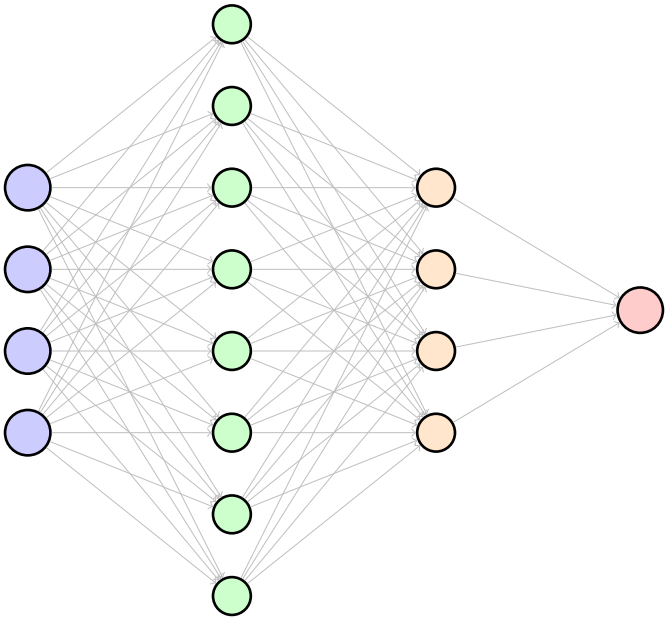
深度学习核心理论与技术

神经网络 · CNN · RNN · Transformer · 优化技术

深入理解深度学习的核心原理与实践应用



深度学习核心理论与技术

2026 年 1 月 6 日

目录

Part I

第一部分：神经网络基础

1 引言

深度学习（Deep Learning）是机器学习的一个子领域，通过构建具有多个隐藏层的神经网络来学习数据的层次化表示。深度学习在计算机视觉、自然语言处理、语音识别、游戏 AI 等领域取得了突破性进展，成为当前人工智能领域最活跃的研究方向之一。

深度学习的核心优势：

- **自动特征提取**：无需人工设计特征，网络能够自动学习数据的层次化特征表示
- **处理复杂非线性关系**：通过多层非线性变换，能够建模高度复杂的函数关系
- **端到端学习**：直接从原始输入学习到最终输出，减少中间环节的信息损失
- **强大的表示能力**：深度网络具有强大的函数逼近能力，能够学习任意复杂的映射关系
- **迁移学习能力**：预训练模型可以迁移到相关任务，提高学习效率

深度学习面临的挑战：

- **数据需求量大**：通常需要大量标注数据才能训练出有效的模型
- **计算资源消耗**：训练深度网络需要强大的计算资源（GPU/TPU）
- **可解释性不足**：深度网络的决策过程往往缺乏可解释性，难以理解其内部机制
- **过拟合风险**：模型容量大，容易在训练数据上过拟合
- **实时性要求**：某些应用场景对推理速度有严格要求
- **数据稀疏性**：在某些领域（如医疗、金融）高质量数据稀缺

本文档系统性地介绍深度学习的核心理论、主要架构和关键技术，涵盖神经网络基础、深度网络架构、优化技术、表示学习、注意力机制以及强化学习等内容。

2 神经网络基础

2.1 感知机

感知机（Perceptron）是最简单的神经网络模型，由 Frank Rosenblatt 于 1957 年提出，是神经网络和深度学习的起点。

定义 2.1 (感知机). *感知机是一个二分类线性分类模型，其输入为特征向量* $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$，*输出为类别标签* $y \in \{-1, +1\}$。

感知机的数学表达式为：

$$f(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & if\ \mathbf{w}^T\mathbf{x} + b < 0 \end{cases} \tag{1}$$

其中， $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$ *是权重向量，* b *是偏置项。*

感知机的几何解释：感知机实际上是在特征空间中构造一个超平面 $\mathbf{w}^T\mathbf{x} + b = 0$，将数据分为两类。权重向量 $\mathbf{w}$ 垂直于超平面，指向正类区域。

感知机学习算法：使用错误驱动的学习规则，当分类错误时更新权重：

Algorithm 1 感知机学习算法

Require: 训练数据集 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$，学习率 η

Ensure: 权重向量 $\mathbf{w}$ 和偏置 b

初始化 $\mathbf{w} = \mathbf{0}$，$b = 0$
repeat
  for 每个样本 $(\mathbf{x}_i, y_i)$ **do**
    if $y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$
      $b \leftarrow b + \eta y_i$
    end if
  end for
until 没有分类错误

感知机的局限性：感知机只能解决线性可分问题。对于线性不可分问题（如异或问题），单层感知机无法解决，这促使了多层感知机的提出。

2.2 多层感知机

多层感知机（Multi-Layer Perceptron, MLP）通过引入隐藏层和激活函数，能够解决非线性分类问题。

定义 2.2 (多层感知机). 多层感知机是由多个全连接层组成的神经网络，每层包含多个神经元。一个 L 层的 MLP 可以表示为:

$$\mathbf{h}^{(0)} = \mathbf{x} \tag{2}$$

$$\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}), \quad l = 1, 2, \ldots, L-1 \tag{3}$$

$$\mathbf{y} = \mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)} \tag{4}$$

其中，$\mathbf{W}^{(l)}$ 是第 l 层的权重矩阵，$\mathbf{b}^{(l)}$ 是偏置向量，σ 是激活函数。

2.2.1 隐藏层详解

概念解释: 隐藏层（Hidden Layer）是人工神经网络中的核心组成部分，位于输入层和输出层之间。之所以称为" 隐藏"，是因为这些层的输出不直接暴露给外部世界（不像输入层接收原始数据、输出层给出最终预测），而是用于内部特征表示的学习。

数学表示:

从数学和结构上看:

- **输入层**: 接收原始特征，记为 $\mathbf{x} \in \mathbb{R}^d$
- **隐藏层**: 对输入进行非线性变换，提取更高层次的抽象特征。第 l 层的输出为:

$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right) \tag{5}$$

  其中:

  - $\mathbf{W}^{(l)}$ 是权重矩阵（可学习参数）
  - $\mathbf{b}^{(l)}$ 是偏置向量
  - $\sigma(\cdot)$ 是激活函数（如 ReLU、Sigmoid、Tanh）
  - $\mathbf{h}^{(0)} = \mathbf{x}$

- **输出层**: 基于最后一层隐藏层的表示，生成最终预测

关键作用: 隐藏层通过多层非线性组合，使网络能够拟合复杂的函数（如图像识别、自然语言理解等）。没有隐藏层（即只有输入 $\rightarrow$ 输出），网络就退化为线性模型，表达能力极其有限。

配置隐藏层:

当你在代码中配置隐藏层时，实际上在配置:

- **神经元数量**：每层的神经元数量（即维度），也叫” 隐藏单元数” 或” 宽度（width）”
- **权重矩阵形状**：如 100×256 表示输入 100 维，输出 256 维
- **激活函数**：指定该隐藏层使用的激活函数（如 ReLU）
- **层数**：决定了网络的深度（depth）

示例：

```
# 配置示例：输入100维 → 隐藏层1：256个神经元 → 隐藏层2：128个神经元 → 输
    出1维
# 这实际上定义了：
# - 第1层：100 × 256 的权重矩阵，256个神经元
# - 第2层：256 × 128 的权重矩阵，128个神经元
# - 输出层：128 × 1 的权重矩阵，1个神经元
```

Listing 1: 隐藏层配置示例

超参数影响：

- 神经元越多、层数越深 → 模型越复杂，拟合能力越强，但也更容易过拟合
- 在深度学习中，常用 2 4 层 MLP，每层 128 512 个神经元

特征学习的层次性：

每一个” 隐藏层” 其实就是一个向量空间变换器：把低层特征映射到高层语义特征。例如：

- 第 1 层可能学到” 边缘、线条” 等低级特征
- 第 2 层可能学到” 形状、纹理” 等中级特征
- 第 3 层可能学到” 物体部件” 等高级特征
- 最终输出层预测” 完整物体” 的类别

2.2.2 输入层、隐藏层、输出层的明确区分

核心定义：

隐藏层（Hidden Layer）的定义是：既不是输入层，也不是输出层的所有中间层。

- **输入层（Input Layer）**：负责接收原始数据（如特征向量、像素值等），不是隐藏层。

- **输出层（Output Layer）**：负责产生最终预测（如分类概率、回归值等），也不是隐藏层。
- **隐藏层**：只有夹在输入层和输出层之间的层才叫隐藏层。

举例说明：

对于一个 3 层网络（输入层 → 隐藏层 → 输出层），只有中间那层是隐藏层。如果网络有更多层，如输入层 → 隐藏层 1 → 隐藏层 2 → 输出层，那么隐藏层 1 和隐藏层 2 都是隐藏层。

术语使用规范：

- **输入层**：不是隐藏层。无参数，仅传递数据。
- **中间层**：是隐藏层。如 `Linear(256,128) + ReLU`，包含可学习参数。
- **输出层**：不是隐藏层。如 `Linear(128, 1)`，用于最终映射。

重要说明：

虽然输出层通常是全连接层（用 Linear 实现，有权重和偏置），但我们不会称它为” 隐藏层”，因为它的功能是输出，不是隐藏表示。当说” 模型有 2 个隐藏层” 时，指的是中间有 2 个可学习的层，不包括输入和输出。

2.2.3 编码（Encoding）与输入层的关系

核心概念区分：

- **编码（Encoding）**：将原始数据（如文本、类别、SQL 语句、图像）转换为数值向量的过程。这是预处理步骤，发生在模型之外或模型最前端。
- **输入层（Input Layer）**：神经网络的第一层，接收已经编码好的数值向量作为输入。它本身不做编码，只是” 起点”。

数学表示：

编码过程：$\mathbf{x} = \text{encode}(\text{原始输入})$，其中 $\mathbf{x} \in \mathbb{R}^d$ 是编码后的数值向量。

输入层：$\mathbf{h}^{(0)} = \mathbf{x}$，即输入层就是编码后的向量本身，不做任何变换。

举例说明：

例 2.1 (文本分类任务)**.**

- **原始输入**：文本*”Hello World”*

- **编码过程**：
  - 分词：*["Hello", "World"]*
  - 词嵌入（*Embedding*）：将每个词转换为向量
  - 结果：$\mathbf{x} \in \mathbb{R}^d$（如 $d = 128$）
- **输入层**：接收这个 $\mathbf{x}$ 向量，不做任何变换，直接传给第一个全连接层

例 2.2 (图像分类任务).

- **原始输入**：一张 28×28 的图片（像素值 *0 255*）
- **编码过程**：
  - 归一化到 $[0, 1]$
  - 展平成 784 维向量
- **输入层**：接收这 784 维向量，直接送入第一个全连接层或卷积层

关键结论：

- **" 输入层就是做 encoding"**：不准确。输入层不执行 encoding，它只是接收已经编码好的数据。
- **"encoding 的结果作为输入层的值"**：正确。这是标准流程，编码后的向量 $\mathbf{x}$ 就是输入层的值。
- **"Embedding 层是输入层"**：不准确。`nn.Embedding` 是可学习的 encoding 模块，属于模型的第一层可学习模块，但它已经超出了" 输入层" 的范畴。

代码示例：

```
import torch
import torch.nn as nn

# 假设原始输入是词索引: [2, 5, 1]
embedding = nn.Embedding(num_embeddings=1000, embedding_dim=64)  # 这是
    encoding!
fc1 = nn.Linear(64, 128)  # 这才是真正的第一层（隐藏层）

x = torch.tensor([2, 5, 1])
emb = embedding(x)  # shape: (3, 64) ← encoded vector
out = fc1(emb)  # 输入给全连接层
```

Listing 2: 编码与输入层在代码中的体现

总结:

输入层不等于编码，但输入层的值等于编码的结果。编码是数据预处理或模型前端的转换步骤，目的是把原始信息变成神经网络能处理的数值向量。输入层只是这个向量进入网络的入口，本身不做计算（无参数）。

更准确的说法是：”我们先把数据 encode 成向量，然后喂给神经网络的输入层。”

2.2.4 输入层不做线性变换

核心结论:

输入层本身不做任何线性变换（也不做非线性变换）。输入层只是一个”数据入口”或”占位符”，它没有可学习的参数（如权重 $\mathbf{W}$ 和偏置 $\mathbf{b}$），也不执行任何计算（包括线性变换）。

什么是线性变换:

在线性代数和神经网络中，线性变换（更准确说是仿射变换）指:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{6}$$

其中:

- $\mathbf{W}$: 权重矩阵（可学习参数）
- $\mathbf{b}$: 偏置向量（可学习参数）

这是全连接层（Linear/Dense Layer）的核心操作。只有包含可学习参数并执行上述运算的层，才做了线性变换。

输入层的本质:

输入层等于原始特征向量 $\mathbf{x}$ 本身。它的作用仅仅是将预处理或编码后的数据传递给网络的第一层可学习模块。

在数学上，输入层就是 $\mathbf{h}^{(0)} = \mathbf{x}$。没有 $\mathbf{W}$，没有 $\mathbf{b}$，没有运算。

类比说明:

输入层就像函数的参数，而不是函数体内的计算:

```
def neural_network(x):  # ← x 就是"输入层"
    z1 = W1 @ x + b1  # ← 第一个全连接层（才开始线性变换）
    a1 = relu(z1)
    ...
```

Listing 3: 输入层与计算层的区别

这里 **x** 本身不做任何事，只是被使用。

常见误解澄清：

- **误解：” 输入层是第一个 Linear 层”**
  澄清：错！第一个 `Linear` 层是第一个隐藏层（或输出层），不是输入层。输入层没有参数，而 `Linear` 层有可学习的权重和偏置。
- **误解：” 输入层有神经元，所以会计算”**
  澄清：错！输入” 神经元” 只是对输入维度的形象说法（如”100 维输入” 说成”100 个输入神经元”），但它们没有激活函数、没有权重，不进行计算。输入层只是数据的载体。
- **误解：”Embedding 是输入层”**
  澄清：不准确。`nn.Embedding` 是一个可学习的 encoding 层，属于模型的第一层可学习模块，但它已经超出了” 输入层” 的范畴。输入层是无参数的，而 `Embedding` 有可学习的参数。

图示说明：

$$\text{原始数据} \to [\text{Encoding}] \to \mathbf{x} \in \mathbb{R}^{100}\ \text{（这就是输入层，无计算）}$$
$$\downarrow$$
$$[\texttt{Linear}(\mathbf{W}_1, \mathbf{b}_1)]\ \text{（第一个全连接层，开始线性变换）}$$
$$\downarrow$$
$$\text{ReLU}(...)$$

特殊情况说明：

在极少数文献或框架中，有人把标准化（Normalization）或固定投影放在最前面，例如：

```
x_norm = (x - mean) / std  # 数据标准化
z = W @ x_norm + b
```

Listing 4: 标准化预处理

但请注意：标准化是预处理，不属于网络参数。即使把它写进模型，也通常视为数据 pipeline 的一部分，而非” 输入层的计算”。真正的” 输入层” 在理论和实践中都被定义为无参数、无变换的数据载体。

总结：

- **输入层会做线性变换吗?**
  不会。输入层没有权重 **W** 和偏置 **b**，不执行任何运算。

- **线性变换从哪开始?**
  从第一个全连接层（`Linear/Dense Layer`）开始。这是网络中的第一个可学习模块。
- **输入层的作用是什么?**
  接收已编码的数值向量，作为网络计算的起点。它只是数据的入口，不做任何变换。

所以可以明确地说：”输入层只是数据的入口，真正的计算从第一层全连接层才开始。”

2.3 激活函数详解

激活函数（Activation Function）是神经网络中的关键组件，它引入非线性变换，使网络能够学习复杂的非线性函数关系。没有激活函数，多层神经网络就退化为单层线性模型。

2.3.1 常用激活函数

1. Sigmoid 函数

数学公式:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

导数:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{8}$$

输出范围: $(0, 1)$

用处:

- 二分类任务的输出层，输出概率
- 需要将输出映射到 $(0, 1)$ 区间的场景
- 门控机制（如 LSTM、GRU）

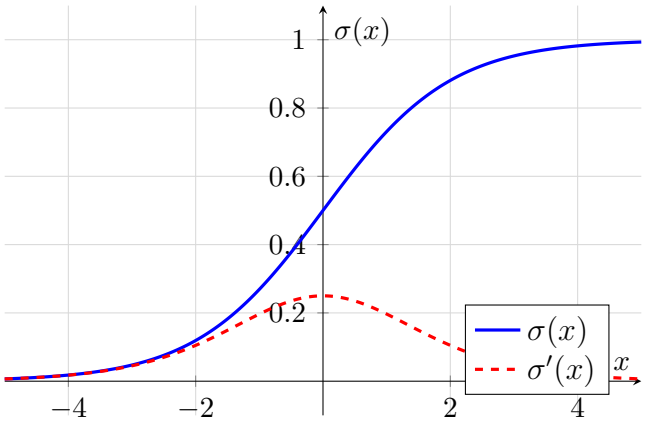
为什么用 Sigmoid:

- **概率解释**: 输出可以解释为概率，便于理解
- **平滑可导**: 处处可导，梯度计算方便
- **历史原因**: 早期神经网络常用，与生物神经元激活模式相似

缺点:

- **梯度消失**：当输入很大或很小时，梯度接近 0，导致深层网络难以训练
- **非零中心**：输出均值不为 0，导致梯度更新效率低
- **计算成本**：涉及指数运算，计算较慢

函数图像：



2. Tanh 函数（双曲正切）

数学公式：

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1 \tag{9}$$

导数：

$$\tanh'(x) = 1 - \tanh^2(x) \tag{10}$$

输出范围：$(-1, 1)$

用处：

- RNN 中的隐藏层激活函数
- 需要零中心化输出的场景
- 数据归一化到 $(-1, 1)$ 区间

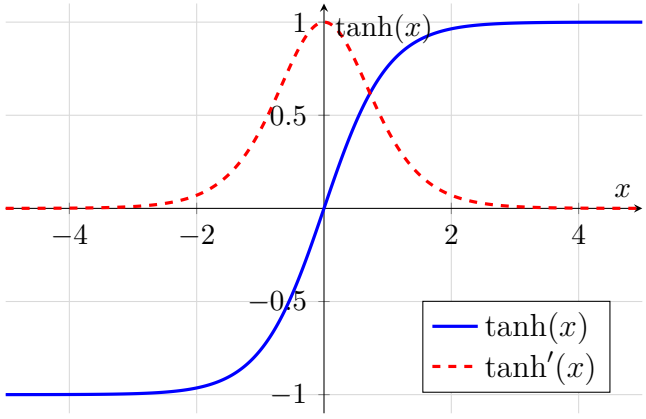
为什么用 Tanh：

- **零中心化**：输出均值为 0，梯度更新更高效
- **比 Sigmoid 更好**：在大多数情况下，tanh 比 sigmoid 表现更好
- **对称性**：关于原点对称，数学性质更好

缺点：

- **梯度消失**：与 Sigmoid 类似，在饱和区域梯度接近 0
- **计算成本**：涉及指数运算

函数图像：



3. ReLU 函数（Rectified Linear Unit）

数学公式：

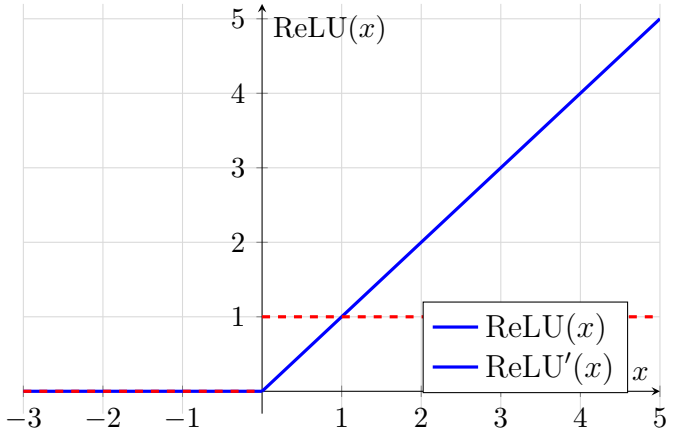
$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{11}$$

导数：

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{12}$$

输出范围：$[0, +\infty)$

函数图像：



用处：

- 深度神经网络隐藏层的首选激活函数
- CNN、MLP 等大多数现代神经网络架构
- 需要稀疏激活的场景

为什么用 ReLU：

- **缓解梯度消失**：在正区间梯度恒为 1，不会衰减
- **计算高效**：只需比较和选择操作，计算速度快
- **稀疏激活**：约 50% 的神经元会被激活，提高计算效率
- **生物学合理性**：模拟生物神经元的单侧抑制特性

缺点：

- **死亡 ReLU 问题**：如果神经元输出始终为负，梯度为 0，无法更新
- **非零中心**：输出均值不为 0
- **无界**：输出可以无限大，可能导致数值不稳定

4. Leaky ReLU

数学公式：

$$\text{LeakyReLU}(x) = \max(\alpha x, x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{13}$$

其中 α 是小的正数（通常取 0.01）。

导数：

$$\text{LeakyReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0 \end{cases} \tag{14}$$

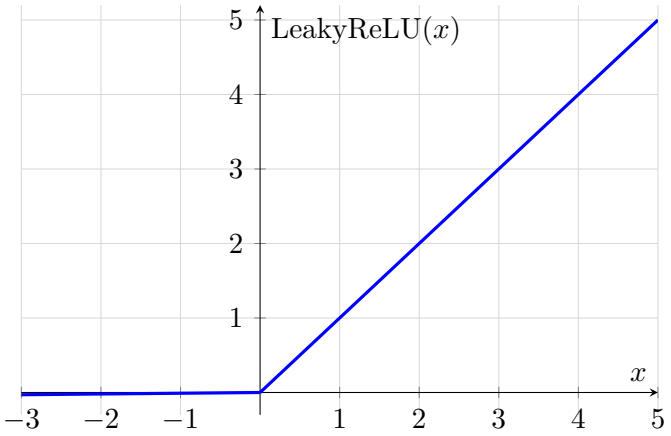
用处：

- 解决 ReLU 的死亡神经元问题
- 需要负值输出的场景
- 对梯度消失敏感的网络

为什么用 Leaky ReLU：

- **避免死亡神经元**：负区间有小的梯度，神经元不会完全” 死亡”
- **保持 ReLU 优点**：正区间仍保持线性，计算高效
- **更好的梯度流**：负区间也有梯度，信息可以反向传播

函数图像（$\alpha = 0.01$）：



5. ELU（Exponential Linear Unit）

数学公式：

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \tag{15}$$

其中 α 通常取 1.0。

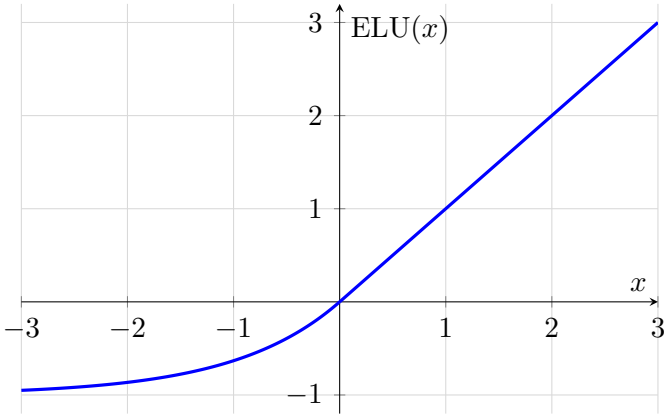
用处：

- 需要平滑负值输出的场景
- 对噪声敏感的任务
- 需要零均值输出的网络

为什么用 ELU：

- **平滑性**：负区间平滑，避免 ReLU 的不连续性
- **零均值**：输出均值接近 0，有助于训练
- **负值处理**：负区间有梯度，避免死亡神经元

函数图像（$\alpha = 1.0$）：



6. Softmax 函数

数学公式：

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}} \tag{16}$$

其中 C 是类别数，$\mathbf{x} \in \mathbb{R}^C$。

输出范围：$(0, 1)$，且 $\sum_{i=1}^{C} \text{softmax}(\mathbf{x})_i = 1$

用处：

- 多分类任务的输出层
- 需要输出概率分布的场景
- 注意力机制中的注意力权重计算

为什么用 Softmax：

- **概率分布**：输出是有效的概率分布，总和为 1
- **与交叉熵配合**：与交叉熵损失函数配合使用，梯度形式简单
- **可解释性**：输出可以解释为各类别的概率

2.3.2 激活函数选择指南

激活函数选择指南：

- **Sigmoid**：

  - 适用场景：二分类输出层、门控机制
  - 选择原因：输出概率、历史原因

- **Tanh**：
  - 适用场景：RNN 隐藏层、需要零中心化
  - 选择原因：零均值、比 Sigmoid 更好

- **ReLU**：
  - 适用场景：深度网络隐藏层（首选）
  - 选择原因：缓解梯度消失、计算高效

- **Leaky ReLU**：
  - 适用场景：解决死亡神经元问题
  - 选择原因：负区间有梯度

- **ELU**：
  - 适用场景：需要平滑负值输出
  - 选择原因：平滑、零均值

- **Softmax**：
  - 适用场景：多分类输出层
  - 选择原因：概率分布、与交叉熵配合

应用场景：

例 2.3 (手写数字识别). *MNIST 数据集是经典的图像分类任务，使用 MLP 可以达到较高的准确率。输入是 $28 \times 28 = 784$ 维的像素向量，输出是 10 个类别的概率分布。*

2.4 前向传播

前向传播（Forward Propagation）是神经网络从输入到输出的计算过程。

计算复杂度：对于 L 层网络，每层有 n_l 个神经元，前向传播的时间复杂度为 $O(\sum_{l=1}^{L} n_{l-1} n_l)$。

Algorithm 2 前向传播算法

Require: 输入 $\mathbf{x}$，网络参数 $\{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L}$

Ensure: 输出 $\mathbf{y}$

$\mathbf{h}^{(0)} \leftarrow \mathbf{x}$

for $l = 1$ to $L-1$ **do**

  $\mathbf{z}^{(l)} \leftarrow \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$

  $\mathbf{h}^{(l)} \leftarrow \sigma(\mathbf{z}^{(l)})$

end for

$\mathbf{z}^{(L)} \leftarrow \mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}$

$\mathbf{y} \leftarrow \mathbf{z}^{(L)}$（或应用 softmax 等函数）

2.5 反向传播

反向传播（Backpropagation）是训练神经网络的核心算法，通过链式法则计算损失函数对网络参数的梯度。

定义 2.3 (反向传播). *给定损失函数 $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$，反向传播算法计算梯度 $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}$ 和 $\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}}$。*

对于输出层：

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \tag{17}$$

对于隐藏层 $l = L-1, L-2, \ldots, 1$：

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(l)}} = (\mathbf{W}^{(l+1)})^T \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l+1)}} \tag{18}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(l)}} \odot \sigma'(\mathbf{z}^{(l)}) \tag{19}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} (\mathbf{h}^{(l-1)})^T \tag{20}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} \tag{21}$$

其中，$\odot$ 表示逐元素相乘（Hadamard 积），σ' 是激活函数的导数。

梯度消失问题：在深层网络中，梯度在反向传播过程中可能指数级衰减，导致底层参数难以更新。使用 ReLU 激活函数、残差连接、批量归一化等技术可以缓解这一问题。

2.6 损失函数

损失函数（Loss Function）用于衡量模型预测值与真实值之间的差异，是训练神经网络的核心目标函数。选择合适的损失函数对模型性能至关重要。

Algorithm 3 反向传播算法

Require: 前向传播的输出 $\mathbf{y}$，真实标签 $\hat{\mathbf{y}}$，损失函数 $\mathcal{L}$

Ensure: 梯度 $\{\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}}\}_{l=1}^{L}$

1: 计算输出层梯度：$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(L)}}$

2: **for** $l = L-1$ down to 1 **do**

3: $\quad \delta^{(l)} = (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(\mathbf{z}^{(l)})$

4: $\quad \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} (\mathbf{h}^{(l-1)})^T$

5: $\quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$

6: **end for**

2.6.1 回归任务的损失函数

均方误差（Mean Squared Error, MSE）：

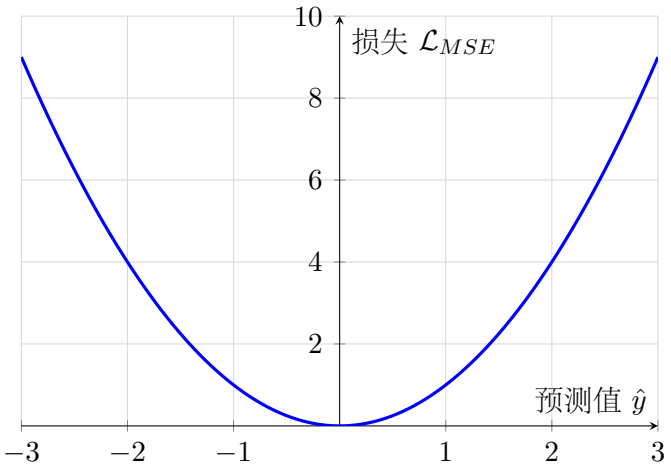
$$\mathcal{L}_{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{22}$$

其中，y_i 是真实值，$\hat{y}_i$ 是预测值，n 是样本数量。

特点：

- 对大误差敏感，惩罚较大
- 梯度与误差成正比，训练稳定
- 适用于回归问题，假设误差服从高斯分布

函数图像（假设真实值 $y = 0$）：



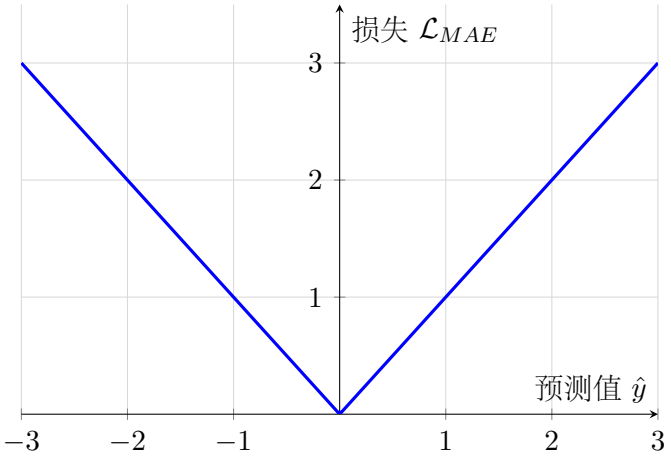
平均绝对误差（Mean Absolute Error, MAE）：

$$\mathcal{L}_{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{23}$$

特点：

- 对大误差不敏感，对异常值更鲁棒
- 梯度为常数（± 1），训练可能不稳定
- 适用于需要鲁棒性的回归问题

函数图像（假设真实值 $y = 0$）：

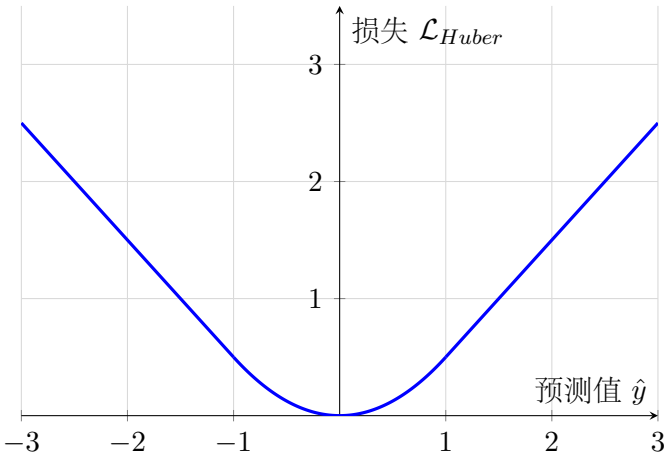


Huber Loss：结合 MSE 和 MAE 的优点：

$$\mathcal{L}_{Huber}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta |y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{24}$$

其中，δ 是超参数（通常取 1.0）。当误差小于 δ 时，使用 MSE；否则使用 MAE。

函数图像（假设真实值 $y = 0$，$\delta = 1.0$）：



2.6.2 分类任务的损失函数

交叉熵损失（Cross-Entropy Loss）：对于二分类问题：

$$\mathcal{L}_{CE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \tag{25}$$

对于多分类问题（C 个类别）：

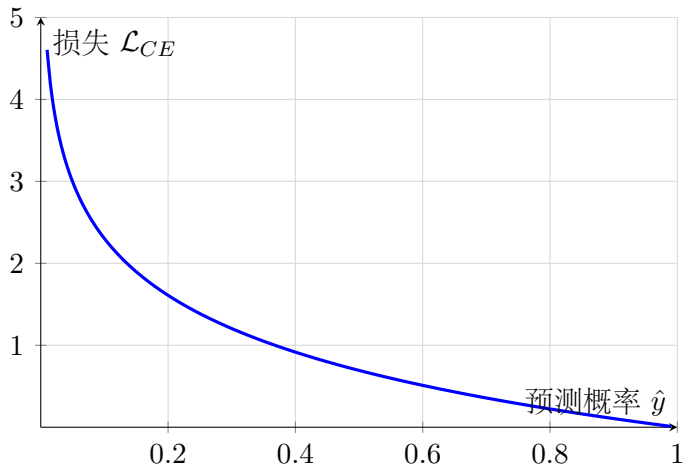
$$\mathcal{L}_{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{26}$$

其中，$y_{i,c}$ 是真实标签的 one-hot 编码，$\hat{y}_{i,c}$ 是模型预测的概率。

特点：

- 与 softmax 激活函数配合使用效果最佳
- 梯度形式简单：$\frac{\partial \mathcal{L}_{CE}}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$
- 适用于分类问题，假设输出服从多项分布

函数图像（二分类，真实标签 $y=1$）：



Focal Loss：解决类别不平衡问题：

$$\mathcal{L}_{Focal} = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{27}$$

其中，p_t 是模型预测的正确类别的概率，α_t 是平衡因子，γ 是聚焦参数（通常 $\gamma=2$）。

特点：

- $(1-p_t)^{\gamma}$ 项降低易分类样本的权重
- 专注于难分类样本，提高模型性能
- 在目标检测任务中广泛应用（如 RetinaNet）

2.6.3 其他损失函数

对比损失（Contrastive Loss）：用于学习相似性：

$$\mathcal{L}_{Contrastive} = \begin{cases} \frac{1}{2}d^2 & \text{if } y = 1 \text{ (相似)} \\ \frac{1}{2}\max(0, m-d)^2 & \text{if } y = 0 \text{ (不相似)} \end{cases} \tag{28}$$

其中，d 是两个样本的欧氏距离，m 是边界参数（margin）。

三元组损失（Triplet Loss）：用于度量学习：

$$\mathcal{L}_{Triplet} = \max(0, d(a,p) - d(a,n) + m) \tag{29}$$

其中，a 是锚样本（anchor），p 是正样本（positive），n 是负样本（negative），$d(\cdot,\cdot)$ 是距离函数，m 是边界。

应用场景：

- 人脸识别：FaceNet 使用三元组损失
- 图像检索：学习图像的相似性表示
- 推荐系统：学习用户和物品的嵌入

感知损失（Perceptual Loss）：用于图像生成任务：

$$\mathcal{L}_{Perceptual} = \sum_{l} \lambda_l \|\phi_l(\hat{I}) - \phi_l(I)\|_2^2 \tag{30}$$

其中，ϕ_l 是预训练网络（如 VGG）的第 l 层特征，λ_l 是权重。

应用场景：

- 图像超分辨率：SRGAN
- 风格迁移：保持内容的同时改变风格
- 图像修复：生成缺失区域

2.6.4 损失函数的选择原则

1. **任务类型**：
    - 回归任务：MSE、MAE、Huber Loss
    - 二分类：二元交叉熵

- 多分类：多元交叉熵
- 多标签分类：二元交叉熵（每个类别独立）

2. **数据分布**：

- 类别不平衡：Focal Loss、加权交叉熵
- 异常值多：MAE、Huber Loss
- 高斯噪声：MSE

3. **训练稳定性**：

- 梯度爆炸风险：使用梯度裁剪
- 梯度消失：选择合适的激活函数和损失函数组合

4. **应用需求**：

- 需要可解释性：使用简单的损失函数
- 需要特定属性：使用定制损失函数（如感知损失）

注 2.1. *在实际应用中，可以组合多个损失函数：*

$$\mathcal{L}_{Total} = \lambda_1 \mathcal{L}_1 + \lambda_2 \mathcal{L}_2 + \cdots + \lambda_k \mathcal{L}_k \tag{31}$$

其中，λ_i 是各损失函数的权重，需要根据任务需求调整。

2.6.5 损失函数使用原因总结

常见损失函数的使用原因：

- **MSE（均方误差）**：
  - 主要用途：回归任务
  - 为什么用它：假设误差服从高斯分布，对大误差敏感，梯度稳定
- **MAE（平均绝对误差）**：
  - 主要用途：回归任务（异常值多）
  - 为什么用它：对异常值鲁棒，梯度为常数
- **Huber Loss**：
  - 主要用途：回归任务（平衡鲁棒性和稳定性）

  - 为什么用它：结合 MSE 和 MAE 优点，小误差用 MSE，大误差用 MAE

- **交叉熵（Cross-Entropy）**：
  - 主要用途：分类任务
  - 为什么用它：与 softmax 配合，梯度形式简单，假设输出为多项分布

- **Focal Loss**：
  - 主要用途：类别不平衡的分类任务
  - 为什么用它：降低易分类样本权重，专注于难样本

- **对比损失（Contrastive Loss）**：
  - 主要用途：相似性学习
  - 为什么用它：学习样本间的相似性关系

- **三元组损失（Triplet Loss）**：
  - 主要用途：度量学习
  - 为什么用它：学习样本间的相对距离关系

- **感知损失（Perceptual Loss）**：
  - 主要用途：图像生成
  - 为什么用它：在特征空间计算损失，更符合人类感知

2.7 评估函数（评估指标）

评估函数（Evaluation Metrics）用于衡量模型在测试集上的性能，与损失函数不同，评估函数通常更关注实际应用中的性能表现。

2.7.1 分类任务的评估指标

1. 准确率（Accuracy）

数学公式：

$$\text{Accuracy} = \frac{\text{正确预测数}}{\text{总样本数}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{32}$$

其中，TP（True Positive）是真阳性，TN（True Negative）是真阴性，FP（False Positive）是假阳性，FN（False Negative）是假阴性。

用处：

- 类别平衡的分类任务
- 需要整体性能评估的场景
- 快速了解模型整体表现

为什么用准确率：

- **直观易懂**：最直观的性能指标，容易理解
- **计算简单**：计算成本低，适合大规模评估
- **通用性强**：适用于所有分类任务

缺点：

- **类别不平衡时失效**：如果 99% 的样本是正类，预测全为正类也能达到 99% 准确率
- **忽略错误类型**：不区分假阳性 and 假阴性

2. 精确率（Precision）

数学公式：

$$\text{Precision} = \frac{TP}{TP + FP} \tag{33}$$

用处：

- 需要减少假阳性的场景（如垃圾邮件检测）
- 关注” 预测为正类的样本中，有多少是真的正类”

为什么用精确率：

- **减少误报**：当假阳性代价高时，精确率更重要
- **资源有限**：当处理正类样本需要消耗资源时，精确率很重要

3. 召回率（Recall）

数学公式：

$$\text{Recall} = \frac{TP}{TP + FN} = \text{Sensitivity} \tag{34}$$

用处：

- 需要减少假阴性的场景（如疾病诊断）

- 关注” 所有正类样本中，有多少被正确识别”

为什么用召回率：

- **减少漏报**：当假阴性代价高时，召回率更重要
- **全面覆盖**：需要尽可能找到所有正类样本

4. F1 分数（F1-Score）

数学公式：

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN} \tag{35}$$

用处：

- 需要平衡精确率和召回率的场景
- 类别不平衡的分类任务
- 单一指标评估模型性能

为什么用 F1 分数：

- **平衡指标**：综合考虑精确率和召回率
- **调和平均数**：使用调和平均数，对低值更敏感
- **单一指标**：用一个数字概括模型性能，便于比较

5. ROC 曲线和 AUC

ROC 曲线：以假阳性率（FPR）为横轴，真阳性率（TPR，即召回率）为纵轴的曲线。

AUC（Area Under Curve）：ROC 曲线下的面积。

数学公式：

$$\text{FPR} = \frac{FP}{FP + TN} \tag{36}$$

$$\text{TPR} = \frac{TP}{TP + FN} = \text{Recall} \tag{37}$$

用处：

- 二分类任务的性能评估
- 需要比较不同模型的整体性能

- 类别不平衡的任务

为什么用 AUC：

- **阈值无关**：不依赖于分类阈值，评估模型整体性能
- **类别不平衡鲁棒**：对类别不平衡相对鲁棒
- **概率解释**：AUC 可以解释为” 随机选择一个正样本和一个负样本，模型对正样本的预测概率大于负样本的概率”

2.7.2 回归任务的评估指标

1. 均方误差（MSE）

数学公式：

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{38}$$

用处：

- 回归任务的标准评估指标
- 需要惩罚大误差的场景

为什么用 MSE：

- **标准指标**：最常用的回归评估指标
- **可导性**：可导，便于优化
- **大误差敏感**：对大误差敏感，符合实际需求

2. 均方根误差（RMSE）

数学公式：

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{39}$$

用处：

- 需要与原数据相同量纲的评估指标
- 更直观的误差表示

为什么用 RMSE:

- **量纲一致**：与原数据量纲相同，更直观
- **可解释性**：可以直接理解为" 平均误差"

3. 平均绝对误差（MAE）

数学公式:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{40}$$

用处:

- 需要鲁棒性评估的场景
- 异常值较多的数据

为什么用 MAE:

- **鲁棒性**：对异常值不敏感
- **直观性**：直接表示平均误差，易于理解

4. 决定系数（R^2 Score）

数学公式:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{\text{SS}_{res}}{\text{SS}_{tot}} \tag{41}$$

其中 $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ 是真实值的均值。

输出范围：$(-\infty, 1]$，越接近 1 越好。

用处:

- 需要相对性能评估的场景
- 比较不同模型的解释能力

为什么用 R^2:

- **相对性能**：表示模型相对于简单均值预测的改进程度
- **标准化**：值域固定，便于比较不同数据集上的模型
- **可解释性**：可以解释为" 模型解释了多少方差"

2.7.3 评估指标选择指南

评估指标选择指南：

- **准确率（Accuracy）**：
  - 适用任务：类别平衡的分类任务
  - 选择原因：直观易懂，计算简单
- **精确率（Precision）**：
  - 适用任务：需要减少假阳性
  - 选择原因：关注预测正类的准确性
- **召回率（Recall）**：
  - 适用任务：需要减少假阴性
  - 选择原因：关注找到所有正类
- **F1 分数**：
  - 适用任务：需要平衡精确率和召回率
  - 选择原因：单一指标，平衡两者
- **AUC（ROC 曲线下面积）**：
  - 适用任务：二分类任务，类别不平衡
  - 选择原因：阈值无关，整体性能
- **MSE/RMSE（均方误差/均方根误差）**：
  - 适用任务：回归任务
  - 选择原因：标准指标，大误差敏感
- **MAE（平均绝对误差）**：
  - 适用任务：回归任务（异常值多）
  - 选择原因：鲁棒性强
- **R^2（决定系数）**：
  - 适用任务：回归任务（相对性能）
  - 选择原因：标准化，可解释性强

3 简易神经网络完整示例

本节通过一个完整的简易神经网络示例，展示深度学习的核心概念，包括编码（Encode）、解码（Decode）、训练和推理的全过程。

3.1 网络架构设计

任务：使用一个简单的三层神经网络进行二分类任务（如判断邮件是否为垃圾邮件）。

网络结构：

- **输入层**：$d_{in} = 4$ 个特征（如邮件的关键词频率）
- **隐藏层 1**：$h_1 = 8$ 个神经元，使用 ReLU 激活函数
- **隐藏层 2**：$h_2 = 4$ 个神经元，使用 ReLU 激活函数
- **输出层**：$d_{out} = 1$ 个神经元，使用 Sigmoid 激活函数（输出概率）

数学表示：

对于输入 $\mathbf{x} \in \mathbb{R}^4$，网络的前向传播为：

$$\mathbf{h}^{(1)} = \text{ReLU}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \quad \mathbf{W}^{(1)} \in \mathbb{R}^{8\times4}, \mathbf{b}^{(1)} \in \mathbb{R}^8 \tag{42}$$

$$\mathbf{h}^{(2)} = \text{ReLU}(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \quad \mathbf{W}^{(2)} \in \mathbb{R}^{4\times8}, \mathbf{b}^{(2)} \in \mathbb{R}^4 \tag{43}$$

$$\hat{y} = \sigma(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}) \quad \mathbf{W}^{(3)} \in \mathbb{R}^{1\times4}, \mathbf{b}^{(3)} \in \mathbb{R}^1 \tag{44}$$

其中 σ 是 Sigmoid 函数：$\sigma(x) = \frac{1}{1+e^{-x}}$。

3.2 从零实现神经网络

完整代码实现：

```
import numpy as np
import pandas as pd

class SimpleNeuralNetwork:
    """简易三层神经网络"""

    def __init__(self, input_size=4, hidden1_size=8, hidden2_size=4,
   output_size=1):
```

```
        """
        初始化神经网络

        参数:
            input_size: 输入特征维度
            hidden1_size: 第一个隐藏层神经元数量
            hidden2_size: 第二个隐藏层神经元数量
            output_size: 输出维度
        """
        # 初始化权重矩阵（使用 Xavier 初始化）
        self.W1 = np.random.randn(hidden1_size, input_size) * np.sqrt(2.0
 / input_size)
        self.b1 = np.zeros((hidden1_size, 1))

        self.W2 = np.random.randn(hidden2_size, hidden1_size) * np.sqrt
(2.0 / hidden1_size)
        self.b2 = np.zeros((hidden2_size, 1))

        self.W3 = np.random.randn(output_size, hidden2_size) * np.sqrt
(2.0 / hidden2_size)
        self.b3 = np.zeros((output_size, 1))

    def relu(self, x):
        """ReLU 激活函数"""
        return np.maximum(0, x)

    def relu_derivative(self, x):
        """ReLU 的导数"""
        return (x > 0).astype(float)

    def sigmoid(self, x):
        """Sigmoid 激活函数"""
        # 防止溢出
        x = np.clip(x, -500, 500)
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        """Sigmoid 的导数"""
        s = self.sigmoid(x)
        return s * (1 - s)
```

```
    def forward(self, X):
        """
        前向传播（Encode）

        参数:
            X: 输入数据 (batch_size, input_size)

        返回:
            y_pred: 预测输出 (batch_size, output_size)
            cache: 中间结果（用于反向传播）
        """
        # 输入层 → 隐藏层1
        self.z1 = X.dot(self.W1.T) + self.b1.T  # (batch_size,
hidden1_size)
        self.a1 = self.relu(self.z1)  # 激活

        # 隐藏层1 → 隐藏层2
        self.z2 = self.a1.dot(self.W2.T) + self.b2.T  # (batch_size,
hidden2_size)
        self.a2 = self.relu(self.z2)  # 激活

        # 隐藏层2 → 输出层
        self.z3 = self.a2.dot(self.W3.T) + self.b3.T  # (batch_size,
output_size)
        self.a3 = self.sigmoid(self.z3)  # 激活（输出概率）

        return self.a3

    def backward(self, X, y, y_pred):
        """
        反向传播（计算梯度）

        参数:
            X: 输入数据 (batch_size, input_size)
            y: 真实标签 (batch_size, 1)
            y_pred: 预测输出 (batch_size, output_size)
        """
        m = X.shape[0]  # 批次大小
```

```
        # 输出层误差
        dz3 = y_pred - y  # (batch_size, output_size)
        dW3 = (1/m) * dz3.T.dot(self.a2)  # (output_size, hidden2_size)
        db3 = (1/m) * np.sum(dz3, axis=0, keepdims=True).T  # (
    output_size, 1)

        # 隐藏层2误差
        da2 = dz3.dot(self.W3)  # (batch_size, hidden2_size)
        dz2 = da2 * self.relu_derivative(self.z2)  # 链式法则
        dW2 = (1/m) * dz2.T.dot(self.a1)  # (hidden2_size, hidden1_size)
        db2 = (1/m) * np.sum(dz2, axis=0, keepdims=True).T  # (
    hidden2_size, 1)

        # 隐藏层1误差
        da1 = dz2.dot(self.W2)  # (batch_size, hidden1_size)
        dz1 = da1 * self.relu_derivative(self.z1)  # 链式法则
        dW1 = (1/m) * dz1.T.dot(X)  # (hidden1_size, input_size)
        db1 = (1/m) * np.sum(dz1, axis=0, keepdims=True).T  # (
    hidden1_size, 1)

        return dW1, db1, dW2, db2, dW3, db3

    def update_parameters(self, dW1, db1, dW2, db2, dW3, db3,
    learning_rate):
        """更新参数（梯度下降）"""
        self.W1 -= learning_rate * dW1
        self.b1 -= learning_rate * db1
        self.W2 -= learning_rate * dW2
        self.b2 -= learning_rate * db2
        self.W3 -= learning_rate * dW3
        self.b3 -= learning_rate * db3

    def compute_loss(self, y_pred, y):
        """
        计算损失函数（二元交叉熵）

        数学公式:
        L = -\frac{1}{m}\sum_{i=1}^m [y_i \log(\hat{y}_i) + (1-y_i)\log
    (1-\hat{y}_i)]
        """
```

```
        m = y.shape[0]
        # 防止 log(0)
        epsilon = 1e-15
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
        loss = -(1/m) * np.sum(y * np.log(y_pred) + (1 - y) * np.log(1 -
y_pred))
        return loss

    def train(self, X_train, y_train, epochs=1000, learning_rate=0.01,
verbose=True):
        """
        训练神经网络

        参数:
            X_train: 训练数据 (n_samples, input_size)
            y_train: 训练标签 (n_samples, 1)
            epochs: 训练轮数
            learning_rate: 学习率
            verbose: 是否打印训练过程
        """
        losses = []

        for epoch in range(epochs):
            # 前向传播
            y_pred = self.forward(X_train)

            # 计算损失
            loss = self.compute_loss(y_pred, y_train)
            losses.append(loss)

            # 反向传播
            dW1, db1, dW2, db2, dW3, db3 = self.backward(X_train, y_train
, y_pred)

            # 更新参数
            self.update_parameters(dW1, db1, dW2, db2, dW3, db3,
learning_rate)

            # 打印训练进度
            if verbose and (epoch + 1) % 100 == 0:
```

```
                print(f"Epoch {epoch+1}/{epochs}, Loss: {loss:.4f}")

        return losses

    def predict(self, X):
        """
        预测（Decode/推理）

        参数:
            X: 输入数据 (n_samples, input_size)

        返回:
            predictions: 预测结果 (n_samples, 1)，值在 [0, 1] 之间
        """
        y_pred = self.forward(X)
        return y_pred

    def predict_class(self, X, threshold=0.5):
        """
        预测类别（二分类）

        参数:
            X: 输入数据
            threshold: 分类阈值

        返回:
            classes: 预测类别 (0 或 1)
        """
        y_pred = self.predict(X)
        return (y_pred > threshold).astype(int)

# 使用示例
# 1. 准备数据（编码：将原始数据转换为数值特征）
np.random.seed(42)
n_samples = 1000
X_train = np.random.randn(n_samples, 4)  # 4个特征
y_train = (X_train.sum(axis=1) > 0).astype(float).reshape(-1, 1)  # 简单
    二分类任务

# 2. 创建和训练模型
```

```
model = SimpleNeuralNetwork(input_size=4, hidden1_size=8, hidden2_size=4,
    output_size=1)
losses = model.train(X_train, y_train, epochs=1000, learning_rate=0.01)

# 3. 预测（解码：将网络输出转换为类别）
X_test = np.random.randn(10, 4)
predictions = model.predict(X_test)
classes = model.predict_class(X_test)

print("\n预测结果:")
print(f"预测概率: {predictions.flatten()}")
print(f"预测类别: {classes.flatten()}")
```

Listing 5: 简易神经网络完整实现（使用 NumPy）

3.3 概念对应关系

通过这个简易神经网络示例，我们可以清楚地看到深度学习各个概念的对应关系：

深度学习概念在代码中的对应：

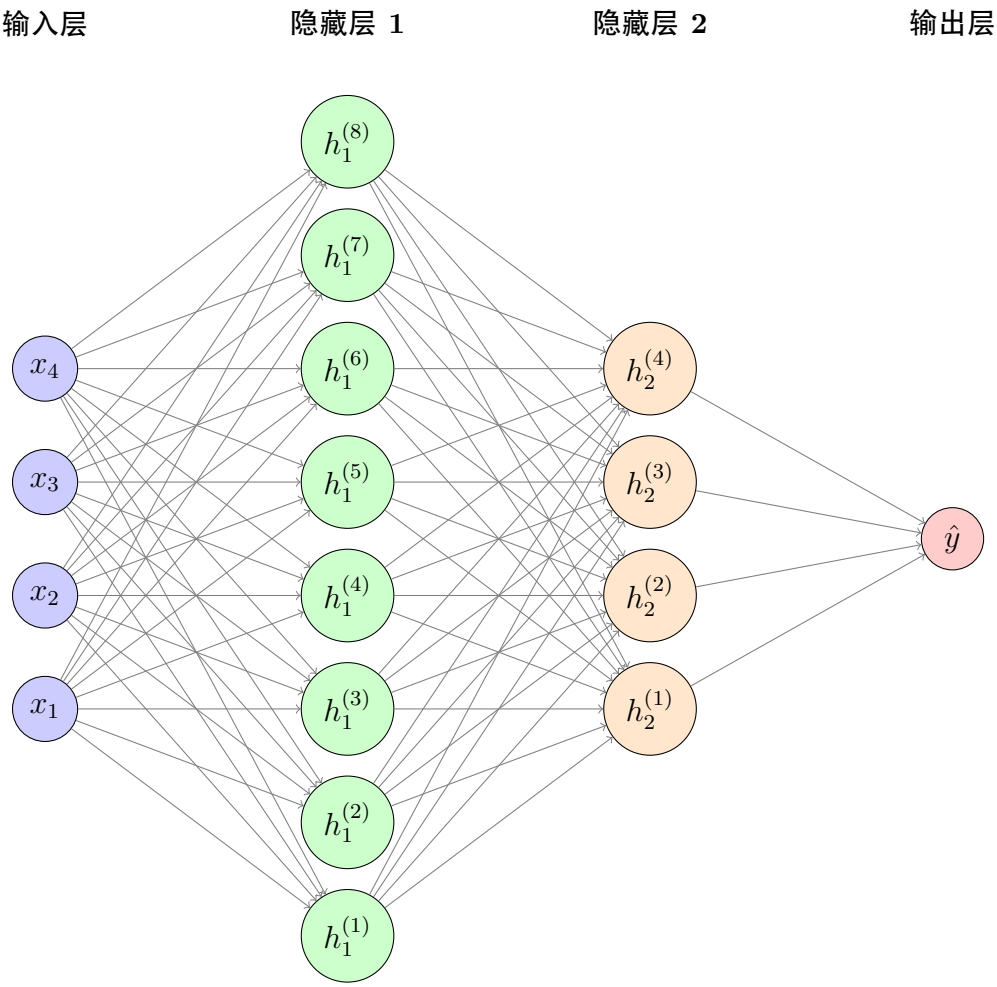
- **输入层**：`X`（输入数据矩阵）
- **隐藏层 1**：`self.a1`（8 个神经元）
- **隐藏层 2**：`self.a2`（4 个神经元）
- **输出层**：`self.a3`（1 个神经元，输出概率）
- **权重矩阵**：`self.W1`, `self.W2`, `self.W3`
- **偏置向量**：`self.b1`, `self.b2`, `self.b3`
- **激活函数**：`relu()`, `sigmoid()`
- **前向传播（Encode）**：`forward()` 方法
- **反向传播**：`backward()` 方法
- **损失函数**：`compute_loss()`（交叉熵）
- **梯度下降**：`update_parameters()`
- **训练**：`train()` 方法
- **推理（Decode）**：`predict()` 和 `predict_class()`

3.4 网络结构图示

网络架构图：

输入层 (4 个神经元) → 隐藏层 1 (8 个神经元, ReLU) → 隐藏层 2 (4 个神经元, ReLU) → 输出层 (1 个神经元, Sigmoid)

详细结构图示：



连接说明：

- 输入层的每个神经元（x_1, x_2, x_3, x_4）与隐藏层 1 的所有 8 个神经元全连接
- 隐藏层 1 的每个神经元与隐藏层 2 的所有 4 个神经元全连接
- 隐藏层 2 的每个神经元与输出层的 1 个神经元全连接

- 每个连接都有一个权重参数，每个神经元都有一个偏置参数

图示说明：

- 每个圆圈代表一个**神经元**（Neuron）
- 每条箭头线代表一个**权重**（Weight）连接
- **输入层**（蓝色）：4 个神经元，接收 4 个特征 $\mathbf{x} = [x_1, x_2, x_3, x_4]^T$
- **隐藏层 1**（绿色）：8 个神经元，每个神经元接收 4 个输入，使用 ReLU 激活函数
- **隐藏层 2**（橙色）：4 个神经元，每个神经元接收 8 个输入，使用 ReLU 激活函数
- **输出层**（红色）：1 个神经元，接收 4 个输入，使用 Sigmoid 激活函数，输出概率 $\hat{y} \in [0, 1]$

连接关系：

- 输入层 → 隐藏层 1：$4 \times 8 = 32$ 个权重连接
- 隐藏层 1 → 隐藏层 2：$8 \times 4 = 32$ 个权重连接
- 隐藏层 2 → 输出层：$4 \times 1 = 4$ 个权重连接
- 总参数量：$(4 \times 8 + 8) + (8 \times 4 + 4) + (4 \times 1 + 1) = 40 + 36 + 5 = 81$ 个参数

3.5 训练过程详解

训练步骤：

1. **前向传播（Encode）**：
   - 输入数据 $\mathbf{x}$ 经过网络，逐层计算
   - 每层计算：$\mathbf{z} = \mathbf{W}\mathbf{h} + \mathbf{b}$，然后应用激活函数
   - 最终得到预测输出 $\hat{y}$
2. **计算损失**：
   - 比较预测值 $\hat{y}$ 和真实值 y
   - 使用交叉熵损失函数：$L = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$
3. **反向传播**：
   - 从输出层开始，逐层向前计算梯度

- 使用链式法则：$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \mathbf{h}^{(l)}} \frac{\partial \mathbf{h}^{(l)}}{\partial W^{(l)}}$
- 计算所有权重和偏置的梯度

4. **参数更新**：
   - 使用梯度下降更新参数：$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}}$
   - 其中 η 是学习率
5. **重复迭代**：重复步骤 1-4，直到损失收敛

3.6 推理过程（Decode）

推理步骤：

1. **输入编码**：将原始输入（如文本、图像特征）编码为数值向量 $\mathbf{x}$
2. **前向传播**：数据通过网络，得到输出 $\hat{y} \in [0, 1]$
3. **输出解码**：
   - 对于二分类：如果 $\hat{y} > 0.5$，预测为正类（1），否则为负类（0）
   - 对于多分类：使用 softmax 将输出转换为概率分布，选择概率最大的类别

数学表示：

编码过程：$\mathbf{x} = \text{encode}(\text{原始输入})$

前向传播：$\hat{y} = f(\mathbf{x}; \theta)$，其中 f 是神经网络，θ 是参数

解码过程：预测类别 $= \text{decode}(\hat{y}) = \begin{cases} 1 & \text{if } \hat{y} > 0.5 \\ 0 & \text{otherwise} \end{cases}$

Part II

第二部分：深度网络架构

4 深度网络架构

4.1 全连接层（Fully Connected Layer）

概念解释：全连接层（Fully Connected Layer），也称为密集层（Dense Layer）或线性层（Linear Layer），是神经网络中最基础的层类型。在全连接层中，每个神经元都与前一层的所有神经元相连。

数学表示：

对于输入 $\mathbf{x} \in \mathbb{R}^{d_{in}}$，全连接层的输出为：

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{45}$$

其中：

- $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$：权重矩阵
- $\mathbf{b} \in \mathbb{R}^{d_{out}}$：偏置向量
- d_{in}：输入维度
- d_{out}：输出维度（神经元数量）

参数量：对于输入维度 d_{in}、输出维度 d_{out} 的全连接层，参数量为：

$$\text{参数量} = d_{in} \times d_{out} + d_{out} = d_{out}(d_{in} + 1) \tag{46}$$

通俗解释：全连接层就像一个” 完全连接的转换器”。想象一个房间里有 d_{in} 个人（输入神经元），另一个房间里有 d_{out} 个人（输出神经元）。每个人都要和另一个房间的每个人握手（连接），每次握手都有一个” 权重”（连接强度）。最后，输出房间的每个人计算自己收到的所有” 握手强度” 的总和，再加上自己的” 基础值”（偏置），就得到了输出。

应用场景：

- MLP 中的主要层类型
- CNN 中的分类层（在卷积和池化之后）

- Transformer 中的前馈网络（Feed-Forward Network）

优势与局限：

优势：

- 表达能力强大，可以学习任意复杂的映射关系
- 实现简单，计算高效（矩阵乘法）

局限：

- 参数量大，容易过拟合
- 忽略了输入的空间结构（如图像的局部相关性）
- 对输入尺寸敏感，需要固定输入维度

4.1.1 输入层和输出层与全连接层的关系

核心问题：输入层和输出层是”全连接层”吗?

这个问题需要分两部分回答：

（1）输入层本身通常不被视为”全连接层”：

输入层只是一个数据接口，它没有可学习的权重或偏置。它只是把原始数据 $\mathbf{x} \in \mathbb{R}^d$ 传递给下一层。

全连接层必须包含可学习的参数（$\mathbf{W}, \mathbf{b}$）并执行 $\mathbf{Wx} + \mathbf{b}$ 运算，而输入层不做任何计算。

结论：输入层不等于全连接层（它甚至不算一个”层”在参数意义上）。

（2）输出层通常是全连接层：

在绝大多数神经网络中（包括分类任务、回归任务），输出层是一个全连接层。

例如：

```
nn.Linear(128, 1)  # 从128维隐藏表示映射到1个输出（如回归值）
nn.Linear(128, 10)  # 从128维隐藏表示映射到10个输出（如10分类）
```

Listing 6: 输出层作为全连接层

这就是标准的全连接层（带权重和偏置）。

结论：输出层通常是全连接层（但它的角色是”输出”，不是”隐藏”）。

术语使用规范：

层类型与全连接层的关系：

- **输入层：**
  - 是隐藏层？否
  - 是全连接层？通常不算（无参数，仅传递数据）
- **中间层：**
  - 是隐藏层？是
  - 是全连接层？常是（如 MLP 中的 `Linear(256,128) + ReLU`）
- **输出层：**
  - 是隐藏层？否
  - 是全连接层？通常是（如 `Linear(128, 1)`，用于最终映射）

重要说明：

虽然输出层是全连接层，但我们不会称它为”隐藏层”，因为它的功能是输出，不是隐藏表示。当配置 `nn.Linear(in, out)` 时，无论它在中间还是最后，都是全连接层，但只有中间的才是隐藏层。

总结：

- **第一层（输入层）和最后一层（输出层）是不是隐藏层？**
  都不是。隐藏层仅指中间层，即夹在输入层和输出层之间的所有层。
- **它们是不是全连接层？**
  - **输入层**：不是。输入层无参数，不算计算层，只是数据的载体。
  - **输出层**：通常是。输出层用 `Linear` 实现，有 $\mathbf{W}$ 和 $\mathbf{b}$ 参数，执行线性变换。

代码示例：

```
import torch.nn as nn

model = nn.Sequential(
    # 输入层：只是数据入口，不是全连接层
    # x 直接传入下一层
```

```
    # 隐藏层1: 全连接层 + 激活函数
    nn.Linear(100, 256),  # ← 这是全连接层，也是隐藏层
    nn.ReLU(),

    # 隐藏层2: 全连接层 + 激活函数
    nn.Linear(256, 128),  # ← 这是全连接层，也是隐藏层
    nn.ReLU(),

    # 输出层: 全连接层（但不是隐藏层）
    nn.Linear(128, 1)  # ← 这是全连接层，但不是隐藏层
)
```

Listing 7: 层类型的代码示例

小贴士:

- 当说”模型有 2 个隐藏层”时，指的是中间有 2 个可学习的层，不包括输入和输出
- 当配置 `nn.Linear(in, out)` 时，无论它在中间还是最后，都是全连接层，但只有中间的才是隐藏层
- 输入层没有参数，输出层有参数（$\mathbf{W}$ 和 $\mathbf{b}$），但输出层不是隐藏层

4.2 卷积层（Convolutional Layer）

概念解释：卷积层是卷积神经网络的核心组件，通过卷积操作提取局部特征。卷积操作利用局部连接和权重共享，大幅减少参数量。

数学表示:

对于二维卷积，给定输入特征图 $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ 和卷积核 $\mathbf{K} \in \mathbb{R}^{k_h \times k_w \times C \times C'}$，卷积操作定义为:

$$(\mathbf{X} * \mathbf{K})_{i,j,c'} = \sum_{c=1}^{C} \sum_{u=0}^{k_h-1} \sum_{v=0}^{k_w-1} \mathbf{X}_{i+u,j+v,c} \cdot \mathbf{K}_{u,v,c,c'} \tag{47}$$

其中，$*$ 表示卷积操作。

参数量：对于卷积核大小为 $k_h \times k_w$，输入通道数 C，输出通道数 C' 的卷积层:

$$\text{参数量} = k_h \times k_w \times C \times C' + C' \tag{48}$$

通俗解释：卷积层就像一个”滑动窗口特征提取器”。想象你拿着一块小模板（卷积核）在图像上滑动，每次滑动时，模板会”扫描”一小块区域，提取这块区域的特征。不同的模板提

取不同的特征（如边缘、纹理、形状等）。通过多个模板（多个卷积核），可以同时提取多种特征。

核心特性：

- **局部连接**：每个神经元只连接输入的一个局部区域
- **权重共享**：同一卷积核在整个输入上滑动，共享参数
- **平移不变性**：对输入的位置变化具有鲁棒性

应用场景：

- 图像分类：提取图像的层次化特征
- 目标检测：定位和识别图像中的物体
- 语义分割：对图像的每个像素进行分类

4.3 池化层（Pooling Layer）

概念解释：池化层用于降低特征图的空间维度，减少参数量和计算量，同时提供一定的平移不变性。

数学表示：

最大池化（Max Pooling）：

$$\text{MaxPool}(\mathbf{X})_{i,j} = \max_{u,v \in \text{window}} \mathbf{X}_{i+u,j+v} \tag{49}$$

平均池化（Average Pooling）：

$$\text{AvgPool}(\mathbf{X})_{i,j} = \frac{1}{|\text{window}|} \sum_{u,v \in \text{window}} \mathbf{X}_{i+u,j+v} \tag{50}$$

其中，window 是池化窗口（如 2×2）。

参数量：池化层没有可学习参数，只有超参数（窗口大小、步长）。

通俗解释：池化层就像一个” 信息压缩器”。想象你把一张大照片分成很多 2×2 的小块，然后对每块做以下操作：

- **最大池化**：保留每块中最亮的像素（最显著的特征）
- **平均池化**：计算每块的平均亮度（平滑的特征）

这样，照片的尺寸就缩小了一半，但保留了最重要的信息。

作用：

- **降维**：减少特征图的空间尺寸，降低计算量
- **特征不变性**：对小的平移和变形具有鲁棒性
- **防止过拟合**：减少参数量，降低模型复杂度

比较：

最大池化 vs 平均池化：

- **保留信息**：
  - 最大池化：保留最显著特征
  - 平均池化：保留整体特征
- **对噪声**：
  - 最大池化：更鲁棒
  - 平均池化：更敏感
- **适用场景**：
  - 最大池化：特征检测
  - 平均池化：平滑特征

4.4 全连接网络

全连接网络（Fully Connected Network, FCN）是由多个全连接层组成的深度网络架构。

架构特点：

- 参数量大：对于 L 层网络，参数量为 $\sum_{l=1}^{L}(n_{l-1}+1)n_l$
- 计算密集：需要大量的矩阵乘法运算
- 适合处理向量化输入：如图像展平后的向量、特征向量等

应用场景：

例 4.1 (图像分类). *在 CIFAR-10 数据集中，可以将 $32 \times 32 \times 3$ 的图像展平为 3072 维向量，然后通过多层全连接网络进行分类。*

4.5 卷积神经网络

卷积神经网络（Convolutional Neural Network, CNN）是专门设计用于处理具有网格结构数据（如图像）的深度学习架构。

定义 4.1 (卷积操作). 对于二维卷积，给定输入特征图 $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ 和卷积核 $\mathbf{K} \in \mathbb{R}^{k_h \times k_w \times C \times C'}$，卷积操作定义为：

$$(\mathbf{X} * \mathbf{K})_{i,j,c'} = \sum_{c=1}^{C} \sum_{u=0}^{k_h-1} \sum_{v=0}^{k_w-1} \mathbf{X}_{i+u,j+v,c} \cdot \mathbf{K}_{u,v,c,c'} \tag{51}$$

其中，$*$ 表示卷积操作。

CNN 的核心组件：

1. **卷积层（Convolutional Layer）**：通过卷积操作提取局部特征
   - 局部连接：每个神经元只连接输入的一个局部区域
   - 权重共享：同一卷积核在整个输入上滑动，共享参数
   - 平移不变性：对输入的位置变化具有鲁棒性
2. **池化层（Pooling Layer）**：降低特征图的空间维度，减少参数量和计算量
   - 最大池化：$\text{MaxPool}(\mathbf{X})_{i,j} = \max_{u,v \in \text{window}} \mathbf{X}_{i+u,j+v}$
   - 平均池化：$\text{AvgPool}(\mathbf{X})_{i,j} = \frac{1}{|\text{window}|} \sum_{u,v \in \text{window}} \mathbf{X}_{i+u,j+v}$
3. **全连接层**：在卷积和池化之后，用于最终的分类或回归

经典 CNN 架构：

例 4.2 (LeNet-5). *LeNet-5* 是 *Yann LeCun* 在 *1998* 年提出的用于手写数字识别的 *CNN* 架构，包含两个卷积层、两个池化层和三个全连接层。

例 4.3 (AlexNet). *AlexNet* 在 *2012* 年 *ImageNet* 竞赛中取得突破性成果，包含 *5* 个卷积层和 *3* 个全连接层，首次使用 *ReLU* 激活函数和 *Dropout* 技术。

例 4.4 (VGGNet). *VGGNet* 使用更深的网络（*16-19* 层）和更小的卷积核（3×3），证明了网络深度的重要性。

例 4.5 (ResNet). *ResNet* 引入残差连接，解决了深层网络的退化问题，可以训练超过 *100* 层的网络。

CNN 的优势：

- **参数效率**：通过局部连接和权重共享，大幅减少参数量
- **平移不变性**：对输入的位置变化具有鲁棒性
- **层次化特征学习**：底层学习边缘、纹理等低级特征，高层学习语义、对象等高级特征
- **广泛应用**：在图像分类、目标检测、语义分割等任务中表现优异

应用场景：

- **图像分类**：ImageNet 图像分类挑战
- **目标检测**：YOLO、R-CNN 系列
- **语义分割**：FCN、U-Net
- **人脸识别**：FaceNet、DeepFace
- **医学影像**：CT、MRI 图像分析

4.6 循环神经网络

循环神经网络（Recurrent Neural Network, RNN）是专门设计用于处理序列数据的神经网络架构。

定义 4.2 (RNN). *RNN 通过维护隐藏状态来记忆历史信息。对于输入序列 $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$，RNN 的计算过程为：*

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}_h) \tag{52}$$

$$\mathbf{y}_t = \mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y \tag{53}$$

其中，$\mathbf{h}_t$ 是时刻 t 的隐藏状态，$\mathbf{W}_h$、$\mathbf{W}_x$、$\mathbf{W}_y$ 是权重矩阵。

RNN 的特点：

- **参数共享**：所有时间步共享相同的参数
- **记忆能力**：通过隐藏状态传递历史信息
- **变长输入**：可以处理不同长度的序列

梯度消失和梯度爆炸：在长序列中，RNN 容易出现梯度消失或梯度爆炸问题，导致难以学习长期依赖关系。

LSTM（Long Short-Term Memory）：LSTM 通过引入门控机制解决长期依赖问题。

定义 4.3 (LSTM). *LSTM 单元包含三个门：遗忘门、输入门和输出门。*

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \quad \text{（遗忘门）} \tag{54}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \quad \text{（输入门）} \tag{55}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \tag{56}$$

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t \quad \text{（细胞状态）} \tag{57}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \quad \text{（输出门）} \tag{58}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t) \tag{59}$$

其中，$\mathbf{C}_t$ 是细胞状态，$\odot$ 表示逐元素相乘。

GRU（Gated Recurrent Unit）：GRU 是 LSTM 的简化版本，只有两个门（更新门和重置门），计算效率更高。

应用场景：

- **自然语言处理**：机器翻译、文本生成、情感分析
- **语音识别**：语音转文字
- **时间序列预测**：股票价格预测、天气预测
- **序列标注**：命名实体识别、词性标注

例 4.6 (机器翻译). *Google 的神经机器翻译系统使用编码器-解码器架构，编码器将源语言序列编码为固定维度的向量，解码器生成目标语言序列。*

Part III

第三部分：优化技术与高级主题

5 深度学习优化技术

5.1 梯度下降变体

梯度下降及其变体是训练深度神经网络的核心优化算法。

批量梯度下降（Batch Gradient Descent）：

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{60}$$

其中，$\nabla_\theta \mathcal{L}(\theta_t) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \ell(f(\mathbf{x}_i;\theta_t), y_i)$ 是损失函数在整个训练集上的梯度。

随机梯度下降（Stochastic Gradient Descent, SGD）：

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \ell(f(\mathbf{x}_i;\theta_t), y_i) \tag{61}$$

每次只使用一个样本更新参数，计算速度快但梯度估计方差大。

小批量梯度下降（Mini-batch Gradient Descent）：

$$\theta_{t+1} = \theta_t - \eta \frac{1}{B}\sum_{i \in \mathcal{B}_t} \nabla_\theta \ell(f(\mathbf{x}_i;\theta_t), y_i) \tag{62}$$

使用小批量样本（通常 $B = 32, 64, 128$），在计算效率和梯度稳定性之间取得平衡。

动量法（Momentum）：

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)\nabla_\theta \mathcal{L}(\theta_t) \tag{63}$$

$$\theta_{t+1} = \theta_t - \eta \mathbf{v}_t \tag{64}$$

其中，$\beta \in [0, 1)$ 是动量系数（通常取 0.9）。动量法可以加速收敛并减少震荡。

Adam（Adaptive Moment Estimation）：Adam 结合了动量和自适应学习率的思想：

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_t) \quad \text{（一阶矩估计）} \tag{65}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_t))^2 \quad \text{（二阶矩估计）} \tag{66}$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t} \quad \text{（偏差修正）} \tag{67}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \tag{68}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}\hat{\mathbf{m}}_t \tag{69}$$

其中，$\beta_1 = 0.9$，$\beta_2 = 0.999$，$\epsilon = 10^{-8}$。Adam 是目前最常用的优化算法之一。

5.2 批量归一化

批量归一化（Batch Normalization, BN）通过归一化每层的输入分布来加速训练并提高模型稳定性。

定义 5.1 (批量归一化). 对于小批量 $\mathcal{B} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_B\}$，批量归一化计算：

$$\mu_{\mathcal{B}} = \frac{1}{B}\sum_{i=1}^{B}\mathbf{x}_i \quad \text{（批量均值）} \tag{70}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{B}\sum_{i=1}^{B}(\mathbf{x}_i - \mu_{\mathcal{B}})^2 \quad \text{（批量方差）} \tag{71}$$

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad \text{（归一化）} \tag{72}$$

$$\mathbf{y}_i = \gamma\hat{\mathbf{x}}_i + \beta \quad \text{（缩放和平移）} \tag{73}$$

其中，γ 和 β 是可学习的参数，ϵ 是小的常数（如 10^{-5}）防止除零。

批量归一化的优势：

- **加速训练**：允许使用更大的学习率
- **减少内部协变量偏移**：稳定每层的输入分布
- **正则化效果**：减少对 Dropout 的依赖
- **缓解梯度消失**：使激活值分布在合适的范围内

应用位置：通常放在卷积层或全连接层之后、激活函数之前（或之后，取决于具体实现）。

5.3 Dropout

Dropout 是一种正则化技术，通过在训练过程中随机丢弃部分神经元来防止过拟合。

定义 5.2 (Dropout). *在训练阶段，对于每个神经元，以概率 p（通常 $p = 0.5$）将其输出置为零*:

$$\mathbf{h}_i^{(l)} = \begin{cases} 0 & \text{以概率} p \\ \frac{\mathbf{h}_i^{(l)}}{1-p} & \text{以概率} 1-p \end{cases} \tag{74}$$

在测试阶段，所有神经元都参与计算，但输出需要乘以 $(1-p)$ 以保持期望值不变。

Dropout 的机制:

- 防止神经元之间的共适应，迫使网络学习更鲁棒的特征
- 相当于训练多个不同的子网络，测试时进行集成
- 减少过拟合，提高泛化能力

应用场景：通常应用在全连接层，卷积层也可以使用（Spatial Dropout）。

5.4 残差连接

残差连接（Residual Connection）通过跳跃连接将输入直接传递到输出，解决了深层网络的退化问题。

定义 5.3 (残差块). *残差块（Residual Block）的计算为*:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x} \tag{75}$$

其中，$\mathcal{F}(\mathbf{x})$ 是残差函数（通常是几个卷积层），$\mathbf{x}$ 是输入（通过跳跃连接直接传递）。

残差连接的优势:

- **解决退化问题**：即使残差函数学习到零映射，网络也能保持恒等映射
- **缓解梯度消失**：梯度可以直接通过跳跃连接反向传播
- **允许训练更深的网络**：ResNet 可以训练超过 1000 层的网络

架构变体:

- **ResNet**：基本的残差网络
- **DenseNet**：密集连接，每层都连接到所有后续层
- **Highway Networks**：使用门控机制控制信息流

例 5.1 (ResNet-50). *ResNet-50 包含 50 个卷积层，在 ImageNet 上取得了优异的性能，成为计算机视觉领域的标准架构之一。*

6 表示学习与嵌入

表示学习（Representation Learning）旨在学习数据的有效表示，使得学习任务更容易完成。嵌入（Embedding）是将离散对象（如词、节点）映射到连续向量空间的技术。

6.1 词嵌入

词嵌入（Word Embedding）将词汇表中的词映射到低维连续向量空间，使得语义相似的词在向量空间中距离较近。

Word2Vec：Word2Vec 是 Google 提出的词嵌入方法，包含两种模型：

1. **Skip-gram**：给定中心词，预测上下文词
2. **CBOW（Continuous Bag of Words）**：给定上下文词，预测中心词

Skip-gram 模型：

$$P(w_{t+j}|w_t) = \frac{\exp(\mathbf{v}_{w_{t+j}}^T \mathbf{u}_{w_t})}{\sum_{w \in V} \exp(\mathbf{v}_w^T \mathbf{u}_{w_t})} \tag{76}$$

其中，$\mathbf{u}_w$ 是词 w 作为中心词的向量，$\mathbf{v}_w$ 是词 w 作为上下文词的向量，V 是词汇表。

负采样：为了加速训练，使用负采样近似 softmax：

$$\log \sigma(\mathbf{v}_{w_O}^T \mathbf{u}_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-\mathbf{v}_{w_i}^T \mathbf{u}_{w_I})] \tag{77}$$

其中，w_O 是正样本（真实上下文词），w_i 是负样本（从噪声分布 $P_n(w)$ 中采样）。

GloVe（Global Vectors）：GloVe 结合了全局统计信息和局部上下文窗口：

$$\mathcal{L} = \sum_{i,j=1}^{V} f(X_{ij})(\mathbf{w}_i^T \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{78}$$

其中，X_{ij} 是词 i 和词 j 的共现次数，$f(X_{ij})$ 是权重函数。

应用场景：

- **文本分类**：将词嵌入作为特征输入分类器
- **机器翻译**：作为编码器的输入
- **情感分析**：捕捉词的语义信息
- **推荐系统**：将物品描述转换为向量

6.2 图嵌入

图嵌入（Graph Embedding）将图中的节点映射到低维向量空间，保持图的结构和属性信息。

DeepWalk：DeepWalk 使用随机游走生成节点序列，然后使用 Word2Vec 学习节点嵌入：

Algorithm 4 DeepWalk 算法

Require: 图 $G(V, E)$，窗口大小 w，游走长度 t，嵌入维度 d

Ensure: 节点嵌入 $\Phi : V \to \mathbb{R}^d$

初始化随机游走序列集合 $\mathcal{S} = \emptyset$
for 每个节点 $v_i \in V$ **do**
  for $r = 1$ to γ **do**
    从 v_i 开始执行长度为 t 的随机游走，得到序列 s_i
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_i\}$
  end for
end for
使用 Skip-gram 在序列集合 $\mathcal{S}$ 上学习节点嵌入

Node2Vec：Node2Vec 使用有偏随机游走，平衡广度优先搜索（BFS）和深度优先搜索（DFS）：

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases} \tag{79}$$

其中，π_{vx} 是未归一化的转移概率，Z 是归一化常数。

应用场景：

- **节点分类**：预测节点的类别
- **链接预测**：预测节点之间是否存在边
- **社区检测**：发现图中的社区结构
- **推荐系统**：用户-物品二部图嵌入

7 注意力机制与 Transformer 架构

7.1 注意力机制

注意力机制（Attention Mechanism）允许模型在处理序列时动态地关注不同位置的信息。

定义 7.1 (注意力机制). 给定查询（*Query*）$\mathbf{Q}$、键（*Key*）$\mathbf{K}$ 和值（*Value*）$\mathbf{V}$，注意力机制计算：

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{80}$$

其中，d_k 是键的维度，$\sqrt{d_k}$ 是缩放因子，防止点积过大导致 *softmax* 梯度消失。

注意力机制的直观理解：

- **查询（Query）**：表示当前需要关注什么信息
- **键（Key）**：表示每个位置提供什么信息
- **值（Value）**：表示每个位置的实际内容
- 注意力权重：通过查询和键的相似度计算，决定关注哪些位置

自注意力（Self-Attention）：当查询、键和值都来自同一输入序列时，称为自注意力：

$$\mathbf{Z} = \text{Attention}(\mathbf{X}\mathbf{W}_Q, \mathbf{X}\mathbf{W}_K, \mathbf{X}\mathbf{W}_V) \tag{81}$$

其中，$\mathbf{X}$ 是输入序列，$\mathbf{W}_Q$、$\mathbf{W}_K$、$\mathbf{W}_V$ 是可学习的权重矩阵。

多头注意力（Multi-Head Attention）：使用多个注意力头并行计算，然后拼接：

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O \tag{82}$$

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{83}$$

其中，h 是注意力头的数量（通常 $h = 8$）。

7.2 Transformer 架构

Transformer 是 Vaswani 等人在 2017 年提出的完全基于注意力机制的架构，成为现代 NLP 的基础。

Transformer 的整体架构：

- **编码器（Encoder）**：N 个相同的层堆叠，每层包含：
  - 多头自注意力子层
  - 前馈神经网络子层
  - 残差连接和层归一化
- **解码器（Decoder）**：N 个相同的层堆叠，每层包含：
  - 掩码多头自注意力子层（防止看到未来信息）
  - 编码器-解码器注意力子层
  - 前馈神经网络子层
  - 残差连接和层归一化

位置编码（Positional Encoding）：由于 Transformer 没有循环结构，需要显式编码位置信息：

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{84}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{85}$$

其中，pos 是位置，i 是维度索引，d_{model} 是模型维度。

前馈神经网络：

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{86}$$

通常使用两层全连接网络，中间使用 ReLU 激活函数。

Transformer 的优势：

- **并行计算**：不像 RNN 需要顺序计算，可以并行处理所有位置
- **长距离依赖**：注意力机制可以直接建模任意距离的依赖关系
- **可解释性**：注意力权重可以可视化，了解模型关注的位置

应用场景：

- **机器翻译**：Transformer 在 WMT 2014 数据集上取得了最先进的性能
- **文本生成**：GPT 系列模型基于 Transformer 解码器
- **文本理解**：BERT 基于 Transformer 编码器
- **代码生成**：GitHub Copilot 使用基于 Transformer 的模型

例 7.1 (BERT). *BERT*(*Bidirectional Encoder Representations from Transformers*) 使用 *Transformer* 编码器和掩码语言模型预训练，在多个 *NLP* 任务上取得了突破性成果。

8 强化学习及其应用

强化学习（Reinforcement Learning, RL）是机器学习的一个分支，通过智能体（Agent）与环境（Environment）交互来学习最优策略。

8.1 强化学习基础

定义 8.1 (马尔可夫决策过程). 马尔可夫决策过程（*Markov Decision Process, MDP*）由五元组 $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ 定义：

- $\mathcal{S}$：状态空间
- $\mathcal{A}$：动作空间
- $\mathcal{P}$：状态转移概率，$P(s'|s,a)$
- $\mathcal{R}$：奖励函数，$R(s,a,s')$
- $\gamma \in [0,1]$：折扣因子

强化学习的目标：学习策略 $\pi(a|s)$，使得累积奖励的期望最大：

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{87}$$

其中，$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ 是轨迹。

价值函数：

- **状态价值函数**：$V^\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$
- **动作价值函数**：$Q^\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a]$

8.2 深度强化学习

深度强化学习（Deep Reinforcement Learning）将深度学习与强化学习结合，使用神经网络近似价值函数或策略。

Deep Q-Network (DQN)：DQN 使用深度神经网络近似 Q 函数：

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{88}$$

DQN 的关键技术：

- **经验回放（Experience Replay）**：存储经验 (s_t, a_t, r_t, s_{t+1})，随机采样进行训练
- **目标网络（Target Network）**：使用独立的网络计算目标 Q 值，提高训练稳定性

策略梯度方法：直接优化策略参数：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R_t \right] \tag{89}$$

其中，$R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ 是回报。

Actor-Critic 方法：结合策略梯度（Actor）和价值函数（Critic）：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A_t \right] \tag{90}$$

$$A_t = Q(s_t, a_t) - V(s_t) \quad \text{（优势函数）} \tag{91}$$

应用场景：

- **游戏 AI**：AlphaGo、AlphaStar、OpenAI Five
- **机器人控制**：机器人导航、操作任务
- **自动驾驶**：决策和控制
- **推荐系统**：动态推荐策略
- **资源调度**：云计算资源分配

例 8.1 (AlphaGo). *AlphaGo 使用深度强化学习（结合蒙特卡洛树搜索）击败了世界围棋冠军，展示了深度强化学习在复杂决策问题中的强大能力。*

例 8.2 (OpenAI Five). *OpenAI Five 在 Dota 2 游戏中击败了世界冠军团队，展示了多智能体强化学习的潜力。*

9 深度学习研究案例

9.1 计算机视觉

例 9.1 (ImageNet 图像分类). *ImageNet* 大规模视觉识别挑战 (*ILSVRC*) 推动了深度学习在计算机视觉领域的发展。*AlexNet*(*2012*) 首次使用深度卷积神经网络取得突破，随后的 *VGGNet*、*ResNet*、*DenseNet* 等不断刷新记录，错误率从 *25.8%* 降低到 *2.25%*。

例 9.2 (目标检测). *R-CNN* 系列 (*R-CNN*、*Fast R-CNN*、*Faster R-CNN*) 和 *YOLO* 系列推动了目标检测技术的发展，在 *COCO* 数据集上取得了优异的性能，广泛应用于自动驾驶、安防监控等领域。

9.2 自然语言处理

例 9.3 (机器翻译). *Google* 神经机器翻译 (*GNMT*) 使用编码器-解码器架构和注意力机制，在多个语言对上取得了接近人类水平的翻译质量。

例 9.4 (预训练语言模型). *BERT*、*GPT*、*T5* 等预训练语言模型通过大规模无监督预训练和任务特定微调，在多个 *NLP* 任务上取得了显著提升，推动了 *NLP* 领域的范式转变。

9.3 语音识别

例 9.5 (语音转文字). *Deep Speech*、*Wav2Vec* 等模型使用深度神经网络进行语音识别，在多个数据集上达到了接近人类水平的准确率，广泛应用于语音助手、实时字幕等场景。

9.4 多模态学习

例 9.6 (图像描述生成). *Show and Tell*、*Bottom-Up and Top-Down Attention* 等模型结合 *CNN* 和 *RNN/Transformer*，能够生成图像的文本描述，在 *COCO Captions* 数据集上取得了优异的性能。

10 总结与展望

深度学习作为人工智能领域的重要分支，通过多层神经网络学习数据的层次化表示，在多个领域取得了突破性进展。从感知机到 Transformer，从图像分类到自然语言处理，深度学习不断推动着人工智能的发展。

未来发展方向:

- **更高效的架构**：减少参数量和计算量，提高推理速度
- **更好的可解释性**：理解模型的决策过程，提高可信度
- **少样本学习**：减少对大规模标注数据的依赖
- **多模态融合**：整合文本、图像、语音等多种模态的信息
- **自监督学习**：从无标注数据中学习有效表示
- **神经符号结合**：结合神经网络的表示能力和符号推理的逻辑能力

深度学习将继续在科学研究、工业应用和社会生活中发挥重要作用，推动人工智能技术的进一步发展。

11 作业与练习

11.1 概念题

1. **梯度消失问题**：
   - 解释什么是梯度消失问题，为什么会出现?
   - 列举至少三种缓解梯度消失问题的方法，并说明其原理。
   - 为什么 ReLU 激活函数能够缓解梯度消失问题?
2. **卷积神经网络**：
   - 解释卷积操作中的局部连接和权重共享如何减少参数量。
   - 比较最大池化和平均池化的优缺点。
   - 为什么 CNN 适合处理图像数据?
3. **注意力机制**：
   - 解释自注意力和交叉注意力的区别。
   - 为什么 Transformer 需要位置编码?
   - 多头注意力的优势是什么?
4. **批量归一化**：
   - 解释批量归一化为什么能够加速训练。
   - 批量归一化在训练和测试阶段的区别是什么?

- 为什么批量归一化具有正则化效果?

5. **强化学习**:

- 解释强化学习与监督学习的区别。
- 什么是探索-利用权衡（Exploration-Exploitation Trade-off)?
- DQN 中的经验回放和目标网络的作用是什么?

6. **损失函数**:

- 比较均方误差（MSE）和平均绝对误差（MAE）的优缺点，各适用于什么场景?
- 为什么交叉熵损失函数与 softmax 激活函数配合使用效果最佳?
- 解释 Focal Loss 如何解决类别不平衡问题，其核心思想是什么?
- 在什么情况下应该使用 Huber Loss 而不是 MSE 或 MAE?
- 三元组损失函数在度量学习中的作用是什么？如何选择正负样本?

11.2 编程题

1. **实现多层感知机**:

- 使用 PyTorch 或 TensorFlow 实现一个三层 MLP（输入层、隐藏层、输出层）
- 在 MNIST 数据集上训练模型
- 实现前向传播和反向传播（可以调用框架的自动微分功能）
- 尝试不同的激活函数（Sigmoid、tanh、ReLU）并比较效果

2. **实现简单的 CNN**:

- 实现一个包含卷积层、池化层和全连接层的 CNN
- 在 CIFAR-10 数据集上训练模型
- 可视化卷积层的特征图，观察不同层学习到的特征

3. **实现 LSTM**:

- 使用 PyTorch 或 TensorFlow 实现 LSTM 单元
- 在文本分类任务（如情感分析）上训练模型
- 比较 LSTM 和简单 RNN 的性能差异

4. **实现注意力机制**:

- 实现自注意力机制
- 实现多头注意力机制
- 在序列到序列任务（如机器翻译）中应用注意力机制

5. **实现批量归一化**：

- 实现批量归一化层（包括训练和测试模式）
- 在深度网络中应用批量归一化，观察训练速度和最终性能的变化
- 比较使用和不使用批量归一化的训练曲线

6. **实现 Dropout**：

- 实现 Dropout 层（包括训练和测试模式）
- 在过拟合的模型上应用 Dropout，观察正则化效果
- 可视化 Dropout 对模型权重分布的影响

7. **实现不同的损失函数**：

- 实现 MSE、MAE 和 Huber Loss，比较它们在回归任务上的表现
- 实现交叉熵损失函数，在分类任务上验证其效果
- 实现 Focal Loss，在类别不平衡的数据集上比较其与标准交叉熵的差异
- 实现三元组损失函数，在图像检索任务中应用
- 尝试组合多个损失函数（如内容损失 + 风格损失），观察效果

11.3 综合项目

项目：图像分类系统

- 使用深度学习框架（PyTorch 或 TensorFlow）构建一个完整的图像分类系统
- 实现数据加载、预处理、模型定义、训练和评估的完整流程
- 尝试不同的网络架构（MLP、CNN、ResNet）
- 应用不同的优化技术（批量归一化、Dropout、学习率调度）
- 使用数据增强技术提高模型泛化能力
- 可视化训练过程（损失曲线、准确率曲线）
- 分析模型的错误案例，提出改进方案

项目：文本生成系统

- 使用 RNN/LSTM/GRU 或 Transformer 构建文本生成模型
- 在文本数据集（如小说、新闻）上训练模型
- 实现不同的采样策略（贪婪搜索、随机采样、束搜索）
- 评估生成文本的质量（困惑度、BLEU 分数等）
- 尝试不同的超参数设置，观察对生成质量的影响

项目：强化学习智能体

- 使用深度强化学习（DQN 或策略梯度方法）训练游戏智能体
- 在 OpenAI Gym 环境（如 CartPole、Atari 游戏）中训练
- 实现经验回放和目标网络（对于 DQN）
- 可视化训练过程和智能体的行为
- 分析不同超参数（学习率、折扣因子等）对性能的影响

11.4 研究性作业

1. **文献阅读**：阅读以下经典论文，总结其主要贡献：
   - LeCun et al. (1998). "Gradient-based learning applied to document recognition"
   - Krizhevsky et al. (2012). "ImageNet classification with deep convolutional neural networks"
   - He et al. (2016). "Deep residual learning for image recognition"
   - Vaswani et al. (2017). "Attention is all you need"
   - Devlin et al. (2018). "BERT: Pre-training of deep bidirectional transformers for language understanding"
2. **技术调研**：选择一个深度学习应用领域（如医疗影像、自动驾驶、金融风控），调研该领域的最新进展，包括：
   - 主要技术方法
   - 面临的挑战
   - 最新研究成果

- 实际应用案例

3. **实验设计**：设计一个实验来验证某个深度学习技术（如批量归一化、残差连接）的有效性：

   - 明确研究问题
   - 设计对比实验
   - 确定评估指标
   - 分析实验结果

通过完成以上作业和练习，读者可以深入理解深度学习的核心概念和技术，掌握实际应用的能力，为进一步的研究和应用打下坚实的基础。