强化学习与大模型推荐系统

刘泉亮 史晓雨 尚明生 编著

第 7 章　处理大规模离散动作空间

强化学习算法主要作用于推荐链路的排序阶段，负责对多路召回生成的候选项目集进行评分和排序，并产生推荐结果。在典型工业场景中，候选项目集的规模通常达到上万量级，构成了一个规模很大的离散动作空间，每个候选项目对应着强化学习框架中的一个离散动作，强化学习推荐系统需要评估每个候选动作的价值。在这样大规模的空间中寻找最优策略需要特殊的算法设计，否则难以承受遍历评估每一个项目的计算成本。

推荐系统的复杂性还体现在其结果需要返回 Top-K 项目列表，这意味着算法需要在每次推荐过程中连续做出多个决策，每一次决策都面临着高计算成本的挑战。此外，由于用户兴趣和项目集的动态变化，模型还需要适应不断演变的状态和动作空间。这些复杂因素叠加导致的高强度计算需求，不仅会导致系统延迟超出可接受范围（通常要求在几十到几百毫秒内完成），还会大量增加计算资源消耗，影响服务成本和用户体验。

本章将讲解 2 类可以用于处理强化学习推荐系统中大规模离散动作空间问题的方法：

1）介绍通过动作表征来向量化大规模离散动作空间的方法，策略不再按价值选择具体推荐项，而是先生成一个目标向量，然后在动作空间中定位与之最匹配的推荐项。

2）介绍通过构建树型分层策略来解构大规模离散动作空间的方法，一次推荐被视为沿着一棵分层聚类树从根结点逐层行进到某个叶节点（推荐项）的系列路径选择。

7.1 动作表征学习

7.1.1 表征空间近邻

表征空间近邻法指的是 DeepMind 的研究者们提出的 Wolpertinger 方法，其名称的含义与《封神演义》中的“四不像”类似，大概说明了它是一种由不同技术（连续空间的策略、k 近邻搜索等）组合在一起的方法。这里为了突出 Wolpertinger 方法的重点，就将它简单地称为“表征空间近邻法”。

1）表征空间近邻法的基本原理。传统方法考虑在一个大规模的离散动作空间中，采用基于值函数的方法进行推荐，动作值函数 $Q(s,a)$ 计算每个候选动作的价值，然后选取其中价值最高的项目展示给用户：

$$\pi_Q(s) = \arg\max_{a \in \mathcal{A}} Q(s,a) \tag{7-1}$$

若候选项目有一万个，那么公式（7-1）中就需要执行一万次计算才能确定最优项目。表征空间近邻法则是将动作从离散空间映射到连续表征空间，不再使用索引来直接指示动作，而是将

动作向量化。若把动作空间限定到一个低维（例如 10 维）向量中，那么每次推荐只需要输出一个低维向量就能找到最优项目，避免逐一评估动作价值带来的计算开销。

这就需要强化推荐算法从基于值函数的方法转向 AC 框架。确定性策略梯度方法 DDPG 很适合这种问题定义，其中 Actor 负责生成动作向量，Critic 负责评估动作向量的价值，二者通过迭代更新不断优化。不过，这种将离散动作映射到连续空间的做法还面临一个问题：策略生成的连续动作向量可能与任何真实的动作向量都不吻合。这种情况下，算法还需要在连续空间中找到与所生成动作向量的最相近的真实动作向量，这便是“近邻”的意义所在。

设动作表征空间为 $\mathbb{R}^n$，$f_{\boldsymbol{\theta}^\pi}$ 是由以 $\boldsymbol{\theta}^\pi$ 为参数的 Actor 网络，代表由状态空间 $\mathcal{S}$ 到动作表征空间 $\mathbb{R}^n$ 的映射。那么对于一个状态 s，可以映射到虚拟动作 $\hat{\boldsymbol{a}}$：

$$\begin{aligned} f_{\boldsymbol{\theta}^\pi} &: \mathcal{S} \to \mathbb{R}^n \\ f_{\boldsymbol{\theta}^\pi}(s) &= \hat{\boldsymbol{a}} \end{aligned} \tag{7-2}$$

其中，动作 $\hat{\boldsymbol{a}}$ 就是策略生成的连续动作向量，它大概率不对应任何一个真实动作向量，即 $\hat{\boldsymbol{a}} \notin \boldsymbol{A}$。

为了解决这个问题，我们可以设置一个 k 近邻映射 g 将 $\hat{\boldsymbol{a}}$ 从连续空间还原到离散空间：

$$\begin{aligned} g &: \mathbb{R}^n \to \boldsymbol{A} \\ g_k(\hat{\boldsymbol{a}}) &= \arg\max_{\boldsymbol{a}\in\boldsymbol{A}}{}^k \|\boldsymbol{a} - \hat{\boldsymbol{a}}\|_2 \end{aligned} \tag{7-3}$$

其中，g_k 用于获取与 $\hat{\boldsymbol{a}}$ 的距离最近的 k 个动作，当 k =1 时相当于查找最近邻动作。

同时，必须考虑到并不是所有近邻动作都符号用户的需求，因此还要使用 Critic 网络来评估这些近邻动作的价值，然后输出其中价值最高的动作：

$$\pi_{\boldsymbol{\theta}}(s) = \arg\max_{\boldsymbol{a}\in g_k \circ f_{\boldsymbol{\theta}^\pi}(s)} Q_{\boldsymbol{\theta}^Q}(s, \boldsymbol{a}) \tag{7-4}$$

其中，$\boldsymbol{\theta}^\pi$ 是 Actor 网络的参数，$\boldsymbol{\theta}^Q$ 是 Critic 网络的参数。

公式（7-4）只需要在 k 个最近邻动作上评估价值，相比于公式（7-1）遍历整个动作空间的计算成本小了很多。

图 7-1 描述了 Wolpertinger 架构生成动作的过程。Actor 网络生成一个虚拟动作后，从所有真实动作的嵌入向量中获取 k 个近邻的动作，然后评估这些近邻动作的价值，并选择其中价值最高的动作。近邻动作的数量 k 是自定义的超参数，可以通过调整它来权衡策略质量和评估复杂度。

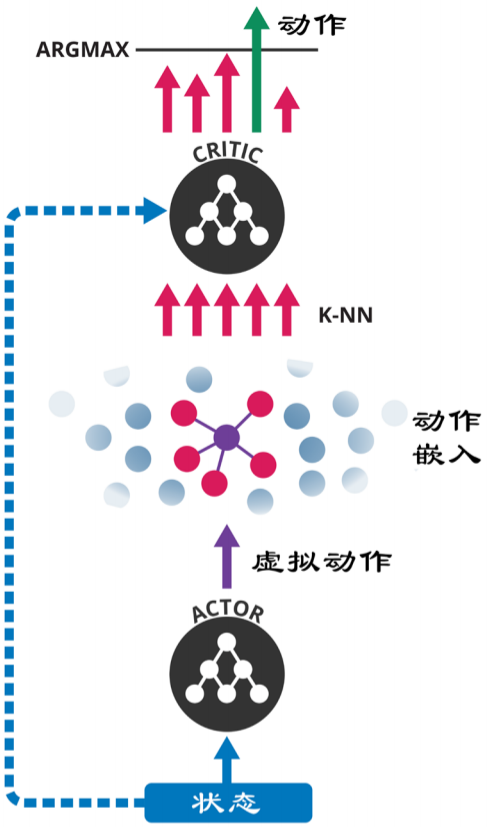

图 7-1 Wolpertinger 架构生成动作的过程

2）表征空间近邻法的训练。该算法的训练目的是找到最优策略 π_{θ^*}，以最大化一个回合内的预期回报：

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}[G \mid \pi_{\boldsymbol{\theta}}] \tag{7-5}$$

利用 DDPG 算法通过训练 f_{θ^π} 和 Q_{θ^Q}，首先，Critic 根据经验池中的样本来学习 Q 函数，经验池中样本的真实动作由策略 $\pi_{\boldsymbol{\theta}}(s)$ 生成。在 Actor 计算策略梯度时则使用虚拟动作，即 $\hat{a} = f_{\theta^\pi}(s)$。真实动作在连续空间中的映射是可以是预训练的，也可以在学习的过程中同步训练。

DDPG 算法的训练流程如表 7-1 所示。

表 7-1 DDPG 算法的训练流程

DDPG 算法的训练流程
1. 使用权重 $\boldsymbol{\theta}^Q$ 和 $\boldsymbol{\theta}^\pi$ 初始化 Critic Q_{θ^Q} 和 Actor f_{θ^π}；
2. 使用权重 $\boldsymbol{\theta}^{Q'} \leftarrow \boldsymbol{\theta}^Q$ 和 $\boldsymbol{\theta}^{\pi'} \leftarrow \boldsymbol{\theta}^\pi$ 初始化目标网络 Q_{θ^Q} 和 f_{θ^π}；
3. 初始化映射 g 和回放池 $\mathcal{B}$；
4. **for** 回合=1,M **do**
5. **for** 时间步=1, T **do**
6. 根据当前策略和探索方法选择动作 $a_t = \pi_{\boldsymbol{\theta}}(s_t)$；
7. 执行动作 a_t 并获取奖励 r_t，转移到新状态 s_{t+1}；

8. 存储轨迹 (s_t, a_t, r_t, s_{t+1}) 到 $\mathcal{B}$ 中;
9. 从 $\mathcal{B}$ 中随机采样 N 个小批次的轨迹 (s_i, a_i, r_i, s_{i+1});
10. 计算 TD 目标 $y_i = r_i + \gamma Q_{\boldsymbol{\theta}^{Q'}}(s_{i+1}, \pi_{\boldsymbol{\theta}'}(s_{i+1}))$;
11. 最小化损失来更新 Critic:

$$\mathcal{L}(\boldsymbol{\theta}^Q) = \frac{1}{N}\sum_i [y_i - Q_{\boldsymbol{\theta}^Q}(s_i, a_i)]^2$$

12. 根据采样梯度来更新 Actor:

$$\nabla_{\boldsymbol{\theta}^\pi} f_{\boldsymbol{\theta}^\pi}|_{s_i} = \frac{1}{N}\sum_i \nabla_a Q_{\boldsymbol{\theta}^Q}(s, \hat{a})|_{\hat{a}=f_{\boldsymbol{\theta}^\pi}(s_i)} \cdot \nabla_{\boldsymbol{\theta}^\pi} f_{\boldsymbol{\theta}^\pi}(s)|_{s_i}$$

13. 更新目标网络:

$$\boldsymbol{\theta}^{Q'} \leftarrow \tau\boldsymbol{\theta}^Q + (1-\tau)\boldsymbol{\theta}^{Q'}$$
$$\boldsymbol{\theta}^{\pi'} \leftarrow \tau\boldsymbol{\theta}^\pi + (1-\tau)\boldsymbol{\theta}^{\pi'}$$

14. **end for**
15. **end for**

3）表征空间近邻法的效率与性能。表征空间近邻法的时间复杂度与近邻动作的数量 k 呈线性关系。k 降低到一定程度虽然可以极大地减小时间复杂度，但是决策性能也随之大幅度下降，在实际应用中需要仔细选择 k 的值以达成效率与性能的权衡。反过来讲，表征空间近邻法的 k 值从 1 逐渐增加过程中，性能增益的边际效益会迅速减少。在推荐系统中，k 的值只需满足 $k/|\mathcal{A}| > 10\%$，就可以保持一个较少的推荐性能损耗，同时大幅度减小时间复杂度。

考虑以下简易场景，对于一个随机的虚拟动作 $\hat{a}$，假设该虚拟动作的近邻动作有概率 p 是一个不良动作，即动作价值低于 $Q(s,\hat{a}) - c$，正常近邻动作的动作价值均匀分布在区间 $[Q(s,\hat{a}) - b, Q(s,\hat{a}) + b]$ 内，其中 $b \le c$。将近邻的 k 个动作表示为整数 $\{1,\ldots,k\}$，那么 k 个近邻动作中的期望最大值为：

$$\mathbb{E}\left[\max_{i\in\{1,\ldots,k\}} Q(s,i) \mid s, \hat{a}\right] = Q(s,a) + b - \left(p^k(c-b) + \frac{2b}{k+1}\frac{1-p^{k+1}}{1-p}\right) \tag{7-6}$$

一个动作的最大动作价值是 $Q(s,\hat{a}) + b$，所以理论上近邻动作的价值一定小于等于最大动作价值，且减小的期望程度为 $p^k(c-b) + [2b/(k+1)][(1-p^{k+1})/(1-p)]$。期望程度的第一项的时间复杂度为 $\mathcal{O}(p^k)$，第二项的时间复杂度为 $\mathcal{O}(1/(k+1))$，当 k 的值较小时，这两项的值会随着 k 的增大而显著减小，也就是期望最大值会快速接近 $Q(s,\hat{a}) + b$。随着 k 值达到一个较大的水平，例如 $10\% \cdot |\mathcal{A}|$，边际收益会迅速减小，期望最大值接近 $Q(s,\hat{a}) + b$ 的速度变得缓慢。

7.1.2 矩阵重构

我们引用 FCPO 算法来一次性得到完整的推荐列表，简称为矩阵重构法。

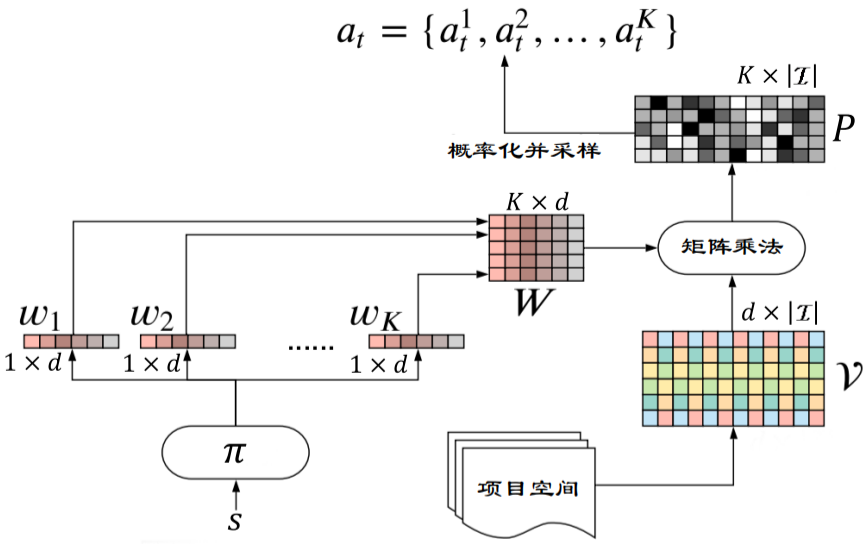

图 7-2 矩阵重构法图示

如图 7-2 所示，将动作用一个 d 维向量表示，那么整个候选动作空间表示为 $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{|\mathcal{I}| \times d}$ 。随机策略 π 有两个输出头，分别用于生成均值向量和方差向量，两个向量的长度均为 d：

$$(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \pi\left((\boldsymbol{\mu}, \boldsymbol{\Sigma}) \mid s\right) \tag{7-7}$$

在进行推荐时，策略 π 共生成 K 个均值和方差向量，然后代入正态分布对每个维度独立采样得到矩阵 $\boldsymbol{W}$ ：

$$\boldsymbol{W} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \in \mathbb{R}^{K \times d} \tag{7-8}$$

其中，均值向量 $\boldsymbol{\mu} \in \mathbb{R}^{Kd}$ ，协方差矩阵 $\boldsymbol{\Sigma} \in \mathbb{R}^{Kd \times Kd}$ ， $\boldsymbol{\Sigma}$ 由方差向量通过对角线建模得来。

然后将矩阵 $\boldsymbol{W}$ 与 $\boldsymbol{\mathcal{V}}$ 相乘，并做 softmax，得到项目的候选概率矩阵：

$$\boldsymbol{P} = \text{softmax}(\boldsymbol{W}^\top \boldsymbol{\mathcal{V}}) \tag{7-9}$$

矩阵 $\boldsymbol{P}$ 中第 k 行即推荐列表中第 k 个位置，其候选项目的概率列表为：

$$\boldsymbol{P}_k = \text{softmax}(\boldsymbol{W}^\top \boldsymbol{\mathcal{V}}), \quad k = 1, \ldots, K \tag{7-10}$$

之后，选取 $\boldsymbol{P}$ 矩阵中每一行中概率最大的索引作为推荐项，$\boldsymbol{P}$ 矩阵共有 K 行，因此推荐列表中总共有 K 个项目：

$$a_t^k = \underset{i \in \{1, \ldots, |\mathcal{I}\}}{\arg\max} P_{k,i}, \quad \forall k = 1, \ldots, K \tag{7-11}$$

其中， $P_{k,i}$ 表示在第 k 行取第 i 个项目的概率。

7.2 树型策略梯度

树形策略梯度推荐方法（Tree-structured Policy Gradient Recommendation，TPGR）通过构造基于项目的平衡分层聚类树，利用策略梯度方法学习在树的每个非叶结点选择最优子类的策略来解决大规模离散动作空间问题。

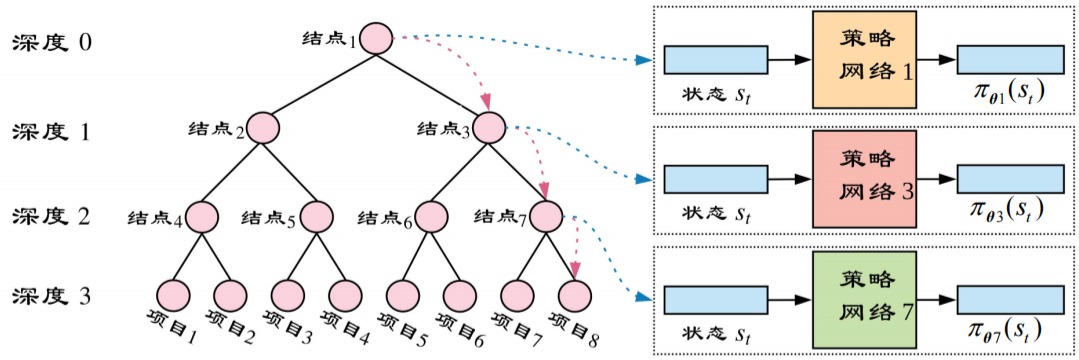

图 7-3 树形策略梯度推荐方法的架构

图 7-3 展示了树形策略梯度推荐方法的整体架构。在聚类树中，每个叶结点都映射到一个具体的项目，每个非叶结点都与一个策略网络相关联。对于一个给定的状态 s_t，最终的推荐结果会在策略网络的指导下，从根到叶结点进行自上而下的移动，并将相应的项目推荐给用户。

1）平衡分层聚类树。聚类树通过对项目进行平衡的分层聚类得到。所构建的聚类树应该是平衡的，即对于每个结点，其子树的高度最多相差 1，并且其子树也要是平衡的。这种划分重复进行，直到每个子簇只关联一个点。为了简化描述和实现，树形策略梯度推荐方法还要求每个非叶结点应当具有相同数量的子结点，记为 c，除了叶结点的父结点，其子结点的数量最多为 c。

可以使用聚类算法来实现项目的平衡分层聚类。聚类的前提在于项目要有合适的表征，可以使用基于评分的方法、基于变分自动编码器的方法或基于矩阵分解的方法：

（1）基于评分的表征方法。项目表征向量为用户-项目评分矩阵中的相应列，项目 j 向量中的每个元素 (i, j) 的值是用户 i 对项目 j 的评分。

（2）基于变分自动编码器的表征方法。利用变分自动编码器来学习每个项目的评分向量的低维表示。

（3）基于矩阵分解的表征方法。利用矩阵分解技术来处理用户-项目评分矩阵，得到每个项目的表征向量。

考虑基于主成分分析或基于 K-means 的聚类算法，以一组项目和一个整数 c 为输入，并将所有的项目划分到 c 个平衡的簇中，即每个簇的项目数量最多相差 1 个。通过反复应用聚类算法，直到每个子簇只关联一个项目，就构建了一个平衡的聚类树。

因此，设项目候选集为 $\mathcal{I}$、平衡聚类树的深度为 d，有：

$$c^{d-1} < |\mathcal{I}| \leq c^d \tag{7-12}$$

即给定项目集 $\mathcal{I}$ 和树深度 d，可以设置 $c = \text{ceil}\lfloor |\mathcal{I}|^{1/d} \rfloor$，表示向上取整，返回大于或等于 $|A|^{1/d}$ 的最小整数。

有了这颗聚类树后，我们假设有一个状态点来指示当前位于哪个结点。做出推荐决策的过程就是将状态点从根结点依次移动到某个叶结点的过程。树中的每个非叶结点是一个策略网络，使用全连接神经网络实现，并在输出层使用 Softmax 激活函数。

假定状态点移动到了结点 v，那么结点 v 上的策略网络就以当前状态作为输入，并输出在 v 的所有子结点上的概率分布，这表示从结点 v 移动到 v 的每个子结点的概率。以图 7-3 为例，该推荐场景包含了 8 个项目，构造了一颗深度为 3 的平衡聚类树。对于给定的状态 s_t，状态点最

初位于根结点（结点$_1$），然后根据根结点对应的策略（策略网络$_1$）给出的概率分布下移到其子结点 3（结点$_3$）。随后，状态点继续移动，直到到达一个具体的叶结点，然后将对应的项目（项目$_8$）推荐给用户。

2）策略训练。树中的每个策略都使用 REINFORCE 算法进行训练，目标是最大化预期的折扣累积奖励，即：

$$\mathcal{J}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{i=1}^{n}\gamma^{i-1}r_i\right] \tag{7-13}$$

其中，奖励为用户对项目标准化后的评分。例如，MovieLens 数据集中的评分是 1-5 星级的形式，那么先要把原始 1-5 星评分线性归一化到[−1,1]区间，得到用户 u 对项目 j 的归一化评分 r_{ij}，若无评分则置为 0。再计算当前交互之前连续出现的正向反馈（$r_{uj}>0$）的个数，以及连续出现的负向反馈（$r_{uj}<0$）的个数，最终定义即时奖励为：

$$r(s,a) = r_{uj} + \alpha \times (c_p - c_n) \tag{7-14}$$

其中，α 是一个非负的超参数；a 是策略采取的动作，表示当前状态 s 下为用户 u 推荐的项目。

参数 $\boldsymbol{\theta}$ 通过随机策略梯度定理来优化，其梯度表达式为：

$$\nabla_{\boldsymbol{\theta}}\mathcal{J}(\pi_{\boldsymbol{\theta}}) \approx \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a\,|\,s)Q_{\pi_{\boldsymbol{\theta}}}(s,a)\right] \tag{7-15}$$

3）时空复杂度分析。在 TPGR 中，策略网络的时间复杂度和空间复杂度由输出层单元的数量 $c = \text{ceil}\lfloor |\mathcal{I}|^{1/d} \rfloor$ 决定。根据复杂度的关系转换式，我们有：

$$\begin{aligned}\mathcal{O}(c+m) &\simeq \mathcal{O}(|\mathcal{I}|^{\frac{1}{d}}+m) \simeq \mathcal{O}(|\mathcal{I}|^{\frac{1}{d}}) \simeq \mathcal{O}(c)\\ \mathcal{O}(m\times c) &\simeq \mathcal{O}(m\times|\mathcal{I}|^{\frac{1}{d}}) \simeq \mathcal{O}(|\mathcal{I}|^{\frac{1}{d}}) \simeq \mathcal{O}(c)\end{aligned} \tag{7-16}$$

其中，m 是一个常量。

由于每个策略都是一个全连接神经网络，因此每个策略网络做出推荐决策的时间复杂度为 $\mathcal{O}(a+b\times c)$。这里的 a 表示输出层之前的时间消耗，b 表示输出层之前的隐藏层的隐藏单元数，它们都是一个常量。根据公式（7-16）可以得到：$\mathcal{O}(a+b\times c)=\mathcal{O}(c)$。

类似的，也可以分析得出每个策略网络的空间复杂度。假设每个策略网络除输出层的参数外的空间占用是 a'，输出层之前的隐藏层的隐藏单元数是 b'，则每个策略网络的空间复杂度是 $\mathcal{O}(a'+b'\times c)=\mathcal{O}(c)$。因此，每个策略网络的决策时间和空间复杂度都与其输出层单元的数量呈线性关系，即时间复杂度为 $\mathcal{O}(c)$。

TPGR 在一次推荐决策中总共会做出 d 个选择，每个选择都基于最多有 c 个输出单元的策略网络，因此 TPGR 完成一次项目推荐的时间复杂度为：

$$\mathcal{O}(d\times c) \simeq \mathcal{O}(d\times|\mathcal{I}|^{\frac{1}{d}}) \tag{7-17}$$

而一般的强化学习推荐方法完成一次推荐的时间复杂度为 $\mathcal{O}(|\mathcal{I}|)$，相比之下 TPGR 确实能在推荐阶段大幅降低时间复杂度。

对于空间复杂度，每个策略网络的空间复杂度是 $\mathcal{O}(c)$，构造的聚类树中，非叶结点（即策略网络）的数量是：

$$\begin{aligned}\mathsf{N} &= 1 + c + c^2 + \cdots + c^{d-1} \\ &= \frac{c^{d-1}}{c-1}\end{aligned} \tag{7-18}$$

根据公式（7-16）有$\mathcal{O}(\mathsf{N}\times c) \simeq \mathcal{O}(c^{d-1}/(c-1)\times c) \simeq \mathcal{O}(c^d) \simeq \mathcal{O}(|\mathcal{I}|)$，可知 TPGR 的空间复杂度为$\mathcal{O}(|\mathcal{I}|)$，这与常规的强化学习推荐方法是一致的。