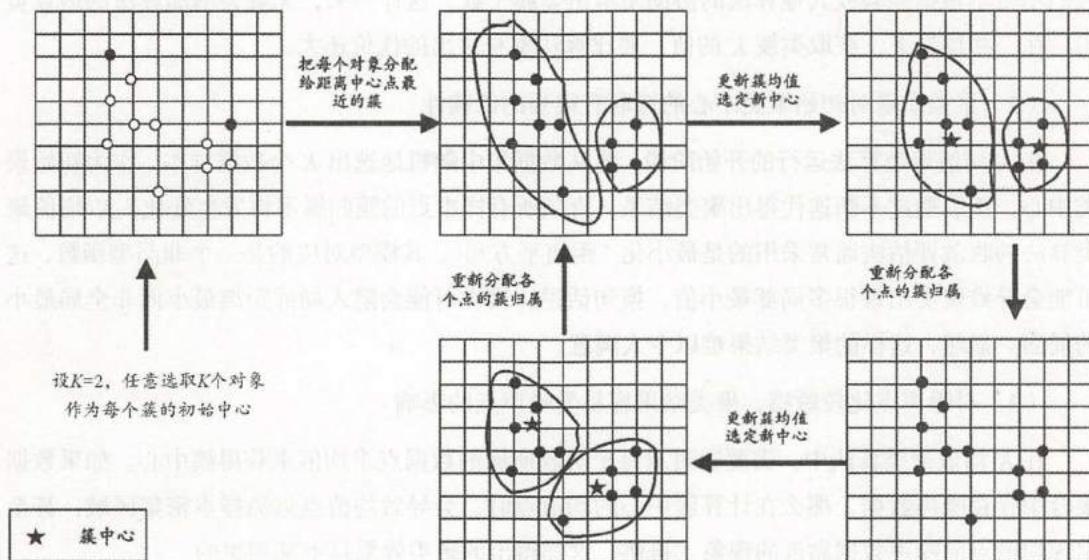


图 3-9  $K$  均值聚类示意图

在  $K$  均值聚类算法中，我们主要需要考虑两个方面的因素：初始簇中心（也称质心）的选取以及距离的度量。常见的选取初始质心的方法是随机挑选  $K$  个点，但这样的簇质量往往很差。因此，挑选质心的常用方法有：

(1) 多次运行调优。即每次使用一组不同的随机的初始质心，最后从中选取具有最小平方误差的簇集。这种策略简单，但效果难料，主要取决于数据集的大小和簇个数  $K$ 。

(2) 根据先验知识（也就是历史经验）来决定  $K$  值大小。

对于另一个因素——如何确定对象之间的距离。根据问题场景不同，其度量方式也是不同的。在欧氏空间中，我们可以通过欧几里得距离来度量两个数据之间的距离，而对于非欧氏空间，可以有 Jaccard 距离、Cosine 距离或 Edit 距离等距离度量方式。

### 3.2.2.4 $K$ 均值聚类算法的优缺点

由于  $K$  均值聚类算法简单易于操作，并且效率非常高，因此该算法得到了广泛的应用。但它也有其自身的不足，大体上有以下 4 点。

(1)  $K$  值需要用户事先给出

通过对  $K$  均值聚类算法的流程分析不难看出，在执行该算法之前，需要给出聚类个数。然而，在实际工作场景中，对给定的数据集要分多少个类，用户往往很难给出合适的答案。此时，人们不得不根据经验或其他算法的协助来给出类簇个数。这样一来，无疑会增加算法的运算负担。在一些场景下，获取类簇  $K$  的值，要比算法本身付出的代价还大。

(2) 聚类质量对初始聚类中心的选取有很强的依赖性

在  $K$  均值聚类算法运行的开始阶段，要从数据集中随机地选出  $K$  个数据样本，作为初始聚类中心，然后通过不断迭代得出聚类结果，直到所有样本点的簇归属不再发生变化。 $K$  均值聚类算法的收敛评估法通常采用的是最小化“距离平方和”，其模型对应的是一个非凸型函数，这可能会导致聚类出现很多局部最小值，换句话说，聚类可能会陷入局部距离最小而非全局最小的局面。显然，这样的聚类结果难以令人满意。

(3) 对噪声点比较敏感，聚类结果容易受噪声点的影响

在  $K$  均值聚类算法中，需要通过对每个类簇所属的数据点求均值来获得簇中心。如果数据集合中存在噪声数据，那么在计算簇中心的均值点时，会导致均值点远离样本密集区域，甚至出现均值点向噪声数据靠近的现象。自然，这样得出的聚类效果是不甚理想的。

(4) 只能发现球形簇，对于其他任意形状的簇，顿感无力

$K$  均值聚类算法常采用欧氏距离来度量不同点之间的距离，这样只能发现数据点分布较均匀的球形簇。采用距离平方和的评估函数，会导致在聚类过程中，为了能让目标函数取到极小值，通常也可能把数据集合较大的类分成一些较小的类，这种趋势也会导致聚类效果不甚理想。

### 3.3 半监督学习

半监督学习(Semi-supervised Learning)的方式，既用到了标签数据，又用到了非标签数据。有一句骂人的话，说某个人“有妈生，没妈教”，抛开这句话骂人的含义，其实它说的是“无监督学习”。但我们绝大多数人，不仅“有妈生，有妈教”，还有小学教、有中学教、有大学教，“有人教”的意思是，有人告诉我们事物的对与错(即对事物打了标签)，然后我们可据此改善自己的性情，慢慢把自己调教得更有教养，这自然就属于“监督学习”。

但总有那么一天，我们会长大。而长大的标志之一，就是自立。何谓自立呢？就是远离父母、走出校园后，没有人告诉你对与错，一切都要基于自己早期已获取的知识，从社会中学习，扩大并更新自己的认知体系，然后当遇到新事物时，我们能泰然自若地处理，而非六神无主。

从这个角度来看，现代人类成长学习的最佳方式当属“半监督学习”！它既不是纯粹的“监督学习”(因为如果完全是这样，就会扼杀我们的创造力和认知体系，也就永远不可能超越我们的父辈和师辈)，也不属于完全的“非监督学习”，因为如果完全这样，我们会如无根之浮萍，会花很多时间重造轮子。前人的思考，我们的阶梯。

那么到底什么是“半监督学习”呢？下面我们给出它的形式化定义。

给定一个来自某个未知分布的有标记示例集 $\{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$ ，其中 $x_i$ 是输入数据， $y_i$ 是标签。对于一个未标记示例集 $U = \{x_{k+1}, x_{k+2}, \dots, x_{k+u}\}$ ， $k \ll u$ ，于是，我们期望学习得到某个函数 $f: X \rightarrow Y$ 可以准确地对未标识的数据 $x_{i+1}$ 预测其标记 $y_i$ 。这里 $x_i \in X$ ，均为 $d$ 维向量， $y_i \in Y$ 为示例 $x_i$ 的标记，示意图如图 3-10 所示。

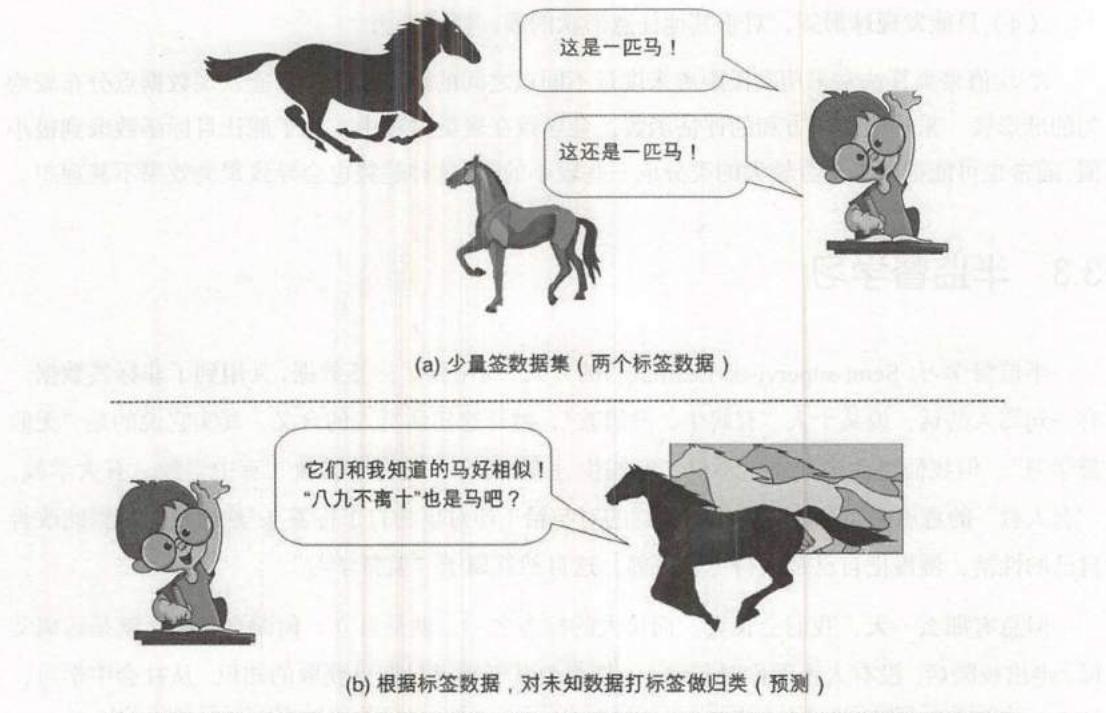


图 3-10 半监督学习

形式化的定义比较抽象，下面我们列举一个现实生活中的例子来辅助说明这个概念。假设我们已经学习到：

(a) 马晓云同学（数据 1）是一个牛人（标签：牛人）。

(b) 马晓腾同学（数据 2）是一个牛人（标签：牛人）。

(c) 假设我们并不知道李晓宏同学（数据 3）是谁，也不知道他牛不牛，但考虑他经常和二马同学共同出入高规格大会，都经常会被达官贵人接见（也就是说他们虽独立，但同分布），我们很容易根据“物以类聚，人以群分”的思想，把李晓宏同学打上标签：他也是一个很牛的人！

这样一来，我们的已知领域（标签数据）就扩大了（由两个扩大到三个），这也就完成了半监督学习。事实上，半监督学习就是以“已知之认知（标签化的分类信息）”，扩大“未知之领域（通过聚类思想将未知事物归类为已知事物）”。但这里隐含了一个基本假设——聚类假设（Cluster Assumption），其核心要义就是：相似的样本，拥有相似的输出。

常见的半监督学习算法有生成式方法、半监督支持向量机（Semi-supervised Support Vector

Machine, 简称 S<sup>3</sup>V<sub>M</sub>, 是 SVM 在半监督学习上的推广应用)、图半监督学习、半监督聚类等。

事实上, 我们对半监督学习的现实需求是非常强烈的。其原因很简单, 就是因为人们能收集到的标签数据非常有限, 而手工标记数据需要耗费大量的人力物力, 但非标签数据却大量存在且触手可及, 这个现象在互联网数据中更为凸出, 因此, 半监督学习就显得尤为重要<sup>[5]</sup>。

人类的知识其实都是这样, 以“半监督”的滚雪球的模式, 越“滚”越大。半监督学习既用到了监督学习的先验知识, 也吸纳了非监督学习的聚类思想, 二者兼顾。

如此一来, 半监督学习就有点类似于中华文化中的“中庸之道”了。下面我们就聊聊机器学习的“中庸之道”。

### 3.4 从“中庸之道”看机器学习

说到“中庸之道”, 很多人立刻想到的就是“平庸之道”, 把它的含义理解为“不偏不倚、不上不下、不左不右、不前不后”。其实, 这是一个很大的误解!

据吴伯凡<sup>[6]</sup>先生介绍, “中”最早其实是一个器具, 它看上去像一个椎子, 为了拿起方便, 就用手柄穿越其中, 即为“中”。

这个“中”可不得了, 它非常重要, 以至于只有少数人才能使用。那都是谁来用呢? 答案就是古代的军事指挥官。在“铁马金戈风沙腾”的战场上, 军旗飘飘, 唯有一人高高站在战车上, 手握其“中”, 其他将士都视其“中”而进退有方(见图 3-11 第二行第一字), 而手握其“中”的人, 称之为“史”(见图 3-11 第一行第一字)。所以现在你知道了吧, 其实“史”最早的本意, 就是手握指挥大权的大官。

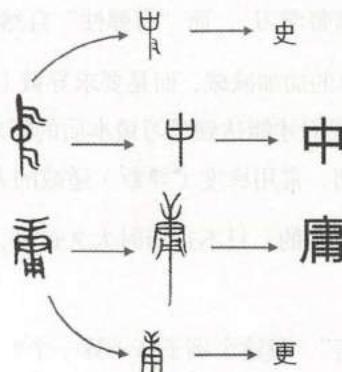


图 3-11 中庸之道, 寓意为何

再后来，“中”有了各种各样的引申含义。比如，“中”还有中心、核心的含义，我们说的“中国”“中华”都蕴含这种含义。此外，“中”的外形也很有意思，它有点像“0和1”的串联组合。

在中原地带的人，在他们的语言里到现在还保留一些古代遗风，比如，河南人在表达“对”或者“是”的时候，他说的是“中（zhóng）”。

其实，“中”还有一个读音叫“中（zhòng）”，比如成语里就有“正中下怀”“百发百中”等。这时，“中（zhòng）”的含义就是恰到好处，不偏离原则，坚守关键点。

下面再来说说“庸”。“庸”的上半部是“庚”，“庚”同音于“更”，即“变化”之意（见图3-11第四行第一字）。而“庸”的下半部是“用”（见图3-11第三行第一个字），“用”的本意为“变化中的不变”，即为“常”。在编程语言中，“常量”即指不变的量。所以，“庸”的最佳解释应该是“富有弹性的坚定”。

那么“中庸”放在一起是什么意思呢？它告诉我们“在变化中保持不变”。其中，所谓“变化”，就是我们所处的环境变化多端，所以也需要我们“随机应变，伺机而动”。而所谓“不变”就是我们要“守住底线，中心原则不变”。二者合在一起，“中庸之道”就是告诉我们要在灵活性（变）和原则性（不变）之间，保持一个最佳的平衡。

那“中庸之道”和机器学习有什么关系呢？其实这就是一个方法论问题。“监督学习”，就是告诉你“正误之道”，即有“不变”之原则。而“非监督学习”，保持开放性，但就有点“随心所欲，变化多端”，不易收敛，很易“无根”。

那“中庸之道”的机器学习应该是怎样的呢？自然就是“半监督学习”，做有弹性的坚定学习。这里的“坚定”自然就是“监督学习”，而“有弹性”自然就是“非监督学习”。

“有弹性”的变化，不是简单的加加减减，而是要求导数（变化），而且还可能是导数的导数（变化中的变化）。只有这样，我们才能达到学习最本质的需求——性能的提升。在机器学习中，我们不正是以提高性能为原则，常用梯度（导数）递减的方式来完成的吗？

所以，祖先的方法论其实是很牛的。只不过历时太久远了，其宝贵的内涵被时间的尘埃蒙蔽了而已。

现在，我们经常提“文化自信”，我这个例子算不算一个？

## 3.5 强化学习

前面我们讨论了机器学习的三大门派。在传统的机器学习分类中，并没有包含强化学习。但实际上，在连接主义学习中，还有一类人类学习常用、机器学习也常用的算法——强化学习（Reinforcement Learning，简称 RL）。

“强化学习”亦称“增强学习”，但它与监督学习和非监督学习都有所不同。强化学习强调的是，在一系列的情景之下，选择最佳决策，它讲究通过多步恰当的决策，来逼近一个最优的目标，因此，它是一种序列多步决策的问题。

强化学习的设计灵感，源于心理学中的行为主义理论，即有机体如何在环境给予的奖励或惩罚刺激下，逐步形成对刺激的预期，从而产生能获得最大利益的习惯性行为。

上面的论述看起来比较抽象，下面我们举一个生活中的例子来说明这个概念。对于儿童教育，有句话非常流行，“好孩子是表扬出来的”。这句话是有道理的，它反映了生物体以奖励为动机的行为。比如，我们知道，想让一个小孩子静下来学习，这是十分困难的。但如果父母在他（她）每复习完一篇课文，就说句“你真棒”并奖励一块巧克力，那么孩子就会明白，只有不断学习，才能获得奖励，从而也就更有劲头复习。

“表扬”本身并不等同于监督学习的“教师信号”（即告诉你行为的正误），却也能逐步引导任务向最优解决方案进发。因此，强化学习也被认为是人类学习的主要模式之一。监督学习、强化学习与非监督学习的区别，如图 3-12 所示。

在外号雅称为“西瓜书”的《机器学习》一书中<sup>[5]</sup>，南京大学的周志华教授就用种西瓜的例子来说明“强化学习”的含义，也别有意义。

考虑一下种西瓜的场景。西瓜从播种到瓜熟蒂落，中间要经过很多步骤。首先得选种，然后播种、定期浇水、施肥、除草、杀虫等，最后收获西瓜。这个过程要经过好几个月。如果把收获高品质的西瓜作为辛勤劳作奖赏的话，那么在种瓜过程中实施某个操作（如浇水、施肥等）时，我们并不能立即得到相应的回报，甚至也难以判断当前操作对最终回报（收获西瓜）有什么影响，因为浇水或施肥并不是越多越好。

然而，即使我们一下子还不能看到辛勤劳作的最终成果，但还是能得到某些操作的部分反馈。例如，瓜秧是否更加茁壮了？通过多次的种瓜经历，我们终于掌握了播种、浇水、施肥等一系列工序的技巧（相当于参数训练），并最终能够收获高品质的西瓜。如果把这个种瓜的过程抽象出来，它就是我们说到的强化学习，示意图如图 3-13 所示。

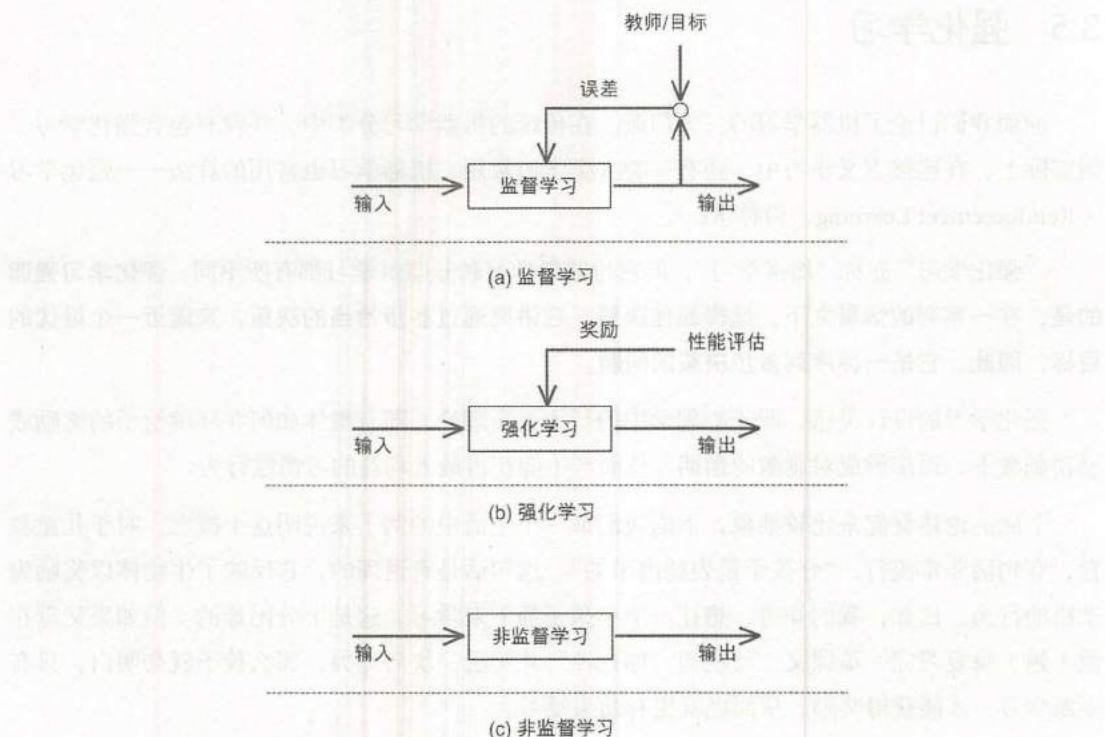


图 3-12 监督学习、强化学习与非监督学习的区别

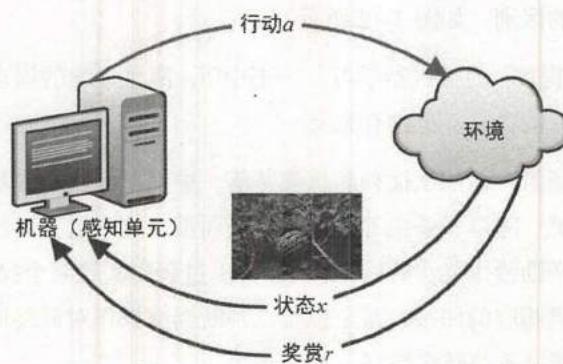


图 3-13 强化学习示意图

在机器学习问题中，环境通常被规范为一个马可夫决策过程（Markov Decision Processes, MDP），许多强化学习算法就是在这种情况下使用动态规划技巧。

马可夫决策过程提供了一个数学架构模型，用于部分随机、部分可由决策者控制的状态下。

可如何进行最佳决策呢？在强化学习场景中，假设机器处于环境  $E$  中，状态空间为  $X$ ，其中每个状态  $x \in X$  是机器所能感知的环境描述。针对种西瓜的例子，状态就是瓜秧的长势。机器所能采取的行动就构成了动作空间  $A$ ，比如种瓜过程中浇多少水、施多少肥、使用何种除草剂等多种可供选择的动作。

若某个动作  $a \in A$  作用于当前状态  $x$  上，那么潜在的转移函数将驱使环境从当前状态按照某种概率  $P$ ，转移到另一个状态。比如，瓜苗状态为缺水，则选择浇水，然后瓜秧的长势有一定的概率长得更加茁壮，也有一定的概率无法恢复。

当从一种状态转移到另外一种状态时，环境会根据潜在的“回报（reward）”函数  $R$ ，反馈给机器一个回报  $r$ 。例如，如果瓜秧健康，则回报为“+5”分（即奖赏）；如果瓜秧凋零，则回报为“-10”分（即惩罚），最终收获了高品质的好西瓜，就重赏“100”分。

综合起来，强化学习的任务可形式化描述为：给定一个四元组  $E = \{X, A, P, R\}$ ，其中转移函数  $P$  指定了状态转移概率，可定义为： $X \times A \times X \mapsto \mathbf{R}$ ，这里的“ $\times$ ”表示取该集合中的一个元素， $\mapsto$  表示某种映射关系， $\mathbf{R}$  表示某实数集合中的一个元素。类似的， $R$  指定了回报，也可以定义为  $X \times A \times X \mapsto \mathbf{R}$ 。

强化学习和监督学习的区别在于，强化学习并不需要出现正确的“输入/输出对”，也不需要精确校正次优化的行为。强化学习更加专注于在线规划，需要在“探索”（在未知的领域）和“利用”（现有知识）之间找到平衡（Tradeoff）。强化学习中的“探索-利用”的交换，这在多臂老虎机问题和有限 MDP 中研究得较多。

与强化学习相关的一则报道是，2017 年 10 月，Google 深度思维团队在著名学术期刊 *Nature*（自然）上发表了一篇论文“Mastering the game of Go without human knowledge”（无须人类知识，精通围棋博弈）<sup>①</sup>，他们设计了 AlphaGo（阿法狗）的升级版 AlphaGo Zero（阿法元），阿法元从零（*Tabula rasa*<sup>②</sup>）开始，不需要人类任何历史围棋棋谱做指导，完全靠强化学习来参悟，自学成才，并以 100 : 0 击败了阿法狗。论文的第一作者、AlphaGo 创始人之一大卫·席尔瓦（David Silver）指出，阿法元远比阿法狗强大，因为它不再被人类的知识所局限，而是能够发现新知识，发现新策略。这确实是机器学习进步的一个重要标志。

更多有关强化学习的资料，请读者参阅参考资料[5]。

<sup>①</sup> 这是一个从哲学中借用的术语，意为“白板”。

## 3.6 本章小结

在本章中，我们主要讲解了机器学习的主要形式，从有无“教师信号”和是否使用标签数据，可分为监督学习、非监督学习、半监督学习及强化学习。简单来说，监督学习是一种利用标签数据的分类技术，它通常使用这些正确的且已标记过的数据来训练神经网络。

非监督学习是一种利用距离的“亲疏远近”，来衡量不同类的聚类技术。非监督学习使用未标记过的数据，即不知道输入数据对应的输出结果是什么，让学习算法自身发现数据的模型和规律，比如聚类和异常检测。非监督学习之所以能进行“异常检测”，就是判断某些点“不合群”，它是聚类的反向应用。

半监督技术则采用“中庸之道”，利用聚类技术扩大已知标签范围，也就是说，训练中使用的数据只有一小部分是标记过的，而大部分是没有标记的，然后逐渐扩大标记数据的范围。

强化学习也使用未标记的数据，它可以通过某种方法（奖惩函数）知道你是离正确答案越来越近，还是越来越远。

如果换一个角度来看，机器学习的研究还分为三大流派：连接主义、行为主义和符号主义。连接主义的主要代表形式是人工神经网络，它的处理对象是原始的数据，这部分研究包括深度学习的最新进展。这将是我们后续章节讨论的重点。

行为主义的代表形式就是强化学习方法（3.5 节做了简要介绍），它的主要处理对象是信息（即奖惩分明的有价值数据，所谓的信息就是有逻辑关联的数据）。最近阿法元的巨大成功，可视为这个流派的最佳代言人。但需要注意（也无须恐慌）的是，AlphaGo Zero 的成功规则，并不能广泛适用于其他人工智能领域。之所以这么说，原因主要有以下两个：

(1) AlphaGo Zero 看似无须数据（不需要人类的棋谱），但恰恰相反，AlphaGo Zero 反而证明了数据在人工智能中的高度依赖性。与普通人工智能项目不同的是，AlphaGo Zero 依靠规则明确的围棋，自动生成了大量的、训练自己的数据。而人类的智能表现，大多是在非明确规则下完成的，也无须大数据来训练自己的智能。比如，回到第 1 章“谈恋爱”的例子，问一下自己，你一辈子能谈几次恋爱（屈指可数吧）。家不是讲理（规则）的地方，谈恋爱也不是吧？

(2) 围棋属于一种彼此信息透明、方案可穷举的全信息博弈游戏。然而，人类的决策，大多是在信息残缺处境下做出的。所以，阿法元的成功（智能表现）虽然惊艳，但想把它的强化学习经验迁移到信息非完备的场景，还有很长的路要走。

下面我们再说说符号主义。符号主义的代表形式是专家系统和知识图谱(Knowledge Graph)，主要用作处理知识和推理(所谓的知识，就是从信息中提炼出规律，并以规律指导我们的行动)。这个流派的历史渊源颇深，在沉寂了一段时间之后，近来又有“老树发新芽”之势，特别是知识图谱，在智能搜索领域有着广泛的应用，但它不是本书讨论的重点，感兴趣的读者可参阅相关资料。

从三大流派处理的对象来看，好像有某种递进的关系：(数据→信息→知识)，但当前人工智能距离人类所擅长的概念产生和理论建立，还相距甚远，尤其是在非公理性逻辑和情感化表征等方面，当下的智能机器更是望“人”兴叹！

“纸上得来终觉浅，绝知此事要躬行。”在前面的章节中，我们介绍了不少与深度学习相关的话题。但如果想让相关话题更接地气的话，需要步入实战环节——编程实现一些常见的算法。从第4章开始，我们就务实地聊聊Python的用法，并逐步介绍机器学习的实战操作，以增强读者的感性认识。

## 3.7 请你思考

通过前面的学习，请你思考如下问题：

- (1) 深度学习算法既有监督学习模式的，也有非监督学习模式的？它有没有半监督学习模式的？如果有，请你分别列举一二。
- (2) 你知道AlphaGo 和其升级版本AlphaGo Zero（阿法元）在技术上有什么不同吗？
- (3) 中国古代的铜钱，也体现有“中庸之道”，你知道它是什么吗？

## 参考资料

- [1] Han J. Data Mining: Concepts and Techniques[M]. Morgan Kaufmann Publishers Inc. 2005.
- [2] 李航. 统计学习方法[M]. 北京: 清华大学出版社, 2012.
- [3] Wu X, Kumar V, Quinlan J R, et al. Top 10 algorithms in data mining[J]. Knowledge and information systems, 2008, 14(1): 1-37.
- [4] 于剑. 机器学习——从公理到算法[M]. 北京: 清华大学出版社, 2017.

- [5] 周志华. 机器学习[M]. 北京: 清华大学出版社, 2016.
- [6] 吴伯凡. 中庸之美. 得到 App. 2017.
- [7] David Silver, Julian Schrittwieser, Karen Simonyan, et al. Mastering the game of Go without human knowledge. 2017.

## 第4章 人生苦短对酒歌， 我用 Python 乐趣多

古人云：“百尺高台起于垒土，千里之行始于足下”。是的，我们的目标高远，但是要想掌握并精通深度学习，路还是要一步一步走。Python 是机器学习领域应用最为广泛的编程语言之一。在本章中，我们将从基础出发，来夯实这个前行的基础。

## 4.1 Python 概要

### 4.1.1 为什么要用 Python

为什么 Python 能在机器学习领域大放异彩呢？说来原因很简单，Python 不仅功能强大，而且易学易用，实为“出工干活居家编程”之语言，而且还拥有高效的高级数据结构，且能有效实现面向对象编程。此外，它还支持动态输入，再加上其解释性语言（Interpreted Language）的本质，便于调试，特别适用于应用程序的快速开发。

简单来说，Python 的设计哲学就是：简单、明确、优雅。布鲁斯·埃克尔（Bruce Eckel）是经典畅销书《Java 编程思想》（Thinking in Java）和《C++ 编程思想》（Thinking in C++）的作者，也是 C++ 标准委员会成员，他曾说，没有一种语言比得上 Python，能使他的工作效率如此之高。为此，他还为 Python 创造了一个经典广告语：“Life is short, you need Python”。国内有人将其翻译为“人生苦短，我用 Python”，我倒是觉得十分贴切。

具体来说，Python 语言具备如下 4 大优点。

（1）**代码编写质量高：**Python 采用强制缩进的方法，代码的可读性好。

（2）**开发效率高，维护成本低：**Python 语法简单，支持动态的类型，可让复杂编程任务变得更为高效。在很多应用场景下，同样的任务实现，Python 的代码量仅为 Java 的 1/5。编程工作量少，才能有时间享受生活。故此才有前面的调侃：人生苦短，我用 Python。由于 Python 代码编写量少，维护成本自然也比较低。

（3）**丰富而完善的开源工具包：**目前 Python 拥有大量标准库和第三方库作为应用开发的支撑，从字符处理、网络编程、信号处理（如有面向复杂科学计算的 SciPy 库），到大数据分析（如有实时内存处理的 Spark、数据清洗用的 Pandas 等）、机器学习（如有包含大量经典机器学习算法的 scikit-learn 库，面向深度学习的 TensorFlow 库等），几乎无所不包。

（4）**广泛的应用程序接口：**除了被广泛使用的第三方程序库之外，在业界还有很多著名公司（如亚马逊、谷歌等）均面向互联网用户，提供了基于 Python 的机器学习应用编程接口（Application Programming Interface，API）。这些公司提供了很多机器学习模块，无须用户自己来编写，极大提高了用户的生产效率。用户只需按照规定的 API 协议与规则会使用即可，其过程就像搭积木一样。如此一来，大大提高了机器学习应用的开发效率。

正是因为有上述诸多优点，Python 逐渐成为最受程序员喜爱的编程语言之一。

正所谓“尺有所短，寸有所长”。虽然前面我们大力介绍了 Python 的优点，但并不是批判

其他语言不行。事实上，每种语言都有其擅长的领域。这好比，韩信做不了萧何那样的谋士，萧何也带不了韩信的兵。图 4-1 所示的是 2018 年 5 月的 TIOBE 编程语言排名，从图中可看出，Python 名列第四。

Mar 2018	Mar 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.941%	-1.44%
2	2		C	12.760%	+5.02%
3	3		C++	6.452%	+1.27%
4	5	^	Python	5.889%	+1.96%
5	4	▼	C#	5.067%	+0.66%
6	6		Visual Basic .NET	4.065%	+0.91%
7	7		PHP	4.010%	+1.00%
8	8		JavaScript	3.916%	+1.25%
9	12	^	Ruby	2.744%	+0.49%
10	-	△	SQL	2.686%	+2.60%

图 4-1 TIOBE 于 2018 年 5 月的编程语言排名

TIOBE 的排名基于所有领域的编程语言。单纯从图 4-1 的排名来看，Python 似乎并没有 Java、C、C++ 应用广泛，但如果垂直细分到机器学习领域，便是另一番景象——Python 这边风景独好！

这么说是有依据的，图 4-2 更能说明问题。依据 Facebook、Twitter、LinkedIn 等网站的汇总信息，图 4-2 显示了近年来机器学习与数据科学领域的职位需求排名。可明显看出，Python 的职位需求高于 R 和 Java，更是领先于 C 和 C++。

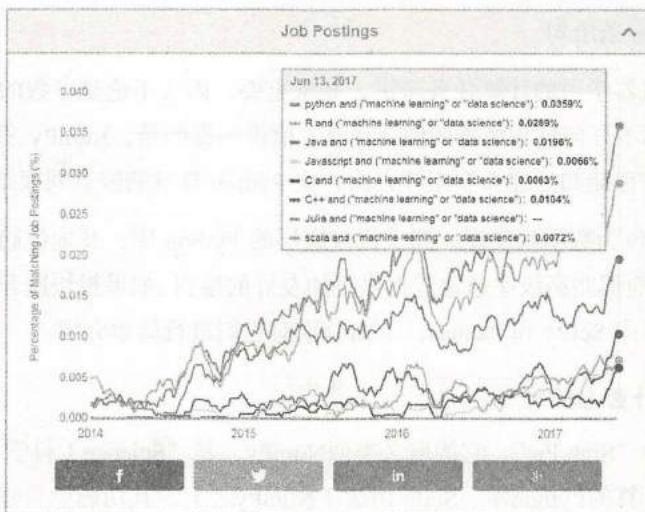


图 4-2 机器学习与数据科学领域的职位需求排名<sup>①</sup>

<sup>①</sup> 图片来源于 <https://www.indeed.com/jobtrends/>。

因此，顺势而为，我们也选择 Python 作为机器学习算法的实现语言。但需要说明的是，简短一章，肯定难以系统介绍 Python。在本章余下的内容中，我们仅讲解与机器学习密切相关的 Python 基础知识。这里，我们推荐的 Python 精进姿势是，快速入门，做中学（learning by doing），指哪打哪，缺啥补啥。这样有任务驱动的 Python 学习方法，效率更高。

## 4.1.2 Python 中常用的库

“它山之石，可以攻玉。”很多 Python 高手已帮我们编写好了许多高质量的类库。很多时候，我们没有必要重造轮子。对于一些优秀的类库，采用“拿来主义”会用就好。下面就介绍一下与机器学习相关的 Python 常用类库。

### 4.1.2.1 数值计算 NumPy

NumPy<sup>①</sup>的发音是['nʌmpɪ], 取自“Numeric（数值）”和“Python”的简写。顾名思义，它是处理数值计算最为基础的类库。NumPy参考了CPython（用C语言实现的Python及其解释器）的设计，其本身也是由C语言开发而成的。

NumPy 除了提供一些数学运算函数之外，还提供了与 MATLAB（由美国 MathWorks 公司出品的著名商业数学软件）相似的功能与操作方式，可让用户高效地直接操作向量或矩阵。比如两个矩阵的加法，在诸如 C/C++ 或 Java 等语言里，我们可能不得不使用多层 for 循环来实现，而在 NumPy 中仅用一条语句。

这些功能对于机器学习的计算任务来讲，非常重要。因为不论是参数的批量计算，还是数据的特征表示，都离不开向量和矩阵的便捷计算。值得一提的是，NumPy 采用了非常独到的数据结构设计，使之在存储和处理大型矩阵方面，比 Python 自身的嵌套列表结构要高效得多。

但 NumPy 被定位为数学基础库，属于比较底层的 Python 库，其地位趋向于成为一个被其他库调用的核心库，而那些高级库通常能提供更加友好的接口。如果想快速开发出可用的程序，建议使用更为高阶的库 SciPy 和 Pandas，下面分别对它们进行简单介绍。

### 4.1.2.2 科学计算 SciPy

SciPy<sup>②</sup>的发音为“Sigh Pie”，它的取义类似NumPy，是“Science（科学）”和“Python”的组合，即面向科学计算的Python库。SciPy构建于NumPy之上，其功能更为强大，在常微分方程

<sup>①</sup> <http://www.numpy.org/>

<sup>②</sup> <https://www.scipy.org/index.html>

求解、线性代数、信号处理、图像处理及稀疏矩阵操作等方面，均有出色的支持。

相比于 NumPy 是一个纯数学的计算模块，SciPy 是一个更为高阶的科学计算库。比如，要做矩阵操作，这是纯数学的基础模块，可在 NumPy 库中找到对应的模块。但如果想要实现特定功能的稀疏矩阵操作，那这个模块就需要在 SciPy 库中找了。SciPy 库需要 NumPy 库的支持。出于这种依赖关系，NumPy 库的安装要先于 SciPy 库。

#### 4.1.2.3 数据清洗 Pandas

Pandas<sup>①</sup>的全称是“Python Data Analysis Library”，这是一款基于 Python 的数据分析库，它同样基于 NumPy 构建而成。

Pandas 库提供了操作大型数据集所需的高效工具，支持带有坐标轴的数据结构，这能防止由于数据没有对齐、处理不同来源、采用不同索引的数据而产生的常见错误。在数据预处理或数据清洗上，Pandas 提供了处理缺失值、转换、合并及其他类 SQL 的功能，这些功能大大减轻了一线从事机器学习的研发人员的负担。在某种程度上，Pandas 是实施数据清洗/整理（Data Wrangling）最好用的工具之一。

#### 4.1.2.4 图形绘制 Matplotlib 与 Seaborn

众所周知，MATLAB、R 及 gnuplot 等都提供了非常出色的绘图功能。事实上，Python 也提供了绘图功能同样强大的类库 Matplotlib<sup>②</sup>。它可以很方便地绘制散点图、折线图、条形图、直方图、饼状图等专业图形。此外，Matplotlib 还提供了一定的互动功能，如图形的缩放和平移等。其输出的常见文件格式有 PDF、SVG、PNG、BMP 和 GIF 等。

类似于 NumPy 是 Pandas 的基础库一样，Matplotlib 也可成为其他更高阶绘图工具的基础库。Seaborn<sup>③</sup>就是这样的高级库，它对 Matplotlib 做了二次封装。Matplotlib 功能虽然很强大，但用好却有较高的门槛。比如，由 Matplotlib 绘制的图形，如果还想更加精致，就需要做大量的微调工作。因此，为简单起见，在某些场合，可用 Seaborn 替代 Matplotlib 绘图，图 4-3 所示的是 Seaborn 绘制的可视化图，其专业程度可见一斑。

<sup>①</sup> <http://pandas.pydata.org/>

<sup>②</sup> <https://matplotlib.org/>

<sup>③</sup> <https://seaborn.pydata.org/>

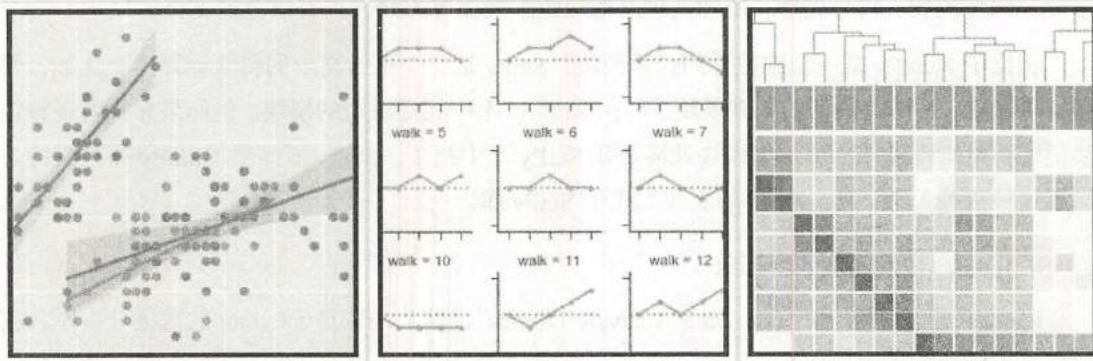


图 4-3 用 Seaborn 绘制的可视化图（图片来源：Seaborn 官网）

#### 4.1.2.5 Anaconda

Python 易学，但用好却不易。其中让初学者头疼的问题之一，就是类库和版本管理的问题。目前，Python 官方同时维护着 2.x 和 3.x 两个版本。有时，二者之间的差异，简直可视为两种不同的语言。

熟悉 Linux 的读者可能或多或少地有过软件包安装失败的经历。失败的源头在于，一个软件包的安装，可能还需要依赖于其他软件包，而普通用户又很难搞清楚谁依赖谁。怎么办呢？不同的 Linux 发布版均达成共识，让专业的工具干专业的活。例如，Ubuntu 提供了 apt-get，CentOS 提供了 yum 等管理工具，使用它们可简化软件的安装与卸载操作。

类似的，在 Python 中安装类库，同样也可能存在类库之间相互依赖、版本彼此冲突等问题。但凡有用户痛苦的地方，就有人提供解决方案。为了解决这一问题，Python 社区提供了方便的软件包管理工具——Anaconda<sup>①</sup>。

Anaconda 是一个用于科学计算的 Python 发行版，支持 Windows、Linux 及 Mac 等三大系统。它提供了强大而方便的类库管理（提供超过 1000 个科学数据包）与环境管理（包依赖）功能，可很方便地解决多版本 Python 并存、切换以及各种第三方库的安装问题。

Anaconda 通常利用 conda 进行类包和环境的管理。conda 的设计理念是，把所有的工具、第三方包都一视同仁当作包（package）对待，甚至包括 conda 自身。在安装完 Anaconda 之后，在命令行就可把 conda 当作一个可执行命令使用。

<sup>①</sup> <https://www.anaconda.com/>

conda 使用最多的参数就是 install 命令。比如，我们要安装前文提到的科学计算库 SciPy(库名为 scipy)，则在命令行输入如下命令即可。

```
conda install scipy
```

如果想查看已经安装的类库，可使用如下命令。

```
conda list
```

如果想卸载已经安装的类库，反向使用 uninstall 命令即可，如下所示。

```
conda uninstall <类库名>
```

#### 4.1.2.6 scikit-learn

机器学习是当下的研究热点，Python 社区更是在此领域引领潮流，scikit-learn<sup>①</sup>便是其中的佼佼者之一。它构建于NumPy、SciPy和Matplotlib之上，提供了一系列经典机器学习算法，如聚类、分类和回归等，并提供统一的接口供用户调用。近十多年来，先后有超过 40 位机器学习专家参与 scikit-learn 代码的维护和更新工作<sup>[1]</sup>，它已成为当前相对成熟的机器学习开源项目。

事实上，除了前面提及的几个常用类库和工具之外，Python 还提供了其他一些实用库。比如，用于网站数据抓取的 Scrappy，用于网络挖掘的 Pattern，用于自然语言处理的 NLTK 和用于深度学习的 TensorFlow 等。

如果读者刚开始学习 Python，建议首先熟悉前文提及的 7 个类库和工具，循序渐进，逐步扩展自己对 Python 的认识。

## 4.2 Python 的版本之争

众所周知，Python 官方同时支持两个版本，Python 2.x 和 Python 3.x。截至 2017 年 8 月，它们的最新版本分别是 2.7 和 3.6。由于一些历史遗留问题，导致这两个版本无法兼容，甚至部分语法都不一致，这给用户带来了极大困扰。

大家不禁要问，Python 2.x 和 Python 3.x 到底是什么关系？用 Python 官方的一句话，可简

---

<sup>①</sup> <http://scikit-learn.org/>

明扼要地道出二者的区别：Python 2.x 是过往的历史，而 Python 3.x 则代表当下和未来（Python 2.x is legacy, Python 3.x is the present and future of the language）。

但历史是有惯性的。到目前为止，仍有为数不容小觑的 Python 2.x 拥趸。本书选择 Python 3.6 作为后续机器学习算法实现的编程载体，原因在于，我们的视角应该站在当下，展望未来。

事实上，我们选择 Python 3，也是有现实考虑的，因为它能更加友好地支持中文。其实，Python 2 和 Python 3 之所以会在版本上发生重大“裂变”，其底层的诱因之一就是对字符编码——Unicode 的支持程度上有显著不同。

Python 3 是强支持 Unicode 的，而 Python 2 是弱支持 Unicode 的。也就是说，Python 2 既支持 ASCII，又支持 Unicode。表面看起来，似乎 Python 2 更加全面且强大。其实不然，因为这违背了 Python 的设计哲学。

Brett Cannon 是 Python 的核心开发者之一。2015 年 12 月，他写了一篇博客，系统地阐述了 Python 3 为什么会存在<sup>[2]</sup>。

Tim Peters（参见图 4-4）是另一位 Python 的核心开发者。他曾精彩地总结了 Python 的设计哲学，江湖人称“Python 之禅”（The Zen of Python<sup>①</sup>）。在这篇只有 19 行的小短文中，有这么一句经典描述：“there should be one—and preferably only one—obvious way to do it（应该有一个——最好只有一个——显而易见的方式去实现）”，这句话的通俗解读就是，“不要给我多选让我累，请给我最优让我醉。”

回到 Python 的讨论上。Python 2 给了用户多个选择，从而导致很多潜在的代码缺陷（bug）。下面举例说明，请问下面的文字代表什么语义呢？

```
'abcd'
```

对于 Python 3 的用户来说，答案简单明了，它就是一个由“a”“b”“c”“d”四个字母构成的字符串（str）对象。

<sup>①</sup> <https://www.python.org/dev/peps/pep-0020/>，读者启动 Python 解释器后，输入 import this 就可以看到全文。

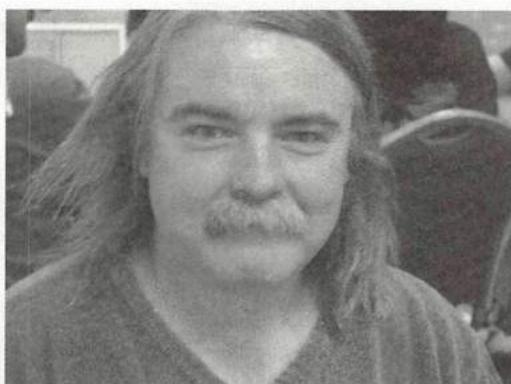


图 4-4 “Python 之禅”作者 Tim Peters

但对于 Python 2 的用户来说，答案可就没那么明确了。除了上述答案之外，它还可以解释为是由 97、98、99、100 构成的 4 个字节。

现在你看到了吧，Python 2 的用户存在“二选一”的解释（也就是存在歧义），文字既能代表文本数据，又能代表二进制数据，有时候会很麻烦。一旦对象脱离用户的控制，程序的结果将无从知晓，进而代码的缺陷可能就潜伏其中。而 Python 3 的答案却是唯一的。前面我们已经提到，Python 的设计理念是，“给我最优，别让我选”。也就是说，Python 2 对文字的处理方式是有悖于其设计哲学的。

也许有人会说，字符串解析有歧义的问题，在 Python 2 中也能得以解决，只要我们用 Unicode，而不是用 str 对象去表示文本，不就行了？这样说是没错。但在现实中，人们并不总会这么做。原因有二：要么嫌太麻烦，因为使用 Unicode 编码会带来额外的工作；要么对性能要求很高，不想承担因 Unicode 解码而带来的性能损失。

回顾 Python 发展史，我们会发现，早在 1991 年 2 月，Python 之父吉多·范罗苏姆（Guido Van Rossum）就发布了 Python 的第一个版本，Python 之父的相片如图 4-5 所示。相比较而言，Unicode 的第一版发布于 1991 年 10 月。也就是说，Python 的诞生早于 Unicode，这就导致 Python 并不是原生态支持 Unicode 的。

后来（1994 年左右），Python 也支持了 Unicode，但为了向后兼容，Python 也同时支持早期的 ASCII 单字节编码。这样一来，Python 2 的用户不得不面临“二选一”（甚至多选一，因为 Python 2 支持字符编码方案远不止这两类）以及彼此间相互转换的问题。这样的政策，实际上违背了 Python 的简捷设计理念。

Cannon 说，避免代码缺陷是一件非常重要却常被人忽视的事。“Python 之禅”中也指出“显

胜于隐（*explicit is better than implicit*）”，其含义就是，歧义和隐性会降低代码的可读性，进而更容易犯错。因此，Python 3 从诞生之日起，就立志于简化语言，并移除一切可能带来模糊性的概念，以减少 Python 代码出错的潜在风险。



图 4-5 Python 之父 Guido Rossum（左）与核心开发者 Brett Cannon（右）

另一方面，Python 的目标是成为一门面向世界的语言，而非仅支持以 ASCII 码为主导的罗马语系。在 Python 2 中，按照是否支持 Unicode，文本信息可被分为两大类。但根据 Python 的设计哲学，它尽量不让用户做选择题，而是直接给出最优解。这就是促使 Unicode 成为 Python 的必选项，而非可选项。

难道“多选”不好吗？这样可以各取所需啊，你或许会有这样的疑问。其实理念不存在绝对的好与坏，每一种存在都有其合理的一面。相比于 Python 2 而言，Python 3 的设计理念，有点类似苹果手机。苹果手机的核心软件、硬件，都属于闭源生态的杰作，普通用户通常难以置喙，别无选择。但苹果手机给用户的体验，也基本上是最优的。反观种类繁多的安卓手机则不同，安卓用户有着高度的选择裁量权，但这也导致用户很容易迷失于茫茫的选择之中，体验未必好。

Python 3 和 Python 2 的设计理念，差异如此之大，如果它想更加完美地体现“Python 之禅”中的设计理念，Python 必须做出重大蜕变，它要抛弃 Python 2 的多选约束，自我革命，涅槃重生！

于是，2008 年 12 月，Python 3 终于诞生了！

Python 3 将文本数据和二进制数据彻底分离，以避免歧义。同时，它也促使所有文本信息都支持 Unicode，从而使得 Python 项目更容易在多种语言下工作。

为了避免让用户陷入“选择困难”，Python 3 还干了一件让业界“瞠目结舌”的事，即干脆不向后兼容 Python 2。其实，这是一个艰难的抉择。原因显而易见，在 Python 3 问世之际，Python 2 及与之兼容的老版本已整整服务世人达 16 年之久，其生态系统已相对成熟。据 Cannon 介绍，时至今日，Python 3 社区的代码总量，想要赶超 Python 2，可能还需要数十年的时间。但是，Python 的开发团队态度决然，愿意承受由 Python 2 向 Python 3 的转型之痛，以维护 Python 的设计哲学。

但鉴于历史的惯性，Python 2 还有着庞大的用户群。所以，Python 官方不得不同时维护这两个不同版本的生态系统。但按 Python 官方的说法，Python 3 会不断吐故纳新、昂首阔步地大发展，而只会对 Python 2 做补丁级的小修小补。

可以相信，随着时间的推移，Python 3 一定会成为编程世界的主流，至少在 Python 社区如此。有一个标志性事件已在验证这个趋势的到来。2017 年 11 月，数据处理的功臣 NumPy 项目宣布，将在 2020 年停止支持 Python 2，因为继续支持 Python 2 正日益成为该项目的负担。<sup>①</sup>

## 4.3 Python 环境配置

Windows 和 Mac OS 是程序员使用较多的操作系统。下面我们将分别讲解这两个操作系统下的 Python 的环境配置。限于篇幅，Linux 下的配置高度类似 Mac OS，这里暂不涉及。

### 4.3.1 Windows 下的安装与配置

由于 Windows 版本的不同，可能会导致安装配置流程有所不同。需要说明的是，本书使用的操作系统版本为 Windows 7 英文版（64 位），Python 版本为 3.6。选择与操作系统匹配的 64 位版本的 Python，可在 Python 官方网站 <https://www.python.org/downloads/windows/> 下载，如图 4-6 所示。

<sup>①</sup> <https://github.com/numpy/numpy/blob/master/doc/neps/nep-0014-dropping-python2.7-proposal.rst>



图 4-6 Python 的 Windows 版本下载

下载完毕后，双击软件包（python-3.6.2-amd64.exe）进行安装。在安装过程中，如果不进行路径修改的话，一路按默认选项安装即可。但如果想自己指定安装路径，则需要选择“自定义安装（Customize installation）”，如图 4-7 所示。



图 4-7 自定义安装界面

然后在后续界面中，在“自定义安装位置（Customize install location）”处输入安装位置（如 C:\python），然后单击“安装（Install）”按钮，如图 4-8 所示。

按安装向导进行安装，直至出现安装成功界面，如图 4-9 所示。

接下来，我们解释一下 Python 的环境变量（Path）。既然本书是面向零基础的读者，这里我们不妨解释一下什么是环境变量。



图 4-8 配置自定义安装路径

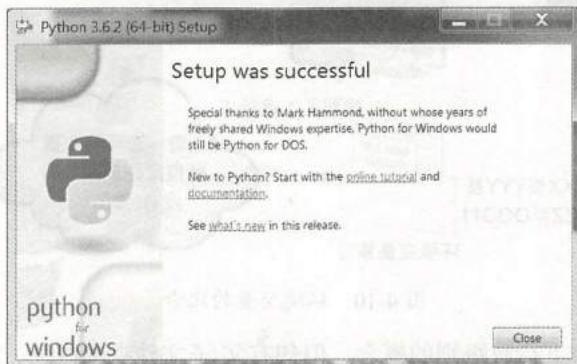


图 4-9 Python 安装成功界面

在介绍环境变量的含义之前，我们先举一个形象的例子，让读者有一个感性的认识。比如，如果我们喊一句：“张三，你妈妈喊你回家吃饭！”可是“张三”为何人？他在哪里呢？对于我们人来说，认识不认识“张三”都能给出一定的响应。如果认识他，可能就会给他带个话；如果不认识他，也可能帮忙吆喝一声“张三，快点回家吧！”

然而，对于操作系统来说，假设“张三”代表的是一条命令，它若不认识“张三”是谁，也不知道“它”来自何处，便会“毫无情趣”地说，不认识“张三”：“not recognized as an internal or external command（错误的内部或外部命令）”，然后拒绝继续服务。

为了让操作系统“认识”张三，我们必须给操作系统有关张三的精确信息，如“XXX省YYY县ZZZ乡QQQ村张三”。但其他问题又来了，如果“张三”代表的命令是用户经常用到的，每次使用“张三”，用户都在终端敲入“XXX省YYY县ZZZ乡QQQ村张三”，是非常烦琐的，有没有更加简单的办法呢？

答案是，当然有！聪明的系统设计人员想出了一个简易的策略，就是使用环境变量。把“XXX

省 XXX 县 YYY 县 ZZZ 乡 QQQ 村”设置为常见的“环境”，当用户在终端敲入“张三”时，系统自动检测环境变量集合里有没有“张三”这个人，如果在“XXX 省 YYY 县 ZZZ 乡 QQQ 村”中找到了，就自动将它替换为这个精确的地址“XXX 省 YYY 县 ZZZ 乡 QQQ 村张三”，然后继续为用户提供服务。如果整个环境变量集合里都没有“张三”，再拒绝服务也不迟，如图 4-10 所示。



图 4-10 环境变量的比喻

操作系统里没有上/下行政级别的概念，但却有父/子文件夹的概念，二者有异曲同工之处。对“XXX 省 YYY 县 ZZZ 乡 QQQ 村”这条定位路径，操作可以用“/”来区分不同级别的文件夹，即 XXX 省/YYY 县/ZZZ 乡/QQQ 村，而“张三”就像这个文件夹下的可执行文件。

下面我们给出环境变量的正式定义。环境变量是指在操作系统指定的运行环境中的一组参数，它包含一个或多个应用程序使用的信息。环境变量一般是多值的，即一个环境变量可以有多个值。在 Windows 环境下，各个值之间以英文状态下的分号“;”（半角的分号）进行分隔。对于 Mac 和 Linux 等系统，则用半角冒号“:”隔开。

对于 Windows 等操作系统来说，一般都有一个系统级的环境变量 Path（路径）。当用户要求操作系统运行一个应用程序，却没有指定应用程序的完整路径时，操作系统首先会在当前路径下寻找该应用程序，如果找不到，便会到环境变量“Path”指定的路径下寻找。若找到该应用程序则执行它，否则会给出错误提示。用户可以通过设置环境变量，来指定程序运行的位置。

例如，Python 程序的解释器 python.exe，这个命令不是 Windows 系统自带的命令，也就是说，它是外部命令，所以用户需要通过设置环境变量来指定这个命令的位置。设置完成后，就可以在任意目录下使用这个命令了，而无须每次都要输入这个命令所在的全路径。需要提醒大

家的是，在 Windows 下，命令和路径是不区分大小写的，所以将路径“C:\python”写作“C:\Python”是一样的。但在 Mac/Linux 环境下则不同，它们严格区分大小写，错敲一个字母都会“失之毫厘，谬以千里”。

那么，该如何在 Windows 中配置这个 Path 环境变量呢？其过程并不复杂，首先选中桌面上的“电脑（Computer）”图标，单击鼠标右键，在弹出的快捷菜单中单击“属性（Properties）”命令，打开如图 4-11 所示的对话框。

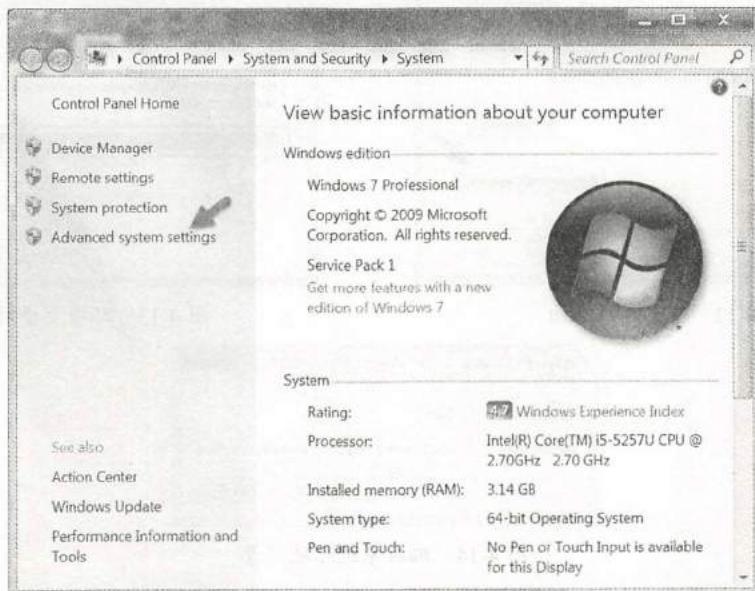


图 4-11 “我的电脑”属性对话框

然后，在图 4-11 所示对话框的左边栏中，单击“高级系统设置（Advanced system settings）”项。接着在弹出的窗口中，选择“高级（Advanced）”选项卡，然后单击“环境变量（Environment Variables）”按钮，如图 4-12 所示。

接下来，在弹出的对话框中拖动下面选区的滚动条，找到需要修改的环境变量“Path”，然后单击“编辑（Edit...）”按钮，如图 4-13 所示。

在弹出对话框中的“变量值（Variable value）”文本框的尾部添加“;C:\python”，如图 4-14 所示。

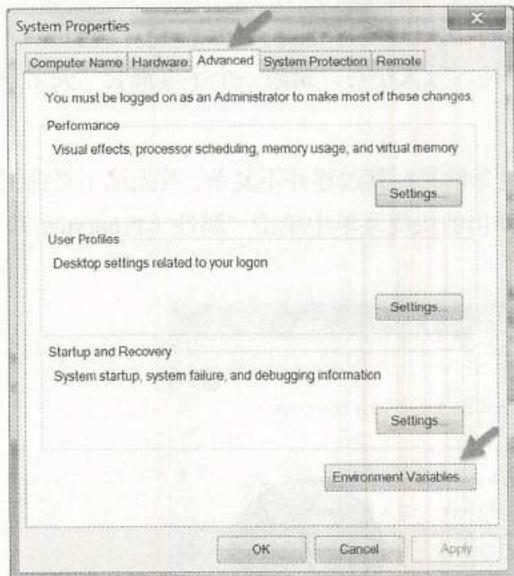


图 4-12 系统属性界面

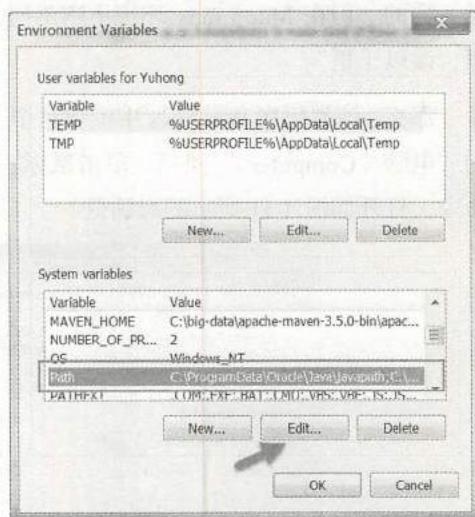


图 4-13 环境变量对话框

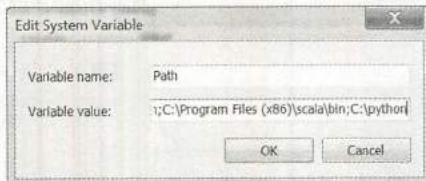


图 4-14 编辑系统环境变量

在这里需要注意两点：(1) 读者要灵活根据自己的 Python 安装位置，设置这个环境变量的值，不要拘泥于和本书完全一样。(2) 路径前面的半角分号“;”不可少，它是与前面一个环境变量值的分隔符。

另外，建议读者把 Python 安装目录下的 Scripts（脚本文件）也放到“Path”环境变量中，在本书中的路径是“C:\python\Scripts”。之所以这么做，是因为该目录下有非常好用的类库安装工具 pip3 和 easy-install 等，它们可以非常便捷地帮助我们在控制台模式安装类库，如图 4-15 所示。

最后，在 Windows 运行窗口中输入“CMD”命令，进入 DOS 控制台状态，然后在控制台输入“python”（后缀名.exe 可省略），检查一下是否出现 Python 特有的命令提示符“>>>”。如果出现，则说明环境变量配置成功，如图 4-16 所示。配置好环境变量，在控制台的任何路径下直接输入“python”都可调出这个解释器。

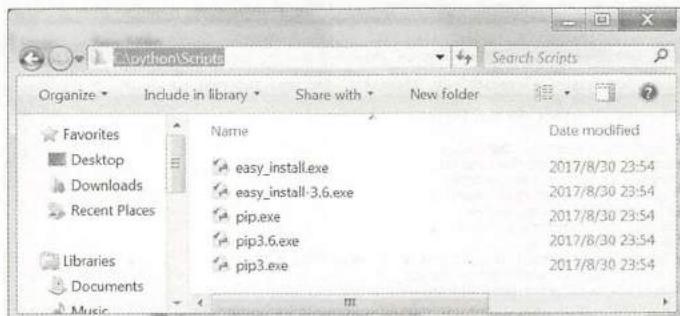


图 4-15 Python 安装目录 Scripts

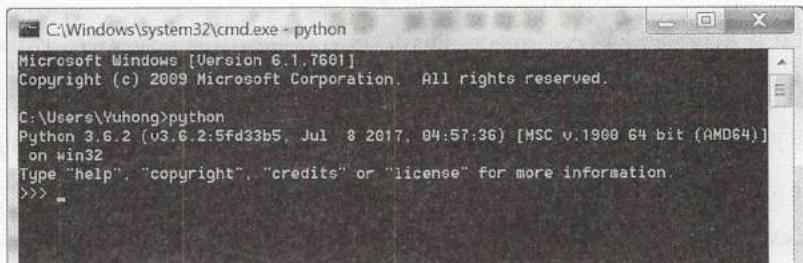


图 4-16 Python 解释器的运行界面

在 Windows 命令行中，按 Ctrl+Z 组合键，再按 Enter 键，即可退出 Python 解释器的运行界面，当然也可以输入 exit() 函数，以杀掉进程的方式退出。

配置好环境之后，如果想安装前面提到的类库（如 NumPy），在 DOS 控制台输入如下命令即可。

```
pip3 install numpy
```

如果想安装其他类库，只需把上述命令参数“numpy”换成其他类库名即可。

从图 4-16 可看出，控制台下的 Python 解释器确实有点简陋。细心的读者可能注意到在图 4-9 中，Python 官方将一个特别的感谢送给一个特别的人——Mark Hammond。是的，他值得感谢。因为是他给 Windows 下的用户提供了一个更加好用的集成开发环境（Integrated Development Environment, IDE）——IDLE。

IDLE 是一个非常友好的开发环境。比如，它有智能的语法提示（输入部分代码，然后按 Tab 键提示补全），它还提供代码颜色的差异显示等。我们可从 Windows 7 左下角的 Windows 图标 → All Programs（所有程序）→ Python 3.6 中找到它，打开后，IDLE 的界面如图 4-17 所示。

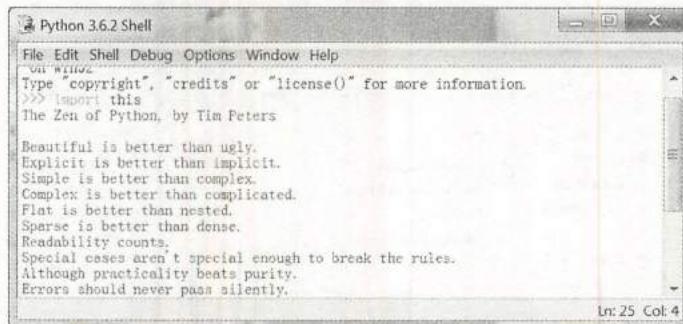


图 4-17 Python 3.6 的 IDLE 界面

可以尝试在提示符>>>下输入“import this”，这时就能输出前文提及的“Python 之禅”小短文。“import”是 Python 导入类库的关键词，后面我们还会讲到它，这里暂不展开介绍。

当然，除了默认安装的 IDLE 开发环境，市面上还有很多第三方的更加专业的 IDE 开发环境，如 Eclipse、IPython 及 PyCharm 等。事实上，万变不离其宗，所有的 IDE 都对 Python 的解释器“python.exe”做了封装。严格来讲，这些 IDE 实质上都是好用的代码编辑器，“好用”意味着可节省开发者的大量时间，因此，有些编辑器被程序员们戏称为“编程神器”。

### 4.3.2 Mac 下的安装与配置

由于 Mac OS 的出色性能，它在程序员世界里应用非常广泛。下面我们将对 Mac OS 版本的 Python 配置进行简单介绍。

#### 4.3.2.1 Python 3 的安装与环境变量配置

通常，Mac OS 会默认安装 Python 2.x 的解释环境。在终端输入“python（全部小写）”，即可得到如图 4-18 所示的界面。再次强调，类 UNIX 系统（如 UNIX、Linux、Mac 等）都是区分大小写的。所以，上述指令不可敲成“Python”，而这一点，在不区分大小的 Windows 中是无所谓的。

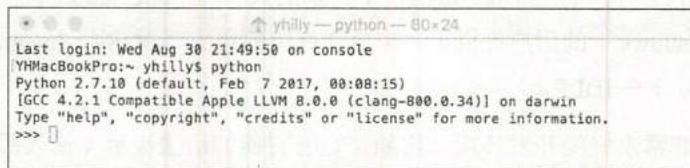


图 4-18 Mac 终端显示默认的 Python 解释器

在控制台中，按下 Ctrl+D 组合键，即可退出 Python 解释器界面。如前文所述，本书使用

的是 Python 3.6，需要读者去 Python 官方网站下载，安装过程类似于 Windows 下的安装过程，这里不再赘述。

安装完毕，可在应用（Application）文件夹或“Launchpad”中找到 IDLE 图标，然后单击该图标，便可启动 Python 3.6，如图 4-19 所示。

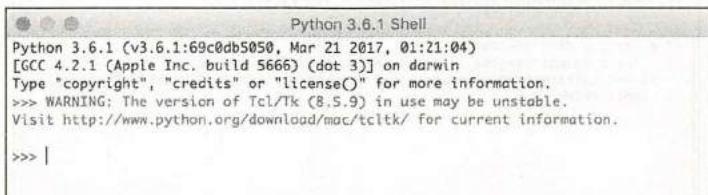


图 4-19 Mac 中的启动解释器 IDLE

与 Windows 操作系统不同的是，类 UNIX 系统中的大部分软件（特别是类库）的安装，是在控制台终端完成的，或许这么安装更加便捷，所以这里我们介绍一下在 Mac 终端安装类库的方法。

在 Mac 中安装 Python 类库，通常使用的工具是 pip（对应 Python 2.x 的版本）或 pip3（对应 Python 3 以上的版本）。可以在终端分别用“python3 --version”和“pip3 --version”来查询 Python 3 和 pip3 的版本号，如图 4-20 所示。

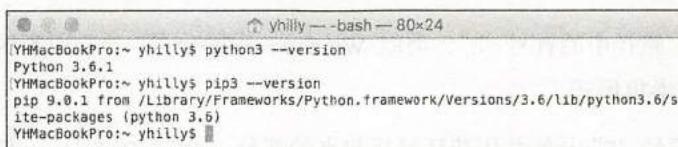


图 4-20 查询 Python 3 和 pip3 的版本号

如果大家没有成功运行上述查询，可检查一下家目录（即路径/user/home/username，或直接输入“cd ~”可进入）中的“.bash\_profile”配置是否正确，主要查看环境变量 PATH 是否涵盖了 Python 3 的安装路径。这里简单介绍一下“.bash\_profile”文件的功能，它的作用就是配置个人的环境变量，类似前面介绍的 Windows 环境变量配置。

所不同的是，Windows 提供了比较友好的图形界面配置环境，而 Mac 提供的是更加高效的字符配置环境。在 Mac、Linux 等系统中，凡是第一个字母为点（.）开始的文件或文件夹，表示隐藏状态，仅用“ls”命令通常是无法显示的，可用“ls -a”来显示它们，这里的“a”表示“所有（all）”文件的意思。

- 在 Mac 终端输入“vim .bash\_profile”，即可看到如图 4-21 所示的界面。