“电子与信息作战丛书”序

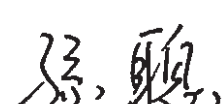
21 世纪是信息科学技术发生深刻变革的时代，电子与信息技术的迅猛发展和广泛应用，推动了武器装备的发展和作战方式的演变，促进了军事理论的创新和编制体制的变革，引发了新的军事革命。电子与信息化作战最终将取代机械化作战，成为未来战争的基本形态。

火力、机动、信息是构成现代军队作战能力的核心要素，而信息能力已成为衡量作战能力高低的首要标志。信息能力，表现在信息的获取、处理、传输、利用和对抗等方面，通过信息优势的争夺和控制加以体现。信息优势，其实质是在获取敌方信息的同时阻止或迟滞敌方获取己方的情报，处于一种动态对抗的过程中，已成为争夺制空权、制海权、陆地控制权的前提，直接影响整个战争的进程和结局。信息优势的建立需要大量地运用具有电子与信息技术、新能源技术、新材料技术、航天航空技术、海洋技术等当代高新技术的新一代武器装备。

如何进一步推动我国电子与信息化作战的研究与发展？如何将电子与信息技术发展的新理论、新方法与新成果转化为新一代武器装备发展的新动力？如何抓住军事变革深刻发展变化的机遇，提升我国自主创新和可持续发展的能力？这些问题的解答都离不开我国国防科技工作者和工程技术人员的上下求索和艰辛付出。

“电子与信息作战丛书”是由设立于沈阳飞机设计研究所的隐身技术航空科技重点实验室与科学出版社在广泛征求专家意见的基础上，经过长期考察、反复论证之后组织出版的。这套丛书旨在传播和推广未来电子与信息作战技术重点发展领域，介绍国内外优秀的科研成果、学术著作，涉及信息感知与处理、先进探测技术、电子战与频谱战、目标特征减缩、雷达散射截面积测试与评估等多个方面。丛书力争起点高、内容新、导向性强，具有一定的原创性。

希望这套丛书的出版，能为我国国防科学技术的发展、创新和突破带来一些启迪和帮助。同时，欢迎广大读者提出好的建议，以促进和完善丛书的出版工作。



中国工程院院士

译 者 序

在人工智能和大数据时代，每个人每天都在自觉或不自觉地制造各类网络信息，也在接收各类网络信息的轮番轰炸，如朋友圈的生活感悟、微博里的所见所闻、知乎里的专业解答等。所有的这些内容或真实存在或道听途说，更有居心不良之人的恶意伪造。其实，伪造具有非常悠久的历史，最早可以追溯到普通照片的伪造和篡改。近年来随着人工智能技术的发展，以深度学习技术为基础的深度伪造视频已出现在网络空间，深刻影响着这个世界。

“假作真时真亦假，真作假时假亦真”，网络空间开始出现很多真假难辨的人工智能生成内容。为了应对这种不良行为，切实加强对新技术、新应用、新业态的管理，我国于 2023 年 1 月 10 日起正式施行《互联网信息服务深度合成管理规定》(以下简称《规定》),《规定》对应用深度合成技术提供互联网信息服务制定了系统性、专门性规定，能够明确各类主体的信息安全义务，为促进深度合成服务规范的发展提供了有力的法律保障。

深度合成技术是指利用深度学习、虚拟现实等生成合成类算法制作文本、图像、音频、视频、虚拟场景等网络信息的技术。从技术发展的角度来说，深度合成技术经历了从自动编码器、生成对抗网络到深度伪造的发展历程。要想深入了解深度合成的工作原理，合理地利用深度合成技术造福人类，了解其中的技术方法是不可或缺的。

本书是一本全面介绍深度合成技术的著作，内容全面细致、知识由浅入深、理论联系实践，基本覆盖了深度合成的核心技术。原著作者兰纳姆是一位经验丰富的软件创新者，其在人工智能和机器学习领域开发出很多应用广泛的软件系统。译者希望能系统地将深度合成技术介绍给渴望新知识、有志在人工智能新时代中勇立潮头的朋友。

本书翻译分工如下：江刚武负责第 1～3 章，魏祥坡负责第 4～6 章，麻顺顺负责第 7、8 章，张贝贝负责第 9、10 章以及附录。全书由江刚武统稿。邱阳、慕中凯、柴飞鸿等学生在示例训练及国产化部署过程中付出了辛勤劳动与努力，译者在此表示感谢。本书的示例下载地址为 https://gitee.com/generatingnewreality/codefortraining。

由于译者的专业知识、语言能力和理论学术水平有限，书中难免存在不妥之处，敬请广大读者批评指正，译者的联系邮箱为 jianggw@163.com。

译　者

2025 年 6 月 10 日

原 书 前 言

现在是一个信息极其丰富、数据快速扩充的数字时代，随着人工智能生成内容（artificial intelligence generated content，AIGC）技术的快速发展，人们所处的现实世界在数字魔术大师和人工智能从业者充满想象力的改造下，逐渐成为一个现实与虚拟并存、真实与伪造交织的数字世界。与此同时，人们非常讨厌的虚假新闻、虚假图片和虚假人物形象也不可避免地进入人们的现实生活中。对许多人来说，这些数字伪造的虚假信息所带来的感觉和困惑是令人非常不舒服的。但是，也有很多人正在积极主动地应对这些数字伪造带来的负面影响，并寻找和探索新的发展机会。

本书将深入研究驱动这种新型数字伪造现实的全新技术，这种技术在人工智能领域和机器学习领域的广义名称为生成式建模。其实，这就是一种人工智能和机器学习技术的具体表现形式，其目的不是进行目标识别分类或者用来预测未来趋势，而是用来生成可以描述具体事物的多媒体内容，如图形、图像或文字等。

生成式建模技术并不是一个全新的概念，而是从深度学习的具体应用中逐渐产生并发展的，目前已经成为人工智能领域的研究热点之一。生成式建模有很多主流应用，大部分应用都在充分展示这项技术的实用价值，但不可避免地存在一些反对的声音。生成式建模技术刚开始应用时，很多人对其产生的人造的、不真实的事物会产生一种真实存在但广为揣测的担忧，也有人非常讨厌生成式建模技术制作的任何东西。然而，将生成式建模技术进行合理利用，也可以帮助人们提高工作效率，进而惠及各行各业。

本书的主要内容如下：第 1 章重点介绍深度学习理论基础，主要包括深度学习、自动编码器的基本概念，以及如何使用 PyTorch 框架构建简单的模型，并通过示例对相关概念进行进一步阐述，为后续章节的学习奠定基础。第 2 章重点介绍生成式建模的概念，主要阐述生成对抗网络的基础知识，以及如何使用生成对抗网络来生成新的内容，并给出生成对抗网络在生成时装和人脸等方面的具体应用示例。第 3 章重点介绍隐藏空间的基本概念，并探讨如何通过调整超参数、损失函数和网络配置等手段来控制隐藏空间。第 4 章重点介绍生成对抗网络，主要探讨生成对抗网络的五种变体，以及在模型学习和生成内容方式等方面的关键差异，加深对生成对抗网络概念的理解和认识。第 5 章重点介绍图像到图像的内容生成技术，在详细分析生成对抗网络高级应用的基础上，通过理解图像变换原理

来提高生成内容的质量，示例将使用各种强大的生成对抗网络模型，对配对图像和未配对图像之间的图像变换过程进行比对分析。第 6 章重点介绍残差生成对抗网络，将持续研究这一类网络模型的生成原理，以提高生成多样化和更接近真实事物特征的能力，本章中的生成对抗网络都使用残差网络，可以帮助识别和学习更加真实的事物特征。第 7 章重点介绍注意力机制，将自然语言处理领域广泛使用的注意力机制引入深度学习网络，以提供将相关特征和其他特征进行识别和映射的独特能力，示例结果展现出将注意力机制与生成对抗网络结合后的良好表现。第 8 章重点介绍高级生成器，深入探讨当前性能最佳的生成对抗网络，示例来自开源代码库，可以展示生成式建模在短时间内能够达到的非凡效果。第 9 章重点介绍深度伪造和换脸，详细阐述生成式建模在深度伪造方面的应用，示例描述基于开源桌面软件的深度伪造流程。第 10 章重点介绍深度伪造内容的检测，由创建深度伪造模型、深度伪造内容转向检测深度伪造内容，阐述目前正在开展的伪造内容检测技术和研究。将来，这些技术和工具可以帮助人们更好地接受数字现实以及理解什么是真实世界。

为了更好地理解书中内容，建议读者全面参与并认真完成书中的 40 个示例。所有的示例都可在 Google Colab 平台上进行测试和运行。有些示例可能需要几天的时间才能完成训练，但大部分示例可以在 1 小时内完成。

非常感谢读者抽出宝贵时间阅读本书，希望可以加深您对人工智能和机器学习专业知识的理解，并拓展您在这个领域的发展机会。当然，您所选择的这段学习旅程还是非常具有挑战性的，甚至可能会给您带来一定的挫折和困难。但是，当您第一次生成一张深度伪造的人脸图像时，肯定会感到非常惊奇，并对这项技术充满敬畏之心。

目　录

第 1 章　深度学习基础

纵观历史，人类一直在努力弄清楚什么是现实世界以及现实世界意味着什么。无论是从原始狩猎采集者到古希腊哲学家所经历的漫长岁月，还是近代文艺复兴时期，人类对现实世界的理解和解释都是随着时间的推移而逐渐走向成熟的。在过去被认为是神秘主义的事物，现在许多已经被科学解释和定义。

十几年前，很多人认为，人类开始步入可以真正理解现实世界的正轨。可是现在，随着人工智能技术的飞速发展，人们每天仍然能看到很多新生事物在周围涌现，如 AlphaGo、ChatGPT、元宇宙、智能驾驶等，正是深度学习和神经网络使这些新生事物的涌现成为可能。

多年来，深度学习和神经网络一直位于计算机科学的前沿交叉领域，非常神秘。对许多人来说，深度学习的抽象概念和深奥数学原理也使人们望而却步。虽然主流科学界多年来一直在持续探索深度学习和神经网络，但在许多传统行业，它们仍然是研究的盲区。尽管存在很多技术上的障碍，但深度学习仍然成为 21 世纪人工智能和机器学习领域最具发展潜力的新兴研究热点。

本书将探讨深度学习和神经网络底层的工作方式，不仅要学习神经网络的内部工作原理及作用机理，还要介绍如何配置神经网络来生成自己的内容和现实，这部分要阐述在不同版本的深度学习内容生成中的具体方法，包括换脸、提升旧视频的质量以及创造全新现实。

本章首先从深度学习的基础知识开始，举例说明如何为典型的机器学习任务构建神经网络，进而研究深度学习如何对数据进行回归和分类，并在内部了解具体的深度学习过程。然后继续介绍神经网络如何通过卷积算法专门提取数据中的特征，并使用有监督的深度学习模型构建一个完整的图像分类器。本章具体内容主要包括前提条件、感知器、多层感知器、PyTorch 和深度学习、回归分析和分类等。

1.1　前 提 条 件

尽管很多机器学习和深度学习的基本概念较为基础，但本书内容的深度还是远远超过深度学习的一般理解与学习的。要想通过深度学习网络成功地生成图像、文字等内容，最好能满足以下前提条件。

1. 数学基础

尽管在深度学习和生成式建模的实现过程中，大部分数学问题都将使用代码库进行处理，但读者仍然需要了解和使用以下数学知识：

(1)线性代数，用于处理矩阵和方程组。

(2)统计学和概率，理解如何描述统计工作和概率基本理论。

(3)微积分，掌握微积分的基础原理，以及如何利用微积分来处理函数的变化率。

2. 编程知识

本书将用到 Python 编程语言的很多知识，如果不了解 Python 语言，一定要提前学习相关课程。此外，还需要仔细了解以下几个库：

(1)NumPy[①]。NumPy(读作“numb-pie”)是用于操作数组或张量的算法库，也是机器学习和深度学习的基础。

(2)PyTorch[②]。PyTorch 是一个开源的 Python 机器学习库，是本书中各个深度学习项目的基础。

(3)MatPlotLib[③]。MatPlotLib 是基础的图形库，本书中的大部分输出成果将会用到这个库，具体将通过大量示例来说明它的用法。

3. 数据科学和机器学习

数据科学和机器学习有助于人们了解机器学习中使用的统计方法，以及处理数据时需要注意的事项。

4. 计算机

本书中的所有示例都是在云端开发的，虽然可以在移动设备上使用，但还是建议使用计算机进行练习，以达到最佳效果。附录 A 提供了在本地计算机上设置和使用代码示例的说明。如果运行示例需要配备高级图形处理器(graphics processing unit，GPU)，或者需要运行一个超过 12h 的示例，计算机的性能也是一个需要考虑的前提条件。

5. 时间

生成式建模通常非常耗时，本书中的有些示例可能需要数小时甚至数天的时

① NumPy 是 http://numpy.org 上的开源项目。

② PyTorch 是 http://pytorch.org 上的开源项目。

③ MatPlotLib 是与 Python 一起大量使用的开放源码包，可在网站 https://matplotlib.org 上获取。

间才能完成，需要耐心等待。

6. 开放学习

本书为读者提供了大部分示例和资料，也可以借助其他相关资料进行扩展学习。

虽然强烈建议读者具备上述前提条件，但是开放的心态和学习的意愿是最重要的，这也是提高学习效果的首要条件。

1.2 感 知 器

虽然存在一些争议，但大多数人都认识到，神经网络的灵感来源于大脑，更具体地说，是脑细胞或神经元。早在 1958 年，Rosenblatt 就开发了基本的感知器模型，后来人们对其进行了改进。图 1-1 给出了基于生物神经元的感知器模型[①]，图中 w_1、w_2、w_3 表示输入值的权重。后来，Minsky 等在其著作《感知器》中，对感知器进行了深入分析，并对其进行了苛刻的批评，特别是不能有效处理异或（XOR）这种简单的布尔逻辑问题[②]。尽管后来发现这些批评大多是毫无根据的，但在当时却直接导致神经网络研究的衰落，并引发了第一个人工智能寒冬。人工智能寒冬是指所有人工智能的研究和开发都被停止或搁置，这些寒冬通常是由一系列阻碍该领域有所进展的主要障碍带来的。第一个寒冬是 Minsky 等对感知器的批评，他们认为这是由不能解决异或问题造成的。目前，已经出现了两个人工智能寒冬，这些寒冬的具体日期有待商榷，可能会因具体学科的差异而有所不同。

或许正是这种与大脑的关联导致对感知器和深度学习的一些批评，使得这种关联增加了神经网络的神秘性和不确定性。然而，感知器本身只是一种数学连接模型，并且常常把这种机器学习理论称为联结主义。要说生物神经元与感知器模型有什么区别，应该是感知器模型只与神经元的连接方式类似，实际上也仅此而已。事实上，大脑神经元的功能要复杂得多，其工作原理与感知器完全不同。

再次分析图 1-1 中的感知器模型，进一步探究感知器是如何接收由方框表示的多个输入的。这些输入值乘以权重值（w_1、w_2、w_3），以加权方式调整下一阶段的输入强度。在此之前，还有另一个偏差输入，其值为 1.0，也可将其乘以另一个权重进行输入，偏差允许感知器抵消前面输入的结果。将所有输入值和偏差进行

① Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain[J]. Psychological Review, 1958, 65(6): 386-408.

② Minsky M, Papert S. Perceptrons: An Introduction to Computational Geometry[M]. Cambridge :The MIT Press, 1969.

加权，在求和函数中进行汇总，然后将求和函数的结果传递给激活函数。激活函数的目的是进一步缩放、挤压或切断要输出的值。下面通过练习 1-1 创建一个简单的感知器模型。

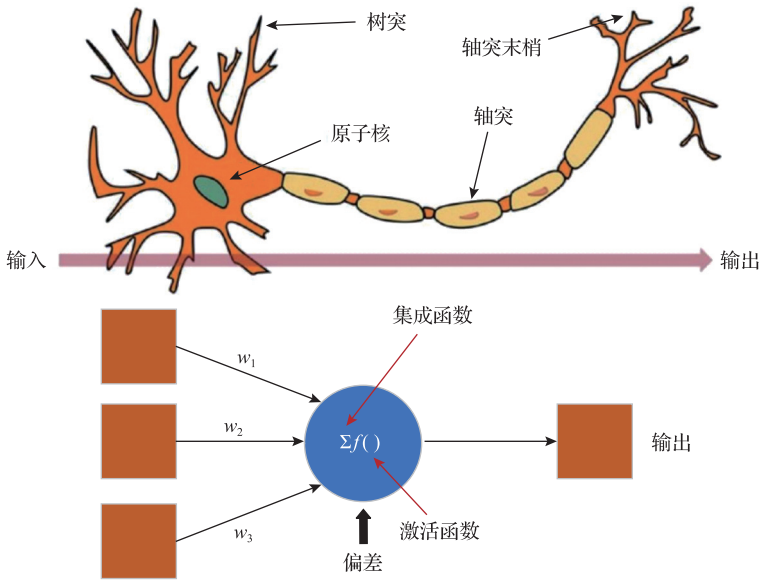


图 1-1　基于生物神经元的感知器模型

练习 1-1：创建感知器模型。

(1) 打开 GitHub 网站上的 GEN_1_XOR_perceptron.ipynb 文件。如果您不确定如何访问源代码，请查看附录 B。

(2) 在交互式笔记本的第一个代码块中，可以看到 NumPy 库和 MatPlotLib 库的一些导入，MatPlotLib 用于显示绘图。

```
Import numpy as np
Import matplotlib. pyplot as plt
```

(3) 滚动到异或问题代码块，如下所示，这是设置数据的地方，这些数据由用来训练感知器的 X 值和 Y 值组成。X 值表示输入，Y 值表示期望输出，通常将 Y 称为标签或预期输出。使用 numpy.np 模块，通过 np.array 创建张量的输入列表。在 numpy.np 模块的底部，输出这些张量的形状。

```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
Y = np.array([0,1,1,0])
print(X.shape)
print(Y.shape)
```

(4) 在这个初始测试问题中，使用的值来自如下所示的异或逻辑真值表。

输入		输出
X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

(5) 向下滚动并执行以下代码块。该代码块使用 matplotlib.plt 模块来输出同一真值表的三维表示，使用数组索引切片来显示 X 的第一列，然后是 Y，X 的最后一列作为第三维度。

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:,0], Y, X[:,1], c='r', marker='o')
```

(6) 创建感知器的第一步是确定输入的数量，并为这些输入创建权重，该过程将通过以下代码来实现。在这段代码中，通过取 X.shape[1]的第一个值来获得输入的数量，即 2。然后用 np.random.rand 随机初始化权重，并为输入添加偏差。回顾一下，偏差是感知器可以抵消函数的一种方式。

```
no_of_inputs = X.shape[1]
weights = np.random.rand(no_of_inputs + 1)
print(weights.shape)
```

(7) 权重初始化为随机值后，就有了一个有效的感知器，可以通过运行下一个代码块对感知器进行测试。在这个模块中，循环输入 X，并使用 np 的点积进行乘法和加法运算。此运算的结果即为感知器输出量的总和。这个代码块的输出还没有任何意义，因为仍然需要训练权重。

```
for i in range(len(X)):
  inputs = X[i]
  print(inputs)
  summation = np.dot(inputs, weights[1:]) + weights[0]
print(summation)
```

(8) 在下一个代码块中训练代码，用于训练感知器中的权重。可以在被称为训练轮次(epoch)的周期内，反复训练感知器或神经网络的数据。在每个周期或迭代期间，将每个数据样本单独或分批输入感知器或神经网络。在输入每个样本时，将求和函数的输出与标签或预测值 Y 进行比较，预测值和标签之间的差异称为损失。基于这一损失，可以根据稍后将详细讨论的公式来调整权重。整个训练代码如下所示：

```
learning_rate = 0.1
epochs = 100
history = []
for _ in range(epochs):
for inputs, label in zip(X, Y):
```

```
    prediction = summation = np.dot(inputs, weights[1:]) + weights[0]
    loss = label - prediction
  history.append(loss*loss)
  print(f"loss = {loss*loss}")
  weights[1:] += learning_rate * loss * inputs
  weights[0] += learning_rate * loss
```

(9)在运行完最后一个代码单元，运行最后一个编码单元，生成损失图，感知器“异或”训练的损失输出如图 1-2 所示。

```
plt.plot(history)
```

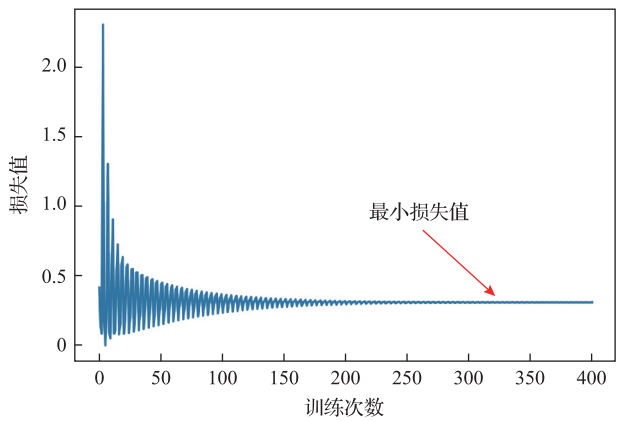


图 1-2　感知器“异或”训练的损失输出

练习 1-1 最终只能获得 0.25 的最小损失值，其结果并不那么优秀。如果运行更多轮次的训练，结果也不会变得更好。也就是说，尽管单个感知器能够解决一些更困难的问题，但是无法解决简单的异或问题，这就是 Minsky 等在其著作《感知器》一书中提出的核心批评观点。

在研究利用感知器解决更复杂问题之前，先回顾下前面示例中的代码学习行，并了解它们是如何工作的。为了便于复习，下面列出部分学习的代码行：

```
prediction = summation = np.dot(inputs, weights[1:]) + weights[0]
loss = label - prediction
...
weights[1:] += learning_rate * loss * inputs
weights[0] += learning_rate * loss
```

在这里，首先使用 np.dot 函数进行点乘求和计算得到预测值，然后通过计算标签值和预测值的差来计算损失值，最后使用如式(1-1)所示的函数更新权重：

$$w_i = w_i + \mathrm{lr} \times \mathrm{loss} \times \mathrm{input} \tag{1-1}$$

式中，w_i 为与输入矩阵匹配的权重值；lr 为学习率；loss 为标签值和预测值之间的差异；input 为感知器的输入数组。

在每次将数组输入感知器的过程中，就是用这个简单公式来更新权重。其中，学习率用于调整更新量，通常为 0.01 或者更小的值，要通过设置合理的学习率，将每次的迭代更新量控制在较小的范围之内，否则每次迭代更新都会导致感知器过度学习或学习不足。学习率称为超参数，是需要经常手动调整的第一个变量。之所以将其称为超参数，是因为一般把内部权重值称为参数，把这类可以影响和控制全局的变量称为超参数。

单个感知器或者单层感知器的最大缺点就是只能解决线性问题，由于异或问题不是线性问题，所以需要引入 2 个以上的感知器，也就是多层感知器（multi-layer perceptron，MLP）。在开始这样做之前，可以重新回顾一下感知器，看看它能解决什么样的问题。

练习 1-2 将研究如何使用感知器这样的线性工具解决更难的问题。现在是要解决二维线性回归问题，在十几年前，这类问题很难应用经典的回归方法来解决，更多关于回归的内容将在后面章节中介绍。现在开始练习 1-2。

练习 1-2：感知器和线性回归。

(1) 打开 GitHub 网站上的 GEN_1_perceptron_class.ipynb 文件。如果不知道如何访问源代码，请查看附录 B。

(2) 通过运行线性回归问题代码块设置数据，如下所示：

```
X = np.array([[1,2,3],[3,4,5],[5,6,7],[7,8,9],[9,8,7]])
Y = np.array([1,2,3,4,5])
print(X.shape)
print(Y.shape)
```

(3) 下一个代码块在图形上显示输入点：

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:,0], X[:,1], X[:,2], c='r', marker='o')
```

(4) 在本例中，仅在图 1-3 所示的图形上显示三维输入点。目标是训练感知器，使其能够学习如何将这些点映射到输出标签 Y 上。

(5) 进入代码部分，在这里设置参数和超参数。在练习 1-2 中，需要调整超参数、迭代次数和学习率，此处将学习率降低到 0.01。然而，在解决线性回归问题时，感知器可以比在解决异或问题时更快地学会映射输入值，所以也会减少迭代次数。

```
no_of_inputs = X.shape[1]
epochs = 50
```

```
learning_rate = 0.01
weights = np.random.rand(no_of_inputs + 1)
print(weights.shape)
```

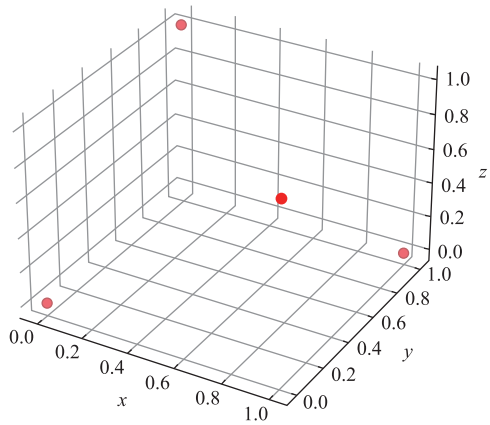


图 1-3　在三维图形上绘制的输入点

(6) 本练习将引入一个激活函数。通过激活函数可以缩放输出量的大小，以获得更好的输入或预测。在这个示例中，使用了一个修正后的线性函数，即修正线性单元(rectified linear unit, ReLU)，该函数能有效消除小于和等于 0 的输出，避免单纯线性输出可能产生的负值输出量。

```
def relu_activation(sum):
  if sum > 0: return sum
  else: return 0
```

(7) 把感知器的全部功能嵌入 Python 类中，以便更好地封装和重用。以下代码就是之前所有感知器和设置代码的组合。

```
class Perceptron(object):
   def __init__(self, no_of_inputs, activation):
     self.learning_rate = learning_rate
     self.weights = np.zeros(no_of_inputs + 1)
     self.activation = activation
  def predict(self, inputs):
    summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
    return self.activation(summation)
  def train(self, training_inputs, training_labels, epochs=100,
  learning_rate=0.01):
    history = []
    for _ in range(epochs):
```

```
        for inputs, label in zip(training_inputs, training_labels):
            prediction = self.predict(inputs)
            loss = (label - prediction)
            loss2 = loss*loss
            history.append(loss2)
            print(f"loss = {loss2}")
            self.weights[1:] += self.learning_rate * loss * inputs
            self.weights[0] += self.learning_rate * loss
        return history
```

(8) 对类进行实例化，并开始训练。

```
perceptron = Perceptron(no_of_inputs, relu_activation)
history = perceptron.train(X,Y, epochs=epochs)
```

(9) 图 1-4 显示了训练函数调用的历史输出，以及运行最后一组单元的结果。可以清楚地看到损失值几乎降低到 0，这意味着感知器能够根据输入值有效地预测和映射结果。

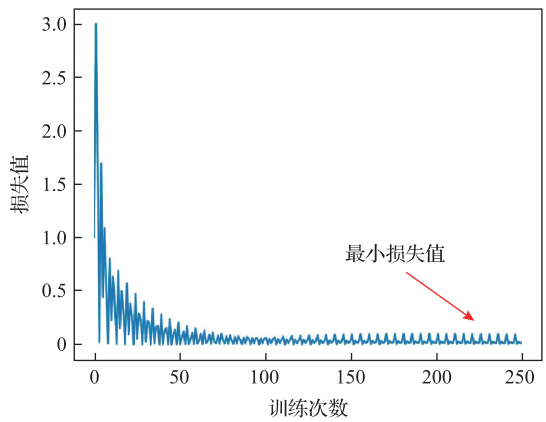


图 1-4　线性回归问题中感知器的输出损失

在图 1-4 中，可以看到网络模型的损失值存在明显的高频振动，这种现象可能是由学习率过高引起的，也可能与数据输入网络模型的方式有关。本书后面将会研究如何解决这类问题。

在将输入映射到预期输出方面，练习 1-2 的结果表现得更为成功。即使是处理典型的比较难的数学问题也有很好的结果，这也是在第一个“人工智能寒冬”中，感知器得以存活的重要原因。直到“人工智能寒冬”过去，大家才发现，如果将感知器堆叠成层，则可以实现更多功能，而且模型的能力更强，特别是能很好地解决异或(XOR)问题。

1.3　多层感知器

从原理上讲，将感知器堆叠成层的概念并不困难。图 1-5 给出了多层感知器网络模型示例，其第一层称为输入层，最后一层称为输出层，中间的则称为中间层或隐藏层。

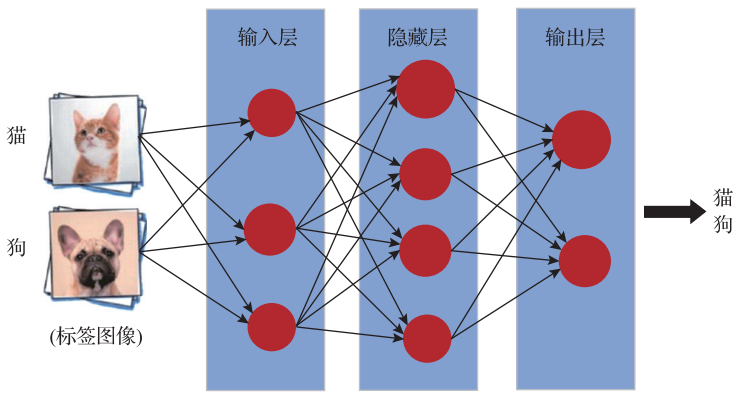


图 1-5　多层感知器网络模型示例

在图 1-5 中，猫和狗的图像输入多层感知器网络模型中，根据输出结果对输入图像进行分类，后面将讨论对输出进行分类的具体方法。图中的每个节点或圆圈代表一个感知器，每个感知器完全连接到网络模型中的各个层。这类网络模型称为全连接顺序网络。

通过网络模型向前传递的预测与感知器的运行原理基本一致，唯一的区别是上一层的输出成为下一层的输入，以此类推。通过将输入值传递给网络模型来计算输出值的过程称为前向传递或预测。从计算过程来看，使用点积函数后，数据传输线路中的前向传递非常有效，这是神经网络模型的优势。

在 1.2 节中，使用 np.dot 函数对输入的权重值进行求和。该函数在图形处理器上进行了优化，执行速度非常快。因此，即使有 100 万个输入值(这是可能的)，也可以在图形处理器上的一次操作中完成计算。np.dot 函数在图形处理器上被优化的原因是计算机三维图形学的发展，点积运算在图形处理中相当常见，从某种意义上来说，游戏和图形引擎的发展对人工智能和深度学习有很大帮助。

虽然前向传递或预测可以快速运行，但其计算难度并不大。但是，与之相反的训练更新或者后向传递就不那么容易了。当多个感知器堆叠时，会面临之前应用的简单更新公式无法跨网络层工作的问题。

在更新多层网络时，遇到的具体问题是，不仅要确定每个层的损失量，还要确定该层中每个感知器的损失量，即不能仅从预测中减去损失，然后将该值应用