# 深度学习

夏同　来源: 龚月姣《学习与优化》

2026 年 1 月 28 日

# 目录

# 第一章　神经网络与深度学习基础

## 1.1　启发性思考

深度学习的核心思想是模仿人脑处理信息的方式。人脑中约有 860 亿个神经元，每个神经元通过树突接收信号，在细胞体整合，若超过某个阈值则通过轴突释放信号（电脉冲）至其他神经元。这一“全有或全无”的激发模式启发了最早的数学模型——**人工神经元**（或称感知器）。

本章将系统学习：

- 人工神经元的基本组成：**权重**、**偏置**、**激活函数**
- 几种经典激活函数（Sigmoid, tanh, ReLU）的特性与对比
- **前馈神经网络**的结构与矩阵表示
- **Softmax 函数**及其温度系数的妙用
- 神经网络训练的完整流程：**前向传播**、**损失函数**、**梯度下降**与核心算法——**反向传播（Backpropagation, BP）**

## 1.2　神经元：深度学习的基石

### 1.2.1　数学模型

一个神经元接收 $n$ 个输入 $x_1, x_2, \ldots, x_n$，每个输入对应一个权重 $w_i$，并有一个偏置 $b$。神经元首先计算加权和（称为**预激活值**）：

$$z = \sum_{i=1}^{n} w_i x_i + b$$

然后将 $z$ 送入一个非线性函数 $f$，得到该神经元的**激活输出**：

$$a = f(z)$$

### 1.2.2　关键组件详解

**定义 1.2.1** (权重（Weight）). 权重 $w_i$ 衡量第 $i$ 个输入对神经元输出的**重要性**或**贡献度**。在训练过程中，权重会被不断调整，以学习输入与输出之间的映射关系。

- **理性理解**：权重是线性变换的参数，决定了输入空间到输出空间的投影方向与尺度。
- **感性理解**：好比人际交往中，不同朋友说的话对你决策的影响程度不同，权重就是这种“信任度”或“影响力”的量化。

**定义 1.2.2** (偏置（Bias）). 偏置 $b$ 是一个常数项，它**平移**了激活函数的阈值。即使所有输入为零，神经元也可能被激活（若 $b > 0$）。

- **理性理解**：偏置提供了模型的平移自由度，使决策边界不必通过原点，增强了模型的表达能力。
- **感性理解**：可以理解为一个人的“先天倾向”或“初始立场”。例如，即使没有任何外部信息（输入为零），一个人也可能因为内在性格（偏置）而倾向于做出某个决定。

**定义 1.2.3** (激活函数（Activation Function）). 激活函数 $f$ 引入了**非线性**。如果没有非线性，无论堆叠多少层，整个网络仍然等价于一个线性模型，无法学习复杂的模式。

- **理性理解**：非线性变换是神经网络能够成为“通用函数逼近器”的关键。
- **感性理解**：就像人脑神经元对输入信号的响应不是简单的线性叠加，而是有一个“激发”或“抑制”的非线性过程。

## 1.3　激活函数：从 Sigmoid 到 ReLU

### 1.3.1　经典激活函数

**Sigmoid 函数**

$$\sigma(z) = \frac{1}{1 + e^{-z}} \in (0, 1)$$

- **优点**：输出平滑、连续，导数易于计算 $\sigma'(z) = \sigma(z)(1 - \sigma(z))$。输出范围 $(0, 1)$ 适合表示概率。
- **缺点**：
  1. **梯度消失（Vanishing Gradient）**：当 $|z|$ 很大时，导数接近 0。在深层网络中，梯度反向传播时连乘会导致前面层的梯度极小，参数更新缓慢甚至停滞。
  2. 输出不是零中心的（均值 $>0$），可能导致后续层的输入发生偏移，影响梯度下降效率。
  3. 计算涉及指数，相对较慢。
- **用途**：二分类问题的输出层（表示概率）。

**双曲正切函数（tanh）**

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \in (-1, 1)$$

- **优点**：输出是**零中心**的（均值为 0），这通常有助于加速收敛。同样平滑连续。
- **缺点**：同样存在梯度消失问题。
- **用途**：适合需要输出有界且零中心的场景，如 RNN 的隐藏状态。

**线性整流单元（ReLU）**

$$\mathrm{ReLU}(z) = \max(0, z)$$

- **优点**：
  1. **计算高效**：只需比较和取最大值。
  2. **缓解梯度消失**：在正半轴导数为 1，梯度可以无衰减地反向传播。
  3. 在实践中，ReLU 网络通常比 Sigmoid/tanh 网络**收敛快得多**。
- **缺点**：
  1. **死亡 ReLU 问题（Dying ReLU）**：一旦输入落入负半轴（例如，由于大的负偏置或激烈的权重更新），输出和梯度都为零，神经元“死亡”，不再参与后续学习。
  2. 输出非零中心，且无上界，可能导致梯度爆炸（需配合适当初始化与归一化）。

### 1.3.2　激活函数对比与进阶选择

表 1.1: 激活函数对比

| 函数 | 输出范围 | 零中心 | 梯度消失 | 计算速度 |
|---|---|---|---|---|
| Sigmoid | (0,1) | 否 | 严重 | 慢 |
| tanh | (-1,1) | **是** | 严重 | 慢 |
| ReLU | $[0, \infty)$ | 否 | **缓解** | **极快** |

**ReLU 的改进变体**

- **Leaky ReLU**：$f(z) = \max(\alpha z, z)$，其中 $\alpha$ 是一个小的正数（如 0.01）。在负半轴保留一个微小斜率，避免神经元完全死亡。
- **PReLU（Parametric ReLU）**：将 $\alpha$ 作为可学习参数，让网络自行决定负半轴的斜率。
- **ELU（Exponential Linear Unit）**：$f(z) = \begin{cases} z, & z > 0 \\ \alpha(e^z - 1), & z \le 0 \end{cases}$。负半轴平滑渐近，输出接近零中心。
- **GELU / SiLU（Sigmoid Linear Unit）**：$f(z) = z \cdot \sigma(z)$。这是一种平滑的非单调激活函数，被 BERT、GPT 等 Transformer 模型采用，性能优异。

**实践建议**：现代深度网络中，ReLU 及其变体（尤其是 GELU）已成为隐藏层的默认选择。Sigmoid 多用于输出层（二分类概率），tanh 在特定架构（如 LSTM）中仍有应用。

# 1.4　前馈神经网络：层叠的威力

## 1.4.1　网络结构

前馈神经网络（Feed-forward Neural Network）由多层神经元组成，信息**单向流动**：从输入层，经过若干隐藏层，到达输出层。

- **不存在循环或反馈**（有循环的模型是循环神经网络 RNN）。
- 每一层的神经元接收前一层所有神经元的输出作为输入，并输出给下一层所有神经元（全连接）。

## 1.4.2　优雅高效的矩阵表示

设网络共 $L$ 层（输入层为第 0 层，输出层为第 $L$ 层）。第 $l$ 层有 $d^{(l)}$ 个神经元。

- 第 $l$ 层的**权重矩阵** $W^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l-1)}}$：$W_{ij}^{(l)}$ 表示第 $l-1$ 层第 $j$ 个神经元到第 $l$ 层第 $i$ 个神经元的连接权重。
- 第 $l$ 层的**偏置向量** $b^{(l)} \in \mathbb{R}^{d^{(l)}}$。
- 第 $l$ 层的**输入**（即前一层的激活输出）记为 $a^{(l-1)} \in \mathbb{R}^{d^{(l-1)}}$。

则第 $l$ 层的**预激活值** $z^{(l)}$ 和**激活输出** $a^{(l)}$ 为：

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}, \quad a^{(l)} = f^{(l)}(z^{(l)})$$

其中 $f^{(l)}$ 是第 $l$ 层的激活函数。这种矩阵形式极其适合 GPU 并行计算。

## 1.4.3　Softmax 函数：多分类的“裁判”

### 标准 Softmax

对于 $K$ 类的分类问题，输出层通常有 $K$ 个神经元，输出一个未经归一化的得分向量（logits）$z \in \mathbb{R}^K$。Softmax 将其转换为一个合法的概率分布：

$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad i = 1, \ldots, K$$

- **指数作用**：将实数映射到正数，并**放大分数差距**（高分更高，低分更低）。
- **归一化**：确保所有输出之和为 1，符合概率公理。
- **用途**：多分类任务的输出层，常与交叉熵损失函数搭配。

### 温度系数（Temperature）

引入温度系数 $T > 0$，控制概率分布的“软硬”程度：

$$\text{Softmax}(z, T)_i = \frac{e^{z_i/T}}{\sum_{j=1}^{K} e^{z_j/T}}$$

- $T = 1$：标准 Softmax。
- $T > 1$（高温）：概率分布更“平坦”，模型对各类别的不确定性增加。常用于**知识蒸馏**（Teacher 模型使用高温度产生软标签，Student 模型学习之）。
- $T < 1$（低温）：概率分布更“尖锐”，模型置信度更高。当 $T \to 0^+$ 时，Softmax 退化为 argmax（即 one-hot 向量）。

## 1.5　神经网络的训练：让模型学会思考

训练神经网络本质是寻找一组参数（所有权重和偏置），使模型在训练数据上的预测损失最小。这是一个典型的优化问题。

### 1.5.1　步骤一：定义网络结构

- **输入层维度**：由数据特征决定。
- **隐藏层数量与宽度**：决定模型容量。层数越多、每层神经元越多，模型拟合复杂函数的能力越强，但也更容易过拟合、计算成本更高。
- **输出层维度**：由任务决定（如二分类为 1，多分类为类别数，回归为 1）。

**数据预处理**：通常将输入特征标准化（均值 0，方差 1）或归一化到 $[0, 1]$，以加速收敛、提升性能。

### 1.5.2　步骤二：定义损失函数（Loss Function）

损失函数量化模型预测 $\hat{y}$ 与真实标签 $y$ 之间的差距。

- **均方误差（MSE）**：用于回归任务。

$$J(W, b) = \frac{1}{2m} \sum_{i=1}^{m} (h_{W,b}(x^{(i)}) - y^{(i)})^2$$

其中 $m$ 是样本数，乘以 $\frac{1}{2}$ 是为了后续求导方便（与平方项的导数 2 抵消）。

- **交叉熵损失（Cross-Entropy）**：用于分类任务，尤其是与 Softmax 输出层结合。

$$J(W, b) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log(\hat{y}_k^{(i)})$$

其中 $K$ 是类别数，$y_k^{(i)}$ 是样本 $i$ 的真实标签（one-hot 向量），$\hat{y}_k^{(i)}$ 是模型预测的概率。

**注记 1.5.1.** 交叉熵衡量两个概率分布之间的“距离”。当预测分布与真实分布完全一致时，交叉熵最小（等于真实分布的熵）。

### 1.5.3　步骤三：梯度下降优化

目标：最小化 $J(W,b)$。梯度下降（Gradient Descent）是核心优化算法。

$$W^{(l)} := W^{(l)} - \alpha \frac{\partial J}{\partial W^{(l)}}, \quad b^{(l)} := b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}}$$

其中 $\alpha$ 是**学习率（Learning Rate）**，控制更新步长。

- **理性**：沿着损失函数梯度的反方向（最速下降方向）更新参数。
- **感性**：好比在山区蒙眼下山，用脚试探周围最陡的下坡方向，然后迈一小步，重复此过程。

**核心挑战**：如何高效计算所有参数的梯度 $\frac{\partial J}{\partial W^{(l)}}$ 和 $\frac{\partial J}{\partial b^{(l)}}$？这正是反向传播要解决的。

## 1.6　反向传播：深度学习的引擎

反向传播（Backpropagation, BP）不是独立的优化算法，而是**利用链式法则高效计算梯度**的方法。它是神经网络训练的核心。

### 1.6.1　误差项（Error Term）的定义

对于第 $l$ 层的第 $i$ 个神经元，定义其误差项 $\delta_i^{(l)}$ 为损失函数对该神经元预激活值 $z_i^{(l)}$ 的偏导数：

$$\delta_i^{(l)} = \frac{\partial J}{\partial z_i^{(l)}}$$

误差项衡量了该神经元对最终损失的“责任”大小。

### 1.6.2　反向传播的四步流程

#### 1. 前向传播

给定输入 $x$，依次计算并保存每一层的 $z^{(l)}$ 和 $a^{(l)}$（$a^{(0)} = x$）。

#### 2. 计算输出层误差

$$\delta^{(L)} = \nabla_{a^{(L)}} J \odot f'(z^{(L)})$$

其中 $\nabla_{a^{(L)}} J$ 是损失对输出层激活的梯度，$\odot$ 表示逐元素乘法（Hadamard 积）。

- 对于 MSE 损失和线性输出：$\nabla_{a^{(L)}} J = a^{(L)} - y$。
- 对于交叉熵损失和 Softmax 输出：$\nabla_{a^{(L)}} J = a^{(L)} - y$（这是一个美妙的巧合，简化了计算）。

**3. 反向传播误差（关键递推公式）**

从 $l = L-1$ 到 $l = 1$，逐层计算：

$$\boxed{\delta^{(l)} = \left((W^{(l+1)})^\top \delta^{(l+1)}\right) \odot f'(z^{(l)})}$$

- **理性解释：**后一层的误差 $\delta^{(l+1)}$ 通过权重矩阵的转置 $(W^{(l+1)})^\top$ 传播回前一层，再乘以当前层激活函数的导数，进行调制。
- **感性解释：**将最终错误（损失）归咎于每一层神经元的“失误”。高层神经元的错误分摊给其输入连接的底层神经元，分摊比例就是连接的权重，同时还要考虑该底层神经元当时是否处于“敏感”状态（由 $f'(z^{(l)})$ 体现）。

**4. 计算参数梯度**

一旦得到 $\delta^{(l+1)}$，第 $l$ 层参数的梯度就非常简洁：

$$\boxed{\frac{\partial J}{\partial W^{(l)}} = \delta^{(l+1)}(a^{(l)})^\top}, \quad \boxed{\frac{\partial J}{\partial b^{(l)}} = \delta^{(l+1)}}$$

- **权重梯度：**等于后一层的误差 $\delta^{(l+1)}$ 与前一层的激活 $a^{(l)}$ 的外积。
- **偏置梯度：**就是后一层的误差 $\delta^{(l+1)}$ 本身。

**5. 更新参数**

使用梯度下降（或其变体如 Adam）更新所有参数：

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial J}{\partial W^{(l)}}, \quad b^{(l)} \leftarrow b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}}$$

**6. 批处理与梯度累加**

实际中我们使用小批量（mini-batch）数据计算梯度。步骤 4 中计算出的梯度通常是该批次内所有样本梯度的**平均值**。这既利用了向量化计算的效率，又引入了噪声，有助于逃离局部极小点。

### 1.6.3　反向传播推导（选读）

理解推导有助于深入掌握 BP 本质。核心是链式法则。我们欲求 $\frac{\partial J}{\partial W_{ij}^{(l)}}$。注意 $J$ 通过 $z_i^{(l+1)}$ 依赖于 $W_{ij}^{(l)}$。

$$\begin{aligned}
\frac{\partial J}{\partial W_{ij}^{(l)}} &= \frac{\partial J}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial W_{ij}^{(l)}} \\
&= \delta_i^{(l+1)} \cdot \frac{\partial}{\partial W_{ij}^{(l)}} \left( \sum_k W_{ik}^{(l)} a_k^{(l)} + b_i^{(l)} \right) \\
&= \delta_i^{(l+1)} \cdot a_j^{(l)}
\end{aligned}$$

这正是 $\frac{\partial J}{\partial W^{(l)}} = \delta^{(l+1)}(a^{(l)})^\top$ 的第 $(i,j)$ 个元素。

误差反向传播公式的推导类似，考虑 $J$ 通过所有 $z_j^{(l+1)}$ 依赖于 $z_i^{(l)}$：

$$\begin{aligned}\delta_i^{(l)} = \frac{\partial J}{\partial z_i^{(l)}} &= \sum_j \frac{\partial J}{\partial z_j^{(l+1)}} \cdot \frac{\partial z_j^{(l+1)}}{\partial z_i^{(l)}} \\ &= \sum_j \delta_j^{(l+1)} \cdot \frac{\partial}{\partial z_i^{(l)}} \left( \sum_k W_{jk}^{(l+1)} f(z_k^{(l)}) + b_j^{(l+1)} \right) \\ &= \sum_j \delta_j^{(l+1)} \cdot W_{ji}^{(l+1)} \cdot f'(z_i^{(l)}) \\ &= \left( \sum_j W_{ji}^{(l+1)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})\end{aligned}$$

括号内即为 $((W^{(l+1)})^\top \delta^{(l+1)})_i$，因此有 $\delta^{(l)} = ((W^{(l+1)})^\top \delta^{(l+1)}) \odot f'(z^{(l)})$。

## 1.7　引言：从拟合数据到泛化世界

深度学习的核心目标是构建能够在**未见过的数据**上表现良好的模型。这就像学生在期末考试中遇到新题时能灵活运用所学知识，而不是只会做练习题。本章将系统学习如何评估模型性能、诊断常见问题、优化模型泛化能力，并探索深度学习从神经网络到现代架构的发展历程。

## 1.8　模型性能评估：不只是看分数

### 1.8.1　数据集划分的艺术

为了模拟模型在真实世界中的表现，我们需要将数据划分为三个互斥的部分：

- **训练集（Training Set）**：模型的" 教科书"，用于学习参数（权重和偏置）。
- **验证集（Validation Set）**：模型的" 模拟考试"，用于：
  1. **超参数调优**：选择学习率、网络层数、神经元数量等。
  2. **模型选择**：比较不同架构的性能。
  3. **早停法（Early Stopping）**的参考依据。
- **测试集（Test Set）**：模型的" 期末考试"，在整个训练和调优过程中只使用一次，用于给出模型性能的**无偏估计**。

**黄金法则**：测试集必须与训练/验证集完全隔离，在整个流程的最后才使用一次。任何基于测试集结果的模型调整都会导致对泛化能力的**乐观估计**。

### 1.8.2　K-折交叉验证：小数据集的救星

当数据量有限时，简单的单次划分可能因随机性导致评估结果不稳定。K-折交叉验证提供了更稳健的解决方案。

**算法流程：**

1. 将训练数据（不包含测试集）随机划分为 K 个大小相似的互斥子集（" 折"），通常 K=5 或 10。
2. 进行 K 次循环，在第 i 次迭代中：
   - 使用第 i 折作为验证集
   - 使用其余 K-1 折作为训练集
3. 将 K 次迭代得到的性能指标取平均值，作为模型性能的最终评估。

**注记 1.8.1. 优点：**

- 充分利用有限数据：每个样本都既当过训练数据也当过验证数据。
- 减少评估结果的随机性：基于多次评估的平均结果更可靠。

**缺点：**计算成本是单次划分的 $K$ 倍。

## 1.9　诊断模型问题：欠拟合与过拟合

模型性能不佳通常源于两个根本问题：


(a) 误差随模型复杂度的变化　(b) 早停法的原理

图 1.1: 欠拟合与过拟合的图示

### 1.9.1　欠拟合（Underfitting）

- **表现：**训练误差和测试误差都很高。
- **原因：**模型太简单，无法捕捉数据中的基本规律。
- **类比：**用线性方程去拟合抛物线数据。
- **解决方案：**
  1. 增加模型复杂度（更多层、更多神经元）。
  2. 使用更强大的特征。

3. 减少正则化强度。
4. 延长训练时间。

### 1.9.2　过拟合（Overfitting）

- **表现**：训练误差很低，但测试误差很高。
- **原因**：模型过于复杂，记住了训练数据中的噪声和偶然性。
- **类比**：死记硬背所有例题，但遇到新题不会做。
- **解决方案**：
  1. 获取更多训练数据（最有效但成本高）。
  2. 使用正则化技术。
  3. 简化模型结构。
  4. 早停法。
  5. Dropout（仅限神经网络）。

### 1.9.3　偏差-方差权衡：统计学习理论的视角

泛化误差可以分解为三个部分：

$$\underbrace{\mathbb{E}[(y-\hat{f}(x))^2]}_{\text{泛化误差}}=\underbrace{[\text{Bias}(\hat{f}(x))]^2}_{\text{偏差}}+\underbrace{\text{Var}(\hat{f}(x))}_{\text{方差}}+\underbrace{\sigma^2}_{\text{不可约减误差}}$$

- **偏差（Bias）**：模型预测的期望值与真实值之间的差距，反映了模型的**系统性错误**。
  - 高偏差：模型过于简单，无法拟合数据的基本模式（欠拟合）。
  - **感性理解**：就像用直尺测量曲面，无论测多少次都有系统误差。
- **方差（Variance）**：模型对训练数据变化的敏感度，反映了模型的**不稳定性**。
  - 高方差：模型过于复杂，对训练数据中的随机噪声敏感（过拟合）。
  - **感性理解**：就像用过于灵敏的天平，每次读数都因微小扰动而不同。
- **不可约减误差（Irreducible Error）**：数据本身的噪声，任何模型都无法消除。

**权衡**：增加模型复杂度通常降低偏差但增加方差，减少模型复杂度则相反。理想的模型需要在两者之间找到平衡点。

## 1.10　正则化技术：防止过拟合的利器

正则化通过在损失函数中添加**惩罚项**来约束模型复杂度，防止过拟合。

### 1.10.1　L1 和 L2 正则化

在损失函数中添加参数的惩罚项：

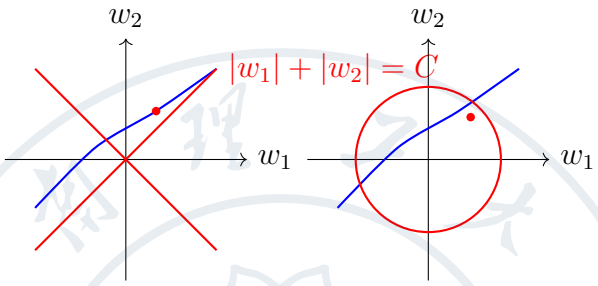
$$J_{\text{reg}}(W,b)=J(W,b)+\lambda\cdot R(W)$$

其中 $\lambda > 0$ 是正则化强度超参数。

表 1.2: L1 vs L2 正则化对比

| 类型 | 惩罚项 $R(W)$ | 数学形式 | 效果 |
|---|---|---|---|
| L1 正则化（Lasso） | $\lambda \sum_{i=1}^{n} \lvert w_i \rvert$ | 绝对值之和 | **稀疏化：**将不重要特征的权重压缩到 0 |
| L2 正则化（Ridge） | $\frac{\lambda}{2} \sum_{i=1}^{n} w_i^2$ | 平方和 | **权重衰减：**使所有权重更小、更平均 |

**几何解释**




L1 正则化（菱形约束）　L2 正则化（圆形约束）

图 1.2: L1 和 L2 正则化的几何解释

L1 正则化的约束区域是菱形，最优解往往在角点上（某些权重为 0），从而实现特征选择。L2 正则化的约束区域是圆形，最优解使所有权重都较小且分布均匀。

### 1.10.2　Dropout：训练时随机丢弃神经元

Dropout 是一种在**训练阶段**使用的正则化技术：

- 每个训练步骤中，以概率 $p$ 随机”丢弃”（设为 0）每个神经元的输出。
- 迫使网络不依赖于任何单个神经元，学习更鲁棒的特征。
- 测试阶段使用所有神经元，但输出要乘以 $1 - p$（或训练时缩放）。

### 1.10.3　早停法（Early Stopping）

监控验证集误差，当验证误差连续多个 epoch 不再下降（甚至上升）时停止训练。

- **优点：**简单有效，不需要改变模型结构。
- **缺点：**需要额外的验证集，停止时机不易确定。

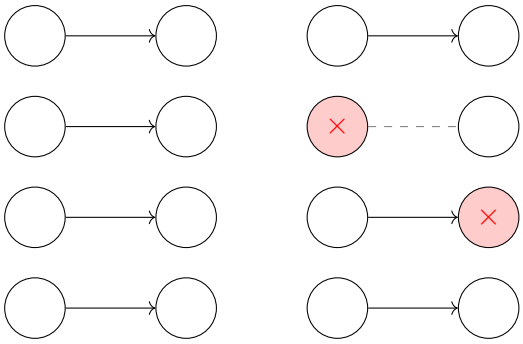



图 1.3: Dropout 示意图

## 1.11　从神经网络到深度学习：历史脉络

### 1.11.1　发展历程

**早期探索（1940s-1980s）**

- 1943 年：McCulloch 和 Pitts 提出 M-P 神经元模型，模拟生物神经元。
- 1958 年：Rosenblatt 提出感知机（Perceptron），可视为单层神经网络。
- 1969 年：Minsky 和 Papert 出版《Perceptrons》，指出单层感知机无法解决 XOR 等非线性问题，导致神经网络研究进入寒冬。

**复兴前夜（1980s-2006）**

- 1986 年：Rumelhart 等人提出反向传播算法，解决了多层网络训练问题。
- 但仍受限于：数据不足、计算力有限、优化困难（梯度消失/爆炸）。

**复兴之年（2006）**

- Hinton 等人发表《A Fast Learning Algorithm for Deep Belief Nets》，提出：
  1. **逐层贪婪预训练**：逐层训练受限玻尔兹曼机（RBM）。
  2. **微调**：用反向传播微调整个网络。
- 解决了深度网络训练难题，开启了深度学习新时代。

**爆发之年（2012）**

- AlexNet 在 ImageNet 竞赛中夺冠，将 Top-5 错误率从 26% 降至 15%。
- 关键创新：使用 ReLU、Dropout、GPU 加速。
- 证明学习到的特征可超越手工设计特征。

### 1.11.2　深度学习的四大支柱

表 1.3: 深度学习的四大支柱

| 支柱 | 内容 | 意义 |
| --- | --- | --- |
| **数据** | 大规模标注数据（ImageNet 等），互联网产生海量数据 | 燃料，数据量决定模型性能上限 |
| **算力** | GPU 并行计算，TPU 专用芯片，云计算平台 | 引擎，使训练深层网络成为可能 |
| **算法** | 新激活函数（ReLU），正则化（Dropout），优化器（Adam），新架构（ResNet） | 智慧，提升模型性能与训练稳定性 |
| **开源生态** | TensorFlow, PyTorch | 土壤，加速研究与应用普及 |

### 1.11.3　为什么叫”深度学习”?

**深**指网络层数多（通常 >3 层）。与传统机器学习相比：

表 1.4: 传统机器学习 vs 深度学习

| 传统机器学习 | 深度学习 |
| --- | --- |
| **特征工程**：人工设计特征 | **特征学习**：自动从数据中学习特征 |
| **流水线式**：多个独立步骤 | **端到端**：单个模型完成所有任务 |
| **浅层模型**：1-2 层 | **深层模型**：数十至数百层 |
| **可解释性强** | **可解释性弱**（黑盒） |
| **数据需求少** | **数据需求大** |

**深度学习的关键优势**：通过多层非线性变换，自动学习**层次化特征表示**：

- 浅层：边缘、纹理、颜色
- 中层：部件、形状
- 深层：语义概念、对象

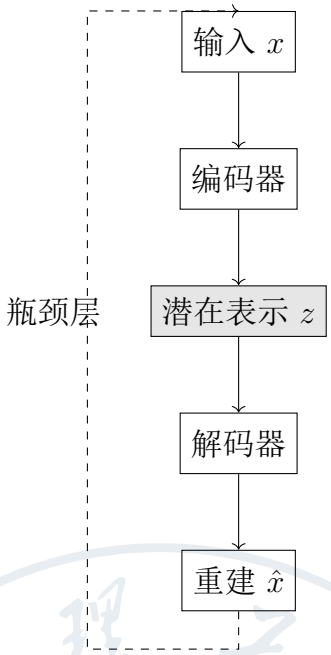



图 1.4: 自编码器结构示意图

## 1.12　关键模型架构

### 1.12.1　自编码器（Autoencoders, AEs）

**核心思想**：学习数据的压缩表示（编码），然后重建输入。

$$\text{编码器：} z = f(x) = \sigma(Wx + b)$$
$$\text{解码器：} \hat{x} = g(z) = \sigma(W'z + b')$$
$$\text{损失函数：} L(x, \hat{x}) = \|x - \hat{x}\|^2$$

**变体与应用：**

- **稀疏自编码器**：在损失函数中添加稀疏约束。
- **去噪自编码器**：输入加噪声，要求重建原始干净数据。
- **变分自编码器（VAE）**：学习数据的概率分布，可生成新样本。

### 1.12.2　卷积神经网络（CNN）

专门处理网格状数据（如图像），核心思想：**局部连接、权重共享、平移不变性**。

**核心组件**

1. **卷积层**：使用卷积核在输入上滑动，计算局部加权和。

$$(I * K)_{i,j} = \sum_m \sum_n I_{i+m,j+n} \cdot K_{m,n}$$

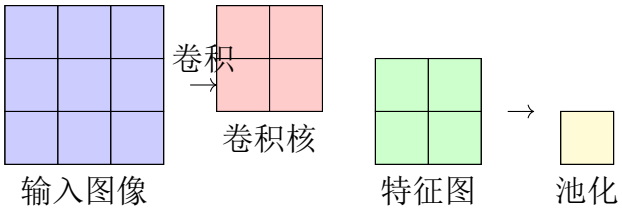



图 1.5: CNN 基本操作：卷积与池化

2. **池化层**：下采样，减少参数量，增加平移不变性。
   - 最大池化：$\text{MaxPool}(x)_{i,j} = \max(x_{2i:2i+1,2j:2j+1})$
   - 平均池化：$\text{AvgPool}(x)_{i,j} = \text{mean}(x_{2i:2i+1,2j:2j+1})$
3. **全连接层**：将特征图展平后分类。

#### 经典架构

- **AlexNet**（2012）：首次使用 ReLU、Dropout、GPU 训练。
- **VGG**（2014）：使用 3×3 小卷积核堆叠深层网络。
- **ResNet**（2015）：残差连接解决梯度消失，训练极深网络（152 层）。

### 1.12.3　循环神经网络（RNN）与 LSTM

专门处理序列数据，具有**记忆能力**。

#### 标准 RNN

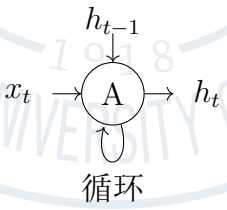



图 1.6: RNN 的表示

RNN 的状态更新公式：

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

**问题**：标准 RNN 使用 tanh 激活函数，虽然能缓解梯度爆炸，但仍有梯度消失问题，难以学习长期依赖。

#### LSTM：长短期记忆网络

LSTM 通过门控机制解决长期依赖问题。

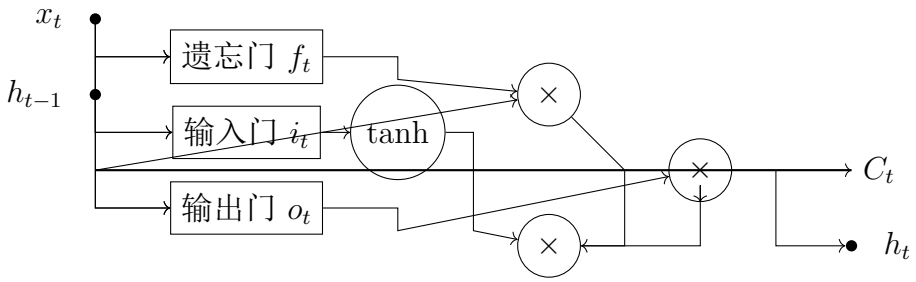



图 1.7: LSTM 单元结构

**LSTM 的三个门：**

1. **遗忘门**：决定从细胞状态中丢弃什么信息

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

2. **输入门**：决定将哪些新信息存入细胞状态

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

3. **输出门**：决定输出什么信息

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

**细胞状态更新：**

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

**注记 1.12.1. 为什么 *LSTM* 能解决长期依赖？**

- 细胞状态 $C_t$ 像一条”传送带”，信息可以几乎不变地流过。
- 门控机制选择性地让信息通过。
- 梯度在细胞状态上可以稳定传播，不易消失。

### 1.12.4　生成对抗网络（GANs）

**核心思想**：两个神经网络相互博弈

- **生成器** $G$：学习真实数据分布，生成以假乱真的样本。
- **判别器** $D$：区分真实数据与生成数据。

**目标函数（极小极大博弈）：**

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

**训练过程：**

1. 固定 $G$，训练 $D$：最大化区分真实与假数据的能力。
2. 固定 $D$，训练 $G$：最小化 $\log(1 - D(G(z)))$，即让生成数据更易骗过 $D$。

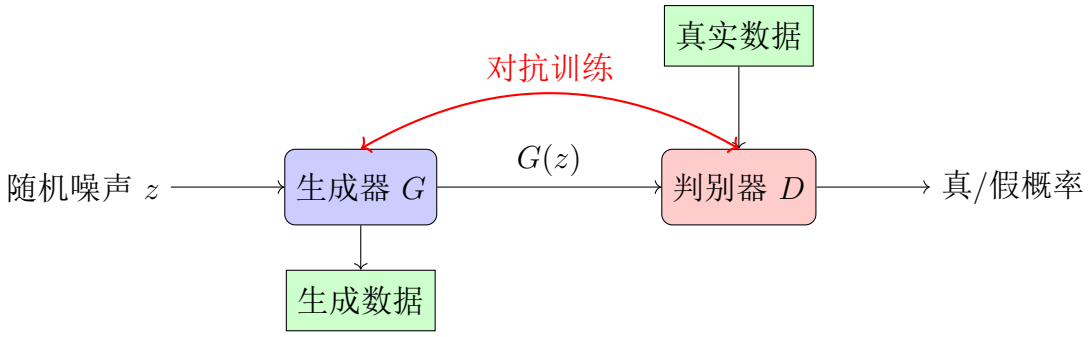



图 1.8: GAN 的基本结构

### 1.12.5　扩散模型（Diffusion Models）

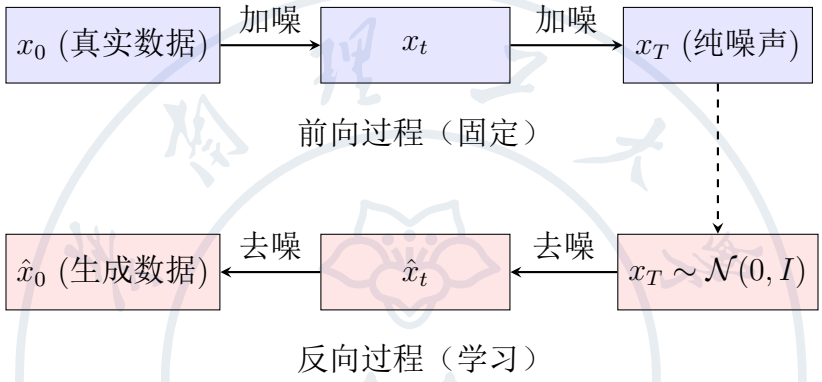



图 1.9: 扩散模型的前向与反向过程

**核心思想：**

1. **前向过程：**逐步向数据添加高斯噪声，直到变成纯噪声。

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

2. **反向过程：**学习去噪过程，从噪声中重建数据。

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

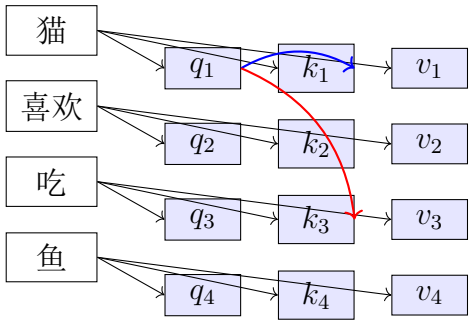
**优势：**生成质量高，训练稳定，无需对抗训练。

## 1.13　Transformer：注意力就是全部

### 1.13.1　自注意力机制（Self-Attention）

传统 RNN/LSTM 的问题是顺序处理、难以并行、长距离依赖衰减。Transformer 的解决方案：完全基于注意力。

**Query, Key, Value 概念：**




注意力得分：

$$\begin{aligned}\alpha_{ij} &= 0.8v_1 + 0.1v_2 + 0.1v_3 + 0.0v_4 \\ &= 0.2v_1 + 0.6v_2 + 0.1v_3 + 0.1v_4 \\ &= 0.1v_1 + 0.2v_2 + 0.6v_3 + 0.1v_4 \\ &= 0.0v_1 + 0.1v_2 + 0.1v_3 + 0.8v_4\end{aligned}$$

图 1.10: 自注意力机制示例：”吃”与”鱼”有强关联

- **Query（查询）**：当前要计算注意力的位置。
- **Key（键）**：所有位置，用于与 Query 计算相关性。
- **Value（值）**：所有位置的实际信息。

**计算过程：**

1. 对每个输入 $x_i$，通过可学习权重矩阵计算 $q_i = W^Q x_i$, $k_i = W^K x_i$, $v_i = W^V x_i$。
2. 计算注意力分数：$\alpha_{ij} = \text{softmax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right)$，其中 $d_k$ 是 Key 的维度（缩放因子防止梯度消失）。
3. 加权求和：$c_i = \sum_j \alpha_{ij} v_j$。

**矩阵形式：**

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

### 1.13.2　多头注意力（Multi-Head Attention）

单一注意力机制可能只关注特定类型的模式。多头注意力允许模型同时关注不同子空间的信息。

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

### 1.13.3　Transformer 架构

**编码器（Encoder）**

- 由 N 个相同层堆叠。
- 每层包含：多头自注意力 + 前馈神经网络 + 残差连接 + 层归一化。

**解码器（Decoder）**

- 也由 N 个相同层堆叠。
- 每层包含：
  1. **掩码多头自注意力**：防止当前位置看到未来信息（因果掩码）。

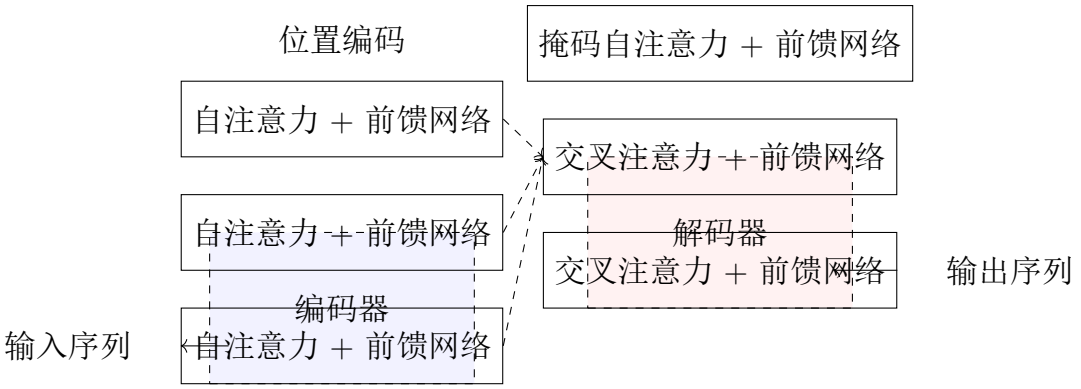



图 1.11: Transformer 架构

2. **多头交叉注意力**：Query 来自解码器，Key/Value 来自编码器。
3. 前馈神经网络 + 残差连接 + 层归一化。

### 1.13.4　位置编码（Positional Encoding）

自注意力机制本身不包含位置信息，需要额外注入位置编码：

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

其中 $pos$ 是位置，$i$ 是维度索引。这种正弦编码能够捕捉相对位置关系。

### 1.13.5　Transformer vs RNN/LSTM

表 1.5: Transformer 与 RNN/LSTM 对比

| 特性 | **Transformer** | **RNN/LSTM** |
|---|---|---|
| **核心机制** | 自注意力（全局依赖） | 循环结构（顺序依赖） |
| **并行能力** | **高**（所有位置并行计算） | **低**（必须顺序计算） |
| **长距离依赖** | **O(1)** 路径长度 | **O(n)** 路径长度，易梯度消失 |
| **计算复杂度** | $O(L^2 \cdot d)$，$L$ 为序列长度 | $O(L \cdot d^2)$ |
| **显存占用** | 高（需存储注意力矩阵） | 低 |
| **可解释性** | **高**（注意力权重可视化） | 低（隐藏状态难解释） |
| **位置信息** | 需显式添加位置编码 | 天然包含位置信息 |

## 1.14　为什么 Transformer 能够” 通吃”?

### 1.14.1　从专家模型到通用模型

- **旧范式**：各司其职的专家模型
  - 图像：CNN（卷积核捕捉局部特征）
  - 序列：RNN/LSTM（循环结构捕捉时序）
  - 归纳偏置强，但领域受限。
- **新范式**：万物皆序列的通用模型
  - 文本：词序列 $[w_1, w_2, \ldots, w_n]$
  - 图像：切分为 patch 序列
  - 音频：帧序列
  - 视频：帧序列的序列
- **统一处理**：所有数据都转化为标记序列，用 Transformer 处理。

### 1.14.2　Transformer 的核心优势

1. **全局依赖建模**：自注意力直接建立任意两个位置的联系。
2. **并行计算**：极大加速训练。
3. **可扩展性**：模型规模（参数量、数据量、计算量）增加带来稳定性能提升。
4. **多模态统一**：相同的架构处理不同模态数据。

## 1.15　总结与思考

本章从模型评估和优化出发，探讨了深度学习的关键问题：

- **评估**：通过数据集划分和交叉验证获得可靠的性能估计。
- **诊断**：识别欠拟合和过拟合，理解偏差-方差权衡。
- **优化**：使用正则化、Dropout、早停法提升泛化能力。
- **发展**：从简单神经网络到深度学习的演进，四大支柱的支撑。
- **架构**：CNN、RNN/LSTM、GAN、扩散模型、Transformer 等各有所长。
- **未来**：Transformer 展示了通用人工智能架构的潜力，大模型时代已经到来。

**思考**：深度学习的发展不仅是算法的进步，更是数据、算力、算法和开源生态协同发展的结果。理解这些基础原理，才能更好地应用和创新。

# 第二章　深度强化学习之基础

## 2.1　引言：从有答案的学习到探索的学习

### 2.1.1　监督学习的辉煌与局限

#### 监督学习的核心思想

监督学习（Supervised Learning）是机器学习中最经典、最直观的范式。它的核心思想可以概括为：

**从”有标签”的数据中学习。**

- **学习方式：**模型通过学习大量的（输入，正确输出）样本对，来建立输入到输出的映射关系。
- **类比：**就像学生跟着老师学习，老师为每个问题提供标准答案。
- **数学模型：**给定训练集 $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$，学习函数 $f : \mathcal{X} \to \mathcal{Y}$，使得 $f(x^{(i)}) \approx y^{(i)}$。

#### 监督学习的成功案例

1. **图像分类：**输入图片，输出类别标签（猫/狗/车等）。
2. **语音识别：**输入音频波形，输出对应文本。
3. **机器翻译：**输入源语言句子，输出目标语言句子。

#### 监督学习的局限性

尽管监督学习非常强大，但它并非万能。在某些场景下会面临严峻挑战：

表 2.1: 监督学习的局限性

| 挑战类型 | 具体描述与示例 |
| --- | --- |
| 缺乏”标准答案” | • 许多复杂决策问题没有唯一的”正确答案”<br>• **示例**：围棋中每一步的好坏难以用简单的对错衡量<br>• **示例**：自动驾驶中的决策权衡（安全 vs 效率） |
| 序贯决策需求 | • 需要做一系列相关决策，而每一步的优劣要到很久之后才能判断<br>• **示例**：下棋时需要规划多步之后的局面<br>• **示例**：机器人需要完成多步动作才能达到目标 |
| 动态变化的环境 | • 环境不断变化，今天的”最优解”明天可能失效<br>• **示例**：金融市场交易策略<br>• **示例**：游戏 AI 应对不同玩家风格 |
| 探索与利用的权衡 | • 如何在已知的好方法和尝试新方法之间取得平衡<br>• **示例**：推荐系统：推荐用户已知喜欢的内容 vs 尝试新内容 |

### 2.1.2　强化学习：一种新的学习范式

**从”知道答案”到”探索答案”**

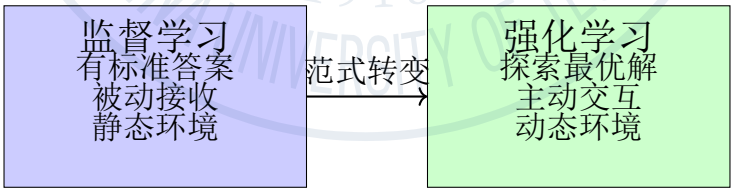



图 2.1: 从监督学习到强化学习的范式转变

**强化学习（Reinforcement Learning, RL）**的核心思想：
**通过与环境的”互动”和”试错”来学习。**

- **没有现成答案**：学习者（智能体）通过行动获得奖励或惩罚。
- **学习目标**：学会采取能最大化**长期累积奖励**的策略。
- **类比**：就像婴儿学习走路，通过尝试和摔倒（惩罚）来学会平衡和前进（奖励）。

**人类学习与强化学习的类比**

表 2.2: 人类学习与强化学习的对比

| 学习要素 | 人类学习 | 强化学习 |
|---|---|---|
| 交互 | 与环境互动（如触摸物体） | 智能体与环境互动 |
| 反馈 | • 打碎杯子 → 受到惩罚<br>• 成绩提升 → 获得奖励 | • 错误动作 → 负奖励<br>• 正确动作 → 正奖励 |
| 分析 | • ” 打碎杯子” 的收益小<br>• ” 好成绩” 的收益大 | • 评估动作的期望回报<br>• 更新策略以最大化回报 |
| 学习 | • 减小力度<br>• 努力学习 | • 调整策略参数<br>• 更新价值函数 |

## 2.2　马尔可夫决策过程：强化学习的数学基础

### 2.2.1　马尔可夫决策过程的基本概念

马尔可夫决策过程（Markov Decision Process, MDP）是强化学习的标准数学框架，它形式化了智能体与环境交互的过程。

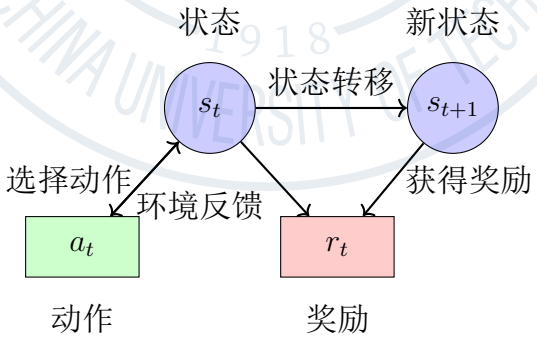



图 2.2: 马尔可夫决策过程的基本流程

**MDP 的五个核心要素**

一个 MDP 由以下五元组定义：$(\mathcal{S}, \mathcal{A}, P, R, \gamma)$

**定义 2.2.1** (状态空间 $\mathcal{S}$). 状态（*State*）是对当前环境的完整描述。状态空间是所有可能状态的集合。

- **理性理解**：状态是环境的”快照”，包含了做出决策所需的全部信息。
- **感性理解**：就像下棋时的棋盘局面，包含了所有棋子的位置信息。
- **示例**：在自动驾驶中，状态可能包括车辆位置、速度、周围车辆位置等。

**定义 2.2.2** (动作空间 $\mathcal{A}$). 动作（*Action*）是智能体可以执行的操作。动作空间是所有可能动作的集合。

- **理性理解**：动作是智能体对环境施加的控制信号。
- **感性理解**：就像棋手可以走的合法着法。
- **分类**：
  1. **离散动作空间**：动作数量有限（如上/下/左/右）
  2. **连续动作空间**：动作是连续的（如转向角度、油门大小）

**定义 2.2.3** (状态转移函数 $P$). 状态转移函数定义了环境动力学：$p(s' \mid s, a) = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$

- **理性理解**：在状态 $s$ 执行动作 $a$ 后，环境转移到状态 $s'$ 的概率。
- **感性理解**：就像知道在某个棋局走某步后，出现各种可能局面的概率。
- **马尔可夫性**：下一状态 $s'$ 只依赖于当前状态 $s$ 和动作 $a$，不依赖于更早的历史。

$$\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a, S_{t-1}, A_{t-1}, \ldots) = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$$

**定义 2.2.4** (奖励函数 $R$). 奖励函数定义了立即奖励：$R(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$

- **理性理解**：在状态 $s$ 执行动作 $a$ 后，期望获得的立即奖励。
- **感性理解**：就像游戏中的得分或扣分。
- **设计原则**：奖励函数的设计是强化学习成功的关键，需要准确反映任务目标。
  - **稀疏奖励**：只在关键时刻给予奖励（如获胜时 *+1*，失败时*-1*）
  - **稠密奖励**：每一步都有小奖励（如离目标越近奖励越大）

**定义 2.2.5** (折扣因子 $\gamma$). 折扣因子 $\gamma \in [0, 1]$ 权衡即时奖励与未来奖励的重要性。

$$U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots$$

- $\gamma = 0$：只关心即时奖励（短视）
- $\gamma \to 1$：几乎平等对待所有未来奖励（远见）
- **数学意义**：确保无限序列的回报有界（当 $\gamma < 1$ 时）
- **实际意义**：反映未来奖励的不确定性（”一鸟在手胜过双鸟在林”）

### 2.2.2　MDP 的核心概念

**策略函数 $\pi$**

**定义 2.2.6** (策略函数). 策略 $\pi(a \mid s)$ 是在状态 $s$ 下选择动作 $a$ 的概率分布：

$$\pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s)$$

- **确定性策略**：$\pi(s)$ 输出一个确定的动作（如 $\pi(s) =$ "向右移动"）
- **随机性策略**：$\pi(a \mid s)$ 输出动作的概率分布（如 $\pi(\text{"右"} \mid s) = 0.8, \pi(\text{"左"} \mid s) = 0.2$）
- **理性理解**：策略是智能体的"行为准则"或"决策规则"。
- **感性理解**：就像一个人的性格或习惯，决定了在什么情况下会做什么选择。

#### 轨迹与回报

**定义 2.2.7** (轨迹（Trajectory）). 一个轨迹（或回合，*Episode*）是从初始状态到终止状态的完整交互序列：

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots)$$

其中 $s_0$ 是初始状态，智能体根据策略选择动作，环境根据转移函数和奖励函数给出新状态和奖励。

**定义 2.2.8** (回报（Return）). 从时刻 $t$ 开始的累计折扣奖励称为回报：

$$U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

- **理性理解**：回报是未来所有奖励的加权和，权重随时间指数衰减。
- **感性理解**：就像投资回报，近期的收益比远期的收益更有价值。

#### Rollout（或轨迹采样）

- **定义**：从当前状态开始，按照某个策略 $\pi$ 进行交互，生成一段轨迹的过程。
- **目的**：评估策略在当前状态下的表现，或收集训练数据。
- **示例**：在蒙特卡洛树搜索中，从当前棋局开始，随机走棋直到终局，得到胜负结果。

## 2.3　价值函数：评估策略的优劣

### 2.3.1　价值函数的定义与意义

由于未来的奖励具有随机性（来自策略的随机性和环境的随机性），我们关心的是期望回报。

#### 状态价值函数 $V_\pi(s)$

**定义 2.3.1** (状态价值函数). 状态价值函数 $V_\pi(s)$ 表示从状态 $s$ 开始，遵循策略 $\pi$ 所能获得的期望回报：

$$V_\pi(s) = \mathbb{E}_\pi[U_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s\right]$$

- **理性理解**：$V_\pi(s)$ 量化了状态 $s$ 的"好坏"程度。

- **感性理解**：就像评估一个棋局的"优势程度"，数值越大表示局面越好。
- **性质**：$V_\pi(s)$ 只依赖于状态 $s$ 和策略 $\pi$，不依赖于具体采取的动作。

**动作价值函数 $Q_\pi(s,a)$**

**定义 2.3.2** (动作价值函数（Q 函数）). 动作价值函数 $Q_\pi(s,a)$ 表示在状态 $s$ 下执行动作 $a$，然后遵循策略 $\pi$ 所能获得的期望回报：

$$Q_\pi(s,a) = \mathbb{E}_\pi[U_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s, A_t = a\right]$$

- **理性理解**：$Q_\pi(s,a)$ 量化了在状态 $s$ 下选择动作 $a$ 的"好坏"程度。
- **感性理解**：就像评估在某个棋局下走某一步的"优劣程度"。
- **关键区别**：$Q_\pi(s,a)$ 额外考虑了当前选择的动作 $a$。

### 2.3.2　价值函数的关系

状态价值函数和动作价值函数之间存在密切关系：

**从 $Q_\pi$ 到 $V_\pi$**

状态价值是该状态下所有可能动作的期望价值：

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \cdot Q_\pi(s,a)$$

- **解释**：在状态 $s$ 下，按照策略 $\pi$ 随机选择动作，$V_\pi(s)$ 是 $Q_\pi(s,a)$ 的加权平均。
- **示例**：如果策略 $\pi$ 在状态 $s$ 下以 0.7 概率选择动作 $a_1$（$Q = 10$），以 0.3 概率选择动作 $a_2$（$Q = 5$），则 $V_\pi(s) = 0.7 \times 10 + 0.3 \times 5 = 8.5$。

**从 $V_\pi$ 到 $Q_\pi$**

动作价值可以分解为立即奖励加上折扣后的下一个状态价值的期望：

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \cdot V_\pi(s')$$

- **解释**：执行动作 $a$ 后，获得立即奖励 $R(s,a)$，然后环境转移到状态 $s'$，从 $s'$ 开始遵循策略 $\pi$ 的期望回报是 $V_\pi(s')$。
- **示例**：在状态 $s$ 执行动作 $a$，有 0.8 概率转移到 $s_1$（$V = 20$），0.2 概率转移到 $s_2$（$V = 5$），立即奖励为 2，$\gamma = 0.9$，则 $Q_\pi(s,a) = 2 + 0.9 \times (0.8 \times 20 + 0.2 \times 5) = 2 + 0.9 \times 17 = 17.3$。

### 2.3.3　最优价值函数

强化学习的最终目标是找到最优策略 $\pi^*$，使得从任何状态出发都能获得最大期望回报。

#### 最优状态价值函数

**定义 2.3.3** (最优状态价值函数). 最优状态价值函数 $V^*(s)$ 是所有可能策略中能获得的最大状态价值：

$$V^*(s) = \max_{\pi} V_\pi(s), \quad \forall s \in \mathcal{S}$$

- **理性理解：** $V^*(s)$ 是从状态 $s$ 出发，采用最优策略所能获得的最大期望回报。
- **感性理解：** 就像”棋神”在当前局面下能获得的最佳结果。

#### 最优动作价值函数

**定义 2.3.4** (最优动作价值函数). 最优动作价值函数 $Q^*(s,a)$ 是所有可能策略中能获得的最大动作价值：

$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

- **理性理解：** $Q^*(s,a)$ 是在状态 $s$ 下执行动作 $a$，然后采用最优策略所能获得的最大期望回报。
- **感性理解：** 就像知道在某个局面下走某一步，后续采用最佳着法能获得的最好结果。

#### 最优价值函数的关系

最优状态价值和最优动作价值之间也存在类似关系：

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \cdot V^*(s')$$

将两式结合，得到：

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \cdot V^*(s') \right\}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \cdot \max_{a' \in \mathcal{A}} Q^*(s',a')$$

## 2.4　贝尔曼方程：动态规划的核心

### 2.4.1　贝尔曼期望方程

贝尔曼期望方程（Bellman Expectation Equation）建立了当前价值与未来价值之间的关系。

**状态价值函数的贝尔曼方程**

**定理 2.4.1** (状态价值函数的贝尔曼方程). *对于任意策略 $\pi$ 和状态 $s$，有：*

$$V_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) \left[ R(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) \cdot V_\pi(s') \right]$$

**推导过程：**

$$\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[U_t \mid S_t = s] \\
&= \mathbb{E}_\pi \left[ R_t + \gamma U_{t+1} \mid S_t = s \right] \\
&= \sum_{a\in\mathcal{A}} \pi(a \mid s) \mathbb{E}_\pi \left[ R_t + \gamma U_{t+1} \mid S_t = s, A_t = a \right] \\
&= \sum_{a\in\mathcal{A}} \pi(a \mid s) \left[ R(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) \cdot \mathbb{E}_\pi[U_{t+1} \mid S_{t+1} = s'] \right] \\
&= \sum_{a\in\mathcal{A}} \pi(a \mid s) \left[ R(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) \cdot V_\pi(s') \right]
\end{aligned}$$

**直观理解：**当前状态的价值 = 当前动作的期望立即奖励 + 折扣后的下一状态价值的期望。

**动作价值函数的贝尔曼方程**

**定理 2.4.2** (动作价值函数的贝尔曼方程). *对于任意策略 $\pi$、状态 $s$ 和动作 $a$，有：*

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) \cdot \sum_{a'\in\mathcal{A}} \pi(a' \mid s') \cdot Q_\pi(s',a')$$

### 2.4.2　贝尔曼最优方程

贝尔曼最优方程（Bellman Optimality Equation）描述了最优价值函数必须满足的条件。

**状态价值函数的最优贝尔曼方程**

**定理 2.4.3** (状态价值函数的最优贝尔曼方程). *最优状态价值函数满足：*

$$V^*(s) = \max_{a\in\mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) \cdot V^*(s') \right\}$$

**直观理解：**最优状态价值 = 所有可能动作中，能获得最大” 立即奖励 + 折扣后最优价值” 的那个动作对应的值。

#### 动作价值函数的最优贝尔曼方程

**定理 2.4.4** (动作价值函数的最优贝尔曼方程). *最优动作价值函数满足：*

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) \cdot \max_{a' \in \mathcal{A}} Q^*(s',a')$$

**直观理解：**最优动作价值 = 立即奖励 + 折扣后下一状态的最优动作价值的最大值。

### 2.4.3　贝尔曼方程的矩阵形式

对于有限状态空间，贝尔曼方程可以写成矩阵形式，便于理论分析和计算。

#### 状态价值函数的矩阵形式

设状态空间 $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$，定义：

- 状态价值向量：$\mathbf{V}_\pi = [V_\pi(s_1), V_\pi(s_2), \ldots, V_\pi(s_N)]^\top$
- 立即奖励向量：$\mathbf{R}_\pi = [R_\pi(s_1), R_\pi(s_2), \ldots, R_\pi(s_N)]^\top$，其中 $R_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) R(s,a)$
- 状态转移矩阵：$\mathbf{P}_\pi$，其中 $[\mathbf{P}_\pi]_{ij} = \sum_{a \in \mathcal{A}} \pi(a \mid s_i) p(s_j \mid s_i, a)$

则贝尔曼期望方程可写为：

$$\mathbf{V}_\pi = \mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{V}_\pi$$

#### 解析解与迭代解

1. **解析解：**理论上可以直接求解：

$$\mathbf{V}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{R}_\pi$$

   但矩阵求逆的复杂度为 $O(N^3)$，当状态数 $N$ 很大时计算不可行。

2. **迭代解：**实际中常用迭代方法：

$$\mathbf{V}_\pi^{(k+1)} = \mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{V}_\pi^{(k)}$$

   当 $k \to \infty$ 时，$\mathbf{V}_\pi^{(k)} \to \mathbf{V}_\pi$。

## 2.5　强化学习算法分类

强化学习算法可以从多个维度进行分类，理解这些分类有助于我们选择合适的算法解决特定问题。

### 2.5.1　有模型 vs 无模型

表 2.3: 有模型方法与无模型方法的对比

| 有模型方法（Model-Based） | 无模型方法（Model-Free） |
| --- | --- |
| **核心思想**：拥有环境模型（$P$ 和 $R$），可以通过规划（Planning）求解最优策略 | **核心思想**：不依赖环境模型，通过与环境的直接交互学习最优策略 |
| **环境知识**：需要知道状态转移概率 $p(s' \mid s, a)$ 和奖励函数 $R(s, a)$ | **环境知识**：不需要知道 $P$ 和 $R$，只需与环境交互获得样本 $(s, a, r, s')$ |
| **典型算法**：<br>• 值迭代（Value Iteration）<br>• 策略迭代（Policy Iteration）<br>• 蒙特卡洛树搜索（MCTS） | **典型算法**：<br>• Q-learning<br>• SARSA<br>• DQN<br>• 策略梯度方法 |
| **优点**：<br>• 样本效率高（可用模型生成虚拟数据）<br>• 可进行规划，无需实际交互 | **优点**：<br>• 更通用，不依赖环境模型<br>• 适用于复杂、未知环境 |
| **缺点**：<br>• 需要准确的环境模型<br>• 模型误差会累积影响策略 | **缺点**：<br>• 样本效率较低<br>• 需要大量交互数据 |

**有模型方法的进一步分类**

1. **给定环境模型**：环境模型已知且准确
   - **示例**：棋类游戏（规则明确）
   - **应用**：AlphaGo 中的蒙特卡洛树搜索
2. **学习环境模型**：从交互数据中学习环境模型
   - **方法**：先用数据学习 $p(s' \mid s, a)$ 和 $R(s, a)$ 的估计
   - **算法**：Dyna 框架（Sutton, 1991）
   - **流程**：
     (a) 用真实交互数据更新模型
     (b) 用模型生成虚拟数据
     (c) 用虚拟数据更新价值函数和策略

### 2.5.2　基于价值 vs 基于策略

表 2.4: 基于价值方法与基于策略方法的对比

| 基于价值方法（Value-Based） | 基于策略方法（Policy-Based） |
| --- | --- |
| **核心思想：**学习价值函数（$V$ 或 $Q$），通过价值函数导出策略 | **核心思想：**直接学习策略函数 $\pi(a \mid s;\theta)$，优化策略参数 $\theta$ |
| **策略表示：**通常是确定性的，$\pi(s) = \arg\max_a Q(s,a)$ | **策略表示：**可以是确定性的或随机性的，直接参数化 |
| **典型算法：**<br>• Q-learning<br>• DQN<br>• SARSA | **典型算法：**<br>• REINFORCE<br>• 策略梯度<br>• TRPO/PPO |
| **优点：**<br>• 通常更稳定，收敛性较好<br>• 样本效率相对较高 | **优点：**<br>• 能处理连续动作空间<br>• 能学习随机策略<br>• 收敛到局部最优 |
| **缺点：**<br>• 难以处理连续动作空间<br>• 通常得到确定性策略 | **缺点：**<br>• 方差大，训练不稳定<br>• 样本效率较低 |

#### 基于价值方法的策略推导

在基于价值的方法中，策略通常从价值函数中推导出来：

1. **贪婪策略：**总是选择价值最高的动作

$$\pi(s) = \arg\max_{a\in\mathcal{A}} Q(s,a)$$

2. **$\epsilon$-贪婪策略：**以 $1-\epsilon$ 的概率选择最优动作，以 $\epsilon$ 的概率随机选择动作

$$\pi(a \mid s) = \begin{cases} 1-\epsilon+\frac{\epsilon}{|\mathcal{A}|} & \text{if } a = \arg\max_{a'} Q(s,a') \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$

3. **Boltzmann 探索（Softmax）：**按价值大小分配选择概率

$$\pi(a \mid s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{a'\in\mathcal{A}} \exp(Q(s,a')/\tau)}$$

其中 $\tau > 0$ 是温度参数，控制探索程度。

#### 基于策略方法的目标函数

基于策略的方法直接优化策略参数 $\theta$ 以最大化期望回报：

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[R(\tau)] = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=0}^{T} \gamma^t r_t \right]$$

其中 $p_\theta(\tau)$ 是由策略 $\pi_\theta$ 生成轨迹 $\tau$ 的概率。

使用策略梯度定理，可以得到梯度：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t \right]$$

其中 $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ 是从时刻 $t$ 开始的累计回报。

### 2.5.3　蒙特卡洛 vs 时序差分

表 2.5: 蒙特卡洛方法与时序差分方法的对比

| **蒙特卡洛方法（Monte Carlo, MC）** | **时序差分方法（Temporal Difference, TD）** |
|---|---|
| **更新时机**：必须等到回合结束才能更新 | **更新时机**：每一步都可以立即更新 |
| **目标值**：使用实际回报 $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ | **目标值**：使用 TD 目标 $r_t + \gamma V(s_{t+1})$ |
| **偏差-方差**：无偏但高方差 | **偏差-方差**：有偏但低方差 |
| **更新公式（状态价值）**：$V(s_t) \leftarrow V(s_t) + \alpha[G_t - V(s_t)]$ | **更新公式（状态价值）**：$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)]$ |
| **收敛性**：保证收敛到真实价值 | **收敛性**：在一定条件下收敛 |
| **适用场景**：<br>• 回合制任务<br>• 需要无偏估计 | **适用场景**：<br>• 连续任务<br>• 在线学习<br>• 需要快速更新 |

#### 蒙特卡洛方法的偏差与方差分析

- **无偏性**：蒙特卡洛估计是真实期望的无偏估计，因为 $G_t$ 是 $V(s_t)$ 的无偏估计。

$$\mathbb{E}[G_t \mid S_t = s] = V_\pi(s)$$

- **高方差**：$G_t$ 依赖于整个后续轨迹，随机性积累导致高方差。

$$\mathrm{Var}[G_t] = \mathrm{Var}\left[ \sum_{k=t}^{T} \gamma^{k-t} R_k \right] = \sum_{k=t}^{T} \gamma^{2(k-t)} \mathrm{Var}[R_k] + \text{交叉项}$$

**时序差分方法的偏差与方差分析**

- **有偏性**：TD 目标 $r_t + \gamma V(s_{t+1})$ 是 $V(s_t)$ 的有偏估计，因为 $V(s_{t+1})$ 本身是估计值。

$$\mathbb{E}[r_t + \gamma V(s_{t+1}) \mid S_t = s] \neq V_\pi(s) \quad \text{除非} V(s_{t+1}) = V_\pi(s_{t+1})$$

- **低方差**：TD 目标只依赖于单步奖励和下一个状态的价值估计，随机性较少。

$$\mathrm{Var}[r_t + \gamma V(s_{t+1})] \approx \mathrm{Var}[r_t] + \gamma^2 \mathrm{Var}[V(s_{t+1})]$$

### 2.5.4　在线策略 vs 离线策略

表 2.6: 在线策略方法与离线策略方法的对比

| **在线策略方法（On-Policy）** | **离线策略方法（Off-Policy）** |
|---|---|
| **行为策略与目标策略**：相同 | **行为策略与目标策略**：可以不同 |
| **数据收集**：使用当前策略收集数据 | **数据收集**：可以使用任意策略收集数据 |
| **数据利用**：只能使用当前策略产生的数据 | **数据利用**：可以使用历史数据、专家数据等 |
| **探索方式**：策略本身必须有一定的探索性 | **探索方式**：行为策略负责探索，目标策略可以更贪婪 |
| **典型算法**：<br>• SARSA<br>• REINFORCE<br>• TRPO<br>• PPO | **典型算法**：<br>• Q-learning<br>• DQN<br>• DDPG<br>• SAC |
| **优点**：<br>• 理论分析相对简单<br>• 通常更稳定 | **优点**：<br>• 数据重用，样本效率高<br>• 可以学习多个策略<br>• 支持从示范中学习 |
| **缺点**：<br>• 样本效率低<br>• 探索与利用需要平衡 | **缺点**：<br>• 理论分析复杂<br>• 可能存在收敛问题 |

**SARSA：在线策略 TD 控制算法**

SARSA（State-Action-Reward-State-Action）是经典的在线策略 TD 控制算法：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

- **名称来源**：使用五元组 $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$
- **在线策略**：$A_t$ 和 $A_{t+1}$ 都来自当前策略（通常为 $\epsilon$-贪婪策略）
- **更新目标**：$R_t + \gamma Q(S_{t+1}, A_{t+1})$，其中 $A_{t+1}$ 是实际采取的动作

#### Q-learning：离线策略 TD 控制算法

Q-learning 是最著名的离线策略 TD 控制算法：

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$

- **离线策略**：$A_t$ 来自行为策略（如 $\epsilon$-贪婪），但更新时使用目标策略（贪婪策略）
- **更新目标**：$R_t + \gamma \max_{a'} Q(S_{t+1}, a')$，使用最优动作的价值
- **收敛性**：在一定条件下，Q-learning 能收敛到最优 Q 函数

#### 经验回放（Experience Replay）

经验回放是深度 Q 网络（DQN）中的关键技术，属于离线策略方法：

1. **存储经验**：将经验元组 $(s_t, a_t, r_t, s_{t+1})$ 存入回放缓冲区（Replay Buffer）
2. **随机采样**：训练时从缓冲区中随机采样小批量经验
3. **打破相关性**：随机采样打破了经验之间的时间相关性，提高稳定性
4. **数据重用**：同一份经验可以多次用于训练，提高样本效率

## 2.6　Actor-Critic 方法：价值与策略的结合

### 2.6.1　Actor-Critic 框架

Actor-Critic 方法结合了基于价值方法和基于策略方法的优点：

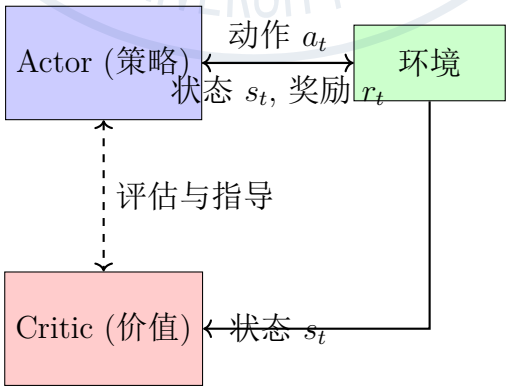



图 2.3: Actor-Critic 框架示意图

- **Actor（执行者）**：负责选择动作，即策略 $\pi_\theta(a \mid s)$
- **Critic（评论者）**：负责评估状态或状态-动作对的价值，即价值函数 $V_\phi(s)$ 或 $Q_\phi(s, a)$

- **协同工作：**
  1. Actor 根据当前策略选择动作
  2. Critic 评估 Actor 选择的动作的好坏
  3. Critic 的评估指导 Actor 更新策略

### 2.6.2　优势函数（Advantage Function）

优势函数是 Actor-Critic 方法中的核心概念，用于衡量一个动作相对于平均水平的优势：

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

- **直观理解：**在状态 $s$ 下选择动作 $a$ 比随机选择动作平均好多少
- **性质：**
  - 如果 $A_\pi(s,a) > 0$，说明动作 $a$ 优于平均水平
  - 如果 $A_\pi(s,a) < 0$，说明动作 $a$ 劣于平均水平
  - $\mathbb{E}_{a\sim\pi(\cdot|s)}[A_\pi(s,a)] = 0$（所有动作的优势平均为 0）

### 2.6.3　策略梯度与 Actor-Critic

结合优势函数的策略梯度定理：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot A_\pi(s_t, a_t)\right]$$

- **REINFORCE 算法：**使用蒙特卡洛回报 $G_t$ 作为优势估计

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot (G_t - b(s_t))$$

  其中 $b(s_t)$ 是基线（baseline），通常取 $V(s_t)$ 以减少方差。
- **Actor-Critic 算法：**使用 Critic 估计的优势函数

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot A_\phi(s_t, a_t)$$

  其中 $A_\phi(s_t, a_t)$ 是 Critic 估计的优势函数。

### 2.6.4　现代 Actor-Critic 算法

**优势 Actor-Critic（A2C/A3C）**

- **A2C（同步优势 Actor-Critic）：**同步更新多个 Worker
- **A3C（异步优势 Actor-Critic）：**异步更新多个 Worker，提高训练速度

- **优势估计**：使用 n 步 TD 误差估计优势

$$A(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}) - V(s_t)$$

**近端策略优化（PPO）**

PPO 通过限制策略更新的幅度来提高训练稳定性：

$$J^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

其中 $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ 是概率比，$\hat{A}_t$ 是优势估计，$\epsilon$ 是超参数。

**深度确定性策略梯度（DDPG）**

DDPG 结合了 DQN 和确定性策略梯度，适用于连续动作空间：

- **Actor**：确定性策略 $\mu_\theta(s)$，输出连续动作
- **Critic**：动作价值函数 $Q_\phi(s, a)$
- **关键技术**：
  1. 经验回放
  2. 目标网络（Target Network）
  3. 确定性策略梯度定理：

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s\sim\mathcal{D}} \left[ \nabla_\theta \mu_\theta(s) \nabla_a Q_\phi(s, a)\big|_{a=\mu_\theta(s)} \right]$$

## 2.7　总结与展望

### 2.7.1　强化学习算法分类总结

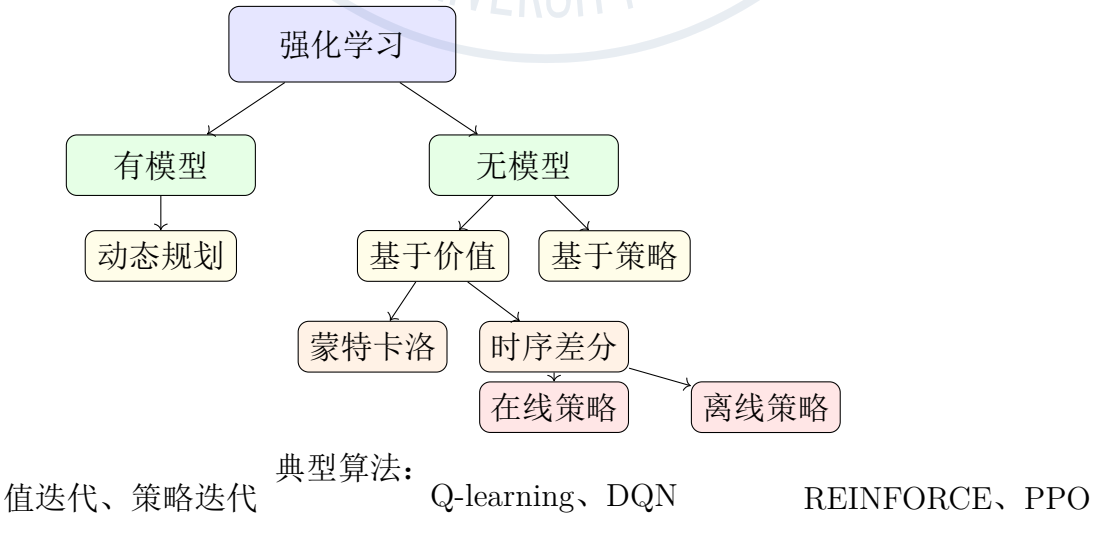



图 2.4: 强化学习算法分类树

### 2.7.2　强化学习的挑战与前沿

#### 主要挑战

1. **样本效率**：强化学习通常需要大量交互数据
   - **解决方法**：经验回放、模型学习、模仿学习
2. **探索与利用的平衡**
   - **解决方法**：$\epsilon$-贪婪、UCB、Thompson 采样、好奇心驱动探索
3. **稳定性与收敛性**
   - **解决方法**：目标网络、软更新、信任域方法
4. **稀疏奖励问题**
   - **解决方法**：课程学习、分层强化学习、内在动机
5. **安全性**
   - **解决方法**：约束强化学习、安全探索

#### 前沿方向

1. **深度强化学习（Deep RL）**：将深度学习与强化学习结合
   - **DQN**：深度 Q 网络，开启深度强化学习时代
   - **AlphaGo**：结合蒙特卡洛树搜索与深度神经网络
   - **AlphaZero**：从零开始自我对弈学习
2. **多智能体强化学习（Multi-Agent RL）**
   - **协作**：智能体协作完成共同任务
   - **竞争**：智能体在竞争环境中学习
   - **混合**：既有协作又有竞争
3. **元强化学习（Meta-RL）**
   - **目标**：学会如何快速学习新任务
   - **应用**：Few-shot 学习、快速适应
4. **离线强化学习（Offline RL）**
   - **特点**：从固定的数据集学习，不与环境交互
   - **应用**：医疗、自动驾驶等高风险领域
5. **强化学习与大型语言模型结合**
   - **RLHF**：基于人类反馈的强化学习
   - **应用**：ChatGPT 等对话系统的对齐

强化学习是一门理论与实践并重的学科，需要在理解数学原理的基础上，通过实际编码和调参来积累经验。随着计算能力的提升和算法的进步，强化学习正在越来越多的领域展现出强大的潜力。

# 第三章　深度强化学习之价值学习

## 3.1　价值学习：从评估到决策

### 3.1.1　价值学习的核心思想

在强化学习中，价值学习（Value-Based Learning）是一类重要的方法。与直接学习策略（策略学习）不同，价值学习的核心思想是学习一个**价值函数**（Value Function），这个函数用于评估在某个状态（或状态-动作对）下的" 好坏" 程度。

**定义 3.1.1** (最优动作价值函数). 最优动作价值函数 $Q^*(s,a)$ 定义为在所有可能的策略中，从状态 $s$ 开始、执行动作 $a$ 后能获得的最大期望回报：

$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

其中 $Q_\pi(s,a)$ 是在策略 $\pi$ 下的动作价值函数。

**关键洞察**：一旦我们知道了最优动作价值函数 $Q^*(s,a)$，最优策略就可以直接得到：在每个状态 $s$ 下，选择使 $Q^*(s,a)$ 最大的动作：

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a)$$

**价值学习的优点：**

- **直观性**：Q 值直接反映了动作的" 好坏"。
- **稳定性**：相比策略学习方法，价值学习通常更稳定。
- **可解释性**：Q 值表或 Q 网络提供了对决策过程的直观理解。

**价值学习的挑战：**

- **维度灾难**：当状态空间或动作空间很大时，存储所有状态-动作对的 Q 值变得不可行。
- **连续空间**：对于连续状态或动作空间，无法枚举所有可能情况。
- **泛化能力**：需要从有限的经验中泛化到未见过的状态。

# 3.2　Q-Learning：经典的价值学习算法

## 3.2.1　算法原理

Q-Learning 是强化学习中最经典、最重要的算法之一。它是一种**离线策略**（off-policy）的**时序差分**（Temporal Difference, TD）学习方法。

**定义 3.2.1** (Q-Learning 更新规则). *Q-Learning 通过以下规则更新 $Q$ 值估计：*

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

*其中：*

- $\alpha \in (0,1]$ *是学习率（步长）*
- $\gamma \in [0,1]$ *是折扣因子*
- $r$ *是执行动作 $a$ 后获得的即时奖励*
- $s'$ *是转移后的新状态*
- $\max_{a'} Q(s',a')$ *是在新状态 $s'$ 下所有可能动作的最大 $Q$ 值*

**直观理解：**

- $Q(s,a)$ 是当前对在状态 $s$ 下执行动作 $a$ 的价值的估计。
- $r + \gamma \max_{a'} Q(s',a')$ 是 TD 目标，包含两部分：
  1. 即时奖励 $r$
  2. 折扣后的未来最大可能价值 $\gamma \max_{a'} Q(s',a')$
- $r + \gamma \max_{a'} Q(s',a') - Q(s,a)$ 是 TD 误差，表示当前估计与目标之间的差距。
- 学习率 $\alpha$ 控制更新步长：$\alpha$ 越大，更新越快，但可能不稳定；$\alpha$ 越小，更新越慢，但更稳定。

## 3.2.2　表格 Q-Learning

在状态空间和动作空间都很小的情况下，我们可以使用表格形式存储所有状态-动作对的 Q 值。

表 3.1: 表格 Q-Learning 中的 Q 值表示例

| 状态 | 动作 1 | 动作 2 | 动作 3 | 动作 4 |
|---|---|---|---|---|
| 状态 0 | $Q(0,1)$ | $Q(0,2)$ | $Q(0,3)$ | $Q(0,4)$ |
| 状态 1 | $Q(1,1)$ | $Q(1,2)$ | $Q(1,3)$ | $Q(1,4)$ |
| 状态 2 | $Q(2,1)$ | $Q(2,2)$ | $Q(2,3)$ | $Q(2,4)$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 状态 m | $Q(m,1)$ | $Q(m,2)$ | $Q(m,3)$ | $Q(m,4)$ |

**Algorithm 1** 表格 Q-Learning 算法

**Require:** 学习率 $\alpha$，折扣因子 $\gamma$，探索率 $\epsilon$

**Ensure:** 最优动作价值函数 $Q^*(s,a)$

初始化 $Q(s,a)$ 为任意值（通常为 0 或随机小值）

**for** 每个回合（episode） **do**

　初始化状态 $s$

　**while** $s$ 不是终止状态 **do**

　　根据 $\epsilon$-贪婪策略从 $s$ 选择动作 $a$：

$$a = \begin{cases} \text{随机动作} & \text{以概率}\epsilon \\ \arg\max_{a'} Q(s,a') & \text{以概率}1-\epsilon \end{cases}$$

　　执行动作 $a$，观察奖励 $r$ 和新状态 $s'$

　　更新 Q 值：$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

　　$s \leftarrow s'$

　**end while**

**end for**

### 3.2.3　探索与利用的平衡：$\epsilon$-贪婪策略

在 Q-Learning 中，$\epsilon$-贪婪策略用于平衡探索（尝试新动作）和利用（选择已知最佳动作）：

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & \text{如果}a = \arg\max_{a'} Q(s,a') \\ \frac{\epsilon}{|\mathcal{A}|} & \text{其他动作} \end{cases}$$

- $\epsilon \in [0,1]$ 是探索率
- 以概率 $1-\epsilon$ 选择当前认为最好的动作（利用）
- 以概率 $\epsilon$ 随机选择动作（探索）
- 通常随着训练进行，$\epsilon$ 会逐渐减小（从高探索到高利用）

### 3.2.4　Q-Learning 与 SARSA 的比较

表 3.2: Q-Learning 与 SARSA 的比较

| **Q-Learning（离线策略）** | **SARSA（在线策略）** |
|---|---|
| **更新公式：** $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ | **更新公式：** $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$ |
| **目标策略：** 贪婪策略 $\pi(s) = \arg\max_a Q(s,a)$ | **目标策略：** 与行为策略相同（通常为 $\epsilon$-贪婪） |
| **行为策略：** $\epsilon$-贪婪策略 | **行为策略：** $\epsilon$-贪婪策略 |
| **TD 目标：** $r + \gamma \max_{a'} Q(s',a')$ | **TD 目标：** $r + \gamma Q(s',a')$ |
| **学习对象：** 最优 Q 函数 $Q^*$ | **学习对象：** 当前策略的 Q 函数 $Q_\pi$ |
| **探索影响：** 行为策略探索不影响目标策略 | **探索影响：** 探索直接影响学习策略 |
| **收敛性：** 收敛到最优策略（在适当条件下） | **收敛性：** 收敛到 $\epsilon$-贪婪策略的最优策略 |

**关键区别：**

- Q-Learning 在计算 TD 目标时使用 $\max_{a'} Q(s',a')$，假设下一个动作是最优的。
- SARSA 在计算 TD 目标时使用实际采取的下一个动作 $Q(s',a')$，更” 谨慎”。
- 这导致 Q-Learning 更” 乐观”，SARSA 更” 保守”。

## 3.3　深度 Q 网络：当 Q-Learning 遇见深度学习

### 3.3.1　表格方法的局限性

表格 Q-Learning 虽然直观，但面临严重限制：

1. **维度灾难：** 状态空间随维度指数增长。例如：
   - 围棋：$10^{170}$ 个状态，无法存储
   - Atari 游戏：$256^{84\times84}$ 个可能状态，天文数字
   - 连续状态空间：无限多个状态
2. **泛化能力差：** 每个状态-动作对独立学习，无法泛化到相似状态。
3. **内存需求巨大：** 存储所有 Q 值需要大量内存。
4. **学习效率低：** 每个状态需要单独学习，无法利用状态间的相似性。

### 3.3.2　深度 Q 网络的基本思想

深度 Q 网络（Deep Q-Network, DQN）的核心思想是用神经网络来近似 Q 函数，从而解决维度灾难问题。

**定义 3.3.1** (深度 Q 网络). 深度 $Q$ 网络是一个参数化的函数 $Q(s,a;\theta)$，其中 $\theta$ 是神经网络的参数。对于给定的状态 $s$，网络输出所有可能动作的 $Q$ 值：

$$f(s;\theta) \approx [Q(s,a_1), Q(s,a_2), \ldots, Q(s,a_n)]$$

其中 $n$ 是动作空间的大小。

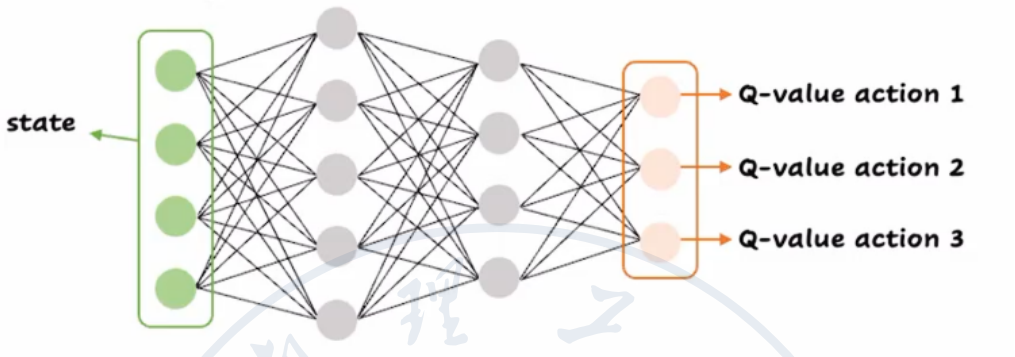



图 3.1: 深度 Q 网络结构示意图：输入状态，输出每个动作的 Q 值

**网络架构：**

- **输入**：状态表示（如图像像素、特征向量等）
- **隐藏层**：多个全连接层或卷积层，用于提取特征
- **输出层**：每个输出节点对应一个动作的 Q 值

### 3.3.3　DQN 的训练目标与损失函数

DQN 的训练目标是找到参数 $\theta$，使得 $Q(s,a;\theta)$ 近似最优 Q 函数 $Q^*(s,a)$。

**定义 3.3.2** (DQN 损失函数). 使用均方误差（*MSE*）作为损失函数：

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[(Q(s,a;\theta) - y)^2\right]$$

其中：

- $(s,a,r,s')$ 是从经验回放缓冲区 $\mathcal{D}$ 中采样的转移
- $y$ 是 *TD* 目标：$y = r + \gamma \max_{a'} Q(s',a';\theta^-)$
- $\theta^-$ 是目标网络的参数（与在线网络 $\theta$ 不同）

**梯度计算：**

$$\nabla_\theta L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[(Q(s,a;\theta) - y)\cdot\nabla_\theta Q(s,a;\theta)\right]$$

**参数更新：**

$$\theta \leftarrow \theta - \alpha\cdot\nabla_\theta L(\theta)$$

其中 $\alpha$ 是学习率。

### 3.3.4　DQN 的训练流程

**Algorithm 2** 深度 Q 网络（DQN）算法

**Require:** 经验回放缓冲区容量 $N$，目标网络更新频率 $C$，折扣因子 $\gamma$，探索率 $\epsilon$

初始化在线网络 $Q(\cdot;\theta)$ 的参数 $\theta$ 随机
初始化目标网络 $Q(\cdot;\theta^-)$ 的参数 $\theta^- \leftarrow \theta$
初始化经验回放缓冲区 $\mathcal{D}$ 为空，容量为 $N$
**for** 每个回合（episode） **do**
　初始化状态 $s_1$ 和预处理 $\phi_1 = \phi(s_1)$
　**for** $t = 1$ 到 $T$ **do**
　　以概率 $\epsilon$ 选择随机动作 $a_t$，否则 $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
　　执行动作 $a_t$，观察奖励 $r_t$ 和下一状态 $s_{t+1}$
　　预处理 $\phi_{t+1} = \phi(s_{t+1})$
　　存储转移 $(\phi_t, a_t, r_t, \phi_{t+1})$ 到 $\mathcal{D}$
　　从 $\mathcal{D}$ 中随机采样小批量转移 $(\phi_j, a_j, r_j, \phi_{j+1})$
　　计算 TD 目标：

$$y_j = \begin{cases} r_j & \text{如果}\phi_{j+1}\text{ 是终止状态} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta^-) & \text{否则} \end{cases}$$

　　计算损失：$L = \frac{1}{m}\sum_{j=1}^{m}(y_j - Q(\phi_j, a_j; \theta))^2$
　　使用梯度下降更新 $\theta$
　　每 $C$ 步更新目标网络：$\theta^- \leftarrow \theta$
　**end for**
**end for**

## 3.4　DQN 的核心技术

### 3.4.1　经验回放（Experience Replay）

经验回放是 DQN 成功的关键技术之一，解决了两个核心问题：

1. **数据效率**：每个转移可以被多次使用，提高数据利用率。
2. **相关性破坏**：随机采样打破了连续状态之间的相关性，使训练更稳定。

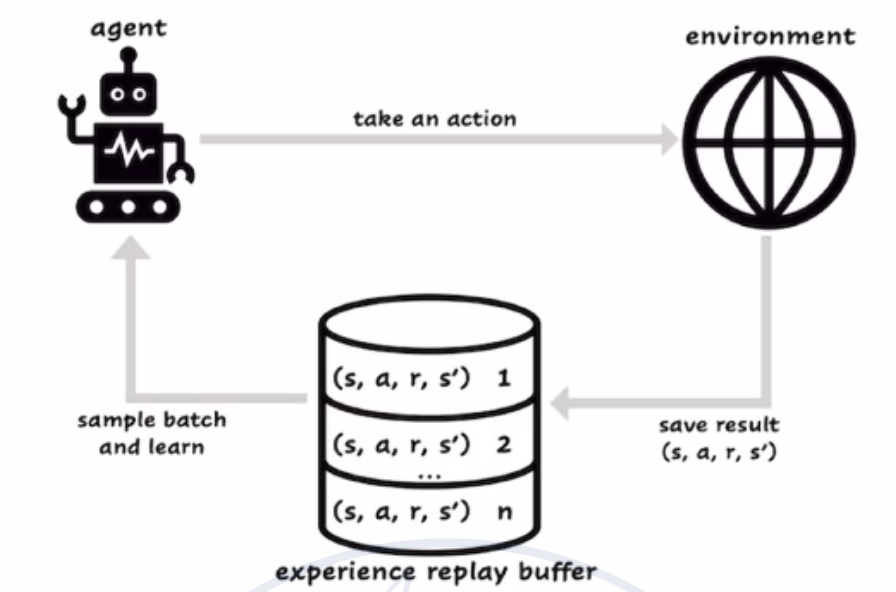



图 3.2: 经验回放机制：存储历史转移并随机采样用于训练

**定义 3.4.1** (经验回放缓冲区). *经验回放缓冲区 $\mathcal{D}$ 是一个固定大小的循环缓冲区，存储转移元组 $(s, a, r, s', done)$，其中：*

- $s$：当前状态
- $a$：执行的动作
- $r$：获得的奖励
- $s'$：转移后的新状态
- $done$：是否到达终止状态

**经验回放的工作原理：**

1. **收集**：智能体与环境交互，将每个转移存储到缓冲区。
2. **采样**：训练时随机从缓冲区采样小批量转移。
3. **学习**：使用采样到的转移计算损失并更新网络。

**经验回放的优势：**

- **打破相关性**：连续的状态高度相关，直接用于训练会导致梯度更新方向高度相关，训练不稳定。
- **数据重用**：每个转移可以被多次用于训练，提高数据效率。
- **平滑分布**：随机采样使数据分布更平稳，减少方差。

### 3.4.2　目标网络（Target Network）

目标网络是 DQN 的另一项关键技术，解决了训练中的不稳定性问题。

**定义 3.4.2** (目标网络). *目标网络 $Q(\cdot;\theta^-)$ 是与在线网络 $Q(\cdot;\theta)$ 结构相同的网络，但参数更新更慢：*

- **硬更新**：每 $C$ 步将在线网络的参数复制给目标网络：$\theta^- \leftarrow \theta$
- **软更新**：每次迭代按比例更新：$\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$，其中 $\tau \ll 1$

**目标网络的作用：**

1. **稳定训练**：TD 目标 $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ 在一段时间内固定，减少了目标的波动。
2. **避免发散**：防止 Q 值估计的”追逐尾巴”现象（目标随估计不断变化，导致训练发散）。
3. **缓解过估计**：目标网络使用旧参数，减少了最大化操作带来的过估计问题。

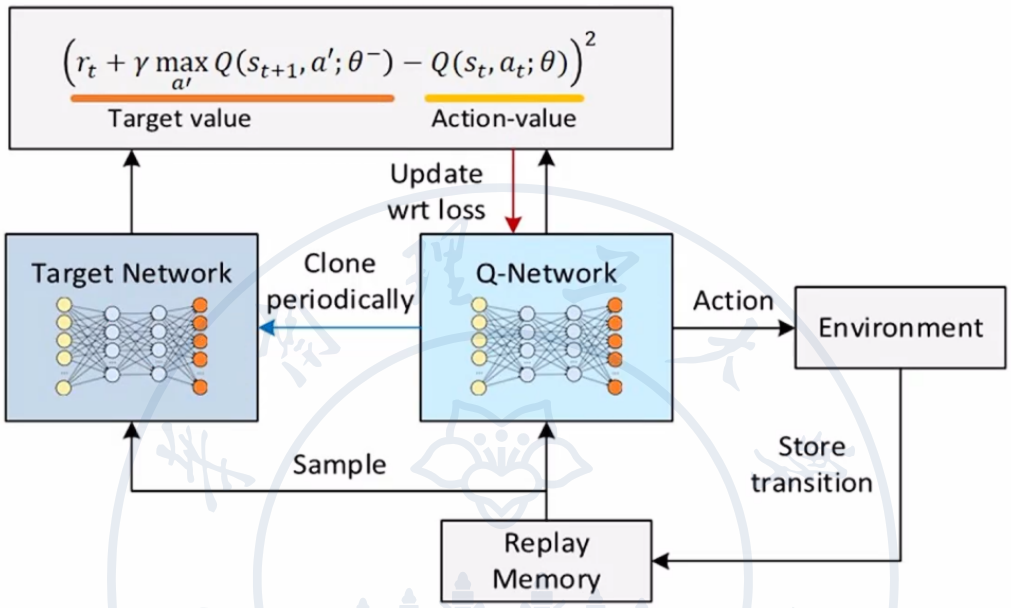



图 3.3: DQN 损失计算：使用目标网络计算 TD 目标

**为什么需要目标网络？**

考虑没有目标网络的情况：

$$y = r + \gamma \max_{a'} Q(s', a'; \theta)$$

损失函数：

$$L(\theta) = \mathbb{E}[(Q(s, a; \theta) - y)^2]$$

问题：目标 $y$ 依赖于正在优化的参数 $\theta$，导致：

1. 目标不断变化，训练不稳定。
2. Q 值可能发散（”追逐尾巴”问题）。

使用目标网络后：

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

目标网络参数 $\theta^-$ 更新较慢，提供了更稳定的学习目标。

# 3.5　DQN 的改进与扩展

## 3.5.1　优先经验回放（Prioritized Experience Replay）

标准经验回放均匀采样，但不同转移的重要性不同。优先经验回放根据 TD 误差的绝对值赋予不同采样优先级。

**定义 3.5.1** (采样优先级). *转移 $i$ 的采样概率为：*

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

*其中：*

- $p_i$ *是转移* $i$ *的优先级*
- $\alpha \in [0,1]$ *控制优先程度的强度（*$\alpha = 0$ *时退化为均匀采样）*

**优先级计算方法：**

1. **基于 TD 误差：** $p_i = |\delta_i| + \epsilon$，其中 $\delta_i$ 是转移 $i$ 的 TD 误差，$\epsilon$ 是小的正常数。
2. **基于排名：** $p_i = \frac{1}{\text{rank}(i)}$，其中 $\text{rank}(i)$ 是基于 $|\delta_i|$ 的排名。

**重要性采样权重：**由于非均匀采样引入偏差，需要重要性采样权重校正：

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$$

其中 $\beta \in [0,1]$ 控制校正程度，训练中从 $\beta_{\text{初始}}$ 线性增加到 1。

**损失函数：**

$$L(\theta) = \sum_{i=1}^{m} w_i \cdot (y_i - Q(s_i, a_i; \theta))^2$$

**为什么重要性采样有效？**

考虑两种学习方式：

1. **均匀采样：**学习率 $\alpha$，用样本 $(s_j, a_j, r_j, s_{j+1})$ 更新一次。
2. **优先采样：**学习率 $\alpha_j = w_j \cdot \alpha$，用重要样本更新多次。

虽然学习率减小了，但重要样本被更频繁地使用，相当于用更多计算量从重要样本中提取更多信息。

## 3.5.2　Double DQN

标准 DQN 存在过估计（overestimation）问题：max 操作倾向于选择被高估的动作。

**定理 3.5.1** (过估计定理). *设 $X_1, \ldots, X_n$ 是独立同分布的随机变量，均值为 $\mu$，方差为 $\sigma^2$。则：*

$$\mathbb{E}[\max(X_1, \ldots, X_n)] \geq \mu$$

*等号成立当且仅当 $n = 1$ 或 $\sigma^2 = 0$。*

**证明**：由 Jensen 不等式，max 是凸函数，所以 $\mathbb{E}[\max(X_i)] \geq \max(\mathbb{E}[X_i]) = \mu$。

在 DQN 中，Q 值估计包含噪声（来自函数近似、环境随机性等），max 操作会放大正误差，导致过估计。

**定义 3.5.2** (Double DQN). *Double DQN（DDQN）解耦动作选择和价值评估：*

$$y = r + \gamma Q\left(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-\right)$$

其中：

- 使用在线网络 $\theta$ 选择动作：$a^* = \arg\max_{a'} Q(s', a'; \theta)$
- 使用目标网络 $\theta^-$ 评估价值：$Q(s', a^*; \theta^-)$

表 3.3: 不同 Q-Learning 变种的比较

| 算法 | 动作选择 | 价值评估 | 过估计程度 |
|---|---|---|---|
| 原始 Q-Learning | DQN 网络 | DQN 网络 | 严重 |
| DQN+ 目标网络 | 目标网络 | 目标网络 | 中等 |
| Double DQN | DQN 网络 | 目标网络 | 轻微 |

**Double DQN 的优势：**

1. 减少过估计，提高价值估计的准确性。
2. 在某些任务上性能显著优于标准 DQN。
3. 实现简单，只需修改 TD 目标计算方式。

### 3.5.3 Dueling DQN

Dueling DQN 改进了网络结构，将 Q 值分解为状态价值 $V(s)$ 和优势函数 $A(s, a)$。

**定义 3.5.3** (Dueling 架构).

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha)$$

其中：

- $V(s; \theta, \beta)$：状态价值函数，评估状态 $s$ 的好坏
- $A(s, a; \theta, \alpha)$：优势函数，评估动作 $a$ 相对于平均水平的优势
- 减去的项确保优势函数的平均值为 0，解决可识别性问题

**可识别性问题**：如果直接定义 $Q(s, a) = V(s) + A(s, a)$，那么对于常数 $c$，有：

$$Q(s, a) = [V(s) + c] + [A(s, a) - c]$$

即 $V(s)$ 和 $A(s,a)$ 不唯一。通过减去优势函数的平均值解决这一问题。

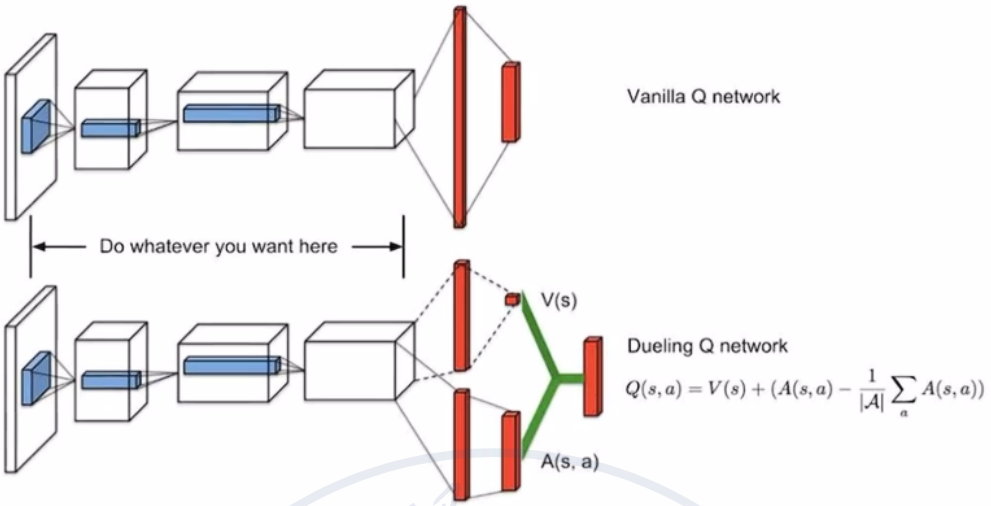



图 3.4: Dueling DQN 网络结构：将 Q 值分解为状态价值和优势函数

**Dueling DQN 的优势：**

1. **更好的泛化：**可以学习哪些状态有价值，而不需要了解每个动作的影响。
2. **更稳定的学习：**状态价值提供基线，减少方差。
3. **更好的策略评估：**在状态价值相似但动作价值不同的情况下表现更好。

### 3.5.4 Multi-Step DQN

标准 DQN 使用单步 TD 目标，只考虑一步奖励。Multi-Step DQN 考虑多步奖励，减少短视行为。

**定义 3.5.4** (n 步 TD 目标). *$n$ 步 TD 目标考虑未来 $n$ 步的奖励：*

$$y_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \max_{a'} Q(s_{t+n}, a'; \theta^-)$$

*特别地：*

- $n = 1$：*标准 DQN*，$y_t^{(1)} = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$
- $n = \infty$：*蒙特卡洛方法，考虑整个回合的回报*

**多步学习的权衡：**

- **偏差-方差权衡：**n 越大，偏差越小（使用更多真实奖励），方差越大（依赖更多随机步骤）。
- **更新延迟：**需要等待 n 步后才能更新，学习延迟增加。

**实现方式：**

1. 存储 n 步转移：$(s_t, a_t, r_t, \ldots, r_{t+n-1}, s_{t+n})$

2. 计算 n 步 TD 目标
3. 用标准 DQN 方式更新

### 3.5.5 Noisy DQN

传统 $\epsilon$-贪婪探索在动作空间添加噪声，Noisy DQN 在参数空间添加噪声，实现更高效的探索。

**定义 3.5.5** (Noisy DQN). *Noisy DQN 在网络的权重中添加可学习的噪声：*

$$\tilde{Q}(s,a,\xi;\mu,\sigma) = Q(s,a;\mu+\sigma\odot\xi)$$

其中：

- $\mu$ 和 $\sigma$ 是可学习的参数
- $\xi$ 是随机噪声，从标准正态分布采样：$\xi \sim \mathcal{N}(0,1)$
- $\odot$ 表示逐元素乘法

**噪声注入方式：**

1. **因子化高斯噪声**：减少参数数量，提高效率。
2. **独立高斯噪声**：每个参数独立添加噪声。

**训练过程：**

1. 每个回合开始时采样噪声 $\xi$，并在整个回合中固定。
2. 使用带噪声的网络选择动作：$a_t = \arg\max_a \tilde{Q}(s_t,a,\xi;\mu,\sigma)$
3. 收集经验并计算损失。
4. 通过损失函数同时优化 $\mu$ 和 $\sigma$。

表 3.4: $\epsilon$-贪婪与 Noisy DQN 的比较

| $\epsilon$-贪婪探索 | **Noisy DQN 探索** |
|---|---|
| **噪声位置**：动作空间 | **噪声位置**：参数空间 |
| **更新频率**：每个时间步独立决定 | **更新频率**：每个回合固定 |
| **探索连贯性**：低（独立随机） | **探索连贯性**：高（连贯的探索方向） |
| **探索与策略耦合**：解耦 | **探索与策略耦合**：强耦合（噪声是网络一部分） |
| **学习稳定性**：较低（频繁切换） | **学习稳定性**：较高（一致的方向） |
| **超参数**：需要手动调整 $\epsilon$ | **超参数**：噪声参数自动学习 |

### 3.5.6 Distributional DQN

传统 DQN 学习 Q 值的期望，Distributional DQN 学习 Q 值的完整分布。

**定义 3.5.6** (价值分布). 价值分布 $Z(s,a)$ 是一个随机变量，表示从状态 $s$ 执行动作 $a$ 后获得的随机回报。传统 $Q$ 值是价值分布的期望：

$$Q(s,a) = \mathbb{E}[Z(s,a)]$$

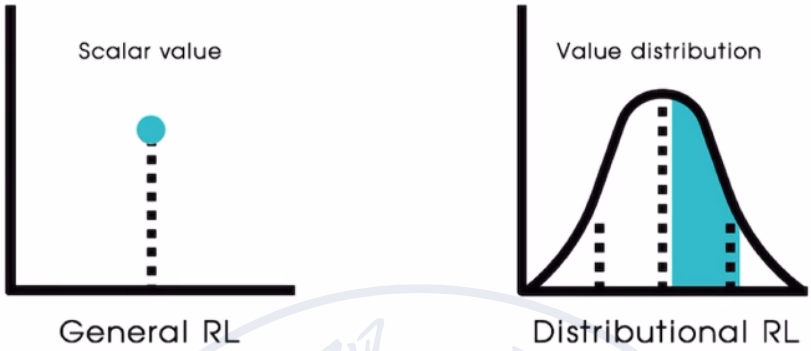



图 3.5: Distributional DQN：学习价值分布而非期望值

**C51 算法**

C51（Categorical 51）是 Distributional DQN 的一种实现，使用 51 个类别的分类分布表示价值分布。

**核心思想：**

1. 将价值范围 $[V_{\min}, V_{\max}]$ 离散化为 $N = 51$ 个等间距的支持点（atoms）：$z_i = V_{\min} + i \cdot \Delta z$，其中 $\Delta z = \frac{V_{\max} - V_{\min}}{N-1}$。
2. 对于每个状态-动作对，网络输出这 51 个点的概率：$p_i(s,a) = \mathbb{P}(Z(s,a) = z_i)$。
3. 价值分布的期望为：$Q(s,a) = \sum_{i=1}^{N} p_i(s,a) \cdot z_i$。

**投影贝尔曼更新：**对于转移 $(s,a,r,s')$，目标分布为：

$$\mathcal{T} z_j = r + \gamma z_j$$

将 $\mathcal{T} z_j$ 投影到原始支持点上，得到目标概率分布。

**损失函数：**使用交叉熵损失：

$$L(\theta) = -\sum_{i=1}^{N} \hat{p}_i \log p_i(s,a;\theta)$$

其中 $\hat{p}_i$ 是目标分布在第 $i$ 个 atom 上的概率。

**C51 的优势：**

1. **更丰富的学习信号：**分布提供比标量更多的信息。
2. **更好的风险意识：**可以区分高方差和低方差情况。
3. **更稳定的训练：**分布学习通常更稳定。

**QR-DQN 和 IQN**

除了 C51，还有其他分布 RL 算法：

- **QR-DQN（Quantile Regression DQN）**：学习价值分布的分位数。
- **IQN（Implicit Quantile Networks）**：隐式学习分位数，更灵活。

## 3.6　Rainbow：集成多种改进

Rainbow 算法集成了 DQN 的六种主要改进：

1. 经验回放
2. 目标网络
3. Double DQN
4. Dueling 网络
5. Multi-Step 学习
6. Noisy 网络
7. Distributional RL

**Rainbow 的关键设计：**

1. **多步损失**：使用 n 步 TD 目标。
2. **优先经验回放**：使用分布 TD 误差作为优先级。
3. **Noisy 网络**：用于探索。
4. **Distributional RL**：学习价值分布。
5. **Dueling 架构**：分解状态价值和优势。
6. **Double DQN**：解耦动作选择和评估。

**消融实验结果：**

- **最大贡献**：多步学习和优先经验回放。
- **中等贡献**：Distributional RL 和 Noisy 网络。
- **较小贡献**：Dueling 网络和 Double DQN。
- **组合效应**：所有改进组合的效果远超单个改进。

## 3.7　总结与比较

### 3.7.1　DQN 变种对比

表 3.5: DQN 主要变种的比较

| 算法 | 核心创新 | 解决的问题 | 实现复杂度 |
|---|---|---|---|
| **DQN** | 神经网络近似 Q 函数 | 维度灾难，连续状态空间 | 中等 |
| **优先经验回放** | 按 TD 误差优先级采样 | 样本效率低 | 低 |
| **Double DQN** | 解耦动作选择与评估 | Q 值过估计 | 低 |
| **Dueling DQN** | Q=V+A 架构 | 状态价值与动作优势分离 | 低 |
| **Multi-Step DQN** | n 步 TD 目标 | 短视偏差 | 低 |
| **Noisy DQN** | 参数空间噪声探索 | 探索效率低 | 中等 |
| **Distributional DQN** | 学习价值分布 | 风险意识，丰富信号 | 高 |
| **Rainbow** | 集成所有改进 | 单一改进的局限性 | 高 |

### 3.7.2　实际应用建议

1. **从标准 DQN 开始**：实现基础 DQN，包含经验回放和目标网络。
2. **逐步添加改进**：按优先级添加：Double DQN → 优先经验回放 → Multi-Step → 其他。
3. **超参数调优**：
   - 学习率：通常 $10^{-4}$ 到 $10^{-3}$
   - 折扣因子 $\gamma$：0.99（长期任务）或 0.95（短期任务）
   - 回放缓冲区大小：$10^5$ 到 $10^6$
   - 批量大小：32 到 256
4. **监控训练**：
   - 观察回报曲线
   - 监控 Q 值范围（避免发散）
   - 检查探索率衰减

### 3.7.3　未来方向

1. **分布式强化学习**：多个智能体并行收集经验，加速训练。
2. **元强化学习**：学习快速适应新任务的能力。

3. **基于模型的 DQN**：结合模型预测与价值学习。
4. **多任务学习**：一个智能体学习多个相关任务。
5. **安全约束**：在价值学习中加入安全约束。

价值学习作为强化学习的核心方法，从经典的 Q-Learning 到现代的 Rainbow，经历了显著的发展。理解这些算法的原理、优势和局限性，对于在实际问题中选择合适的算法至关重要。

# 第四章　深度强化学习之策略学习

## 4.1　引言：为什么需要策略学习？

### 4.1.1　价值学习方法的局限性

在前面的学习中，我们探讨了价值学习（Value-Based Learning）方法，如 Q-Learning、DQN 及其变种。这些方法通过学习价值函数（Value Function）来间接得到策略，其核心思想是：找到最优动作价值函数 $Q^*(s,a)$，然后通过贪心策略 $\pi^*(s) = \arg\max_a Q^*(s,a)$ 得到最优策略。

然而，价值学习方法在某些场景下存在明显局限性：

表 4.1: 价值学习方法的局限性

| 局限性类型 | 详细说明与示例 |
| --- | --- |
| **大动作空间问题** | • 当动作空间很大时，需要计算每个动作的 Q 值，计算成本高<br>• 例如：机器人控制可能有几十个连续关节，每个关节有多个自由度<br>• 在 Atari 游戏中，虽然动作空间有限，但某些游戏动作空间也很大 |
| **连续动作空间** | • 连续动作空间中无法通过 $\arg\max$ 选择动作<br>• 例如：自动驾驶的转向角度是连续值<br>• 机械臂控制的关节角度是连续值<br>• 需要额外的优化过程来找到最大化 Q 值的动作 |
| **随机策略表示困难** | • 某些问题的最优策略本质上是随机的<br>• 例如：石头剪刀布游戏中，最优策略是以 1/3 概率选择每个动作<br>• 部分可观察马尔可夫决策过程（POMDP）中可能需要随机策略<br>• 探索时也需要一定随机性 |
| **策略表示受限** | • 通常只能表示确定性策略或 $\epsilon$-贪婪策略<br>• 难以表示更复杂的策略形式<br>• 策略缺乏明确的概率解释 |

### 4.1.2　策略学习的优势

策略学习（Policy Learning）或策略梯度（Policy Gradient）方法直接对策略进行参数化和优化，具有以下优势：

- **直接优化目标**：直接优化累积奖励，不通过价值函数作为中间步骤
- **自然处理连续动作**：策略网络可以直接输出连续动作或动作分布参数
- **支持随机策略**：策略网络可以直接输出动作的概率分布
- **更好的收敛性**：在某些问题中，策略梯度方法有更好的收敛保证
- **探索与利用的自然平衡**：随机策略本身带有探索性质

### 4.1.3　策略学习的基本框架

策略学习的核心思想是：使用参数化的函数（通常是神经网络）直接表示策略 $\pi_\theta(a|s)$，然后通过优化参数 $\theta$ 来最大化期望累积奖励。

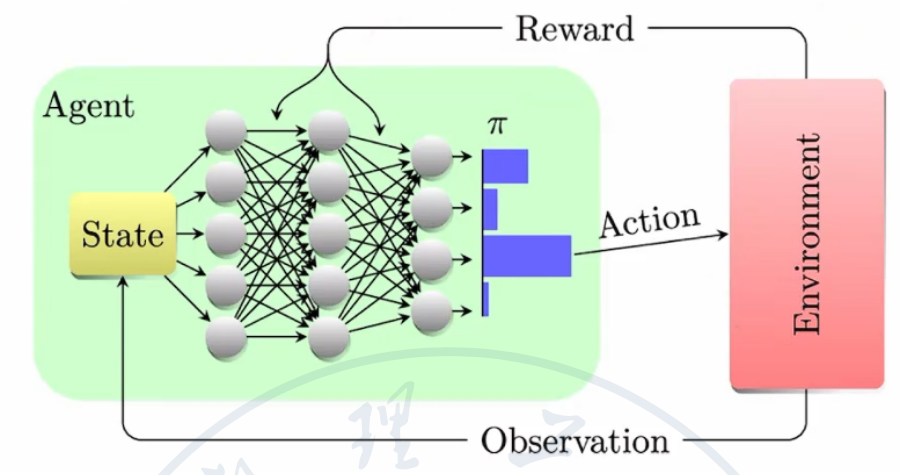



图 4.1: 策略网络：输入状态，输出动作的概率分布

策略网络 $\pi_\theta(a|s)$ 接收状态 $s$ 作为输入，输出动作 $a$ 的概率分布（对于离散动作空间）或动作分布的参数（对于连续动作空间）。

## 4.2　策略梯度定理：理论基础

### 4.2.1　目标函数的定义

我们的目标是最大化期望累积奖励。考虑折扣奖励情况，定义目标函数为：

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=0}^{T} \gamma^t r_{t+1} \right]$$

其中：

- $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T+1})$ 是一条轨迹
- $p_\theta(\tau)$ 是在策略 $\pi_\theta$ 下生成轨迹 $\tau$ 的概率
- $\gamma \in [0, 1]$ 是折扣因子
- $T$ 可以是有限的（分幕式任务）或无限的（持续式任务）

### 4.2.2　轨迹概率的分解

根据马尔可夫决策过程的性质，轨迹概率可以分解为：

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

其中：

- $p(s_0)$ 是初始状态分布
- $\pi_\theta(a_t|s_t)$ 是策略在状态 $s_t$ 下选择动作 $a_t$ 的概率
- $p(s_{t+1}|s_t, a_t)$ 是状态转移概率

### 4.2.3　策略梯度推导

我们需要计算目标函数 $J(\theta)$ 关于参数 $\theta$ 的梯度。直接计算梯度很困难，因为目标函数中包含对轨迹的期望。策略梯度定理提供了计算这个梯度的方法。

**定理 4.2.1** (策略梯度定理). *对于参数化的策略 $\pi_\theta(a|s)$，目标函数 $J(\theta)$ 的梯度为：*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=0}^{T}\gamma^t r_{t+1}\right)\right]$$

**证明：**

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{T}\gamma^t r_{t+1}\right] \\
&= \nabla_\theta \int p_\theta(\tau)R(\tau)d\tau \quad （其中 R(\tau) = \sum_{t=0}^{T}\gamma^t r_{t+1}） \\
&= \int \nabla_\theta p_\theta(\tau)R(\tau)d\tau \\
&= \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)R(\tau)d\tau \quad （使用对数导数技巧） \\
&= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau)R(\tau)\right]
\end{aligned}$$

接下来计算 $\nabla_\theta \log p_\theta(\tau)$：

$$\log p_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T}\left[\log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)\right]$$

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)$$

因为 $\log p(s_0)$ 和 $\log p(s_{t+1}|s_t, a_t)$ 与 $\theta$ 无关，所以它们的梯度为零。代入得：

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

证毕。

### 4.2.4　直观理解

策略梯度定理可以直观地理解：

- $\nabla_\theta \log \pi_\theta(a_t|s_t)$ 是指数族分布的性质，表示在当前参数下，增加动作 $a_t$ 在状态 $s_t$ 下概率的方向
- $R(\tau)$ 是整条轨迹的回报，作为权重
- 如果一条轨迹的回报 $R(\tau)$ 很高，就增加这条轨迹中所有动作的概率
- 如果一条轨迹的回报 $R(\tau)$ 很低，就减少这条轨迹中所有动作的概率

## 4.3　REINFORCE 算法：蒙特卡洛策略梯度

### 4.3.1　基本 REINFORCE 算法

REINFORCE（REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility）是最基础的策略梯度算法。它直接使用蒙特卡洛采样来估计策略梯度。

**Algorithm 3** REINFORCE 算法

**Require:** 策略参数 $\theta$，学习率 $\alpha$，折扣因子 $\gamma$

**Ensure:** 最优策略参数 $\theta^*$

初始化策略参数 $\theta$
**for** 每个回合（episode） **do**
　使用当前策略 $\pi_\theta$ 生成一条轨迹：$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T+1})$
　计算轨迹回报：$R(\tau) = \sum_{t=0}^{T} \gamma^t r_{t+1}$
　**for** $t = 0$ 到 $T$ **do**
　　计算梯度：$g_t = \nabla_\theta \log \pi_\theta(a_t|s_t)$
　**end for**
　估计梯度：$\nabla_\theta J(\theta) \approx \left(\sum_{t=0}^{T} g_t\right) R(\tau)$
　更新参数：$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
**end for**

### 4.3.2　REINFORCE 的改进：因果关系修正

基本 REINFORCE 算法有一个问题：在时刻 $t$ 采取的动作 $a_t$ 不应该影响 $t$ 时刻之前的奖励。因此，我们使用从时刻 $t$ 开始的折扣回报：

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_{k+1}$$

改进后的策略梯度为：

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

对应的算法更新为：

**Algorithm 4** REINFORCE with Causality

**Require:** 策略参数 $\theta$，学习率 $\alpha$，折扣因子 $\gamma$

初始化策略参数 $\theta$
**for** 每个回合（episode） **do**
  使用当前策略 $\pi_\theta$ 生成一条轨迹：$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T+1})$
  计算每个时刻的回报 $G_t$：
  **for** $t = T$ 到 0（反向计算） **do**
    **if** $t = T$ **then**
      $G_T = r_{T+1}$
    **else**
      $G_t = r_{t+1} + \gamma G_{t+1}$
    **end if**
  **end for**
  **for** $t = 0$ 到 $T$ **do**
    计算梯度：$g_t = \nabla_\theta \log \pi_\theta(a_t|s_t)$
    更新参数：$\theta \leftarrow \theta + \alpha g_t G_t$
  **end for**
**end for**

### 4.3.3 REINFORCE 的改进：引入基线

REINFORCE 算法的另一个问题是高方差。即使使用因果关系修正，回报 $G_t$ 的方差仍然很大。为了降低方差，我们引入基线（baseline）$b(s_t)$：

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

基线 $b(s_t)$ 可以是任何不依赖于动作 $a_t$ 的函数。常用的基线是状态值函数 $V(s_t)$ 的估计。可以证明，只要基线不依赖于动作，引入基线不会改变梯度的期望值：

证明.

$$\begin{aligned}\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{T}\nabla_\theta\log\pi_\theta(a_t|s_t)b(s_t)\right] &= \sum_{t=0}^{T}\mathbb{E}_{s_t}\left[\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}\left[\nabla_\theta\log\pi_\theta(a_t|s_t)b(s_t)\right]\right]\\ &= \sum_{t=0}^{T}\mathbb{E}_{s_t}\left[b(s_t)\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}\left[\nabla_\theta\log\pi_\theta(a_t|s_t)\right]\right]\\ &= \sum_{t=0}^{T}\mathbb{E}_{s_t}\left[b(s_t)\cdot 0\right]=0\end{aligned}$$

□

最简单的基线是平均回报：$b=\frac{1}{N}\sum_{i=1}^{N}R(\tau^i)$。改进后的 REINFORCE 算法：

**Algorithm 5** REINFORCE with Baseline

**Require:** 策略参数 $\theta$，学习率 $\alpha$，折扣因子 $\gamma$

初始化策略参数 $\theta$
**for** 每个回合（episode） **do**
　使用当前策略 $\pi_\theta$ 生成一条轨迹，计算 $G_t$
　计算基线 $b$（可以使用多个轨迹的平均回报）
　**for** $t=0$ 到 $T$ **do**
　　计算优势：$A_t=G_t-b$
　　计算梯度：$g_t=\nabla_\theta\log\pi_\theta(a_t|s_t)$
　　更新参数：$\theta\leftarrow\theta+\alpha g_t A_t$
　**end for**
**end for**

### 4.3.4　REINFORCE 的优缺点

表 4.2: REINFORCE 算法的优缺点

| 优点 | 缺点 |
|---|---|
| 简单直观，易于实现 | 高方差，训练不稳定 |
| 可以处理连续动作空间 | 样本效率低（需要完整轨迹） |
| 可以学习随机策略 | 更新频率低（每回合更新一次） |
| 理论上有收敛保证 | 对超参数敏感 |
| 探索充分（通过随机策略） | 可能收敛到局部最优 |

# 4.4　Actor-Critic 方法：结合价值函数

## 4.4.1　Actor-Critic 基本思想

REINFORCE 算法使用蒙特卡洛方法估计回报 $G_t$，虽然无偏但方差大。Actor-Critic 方法引入价值函数（Critic）来估计优势函数，从而降低方差。

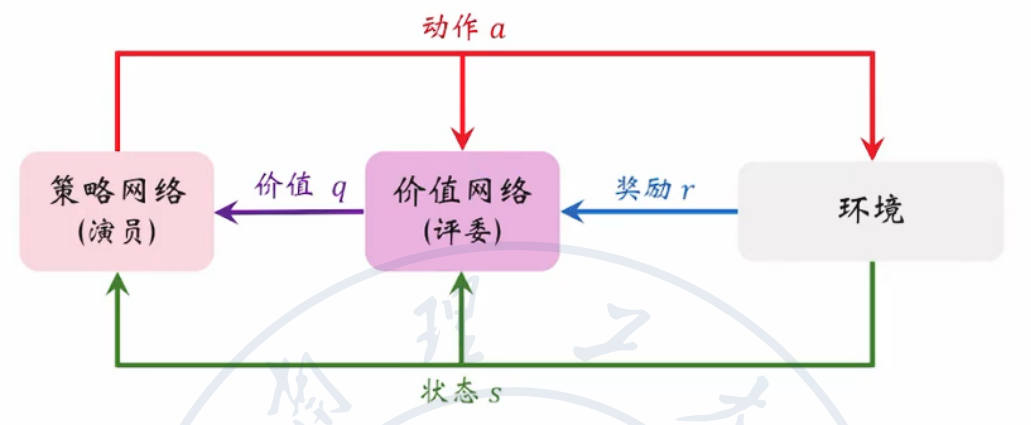



图 4.2: Actor-Critic 框架示意图

Actor-Critic 方法包含两个组件：

- **Actor（演员）**：策略网络 $\pi_\theta(a|s)$，负责选择动作
- **Critic（评论家）**：价值网络 $V_\phi(s)$，负责评估状态的价值

## 4.4.2　优势函数（Advantage Function）

在 Actor-Critic 中，我们使用优势函数 $A(s_t, a_t)$ 替代回报 $G_t$：

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

其中 $Q(s_t, a_t)$ 是动作价值函数，$V(s_t)$ 是状态价值函数。优势函数表示在状态 $s_t$ 下执行动作 $a_t$ 相对于平均水平的优势。

在实际中，我们使用时序差分（TD）误差来估计优势函数：

$$A(s_t, a_t) \approx r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

### 4.4.3　Actor-Critic 算法

**Algorithm 6** Actor-Critic 算法

**Require:** 策略参数 $\theta$，价值参数 $\phi$，学习率 $\alpha_\theta, \alpha_\phi$，折扣因子 $\gamma$

初始化策略参数 $\theta$ 和价值参数 $\phi$
**for** 每个时间步 **do**
　在状态 $s_t$，根据策略 $\pi_\theta(\cdot|s_t)$ 选择动作 $a_t$
　执行动作 $a_t$，获得奖励 $r_{t+1}$，转移到新状态 $s_{t+1}$
　计算 TD 误差：$\delta_t = r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
　更新 Critic：$\phi \leftarrow \phi + \alpha_\phi \delta_t \nabla_\phi V_\phi(s_t)$
　更新 Actor：$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \log \pi_\theta(a_t|s_t)\delta_t$
**end for**

### 4.4.4　Actor-Critic 的优缺点

表 4.3: Actor-Critic 算法的优缺点

| 优点 | 缺点 |
|---|---|
| 方差低，训练稳定 | 有偏差（依赖价值函数估计） |
| 样本效率高（单步更新） | 需要同时训练两个网络 |
| 可以处理连续任务 | 可能不稳定（两个网络相互影响） |
| 更新频率高（每一步更新） | 超参数更多（两个学习率） |

## 4.5　A2C 和 A3C：并行化 Actor-Critic

### 4.5.1　优势 Actor-Critic（A2C）

A2C（Advantage Actor-Critic）是 Actor-Critic 的改进版本，主要改进包括：

1. **明确使用优势函数：**使用 $A(s_t, a_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
2. **多步回报：**使用 n 步回报来估计优势函数
3. **并行训练：**多个环境并行收集数据

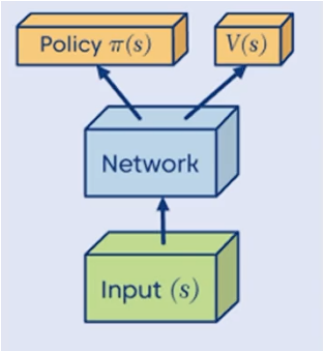



图 4.3: A2C 架构示意图：多个 Worker 并行收集数据，同步更新

### 4.5.2　A2C 算法

**Algorithm 7** A2C 算法

**Require:** 全局策略参数 $\theta$，全局价值参数 $\phi$，学习率 $\alpha_\theta, \alpha_\phi$，折扣因子 $\gamma$，Worker 数量 $N$

初始化全局参数 $\theta, \phi$
**repeat**
　重置所有 Worker 的梯度：$d\theta \leftarrow 0$, $d\phi \leftarrow 0$
　同步所有 Worker 参数：$\theta_i \leftarrow \theta$, $\phi_i \leftarrow \phi$
　**for** 每个 Worker $i = 1$ 到 $N$ **do**
　　Worker $i$ 运行策略 $\pi_{\theta_i}$ 收集轨迹数据
　　计算优势估计 $A_t$（可以使用 n 步回报或 GAE）
　　计算策略梯度：$d\theta_i = \sum_t \nabla_{\theta_i} \log \pi_{\theta_i}(a_t|s_t) A_t$
　　计算价值梯度：$d\phi_i = \sum_t \nabla_{\phi_i} (V_{\phi_i}(s_t) - R_t)^2$
　**end for**
　汇总梯度：$d\theta = \frac{1}{N} \sum_i d\theta_i$, $d\phi = \frac{1}{N} \sum_i d\phi_i$
　更新全局参数：$\theta \leftarrow \theta + \alpha_\theta d\theta$, $\phi \leftarrow \phi + \alpha_\phi d\phi$
**until** 收敛

### 4.5.3　异步优势 Actor-Critic（A3C）

A3C（Asynchronous Advantage Actor-Critic）是 A2C 的异步版本。在 A3C 中，每个 Worker 异步地更新全局网络参数，不需要等待其他 Worker。

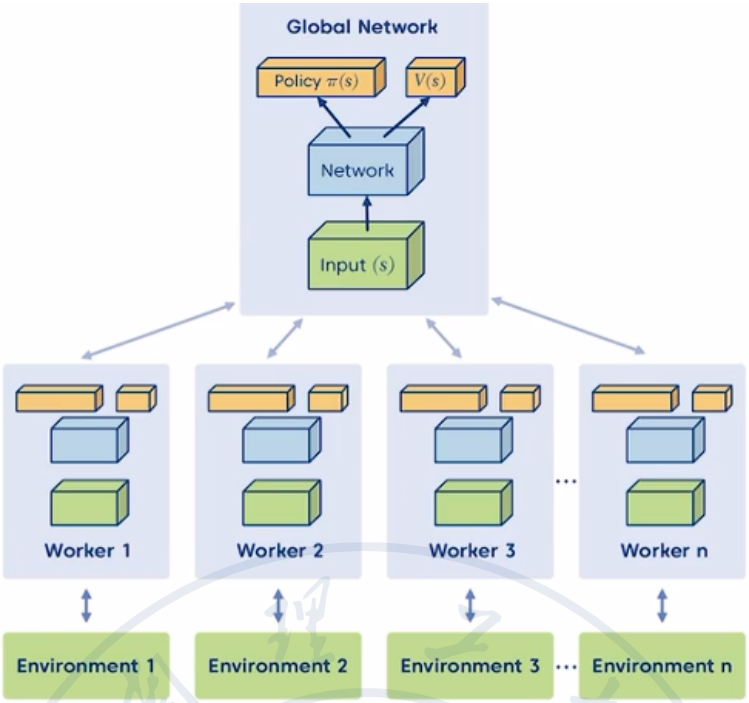



图 4.4: A3C 架构示意图：多个 Worker 异步更新全局网络

A3C 的关键特点：

- **异步更新**：Worker 独立运行，不需要同步
- **探索多样性**：不同 Worker 可能处于不同状态，增加探索
- **硬件效率**：充分利用多核 CPU

### 4.5.4　A2C vs A3C 比较

表 4.4: A2C 与 A3C 比较

| 特性 | A2C | A3C |
|---|---|---|
| 更新方式 | 同步 | 异步 |
| 通信开销 | 高（需要同步） | 低（异步） |
| 实现复杂度 | 较低 | 较高 |
| 收敛速度 | 稳定但可能较慢 | 可能更快但波动大 |
| 硬件利用 | 需要同步，效率较低 | 充分利用多核，效率高 |
| 探索多样性 | 一般 | 好（不同 Worker 不同状态） |

# 4.6　TRPO：置信域策略优化

## 4.6.1　TRPO 的基本思想

TRPO（Trust Region Policy Optimization）的核心思想是：在策略更新时，限制新策略与旧策略的差异，确保每次更新都在一个” 可信赖” 的区域内，从而保证策略性能单调提升。

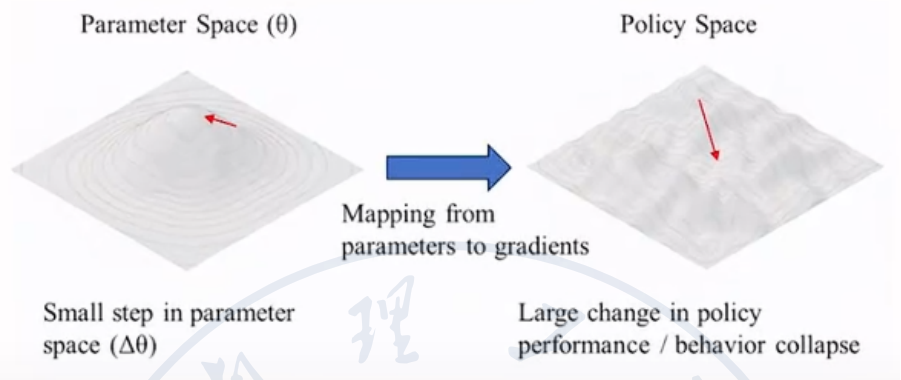



图 4.5: TRPO 的置信域优化：限制策略更新幅度

## 4.6.2　TRPO 的数学形式

TRPO 将策略优化问题形式化为带约束的优化问题：

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}},a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}A_{\theta_{\text{old}}}(s,a)\right] \\ \text{s.t.} \quad & \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s)\|\pi_\theta(\cdot|s))\right] \le \delta \end{aligned}$$

其中：

- $\rho_{\theta_{\text{old}}}$ 是旧策略下的状态分布
- $D_{\text{KL}}$ 是 KL 散度，度量两个策略的差异
- $\delta$ 是置信域半径，限制策略更新的幅度

## 4.6.3　重要性采样（Importance Sampling）

TRPO 使用重要性采样来利用旧策略收集的数据评估新策略：

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[A(s,a)] = \mathbb{E}_{a\sim\pi_{\theta_{\text{old}}}(\cdot|s)}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}A(s,a)\right]$$

重要性采样比率 $\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ 衡量了新策略与旧策略在动作选择上的差异。

### 4.6.4　代理目标函数（Surrogate Objective）

TRPO 优化的是代理目标函数：

$$L(\theta) = \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_{\theta_{\text{old}}}(s,a)\right]$$

这个代理目标函数是原始目标函数 $J(\theta)$ 的一阶近似。

### 4.6.5　TRPO 的求解

TRPO 的约束优化问题通过以下步骤求解：

1. 线性化目标函数：$L(\theta) \approx g^T(\theta - \theta_{\text{old}})$
2. 二次化约束：$\frac{1}{2}(\theta - \theta_{\text{old}})^T H(\theta - \theta_{\text{old}}) \le \delta$
3. 求解约束优化问题：$\theta^* = \theta_{\text{old}} + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$

其中 $g = \nabla_\theta L(\theta)|_{\theta=\theta_{\text{old}}}$ 是梯度，$H$ 是 KL 散度的 Hessian 矩阵（Fisher 信息矩阵）。在实际实现中，使用共轭梯度法计算 $H^{-1}g$，避免直接计算和存储 $H^{-1}$。

### 4.6.6　TRPO 算法

**Algorithm 8** TRPO 算法

**Require:** 初始策略参数 $\theta_0$，置信域半径 $\delta$，回溯系数 $\alpha$

**for** $k = 0, 1, 2, \ldots$ **do**
　使用策略 $\pi_{\theta_k}$ 收集轨迹数据
　估计优势函数 $A_{\theta_k}(s,a)$
　计算代理目标 $L(\theta)$ 和其梯度 $g$
　使用共轭梯度法计算更新方向 $v \approx H^{-1}g$
　计算步长 $\beta = \sqrt{\frac{2\delta}{v^T H v}}$
　候选参数：$\theta_{\text{candidate}} = \theta_k + \beta v$
　回溯线性搜索：找到最大的 $\alpha^j$ 使得 $L(\theta_k + \alpha^j \beta v) \ge L(\theta_k)$ 且满足 KL 约束
　更新参数：$\theta_{k+1} = \theta_k + \alpha^j \beta v$
**end for**

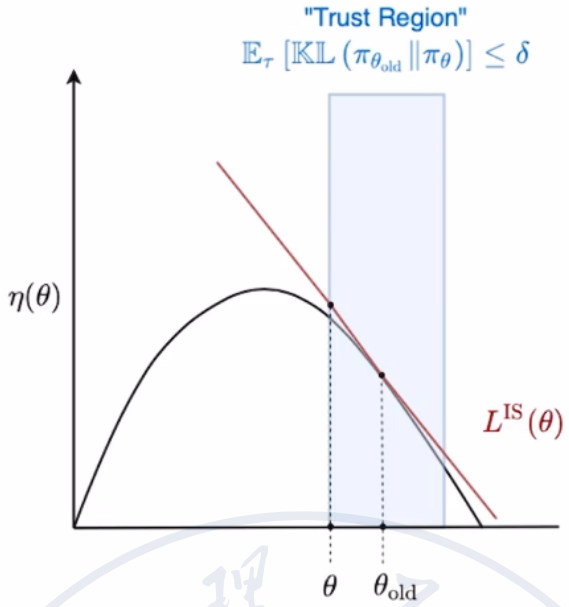



图 4.6: TRPO 的置信域优化过程

### 4.6.7　TRPO 的优缺点

表 4.5: TRPO 算法的优缺点

| 优点 | 缺点 |
| --- | --- |
| 理论保证单调提升 | 计算复杂，需要二阶优化 |
| 更新稳定，避免策略崩溃 | 实现难度大 |
| 适用于复杂任务 | 与某些网络结构不兼容（如 dropout） |
| 样本效率较高 | 超参数敏感（置信域半径 $\delta$） |

## 4.7　PPO：近端策略优化

### 4.7.1　PPO 的基本思想

PPO（Proximal Policy Optimization）是 TRPO 的简化版本，通过修改目标函数来实现策略更新的约束，避免了 TRPO 中复杂的二阶优化。

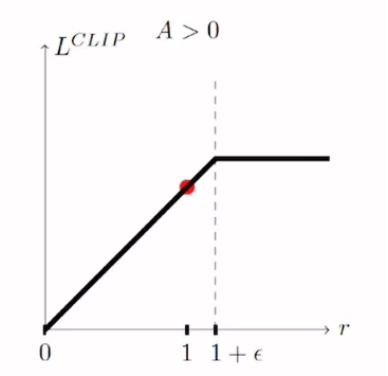



图 4.7: PPO 的裁剪机制：限制重要性采样比率

### 4.7.2　PPO-Clip 目标函数

PPO-Clip 通过裁剪重要性采样比率来限制策略更新：

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right]$$

其中：

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ 是重要性采样比率
- $A_t$ 是优势函数估计
- $\epsilon$ 是裁剪参数，通常取 0.1 或 0.2
- $\text{clip}(x, a, b)$ 将 $x$ 限制在 $[a, b]$ 区间内

### 4.7.3　PPO-Clip 的直观理解

- 当 $A_t > 0$ 时，动作 $a_t$ 优于平均水平，我们希望增加其概率
  - 如果 $r_t(\theta) < 1+\epsilon$，使用 $r_t(\theta) A_t$
  - 如果 $r_t(\theta) \geq 1+\epsilon$，使用 $(1+\epsilon) A_t$，限制更新幅度
- 当 $A_t < 0$ 时，动作 $a_t$ 劣于平均水平，我们希望减少其概率
  - 如果 $r_t(\theta) > 1-\epsilon$，使用 $r_t(\theta) A_t$
  - 如果 $r_t(\theta) \leq 1-\epsilon$，使用 $(1-\epsilon) A_t$，限制更新幅度

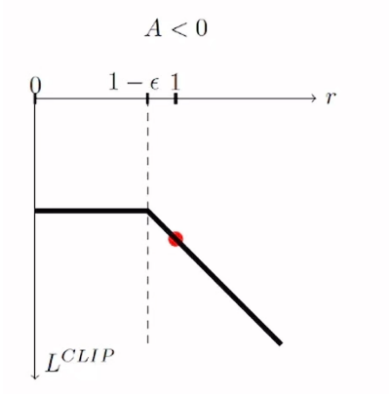



图 4.8: PPO-Clip 目标函数的两种情况

### 4.7.4　PPO-Penalty 目标函数

PPO-Penalty 通过在目标函数中添加 KL 散度惩罚项来约束策略更新：

$$L^{\mathrm{KL}}(\theta) = \mathbb{E}_t \left[ r_t(\theta) A_t - \beta D_{\mathrm{KL}}(\pi_{\theta_{\mathrm{old}}}(\cdot|s_t) \| \pi_\theta(\cdot|s_t)) \right]$$

其中 $\beta$ 是自适应参数，根据 KL 散度的大小调整。

### 4.7.5　广义优势估计（GAE）

GAE（Generalized Advantage Estimation）是一种权衡偏差和方差的优势估计方法：

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

其中 $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ 是 TD 误差。

GAE 的参数 $\lambda$ 控制偏差-方差权衡：

- $\lambda = 0$：$\hat{A}_t = \delta_t$，方差小但偏差大
- $\lambda = 1$：$\hat{A}_t = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}$，偏差小但方差大
- 通常取 $\lambda = 0.95$ 或 0.97，平衡偏差和方差

### 4.7.6　PPO 算法

**Algorithm 9** PPO-Clip 算法

**Require:** 初始策略参数 $\theta_0$，初始价值参数 $\phi_0$，裁剪参数 $\epsilon$，学习率 $\alpha$

1: **for** $k = 0, 1, 2, \ldots$ **do**

2: 使用策略 $\pi_{\theta_k}$ 收集轨迹数据 $\mathcal{D}_k$

3: 计算优势估计 $\hat{A}_t$（使用 GAE）

4: 计算目标价值 $\hat{V}_t = \hat{A}_t + V_{\phi_k}(s_t)$

5: 优化 PPO-Clip 目标函数（多个 epoch）：

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)$$

6: 优化价值函数（多个 epoch）：

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{V}_t\right)^2$$

7: **end for**

### 4.7.7　联合损失函数

在实际实现中，PPO 通常使用联合损失函数，同时优化策略和价值函数：

$$L(\theta, \phi) = L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\phi) + c_2 S[\pi_\theta](s_t)$$

其中：

- $L^{\text{CLIP}}(\theta)$ 是 PPO-Clip 策略损失
- $L^{\text{VF}}(\phi) = (V_\phi(s_t) - V_t^{\text{target}})^2$ 是价值函数损失
- $S[\pi_\theta](s_t)$ 是策略熵，鼓励探索
- $c_1, c_2$ 是系数，平衡不同项

### 4.7.8　PPO 的优缺点

表 4.6: PPO 算法的优缺点

| 优点 | 缺点 |
| --- | --- |
| 实现简单，计算效率高 | 理论保证较弱 |
| 性能稳定，适用于多种任务 | 超参数较多（$\epsilon, c_1, c_2, \lambda$ 等） |
| 样本效率高 | 对超参数敏感 |
| 与多种网络结构兼容 | 可能收敛到局部最优 |
| 支持连续和离散动作空间 | 需要调整学习率调度 |

## 4.8　策略学习方法的比较与应用

### 4.8.1　方法对比

表 4.7: 策略学习方法对比

| 方法 | 核心思想 | 优点 | 缺点 | 适用场景 |
| --- | --- | --- | --- | --- |
| REINFORCE | 蒙特卡洛策略梯度 | 简单，无偏 | 高方差，低效 | 简单任务，探索算法 |
| Actor-Critic | 价值函数引导 | 方差低，高效 | 有偏差，不稳定 | 中等复杂度任务 |
| A2C/A3C | 并行化 AC | 稳定，高效 | 实现复杂 | 需要并行化的任务 |
| TRPO | 置信域优化 | 理论保证，稳定 | 计算复杂 | 高精度控制任务 |
| PPO | 裁剪目标函数 | 简单高效，稳定 | 超参数多 | 通用强化学习任务 |

### 4.8.2　选择指南

1. **简单任务**：可以从 REINFORCE 或 Actor-Critic 开始
2. **中等复杂度任务**：推荐使用 A2C 或 PPO
3. **复杂控制任务**：考虑 TRPO 或 PPO
4. **需要高样本效率**：使用 PPO with GAE
5. **计算资源有限**：使用 PPO（实现简单）
6. **需要理论保证**：使用 TRPO

### 4.8.3　实际应用建议

- **网络架构**：

  - 使用共享主干网络提取特征
  - 策略头输出动作分布参数
  - 价值头输出状态价值
  - 适当使用归一化技术
- **超参数调优：**
  - 学习率：通常 $10^{-4}$ 到 $10^{-3}$
  - 折扣因子 $\gamma$：0.99（长期任务）或 0.95（短期任务）
  - PPO 裁剪参数 $\epsilon$：0.1 到 0.3
  - GAE 参数 $\lambda$：0.9 到 0.99
  - 批量大小：128 到 2048
- **训练技巧：**
  - 使用多个环境并行收集数据
  - 适当增加熵奖励鼓励探索
  - 使用学习率衰减
  - 监控 KL 散度避免策略崩溃

## 4.9　总结与展望

### 4.9.1　策略学习的发展历程

策略学习方法经历了从简单到复杂的发展过程：

- **早期**：REINFORCE 算法提出策略梯度基本思想
- **发展**：Actor-Critic 方法引入价值函数降低方差
- **成熟**：TRPO 提供理论保证的稳定优化
- **普及**：PPO 简化实现，成为实际应用标准
- **前沿**：分布策略梯度，元策略学习等

### 4.9.2　关键技术创新

1. **重要性采样**：允许重用旧数据，提高样本效率
2. **优势函数**：降低梯度方差，提高训练稳定性
3. **置信域方法**：限制策略更新幅度，避免策略崩溃
4. **裁剪技术**：简化置信域约束，提高计算效率
5. **并行化**：提高数据收集效率，加速训练
6. **熵奖励**：鼓励探索，防止过早收敛

### 4.9.3　未来发展方向

1. **样本效率提升**：结合模型学习，元学习
2. **探索策略改进**：好奇心驱动，内在动机
3. **多任务学习**：共享表示，迁移学习
4. **分层策略**：技能学习，选项发现
5. **安全约束**：约束策略优化，安全探索
6. **分布式训练**：大规模并行，分布式优化

### 4.9.4　学习建议

1. **理论基础**：深入理解策略梯度定理，重要性采样
2. **实践能力**：实现基本算法，调试超参数
3. **工具掌握**：熟练使用 RL 框架（如 Stable Baselines3）
4. **代码阅读**：学习开源实现，理解细节
5. **论文跟踪**：关注最新进展，学习新方法

策略学习作为强化学习的核心方法之一，已经在游戏、机器人控制、自然语言处理等领域取得了显著成功。理解策略学习的原理和方法，掌握 PPO 等现代算法，对于解决实际强化学习问题具有重要意义。