LBOPB 离散版 SAC (Discrete SAC) 算法需求描述

- 作者：GaoZheng
- 日期：2025-10-23
- 版本：v1.0.0

注：“O3理论/O3元数学理论/主纤维丛版广义非交换李代数(PFB-GNLA)”相关理论参见： 作者（GaoZheng）网盘分享 或 作者（GaoZheng）开源项目 或 作者（GaoZheng）主页，欢迎访问！

摘要

本文**离散版 Soft Actor-Critic (SAC)**——制定了详细的技术需求。其核心目标是将为连续动作空间设计的标准 SAC 算法，成功适配到 LBOPB 框架特有的、由海量离散生物学算子构成的动作空间中。

---

1. 概述 (Overview)

1.1. 项目目标:
本项目旨在实现一个离散版本的 Soft Actor-Critic (SAC) 算法，作为 GRL-Prophet 框架的核心强化学习引擎。该引擎将学习在一个由 LBOPB（生命总算子主纤维丛）定义的、高维度的、离散的算子空间中，生成最优的算子序列，以实现预设的生物学或药理学目标。

1.2. 核心挑战:
标准的 SAC 算法是为连续动作空间（如机器人关节控制）设计的。而 LBOPB 的动作空间是由海量的、离散的生物学基础算子（如 `Inflammation`，`Bind`，`Dose` 等）构成的。因此，核心需求是将 SAC 的关键组件（策略网络、Q 网络、熵最大化机制）从连续域无损地迁移到离散域。

1.3. 算法定位:
该算法是一个**离策略（Off-Policy）**、**最大熵（Maximum Entropy）**的深度强化学习算法，专注于在复杂环境中进行高效的数据利用和鲁棒的探索。

---

2. 核心算法组件 (Core Algorithm Components)

该算法由三个主要的神经网络构成：策略网络（Actor）、Q 网络（Critic）和目标 Q 网络。

2.1. 策略网络 (Actor - The Policy Network)

- **功能**: 决定在给定状态下，选择每一个离散算子的概率。
- **输入**:
  - `state` (状态向量): 一个高维浮点数向量，代表了 LBOPB 环境中所有七个幺半群当前状态的综合表示。
- **网络结构**: 一个多层感知机（MLP）或更复杂的网络（如 Transformer Encoder）。
- **输出**:
  - `action_logits` (动作 logits): 一个维度为 `N_actions` 的向量，其中 `N_actions` 是离散算子的总数量。该向量的每一个元素对应一个算子的原始分数。
- **激活函数**: 输出层之后必须连接一个 **Softmax** 函数，将 `logits` 转换为一个合法的**概率分布** `π(a|s)`，表示在状态 `s` 下选择动作 `a` 的概率。

2.2. Q 网络 (Critic - The Q-Network)

- **功能**: 评估在给定状态下，执行每一个离散算子的长期价值（Q-value）。
- **输入**:
  - `state` (状态向量): 与策略网络相同的状态向量。
- **网络结构**: 一个多层感知机（MLP）或与策略网络相似的结构。
- **输出**:
  - `q_values` (Q 值向量): 一个维度为 `N_actions` 的向量。该向量的每一个元素 `q_values[i]` 代表了在当前状态下，执行第 `i` 个算子的 Q 值。
- **设计模式**: 采用**双 Q 网络（Double Q-Networks）**设计，即同时训练两个结构相同的 Q 网络（Q1 和 Q2），以缓解 Q 值的高估问题。在计算目标 Q 值时，取两者中较小的一个。

2.3. 目标 Q 网络 (Target Q-Networks)

- **功能**: 在计算目标 Q 值时，提供一个稳定的、更新较慢的网络，以增强训练的稳定性。
- **结构**: 与 Q 网络（Q1 和 Q2）完全相同。
- **更新机制**: 采用**软更新（Soft Update / Polyak Averaging）**。在每次训练迭代后，目标 Q 网络的权重 `w_target` 会缓慢地向主 Q 网络的权重 `w_main` 靠拢：`w_target ← τ * w_main + (1 - τ) * w_target`，其中 `τ` 是一个很小的超参数（如 0.005）。

---

3. 关键机制适配 (Key Mechanism Adaptations)

3.1. 熵计算 (Entropy Calculation)

- **需求**: SAC 的核心是最大化策略的熵，以鼓励探索。对于离散（分类）分布，熵的计算公式为：
  `H(π(·|s)) = -Σ [π(a|s) * log(π(a|s))]`
- **实现**: 在策略网络的损失函数中，需要加入这一熵项，并通过一个可自动调整的温度系数 `α` 来加权。 `α` 的目标是维持策略熵在一个目标水平。

3.2. Q 值与 V 值的计算 (Q-value and V-value Calculation)

- **V 值 (状态价值)**: 在离散 SAC 中，状态 `s` 的价值 `V(s)` 可以直接通过 Q 值和策略概率计算得出，无需单独的 V 网络：
  `V(s) = Σ [π(a|s) * (Q(s,a) - α * log(π(a|s)))]`
  这可以被看作是策略 `π` 在所有动作上的期望 Q 值，并减去策略本身的熵。
- **目标 Q 值 (Target Q-value)**: 用于训练 Critic 网络的“真值”标签 `y` 的计算方式为：
  `y(r, s') = r + γ * V(s')`
  其中 `r` 是即时奖励， `γ` 是折扣因子， `V(s')` 是下一个状态的价值。

3.3. 策略网络损失 (Actor Loss)

- **需求**: 策略网络的目标是更新参数，使其输出的动作能够获得更高的 Q 值。其损失函数为：
  `Loss_actor = E [α * log(π(a|s)) - Q(s,a)]`
  直观上，对于一个给定的状态，如果某个动作的 Q 值高于当前策略的平均价值，策略网络就会调整参数以增加选择该动作的概率。

---

4. 环境接口 (Environment Interface)

Discrete SAC 算法需要与 LBOPB 模拟环境进行交互，交互的数据格式必须标准化。

- `state` **(状态)**: LBOPB 环境在每一步必须提供一个标准化的、固定维度的向量来描述其“立体状态”。这需要一个函数 `vectorize_state(pem_state, pdem_state, ...)` 。
- `action` **(动作)**: 算法将输出一个整数 `a` ，代表被选中的算子在全局算子列表中的索引。
- `reward` **(奖励)**: LBOPB 环境在执行一个动作后，必须返回一个标量浮点数作为奖励。该奖励应基于各幺半群 `risk` 和 `cost` 的变化量来设计，以提供稠密的奖励信号。
- `next_state` **(下一状态)**: 执行动作后的新状态向量。
- `done` **(完成标志)**: 一个布尔值，表示一个序列（Episode）是否结束（例如，达到目标状态或达到最大步数）。

---

5. 训练流程 (Training Loop)

1. **初始化**: 创建策略网络、两个 Q 网络、两个目标 Q 网络，以及一个空的经验回放池（Replay Buffer）。
2. **数据收集**:
   - **离线监督集构建（用于训练）**：从 `operator_crosswalk.json` 按规则节选，生成 `operator_crosswalk_train.json` 专用于训练；内容应覆盖“最优路径（算子包）”、以及“次优路径（算子包）”与“瑕疵路径（算子包）”。该文件仅用于预热策略与预填回放池，避免与评测数据混淆。
   - **在线交互**: 智能体根据当前策略 `π(·|s)` 选择一个动作 `a`，与环境交互，获得 `(s, a, r, s', d)` 五元组，并将其存入回放池。
3. **采样与训练**:
   - 当回放池中的数据量足够时，每次迭代从中随机采样一个批次（mini-batch）的经验。
   - 使用采样的批次数据，根据上述损失函数，更新两个 Q 网络和策略网络的参数。
   - 软更新目标 Q 网络的参数。
   - (可选) 更新温度系数 `α`。
4. **循环**: 重复步骤 2 和 3，直到算法收敛或达到最大训练步数。

---

6. 超参数 (Hyperparameters)

- `learning_rate`: 学习率 (Actor, Critic, Alpha)
- `gamma`: 折扣因子
- `buffer_size`: 经验回放池大小
- `batch_size`: 批处理大小
- `tau`: 目标网络软更新系数
- `target_entropy`: 目标熵 (用于自动调整 `α`)
