强化学习算法综合比较实验（LunarLander-v2）

1. 项目目标

在LunarLander-v2环境中实现并比较5种强化学习算法，分析其在连续状态空间问题中的表现差异，掌握不同算法范式的特点。

2. 环境说明

- 环境： `LunarLander-v2` （默认参数）
- 状态空间：8维连续向量（坐标、速度、角度等）
- 动作空间：4个离散动作（无操作、左引擎、主引擎、右引擎）
- 成功标准：单次着陆奖励 > 100分（完美着陆可达140分）

3. 算法实现要求

(1) REINFORCE

（策略梯度基础算法）

```
class REINFORCE:
    def __init__(self, state_dim, action_dim):
        self.policy_net = nn.Sequential(
            nn.Linear(state_dim, 64),
            nn.ReLU(),
            nn.Linear(64, action_dim),
            nn.Softmax(dim=-1)
        )

    def update(self, trajectories):
        """输入：完整的episode轨迹 (states, actions, rewards)"""
        # 需完成：回报计算、梯度上升
```

(2) QAC (Q Actor-Critic)

（值函数辅助的策略梯度）

```
class QAC:
    def __init__(self, state_dim, action_dim):
        self.actor = PolicyNetwork(state_dim, action_dim)  # 策略网络
        self.critic = ValueNetwork(state_dim)  # 状态值函数网络
        # 需实现：TD误差计算、策略梯度更新

    def update(self, state, action, reward, next_state):
        # 需完成：Critic的TD学习 + Actor的梯度更新
```

(3) Advantage AC

（带优势函数的Actor-Critic）

```
class A2C:
    def __init__(self, state_dim, action_dim):
        self.actor = PolicyNetwork(state_dim, action_dim)
        self.critic = ValueNetwork(state_dim)

    def compute_advantage(self, rewards, values):
        # 需完成：优势函数计算
```

(4) Natural AC

（自然策略梯度）

```
class NaturalAC:
    def __init__(self, state_dim, action_dim):
        # 需实现：Fisher信息矩阵近似
        self.fisher_matrix = torch.zeros((param_dim, param_dim))

    def natural_gradient(self, gradients):
        # 需完成：共轭梯度法求解
```

(5) Model-Based RL (Dyna-Q）

```
class DynaQ:
    def __init__(self, state_dim, action_dim, bins=[10, 10, 8, 8, 6, 6]):
        # 状态离散化工具
        self.discretizer = Discretizer(bins)

        # Q-table（离散状态空间）
        self.q_table = np.zeros((*bins, action_dim))

        # 环境模型 (s,a) -> (r, s')
        self.model = defaultdict(dict)

        # 参数设置
        self.alpha = 0.1  # 学习率
        self.gamma = 0.99  # 折扣因子
        self.epsilon = 1.0  # 探索率
        self.planning_steps = 5  # 每次交互后的规划步数

    def get_discrete_state(self, state):
        """连续状态离散化"""

    def update(self, s, a, r, s_, done):
        """真实经验学习 + 模型规划"""

    def _planning(self):
        """从模型随机采样进行规划"""

    def choose_action(self, state, training=True):
        """ε-贪婪策略"""
```

4. 实验要求

(1) 统一实验设置

- 训练episodes：2000次（Dyna-Q可适当增加）
- 评估频率：每100 episodes运行10次测试取平均
- 超参数范围：

```
learning_rate = [0.001, 0.01, 0.1]
discount_factor = 0.99
exploration = LinearDecay(1.0, 0.01, 1000)
```

(2) 评估指标

指标	计算方法
平均奖励	最近100 episodes的均值
成功率	奖励>100的episode比例
收敛速度	达到80%成功率所需episodes
样本效率	相同样本量下的性能对比

(3) 可视化要求

```
def plot_results(algorithms):
    """生成以下图表：
    1. 训练曲线（奖励vs episodes）
    2. 最终性能箱线图
    3. 策略可视化对比（同一初始状态的着陆轨迹）
    """
```

5. 报告内容

1. **算法对比分析**

算法	优点	缺点	适用场景
REINFORCE	***	***	***
***	***	***	***

2. **关键发现**：
   - 优势函数如何降低方差
   - 模型学习对样本效率的影响
   - 自然梯度与普通策略梯度的收敛性差异

6. 评分标准

部分	分值	要求
代码实现	40	5种算法完整实现，结构清晰
实验设计	20	控制变量合理，超参数选择有说明
分析深度	30	能解释性能差异的理论原因
可视化	10	图表专业，包含必要标注

项目交付物

- **代码**：完整实现（.py或.ipynb）
- **报告**：PDF（含实验设计、结果分析、可视化图表）
- **演示**：1分钟视频展示最佳策略