

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	12
1.1 启发性思考	12
1.2 神经元：深度学习的基石	12
1.2.1 数学模型	12
1.2.2 关键组件详解	12
1.3 激活函数：从 Sigmoid 到 ReLU	13
1.3.1 经典激活函数	13
1.3.2 激活函数对比与进阶选择	14
1.4 前馈神经网络：层叠的威力	15
1.4.1 网络结构	15
1.4.2 优雅高效的矩阵表示	15
1.4.3 Softmax 函数：多分类的“裁判”	15
1.5 神经网络的训练：让模型学会思考	16
1.5.1 步骤一：定义网络结构	16
1.5.2 步骤二：定义损失函数（Loss Function）	16
1.5.3 步骤三：梯度下降优化	17
1.6 反向传播：深度学习的引擎	17
1.6.1 误差项（Error Term）的定义	17
1.6.2 反向传播的四步流程	17
1.6.3 反向传播推导（选读）	18
1.7 引言：从拟合数据到泛化世界	19
1.8 模型性能评估：不只是看分数	19
1.8.1 数据集划分的艺术	19
1.8.2 K-折交叉验证：小数据集的救星	20
1.9 诊断模型问题：欠拟合与过拟合	20
1.9.1 欠拟合（Underfitting）	20
1.9.2 过拟合（Overfitting）	21
1.9.3 偏差-方差权衡：统计学习理论的视角	21
1.10 正则化技术：防止过拟合的利器	21

1.10.1 L1 和 L2 正则化	21
1.10.2 Dropout: 训练时随机丢弃神经元	22
1.10.3 早停法 (Early Stopping)	22
1.11 从神经网络到深度学习: 历史脉络	23
1.11.1 发展历程	23
1.11.2 深度学习的四大支柱	24
1.11.3 为什么叫“深度学习”?	24
1.12 关键模型架构	25
1.12.1 自编码器 (Autoencoders, AEs)	25
1.12.2 卷积神经网络 (CNN)	25
1.12.3 循环神经网络 (RNN) 与 LSTM	26
1.12.4 生成对抗网络 (GANs)	27
1.12.5 扩散模型 (Diffusion Models)	28
1.13 Transformer: 注意力就是全部	28
1.13.1 自注意力机制 (Self-Attention)	28
1.13.2 多头注意力 (Multi-Head Attention)	29
1.13.3 Transformer 架构	29
1.13.4 位置编码 (Positional Encoding)	30
1.13.5 Transformer vs RNN/LSTM	30
1.14 为什么 Transformer 能够“通吃”?	31
1.14.1 从专家模型到通用模型	31
1.14.2 Transformer 的核心优势	31
1.15 总结与思考	31
第二章 深度强化学习之基础	32
2.1 引言: 从有答案的学习到探索的学习	32
2.1.1 监督学习的辉煌与局限	32
2.1.2 强化学习: 一种新的学习范式	33
2.2 马尔可夫决策过程: 强化学习的数学基础	34
2.2.1 马尔可夫决策过程的基本概念	34
2.2.2 MDP 的核心概念	35
2.3 价值函数: 评估策略的优劣	36
2.3.1 价值函数的定义与意义	36
2.3.2 价值函数的关系	37
2.3.3 最优价值函数	37
2.4 贝尔曼方程: 动态规划的核心	38
2.4.1 贝尔曼期望方程	38
2.4.2 贝尔曼最优方程	39

2.4.3 贝尔曼方程的矩阵形式	40
2.5 强化学习算法分类	40
2.5.1 有模型 vs 无模型	41
2.5.2 基于价值 vs 基于策略	42
2.5.3 蒙特卡洛 vs 时序差分	43
2.5.4 在线策略 vs 离线策略	44
2.6 Actor-Critic 方法：价值与策略的结合	45
2.6.1 Actor-Critic 框架	45
2.6.2 优势函数（Advantage Function）	46
2.6.3 策略梯度与 Actor-Critic	46
2.6.4 现代 Actor-Critic 算法	46
2.7 总结与展望	47
2.7.1 强化学习算法分类总结	47
2.7.2 强化学习的挑战与前沿	48
第三章 深度强化学习之价值学习	49
3.1 价值学习：从评估到决策	49
3.1.1 价值学习的核心思想	49
3.2 Q-Learning：经典的价值学习算法	50
3.2.1 算法原理	50
3.2.2 表格 Q-Learning	50
3.2.3 探索与利用的平衡： ϵ -贪婪策略	51
3.2.4 Q-Learning 与 SARSA 的比较	52
3.3 深度 Q 网络：当 Q-Learning 遇见深度学习	52
3.3.1 表格方法的局限性	52
3.3.2 深度 Q 网络的基本思想	52
3.3.3 DQN 的训练目标与损失函数	53
3.3.4 DQN 的训练流程	54
3.4 DQN 的核心技术	54
3.4.1 经验回放（Experience Replay）	54
3.4.2 目标网络（Target Network）	55
3.5 DQN 的改进与扩展	57
3.5.1 优先经验回放（Prioritized Experience Replay）	57
3.5.2 Double DQN	57
3.5.3 Dueling DQN	58
3.5.4 Multi-Step DQN	59
3.5.5 Noisy DQN	60
3.5.6 Distributional DQN	60

3.6	Rainbow: 集成多种改进	62
3.7	总结与比较	63
3.7.1	DQN 变种对比	63
3.7.2	实际应用建议	63
3.7.3	未来方向	63
第四章 深度强化学习之策略学习		65
4.1	引言: 为什么需要策略学习?	65
4.1.1	价值学习方法的局限性	65
4.1.2	策略学习的优势	66
4.1.3	策略学习的基本框架	67
4.2	策略梯度定理: 理论基础	67
4.2.1	目标函数的定义	67
4.2.2	轨迹概率的分解	67
4.2.3	策略梯度推导	68
4.2.4	直观理解	69
4.3	REINFORCE 算法: 蒙特卡洛策略梯度	69
4.3.1	基本 REINFORCE 算法	69
4.3.2	REINFORCE 的改进: 因果关系修正	69
4.3.3	REINFORCE 的改进: 引入基线	70
4.3.4	REINFORCE 的优缺点	71
4.4	Actor-Critic 方法: 结合价值函数	72
4.4.1	Actor-Critic 基本思想	72
4.4.2	优势函数 (Advantage Function)	72
4.4.3	Actor-Critic 算法	73
4.4.4	Actor-Critic 的优缺点	73
4.5	A2C 和 A3C: 并行化 Actor-Critic	73
4.5.1	优势 Actor-Critic (A2C)	73
4.5.2	A2C 算法	74
4.5.3	异步优势 Actor-Critic (A3C)	74
4.5.4	A2C vs A3C 比较	75
4.6	TRPO: 置信域策略优化	76
4.6.1	TRPO 的基本思想	76
4.6.2	TRPO 的数学形式	76
4.6.3	重要性采样 (Importance Sampling)	76
4.6.4	代理目标函数 (Surrogate Objective)	77
4.6.5	TRPO 的求解	77
4.6.6	TRPO 算法	77

4.6.7 TRPO 的优缺点	78
4.7 PPO: 近端策略优化	78
4.7.1 PPO 的基本思想	78
4.7.2 PPO-Clip 目标函数	79
4.7.3 PPO-Clip 的直观理解	79
4.7.4 PPO-Penalty 目标函数	79
4.7.5 广义优势估计 (GAE)	80
4.7.6 PPO 算法	80
4.7.7 联合损失函数	80
4.7.8 PPO 的优缺点	81
4.8 策略学习方法的比较与应用	81
4.8.1 方法对比	81
4.8.2 选择指南	81
4.8.3 实际应用建议	82
4.9 总结与展望	82
4.9.1 策略学习的发展历程	82
4.9.2 关键技术创新	82
4.9.3 未来发展方向	83
4.9.4 学习建议	83
第五章 深度强化学习之连续控制	84
5.1 引言: 连续控制问题的挑战	84
5.1.1 离散动作空间 vs 连续动作空间	84
5.1.2 连续控制问题的实例	84
5.2 价值学习方法在连续动作空间中的困境	85
5.2.1 Q-Learning/DQN 的核心机制回顾	85
5.2.2 连续动作空间中的挑战	86
5.2.3 传统策略梯度方法的困境	86
5.3 离散化方法: 一种直观的解决方案	86
5.3.1 离散化的基本思想	86
5.3.2 一维离散化	86
5.3.3 多维离散化	87
5.3.4 离散化的数学表示	87
5.3.5 离散化的困境与局限性	87
5.3.6 离散化的适用场景	88
5.4 随机策略梯度方法: 参数化概率分布	89
5.4.1 核心思想	89
5.4.2 为什么选择高斯分布?	89

5.4.3 单变量高斯分布	89
5.4.4 多变量高斯分布与独立性假设	90
5.4.5 策略网络的输出设计	90
5.4.6 动作选择过程	90
5.4.7 计算对数概率	91
5.4.8 数值稳定性问题与解决方案	91
5.4.9 PPO 在连续动作空间中的实现	91
5.5 确定性策略梯度方法	92
5.5.1 确定性策略与随机策略的对比	92
5.5.2 确定性策略梯度定理	93
5.5.3 探索策略：手动添加噪声	93
5.5.4 Critic 指导 Actor 更新	93
5.6 DDPG：深度确定性策略梯度	94
5.6.1 DDPG 算法框架	94
5.6.2 DDPG 算法流程	95
5.6.3 DDPG 的关键技术	95
5.6.4 DDPG 的改进：TD3	96
5.6.5 DDPG 的改进：SAC	96
5.7 方法对比与总结	97
5.7.1 三种方法全面对比	97
5.7.2 选择指南	97
5.7.3 实际应用建议	97
5.7.4 常见问题与解决方案	98
5.8 前沿发展与未来方向	98
5.8.1 当前研究趋势	98
5.8.2 重要算法进展	99
第六章 进化计算之起源与遗传算法	100
6.1 引言：两种哲学思想的碰撞	100
6.1.1 理性主义与经验主义的对立	100
6.1.2 数学优化与进化计算的哲学对应	100
6.1.3 现实世界优化问题的复杂性	101
6.2 遗传算法的提出	102
6.2.1 自然进化的启示	102
6.2.2 生物进化与遗传算法的类比	102
6.3 遗传算法的总体流程	103
6.4 遗传算法的关键组成部分	103
6.4.1 编码策略（Encoding Strategy）	103

6.4.2 适应度函数 (Fitness Function)	104
6.4.3 选择算子 (Selection Operator)	105
6.4.4 交叉算子 (Crossover Operator)	106
6.4.5 变异算子 (Mutation Operator)	107
6.4.6 精英策略 (Elitist Strategy)	108
6.5 遗传算法的参数设置	108
6.5.1 种群大小 (Population Size, N)	108
6.5.2 交叉概率 (Crossover Probability, P_c)	108
6.5.3 变异概率 (Mutation Probability, P_m)	109
6.6 遗传算法的特点	109
6.7 应用 1: 函数优化 (4D Rastrigin 函数)	109
6.7.1 问题定义	109
6.7.2 实例演示	110
6.8 应用 2: 旅行商问题 (Traveling Salesman Problem, TSP)	112
6.8.1 问题定义	112
6.8.2 编码策略: 排列编码 (Permutation Encoding)	112
6.8.3 遗传算子设计	112
6.8.4 GA 求解 TSP 的整体流程	116
6.9 总结	116
第七章 进化计算之连续优化三剑客	118
7.1 引言: 复杂优化问题的挑战与进化计算的意义	118
7.1.1 优化问题的本质与分类	118
7.1.2 传统优化方法的局限性	118
7.1.3 进化计算的优势与哲学基础	119
7.2 适应度地形图: 理解优化问题的几何视角	119
7.2.1 适应度地形图的基本概念	119
7.2.2 地形特征对优化难度的影响	120
7.2.3 适应度地形图的量化分析	121
7.2.4 探索与开发的平衡	122
7.3 差分进化算法: 基于向量差分的全局优化器	123
7.3.1 差分进化的历史背景与基本原理	123
7.3.2 DE 的变异策略详解	124
7.3.3 DE 的交叉算子	125
7.3.4 DE 的选择算子与精英保留	126
7.3.5 DE 的参数设置与调优	126
7.3.6 DE 的改进算法	127
7.3.7 DE 的收敛性分析	128

7.3.8 DE 的优缺点总结	128
7.4 粒子群优化算法：模拟群体智能的优化器	128
7.4.1 粒子群优化的生物学基础	128
7.4.2 标准 PSO 算法	129
7.4.3 PSO 的参数分析与设置	131
7.4.4 PSO 的拓扑结构	132
7.4.5 PSO 的变体算法	133
7.4.6 PSO 的收敛性分析	134
7.4.7 PSO 的优缺点总结	134
7.5 协方差矩阵自适应进化策略：学习问题结构的优化器	135
7.5.1 从简单进化策略到 CMA-ES	135
7.5.2 CMA-ES 的核心组件	136
7.5.3 CMA-ES 的完整算法	137
7.5.4 CMA-ES 的参数设置	138
7.5.5 CMA-ES 的变体与改进	138
7.5.6 CMA-ES 的理论性质	139
7.5.7 CMA-ES 的优缺点总结	139
7.6 算法对比与综合应用	140
7.6.1 三剑客对比分析	140
7.6.2 性能基准测试	140
7.6.3 混合策略与自适应选择	141
7.6.4 实际应用案例	141
7.7 前沿方向与未来展望	142
7.7.1 大规模优化	142
7.7.2 多目标优化	142
7.7.3 昂贵优化	142
7.7.4 动态优化	142
7.7.5 学习优化（Learning to Optimize）	142
7.7.6 理论发展	143
7.8 实践指南与经验总结	144
7.8.1 算法选择流程	144
7.8.2 参数调优建议	144
7.8.3 常见陷阱与避免方法	145
7.8.4 性能评估指标	145
7.9 结论	145

第八章 进化计算之拓展篇	146
8.1 引言：从连续优化到组合优化与程序生成	146
8.1.1 连续优化与组合优化的区别	146
8.1.2 自然启发的优化思想	147
8.2 蚁群优化算法：模拟蚂蚁觅食的智能优化	147
8.2.1 自然界的蚂蚁觅食行为	147
8.2.2 蚁群优化算法的提出	147
8.2.3 基本蚁群系统（Ant System, AS）	148
8.2.4 AS 算法实例：四城市 TSP 问题	151
8.2.5 蚁群系统（Ant Colony System, ACS）的改进	152
8.2.6 ACO 参数设置与调优	155
8.2.7 ACO 与其他算法的对比	155
8.2.8 ACO 的应用领域	156
8.3 遗传编程：自动程序生成的进化方法	156
8.3.1 遗传编程的基本思想	156
8.3.2 GP 的个体表示：程序树	157
8.3.3 GP 的基本步骤	157
8.3.4 GP 的参数设置	160
8.3.5 GP 的应用案例：符号回归	160
8.3.6 GP 的优缺点与挑战	162
8.3.7 GP 与其他算法的对比	163
8.3.8 GP 的应用领域	163
8.4 总结与展望	164
8.4.1 算法对比总结	164
8.4.2 选择指南	164
8.4.3 未来发展方向	164
8.4.4 实践建议	165
第九章 学习辅助的自动算法设计	166
9.1 引言：优化算法的根本局限与自动算法设计的必要性	166
9.1.1 没有免费的午餐定理：优化算法的根本局限性	166
9.1.2 NFL 定理的启示与自动算法设计的动机	167
9.2 传统算法设计方法与元学习	167
9.2.1 经验法则方法	167
9.2.2 元学习方法	168
9.3 学习辅助的优化：L2O 与 NCO	169
9.3.1 L2O 与 NCO 的提出背景	169
9.3.2 神经组合优化（NCO）	169

9.3.3 基于学习的优化 (L2O)	170
9.4 元黑箱优化：自动算法设计的统一框架	171
9.4.1 元黑箱优化的核心思想	171
9.4.2 MetaBBO 的数学形式化	171
9.4.3 MetaBBO 的学习方式分类	172
9.4.4 基于元层学习范式的分类	174
9.5 自动算法设计的具体技术	174
9.5.1 自动算法选择 (Algorithm Selection, AS)	174
9.5.2 自动算法配置 (Algorithm Configuration, AC)	176
9.5.3 自动算法生成 (Algorithm Generation, AG)	177
9.6 MetaBBO 的核心挑战与未来方向	178
9.6.1 核心挑战	178
9.6.2 未来研究方向	179
9.6.3 方法选择与实施	180
9.6.4 评估与部署	180
9.7 总结与展望	181
9.7.1 核心观点总结	181
9.7.2 发展趋势	181
第十章 提纲挈领	182
10.1 学习与优化的宏大图景	182
10.1.1 学习与优化的基本问题	182
10.1.2 三大支柱的融合	182
10.2 神经网络：深度学习的基石	183
10.2.1 从生物神经元到人工神经网络	183
10.2.2 激活函数：引入非线性的关键	183
10.2.3 前馈网络：层叠的威力	183
10.2.4 反向传播：深度学习的引擎	183
10.2.5 过拟合与正则化	183
10.3 强化学习：从交互中学习	184
10.3.1 强化学习的基本框架	184
10.3.2 马尔可夫决策过程 (MDP)	184
10.3.3 价值函数与贝尔曼方程	184
10.3.4 强化学习算法分类	184
10.3.5 探索与利用的权衡	185
10.4 进化计算：自然启发的优化	185
10.4.1 进化计算的基本思想	185
10.4.2 遗传算法 (GA)	185

10.4.3 差分进化 (DE)	186
10.4.4 粒子群优化 (PSO)	186
10.4.5 蚁群优化 (ACO)	186
10.4.6 协方差矩阵自适应进化策略 (CMA-ES)	186
10.5 自动算法设计：元学习的兴起	187
10.5.1 没有免费的午餐定理 (NFL)	187
10.5.2 元学习：学会学习	187
10.5.3 学习优化 (L2O)	187
10.5.4 神经组合优化 (NCO)	187
10.5.5 自动算法设计的方法论	187
10.5.6 元黑箱优化 (MetaBBO)	188
10.6 概念串联：学习与优化的统一视角	188
10.6.1 从监督学习到元学习	188
10.6.2 探索与利用 → 永恒主题	188
10.6.3 表示学习的关键作用	189
10.6.4 从手工设计到自动学习	189
10.7 未来方向与挑战	189
10.7.1 可解释性与可靠性	189
10.7.2 数据效率与计算可持续性	189
10.7.3 多模态与跨领域学习	189
10.7.4 人机协作与社会影响	190
10.8 结语	190

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, \dots, 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)

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

- **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, ∞)	否	缓解	极快

ReLU 的改进变体

- **Leaky ReLU:** $f(z) = \max(\alpha z, z)$ ，其中 α 是一个很小的正数（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 α 作为可学习参数，让网络自行决定负半轴的斜率。
- **ELU (Exponential Linear Unit):** $f(z) = \begin{cases} z, & z > 0 \\ \alpha(e^z - 1), & z \leq 0 \end{cases}$ 。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：

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

- **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”，模型对各類别的不确定性增加。常用于知识蒸馏 (Teacher 模型使用高溫度产生软标签，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 $T \rightarrow 0^+$ 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 α 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:

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

- 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:

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

- 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：



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 w_i $	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

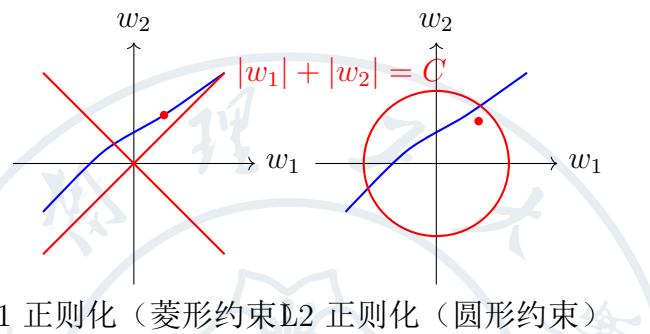


图 1.2: L1 和 L2 正则化的几何解释

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。

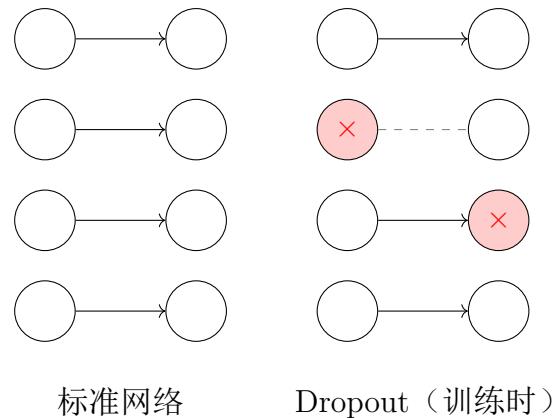


图 1.3: Dropout 示意图

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:
 - 逐层贪婪预训练: 逐层训练受限玻尔兹曼机 (RBM)。
 - 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), 燃料, 数据量决定模型性能上限 互联网产生海量数据	
算力	GPU 并行计算, TPU 专用芯片, 引擎, 使训练深层网络成为可能 云计算平台	
算法	新激活函数 (ReLU), 正则化 智慧, 提升模型性能与训练稳定性 (Dropout), 优化器 (Adam), 新 性 架构 (ResNet)	
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

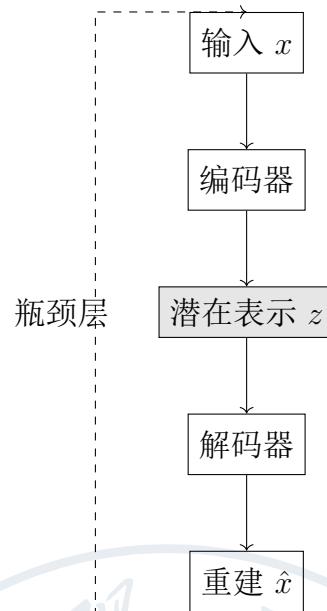


图 1.4: 自编码器结构示意图

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。

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

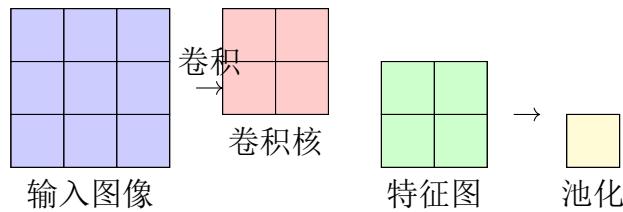


图 1.5: CNN 基本操作：卷积与池化

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

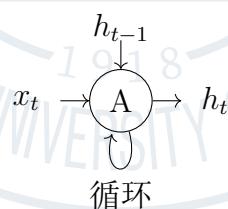


图 1.6: RNN 的表示

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。

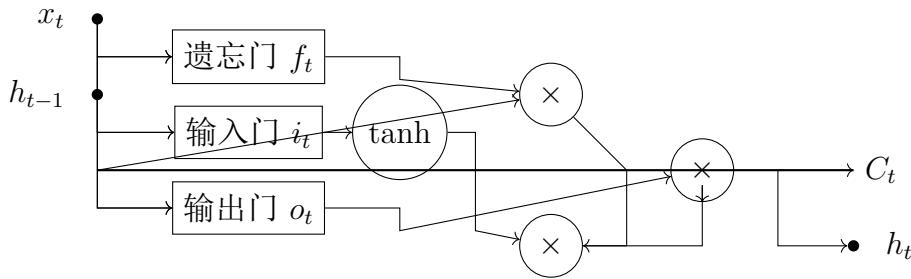


图 1.7: LSTM 单元结构

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 。

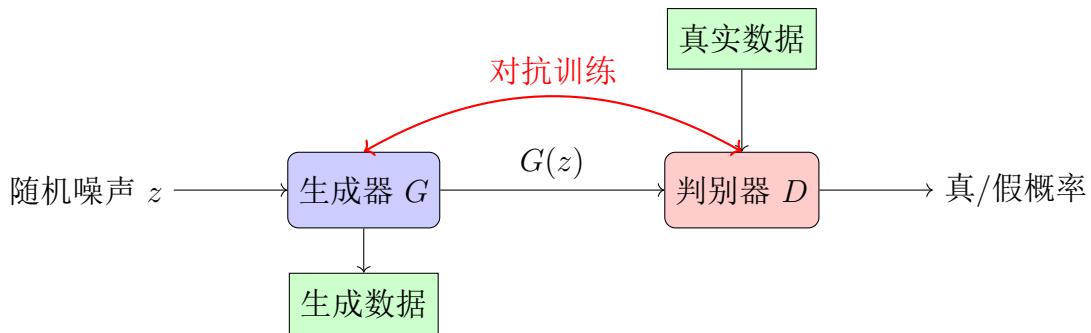


图 1.8: GAN 的基本结构

1.12.5 扩散模型 (Diffusion Models)

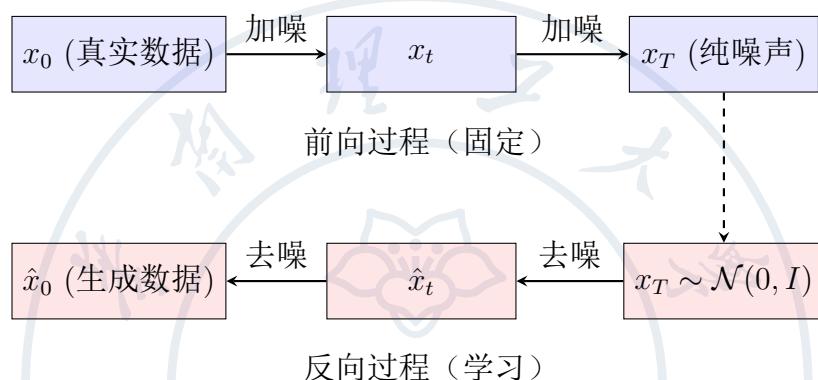


图 1.9: 扩散模型的前向与反向过程

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:

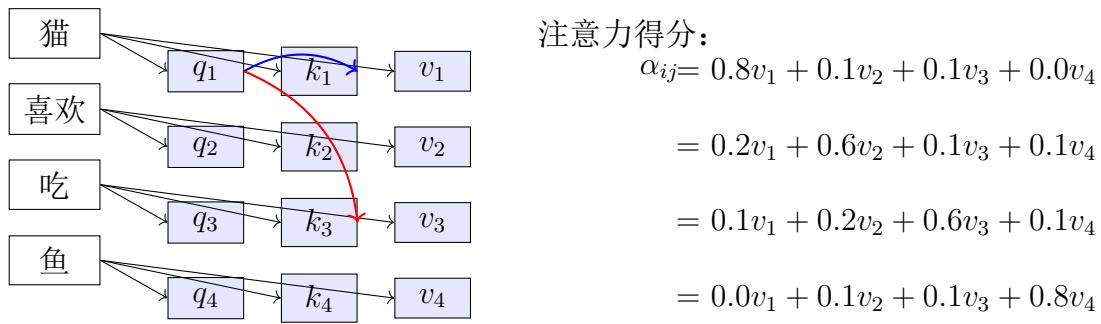


图 1.10: 自注意力机制示例：“吃”与“鱼”有强关联

- **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。

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

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)。

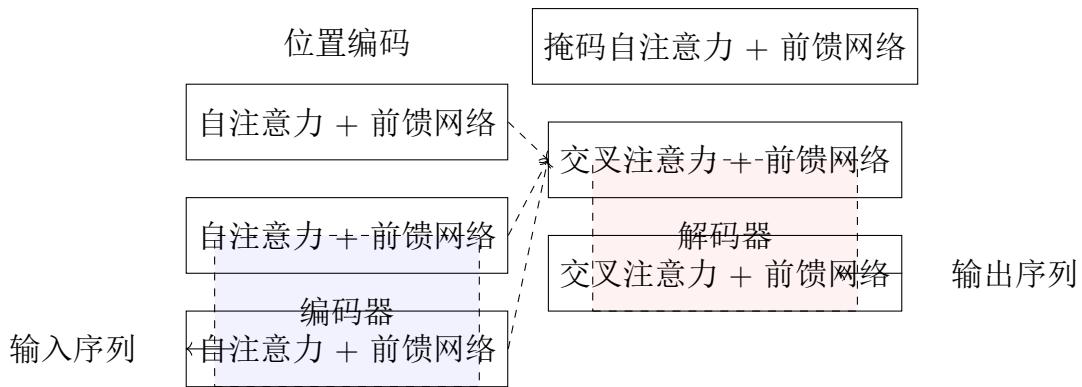


图 1.11: Transformer 架构

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:

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

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 $[w_1, w_2, \dots, w_n]$
 - 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 $f: \mathcal{X} \rightarrow \mathcal{Y}$ ，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”	<ul style="list-style-type: none"> 许多复杂决策问题没有唯一的”正确答案” 示例：围棋中每一步的好坏难以用简单的对错衡量 示例：自动驾驶中的决策权衡（安全 vs 效率）
序贯决策需求	<ul style="list-style-type: none"> 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 示例：下棋时需要规划多步之后的局面 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	<ul style="list-style-type: none"> 环境不断变化，今天的”最优解”明天可能失效 示例：金融市场交易策略 示例：游戏 AI 应对不同玩家风格
探索与利用的权衡	<ul style="list-style-type: none"> 如何在已知的好方法和尝试新方法之间取得平衡 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	<ul style="list-style-type: none"> 打碎杯子 → 受到惩罚 成绩提升 → 获得奖励 	<ul style="list-style-type: none"> 错误动作 → 负奖励 正确动作 → 正奖励
分析	<ul style="list-style-type: none"> ”打碎杯子”的收益小 ”好成绩”的收益大 	<ul style="list-style-type: none"> 评估动作的期望回报 更新策略以最大化回报
学习	<ul style="list-style-type: none"> 减小力度 努力学习 	<ul style="list-style-type: none"> 调整策略参数 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。

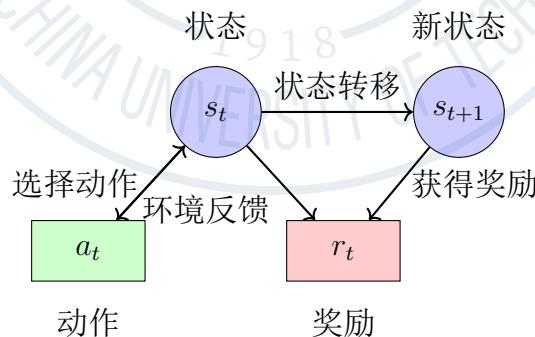


图 2.2: 马尔可夫决策过程的基本流程

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: $p(s' | s, a) = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$

- **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。

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

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

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

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

- $\gamma = 0$: 只关心即时奖励 (短视)
- $\gamma \rightarrow 1$: 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 π

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

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

- **确定性策略:** $\pi(s)$ 输出一个确定的动作 (如 $\pi(s) = \text{"向右移动"}$)
- **随机性策略:** $\pi(a | s)$ 输出动作的概率分布 (如 $\pi(\text{"右"} | s) = 0.8, \pi(\text{"左"} | s) = 0.2$)
- **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:

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

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:

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

- **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 π 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 π 所能获得的期望回报:

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

- **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 π ，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 π 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_π 到 V_π

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 π 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 π 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_π 到 Q_π

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 π 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 π^* , 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:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} Q^*(s, a) \\ Q^*(s, a) &= R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) \cdot V^*(s') \end{aligned}$$

将两式结合, 得到:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) \cdot V^*(s') \right\} \\ Q^*(s, a) &= R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) \cdot \max_{a' \in \mathcal{A}} Q^*(s', a') \end{aligned}$$

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 π 和状态 s , 有:

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

推导过程:

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

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 π 、状态 s 和动作 a , 有:

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

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:

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

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:

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

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 $\mathcal{S} = \{s_1, s_2, \dots, s_N\}$, 定义:

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

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 $p(s' s, a)$ 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: <ul style="list-style-type: none"> 值迭代 (Value Iteration) 策略迭代 (Policy Iteration) 蒙特卡洛树搜索 (MCTS) 	典型算法: <ul style="list-style-type: none"> Q-learning SARSA DQN 策略梯度方法
优点: <ul style="list-style-type: none"> 样本效率高 (可用模型生成虚拟数据) 可进行规划, 无需实际交互 	优点: <ul style="list-style-type: none"> 更通用, 不依赖环境模型 适用于复杂、未知环境
缺点: <ul style="list-style-type: none"> 需要准确的环境模型 模型误差会累积影响策略 	缺点: <ul style="list-style-type: none"> 样本效率较低 需要大量交互数据

有模型方法的进一步分类

- 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
- 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 $p(s' | s, a)$ 和 $R(s, a)$ 的估计
 - 算法: Dyna 框架 (Sutton, 1991)
 - 流程:
 - 用真实交互数据更新模型
 - 用模型生成虚拟数据
 - 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 $\pi(a s; \theta)$, 优化策略参数 θ
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:	典型算法:
<ul style="list-style-type: none"> • Q-learning • DQN • SARSA 	<ul style="list-style-type: none"> • REINFORCE • 策略梯度 • TRPO/PPO
优点:	优点:
<ul style="list-style-type: none"> • 通常更稳定, 收敛性较好 • 样本效率相对较高 	<ul style="list-style-type: none"> • 能处理连续动作空间 • 能学习随机策略 • 收敛到局部最优
缺点:	缺点:
<ul style="list-style-type: none"> • 难以处理连续动作空间 • 通常得到确定性策略 	<ul style="list-style-type: none"> • 方差大, 训练不稳定 • 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. **ϵ -贪婪策略:** 以 $1 - \epsilon$ 的概率选择最优动作, 以 ϵ 的概率随机选择动作

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

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

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

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 θ 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 π_θ 生成轨迹 τ 的概率。

使用策略梯度定理, 可以得到梯度:

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

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:	适用场景:
<ul style="list-style-type: none"> 回合制任务 需要无偏估计 	<ul style="list-style-type: none"> 连续任务 在线学习 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。

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

- 高方差: G_t 依赖于整个后续轨迹, 随机性积累导致高方差。

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

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。

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

- **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。

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

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:	典型算法:
<ul style="list-style-type: none"> • SARSA • REINFORCE • TRPO • PPO 	<ul style="list-style-type: none"> • Q-learning • DQN • DDPG • SAC
优点:	优点:
<ul style="list-style-type: none"> • 理论分析相对简单 • 通常更稳定 	<ul style="list-style-type: none"> • 数据重用, 样本效率高 • 可以学习多个策略 • 支持从示范中学习
缺点:	缺点:
<ul style="list-style-type: none"> • 样本效率低 • 探索与利用需要平衡 	<ul style="list-style-type: none"> • 理论分析复杂 • 可可能存在收敛问题

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:

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

- **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 ϵ -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 ϵ -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：

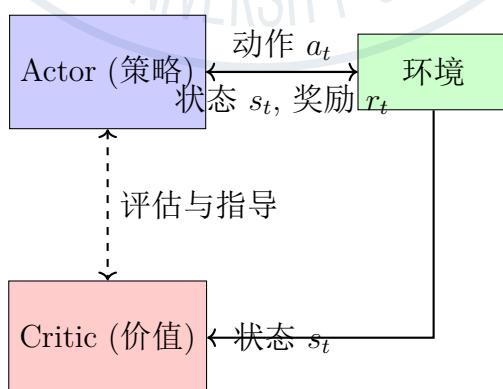


图 2.3: Actor-Critic 框架示意图

- **Actor (执行者):** 负责选择动作，即策略 $\pi_\theta(a | s)$
- **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：

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

- REINFORCE 算法：使用蒙特卡洛回报 G_t 作为优势估计

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

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

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

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, ϵ 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

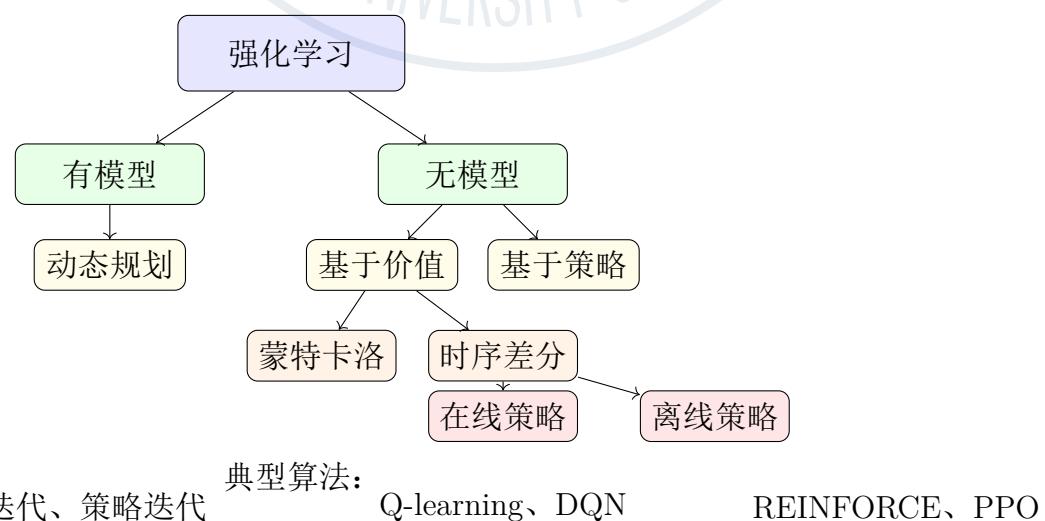


图 2.4: 强化学习算法分类树

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： ϵ -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 π 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:
 - 即时奖励 r
 - 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 α 控制更新步长: α 越大, 更新越快, 但可能不稳定; α 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 α , 折扣因子 γ , 探索率 ϵ

Ensure: 最优动作价值函数 $Q^*(s, a)$

1: 初始化 $Q(s, a)$ 为任意值 (通常为 0 或随机小值)

2: **for** 每个回合 (episode) **do**

3: 初始化状态 s

4: **while** s 不是终止状态 **do**

5: 根据 ϵ -贪婪策略从 s 选择动作 a :

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

6: 执行动作 a , 观察奖励 r 和新状态 s'

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

8: $s \leftarrow s'$

9: **end while**

10: **end for**

3.2.3 探索与利用的平衡: ϵ -贪婪策略

在 Q-Learning 中, ϵ -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 ϵ 随机选择动作 (探索)
- 通常随着训练进行, ϵ 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 ϵ -贪婪)
行为策略: ϵ -贪婪策略	行为策略: ϵ -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_π
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 ϵ -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: $256^{84 \times 84}$ 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 θ 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:

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

其中 n 是动作空间的大小。

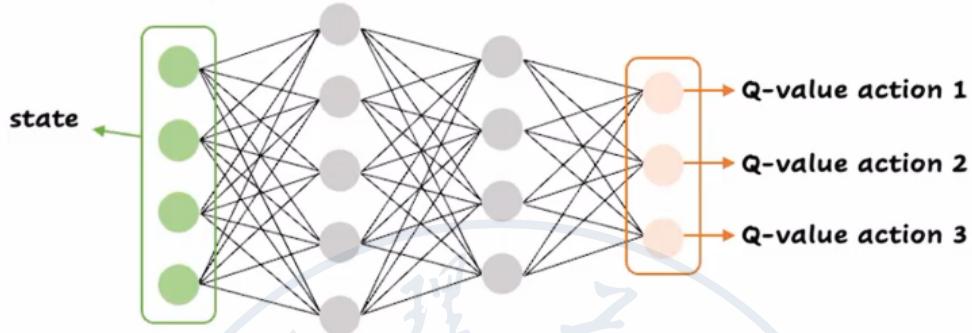


图 3.1: 深度 Q 网络结构示意图: 输入状态, 输出每个动作的 Q 值

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 θ , 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:

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

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: $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$
- θ^- 是目标网络的参数 (与在线网络 θ 不同)

梯度计算:

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

参数更新:

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

其中 α 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 γ , 探索率 ϵ

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

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

- 13: 计算损失: $L = \frac{1}{m} \sum_{j=1}^m (y_j - Q(\phi_j, a_j; \theta))^2$
 - 14: 使用梯度下降更新 θ
 - 15: 每 C 步更新目标网络: $\theta^- \leftarrow \theta$
 - 16: **end for**
 - 17: **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。

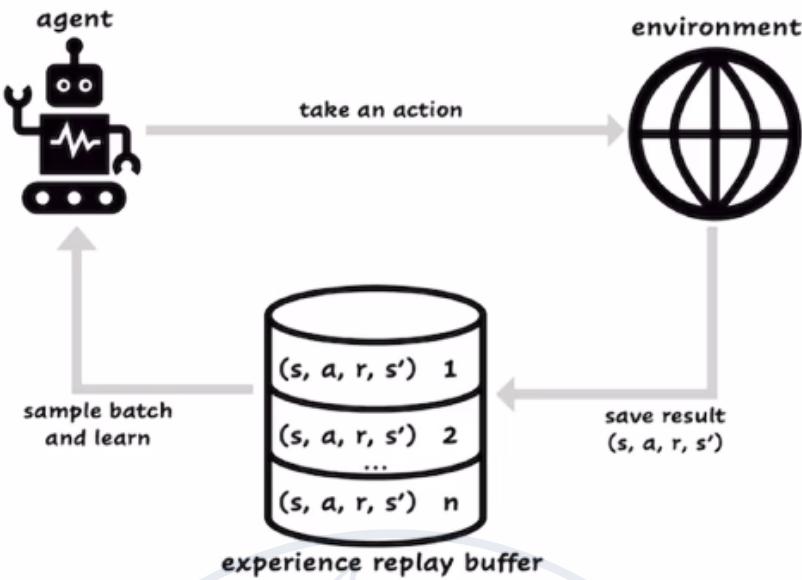


图 3.2: 经验回放机制: 存储历史转移并随机采样用于训练

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。

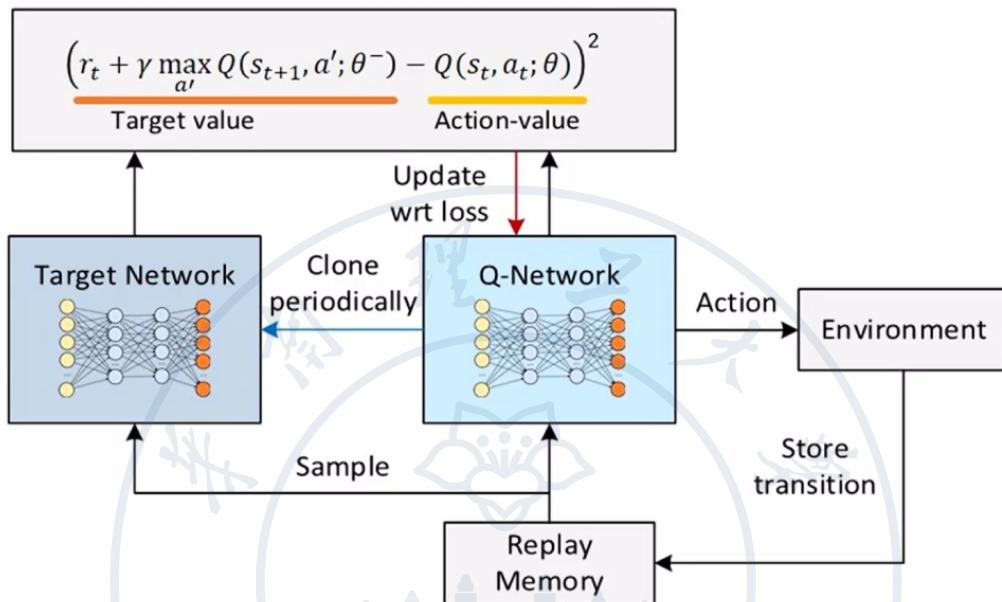


图 3.3: DQN 损失计算：使用目标网络计算 TD 目标

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 θ ，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 θ^- 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 δ_i 是转移 i 的 TD 误差, ϵ 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 α , 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 存有过估计 (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, \dots, X_n 是独立同分布的随机变量，均值为 μ ，方差为 σ^2 。则：

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

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 θ 选择动作: $a^* = \arg \max_{a'} Q(s', a'; \theta)$
- 使用目标网络 θ^- 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。

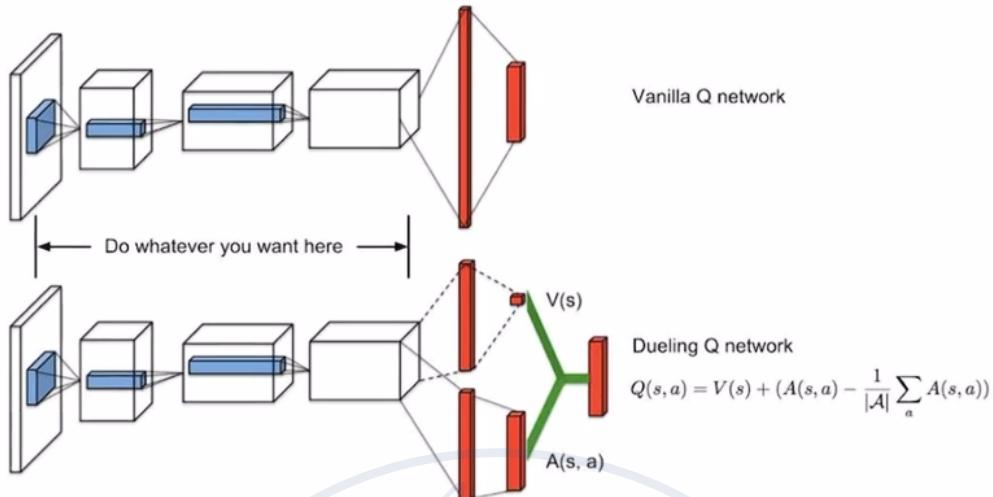


图 3.4: Dueling DQN 网络结构: 将 Q 值分解为状态价值和优势函数

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: $(s_t, a_t, r_t, \dots, r_{t+n-1}, s_{t+n})$

2. 计算 n 步 TD 目标
3. 用标准 DQN 方式更新

3.5.5 Noisy DQN

传统 ϵ -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：

- μ 和 σ 是可学习的参数
- ξ 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 ξ ，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 μ 和 σ 。

表 3.4: ϵ -贪婪与 Noisy DQN 的比较

ϵ -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 ϵ	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:

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

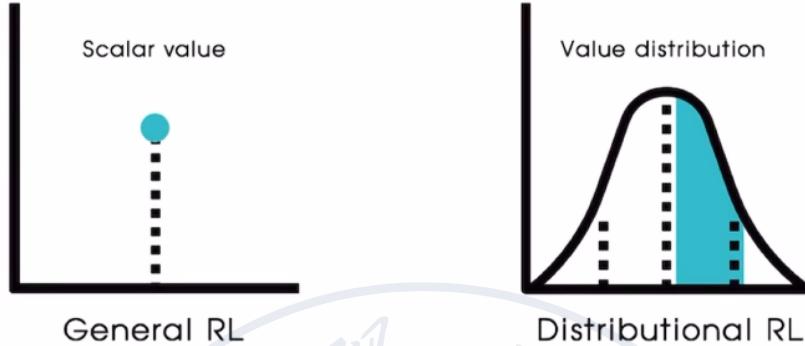


图 3.5: Distributional DQN: 学习价值分布而非期望值

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

- 从标准 DQN 开始: 实现基础 DQN, 包含经验回放和目标网络。
- 逐步添加改进: 按优先级添加: Double DQN → 优先经验回放 → Multi-Step → 其他。
- 超参数调优:
 - 学习率: 通常 10^{-4} 到 10^{-3}
 - 折扣因子 γ : 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
- 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

- 分布式强化学习: 多个智能体并行收集经验, 加速训练。
- 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	<ul style="list-style-type: none"> 当动作空间很大时，需要计算每个动作的 Q 值，计算成本高 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 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	<ul style="list-style-type: none"> 连续动作空间中无法通过 $\arg \max$ 选择动作 例如：自动驾驶的转向角度是连续值 机械臂控制的关节角度是连续值 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	<ul style="list-style-type: none"> 某些问题的最优策略本质上是随机的 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 部分可观察马尔可夫决策过程（POMDP）中可能需要随机策略 探索时也需要一定随机性
策略表示受限	<ul style="list-style-type: none"> 通常只能表示确定性策略或 ϵ-贪婪策略 难以表示更复杂的策略形式 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 θ 来最大化期望累积奖励。

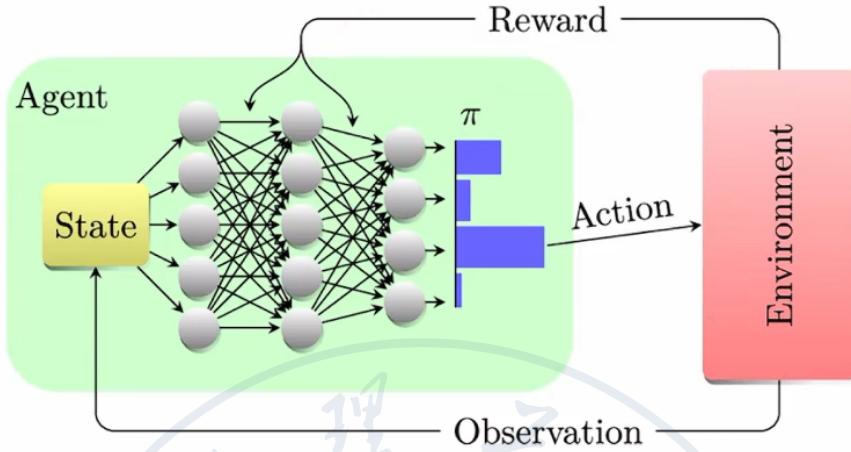


图 4.1: 策略网络：输入状态，输出动作的概率分布

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：

- $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_T, a_T, r_{T+1})$ 是一条轨迹
- $p_\theta(\tau)$ 是在策略 π_θ 下生成轨迹 τ 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 θ 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：

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

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

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

因为 $\log p(s_0)$ 和 $\log p(s_{t+1}|s_t, a_t)$ 与 θ 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 \times Offset Reinforcement \times 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 θ , 学习率 α , 折扣因子 γ

Ensure: 最优策略参数 θ^*

- 1: 初始化策略参数 θ
 - 2: **for** 每个回合 (episode) **do**
 - 3: 使用当前策略 π_{θ} 生成一条轨迹: $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_T, a_T, r_{T+1})$
 - 4: 计算轨迹回报: $R(\tau) = \sum_{t=0}^T \gamma^t r_{t+1}$
 - 5: **for** $t = 0$ 到 T **do**
 - 6: 计算梯度: $g_t = \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$
 - 7: **end for**
 - 8: 估计梯度: $\nabla_{\theta} J(\theta) \approx \left(\sum_{t=0}^T g_t \right) R(\tau)$
 - 9: 更新参数: $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$
 - 10: **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 θ , 学习率 α , 折扣因子 γ

```

1: 初始化策略参数  $\theta$ 
2: for 每个回合 (episode) do
3:   使用当前策略  $\pi_{\theta}$  生成一条轨迹:  $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_T, a_T, r_{T+1})$ 
4:   计算每个时刻的回报  $G_t$ :
5:   for  $t = T$  到 0 (反向计算) do
6:     if  $t = T$  then
7:        $G_T = r_{T+1}$ 
8:     else
9:        $G_t = r_{t+1} + \gamma G_{t+1}$ 
10:    end if
11:   end for
12:   for  $t = 0$  到  $T$  do
13:     计算梯度:  $g_t = \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$ 
14:     更新参数:  $\theta \leftarrow \theta + \alpha g_t G_t$ 
15:   end for
16: 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.

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

□

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 θ , 学习率 α , 折扣因子 γ

- 1: 初始化策略参数 θ
 - 2: **for** 每个回合 (episode) **do**
 - 3: 使用当前策略 π_θ 生成一条轨迹, 计算 G_t
 - 4: 计算基线 b (可以使用多个轨迹的平均回报)
 - 5: **for** $t = 0$ 到 T **do**
 - 6: 计算优势: $A_t = G_t - b$
 - 7: 计算梯度: $g_t = \nabla_\theta \log \pi_\theta(a_t | s_t)$
 - 8: 更新参数: $\theta \leftarrow \theta + \alpha g_t A_t$
 - 9: **end for**
 - 10: **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。

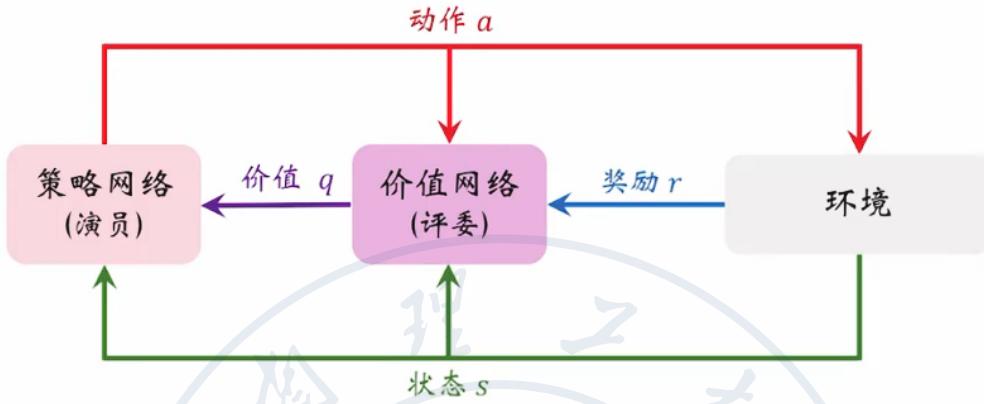


图 4.2: Actor-Critic 框架示意图

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 θ , 价值参数 ϕ , 学习率 $\alpha_\theta, \alpha_\phi$, 折扣因子 γ

- 1: 初始化策略参数 θ 和价值参数 ϕ
 - 2: **for** 每个时间步 **do**
 - 3: 在状态 s_t , 根据策略 $\pi_\theta(\cdot|s_t)$ 选择动作 a_t
 - 4: 执行动作 a_t , 获得奖励 r_{t+1} , 转移到新状态 s_{t+1}
 - 5: 计算 TD 误差: $\delta_t = r_{t+1} + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
 - 6: 更新 Critic: $\phi \leftarrow \phi + \alpha_\phi \delta_t \nabla_\phi V_\phi(s_t)$
 - 7: 更新 Actor: $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \log \pi_\theta(a_t|s_t) \delta_t$
 - 8: **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

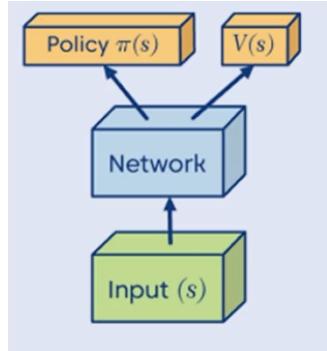


图 4.3: A2C 架构示意图: 多个 Worker 并行收集数据, 同步更新

4.5.2 A2C 算法

Algorithm 7 A2C 算法

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

- 1: 初始化全局参数 θ, ϕ
 - 2: **repeat**
 - 3: 重置所有 Worker 的梯度: $d\theta \leftarrow 0, d\phi \leftarrow 0$
 - 4: 同步所有 Worker 参数: $\theta_i \leftarrow \theta, \phi_i \leftarrow \phi$
 - 5: **for** 每个 Worker $i = 1$ 到 N **do**
 - 6: Worker i 运行策略 π_{θ_i} 收集轨迹数据
 - 7: 计算优势估计 A_t (可以使用 n 步回报或 GAE)
 - 8: 计算策略梯度: $d\theta_i = \sum_t \nabla_{\theta_i} \log \pi_{\theta_i}(a_t | s_t) A_t$
 - 9: 计算价值梯度: $d\phi_i = \sum_t \nabla_{\phi_i} (V_{\phi_i}(s_t) - R_t)^2$
 - 10: **end for**
 - 11: 汇总梯度: $d\theta = \frac{1}{N} \sum_i d\theta_i, d\phi = \frac{1}{N} \sum_i d\phi_i$
 - 12: 更新全局参数: $\theta \leftarrow \theta + \alpha_\theta d\theta, \phi \leftarrow \phi + \alpha_\phi d\phi$
 - 13: **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。

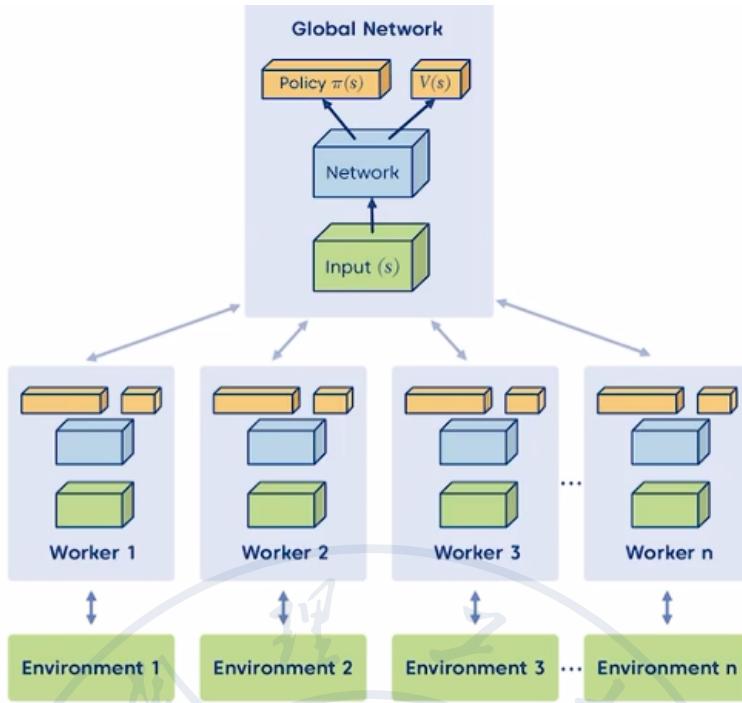


图 4.4: A3C 架构示意图: 多个 Worker 异步更新全局网络

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。

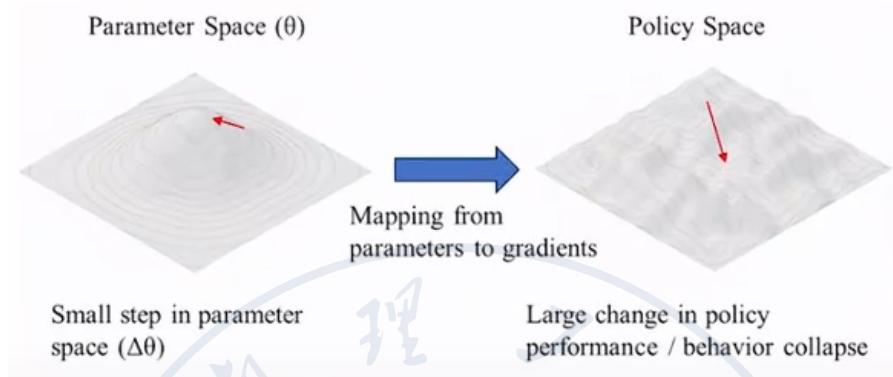


图 4.5: TRPO 的置信域优化：限制策略更新幅度

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：

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

其中：

- $\rho_{\theta_{\text{old}}}$ 是旧策略下的状态分布
- D_{KL} 是 KL 散度，度量两个策略的差异
- δ 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: $\frac{1}{2}(\theta - \theta_{\text{old}})^T H(\theta - \theta_{\text{old}}) \leq \delta$
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 θ_0 , 置信域半径 δ , 回溯系数 α

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

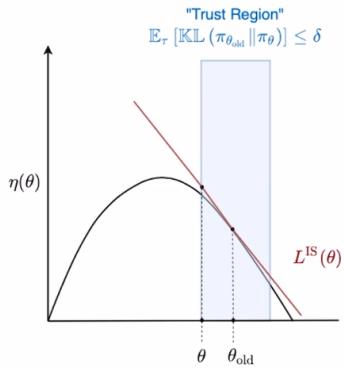


图 4.6: TRPO 的置信域优化过程

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 δ ）

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。

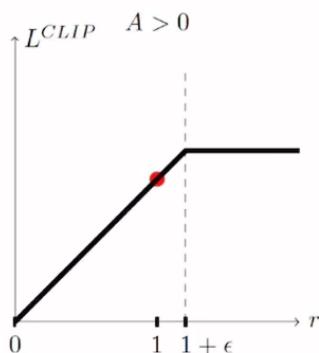


图 4.7: PPO 的裁剪机制：限制重要性采样比率

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：

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

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
- ϵ 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

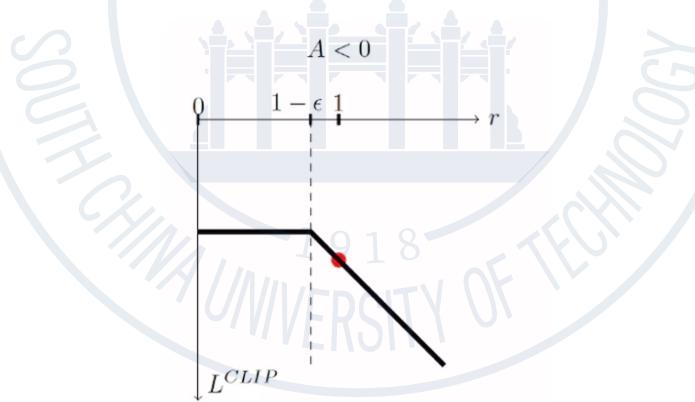


图 4.8: PPO-Clip 目标函数的两种情况

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：

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

其中 β 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:

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

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

GAE 的参数 λ 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 θ_0 , 初始价值参数 ϕ_0 , 裁剪参数 ϵ , 学习率 α

- 1: **for** $k = 0, 1, 2, \dots$ **do**
- 2: 使用策略 π_{θ_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 γ : 0.99 (长期任务) 或 0.95 (短期任务)
 - PPO 裁剪参数 ϵ : 0.1 到 0.3
 - GAE 参数 λ : 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。

第五章 深度强化学习之连续控制

5.1 引言：连续控制问题的挑战

5.1.1 离散动作空间 vs 连续动作空间

在强化学习中，动作空间（Action Space）是智能体（Agent）可以执行的所有可能动作的集合。根据动作的性质，我们可以将动作空间分为两大类：

定义 5.1.1 (离散动作空间 (Discrete Action Space)). 离散动作空间由有限个、可数的动作组成。智能体在每个时间步从这些动作中选择一个执行。

- **特点：**动作数量有限，可以枚举
- **示例：**
 - Atari 游戏：上、下、左、右（4 个动作）
 - 棋类游戏：棋盘上的合法落子位置（几十到几百个动作）
 - 导航问题：前进、后退、左转、右转、停止
- **数学表示：** $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$, 其中 n 是有限正整数

定义 5.1.2 (连续动作空间 (Continuous Action Space)). 连续动作空间中的动作是在一个或多个维度上取值的连续实数。智能体在每个时间步需要指定每个动作维度的具体数值。

- **特点：**动作是连续的，不可枚举
- **示例：**
 - 自动驾驶：方向盘转角（ -90° 到 $+90^\circ$ 的连续值）、油门/刹车（0 到 1 的连续值）
 - 机器人控制：机械臂的 6 个关节角度，每个关节在特定范围内连续变化
 - 无人机控制：三个方向的推力大小，每个推力是连续值
- **数学表示：** $\mathcal{A} \subseteq \mathbb{R}^d$, 其中 d 是动作维度

5.1.2 连续控制问题的实例

为了更直观地理解连续控制问题，让我们看几个具体例子：

示例 5.1.1 (自动驾驶). 自动驾驶汽车需要同时控制多个连续变量：

$$a = \begin{bmatrix} \text{方向盘转角} \\ \text{加速度} \end{bmatrix} = \begin{bmatrix} -15.5^\circ \\ 0.75 \end{bmatrix}$$

其中：

- 方向盘转角：通常在 $[-90^\circ, +90^\circ]$ 范围内连续变化
- 加速度：在 $[-1, 1]$ 范围内连续变化，负值表示刹车

这是一个 2 维连续动作空间。

示例 5.1.2 (6 自由度机械臂控制). 工业机器人需要精确控制每个关节的角度或力矩：

$$a = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix} = \begin{bmatrix} 0.5 \text{ rad} \\ 1.2 \text{ rad} \\ -0.3 \text{ rad} \\ 0.8 \text{ rad} \\ 0.1 \text{ rad} \\ -0.5 \text{ rad} \end{bmatrix}$$

其中每个 θ_i 表示第 i 个关节的角度，这是一个 6 维连续动作空间。

示例 5.1.3 (人形机器人行走). 人形机器人需要控制多个关节的力矩来实现稳定行走：

$$a = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_{12} \end{bmatrix}$$

其中 τ_i 表示第 i 个关节的力矩，通常有 12-20 个关节需要控制。

5.2 价值学习方法在连续动作空间中的困境

5.2.1 Q-Learning/DQN 的核心机制回顾

在离散动作空间中，Q-Learning 和 DQN 等价值学习方法的核心是计算状态-动作价值函数 $Q(s, a)$ ，然后通过最大化操作选择最优动作：

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

这个机制在离散动作空间中运行良好，因为：

1. 可以枚举所有动作 $a \in \mathcal{A}$
2. 为每个动作计算 $Q(s, a)$
3. 比较所有 $Q(s, a)$ 值，选择最大的

5.2.2 连续动作空间中的挑战

在连续动作空间中, $\arg \max$ 操作面临根本性困难:

1. **无限动作数量:** 动作空间是无限的, 无法枚举所有可能的动作
2. **连续优化问题:** $\arg \max_{a \in \mathbb{R}^d} Q(s, a)$ 是一个连续优化问题
3. **计算复杂度:** 每次选择动作都需要解决一个优化问题, 计算代价高昂
4. **局部最优:** $Q(s, a)$ 函数可能非凸, 难以找到全局最优

具体分析: 假设 $Q(s, a)$ 是一个关于 a 的复杂非线性函数, 找到使 $Q(s, a)$ 最大的 a 需要:

$$\nabla_a Q(s, a) = 0$$

然后检查二阶条件 $\nabla_a^2 Q(s, a) \prec 0$ (负定)。这在实际中几乎不可能实时完成。

5.2.3 传统策略梯度方法的困境

对于离散动作空间, 策略网络 $\pi_\theta(a|s)$ 通常使用 Softmax 输出层, 为每个离散动作输出一个概率:

$$\pi_\theta(a|s) = \frac{\exp(f_\theta(s, a))}{\sum_{a' \in \mathcal{A}} \exp(f_\theta(s, a'))}$$

其中 $f_\theta(s, a)$ 是网络为动作 a 计算的得分。

在连续动作空间中, 这种方法失效, 因为:

1. 无法计算分母中的归一化常数 (需要对无限动作积分)
2. 无法为无限个动作分别计算得分

5.3 离散化方法: 一种直观的解决方案

5.3.1 离散化的基本思想

面对连续动作空间的挑战, 一个最直观的想法是离散化 (Discretization): 将连续的无限动作空间转化为离散的有限动作空间, 然后应用已有的离散动作算法。

定义 5.3.1 (离散化). 离散化是将连续动作空间 $\mathcal{A} \subseteq \mathbb{R}^d$ 划分为有限个离散区域, 每个区域用一个代表性动作 (通常是中心点) 表示的过程。

5.3.2 一维离散化

对于一维连续动作, 离散化过程很简单:

示例 5.3.1 (方向盘转角离散化). 假设方向盘转角范围是 $[-90^\circ, +90^\circ]$, 我们可以将其离散化为 5 个档位:

$$\mathcal{A}_{disc} = \{-45^\circ, -15^\circ, 0^\circ, +15^\circ, +45^\circ\}$$

智能体只能选择这 5 个角度之一。

5.3.3 多维离散化

对于多维连续动作，离散化需要更复杂的处理：

示例 5.3.2 (自动驾驶的二维动作离散化). 考虑自动驾驶的两个连续动作维度：

- 方向盘转角 D_1 : 离散化为 5 个档位: $\{-45^\circ, -15^\circ, 0^\circ, +15^\circ, +45^\circ\}$
- 油门/刹车 D_2 : 离散化为 3 个档位: {全力加速, 保持速度, 紧急刹车}

总动作空间是这两个维度的笛卡尔积:

$$\mathcal{A}_{disc} = D_1 \times D_2$$

总动作数为 $5 \times 3 = 15$, 包括 $(-45^\circ, \text{全力加速})$ 、 $(-45^\circ, \text{保持速度})$ 等所有组合。

5.3.4 离散化的数学表示

更一般地, 对于 d 维连续动作空间, 将每个维度 i 离散化为 k_i 个档位, 则总动作数为:

$$|\mathcal{A}_{disc}| = \prod_{i=1}^d k_i$$

这是一个组合爆炸问题: 随着维度增加, 动作数量呈指数增长。

5.3.5 离散化的困境与局限性

尽管离散化方法直观简单, 但它面临严重问题:

表 5.1: 离散化方法的局限性

问题	详细分析
维度灾难 (Curse of Dimensionality)	<ul style="list-style-type: none"> 动作数量随维度指数增长: k^d 例如: 6 自由度机械臂, 每个维度离散化为 10 个级别 $ \mathcal{A}_{\text{disc}} = 10^6 = 1,000,000$ <ul style="list-style-type: none"> 从一百万个动作中选择最优动作计算量巨大 存储 $Q(s, a)$ 表需要大量内存
精度损失	<ul style="list-style-type: none"> 离散化丢失了连续控制能力 真正的最优动作可能在两个离散点之间 例如: 最优转向角可能是 -17.3°, 但离散化后只能选择 -15° 或 -45° 导致策略次优, 控制抖动
探索效率低	<ul style="list-style-type: none"> 离散动作之间的跳跃导致探索不连续 难以学习平滑的控制策略 在边界附近可能产生振荡
不适用于高精度控制	<ul style="list-style-type: none"> 机器人手术、精密加工等需要高精度连续控制 离散化无法满足精度要求 可能导致系统不稳定

5.3.6 离散化的适用场景

尽管有诸多限制, 离散化在特定情况下仍有价值:

- 动作维度很低 (1-2 维)
- 控制精度要求不高
- 快速原型验证
- 与现有离散算法兼容

但对于大多数实际连续控制问题, 我们需要更优雅的解决方案。

5.4 随机策略梯度方法：参数化概率分布

5.4.1 核心思想

随机策略梯度（Stochastic Policy Gradient）方法的核心理念是：策略网络不直接输出动作，而是输出一个概率分布的参数，然后从这个分布中采样得到动作。

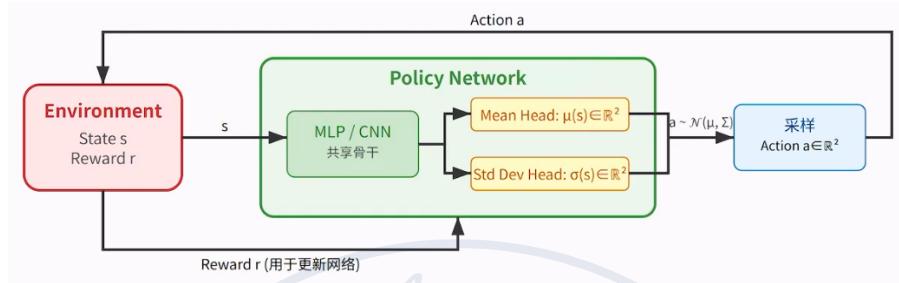


图 5.1: 随机策略网络：输出概率分布参数，从中采样动作

5.4.2 为什么选择高斯分布？

高斯分布（正态分布）是连续动作空间中最常用的概率分布，原因如下：

1. **参数简单：**只需均值和标准差两个参数
 - 均值 μ : 策略认为“最优”的动作
 - 标准差 σ : 探索的程度
2. **数学性质良好：**
 - 中心极限定理保证许多随机变量的和近似服从高斯分布
 - 具有最大熵性质，是最“随机”的分布
3. **计算方便：**
 - 概率密度函数有解析表达式
 - 采样、计算对数概率都很简单
4. **物理意义明确：**
 - 许多物理系统的噪声是高斯分布
 - 符合“大部分情况下接近均值，偶尔有较大偏差”的直觉

5.4.3 单变量高斯分布

对于一维连续动作，策略网络输出均值和标准差：

$$a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)^2)$$

其中：

- $\mu_\theta(s)$: 策略网络在状态 s 下输出的均值

- $\sigma_\theta(s)$: 策略网络在状态 s 下输出的标准差（必须为正）

概率密度函数为：

$$\pi_\theta(a|s) = \frac{1}{\sigma_\theta(s)\sqrt{2\pi}} \exp\left(-\frac{(a - \mu_\theta(s))^2}{2\sigma_\theta(s)^2}\right)$$

5.4.4 多变量高斯分布与独立性假设

对于 d 维连续动作，理论上应该使用多元高斯分布：

$$a \sim \mathcal{N}(\mu_\theta(s), \Sigma_\theta(s))$$

其中 $\Sigma_\theta(s)$ 是 $d \times d$ 的协方差矩阵。

但实际上通常做独立性假设：各动作维度相互独立。此时协方差矩阵是对角矩阵：

$$\Sigma_\theta(s) = \text{diag}(\sigma_1^2, \sigma_2^2, \dots, \sigma_d^2)$$

概率密度函数简化为各维度概率密度的乘积：

$$\pi_\theta(a|s) = \prod_{i=1}^d \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(a_i - \mu_i)^2}{2\sigma_i^2}\right)$$

5.4.5 策略网络的输出设计

在实现中，策略网络需要输出概率分布的参数：

1. 均值 $\mu_\theta(s)$: 网络直接输出 d 维向量
2. 标准差 $\sigma_\theta(s)$: 两种常见设计
 - 状态相关：网络输出另一个 d 维向量，通过 \exp 函数确保正性
 - 状态无关： σ 作为可学习参数，不依赖状态

为了保证标准差为正，通常输出 $\log \sigma$ ，然后通过指数函数转换：

$$\sigma = \exp(\log \sigma)$$

5.4.6 动作选择过程

在状态 s_t 下，策略网络输出参数后，动作选择过程为：

Algorithm 10 随机策略的动作选择

Require: 状态 s_t , 策略网络参数 θ

Ensure: 动作 a_t

- 1: 前向传播: $[\mu_\theta(s_t), \log \sigma_\theta(s_t)] = \text{PolicyNetwork}(s_t)$
- 2: 计算标准差: $\sigma_\theta(s_t) = \exp(\log \sigma_\theta(s_t))$
- 3: 从标准正态分布采样: $\epsilon \sim \mathcal{N}(0, I)$
- 4: 计算动作: $a_t = \mu_\theta(s_t) + \sigma_\theta(s_t) \odot \epsilon$, 其中 \odot 是逐元素乘法
- 5: 执行动作 a_t , 获得奖励 r_{t+1} 和下一状态 s_{t+1}

这个过程称为“重参数化技巧”(Reparameterization Trick)，它使采样过程可微分，便于梯度传播。

5.4.7 计算对数概率

在策略梯度方法中，需要计算 $\log \pi_\theta(a|s)$ 及其梯度。对于单变量高斯分布：

$$\begin{aligned}\log \pi_\theta(a|s) &= \log \left(\frac{1}{\sigma \sqrt{2\pi}} \exp \left(-\frac{(a-\mu)^2}{2\sigma^2} \right) \right) \\ &= -\frac{(a-\mu)^2}{2\sigma^2} - \log \sigma - \log \sqrt{2\pi}\end{aligned}$$

对于 d 维独立高斯分布：

$$\begin{aligned}\log \pi_\theta(a|s) &= \log \left(\prod_{i=1}^d \pi_\theta(a_i|s) \right) \\ &= \sum_{i=1}^d \log \pi_\theta(a_i|s) \\ &= -\frac{1}{2} \sum_{i=1}^d \left(\frac{(a_i - \mu_i)^2}{\sigma_i^2} + 2 \log \sigma_i + \log(2\pi) \right)\end{aligned}$$

5.4.8 数值稳定性问题与解决方案

计算概率时可能遇到数值下溢问题：多个小概率相乘结果趋近于 0。解决方案是在对数空间计算：

问题：

$$\pi_\theta(a|s) = \prod_{i=1}^d \pi_\theta(a_i|s) \rightarrow 0 \quad \text{当 } d \text{ 很大时}$$

解决方案：使用对数概率

$$\log \pi_\theta(a|s) = \sum_{i=1}^d \log \pi_\theta(a_i|s)$$

在 PPO 中计算重要性采样比率时：

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} = \exp (\log \pi_\theta(a_t|s_t) - \log \pi_{\theta_k}(a_t|s_t))$$

5.4.9 PPO 在连续动作空间中的实现

将上述技术整合到 PPO 框架中：

Algorithm 11 PPO for Continuous Control

Require: 初始化策略参数 θ , 价值参数 ϕ , 裁剪参数 ϵ , 学习率 α

- 1: **for** 迭代 $k = 0, 1, 2, \dots$ **do**
 - 2: 使用当前策略 π_{θ_k} 收集轨迹数据 \mathcal{D}_k
 - 3: 计算优势估计 \hat{A}_t (使用 GAE)
 - 4: **for** epoch = 1 to M **do**
 - 5: 从 \mathcal{D}_k 中采样小批量数据
 - 6: 计算对数概率: $\log \pi_\theta(a_t|s_t)$ 和 $\log \pi_{\theta_k}(a_t|s_t)$
 - 7: 计算重要性采样比率: $r_t(\theta) = \exp(\log \pi_\theta(a_t|s_t) - \log \pi_{\theta_k}(a_t|s_t))$
 - 8: 计算裁剪目标函数:
- $$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]$$
- 9: 更新策略参数: $\theta \leftarrow \theta + \alpha \nabla_\theta L^{\text{CLIP}}(\theta)$
 - 10: **end for**
 - 11: **end for**

5.5 确定性策略梯度方法

5.5.1 确定性策略与随机策略的对比

表 5.2: 随机策略 vs 确定性策略

随机策略 (Stochastic Policy)	确定性策略 (Deterministic Policy)
输出动作的概率分布 $\pi_\theta(a s)$	直接输出动作 $a = \mu_\theta(s)$
从分布中采样得到动作	动作被唯一确定
自带探索能力	需要手动添加噪声探索
通常用于在线策略算法	天然适合离线策略算法
示例: PPO、TRPO、A3C	示例: DPG、DDPG、TD3

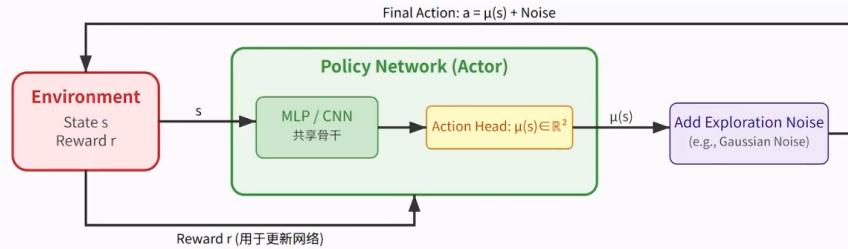


图 5.2: 确定性策略: 直接输出动作值

5.5.2 确定性策略梯度定理

定理 5.5.1 (确定性策略梯度定理). 对于确定性策略 $a = \mu_\theta(s)$, 目标函数 $J(\theta) = \mathbb{E}_{s \sim \rho^\mu}[Q^\mu(s, \mu_\theta(s))]$ 的梯度为:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]$$

其中 ρ^μ 是策略 μ 下的状态分布。

直观理解:

- $\nabla_a Q^\mu(s, a)$: Critic 告诉 Actor 动作应该如何变化以提高 Q 值
- $\nabla_\theta \mu_\theta(s)$: Actor 的参数如何影响输出动作
- 两者结合: 通过链式法则, 确定如何更新 Actor 参数以提高 Q 值

5.5.3 探索策略: 手动添加噪声

确定性策略本身没有探索能力, 需要在训练时手动添加噪声:

Algorithm 12 确定性策略的探索

Require: 状态 s_t , 确定性策略 μ_θ , 探索噪声 \mathcal{N}

Ensure: 动作 a_t

- 1: 计算确定性动作: $a_{\text{det}} = \mu_\theta(s_t)$
 - 2: 添加探索噪声: $a_t = a_{\text{det}} + \mathcal{N}$
 - 3: 执行动作 a_t , 收集经验存入回放缓冲区
-

常用的噪声类型:

- 高斯噪声: $\mathcal{N} \sim \mathcal{N}(0, \sigma^2)$
- Ornstein-Uhlenbeck 过程: 具有时间相关性的噪声, 适合物理系统
- 参数空间噪声: 在策略参数中添加噪声

5.5.4 Critic 指导 Actor 更新

在确定性策略梯度中, Critic 网络 $Q_\phi(s, a)$ 指导 Actor 更新:

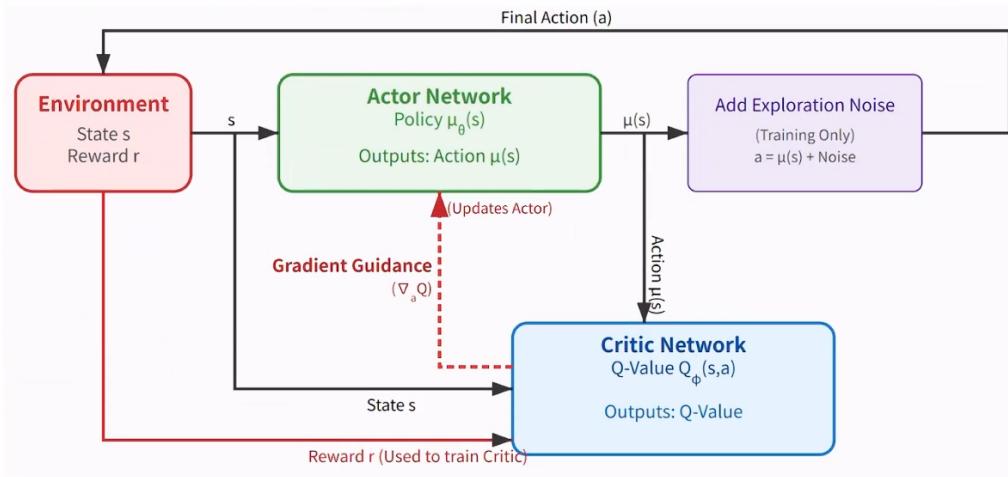


图 5.3: Critic 指导 Actor 更新: 通过 Q 函数对动作的梯度

更新过程:

1. Actor 输出动作: $a = \mu_\theta(s)$
2. Critic 评估动作: $Q = Q_\phi(s, a)$
3. 计算 Q 对动作的梯度: $\nabla_a Q_\phi(s, a)$
4. 通过链式法则更新 Actor:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \mu_\theta(s) \nabla_a Q_\phi(s, a)]$$

5.6 DDPG: 深度确定性策略梯度

5.6.1 DDPG 算法框架

DDPG (Deep Deterministic Policy Gradient) 结合了 DQN 和 DPG 的优点:

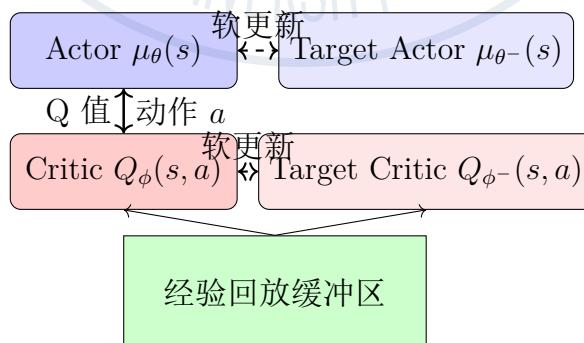


图 5.4: DDPG 算法框架: Actor-Critic 结构 + 目标网络 + 经验回放

5.6.2 DDPG 算法流程

Algorithm 13 DDPG 算法

Require: 学习率 $\alpha_\theta, \alpha_\phi$, 软更新系数 τ , 折扣因子 γ , 探索噪声 \mathcal{N} , 批量大小 N

- 1: 随机初始化 Actor 网络 $\mu_\theta(s)$ 和 Critic 网络 $Q_\phi(s, a)$ 的参数
- 2: 初始化目标网络: $\theta^- \leftarrow \theta$, $\phi^- \leftarrow \phi$
- 3: 初始化经验回放缓冲区 \mathcal{D}
- 4: **for** 回合 = 1 to M **do**
- 5: 初始化环境, 获得初始状态 s_1
- 6: **for** 时间步 $t = 1$ to T **do**
- 7: 选择动作: $a_t = \mu_\theta(s_t) + \mathcal{N}_t$, 其中 \mathcal{N}_t 是探索噪声
- 8: 执行动作 a_t , 获得奖励 r_t 和下一状态 s_{t+1}
- 9: 存储转移 (s_t, a_t, r_t, s_{t+1}) 到 \mathcal{D}
- 10: 从 \mathcal{D} 中采样 N 个转移的批量 $\{(s_i, a_i, r_i, s_{i+1})\}$
- 11: 计算目标 Q 值: $y_i = r_i + \gamma Q_{\phi^-}(s_{i+1}, \mu_{\theta^-}(s_{i+1}))$
- 12: 更新 Critic: 最小化损失 $L = \frac{1}{N} \sum_i (y_i - Q_\phi(s_i, a_i))^2$
- 13: 更新 Actor: 使用确定性策略梯度

$$\nabla_\theta J \approx \frac{1}{N} \sum_i \nabla_a Q_\phi(s_i, a)|_{a=\mu_\theta(s_i)} \nabla_\theta \mu_\theta(s_i)$$

- 14: 软更新目标网络:

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$$

$$\phi^- \leftarrow \tau\phi + (1 - \tau)\phi^-$$
 - 15: **end for**
 - 16: **end for**
-

5.6.3 DDPG 的关键技术

目标网络 (Target Networks)

- 解决“追逐移动目标”问题
- 使用软更新: $\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$
- τ 通常很小 (如 0.001), 使目标网络变化缓慢

经验回放 (Experience Replay)

- 打破数据相关性
- 提高数据利用率
- 支持离线策略学习

探索噪声

- 时间相关噪声: Ornstein-Uhlenbeck 过程

$$d\mathcal{N}_t = \theta(\mu - \mathcal{N}_t)dt + \sigma dW_t$$

- 随时间衰减: 训练后期减少探索

5.6.4 DDPG 的改进: TD3

TD3 (Twin Delayed DDPG) 解决了 DDPG 的三个问题:

- 过估计问题: 使用两个 Critic 网络, 取最小值

$$y = r + \gamma \min_{j=1,2} Q_{\phi_j^-}(s', \mu_{\theta^-}(s') + \epsilon)$$

- 方差累积: 添加目标策略平滑

$$a' = \mu_{\theta^-}(s') + \text{clip}(\mathcal{N}(0, \sigma), -c, c)$$

- 策略更新延迟: Critic 更新多次后, Actor 才更新一次

5.6.5 DDPG 的改进: SAC

SAC (Soft Actor-Critic) 结合了随机策略和离线策略学习的优点:

- 最大化熵目标:

$$J(\theta) = \mathbb{E} \left[\sum_t r(s_t, a_t) + \alpha \mathcal{H}(\pi_\theta(\cdot | s_t)) \right]$$

- 自动调整温度参数: α 控制熵的重要性
- 随机策略: 输出高斯分布参数
- 离线策略: 使用经验回放
- 双 Q 网络: 防止过估计

5.7 方法对比与总结

5.7.1 三种方法全面对比

表 5.3: 连续控制方法全面对比

特性	离散化方法	随机策略梯度	确定性策略梯度
核心思想	连续空间离散化	学习概率分布	学习确定性映射
策略输出	离散动作的 Q 值	分布参数 (μ, σ)	动作向量 a
动作选择	$\arg \max Q(s, a)$	从分布中采样	直接输出 + 噪声
探索机制	ϵ -greedy 等	分布采样	手动添加噪声
学习方式	价值学习	策略梯度	确定性策略梯度
样本效率	中等	较低 (在线)	高 (离线)
收敛速度	慢	中等	快
稳定性	中等	高	较低
实现难度	简单	中等	复杂
代表算法	DQN (离散化)	PPO, TRPO, A3C	DDPG, TD3, SAC
适用场景	低维简单控制	需要探索的任务	高维精确控制

5.7.2 选择指南

表 5.4: 方法选择指南

考虑因素	推荐方法
动作维度低 (1-2 维), 精度要求不高	离散化方法
需要充分探索, 环境随机性大	随机策略梯度 (PPO)
样本效率重要, 有大量历史数据	确定性策略梯度 (DDPG/TD3/SAC)
控制精度要求高, 需要平滑控制	确定性策略梯度
安全关键应用, 需要稳定训练	随机策略梯度 (PPO/TRPO)
计算资源有限	离散化或 PPO
需要快速原型验证	离散化或 PPO
高维复杂控制 (如人形机器人)	SAC 或 TD3

5.7.3 实际应用建议

网络架构设计

- 输入归一化: 状态输入标准化到 $[-1, 1]$

- **输出缩放:** 动作输出缩放到实际范围
- **激活函数:** 中间层使用 ReLU, 输出层使用 tanh
- **参数初始化:** 最后一层小权重初始化

训练技巧

1. **热身阶段:** 开始时随机探索收集数据
2. **探索衰减:** 训练后期减少探索噪声
3. **学习率衰减:** 训练后期降低学习率
4. **监控指标:**
 - 回合奖励
 - Q 值范围
 - 策略熵
 - 梯度范数
5. **正则化:**
 - 权重衰减
 - 梯度裁剪
 - 批归一化

5.7.4 常见问题与解决方案

表 5.5: 常见问题与解决方案

问题	解决方案
训练不稳定, 奖励震荡	减小学习率, 增加批量大小, 使用目标网络
探索不足, 陷入局部最优	增加探索噪声, 提高熵奖励, 重启探索
过拟合, 泛化能力差	使用 dropout, 数据增强, 早停法
梯度爆炸/消失	梯度裁剪, 合适的初始化, 批归一化
收敛速度慢	调整学习率, 优化网络结构, 并行收集数据
Q 值过高估计	使用双 Q 学习, 延迟更新, 目标策略平滑
探索与利用不平衡	自适应探索策略, 好奇心驱动探索

5.8 前沿发展与未来方向

5.8.1 当前研究趋势

1. **样本效率提升**
 - 模型基强化学习

- 元学习
- 模仿学习

2. 探索策略改进

- 好奇心驱动探索
- 内在动机
- 基于不确定性的探索

3. 多任务与迁移学习

- 共享表示学习
- 渐进式网络
- 蒸馏学习

4. 安全与稳健性

- 约束强化学习
- 鲁棒强化学习
- 安全探索

5. 分布式与并行训练

- 大规模分布式训练
- 异步优化
- 联邦强化学习

5.8.2 重要算法进展

表 5.6: 近年重要连续控制算法

算法	年份	主要贡献
DDPG	2015	深度确定性策略梯度，结合 DQN 和 DPG
PPO	2017	简单高效的策略梯度算法，成为标准基准
TD3	2018	解决 DDPG 的过估计问题，更稳定
SAC	2018	最大熵强化学习，随机策略 + 离线策略
MPO	2018	相对熵策略搜索，更理论化方法
SAC-v2	2019	自动调整温度参数，更易使用
REDQ	2021	随机集成双 Q 学习，高样本效率
DrQ	2021	数据增强强化学习，改进样本效率

第六章 进化计算之起源与遗传算法

6.1 引言：两种哲学思想的碰撞

6.1.1 理性主义与经验主义的对立

在哲学史上，关于人类如何获取知识，存在两种根本对立的观点：理性主义（Rationalism）和经验主义（Empiricism）。

表 6.1: 理性主义与经验主义的对比

理性主义（Rationalism）	经验主义（Empiricism）
代表人物：笛卡尔、斯宾诺莎、莱布尼茨	代表人物：培根、洛克、牛顿
核心思想：真正的知识来源于理性和逻辑推理，而非感官经验。存在不证自明的“公理”或“天赋观念”。	核心思想：知识来源于感官经验。通过观察、实验、归纳总结形成知识。
方法论：从公理出发，通过严密的演绎推理构建知识体系。	方法论：通过观察现象，归纳总结出一般规律。
优点：逻辑严谨，体系严密。	优点：能够产生新知识，适用范围广，是现代科学的基石。
缺点：如果公理错误，整个体系崩溃；适用范围有限。	缺点：归纳法的结论不保证永远正确。
经典名言：我思故我在（笛卡尔）	经典名言：知识就是力量（培根）

6.1.2 数学优化与进化计算的哲学对应

在优化理论中，我们同样可以看到这两种哲学思想的体现：

表 6.2: 数学优化与进化计算的哲学对应

数学优化（对应理性主义）	进化计算（对应经验主义）
依赖于精确的数学模型（类似于“公理”）	不依赖于精确的数学模型
使用严格的数学推导和证明（演绎法）	模仿自然界的“物竞天择，适者生存”（归纳法）
通过梯度下降、牛顿法等一步步找到最优解	通过“选择、交叉、变异”等随机操作，不断尝试和迭代
优点: 一旦模型建立，求解过程严谨，通常能保证找到最优解或近似最优解。	优点: 几乎可以应用于任何可以定义“好坏”（适应度）的优化问题，适用性极广。
缺点: 对问题要求高，很多现实问题难以建立精确的数学模型。	缺点: 无法从数学上证明一定能找到全局最优解，性能边界难以严格保证。

6.1.3 现实世界优化问题的复杂性

现实世界中，许多优化问题具有多个特性，使得传统数学优化方法难以应用：

1. **非线性 (Nonlinear):** 目标函数或约束条件不是线性的
2. **不可导/导数不连续 (Non-differentiable/Discontinuous Gradient):** 目标函数在某些点不可导或导数不连续
3. **高维度 (High dimensionality):** 决策变量数量很多
4. **非凸、大量的局部最优解 (Non-convex, Numerous local optima):** 存在多个局部最优解
5. **黑箱优化 (Black-Box Optimization, BBO):** 目标函数是黑箱，只能通过输入得到输出，无法获取内部信息

黑箱优化问题举例

- **交通流优化:** 目标是最小化平均通勤时间。输入是交通信号灯时序方案。黑箱是通过SUMO等软件运行一次交通流仿真。
- **计算流体力学 (CFD):** 目标是找到阻力最小的飞机翼型。输入是翼型的几何参数。黑箱是运行一次昂贵的CFD仿真。
- **药物/材料发现:** 目标是找到性能最好的分子/合金。输入是化学结构/配方。黑箱是运行分子动力学模拟或真实的物理实验。



图 6.1: 黑箱优化问题示意图：只能通过输入得到输出，无法了解内部结构

6.2 遗传算法的提出

6.2.1 自然进化的启示

世界上最好的解决问题的工具是什么？

1. **人类大脑**: 创造了车轮、纽约和战争... (Douglas Adams) → 神经计算 (Neural Computation)
2. **进化机制**: 创造了人类大脑... (Charles Robert Darwin) → 进化计算 (Evolutionary Computation)

遗传算法 (Genetic Algorithm, GA) 是由美国密歇根大学的 John Holland 教授在 20 世纪 60 年代末到 70 年代初提出的一种概率性群体搜索算法。它以孟德尔的遗传学说和达尔文的生物进化论为基础，通过模拟生物的进化过程来求解优化问题。

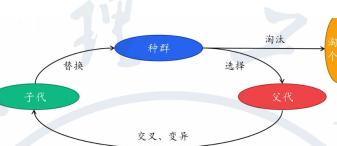


图 6.2: 生物进化过程与遗传算法的对应关系

6.2.2 生物进化与遗传算法的类比

表 6.3: 生物进化过程与遗传算法的对应关系

生物学术语	遗传算法术语	解释
基因 (gene)	决策变量/解的元素	问题的基本构成单元
个体/染色体 (individual/chromosome)	候选解	待求解问题的一个可能解
群体 (population)	候选解集合	一个候选解的有限集合
适应度 (fitness)	适应值/质量评估值	候选解的质量评估值
适者生存 (survival of the fittest)	选择操作	保留较好解，淘汰较差解
交叉 (crossover)	交叉操作	两个个体交换基因信息产生新个体
变异 (mutation)	变异操作	个体基因发生随机变化
进化 (evolution)	迭代优化	种群逐代改进，逼近最优解

6.3 遗传算法的总体流程

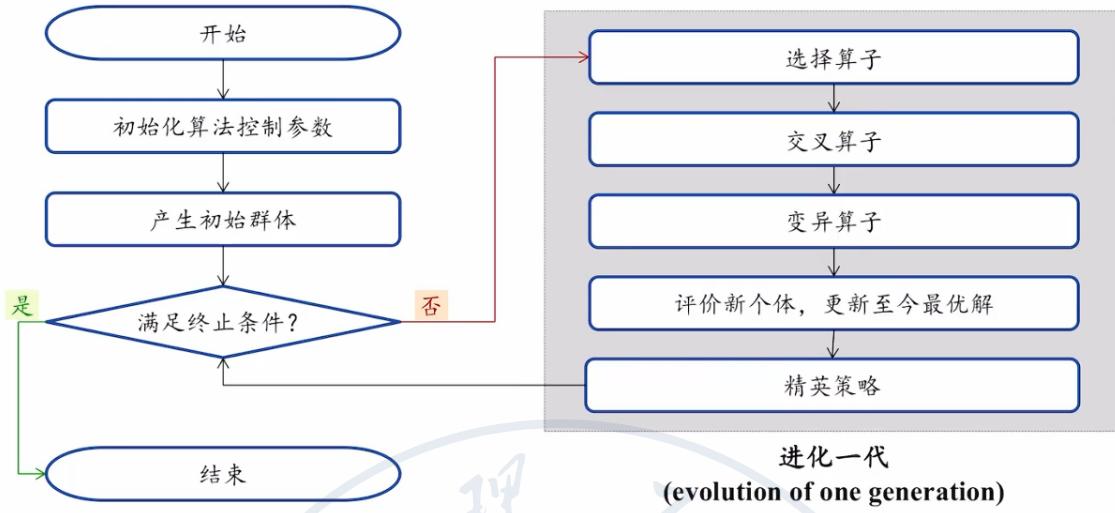


图 6.3: 遗传算法的总体流程

Algorithm 14 遗传算法基本流程

Require: 种群大小 N , 交叉概率 P_c , 变异概率 P_m , 最大迭代次数 G_{\max}

Ensure: 至今最优解

- 1: 随机生成初始种群 P_0 , 大小为 N
 - 2: 评估 P_0 中每个个体的适应度
 - 3: 记录至今最优解
 - 4: **for** $g = 1$ to G_{\max} **do**
 - 5: 选择操作: 从当前种群中选择 N 个个体形成交配池
 - 6: 交叉操作: 以概率 P_c 对交配池中的个体进行交叉, 产生新个体
 - 7: 变异操作: 以概率 P_m 对新个体进行变异
 - 8: 评估新个体的适应度
 - 9: 执行精英策略: 用至今最优解替换当前最差解
 - 10: 更新至今最优解
 - 11: **end for**
 - 12: **return** 至今最优解
-

6.4 遗传算法的关键组成部分

6.4.1 编码策略 (Encoding Strategy)

编码策略是指如何将问题的一个解映射为遗传算法中的个体（染色体）。编码策略的设计强调全面性和唯一性，与算法的性能密切相关。

表 6.4: 常见的编码策略

编码类型	表示方法	适用问题
二进制编码	每个决策变量用二进制数表示	离散优化问题, 特别是 0-1 规划问题
整数编码	每个决策变量用整数表示	离散优化问题, 如调度问题
实数编码	每个决策变量用实数表示	连续优化问题, 如函数优化
排列编码	将有限集合内的元素进行排列	组合优化问题, 如 TSP、调度问题

编码策略的选择原则

1. 合法性: 编码应能表示所有可能的合法解
2. 唯一性: 每个解应有唯一的编码表示
3. 完备性: 编码应包含问题的所有信息
4. 简洁性: 编码应尽量简洁, 减少冗余
5. 可操作性: 编码应便于遗传操作 (选择、交叉、变异)

6.4.2 适应度函数 (Fitness Function)

适应度函数用于评价个体所对应解的优劣, 是遗传算法选择的依据。

适应度函数的设计原则

1. 能直观反映解的质量
2. 能体现解质量的差异
3. 与遗传算子的特性相匹配

约束处理

对于约束优化问题, 常用的方法是使用带惩罚项的适应度函数:

$$f(\mathbf{x}) = \text{fitness}(\mathbf{x}) + \lambda \cdot \text{penalty}(\mathbf{x})$$

其中:

- $\text{fitness}(\mathbf{x})$ 是原始适应度
- $\text{penalty}(\mathbf{x})$ 是惩罚项, 反映解违反约束的程度
- λ 是惩罚系数

惩罚函数的设计应保证: 最好不合法解的适应度值差于最差合法解的适应度值。

6.4.3 选择算子 (Selection Operator)

选择算子体现生物进化过程中的“优胜劣汰”原则，适应度好的个体有较大的机会被选中。

轮盘赌选择 (Roulette Wheel Selection)

轮盘赌选择是最经典的选择算子。每个个体的选择概率与其适应度成正比：

$$P_i = \frac{f(\mathbf{x}_i)}{\sum_{j=1}^N f(\mathbf{x}_j)}$$

其中 $f(\mathbf{x}_i)$ 是个体 \mathbf{x}_i 的适应度。

问题：

1. 对适应度绝对值敏感
2. 出现“超级个体”会垄断种群
3. 无法直接处理负适应度值

改进方案：使用 Softmax 函数计算选择概率：

$$P_i = \frac{\exp(f(\mathbf{x}_i)/T)}{\sum_{j=1}^N \exp(f(\mathbf{x}_j)/T)}$$

其中 T 是温度参数，控制选择压力。

优点：

1. 不受适应度尺度影响（平移不变性）
2. 可调节选择压力（通过温度参数 T ）
3. 天然兼容负适应度值

锦标赛选择 (Tournament Selection)

锦标赛选择是另一种常用的选择算子：

Algorithm 15 锦标赛选择

Require: 种群 P , 种群大小 N , 锦标赛规模 K

Ensure: 选择出的个体

- 1: 初始化空列表 $selected$
 - 2: **while** $|selected| < N$ **do**
 - 3: 从种群中随机选择 K 个个体
 - 4: 从这 K 个个体中选择适应度最高的个体
 - 5: 将该个体加入 $selected$ 列表
 - 6: **end while**
 - 7: **return** $selected$
-

特点：

1. 选择概率与适应度的取值范围无关（只与排序相关）
2. 需要设置参数 K （锦标赛规模）
3. 较差个体被选中的机会较小
4. 好个体可能会在种群中重复出现

6.4.4 交叉算子（Crossover Operator）

交叉算子将两个或多个个体重新组合，期望将优势基因“强强联合”，产生适应度更高的新个体。

单点交叉（One-Point Crossover）

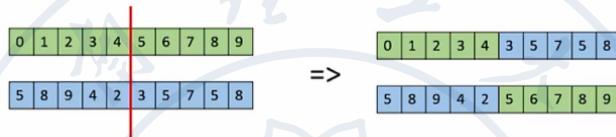


图 6.4: 单点交叉示意图：随机选择一个交叉点，交换两个父代个体交叉点后的部分

对于二进制编码，单点交叉的操作过程：

1. 随机选择一个交叉点 k ($1 \leq k < L$, L 为染色体长度)
2. 父代 1: $A = a_1 a_2 \dots a_k a_{k+1} \dots a_L$
3. 父代 2: $B = b_1 b_2 \dots b_k b_{k+1} \dots b_L$
4. 子代 1: $A' = a_1 a_2 \dots a_k b_{k+1} \dots b_L$
5. 子代 2: $B' = b_1 b_2 \dots b_k a_{k+1} \dots a_L$

两点交叉（Two-Point Crossover）

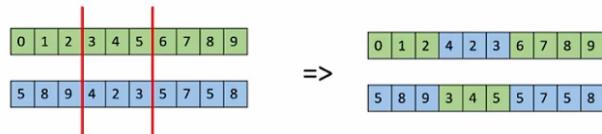


图 6.5: 两点交叉示意图：随机选择两个交叉点，交换两个父代个体交叉点间的部分

对于二进制编码，两点交叉的操作过程：

1. 随机选择两个交叉点 k_1, k_2 ($1 \leq k_1 < k_2 < L$)
2. 父代 1: $A = a_1 \dots a_{k_1} a_{k_1+1} \dots a_{k_2} a_{k_2+1} \dots a_L$
3. 父代 2: $B = b_1 \dots b_{k_1} b_{k_1+1} \dots b_{k_2} b_{k_2+1} \dots b_L$

4. 子代 1: $A' = a_1 \dots a_{k_1} b_{k_1+1} \dots b_{k_2} a_{k_2+1} \dots a_L$
5. 子代 2: $B' = b_1 \dots b_{k_1} a_{k_1+1} \dots a_{k_2} b_{k_2+1} \dots b_L$

算术交叉 (Arithmetic Crossover)

对于实数编码，常用的交叉算子是算术交叉：

$$\begin{aligned}\text{子代 1} &= \alpha \cdot \text{父代 1} + (1 - \alpha) \cdot \text{父代 2} \\ \text{子代 2} &= (1 - \alpha) \cdot \text{父代 1} + \alpha \cdot \text{父代 2}\end{aligned}$$

其中 $\alpha \in [0, 1]$ 是随机数或固定值。

6.4.5 变异算子 (Mutation Operator)

变异算子旨在为算法带来新的基因材料，增加种群的多样性，防止算法过早收敛。

基本变异算子

对于每个基因，以变异概率 P_m 决定是否进行变异：

Algorithm 16 基本变异操作

```

1: for 每个个体  $\mathbf{x} = (x_1, x_2, \dots, x_D)$  do
2:   for 每个基因  $i = 1$  to  $D$  do
3:     生成随机数  $r \sim U(0, 1)$ 
4:     if  $r < P_m$  then
5:       对  $x_i$  实施变异操作
6:     end if
7:   end for
8: end for

```

均匀变异 (Uniform Mutation)

$$x'_i = \text{random}(l_i, u_i)$$

其中 l_i 和 u_i 是基因 x_i 的下界和上界。

高斯变异 (Gaussian Mutation)

$$x'_i = x_i + N(0, \sigma^2)$$

其中 $N(0, \sigma^2)$ 是均值为 0、方差为 σ^2 的正态分布随机数。

6.4.6 精英策略 (Elitist Strategy)

精英策略旨在将优势基因保存在群体中，避免交叉、变异操作导致优势基因流失。

Algorithm 17 精英策略

Require: 当前种群 P_g , 至今最优解 \mathbf{x}_{best}

Ensure: 更新后的种群

1: 找到当前种群中的最差解 $\mathbf{x}_{\text{worst}}$

2: 用至今最优解 \mathbf{x}_{best} 替换 $\mathbf{x}_{\text{worst}}$

3: **return** 更新后的种群

6.5 遗传算法的参数设置

6.5.1 种群大小 (Population Size, N)

在固定的总计算预算（最大函数评估次数 MaxFEs）下：

$$\text{MaxFEs} = \text{种群大小}(N) \times \text{世代代数}(G)$$

两种策略的权衡：

表 6.5: 种群大小设置的两种策略

大种群策略 ($N \uparrow, G \downarrow$)	小种群策略 ($N \downarrow, G \uparrow$)
思路：广度优先，并行探索大量不同的解	思路：深度优先，对少数解进行深度挖掘
优点：多样性好，能覆盖更广的搜索空间，有效避免早熟	优点：收敛速度快，能快速迭代优化
缺点：进化不充分，可能来不及对优良个体进行精细优化	缺点：多样性差，极易陷入局部最优且无法跳出
适用场景：复杂、多峰问题，搜索空间大	适用场景：相对简单、单峰问题，计算资源有限

设置建议：常用范围为 30-200。对于复杂问题，通常需要更大的 N 来保证足够的探索。

6.5.2 交叉概率 (Crossover Probability, P_c)

交叉概率控制着个体间进行基因重组的频率。

- P_c 过高 (如 > 0.95): 可能频繁破坏已有的优良基因组合，导致解的质量不稳定
- P_c 过低 (如 < 0.5): 搜索停滞，产生新解的能力不足，探索效率低下

设置建议：通常设置一个较高的值，以保证算法有足够的探索能力。经验范围：0.5-0.95。

6.5.3 变异概率 (Mutation Probability, P_m)

变异概率控制着个体发生随机变异的概率。

- P_m 过高 (如 > 0.1): 算法退化为随机搜索, 优良基因被持续破坏, 难以收敛
- P_m 过低 (如 < 0.005): 无法有效引入新基因, 一旦陷入局部最优便难以逃脱

设置建议: 必须设置一个很低的值, 它应是一个辅助性的小概率事件。经验范围: 0.005-0.1。

一个常用的经验法则是 P_m 与染色体长度 D 成反比: $P_m \propto 1/D$ 。

6.6 遗传算法的特点

表 6.6: 遗传算法的特点

特点	详细说明
群体性随机搜索	从点集开始, 以点集结束, 通过群体进行搜索
灵活性强	只在设计适应度函数和处理约束时需要先验知识
长于全局搜索	以适应度为导向, 但不排斥接纳较差的新解, 有助于跳出局部最优
隐含并行性	群体中每个个体的操作相似, 易于并行实现
局部搜索能力弱	精度较低, 通常需要与局部搜索算子相结合
实时性差	需要多次迭代, 收敛速度可能较慢
参数敏感	需要设置控制参数且与问题相关性较大

6.7 应用 1: 函数优化 (4D Rastrigin 函数)

6.7.1 问题定义

目标函数: 4D Rastrigin 函数

$$f(x_1, x_2, x_3, x_4) = 40 + \sum_{i=1}^4 [x_i^2 - 10 \cos(2\pi x_i)]$$

约束条件: 每个决策变量 x_i 的取值范围为 $[-5.12, 5.12]$

适应度函数: 由于是求最小值, 适应度与目标函数值成反比

$$\text{Fitness} = \frac{1}{f(\mathbf{x})}$$

6.7.2 实例演示

步骤 1：初始化种群 P_0 及适应度评估

随机生成一个大小为 4 的初始种群，并评估其适应度：

表 6.7: 初始种群 P_0

编号	染色体 (x_1, x_2, x_3, x_4)	目标函数值 $f(\mathbf{x})$	适应度
个体 1	[4.11, -3.22, 2.55, -4.81]	114.31	0.0087
个体 2	[0.10, -0.20, 0.15, -0.05]	3.97	0.2519
个体 3	[-3.11, 3.88, -4.01, 1.23]	108.80	0.0092
个体 4	[1.10, 0.80, -1.20, -0.90]	19.98	0.0500

观察：个体 2 表现最优，其适应度 (0.2519) 远超其他个体。记录至今最优解（精英）：个体 2，其解为 [0.10, -0.20, 0.15, -0.05]。

步骤 2：选择操作（轮盘赌选择）

计算总适应度： $F_{\text{total}} = 0.0087 + 0.2519 + 0.0092 + 0.0500 = 0.3198$

计算每个个体的选择概率和累积概率：

表 6.8: 选择概率计算

编号	适应度	选择概率	累积概率
个体 1	0.0087	0.027	0.027
个体 2	0.2519	0.788	0.815
个体 3	0.0092	0.029	0.844
个体 4	0.0500	0.156	1.000

通过轮盘赌选择，假设选中的个体为：个体 2、个体 4、个体 2、个体 1。

步骤 3：交叉操作（算术交叉）

设定交叉概率 $P_c = 0.9$ 。

1. 配对 1：(个体 2, 个体 4)。生成随机数 $r = 0.85 < 0.9$ ，执行交叉。

设随机数 $\alpha = 0.8$ ：

$$\text{后代 } 1 = 0.8 \cdot \text{个体 } 2 + 0.2 \cdot \text{个体 } 4 = [0.30, 0.00, -0.21, -0.25]$$

$$\text{后代 } 2 = 0.2 \cdot \text{个体 } 2 + 0.8 \cdot \text{个体 } 4 = [0.90, 0.60, -0.93, -0.73]$$

2. 配对 2：(个体 2, 个体 1)。生成随机数 $r = 0.75 < 0.9$ ，执行交叉。

设随机数 $\alpha = 0.9$:

$$\text{后代 } 3 = 0.9 \cdot \text{个体 } 2 + 0.1 \cdot \text{个体 } 1 = [0.50, -0.50, 0.39, -0.53]$$

$$\text{后代 } 4 = 0.1 \cdot \text{个体 } 2 + 0.9 \cdot \text{个体 } 1 = [3.71, -2.92, 2.31, -4.33]$$

步骤 4: 变异操作 (高斯变异)

设定变异概率 $P_m = 0.1$, 变异强度 $\sigma = 0.1$ 。

遍历 4 个后代的共 16 个基因, 假设只有一个随机数小于 P_m , 且该事件发生在后代 2 的第 4 个基因 x_4 上:

$$x'_4 = -0.73 + N(0, 0.1^2) = -0.73 + 0.08 = -0.65$$

最终后代 2': $[0.90, 0.60, -0.93, -0.65]$

步骤 5: 形成临时新种群并评估

表 6.9: 临时新种群

编号	染色体 (x_1, x_2, x_3, x_4)	目标函数值 $f(\mathbf{x})$	适应度
后代 1	$[0.30, 0.00, -0.21, -0.25]$	2.03	0.4926
后代 2'	$[0.90, 0.60, -0.93, -0.65]$	12.01	0.0833
后代 3	$[0.50, -0.50, 0.39, -0.53]$	8.85	0.1130
后代 4	$[3.71, -2.92, 2.31, -4.33]$	100.25	0.0100

步骤 6: 执行精英策略

- 找到当代最差解: 后代 4 (适应度 0.0100)
- 获取至今最优解: 个体 2 (来自 P_0 , 适应度 0.2519)
- 执行替换: 后代 4 被个体 2 替换

步骤 7: 最终新一代种群 P_1

表 6.10: 新一代种群 P_1

编号	染色体 (x_1, x_2, x_3, x_4)	来源	目标函数值 $f(\mathbf{x})$	适应度
新个体 1	$[0.30, 0.00, -0.21, -0.25]$	后代 1	2.03	0.4926
新个体 2	$[0.90, 0.60, -0.93, -0.65]$	后代 2'	12.01	0.0833
新个体 3	$[0.50, -0.50, 0.39, -0.53]$	后代 3	8.85	0.1130
新个体 4	$[0.10, -0.20, 0.15, -0.05]$	精英替换	3.97	0.2519

观察: 新一代种群的最优适应度从 0.2519 提升到 0.4926, 说明算法在向更好的方向进化。

6.8 应用 2：旅行商问题（Traveling Salesman Problem, TSP）

6.8.1 问题定义

旅行商问题：给定一个城市集合，找到一条访问每个城市一次且仅一次，并最终返回出发点的最短回路。

- 城市可以代表：客户、传感器、DNA 片段、焊接点等
- 距离可以表示：时间、成本、相似性度量等

数学形式化

设城市集合为 $C = \{1, 2, \dots, n\}$ ，距离矩阵为 $D = (d_{ij})_{n \times n}$ 。目标是找到城市的一个排列 $\pi = (\pi(1), \pi(2), \dots, \pi(n))$ ，最小化总路径长度：

$$\text{Minimize } Z = \sum_{i=1}^{n-1} d_{\pi(i), \pi(i+1)} + d_{\pi(n), \pi(1)}$$

实例数据

距离矩阵：

$$D = \begin{bmatrix} \infty & 20 & 42 & 35 \\ 20 & \infty & 30 & 34 \\ 42 & 30 & \infty & 12 \\ 35 & 34 & 12 & \infty \end{bmatrix}$$

6.8.2 编码策略：排列编码（Permutation Encoding）

对于 TSP，最有效的编码方式是排列编码：

- 染色体是一个长度为 n 的数组，包含从 1 到 n 的所有城市 ID，每个 ID 只出现一次
- 数组的顺序代表旅行的顺序
- 示例：染色体 $[1, 4, 3, 2]$ 代表路径 $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$

这种编码方式天然满足 TSP“访问每个城市一次且仅一次”的核心约束。

6.8.3 遗传算子设计

种群初始化

1. **随机初始化：**多次随机打乱一个包含所有城市 ID 的数组
 - 创建有序数组，如 $[1, 2, 3, 4]$
 - 对该数组执行随机置换算法，得到随机排列，如 $[3, 1, 4, 2]$

- 重复此过程，直到生成足够数量的初始个体

2. 贪心初始化：提高初始种群的平均质量

- 从一个随机城市出发
- 每一步都选择前往所有未访问城市中最近的一个
- 直到所有城市都被访问

示例（从城市 1 出发）：

- 路径：[1]
- 下一站（距 1 最近）：2 → 路径：[1, 2]
- 下一站（距 2 最近且未访问）：3 → 路径：[1, 2, 3]
- 下一站（只剩 4）：4 → 路径：[1, 2, 3, 4]
- 总距离： $d_{12} + d_{23} + d_{34} + d_{41} = 20 + 30 + 12 + 35 = 97$

最终种群可以由 1 个贪心解和多个随机解组成，以兼顾开局优势和种群多样性。

适应度与锦标赛选择

- 适应度：**在 TSP 中，目标是最小化距离，因此个体的“适应性”与距离成反比
- 锦标赛选择：**这种选择策略只关心个体间的优劣排序，不需要将距离转换为适应度值

锦标赛选择流程（以锦标赛大小 $K = 2$ 为例）：

- 从当前种群中随机选出 2 个个体
- 直接比较这两个个体的总旅行距离
- 总距离更短的那个个体“获胜”，被复制到下一代的交配池中
- 重复以上步骤，直到交配池满员

维持合法性的交叉算子：部分匹配交叉（PMX）

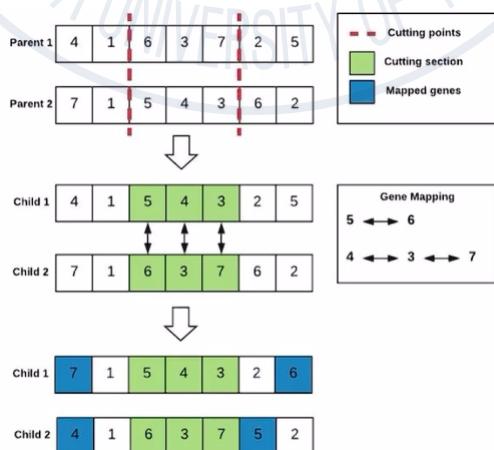


图 6.6: 部分匹配交叉（PMX）示意图

PMX 操作步骤：

1. 随机选择两个交叉点
2. 交换两个父代个体交叉点间的片段
3. 根据映射关系修复冲突，确保每个城市只出现一次

示例：

- 父代 1: [1, 2, 3, 4, 5, 6, 7, 8, 9]
- 父代 2: [4, 5, 2, 1, 8, 7, 6, 9, 3]
- 交叉点: 3 和 6
- 交换中间片段：
 - 子代 1 中间: [1, 8, 7, 6]
 - 子代 2 中间: [3, 4, 5, 6]
- 修复冲突，得到合法子代

维持合法性的变异算子

1. 交换变异 (Swap Mutation): 随机选择染色体上的两个位置，并交换这两个位置上的城市

1 2 3 4 5 6 7 8 9 0 → 1 6 3 4 5 2 7 8 9 0

2. 乱序变异 (Scramble Mutation): 随机选择染色体的一个子集，然后将这个子集中的城市进行随机打乱

0 1 2 2 3 4 5 6 7 8 9 → 0 1 3 6 2 4 5 7 8 9

3. 倒置变异 (Inversion Mutation): 随机选择染色体中的一个子段，然后将其完全逆序

0 1 2 3 4 5 6 7 8 9 → 0 1 6 5 4 3 2 7 8 9

局部搜索增强：2-Opt 算法

为了结合 GA 的全局探索能力和局部搜索的精细挖掘能力，可以引入 2-Opt 算法：

Algorithm 18 2-Opt 局部搜索

Require: 当前路径 π

Ensure: 改进后的路径

```
1: 设置  $improved \leftarrow True$ 
2: while  $improved$  do
3:    $improved \leftarrow False$ 
4:   for  $i = 1$  to  $n - 2$  do
5:     for  $j = i + 2$  to  $n$  do
6:       计算当前边  $(i, i + 1)$  和  $(j, j + 1)$  的长度  $d_1$ 
7:       计算交换后边  $(i, j)$  和  $(i + 1, j + 1)$  的长度  $d_2$ 
8:       if  $d_2 < d_1$  then
9:         交换路径中  $i + 1$  到  $j$  之间的城市顺序
10:         $improved \leftarrow True$ 
11:      end if
12:    end for
13:  end for
14: end while
15: return 改进后的路径
```

2-Opt 操作：随机移除两条不相交的边，用唯一可能的方式重连，若路径变短则接受新路径。

6.8.4 GA 求解 TSP 的整体流程

Algorithm 19 GA for TSP with Local Search

Require: 城市数量 n , 距离矩阵 D , 种群大小 N , 最大迭代次数 G_{\max}

Ensure: 至今最优路径

- 1: 初始化种群 P_0 (包含随机解和贪心解)
 - 2: 评估 P_0 中每个个体的路径长度
 - 3: 记录至今最优解
 - 4: **for** $g = 1$ to G_{\max} **do**
 - 5: 锦标赛选择: 从当前种群中选择 N 个个体形成交配池
 - 6: PMX 交叉: 以概率 P_c 对交配池中的个体进行交叉
 - 7: 变异操作: 以概率 P_m 对新个体进行变异 (交换、倒置等)
 - 8: 评估新个体的路径长度
 - 9: 执行精英策略: 用至今最优解替换当前最差解
 - 10: 局部搜索: 对本代最优解执行 2-Opt 优化
 - 11: 更新至今最优解
 - 12: **end for**
 - 13: **return** 至今最优解
-

6.9 总结

遗传算法作为一种基于自然选择和遗传机制的优化算法, 具有以下特点:

1. **哲学基础:** 体现了经验主义的哲学思想, 通过“尝试-评估-选择”的迭代过程逼近最优解
2. **算法特点:**
 - 群体性搜索, 避免陷入局部最优
 - 适用于黑箱优化、非凸、不可导等复杂问题
 - 灵活性强, 只需定义适应度函数
 - 隐含并行性, 易于并行实现
3. **关键组件:**
 - 编码策略: 将问题解映射为染色体
 - 适应度函数: 评价解的质量
 - 遗传算子: 选择、交叉、变异
 - 精英策略: 保留历史最优解
4. **参数设置:**
 - 种群大小 N : 权衡探索广度与深度
 - 交叉概率 P_c : 控制基因重组频率
 - 变异概率 P_m : 控制引入新基因的概率

5. 应用领域:

- 函数优化: 连续、离散、混合优化问题
- 组合优化: TSP、调度、布局等问题
- 机器学习: 特征选择、参数调优
- 工程设计: 结构优化、电路设计等

6. 改进方向:

- 自适应参数调整
- 多种群协同进化
- 与局部搜索结合 (模因算法)
- 并行化实现

遗传算法虽然不能保证找到全局最优解,但在许多复杂优化问题上表现出了良好的性能。它的成功验证了自然进化机制作为一种优化策略的有效性,也为解决复杂优化问题提供了新的思路和方法。



第七章 进化计算之连续优化三剑客

7.1 引言：复杂优化问题的挑战与进化计算的意义

7.1.1 优化问题的本质与分类

优化是数学和工程领域的核心问题之一，其本质是在给定的约束条件下，寻找使目标函数达到最优（最大或最小）的决策变量值。优化问题可以形式化地表示为：

$$\begin{aligned} \min_{x \in \mathcal{X}} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \quad i = 1, \dots, m \\ & h_j(x) = 0, \quad j = 1, \dots, p \end{aligned}$$

其中， x 是决策变量， \mathcal{X} 是决策空间， $f(x)$ 是目标函数， $g_i(x)$ 和 $h_j(x)$ 分别是不等式约束和等式约束。

根据决策变量的性质，优化问题可分为：

- **连续优化：** 决策变量在连续空间取值
- **离散优化：** 决策变量在离散集合取值
- **混合整数优化：** 同时包含连续和离散变量

根据目标函数和约束的性质，优化问题可分为：

- **线性规划：** 目标函数和约束均为线性
- **非线性规划：** 目标函数或约束至少有一个非线性
- **凸优化：** 目标函数为凸函数，约束为凸集
- **非凸优化：** 目标函数或约束非凸

7.1.2 传统优化方法的局限性

传统优化方法如梯度下降法、牛顿法、内点法等在解决凸优化、光滑问题时表现出色，但在面对现实世界中的复杂优化问题时，往往面临以下挑战：

表 7.1: 传统优化方法的局限性

挑战类型	具体表现与影响
非凸性	目标函数存在多个局部最优解，传统方法容易陷入局部最优
不可微性	目标函数或约束不可导，无法使用基于梯度的方法
高维度	决策变量数量多，搜索空间呈指数增长，出现“维度灾难”
计算昂贵	目标函数评估成本高（如一次仿真需要数小时）
黑箱问题	目标函数形式未知，仅能通过输入输出进行评估
多峰性	存在多个局部最优解，需要全局搜索能力
噪声	目标函数值包含随机噪声，影响梯度估计

7.1.3 进化计算的优势与哲学基础

进化计算 (Evolutionary Computation, EC) 是一类受生物进化过程启发的随机优化算法。与传统的基于梯度的方法不同，进化计算具有以下优势：

- **无需梯度信息**: 仅需目标函数值，不要求可微性
- **全局搜索能力**: 通过种群多样性避免陷入局部最优
- **鲁棒性强**: 对问题形式、噪声等不敏感
- **并行性**: 种群中的个体可以并行评估
- **通用性**: 适用于连续、离散、混合优化问题

从哲学角度看，进化计算体现了“经验主义”的思想：不依赖于问题的精确数学模型，而是通过“试错-选择”的迭代过程，从经验中学习，逐步逼近最优解。这与基于“理性主义”的传统优化方法形成鲜明对比。

7.2 适应度地形图：理解优化问题的几何视角

7.2.1 适应度地形图的基本概念

适应度地形图 (Fitness Landscape) 是理解优化问题复杂性的重要工具。它将解空间映射为一个地形表面，其中：

- **位置**: 表示一个候选解
- **高度**: 表示该解的适应度（目标函数值）
- **地形特征**: 反映了问题的搜索特性

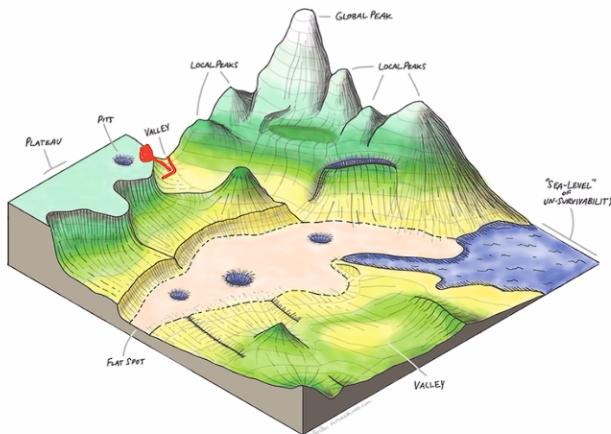


图 7.1: 适应度地形图的三维可视化: 将优化问题转化为在地形中寻找最高峰的问题

7.2.2 地形特征对优化难度的影响

局部最优与全局最优

定义 7.2.1 (局部最优解). 对于最小化问题, 点 x^* 是局部最优解, 如果存在 $\delta > 0$, 使得对于所有满足 $\|x - x^*\| < \delta$ 的 x , 都有 $f(x^*) \leq f(x)$ 。

定义 7.2.2 (全局最优解). 对于最小化问题, 点 x^{**} 是全局最优解, 如果对于所有 $x \in \mathcal{X}$, 都有 $f(x^{**}) \leq f(x)$ 。

地形中可能包含多个局部最优点 (山峰), 但只有一个全局最优点 (最高峰)。局部最优点的数量越多, 问题越难求解。

早熟收敛现象

早熟收敛 (Premature Convergence) 是指优化算法过早陷入局部最优, 无法找到全局最优的现象。这通常发生在:

- 种群多样性不足
- 选择压力过大
- 变异率过低

停滞区: 高原与平坦区

定义 7.2.3 (高原区域). 在适应度地形图中, 如果存在一个区域 R , 对于任意 $x, y \in R$, 都有 $|f(x) - f(y)| < \epsilon$ (ϵ 很小), 则称 R 为高原区域。

高原区域对优化算法的挑战:

- 梯度信息几乎为零
- 缺乏搜索方向指引
- 算法可能随机游走, 效率低下

深谷与悬崖

深谷（窄而深的山谷）和悬崖（适应度急剧变化）会阻碍算法的搜索：

- 深谷可能导致算法陷入
- 悬崖可能导致算法“跳过”有希望的区域

7.2.3 适应度地形图的量化分析

崎岖度 (Ruggedness)

崎岖度量化了地形的波动程度。可以通过自相关函数计算：

$$\rho(k) = \frac{\mathbb{E}[(f(x_t) - \mu)(f(x_{t+k}) - \mu)]}{\sigma^2}$$

其中， x_t 是随机游走路径上的点， μ 和 σ^2 是适应度的均值和方差。自相关长度 τ 定义为 $\rho(\tau) = 1/e$ ， τ 越小，地形越崎岖。

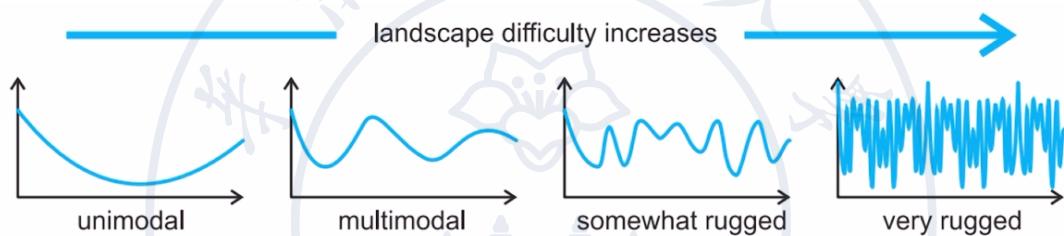


图 7.2: 不同崎岖度的适应度地形：(a) 平滑地形，(b) 中等崎岖地形，(c) 高度崎岖地形

中立性 (Neutrality)

中立性指存在大量适应度相同或相近的解。中立网络 (Neutral Network) 是适应度相等的解构成的连通集合。

中立性的影响：

- 允许算法在不降低适应度的情况下探索
- 可能延缓收敛速度
- 有助于维持种群多样性

欺骗性 (Deceptiveness)

欺骗性地形会误导搜索方向，使算法远离全局最优。形式化定义为：

定义 7.2.4 (欺骗性问题). 如果对于全局最优解 x^{**} 和某个局部最优解 x^* ，在 x^{**} 的某些邻域内，平均适应度低于在 x^* 的相应邻域内的平均适应度，则称该问题是欺骗性的。

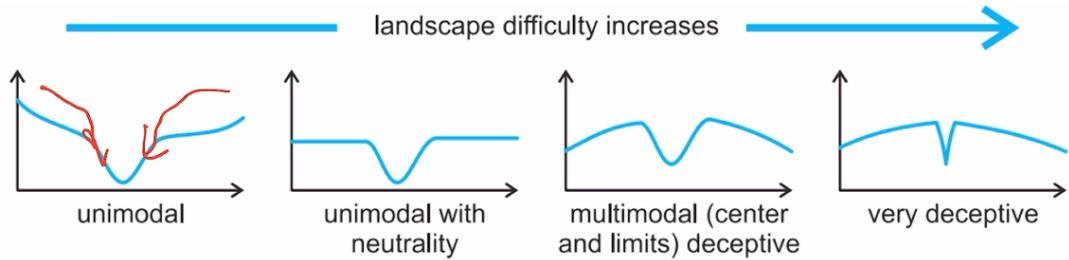


图 7.3: 欺骗性地形示意图：局部最优区域（右侧）的平均适应度高于全局最优区域（左侧），容易误导搜索方向

上位性 (Epistasis)

上位性指基因（决策变量）间的相互作用。在高上位性问题中，一个变量的最优值依赖于其他变量的取值。

上位性的数学表达：

$$\text{Epistasis} = \frac{\text{Var}(f) - \sum_{i=1}^n \text{Var}(f_i)}{\text{Var}(f)}$$

其中， f_i 是仅改变第 i 个变量时的适应度变化。

上位性对算法设计的影响：

- 简单突变算子效果差
- 需要能够捕获变量交互的算子
- 增加问题难度

维度灾难 (Curse of Dimensionality)

随着问题维度 d 的增加：

- 搜索空间体积呈指数增长： $V \propto r^d$
- 采样密度急剧下降： n 个点在 d 维空间中的平均距离为 $O(n^{-1/d})$
- 距离集中现象：高维空间中任意两点间的距离趋于相似

7.2.4 探索与开发的平衡

探索 (Exploration) 与开发 (Exploitation) 的平衡是进化计算的核心问题：

定义 7.2.5 (探索与开发). • 探索：在搜索空间中广泛搜索，发现新的有希望区域

- 开发：在已知的有希望区域内精细搜索，提高解的质量

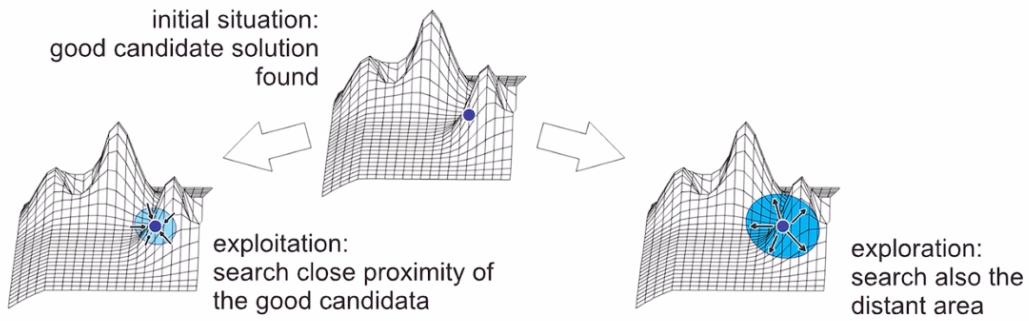


图 7.4: 探索与开发的权衡: 过度探索导致收敛慢, 过度开发导致早熟收敛

探索与开发的数学形式化

设 P_t 为第 t 代的种群, S 为选择算子, V 为变异算子, C 为交叉算子, 则进化算法可表示为:

$$P_{t+1} = S(V(C(P_t)))$$

探索程度可用种群多样性度量:

$$\text{Diversity}(P) = \frac{1}{N(N-1)} \sum_{i=1}^N \sum_{j \neq i}^N d(x_i, x_j)$$

其中 $d(\cdot, \cdot)$ 是距离度量。

开发程度可用种群平均适应度的改善度量:

$$\text{Exploitation}(t) = \frac{f_{\text{avg}}(t) - f_{\text{avg}}(t-1)}{f_{\text{avg}}(t-1)}$$

平衡策略

- **自适应参数调整:** 根据搜索进度动态调整探索与开发
- **多种群策略:** 不同种群侧重不同的探索开发平衡
- **混合算法:** 结合全局搜索和局部搜索算法
- **记忆机制:** 保存历史信息指导搜索方向

7.3 差分进化算法: 基于向量差分的全局优化器

7.3.1 差分进化的历史背景与基本原理

差分进化 (Differential Evolution, DE) 由 Rainer Storn 和 Kenneth Price 于 1995 年提出, 是一种简单而高效的连续优化算法。DE 的核心思想是利用种群中个体间的差异来生成新个体。

DE 的基本流程

DE 算法遵循进化计算的一般框架，但具有独特的变异机制：

Algorithm 20 标准差分进化算法

Require: 目标函数 $f : \mathbb{R}^D \rightarrow \mathbb{R}$, 搜索边界 $[x_{\min}, x_{\max}]$

Ensure: 最优解 x_{best} 和最优值 f_{best}

- 1: 初始化参数：种群大小 NP , 缩放因子 F , 交叉概率 CR
 - 2: 随机初始化种群 $P_0 = \{x_{i,0} | i = 1, \dots, NP\}$, 其中 $x_{i,0} \sim U(x_{\min}, x_{\max})$
 - 3: 评估初始种群适应度 $f(x_{i,0})$
 - 4: 记录最优个体 $x_{\text{best},0}$ 和最优值 $f_{\text{best},0}$
 - 5: **for** $g = 1$ to G_{\max} **do**
 - 6: **for** $i = 1$ to NP **do**
 - 7: **变异**: 根据变异策略生成变异向量 $v_{i,g}$
 - 8: **交叉**: 生成试验向量 $u_{i,g}$: $u_{j,i,g} = \begin{cases} v_{j,i,g}, & \text{if } \text{rand} \leq CR \text{ or } j = j_{\text{rand}} \\ x_{j,i,g}, & \text{otherwise} \end{cases}$
 - 9: **边界处理**: 修复越界的试验向量
 - 10: **选择**: $x_{i,g+1} = \begin{cases} u_{i,g}, & \text{if } f(u_{i,g}) \leq f(x_{i,g}) \\ x_{i,g}, & \text{otherwise} \end{cases}$
 - 11: **end for**
 - 12: 更新最优解 $x_{\text{best},g+1}$ 和 $f_{\text{best},g+1}$
 - 13: **end for**
 - 14: **return** $x_{\text{best}}, f_{\text{best}}$
-

7.3.2 DE 的变异策略详解

变异是 DE 算法的核心，通过差分向量引导搜索方向。常见的变异策略有：

DE/rand/1

最基本的变异策略，使用三个随机个体：

$$v_{i,g} = x_{r_1,g} + F \cdot (x_{r_2,g} - x_{r_3,g})$$

其中 r_1, r_2, r_3 是互不相同的随机索引，且 $r_i \neq i$ 。

DE/best/1

利用当前最优个体引导搜索：

$$v_{i,g} = x_{\text{best},g} + F \cdot (x_{r_1,g} - x_{r_2,g})$$

DE/current-to-best/1

结合当前个体和最优个体的信息：

$$v_{i,g} = x_{i,g} + F \cdot (x_{\text{best},g} - x_{i,g}) + F \cdot (x_{r_1,g} - x_{r_2,g})$$

DE/rand/2 和 DE/best/2

使用两个差分向量增强探索能力：

$$\text{DE/rand/2 : } v_{i,g} = x_{r_1,g} + F \cdot (x_{r_2,g} - x_{r_3,g}) + F \cdot (x_{r_4,g} - x_{r_5,g})$$

$$\text{DE/best/2 : } v_{i,g} = x_{\text{best},g} + F \cdot (x_{r_1,g} - x_{r_2,g}) + F \cdot (x_{r_3,g} - x_{r_4,g})$$

变异策略的选择与适应度地形的匹配

表 7.2: 不同变异策略的特点与适用场景

策略	特点	适用场景
DE/rand/1	探索能力强，收敛慢	多峰问题，全局搜索
DE/best/1	开发能力强，易早熟	单峰问题，快速收敛
DE/current-to-best/1	平衡探索与开发	一般问题
DE/rand/2	强探索，避免早熟	复杂多峰问题
DE/best/2	强开发，快速收敛	相对简单问题

7.3.3 DE 的交叉算子

交叉算子将变异向量与原向量结合，产生试验向量。

二项交叉 (Binomial Crossover)

最常用的交叉方式，每个维度独立决定来自变异向量还是原向量：

$$u_{j,i,g} = \begin{cases} v_{j,i,g}, & \text{if } \text{rand}_j(0, 1) \leq CR \text{ or } j = j_{\text{rand}} \\ x_{j,i,g}, & \text{otherwise} \end{cases}$$

其中 j_{rand} 是随机选择的维度，确保试验向量至少有一维来自变异向量。

指数交叉 (Exponential Crossover)

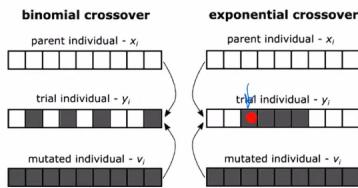


图 7.5: 指数交叉示意图: 连续选择来自变异向量的维度, 直到随机数大于 CR

指数交叉的算法步骤:

1. 随机选择起始位置 n
2. 设置 $L = 0$
3. 当 $\text{rand}(0, 1) \leq CR$ 且 $L < D$ 时:
 - $u_{n,i,g} = v_{n,i,g}$
 - $n = (n + 1) \bmod D$
 - $L = L + 1$
4. 剩余维度从原向量复制

7.3.4 DE 的选择算子与精英保留

DE 采用贪婪选择策略:

$$x_{i,g+1} = \begin{cases} u_{i,g}, & \text{if } f(u_{i,g}) \leq f(x_{i,g}) \\ x_{i,g}, & \text{otherwise} \end{cases}$$

这种选择策略的优缺点:

- **优点:** 简单高效, 保证种群质量不下降
- **缺点:** 可能过于贪婪, 丢失多样性

7.3.5 DE 的参数设置与调优

种群大小 NP

- 通常设置为 $5D$ 到 $10D$, 其中 D 是问题维度
- 大种群: 探索能力强, 但计算成本高
- 小种群: 收敛快, 但易早熟

缩放因子 F

控制差分向量的幅度:

- 通常 $F \in [0.4, 1.0]$

- 大 F : 探索能力强, 但可能振荡
 - 小 F : 开发能力强, 但易陷入局部最优
- 经验公式: $F = 0.5 \times (1 + \text{rand}(0, 1))$

交叉概率 CR

控制来自变异向量的基因比例:

- 通常 $CR \in [0.1, 0.9]$
- 大 CR : 更多新基因, 探索能力强
- 小 CR : 更多原基因, 开发能力强

自适应参数调整策略

1. 模糊自适应 DE: 根据搜索状态动态调整参数
2. 自适应性 DE: 每个个体有自己的参数, 优秀个体的参数得以保留
3. 基于成功历史的自适应: 记录成功变异的参数, 指导新参数生成

7.3.6 DE 的改进算法

SaDE: 自适应差分进化

SaDE 为每个个体分配独立的参数, 并通过学习历史成功参数来调整:

$$F_i \sim \mathcal{N}(0.5, 0.3)$$

$$CR_i \sim \mathcal{N}(CR_m, 0.1)$$

其中 CR_m 是历史成功 CR 值的中位数。

JADE: 带外部存档的自适应 DE

JADE 引入两个改进:

1. 使用柯西分布生成 F , 增强探索能力
2. 维护外部存档保存历史信息

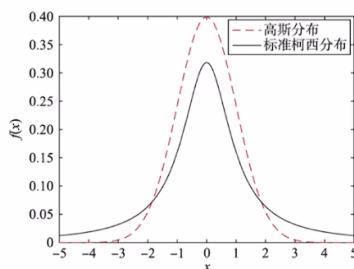


图 7.6: 高斯分布与标准柯西分布对比: 柯西分布具有更重的尾部, 能产生更大的变异步长

SHADE 和 L-SHADE

SHADE 在 JADE 基础上引入历史记忆库，L-SHADE 进一步加入线性种群缩减策略：

$$NP_{g+1} = \text{round} \left(NP_{\min} + (NP_{\max} - NP_{\min}) \times \frac{\text{MaxFEs} - \text{FEs}}{\text{MaxFEs}} \right)$$

7.3.7 DE 的收敛性分析

虽然 DE 缺乏严格的收敛性证明，但可以通过马尔可夫链分析其性质。设 P_g 为第 g 代种群，DE 的更新可看作一个马尔可夫过程：

$$P_{g+1} = T(P_g)$$

其中 T 是转移算子。

在适当条件下，DE 算法以概率 1 收敛到全局最优：

$$\lim_{g \rightarrow \infty} P(f(x_{\text{best},g}) = f^*) = 1$$

7.3.8 DE 的优缺点总结

优点

- 结构简单，易于实现
- 参数少，调节相对容易
- 全局搜索能力强
- 对旋转不变性问题有一定鲁棒性
- 无需梯度信息

缺点

- 对高维问题可能收敛慢
- 参数设置对性能影响大
- 缺乏严格的收敛性证明
- 对离散问题处理不佳

7.4 粒子群优化算法：模拟群体智能的优化器

7.4.1 粒子群优化的生物学基础

粒子群优化（Particle Swarm Optimization, PSO）由 Kennedy 和 Eberhart 于 1995 年提出，灵感来源于鸟群觅食行为。鸟群在寻找食物时，每只鸟会根据自身经验和群体经验调整飞行方向。

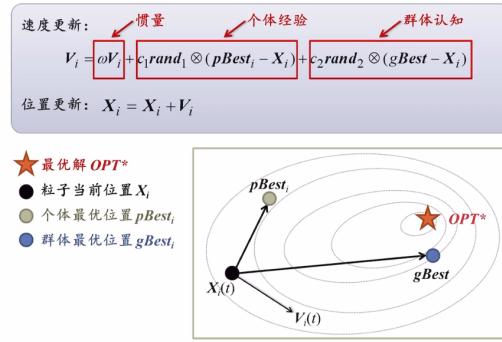


图 7.7: PSO 算法示意图: 粒子向个体历史最优和群体历史最优的加权方向移动

7.4.2 标准 PSO 算法

粒子表示与初始化

每个粒子 i 在 D 维空间中的状态表示为:

- 位置: $x_i = (x_{i1}, x_{i2}, \dots, x_{iD})$
- 速度: $v_i = (v_{i1}, v_{i2}, \dots, v_{iD})$
- 个体历史最优位置: $p_i = (p_{i1}, p_{i2}, \dots, p_{iD})$
- 个体历史最优值: $p_{best,i}$

群体历史最优位置记为 $g = (g_1, g_2, \dots, g_D)$, 对应值为 g_{best} 。

速度与位置更新公式

标准 PSO 的更新公式为:

$$v_{id}^{t+1} = wv_{id}^t + c_1r_1(p_{id}^t - x_{id}^t) + c_2r_2(g_d^t - x_{id}^t) \quad (7.1)$$

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1} \quad (7.2)$$

其中:

- w : 惯性权重
- c_1, c_2 : 加速系数
- r_1, r_2 : $[0, 1]$ 均匀分布的随机数

算法流程

Algorithm 21 标准粒子群优化算法

Require: 目标函数 $f(x)$, 种群大小 N , 最大迭代次数 T_{\max}

Ensure: 全局最优解 g 和最优值 g_{best}

```

1: 初始化粒子位置  $x_i^0$  和速度  $v_i^0$ ,  $i = 1, \dots, N$ 
2: 初始化  $p_i^0 = x_i^0$ ,  $p_{\text{best},i}^0 = f(x_i^0)$ 
3: 初始化  $g^0 = \arg \min_i p_{\text{best},i}^0$ ,  $g_{\text{best}}^0 = \min_i p_{\text{best},i}^0$ 
4: for  $t = 0$  to  $T_{\max} - 1$  do
5:   for  $i = 1$  to  $N$  do
6:     for  $d = 1$  to  $D$  do
7:       更新速度:  $v_{id}^{t+1} = wv_{id}^t + c_1r_1(p_{id}^t - x_{id}^t) + c_2r_2(g_d^t - x_{id}^t)$ 
8:       限制速度:  $v_{id}^{t+1} = \min(\max(v_{id}^{t+1}, -v_{\max}), v_{\max})$ 
9:       更新位置:  $x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1}$ 
10:    end for
11:    评估适应度:  $f_i^{t+1} = f(x_i^{t+1})$ 
12:    if  $f_i^{t+1} < p_{\text{best},i}^t$  then
13:      更新个体最优:  $p_i^{t+1} = x_i^{t+1}$ ,  $p_{\text{best},i}^{t+1} = f_i^{t+1}$ 
14:    else
15:       $p_i^{t+1} = p_i^t$ ,  $p_{\text{best},i}^{t+1} = p_{\text{best},i}^t$ 
16:    end if
17:  end for
18:  更新全局最优:  $g^{t+1} = \arg \min_i p_{\text{best},i}^{t+1}$ ,  $g_{\text{best}}^{t+1} = \min_i p_{\text{best},i}^{t+1}$ 
19: end for
20: return  $g^{T_{\max}}$ ,  $g_{\text{best}}^{T_{\max}}$ 

```

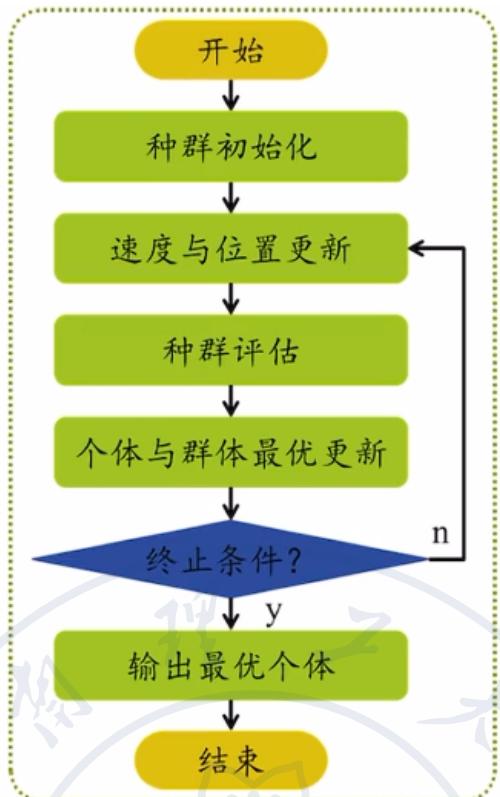


图 7.8: PSO 速度更新示意图: 粒子向个体历史最优和群体历史最优的加权组合方向移动

7.4.3 PSO 的参数分析与设置

惯性权重 w

惯性权重控制粒子保持原速度的程度:

- 大 w : 探索能力强, 适合全局搜索
 - 小 w : 开发能力强, 适合局部搜索
- 常用策略: 线性递减惯性权重

$$w(t) = w_{\max} - (w_{\max} - w_{\min}) \times \frac{t}{T_{\max}}$$

通常 $w_{\max} = 0.9$, $w_{\min} = 0.4$ 。

加速系数 c_1, c_2

加速系数控制个体认知和社会认知的权重:

- c_1 : 个体学习因子, 控制向个体历史最优的学习
- c_2 : 社会学习因子, 控制向群体历史最优的学习

经验设置: $c_1 = c_2 = 2.0$

最大速度 v_{\max}

最大速度限制粒子的移动范围:

$$v_{\max} = \delta \times (x_{\max} - x_{\min})$$

其中 $\delta \in [0.1, 0.5]$ 。

种群大小 N

通常 $N = 20 \sim 50$, 复杂问题需要更大种群。

7.4.4 PSO 的拓扑结构

拓扑结构定义粒子间的信息交流方式, 影响算法的探索开发平衡。

常见拓扑结构

1. 全局拓扑 (星型拓扑): 所有粒子与全局最优粒子连接
 - 收敛快, 但易早熟
2. 环形拓扑: 每个粒子只与左右邻居连接
 - 收敛慢, 但多样性好
3. 冯·诺依曼拓扑: 粒子排列在网格上, 与上下左右邻居连接
 - 平衡收敛与多样性
4. 小世界网络: 具有高集聚系数和短平均路径长度

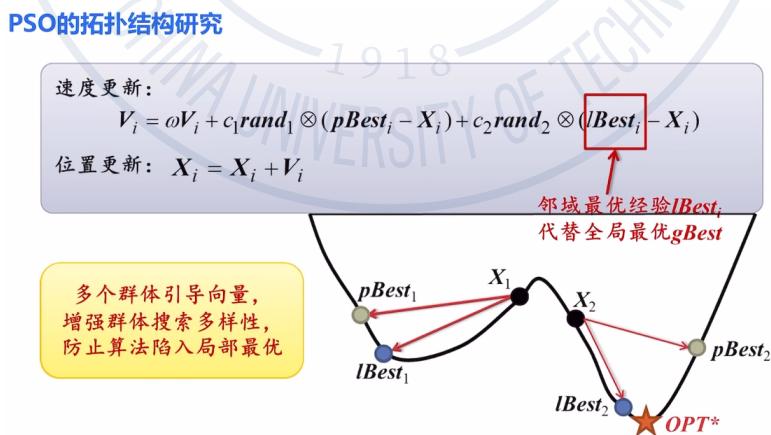


图 7.9: PSO 的不同拓扑结构: (a) 全局拓扑, (b) 环形拓扑, (c) 冯·诺依曼拓扑, (d) 小世界网络

小世界网络的构建

小世界网络结合了规则网络的高集聚性和随机网络的短路径特性:

Algorithm 22 小世界网络构建算法 (Watts-Strogatz 模型)

Require: 节点数 N , 初始邻居数 K , 重连概率 p

Ensure: 网络邻接矩阵 A

```

1: 构建环形网络: 每个节点连接  $K/2$  个左侧邻居和  $K/2$  个右侧邻居
2: for  $i = 1$  to  $N$  do
3:   for 每个连接  $(i, j)$ , 其中  $j > i$  do
4:     if  $\text{rand}(0, 1) < p$  then
5:       断开连接  $(i, j)$ 
6:       随机选择节点  $k \neq i$ , 且不与  $i$  连接
7:       建立新连接  $(i, k)$ 
8:     end if
9:   end for
10: end for

```

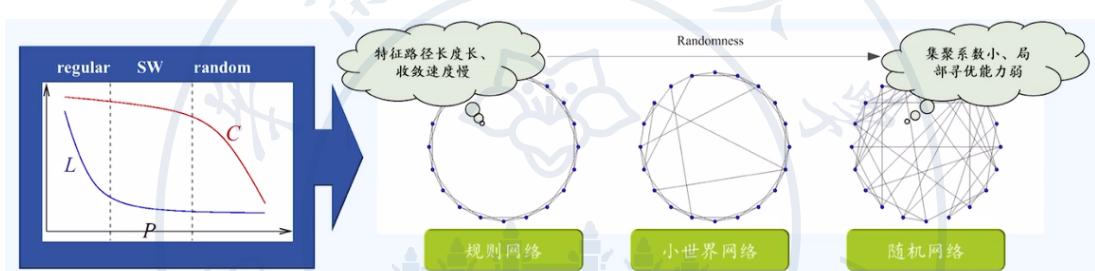


图 7.10: 小世界网络拓扑的优势: 结合局部紧密连接和全局短路径, 有利于信息传播和多样性保持

7.4.5 PSO 的变体算法

标准 PSO 的改进

1. 带收缩因子的 PSO:

$$v_{id}^{t+1} = \chi[v_{id}^t + c_1 r_1(p_{id}^t - x_{id}^t) + c_2 r_2(g_d^t - x_{id}^t)]$$

其中 $\chi = \frac{2}{|2-\varphi-\sqrt{\varphi^2-4\varphi}|}$, $\varphi = c_1 + c_2 > 4$ 。

2. 自适应 PSO: 动态调整参数

- 根据种群多样性调整惯性权重
- 根据搜索进度调整加速系数

3. 多目标 PSO: 处理多目标优化问题

- 维护外部存档保存 Pareto 最优解
- 使用拥挤距离保持多样性

混合 PSO 算法

1. **PSO-DE 混合:** 利用 DE 的变异增强 PSO 的探索能力
2. **PSO-局部搜索混合:** 在 PSO 迭代中加入局部搜索
3. **量子 PSO:** 引入量子力学概念, 增强全局搜索能力

7.4.6 PSO 的收敛性分析

简化 PSO 模型

考虑简化的一维 PSO 模型:

$$v^{t+1} = wv^t + c_1r_1(p - x^t) + c_2r_2(g - x^t)$$

$$x^{t+1} = x^t + v^{t+1}$$

可以写成矩阵形式:

$$\begin{bmatrix} x^{t+1} \\ v^{t+1} \end{bmatrix} = \begin{bmatrix} 1 - \varphi & w \\ -\varphi & w \end{bmatrix} \begin{bmatrix} x^t \\ v^t \end{bmatrix} + \begin{bmatrix} c_1r_1p + c_2r_2g \\ c_1r_1p + c_2r_2g \end{bmatrix}$$

其中 $\varphi = c_1r_1 + c_2r_2$ 。

收敛条件

系统稳定的充分条件是特征值的模小于 1:

$$|\lambda_{1,2}| < 1$$

其中 $\lambda_{1,2}$ 是系统矩阵的特征值。

计算可得收敛条件为:

$$w < 1, \quad \varphi > 0, \quad 2w - \varphi - 2 < 0$$

7.4.7 PSO 的优缺点总结

优点

- 概念直观, 易于理解和实现
- 参数较少, 调节相对简单
- 收敛速度快
- 具有天然的并行性
- 对连续优化问题表现良好

缺点

- 易早熟收敛，陷入局部最优
- 对高维复杂问题效果下降
- 缺乏严格的全局收敛性证明
- 对离散优化问题需要特殊处理

7.5 协方差矩阵自适应进化策略：学习问题结构的优化器

7.5.1 从简单进化策略到 CMA-ES

进化策略（Evolution Strategies, ES）由 Rechenberg 和 Schwefel 在 20 世纪 60 年代提出，是最早的进化算法之一。简单 ES 使用高斯分布进行变异：

简单进化策略

(1 + 1)-ES 算法：

Algorithm 23 (1 + 1)-ES 算法

Require: 初始解 x , 初始步长 σ , 目标函数 f

```

1: 设置成功计数器  $s = 0$ 
2: for  $t = 1$  to  $T_{\max}$  do
3:   生成试验解:  $x' = x + \sigma \cdot N(0, I)$ 
4:   if  $f(x') < f(x)$  then
5:     接受试验解:  $x = x'$ 
6:     增加成功计数器:  $s = s + 1$ 
7:   end if
8:   if  $t \bmod n = 0$  then
9:     if  $s/n < 1/5$  then
10:      减小步长:  $\sigma = \sigma \cdot c_d \{c_d < 1\}$ 
11:    else if  $s/n > 1/5$  then
12:      增大步长:  $\sigma = \sigma \cdot c_i \{c_i > 1\}$ 
13:    end if
14:    重置计数器:  $s = 0$ 
15:  end if
16: end for

```

CMA-ES 的基本思想

协方差矩阵自适应进化策略 (Covariance Matrix Adaptation Evolution Strategy, CMA-ES)由 Hansen 和 Ostermeier 在 1996 年提出。与简单 ES 使用各向同性高斯分布不同,CMA-ES 使用完整的多元高斯分布:

$$x \sim \mathcal{N}(m, \sigma^2 C)$$

其中:

- m : 分布均值
- σ : 全局步长
- C : 协方差矩阵, 描述变量间的相关性和尺度

7.5.2 CMA-ES 的核心组件

采样新解

在第 g 代, 从当前分布采样 λ 个子代:

$$x_k^{(g+1)} = m^{(g)} + \sigma^{(g)} y_k, \quad y_k \sim \mathcal{N}(0, C^{(g)}), \quad k = 1, \dots, \lambda$$

选择与重组

选择 μ 个最优个体进行加权重组:

$$m^{(g+1)} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}^{(g+1)}$$

其中 $x_{i:\lambda}$ 是第 i 个最优个体, w_i 是权重, 满足 $\sum_{i=1}^{\mu} w_i = 1$, 通常 $w_1 \geq w_2 \geq \dots \geq w_{\mu} > 0$ 。

步长控制

CMA-ES 通过累积路径 (evolution path) 控制步长:

$$p_{\sigma}^{(g+1)} = (1 - c_{\sigma}) p_{\sigma}^{(g)} + \sqrt{c_{\sigma}(2 - c_{\sigma}) \mu_{\text{eff}}} C^{(g)-1/2} \frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}$$

其中:

- c_{σ} : 学习率
- $\mu_{\text{eff}} = 1 / \sum_{i=1}^{\mu} w_i^2$: 有效选择质量

步长更新:

$$\sigma^{(g+1)} = \sigma^{(g)} \exp \left(\frac{c_{\sigma}}{d_{\sigma}} \left(\frac{\|p_{\sigma}^{(g+1)}\|}{\mathbb{E}\|\mathcal{N}(0, I)\|} - 1 \right) \right)$$

其中 d_{σ} 是阻尼系数, $\mathbb{E}\|\mathcal{N}(0, I)\| \approx \sqrt{n} + O(1/n)$ 是 n 维标准正态分布随机向量的期望范数。

协方差矩阵自适应

CMA-ES 通过两种方式更新协方差矩阵：

1. 秩 μ 更新：利用当前代的信息

$$C_{\mu}^{(g+1)} = \sum_{i=1}^{\mu} w_i y_{i:\lambda}^{(g+1)} (y_{i:\lambda}^{(g+1)})^{\top}$$

2. 秩 1 更新：利用进化路径的信息

$$p_c^{(g+1)} = (1 - c_c)p_c^{(g)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}$$

$$C_1^{(g+1)} = p_c^{(g+1)} (p_c^{(g+1)})^{\top}$$

综合更新：

$$C^{(g+1)} = (1 - c_1 - c_{\mu})C^{(g)} + c_1 C_1^{(g+1)} + c_{\mu} C_{\mu}^{(g+1)}$$

其中 c_1 和 c_{μ} 是学习率。

7.5.3 CMA-ES 的完整算法

Algorithm 24 协方差矩阵自适应进化策略 (CMA-ES)

Require: 初始解 m , 初始步长 σ , 种群大小 λ

- 1: 设置参数: $c_{\sigma}, d_{\sigma}, c_c, c_1, c_{\mu}$, 权重 w_i
 - 2: 初始化: $C = I$, $p_{\sigma} = 0$, $p_c = 0$
 - 3: 计算 $\mu_{\text{eff}} = 1 / \sum_{i=1}^{\mu} w_i^2$
 - 4: **for** $g = 0$ to $G_{\text{max}} - 1$ **do**
 - 5: 采样: $x_k = m + \sigma \cdot y_k$, $y_k \sim \mathcal{N}(0, C)$, $k = 1, \dots, \lambda$
 - 6: 评估适应度并排序: $f(x_{1:\lambda}) \leq \dots \leq f(x_{\lambda:\lambda})$
 - 7: 重组: $m' = \sum_{i=1}^{\mu} w_i x_{i:\lambda}$
 - 8: 更新进化路径:
 - 9: $p_{\sigma} = (1 - c_{\sigma})p_{\sigma} + \sqrt{c_{\sigma}(2 - c_{\sigma})\mu_{\text{eff}}} C^{-1/2} \frac{m' - m}{\sigma}$
 - 10: $p_c = (1 - c_c)p_c + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \frac{m' - m}{\sigma}$
 - 11: 更新步长: $\sigma = \sigma \cdot \exp \left(\frac{c_{\sigma}}{d_{\sigma}} (\|p_{\sigma}\| / \mathbb{E} \|\mathcal{N}(0, I)\| - 1) \right)$
 - 12: 更新协方差矩阵:
 - 13: $C = (1 - c_1 - c_{\mu})C + c_1 p_c p_c^{\top} + c_{\mu} \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^{\top}$
 - 14: 更新均值: $m = m'$
 - 15: **end for**
-

7.5.4 CMA-ES 的参数设置

默认参数设置

Hansen 建议的默认参数设置：

- 种群大小： $\lambda = 4 + \lfloor 3 \ln n \rfloor$
- 父代数量： $\mu = \lfloor \lambda / 2 \rfloor$
- 重组权重： $w_i = \frac{\ln(\mu+0.5) - \ln i}{\sum_{j=1}^{\mu} (\ln(\mu+0.5) - \ln j)}$
- 步长学习率： $c_\sigma = \frac{\mu_{\text{eff}} + 2}{n + \mu_{\text{eff}} + 5}$
- 阻尼系数： $d_\sigma = 1 + 2 \max(0, \sqrt{\frac{\mu_{\text{eff}} - 1}{n + 1}} - 1) + c_\sigma$
- 协方差路径学习率： $c_c = \frac{4 + \mu_{\text{eff}}/n}{n + 4 + 2\mu_{\text{eff}}/n}$
- 协方差秩 1 学习率： $c_1 = \frac{2}{(n + 1.3)^2 + \mu_{\text{eff}}}$
- 协方差秩 μ 学习率： $c_\mu = \min(1 - c_1, \frac{2(\mu_{\text{eff}} - 2 + 1/\mu_{\text{eff}})}{(n + 2)^2 + \mu_{\text{eff}}})$

参数解释与调整

- 种群大小 λ : 影响探索能力，大 λ 增强全局搜索
- 学习率 c_1, c_μ : 控制协方差更新速度，小值稳定，大值快速适应
- 有效样本数 μ_{eff} : 衡量选择强度，影响参数更新

7.5.5 CMA-ES 的变体与改进

Active-CMA-ES

利用不成功的搜索方向更新协方差矩阵：

$$C^{(g+1)} = (1 - c_1 - c_\mu - c_\alpha)C^{(g)} + c_1 C_1^{(g+1)} + c_\mu C_\mu^{(g+1)} - c_\alpha C_\alpha^{(g+1)}$$

其中 C_α 来自最差个体的信息。

sep-CMA-ES

使用对角协方差矩阵的简化版本，计算复杂度从 $O(n^2)$ 降到 $O(n)$ ，适用于高维问题。

BI-POP-CMA-ES

使用两个种群：一个用小 λ 快速收敛，一个用大 λ 增强探索。

CMA-ES with Margin

防止在边界问题中过早收敛到边界。

7.5.6 CMA-ES 的理论性质

不变性

CMA-ES 具有以下不变性：

- 平移不变性： $f(x)$ 和 $f(x + c)$ 难度相同
- 旋转不变性： $f(x)$ 和 $f(Rx)$ 难度相同， R 是正交矩阵
- 缩放不变性： $f(x)$ 和 $a f(x)$ 难度相同

收敛性分析

在适当条件下，CMA-ES 以概率 1 收敛到全局最优：

$$\lim_{g \rightarrow \infty} P(f(m^{(g)}) = f^*) = 1$$

收敛速度：在球面函数上，CMA-ES 达到线性收敛：

$$\mathbb{E}[\ln(\sigma^{(g+1)} / \sigma^{(g)})] = -\frac{c_\sigma}{d_\sigma} \cdot \frac{\mu_{\text{eff}}}{n}$$

7.5.7 CMA-ES 的优缺点总结

优点

- 强大的局部搜索能力
- 自适应学习问题结构
- 旋转不变性，对旋转问题表现优异
- 理论分析较为完善
- 参数设置有系统方法

缺点

- 计算复杂度高， $O(n^2)$ 内存， $O(n^3)$ 计算
- 对高维问题 ($n > 100$) 效率下降
- 需要相对较大的种群
- 实现较为复杂

7.6 算法对比与综合应用

7.6.1 三剑客对比分析

表 7.3: DE、PSO、CMA-ES 算法全面对比

特性	差分进化 (DE)	粒子群优化 (PSO)	CMA-ES
提出时间	1995 年	1995 年	1996 年
灵感来源	生物进化	鸟群觅食	自然进化
核心机制	向量差分变异	个体-社会学习	协方差自适应
搜索分布	基于差分向量	基于速度位置	多元高斯分布
参数数量	少 (3 个)	少 (3-4 个)	多 (10+ 个)
实现难度	简单	简单	复杂
计算复杂度	$O(NP \cdot D)$	$O(N \cdot D)$	$O(D^2)$ 内存, $O(D^3)$ 计算
收敛速度	中等	快	慢但精确
全局搜索	强	中等	中等
局部搜索	中等	强	很强
旋转不变性	无	无	有
理论分析	较少	中等	丰富
适应自相关	中等	弱	强
高维性能	好	中等	差 ($D > 100$)
噪声鲁棒性	好	中等	好
约束处理	中等	中等	困难

7.6.2 性能基准测试

测试函数集

常用的基准测试函数:

1. 单峰函数: 评估局部搜索能力

- Sphere 函数: $f_1(x) = \sum_{i=1}^n x_i^2$
- Ellipsoid 函数: $f_2(x) = \sum_{i=1}^n 10^{6\frac{i-1}{n-1}} x_i^2$

2. 多峰函数: 评估全局搜索能力

- Rastrigin 函数: $f_3(x) = 10n + \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i)]$
- Schwefel 函数: $f_4(x) = 418.9829n - \sum_{i=1}^n x_i \sin(\sqrt{|x_i|})$

3. 旋转函数: 评估旋转不变性

- 旋转 Ellipsoid 函数: $f_5(x) = f_2(Rx)$, R 是随机旋转矩阵

实验结果

在 CEC (Congress on Evolutionary Computation) 基准测试上的典型结果：

- **低维问题** ($D \leq 30$): CMA-ES 通常最优
- **中维问题** ($30 < D \leq 100$): DE 和 CMA-ES 竞争
- **高维问题** ($D > 100$): DE 通常最优
- **多峰问题**: DE 表现优异
- **噪声问题**: PSO 和 DE 表现较好

7.6.3 混合策略与自适应选择

算法选择器

根据问题特征自动选择算法：

- **维度**: 高维选 DE, 低维选 CMA-ES
- **多峰性**: 多峰选 DE, 单峰选 CMA-ES
- **计算预算**: 预算少选 PSO, 预算多选 CMA-ES
- **旋转性**: 旋转问题选 CMA-ES

混合算法设计

1. **DE-CMA-ES 混合**: 用 DE 进行全局探索, 用 CMA-ES 进行局部开发
2. **PSO-DE 混合**: 用 PSO 快速收敛, 用 DE 增强多样性
3. **层次混合**: 不同层次使用不同算法

7.6.4 实际应用案例

工程优化问题

- **航空航天**: 机翼形状优化, 使用 CMA-ES
- **汽车工业**: 发动机参数优化, 使用 DE
- **电力系统**: 电网调度优化, 使用 PSO
- **化学工程**: 反应器设计优化, 使用 DE

机器学习超参数优化

- **神经网络**: 结构优化, 使用 CMA-ES
- **支持向量机**: 参数优化, 使用 DE
- **集成学习**: 权重优化, 使用 PSO

金融优化

- 投资组合：资产配置优化，使用 PSO
- 风险管理：VaR 优化，使用 DE
- 交易策略：参数优化，使用 CMA-ES

7.7 前沿方向与未来展望

7.7.1 大规模优化

处理 $D > 1000$ 维的问题：

- 协方差矩阵压缩：低秩近似，减少计算量
- 变量分组：利用问题结构，分而治之
- 随机子空间：在随机子空间中优化

7.7.2 多目标优化

同时优化多个目标：

- 多目标 DE：NSDE, GDE3
- 多目标 PSO：MOPSO, SMPSO
- 多目标 CMA-ES：MO-CMA-ES

7.7.3 昂贵优化

目标函数评估成本高：

- 代理模型：用廉价模型近似目标函数
- 贝叶斯优化：结合代理模型和获取函数
- 进化算法 + 代理模型：EA 作为全局优化器

7.7.4 动态优化

优化问题随时间变化：

- 变化检测：检测问题变化
- 响应策略：重启，多样性注入，记忆利用
- 预测模型：预测变化趋势

7.7.5 学习优化 (Learning to Optimize)

使用机器学习改进优化算法：

L2O-RNN

用循环神经网络学习优化器的更新规则:

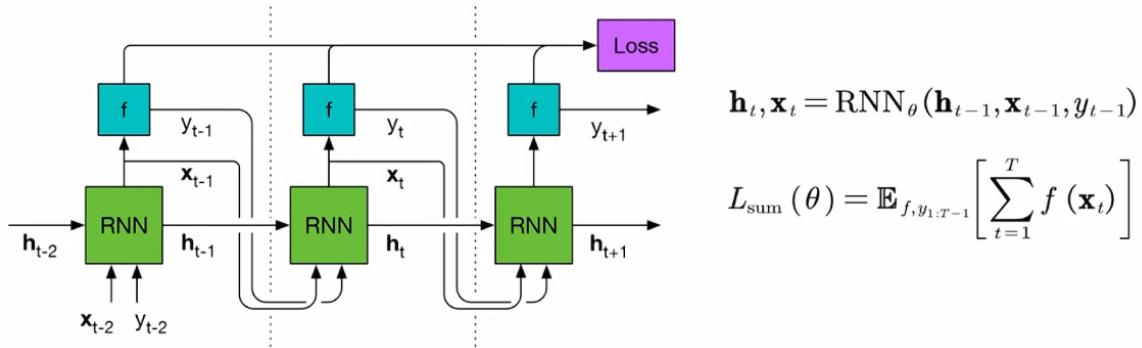


图 7.11: L2O-RNN 框架: 使用 RNN 学习优化器的更新规则, 可以泛化到不同问题

OPRO: 通过提示进行优化

利用大语言模型的推理能力进行优化:

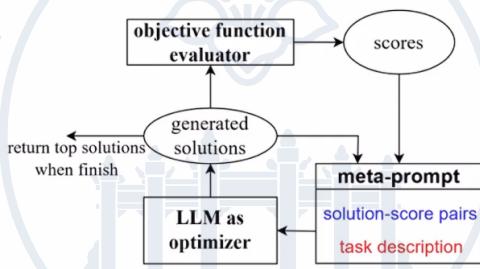


图 7.12: OPRO 框架: 通过自然语言提示指导大语言模型进行优化, 展示了 AI 在优化问题中的新应用

7.7.6 理论发展

- 收敛性理论: 建立更严格的理论基础
- 复杂度分析: 分析算法的计算复杂度
- No-Free-Lunch 定理: 理解算法的根本限制

7.8 实践指南与经验总结

7.8.1 算法选择流程

Algorithm 25 进化算法选择流程

Require: 优化问题特征

Ensure: 推荐算法

```
1: if 维度  $D > 100$  then
2:   return DE {高维问题首选 DE}
3: else if 计算预算有限 then
4:   return PSO {快速收敛}
5: else if 问题旋转不变 then
6:   return CMA-ES {旋转问题首选 CMA-ES}
7: else if 多峰性强 then
8:   return DE {多峰问题首选 DE}
9: else if 需要高精度解 then
10:  return CMA-ES {局部搜索能力强}
11: else
12:  return DE {默认选择}
13: end if
```

7.8.2 参数调优建议

通用建议

- 种群大小：从经验规则开始，逐步调整
- 多次运行：至少运行 30 次，统计结果
- 记录日志：记录搜索过程，便于分析
- 可视化：可视化搜索过程，直观理解

问题特定调优

1. 分析问题特征：维度，多峰性，可微性，约束等
2. 选择合适算法：根据特征选择算法
3. 设置初始参数：使用经验值
4. 小规模试验：在小规模问题上调参
5. 逐步调整：根据结果微调参数

7.8.3 常见陷阱与避免方法

表 7.4: 进化算法常见陷阱与解决方案

问题	表现	解决方案
早熟收敛	种群多样性迅速下降，陷入局部最优	增加种群大小，增加变异率，使用多种群
收敛过慢	多代无显著改进	减小种群大小，增加选择压力，使用局部搜索
参数敏感	不同参数结果差异大	使用自适应参数，多次试验取平均
维度灾难	高维性能急剧下降	使用降维技术，变量分组，问题分解
计算昂贵	评估次数过多	使用代理模型，提前终止，并行计算

7.8.4 性能评估指标

- **解质量:** 最优值, 平均值, 标准差
- **收敛速度:** 达到特定精度所需评估次数
- **鲁棒性:** 多次运行结果的一致性
- **成功率:** 达到全局最优的概率
- **计算效率:** 时间复杂度和空间复杂度

7.9 结论

进化计算作为一类受自然启发的优化算法，在解决复杂优化问题中展现出强大能力。DE、PSO 和 CMA-ES 作为连续优化领域的三大代表性算法，各有特色：

- **DE:** 以简洁的差分变异机制实现强大的全局搜索，适合高维、多峰问题
- **PSO:** 通过个体与群体经验的平衡实现快速收敛，适合中等维度、快速求解
- **CMA-ES:** 通过协方差矩阵自适应学习问题结构，实现精确的局部搜索，适合低维、精确求解

在实际应用中，应根据问题特征选择合适的算法，必要时可以结合多种算法的优势。随着计算技术的发展和新理论的提出，进化计算将继续在科学和工程领域发挥重要作用。

未来，进化计算将与机器学习、高性能计算、自动调参等技术深度融合，向着更智能、更高效、更自适应的方向发展。理解这些算法的原理、特性和应用场景，对于解决实际优化问题具有重要意义。

第八章 进化计算之拓展篇

8.1 引言：从连续优化到组合优化与程序生成

在前面的学习中，我们深入探讨了进化计算在连续优化问题上的应用，特别是差分进化（DE）、粒子群优化（PSO）和协方差矩阵自适应进化策略（CMA-ES）这三种经典算法。然而，现实世界中的优化问题不仅仅局限于连续空间，还存在大量的组合优化问题，如旅行商问题（TSP）、调度问题、路径规划等。同时，我们也不仅仅满足于优化参数，有时我们希望自动生成解决问题的程序或策略。本章将介绍进化计算的两个重要拓展方向：蚁群优化（Ant Colony Optimization, ACO）和遗传编程（Genetic Programming, GP）。

8.1.1 连续优化与组合优化的区别

表 8.1: 连续优化与组合优化的对比

特征	连续优化	组合优化
决策变量	连续实数	离散取值（整数、排列、集合等）
搜索空间	无限、连续、通常有界	有限、离散、通常规模巨大
典型问题	函数优化、参数调优	TSP、调度、装箱、图着色
算法特点	基于梯度或随机搜索	基于启发式、局部搜索、元启发式
解的质量评估	目标函数值	目标函数值，但可能涉及约束满足
挑战	局部最优、高维度、多峰性	组合爆炸、约束复杂、NP 难

组合优化问题通常具有以下特点：

- **有限但巨大的搜索空间：** 虽然理论上解的数量有限，但实际上往往无法枚举
- **NP 难问题：** 大多数有实际意义的组合优化问题都是 NP 难的
- **约束复杂：** 通常包含多种类型的约束条件
- **离散结构：** 解通常具有组合结构，如排列、划分、匹配等

8.1.2 自然启发的优化思想

进化计算的核心思想是受自然现象启发，通过模拟自然界的智能行为来解决复杂的优化问题。我们已经学习了模拟生物进化的遗传算法（GA）、模拟鸟群觅食的粒子群优化（PSO）等。本章将介绍另外两种受自然启发的算法：

- **蚁群优化（ACO）**: 模拟蚂蚁群体觅食行为，通过信息素的正反馈机制寻找最优路径
- **遗传编程（GP）**: 模拟生物进化过程，自动生成计算机程序

这两种算法分别针对不同类型的优化问题：ACO 主要解决组合优化问题，GP 则用于自动程序设计。

8.2 蚁群优化算法：模拟蚂蚁觅食的智能优化

8.2.1 自然界的蚂蚁觅食行为

在自然界中，蚂蚁群体展现出令人惊叹的集体智能。尽管单个蚂蚁的能力有限，但蚁群却能高效地找到从巢穴到食物源的最短路径。这种能力主要依赖于一种称为信息素（Pheromone）的化学物质。

双桥实验

著名的双桥实验揭示了蚂蚁觅食的智能行为：

实验过程：

1. 在蚁巢和食物源之间设置两条不同长度的路径
2. 初始时，蚂蚁随机选择路径
3. 由于短路径上的往返时间更短，信息素积累更快
4. 随着时间推移，短路径上的信息素浓度越来越高
5. 最终，大多数蚂蚁都选择短路径

关键机制：

- **正反馈**: 更多的蚂蚁选择信息素浓度高的路径
- **自组织**: 没有中央控制，通过局部相互作用形成全局模式
- **随机性**: 蚂蚁的选择有一定随机性，避免陷入局部最优

8.2.2 蚁群优化算法的提出

蚁群优化算法由 Marco Dorigo 于 1992 年在其博士论文中首次提出，随后在 1996 年完善并正式发表。ACO 是群体智能（Swarm Intelligence）的代表性算法之一。

表 8.2: 自然界蚂蚁觅食行为与 ACO 算法的对应关系

自然现象	ACO 算法概念	数学表示/解释
觅食空间	问题的搜索空间	所有可能解的集合
蚂蚁	搜索代理	算法中的个体，负责构建解
蚂蚁构建的路径	一个候选解	如 TSP 中的城市访问序列
信息素	人工信息素	记录路径优劣的数值，通常表示为矩阵
信息素挥发	信息素更新机制	避免算法早熟，保持探索能力
正反馈	强化学习机制	优秀解的信息素增强，吸引更多蚂蚁
最优路径	问题的最优解	算法找到的最佳解

8.2.3 基本蚁群系统 (Ant System, AS)

基本蚁群系统是最早的 ACO 算法，以 TSP（旅行商问题）为典型应用场景。TSP 问题可以形式化描述为：

给定 n 个城市和城市间的距离矩阵 $D = (d_{ij})_{n \times n}$ ，寻找一条访问每个城市恰好一次并回到起点的最短回路。

AS 算法的核心要素

1. 信息素矩阵 τ : τ_{ij} 表示边 (i, j) 上的信息素浓度
2. 启发式信息 η : $\eta_{ij} = 1/d_{ij}$, 表示从城市 i 到 j 的启发式吸引力
3. 蚂蚁的路径构建：每只蚂蚁按照概率选择下一个城市
4. 信息素更新：包括挥发和增强两个过程

路径构建：随机比例规则

蚂蚁 k 在城市 i 选择下一个城市 j 的概率为：

$$p_k(i, j) = \begin{cases} \frac{[\tau(i, j)]^\alpha [\eta(i, j)]^\beta}{\sum_{u \in J_k(i)} [\tau(i, u)]^\alpha [\eta(i, u)]^\beta}, & \text{if } j \in J_k(i) \\ 0, & \text{otherwise} \end{cases}$$

其中：

- $J_k(i)$: 蚂蚁 k 在城市 i 可以访问的、且未访问过的城市集合
- α : 信息素重要程度参数，控制信息素的影响

- β : 启发式信息重要程度参数, 控制启发式信息的影响
- $\tau(i, j)$: 边 (i, j) 上的信息素浓度
- $\eta(i, j)$: 启发式信息, 通常 $\eta(i, j) = 1/d_{ij}$

参数 α 和 β 的作用:

- $\alpha = 0$: 算法退化为贪心算法, 只考虑距离因素
- $\beta = 0$: 算法只依赖信息素, 可能陷入局部最优
- 通常设置: $\alpha = 1$, $\beta = 2 \sim 5$

信息素更新机制

信息素更新包括两个过程: 挥发和增强。

1. 信息素挥发: 所有边上的信息素按一定比例挥发

$$\tau(i, j) \leftarrow (1 - \rho) \cdot \tau(i, j)$$

其中 $\rho \in (0, 1]$ 是挥发系数, 通常 $\rho = 0.5$ 。

2. 信息素增强: 蚂蚁在其经过的边上释放信息素

$$\tau(i, j) \leftarrow \tau(i, j) + \sum_{k=1}^m \Delta\tau_k(i, j)$$

其中 m 是蚂蚁数量, $\Delta\tau_k(i, j)$ 是蚂蚁 k 在边 (i, j) 上释放的信息素量:

$$\Delta\tau_k(i, j) = \begin{cases} Q/L_k, & \text{if 蚂蚁 } k \text{ 经过边 } (i, j) \\ 0, & \text{otherwise} \end{cases}$$

其中 Q 是常数, L_k 是蚂蚁 k 构建路径的长度。

初始信息素设置

初始信息素浓度 τ_0 的设置对算法性能有重要影响:

$$\tau_0 = \frac{m}{L_{nn}}$$

其中 L_{nn} 是使用最近邻贪心算法得到的路径长度。

设置原理:

- 与问题规模相关: L_{nn} 反映了问题难度
- 与蚂蚁数量相关: m 越大, 初始信息素相对越小
- 量纲一致性: 与信息素增强公式中的 Q/L_k 量纲一致

AS 算法流程

Algorithm 26 基本蚁群系统 (Ant System, AS) 算法

Require: TSP 实例, 蚂蚁数量 m , 参数 α, β, ρ, Q , 最大迭代次数 T_{\max}

Ensure: 最优路径 T^* 和其长度 L^*

```

1: 初始化: 计算启发式信息  $\eta_{ij} = 1/d_{ij}$ , 初始信息素  $\tau_{ij} = \tau_0$ 
2: 计算最近邻路径长度  $L_{\text{nn}}$ , 设置  $\tau_0 = m/L_{\text{nn}}$ 
3: 设置最优路径  $T^* = \emptyset$ ,  $L^* = \infty$ 
4: for  $t = 1$  to  $T_{\max}$  do
5:   for  $k = 1$  to  $m$  do
6:     随机选择起始城市  $s_k$ 
7:     初始化已访问城市列表  $\text{visited}_k = \{s_k\}$ 
8:     当前城市  $c = s_k$ 
9:     while  $|\text{visited}_k| < n$  do
10:       根据公式计算选择每个未访问城市的概率  $p_k(c, j)$ 
11:       使用轮盘赌选择下一个城市  $j$ 
12:       将  $j$  加入  $\text{visited}_k$ , 更新当前城市  $c = j$ 
13:     end while
14:     计算路径长度  $L_k$ , 包括返回起点的距离
15:     if  $L_k < L^*$  then
16:       更新  $L^* = L_k$ ,  $T^* = \text{visited}_k$ 
17:     end if
18:   end for
19:   信息素挥发:  $\tau_{ij} = (1 - \rho)\tau_{ij}$ , 对所有  $i, j$ 
20:   信息素增强: 对每只蚂蚁  $k$ , 对其路径上的每条边  $(i, j)$ ,  $\tau_{ij} = \tau_{ij} + Q/L_k$ 
21: end for
22: return  $T^*, L^*$ 
  
```

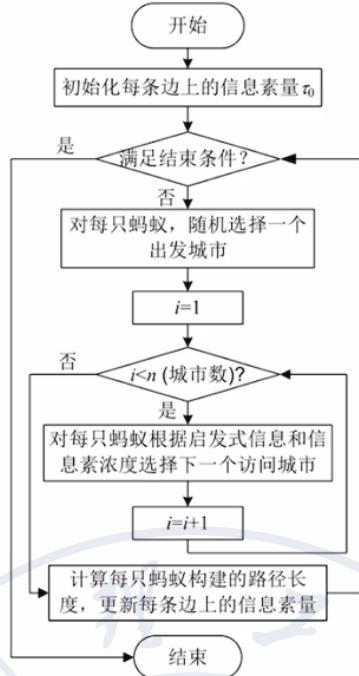


图 8.1: AS 算法流程示意图：包括路径构建和信息素更新两个主要阶段

8.2.4 AS 算法实例：四城市 TSP 问题

示例 8.2.1 (四城市 TSP 问题). 距离矩阵为：

$$D = \begin{bmatrix} \infty & 3 & 1 & 2 \\ 3 & \infty & 5 & 4 \\ 1 & 5 & \infty & 2 \\ 2 & 4 & 2 & \infty \end{bmatrix}$$

参数设置： $m = 3$, $\alpha = 1$, $\beta = 2$, $\rho = 0.5$, $Q = 1$ 。

步骤 1：初始化

1. 计算最近邻路径：从 A 出发，最近邻为 C(1), 然后 D(2), 然后 B(4), 返回 A(3), 总长 $L_{nn} = 10$
2. 初始信息素： $\tau_0 = m/L_{nn} = 3/10 = 0.3$
3. 信息素矩阵：

$$\tau^{(0)} = \begin{bmatrix} 0 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0 \end{bmatrix}$$

4. 启发式信息矩阵:

$$\eta = \begin{bmatrix} 0 & 1/3 & 1 & 1/2 \\ 1/3 & 0 & 1/5 & 1/4 \\ 1 & 1/5 & 0 & 1/2 \\ 1/2 & 1/4 & 1/2 & 0 \end{bmatrix}$$

步骤 2: 路径构建 (以蚂蚁 1 为例)

假设蚂蚁 1 从城市 A 出发:

1. 当前在城市 A, 可访问城市: B, C, D

2. 计算选择概率:

$$p(A, B) = \frac{0.3^1 \times (1/3)^2}{0.3^1 \times (1/3)^2 + 0.3^1 \times 1^2 + 0.3^1 \times (1/2)^2} = \frac{0.0333}{0.4083} = 0.0815$$

$$p(A, C) = \frac{0.3}{0.4083} = 0.7348$$

$$p(A, D) = \frac{0.075}{0.4083} = 0.1837$$

3. 轮盘赌选择: 假设随机数 0.05, 选择城市 B

4. 以此类推, 最终得到路径 A→B→D→C→A, 长度 $L_1 = 3 + 4 + 2 + 1 = 10$

步骤 3: 信息素更新

1. 信息素挥发: $\tau_{ij} = 0.5 \times \tau_{ij}$

2. 信息素增强: 假设三只蚂蚁的路径长度分别为 10, 10, 12

- 蚂蚁 1 在边 (A,B) 上增强: $\Delta\tau_1(A, B) = 1/10 = 0.1$
- 蚂蚁 2 在边 (A,B) 上增强: $\Delta\tau_2(A, B) = 0.1$
- 蚂蚁 3 未经过边 (A,B)

3. 边 (A,B) 更新后: $\tau_{AB} = 0.5 \times 0.3 + 0.1 + 0.1 = 0.35$

8.2.5 蚁群系统 (Ant Colony System, ACS) 的改进

基本 AS 算法存在收敛速度慢、易陷入局部最优等问题。Dorigo 和 Gambardella 于 1997 年提出了改进的蚁群系统 (ACS)。

ACS 的三项改进

1. 伪随机比例规则: 平衡探索与开发
2. 局部信息素更新: 增加探索能力
3. 全局信息素更新: 只更新最优路径

伪随机比例规则

蚂蚁 k 在城市 i 选择下一个城市 j 的规则:

$$j = \begin{cases} \arg \max_{u \in J_k(i)} \{[\tau(i, u)]^\alpha [\eta(i, u)]^\beta\}, & \text{if } q \leq q_0 \\ S, & \text{otherwise} \end{cases}$$

其中:

- q 是 $[0, 1]$ 均匀分布的随机数
- q_0 是参数, 控制开发与探索的平衡
- S 是按照 AS 的概率分布随机选择的结果

物理意义:

- 以概率 q_0 选择当前最优的边 (开发)
- 以概率 $1 - q_0$ 按照概率分布随机选择 (探索)
- 通常 $q_0 = 0.7 \sim 0.99$

局部信息素更新

蚂蚁在构建路径的过程中, 每经过一条边 (i, j) , 立即进行局部更新:

$$\tau(i, j) = (1 - \xi) \cdot \tau(i, j) + \xi \cdot \tau_0$$

其中 $\xi \in (0, 1)$ 是局部挥发系数, 通常 $\xi = 0.1$ 。

作用:

- 降低已访问边的信息素, 鼓励探索新路径
- 避免算法过早收敛
- 增加解的多样性

全局信息素更新

在每轮迭代结束后, 只对全局最优路径 T_{gb} 进行更新:

$$\tau(i, j) = (1 - \rho) \cdot \tau(i, j) + \rho \cdot \Delta\tau(i, j), \quad \forall (i, j) \in T_{gb}$$

其中 $\Delta\tau(i, j) = 1/L_{gb}$, L_{gb} 是全局最优路径长度。

与 AS 的区别:

- AS 更新所有蚂蚁经过的边
- ACS 只更新全局最优路径上的边
- 计算复杂度从 $O(n^2)$ 降到 $O(n)$

信息素边界限制

ACS 中信息素被限制在 $[\tau_{\min}, \tau_{\max}]$ 范围内:

- $\tau_{\min} = \tau_0$, 通常 $\tau_0 = 1/(nL_{nn})$
- $\tau_{\max} = 1/(\rho L_{gb})$

作用:

- 避免某些边的信息素过高导致早熟
- 保证所有边都有被选择的机会

ACS 算法流程

Algorithm 27 蚁群系统 (Ant Colony System, ACS) 算法

Require: TSP 实例, 蚂蚁数量 m , 参数 $\alpha, \beta, \rho, \xi, q_0$, 最大迭代次数 T_{\max}

Ensure: 最优路径 T_{gb} 和其长度 L_{gb}

```

1: 初始化: 计算启发式信息  $\eta_{ij} = 1/d_{ij}$ , 初始信息素  $\tau_{ij} = \tau_0$ 
2: 计算最近邻路径长度  $L_{nn}$ , 设置  $\tau_0 = 1/(nL_{nn})$ 
3: 设置全局最优路径  $T_{gb} = \emptyset$ ,  $L_{gb} = \infty$ 
4: for  $t = 1$  to  $T_{\max}$  do
5:   初始化每只蚂蚁的起始城市
6:   for  $k = 1$  to  $m$  do
7:     当前城市  $i =$  起始城市
8:     初始化已访问城市列表  $visited_k = \{i\}$ 
9:     while  $|visited_k| < n$  do
10:       按照伪随机比例规则选择下一个城市  $j$ 
11:       局部信息素更新:  $\tau(i, j) = (1 - \xi)\tau(i, j) + \xi\tau_0$ 
12:       将  $j$  加入  $visited_k$ , 更新当前城市  $i = j$ 
13:     end while
14:     计算路径长度  $L_k$ 
15:     if  $L_k < L_{gb}$  then
16:       更新  $L_{gb} = L_k$ ,  $T_{gb} = visited_k$ 
17:     end if
18:   end for
19:   全局信息素更新: 对  $T_{gb}$  上的每条边  $(i, j)$ ,  $\tau(i, j) = (1 - \rho)\tau(i, j) + \rho/L_{gb}$ 
20:   限制信息素范围:  $\tau(i, j) = \max(\tau_{\min}, \min(\tau_{\max}, \tau(i, j)))$ 
21: end for
22: return  $T_{gb}, L_{gb}$ 

```

8.2.6 ACO 参数设置与调优

表 8.3: ACO 算法参数设置指南

参数	符号	推荐范围	影响
蚂蚁数量	m	n (城市数)	探索能力, 计算成本
信息素重要性	α	1	信息素的影响程度
启发式重要性	β	2-5	启发式信息的影响程度
全局挥发率	ρ	AS: 0.5, ACS: 0.1	信息素挥发速度, 影响收敛
局部挥发率	ξ	ACS: 0.1	增加探索, 防止早熟
开发概率	q_0	ACS: 0.7-0.99	控制开发与探索平衡
初始信息素	τ_0	AS: m/L_{nn} , ACS: $1/(nL_{nn})$	影响初期搜索行为

调优建议:

1. 从默认参数开始
2. 根据问题规模调整蚂蚁数量
3. 通过实验调整 α 和 β 的平衡
4. 收敛过快时减小 q_0 或增加 ξ
5. 收敛过慢时增加 q_0 或减小 ξ

8.2.7 ACO 与其他算法的对比

ACO vs GA

表 8.4: ACO 与 GA 的对比

维度	蚁群优化 (ACO)	遗传算法 (GA)
灵感来源	蚂蚁群体觅食行为	生物进化理论
解生成方式	逐步构建 (自回归)	全局重构 (非自回归)
约束处理	预防式: 构建时避免违反	修复式: 生成后修复或惩罚
信息共享	间接共享: 通过信息素	直接混合: 通过交叉操作
环境适应性	适应动态环境	静态优化为主
计算特点	正反馈、分布式	选择、交叉、变异
适用问题	路径规划、调度、分配	函数优化、参数调优、组合优化

ACO vs Q-Learning

表 8.5: ACO 与 Q-Learning 的对比

维度	蚁群优化 (ACO)	Q-Learning
抽象本质	基于经验的迭代优化	基于经验的迭代优化
主体性质	群体经验的分布式存储	单智能体的价值函数
适应环境	调整信息素矩阵	调整 Q-table
更新触发	一轮迭代后全局更新	单步动作后立即更新
核心数学	正反馈 + 挥发衰减	贝尔曼方程 + 时序差分
决策依据	启发式信息 + 群体经验	纯价值函数
探索控制	随机比例规则 + 挥发率	ϵ -greedy 或 Boltzmann
优化侧重	全局优化效果好	局部决策灵敏度高

8.2.8 ACO 的应用领域

ACO 算法已在多个领域成功应用：

1. 旅行商问题 (TSP) 及变种：车辆路径问题、带时间窗的 VRP
2. 调度问题：作业车间调度、流水车间调度
3. 网络路由：通信网络路由、无线传感器网络
4. 数据挖掘：聚类分析、特征选择
5. 图像处理：边缘检测、图像分割

8.3 遗传编程：自动程序生成的进化方法

8.3.1 遗传编程的基本思想

遗传编程由 John Koza 于 1992 年提出，是遗传算法的自然扩展。与 GA 优化参数不同，GP 优化的是计算机程序的结构。

核心思想

通过模拟自然选择过程，自动生成解决特定问题的计算机程序：

- 程序表示为树状结构
- 通过适应度评估程序的性能
- 使用遗传操作（选择、交叉、变异）进化程序种群
- 最终得到高性能的程序

与 GA 的关系

- **共同点:** 都基于进化计算框架, 使用选择、交叉、变异操作
- **不同点:**
 - GA 优化固定长度的字符串 (参数)
 - GP 优化可变大小的树结构 (程序)
 - GA 的交叉是等位基因交换
 - GP 的交叉是子树交换

8.3.2 GP 的个体表示: 程序树

GP 使用树状结构表示程序, 称为程序树 (Program Tree) 或语法树 (Syntax Tree)。

基元集 (Primitive Set)

基元集是 GP 可用的基本构建块, 包括:

1. **函数集 (Function Set):** 内部节点, 有子节点
 - 算术运算: +, -, *, /, sin, cos, exp, log
 - 布尔运算: AND, OR, NOT
 - 条件运算: IF-THEN-ELSE
 - 循环运算: WHILE, FOR
 - 其他函数: 取决于具体问题
2. **终止符集 (Terminal Set):** 叶节点, 无子节点
 - 变量: x, y, z
 - 常量: 1, 2, 3.14, true, false
 - 零参数函数: rand(), time()

基元集的设计原则

1. **充分性:** 基元集应能表达解空间中的所有可能解
2. **封闭性:** 任何函数节点的返回值类型应与子节点类型匹配
3. **适当性:** 基元集应与问题相关, 不过大也不过小
4. **效率性:** 避免不必要的基元, 减少搜索空间

8.3.3 GP 的基本步骤

GP 遵循进化计算的基本框架, 但操作针对树结构:

Algorithm 28 遗传编程基本流程

Require: 问题描述, 适应度函数, 基元集, 参数设置

Ensure: 最优程序

- 1: 初始化: 生成初始种群 (随机程序树)
 - 2: 评估: 计算每个个体的适应度
 - 3: **while** 未满足终止条件 **do**
 - 4: 选择: 根据适应度选择父代
 - 5: 繁殖: 对父代应用遗传操作生成子代
 - 交叉: 交换两个父代的子树
 - 变异: 替换父代的子树
 - 复制: 直接复制父代
 - 6: 评估: 计算子代的适应度
 - 7: 替换: 用子代替换部分或全部父代
 - 8: **end while**
 - 9: **return** 最优程序
-

种群初始化

有三种常用的初始化方法:

1. 增长法 (Grow Method)
 - 从函数集随机选择根节点
 - 对每个参数位置, 从整个基元集随机选择
 - 如果选择终止符, 则该分支停止生长
 - 如果选择函数, 则继续生长
 - 达到最大深度时, 必须选择终止符
2. 完全法 (Full Method)
 - 从函数集随机选择根节点
 - 直到深度 $d_{\max} - 1$, 都从函数集选择
 - 在深度 d_{\max} , 从终止符集选择
 - 生成的都是深度为 d_{\max} 的完整树
3. 分层对半分法 (Ramped Half-and-Half)
 - 将种群按深度分层
 - 每层中, 50% 用增长法, 50% 用完全法
 - 结合两种方法的优点, 增加多样性

适应度评估

适应度函数衡量程序解决目标问题的能力:

- **符号回归:** 预测值与真实值的误差
- **分类问题:** 分类准确率
- **控制问题:** 控制性能指标
- **程序生成:** 通过测试用例的比例

常用适应度度量:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i))^2$$

$$\text{准确率} = \frac{\text{正确分类数}}{\text{总样本数}}$$

选择操作

与 GA 类似, 常用选择方法:

- **轮盘赌选择:** 按适应度比例选择
- **锦标赛选择:** 随机选择 k 个个体, 选择最优的
- **精英选择:** 保留最优个体直接进入下一代

交叉操作

GP 交叉是子树交换:

1. 随机选择两个父代个体
2. 在每个父代中随机选择一个交叉点
3. 交换以交叉点为根的子树
4. 生成两个子代个体

变异操作

两种主要变异方式:

1. **子树变异 (Subtree Mutation)**
 - 随机选择变异点
 - 删 除以该点为根的子树
 - 用随机生成的子树替换
2. **单点变异 (Point Mutation)**
 - 随机选择变异点
 - 用同类型的基元替换
 - 保持树结构不变

复制操作

以一定概率直接复制父代到子代, 保留优秀个体。

8.3.4 GP 的参数设置

表 8.6: GP 算法参数设置指南

参数	推荐范围	说明
种群大小	500-5000	越大探索能力越强，计算成本越高
最大深度	3-10	控制程序复杂度，防止膨胀
交叉率	0.8-0.95	主要搜索操作，促进多样性
变异率	0.01-0.1	引入新基因，防止早熟
复制率	0-0.1	保留优秀个体
选择方法	锦标赛	常用锦标赛大小 $k = 7$
精英数量	1-5	保留最优个体
最大代数	50-1000	根据问题复杂度调整

8.3.5 GP 的应用案例：符号回归

符号回归是 GP 的经典应用，目标是发现数据背后的数学表达式。

问题描述

给定数据集 $\{(x_i, y_i)\}_{i=1}^n$ ，寻找数学表达式 $f(x)$ ，使得 $f(x_i) \approx y_i$ 。

示例 8.3.1 (二次多项式回归). 目标函数: $f(x) = x^2 + x + 1$ 数据范围: $x \in [-1, 1]$ 采样点: $x = -1.0, -0.9, \dots, 0.9, 1.0$

GP 设置

1. 终止符集: $T = \{x, \mathbb{R}\}$, 其中 \mathbb{R} 是随机常数
2. 函数集: $F = \{+, -, \times, \%\}$, 其中 $\%$ 是保护除法 (除 0 返回 1)
3. 适应度函数: MSE

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i))^2$$

4. 参数设置: 种群大小 4, 最大深度 2, 交叉率 0.5, 变异率 0.25, 复制率 0.25
5. 终止条件: $\text{MSE} < 0.1$

GP 求解过程

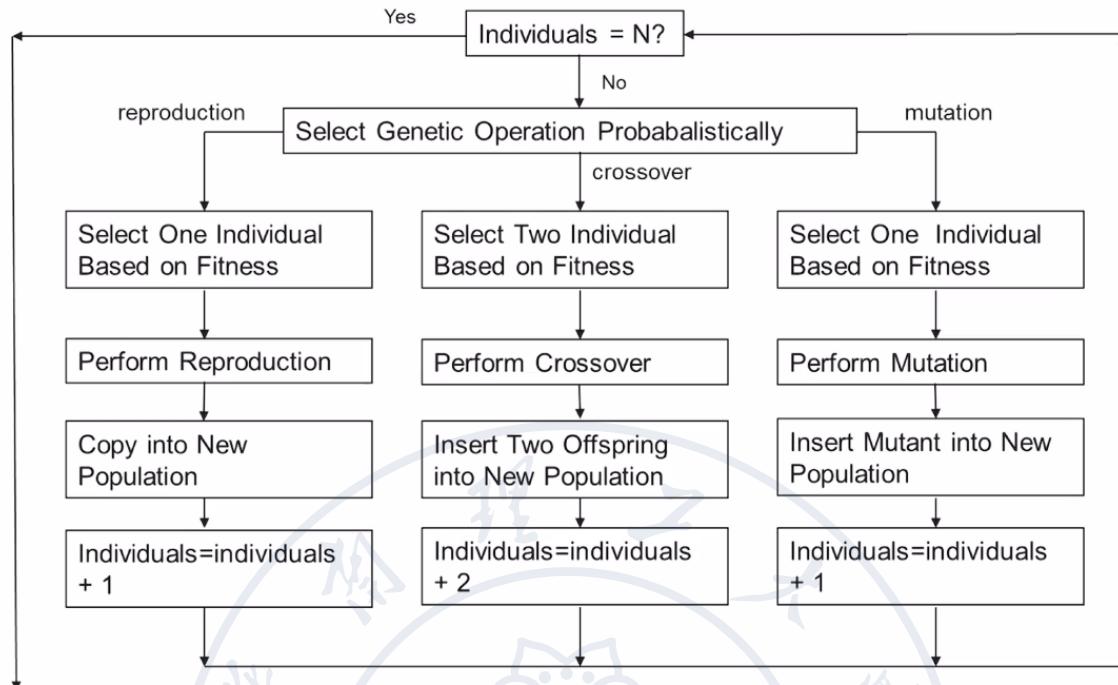


图 8.2: GP 求解符号回归问题的完整过程: 从初始化到找到最优解

第 1 代:

- 个体 1: $x + 1$, MSE=7.7
- 个体 2: $x^2 + 1$, MSE=11.00
- 个体 3: 2, MSE=17.98
- 个体 4: x , MSE=28.70

遗传操作:

- 复制: 个体 1 直接复制
- 变异: 个体 3 变异为 $x\%x$ (即 1)
- 交叉 1: 个体 1 和 4 交叉, 得到 x 和 $x + 1$
- 交叉 2: 个体 1 和 2 交叉, 得到 $x^2 + x + 1$

第 2 代:

- 个体 1: $x + 1$, MSE=7.7
- 个体 2: 1, MSE=11.00
- 个体 3: x , MSE=28.70
- 个体 4: $x^2 + x + 1$, MSE=0.0

找到最优解 $f(x) = x^2 + x + 1$, 终止。

不同回归方法对比

表 8.7: 不同回归方法的对比

方法	模型形式	优点	缺点
线性回归	$y = \beta_0 + \beta_1 x$	简单、可解释	只能拟合线性关系
多项式回归	$y = \sum_{i=0}^k \beta_i x^i$	可拟合非线性	阶数选择困难，易过拟合
神经网络	$y = f(Wx + b)$	强大拟合能力	黑箱、需大量数据、易过拟合
符号回归	任意表达式	可解释、自动发现	计算复杂、可能膨胀

8.3.6 GP 的优缺点与挑战

优点

- 自动程序设计：无需预先指定程序结构
- 可解释性：生成的程序易于理解
- 灵活性：适用于多种问题类型
- 创造性：可能发现人类未想到的解决方案

缺点与挑战

1. 计算复杂度高：程序评估和进化成本高
2. 膨胀现象（Bloat）：程序无意义地增长
3. 过拟合：在训练集上表现好，测试集差
4. 参数敏感：需要仔细调参
5. 局部最优：可能陷入局部最优解

膨胀现象的原因与对策

膨胀原因：

- 搜索偏差：大程序有更多交叉点
- 无用代码：不影响适应度的代码不被淘汰
- 适应度地形：大程序可能在小邻域内有更好解

对策：

- 深度限制：限制程序最大深度
- 大小限制：限制程序节点数

- 简化操作：定期简化程序
- 适应度惩罚：对复杂程序进行惩罚

8.3.7 GP 与其他算法的对比

GP vs 深度强化学习 (DRL)

表 8.8: GP 与 DRL 的对比

维度	遗传编程 (GP)	深度强化学习 (DRL)
策略表示	树结构 (符号表达式)	神经网络 (数值函数)
可微性	不可微	可微
训练方式	进化算法 (选择、交叉、变异)	梯度下降 (反向传播)
先验知识	不依赖，从基元集构建	依赖网络结构设计
可解释性	强，符号表达式易于理解	弱，黑箱模型难以解释
探索机制	通过变异和交叉探索	通过动作随机性探索
样本效率	通常较低	可以较高 (有重用机制)
计算需求	大量程序评估	大量环境交互
适用问题	符号回归、程序生成、控制	游戏、机器人控制、决策
优势	可解释、无需梯度、创造性	强大表示能力、端到端学习
劣势	计算昂贵、易膨胀、参数敏感	需要大量数据、不稳定、难调参

作为策略的 GP

GP 可以视为一种策略表示方法：

$$\pi(s) = \text{GP-Program}(s)$$

其中程序以状态 s 为输入，输出动作 a 。

增强 GP 处理序列决策：

- 引入状态变量作为终止符
- 添加记忆机制
- 使用迭代或递归结构

8.3.8 GP 的应用领域

1. 符号回归：发现数据背后的数学规律

2. 程序生成：自动生成算法、函数
3. 控制器设计：机器人控制、过程控制
4. 电路设计：数字电路、模拟电路
5. 游戏 AI：游戏策略、NPC 行为
6. 数据挖掘：特征构建、模式发现
7. 图像处理：滤波器设计、特征提取

8.4 总结与展望

8.4.1 算法对比总结

表 8.9: 进化计算算法综合对比

算法	灵感来源	表示方式	适用问题	特点
GA	生物进化	固定长度字符串	优化、搜索	通用、易实现
DE	向量差分	实数向量	连续优化	简单、高效
PSO	鸟群觅食	位置速度	连续优化	收敛快、参数少
CMA-ES	自然进化	高斯分布	连续优化	自适应、旋转不变
ACO	蚂蚁觅食	路径/排列	组合优化	正反馈、分布式
GP	生物进化	树结构	程序生成	自动设计、可解释

8.4.2 选择指南

- 连续参数优化：DE、PSO、CMA-ES
- 组合优化：ACO、GA
- 自动程序设计：GP
- 需要可解释性：GP、符号回归
- 动态环境：ACO、PSO
- 高维问题：DE、CMA-ES
- 计算资源有限：PSO、简单 GA

8.4.3 未来发展方向

1. 混合算法：结合不同算法的优势

- ACO+ 局部搜索：增强局部搜索能力
 - GP+ 神经网络：结合符号和连接主义
 - 进化 + 强化学习：进化神经网络结构
2. 大规模并行：利用 GPU、分布式计算加速
 3. 自适应机制：自动调整参数和策略
 4. 理论分析：深入理解算法收敛性和复杂性
 5. 新应用领域：量子计算、生物信息、金融科技
 6. 可解释 AI：GP 在可解释机器学习中的应用
 7. 自动机器学习（AutoML）：自动选择算法、调参

8.4.4 实践建议

1. 理解问题本质：分析问题特点，选择合适算法
2. 从简单开始：先尝试简单算法和默认参数
3. 逐步优化：根据结果调整参数和策略
4. 多次运行：统计多次运行结果，评估稳定性
5. 可视化分析：可视化搜索过程，理解算法行为
6. 记录实验：详细记录实验设置和结果
7. 理论结合实践：理解算法原理，指导实践应用

进化计算作为一类受自然启发的优化方法，在解决复杂问题中展现出强大能力。ACO 和 GP 分别针对组合优化和程序生成问题，扩展了进化计算的应用范围。理解这些算法的原理、特点和应用场景，对于解决实际问题具有重要意义。随着计算技术的进步和理论的发展，进化计算将继续在科学和工程领域发挥重要作用。

第九章 学习辅助的自动算法设计

9.1 引言：优化算法的根本局限与自动算法设计的必要性

9.1.1 没有免费的午餐定理：优化算法的根本局限性

在优化理论中，有一个深刻而基本的定理——没有免费的午餐定理（No Free Lunch Theorem, NFL），由 Wolpert 和 Macready 于 1997 年正式提出。这个定理揭示了所有优化算法的根本局限性。

定理 9.1.1 (没有免费的午餐定理). 对于任意两个优化算法 A 和 B ，在所有可能的优化问题上的平均性能是相等的。换句话说，不存在一个算法在所有问题上都比其他算法表现得更好。

数学形式化：设 \mathcal{F} 是所有可能的目标函数集合， d_m^y 是在 m 次评估后得到的观察序列， $P(d_m^y|f, m, A)$ 是在算法 A 和函数 f 下得到序列 d_m^y 的概率，则：

$$\sum_f P(d_m^y|f, m, A) = \sum_f P(d_m^y|f, m, B)$$

对任意算法 A, B 成立。

直观理解：

- 如果一个算法在某些问题上表现优异，必然在其他问题上表现较差
- 算法的“优异”是相对于特定问题类而言的
- 没有“万能”的优化算法

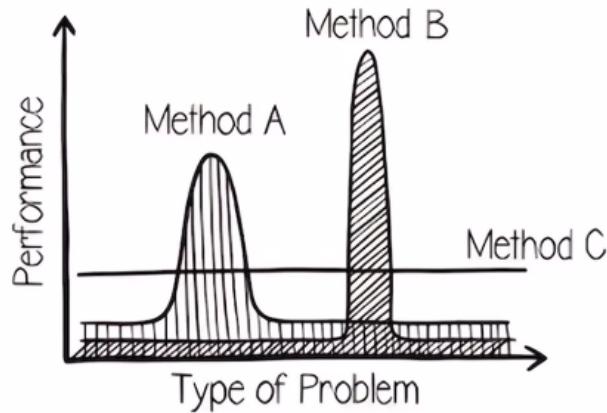


图 9.1: 没有免费的午餐定理示意图: 不同的算法在不同类型的问题上表现不同, 没有绝对的“最佳算法”

9.1.2 NFL 定理的启示与自动算法设计的动机

NFL 定理告诉我们, 为了在特定问题上获得最佳性能, 我们必须根据问题的特性设计或选择合适的算法。这引出了自动算法设计 (Automated Algorithm Design, AAD) 的核心动机:

1. 问题适应性: 算法应该能够适应具体问题的特性
2. 自动调参: 避免繁琐的手工参数调整
3. 算法组合: 结合不同算法的优势
4. 元优化: 在算法设计层面进行优化

核心思想: 与其寻找“万能算法”, 不如开发能够自动为给定问题设计和调整算法的方法。

9.2 传统算法设计方法与元学习

9.2.1 经验法则方法

经验法则 (Rule-based Approaches) 是基于领域专家知识和经验的设计方法。

定义 9.2.1 (经验法则方法). 基于对问题特性和算法行为的先验知识, 制定一系列规则来选择、配置或组合算法。

典型方法:

- **规则库:** 建立“如果-那么”规则
- **启发式规则:** 基于经验总结的简单规则
- **案例推理:** 参考历史类似问题的解决方案

示例:

- 如果问题维度高, 则使用差分进化
- 如果问题光滑, 则使用梯度下降

- 如果问题有约束，则使用罚函数法

优点:

- 简单直观
- 实现容易
- 可解释性强

缺点:

- 依赖专家知识
- 难以覆盖复杂情况
- 缺乏自适应能力
- 可能不适用于新问题

9.2.2 元学习方法

元学习 (Meta-Learning) 或“学会学习” (Learning to Learn)，是使用机器学习方法来自动生成学习算法设计策略。

定义 9.2.2 (元学习). 从多个学习任务中提取经验，学习如何快速适应新任务的方法。在自动算法设计中，指从多个优化问题中学习如何为新的优化问题选择、配置或设计算法。

核心思想:

- 从历史问题-算法性能数据中学习
- 建立问题特征与算法性能的映射
- 为新的问题推荐合适的算法

基本框架:

1. **特征提取:** 从问题中提取特征
2. **数据收集:** 收集问题特征与算法性能数据
3. **模型训练:** 训练预测模型
4. **预测应用:** 为新问题预测最佳算法

关键技术:

- **问题特征化:** 如何有效描述优化问题
- **性能预测模型:** 如何准确预测算法性能
- **特征选择:** 选择最具判别性的特征

优点:

- 自动化程度高
- 适应性强
- 不依赖专家知识

缺点:

- 需要大量训练数据
- 特征工程复杂

- 模型可解释性差

9.3 学习辅助的优化：L2O 与 NCO

9.3.1 L2O 与 NCO 的提出背景

2016 年前后，两个开创性的研究方向几乎同时出现：

- L2O (Learning to Optimize): 让机器学习如何优化
- NCO (Neural Combinatorial Optimization): 用神经网络解决组合优化

这两个方向共同开启了“让机器自动学习如何优化”的新时代。

9.3.2 神经组合优化 (NCO)

定义 9.3.1 (神经组合优化). 使用神经网络自动学习解决组合优化问题的启发式策略，通过端到端训练直接生成高质量解。

核心特点：

- 专门针对组合优化问题
- 使用编码器-解码器架构
- 利用注意力机制
- 通过强化学习训练

NCO 的基本架构

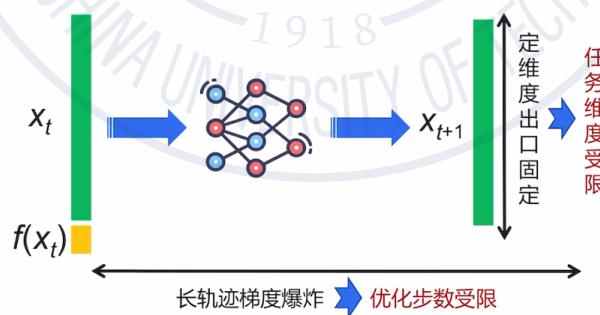


图 9.2: NCO 基本架构：编码器理解问题全局，解码器逐步构建解，注意力机制动态选择

编码器：

- 输入：问题实例（如城市坐标）
- 输出：问题表示向量
- 常用：图神经网络、Transformer

解码器：

- 输入：当前部分解和编码器输出
- 输出：下一步动作的概率分布
- 常用：RNN、LSTM、注意力机制

NCO 的训练方法

强化学习训练：

$$J(\theta) = \mathbb{E}_{p_\theta(\pi|s)}[R(\pi)]$$

其中 $R(\pi)$ 是解 π 的奖励（如路径长度的负值）。

策略梯度更新：

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta(\pi|s)}[(R(\pi) - b)\nabla_\theta \log p_\theta(\pi|s)]$$

b 是基线，用于减小方差。

NCO 的优势与局限

优势：

- 推理速度快（毫秒级）
- 可学习复杂启发式
- 端到端训练
- 可迁移性

局限：

- 训练计算量大
- 泛化能力有限
- 可解释性差
- 对问题规模敏感

9.3.3 基于学习的优化 (L2O)

定义 9.3.2 (基于学习的优化). 使用机器学习方法（特别是深度学习）来学习优化算法的更新规则，替代传统的手工设计优化器。

与 NCO 的区别：

- L2O 更广义，可处理连续和离散优化
- L2O 关注优化过程，而非直接生成解
- L2O 可视为“优化器的优化器”

L2O 的基本思想

传统优化器（如梯度下降）：

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

L2O 优化器：

$$x_{t+1} = x_t + g_\theta(\nabla f(x_t), H_t)$$

其中 g_θ 是学习的更新函数， H_t 是历史信息。

L2O 的典型方法

1. RNN 优化器：使用 RNN 学习更新规则
2. 元学习优化器：学习快速适应的优化器
3. 学习率调度器：学习最优的学习率调整策略

9.4 元黑箱优化：自动算法设计的统一框架

9.4.1 元黑箱优化的核心思想

元黑箱优化（Meta Black-Box Optimization, MetaBBO）是自动算法设计的统一框架。

定义 9.4.1 (元黑箱优化). 在元层次上学习如何优化，关注点从解决单一问题上升到学习能够处理一类问题的通用优化策略，优化的对象是黑箱函数。

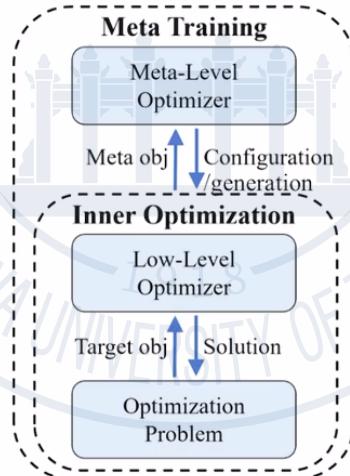


图 9.3: 元黑箱优化框架：在元层次学习优化策略，应用于底层黑箱优化问题

9.4.2 MetaBBO 的数学形式化

MetaBBO 的目标是学习一个策略 π_θ ，使其在一类问题 \mathcal{P} 上期望性能最优：

$$\max_{\theta} J(\theta) = \mathbb{E}_{f \sim \mathcal{P}}[R(\mathcal{A}, \pi_\theta, f)]$$

其中：

- \mathcal{A} : 底层优化器（可以是已有算法、算法池或生成的算法）

- π_θ : 元策略, 参数为 θ
- f : 目标函数 (黑箱)
- R : 性能度量

状态-动作-奖励框架

在每次迭代 t , 对于问题 f_i :

$$\text{状态} : s_i^t = \text{sf}(\mathcal{A}, f_i, t) \quad (9.1)$$

$$\text{动作} : a_i^t = \pi_\theta(s_i^t) \quad (9.2)$$

$$\text{奖励} : r_i^t = \text{perf}(\mathcal{A}, a_i^t, f_i) \quad (9.3)$$

其中:

- $\text{sf}(\cdot)$: 状态提取函数
- $\text{perf}(\cdot)$: 性能评估函数

9.4.3 MetaBBO 的学习方式分类

根据学习方式, MetaBBO 可以分为四类:

强化学习方法

将算法设计过程建模为马尔可夫决策过程 (MDP):

- 状态: 问题特征 + 优化状态
- 动作: 算法设计决策
- 奖励: 性能改进
- 策略: 状态到动作的映射

优势:

- 适用于序列决策
- 可处理延迟奖励
- 探索-利用平衡

挑战:

- 奖励稀疏
- 状态空间大
- 训练不稳定

监督学习方法

从历史数据中学习算法设计模式:

- 输入: 问题特征

- **输出:** 最佳算法/参数
- **损失函数:** 预测误差

优势:

- 训练稳定
- 样本效率高
- 实现简单

挑战:

- 需要标注数据
- 泛化能力有限
- 难以处理动态调整

进化学习方法

使用进化学算法搜索最优算法设计:

- **个体表示:** 算法结构/参数
- **适应度:** 算法性能
- **进化操作:** 选择、交叉、变异

优势:

- 无需梯度
- 全局搜索能力强
- 可发现新颖设计

挑战:

- 计算代价高
- 收敛速度慢
- 可解释性差

上下文学习方法（大语言模型）

利用大语言模型的推理能力进行算法设计:

- **提示工程:** 设计合适的提示
- **上下文学习:** few-shot 或 zero-shot 学习
- **思维链:** 逐步推理

优势:

- 无需训练
- 泛化能力强
- 可解释性好

挑战:

- 计算成本高

- 结果不稳定
- 需要精心设计提示

9.4.4 基于元层学习范式的分类

从输出形式角度, MetaBBO 可分为两类:

直接生成答案

系统直接输出完整的算法实体或具体建议:

1. 基于搜索/演化
 - 代表: 遗传编程
 - 特点: 离线搜索, 代价高但一次完成
 - 输出: 优化算法程序
2. 基于即时推理
 - 代表: 大语言模型上下文学习
 - 特点: 在线推理, 快速灵活
 - 输出: 算法建议或代码

生成解题方法

系统生成解决问题的策略或方法:

1. 基于学习
 - 代表: 强化学习、监督学习
 - 特点: 学习优化策略
 - 输出: 决策函数

9.5 自动算法设计的具体技术

9.5.1 自动算法选择 (Algorithm Selection, AS)

定义 9.5.1 (自动算法选择). 给定问题实例和算法集合, 自动选择最适合该问题的算法。

问题形式化

设:

- \mathcal{I} : 问题实例集合
- $\mathcal{A} = \{A_1, A_2, \dots, A_k\}$: 算法集合
- $c : \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$: 性能度量函数

目标是找到映射 $s : \mathcal{I} \rightarrow \mathcal{A}$, 使得:

$$\min_s \mathbb{E}_{i \sim \mathcal{I}} [c(i, s(i))]$$

关键技术

1. 特征工程

- 问题特征: 维度、线性性、凸性等
- 实例特征: 具体数值特征
- 元特征: 易于计算的统计特征

2. 性能预测

- 回归模型: 预测运行时间/解质量
- 分类模型: 预测最佳算法
- 排序模型: 预测算法排序

3. 选择策略

- 单算法选择: 选择预测最佳的算法
- 算法调度: 不同阶段使用不同算法
- 集成选择: 结合多个算法的预测

典型方法

1. 基于实例的方法

- k-近邻: 找到相似实例, 使用其最佳算法
- 聚类分析: 将实例聚类, 每类有推荐算法

2. 基于模型的方法

- 决策树: 可解释性强
- 随机森林: 鲁棒性好
- 神经网络: 表达能力强

3. 基于性能模型的方法

- 回归模型: 预测算法性能
- 排序学习: 学习算法排序

评估指标

- 选择准确率: 选择正确算法的比例
- 性能损失: 相对于最优算法的性能差距
- 计算开销: 特征提取和选择的时间

9.5.2 自动算法配置 (Algorithm Configuration, AC)

定义 9.5.2 (自动算法配置). 给定算法和参数空间, 自动找到使算法在目标问题类上性能最优的参数配置。

问题形式化

设:

- \mathcal{A}_θ : 参数化算法, $\theta \in \Theta$
- \mathcal{I} : 问题实例集合
- $c : \Theta \times \mathcal{I} \rightarrow \mathbb{R}$: 性能度量

目标是找到:

$$\theta^* = \arg \min_{\theta \in \Theta} \mathbb{E}_{i \sim \mathcal{I}}[c(\theta, i)]$$

配置空间设计

1. 参数类型

- 连续参数: 实数, 有界或无界
- 离散参数: 整数或枚举值
- 条件参数: 依赖其他参数的值

2. 空间结构

- 层次结构: 参数间有依赖关系
- 条件结构: 某些参数只在特定条件下有效

配置方法

1. 基于模型的方法

- 贝叶斯优化: 建立代理模型, 指导搜索
- 随机森林: 处理混合类型参数
- 高斯过程: 处理连续参数

2. 基于序列的方法

- 序列模型优化: 迭代构建和优化模型
- 超带: 早停机制, 提高效率

3. 基于进化的方法

- 协方差矩阵自适应: 处理连续参数
- 遗传算法: 处理混合类型参数

配置策略

1. **默认配置**: 为算法找到通用良好的配置
2. **实例特定配置**: 为每个实例单独配置

3. 配置调度：不同阶段使用不同配置

评估方法

- 交叉验证：在实例集合上评估
- 统计检验：比较配置性能
- 成本效益分析：考虑配置时间和性能提升

9.5.3 自动算法生成（Algorithm Generation, AG）

定义 9.5.3 (自动算法生成). 自动创建全新的优化算法，而不仅仅是选择或配置现有算法。

生成空间

算法生成空间包括：

- 组件库：基本操作和函数
- 组合规则：如何组合组件
- 控制结构：循环、条件、递归
- 终止条件：停止准则

生成方法

1. 基于遗传编程的方法
 - 个体表示：算法程序树
 - 适应度：算法性能
 - 进化操作：交叉、变异
2. 基于强化学习的方法
 - 状态：当前算法部分
 - 动作：添加组件或结构
 - 奖励：最终算法性能
3. 基于神经网络的方法
 - 序列生成：按顺序生成算法组件
 - 图生成：生成算法计算图
 - 程序合成：生成完整算法代码

评估与验证

1. 功能正确性
 - 语法正确：算法可执行
 - 语义正确：算法逻辑合理

2. 性能评估

- 基准测试：标准问题集
- 鲁棒性测试：不同问题类型
- 效率测试：运行时间和内存

3. 可解释性分析

- 算法结构：是否清晰
- 组件功能：是否可理解
- 决策逻辑：是否合理

应用实例

1. 自动设计局部搜索启发式
2. 生成元启发式算法
3. 创建混合算法
4. 发现新颖优化策略

9.6 MetaBBO 的核心挑战与未来方向

9.6.1 核心挑战

挑战 1：增强算法设计 Agent 的泛化能力

1. 问题表示学习

- 人工特征的限制：不能充分表达问题本质
- 学习特征表示：自动学习有意义的特征
- 多尺度特征：捕捉问题的不同层面

2. 训练数据多样性

- 基准测试集局限：不能覆盖所有问题类型
- 问题生成：自动生成多样化训练问题
- 课程学习：从简单到复杂逐步学习

3. 跨问题迁移

- 领域自适应：适应不同问题领域
- 元迁移学习：利用相关问题的知识
- 少样本学习：在少量样本上快速适应

挑战 2：建立公平鲁棒的评估体系

1. 标准化基准

- 统一测试集：建立权威基准

- 多样化问题：覆盖不同类型和难度
- 可重复实验：提供详细实验设置

2. 评估指标

- 多维评估：性能、效率、鲁棒性
- 统计显著性：严格统计检验
- 成本效益：考虑计算资源

3. 比较框架

- 公平比较：相同计算预算
- 消融实验：分析组件贡献
- 敏感性分析：参数敏感性

9.6.2 未来研究方向

理论方向

1. 泛化理论

- 学习优化器的泛化界
- 算法设计的 PAC 学习框架
- 元学习的收敛性分析

2. 复杂性理论

- 自动算法设计的计算复杂性
- 表示能力的理论分析
- 最优算法设计的信息论下界

方法方向

1. 可解释的自动设计

- 可解释的神经网络优化器
- 模块化算法设计
- 算法设计的可视化分析

2. 高效学习框架

- 在线学习优化器
- 自适应算法设计
- 分布式自动设计

3. 多目标自动设计

- 多目标优化器的自动设计
- 权衡性能与效率
- 帕累托最优算法设计

应用方向

1. 领域特定自动设计
 - 科学计算优化器
 - 机器学习训练器
 - 工程优化算法
2. 大规模自动设计
 - 大规模优化问题
 - 分布式优化算法
 - 高维优化器设计
3. 动态环境自动设计
 - 时变问题优化器
 - 在线学习算法
 - 自适应控制系统

9.6.3 方法选择与实施

表 9.1: 自动算法设计方法选择指南

需求	推荐方法	实施难度	预期效果
快速选择算法	自动算法选择	低	中等
精细调参	自动算法配置	中	高
创造新算法	自动算法生成	高	高但风险大
在线适应	强化学习方法	高	高
少样本学习	上下文学习	中	中等

9.6.4 评估与部署

1. 离线评估
 - 交叉验证
 - 统计检验
 - 消融实验
2. 在线测试
 - A/B 测试
 - 逐步部署
 - 监控反馈
3. 持续改进
 - 收集新数据

- 更新模型
- 迭代优化

9.7 总结与展望

9.7.1 核心观点总结

1. 没有免费的午餐定理是自动算法设计的根本动机
2. 元学习提供了自动算法设计的理论基础
3. L2O 和 NCO 展示了学习辅助优化的潜力
4. MetaBBO 是自动算法设计的统一框架
5. 自动算法选择、配置、生成是具体实现技术
6. 泛化能力和评估体系是当前主要挑战

9.7.2 发展趋势

- 理论深化：建立更坚实的理论基础
- 方法融合：结合不同方法的优势
- 应用扩展：扩展到更多领域和问题
- 效率提升：降低计算成本和数据需求
- 可解释性增强：提高算法的透明度和可信度

自动算法设计代表了优化领域的未来方向，它将人工智能与优化理论深度融合，为解决复杂优化问题提供了新的可能。随着技术的不断发展，我们有理由相信，自动算法设计将在科学、工程、经济等各个领域发挥越来越重要的作用。

第十章 提纲挈领

10.1 学习与优化的宏大图景

学习与优化是人工智能领域的核心支柱，它研究如何让机器通过经验改进性能。本笔记系统梳理了从基础神经网络到前沿自动算法设计的完整知识体系，旨在揭示这些方法背后的统一思想和发展脉络。

10.1.1 学习与优化的基本问题

所有学习与优化问题都可以归结为一个核心挑战：如何在复杂环境中做出智能决策？这包括：

- **预测问题：**从数据中学习映射关系（如图像分类、语音识别）：根据已知的输入数据（如一张图片、一段语音），推断出对应的输出结果（如图片中的物体类别、语音对应的文字）。
- **决策问题：**在不确定环境中序列决策（如游戏 AI、机器人控制）
- **优化问题：**在约束条件下寻找最优解（如资源分配、路径规划）

传统方法依赖于手工设计的算法和特征，但现代方法让机器自动学习如何学习和优化自身。

10.1.2 三大支柱的融合

学习与优化的发展体现了三大思想的融合：

1. **连接主义：**神经网络模拟人脑，通过层次化表示学习复杂模式
2. **行为主义：**强化学习通过试错学习最优策略，模拟生物学习过程
3. **进化论：**进化计算受自然选择启发，通过种群进化寻找最优解

这三种范式从不同角度解决学习问题，最终在元学习框架下统一。

10.2 神经网络：深度学习的基石

10.2.1 从生物神经元到人工神经网络

人脑的运作机制启发了神经网络的基本设计。单个神经元接收输入信号，整合后超过阈值则激发。人工神经元数学化这一过程：

$$a = f(\sum w_i x_i + b)$$

其中 w_i 是权重（连接强度）， b 是偏置（激发阈值）， f 是激活函数（非线性变换）。

10.2.2 激活函数：引入非线性的关键

激活函数使神经网络能够拟合复杂函数：

- **Sigmoid:** 将输入压缩到 (0,1)，适合概率输出，但易梯度消失
- **tanh:** 输出范围 (-1,1)，零中心，缓解梯度消失但仍存在
- **ReLU:** $\max(0, x)$ ，计算简单，有效缓解梯度消失，但存在“死亡 ReLU”问题
- 改进变体：Leaky ReLU、ELU、GELU 等平衡性能与稳定性

10.2.3 前馈网络：层叠的威力

将神经元分层组织，前一层的输出作为后一层的输入，构成前馈神经网络。深度网络通过多层非线性变换学习层次化特征：

- 底层学习简单特征（边缘、纹理）
- 中层学习组合特征（形状、部件）
- 高层学习抽象概念（物体、场景）

10.2.4 反向传播：深度学习的引擎

反向传播算法高效计算损失函数对网络参数的梯度，是神经网络训练的核心：

1. **前向传播:** 输入数据通过网络，计算各层输出
2. **误差计算:** 比较网络输出与真实值，计算损失
3. **反向传播:** 利用链式法则从输出层向输入层传播误差，计算梯度
4. **参数更新:** 沿梯度反方向更新权重和偏置

这一过程反复迭代，使网络输出逐渐接近期望值。

10.2.5 过拟合与正则化

当网络过于复杂时，易记忆训练数据中的噪声而非学习一般规律，导致过拟合。应对策略：

- **数据增强**: 通过对训练数据施加变换扩充数据集
- **正则化**: 在损失函数中添加惩罚项, 限制模型复杂度
- **Dropout**: 训练时随机“丢弃”部分神经元, 强制网络学习冗余表示
- **早停法**: 监控验证集性能, 性能不再提升时停止训练

10.3 强化学习：从交互中学习

10.3.1 强化学习的基本框架

强化学习研究智能体如何通过与环境交互学习最优行为策略。其核心要素：

- **智能体 (Agent)**: 学习者或决策者
- **环境 (Environment)**: 智能体交互的外部系统
- **状态 (State)**: 环境的当前情况描述
- **动作 (Action)**: 智能体可执行的操作
- **奖励 (Reward)**: 环境对智能体动作的反馈
- **策略 (Policy)**: 状态到动作的映射函数

智能体目标是在每个状态下选择动作, 最大化长期累积奖励。

10.3.2 马尔可夫决策过程 (MDP)

MDP 为强化学习提供数学框架, 假设未来状态只依赖于当前状态和动作 (马尔可夫性)。MDP 由五元组定义: $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, 分别表示状态空间、动作空间、状态转移概率、奖励函数和折扣因子。

10.3.3 价值函数与贝尔曼方程

价值函数评估状态或状态-动作对的好坏:

- **状态价值函数 $V(s)$** : 从状态 s 开始遵循某策略的期望累积奖励
- **动作价值函数 $Q(s, a)$** : 在状态 s 执行动作 a 后遵循某策略的期望累积奖励

贝尔曼方程描述了价值函数间的递归关系, 是强化学习算法的理论基础。

10.3.4 强化学习算法分类

根据学习方式, 强化学习算法可分为:

基于价值的方法

学习价值函数, 通过价值函数导出策略, 如 Q-learning、DQN:

- **优点**: 通常更稳定, 样本效率较高

- 缺点：难以处理连续动作空间，通常得到确定性策略

深度 Q 网络 (DQN) 结合神经网络与 Q-learning，解决高维状态空间问题，引入经验回放和目标网络提升稳定性。

基于策略的方法

直接学习策略函数，如 REINFORCE、策略梯度：

- 优点：自然处理连续动作空间，可学习随机策略
- 缺点：方差大，训练不稳定，样本效率较低

Actor-Critic 方法

结合价值函数与策略函数，Actor 负责执行策略，Critic 负责评估策略：

- 优势 Actor-Critic (A2C/A3C)：多 Worker 并行收集经验
- 近端策略优化 (PPO)：通过裁剪目标函数平衡探索与利用
- 深度确定性策略梯度 (DDPG)：处理连续动作空间

10.3.5 探索与利用的权衡

强化学习核心挑战之一是如何平衡探索新行为与利用已知好行为：

- ϵ -贪婪策略：以 $1 - \epsilon$ 概率选择最优动作，以 ϵ 概率随机探索
- 乐观初始化：高估未知状态-动作对的价值，鼓励探索
- 基于不确定性的探索：根据价值估计的不确定性调整探索程度

10.4 进化计算：自然启发的优化

10.4.1 进化计算的基本思想

进化计算受生物进化机制启发，通过模拟自然选择过程解决优化问题。其核心原理：

1. 种群：一组候选解（个体）
2. 选择：适应度高的个体更可能被选中繁殖
3. 繁殖：通过交叉和变异产生新个体
4. 替代：新个体替代部分旧个体，形成新一代

这一过程反复迭代，种群整体适应度逐渐提高。

10.4.2 遗传算法 (GA)

遗传算法是进化计算的典型代表，其流程：

1. 编码：将解表示为染色体（如二进制串、实数向量）

2. **初始化:** 随机生成初始种群
3. **评估:** 计算每个个体的适应度
4. **选择:** 根据适应度选择父代（如轮盘赌选择、锦标赛选择）
5. **交叉:** 交换父代部分基因产生子代（如单点交叉、两点交叉）
6. **变异:** 以小概率随机改变基因值
7. **替代:** 用子代替换部分或全部父代

遗传算法适合黑箱优化、多峰函数优化等复杂问题。

10.4.3 差分进化 (DE)

差分进化专门针对连续优化问题，通过向量差分产生新个体：

1. **变异:** 随机选择三个个体，基于其差分向量生成变异个体
2. **交叉:** 变异个体与原个体按概率交叉
3. **选择:** 贪婪选择适应度更高的个体进入下一代

DE 参数少、实现简单，在连续优化中表现优异。

10.4.4 粒子群优化 (PSO)

PSO 模拟鸟群觅食行为，每个粒子（候选解）根据个体经验和群体经验调整飞行方向：

- **个体最佳位置:** 粒子历史上适应度最高的位置
- **群体最佳位置:** 整个群体中适应度最高的位置
- **速度更新:** 向个体最佳和群体最佳的加权方向飞行

PSO 收敛快、参数少，适合连续优化问题。

10.4.5 蚁群优化 (ACO)

ACO 模拟蚂蚁觅食行为，通过信息素正反馈寻找最优路径：

1. **路径构建:** 蚂蚁根据信息素浓度和启发式信息概率选择路径
2. **信息素更新:** 路径越短，信息素增强越多；同时信息素会挥发

ACO 在组合优化（如旅行商问题）中表现优异。

10.4.6 协方差矩阵自适应进化策略 (CMA-ES)

CMA-ES 是一种先进的进化策略，自适应调整搜索分布：

- 使用多元正态分布表示搜索分布
- 自适应调整分布均值和协方差矩阵
- 通过进化路径记录搜索方向

CMA-ES 在复杂连续优化中表现卓越，具旋转不变性。

10.5 自动算法设计：元学习的兴起

10.5.1 没有免费的午餐定理（NFL）

NFL 定理指出：没有任何优化算法在所有问题上都表现最佳。算法在某类问题上优异，必然在其他问题上较差。这揭示了自动算法设计的必要性——我们需要根据问题特性自动选择或设计合适算法。

10.5.2 元学习：学会学习

元学习旨在让机器学习如何学习，包括：

- 学习优化算法：让机器自动学习优化规则（如 L2O）
- 学习网络架构：自动设计神经网络结构（如 NAS）
- 学习超参数：自动调整算法超参数

10.5.3 学习优化（L2O）

L2O 用神经网络学习优化规则，替代手工设计的优化器（如梯度下降）：

- RNN 优化器：使用 RNN 学习参数更新规则
- 元学习优化器：学习快速适应新任务的优化器

L2O 可学习复杂优化策略，在特定问题上超越传统优化器。

10.5.4 神经组合优化（NCO）

NCO 用神经网络解决组合优化问题，端到端学习启发式策略：

- 编码器-解码器架构：编码器理解问题结构，解码器逐步构建解
- 注意力机制：动态关注问题不同部分
- 强化学习训练：通过策略梯度优化解质量

NCO 在旅行商等问题上接近专用算法性能，且推理速度极快。

10.5.5 自动算法设计的方法论

自动算法设计可分为三个层次：

自动算法选择（Algorithm Selection）

给定问题实例和算法池，自动选择最适合的算法：

1. 特征提取：从问题中提取特征（如维度、线性性）
2. 性能预测：建立特征到算法性能的映射
3. 算法推荐：为新材料推荐预测性能最佳的算法

自动算法配置 (Algorithm Configuration)

给定参数化算法，自动找到最优参数配置：

1. **参数空间定义**: 确定参数类型、范围及依赖关系
2. **配置搜索**: 使用贝叶斯优化等方法搜索最优配置
3. **性能评估**: 在验证集上评估配置性能

自动算法生成 (Algorithm Generation)

自动创建全新算法，而不仅仅是选择或配置现有算法：

1. **组件库定义**: 定义基本算法组件（如选择、交叉、变异）
2. **组合规则**: 定义组件组合方式
3. **算法搜索**: 在算法空间搜索高性能算法

10.5.6 元黑箱优化 (MetaBBO)

MetaBBO 是自动算法设计的统一框架，在元层次学习优化策略：

- **状态**: 问题特征和优化状态
- **动作**: 算法设计决策（如选择什么算法、如何配置）
- **奖励**: 优化性能改进

MetaBBO 可基于强化学习、监督学习或进化方法实现。

10.6 概念串联：学习与优化的统一视角

10.6.1 从监督学习到元学习

学习与优化的发展体现了层次的不断提升：

1. **监督学习**: 从标注数据学习输入到输出的映射
2. **强化学习**: 从交互中学习决策策略
3. **进化计算**: 通过种群进化寻找最优解
4. **元学习**: 学习如何学习，自动设计学习算法

每一层次都解决更广义的学习问题，减少对先验知识的依赖。

10.6.2 探索与利用 → 永恒主题

所有学习与优化方法都面临探索（尝试新可能性）与利用（优化已知好解）的权衡：

- **监督学习**: 正则化控制模型复杂度，平衡拟合与泛化
- **强化学习**: ϵ -贪婪、乐观初始化等平衡探索与利用
- **进化计算**: 选择压力控制收敛速度，变异引入新基因

- **自动算法设计:** 在算法空间探索新算法, 利用已知好算法

10.6.3 表示学习的关键作用

合适的表示是学习成功的关键:

- **神经网络:** 通过隐藏层学习数据的分层表示
- **强化学习:** 价值函数表示状态或状态-动作对的好坏
- **进化计算:** 编码方式影响搜索效率(如二进制编码、实数编码)
- **自动算法设计:** 如何表示算法本身成为新挑战

10.6.4 从手工设计到自动学习

学习与优化的发展趋势是从依赖专家知识到自动学习:

- **特征工程:** 手工设计特征 → 表示学习自动学习特征
- **算法选择:** 依赖专家经验 → 自动算法选择
- **超参数调优:** 网格搜索 → 贝叶斯优化自动调参
- **算法设计:** 手工设计 → 自动算法生成

这一趋势使 AI 系统越来越自主, 减少对人类专家的依赖。

10.7 未来方向与挑战

10.7.1 可解释性与可靠性

随着学习系统越来越复杂, 可解释性和可靠性成为关键挑战:

- **可解释 AI:** 理解复杂模型决策过程, 增加透明度
- **对抗鲁棒性:** 抵御恶意攻击, 保证系统安全
- **分布外泛化:** 在训练分布外数据上保持性能

10.7.2 数据效率与计算可持续性

当前许多先进方法依赖大量数据和计算资源, 提升效率至关重要:

- **小样本学习:** 从少量样本快速学习新任务
- **持续学习:** 不断学习新知识而不遗忘旧知识
- **绿色 AI:** 减少训练和推理的计算成本与能耗

10.7.3 多模态与跨领域学习

现实世界问题常涉及多种信息源和领域:

- **多模态学习:** 整合视觉、语言、语音等多种信息

- **跨领域学习：**将知识从源领域迁移到目标领域
- **多任务学习：**同时学习多个相关任务，共享表示

10.7.4 人机协作与社会影响

学习系统越来越多融入人类社会：

- **人机协作：**人类与 AI 系统协同解决问题
- **伦理对齐：**确保 AI 系统价值观与人类一致
- **普惠 AI：**让更多人从 AI 技术进步中受益

10.8 结语

学习与优化是 AI 领域最活跃和富有成果的方向之一。从模拟神经网络的连接主义，到受行为心理学启发的强化学习，再到借鉴生物进化的进化计算，这些方法从不同角度解决智能的核心问题——如何从经验中学习改进。

当前，我们正见证这些范式的深度融合。元学习框架将学习算法本身作为优化对象，使机器能够自动设计解决特定问题的最佳算法。这一趋势最终可能通向更广义的 AI——不仅能够解决特定任务，还能自动获取解决新任务的能力。

本笔记试图勾勒这一宏大图景，将分散的概念串联成连贯体系。理解这一发展脉络，不仅有助于掌握现有技术，更为迎接未来突破奠定基础。学习与优化的探索远未结束，而只是刚刚开始。