arXiv:2301.00942v1 [cs.LG] 3 Jan 2023

深度学习与计算物理（讲义）

Deep Ray、Orazio Pinti 和 Assad A. Oberai[1]

[1]南加利福尼亚大学航空航天与机械工程系，美国加利福尼亚州洛杉矶

目录

前言 **3**

1 引言 4

1.1 计算物理.4 1.2 机器学习.5 1.2.1 机器学习示例.5 1.2.2 基于学习任务的机器学习算法类型.5 1.3 人工智能、机器学习与深度学习.6 1.4 机器学习与计算物理.7

2 深度神经网络简介 9

2.1 多层感知机架构.9 2.2 激活函数.10 2.2.1 线性激活.12 2.2.2 整流线性单元（ReLU）. 12 2.2.3 带泄漏的 ReLU.12 2.2.4 逻辑函数.12 2.2.5 双曲正切函数.13 2.2.6 正弦函数.13 2.3 网络的表达能力.13 2.3.1 通用逼近结果.15 2.4 神经网络的训练、验证与测试.15 2.5 泛化能力.17 2.5.1 正则化.17 2.6 梯度下降.18 2.7 一些高级优化算法.20 2.7.1 动量法.20 2.7.2 Adam.21 2.7.3 随机优化.21 2.8 使用反向传播计算梯度.23 2.9 回归与分类.25

3 残余屿 网络 **27**

3.1 深度网络中的梯度消失.27 3.2 残差网络.28 3.3 与常微分方程的联系.29 3.4 神经常微分方程.30

4 使用多层感知机求解偏微分方程 **33**

4.1 有限差分法.34 4.2 谱配置法.35 4.3 物理信息神经网络（PINNs）.38 4.4 将 PINNs 扩展到更一般的偏微分方程.40 4.5 物理信息神经网络的误差分析.41 4.6 使用物理信息神经网络进行数据同化.42

5 卷积神经网络 **44**

5.1 函数和图像.44 5.2 函数的卷积.44 5.2.1 例 1.45 5.2.2 例 2.46 5.3 离散卷积.46 5.4 与有限差分的联系.48 5.5 卷积层.48 5.5.1 平均池化和最大池化.50 5.5.2 多通道输入的卷积.51 5.6 卷积神经网络（CNN）.51 5.7 转置卷积层.53 5.8 图像到图像变换.56

6 算子网络 **57**

6.1 物理信息神经网络的问题.57 6.2 参数化偏微分方程.57 6.3 算子.58 6.4 深度算子网络（DeepONet）架构.59 6.5 DeepONet 训练.61 6.6 DeepONets 误差分析.62 6.7 物理信息 DeepONet.63 6.8 傅里叶神经算子——架构.64 6.9 傅里叶神经算子的离散化.66 6.10 傅里叶变换的使用.67

7 概率　概率深度学习 7 **0**

7.1 概率论的关键要素.70 7.2 随机变量.72 7.2.1 累积分布函数.72 7.2.2 概率密度函数.74 7.2.3 重要随机变量示例.75 7.2.4 随机变量的期望与方差.76 7.2.5 随机变量对.77 7.3 无监督概率深度学习算法.79 7.3.1 生成对抗网络.79 7.4 有监督概率深度学习算法.82 7.4.1 条件生成对抗网络.83

序言

这些讲义是为一门课程编写的，该课程由南加利福尼亚大学开发并授课。具备扎实应用数学背景的普通工程专业研究生应该能够理解这些讲义。

本讲义的主要目标是向熟悉线性代数和偏微分方程概念的学生介绍深度学习中的一些选题。本讲义利用深度学习算法与较为传统的计算物理技术之间的紧密联系，以实现两个目标。首先，借助计算物理中的概念，帮助读者理解深度学习算法。深度学习中的许多概念都能与计算物理中的类似概念建立联系，这一点并不令人意外；我们可以利用这种联系更好地理解这些算法。其次，有几种新颖的深度学习算法可用于解决计算物理中的难题。因此，对于有兴趣模拟物理现象的人而言，这些算法提供了一套互补的工具。

第1章

引言

本课程探讨处于计算物理 与机器学习交界处的主题。在理解为何需要结合这两个重要概念之前，我们首先需要分别了解它们各自的含义。

1.1 计算物理

计算物理在解决科学和工程领域的许多问题时发挥着基础性作用。为了理解这一概念，我们简要概述解决物理问题所涉及的关键步骤：

1. 考察一种物理现象，并收集某个感兴趣的可观测量的测量值。例如，研究海洋波浪时，可从海洋浮标获取水位高度和波向的测量值。

2. 根据观测结果，提出一条物理定律。例如，观测发现，封闭系统中的流体质量始终守恒。

3. 用数学语言描述该定律。这可以借助常微分方程（ODE）、偏微分方程（PDE）、积分方程等。

4. 建立数学模型后，求解该系统。可以通过两种方式获得解：

   (a) 在某些情况下，可以得到解的精确解析形式。例如，可以使用分离变量法、拉普拉斯变换、傅里叶变换或积分因子来求解常微分方程/偏微分方程。(b) 在大多数情况下，无法得到解的精确表达式，因此必须使用数值算法进行适当的近似。例如，可以使用前向欧拉法或后向欧拉法、中点法则或龙格–库塔格式来求解常微分方程系统；也可以使用有限差分法/有限体积法/有限元法来求解偏微分方程。

5. 设计好用于求解（精确或近似）的算法后，使用它来验证数学模型，也就是检查预测是否与收集到的数据吻合。

所有这些步骤大致描述了计算物理所包含的内容。

1.2 机器学习

与计算物理不同，机器学习（ML）不需要提出物理定律。其涉及的一般步骤如下：

1. 通过观察物理现象、对某个可观测量进行实时测量，或使用近似模拟该现象的数值求解器来收集数据。2. 使用收集到的数据训练合适的算法，旨在发现不同样本之间的模式或关系。具体示例请参见第1.2.1节。3. 训练完成后，使用机器学习算法进行未来预测，并利用额外收集的数据对其进行验证。

1.2.1 机器学习示例

1. **回归算法：** 给定一组成对数据 $\{(x_i,\ y_i):\ 1 \le i \le N\}$，它对应某个未知函数 $y = f(x)$，拟合一个多项式（或任何其他基函数）来处理这组数据，以近似 f。例如，求出线性函数 $\tilde{f}(x;\ a,\ b) = ax + b$ 的系数 $a,\ b$，使误差最小

$$\mathcal{E}(a,b) = \sum_{i=1}^{N} |y_i - \tilde{f}(x_i)|^2.$$

如果 $(a^*,\ b^*) = \arg\min_{a,\ b} \mathcal{E}(a,\ b)$，那么我们可以将 $f^*(x) := f(x;\ a^*,\ b^*)$ 视为 $f(x)$ 的近似（见图1.1(a)）。

2. **决策树：** 我们获得了一个样本总体的数据集，其中包含以下特征：年龄、收入。此外，数据被分为两组：A组中的个体拥有房屋，而B组中的个体没有。然后，给定一个新数据点的特征，我们希望预测这个新个体拥有房屋的概率。决策树可用于解决这一分类问题。典型决策树的工作方式是通过划分数据，使数据集中的样本按组别得到最大程度的分离（见图1.1(b)）。然后，算法根据这些划分，确定新数据点属于特定类别/组别的概率。

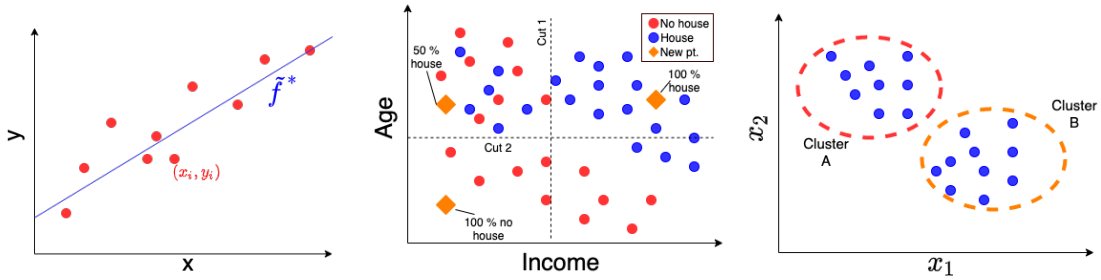
3. **聚类算法：** 给定一组数据，其中每个样本具有若干特征，找出数据中的簇/模式（见图1.1(c)）。

1.2.2 基于学习任务的机器学习算法类型

广义而言，机器学习算法有四种类型：

1. **监督学习：** 给定数据 $\mathcal{S} = \{(\boldsymbol{x}_i,\ \boldsymbol{y}_i):\ 1 \le i \le N\}$，预测某个新 $\hat{\boldsymbol{x}}$ 对应的 $\hat{\boldsymbol{y}}$，使得 $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \notin \mathcal{S}$。例如，给定一组图像及其图像标签（如狗、猫、牛等），训练一个分类机器学习算法来学习图像与标签之间的关系，并用它预测新图像的标签。

2. **无监督学习：** 给定数据 $\mathcal{S} = \{\boldsymbol{x}_i \in \Omega_x :\ 1 \le i \le N\}$，找出 Ω_x 不同区域之间的关系。例如，找出数据集中的簇，或者找出描述这些数据扩散情况的概率分布 $p_x(\boldsymbol{x})$ 的表达式，并据此生成新的样本。



(a) 线性回归　　(b) 决策树　　(c) 聚类

图 1.1：机器学习示例

3. **半监督学习：** 这类方法介于监督学习和无监督学习之间。它们通常结合有标签数据和无标签数据进行训练。例如，假设我们有 10,000 张无标签图像和仅 50 张有标签图像。我们能否利用这个数据集开发一种图像分类算法?

4. **强化学习：** 属于这一类的方法通过对所采取决策给予奖励或惩罚来进行学习。因此，算法会学习出一条合适的路径/策略，以最大化奖励。这类方法用于训练算法，使其能够下国际象棋或围棋。

在本课程中，我们将主要关注前两种机器学习算法类型。

1.3　人工智能、机器学习与深度学习　g

有时，人工智能（AI）、机器学习和深度学习（DL）这几个术语会被互换使用。实际上，它们是三个彼此相关但又有所不同的概念。通过观察图 1.2 中的维恩图，可以理解它们之间的关系。

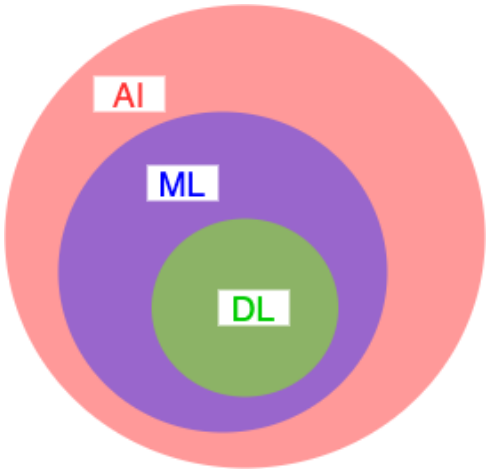


图 1.2：人工智能、机器学习与深度学习之间的关系

人工智能指的是具有类似人类智能的系统。机器学习是人工智能系统的关键组成部分，但系统还涉及其他要素。自动驾驶汽车是人工智能的典型例子。

我们来仔细看看这类系统的设计（见图1.3）。汽车上装有一台摄像机，用来实时拍摄前方道路的图像/视频。随后，这些帧会传递给机器学习算法，进行语义分割，即分割出帧中的不同区域，并对每个区域中的物体类型（汽车、树木、道路、天空等）进行分类。分割完成后，结果会传递给决策系统，由它根据分割后的图像决定汽车接下来应采取的行动。随后，这些信息会传递给控制模块，由它实际控制汽车的机械动作。整个过程模仿了真实驾驶员的行为，因此属于人工智能。

另一方面，机器学习（ML）是该系统中通过数据训练的分量。也就是说，它们通过数据进行学习。在上面的例子中，语义分割器就是这样一个系统。有许多机器学习算法可以利用数据执行这项任务，我们将在本课程中学习其中一些。决策系统也可以是机器学习分量，即从先前的数据中学习应当做出的适当决策。不过，它也可以不采用机器学习。例如，可以使用基于规则的专家系统。

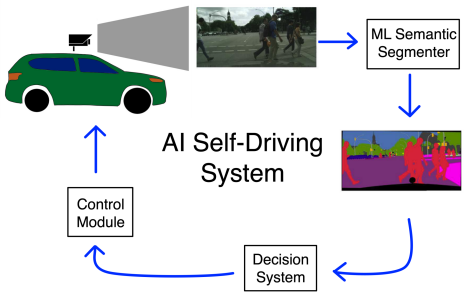


图1.3：自动驾驶汽车人工智能系统示意图。部分插图取自 [32]。

深度学习是机器学习算法的一个子集。深度学习架构中最简单的形式称为前馈网络，由多个非线性变换层组成。非线性变换逐分量地作用于中间输出的仿射变换结果。这种架构在一定程度上受到生物体中枢神经系统传递信号方式的启发。我们将在第2章中更详细地学习深度学习架构。

1.4 机器学习与计算物理

现在我们对计算物理和机器学习有了更深入的了解，下一个显而易见的问题是：“为什么要研究二者的结合？”以下列出几点动机：

- 对于复杂的“物理”数据模式，机器学习为表示数学定律提供了另一种途径。设想一个包含两个重要分量的物理过程。其中一个分量我们已经充分理解，并且有可靠的数学模型；另一个分量则尚未得到充分理解，也没有数学描述。在这种情况下，我们可以对第一个分量采用计算物理，对第二个分量采用机器学习。一个具体例子是受能量守恒定律和复杂本构模型支配的系统。对于前者，我们可能已经有充分理解的数学模型；而对于后者，我们可能不得不依靠机器学习来开发模型。

- ML 通常非常依赖大量数据。但物理学知识可以帮助限制 e

输入和解／预测所在的流形。有了这些约束，我们就能减少训练机器学习算法所需的数据量。

- 用于分析计算物理的工具（泛函分析、数值分析、关于收敛至精确解的概念、概率框架）可以延伸应用于机器学习。将这些工具应用于机器学习，有助于我们更好地理解并设计更优秀的机器学习算法。

我们简要概述本课程将涉及的各个主题：

- 深度神经网络（多层感知机）及其收敛性。
- 残差网络及其与非线性常微分方程（神经常微分方程）的联系。
- 递归网络及其与非线性常微分方程的联系。
- 卷积神经网络及其与偏微分方程的联系。
- 随机梯度下降及其与常微分方程的关系。
- 用于求解偏微分方程的深度学习算法。
- 用于逼近算子的深度学习算法。
- 生成对抗算法及其与计算物理的联系。

第2章

深度神经网络简介

在本章中，我们将更加仔细地了解目前已知的最简单网络架构，即多层感知机 （MLP）。

2.1 多层感知机架构

我们将目标定义为使用多层感知机近似函数 $\boldsymbol{f}$： $\boldsymbol{x} \in \mathbb{R}^d \mapsto \boldsymbol{y} \in \mathbb{R}^D$，并将其记作 $\boldsymbol{\mathcal{F}}$。多层感知机的计算单元称为人工神经元，它们依次堆叠在若干连续的层中。$\boldsymbol{\mathcal{F}}$ 的第零层称为输入层，它不是计算层，只负责向网络提供输入（维度为 d）。$\boldsymbol{\mathcal{F}}$ 的最后一层称为输出层，负责输出网络的预测结果（维度为 D）。中间的其他层均称为隐藏层。一层中神经元的数量决定了该层的宽度。图 2.1 展示了一个具有 2 个隐藏层的多层感知机示意图。

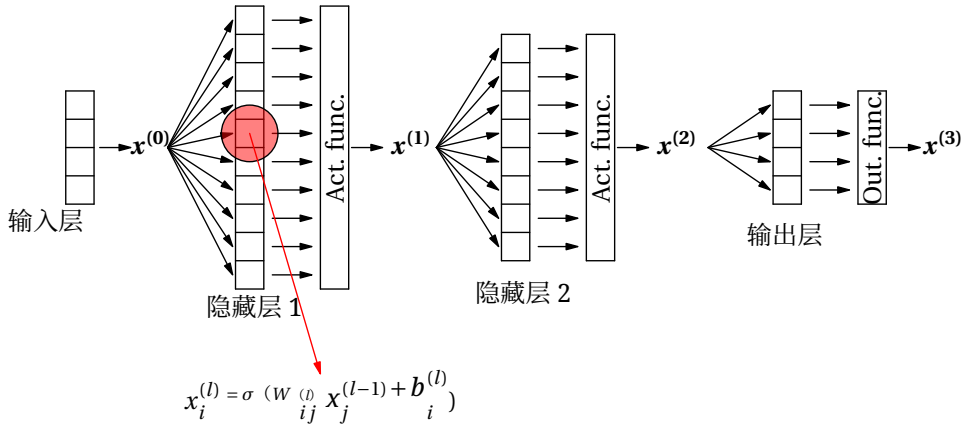


图 2.1：具有 2 个隐藏层的多层感知机

为了理解多层感知机内部发生的运算，我们先定义一些符号说明。考虑一个包含 L 个隐藏层的网络，第 (l) 层的宽度记为 H_l，其中 l =0，1，……，$L+1$。请注意，为了与我们试图近似的函数 f 保持一致，必须满足 $H_0 = d$ 和 $H_{L+1} = D$。我们将第 l 层的输出向量记为 $\boldsymbol{x}^{(l)} \in \mathbb{R}^{H_l}$，它将作为下一层的输入。令 $\boldsymbol{x}^{(0)} = \boldsymbol{x} \in \mathbb{R}^d$ 表示输入层提供的输入信号。在每一层 l、 $1 \leq l \leq L+1$ 中，第 i 个神经元会对该层的输入 L 进行一次

仿射变换，随后进行一次非线性变换

$$x_i^{(l)} = \sigma\Big(\underbrace{W_{ij}^{(l)} x_j^{(l-1)}}_{\text{Einstein sum}} + b_i^{(l)}\Big), \quad 1 \le i \le H_l, \quad 1 \le j \le H_{l-1} \tag{2.1}$$

其中， $W_{ij}^{(l)}$ 和 $b_i^{(l)}$ 分别称为与第 i 个神经元相关联的权重和偏置，该神经元位于第 l 层；函数 $\sigma(.)$ 称为激活函数，在帮助网络表示非线性复杂函数方面发挥着关键作用。若令 $\boldsymbol{W}^{(l)} \in \mathbb{R}^{H_{l-1} \times H_l}$ 为第 l 层的权重矩阵， $\boldsymbol{b}^{(l)} \in \mathbb{R}^{H_l}$ 为第 l 层的偏置向量，则可以将整个层的作用重写为

$$\boldsymbol{x}^{(l)} = \sigma\left(\mathcal{A}^{(l)}(\boldsymbol{x}^{(l-1)})\right), \quad \mathcal{A}^{(l)}(\boldsymbol{x}^{(l-1)}) = \boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{2.2}$$

其中，激活函数是逐分量应用的。因此，整个网络 $\boldsymbol{\mathcal{F}}$： $\mathbb{R}^d \mapsto \mathbb{R}^D$ 的作用在数学上可以看作是仿射变换与逐分量激活交替组合而成的复合函数

$$\boldsymbol{\mathcal{F}}(\boldsymbol{x}) = \mathcal{A}^{(L+1)} \circ \sigma \circ \mathcal{A}^{(L)} \circ \sigma \circ \mathcal{A}^{(L-1)} \circ \cdots \circ \sigma \circ \mathcal{A}^{(1)}(\boldsymbol{x}). \tag{2.3}$$

这里我们提出几点说明:

1. 为了简化表示，我们假设网络的所有层都使用相同的激活函数。不过，这并非严格规定。事实上，近期有证据表明，逐层交替使用激活函数可以构建出更好的神经网络 [33]。

2. 有时，输出层末尾可能使用输出函数 O 而不是激活函数，通常是为了将输出重新表达为合适的形式。我们将在本课程后面看到此类函数的示例。

3. 我们将使用网络的深度 一词来表示多层感知机中的计算层数，即隐藏层和输出层的数量。根据上述符号说明，该数量为 $L+1$ 。

网络的参数是所有权重和偏置，我们将其表示为

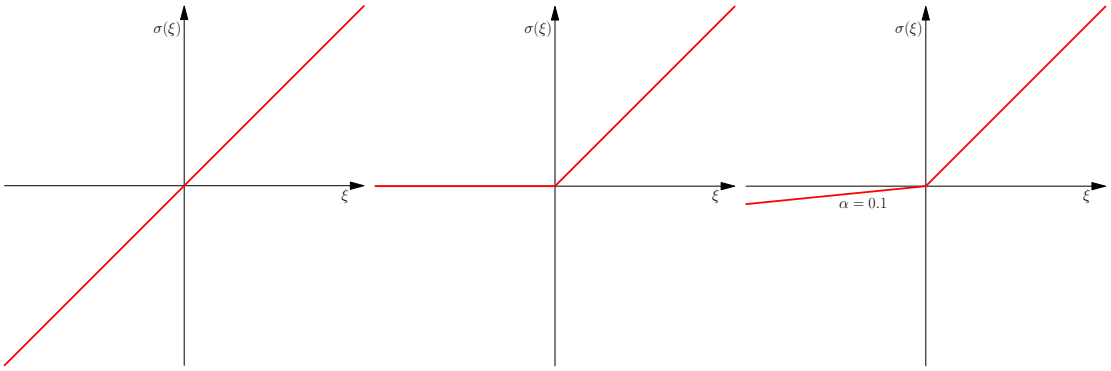
$$\boldsymbol{\theta} = \{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\}_{l=1}^{L+1} \in \mathbb{R}^{N_\theta}$$

其中 N_θ 表示网络的参数总数。网络 $\boldsymbol{\mathcal{F}}(\boldsymbol{x};\ \boldsymbol{\theta})$ 表示一个参数化函数族，其中需要适当地选择 $\boldsymbol{\theta}$ ，使得网络在输入 $\boldsymbol{x}$ 处近似目标函数 $f(\boldsymbol{x})$。

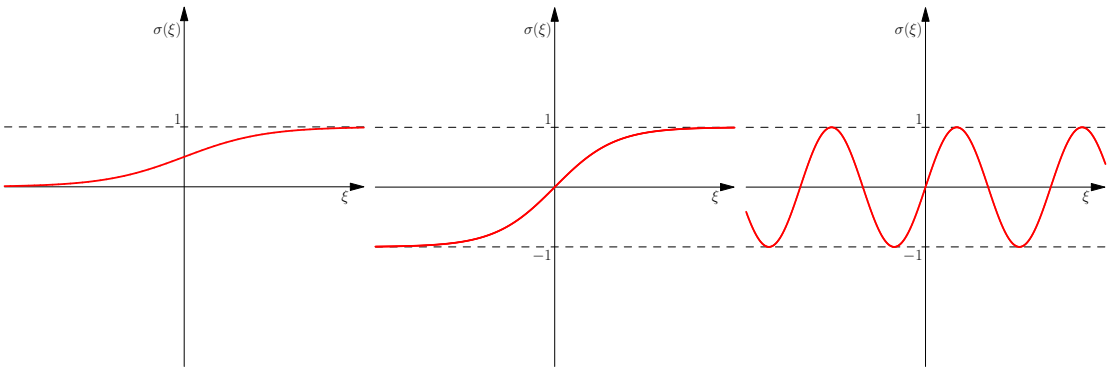
问题 2.1.1. 证明 $N_\theta = \sum_{l=1}^{L+1} (H_{l-1}+1)H_l$.

2.2 激活函数

激活函数可能是多层感知机中最重要的组成部分。文献中有大量可用的激活函数，每一种都有其自身的优点和缺点。让我们来看看其中几种选项（另请参见图 2.2）。



(a) 线性函数　(b) ReLU　(c) 带泄漏的 ReLU



(d) 逻辑函数　(e) 双曲正切函数　(f) 正弦函数

图 2.2：激活函数示例

2.2.1 线性激活

最简单的激活函数对应于 $\sigma(\xi) = \xi$。该函数的一些特征如下：

- 该函数无限光滑，但二阶导数之后的所有导数均为零。
- 该函数的值域为 $(-\infty, \infty)$。
- 在所有层中使用线性激活函数，会使整个网络简化为输入 $\boldsymbol{x}$ 的单个仿射变换。换言之，该网络只不过是目标函数 $\boldsymbol{f}$ 的线性近似；如果 $\boldsymbol{f}$ 是高度非线性的，这种近似就没有用处。

2.2.2 修正线性单元（ReLU）

该函数是分段线性的，定义如下：

$$\sigma(\xi) = \max\{0, \xi\} = \begin{cases} \xi, & \text{if } \xi \geq 0 \\ 0, & \text{if } \xi < 0 \end{cases} \tag{2.4}$$

这是实践中最常用的激活函数之一。该函数具有以下一些特征：

- 该函数是连续的，而其导数是分段常数，并存在跳跃间断$\xi = 0$。二阶导数将是集中在 $\xi = 0$ 处的狄拉克函数。换句话说，高阶导数（阶数大于 1）没有明确的定义。
- 该函数的值域为 $[0, \infty)$。

2.2.3 带泄漏的 ReLU

如果神经元的仿射变换结果为负，ReLU 激活函数会使该神经元的输出为零。这可能导致神经网络训练过程中出现神经元死亡 [15]，即神经元完全退出网络，不再对最终预测作出贡献。为克服这一问题，人们设计了带泄漏版本的 ReLU

$$\sigma(\xi; \alpha) = \begin{cases} \xi, & \text{if } \xi \geq 0 \\ \alpha\xi, & \text{if } \xi < 0 \end{cases} \tag{2.5}$$

其中 α 成为网络的超参数。此函数的一些特征如下：

- 带泄漏的 ReLU 的导数与 ReLU 的导数行为相同。
- 该函数的值域为 $(-\infty, \infty)$。

2.2.4 逻辑函数

逻辑函数或 Sigmoid 激活函数定义为

$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}} \tag{2.6}$$

并具有以下性质

- 该函数具有无限阶光滑性且单调。
- 该函数的值域为 (0，1)，也就是说，该函数有界。这类函数在表示概率时很有用。
- 由于导数在远离 $\xi = 0$ 时会迅速衰减至零，因此训练过程中，这种激活函数可能导致网络收敛缓慢。

2.2.5 双曲正切函数

双曲正切函数可以看作逻辑函数的对称扩展

$$\sigma(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}} \tag{2.7}$$

并具有以下性质

- 该函数处处光滑且单调。
- 该函数的值域为（-1,1），也就是说，该函数有界。注意，它将零输入映射为零，同时将正（负）输入推向 $+1$(-1)。
- 与逻辑函数类似，tanh 的导数远离$\xi = 0$ 时会迅速衰减至零，因此在网络训练过程中可能导致收敛缓慢。

2.2.6 正弦函数

近年来，正弦函数，即 $\sigma(\xi) = \sin(\xi)$，被提出作为一种高效的激活函数[27]。它兼具上述所有激活函数的优点:

- 该函数处处光滑。
- 该函数的值域为 $(-1,1)$，即该函数有界。
- 该函数的所有导数都不会衰减至零。

问题 2.2.1。 你能否设想一种采用正弦函数作为激活函数的多层感知机架构，使其产生与傅里叶级数展开非常相似的近似?

2.3 网络的表达能力

让我们试着理解 N_θ 增加的影响。为此，我们考虑一个使用 ReLU 激活函数的简单例子，即 $\sigma(\xi) = \max\{\xi, 0\}$。我们设定 $d = D = 1, L = 1$，并将参数

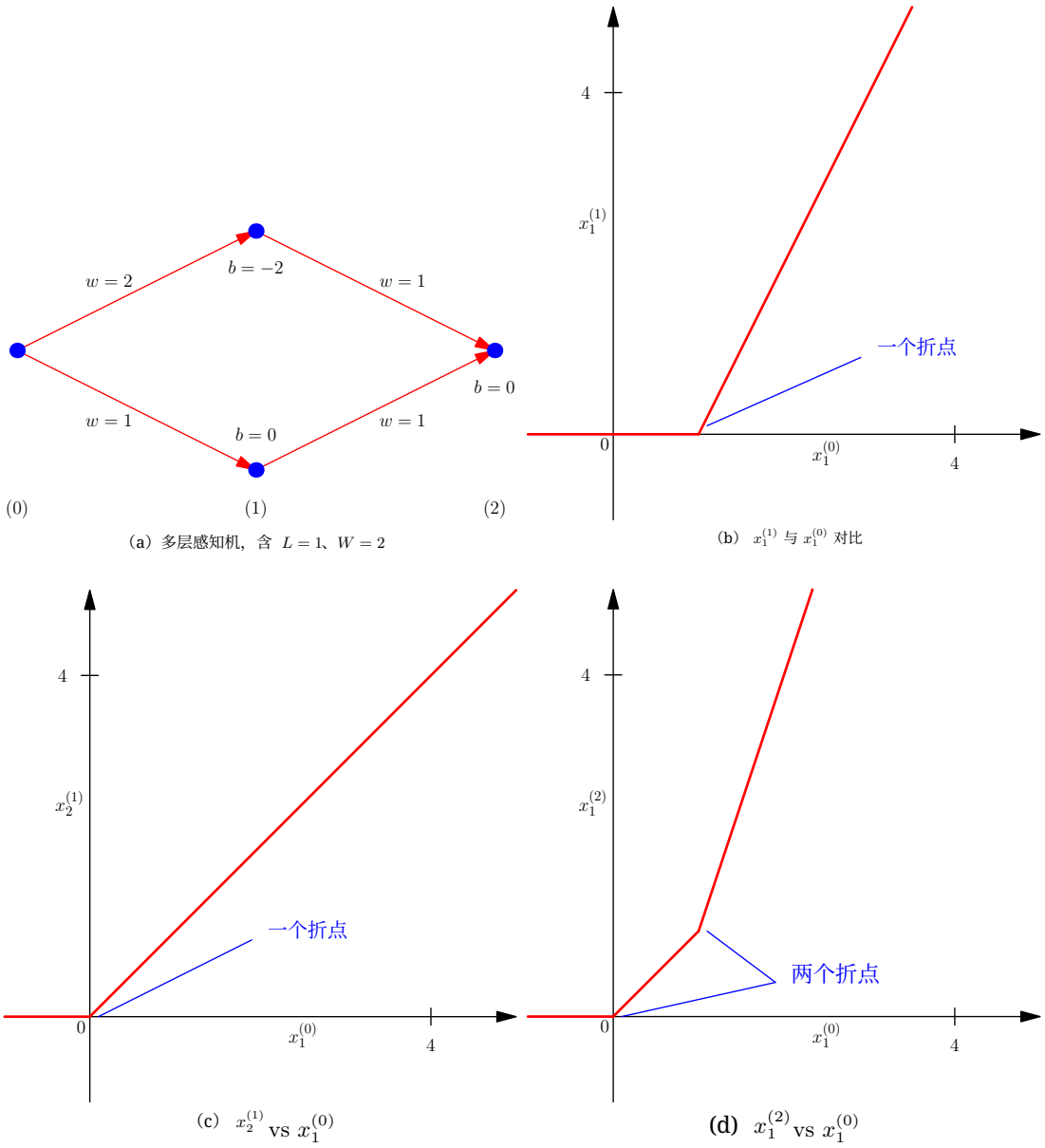
$$\boldsymbol{W}^{(1)} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{b}^{(1)} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}, \quad \boldsymbol{W}^{(2)} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad b^{(2)} = 0.$$

如图 2.3(a) 所示。于是，各层的输出为

$$x_1^{(1)} = \max\{2x_1^{(0)} - 2, 0\}, \quad x_2^{(1)} = \max\{x_1^{(0)}, 0\}, \quad x_1^{(2)} = \max\{2x_1^{(0)} - 2, 0\} + \max\{x_1^{(0)}, 0\}.$$

请注意，虽然隐藏层的输出 $\boldsymbol{x}^{(1)}$（见图 2.3(b) 和 (c)）只有一个拐角/折点，但最终输出却有两个折点（见图 2.3(d)）。

我们将这一表述推广到一个更大的网络，其中包含 L 个隐藏层，每层宽度为 H。于是可以预期，$x_i^{(1)}$, $1 \le i \le H$ 将只有一个折点，其位置和角度取决于隐藏层中每个神经元对应的权重和偏置。向量 $\boldsymbol{x}^{(1)}$ 被传递到下一个隐藏层，在那里，每个神经元会组合这些单个折点，并产生可能具有 H 个折点的输出。同样，第二个隐藏层中每个神经元的输出所含 H 折点的位置和角度也会各不相同。折点的位置之所以不同，是因为每个神经元都可以有不同的偏置，因此能够产生不同的平移。依此类推，可以预期，随着信号经过宽度为 H的各个隐藏层，折点的数量会不断增加，H, H^2, H^3 。一般而言，折点总数可增长至 H^L。换言之，随着网络深度（和宽度）的增加，网络的表达能力也会增强。



(a) 多层感知机，含 $L=1$、$W=2$

(b) $x_1^{(1)}$ 与 $x_1^{(0)}$ 对比

(c) $x_2^{(1)}$ vs $x_1^{(0)}$

(d) $x_1^{(2)}$ vs $x_1^{(0)}$

图 2.3：理解神经网络表达能力的示例

2.3.1 通用逼近结果

为了以数学上严谨的方式量化网络的表达能力，我们来考察一些关于多层感知机逼近性质的结果。对于这些结果，我们假设 $K \subset \mathbb{R}^d$ 是一个闭有界集。

定理 2.3.1（Pinkus, 1999 [23]）。设 f： $K \to \mathbb{R}$，即 $D = 1$，是一个连续函数。则给定一个 $\epsilon > 0$，存在一个具有单个隐藏层的多层感知机（$L = 1$）、任意宽度 H以及非多项式连续激活函数 σ，使得

$$\max_{\boldsymbol{x} \in K} |\mathcal{F}(\boldsymbol{x}; \boldsymbol{\theta}) - f(\boldsymbol{x})| \leq \epsilon.$$

定理 2.3.2（Kidger, 2020 [9]）。设 $\boldsymbol{f}$： $K \to \mathbb{R}^D$ 是一个连续向量值函数。则给定一个 $\epsilon > 0$，存在一个具有任意数量隐藏层 L的多层感知机，每层具有宽度 $H \geq d + D + 2$，并采用连续激活函数 σ（满足一些额外的温和条件），使得

$$\max_{\boldsymbol{x} \in K} \|\boldsymbol{\mathcal{F}}(\boldsymbol{x}; \boldsymbol{\theta}) - \boldsymbol{f}(\boldsymbol{x})\| \leq \epsilon.$$

定理 2.3.3 (Yarotsky, 2021 [33])。设 f： $K \to \mathbb{R}$ 为一个具有两个连续导数的函数，即 $f \in C^2(K)$。考虑一个具有 *ReLU* 激活函数且满足 $H \geq 2d + 10$ 的多层感知机。那么，存在一个具有此配置的网络，使得误差收敛至

$$\max_{\boldsymbol{x} \in K} |\mathcal{F}(\boldsymbol{x}; \boldsymbol{\theta}) - \boldsymbol{f}(\boldsymbol{x})| \leq C(N_\theta)^{-4}$$

其中 C 是一个取决于网络参数数量的常数。

上述数值结果有助于揭开神经网络“黑箱”特性的神秘面纱，并为设计网络架构提供实用指南。

2.4 神经网络的训练、验证与测试

现在我们已经更好地理解了多层感知机的架构，接下来将讨论如何设定这些网络的参数，以逼近某个目标函数。我们的讨论将限定在监督学习框架内。

假设给定一个由成对样本 $\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \text{ 组成的数据集：} 1 \leq i \leq N\}$，这些样本对应于一个目标函数 $\boldsymbol{f} : \boldsymbol{x} \mapsto \boldsymbol{y}$。我们希望使用神经网络来逼近这个函数

$$\mathcal{F}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\Theta})$$

其中 $\boldsymbol{\theta}$ 是前文定义的网络参数，而 $\boldsymbol{\Theta}$ 对应于网络的超参数，例如深度 $L+1$、宽度 H、激活函数类型 σ等。设计稳健网络的策略包括三个步骤：

1. 在训练 阶段，求出 $\boldsymbol{\theta}$ 的最优值（在 $\boldsymbol{\Theta}$固定的情况下）。
2. 在验证 阶段，求出 $\boldsymbol{\Theta}$ 的最优值。
3. 在测试 阶段，使用未见数据测试网络的性能。

为了完成这三项任务，通常首先将数据集 $\mathcal{S}$ 划分为三个不同的部分：包含 N_{train} 个样本的训练集 、包含 N_{val} 个样本的验证集 和包含N_{test} 个样本的测试集 ，其中 $N = N_{\text{train}} + N_{\text{val}} + N_{\text{test}}$。通常，约 60% 的样本用作训练样本，20% 用作验证样本，其余 20% 用于测试。

划分数据集是必要的，因为神经网络是过度参数化的函数。用于对数据建模的大量自由度可能导致数据过拟合。当数据中存在的误差或噪声对网络行为的影响超过底层输入输出关系本身时，就会出现这种情况。因此，一部分数据用于确定 $\boldsymbol{\theta}$，另一部分用于确定超参数 $\boldsymbol{\Theta}$。其余数据则留作测试训练后的网络在未见数据上的表现，即网络泛化 能力的测试。

现在让我们进一步讨论如何在这三个阶段中使用这种划分：

训练： 训练网络时，会使用训练集 $\mathcal{S}_{\text{train}}$ 来解决以下优化问题：求

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \Pi_{\text{train}}(\boldsymbol{\theta}), \quad \text{where} \quad \Pi_{\text{train}}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}} \sum_{\substack{i=1 \\ (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{S}_{\text{train}}}}^{N_{\text{train}}} \|\boldsymbol{y}_i - \boldsymbol{\mathcal{F}}(\boldsymbol{x}_i; \boldsymbol{\theta}, \boldsymbol{\Theta})\|^2$$

对于某个固定的 $\boldsymbol{\Theta}$。最优的 $\boldsymbol{\theta}^*$ 通过适当的基于梯度的算法获得（稍后将讨论）。函数 Π_{train} 称为损失函数。在上面的示例中，我们使用了均方损失函数。稍后我们将考虑其他类型的损失函数。

验证： 验证网络时，会使用验证集 $\mathcal{S}_{\text{val}}$ 来解决以下优化问题：求

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \Pi_{\text{val}}(\boldsymbol{\Theta}), \quad \text{where} \quad \Pi_{\text{val}}(\boldsymbol{\Theta}) = \frac{1}{N_{\text{val}}} \sum_{\substack{i=1 \\ (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{S}_{\text{val}}}}^{N_{\text{val}}} \|\boldsymbol{y}_i - \boldsymbol{\mathcal{F}}(\boldsymbol{x}_i; \boldsymbol{\theta}^*, \boldsymbol{\Theta})\|^2.$$

最优的 $\boldsymbol{\Theta}^*$ 通过随机网格搜索或张量网格搜索等技术获得。

测试： 获得由 $\boldsymbol{\theta}^*$ 和 $\boldsymbol{\Theta}^*$表征的"最佳"网络后，会在测试集 $\mathcal{S}_{\text{test}}$ 上对其进行评估，以估计该网络在前两个阶段未使用过的数据上的性能。

$$\Pi_{\text{test}} = \frac{1}{N_{\text{test}}} \sum_{\substack{i=1 \\ (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{S}_{\text{test}}}}^{N_{\text{test}}} \|\boldsymbol{y}_i - \boldsymbol{\mathcal{F}}(\boldsymbol{x}_i; \boldsymbol{\theta}^*, \boldsymbol{\Theta}^*)\|^2.$$

This 测试误差也称为网络的（近似）泛化误差 。

让我们看一个例子，以便更好地理解如何得到这样的网络。

例 2.4.1. 考虑一个多层感知机，其中除以下可灵活选择的选项外，所有超参数均已固定

$$\sigma \in \{ReLU,\ tanh\}, \quad L \in \{10, 20\}.$$

我们使用以下算法

1. 对每一组可能的 σ, L ：

   (a) 求 $\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}}\, \Pi_{train}(\boldsymbol{\theta})$*(b)*
   用这个 $\boldsymbol{\theta}^*$，计算 $\Pi_{val}(\boldsymbol{\Theta})$

2. 选择使 Π_{val}（$\boldsymbol{\Theta}$） 取值最小的 $\boldsymbol{\Theta}^*$ 。

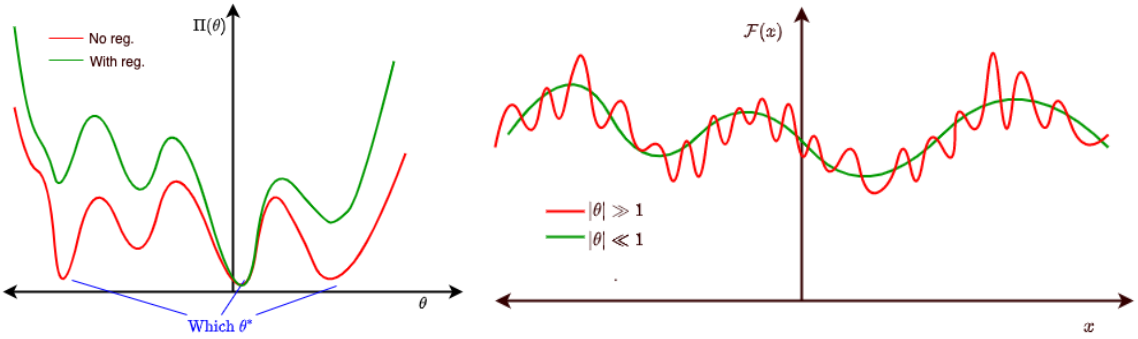
3. 最后，报告 Π_{test} 对于此 $\boldsymbol{\Theta}^*$ 以及相应的 $\boldsymbol{\theta}^*$。

2.5 泛化能力

如果我们训练一个 Π_{train} 和 Π_{val}值都很小的网络，这能确保 Π_{test} 也很小吗？我们通过研究训练后的网络的泛化能力来探讨这个问题，即该网络在训练/验证过程中未见过的数据上表现良好的能力。如果网络经过训练后对训练数据过拟合，通常会导致其在测试数据上的预测效果不佳。通常，如果 $\mathcal{S}_{\text{train}}$、 $\mathcal{S}_{\text{val}}$ 和 $\mathcal{S}_{\text{test}}$ 取自同一数据分布，那么较小的Π_{train}， Π_{val} 可能会使 Π_{test}也较小。接下来，我们来看看一种常用的数据防过拟合技术，称为正则化。

2.5.1 正则化

神经网络，尤其是多层感知机，几乎总是参数过多的，即 $N_\theta \gg N$ 其中 N 是训练样本的数量。这会导致网络模型高度非线性，其损失函数 $\Pi(\boldsymbol{\theta})$（为简洁起见，我们省略下标“train”）的曲面可能包含许多局部极小值（见图 2.4(a)）。那么，我们如何确定哪个极小值能带来更好的泛化能力呢？为了将 $\boldsymbol{\theta}^*$ 的选择引向更有利的方向，可以采用正则化技术。



(a) 损失函数曲面　　(b) 网络敏感度

图 2.4：正则化对损失函数的影响。为便于说明，我们假设 θ 是一个标量。

最简单的正则化方法是在损失函数中增加一个惩罚项：

$$\Pi(\boldsymbol{\theta}) \longrightarrow \Pi(\boldsymbol{\theta}) + \alpha\|\boldsymbol{\theta}\|, \quad \alpha \geq 0$$

其中， α 是正则化超参数， $\|\boldsymbol{\theta}\|$ 是网络参数$\boldsymbol{\theta}$的适当范数。这一增补可以改变 $\Pi(\boldsymbol{\theta})$的地形，如图 2.4(a) 所示。换句话说，这种正则化会促使选择对应于较小参数值 $\boldsymbol{\theta}$的极小值。

较小的 $\boldsymbol{\theta}$ 为何会是更好的选择并不明显。为了理解其原因，请考虑网络中间输出

$$x_1^{(1)} = \sigma(W_{1j}^{(1)} x_j^{(0)} + b_1^{(1)}),$$

由此得到

$$\frac{\partial x_1^{(1)}}{\partial x_1^{(0)}} = \sigma'(W_{1j}^{(1)} x_j^{(0)} + b_1^{(1)}) W_{11}^{(1)} \propto W_{11}^{(1)}.$$

由于这个导数随 $W^{(1)}{}_{11}$ 缩放，这意味着 $|\frac{\partial \boldsymbol{\mathcal{F}}(\boldsymbol{x})}{\partial x_1^{(0)}}|$ 也随 $W^{(1)}{}_{11}$ 缩放。如果$|W_{11}^{(1)}| \gg 1$，那么网络会对输入 $x_1^{(0)}$ 的微小变化也非常敏感，也就是说，该网络将是病态的。如图 2.4(b) 所示，适当的正则化有助于避免过拟合。

下面来看看几种常见的正则化类型：

- l_2 **正则化**：这里我们在正则化项中使用 l_2 范数

$$\|\boldsymbol{\theta}\| = \|\boldsymbol{\theta}\|_2 = \left(\sum_{i=1}^{N_\theta} \theta_i^2\right)^{1/2}.$$

- l_1 **正则化**：这里我们在正则化项中使用 l_1 范数

$$\|\boldsymbol{\theta}\| = \|\boldsymbol{\theta}\|_1 = \sum_{i=1}^{N_\theta} |\theta_i|,$$

这会促进 $\boldsymbol{\theta}$ 的稀疏性。

2.6 梯度下降

回想一下，我们希望在训练阶段求解最小化问题 $\boldsymbol{\theta}^* = \arg\min \Pi(\boldsymbol{\theta})$。这个最小化问题可以通过梯度下降（GD）求解，也称为最速下降。考虑关于 $\boldsymbol{\theta}_0$ 的泰勒展开

$$\Pi(\boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}) = \Pi(\boldsymbol{\theta}_0) + \frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_0) \cdot \Delta\boldsymbol{\theta} + \frac{\partial^2 \Pi}{\partial \theta_i \theta_j}(\hat{\boldsymbol{\theta}}) \Delta\theta_i \Delta\theta_j$$

对于 $\boldsymbol{\theta}_0$ 的一个小邻域内的某个 $\hat{\boldsymbol{\theta}}$。当 $|\Delta\boldsymbol{\theta}|$ 很小且假设 $\frac{\partial^2 \Pi}{\partial \theta_i \theta_j}$ 有界时，我们可以忽略二阶项，只考虑近似

$$\Pi(\boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}) \approx \Pi(\boldsymbol{\theta}_0) + \frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_0) \cdot \Delta\boldsymbol{\theta}.$$

为了尽可能降低损失函数相较于其在 $\boldsymbol{\theta}_0$ 处的取值，即最小化 $\Delta\Pi = \Pi(\boldsymbol{\theta}_0 + \Delta\boldsymbol{\theta}) - \Pi(\boldsymbol{\theta}_0)$，我们需要选择与梯度方向相反的步 $\Delta\boldsymbol{\theta}$，即：

$$\Delta\boldsymbol{\theta} = -\eta \frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_0)$$

其中步长为 $\eta \geq 0$，也称为学习率。这是另一个需要在验证阶段调优的超参数。这就是梯度下降算法的关键，可总结如下：

1. 初始化 $k = 0$ 和 $\boldsymbol{\theta}_0$
2. 当 $|\Pi(\boldsymbol{\theta}_k)| > \epsilon_1$ 时，执行

   (a) 计算 $\frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_k)$ (b) 更新
   $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_k)$ (c) 增加
   $k = k + 1$

收敛： 假设 $\Pi(\boldsymbol{\theta})$ 是凸且可微的，其梯度是利普希茨连续的，利普希茨常数为 $\mathcal{K}$。那么对于一个 $\eta \leq 1/\mathcal{K}$，梯度下降更新的收敛性为

$$\|\boldsymbol{\theta}^* - \boldsymbol{\theta}_k\|_2 \leq \frac{C}{k}.$$

然而，在大多数情况下 $\Pi(\boldsymbol{\theta})$并非凸函数。如果存在多个极小值，那么梯度下降会倾向于选择哪种极小值？为回答这个问题，请考虑图 2.5 所示的标量 θ 的损失函数，它有两个谷底。假设每个谷底中 $\Pi(\theta)$的形状都可以近似为一条（以中心为顶点的）抛物线

$$\Pi(\theta) \approx \frac{1}{2}a\theta^2$$

其中 $a > 0$ 是每个谷底的曲率。注意，左侧谷底的曲率远小于右侧谷底的曲率。我们选取一个固定的学习率 η，并在任一谷底中选取一个起始值 θ_0。那么，

$$\frac{\partial \Pi}{\partial \theta}(\theta_0) = a\theta_0$$

梯度下降更新后的新点将为 $\theta_1 = \theta_0(1 - a\eta)$。同样不难看出，之后所有的迭代点都满足 $\theta_{k+1} = \theta_k(1 - a\eta)$。为了收敛，我们需要

$$\left|\frac{\theta_{k+1}}{\theta_k}\right| < 1 \quad \Longrightarrow \quad |1 - a\eta| < 1.$$

由于谷底中的 $a > 0$，我们需要对学习率施加以下条件

$$-1 < 1 - a\eta \implies a\eta < 2.$$

如果我们固定 η，那么要实现收敛，局部曲率必须满足 $a < 2/\eta$。换句话说，梯度下降更倾向于收敛到曲率较平缓、较小的极小值，也就是说，它会倾向于左侧谷地中的极小值。如果起点位于右侧谷地，我们可能会不断越过右侧极小值，弹向另一侧的谷壁，直到梯度下降算法将我们弹射θ_k 出谷地。之后，它会进入曲率较小的左侧谷地，并逐渐向其中的极小值移动。

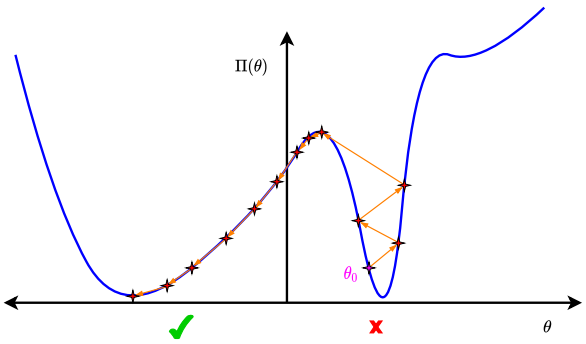


图 2.5：梯度下降偏好更平缓的极小值。

虽然很明显，梯度下降偏好平坦极小值点，但尚不清楚平坦极小值点为何更好。有实验证据表明，在平坦极小值点处获得的参数值往往具有更好的泛化能力，因此更值得选择。

2.7 一些高级优化算法

我们讨论了如何使用梯度下降来解决训练神经网络时涉及的优化问题。接下来，我们来看看几种受梯度下降启发的先进且流行的优化技术。

一般来说，大多数优化算法的更新公式都会用到以下公式

$$[\boldsymbol{\theta}_{k+1}]_i = [\boldsymbol{\theta}_k]_i - [\boldsymbol{\eta}_k]_i [\boldsymbol{g}_k]_i, \quad 1 \leq i \leq N_\theta, \tag{2.8}$$

其中，$[\boldsymbol{\eta}_k]_i$ 是分量学习率，而向量值函数 $\boldsymbol{g}$ 依赖于梯度或对梯度进行近似。注意，记号 $[.]_i$ 用于表示向量的第 i个分量。还要注意，学习率可以依赖于迭代次数 k。梯度下降法使用的是

$$[\boldsymbol{\eta}_k]_i = \eta, \quad \boldsymbol{g}_k = \frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_k).$$

梯度下降法的一个问题是，如果 η 选择不当，收敛到极小值的速度可能会非常慢。例如，考虑图 2.6 所示的目标函数地形，其沿 $[\theta]_2$ 方向的梯度比沿 $[\theta]_1$ 方向的梯度更陡。如果我们从图中所示的某个点出发，那么当 η 过大（但仍处于稳定边界内）时，更新过程将不断以之字形朝极小值前进。理想情况下，对于图 2.6 所示的情形，我们希望沿$[\theta]_1$ 方向采取较大的步幅，相比之下沿 $[\theta]_2$ 方向的步幅较小，从而更快地到达极小值。

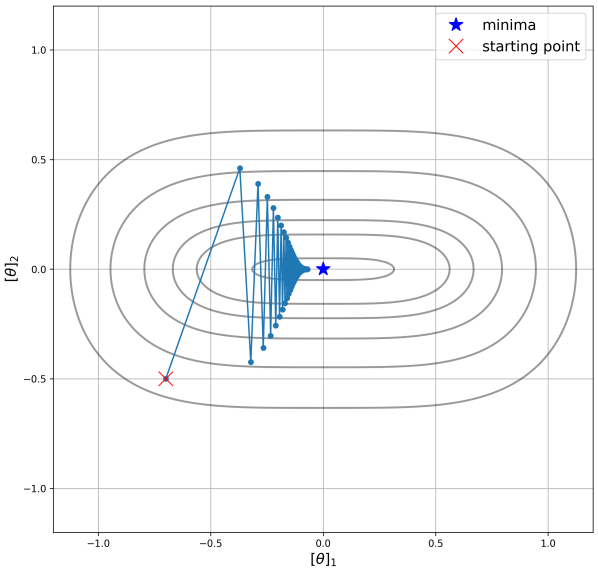


图 2.6：使用梯度下降的之字形更新。

下面介绍两种流行的方法，它们能够克服梯度下降所面临的一些问题。

2.7.1 动量法

动量法利用梯度的历史信息，而不只是前一步的梯度。更新公式如下：

$$[\boldsymbol{\eta}_k]_i = \eta, \quad \boldsymbol{g}_k = \beta_1 \boldsymbol{g}_{k-1} + (1-\beta_1)\frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_k), \quad \boldsymbol{g}_{-1} = 0$$

其中， $\boldsymbol{g}_k$ 是梯度的加权移动平均。通过抵消梯度沿 $[\theta]_2$方向的分量，这种加权预计能够平滑图 2.6 中的之字形轨迹，使参数更平稳地趋向极小值。 β_1 的常用取值为 0.9。

2.7.2 Adam

Adam 优化算法由 Kingma 和 Ba [10], 提出，它利用梯度的历史信息以及梯度的二阶矩（用于衡量梯度的模）。对于初始学习率 η，更新公式为

$$\begin{aligned}\boldsymbol{g}_k &= \beta_1 \boldsymbol{g}_{k-1} + (1-\beta_1)\frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_k)\\ [\boldsymbol{G}_k]_i &= \beta_2 [\boldsymbol{G}_{k-1}]_i + (1-\beta_2)\left(\frac{\partial \Pi}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_k)\right)^2\\ [\boldsymbol{\eta}_k]_i &= \frac{\eta}{\sqrt{[\boldsymbol{G}_k]_i + \epsilon}}\end{aligned} \tag{2.9}$$

其中， $\boldsymbol{g}_k$ 和 $\boldsymbol{G}_k$ 分别是梯度及其平方的加权移动均值。超参数的推荐取值为 $\beta_1 = 0.9$、$\beta_2 = 0.999$ 和$\epsilon = 10^{-8}$。请注意，每个分量的学习率各不相同。具体来说，某个分量的梯度模越大，其学习率就越小。回到图 2.6 中的例子，这意味着 θ_2 的学习率会低于 θ_1，因此有助于缓解优化算法的锯齿形路径。

备注 2.7.1. *Adam* 算法还会对 $\boldsymbol{g}_k$ 和 $\boldsymbol{G}_k$ 执行额外的校正步骤，以提高算法的效率。详情请见 *[10]*。

2.7.3 随机优化

注意到，训练损失可以改写为

$$\Pi(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}} \Pi_i(\boldsymbol{\theta}), \quad \Pi_i(\boldsymbol{\theta}) = \|\boldsymbol{y}_i - \boldsymbol{\mathcal{F}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\Theta})\|^2$$

因此，损失函数的梯度为

$$\frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}} \frac{\partial \Pi_i}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta})$$

然而，对梯度求和的代价可能非常高，因为 N_{train} 通常非常大， $N_{\text{train}} \sim 10^6$。解决这一问题的一个简单方法是使用以下更新公式（此处以梯度下降法为例）

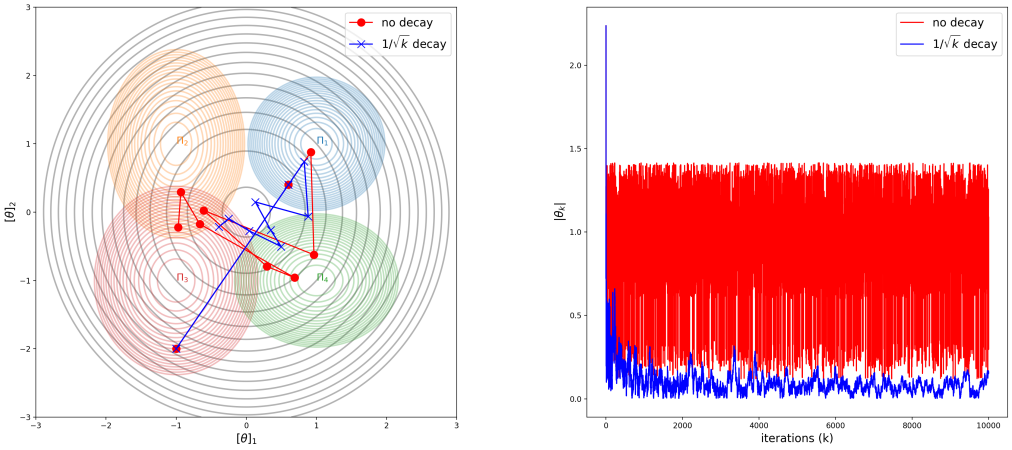
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \frac{\partial \Pi_i}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}_k), \tag{2.10}$$

其中，每次更新步骤 k 都会随机选择 i 。这称为随机梯度下降。值得注意的是，假设 $\Pi_i(\boldsymbol{\theta})$ 是凸且可微的，并且 $\eta_k \sim 1/\sqrt{k}$ [19]，这种改进后的算法确实会收敛。为说明 η_k 为什么需要逐渐减小，考虑 $\boldsymbol{\theta} \in \mathbb{R}^2$ 的玩具函数

$$\begin{aligned}&\Pi_1(\boldsymbol{\theta}) = ([\theta]_1 - 1)^2 + ([\theta]_2 - 1)^2, \quad \Pi_2(\boldsymbol{\theta}) = ([\theta]_1 + 1)^2 + 0.5([\theta]_2 - 1)^2,\\ &\Pi_3(\boldsymbol{\theta}) = 0.7([\theta]_1 + 1)^2 + 0.5([\theta]_2 + 1)^2, \quad \Pi_4(\boldsymbol{\theta}) = 0.7([\theta]_1 - 1)^2 + \frac{1}{2}([\theta]_2 + 1)^2,\\ &\Pi(\boldsymbol{\theta}) = \frac{1}{4}\left(\Pi_1(\boldsymbol{\theta}) + \Pi_2(\boldsymbol{\theta}) + \Pi_3(\boldsymbol{\theta}) + \Pi_4(\boldsymbol{\theta})\right).\end{aligned} \tag{2.11}$$

这些函数的等高线图如图 2.7(a) 所示，其中黑色等高线对应于 $\Pi(\boldsymbol{\theta})$。注意， $\boldsymbol{\theta}^* = (0,0)$ 是 $\Pi(\boldsymbol{\theta})$ 的唯一极小值。我们考虑使用随机梯度下降算法求解，分别采用恒定学习率 $\eta_k = 0.4$ 和递减学习率 $\eta_k = 0.4/\sqrt{k}$。从 $\boldsymbol{\theta}_0 = (-1.0, 2.0)$ 开始，并在每一步随机选择 $i \in 1, 2, 3, 4$，我们运行算法 10,000 次迭代。图 2.7(a) 绘出了采用每种学习率时的前 10 步。显然，如果学习率没有衰减，随机梯度下降算法会不断越过极小值。事实上，这种情况会持续到后续所有迭代，正如图 2.7(b) 所示，更新的范数并未衰减（我们预期它会衰减至 $|\boldsymbol{\theta}^*| = 0$）。另一方面，如果学习率按 $1/\sqrt{k}$ 递减，我们就会迅速接近 $\boldsymbol{\theta}^*$。

随着我们逐渐接近极小值而减小步长的原因是：当我们远离 Π 的极小值时， Π 的梯度向量与所有单独的 Π_i 都相当一致。然而，当我们逐渐接近 Π 的极小值时，情况并非如此，因此必须采取更小的步长，以免被甩到远离极小值的区域。



(a) 函数等高线与路径　　(b) 更新量的范数

图 2.7：学习率衰减与不衰减时的随机梯度下降算法。

I在实践中，随机优化算法不会出于以下原因而被使用 :

1. 尽管损失函数会随着迭代次数增加而下降，但在极小值附近会以混乱的方式波动，始终无法达到极小值。

2. 一次处理所有样本可能会带来高昂的计算开销，而一次只处理一个样本则会严重浪费计算和内存资源。

不过，可以采用小批量优化来折中。在这种策略中，包含 N_{train} 个样本的数据集被划分为 N_{batch} 个互不相交的子集，称为小批量。每个小批量包含 $N_{\text{train}} = N_{\text{train}}/N_{\text{batch}}$ 个样本，这个数量也称为批次大小。因此，损失函数的梯度可以近似为

$$\frac{\partial \Pi}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \frac{\partial \Pi_i}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}) \approx \frac{1}{\overline{N}_{\text{train}}} \sum_{i \in \text{batch}(j)} \frac{\partial \Pi_i}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}). \tag{2.12}$$

请注意，取 $N_{\text{batch}}=1$ 会得到原始的优化算法，而取 $N_{\text{batch}}=N_{\text{train}}$ 则会得到随机梯度下降算法。通常会选择一个批次大小，使得每次能够加载到随机存取存储器中的数据量最大。我们将遍历完整训练集中的所有样本（或小批量）一次定义为一个训练轮次。下面介绍小批量随机优化算法：

1. 对于训练轮次 $=1,\ \ldots\ldots,\ \mathrm{J}$

   (a) 随机打乱整个训练集 (b) 创建 N_{batch} 个小批量 (c) 对于 $i=1,\ \cdots,\ N_{\text{batch}}$i. 使用 (2.12) 计算该批次的梯度。 ii. 使用此梯度和你偏好的优化算法（梯度下降、动量或 Adam）更新 $\boldsymbol{\theta}$ 。

注释 2.7.2。 有一项有趣的研究 *[31]* 表明，随机梯度下降实际上可能有助于选择泛化能力更好的极小值。在该研究中，作者证明，随机梯度下降倾向于选择曲率更均匀的极小值。也就是说，损失函数各分量的曲率分布较为集中，并以一个较小的值为中心。这与整体曲率可能较小、但损失函数各分量的曲率分布更分散的极小值形成对比。随后，他们还（通过实证）表明，相较于异质极小值，同质性更高的极小值往往具有更好的泛化能力。

2.8 使用反向传播计算梯度

我们需要理解的训练算法的最后一部分，是在训练网络时如何实际计算梯度。回想一下，层 $l+1$ 的输出 $\boldsymbol{x}^{(l+1)}$ 定义为

$$\text{Affine transform:}\quad \xi_i^{(l+1)}=W_{ij}^{(l+1)}x_j^{(l)}+b_i^{(l+1)},\quad 1\le i\le H_{l+1} \tag{2.13}$$

$$\text{Non-linear transform:}\quad x_i^{(l+1)}=\sigma\left(\xi_i^{(l+1)}\right),\quad 1\le i\le H_{l+1}. \tag{2.14}$$

给定一个训练样本（$\boldsymbol{x}$，$\boldsymbol{y}$），令 $\boldsymbol{x}^{(0)}=\boldsymbol{x}$。可以通过前向传播计算（针对这个特定样本的）损失函数/目标函数的值：

1. 对于 $l=1,\ \ldots,\ L+1$

   (a) 使用 (2.13) 计算 $\xi^{(l)}$ 。
   (b) 使用 (2.14) 计算 $\boldsymbol{x}^{(l)}$ 。

2. 计算给定样本的损失函数

$$\Pi(\boldsymbol{\theta})=\|\boldsymbol{y}-\boldsymbol{\mathcal{F}}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\Theta})\|^2.$$

这一运算可以简洁地表示为图 2.8 所示的计算图。在该图中，计算图的下半部分表示损失函数的计算 Π。

当然，我们需要对训练集中的所有样本（或随机优化中的一个小批量）重复这一步。为简单起见，我们将讨论限定为单个样本的损失及其梯度的计算。

为了更新网络参数，我们需要 $\frac{\partial \Pi}{\partial \boldsymbol{\theta}}$，更准确地说是 $\frac{\partial \Pi}{\partial \boldsymbol{W}^{(l)}}$， $\frac{\partial \Pi}{\partial \boldsymbol{b}^{(l)}}$ 对$1 \leq l \leq L+1$。~~我们~~将先推导 $\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}}$ 和 $\frac{\partial \Pi}{\partial \boldsymbol{x}^{(l)}}$ 的表达式，再据此推导这些导数的表达式。

从计算图中可以很容易看出，网络中的每个隐藏变量是如何转换为下一个变量的。认识到这一点并反复应用链式法则，可得

$$\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} = \frac{\partial \Pi}{\partial \boldsymbol{x}^{(L+1)}} \cdot \frac{\partial \boldsymbol{x}^{(L+1)}}{\partial \boldsymbol{\xi}^{(L+1)}} \cdot \frac{\partial \boldsymbol{\xi}^{(L+1)}}{\partial \boldsymbol{x}^{(L)}} \cdots \frac{\partial \boldsymbol{x}^{(l+1)}}{\partial \boldsymbol{\xi}^{(l+1)}} \cdot \frac{\partial \boldsymbol{\xi}^{(l+1)}}{\partial \boldsymbol{x}^{(l)}} \cdot \frac{\partial \boldsymbol{x}^{(l)}}{\partial \boldsymbol{\xi}^{(l)}}. \tag{2.15}$$

为了计算这个表达式，我们需要计算以下各项：

$$\frac{\partial \Pi}{\partial \boldsymbol{x}^{(L+1)}} = -2(\boldsymbol{y} - \boldsymbol{x}^{(L+1)})^T \tag{2.16}$$

$$\frac{\partial \boldsymbol{\xi}^{(l+1)}}{\partial \boldsymbol{x}^{(l)}} = \boldsymbol{W}^{(l+1)} \tag{2.17}$$

$$\frac{\partial \boldsymbol{x}^{(l)}}{\partial \boldsymbol{\xi}^{(l)}} = \boldsymbol{S}^{(l)} \equiv \mathrm{diag}[\sigma'(\xi_1^{(l)}), \cdots, \sigma'(\xi_{H_l}^{(l)})], \tag{2.18}$$

其中，最后两个关系对任意网络层都成立； l， H_l 是相应层的宽度， σ' 表示激活函数对其自变量的导数。将这些关系代入 (2.15)，可得：

$$\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} = \frac{\partial \Pi}{\partial \boldsymbol{x}^{(L+1)}} \cdot \boldsymbol{S}^{(L+1)} \cdot \boldsymbol{W}^{(L+1)} \cdots \boldsymbol{S}^{(l+1)} \cdot \boldsymbol{W}^{(l+1)} \cdot \boldsymbol{S}^{(l)}. \tag{2.19}$$

取转置，并注意到 $\boldsymbol{\Sigma}^{(l)}$ 是对角矩阵，因此也是对称矩阵，最终可得

$$\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} = \boldsymbol{S}^{(l)} \boldsymbol{W}^{(l+1)T} \boldsymbol{S}^{(l+1)} \cdots \boldsymbol{W}^{(L+1)T} \boldsymbol{S}^{(L+1)} [-2(\boldsymbol{y} - \boldsymbol{x}^{(L+1)})]. \tag{2.20}$$

这一计算过程也可以用计算图表示。事实上，如图 2.8 所示，可以将其附加到原始计算图上，其中这部分计算显示在计算图的上方一行。请注意，我们现在是沿着反向遍历的，因此称为反向传播。

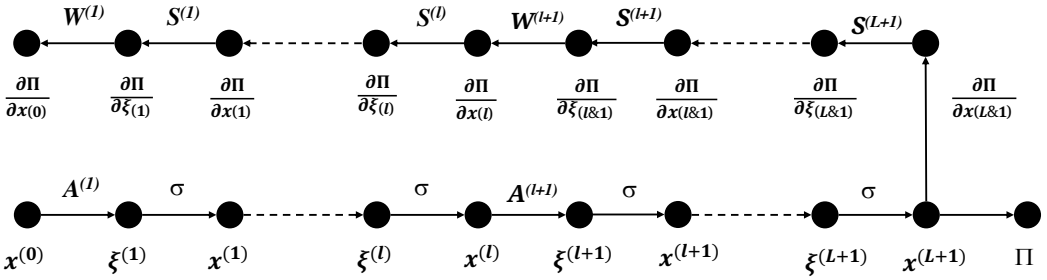


图 2.8：用于计算损失函数及其相对于隐藏／潜在向量的导数的计算图。

最后一步是求出 $\frac{\partial \Pi}{\partial \boldsymbol{W}^{(l)}}$ 的显式表达式。这可以通过识别出以下关系来完成：

$$\frac{\partial \Pi}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} \cdot \frac{\partial \boldsymbol{\xi}^{(l)}}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} \otimes \boldsymbol{x}^{(l-1)}, \tag{2.21}$$

其中， $[\boldsymbol{x}\otimes\boldsymbol{y}]_{ij}=x_iy_j$ 是外积。因此，为了求出 $\frac{\partial\Pi}{\partial\boldsymbol{W}^{(l)}}$，我们需要 $\boldsymbol{x}^{(l-1)}$（它在前向阶段求得）以及 $\frac{\partial\Pi}{\partial\boldsymbol{\xi}^{(l)}}$（它在反向传播过程中求得）。——

问题 2.8.1。 你能推导出一组类似的表达式以及相应的算法来计算 $\frac{\partial\Pi}{\partial\boldsymbol{b}^{(l)}}$ 吗？

——

问题 2.8.2。 你能推导出 $\frac{\partial\boldsymbol{x}^{(L+1)}}{\partial\boldsymbol{x}^{(0)}}$ 的显式表达式吗？也就是说，能否写出网络输出相对于其输入的导数表达式？这是一个非常有用的量，可用于物理信息神经网络和 *Wasserstein*生成对抗网络等算法。

2.9 回归与分类

到目前为止，给定带标签数据集 $\mathcal{S}=\{(\boldsymbol{x}_i,\ \boldsymbol{y}_i):\ 1\le i\le N\}$，我们考虑了两种类型的损失

- 均方误差（MSE）

$$\Pi(\boldsymbol{\theta})=\frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}}\|\boldsymbol{y}_i-\boldsymbol{\mathcal{F}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\Theta})\|^2.$$

- 平均绝对误差（MAE）

$$\Pi(\boldsymbol{\theta})=\frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}}\|\boldsymbol{y}_i-\boldsymbol{\mathcal{F}}(\boldsymbol{x}_i;\boldsymbol{\theta},\boldsymbol{\Theta})\|.$$

具有上述损失函数的神经网络可用于解决各种回归问题，这些问题中的底层函数高度非线性，且输入/输出是多维的。

例 2.9.1. 给定房屋/公寓的特征，例如邮政编码、卧室/浴室数量、地毯面积、建筑年限等，预测市场售价或房屋在市场上挂牌的天数等结果。

现在让我们来看一些分类问题的例子，在这些问题中，网络的输出通常属于一个有限的离散集合。

例 2.9.2. 给定*2019*冠状病毒病患者的症状和血液标志物，预测他们是否需要入住重症监护室。因此，这个问题的输入和输出为

$$\boldsymbol{x}=[pulse\ rate, temperature, SPO_2, procalcitonin, ...]$$
$$\boldsymbol{y}=[p_1,p_2]$$

其中， p_1 表示入住重症监护室的概率，而 p_2 表示不入住的概率。注意， $0\le p_1$、 $p_2\le 1$ 和 $p_1+p_2=1$。

例 2.9.3. 给定一组动物图像，预测动物是狗、猫还是鸟。在这种情况下，输入和输出应为

$$\boldsymbol{x}=the\ image$$
$$\boldsymbol{y}=[p_1,p_2,p_2]$$

其中， p_1、 p_2、 p_3 分别表示属于狗、猫或鸟的概率。

由于分类问题的输出对应的是概率，因此我们需要对网络做一些调整

1. 在输出层末端使用输出函数，将输出向量适当地转换为所需形式，即概率向量。通常使用 *Softmax* 函数 来完成这一转换

$$x_i^{(L+1)} = \frac{\exp\left(\xi_i^{(L+1)}\right)}{\sum_{j=1}^{C} \exp\left(\xi_j^{(L+1)}\right)}$$

其中 C 是类别的数量（也是输出维度）。验证经过此转换后， $\boldsymbol{x}^{(L+1)}$ 的各分量构成凸组合，即 $x_i^{(L+1)} \in [0,1]$和 $\sum_{i=1}^{C} x_i^{(L+1)} = 1$。

2. 各个样本的输出标签需要进行独热编码。换言之，对于样本（$\boldsymbol{x}$， $\boldsymbol{y}$），输出标签 $\boldsymbol{y}$ 的维度应为 $D = C$，并且只有表示该样本所属类别 $\boldsymbol{x}$ 的分量为 1，其余分量均为 0。例如，在例 2.9.3 中

$$\boldsymbol{y} = \begin{cases} [1,0,0]^\top & \text{if } \boldsymbol{x} \text{ is a dog,} \\ [0,1,0]^\top & \text{if } \boldsymbol{x} \text{ is a cat,} \\ [0,0,1]^\top & \text{if } \boldsymbol{x} \text{ is a pig.} \end{cases}$$

3. 虽然仍可使用均方误差或均方绝对误差作为损失函数，但最好使用交叉熵损失函数

$$\Pi(\boldsymbol{\theta}) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \sum_{c=1}^{C} -y_{ci} \log(\mathcal{F}_c(\boldsymbol{x}_i; \boldsymbol{\theta})), \tag{2.22}$$

其中 y_{ci} 是第 i个样本真实标签的第 c个分量。式（2.22）中的损失函数将 y_c 和 $\mathcal{F}_c$ 视为概率分布，并衡量二者之间的差异。可以证明，它与这两个分布之间的 Kullback-Liebler 散度有关。与均方误差相比，这个损失函数会对置信度很高但预测错误的结果施加严厉惩罚。例 2.9.4 展示了这一点

例 2.9.4. 考虑一个二分类问题，即 $C = 2$。对于给定的 $\boldsymbol{x}$，令 $\boldsymbol{y} = [0,1]$，并令预测为$\boldsymbol{\mathcal{F}} = [p, 1-p]$。显然， p 的值越小越好。因此，任何合理的代价函数都应对 p的较大值施加惩罚。现在，让我们用各种损失函数来评估误差

- 均方误差损失 $= (0-p)^2 + (1-1+p)^2 = 2p^2$。
- 交叉熵损失 $= -(0\log(p) + 1\log(1-p) = -log(1-p)$.

请注意，这两种损失都会惩罚较大的 p。此外，当 $p = 0$ 时，两种损失都为零。然而，当 $p \to 1$ （这会导致错误的预测）时，均方误差损失 $\to 2$，而交叉熵损失 $\to \infty$。也就是说，它会强烈惩罚错误且置信度高的预测。

第3章

残差神经网络

残差网络（或 ResNet）由何等人于 [8] 在 2015年提出。本章将讨论这些网络是什么、它们为何被提出，以及它们与常微分方程的关系。

3.1 深度网络中的梯度消失

训练神经网络时，梯度 $\frac{\partial \Pi}{\partial \boldsymbol{W}^{(l)}}$，$\frac{\partial \Pi}{\partial \boldsymbol{b}^{(l)}}$ 可能会变得非常小。例如，考虑一个非常深的网络，比如 $L \geq 20$。如果 $\left|\frac{\partial \Pi}{\partial \boldsymbol{W}^{(l)}}\right| \ll 1$ 对于 $l \leq \bar{l}$ 成立，那么网络最初的 l 层的贡献可以忽略不计，因为它们的权重对损失函数的影响很小。由于这种深度截断，深度网络在表达能力方面的优势便不复存在。

那么，为什么会发生这种情况？回顾第2.8节，

$$\frac{\partial \Pi}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} \otimes \boldsymbol{x}^{(l-1)}$$

and

$$\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} = \boldsymbol{\Sigma}^{(l)} \prod_{m=l+1}^{L+1} (\boldsymbol{W}^{(m)T} \boldsymbol{\Sigma}^{(m)}) \frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(L+1)}}. \tag{3.1}$$

对于任意矩阵 $\boldsymbol{A}$，令 $\tau(\boldsymbol{A})$ 表示最大奇异值。那么我们可以将 $|\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}}|$ 界定为

$$|\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}}| \leq \tau(\boldsymbol{\Sigma}^{(l)}) \prod_{m=l+1}^{L+1} (\tau(\boldsymbol{W}^{(m)}) \tau(\boldsymbol{\Sigma}^{(m)}))| \frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(L+1)}}|. \tag{3.2}$$

回想一下，$\boldsymbol{\Sigma}^{(m)} \equiv \text{diag}[\sigma'(\xi_1^{(m)}) \cdots, \sigma'(\xi_{H_l}^{(m)})]$, 其中 σ' 表示 σ 对其自变量的导数。对于 ReLU，其值为 0 或 1。因此 $\tau(\boldsymbol{\Sigma}^{(m)}) = 1$。

此外，为了保证稳定性，我们希望 $\tau(\boldsymbol{W}_{(m)}) < 1$。否则，网络的输出可能会变得无界。实践中，这通过正则化项来实现。

将其代入上面的方程，我们得到

$$|\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}}| \leq \prod_{m=l+1}^{L+1} (\tau(\boldsymbol{W}^{(m)}))| \frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(L+1)}}|, \tag{3.3}$$

乘积中的每一项都是小于 1 的标量。随着项数增加，也就是说 $L - l \gg 1$，这个乘积可能会变得非常小，而且确实会如此。这种情况通常发生在

$L-l\approx 20$，此时 $|\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}}|$，因此 $|\frac{\partial \Pi}{\partial \boldsymbol{W}^{(l)}}|$ 也会变得非常小。这个问题称为梯度消失问题。它会出现在深度网络中，此时内部层中的权重（例如 $L-l>20$）对网络没有贡献。

在 [8], 中，作者展示了更深的网络实际上可能导致训练误差和验证误差增加（见图 3.1）。因此，超过某个临界点后，增加网络的深度可能适得其反。根据我们先前关于梯度消失的讨论，我们知道了其中的原因。鉴于此，我们希望设计一种网络架构，通过确保 $\left|\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(L+1)}}\right| \approx \left|\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(1)}}\right|$ 来解决梯度消失问题。这意味着，当网络的权重趋近于较小的值时，网络应趋近于恒等映射，而不是零映射。这正是残差网络架构的核心思想。

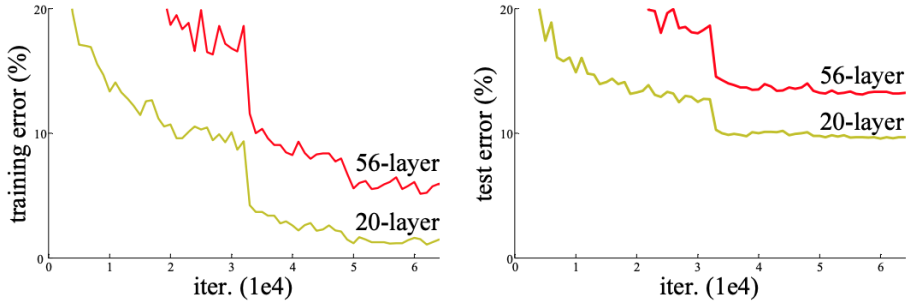


图 3.1：使用“普通”深度网络在 CIFAR-10 数据上的训练误差（左）和测试误差（右）（摘自 [8]）。

3.2 残差网络

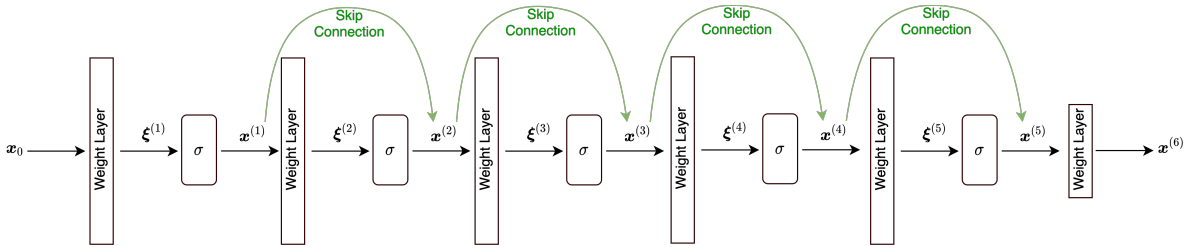


图 3.2：带有跳跃连接、深度为 6 的残差网络。

考虑一个深度为 6 的多层感知机（如图 3.2 所示），其每个隐藏层的宽度均固定为 H。我们按以下方式在隐藏层之间添加跳跃连接

$$x_i^{(l)} = \sigma(W_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)}) + x_i^{(l-1)}, \quad 2 \le l \le L. \tag{3.4}$$

我们可以得出以下观察结果：

1. 如果所有权重（以及偏置）都为零，那么 $\boldsymbol{x}^{(5)} = \boldsymbol{x}^{(1)}$，进而将会

$$\frac{\partial \Pi}{\partial \boldsymbol{x}^{(1)}} = \frac{\partial \Pi}{\partial \boldsymbol{x}^{(5)}},$$

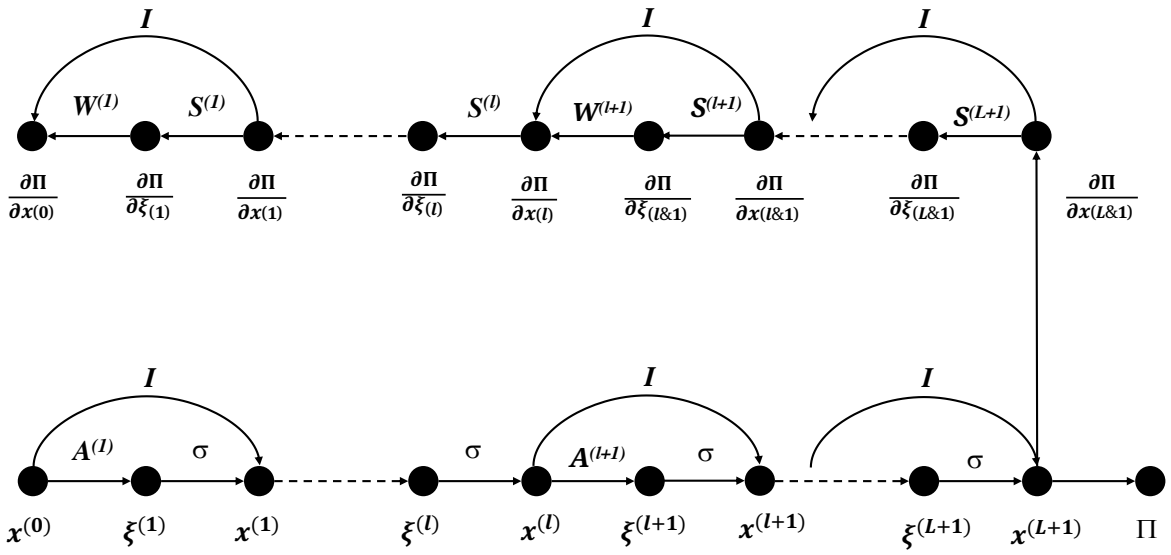


图 3.3：残差网络中前向传播与反向传播的计算图。

也就是说，我们不会遇到梯度消失问题。

2. 图 3.3 展示了残差网络的前向传播与反向传播计算图。从图中可以看出，$\frac{\partial \boldsymbol{x}^{(l+1)}}{\partial \boldsymbol{x}^{(l)}}$ 的表达式现在涉及遍历两个分支并将它们相加。因此，我们有

$$\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} = \boldsymbol{\Sigma}^{(l)} \prod_{m=l+1}^{L+1} (\boldsymbol{I} + \boldsymbol{W}^{(m)T} \boldsymbol{\Sigma}^{(m)}) \frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(L+1)}}. \tag{3.5}$$

现在，假设通过正则化 $|\boldsymbol{W}^{(m)}| \ll 1$，我们有

$$\frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(l)}} = \boldsymbol{\Sigma}^{(l)} \big(\boldsymbol{I} + \sum_{m=l+1}^{L+1} \boldsymbol{W}^{(m)\ T} \boldsymbol{\Sigma}^{(m)} + \text{ higher order terms} \big) \frac{\partial \Pi}{\partial \boldsymbol{\xi}^{(L+1)}}. \tag{3.6}$$

在上面的表达式中，即使各个矩阵的元素都很小，它们的和也未必趋近于零矩阵。这意味着，即使要求权重较小（通过正则化实现），我们仍可以使输入层和输出层附近的梯度产生有限且显著的变化。

注释 3.2.1. 上述分析可以扩展到 H 并非固定的情况，但分析过程就没有那么简洁了。参见 *[8]*，了解如何做到这一点。

3.3 与常微分方程的联系

我们首先考虑带有 $d = D = H$ 的残差网络的特殊情况。回想关系式 (3.4)，我们可以将其改写为

$$\frac{\boldsymbol{x}^{(l)} - \boldsymbol{x}^{(l-1)}}{\Delta t} = \frac{1}{\Delta t} \sigma(\boldsymbol{W}^{(l)} \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}) = \frac{1}{\Delta t} \sigma(\boldsymbol{\xi}^{(l)}) \tag{3.7}$$

其中某个标量为 Δt。请注意，$\boldsymbol{\xi}^{(l)}$ 是一个由 $\boldsymbol{\theta}^{(l)}=[\boldsymbol{W}^{(l)},\boldsymbol{b}^{(l)}]$ 参数化的 $\boldsymbol{x}^{(l-1)}$ 函数。因此，我们可以进一步将 (3.7) 改写为

$$\frac{\boldsymbol{x}^{(l)}-\boldsymbol{x}^{(l-1)}}{\Delta t}=\boldsymbol{V}(\boldsymbol{x}^{(l-1)};\boldsymbol{\theta}^{(l)}). \tag{3.8}$$

现在考虑一个一阶系统的（可能为非线性的）常微分方程，其中给定 $\boldsymbol{x}(0)$ 和

$$\dot{\boldsymbol{x}}\equiv\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}=\boldsymbol{V}(\boldsymbol{x},t) \tag{3.9}$$

我们希望求得 $\boldsymbol{x}(T)$。为了对其进行数值求解，可以使用时间步长 Δt 将时间域均匀划分，并得到时间节点 $t^{(l)}=l\Delta t$、$0\le l\le L+1$，其中 $(L+1)\Delta t=T$。将离散解定义为 $\boldsymbol{x}^{(l)}=\boldsymbol{x}(l\Delta t)$。于是，给定 $\boldsymbol{x}^{(l-1)}$，我们可以使用时间积分器来近似解 $\boldsymbol{x}^{(l)}$。我们可以考虑一种受前向欧拉积分器启发的方法，其中 (3.9) 式的左端项近似为

$$LHS\approx\frac{\boldsymbol{x}^{(l)}-\boldsymbol{x}^{(l-1)}}{\Delta t}.$$

而右端项则使用参数 $\boldsymbol{\theta}^{(l)}$ 近似为

$$RHS\approx\boldsymbol{V}(\boldsymbol{x}^{(l-1)};t^{(l)})=\boldsymbol{V}(\boldsymbol{x}^{(l-1)};\boldsymbol{\theta}^{(l)}).$$

其中，我们允许参数在每个时间步长上有所不同。将这两者结合起来，就恰好得到 (3.8) 式中残差网络的关系。换句话说，残差网络不过是非线性常微分方程系统的离散化。下面我们进一步发表评论，以强化这一联系。

- 在一个经过充分训练的残差网络中，给定 $\boldsymbol{x}_{(0)}$ 和网络的权重，我们预测$\boldsymbol{x}_{(L+1)}$。

- 在一个常微分方程系统中，给定 $\boldsymbol{x}(0)$ 和 $\boldsymbol{V}(\boldsymbol{x},t)$，我们预测 $\boldsymbol{x}(T)$。
- 训练残差网络意味着确定网络的参数 $\boldsymbol{\theta}$，使得当 $\boldsymbol{x}_{(0)}=\boldsymbol{x}_i$ 时，对于 $i=1,\cdots,N_{\text{train}}$，$\boldsymbol{x}_{(L+1)}$ 尽可能接近 $\boldsymbol{y}_i$。
- 从类比的常微分方程视角来看，训练意味着确定右端项 $\boldsymbol{V}(\boldsymbol{x},t)$，使得当 $\boldsymbol{x}(0)=\boldsymbol{x}_i$ 时，对于$i=1,\cdots,N_{\text{train}}$，$\boldsymbol{x}(T)$ 尽可能接近 $\boldsymbol{y}_i$。

- 在残差网络中，我们要寻找一个“单一的” $\boldsymbol{V}(\boldsymbol{x},t)$，使其能将 $\boldsymbol{x}_i$ 映射到 $\boldsymbol{y}_i$，适用于所有$1\le i\le N_{\text{train}}$。

3.4 神经常微分方程

受残差网络与常微分方程之间联系的启发，[4] 提出了神经常微分方程。考虑如下常微分方程系统:

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}=\boldsymbol{V}(\boldsymbol{x},t) \tag{3.10}$$

给定 $\boldsymbol{x}(0)$，我们希望求出 $\boldsymbol{x}(T)$。在 [4], 右端项中，即 $\boldsymbol{V}(\boldsymbol{x},\ t)$，由一个参数为 $\boldsymbol{\theta}$ 的前馈神经网络定义（见图 3.4）。网络的输入为 $(\boldsymbol{x},\ t)$，输出为 $\boldsymbol{V}(\boldsymbol{x},\ t)$（其维度与 $\boldsymbol{x}$相同）。根据这一描述，使用合适的时间推进格式（如前向欧拉法、龙格-库塔法等）求解系统 (3.10)。

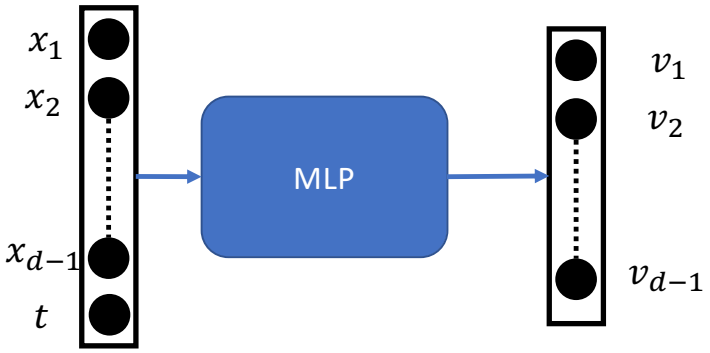


图 3.4：用于建模神经常微分方程中右端项的前馈神经网络。因变量的数量 $= d - 1$。

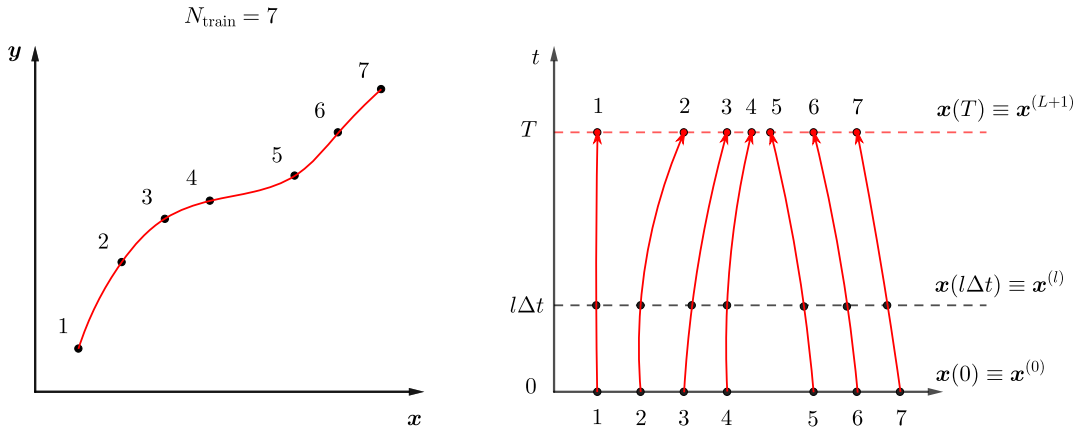


图3.5：回归问题与神经常微分方程之间的类比。

那么，我们如何使用这个网络来求解回归问题呢？假设给定带标签的训练数据 $\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : 1 \leq i \leq N_{\text{train}}\}$。这里假设 $\boldsymbol{x}_i$ 和 $\boldsymbol{y}_i$ 的维度均为 $d - 1$。关键思想是将 $\boldsymbol{x}_i$ 视为 $d - 1$维空间中的点，这些点表示系统的初始状态；将 $\boldsymbol{y}_i$ 视为表示最终状态的点。这样，回归问题就转化为寻找 (3.10) 的右端项，使其以尽可能小的误差将初始点映射到最终点。换言之，寻找满足以下条件的参数 $\boldsymbol{\theta}$

$$\Pi(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} |\boldsymbol{x}_i(T; \boldsymbol{\theta}) - \boldsymbol{y}_i|^2$$

被最小化。这里，$\boldsymbol{x}_i(T; \boldsymbol{\theta})$ 表示满足 (3.10) 的解（在时间 $t = T$），其中 $\boldsymbol{x}(0) = \boldsymbol{x}_i$，右端项由前馈神经网络 $\boldsymbol{V}(\boldsymbol{x}, t; \boldsymbol{\theta})$ 表示。请注意，$\boldsymbol{y}_i$ 是测量得到的输出值。当 $\boldsymbol{x}_i$ 和 $\boldsymbol{y}_i$ 的维度不同时，也可以用一种相对直接的方式扩展这一方法。总之，在神经常微分方程中，人们将一个回归问题转化为寻找常微分方程系统的非线性时变右端项的问题。

下面列出神经常微分方程与残差网络相比的优势和差异 :

• 如果 我们将神经常微分方程中的时间步数视为残差网络中隐 n
w

层 L，那么这两种方法的计算成本均为 $\mathcal{O}(L)$。这是执行一次前向传播和一次反向传播所需的成本。然而，记忆成本（即存储每一层权重所需的成本）有所不同。对于神经常微分方程，所有权重都属于用于表示函数 $\boldsymbol{V}(\boldsymbol{x},\ t;\boldsymbol{\theta})$ 的前馈网络。因此，权重数量与求解常微分方程所用的时间步数独立。另一方面，对于残差网络，权重数量随层数线性增长，因此存储这些权重的成本按 $\mathcal{O}(L)$ 增长。

- 在神经常微分方程中，我们可以取极限 $\Delta t \to 0$ 并研究其收敛性，因为这不会改变用于表示右端项的网络规模。然而，对于残差网络，这在计算上并不可行，因为 $\Delta t \to 0$ 对应于网络深度 $L \to \infty$!

- 残差网络使用一种前向欧拉法类型的方法，而在神经常微分方程中，可以使用任意时间积分器。尤其是，其他高阶显式时间积分器（如龙格-库塔方法）能够以更快的速度收敛到“精确”解。

第4章

使用多层感知机求解偏微分方程

有许多数值方法可用于求解偏微分方程。其中一些方法如下:

- 有限差分方法
- 有限体积方法
- 有限元方法
- 谱 Galerkin 方法和配置点方法
- 深度神经网络!

为了更好地理解其中的一些方法，尤其是深度神经网络，我们来考虑一个简单的模型问题，描述一维标量对流-扩散问题：求区间 $x \in (0,\ l)$ 上的 $u(x)$，使得

$$\begin{aligned} a\frac{\mathrm{d}u}{\mathrm{d}x} - \kappa\frac{\mathrm{d}^2u}{\mathrm{d}x^2} &= f(x), \quad x \in (0,\ell) \\ u(0) &= 0 \\ u(\ell) &= 1 \end{aligned} \tag{4.1}$$

其中，a 表示对流速度，κ 是扩散系数，而 $f(x)$ 是源项。这类方程用于模拟许多物理现象，例如流体输送污染物，或电子在半导体中的流动。该问题的多维形式为

$$\begin{aligned} \boldsymbol{a}\cdot\nabla\boldsymbol{u}(\boldsymbol{s}) - \kappa\Delta\boldsymbol{u}(\boldsymbol{s}) &= \boldsymbol{f}(\boldsymbol{s}), \quad \boldsymbol{s} \in \Omega \\ \boldsymbol{u}(\boldsymbol{s}) &= \boldsymbol{g}(\boldsymbol{s}), \quad \boldsymbol{s} \in \partial\Omega \end{aligned} \tag{4.2}$$

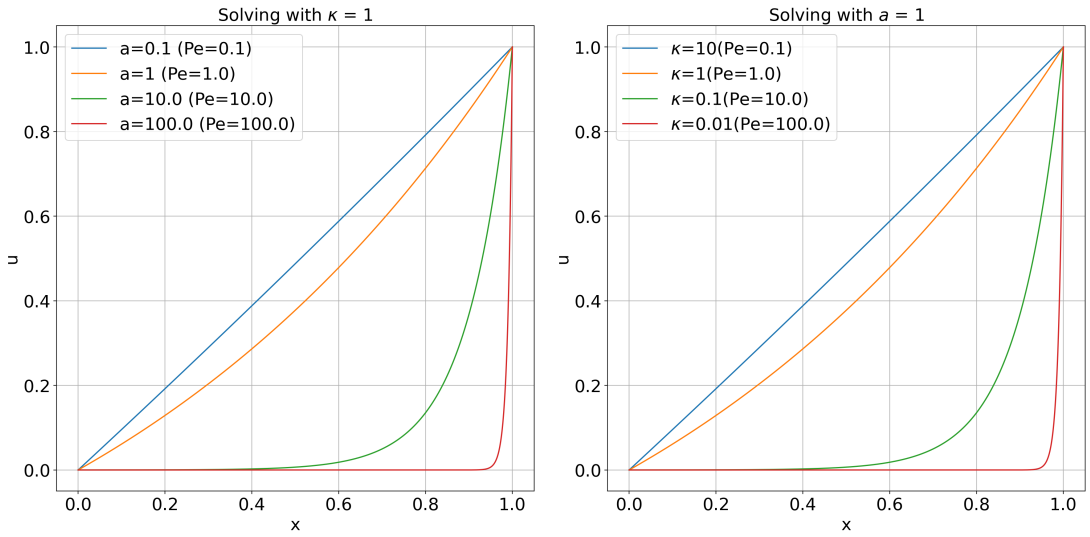
请注意，该模型问题是一个线性偏微分方程（一维情形下为常微分方程）。将速度 a 替换为 u，便得到黏性Burgers方程。

当 $f \equiv 0$ 时，方程 (4.1) 的解可以解析地写为

$$u(x) = \frac{1-\exp(ax/\kappa)}{1-\exp(a\ell/\kappa)}$$

其中，$a\ell/\kappa$ 称为佩克莱数（Pe），用于衡量对流强度与扩散强度之比。我们在图 4.1 中绘出了 a 和 κ 取不同值时的解。请注意，当 Pe 较小时，解基本上是一条直线。但随着 Pe 增大，解开始弯曲，并在右边界附近形成更陡峭的边界层。该边界层的厚度为 $\delta \approx \mathrm{Pe}\ \times l$。

现在，我们将考虑几种数值求解这一示例问题的方法。



(a) 固定 κ (b) 固定 a

图 4.1： $\ell = 1$ 条件下（4.1）的精确解。

4.1 有限差分法

有限差分格式的关键步骤如下：

1. 将区域离散化为网格点，目标是求出这些点上的解。2. 在这些点上用有限差分近似导数。由此得到一个（线性或非线性）代数方程系统。3. 使用合适的算法求解该系统，从而得到解。

将这些步骤应用于（4.1），可得：

1. 将区域离散为 $N+1$ 个点，其中 $x_i = ih$、 $0 \le i \le N$，且 $h = \ell/N$。我们希望求解 $u(x_i) = u_i$。我们还从边界条件得知 $u_0 = 0$ 和 $u_N = 1$。

2. 使用近似式

$$\frac{\mathrm{d}u}{\mathrm{d}x}(x_i) = \frac{u_{i+1} - u_{i-1}}{2h} + \mathcal{O}(h^2)$$
$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2}(x_i) = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \mathcal{O}(h^2)$$

请注意，上述两种近似均具有二阶精度。它们是“中心差分”近似，因为它们对第 i 个点两侧的点赋予了相同的权重。值得一提的是，当佩克莱数很大时，对流项的中心差分近似并不理想，因为它会导致数值不稳定性。在这种情况下，通常更倾向于采用“迎风”近似。将这些近似应用于

在 $x_i, 1 \le i \le N-1$ 处的偏微分方程

$$
\begin{aligned}
& a\frac{u_{i+1}-u_{i-1}}{2h} - \kappa\frac{u_{i+1}-2u_i+u_{i-1}}{h^2} = f_i \\
\iff\ & u_{i+1}\underbrace{\left(\frac{a}{2h}-\frac{\kappa}{h^2}\right)}_{\gamma} + u_i\underbrace{\left(\frac{2\kappa}{h^2}\right)}_{\beta} + u_{i-1}\underbrace{\left(-\frac{a}{2h}-\frac{\kappa}{h^2}\right)}_{\alpha} = f_i
\end{aligned}
$$

考察解未知的每个节点（请回忆， $u_0 = 0$ 和 $u_N = 1$ 是已知的），

$$
\begin{aligned}
\beta u_1 + \gamma u_2 &= -\alpha u_0 + f_1 \\
\alpha u_{i-1} + \beta u_i + \gamma u_{i+1} &= f_i, \quad \forall\ 2 \le i \le N-2 \\
\alpha u_{N-2} + \beta u_{N-1} &= -\gamma u_N + f_{N-1}
\end{aligned}
\tag{4.3}
$$

将 (4.3) 中的所有 $N-1$ 方程合并，可得以下线性系统

$$
\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} \tag{4.4}
$$

其中，三对角矩阵 $\boldsymbol{K}$ 以及 (4.4) 中的其他向量定义如下

$$
\begin{aligned}
\boldsymbol{K} &= \begin{bmatrix} \beta & \gamma & & 0 \\ \alpha & \ddots & \ddots & \\ & \ddots & \ddots & \gamma \\ 0 & & \alpha & \beta \end{bmatrix} \in \mathbb{R}^{(N-1)\times(N-1)}, \\
\boldsymbol{u} &= \begin{bmatrix} u_1 & u_2 & \cdots & u_{N-2} & u_{N-1} \end{bmatrix}^\top \in \mathbb{R}^{N-1}, \\
\boldsymbol{f} &= \begin{bmatrix} -\alpha u_0 + f_1 & f_2 & f_3 \cdots & f_{N-2} & f_{N-1} & -\gamma u_N + f_{N-1} \end{bmatrix}^\top \in \mathbb{R}^{N-1}
\end{aligned}
$$

3. 求解 $\boldsymbol{u} = \boldsymbol{K}^{-1}\boldsymbol{f}$。

注意:

- 实际应用中，我们从不真正求 $\boldsymbol{K}$ 的逆矩阵，因为计算代价很高。相反，我们使用高效的数值算法来求解系统 (4.4)。例如，对于这个特定的系统，可以使用托马斯三对角算法，它是高斯消元法的简化版本。

- 我们只能得到一个近似值 $u_i \approx u(x_i)$。为减小近似误差，我们可以缩小网格尺寸 h。或者，我们可以使用更高阶的有限差分近似，这会形成一个“更宽的模板”，用于近似每个点处的导数。

- 我们可以把每个“应用”偏微分方程的点视为一个配置点。在配置点处应用偏微分方程这一思路，也为我们接下来要介绍的方法所采用。使用多层感知机求解偏微分方程的方法也采用了这一思路。

4.2 谱配置法

谱配置法旨在求得一个解，将其表示为一组光滑的全局基函数的展开。基函数预先选定，而展开的系数则是未知数，通过要求偏微分方程的数值解在一组所谓的配置点上精确成立来计算这些系数。具体而言，这种方法包括以下步骤。

1. 选择一组具有以下性质的全局基函数：

   (a) 在所考虑的函数空间中构成完备基函数。(b) 足够光滑，因而可以计算导数。(c) 易于计算。(d) 导数易于计算。

例如，可以使用定义在 $\xi \in (-1,1)$ 上的切比雪夫多项式，其递推关系如下：

$$T_0(\xi) = 1, \quad T_1(\xi) = \xi, \quad T_{n+1}(\xi) = 2\xi T_n(\xi) - T_{n-1}(\xi)$$

图 4.2 展示了前几个切比雪夫多项式。请注意，这组基函数满足上述所有要求。它易于在任意点求值，因为可以利用上述递推关系以及两个低阶多项式的值，计算后续阶数的切比雪夫多项式。还可以对上述递推关系求导，从而得到各阶导数的递推关系。

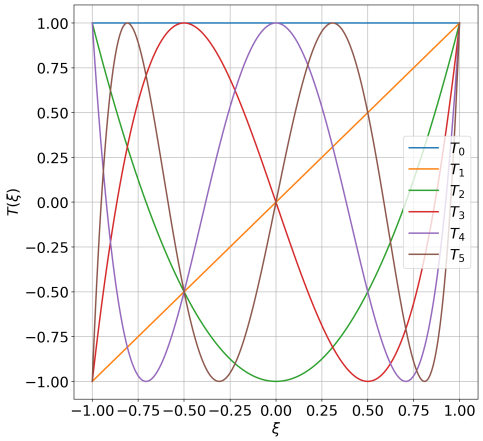


图 4.2：前几个切比雪夫多项式。

2. 将解写成基函数 $\{\phi_n(x)\}_{n=0}^N$ 的线性组合

$$u(x) = \sum_{n=0}^{N} u_n \phi_n(x) \tag{4.5}$$

其中 u_n 为基系数。对于我们的示例问题 (4.1)（假设 $\ell = 1$），我们将使用切比雪夫多项式 $\phi_n(x) = T_n(2x-1)$，并将自变量变换为使这些函数适用于区间 (0,1)。

3. 求出偏微分方程的导数，对于我们的示例问题，方程为

$$\begin{aligned} \frac{\mathrm{d}u}{\mathrm{d}x}(x) &= \sum_{n=0}^{N} u_n \phi_n'(x) = \sum_{n=0}^{N} u_n 2T_n'(2x-1) \\ \frac{\mathrm{d}^2u}{\mathrm{d}x^2}(x) &= \sum_{n=0}^{N} u_n \phi_n''(x) = \sum_{n=0}^{N} u_n 4T_n''(2x-1) \end{aligned} \tag{4.6}$$

4. 使用偏微分方程的边界条件。对于式 (4.1) 的具体情况，

$$\begin{aligned} u(0)=0 &\implies \sum_{n=0}^{N} u_n\phi_n(0)=\sum_{n=0}^{N} u_nT_n(-1)=0, \\ u(1)=1 &\implies \sum_{n=0}^{N} u_n\phi_n(1)=\sum_{n=0}^{N} u_nT_n(1)=1 \end{aligned} \tag{4.7}$$

这会产生关于 $N+1$ 系数的 2 个线性方程。接着，我们在区域内部选取一组（适当选定的）节点 $x_i, 1\le i\le N-1$，即配置点，并使用第 3 步中求得的导数，在这些 $N-1$ 节点处计算偏微分方程

$$\begin{aligned} & a\sum_{n=0}^{N} u_n\phi_n'(x_i)-\kappa\sum_{n=0}^{N} u_n\phi_n''(x_i)=f(x_i) \\ \implies & \sum_{n=0}^{N} u_n\left(2aT_n'(2x_i-1)-4\kappa T_n''(2x_i-1)\right)=f(x_i) \end{aligned} \tag{4.8}$$

这会产生关于 $N+1$ 系数的额外 $N-1$ 个方程。结合 (4.7) 和 (4.8)，得到以下线性系统

$$\boldsymbol{K}\boldsymbol{u}=\boldsymbol{f} \tag{4.9}$$

其中

$$\begin{aligned} &\boldsymbol{K}\in\mathbb{R}^{(N+1)\times(N+1)}, \\ &\boldsymbol{u}=\begin{bmatrix} u_0 & u_2 & \cdots & u_{N-1} & u_N \end{bmatrix}^\top\in\mathbb{R}^{N+1}, \\ &\boldsymbol{f}=\begin{bmatrix} 0 & f(x_1) & f(x_1) & \cdots & f(x_{N-2}) & f(x_{N-1}) & 1 \end{bmatrix}^\top\in\mathbb{R}^{N+1} \end{aligned}$$

5. 求解 $\boldsymbol{u}=\boldsymbol{K}^{-1}\boldsymbol{f}$。

我们需要恰当地选择配置点/求积点 x_i，使 $\boldsymbol{K}$ 具有理想的性质，从而让线性系统 (4.9) 更容易求解。这些性质包括可逆性、正定性、稀疏性等。

注 4.2.1. 这种方法称为配置法，因为偏微分方程是在指定的配置点/求积点 x_i处计算的。

注 4.2.2. 处理非线性偏微分方程时，最终会得到一个关于系数 $u_0, ..., u_N$ 的代数方程非线性系统，通常用牛顿法求解。

我们来看一种用于求解谱方法展开系数的最小二乘变体。和前面一样，我们仍然使用 (4.5) 表示解，并计算其导数。然后，通过最小化以下损失函数来求出系数 $\boldsymbol{u}$

$$\begin{aligned} \Pi(\boldsymbol{u}) &= \Pi_{\text{int}}(\boldsymbol{u})+\lambda\Pi_{\text{bc}}(\boldsymbol{u}) \\ \Pi_{\text{bc}}(\boldsymbol{u}) &= \left|\sum_{n=0}^{N} u_n\phi_n(0)-0\right|^2+\left|\sum_{n=0}^{N} u_n\phi_n(1)-1\right|^2 \\ \Pi_{\text{int}}(\boldsymbol{u}) &= \frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}}\left|a\sum_{n=0}^{N} u_n\phi_n'(x_i)-\kappa\sum_{n=0}^{N} u_n\phi_n''(x_i)-f(x_i)\right|^2 \end{aligned} \tag{4.10}$$

这可以使用我们在第2章中介绍过的任意一种基于梯度的方法求解。这种方法在处理非线性偏微分方程时尤其有用。事实上，在这些情况下，无法像 (4.9) 那样用系数表示一个线性系统。这里有几点需要注意

- λ 是一个用于对内部损失和边界损失进行不同缩放的参数。
- 内部点的数量 x_i 可以独立于基函数的数量进行选择。换句话说， $N_{\text{训练}}$ 不必与 N相同。
- 我们将在下一节看到，这种谱方法的变体与使用深度神经网络求解偏微分方程的方式非常相似。

4.3 物理信息神经网络（PINNs）

使用神经网络求解偏微分方程的想法由Lagaris等人在20世纪90年代至21世纪初提出 [11]。随着人们重新对使用机器学习工具求解偏微分方程产生兴趣，Raissi等人在2019年重新发现了这一想法 [24], ，并将其称为PINNs（物理信息神经网络）。PINNs的基本思想与回归类似，不同之处在于损失函数 $\Pi(\boldsymbol{\theta})$包含所研究偏微分方程中的微分算子。下面我们以一维标量偏微分方程为例概述主要步骤，这些步骤很容易扩展到多维偏微分方程组。我们建议读者思考这种方法与上一节所述谱配置法之间的相似之处和差异。

1. 选择一个神经网络作为偏微分方程解的函数表示：

$$u = \mathcal{F}(x; \boldsymbol{\theta}). \tag{4.11}$$

这种表示需要具备一些关键性质：

(a) 这种表示是否完备？也就是说，能否用这种表示准确逼近所需的函数类别？答案是肯定的，这是因为神经网络具有通用逼近定理（见第2.3.1节）。(b) 这种表示是否光滑？如果激活函数是光滑的，例如双曲正切函数、正弦函数等，答案就是肯定的。注意，我们不能使用 ReLU，因为它没有足够阶数的光滑导数。(c) 这种表示是否容易求值？答案是肯定的，因为前向传播过程很快。(d) 导数是否容易求值？答案是肯定的，因为可以使用反向传播。下文将对此进行详细讨论。

2. 给定表示式 (4.11)，我们需要求出 $\boldsymbol{\theta}$ ，使偏微分方程以某种合适的形式得到满足。请将此与式 (4.5) 给出的谱配置近似进行比较；在后者中，我们需要确定系数 u_n。注意，在式 (4.5) 中，对系数 u_n 的依赖是线性的，而在式 (4.11) 中，对 $\boldsymbol{\theta}$ 的依赖可能高度非线性。

3. 接下来，我们要计算该表示的导数。考虑图 4.3 所示的网络计算图。它由交替进行的仿射变换和逐分量非线性变换组成。输出相对于输入的导数可以通过反向传播求得。图 4.3 中的计算图是通过

只需在图 2.8 所示的计算图中设置 $\Pi = \boldsymbol{x}^{(L+1)}$ 得到的。此外，一旦我们认识到 $\frac{\partial \boldsymbol{x}^{(L+1)}}{\partial \boldsymbol{x}^{(L+1)}} = \mathbf{1}$ 是单位矩阵，就可以轻易地从该计算图中看出

$$\frac{\partial \boldsymbol{x}^{(L+1)}}{\partial \boldsymbol{x}^{(0)}} = \boldsymbol{W}^{(L+1)}\boldsymbol{S}^{(L+1)}\boldsymbol{W}^{(L)}\boldsymbol{S}^{(L)}\cdots\boldsymbol{W}^{(2)}\boldsymbol{S}^{(2)}\boldsymbol{W}^{(1)}\boldsymbol{S}^{(1)}$$

因此，要计算$\mathrm{d}u\ \mathrm{d}x$，需要扩展原始计算图，添加一个反向分支，用于计算向量 $\boldsymbol{\xi}^{(l)}$ 各分量的激活函数导数（见图 4.3）。二阶导数$\mathrm{d}^2u\ \mathrm{d}x^2$ 通过对扩展后的计算图执行反向传播来计算。要计算更高阶导数，还需要以类似方式进一步扩展计算图。在 PyTorch 中调用“自动微分”时，后台正是如此处理的。

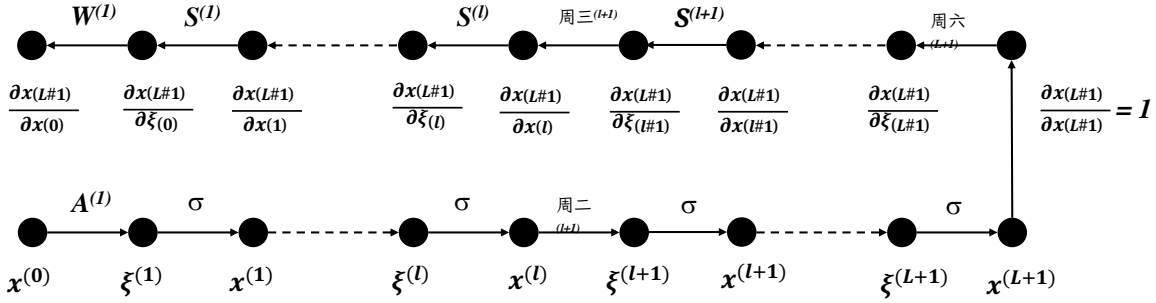


图 4.3：用于计算相对于网络输入的导数的扩展计算图。

4. 将解的泛函表示 (4.11) 代入偏微分方程，以求出参数 $\boldsymbol{\theta}$。为此，我们首先定义一组点 $\mathcal{S} = \{x_i : 1 \le i \le N_{\text{train}}\}$，用于训练网络，类似于谱配置法中的配置点集合。随后，我们需要定义损失函数（针对示例问题 (4.1) 特别设定）

$$\begin{aligned}
\Pi(\boldsymbol{\theta}) &= \Pi_{\text{int}}(\boldsymbol{\theta}) + \lambda_b \Pi_{\text{b}}(\boldsymbol{\theta}),\\
\Pi_{\text{b}}(\boldsymbol{\theta}) &= \left(\mathcal{F}(0;\boldsymbol{\theta}) - 0\right)^2 + \left(\mathcal{F}(1;\boldsymbol{\theta}) - 1\right)^2,\\
\Pi_{\text{int}}(\boldsymbol{\theta}) &= \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left(a\mathcal{F}'(x_i;\boldsymbol{\theta}) - \kappa\mathcal{F}''(x_i;\boldsymbol{\theta})_n - f(x_i)\right)^2.
\end{aligned} \tag{4.12}$$

训练网络后，也就是求解最小化问题 $\boldsymbol{\theta}^* = \arg\min\ \Pi(\boldsymbol{\theta}), \boldsymbol{\theta}$，解表示为 $u^*(x) = \mathcal{F}(x;\ \boldsymbol{\theta}^*)$。请注意，这与谱配置法的最小二乘变体完全相同：通过最小化类似的损失来求出系数 u_n。

我们作几点说明：

- 当我们能够找到满足 $\Pi(\boldsymbol{\theta}^*) = 0$ 的 $\boldsymbol{\theta}^*$ 时，这意味着 $\Pi_{\text{积分}}(\boldsymbol{\theta}^*) = 0$，且 $\Pi_{\text{边界}}(\boldsymbol{\theta}^*) = 0$。换言之，偏微分方程残差在配置点处为零。只要配置点充分覆盖整个区域，就能得到良好的解。

- 有多种方法可以提高物理信息神经网络的准确性，例如

– 增加配置点的数量。– 更改用于加权边界损失的超参数 λ_b 。– 增大网络的规模，也就是增加 N_θ。

- 微分方程的边界条件（BCs）承载着我们试图描述的现象的基本物理属性，因此数值解满足这些条件至关重要。在物理信息神经网络的框架中，BCs 通过惩罚项 $\Pi_b(\boldsymbol{\theta})$ 作为软约束施加。因此，超参数 λ_b 在网络训练中发挥着关键作用，因为它在最小化过程中平衡了两个损失项之间的相互作用。如果不同损失项的梯度没有得到适当的缩放，那么收敛到同时满足 BCs 和偏微分方程本身的解可能会极其缓慢。对于刚性偏微分方程，这一问题尤为突出。为了解决这个问题，研究者提出了多种自适应技术，以便在训练过程中调整 λ_b 的取值 [29, 16, 3]。

4.4 将物理信息神经网络扩展到更一般的偏微分方程

考虑一个一般的偏微分方程：求解 $\boldsymbol{u}: \Omega \subset \mathbb{R}^d \to \mathbb{R}^D$，使得

$$\begin{aligned} L(\boldsymbol{u}(\boldsymbol{x})) &= \boldsymbol{f}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \\ B(\boldsymbol{u}(\boldsymbol{x})) &= \boldsymbol{g}(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{4.13}$$

其中 L 是微分算子， $\boldsymbol{f}$ 是已知强迫项， B 是边界算子， $\boldsymbol{g}$ 是边界条件的非齐次部分（也已给定）。

例如，我们可以考虑三维不可压缩纳维-斯托克斯方程，在 $\Omega = \Omega_S \times [0,T]$ 上求解速度场 $\boldsymbol{v} = [v_1, v_2, v_3]$ 和压力 p。这里，Ω_S 是三维空间域， $[0,T]$ 是感兴趣的时间区间。方程如下：

$$\begin{aligned} \frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{v} \cdot \nabla \boldsymbol{v} + \nabla p - \mu \Delta \boldsymbol{v} &= \boldsymbol{f}, \quad \forall\ (\boldsymbol{s}, t) \in \Omega \\ \nabla \cdot \boldsymbol{u} &= 0, \quad \forall\ (\boldsymbol{s}, t) \in \Omega \\ \boldsymbol{v} &= \boldsymbol{0}, \quad \forall\ (\boldsymbol{s}, t) \in \partial\Omega_S \times [0,T] \\ \boldsymbol{v}(\boldsymbol{s}, 0) &= \boldsymbol{v}_0(\boldsymbol{s}), \quad \forall\ \boldsymbol{s} \in \Omega_S. \end{aligned} \tag{4.14}$$

上面的第一个方程是线动量平衡。第二个方程强制满足质量守恒。第三个方程是无滑移边界条件，适用于边界刚性且固定的情况。第四个方程给出了初始速度场。

要为 (4.13) 设计物理信息神经网络，网络的输入应为自变量 $\boldsymbol{x}$，输出应为解向量 $\boldsymbol{u}$。对于纳维-斯托克斯系统 (4.14) 这一具体情况，网络的输入为 $[s_1, s_2, s_3, t] \in \mathbb{R}^4$，而输出向量为 $\boldsymbol{u} = [v_1, v_2, v_3, p] \in \mathbb{R}^4$。步骤如下：

1. 构造损失函数
   - 定义内部残差 $R(\boldsymbol{u}) = L(\boldsymbol{u}) - \boldsymbol{f}$。
   - 定义边界残差 $R_b(\boldsymbol{u}) = B(\boldsymbol{u}) - \boldsymbol{g}$。
   - 在区域内部选择合适的 N_v 配置点，并在区域边界上选择 N_b 点来评估残差。这些点可以根据求积规则选取，例如高斯、洛巴托、均匀分布、随机分布等。

那么损失函数为

$$\Pi(\boldsymbol{\theta}) = \Pi_{\text{int}}(\boldsymbol{\theta}) + \lambda_b \Pi_{\text{b}}(\boldsymbol{\theta})$$
$$\Pi_{\text{int}}(\boldsymbol{\theta}) = \frac{1}{N_v} \sum_{i=1}^{N_v} |R(\boldsymbol{\mathcal{F}}(\boldsymbol{x}_i; \boldsymbol{\theta})|^2$$
$$\Pi_{\text{b}}(\boldsymbol{\theta}) = \frac{1}{N_b} \sum_{i=1}^{N_b} |R_b(\boldsymbol{\mathcal{F}}(\boldsymbol{x}_i; \boldsymbol{\theta})|^2$$

2. 训练网络：求 $\boldsymbol{\theta}^* = \arg\min \Pi(\boldsymbol{\theta})$，并将解设为 $\boldsymbol{u}^* = \boldsymbol{\mathcal{F}}(\boldsymbol{x};\ \boldsymbol{\theta}\boldsymbol{\theta}$ *)

这里我们做一些说明:

- 这 隐含地假设还在损失 $\Pi(\boldsymbol{\theta})$ 中添加了一个权重正则化项 .
- $\boldsymbol{u}^*(\boldsymbol{x})$ 是偏微分方程的精确解吗？答案是否定的!

  – 首先， $\Pi(\boldsymbol{\theta}^*)$ 可能不为零。– 即使 $\Pi(\boldsymbol{\theta}^*)$ 恒为零，也只意味着残差在配置点处为零。然而，这并不能保证残差在区域内处处为零。为此 N_v， $N_b \to \infty$。– 此外，对于固定的网络（N_θ 固定），我们无法表示所有函数。为此，我们需要 $N_\theta \to \infty$。

• 在实际应用中，我们只计算 $\Pi(\boldsymbol{\theta}^*)$。解误差 $\|\boldsymbol{e}\| = \|\boldsymbol{u}^* - \boldsymbol{u}\|$ 与这个损失值有关吗？如果有关，只要损失足够小，我们是否就能说这个误差也会很小？这正是我们试图在下一节回答的问题。

4.5 物理信息神经网络的误差分析

为了评估误差 $\boldsymbol{e} = \boldsymbol{u}^* - \boldsymbol{u}$，我们需要知道精确解 $\boldsymbol{u}$，但通常无法获得。我们考虑一种解决这一问题的方法：将讨论范围限制在线性偏微分方程上，即 L 和 B 都是线性算子。

注意，如果 $\boldsymbol{u}$ 是精确解，那么

$$L(\boldsymbol{e}) = L(\boldsymbol{u}^* - \boldsymbol{u}) = L(\boldsymbol{u}^*) - L(\boldsymbol{u}) = L(\boldsymbol{u}^*) - \boldsymbol{f} = R(\boldsymbol{u}^*) \tag{4.15}$$

and

$$B(\boldsymbol{e}) = B(\boldsymbol{u}^* - \boldsymbol{u}) = B(\boldsymbol{u}^*) - B(\boldsymbol{u}) = B(\boldsymbol{u}^*) - \boldsymbol{g} = R_b(\boldsymbol{u}^*) \tag{4.16}$$

因此，(4.15) 和 (4.16) 导出了一个由多层感知机解的残差驱动的 $\boldsymbol{e}$ 偏微分方程 n,

$$\begin{aligned} L(\boldsymbol{e}) &= R(\boldsymbol{u}^*), \quad \text{in } \Omega \\ B(\boldsymbol{e}) &= R_b(\boldsymbol{u}^*), \quad \text{on } \Omega \end{aligned} \tag{4.17}$$

如果 $\boldsymbol{u}^*$ 的残差为零，那么 $\boldsymbol{e} = 0$。遗憾的是，这些残差并不为零。我们最多只能说，它们在配置点处很小。然而，根据适定偏微分方程稳定性理论，我们有

$$\|\boldsymbol{e}\|_{L^2(\Omega)} \leq C_1 \left(\|R(\boldsymbol{u}^*)\|_{L^2(\Omega)} + \|R_b(\boldsymbol{u}^*)\|_{L^2(\partial\Omega)} \right) \tag{4.18}$$

其中， C_1 是一个稳定性常数，它取决于偏微分方程、区域 Ω 等。这是所有适定偏微分方程都满足的条件。它表明，如果驱动偏微分方程的项很小，那么偏微分方程的解也会很小。这个方程告诉我们，只要能控制多层感知机解的残差，就能控制误差。然而，在实践中，我们知道并控制的是Π_{int}， Π_{b}，而不是 $\|R(\boldsymbol{u}^*)\|^2_{L^2(\Omega)}$ $\|R_b(\boldsymbol{u}^*)\|^2_{L^2(\partial\Omega)}$。那么问题就变成了，这些量之间是否有关联。下文的分析将回答这个问题，

$$\begin{aligned}\|R(\boldsymbol{u}^*)\|_{L^2(\Omega)} &= \left|m_\Omega \Pi_{\text{int}}(\boldsymbol{\theta}^*)^{1/2} + \|R(\boldsymbol{u}^*)\|_{L^2(\Omega)} - m_\Omega \Pi_{\text{int}}(\boldsymbol{\theta}^*)^{1/2}\right| \\ &\leq m_\Omega \Pi_{\text{int}}(\boldsymbol{\theta}^*)^{1/2} + \left|\|R(\boldsymbol{u}^*)\|_{L^2(\Omega)} - m_\Omega \Pi_{\text{int}}(\boldsymbol{\theta}^*)^{1/2}\right| \\ &\leq m_\Omega \Pi_{\text{int}}(\boldsymbol{\theta}^*)^{1/2} + C_2(N_v)^{-\alpha}\end{aligned} \tag{4.19}$$

其中， m_Ω 是区域 Ω 的测度， C_2 和 $\alpha > 0$ 取决于所选求积点的类型。上式第一行通过加上并减去项 $m_\Omega \Pi_{\text{int}}(\boldsymbol{\theta}^*)^{1/2}$ 得到。第二行利用了三角不等式。第三行说明了用在求积点处求得的有限个函数值之和近似积分时产生的误差。

边界残差同理

$$\|R_b(\boldsymbol{u}^*)\|_{L^2(\partial\Omega)} \leq m_{\partial\Omega} \Pi_{\text{b}}(\boldsymbol{\theta}^*)^{1/2} + C_3(N_b)^{-\beta} \tag{4.20}$$

其中， $m_{\partial\Omega}$ 是 $\partial\Omega$ 的测度， C_3 和 $\beta > 0$ 取决于边界求积点的类型。

利用 (4.18)、(4.19) 和 (4.20)，可得

$$\|\boldsymbol{e}\|_{L^2(\Omega)} \leq C_1 \left(\underbrace{m_\Omega \Pi_{\text{int}}(\boldsymbol{\theta}^*)^{1/2} + m_{\partial\Omega} \Pi_{\text{b}}(\boldsymbol{\theta}^*)^{1/2}}_{\text{reduced by } N_\theta \uparrow} + \underbrace{C_2(N_v)^{-\alpha} + C_3(N_b)^{-\beta}}_{\text{reduced by } N_v, N_b \uparrow} \right) \tag{4.21}$$

该方程表明，可以通过减小损失函数（增大 N_θ）以及增加内部和边界配置点的数量，来控制物理信息神经网络解的误差。关于这一分析的更多细节，请参阅 [18]。

4.6 使用物理信息神经网络进行数据同化

科学和工程领域经常会遇到数据同化问题。在这类问题中，我们可以对某个量进行少量稀疏测量，并利用这些测量值在细网格上求出该量在各处的值。此外，我们还已知一条物理原理（以偏微分方程的形式表示），感兴趣的变量必须满足该原理。

假设我们获得了区域 Ω 上某个量 u 的一组稀疏测量

$$u_i = u(\boldsymbol{x}_i), \quad \boldsymbol{x}_i \in \Omega, \quad 1 \leq i \leq M$$

此外，我们知道 u 在 Ω 上满足某个约束条件 $R(u) = 0$。那么，数据同化就是利用这些信息求出 u 在 Ω 中任意 $\boldsymbol{x} \in$处的值。

我们可以使用物理信息神经网络来求解这个问题。首先，我们用神经网络 $\mathcal{F}(\boldsymbol{x}, \boldsymbol{\theta})$ 表示 u。接下来，我们定义一个损失函数

$$\Pi(\boldsymbol{\theta}) = \underbrace{\frac{\lambda_I}{M} \sum_{i=1}^{M} (u_i - \mathcal{F}(\boldsymbol{x}_i, \boldsymbol{\theta}))^2}_{\text{data matching}} + \underbrace{\frac{1}{N_v} \sum_{i=M+1}^{M+N_v} |R(\mathcal{F}(\boldsymbol{x}_i, \boldsymbol{\theta}))|^2}_{\text{physical constraint}} + \underbrace{\lambda \|\boldsymbol{\theta}\|^2}_{\text{smoothness}}$$

其中 $\boldsymbol{x}_i,\quad M+1 \leq i \leq M+N_v$ 是用于计算残差的一些配置点，而 $\lambda_I,\ \lambda$ 是超参数。然后，我们通过求取 $\boldsymbol{\theta}^* = \arg\min \Pi(\boldsymbol{\theta})$ 来训练网络，并$\boldsymbol{\theta}$将物理信息神经网络的解设为 $\boldsymbol{u}^* = \boldsymbol{\mathcal{F}}(\boldsymbol{x};\ \boldsymbol{\theta}^*)$。

第5章

卷积神经网络

在前几章中，我们已经了解了如何使用全连接层构建神经网络。现在我们来看看另一类层，称为卷积层，它们在处理图像输入时非常有用。这类任务包括将图像分类到不同类别、对图像进行语义分割，以及将图像从一种类型转换为另一种类型。

5.1 函数与图像

考虑定义在 $\boldsymbol{x} \in [a,b] \times [c,d] \subset \mathbb{R}^2$ 上的函数 $u(\boldsymbol{x})$。那么，我们可以将该函数的离散化版本可视化为图像 $\boldsymbol{U} \in \mathbb{R}^{N_1 \times N_2}$，其中

$$U[i,j] = u(ih, jh), \quad 1 \le i \le N_1,\ 1 \le j \le N_2,\ h = \text{ pixel size.} \tag{5.1}$$

请注意，式 (5.1) 中的图像定义了一个灰度图像，其中 u 在每个像素处的值就是强度。如果我们处理的是彩色图像，那么它将是一个三维张量，第三个维度对应红、蓝、绿通道。换句话说，$\boldsymbol{U} \in \mathbb{R}^{N_1 \times N_2 \times 3}$。

如果我们想使用一个以大小为 100×100 的彩色二维图像作为输入的全连接神经网络（多层感知机），那么将整个图像展开为单个向量后，输入维度将达到 3×10^4，数值非常大。这会导致连接层规模过大，在计算上不可行。其次，展开图像时，我们会丢失原始图像的所有空间上下文。最后，我们会认为，边缘检测等局部操作在图像的任何区域都应当相同。考虑全连接层的权重，它们可表示为矩阵 W_{ij}，其中 i 索引表示线性变换的输出，j 索引表示输入。如果对每个输出索引执行的操作都相同，那么我们就会对每个 i 执行相同的操作，因此不再需要矩阵。为了解决这些问题，我们可以对函数使用卷积算子。

5.2 函数卷积

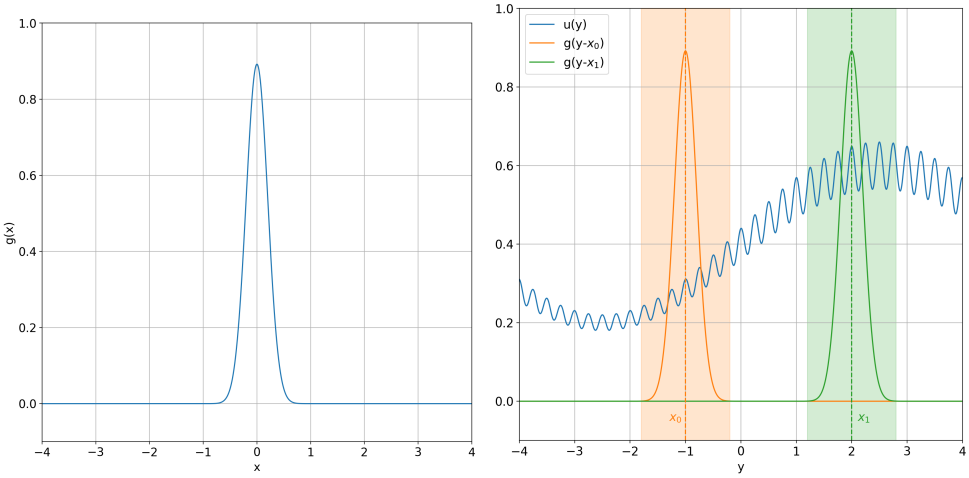
卷积算子将函数映射为函数。考虑函数 $u(\boldsymbol{x})$，$\boldsymbol{x} \in \mathbb{R}^d$，以及一个足够光滑、随 $|\boldsymbol{x}| \to \infty$ 衰减的核函数 $g(\boldsymbol{x})$。那么卷积算子定义为

$$\overline{u}(\boldsymbol{x}) = \int_{\mathbb{R}^d} g(\boldsymbol{y} - \boldsymbol{x}) u(\boldsymbol{y}) \mathrm{d}\boldsymbol{y} \tag{5.2}$$

我们可以将卷积算子理解为通过改变 $\boldsymbol{x}$ 对 u 进行采样。例如，在一维情况下，设 $u(x)$ 和 $g(x)$ 如图 5.1 所示，并且

$$\overline{u}(x) = \int_{\mathbb{R}} g(y-x)u(y)\mathrm{d}y.$$

考虑一个点 x_0。然后， $g(y-x_0)$ 将卷积核平移到位置 x_0，此时会对橙色阴影区域中的函数 u 进行采样。同样地，对于另一个点 x_1， $g(y-x_1)$ 将卷积核平移到位置 x_1，此时会对绿色阴影区域中的函数 u 进行采样。因此，随着卷积核移动，它会在不同的窗口中对 u 进行采样。请注意，无论 x 的取值如何，都会应用相同的操作。现在让我们考虑几个典型的核函数。



（a）卷积核 $g(x)$　　（b）采样

图 5.1：使用平移卷积核在一维中进行采样。

5.2.1 例 1

一种常用的选择是高斯核

$$g(\boldsymbol{\xi}) = \rho(|\boldsymbol{\xi}|),$$

其中，对于任意标量 r，

$$\rho(r) = \frac{\exp(-r^2/2\sigma^2)}{\sqrt{(2\pi\sigma^2)^d}}.$$

它用作平滑/模糊滤波器。这里， d 是维度， σ 表示扩散度。 $\sigma = 0.2$时，该卷积核在一维中的形式如图 5.1(a) 所示。该卷积核的一些重要性质如下：

- 它是各向同性的，因为它只取决于 $\boldsymbol{\xi}$的模。
- 在整个区域上的积分为 1。
- 它由 σ 参数化， σ 表示频率截止，因为比 σ 更精细的尺度会被卷积滤除。这种平滑效应如图 5.2 所示。

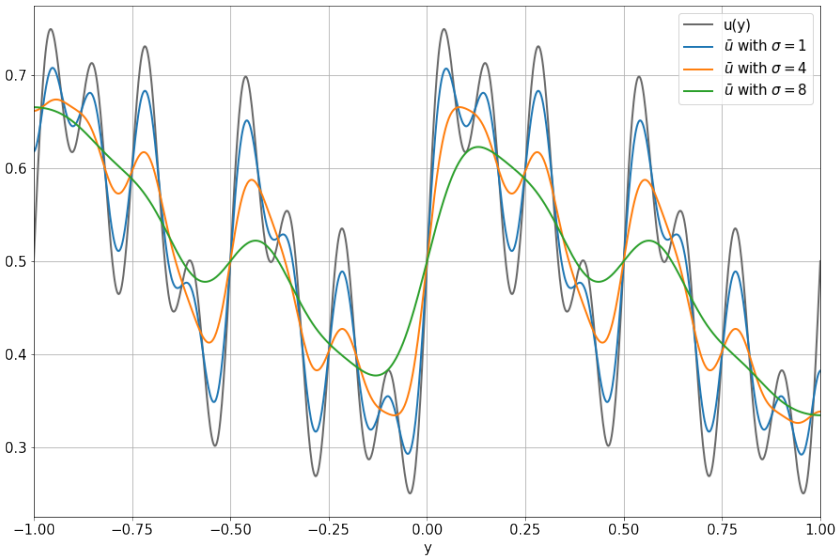


图 5.2：带高斯核的一维卷积。

5.2.2 例 2

我们再来看一个例子：某个卷积核可以生成 u 的平滑版本的导数。在二维中，我们希望它看起来像这样：

$$\begin{aligned}\overline{u}(\boldsymbol{x}) &= \frac{\partial}{\partial x_1}\left(\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\rho(|\boldsymbol{y}-\boldsymbol{x}|)u(\boldsymbol{y})dy_1dy_2\right)\\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{\partial\rho(|\boldsymbol{y}-\boldsymbol{x}|)}{\partial x_1}u(\boldsymbol{y})dy_1dy_2\\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\underbrace{\left(-\frac{\partial\rho(|\boldsymbol{y}-\boldsymbol{x}|)}{\partial y_1}\right)}_{\text{required kernel}}u(\boldsymbol{y})dy_1dy_2\end{aligned} \tag{5.3}$$

图 5.3 展示了这个卷积核在一维和二维中的形式。请注意，这个卷积核的作用类似于平滑后的有限差分运算。也就是说，卷积核中心左侧的区域赋予负权重，右侧的区域赋予正权重。

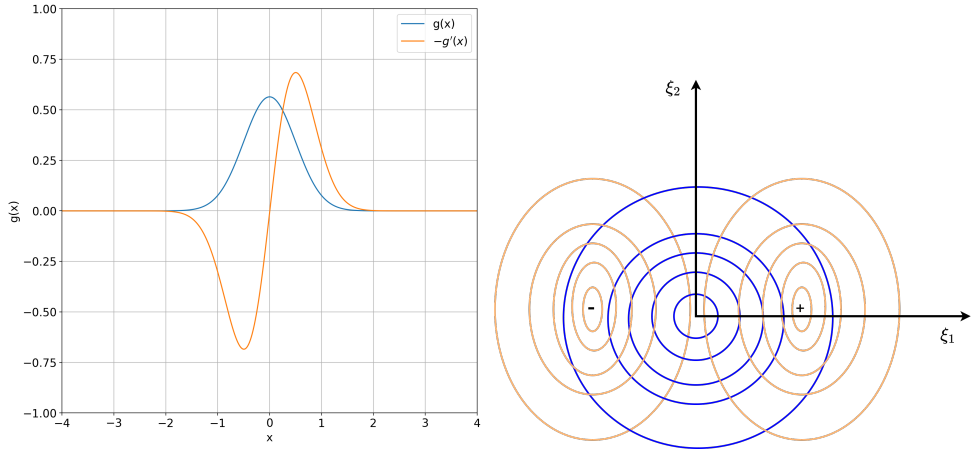
5.3 离散卷积

为了计算二维离散卷积，考虑式 (5.2)，并使用求积对其进行离散化。于是我们有

$$\overline{U}[i,j] = \sum_{m=1}^{N_1}\sum_{n=1}^{N_2} g[m-i,n-j]U[m,n] \tag{5.4}$$

其中，我们将测度 h^2 吸收到卷积核的定义中。与连续情形一样，我们假设 g 在超过某个距离后趋于零

$$_g[p,q] = 0, \qquad |p|,\ |q| > N \text{（卷积核宽度的度量）。}$$



(a) 一维　(b) 二维，沿 1 方向求导

图 5.3：导数核。蓝色曲线表示高斯核，橙色曲线表示导数。

因此，通过排除所有卷积结果为零的像素，求和范围缩小了，

$$\overline{U}[i,j] = \sum_{m=i-\bar{N}}^{i+\bar{N}} \sum_{n=j-\bar{N}}^{j+\bar{N}} g[m-i, n-j]U[m,n]. \tag{5.5}$$

设 $m' = m - i$ 和 $n' = n - j$。那么

$$\overline{U}[i,j] = \sum_{m'=-\bar{N}}^{\bar{N}} \sum_{n'=-\bar{N}}^{\bar{N}} g[m', n']U[i+m', j+n']. \tag{5.6}$$

这正是深度学习中应用卷积的方式。因此，卷积完全由以下内容决定：

$$g[m,n], \quad |m|, |n| \leq \bar{N},$$

它们将成为卷积层的权重，权重数量为 $(\bar{N}+1)^2$。

我们来看一些例子：

- 一个平滑核可以是

$$\frac{1}{8}\begin{bmatrix} \frac{1}{4} & 1 & \frac{1}{4} \\ 1 & 3 & 1 \\ \frac{1}{4} & 1 & \frac{1}{4} \end{bmatrix} \approx \text{Gaussian kernel with some } \sigma$$

- 沿 x 方向和 y 方向求导所用的卷积核为

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

- 同样，沿 x 和 y 方向的二阶导数由以下形式的卷积核给出

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & -1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

- 而拉普拉斯算子由以下类型的卷积核给出

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

¯**注释 5.3.1.** 不同方向上的 N 可以不同。也就是说，沿每个方向的卷积核可以具有不同的宽度。

5.4 与有限差分的联系

卷积的概念与有限差分格式的模板之间有着密切的联系。下文将对此进行阐述。

假设函数 $u(x_1, x_2)$ 定义在一个有限网格上，其中网格点以 (i, j) 为索引，网格大小为 h。那么我们使用记号 $U[i, j] = u(x_1^i, x_2^j)$。在 (i, j) 处进行泰勒级数展开

$$\frac{U[i+1,j] - U[i-1,j]}{2h} = \frac{u(x_1^{i+1}, x_2^j) - u(x_1^{i-1}, x_2^j)}{2h} = \frac{\partial}{\partial x_1} u(x_1^i, x_2^j) + \mathcal{O}(h^2).$$

上面的运算与 权重由.....给出的离散卷积运算完全相同 y

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \frac{1}{2h} \approx \frac{\partial u}{\partial x_1}.$$

因此，我们可以说，这种卷积运算近似于沿 1 方向求导。类似地，我们可以证明

$$\frac{U[i+1,j] - 2U[i,j] + U[i-1,j]}{h^2} = \frac{\partial}{\partial x_1^2} u(x_1^i, x_2^j) + \mathcal{O}(h^2).$$

因此，使用以下卷积核进行卷积

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} \frac{1}{h^2} \approx \frac{\partial u}{\partial x_1^2},$$

近似于计算沿 1 方向的二阶导数。

5.5 卷积层

需要记住的要点是:

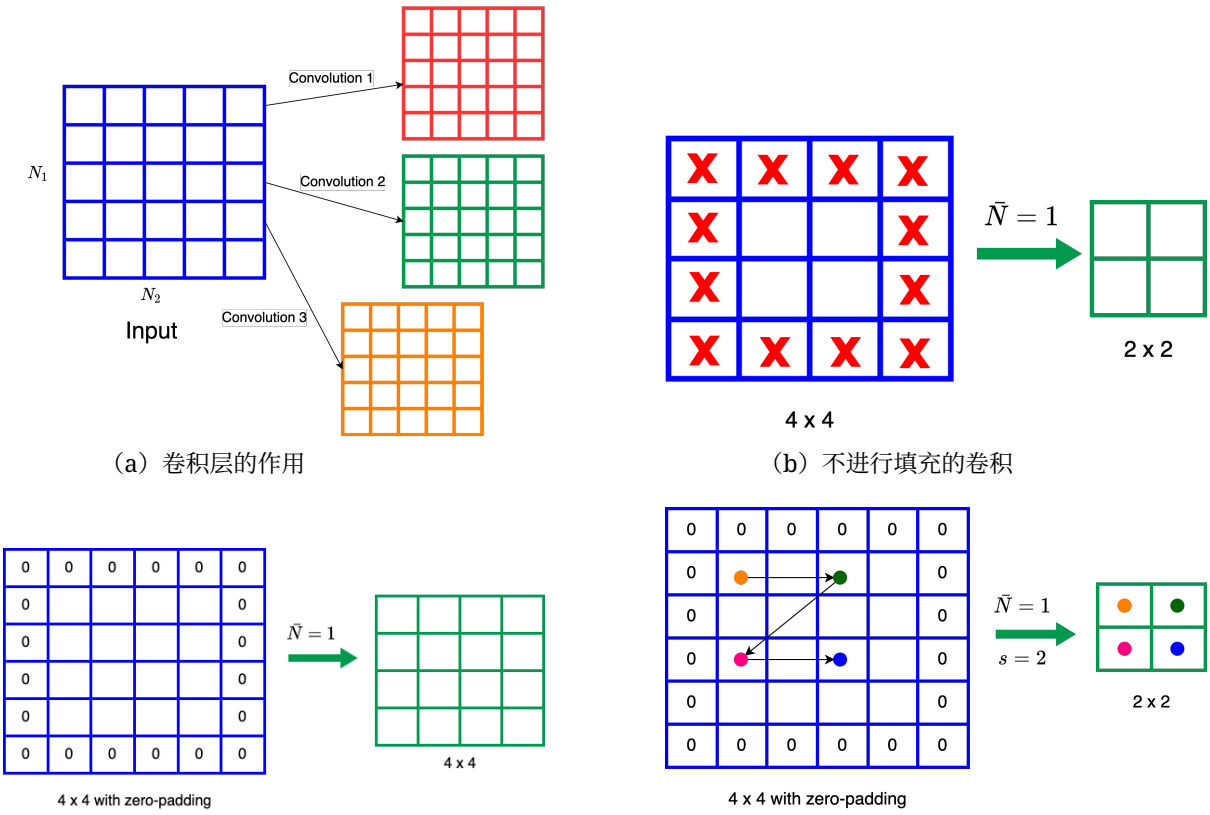
- 每个卷积层由若干离散卷积组成，每个卷积都有自己的卷积核。
- 卷积核的权重决定了它的作用（平滑、一阶导数、二阶导数等），这些都是可学习参数，并在网络训练时确定。因此，理解学习过程的方式是：网络会学习适用于其任务的运算（卷积）。例如，这项任务可以是分类问题。

假设我们有一幅 $N_1 \times N_2$ 图像作为输入。那么，我们将在一个卷积层中进行多个卷积，每个卷积都会生成一幅不同的图像，如图 5.4(a) 所示。该层的可训练参数是每个卷积核的权重。假设卷积核在每个方向上的宽度为 $\bar{N}$，且共有 P 个卷积核，那么可训练权重的数量将为 $P \times (2\bar{N}+1)^2$。

接下来，我们考虑应用单个卷积核操作后输出图像的大小。请注意，由于在图像边界之外没有可用的像素值，因此我们无法在边界像素上应用卷积核（见图 5.4(b)）。因此，在应用卷积核时，我们必须在每个边界处跳过 $\bar{N}$ 个像素，从而得到形状为 $(N_1 - N + 1) \times (N_2 - N + 1)$ 的输出图像。

解决这个问题的一种方法是用每个边缘填充 $\bar{N}$ 个值为零的像素。现在，我们可以在边界像素上应用卷积核，输出图像将与输入图像大小相同，如图 5.4(c) 所示。

卷积的另一个特性称为步幅，它决定了卷积核在图像上移动时，每次移动的像素数。在上面的示例中，两个方向上的步幅都是 1。实际应用中，我们可以选择步幅 > 1，进一步缩小输出图像的尺寸。例如，如果两个方向上的步幅都设为 S（并进行零填充），那么输出图像在每个方向上的尺寸都会缩小为原来的 S（见图 5.4(d)）。



（a）卷积层的作用

（b）不进行填充的卷积

（c）采用零填充的卷积

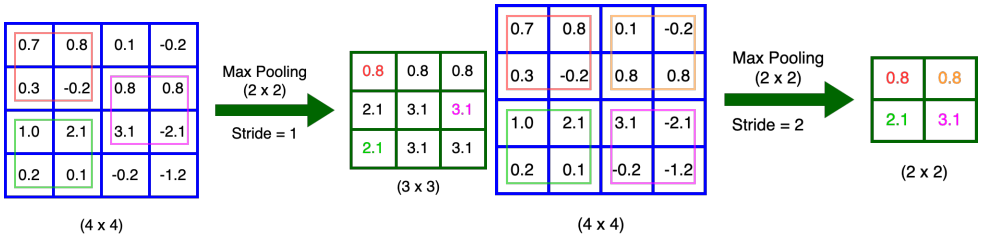
(d) 使用零填充且步幅为 2 的卷积

图 5.4：卷积层/卷积核的作用。

5.5.1 平均池化和最大池化

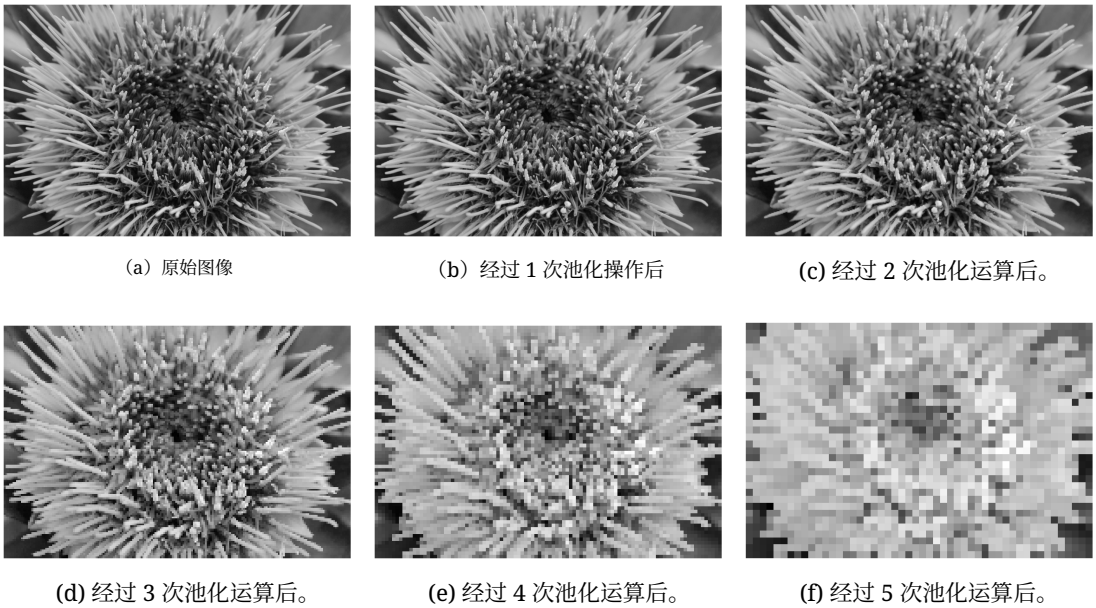
池化操作通常用于缩小图像尺寸，并让你能够以不同尺度遍历图像。若对尺寸为 $N \times N$ 的图像按大小为$S \times S$的图像块进行操作，新图像的维度将为 $\frac{N}{S} \times \frac{N}{S}$，其中 S 是池化操作的步幅。图 5.5 展示了 $S = 2$ 的情况。请注意，通常会选择像素块来计算最大值或平均值，其大小为 ($S \times S$)，其中 S 是步幅。图 5.5(b) 符合这一情况，但图 5.5(a) 不符合。

此外，我们在图 5.6 中展示了池化如何让我们遍历图像的各种尺度；随着应用更多池化操作，图像会变得更加粗糙。请注意，池化操作不包含任何可训练参数。池化操作与设计用于求解代数方程线性系统的多重网格预条件器时所使用的类似算子有很强的对应关系。



（a）步幅为 1　　（b）步幅为 2

图 5.5：对图像应用最大池化，图像块大小为 (2×2)。



（a）原始图像　（b）经过 1 次池化操作后　(c) 经过 2 次池化运算后。

(d) 经过 3 次池化运算后。　(e) 经过 4 次池化运算后。　(f) 经过 5 次池化运算后。

图 5.6　：在图像上反复应用最大池化，图像块大小为 (2×2)，步幅为　2.

5.5.2 多通道输入的卷积

假设卷积层的输入大小为 $N_1 \times N_2 \times C$，其中 C 是输入图像的通道数。那么，卷积层会对该输入应用 P 次卷积，并生成大小为 $M_1 \times M_2 \times P$ 的输出。请注意，输出图像的空间分辨率和通道数都可能与输入图像不同。此外，如果该层中的单个 ¯ 卷积使用宽度为 $k = 2N + 1$ 的卷积核，那么卷积核的形状将为 $k \times k \times C$，也就是说，对于输入图像的 C 个输入通道中的每一个，卷积核都包含 $k \times k$ 个权重。每次卷积都会作用于输入，并生成形状为 $M_1 \times M_2 \times 1$ 的输出图像。每次卷积的输出会堆叠在一起，从而得到卷积层的最终输出。该过程可以写成：

$$\bar{U}[i,j,k] = \sum_{m=-\bar{N}}^{\bar{N}} \sum_{n=-\bar{N}}^{\bar{N}} \sum_{c=1}^{C} g_k[m,n,c] U[i+m, j+n, c], \quad 1 \le i \le M_1,\ 1 \le j \le M_2,\ 1 \le k \le P,$$

其中 g_k 是该层第 k 个卷积的卷积核。请注意，可训练参数的总数为 $(2N+1) \times (2N+1) \times P \times C$。这是卷积神经网络中最常见的卷积层类型，下一节将对此进行介绍。

5.6 卷积神经网络（CNN）

现在，让我们将所有元素组合起来，构成完整的网络。考虑一个图像分类问题。此时，CNN 可表示为 $\boldsymbol{y} = \boldsymbol{\mathcal{F}}(\boldsymbol{x};\, \boldsymbol{\theta})$，其中 $\boldsymbol{x} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ 是具有 N_3 个通道的输入图像，而 $\boldsymbol{y} \in \mathbb{R}^C$ 是概率向量，其第 i 个分量表示输入图像属于总共 C 个类别中的第 i 个类别的概率。这些类别通常采用独热编码。

该网络的一种可能架构如图5.7所示。它由若干卷积层和后续的池化层组成，池化层会降低输入图像的空间分辨率，同时增加通道数。最后一个池化层的输出会被展平为一个向量，然后传递到若干具有某种激活函数（例如 ReLU）的全连接层。最后一个全连接层将向量的大小缩减至C （图中取值为 10），然后将其输入 Softmax 函数，以生成预测的概率向量 $\boldsymbol{y}$。由于我们要解决的是一个分类问题，因此损失函数采用交叉熵函数

$$\Pi(\boldsymbol{\theta}) = -\sum_{i=1}^{N_{\text{train}}} \sum_{c=1}^{C} \left[y_c^{(i)} \log \left(\mathcal{F}_c(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \right) \right].$$

我们训练 卷积神经网络，尝试求取 $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}}$ $\Pi(\boldsymbol{\theta})$，最终得到的网络为 $\boldsymbol{y} = \boldsymbol{\mathcal{F}}(\boldsymbol{x};\, \boldsymbol{\theta}$ *).

我们作出一些重要说明：

1. 卷积运算同样是对输入执行的线性函数，这一点与全连接层的情况相同。唯一的区别在于，在全连接层中，权重将输出中的每个像素与输入中的每个像素相连接；而在卷积层中，权重只将输出中的一个像素与输入中的一个像素块相连接。此外，输入的每个像素块都使用相同的权重。

2. 在图5.7所示的卷积神经网络中，卷积层可以理解为对输入图像的信息进行编码，而全连接层可以理解为

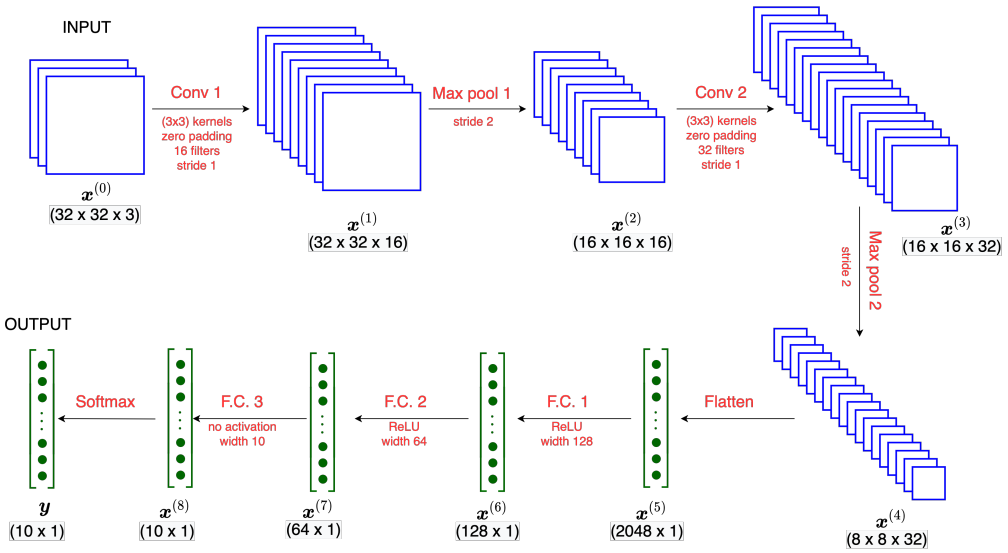


图 5. 7：用于 10 个类别的图像分类问题的 CNN 架构示例 .

利用这些信息解决分类问题。因此，文献中常将卷积层称为执行特征选择。此外，网络中通向“展平”向量的部分有时被称为编码器。

3. 有时激活函数也会与偏置

$$x^{(l+1)}[i,j,k] = \sigma(\sum_{m,n,c} g_k[m,n,c]x^{(l)}[i+m,j+n,c] + b_k)$$

其中，给定输出通道 k 时会使用单个偏置 b_k 。

4. 在上面的例子中，我们讨论了一个图像分类问题。也就是说，这个网络是一个从图像到类别标签的变换。我们还可以考虑其他类似的情况。例如，网络可以是一个从图像到实数的变换。这在计算物理中可能有多种实用应用。考虑这样一种情形：你想创建一个涡量平方积分计算器。也就是说，这个网络以定义在网格上的流体速度分量图像作为输入，并输出整个区域内涡量平方的积分（称为涡量平方积分）。另一个例子是，网络以弹性固体的位移分量作为输入，并输出固体内部储存的总应变能。

5. 我们讨论过的架构可以将图像变换为向量，这在涉及图像分类、图像分析等问题中很有用。不过，还有另一种架构可以执行相反的操作，即将向量映射为图像。这对于涉及图像合成的应用很有用。最后，将这两种架构结合起来，我们就可以将图像变换为向量，再将向量变换为另一幅图像。这种图像到图像变换在图像语义分割等应用中很有用。以下各节将介绍这些概念。

6. 值得花一点时间分析卷积层如何作用于图像，以及它们为何如此有用。在深度学习中处理图像输入时，最初一种简单的做法是将图像展平，然后输入到常规的全连接多层感知机中。

然而，这会带来不同的问题。事实上，对于常规图像，展平后的输入尺寸会非常大。在这种情况下，定义网络架构时有两种可能：

(a) 我们可以将网络前几层的宽度设置为与输入的（较大）维度相当。(b) 我们可以让第二个隐藏层的宽度急剧减小。

这两种策略都会带来问题。第一种情况下，网络规模会过大。因此，可训练参数会过多，需要不切实际的大量数据才能训练网络。第二种情况下，前两层之间的信息压缩会过于激进，导致网络很难学会如何从不同图像中提取合适的特征。此外，将图像展平会丢失像素之间重要的空间关系（如边缘、形状等）。理想情况下，我们希望尽可能利用这些关系，因为它们承载着重要的空间信息。卷积层可以解决这两个问题。它们允许输入为二维图像，同时大幅减少特征提取任务所需的可学习参数数量。事实上，与经典的全连接层相比，卷积核引入的参数数量有限。由于同一个卷积核会应用于图像中的不同像素位置，即 参数共享，因此能够高效、巧妙地利用计算资源。

5.7 转置卷积层

我们已经看到，卷积层和池化层可以用来缩小图像。现在我们来考虑作用相反的层，即放大图像的层。为了了解这种操作的效果，我们来看几个例子。

1. 考虑一个大小为 4 的一维图像

$$\text{Input} = \begin{bmatrix} u_1, & u_2, & u_3, & u_4 \end{bmatrix}$$

考虑一个大小为 3×1 的卷积核

$$\boldsymbol{k} = \begin{bmatrix} x, & y, & z \end{bmatrix}$$

考虑一个卷积层，其卷积核为 $\boldsymbol{k}$，步幅为 1，零填充大小为 1。该层作用于输入后，输出为

$$\text{Output} = \begin{bmatrix} yu_1 + zu_2, & xu_1 + yu_2 + zu_3, & xu_2 + yu_3 + zu_4, & xu_3 + yu_4 \end{bmatrix}$$

卷积算子的步骤包括：填充、点积、步幅。请注意，采用填充并将步幅设为 1，确保了输出与输入大小相同。

2. 考虑另一个卷积，其卷积核为 $\boldsymbol{k}$，采用零填充层 1，但步幅为 2。那么，该层作用于与前面相同的输入时，输出为

$$\text{Output} = \begin{bmatrix} yu_1 + zu_2, & xu_2 + yu_3 + zu_4 \end{bmatrix}$$

请注意，输出的尺寸缩小了一半。换句话说，增大步幅使我们能够对输入进行下采样。我们想要问的是，能否用类似的方法进行上采样？确实可以，只需将卷积层中的每个操作转置即可。

- 我们将使用外积，而不是点积（内积）。
- 我们不再跳过输入中的像素（步幅），而是跳过输出中的像素。
- 我们不需要填充，而是需要裁剪输出。

3. 现在来看一个转置卷积层的示例。考虑一个大小为 2×1 的一维输入图像

$$\text{Input} = \begin{bmatrix} u_1, & u_2 \end{bmatrix}$$

以及一个大小为 3×1 的卷积核

$$\boldsymbol{k} = \begin{bmatrix} x, & y, & z \end{bmatrix}.$$

如果我们将输入与 $\boldsymbol{k}$ 做外积，就会得到

$$\text{Outer product} = \begin{bmatrix} u_1 x & u_1 y & u_1 z \\ u_2 x & u_2 y & u_2 z \end{bmatrix}.$$

如果步幅为 2，我们需要将外积的各行移动 2

$$\text{Striding} = \begin{bmatrix} u_1 x & u_1 y & u_1 z & 0 & 0 \\ 0 & 0 & u_2 x & u_2 y & u_2 z \end{bmatrix}.$$

完成步幅操作后，我们需要将每一列中的各项相加，并裁剪该向量以获得输出

$$\text{输出} = \text{裁剪}(\begin{bmatrix} u_1 x, & u_1 y, & u_1 z + u_2 x, & u_2 y, & u_2 z \end{bmatrix}) = \begin{bmatrix} u_1 x, & u_1 y, & u_1 z + u_2 x, & u_2 y \end{bmatrix}$$

其中，按照惯例，我们裁剪掉最后几个元素，以获得大小为输入的 2 倍的输出。

4. 我们考虑将二维转置卷积应用于大小为 (2×2) 的二维图像。卷积核的形状取为 (3×3)，步幅为 2，并采用填充（裁剪）。该转置卷积的作用如图 5.8(a) 所示：我们首先得到大小为 (5×5) 的图像，然后对其进行裁剪，以得到大小为 (4×4) 的输出。请注意，各个图像块对输出像素的贡献并不相等（图中以数字表示），这会导致棋盘格效应，而这是不理想的。棋盘格效应是指输出图像中像素值在像素之间出现变化。

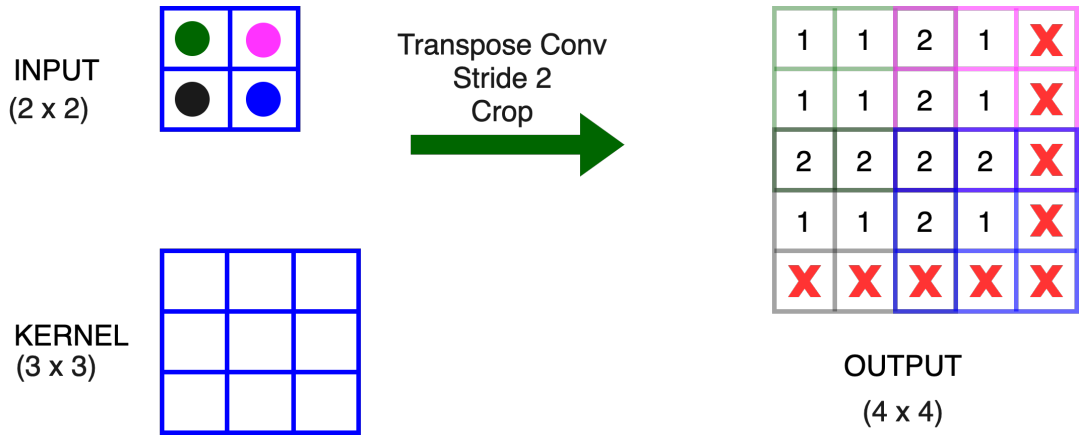
5. 避免棋盘格效应的一种方法，是确保滤波器尺寸为步幅的整数倍。让我们重复前面的例子，但使用一个 (2×2) 卷积核。该操作如图 5.8(b) 所示。在这种情况下，我们不需要进行填充（裁剪），并且每个输出像素的贡献都相同。

我们提出以下几点说明：

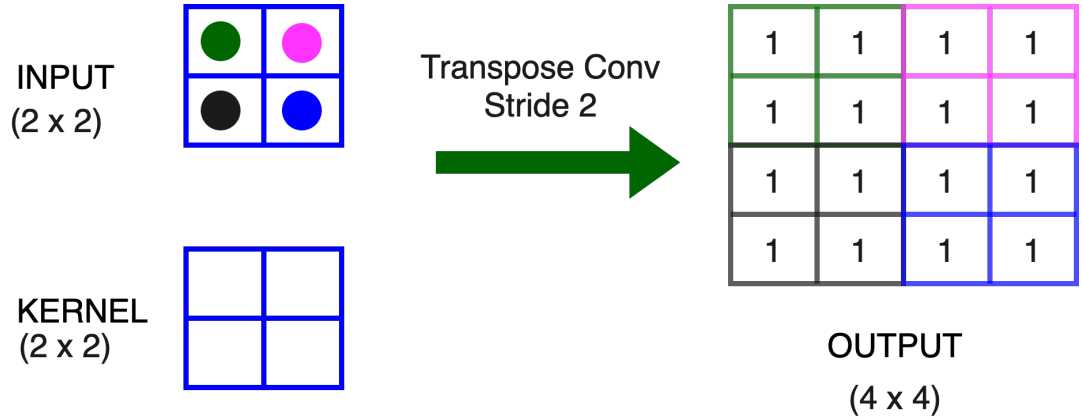
1. 转置卷积层也称为分数步长层，因为输入每前进一步，输出会前进大于一步。这与卷积层中的情况相反。在卷积层中，输入图像每前进大于一步，输出图像才前进一步。

2. 转置卷积是通过可学习参数进行上采样的首选工具。

3. 上采样通常会伴随通道数的减少，这与卷积层中的做法正好相反。



(a) 卷积核大小 (3×3)，存在棋盘格效应



(b) 卷积核大小 (2×2)，不存在棋盘格效应

图 5.8：转置卷积操作示例。输出中标有红色 X 的单元格会被裁剪掉。数字表示对输出像素的值有贡献的图像块数量。

5.8 图像到图像变换

图像到图像变换可以类比于函数到函数变换。这类网络通常用于计算机视觉、超分辨率、风格迁移，以及计算物理领域，例如将偏微分方程的源（右端项）场映射到解。

我们将讨论一种用于此类变换的特定网络，即 U-Net [26]。在 U-Net 中（见图 5.9），有一个向下分支，它接收输入图像，并通过若干卷积层和池化操作对图像进行下采样。沿着该分支向下时，通道数通常会增加。到达最粗尺度后，我们会进入向上分支，该分支对图像进行上采样，并通过转置卷积类操作减少通道数，最终得到输出图像。除了这两个分支，U-Net 还利用跳跃连接，将向下分支中特定尺度的信息与向上分支同一尺度的信息相结合。这些连接与残差网络中使用的连接类似。在 U-Net 的上采样分支中，如果考察某一点的激活函数，会发现它们来自两个不同的来源：一个来自 U-Net 下采样分支中的相同空间尺度，另一个来自 U-Net 上采样分支中更粗的尺度。

注释 5.8.1. *U-Net*架构与多重网格预条件器中通常使用的*V*循环有许多共同特征。

注记 5.8.2. 我们也可以将 *U-net* 视为一种编码器-解码器网络，其额外特点是包含跳跃连接。

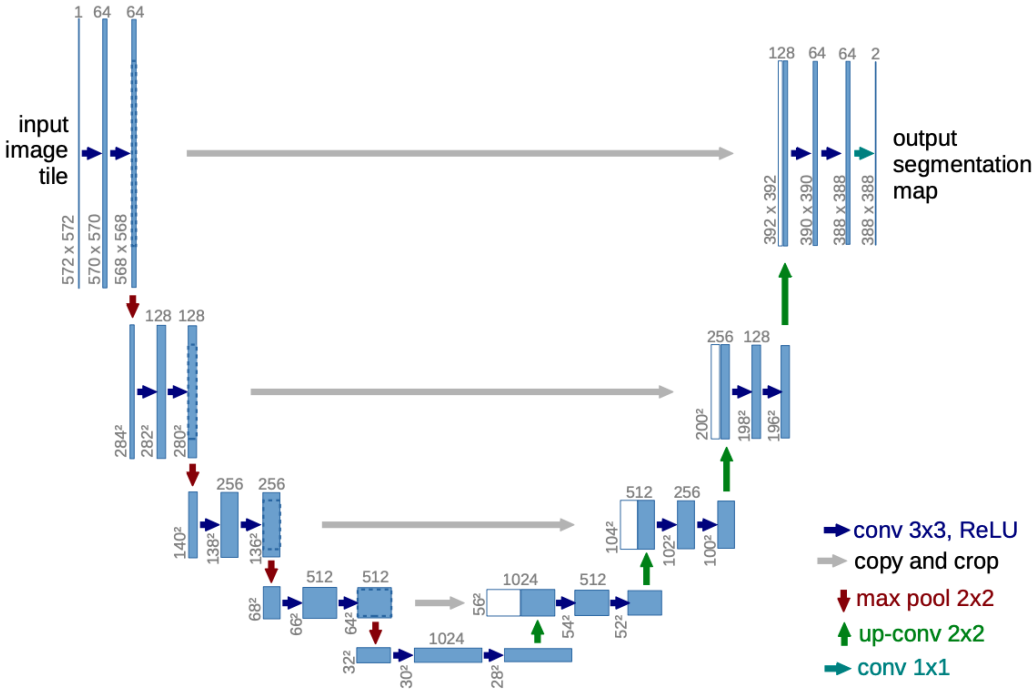


图 5.9：取自 [26]的 U-Net 示例。

第6章

算子网络

6.1 物理信息神经网络的问题

回顾一下，典型的多层感知机 $\boldsymbol{y} = \boldsymbol{\mathcal{F}}(\boldsymbol{x};\ \boldsymbol{\theta})$ 是一个以 $\boldsymbol{x} \in \mathbb{R}^d$ 为输入、给出 $\boldsymbol{y} \in \mathbb{R}^d$ 作为输出且具有可训练权重 $\boldsymbol{\theta}$的函数。此外，正如我们在第4章所讨论的，物理信息神经网络是一种形式为 $\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{\mathcal{F}}(\boldsymbol{x};\ \boldsymbol{\theta})$ 的网络，它以底层偏微分方程的独立变量 $\boldsymbol{x}$ 为输入，并输出（该偏微分方程的）解 $\boldsymbol{u}(\boldsymbol{x})$。该网络通过最小化偏微分方程残差与边界残差的加权和进行训练。不过，这只是针对某个给定边界条件和源项求解偏微分方程的一种情形。例如，如果我们考虑偏微分方程

$$\begin{aligned} \nabla \cdot (\kappa \nabla u) &= f(x), \quad \boldsymbol{x} \in \Omega = [0,1] \times [0,1] \\ u(x) &= g(x), \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{6.1}$$

并训练一个物理信息神经网络 $\mathcal{F}(\boldsymbol{x};\ \boldsymbol{\theta})$，以最小化损失函数

$$\Pi(\boldsymbol{\theta}) = \frac{1}{N_v} \sum_{i=1}^{N_v} |\nabla \cdot (\kappa \nabla \mathcal{F}(\boldsymbol{x}_i; \boldsymbol{\theta})) - f(\boldsymbol{x}_i)|^2 + \frac{\lambda_b}{N_b} \sum_{i=1}^{N_b} |\mathcal{F}(\boldsymbol{x}_i; \boldsymbol{\theta}) - g(\boldsymbol{x}_i)|^2$$

那么，如果 $\boldsymbol{\theta}^* = \arg\min \Pi(\boldsymbol{\theta})$，求解 (6.1) 的物理信息神经网络将是 $u(\boldsymbol{x}) = \mathcal{F}(\boldsymbol{x};\ \boldsymbol{\theta}^*)$。然而，如果我们$\boldsymbol{\theta}$改变 (6.1) 中的 f 或 g，就没有理由认为同一个训练后的网络仍然有效。事实上，对于新的f 和 g，我们需要重新训练该网络（或许使用相同的架构）。这样做可能相当繁琐，我们理想情况下希望避免这种情况。在本章中，我们将介绍克服这一问题的方法。

6.2 参数化偏微分方程

假设式 (6.1) 中的源项 f 是一个参数化函数 $f(\boldsymbol{x};\ \alpha)$。例如，我们可以有

$$f(x_1, x_2; \alpha) = 4\alpha x_1(1 - x_1)x_2(1 - x_2)$$

然后，我们可以训练一个物理信息神经网络，通过考虑一个同时将 $\boldsymbol{x}$ 和 α作为输入的网络来适应参数化，即 $\mathcal{F}(\boldsymbol{x}, \alpha;\ \boldsymbol{\theta})$。这如图6.1所示。该网络可以通过最小化损失函数来训练

$$\Pi(\boldsymbol{\theta}) = \frac{1}{N_a} \sum_{j=1}^{N_a} \left[\frac{1}{N_v} \sum_{i=1}^{N_v} |\nabla \cdot (\kappa \nabla \mathcal{F}(\boldsymbol{x}_i, \alpha_j; \boldsymbol{\theta})) - f(\boldsymbol{x}_i, \alpha_j)|^2 + \frac{\lambda_b}{N_b} \sum_{i=1}^{N_b} |\mathcal{F}(\boldsymbol{x}_i, \alpha_j; \boldsymbol{\theta}) - g(\boldsymbol{x}_i)|^2 \right]$$

请注意，构建损失函数时，我们还必须考虑参数 α 的配置点。如果 $\boldsymbol{\theta}^* = \arg\min\Pi(\boldsymbol{\theta})$，那么参数化偏微分方程的解将是 $\boldsymbol{\theta}$ $u(\boldsymbol{x}, \alpha) = \mathcal{F}(\boldsymbol{x}, \alpha; \boldsymbol{\theta}^*)$。此外，对于 $\alpha = \hat{\alpha}$ 的任意新取值，我们都可以通过计算 $\mathcal{F}(\boldsymbol{x}, \hat{\alpha}; \boldsymbol{\theta}^*)$ 来求得解。如果有办法对函数 $\kappa(\boldsymbol{x})$ 和 $g(\boldsymbol{x})$进行参数化，我们也可以采用相同的方法。

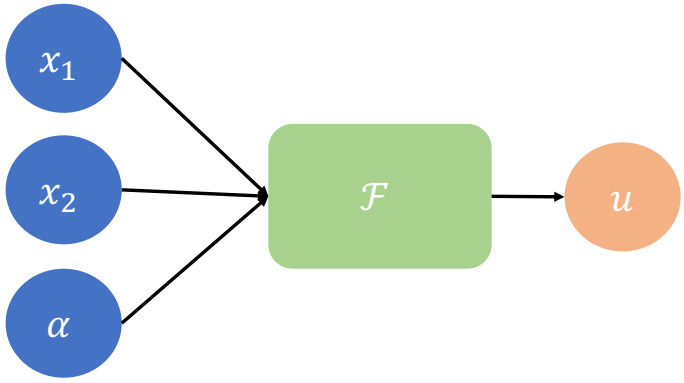


图6.1：具有参数化输入的物理信息神经网络示意图。

然而，如果我们想要得到任意非参数化 f 对应的解，该怎么办？为此，我们需要找到一种方法来近似**将函数映射为函数的算子**。

6.3 算子

考虑一个函数类 $\boldsymbol{a}(\boldsymbol{y}) \in A$，满足 $\boldsymbol{a}: \Omega_Y \to \mathbb{R}^D$。该函数类中的函数可能具有某些性质，例如 $\boldsymbol{a} \in C(\Omega_Y)$ 或 $\boldsymbol{a} \in L^2(\Omega_Y)$。再考虑算子$\mathcal{N}$: $A \mapsto C(\Omega_X)$，其中 $\boldsymbol{u}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{a})(\boldsymbol{x})$，对 $\boldsymbol{x} \in \Omega_X$ 均成立。下面来看一些算子的例子 $\mathcal{N}$。

1. 考虑偏微分方程

$$\begin{aligned} \nabla \cdot (\kappa \nabla u) &= f, \quad \boldsymbol{x} \in \Omega \\ u &= g, \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{6.2}$$

对于这个偏微分方程，$\Omega_X = \Omega_Y = \Omega$，算子 $\mathcal{N}$ 将右端项 f 映射为该偏微分方程的解（温度） u。也就是说， $u = \mathcal{N}(f)(\boldsymbol{x})$。算子的输入和输出由上面的方程联系起来，其中假设 κ 和 g 已知且固定。

2. 考虑偏微分方程

$$\begin{aligned} \nabla \cdot (\kappa \nabla u) &= f, \quad \boldsymbol{x} \in \Omega \\ u &= g, \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{6.3}$$

这与前一个偏微分方程相同，但我们假设模型中的传导率场 κ可能会变化，而不是右端项。于是，$\Omega_X = \Omega_Y = \Omega$，算子 $\mathcal{N}$将电导率 κ 映射为该偏微分方程的解 u。也就是说，$u = \mathcal{N}(\kappa)(\boldsymbol{x})$。算子的输入和输出由上面的方程关联，其中假设$f$ 和 g 是已知且固定的。

3. 再次考虑同一个偏微分方程，但允许电导率和边界条件发生变化

$$\begin{aligned} \nabla \cdot (\kappa \nabla u) &= f(x), \quad \boldsymbol{x} \in \Omega \\ u(x) &= g(x), \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{6.4}$$

那么，算子 $\mathcal{N}$ 将边界条件 g 和电导率 κ 映射为偏微分方程的解u。也就是说，$u = \mathcal{N}(\kappa, g)(\boldsymbol{x})$。在这种情况下，算子的输入是两个函数（$g$，$\kappa$），输出是一个函数。因此，$\Omega_X = \Omega$，而 $\Omega_Y = \Omega \times \partial\Omega$。输入和输出通过上述偏微分方程的解相联系，其中假设 f 已知且固定。

4. 现在考虑定义在三维区域 $\Omega \subset \mathbb{R}^3$上的线性各向同性弹性方程，

$$\begin{aligned} \nabla \cdot \big(\lambda(\nabla \cdot \boldsymbol{u})\boldsymbol{I} + 2\mu\nabla^S(\boldsymbol{u})\big) &= \boldsymbol{f}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \\ \boldsymbol{u}(\boldsymbol{x}) &= \boldsymbol{g}(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega. \end{aligned} \tag{6.5}$$

考虑由 $\boldsymbol{u}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{f})(\boldsymbol{x})$ 定义的算子。这里，输入函数 $\boldsymbol{f}: \Omega \to \mathbb{R}^3$，输出函数 $\boldsymbol{u}: \Omega \to \mathbb{R}^3$。二者通过上述方程相联系，其中$\lambda$，$\mu$，$\boldsymbol{g}$已知且固定。

5. 现在考虑另一个偏微分方程。具体来说，是对流-扩散-反应方程，

$$\begin{aligned} \frac{\partial u}{\partial t} + \boldsymbol{a} \cdot \nabla u - \kappa\nabla^2 u + u(1-u) &= f, \quad (\boldsymbol{x}, t) \in \Omega \times (0, T] \\ u(\boldsymbol{x}, t) &= g(\boldsymbol{x}, t), \quad (\boldsymbol{x}, t) \in \partial\Omega \times (0, T] \\ u(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega. \end{aligned} \tag{6.6}$$

我们希望找到将初始条件 u_0 映射到终止时间 T处解 u 的算子 $\mathcal{N}$，即 $u(\boldsymbol{x}, T) = \mathcal{N}(u_0)(\boldsymbol{x})$。在这种情况下，$\Omega_X = \Omega_Y = \Omega$。此外，输入函数和输出函数通过上述偏微分方程的解彼此相关，其中 $\boldsymbol{a}$，κ，f，g均为给定且固定的。

注释 6.3.1。判断一个算子是线性还是非线性通常很有用。这是因为如果它是线性的，就可以用另一个线性算子很好地近似。在上述情形中，例 *1* 和例 *4* 中的算子是线性的，而例 *2*、*3* 和 *5* 中的算子是非线性的。

我们关注的是能够近似算子 $\mathcal{N}$的网络。下一节将介绍如何实现这一点。这类网络通常称为算子网络。这类网络有两个常见的版本：一种称为深度算子网络，即 DeepONet；另一种称为傅里叶神经算子。下一节将介绍 DeepONet。

6.4 DeepOperatorNetwork (DeepONet) 架构

算子网络最初由 Chen and Chen [5], 提出，他们只考虑了具有单个隐藏层的浅层网络。最近，这一思想被重新发现，并扩展到深层架构 [14] ，这些网络被称为 DeepONets。标准的 DeepONet 由两个神经网络组成。下面我们介绍其构造方法，用于近似算子 $\mathcal{N}: A \to U$，其中A 是由以下形式的函数构成的集合 $a: \Omega_Y \subset \mathbb{R}^d \to \mathbb{R}$，而 U 由以下形式的函数构成$u: \Omega_X \subset \mathbb{R}^D \to \mathbb{R}$。此外，我们假设这两个函数类都可以进行逐点求值。用于近似该算子的 DeepONet 架构如图 6.2 所示。下面将对此进行说明：

- 固定 M 个互不相同的传感器点 $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(M)}$，它们位于 Ω_Y 中。

- 在这些传感器点处对函数 $a \in A$ 采样，得到向量 $\boldsymbol{a} = [a(\boldsymbol{y}^{(1)}), \ldots, a(\boldsymbol{y}^{(M)})]^\top \in \mathbb{R}^M$。

- 将 $\boldsymbol{a}$ 作为输入提供给一个子网络，称为分支网络 $\mathcal{B}(.; \boldsymbol{\theta}_B) : \mathbb{R}^M \to \mathbb{R}^p$，其输出为向量 $\boldsymbol{\beta} = [\beta_1(\boldsymbol{a}), \ldots, \beta_p(\boldsymbol{a})]^\top \in \mathbb{R}^p$。此处 $\boldsymbol{\theta}_B$ 是分支网络的可训练参数。分支的输出维度相对较小，例如 $p \approx 100$。

- 将 $\boldsymbol{x}$ 作为输入提供给第二个子网络，称为主干网络 $\mathcal{T}(.; \boldsymbol{\theta}_T) : \mathbb{R}^D \to \mathbb{R}^p$，其输出为向量 $\boldsymbol{\tau} = [\tau_1(\boldsymbol{x}), \ldots, \tau_p(\boldsymbol{x})]^\top \in \mathbb{R}^p$。此处 $\boldsymbol{\theta}_T$ 是主干网络的可训练参数。

- 对分支网络和主干网络的输出求点积，得到 DeepONet 的最终输出 $\widetilde{\mathcal{N}}(., .; \boldsymbol{\theta}) : \mathbb{R}^D \times \mathbb{R}^M \to \mathbb{R}$，它将近似于 $u(\boldsymbol{x})$ 的值

$$u(\boldsymbol{x}) \approx \widetilde{\mathcal{N}}(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) = \sum_{k=1}^{p} \beta_k(\boldsymbol{a}) \tau_k(\boldsymbol{x}). \tag{6.7}$$

其中，DeepONet 的可训练参数是分支网络和主干网络的参数组合，即 $\boldsymbol{\theta} = [\boldsymbol{\theta}_T, \boldsymbol{\theta}_M]$。

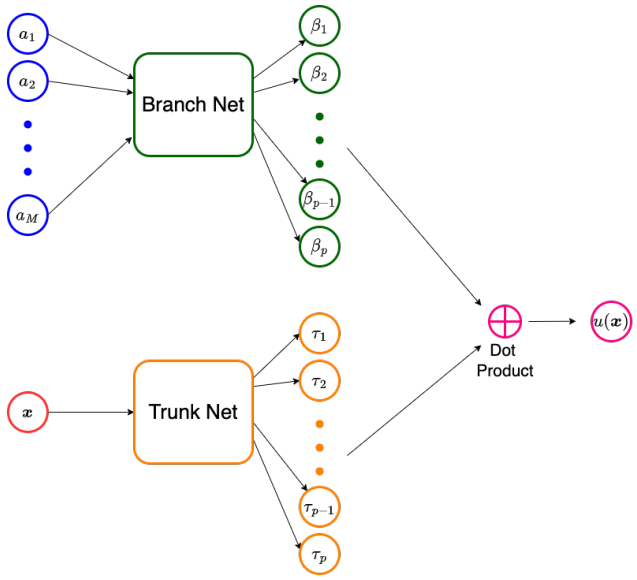


图 6.2：DeepONet 示意图

在上述构造中，DeepONet 训练完成后（我们将在下一节讨论训练），它将近似底层算子 $\mathcal{N}$，使我们能够近似任意 $a \in A$ 和任意 $\boldsymbol{x} \in \Omega_X$ 下的任意 $\mathcal{N}(a)(\boldsymbol{x})$ 的值。请注意，在构造 DeepONet 时，M 传感器点必须预先定义，并且在训练和评估阶段不能更改。

关于 DeepONet 架构，我们可以作出以下观察：

1. 式 (6.7) 的形式是将解表示为一系列系数与函数之和。系数由分支网络确定，而

函数则由主干网络确定。从这个意义上说，DeepONet 的构造与谱方法或有限元方法中使用的构造类似。不过，两者有一个关键区别：在这些方法中，基函数是预先确定的，由用户选取。然而，在 DeepONet 中，这些函数由主干网络确定，其最终形式取决于用于训练 DeepONet 的数据。

2. 分支子网络的架构：当随机选择输入函数的采样点时，分支网络适合采用全连接层。进一步考虑到该网络的输入维度可能相当大 $N_1 \approx 10^4$，而输出通常较小 $p \approx 10^2$，因此可以将该网络视为编码器。

当在均匀网格上选择输入函数的采样点时，分支网络适合采用卷积层。在这种情况下，该网络会将大维度的图像（$N_1 \approx 10^4$）映射为小维度的潜向量，$p \approx 10^2$。因此，卷积神经网络是最合适的表示形式。

3. 广义而言，有两种方法可以提升 DeepONet 的表达能力：增加分支子网络和主干子网络中的网络参数数量，以及增加这些子网络所形成的潜向量的维度 p。

6.5 DeepONet 训练

DeepONet 的训练通常采用监督式学习，并需要成对数据。主要步骤如下：

1. 从集合 A 中选取 N_1 代表性函数 $a^{(i)}, 1 \le i \le N_1$。在 M 传感器点处计算这些函数的值，即对 $1 \le j \le M$，计算 $a_j^{(i)} = a^{(i)}(\boldsymbol{y}^{(j)})$。这样，对于每个 $1 \le i \le N_1$，我们便得到向量 $\boldsymbol{a}^{(i)} = [a^{(i)}(\boldsymbol{y}^{(1)}) \ldots\ldots, a^{(i)}(\boldsymbol{y}^{(M)})]^\top \in \mathbb{R}^M$。

2. 对于每个 $a^{(i)}$，确定算子 $\mathcal{N}$ 给出的相应函数 $u^{(i)}$（通过数值方法或解析方法）。

3. 在 Ω_X 中的 N_2 个点处对函数 $u^{(i)}$ 进行采样，即对 $1 \le k \le N_2$，计算 $u^{(i)}(\boldsymbol{x}^{(k)})$。

4. 构建训练集

$$\mathcal{S} = \left\{ \left(\boldsymbol{a}^{(i)}, \boldsymbol{x}^{(k)}, u^{(i)}(\boldsymbol{x}^{(k)}) \right) : 1 \le i \le N_1,\ 1 \le k \le N_2 \right\}$$

其中将有 $N_1 \times N_2$ 个样本。

5. 定义损失函数

$$\Pi(\boldsymbol{\theta}) = \frac{1}{N_1 N_2} \sum_{i=1}^{N_1} \sum_{k=1}^{N_2} |\widetilde{N}(\boldsymbol{x}^{(k)}, \boldsymbol{a}^{(i)}; \boldsymbol{\theta}) - u^{(i)}(\boldsymbol{x}^{(k)})|^2.$$

6. 训练 DeepONet 相当于求解 $\boldsymbol{\theta}^* = \operatorname{argmin} \Pi(\boldsymbol{\theta})$。$\boldsymbol{\theta}$

7. 训练完成后，给定传感器点处的任意新样本 $a \in A$ M（由此得到向量 $\boldsymbol{a} \in \mathbb{R}^M$）以及一个新点 $\boldsymbol{x} \in \Omega_X$，我们就可以计算出相应的预测 $\sim u^*(\boldsymbol{x}) = N(\boldsymbol{x}, \boldsymbol{a};\ \boldsymbol{\theta}^*)$。

注 6.5.1. 训练集中所有 i 点不必都选取相同的 N_2 点。事实上，可以随机选取这些点，从而得到更多样化的数据集。

备注 6.5.2。 *DeepONet* 可以很容易地扩展到输入包含多个函数的情况。在这种情况下，主干网络保持不变，但分支网络现在以多个向量作为输入。对应于两个输入函数 $a(\boldsymbol{y})$ 和 $b(\boldsymbol{y})$的情况，对它们进行采样后得到向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$，如图 *6.3* 所示。

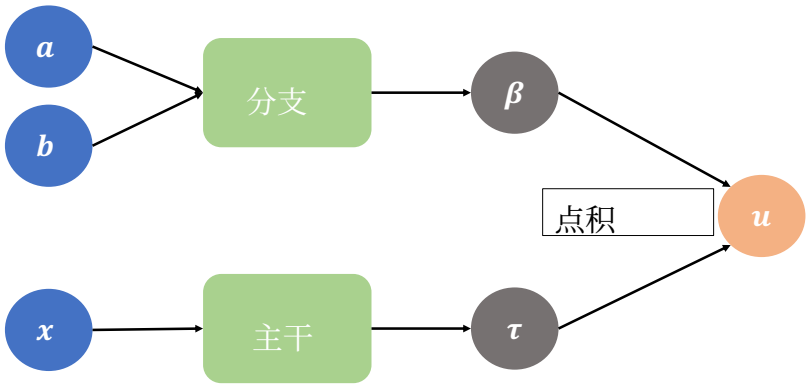


图 6.3：包含两个输入函数的 DeepONet 示意图。

备注 6.5.3。 *DeepONet* 可以很容易地扩展到输出包含多个函数（比如 D 个函数）的情况。在这种情况下，分支网络和主干网络的输出分别得到 D 个向量，每个向量的维度为 p。然后，通过计算这些向量两两之间的点积来获得解。对应于两个输出函数 $u_1(\boldsymbol{x})$和 $u_2(\boldsymbol{x})$的情况如图 *6.3* 所示。

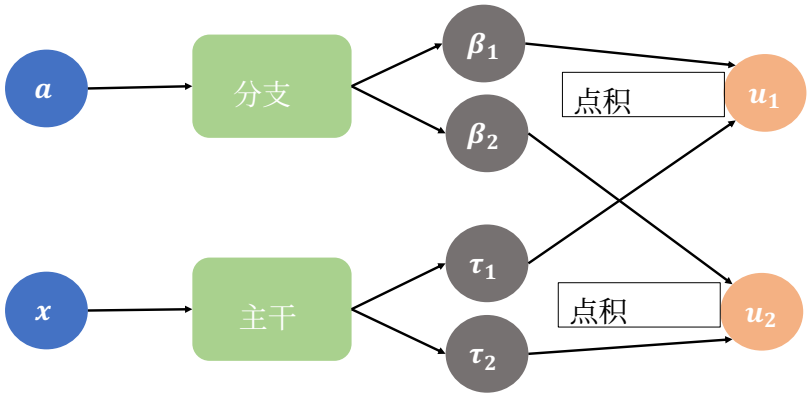


图 6.4：包含两个输出函数的 DeepONet 示意图。

6.6 DeepONets 的误差分析

Chen and Chen [5] 给出了浅层版本 DeepONets 的通用逼近定理

定理 6.6.1。 假设 Ω_X 和 Ω_Y 分别是 $\mathbb{R}^D$ （或更一般地，巴拿赫空间）和 $\mathbb{R}^d$中的紧集。设 $\mathcal{N}$ 是一个非线性连续算子，将 $V \subset C(\Omega_Y)$ 映射到$C(\Omega_X)$。那么，给定 $\epsilon > 0$，存在一个 ***DeepONet***，其分支网络和主干网络包含 M 个传感器以及一个宽度为 P 的单隐藏层，使得

$$\max_{\substack{\boldsymbol{x}\in\Omega_X \\ a\in A}} |\widetilde{\mathcal{N}}(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) - \mathcal{N}(a)(\boldsymbol{x})| < \epsilon$$

对于足够大的 P 和 M。

在 [14], 中，这一结果已扩展到更深层的网络版本，并在 [12] 中通过移除空间上的紧致性假设得到了进一步推广。

最近，在 [20], 中，作者提出了 DeepONet 的误差估计，该估计清楚地指出了 DeepONet 中误差的不同来源。该估计表明，以 $L^2(\Omega)$ 范数衡量的误差满足以下界：

$$\max_{a\in A} \|\widetilde{\mathcal{N}}(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta}) - \mathcal{N}(a)(\boldsymbol{x})\|_{L^2(\Omega)} \leq C\left(\epsilon_h + \sqrt{\epsilon_t} + \epsilon_s + M^{-\alpha_1} + (N_2)^{-\alpha_2}\right) \tag{6.8}$$

其中， ϵ_h 是用于生成训练集中的近似目标解 $u^{(i)}$ 的数值求解器所产生的误差（相对于精确解而言）； ϵ_t 是最终达到的训练误差/损失；而 ϵ_s 则界定任意 $a \in A$ 到用于构建训练集的函数集合 $\{a^{(i)}\}_{i=1}^{N_1}$ 的距离，也就是说，它估计了训练样本对输入空间 A 的覆盖程度。此外，由于输入函数是在 M 个有限传感器节点处求值，而输出是在 N_2 个输出节点处求值，因此会产生额外的离散化（或求积）误差，对应于式 (6.8) 中最后两项。请注意，这与式 (4.21) 中针对物理信息神经网络得到的误差估计类似。

6.7 物理信息 DeepONet

~回顾一下，DeepONet 近似于 $u(\boldsymbol{x}) = \mathcal{N}(a)(\boldsymbol{x}) \approx N(\boldsymbol{x}, \boldsymbol{a}; \boldsymbol{\theta})$。假设 a和u 满足一个偏微分方程。例如，

$$\begin{aligned} \nabla \cdot (\kappa u) &= f \text{ in } \Omega \\ u &= g \text{ on } \partial\Omega \end{aligned} \tag{6.9}$$

其中， κ 和 g 是给定的。为了构造将 f 映射到 u的算子 $\mathcal{N}$，我们需要在外部求解偏微分方程。不过，除此之外，我们还可以使用 PINN 型损失函数，并将其加入总损失函数。这就是文献 [30]提出的物理信息 DeepONet 的思路。因此，对于上述模型偏微分方程，额外的基于物理的损失为：

$$\Pi_p(\boldsymbol{\theta}) = \frac{1}{\bar{N}_1}\frac{1}{\bar{N}_2}\sum_{i=1}^{\bar{N}_1}\sum_{k=1}^{\bar{N}_2}\left|\nabla_{\boldsymbol{x}} \cdot \left(\kappa\nabla_{\boldsymbol{x}}\widetilde{\mathcal{N}}(\boldsymbol{x}^{(k)}, \boldsymbol{f}^{(i)}; \boldsymbol{\theta})\right) - f^{(i)}(\boldsymbol{x}^{(k)})\right|^2. \tag{6.10}$$

此外，还有标准的数据驱动损失函数；对于本例，它定义为：

$$\Pi_d(\boldsymbol{\theta}) = \frac{1}{N_1 N_2}\sum_{i=1}^{N_1}\sum_{k=1}^{N_2}|\widetilde{N}(\boldsymbol{x}^{(k)}, \boldsymbol{f}^{(i)}; \boldsymbol{\theta}) - u^{(i)}(\boldsymbol{x}^{(k)})|^2. \tag{6.11}$$

总损失函数是这两项的加权和：

$$\Pi(\boldsymbol{\theta}) = \Pi_d(\boldsymbol{\theta}) + \lambda\Pi_p(\boldsymbol{\theta}), \tag{6.12}$$

其中， λ 是一个超参数。以下几点需要说明：

1. 基于物理的损失函数中使用的输出传感器点，通常不同于数据驱动损失项中使用的输出传感器点。前者表示我们希望最小化偏微分方程残差的位置，后者则表示通过外部方式获得解的点。基于物理的损失函数部分中输出传感器点的总数记为 N_2，而在数据驱动损失函数中则记为 N_2。

2. 用于构造基于物理的损失函数的输入函数集合，通常不同于用于构造数据驱动损失函数的输入函数集合。前者表示我们希望使偏微分方程残差最小化的函数集合，而后者表示我们通过外部途径获得其解的输入函数集合。用于构造损失函数中基于物理部分的函数总数记为 N_1，而用于构造损失函数中数据驱动部分的函数总数记为 $N-1$。

与前面一样，我们通过求解 $\boldsymbol{\theta}^* = \arg\min \Pi(\boldsymbol{\theta})$ 来训练网络，并对新的 a，用 $u^*(\boldsymbol{x}) = N(\boldsymbol{x}, \boldsymbol{a};\ \boldsymbol{\theta}^*)$ 近似解 $\widetilde{\boldsymbol{\theta}}$。添加额外的基于物理的损失具有以下优点：

1. 它降低了数据驱动损失项对数据量的需求。这意味着我们不必生成那么多偏微分方程的解来训练 DeepONet。

2. 这使网络更加稳健，因为它更有可能针对训练集中的数据驱动损失项未涵盖的输入函数类型，给出准确的解。

6.8 傅里叶神经算子 - 架构

现在，我们来介绍并讨论傅里叶神经算子（FNO）[13]。本节将讨论其架构，下一节再讨论其他方面。

我们构建 FNO 架构的方法是，先从典型的前馈多层感知机架构入手，该网络将一个标量映射为另一个标量，然后系统地对其进行扩展，使扩展后的版本能够将一个标量值函数映射为另一个标量值函数。

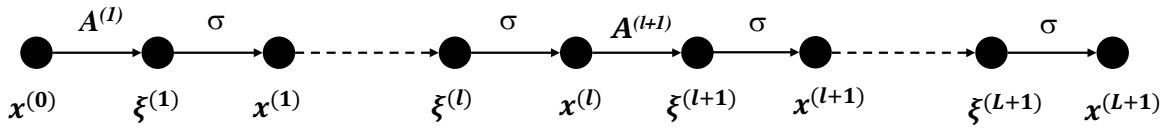


图 6.5：前馈多层感知机的计算图。

在图 6.5 中，我们绘制了一个多层感知机的计算图。我们只关注该网络的前向部分（而非反向传播部分）。为简单起见，我们假设输入 $\boldsymbol{x}^{(0)} = x$ 是一个标量，输出 $\boldsymbol{x}^{(L)} = y$ 也是一个标量。此外，所有其他隐藏变量（ $\boldsymbol{\xi}^{(L+1)}$ 除外）都是包含 H 个分量的向量。也就是说，每一层的宽度都是 H。

这一过程的第一步是将输入和输出替换为函数。现在，输入将是函数 $a(\boldsymbol{x}) : \Omega \mapsto \mathbb{R}^1$。同样，输出是函数 $u(\boldsymbol{x}) : \Omega \mapsto \mathbb{R}^1$。由此得到图 6.6 所示的计算图。

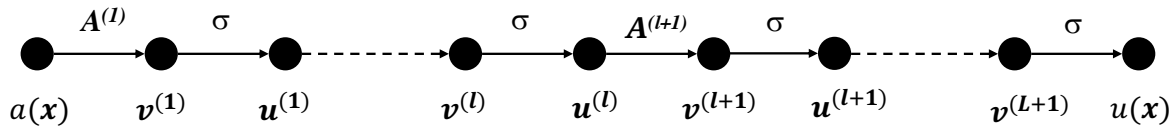


图 6.6：前馈傅里叶神经算子（FNO）网络的计算图。

下一步是考虑隐藏层中的变量。在多层感知机中，这些变量都是具有 H 个分量的向量。在傅里叶神经算子中，它们将是具有 H 个分量的函数。也就是说，

$$\boldsymbol{v}^{(1)}, \cdots, \boldsymbol{v}^{(L)}, \boldsymbol{u}^{(1)}, \cdots, \boldsymbol{u}^{(L)} : \Omega \mapsto \mathbb{R}^H. \tag{6.13}$$

如图 6.6 所示， $\boldsymbol{v}^{(n)}$ 和 $\boldsymbol{u}^{(n)}$ 分别对应于 $\boldsymbol{\xi}^{(n)}$ 和 $\boldsymbol{x}^{(n)}$。此外，由于 $\boldsymbol{\xi}^{(L+1)}$ 是一个标量，我们将 $\boldsymbol{v}^{(L+1)}$ 设为标量值函数。

现在，我们已经完成了将输入、输出以及隐藏层中的变量从向量扩展为函数。接下来，我们需要将多层感知机（MLP）中把向量变换为向量的算子，扩展为傅里叶神经算子（FNO）中把函数变换为函数的算子。

我们从算子 $A^{(1)}$开始。在多层感知机中，它是一个仿射映射，将一个分量的向量映射为具有 H 个分量的向量。将其直接扩展到函数的形式为：

$$\boldsymbol{v}^{(1)}(\boldsymbol{x}) = A^{(1)}(a)(\boldsymbol{x}), \tag{6.14}$$

其中

$$v_i^{(1)}(\boldsymbol{x}) = W_i^{(1)} a(\boldsymbol{x}) + b_i^{(1)}, \qquad i = 1, \cdots, H. \tag{6.15}$$

这里， $W_i^{(1)}$ 和 $b_i^{(1)}$ 是与这一层相关的权重和偏置。

类似地，在多层感知机中，算子 $A^{(L+1)}$ 是一个仿射映射，它将一个具有 H 个分量的向量映射为一个只有 1 个分量的向量。将其自然地扩展到函数上，得到：

$$v^{(L+1)}(\boldsymbol{x}) = A^{(L+1)}(\boldsymbol{u}^{(L)})(\boldsymbol{x}), \tag{6.16}$$

其中

$$v^{(L+1)}(\boldsymbol{x}) = W_i^{(L+1)} u_i^{(L)}(\boldsymbol{x}) + b^{(L+1)}, \qquad i = 1, \cdots, H. \tag{6.17}$$

这里， $W_i^{(L+1)}$ 和 $b^{(L+1)}$ 是与这一层相关的权重和偏置。

接下来，我们描述激活函数对输入函数的作用。它是将激活函数应用于输入函数的逐点取值的一种简单扩展。也就是说，

$$\boldsymbol{u}^{(n)}(\boldsymbol{x}) = \sigma(\boldsymbol{v}^{(n)})(\boldsymbol{x}), \tag{6.18}$$

其中

$$u_i^{(n)}(\boldsymbol{x}) = \sigma(v_i^{(n)}(\boldsymbol{x})), \qquad i = 1, \cdots, H. \tag{6.19}$$

最后，还需要将算子 $A^{(n)}$、 $n = 2$、 $\cdots$、 L 扩展为函数。这些函数定义为：

$$\boldsymbol{v}^{(n+1)}(\boldsymbol{x}) = A^{(n+1)}(\boldsymbol{u}^{(n)})(\boldsymbol{x}), \tag{6.20}$$

其中

$$\begin{aligned} v_i^{(n+1)}(\boldsymbol{x}) &= W_{ij}^{(n+1)} u_j^{(n)}(\boldsymbol{x}) + b_i^{(n+1)} && (6.21) \\ &\quad + \int_\Omega \kappa_{ij}^{(n+1)}(\boldsymbol{y} - \boldsymbol{x}) u_j^{(n)}(\boldsymbol{y}) d\boldsymbol{y}, \qquad i = 1, \cdots, H. && (6.22) \end{aligned}$$

在上面的方程中，默认对哑指标 j （从 1 到 H）求和。方程中出现的新项是卷积。其依据是这样一个观察：一大类线性算子都可以表示为卷积。例如，所谓的格林算子会将线性偏微分方程的右端项（也称为强迫函数）映射为其解。函数 $\kappa_{ij}^{(n+1)}(\boldsymbol{z})$ 称为卷积核。我们注意到，每一层中都有 H^2 个这样的函数。

考察卷积的一个具体案例很有启发性。设 $\Omega = [0, L_1] \times [0, L_2]$，其中两个坐标分别记为 x_1 和 x_2，或记为 y_1 或 y_2。在这种情况下，我们可以将卷积写为：

$$v_i(x_1, x_2) \quad = \quad \int_0^{L_1} \int_0^{L_2} \kappa_{ij}(y_1 - x_1, y_2 - x_2) u_j(y_1, y_2)\, dy_2 dy_1, \qquad i = 1, \cdots, H. \tag{6.23}$$

在上面的方程中，我们省略了上标，因为它们与讨论无关。

注 6.8.1. 我们可以将傅里叶神经算子（*FNO*）理解为一系列仿射变换和卷积，随后接上逐点非线性激活。线性与非线性（激活函数）运算的这种组合，使我们能够利用这一架构逼近非线性算子。

注 6.8.2. 列出 *FNO* 中所有可训练实体很有启发性。首先，我们列出所有可训练参数：

$$W_i^{(1)}, W_{ij}^{(2)}, \cdots, W_{ij}^{(L)}, W_i^{(L+1)}; b_i^{(1)}, b_i^{(2)}, \cdots, b_i^{(L)}, b^{(L+1)}. \tag{6.24}$$

其次，列出所有可训练核函数

$$\kappa_{ij}^{(n)}(\boldsymbol{z}), \qquad n = 2, \cdots, L. \tag{6.25}$$

本节介绍的神经算子直接作用于函数，并将其转换为函数。然而，在计算机上实现这一算子时，函数必须以离散形式表示。下一节将对此进行说明。

6.9 傅里叶神经算子的离散化

上一节所述神经算子中出现的函数如下：

$$a, \boldsymbol{v}^{(1)}, \boldsymbol{u}^{(1)}, \cdots, \boldsymbol{v}^{(L)}, \boldsymbol{u}^{(L)}, v^{(L+1)}, u. \tag{6.26}$$

这些函数均定义在区域 Ω 上。我们用 N 均匀分布点对该区域进行离散化，并用各函数在这些点上的取值来表示它们。

例如，在二维情形下，若 $\Omega = [0, L_1] \times [0, L_2]$，我们将函数$a(x_1, x_2)$ 表示为：

$$a[m, n] = a(x_{1m}, x_{2n}), \quad m = 1 \cdots, N_1, \, n = 1 \cdots, N_2. \tag{6.27}$$

其中

$$x_{1m} \quad = \quad (m-1) \times \frac{L_1}{N_1 - 1} \tag{6.28}$$

$$x_{1n} \quad = \quad (n-1) \times \frac{L_2}{N_2 - 1}. \tag{6.29}$$

所有其他函数也将使用相同的表示方式。

现在，我们还必须考虑这些函数上所有运算的离散版本。下面将以 $\Omega = [0, L_1] \times [0, L_2]$ 的特殊情况为例进行说明。

我们从算子 $A^{(1)}$ 开始。其离散化形式为

$$\boldsymbol{v}^{(1)}[m, n] = A^{(1)}(a)[m, n], \tag{6.30}$$

其中

$$v_i^{(1)}[m, n] = W_i^{(1)} a[m, n] + b_i^{(1)}, \qquad i = 1, \cdots, H. \tag{6.31}$$

类似地，算子 $A^{(L+1)}$ 的离散化形式为，

$$v^{(L+1)}[m, n] = A^{(L+1)}(\boldsymbol{u}^{(L)})[m, n], \tag{6.32}$$

其中

$$v^{(L+1)}[m, n] = W_i^{(L+1)} u_i^{(L)}[m, n] + b^{(L+1)}, \qquad i = 1, \cdots, H. \tag{6.33}$$

接下来，我们描述激活函数对离散化输入函数的作用。其表示为

$$\boldsymbol{u}^{(n)}[m, n] = \sigma(\boldsymbol{v}^{(n)})[m, n], \tag{6.34}$$

其中

$$u_i^{(n)}[m, n] = \sigma(v_i^{(n)}[m, n]), \qquad i = 1, \cdots, H. \tag{6.35}$$

最后，我们还需要构造算子 $A^{(n)}, n = 2, \cdots, L$ 的离散版本。其定义如下：

$$\boldsymbol{v}^{(p+1)}[m, n] = A^{(p+1)}(\boldsymbol{u}^{(p)})[m, n], \tag{6.36}$$

其中

$$v_i^{(p+1)}[m, n] = W_{ij}^{(p+1)} u_j^{(p)}[m, n] + b_i^{(p+1)} \tag{6.37}$$

$$+ \sum_{r=1}^{N_1} \sum_{s=1}^{N_2} \kappa_{ij}^{(p+1)}[r - m, s - n] u_j^{(p)}[r, s] h_1 h_2, \qquad i = 1, \cdots, H, \tag{6.38}$$

其中 $h_1 = \frac{L_1}{N_1 - 1}$ 和 $h_2 = \frac{L_2}{N_2 - 1}$。请注意，卷积中的积分现在被所有网格点上的求和所取代。对于 i 和 m, n 的每个取值，计算该积分需要 $O(N_1 N_2 H)$ 次浮点运算。由于这需要针对 H 个不同的 i 值、 N_1 个 M 值以及 N_2 个 j 值进行计算，离散化卷积运算的总成本为 $O(N_1^2 N_2^2 H) = O(N^2 H^2)$，其中 $N = N_1 \times N_2$。该成本中的 N^2 因子难以接受，使得这一算法的实现并不实用。在下一节中，我们将介绍如何利用傅里叶变换（正变换和逆变换）克服这一瓶颈，从而得到一种实用的算法。这也是该算法被称为“傅里叶神经算子”的原因。

6.10 傅里叶变换的使用

考虑 $\Omega \equiv [0, L_1] \times [0, L_2]$ 上定义的周期函数 $u(x_2, x_2)$。如果该函数足够光滑，则可以用截断的傅里叶级数进行近似，

$$u(x_1, x_2) \approx \sum_{m=-N_1/2}^{N_1/2} \sum_{n=-N_2/2}^{N_2/2} \hat{u}[m, n] e^{2\pi \mathrm{i}(\frac{m x_1}{L_1} + \frac{n x_2}{L_2})}. \tag{6.39}$$

这里 N_1 和 N_2 是偶整数，系数 $\hat{u}[m, n]$ 是傅里叶系数，且 $\mathrm{i} = \sqrt{-1}$。我们注意到，尽管函数 u 是实值的，系数却是复值的。不过，

由于 u 是实值的，它们满足规则 $\hat{u}[-m,-n]=\hat{u}^*[m,n]$，其中 $(.)^*$ 表示复数的复共轭。增大 N_1 和 N_2可以提高近似精度；当这些数趋于无穷大时，我们便恢复等式。上述关系通常称为傅里叶逆变换，因为它将傅里叶系数映射到物理空间中的函数。

傅里叶正变换（将物理空间中的函数映射为傅里叶系数）可由上面的关系通过以下步骤得到：

1. 两边乘以 $e^{-2\pi \mathrm{i}(\frac{rx_1}{L_1}+\frac{sx_2}{L_2})}$，其中 r和 s为整数。

2. 对 Ω 两边积分。

3. 注意到积分 $\int_\Omega e^{2\pi \mathrm{i}(\frac{(m-r)}{L_1}x_1+\frac{(n-s)x_2}{L_2})}dx_1dx_2$仅在 $m=r$和 $n=s$时不为零，此时其值为 L_1L_2。

由此可得最终关系：

$$\hat{u}[r,s]=\frac{1}{L_1L_2}\int_0^{L_1}\int_0^{L_2}u(x_1,x_2)e^{-2\pi \mathrm{i}(\frac{rx_1}{L_1}+\frac{sx_2}{L_2})}dx_1dx_2. \tag{6.40}$$

现在我们来说明如何利用傅里叶变换高效地计算卷积。为此，我们考虑式 (6.23) 中二维卷积的特殊情况。首先，将 $u_j(y_1,\ y_2)=\sum_{m=-N_1/2}^{N_1/2}\sum_{n=-N_2/2}^{N_2/2}\hat{u}_j[m,n]e^{2\pi \mathrm{i}(\frac{my_1}{L_1}+\frac{ny_2}{L_2})}$代入该式，得到：

$$\begin{aligned}
v_i(x_1,x_2) &= \int_0^{L_1}\int_0^{L_2}\kappa_{ij}(y_1-x_1,y_2-x_2)\sum_{m,n}\hat{u}_j[m,n]e^{2\pi \mathrm{i}(\frac{my_1}{L_1}+\frac{ny_2}{L_2})}\,dy_2dy_1\\
&= \sum_{m,n}\hat{u}_j[m,n]\int_0^{L_1}\int_0^{L_2}\kappa_{ij}(y_1-x_1,y_2-x_2)e^{2\pi \mathrm{i}(\frac{my_1}{L_1}+\frac{ny_2}{L_2})}\,dy_2dy_1\\
&= \sum_{m,n}\hat{u}_j[m,n]\int_{-x_1}^{L_1-x_1}\int_{-x_2}^{L_2-x_2}\kappa_{ij}(z_1,z_2)e^{2\pi \mathrm{i}(\frac{m(z_1+x_1)}{L_1}+\frac{n(z_2+x_2)}{L_2})}\,dz_2dz_1\\
&= \sum_{m,n}\hat{u}_j[m,n]e^{2\pi \mathrm{i}(\frac{mx_1}{L_1}+\frac{nx_2}{L_2})}\int_0^{L_1}\int_0^{L_2}\kappa_{ij}(z_1,z_2)e^{2\pi \mathrm{i}(\frac{mz_1}{L_1}+\frac{nz_2}{L_2})}\,dz_2dz_1\\
&= L_1L_2\sum_{m,n}\hat{u}_j[m,n]\hat{\kappa}_{ij}[-m,-n]e^{2\pi \mathrm{i}(\frac{mx_1}{L_1}+\frac{nx_2}{L_2})}.
\end{aligned} \tag{6.41}$$

在上述推导中，从第一行到第二行，我们将求和移到了积分号外，并注意到系数 $\hat{u}_j[m,n]$ 与 y_1 和 y_2 无关。从第二行到第三行，我们引入了变量 $z_1=y_1-x_1$ 和 $z_2=y_2-x_2$。从第三行到第四行，我们利用了函数 $\kappa_{ij}(z_1,\ z_2)$ 具有周期性这一事实。最后，从第四行到第五行，我们运用了傅里叶变换的定义（6.40）。这个最终关系告诉我们，卷积可以通过以下方式计算：

1. 计算 u_j 的傅里叶变换。

2. 计算 κ_{ij} 的傅里叶变换。

3. 计算这两个变换的系数乘积。

4. 计算乘积的傅里叶逆变换。

接下来，我们考虑到只会使用函数u_j 和 κ_{ij}的离散形式。这意味着我们要在有限个网格点上计算傅里叶逆变换 (6.39)。此外，这意味着我们必须对傅里叶变换 (6.40) 中的积分进行近似。其另一种形式如下：

$$\hat{u}[r,s] = \frac{h_1 h_2}{L_1 L_2} \sum_{m=1}^{N_1} \sum_{n=1}^{N_2} u[m,n] e^{-2\pi \mathrm{i}(\frac{r x_{1m}}{L_1} + \frac{s x_{2n}}{L_2})}. \tag{6.42}$$

这里 $h_1 = \frac{L_1}{N_1}$ 和 $h_2 = \frac{L_2}{N_2}$, $x_{1m} = (m-1)h_1$ 和 $x_{2n} = (n-1)h_2$。

最后需要注意的是，计算 (6.39) 和 (6.42) 中的求和需要 $O(N^2)$ 次运算。因此，除非 N 非常小，否则通过傅里叶方法计算卷积并不实用。然而，使用快速傅里叶变换（FFT）可将这一计算成本降至 $O(N\log\ N)$。这样一来，实现卷积的成本便降至 $O(N\log\ NH^2)$，使得傅里叶神经算子的实现切实可行。

第 7 章

概率深度学习

到目前为止，我们讨论了回归和分类问题：给定输入 $\boldsymbol{x}$ ，我们需要计算一个单一输出 $\boldsymbol{y}$。然而，对于许多我们感兴趣的问题而言，这可能还不够。事实上，给定一个 $\boldsymbol{x}$，可能会有许多个 $\boldsymbol{y}$。例如，

1. $\boldsymbol{y}$ 和 $\boldsymbol{x}$ 的测量值中可能含有一些随机噪声。2. $\boldsymbol{y}$ 和 $\boldsymbol{x}$ 可能本质上是随机的。例如， $\boldsymbol{y}$ 可以是在空间某点 $\boldsymbol{x}$ 测得的湍流压力。3. 反问题可能有多个解。例如，正向/直接问题是已知水头传导率来确定温度场，而反问题则是已知（可能含噪声的）温度场来确定传导率场。

因此，我们需要建立一个概率框架，以便使用深度学习算法解决此类问题。回想一下，我们的确定性模型表示为 $\boldsymbol{y}=\boldsymbol{\mathcal{F}}(\boldsymbol{x};\,\boldsymbol{\theta})$。在概率设定中， $\boldsymbol{y}$、 $\boldsymbol{x}$ 和 $\boldsymbol{\theta}$ 被视为随机变量。

在处理随机变量之前，我们需要了解概率论中的一些关键要素，这些要素对于定义随机变量是必不可少的。

7.1 概率论的关键要素

随机试验由一个过程以及一组一个或多个观察/测量值构成。例如，

1. 观察一个开关，判断它是打开还是关闭。
2. 抛一枚硬币3次，并记录正面 H 或反面的 T序列。
3. 抛一枚硬币3次，并统计正面 H 出现的次数。注意，这与前面的实验相同，但测量值不同。
4. 转动一个转盘，并测量最终角度（以弧度为单位）。

结果是实验中不能再分解为更小部分的结果。样本空间（记作 $\mathbb{S}$）是实验所有可能结果的集合。对于上面观察到的四个随机试验，我们有

1. $\mathbb{S}=$ {开、关}。

2. $\mathbb{S} = \{TTT,\ TTH,\ THT,\ HTT,\ THH,\ HTH,\ HHT,\ HHH\}$。

3. $\mathbb{S} = \{0,\ 1,\ 2,\ 3\}$。4.
$\mathbb{S} = \{\psi :\quad \psi \in (0,\quad 2\pi]\}$。

请注意，前 3 个实验与最后一个实验之间有一个根本区别：前者的 $\mathbb{S}$ 是离散且可数的，而后者的 $\mathbb{S}$ 是不可数的。

事件是结果的集合，即 $\mathbb{S}$的一个子集。通常，事件中的结果都满足某个条件。来看一些上述实验的例子

1. $A = \{$在$\}$ 上，或 $A = \{$在，关闭$\} = \mathbb{S}$ 。

2. A 是所有至少有 2 H的结果，即 $A = \{THH,\ HTH,\ HHT,\ HHH\}$。

3. A 是所有至少有 2 H的结果，即 $A = \{2,\ 3\}$。如果我们将 B 定义为所有具有 4 H的结果，那么就没有结果满足这一条件，即 $B = \emptyset$ 空集。

4. A 是所有满足 $\psi > \pi/4$的结果，即 $A = \{\psi :\quad \psi \in (\pi/4,\quad 2\pi]\}$。

事件类 $\mathscr{E}$ 是一个集合，其中包含所有可以定义概率的事件（集合）。当 $\mathbb{S}$ 可数时， $\mathscr{E}$ 是 $\mathbb{S}$的所有子集。当 $\mathbb{S}$ 不可数时， $\mathscr{E}$ 是博雷尔域（或博雷尔代数），即 $\mathbb{S}$中所有开集和闭集的集合。

概率律 是一条规则，它为事件类 $\mathscr{E}$中的所有集合赋予一个概率。我们列出**概率公理**，它们是概率律必须满足的要求。

考虑一个实验的样本空间 $\mathbb{S}$ 及其对应的事件类 $\mathscr{E}$。令$P :\ \mathscr{E} \mapsto [0,1]$ 满足

1. 对所有 $A \in \mathscr{E},\quad P[A] \geq 0$ 。

2. $P[\mathbb{S}] = 1$。

3. 如果 $A_1, A_2, \ldots$ 是满足以下条件的事件：$A_i \cap A_j = \emptyset$ 对所有 $i \neq j$均成立，即这些事件互斥，那么

$$P[\bigcup_{i=1}^{\infty} A_i] = \sum_{i=1}^{\infty} P[A_i].$$

任何满足上述条件的赋值 P 都称为有效概率律。注意，概率类似于质量。它是非负的（公理 1）、守恒的（总质量始终为 1，即公理 2），并且对于不同的点，总质量等于各点质量之和（公理 3）。

如果 $\mathbb{S}$ 是可数的，那么只需为 $\mathbb{S}$中的所有元素（即所有基本结果）定义概率律，同时确保概率非负且总和为 1（前两个公理）即可。让我们依据这些条件，尝试为前三个具有可数 $\mathbb{S}$ 的例子指定概率律。

1. 对于某个 $p \in [0,1]$，在$] = p$上定义 $P[$；在$] = 1 - p$上定义 $P[$。

2. 对于没有记忆的公平骰子，$P[a_i] = 1/8$，其中 $a_i \in \mathbb{S},\ i = 1,\ \cdots,\ 8$。

3. 对于没有记忆的公平骰子，$P[0] = 1/8,\quad P[1] = 3/8,\quad P[2] = 3/8,\quad P[3] = 1/\quad 8.$

注释 7.1.1. 请作为练习，验证这些情形分别满足公理。

对于连续样本空间，只需为所有开区间和闭区间定义概率律，同时确保满足公理 1 和 2。我们来考虑第四个例子，其中有一个不可数的 $\mathbb{S}$。如果转盘完全没有偏置，那么概率就是均匀分布的。那么对于 $b \geq a$，我们说 $P[(a,\ b]] = (b-a)/(2\pi)$。请注意，连续样本空间中的单点集的概率为零（对于任何分布都是如此）。

注释 7.1.2. 从现在起，每当我们谈到样本空间 $\mathbb{S}$时，都会默认指的是三元组 $(\mathbb{S}, \mathscr{E}, P)$。这个三元组也称为“测度空间”。

7.2 随机变量

随机变量 X 是一个从 $\mathbb{S}$ 到实数轴的函数，并满足以下性质：对于所有 $b \in \mathbb{R}$，集合 $A_b = \{\xi \in \mathbb{S}:\ X(\xi) \leq b\}$ 属于 $\mathscr{E}$。请注意，用测度论语言来说，我们要求 X 是一个可测函数。另请注意，根据 A_b的定义，我们要求对于所有 $b \in \mathbb{R}$，都能够计算 $P[A_b]$。

下面为上述示例定义随机变量（RV）：

1. 对于 $\mathbb{S} = \{$上，下$\}$，其中 $P[$上$] = p$、$P[$下$] = 1-p$，定义随机变量

$$X(\xi) = \begin{cases} 0 & \text{if } \xi = \text{off} \\ 1 & \text{if } \xi = \text{on.} \end{cases} \tag{7.1}$$

这也称为伯努利随机变量。

2. 对于 $\mathbb{S} = \{TTT,\ TTH,\ THT,\ HTT,\ THH,\ HTH,\ HHT,\ HHH\}$，其中 $P[a_i] = 1/8$ 对所有 $a_i \in \mathbb{S}$ 均成立，定义

$$X(\xi) = \text{Number of heads in } \xi. \tag{7.2}$$

请注意，这就是实验 3 中描述的随机事件。

3. 这个随机事件本身已经是一个随机变量。

4. 对于 带有 $\mathbb{S} = \{\psi$ 的转盘实验：$\psi \in (0,\ 2\pi]\}$，其中 $P[(a,\ b]] = (b-a)/(2\pi)$，定义 e

$$X(\psi) = \frac{\psi}{2\pi}. \tag{7.3}$$

如果 X 定义在离散样本空间上，则称其为离散随机变量；如果定义在连续样本空间上，则称其为连续随机变量。

如上所述，随机变量的概率解释继承自用于定义它的测度空间。在接下来的章节中，我们将定义随机变量的概率解释。

7.2.1 累积分布函数

随机变量 X 的**累积分布函数**（cdf）定义为

$$F_X(x) = P[\xi : X(\xi) \leq x]$$

它定义了 $\mathbb{R}$ 上 X 取值于区间 $(-\infty,\ x]$ 的概率。现在我们来定义上述示例的累积分布函数：

1. 对于由 (7.1) 定义的伯努利随机变量

- 如果 $x < 0$，那么 $F_X(x) = P[\emptyset] = 0$
- 若 $0 \le x < 1$，则 $F_X(x) = P[\{关\}] = 1 - p$
- 若 $x \ge 1$，则 $F_X(x) = P[\{开，关\}] = 1$

完整的累积分布函数如图 7.1(a) 所示。

2. 对于由 (7.2) 定义的随机变量

- 若 $x < 0$，则 $F_X(x) = P[\emptyset] = 0$
- 若 $0 \le x < 1$，则 $F_X(x) = P[\#ofH = 0] = P[\{TTT\}] = 1/8$
- 若 $1 \le x < 2$，则 $F_X(x) = P[\#ofH = 0，\ 1] = 1/8 + 3/8 = 4/8$
- 若 $3 \le x < 3$，则 $F_X(x) = P[\#ofH = 0，\ 1，\ 2] = 1/8 + 3/8 + 3/8 = 7/8$
- 若 $x \ge 3$，则 $F_X(x) = P[\#ofH = 0，\ 1，\ 2，\ 3] = 1$

完整的累积分布函数如图 7.1(b) 所示。

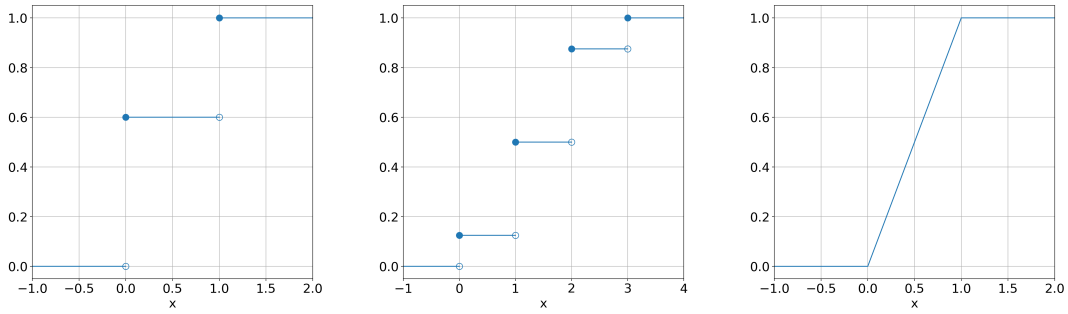
3. 这个随机变量与例 2 相同。

4. 对于由 (7.3) 定义随机变量的转盘实验

$$F_X(x) = P[\psi : X(\psi) \le x] = P[\{\psi : \psi \le 2\pi x\}]$$

- 若 $x < 0$，则 $F_X(x) = P[\emptyset] = 0$
- 若 $0 \le x < 1$，则 $F_X(x) = P[\{\psi \in (0，\ 2\pi x]\}] = \frac{2\pi x}{2\pi} = x$
- 若 $x \ge 1$，则 $F_X(x) = P[\{\psi \le 2\pi\}] = 1$

完整的累积分布函数如图 7.1(c)所示。



（a）伯努利　　（b）3 次抛硬币　　（c）转盘

图 7.1：累积分布函数的示例

让我们讨论一下 F_X 的一些性质：

1. $0 \le F_X(x) \le 1$. 2. $\lim_{x\to\infty} F_X(x) = 1$. 3. $\lim_{x\to-\infty} F_X(x) = 0$.

4. F_X 单调递增。

5. 累积分布函数始终右连续

$$F_X(x) = \lim_{h \to 0+} F_x(x+h).$$

注意，对于离散随机变量（见图 7.1），累积分布函数在有限个区间处不连续。事实上，离散随机变量的累积分布函数可以写成如下形式的有限和

$$F_X(x) = \sum_{k=1}^{K} p_k H(x - x_k), \quad \sum_{k=1}^{K} p_k = 1,$$

其中， p_k 是概率质量， H 是赫维赛德函数

$$H(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}.$$

注 7.2.1。一旦我们得到 F_X ，就可以计算 X 在 $\mathbb{R}$中的“任意”区间内取值的概率，即可以计算 $P[a < X \leq b]$。注意，

$$\begin{aligned} F_X(b) = P[X \leq b] &= P[(X \leq a) \cup (a < X \leq b)] \\ &= P[X \leq a] + P[a < X \leq b] \qquad \textit{(mutually exclusive events)} \\ &= F_X(a) + P[a < X \leq b]. \end{aligned}$$

因此，

$$P[a < X \leq b] = F_X(b) - F_X(a).$$

7.2.2 概率密度函数

我们定义**概率密度函数 (pdf)**。对于连续的 F_X，其定义为

$$f_X(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x) \tag{7.4}$$

它继承了累积分布函数的以下性质：

1. $f_X(x) \geq 0, \quad \forall x \in \mathbb{R}$，因为 F_X 是单调递增的。

2. 极限$_{x \to -\infty} f_X(x) =$ 极限$_{x \to \infty} f_X(x) = 0$.

3. 将 (7.4) 从 $(-\infty, \ x]$ 积分，得到

$$\int_{-\infty}^{x} f_X(y)\mathrm{d}y = F_X(x) - \lim_{x \to -\infty} F_X(x) = F_X(x).$$

4. 此外

$$P[a < X \leq b] = F_X(b) - F_X(a) = \int_{-\infty}^{b} f_X(y)\mathrm{d}y - \int_{-\infty}^{a} f_X(y)\mathrm{d}y = \int_{a}^{b} f_X(y)\mathrm{d}y.$$

因此，概率密度函数在某个区间上的积分给出“概率质量”，即随机变量落在该区间内的概率。这就是概率密度函数被称为“密度”的原因。

5. 此外，

$$\int_{-\infty}^{\infty} f_X(y)\mathrm{d}y = \lim_{x\to\infty} F_X(x) - \lim_{x\to-\infty} F_X(x) = 1.$$

6. 对于非常小的 $h > 0$，我们有如下解释

$$P[a < X \le a + h] = \int_a^{a+h} f_X(y)\mathrm{d}y \approx h f_X(a).$$

注意，当 $h \to 0^+$ 时，$P[a < X \le a + h] \to 0$。也就是说，对于连续随机变量，取到单个值的概率为零。

7.2.3 重要随机变量示例

我们来看看一些重要的随机变量及其对应的累积分布函数和概率密度函数（另见图 7.2）：

1. **均匀随机变量：** 对于某个区间 $(a, b]$，概率密度函数为

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in (a, b] \\ 0 & \text{other wise} \end{cases},$$

而累积分布函数为

$$F_X(x) = \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } x \in (a, b] \\ 1 & \text{if } x > b \end{cases}.$$

2. **指数随机变量：** 用于对关键事件发生后设备或人的寿命建模。在这种情况下，X 表示失效时间，而 $P[X > x] = e^{-\lambda x}$，其中 $\lambda > 0$ 是表示失效率的模型参数。因此，

$$F_X(x) = P[X \le x] = 1 - P[X > x] = 1 - e^{-\lambda x},$$

and

$$f_X(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x) = \lambda e^{-\lambda x}.$$

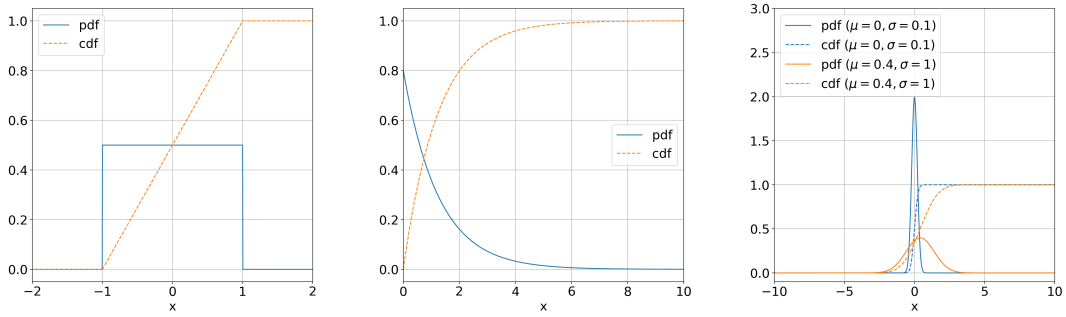
3. **高斯随机变量：** 用于对身高、体重等自然事物建模。事实上，根据中心极限定理，它也是许多随机变量聚合后得到的分布。其概率密度函数为

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

该分布由均值 μ 和方差 σ^2 确定，均值表示该分布的中心，方差表示其扩散度。相应的累积分布函数为

$$F_X(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right], \quad \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\mathrm{d}t.$$

在概率机器学习中，人们广泛使用均匀随机变量和高斯随机变量。



(a) 均匀随机变量（$a = -1,\ b = 1$）　(b) 指数随机变量（$\lambda = 0.8$）　(c) 高斯随机变量

图 7.2：连续随机变量

7.2.4 随机变量的期望与方差

给定一个 对于概率密度函数为 f_X 的随机变量 X，我们可以计算它的**期望值**、**期望**或**均值**a s

$$\mu_X := \mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x) \mathrm{d}x.$$

期望具有以下性质：

- 注意，如果概率密度函数关于 $x = m$ 对称，那么 $\mathbb{E}[X] = m$。要理解这一点，请注意 $(m - x) f_X(x)$ 关于 m 呈反对称。因此

$$0 = \int_{-\infty}^{\infty} (m-x) f_X(x) \mathrm{d}x = m \int_{-\infty}^{\infty} f_X(x) \mathrm{d}x - \int_{-\infty}^{\infty} x f_X(x) \mathrm{d}x \quad \Longrightarrow \quad \int_{-\infty}^{\infty} x f_X(x) \mathrm{d}x = m.$$

  利用这一性质，我们可以轻松得出：均匀随机变量的均值为 $(a + b)/2$，而高斯随机变量的均值为 μ。

- $\mathbb{E}[c] = c$ 对于常数 c。

- 我们可以按如下方式计算随机变量函数的期望值：

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) \mathrm{d}x.$$

- 期望具有线性性质，即

$$\mathbb{E}[g(X) + ch(X)] = \mathbb{E}[g(X)] + c\mathbb{E}[h(X)].$$

随机变量的**方差**衡量其相对于均值的离散程度。其计算方式为

$$\mathrm{VAR}[X] = \int_{-\infty}^{\infty} (x - \mu_X)^2 f_X(x) \mathrm{d}x.$$

此外，我们将**标准差** 记为

$$\sigma_X := \mathrm{STD}[X] = \sqrt{\mathrm{VAR}[X]}.$$

对于均匀随机变量

$$\mathrm{VAR}[X] = \int_a^b \left(x - \frac{b+a}{2}\right)^2 \frac{1}{b-a}\mathrm{d}x = \frac{(b-a)^2}{12}.$$

对于高斯随机变量，我们首先利用概率密度函数的性质写出

$$\int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}\mathrm{d}x = \sqrt{2\pi}\sigma.$$

对两边关于 σ 求导，得到

$$\int_{-\infty}^{\infty} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}(x-\mu)^2\sigma^{-3}\mathrm{d}x = \sqrt{2\pi}$$

经过一些代数运算后，我们得到

$$\mathrm{VAR}[X] = \int_{-\infty}^{\infty} (x-\mu)^2 \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}\mathrm{d}x = \sigma^2.$$

7.2.5 随机变量对

在概率机器学习中，我们会处理多个随机变量。例如，输入、输出和权重都可能是随机变量。因此，我们需要将单个随机变量的概念扩展到随机变量向量。本节首先考虑一对随机变量，以此进行扩展。为一对随机变量定义的大多数概念都适用于随机变量向量。

一对随机变量是从测度空间到以下形式的映射，其事件类为 $\mathscr{E}$: rm

$$\boldsymbol{X} : \mathscr{E} \to \mathbb{R}^2,$$

其中映射可以是离散的，也可以是连续的。我们有时会使用记号 $\boldsymbol{X} = (X,\ Y)$。例如，我们可以将转盘旋转两次，并测量 $\psi_1 \in (0,\ 2\pi]$, $\psi_2 \in (0,\ 2\pi]$。在这种情况下，我们可以定义这两个随机变量

$$X(\psi_1) = \frac{\psi_1}{2\pi}, \quad Y(\psi_2) = \frac{\psi_2}{2\pi}.$$

$\boldsymbol{X}$ 的事件是 $\mathbb{R}^2$中的集合。为了计算事件的概率，我们需要定义**联合累积分布函数** F_{XY}: $\mathbb{R}^2 \to \mathbb{R}$，其中

$$F_{XY}(x,y) = P[X \le x, Y \le y] = P[\xi \in \mathbb{S} : X(\xi) \le x, Y(\xi) \le y].$$

与单个随机变量类似

- 联合累积分布函数是 x、 y 的非递增函数。换句话说，对于 $x \ge x'$ 和 $y \ge y'$

$$F_{XY}(x,y) \ge F_{XY}(x',y').$$

- 当 $_{x\to-\infty} F$ 趋于无穷时，$\lim_{x\to-\infty} F(x,\ y) = 0$; 当 $_{y\to-\infty} F$ 趋于无穷时，$\lim_{y\to-\infty} F(x,\ y) = 0$; 当 $_{x,\ y\to\infty} F(x,\ y) = 1$。

- 我们可以计算

$$P[x_1 < X \le x_2, y_1 < Y \le y_2] = F_{XY}(x_2,y_2) + F_{XY}(x_1,y_1) - F_{XY}(x_1,y_2) - F_{XY}(x_2,y_1)。$$

对于 X，Y连续的联合分布，我们可以将**联合概率密度函数** 定义为

$$f_{XY}(x,y) = \frac{\partial^2 F_{XY}(x,y)}{\partial x \partial y}$$

它具有以下性质

- $f_{XY}(x,\ y)=0$ 作为 $x\to\pm\infty$ 或 $y\to\pm\infty$。
- $F_{XY}(x,\ y)=\int_{-\infty}^{x}\int_{-\infty}^{y}f_{XY}(r,\ s)\mathrm{d}r\mathrm{d}s$。
- $\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f_{XY}(r,\ s)\mathrm{d}r\mathrm{d}s=1$。
- $P[x_1<X\le x_2,y_1<Y\le y_2]=\int_{x_1}^{x_2}\int_{y_1}^{y_2}f_{XY}(x,\ y)\mathrm{d}x\mathrm{d}y.$
- $P[\boldsymbol{X}\in B]=\int\int_{B}f_{XY}(x,\ y)\mathrm{d}x\mathrm{d}y$。

我们来看一些重要的联合随机变量：

1. **联合均匀随机变量：**对于某个区域 $(a,\ b]\times(c,\ d]$，联合概率密度函数为

$$f_{XY}(x,y)=\begin{cases}\frac{1}{(b-a)(d-c)} & \text{if } (x,y)\in(a,b]\times(c,d]\\ 0 & \text{otherwise}\end{cases}. \tag{7.5}$$

2. **联合高斯随机变量：**其联合概率密度函数为

$$f_{XY}(x,y)=\frac{1}{\sqrt{(2\pi)^2\det(\boldsymbol{\Sigma})}}\exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\top}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]$$

其中 $\boldsymbol{x}=(x,\ y)$、$\boldsymbol{\mu}=(\mu_x,\ \mu_y)$ 为均值，$\boldsymbol{\Sigma}$ 称为协方差矩阵

$$\boldsymbol{\Sigma}=\begin{bmatrix}\sigma_x^2 & \rho\sigma_x\sigma_y\\ \rho\sigma_x\sigma_y & \sigma_y^2\end{bmatrix}.$$

协方差矩阵是对称正定的。

我们可以定义随机变量 X 的**边缘概率密度函数**，即假设 Y 取遍所有可能值时 X 的概率密度函数

$$f_X(x)=\int_{-\infty}^{\infty}f_{XY}(x,y)\mathrm{d}y.$$

同理，Y 的边缘概率密度函数为

$$f_Y(y)=\int_{-\infty}^{\infty}f_{XY}(x,y)\mathrm{d}x.$$

如果 $f_{XY}(x,\ y)=f_X(x)f_Y(y)$，则称随机变量 X 和 Y 相互**独立**。

Qu题 7.2.1. 证明，具有联合概率密度函数 (7.5)的联合均匀随机变量相互独立。

考虑函数 $g(\boldsymbol{X})$，它可以是标量值、向量值或张量值，那么它的期望值为

$$\mathbb{E}[g(\boldsymbol{X})]=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}g(\boldsymbol{x})f_{XY}(x,y)\mathrm{d}x\mathrm{d}y$$

只要该积分有定义。例如：

- 对于 $g(\boldsymbol{X})=X$，有 $\mathbb{E}[g(\boldsymbol{X})]=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}xf_{XY}(x,\ y)\mathrm{d}x\mathrm{d}y$。
- 对于 $g(\boldsymbol{X})=\boldsymbol{X}$，我们有向量值期望 $\mathbb{E}[g(\boldsymbol{X})]=[\mathbb{E}[X],\mathbb{E}[Y]]$。
- 对于 $g(\boldsymbol{X})=X+Y$，我们有 $\mathbb{E}[g(\boldsymbol{X})]=\mathbb{E}[X]+\mathbb{E}[Y]$。

$\boldsymbol{X}$ 的**协方差**为

$$\mathrm{COV}[\boldsymbol{X}] = \mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}]) \otimes (\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])]$$

其中

$$\begin{aligned}
\mathrm{COV}[\boldsymbol{X}]_{11} &= \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathrm{VAR}[X] \\
\mathrm{COV}[\boldsymbol{X}]_{22} &= \mathbb{E}[(Y - \mathbb{E}[Y])^2] = \mathrm{VAR}[Y] \\
\mathrm{COV}[\boldsymbol{X}]_{12} &= \mathrm{COV}[\boldsymbol{X}]_{21} = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])].
\end{aligned}$$

X 和 Y 称为**不相关**，如果 $\mathrm{COV}[\boldsymbol{X}]_{12} = 0$。此外，对于相互独立的随机变量，$\mathrm{COV}[\boldsymbol{X}]_{12} = 0$ 。**注意：$\mathrm{COV}[\boldsymbol{X}]_{12} = 0$ 并不意味着这些随机变量相互独立！**

最后，我们感兴趣的是，在已知 $X = \hat{x}$ 的情况下，考察 Y 的概率密度函数。一个不错的猜测是 $f_{XY}(\hat{x}, y)$。然而，这不一定是概率密度函数，因为它在 y 上的积分不一定等于 1。这引出了在已知 $X = \hat{x}$ 的情况下，Y 的**条件概率密度函数**，

$$f_{Y|X}(y|\hat{x}) = \frac{f_{XY}(\hat{x}, y)}{\int_{-\infty}^{\infty} f_{XY}(\hat{x}, y)\mathrm{d}y} = \frac{f_{XY}(\hat{x}, y)}{f_X(\hat{x})}.$$

类似地，我们可以将已知 $Y = \hat{y}$ 时 X 的条件概率密度函数写作

$$f_{X|Y}(x|\hat{y}) = \frac{f_{XY}(x, \hat{y})}{f_Y(\hat{y})}.$$

我们注意到，将回归问题扩展到概率框架，会使我们需要确定给定一个输入实例时输出的条件分布。这将在第 7.4 节中讨论。

7.3 无监督概率深度学习算法

我们从一个包含 N_X 个分量的随机变量向量 $\boldsymbol{X}$ 开始，其概率密度函数由 f_X 给出。假设我们有一个从密度 f_X 中采样得到的数据集，其中包含 $\{\boldsymbol{x}_i\}$ 个样本，记作 $\boldsymbol{x}_i \sim f_X$。例如，这些样本可以是分辨率为 512×512 的汽车 RGB 图像。请注意，这意味着这些样本位于维度为 $N_X = 512 \times 512 \times 3$ 的空间中，这个维度相当高！我们可以将图像中的每个像素视为一个随机变量，其取值由像素强度给出（涵盖全部 3 个通道）。因此，这些图像可以看作某个 N_X 维随机变量的样本，其密度 f_X 未知。此外，由于图像中物体（即汽车）固有的结构，多维随机变量的各个分量预计会高度相关，从而使 f_X 具有非平凡的形式。这种相关性还意味着，或许可以将 $\boldsymbol{X}$ 的维度从 N_X 降低到更小的数值，从而使表示更加简单。

我们希望利用给定的有限样本集 $\{\boldsymbol{x}_i\}$ 来学习数据的密度 f_X ，并从学到的分布中生成新的样本。这类方法称为生成算法。虽然有多种生成算法可供选择，但我们将聚焦于一种特定的深度学习算法，即生成对抗网络，简称生成对抗网络。

7.3.1 生成对抗网络

生成对抗网络最初由 Goodfellow 等人于 [6]2014年 提出。此后，人们提出了许多生成对抗网络变体，它们在网络架构和用于训练生成对抗网络的目标函数方面各不相同

。我们首先介绍抽象的问题设定，随后介绍生成对抗网络的架构和训练过程。

考虑数据集 $\mathcal{S} = \{\boldsymbol{x}_i \in \Omega_X \subset \mathbb{R}^{N_X} : 1 \leq i \leq N_{\text{train}}\}$。我们假设样本是某个密度为 f_X 的随机变量 $\boldsymbol{X}$ 的实现值，即 $\boldsymbol{x}_i \sim f_X$。我们希望训练一个生成对抗网络来学习 f_X，并从中生成新样本。

生成对抗网络通常由两个子网络组成：生成器和判别器（或评论家）。生成器是如下形式的网络：

$$\boldsymbol{g}(.;\boldsymbol{\theta}_g) : \Omega_Z \to \Omega_X, \quad \boldsymbol{g} : \boldsymbol{z} \mapsto \boldsymbol{x} \tag{7.6}$$

其中 $\boldsymbol{\theta}_g$ 是可训练参数，而 $\boldsymbol{z} \in \Omega_Z \subset \mathbb{R}^{N_Z}$ 是随机变量 $\boldsymbol{Z}$ 的实现值，该随机变量服从一种简单分布，例如密度为下式的不相关多元高斯分布：

$$f_Z(\boldsymbol{z}) = \frac{1}{\sqrt{(2\pi)^2 \det(\boldsymbol{\Sigma})}} \exp\left[-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{z}-\boldsymbol{\mu})\right] \quad \text{with} \quad \boldsymbol{\mu} = \boldsymbol{0}, \quad \boldsymbol{\Sigma} = \boldsymbol{I}.$$

通常，$N_Z \ll N_X$，其中 $\boldsymbol{Z}$ 称为生成对抗网络的潜变量。生成器的架构取决于 $\boldsymbol{x}$ 的大小/形状。如果 $\boldsymbol{x}$ 是向量，那么 $\boldsymbol{g}$ 可以是一个输入维度为 N_Z、输出维度为 N_X 的多层感知机。如果 $\boldsymbol{x}$ 是图像，例如形状为 $H \times W \times 3$ 的图像，那么生成器架构将包含若干全连接层，随后将其重塑为具有许多通道的粗略图像，再通过若干转置卷积通道，逐渐提高空间分辨率并压缩通道数量，最终扩展至形状 $H \times W \times 3$。这也称为解码器架构，类似于 U-Net 的向上分支（见图 5.9）。无论哪种情况，对于固定的 $\boldsymbol{\theta}_g$，生成器$\boldsymbol{g}$都会将随机变量 $\boldsymbol{Z}$ 转换为另一个随机变量 $\boldsymbol{X}^g = \boldsymbol{g}(\boldsymbol{Z};\boldsymbol{\theta}_g)$，其密度为 f_X^g，对应于经 $\boldsymbol{g}$ 推前的潜在密度 f_Z。我们希望选择 $\boldsymbol{\theta}_g$，使得 f_X^g 接近未知的目标分布 f_X。

判别器网络的形式为

$$d(.;\boldsymbol{\theta}_d) : \Omega_X \to \mathbb{R} \tag{7.7}$$

其可训练参数为 $\boldsymbol{\theta}_d$。同样，判别器架构取决于$\boldsymbol{x}$的形状。如果 $\boldsymbol{x}$ 是向量，那么 d 可以是一个输入维度为 N_X、输出维度为 1 的多层感知机。如果 $\boldsymbol{x}$ 是图像，那么判别器架构将包含若干卷积层，后接一个展平层和若干全连接层。这与图5.7所示的 CNN 架构类似，但输出为标量且没有输出函数。

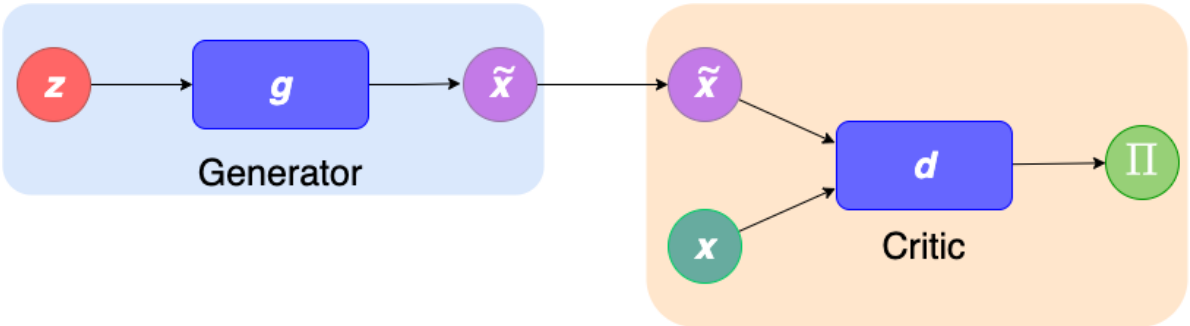


图7.3：生成对抗网络示意图

图7.3展示了生成对抗网络的示意图，以及子网络的输入和输出。生成器和判别器扮演对抗角色。判别器经过训练，能够区分

来自 f_X的真实样本与由 $\boldsymbol{g}$根据密度 f_X^g生成的伪造样本。另一方面，生成器经过训练，通过生成逼真的样本来欺骗判别器，也就是说，生成与从 f_X中采样得到的样本相似的样本。

我们定义描述*Wasserstein* 生成对抗网络（*WGAN*） [2], 的目标函数。与原始生成对抗网络相比，它具有更好的鲁棒性和收敛特性。目标函数为

$$\Pi(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = \underbrace{\frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} d(\boldsymbol{x}_i; \boldsymbol{\theta}_d)}_{\text{critic value on real samples}} - \underbrace{\frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} d(\boldsymbol{g}(\boldsymbol{z}_i; \boldsymbol{\theta}_g); \boldsymbol{\theta}_d)}_{\text{critic value on fake samples}} \tag{7.8}$$

其中， $\boldsymbol{x}_i \in \mathcal{S}$ 是来自真实目标分布 f_X的样本，而 $\boldsymbol{z}_i \sim f_Z$ 则经过$\boldsymbol{g}$ 处理以生成伪造样本。为了区分真实样本和伪造样本，判别器在真实样本上的取值较大且为正，在生成的伪造样本上的取值较大且为负。因此，判别器通过最大化目标函数进行训练。换言之，我们要解决以下问题:

$$\boldsymbol{\theta}_d^*(\boldsymbol{\theta}_g) = \arg\max_{\boldsymbol{\theta}_d} \Pi(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) \quad \text{for any } \boldsymbol{\theta}_g. \tag{7.9}$$

请注意，判别器的最优参数取决于 $\boldsymbol{\theta}_g$。现在，为了欺骗判别器，生成器 $\boldsymbol{g}$ 试图最小化目标函数:

$$\boldsymbol{\theta}_g^* = \arg\min_{\boldsymbol{\theta}_g} \Pi(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d^*). \tag{7.10}$$

因此，训练 WGAN 相当于求解一个极小极大优化问题。需要注意的是，判别器和生成器以对抗的方式运行。也就是说，前者试图最大化目标函数，而后者则试图将其最小化。因此，这种网络被称为生成对抗网络。

实际上,我们需要在判别器损失中加入一个稳定项。因此，判别器的训练目标是
最大化

$$\Pi_c(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = \Pi(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) - \frac{\lambda}{\bar{N}} \sum_{i=1}^{\bar{N}} \left(\left\| \frac{\partial d}{\partial \hat{\boldsymbol{x}}}(\hat{\boldsymbol{x}}_i; \boldsymbol{\theta}_d) \right\| - 1 \right)^2 \tag{7.11}$$

其中 $\hat{\boldsymbol{x}}_i = \alpha \boldsymbol{x}_i + (1-\alpha)\boldsymbol{g}(\boldsymbol{z}_i; \boldsymbol{\theta}_g)$，而 α从 (0，1) 中的均匀随机变量采样。式 (7.11) 中的附加项称为梯度惩罚项，用于约束判别器 d 相对于其输入的梯度（范数）接近 1，从而使其成为 1-利普希茨函数。有关该项的更多细节，感兴趣的读者可参阅 [7]。

迭代算法1用于同时训练 $\boldsymbol{g}$ 和 d，这也称为交替最速下降，其中 η_d 和η_g 分别是判别器和生成器的学习率。注意，我们先对判别器执行 $K > 1$ 个优化步骤，然后对生成器执行一个优化步骤。这是因为我们希望首先求解内部最大化问题，使判别器能够区分真实样本和伪造样本。虽然取非常大的 K会使极小极大问题的求解更加准确，但也会使中等规模网络的训练算法在计算上难以处理。因此，在实践中， K 通常取 4 到 6。

极大极小优化问题很难求解，通常需要经过许多训练轮次才能达到收敛。另一种方法是，对训练数据进行小批量处理，并在每个训练轮次中处理许多小批量，以便在相对较少的训练轮次内完成数量相近的优化步骤。随着迭代的进行， d 越来越善于检测伪造样本，而 $\boldsymbol{g}$ 越来越善于生成能够骗过判别器的样本。

算法1： 训练生成对抗网络的算法

输入： $\boldsymbol{\theta}_d^0$，$\boldsymbol{\theta}_g^0$，K，$N_$训练轮次 η_d，η_g

对于 $n = 1, ..., N_$ 训练轮次 **执行**

$\hat{\boldsymbol{\theta}}_d \leftarrow \boldsymbol{\theta}_d^{(n-1)}$

对于 $k = 1, ..., K$ **do**

最大化更新：

$$\hat{\boldsymbol{\theta}}_d \leftarrow \hat{\boldsymbol{\theta}}_d + \eta_d \frac{\partial \Pi_c}{\partial \boldsymbol{\theta}_d}(\boldsymbol{\theta}_g^{(n-1)},\ \hat{}\boldsymbol{\theta}_d)$$

end

$\boldsymbol{\theta}_d^{(n)} \leftarrow \hat{\boldsymbol{\theta}}_d$

最小化更新：

$$\boldsymbol{\theta}_g^n \leftarrow \boldsymbol{\theta}_g^{(n-1)} - \eta_g \frac{\partial \Pi}{\partial \boldsymbol{\theta}_g}(\boldsymbol{\theta}_g^{(n-1)},\ \boldsymbol{\theta}_d^{(n)})$$

end

在无限容量（$N_{\boldsymbol{\theta}_g}$，$N_{\boldsymbol{\theta}_d} \to \infty$）、无限数据（$N_{\text{train}} \to \infty$）以及完美优化器的假设下，我们可以证明，生成分布 f_X^g **弱收敛**到目标分布 f_X [2]。这等价于说

$$\mathbb{E}_Z[\ell(\boldsymbol{g}(\boldsymbol{Z}; \boldsymbol{\theta}_g^*))] \longrightarrow \mathbb{E}_X[\ell(\boldsymbol{X})], \tag{7.12}$$

对于 Ω_X 上的每个连续有界函数 ℓ，即 $\ell \in C_b(\Omega_X)$。训练好生成对抗网络后，我们可以使用优化后的 $\boldsymbol{g}$，先从 $f_X^g \approx f_X$ 中采样 $\boldsymbol{z} \sim f_Z$，再将其输入生成器，得到样本 $\boldsymbol{x} = \boldsymbol{g}(\boldsymbol{z};\ \boldsymbol{\theta}_g^*)$。此外，由于上述弱收敛，生成样本的统计量（均值、方差等）将收敛到与 f_X 相关的真实统计量。

注释 7.3.1。这里我们提出几点重要说明：

1. GAN 训练完成后，我们通常只保留生成器，而不再需要判别器。训练判别器的主要目的是获得一个合适的 $\boldsymbol{g}$，使其能够生成逼真的样本。*2. WGAN* 名称中出现*"Wasserstein"*一词，是因为可以证明，求解极大极小问题等价于最小化 f_X^g 与 f_X *[2, 28]* 之间的 *Wasserstein-1*距离。*Wasserstein-1*距离是一种常用的度量，用于衡量两个概率测度之间的差异。*3.* 由于潜变量的维度 N_Z 通常远小于 Ω_X 中样本的维度N_X，训练得到的生成器也能为高维数据提供低维表示，这在若干下游任务 *[21, 22]* 中非常有用。

7.4 监督式概率深度学习算法

回想一下这个确定性问题：给定带标签/成对数据集

$$\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : \boldsymbol{x}_i \in \Omega_X \subset \mathbb{R}^{N_X},\ \boldsymbol{y} \in \Omega_Y \subset \mathbb{R}^{N_Y}\}_{i=1}^{N_{\text{train}}} \tag{7.13}$$

我们希望为一个未出现在 $\mathcal{S}$ 中的新 $\boldsymbol{x}$ 找到 $\boldsymbol{y}$。在前面的章节中，我们已经了解了如何使用神经网络来解决这类回归（或分类）问题。

现在我们来考虑这个问题的概率版本。我们假设分别使用随机变量 $\boldsymbol{X}$ 和 $\boldsymbol{Y}$ 对 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 进行建模。此外，假设式（7.13）中的配对样本来自未知的联合分布 f_{XY}。那么，给定一个实现值 $\boldsymbol{X} = \hat{\boldsymbol{x}}$，我们希望使用 $\mathcal{S}$ 来确定条件分布 $f_{Y|X}(\boldsymbol{y}|\hat{\boldsymbol{x}})$，并从中生成样本。

解决这一概率问题有几种常用方法，例如贝叶斯神经网络、变分推断、丢弃法或深度玻尔兹曼机。但我们将重点讨论生成对抗网络的一种扩展，它同样可以解决这类问题。

7.4.1 条件生成对抗网络

条件生成对抗网络最初在 [17] 中提出，用于学习条件分布。我们将讨论这类模型的一种特殊变体，即条件瓦瑟斯坦生成对抗网络（cWGAN），该模型在 [1], 中开发，并在 [25]中用于解决若干基于物理学的（逆）问题。

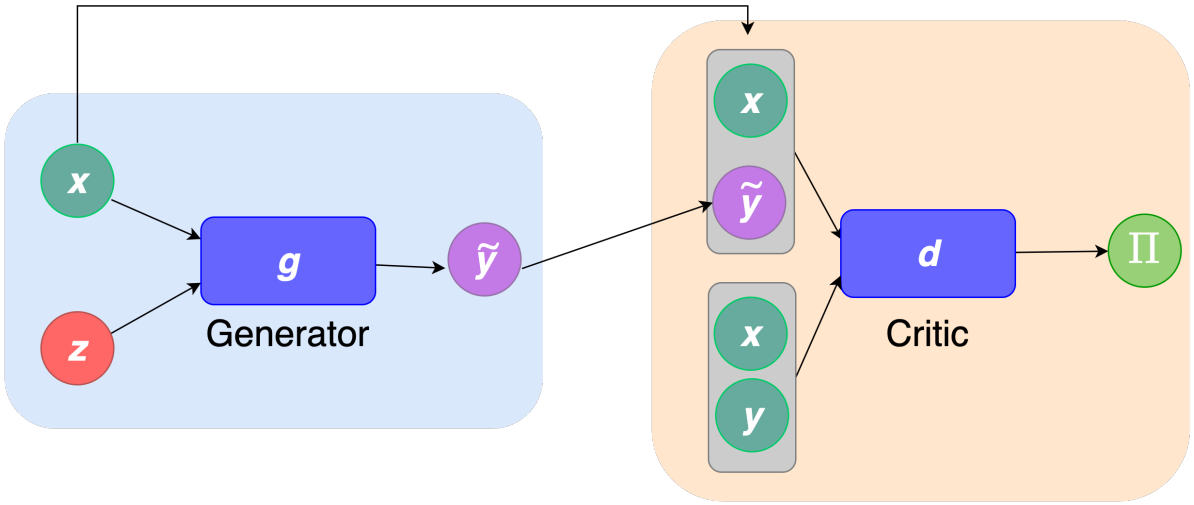


图 7.4：条件生成对抗网络示意图

图 7.4 展示了条件生成对抗网络的示意图。生成器是如下形式的网络：

$$\boldsymbol{g}(.;\boldsymbol{\theta}_g) : \Omega_Z \times \Omega_X \to \Omega_Y, \quad \boldsymbol{g} : (\boldsymbol{z}, \boldsymbol{x}) \mapsto \boldsymbol{y} \tag{7.14}$$

其中 $\boldsymbol{z} \sim f_Z$ 是潜变量。注意，与生成对抗网络不同，条件生成对抗网络中的生成器还将 $\boldsymbol{x}$作为输入。对于给定的 $\boldsymbol{X} = \hat{\boldsymbol{x}}$，对 $\boldsymbol{z} \sim f_Z$ 进行采样会生成许多来自某个诱导条件分布 $f^g_{Y|X}(\boldsymbol{y}|\hat{\boldsymbol{x}})$ 的 $\boldsymbol{y}$样本。目标是指定参数 $\boldsymbol{\theta}_g$，使得对于（几乎）每个 $\hat{\boldsymbol{x}}$ 的取值，$f^g_{Y|X}(\boldsymbol{y}|\hat{\boldsymbol{x}})$ 都能近似真实的条件 $f_{Y|X}(\boldsymbol{y}|\hat{\boldsymbol{x}})$。

判别器是如下形式的网络：

$$d(.;\boldsymbol{\theta}_d) : \Omega_X \times \Omega_Y \to \mathbb{R} \tag{7.15}$$

该函数经过训练，用于区分由真实联合分布 f_{XY} 生成的配对样本（$\boldsymbol{x}$，$\boldsymbol{y}$）与虚假配对（$\boldsymbol{x}$，$\hat{\boldsymbol{y}}$），其中 $\hat{\boldsymbol{y}}$ 由 $\boldsymbol{g}$ 根据（真实的） $\boldsymbol{x}$生成。

cWGAN 的目标函数为

$$\Pi(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = \underbrace{\frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} d(\boldsymbol{x}_i, \boldsymbol{y}_i; \boldsymbol{\theta}_d)}_{\text{critic value on real pairs}} - \underbrace{\frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} d(\boldsymbol{x}_i, \boldsymbol{g}(\boldsymbol{z}_i, \boldsymbol{x}_i; \boldsymbol{\theta}_g); \boldsymbol{\theta}_d)}_{\text{critic value on fake pairs}}. \tag{7.16}$$

与前面一样，训练判别器以最大化目标函数（由式 (7.9) 给出），同时训练生成器以最小化该函数（由式 (7.10) 给出）。此外，优化判别器时需要加入稳定化梯度惩罚项（见 [25]）。生成器和判别器采用前文针对生成对抗网络所述的交替最速下降算法进行训练。

在无限容量（$N_{\boldsymbol{\theta}_g}$, $N_{\boldsymbol{\theta}_d} \to \infty$）、无限数据（$N_{\text{train}} \to \infty$）且存在完美优化器的假设下，我们可以证明 [1]，对于给定的$\boldsymbol{X} = \hat{\boldsymbol{x}}$，生成的条件分布 $f^g_{Y|X}(\boldsymbol{y}|\hat{\boldsymbol{x}})$在弱意义上（取平均）收敛于目标条件分布 $f_{Y|X}(\boldsymbol{y}|\hat{\boldsymbol{x}})$。

参考文献

[1] J. Adler 和 O. Öktem，深度贝叶斯反演。https://arxiv.org/abs/1811.05910，2018年。[2] M. Arjovsky、S. Chintala 和 L. Bottou，*Wasserstein*生成对抗网络，载于《第34届国际机器学习会议论文集》，D. Precup 和 Y. W. Teh 编，第70卷《机器学习研究论文集》，澳大利亚悉尼国际会议中心，2017年8月6日至11日，PMLR，第214–223页。[3] R. Bischof 和 M. Kraus，物理信息深度学习的多目标损失平衡。http://rgdoi.net/10.13140/RG.2.2.20057.24169，2021年。[4] R. T. Q. Chen、Y. Rubanova、J. Bettencourt 和 D. Duvenaud，神经常微分方程。https://arxiv.org/abs/1806.07366，2018年。[5] T. Chen 和 H. Chen，具有任意激活函数的神经网络对非线性算子的通用逼近及其在动力系统中的应用，《IEEE神经网络汇刊》，6（1995年），第911–917页。[6] I. Goodfellow、J. Pouget-Abadie、M. Mirza、B. Xu、D. Warde-Farley、S. Ozair、A. Courville 和 Y. Bengio，生成对抗网络，载于《神经信息处理系统进展》，2014年，第2672–2680页。[7]I. Gulrajani、F. Ahmed、M. Arjovsky、V. Dumoulin 和 A. C. Courville，改进*Wasserstein*生成对抗网络的训练，载于《神经信息处理系统进展》，2017年，第5767–5777页。[8] K. He、X. Zhang、S. Ren 和 J. Sun，用于图像识别的深度残差学习，载于《2016年IEEE计算机视觉与模式识别会议（CVPR）》，2016年，第770–778页。[9] P. Kidger 和 T. Lyons，深而窄网络的通用逼近，载于《第三十三届学习理论会议论文集》，J. Abernethy 和 S. Agarwal 编，第125卷《机器学习研究论文集》，PMLR，2020年7月9日至12日，第2306–2327页。[10] D. P. Kingma 和 J. Ba，*Adam*：一种随机优化方法。https://arxiv.org/abs/1412.6980v9，2017年。[11] I. Lagaris、A. Likas 和 D. Papageorgiou，具有不规则边界的边值问题的神经网络方法，《IEEE神经网络汇刊》，11（2000年），第1041–1049页。

[12] *S. Lanthaler*、*S. Mishra* 和 *G. E. Karniadakis*，《*DeepONet* 的误差估计：无限维深度学习框架》，《数学及其应用汇刊》，*6*（*2022*年）。[13] *Z. Li*、*N. Kovachki*、*K. Azizzadenesheli*、*B. Liu*、*K. Bhattacharya*、*A. Stuart* 和 *A. Anandkumar*，《用于参数偏微分方程的傅里叶神经算子》。*https://arxiv.org/abs/2010.08895*，*2020*年。[14] *L. Lu*、*P. Jin*、*G. Pang*、*Z. Zhang* 和 *G. E. Karniadakis*，《基于算子通用逼近定理的 *DeepONet* 非线性算子学习》，《自然·机器智能》，*3*（*2021*年），第 *218–229* 页。[15] *A. L. Maas*、*A. Y. Hannun*、*A. Y. Ng* 等，《整流非线性可改善神经网络声学模型》，载于《国际机器学习大会论文集》，第 *30* 卷，*2013*年。[16] *L. McClenny* 和 *U. Braga-Neto*，《利用软注意力机制的自适应物理信息神经网络》。*https://arxiv.org/abs/2009.04544*，*2020*年。[17] *M. Mirza* 和 *S. Osindero*，《条件生成对抗网络》。*https://arxiv. org/abs/1411.1784*，*2014*年。[18] *S. Mishra* 和 *R. Molinaro*，《物理信息神经网络逼近偏微分方程的泛化误差估计》，《*IMA*数值分析杂志》，（*2022*年）。[19] *A. Nemirovski*、*A. Juditsky*、*G. Lan* 和 *A. Shapiro*，《随机规划的鲁棒随机逼近方法》，《*SIAM*优化杂志》，*19*（*2009*年），第 *1574–1609* 页。[20] *D. Patel*、*D. Ray*、*M. R. A. Abdelmalik*、*T. J. R. Hughes* 和 *A. A. Oberai*，《变分拟态算子网络》。*https://arxiv.org/abs/2209.12871*，*2022*年。[21] *D. V. Patel* 和 *A. A. Oberai*，《用于量化监督学习不确定性的基于生成对抗网络的先验》，《*SIAM/ASA*不确定性量化杂志》，*9*（*2021*年），第 *1314–1343* 页。[22] *D. V. Patel*、*D. Ray* 和 *A. A. Oberai*，《利用深度生成先验求解基于物理的贝叶斯反问题》，《计算机方法与应用力学和工程》，*400*（*2022*年），第 *115428* 页。[23] *A. Pinkus*，《神经网络中多层感知机模型的逼近理论》，《数值学报》，*8*（*1999*年），第 *143–195* 页。[24] *M. Raissi*、*P. Perdikaris* 和 *G. Karniadakis*，《物理信息神经网络：求解涉及非线性偏微分方程的正问题与反问题的深度学习框架》，《计算物理杂志》，*378*（*2019*年），第 *686–707* 页。[25] *D. Ray*、*H. Ramaswamy*、*D. V. Patel* 和 *A. A. Oberai*，《条件生成对抗网络在基于物理的反问题后验推断中的效能与泛化能力》。*https://arxiv.org/abs/2202.07773*，*2022*年。[26] *O. Ronneberger*、*P. Fischer* 和 *T. Brox*，《*U-Net*：用于生物医学图像分割的卷积网络》，载于《医学图像计算与计算机辅助干预——医学图像计算与计算机辅助干预大会 *2015*》，*N. Navab*、*J. Hornegger*、*W. M. Wells* 和 *A. F. Frangi* 编，尚，*2015*年，施普林格国际出版，第 *234–241* 页。

[27] V. Sitzmann、J. N. P. Martel、A. W. Bergman、D. B. Lindell 和 G. Wetzstein，具有周期激活函数的隐式神经表示。https://arxiv.org/abs/2006.09661，2020年。[28] C. Villani，《最优传输：旧与新》，数学科学基础丛书，施普林格·柏林·海德堡，2008年。[29] S. Wang、Y. Teng 和 P. Perdikaris，理解并缓解物理信息神经网络中的梯度流病态，《SIAM科学计算杂志》，43（2021年），第 A3055–A3081 页。[30] S. Wang、H. Wang 和 P. Perdikaris，利用物理信息 *DeepONet* 学习参数偏微分方程的解算子，《科学进展》，7（2021年）。[31] L. Wu、C. Ma 和 W. E，过参数化学习中随机梯度下降如何选择全局极小值：动力学稳定性视角，载于《神经信息处理系统进展》，S. Bengio、H. Wallach、H. Larochelle、K. Grauman、N. Cesa-Bianchi 和 R. Garnett 编，第 31 卷，Curran Associates, Inc.，2018年。[32] M. Yang、K. Yu、C. Zhang、Z. Li 和 K. Yang，用于街景语义分割的 *DenseASPP*，载于《2018年 IEEE/CVF计算机视觉与模式识别会议》，2018年，第 3684–3692 页。[33] D. Yarotsky 和 A. Zhevnerchuk，深度神经网络逼近率的相图。https://arxiv.org/abs/1906.09477，2019年。