

# 强化学习在电商环境下的若干应用与研究

December 25, 2017

## 背景

随着搜索技术的持续发展，我们已经逐渐意识到监督学习算法在搜索场景的局限性：

- 搜索场景中，只有被当前投放策略排到前面的商品，才会获得曝光机会，从而形成监督学习的正负样本，而曝光出来的商品，只占总的召回商品中的很小一部分，训练样本是高度受当前模型的bias影响的。
- 监督学习的损失函数，和业务关注的指标之间，存在着不一致性
- 用户的搜索、点击、购买行为，是一个连续的序列决策过程，监督模型无法对这个过程进行建模，无法优化长期累积奖赏。

与此同时，强化学习的深度学习化，以及以Atari游戏和围棋游戏为代表的应用在近几年得到了空前的发展，使得我们开始着眼于这项古老而又时尚的技术，并以此为一条重要的技术发展路线，陆陆续续地在多个业务和场景，进行了强化学习建模，取得了一些初步成果，相关的工作已经在整理发表中。同时我们也深知，目前强化学习的算法理论上限和工业界中大规模噪声数据之间，还存在着很大的gap，需要有更多的智慧去填补。

## 基于强化学习的实时搜索排序调控

### 背景

淘宝的搜索引擎涉及对上亿商品的毫秒级处理响应，而淘宝的用户不仅数量巨大，其行为特点以及对商品的偏好也具有丰富性和多样性。因此，要让搜索引擎对不同特点的用户作出针对性的排序，并以此带动搜索引导的成交提升，是一个极具挑战性的问题。传统的Learning to Rank (LTR) 方法主要是在商品维度进行学习，根据商品的点击、成交数据构造学习样本，回归出排序权重。尽管Contextual LTR方法可以根据用户的上下文信息对不同的用户给出不同的排序结果，但它没有考虑到用户搜索商品是一个连续的过程。这一连续过程的不同阶段之间不是孤立的，而是有着紧密的联系。换句话说，用户最终选择购买或不够买商品，不是由某一次排序所决定，而是一连串搜索排序的结果。

实际上，如果把搜索引擎看作智能体 (Agent)、把用户看做环境 (Environment)，则商品的搜索问题可以被视为典型的顺序决策问题 (Sequential

Decision-making Problem) : (1) 在用户每一次请求PV时, Agent做出相应的排序决策, 将商品展示给用户; (2) 用户根据Agent的排序结果, 给出点击、翻页等反馈信号; (3) Agent接收反馈信号, 在新的PV请求时做出新的排序决策; (4) 这样的过程将一直持续下去, 直到用户购买商品或者退出搜索。以前向视角 (Forward View) 来看, 用户在每个PV中的上下文状态与之前所有PV中的上下文状态和Agent的行为有着必然因果关系, 同一个PV中Agent采取的不同排序策略将使得搜索过程朝不同的方向演进; 反过来, 以后向视角 (Backward View) 来看, 在遇到相同的上下文状态时, Agent就可以根据历史演进的结果对排序策略进行调整, 将用户引导到更有利成交的PV中去。Agent每一次策略的选择可以看成一次试错 (Trial-and-Error), 在这种反复不断地试错过程中, Agent将逐步学习到最优的排序策略。而这种在与环境交互的过程中进行试错的学习, 正是强化学习 (Reinforcement Learning, RL) 的根本思想。

强化学习最早可以追溯到巴甫洛夫的条件反射实验, 它从动物行为研究和优化控制两个领域独立发展, 最终经Bellman之手将其抽象为马尔可夫决策过程 (Markov Decision Process, MDP) 问题而完成形式化。对于环境反馈的有利奖赏, Agent将强化引发这种奖赏的动作, 并在以后与环境交互的过程中更偏向于执行该动作。我们尝试将强化学习方法引入商品的搜索排序中, 以优化用户在整个搜索过程中的收益为目标, 根据用户实时行为反馈进行学习, 实现商品排序的实时调控。图1比较直观地展示了用强化学习来优化搜索排序的过程。如图所示, 在三次PV请求之间, Agent做出了两次排序决策 ( $a_1$  和  $a_2$ ), 从而引导了两次PV展示。从效果上来看,  $a_1$  对应PV中并没有发生商品点击, 而  $a_2$  对应PV上发生了3次商品点击。如果将商品点击看成是对排序策略的反馈信号, 那么Agent第二次执行的排序策略  $a_2$  将得到正向的强化激励, 而其第一次排序策略  $a_1$  得到的激励为零。本文接下来的内容将对我们具体的方案进行详细介绍。

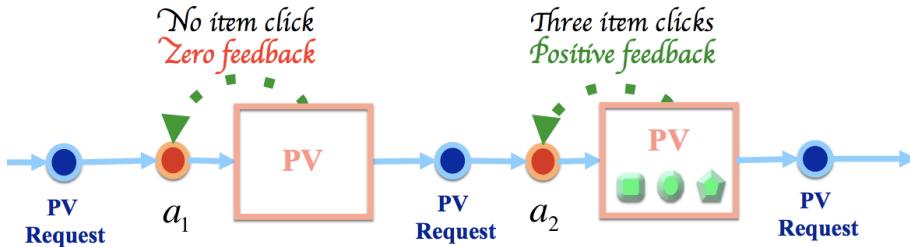


Figure 1: 搜索的序列决策模型

## 问题建模

### 强化学习简介

马尔可夫决策过程 (Markov Decision Process, MDP) 是强化学习的最基本理论模型。一般地, MDP可以由一个四元组 $\langle S, A, R, T \rangle$ 表示: (1)  $S$ 为状态空间 (State Space), 包含了Agent可能感知到的所有环境状态; (2)  $A$ 为动作空间 (Action Space), 包含了Agent在每个状态上可以采取的所有动作; (3)  $R : S \times A \times S \rightarrow \mathbb{R}$ 为奖赏函数 (Reward Function),  $R(s, a, s')$ 表示在

状态 $s$ 上执行动作 $a$ , 并转移到状态 $s'$ 时, Agent从环境获得的奖赏值; (4)  $T : S \times A \times S \rightarrow [0, 1]$ 为环境的状态转移函数(State Transition Function),  $T(s, a, s')$ 表示在状态 $s$ 上执行动作 $a$ , 并转移到状态 $s'$ 的概率。

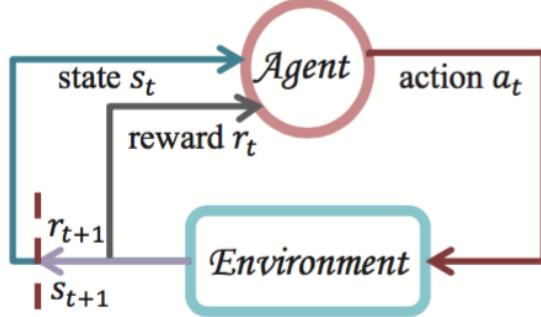


Figure 2: 强化学习agent和环境交互

在MDP中, Agent和环境之间的交互过程可如图2所示: Agent感知当前环境状态 $s_t$ , 从动作空间 $A$ 中选择动作 $a_t$ 执行; 环境接收Agent所选择的动作之后, 给以Agent相应的奖赏信号反馈 $r_{t+1}$ , 并转移到新的环境状态 $s_{t+1}$ , 等待Agent做出新的决策。在与环境的交互过程中, Agent的目标是找到一个最优策略 $\pi^*$ , 使得它在任意状态 $s$ 和任意时间步骤 $t$ 下, 都能够获得最大的长期累积奖赏, 即

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\}, \forall s \in S, \forall t \geq 0. \quad (1)$$

在这里,  $\pi : S \times A \rightarrow [0, 1]$ 表示Agent的某个策略 (即状态到动作的概率分布),  $\mathbb{E}_{\pi}$ 表示策略 $\pi$ 下的期望值,  $\gamma \in [0, 1)$ 为折扣率 (Discount Rate),  $k$ 为未来时间步骤,  $r_{t+k}$ 表示Agent在时间步骤 $(t+k)$ 上获得的即时奖赏。

强化学习主要通过寻找最优状态值函数 (Optimal State Value Function)

$$V^*(s) = \max_{\pi} \mathbb{E}_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\}, \forall s \in S, \forall t \geq 0 \quad (2)$$

或最优状态动作值函数 (Optimal State-Action Value Function)

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a \right\}, \forall s \in S, \forall a \in A, \forall t \geq 0 \quad (3)$$

来学习最优策略 $\pi^*$ 。经典的强化学习算法 (如: SARSA, Q-learning等) 将值函数用状态空间到动作空间的一张表来进行表达, 并通过广义策略迭代 (Generalized Policy Iteration) 方法对最值函数进行逼近, 从而得到最优策略 $\pi^*$ 。然而, 在大规模状态/动作空间问题 (包括连续状态/动作空间问题) 中, 值表形式的值函数所需要的存储空间远远超过了现代计算机的硬件条件, 使得这些经典的算法不再适用。这也即强化学习中的著名的“维度灾难”问题。

值函数估计 (Value Function Approximation) 是解决维度灾难问题的主要手段之一，其主要思想是将状态值函数或动作值函数进行参数化，将值函数空间转化为参数空间，达到泛化(Generalization)的目的。以状态值函数为例，在参数向量 $\theta$ 下，任意状态 $s$ 的值 $V(s)$ 可以表达为

$$V_\theta(s) = f_\theta(\phi(s)). \quad (4)$$

其中， $\phi(s)$ 为状态 $s$ 的特征向量， $f$ 为定义在状态特征空间上的某个函数，其具体形式取决于算法本身。因此，对值函数 $V(s)$ 的学习也就转化为了对参数向量 $\theta$ 的学习。基于对函数 $f$ 采用的不同形式，强化学习领域也发展出了不同的子分支，这其中包括：线性函数估计方法，回归树方法，神经网络方法，基于核的强化学习方法等。值得一提的是，深度强化学习 (Deep Reinforcement Learning, DRL) 本质上属于采用神经网络作为值函数估计器的一类方法，其主要优势在于它能够利用深度神经网络对状态特征进行自动抽取，避免了人工定义状态特征带来的不准确性，使得Agent能够在更原始的状态上进行学习。

### 状态定义

在我们的方案中，用户被视为响应Agent动作的环境，Agent需要感知环境状态进行决策。因此，如何定义环境状态使其能够准确反映出用户对商品的偏好是首要问题。假设用户在搜索的过程中倾向于点击他感兴趣的的商品，并且较少点击他不感兴趣的的商品。基于这个假设，我们将用户的历史点击行为作为抽取状态特征的数据来源。具体地，在每一个PV请求发生时，我们把用户在最近一段时间内点击的商品的特征（包括：价格、转化率、销量等）作为当前Agent感知到的状态，令 $s$ 代表状态，则有

$$s = (price_1, cvr_1, sale_1, \dots, price_n, cvr_n, sale_n). \quad (5)$$

其中， $n$ 表示历史点击商品的个数，为可变参数， $price_i$ 、 $cvr_i$ 、 $sale_i$ 分别代表商品 $i$  ( $0 \leq i \leq n$ ) 的价格、转化率和销量。另外，为了区别不同群体的用户，我们还将用户的长期特征加入到了状态的定义中，最终的状态定义为

$$s = (price_1, cvr_1, sale_1, \dots, price_n, cvr_n, sale_n, power, item, shop). \quad (6)$$

其中， $power$ 、 $item$ 和 $shop$ 分别代表用户的购买力、偏好宝贝以及偏好店铺特征。在具体算法实现时，由于状态特征不同维度的尺度不一样，我们会将所有维度的特征值归一化到 $[0, 1]$ 区间内，再进行后续处理。

### 奖赏函数设定

当状态空间 $S$ 和动作空间 $A$ 确定好之后（动作空间即Agent能够选择排序策略的空间），状态转移函数 $T$ 也随即确定，但奖赏函数 $R$ 仍然是个未知数。奖赏函数 $R$ 定义的是状态与动作之间的数值关系，而我们要解决的问题并非是一个天然存在的MDP，这样的数值关系并不存在。因此，另一个重要的步骤是把我们要达到的目标（如：提高点击率、提高GMV等）转化为具体的奖赏函数 $R$ ，在学习过程中引导Agent完成我们的目标。

幸运的是，这样的转化在我们的场景中并不复杂。如前所述，Agent给出商品排序，用户根据排序的结果进行的浏览、商品点击或购买等行为都可以看成

对Agent的排序策略的直接反馈。我们采取的奖赏函数定义规则如下：(1) 在一个PV中如果仅发生商品点击，则相应的奖赏值为用户点击的商品的数量；(2) 在一个PV中如果发生商品购买，则相应奖赏值为被购买商品的价格；(3) 其他情况下，奖赏值为0。从直观上来理解，第一条规则表达的是提高CTR这一目标，而第二条规则表达的则是提高GMV。在第四章中，我们将利用奖赏塑形 (Reward Shaping) 方法对奖赏函数的表达进行丰富，提高不同排序策略在反馈信号上的区分度。

## 算法设计

### 策略函数

在搜索场景中，排序策略实际上是一组权重向量，我们用 $\mu = (\mu_1, \mu_2, \dots, \mu_m)$ 来表示。每个商品最终的排序次序是由其特征分数和排序权重向量 $\mu$ 的内积所决定的。一个排序权重向量是Agent的一个动作，那么排序权重向量的欧式空间就是Agent的动作空间。根据2.1节对状态的定义可知，我们的状态空间也是连续的数值空间。因此，我们面临的问题是在两个连续的数值空间中学习出最优的映射关系。

策略逼近 (Policy Approximation) 方法是解决连续状态/动作空间问题的有效方法之一。其主要思想和值函数估计方法类似，即用参数化的函数对策略进行表达，通过优化参数来完成策略的学习。通常，这种参数化的策略函数被称为Actor。我们采用确定性策略梯度算法 (Deterministic Policy Gradient, DPG) 算法来进行排序的实时调控优化。在该算法中，Actor的输出是一个确定性的策略 (即某个动作)，而非一个随机策略 (即动作的概率分布)。对于连续动作空间问题，确定性策略函数反而让策略改进 (Policy Improvement) 变得更加方便了，因为贪心求最优动作可以直接由函数输出。

我们采用的Actor以状态的特征为输入，以最终生效的排序权重为输出。假设我们一共调控 $m$  ( $m \geq 0$ ) 个维度的排序权重，对于任意状态 $s \in S$ ，Actor对应的输出为

$$\mu_\theta(s) = (\mu_\theta^1(s), \mu_\theta^2(s), \dots, \mu_\theta^m(s)). \quad (7)$$

其中， $\theta = (\theta_1, \theta_2, \dots, \theta_m)$  为actor的参数向量，对于任意 $i$  ( $1 \leq i \leq m$ )， $\mu_\theta^i(s)$  为第 $i$ 维的排序权重分，具体地有

$$\mu_\theta^i = \frac{C_i \exp(\theta_i^\top \phi(s))}{\sum_{j=1}^m \exp(\theta_j^\top \phi(s))}. \quad (8)$$

在这里， $\phi(s)$  为状态 $s$ 的特征向量， $\theta_1, \theta_2, \dots, \theta_m$  均为长度与 $\phi(s)$ 相等的向量， $C_i$  为第 $i$ 维排序权重分的常数，用来对其量级进行控制（不同维度的排序权重分会有不同的量级）。

### 策略梯度

回顾一下，强化学习的目标是最大化任意状态 $s$ 上的长期累积奖赏（参考2.1节中对 $V^*$ 和 $Q^*$ 的定义）。实际上，我们可以用一个更一般地形式来表达这一目

标，即

$$\begin{aligned}
J(\mu_\theta) &= \int_S \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_0(s') T(s', \mu_\theta(s'), s) R(s, \mu_\theta(s)) ds' ds \\
&= \int_S \rho^\mu(s) R(s, \mu_\theta(s)) ds \\
&= \mathbb{E}_{s \sim \rho^\mu}[R(s, \mu_\theta(s))].
\end{aligned} \tag{9}$$

其中， $\rho^\mu(s) = \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_0(s') T(s', \mu_\theta(s'), s) ds'$  表示状态  $s$  在一直持续的学习过程中被访问的概率， $p_0$  为初始时刻的状态分布， $T$  为环境的状态转移函数。不难推测， $J(\mu_\theta)$  实际上表达的是在确定性策略  $\mu_\theta$  的作用下，Agent 在所有状态上所能够获得的长期累积奖赏期望之和。通俗地讲，也就是 Agent 在学习过程中得到的所有奖赏值。

显然，为了最大化  $J(\mu_\theta)$ ，我们需要求得  $J(\mu_\theta)$  关于参数  $\theta$  的梯度，让  $\theta$  往梯度方向进行更新。根据策略梯度定理 (Policy Gradient Theorem)， $J(\mu_\theta)$  关于  $\theta$  的梯度为

$$\begin{aligned}
\nabla_\theta J(\mu_\theta) &= \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a) |a = \mu_\theta(s)| ds \\
&= \mathbb{E}_{s \sim \rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a) |a = \mu_\theta(s)].
\end{aligned} \tag{10}$$

其中， $Q^\mu(s, a)$  为策略  $\mu_\theta$  下状态动作对 (State-Action Pair)  $(s, a)$  对应的长期累积奖赏。因此，参数  $\theta$  的更新公式可以写为

$$\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a) |a = \mu_\theta(s). \tag{11}$$

在这个公式中， $\alpha_\theta$  为学习率， $\nabla_\theta \mu_\theta(s)$  为一个 Jacobian Matrix，能够很容易地算出来，但麻烦的是  $Q^\mu(s, a)$  及其梯度  $\nabla_a Q^\mu(s, a)$  的计算。因为  $s$  和  $a$  都是连续的数值，我们无法精确获取  $Q^\mu(s, a)$  的值，只能通过值函数估计方法进行近似计算。我们采用线性函数估计方法 (Linear Function Approximation, LFA)，将  $Q$  函数用参数向量  $w$  进行表达：

$$Q^\mu(s, a) \approx Q^w(s, a) = \phi(s, a)^\top w. \tag{12}$$

在这里， $\phi(s, a)$  为状态动作对  $(s, a)$  的特征向量。采用线性值函数估计的好处不仅在于它的计算量小，更重要的是在它能让我们找到合适的  $\phi(s, a)$  的表达，使得  $\nabla_a Q^w(s, a)$  可以作为  $\nabla_a Q^\mu(s, a)$  的无偏估计。一个合适的选择是令  $\phi(s, a) = a^\top \nabla_\theta \mu_\theta(s)$ ，则可以得到

$$\nabla_a Q^\mu(s, a) \approx \nabla_a Q^w(s, a) = \nabla_a(a^\top \nabla_\theta \mu_\theta(s))^\top w = \nabla_\theta \mu_\theta(s)^\top w. \tag{13}$$

因此，策略函数的参数向量  $\theta$  的更新公式可以写为

$$\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s) (\nabla_\theta \mu_\theta(s)^\top w). \tag{14}$$

## 值函数的学习

在更新策略函数 $\mu_\theta$ 的参数向量 $\theta$ 的同时，值函数 $Q^w$ 的参数向量 $w$ 也需要进行更新。最简单地， $w$ 的更新可以参照Q-learning算法[4, 5]的线性函数估计版本进行，对于样本 $(s_t, a_t, r_t, s_{t+1})$ ，有：

$$\begin{aligned}\delta_{t+1} &= r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \\ &= r_t + w_t^\top (\gamma \phi(s_{t+1}, \mu_\theta(s_{t+1})) - \phi(s_t, a_t)) \\ w_{t+1} &= w_t + \alpha_w \delta_{t+1} \phi(s_t, a_t) \\ &= w_t + \alpha_w \delta_{t+1} (a_t^\top \nabla_\theta \mu_\theta(s_t)).\end{aligned}\tag{15}$$

其中， $s_t$ 、 $a_t$ 、 $r_t$ 和 $s_{t+1}$ 为Agent在 $t$ 时刻感知的状态、所做的动作、从环境获得的奖赏反馈和在 $(t+1)$ 时刻感知的状态， $\delta_{t+1}$ 被称作差分误差（Temporal-Difference Error）， $\alpha_w$ 为 $w$ 的学习率。

需注意的是，Q-learning的线性函数估计版本并不能保证一定收敛。并且，在大规模动作空间问题中，线性形式的 $Q$ 函数较难在整个值函数空间范围内精确地估计每一个状态动作对的值。一个优化的办法是引入优势函数（Advantage Function），将 $Q$ 函数用状态值函数 $V(s)$ 和优势函数 $A(s, a)$ 的和进行表达。我们用 $V(s)$ 从全局角度估计状态 $s$ 的值，用 $A(s, a)$ 从局部角度估计动作 $a$ 在状态 $s$ 中的相对于其他动作的优势。具体地，我们有

$$Q(s, a) = A^w(s, a) + V^v(s) = (a - \mu_\theta(s))^\top \nabla_\theta \mu_\theta(s)^\top w + \phi(s)^\top v.\tag{16}$$

在这里， $w$ 和 $v$ 分别为 $A$ 和 $V$ 的参数向量。最后，我们将所有参数 $\theta$ 、 $w$ 和 $v$ 的更新方式总结如下：

$$\begin{aligned}\delta_{t+1} &= r_t + \gamma Q(s_{t+1}, \mu_\theta(s_{t+1})) - Q(s_t, a_t) \\ &= r_t + \gamma \phi(s_{t+1})^\top v_t - ((a_t - \mu_\theta(s_t))^\top \nabla_\theta \mu_\theta(s_t)^\top w_t + \phi(s_t)^\top v_t) \\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) (\nabla_\theta \mu_\theta(s_t)^\top w_t) \\ w_{t+1} &= w_t + \alpha_w \delta_{t+1} \phi(s_t, a_t) = w_t + \alpha_w \delta_{t+1} (a_t^\top \nabla_\theta \mu_\theta(s_t)) \\ v_{t+1} &= v_t + \alpha_v \delta_{t+1} \phi(s_t)\end{aligned}\tag{17}$$

## 奖赏塑形

第二章定义的奖赏函数是一个非常简单化的版本，我们在初步的实验中构造了一个连续状态空间/离散动作空间问题对其进行验证。具体地，我们采用了2.2节定义的状态表示方法，同时人工选取15组固定的排序策略，通过线性值函数估计控制算法GreedyGQ来学习相应的状态动作值函数 $Q(s, a)$ 。从实验结果中，我们发现学习算法虽然最终能够稳定地区分开不同的动作，但它们之间的差别并不大。一方面，这并不会对算法收敛到最优产生影响；但另一个方面，学习算法收敛的快慢却会大受影响，而这是在实际应用中我们必须考虑的问题。

在淘宝主搜这种大规模应用的场景中，我们较难在短时间内观察到不同的排序策略在点击和成交这样的宏观指标上的差别。因此，我们有必要在奖赏函数中引入更多的信息，增大不同动作的区分度。以商品点击为例，考虑不

同的用户A和B在类似的状态中发生的商品点击行为。若A点击商品的价格较高, B点击商品的价格较低, 那么就算A和B点击的商品数量相同, 我们也可以认为对A和B采用的排序策略带来的影响是不同的。同样的, 对于商品的成交, 价格相同而销量不同的商品成交也具有差别。因此, 在原有的基础上, 我们将商品的一些属性特征加入到奖赏函数的定义中, 通过奖赏塑形 (Reward Shaping) 的方法丰富其包含的信息量。

奖赏塑形的思想是在原有的奖赏函数中引入一些先验的知识, 加速强化学习算法的收敛。简单地, 我们可以将“在状态 $s$ 上选择动作 $a$ , 并转移到状态 $s'$ ”的奖赏值定义为

$$R(s, a, s') = R_0(s, a, s') + \Phi(s). \quad (18)$$

其中,  $R_0(s, a, s')$ 为原始定义的奖赏函数,  $\Phi(s)$ 为包含先验知识的函数, 也被称为势函数 (Potential Function)。我们可以把势函数 $\Phi(s)$ 理解学习过程中的子目标 (Local Objective)。例如, 在用强化学习求解迷宫问题中, 可以定义 $\Phi(s)$ 为状态 $s$ 所在位置与出口的曼哈顿距离 (或其他距离), 使得Agent更快地找到潜在的与出口更近的状态。根据上面的讨论, 我们把每个状态对应PV的商品信息纳入Reward的定义中, 将势函数 $\Phi(s)$ 定义为

$$\Phi(s) = \sum_{i=1}^K \text{MLL}(i|\mu_\theta(s)). \quad (19)$$

其中,  $K$ 为状态 $s$ 对应PV中商品的个数,  $i$ 表示的第 $i$ 个商品,  $\mu_\theta(s)$ 为Agent在状态 $s$ 执行的动作,  $\text{MLL}(i|\mu_\theta(s))$ 表示排序策略为 $\mu_\theta(s)$ 时对商品的点击 (或成交) 的极大似然 (Maximum Likelihood) 估计。因此,  $\Phi(s)$ 也就表示在状态 $s$ 上执行动作 $\mu_\theta(s)$ 时, PV中所有商品能够被点击 (或购买) 的极大似然概率之和。

下面我们给出的 $\text{MLL}(i|\mu_\theta(s))$ 具体形式。令商品 $i$ 的特征向量 (即价格、销量、人气分、实时分等特征) 为 $x_i = (x_i^1, x_i^2, \dots, x_i^m)$ , 则 $x_i^\top \mu_\theta(s)$ 即为商品 $i$ 在状态 $s$ 下的最终排序分数。又令 $y_i \in \{0, 1\}$ 为商品 $i$ 实际被点击 (或成交) 的label, 并假设意商品 $i$ 的实际点击 (或成交) 的概率 $p_i$ 与其排序分数 $x_i^\top \mu_\theta(s)$ 满足 $\ln \frac{p_i}{1-p_i} = x_i^\top \mu_\theta(s)$ , 则商品 $i$ 的似然概率为

$$\text{MLL} = p_i^{y_i} (1 - p_i)^{1-y_i} = \left( \frac{1}{1 + \exp(-x_i^\top \mu_\theta(s))} \right)^{y_i} \left( \frac{1}{1 + \exp(x_i^\top \mu_\theta(s))} \right)^{1-y_i}. \quad (20)$$

为简化计算, 我们对 $\text{MLL}$ 取对数, 得到对数似然概率

$$\text{MLL}_{log}(i|\mu_\theta(s)) = y_i x_i^\top \mu_\theta(s) - \ln(1 + \exp(x_i^\top \mu_\theta(s))). \quad (21)$$

将PV中所有商品的对数似然概率综合起来, 则有

$$\Phi(s) = \sum_{i=1}^K y_i x_i^\top \mu_\theta(s) - \ln(1 + \exp(x_i^\top \mu_\theta(s))). \quad (22)$$

我们最终实现的奖赏塑形方法将点击和成交均纳入考虑中, 对于只有点击的PV样本, 其对应的奖赏势函数为

$$\Phi_{clk}(s) = \sum_{i=1}^K y_i^c x_i^\top \mu_\theta(s) - \ln(1 + \exp(x_i^\top \mu_\theta(s))). \quad (23)$$

其中,  $y_i^c$ 是商品*i*被点击与否的label。而对于有成交发生的PV样本, 我们将商品价格因素加入到奖赏势函数中, 得到

$$\Phi_{pay}(s) = \sum_{i=1}^K y_i^p x_i^\top \mu_\theta(s) - \ln(1 + \exp(x_i^\top \mu_\theta(s))) + \ln \text{Price}_i. \quad (24)$$

其中,  $y_i^p$ 和  $\text{Price}_i$ 分别是商品*i*被购买与否的label和它的价格。从直观上来理解,  $\Phi_{clk}(s)$ 和 $\Phi_{pay}(s)$ 将分别引导Agent对点击率和GMV的对数似然进行优化。

实际上, 我们所采用的奖赏塑形方法来自于LTR方法的启发。LTR方法的有效性在于它能够利用商品维度的信息来进行学习, 其最终学习到的排序权重和商品特征有直接相关性。我们通过把商品的特征灌注到奖赏函数中, 能让Agent的动作在具体商品上产生的影响得到刻画, 因此也就能更好地在数值信号上将不同的动作区分开来。另外, 与以往的奖赏塑形方法不同的是, 我们采用的势函数是随着策略的学习变化的, 它让Reward和Action之间产生了相互作用: Action的计算将朝着最大化Reward的方向进行, 而Action的生效投放也反过来影响了Reward的产生。因此, 学习算法实际上是在非独立同分布的数据上进行训练的, 我们将在最后一章对该问题进行探讨。

## 实验效果

在双十一期间, 我们对强化学习方案进行了测试, 下图展示了我们的算法在学习的过程中的误差变化情况, 衡量学习误差的指标为Norm of the Expected TD Update (NEU), 是差分误差 (TD Error) 与状态动作特征向量乘积的期望值, 图中的RNEU表示NEU的平方根。从理论上来讲, RNEU越小表示算法学习到的策略越接近最优。

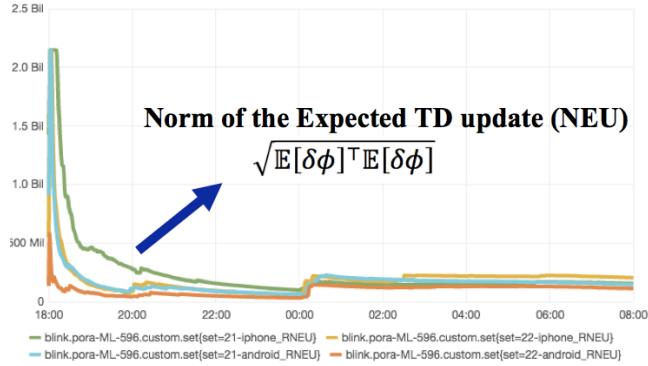


Figure 3: Norm of the Expected TD Update

可以看到, 从启动开始, 每个桶上的RNEU开始逐渐下降。之后, 下降趋势变得比较缓和, 说明学习算法在逐步往最优策略进行逼近。但过了零点之后, 每个桶对应的RNEU指标都出现了陡然上升的情况, 这是因为0点前后用户的行为发生了急剧变化, 导致线上数据分布在0点以后与0点之前产生较大差别。相应地, 学习算法获取到新的reward信号之后, 也会做出适应性地调整。

接下来，我们再对双十一当天排序权重分的变化情况进行考查。我们一共选取了若干个精排权重分来进行实时调控，下面两幅图分别展示了iphone和android中，每个维度的排序权重分在一天内的变化。

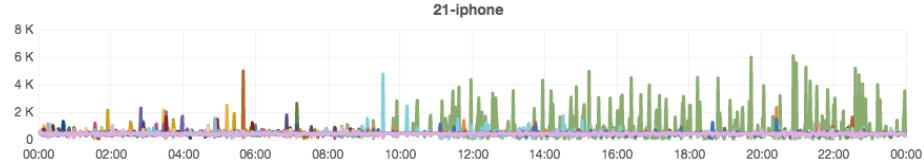


Figure 4: 每个维度的排序权重分在一天内的变化 (iphone)

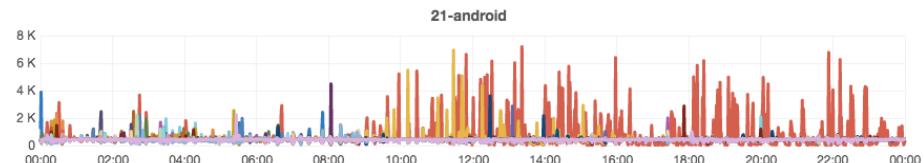


Figure 5: 每个维度的排序权重分在一天内的变化 (android)

从0点到早上10:00这一时间段内，无论是在android端还是iphone端，都没有出现某个维度的排序权重分占绝对主导地位。在11号凌晨和上午，全网大部分的成交其实不在搜索端，在这段时间内，用户产生的数据相对没有这么丰富，可能还不足以将重要的排序权重分凸显出来。而到了10:00以后，我们可以发现某一个维度的排序权重分逐渐开始占据主导，并且其主导过程一直持续到了当天结束。在iphone端占据主导的是某大促相关的分（绿色曲线），而android端的则是某转化率的分（红色曲线）。这其实也从侧面说明了iphone端和android端的用户行为存在较大差别。

在最终的投放效果上，强化学习桶相对于基准桶整体提升了很大幅度，同时强化学习桶在CTR方面的提升高于其它绝大部分非强化学习桶，证明我们所采用的奖赏塑形方法确实有效地将优化CTR的目标融入了奖赏函数中。

## DDPG与梯度融合

在双11之后，我们对之前的方案进行了技术升级，一个直接的优化是DPG升级为DDPG，即将actor模型和critic模型升级为actor网络 $\mu(s|\theta^\mu)$ 和critic网络 $Q(s, a|\theta^Q)$ 。此外，我们还增加了一个ltr loss层 $L(a, X, Y)$ ，用于衡量actor网络输出的 $a$ ，在pointwise ltr上的cross entropy loss，这里 $X = [x_1, x_2, \dots, x_n]$ 是 $n$ 个宝贝的归一化的特征分向量， $Y = [y_1, y_2, \dots, y_n]$ 是对应的点击、成交的label。具体地：

$$L(a, X, Y) = \frac{1}{n} \sum_i^n y_i \log(\sigma(a^T x_i)) + (1 - y_i) \log(1 - \sigma(a^T x_i)) \quad (25)$$

这里  $\sigma(a^T x_i) = 1/(1 + \exp(-a^T x_i))$ 。因此最终的actor的网络的梯度为

$$\begin{aligned} \nabla_{\theta^\mu} J = & -\frac{1}{N} \sum_i^N \nabla_a Q(s, a | \theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s | \theta^\mu)|_{s_i} + \\ & \lambda \nabla_a L(a, X, Y)|_{a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s | \theta^\mu)|_{s_i} \end{aligned} \quad (26)$$

大致的整体框架图如下所示：

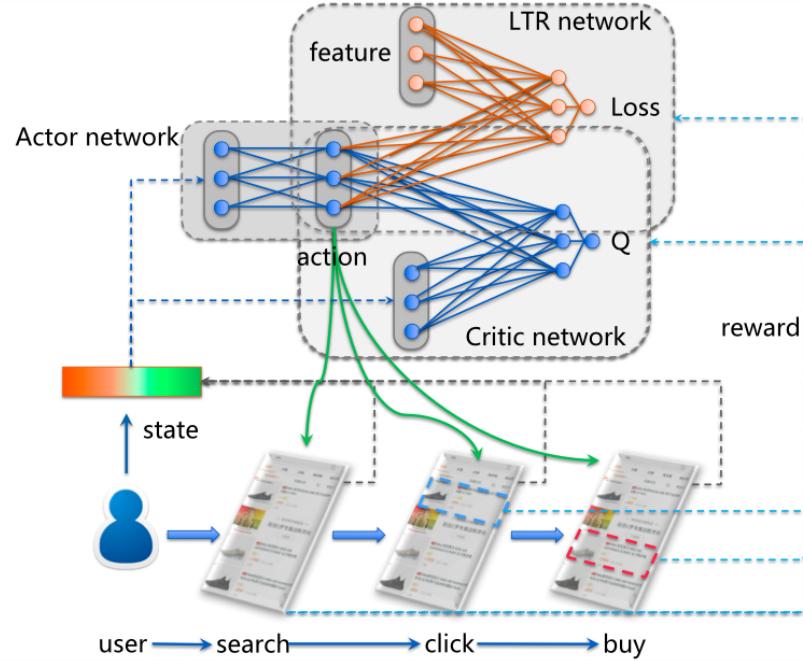


Figure 6: 监督学习和强化学习的多任务学习网络

这个整体实现，较之前的DPG方案，一方面可以受益于深度神经网络强大的表征能力，另一方面也可以从监督学习网络获得很好的梯度，获得较好的初始化，并保证整个训练过程中的稳定性。

## 总结与展望

总的来说，我们将强化学习应用到淘宝的搜索场景中只是一次初步尝试，有很多方面都需要进一步探索，现将我们在未来需要改进的地方以及可能的探索方向归纳如下：

(1) 状态的表示：我们将用户最近点击的商品特征和用户长期行为特征作为状态，其实是基于这样的一个假设，即用户点击过的商品能够较为精确地反映用户的内心活动和对商品的偏好。但实际上，用户对商品的点击通常具有盲目性，无论什么商品可能都想要看一看。也就是说，我们凭借经验所设定的状态并非那么准确。深度强化学习对状态特征的自动抽取能力是它在Atari Game

和围棋上取得成功的重要原因之一。因此，在短期内可以考虑利用深度强化学习对现有方案进行扩展。同时，借助深度神经网络对状态特征的自动抽取，我们也可以发现用户的哪些行为对于搜索引擎的决策是比较重要的。

(2) 奖赏函数的设定：和状态的定义一样，我们在第二章设定的奖赏函数也来自于人工经验。奖赏塑形 (Reward Shaping) 虽然是优化奖赏函数的方法，但其本质上也是启发式函数，其更多的作用在于对学习算法的加速。逆强化学习 (Inverse Reinforcement Learning, IRL) 是避免人工设定的奖赏函数的有效途径之一，也是强化学习研究领域的重要分支。IRL的主要思想是根据已知的专家策略或行为轨迹，通过监督学习的方法逆推出问题模型的奖赏函数。Agent在这样的奖赏函数上进行学习，就能还原出专家策略。对于我们的问题，IRL的现有方法不能完全适用，因为我们的搜索任务并不存在一个可供模仿的专家策略。我们需要更深入思考如何在奖赏函数与我们的目标（提升CTR，提升成交笔数）之间建立紧密的关系。

(3) 多智能体强化学习 (Multi-Agent Reinforcement Learning, MARL)：我们将搜索引擎看作Agent，把用户看成响应Agent动作的环境，属于典型的单智能体强化学习 (Single-Agent RL) 模式。在单智能体强化学习的理论模型（即MDP）中，环境动态 (Environmental Dynamics, 也即奖赏函数和状态转移函数) 是不会发生变化的；而在我们的问题中，用户的响应行为却是非静态的，同时也带有随机性。因此，单智能体强化学习的模式未必是我们的最佳方案。要知道，用户其实也是在一定程度理性控制下的，能够进行自主决策甚至具有学习能力的Agent。从这样的视角来看，或许更好的方式是将用户建模为另外一个Agent，对这个Agent的行为进行显式地刻画，并通过多智能体强化学习[21]方法来达到搜索引擎Agent和用户Agent之间的协同 (Coordination)。

(4) 第四章的末尾提到了奖赏函数与Agent的动作的相互作用带来的非独立同分布数据问题，我们在这里再进行一些讨论。在MDP模型中，奖赏函数 $R(s, a, s')$ （有时又写成 $R(s, a)$ ）是固定的，不会随着Agent策略的变化而变化。然而，在我们提出的奖赏塑形方法中，势函数 $\Phi_{clk}(s)$ 和 $\Phi_{pay}(s)$ 中包含了策略参数 $\theta$ ，使得Agent从环境获得的奖赏信号在不同的 $\theta$ 下有所不同。这也意味着我们的Agent实际上是处于一个具有动态奖赏函数的环境中，这种动态变化不是来自于外部环境，而是源于Agent的策略改变。这有点类似于人的行为与世界的相互作用。这样一来，我们可以将3.2节的 $J(\mu_\theta)$ 重写为

$$\begin{aligned}\bar{J}(\mu_\theta) &= \int_S \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_0(s') T(s', \mu_\theta(s'), s) R_\theta(s, \mu_\theta(s)) ds' ds \\ &= \int_S \rho^\mu(s) R_\theta(s, \mu_\theta) ds.\end{aligned}\tag{27}$$

其中， $R_\theta$ 为Agent的策略参数 $\theta$ 的函数。虽然 $J(\mu_\theta)$ 与 $\bar{J}(\mu_\theta)$ 之间只有一个符号之差，但这微小的变化也许会导致现有的强化学习算法无法适用，我们在未来的工作中将从理论上来深入研究这个问题的解决方法。

# 强化学习为何有用？——延迟奖赏在搜索排序场景中的作用分析

## 背景

我们用强化学习（Reinforcement Learning, RL）在搜索场景中进行了许多的尝试，例如：对商品排序策略进行动态调节、控制个性化展示比例、控制价格T变换等。虽然从顺序决策的角度来讲，强化学习在这些场景中的应用是合理的，但我们并没有回答一些根本性的问题，比如：在搜索场景中采用强化学习和采用多臂老虎机有什么本质区别？从整体上优化累积收益和分别独立优化每个决策步骤的即时收益有什么差别？每当有同行问到这些问题时，我们总是无法给出让人信服的回答。因为我们还没思考清楚一个重要的问题，即：在搜索场景的顺序决策过程中，任意决策点的决策与后续所能得到的结果之间的关联性有多大？从强化学习的角度讲，也就是后续结果要以多大的比例进行回传，以视为对先前决策的延迟激励。也就是说我们要搞清楚延迟反馈在搜索场景中的作用。本文将以继续以搜索场景下调节商品排序策略为例，对这个问题展开探讨。本文余下部分的将组织如下：第二节对搜索排序问题的建模进行回顾，第三节将介绍最近的线上数据分析结果，第四节将对搜索排序问题进行形式化定义，第五节和第六节分别进行理论分析和实验分析并得出结论。

## 搜索排序问题回顾

在淘宝中，对商品进行搜索排序以及重排序涉及搜索引擎与用户之间的不断交互。图7展示了这样的交互过程：（1）用户进入搜索引擎，输入query；（2）搜索引擎根据用户输入的query和用户的特征，从若干个可能的排序动作中 $(a_1, a_2, a_3, \dots)$ 选择其中一个给对应query下的商品进行排序，选择top  $K$ 个商品（ $K$ 一般等于10）；（3）用户在看到展示页面的商品之后，会在页面中进行一些操作，比如：点击、加购感兴趣的商品；（4）当用户进行翻页时，搜索引擎会再次选择一个排序动作，对未展示的商品重新进行排序，并进行商品展示；（5）随着用户不断地翻页，这样的交互过程会一直进行下去，直到用户购买某个商品或者离开搜索引擎。

如果把搜索引擎看作智能体（Agent）、把用户看做环境（Environment），那么图7展示的交互过程对于搜索引擎Agent来讲是一个典型的顺序决策问题。若从强化学习的视角来看，上所展示的过程就是一次Episode，可以重新用图8进行描述。在图8中，蓝色的节点表示一次PV请求，也对应Agent进行状态感知的时刻，红色的节点表示Agent的动作，绿色箭头表示对Agent动作的即时奖赏激励。

需要注意的是，由于搜索引擎每一次的决策都是在PV请求时发生的，所以决策过程中的状态与展示的商品页是一一对应的。更严格地来讲，每一个决策点的状态应该是“这个决策点之前所有商品页面包含的信息总和”，包括这些页面展示的商品信息，以及用户在这些页面上的实时行为。在目前的系统实现中，由于性能、信息获取条件的限制，现有的状态表示中并没有完全囊括这些信息。但抛开具体的状态表示方法不谈，我们可以认为一个商品页就是一个状态。在下一节中，我们将以PV为单位对线上数据进行统计分析，希望能够发现这个搜索排序问题的一些特性。

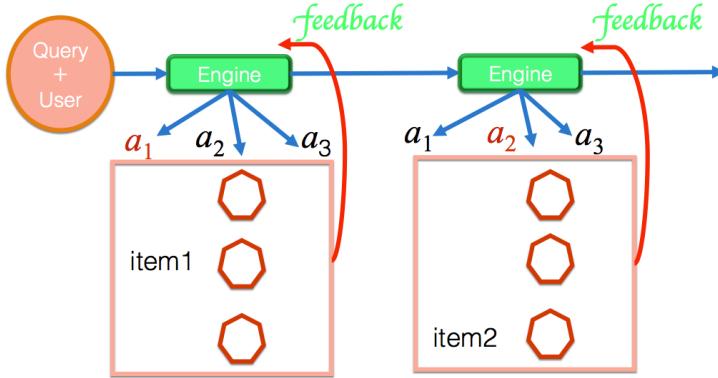


Figure 7: 搜索引擎与用户交互示意图

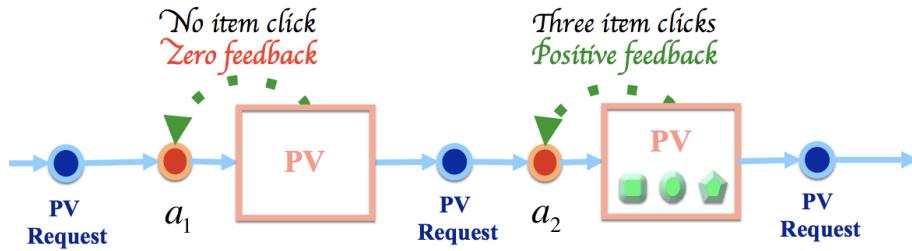


Figure 8: 搜索引擎Agent决策过程示意图

## 数据统计分析

在强化学习中，有很多算法都是分Episode进行训练的。所谓Episode是指一次从初始状态 (Initial State) 到终止状态 (Terminal State) 之间的经历。我们对线上产生的训练数据按照Episode进行了组织，也就是将每个Episode对应的所有商品展示页进行串联，形成“PV->PV->...->End”的一个序列，相当于是把Episode中的所有State进行串联。其中，“End”表示一个Episode的终止状态。在我们的场景中，终止状态表示“用户离开搜索引擎”或者“进行了购买”。由于我们在日志中无法获取“用户离开搜索引擎”这样的事件，所以我们能够完整抽取Episode的数据其实都是有成功购买的PV序列。令 $n$ 表示Episode的序列长度，我们统计了 $n = 1, 2, \dots, 30$ 的Episode占总体成交Episode的比例，结果如图9所示。

从图9的结果中，可以看到Episode的长度越大，其对应的占总体的比例越小。这与“越往后的PV转化率越低”的经验是相符的。从绝对数值上看，超过60%的成交都是在前6个PV中发生的，而 $n = 1, 2, 3$ 的比例更是分别超过了20%、15%和10%。当然，图10的结果来自于对全类目数据的统计。为了消除类目间差异给统计结果带来的影响，我们选取了某这三个成交量较大的类目，分别进行了相同的统计分析，相应的结果展示在图??中。

虽然分类目统计结果与全类目的结果在绝对数值上有一定差别，但还是呈

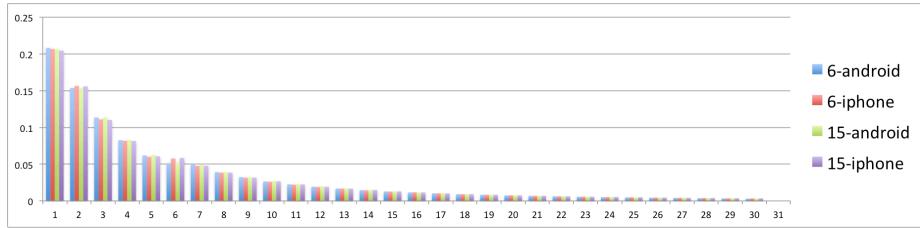


Figure 9: 全类目数据下所有成交Episode的长度分布情况

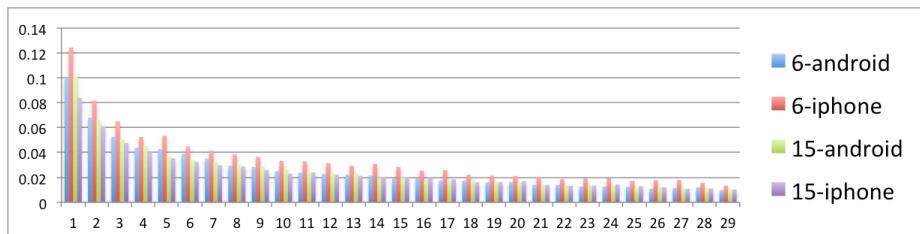


Figure 10: 类目A下所有成交Episode的长度分布情况

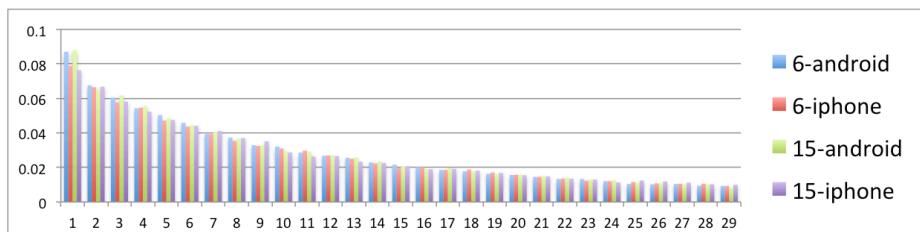


Figure 11: 类目B下所有成交Episode的长度分布情况

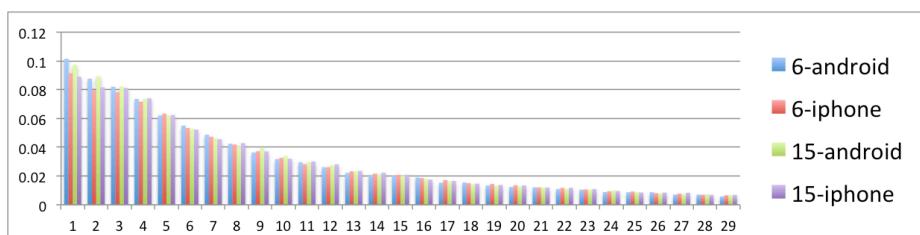


Figure 12: 类目C下所有成交Episode的长度分布情况

现出了相同的趋势。如果不考虑具体的数值，我们至少可以得出一个结论：用户在看过任意数量的商品展示页之后，都有可能发生成交。根据这个结论，我们可以将一次搜索会话过程用图13的抽象示意图来描述。如图所示，垂直方向的箭头由上向下表示用户不停翻页的过程。每翻一页，用户选择商品的范围

就增加一页，PV的History也对应地发生变化。横向地来看，用户在任意的PV History下，都有可能选择购买某个被展示的商品，或者继续往下翻页。当然，如果考虑到用户也有可能离开搜索引擎，我们可以得到图14中的更一般的示意图。

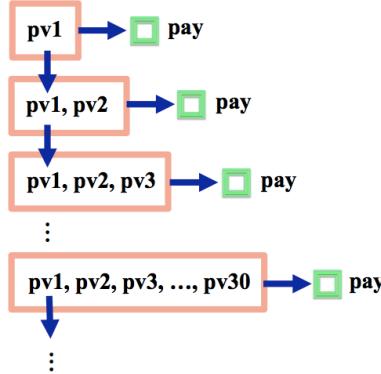


Figure 13: 仅考虑成交和翻页的搜索会话图示

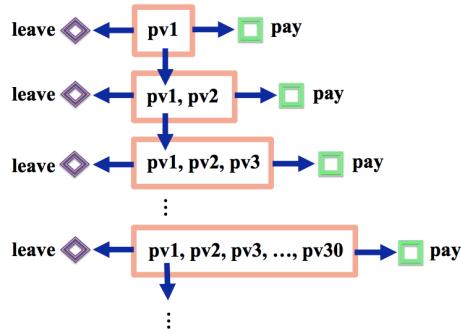


Figure 14: 考虑成交、翻页和用户离开的搜索会话图示

在我们的场景中，“成交”和“离开搜索引擎”均被视为一个Episode的终止状态。如果把图8和马尔可夫决策过程 (MDP) 的状态、状态转移等要素对应起来，就可以发现搜索排序问题的明显特征：任意非终止状态都有一定的概率转移到终止状态。这同一些典型的强化学习应用场景相比有很大不同。比如，在网格世界和迷宫问题中，只有与邻近终点的位置才有非零的概率转移到终止状态。在接下来的内容中，我们将根据搜索排序问题的特点对其进行形式化定义，并在此基础上做相应的理论分析。

### 搜索排序问题形式化

本节提出搜索会话马尔科夫决策过程模型 (Search Session Markov Decision Process, SSMDP)，作为对搜索排序问题的形式化定义。我们首先对搜索会话过程

中的上下文信息和用户行为进行建模，形式化定义商品页、商品页历史、成交转化率等概念，它们是定义状态和状态转移关系的基础。

**定义1. [Top K List]** 给定商品集合 $\mathcal{D}$ ，排序函数 $f$ ，以及一个正整数 $K$  ( $1 \leq K \leq |\mathcal{D}|$ )，关于 $\mathcal{D}$ 和 $f$ 的top  $K$  list，记为 $\mathcal{L}_K(\mathcal{D}, f)$ ，是用函数 $f$ 对 $\mathcal{D}$ 中商品进行打分以后的前 $K$ 个商品的有序列表 $(\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_K)$ 。其中， $\mathcal{I}_k$ 是排在第 $k$ 位的商品 ( $1 \leq k \leq K$ )，并且对于任意 $k' \geq k$ ，都有 $f(\mathcal{I}_k) > f(\mathcal{I}_{k'})$ 。

**定义2. [Item Page]** 令 $\mathcal{D}$ 为关于某个query的商品全集， $K$  ( $K > 0$ ) 为一个页面能够展示的商品数量。对于一个搜索会话的任意时间步 $t$  ( $t \geq 1$ )，其对应的item page  $p_t$ 是关于的第 $(t-1)$ 步的打分函数 $a_{t-1}$ 和未展示商品集合 $\mathcal{D}_{t-1}$ 的top  $K$  list  $\mathcal{L}_K(\mathcal{D}_{t-1}, a_{t-1})$ 。对于初始时间步 $t = 0$ ，有 $\mathcal{D}_0 = \mathcal{D}$ 。对于其他任意时间步 $t \geq 1$ ，有 $\mathcal{D}_t = \mathcal{D}_{t-1} \setminus p_t$ 。

**定义3. [Item Page History]** 令 $q$ 为一个搜索会话的query。对于初始时间步 $t = 0$ ，对应的初始item page history为 $h_0 = q$ 。对于任意其他时间步 $t \geq 1$ ，对应的item page history为 $h_t = (h_{t-1}, p_t)$ 。在这里， $h_{t-1}$ 为第 $(t-1)$ 步的item page history， $p_t$ 为第 $t$ 步的item page。

对于任意时间步骤 $t$ ，item page history  $h_t$ 包含了用户在 $t$ 时刻能够观察到的所以上下文信息。由于商品全集 $\mathcal{D}$ 是一个有限集合，不难发现一个搜索会话最多包含 $\lceil \frac{|\mathcal{D}|}{K} \rceil$ 个item page。对于搜索引擎来讲，它在一个搜索会话中最多决策 $\lceil \frac{|\mathcal{D}|}{K} \rceil$ 次。根据我们之前的数据分析，不同的用户会在不同的时间步上选择购买或者离开。如果我们把所有用户看作一个能够采样出不同用户行为的environment，就意味着这个environment可能会在任意时间步上以一定的成交转化概率 (conversion probability) 或者放弃概率 (abandon probability) 来终止一个搜索会话。我们形式化定义这两种概率如下。

**定义4. [Conversion Probability]** 对于一个搜索会话中的任意item page history  $h_t$  ( $t > 0$ )，令 $B(h_t)$ 表示用户在观察到 $h_t$ 之后发生购买行为的随机事件，则 $h_t$ 的conversion probability，记为 $b(h_t)$ ，就是事件 $B(h_t)$ 在 $h_t$ 下发生的概率。

**定义5. [Abandon Probability]** 对于一个搜索会话中的任意item page history  $h_t$  ( $t > 0$ )，令 $L(h_t)$ 表示用户在观察到 $h_t$ 之后离开搜索会话的随机事件，则 $h_t$ 的abandon probability，记为 $l(h_t)$ ，就是事件 $L(h_t)$ 在 $h_t$ 下发生的概率。

由于 $h_t$ 是在 $(t-1)$ 时刻的item page history  $h_{t-1}$ 上执行动作 $a_{t-1}$ 的直接结果，因此 $b(h_t)$ 和 $l(h_t)$ 也表征了Agent在 $h_{t-1}$ 上执行动作 $a_{t-1}$ 之后环境状态的转移：

(1) 以 $b(h_t)$ 的成交概率终止搜索会话； (2) 以 $l(h_t)$ 的离开概率终止搜索会话； (3) 以 $(1 - b(h_t) - l(h_t))$ 的概率继续搜索会话。方便起见，我们对用户继续进行搜索会话对概率也进行形式化描述。

**定义6. [Continuing Probability]** 对于一个搜索会话中的任意item page history  $h_t$  ( $t > 0$ )，令 $C(h_t)$ 表示用户在观察到 $h_t$ 之后继续停留在会话中的随机事件，则 $h_t$ 的continuing probability，记为 $c(h_t)$ ，就是事件 $C(h_t)$ 在 $h_t$ 下发生的概率。

显然，对于任意item page history  $h$ ，都有 $c(h) = 1 - b(h) - l(h)$ 成立。特殊地，对于初始item page history  $h_0$ 来讲， $C(h_0)$ 是一个必然事件（即 $c(h_0) = 1$ ）。这是因为在第一个item page展示给用户前，不可能存在成交转化事件和离开事件。

基于上面定义的几个概念，我们可以定义search session MDP (SSMDP) 如下： **定义7. [Search Session MDP]** 令 $q$ 为某个query， $\mathcal{D}$ 为和 $q$ 相关的商品全集， $K$ 为一个页面可展示的商品数量，关于 $q$ 、 $\mathcal{D}$ 和 $K$ 的search session MDP是一个元组，记为 $\mathcal{M} = \langle T, \mathcal{H}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$ 。该元组中的每个要素分别为：

- $T = \lceil \frac{|\mathcal{D}|}{K} \rceil$  为搜索会话最大决策步数
- $\mathcal{H} = \bigcup_{t=0}^T \mathcal{H}_t$  为关于  $q$ 、 $\mathcal{D}$  和  $K$  的所有可能的 item page history 的集合，其中  $\mathcal{H}_t$  为  $t$  时刻所有可能 item page history 的集合 ( $0 \leq t \leq T$ )
- $\mathcal{S} = \mathcal{H}_C \cup \mathcal{H}_B \cup \mathcal{H}_L$  为状态空间， $\mathcal{H}_C = \{C(h_t) | \forall h_t \in \mathcal{H}_t, 0 \leq t < T\}$  是包含所有的继续会话事件的非终止状态集合 (nonterminal state set)， $\mathcal{H}_B = \{B(h_t) | \forall h_t \in \mathcal{H}_t, 0 < t \leq T\}$  和  $\mathcal{H}_L = \{L(h_t) | \forall h_t \in \mathcal{H}_t, 0 < t \leq T\}$  分别是包含所有成交转化事件和离开事件的终止状态集合 (terminal state set)
- $\mathcal{A}$  为动作空间，包含搜索引擎所有可能的排序打分函数
- $\mathcal{R} : \mathcal{H}_C \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  为奖赏函数
- $\mathcal{P} : \mathcal{H}_C \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$  为状态转移函数，对于任意时间步  $t$  ( $0 \leq t < T$ )、任意 item page history  $h_t \in \mathcal{H}_t$  和任意动作  $a \in \mathcal{A}$ ，令  $h_{t+1} = (h_t, \mathcal{L}_K(\mathcal{D}_t, a))$ ，则 agent 在状态  $C(h_t)$  上执行动作  $a$  后，环境转移到任意状态  $s' \in \mathcal{S}$  的概率为

$$\mathcal{P}(C(h_t), a, s') = \begin{cases} b(h_{t+1}) & \text{if } s' = B(h_{t+1}), \\ l(h_{t+1}) & \text{if } s' = L(h_{t+1}), \\ c(h_{t+1}) & \text{if } s' = C(h_{t+1}), \\ 0 & \text{otherwise.} \end{cases} \quad (28)$$

在一个 search session MDP 中，环境即是所有可能用户共同构成的总体，环境的状态表征了用户总体在对应 item page history 下的动向（继续会话、成交或离开）。环境状态的转移则直接基于我们之前定义的 conversion probability、abandon probability 以及 continuation probability。奖赏函数  $\mathcal{R}$  可以根据具体的业务目标进行定义。基于让天下没有难做的生意的使命，也就是尽可能多地促进用户与卖家之间的交易，我们给出如下的奖赏函数范例。对于任意时间步  $t$  ( $0 \leq t < T$ )、任意 item page history  $h_t \in \mathcal{H}_t$  和任意动作  $a \in \mathcal{A}$ ，令  $h_{t+1} = (h_t, \mathcal{L}_K(\mathcal{D}_t, a))$ ，则 agent 在状态  $C(h_t)$  上执行动作  $a$  并且环境转移到任意状态  $s' \in \mathcal{S}$  的奖赏为

$$\mathcal{R}(C(h_t), a, s') = \begin{cases} m(h_{t+1}) & \text{if } s' = B(h_{t+1}), \\ 0 & \text{otherwise,} \end{cases} \quad (29)$$

其中， $m(h_{t+1})$  表示 item page history  $h_{t+1}$  对应的成交均价。

## 理论分析

### 马尔可夫性质

上一节定义的 search session MDP (SSMDP) 可以看作是 MDP 模型的一个实例，但要保证 SSMDP 是良定义的，我们需要证明 SSMDP 中的状态都具有马

尔可夫性质 (Markov Property)。马尔可夫性质指的是对于任意的状态动作序列  $s_0, a_0, s_1, a_1, s_2, \dots, s_{t-1}, a_{t-1}, s_t$ , 都有如下等式成立:

$$\Pr(s_t | s_0, a_0, s_1, a_1, \dots, s_{t-1}, a_{t-1}) = \Pr(s_t | s_{t-1}, a_{t-1}). \quad (30)$$

也即是说, 当前状态  $s_t$  的发生概率仅仅取决于最近一个状态动作对  $(s_{t-1}, a_{t-1})$ , 而并非整个序列。我们可以证明对于一个SSMDP, 它的所有状态都具有马尔可夫性质。

**命题1.** 对于任意 search session MDP  $\mathcal{M} = \langle T, \mathcal{H}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$ , 其状态空间  $\mathcal{S}$  中的任意状态都具有马尔可夫性质。证明: 我们只需证明对于任意时间步  $t$  ( $0 \leq t \leq T$ ) 和关于  $t$  的任意状态动作序列  $s_0, a_0, s_1, a_1, \dots, s_{t-1}, a_{t-1}, s_t$ , 都有等式  $\Pr(s_t | s_0, a_0, s_1, a_1, \dots, s_{t-1}, a_{t-1}) = \Pr(s_t | s_{t-1}, a_{t-1})$  成立即可。

除了状态  $s_t$  以外, 序列  $s_0, a_0, s_1, a_1, \dots, s_{t-1}, a_{t-1}, s_t$  中的其他所有状态都是非终止状态 (non-terminal state)。根据状态的定义, 对于任意时间步  $t'$  ( $0 < t' < t$ ), 必然存在一个 item page history  $h_{t'}$  与状态  $s_{t'}$  相对应, 且有  $s_{t'} = C(h(t'))$ 。因此, 整个序列可以重写为  $C(h_0), a_0, C(h_1), a_1, \dots, C(h_{t-1}), a_{t-1}, s_t$ 。需注意的是, 对于任意时间步  $t'$  ( $0 < t' < t$ ), 都有

$$h_{t'} = (h_{t'-1}, \mathcal{L}_K(\mathcal{D}_{t'-1}, a_{t'-1})), \quad (31)$$

成立。其中,  $\mathcal{L}_K(\mathcal{D}_{t'-1}, a_{t'-1})$  也就是关于  $(t' - 1)$  时刻的动作  $a_{t'-1}$  和未展示商品  $\mathcal{D}_{t'-1}$  的 top  $K$  list。给定  $h_{t'-1}$ , 集合  $\mathcal{D}_{t'-1}$  一定是确定的。所以,  $h_{t'}$  也就是状态动作对  $(C(h_{t'-1}), a_{t'-1})$  的必然和唯一结果。那么事件  $(C(h_{t'-1}), a_{t'-1})$  也就能够等价地表示为事件  $h_{t'}$ 。基于此, 我们可以进行如下推导:

$$\begin{aligned} & \Pr(s_t | s_0, a_0, s_1, a_1, \dots, s_{t-1}, a_{t-1}) \\ &= \Pr(s_t | C(h_0), a_0, C(h_1), a_1, \dots, C(h_{t-1}), a_{t-1}) \\ &= \Pr(s_t | h_1, h_2, \dots, h_{t-1}, C(h_{t-1}), a_{t-1}) \\ &= \Pr(s_t | h_{t-1}, C(h_{t-1}), a_{t-1}) \\ &= \Pr(s_t | C(h_{t-1}), a_{t-1}) \\ &= \Pr(s_t | s_{t-1}, a_{t-1}). \end{aligned} \quad (32)$$

第三步推导成立是由于对于任意时间步  $t'$  ( $0 < t' < t$ ),  $h_{t'-1}$  都包含在  $h_{t'}$  中。类似地, 第四步推导成立是由于事件  $C(h_{t-1})$  已经包含了  $h_{t-1}$  的发生。

### 折扣率

在这一小节我们将讨论本文最重要的问题: 延迟奖赏 (delay reward) 对于搜索排序的优化到底有没有作用? 简单地来说, 也就是 search session MDP 的折扣率 (discount rate) 到底应该设多大。在任意 MDP 中, 折扣率  $\gamma$  的大小直接决定了 future rewards 在 agent 的优化目标中所占比重。我们将分析优化长期累积奖赏与优化搜索引擎的经济指标这两个目标之间的关系给出答案。

令  $\mathcal{M} = \langle T, \mathcal{H}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$  为一个关于 query  $q$ 、商品全集  $\mathcal{D}$  和正整数  $K$  ( $K > 0$ ) 的 SSMDP。给定一个确定性策略  $\pi : \mathcal{S} \rightarrow \mathcal{A}$ , 记每个时间步  $t$  ( $0 \leq t \leq T$ ) 对应的 item page history 为  $\pi$  by  $h_t^\pi$ , 我们把在策略  $\pi$  下能够访问的所有状态都展示在图 15 中。

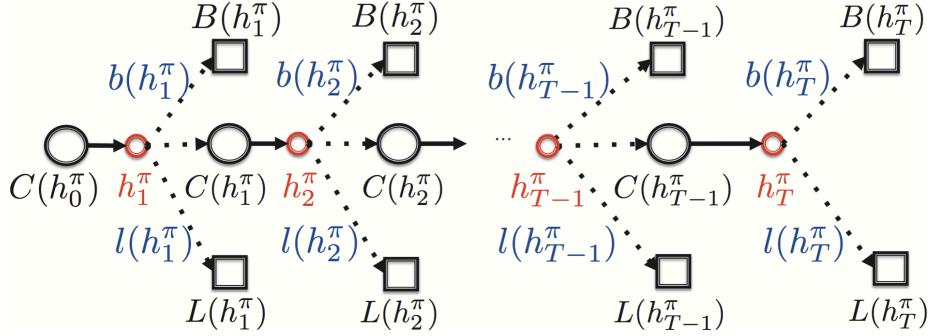


Figure 15: Agent策略 $\pi$ 下能够访问SSMDP中的所有状态

在这个图中，红色的节点表示item page history，注意它们并不是SSMDP的状态。方便起见，在本文接下来的部分，我们将把 $C(h_t^\pi)$ 、 $c(h_t^\pi)$ 、 $b(h_t^\pi)$ 和 $m(h_t^\pi)$ 分别简化记为 $C_t^\pi$ 、 $c_t^\pi$ 、 $b_t^\pi$ 和 $m_t^\pi$ 。

不失一般性，我们设SSMDP  $\mathcal{M}$ 的折扣率为 $\gamma$  ( $0 \leq \gamma \leq 1$ )。由于SSMDP是一个有限时间步MDP (finite-horizon MDP)，所以折扣率可以取到1。对于任意时间步 $t$  ( $0 \leq t < T$ )，状态 $C_t^\pi$ 的state value为

$$\begin{aligned} V_\gamma^\pi(C_t^\pi) &= \mathbb{E}^\pi \left\{ \sum_{k=1}^{T-t} \gamma^{k-1} r_{t+k} | C_t^\pi \right\} \\ &= \mathbb{E}^\pi \left\{ r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-t-1} r_T | C_t^\pi \right\}. \end{aligned} \quad (33)$$

其中，对任意 $k$  ( $1 \leq k \leq T-t$ )， $r_{t+k}$ 为agent在未来时刻( $t+k$ )的item page history  $h_{t+k}^\pi$ 中收到的即时奖赏。根据我们奖赏函数的定义， $r_{t+k}$ 在策略 $\pi$ 下的期望值为 $\mathbb{E}^\pi \{r_{t+k}\} = b_{t+k}^\pi m_{t+k}^\pi$ 。在这里， $m_{t+k}^\pi = m(h_{t+k}^\pi)$ 为item page history  $h_{t+k}^\pi$ 的成交价格期望。由于 $V_\gamma^\pi(C_t^\pi)$ 表达的是在 $C_t^\pi$ 发生的条件下的长期累积奖赏期望，所以我们还要把从 $C_t^\pi$ 到达item page history  $h_{t+k}^\pi$ 的概率考虑进来。记从状态 $C_t^\pi$ 到达 $h_{t+k}^\pi$ 的概率为 $\Pr(C_t^\pi \rightarrow h_{t+k}^\pi)$ ，根据状态转移函数的定义可以得到

$$\Pr(C_t^\pi \rightarrow h_{t+k}^\pi) = \begin{cases} 1.0 & k = 1, \\ \prod_{j=1}^{k-1} c_{t+j}^\pi & 1 < k \leq T-t. \end{cases} \quad (34)$$

从状态 $C_t^\pi$ 到item page history  $h_{t+1}^\pi$ 的概率为1是因为 $h_{t+1}^\pi$ 是状态动作对 $(C_t^\pi, \pi(C_t^\pi))$ 的直接结果。将上面的几个公式综合起来，我们可以进一步计算 $V_\gamma^\pi(C_t^\pi)$ 如下：

$$\begin{aligned} V_\gamma^\pi(C_t^\pi) &= \mathbb{E}^\pi \{ r_{t+1} | C_t^\pi \} + \gamma \mathbb{E}^\pi \{ r_{t+2} | C_t^\pi \} + \cdots + \gamma^{T-t-1} \mathbb{E}^\pi \{ r_T | C_t^\pi \} \\ &= \sum_{k=1}^{T-t} \gamma^{k-1} \Pr(C_t^\pi \rightarrow h_{t+k}^\pi) b_{t+k}^\pi m_{t+k}^\pi \\ &= b_{t+1}^\pi m_{t+1}^\pi + \gamma c_{t+1}^\pi b_{t+2}^\pi m_{t+2}^\pi + \cdots + \gamma^{T-t-1} (\prod_{j=1}^{T-t-1} c_{t+j}^\pi) b_T^\pi m_T^\pi \\ &= b_{t+1}^\pi m_{t+1}^\pi + \sum_{k=2}^{T-t} \gamma^{k-1} ((\prod_{j=1}^{k-1} c_{t+j}^\pi) b_{t+k}^\pi m_{t+k}^\pi). \end{aligned} \quad (35)$$

根据图15中展示的每个item page history的conversion probability以及成交价格期望，我们也可以将搜索引擎在策略 $\pi$ 的作用下在一个搜索会话中引导的成交额期望表达出来，即

$$\begin{aligned}\mathbb{E}_{gmv}^{\pi} &= b_1^{\pi} m_1^{\pi} + c_1^{\pi} b_2^{\pi} m_2^{\pi} + \cdots + (\Pi_{k=1}^T c_k^{\pi}) b_T^{\pi} m_T^{\pi} \\ &= b_1^{\pi} m_1^{\pi} + \sum_{k=2}^T (\Pi_{j=1}^{k-1} c_j^{\pi}) b_k^{\pi} m_k^{\pi}.\end{aligned}\quad (36)$$

通过比较 $\mathbb{E}_{gmv}$ 和 $V_{\gamma}^{\pi}$ ，不难发现当折扣率 $\gamma = 1$ 时，有 $\mathbb{E}_{gmv}^{\pi} = V_{\gamma}^{\pi}(C_0^{\pi})$ 成立。也就是说，当 $\gamma = 1$ 时，最大化长期累积奖赏将直接带来搜索引擎成交额的最大化。当 $\gamma < 1$ 时，由于 $\mathbb{E}_{gmv}^{\pi}$ 是 $V_{\gamma}^{\pi}(C_0^{\pi})$ 的上界，所以最大化 $V_{\gamma}^{\pi}$ 并不一定能够最大化 $\mathbb{E}_{gmv}^{\pi}$ 。

**命题2.**令 $\mathcal{M} = \langle T, \mathcal{H}, \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$ 为任意search session MDP。对于任意确定性策略 $\pi : \mathcal{S} \rightarrow \mathcal{A}$ 和折扣率 $\gamma$  ( $0 \leq \gamma \leq 1$ )，都有式子 $V_{\gamma}^{\pi}(C(h_0)) \leq \mathbb{E}_{gmv}^{\pi}$ 成立，其中 $V_{\gamma}^{\pi}$ 为agent在策略 $\pi$ 和折扣率 $\gamma$ 下的状态值函数， $C(h_0)$ 为搜索会话的初始状态， $\mathbb{E}_{gmv}^{\pi}$ 为搜索引擎在策略 $\pi$ 下的单次搜索会话成交额期望。仅当 $\gamma = 1$ 时，有 $V_{\gamma}^{\pi}(C(h_0)) = \mathbb{E}_{gmv}^{\pi}$ 成立。证明：我们只需证明当 $\gamma < 1$ 时，有 $V_{\gamma}^{\pi}(C(h_0)) < \mathbb{E}_{gmv}^{\pi}$ 成立。这是显然的，因为二者之差，即 $\sum_{k=2}^T (1-\gamma^{k-1})(\Pi_{j=1}^{k-1} c_j^{\pi}) b_k^{\pi} m_k^{\pi}$ 在 $\gamma < 1$ 时一定为正。

至此，我们可以回答之前提出的问题：站在提高搜索引擎成交额的角度，搜索排序问题中考虑延迟奖赏是必要且必须的。从理论上，这是因为最大化无折扣累积奖赏能够直接优化搜索引擎的成交额。究其深层原因，是因为用户在搜索商品的每个步骤（即每个item page history）的行为都是基于之前观察到的所有信息（或者大部分信息）作出的反应，这天然决定了搜索排序问题的sequential decision-making本质。

## 实验分析

我们根据图9设计了一个搜索排序的模拟实验来说明上一节理论分析结果。我们设定一个搜索会话的最大决策步数为30，因为从第2节的数据分析来看，长度为30的PV序列总占比不超过1%。我们通过线上真实数据生成商品候选集合。在每个状态上，agent可以选择动作集 $A = \{a_1, a_2, a_3, a_4, a_5\}$ 中的任意一个动作对商品进行排序，进行页面展示。对于任意时间步 $t$  ( $0 < t \leq 30$ ) 以及任意item page history  $h_t$ ， $h_t$ 通过其最近4个商品页的商品特征来进行表示。同时， $h_t$ 的conversion probability、continuing probability以及abandon probability也直接通过 $h_t$ 的特征生成。

由于实验规模不大，我们可以直接用Tabular强化学习方法对问题进行求解。我们分别采用Q-learning算法、Actor-Critic方法和Dynamic Programming方法，在折扣率 $\gamma$ 为0、0.1、0.5、0.9和1.0的情况下进行测试。每一个实验设置运行算法50次，每一次包含80万次搜索会话，我们记录下学习过程中每个搜索会话的平均成交额以及初始状态 $h_0$ 的state value。这里仅给出部分结果。我们首先对比两个极端情况 $\gamma = 0$ 和 $\gamma = 1.0$ 时，每个算法取得的GMV指标。如图16所示，三个算法在 $\gamma = 1.0$ 时的GMV都要高于 $\gamma = 0$ 时的GMV。单看结果最直观的DP方法，可以发现它在两个折扣因子下的GMV有6%的gap。如前所述，折扣因子 $\gamma = 0$ 即是分别独立优化Agent在每个状态上的即时奖赏，而 $\gamma = 1.0$ 则是直接优化GMV。

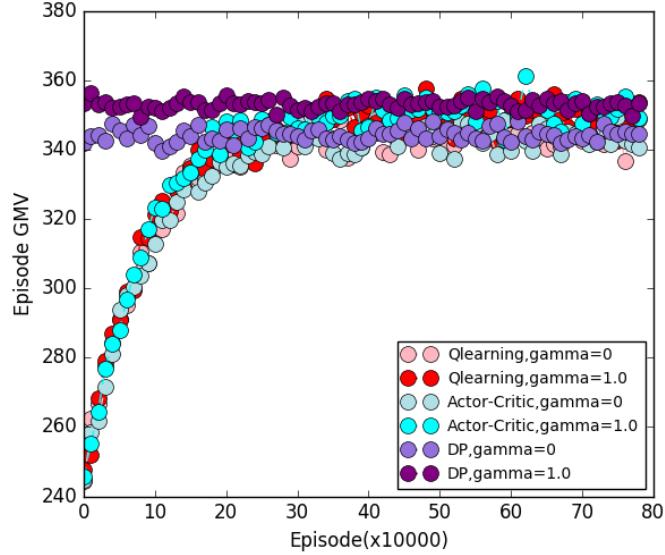


Figure 16:  $\gamma = 0$ 和 $\gamma = 1$ 情况下测试算法的GMV曲线

我们选取Actor-Critic方法，考察它在不同折扣率下的GMV曲线以及初始状态 $h_0$ 的state value  $V_\gamma(h_0)$ 的变化情况，结果如图17所示。可以看到，Actor-Critic方法在不同折扣率下的GMV指标与图16的结果一致，而图右边的值函数变化曲线更直接证明我之前的理论分析。只有折扣因子 $\gamma = 1.0$ 时， $V_\gamma(h_0)$ 的值才和对应的GMV几乎相等。需要注意的是，Actor-Critic方法在 $\gamma$ 等于0.9和1.0时的GMV曲线基本重合，并不能说明优化 $V_{0.9}$ 能够优化GMV，而只能说明优化 $V_{0.9}$ 与优化 $V_{1.0}$ 得到的最优策略恰好相同，二者本质上并不一样。

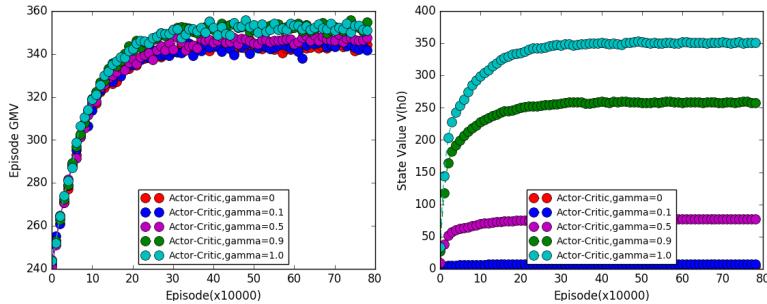


Figure 17: Actor-Critic方法在各个折扣率设置下的GMV曲线和值函数曲线

至此，我们已经通过理论分析和实验证明了搜索排序问题是一个有限时间步的无折扣（Finite-Horizon Undiscounted）的顺序决策问题。当然，我们所有的

结论都是在以最大化成交额为目标的前提下得到的。如果追求点击率或者单纯转化率之类的目标，结论可能会有不同，但都可以采用与本文类似的方法进行分析。

## 基于强化学习的引擎性能优化

### 背景

和谷歌类似，淘宝的搜索排序是建立在数十种排序因子（当然，谷歌有数百种）之上的，随着近年来深度模型的广泛应用，越来越多的复杂且耗时的因子被引入到搜索排序中，这一方面带来了排序效果上的收益，另一方面，也对线上引擎的性能带来了新的挑战，而这样的挑战不仅来自于高耗时排序策略无法全量生效，也来自于双11这样的突发性高流量对引擎的瞬间压力。

通常来说，面对这样的大规模流量访问，当引擎的处理能力不足时，通常有2种做法：一种是算法端准备一个廉价的方案，去掉效果好但耗时高的因子，这个方案比最好的策略差很多，但是引擎肯定可以扛得住；另一种是引擎端执行临时性的降级方案，比如，下线不重要业务、减少召回数量、通过粗排过滤更多宝贝等方法。可以看到，不管哪一种，都是效果对性能的硬妥协（hard compromise），所以我们尝试问自己这样一个问题，can we make it softer, and smarter?

当然，答案是，完全有可能！实际上，当我们观察我们线上的排序因子，我们发现，即使每一个因子的上线初期都经过了A/B测试验证了其有效性，但总体来看，因子之间的相关性仍然很高，我们抽取了一个子集，计算了两两之间的皮尔逊积矩相关系数（Pearson product-moment correlation coefficient），如下图所示：

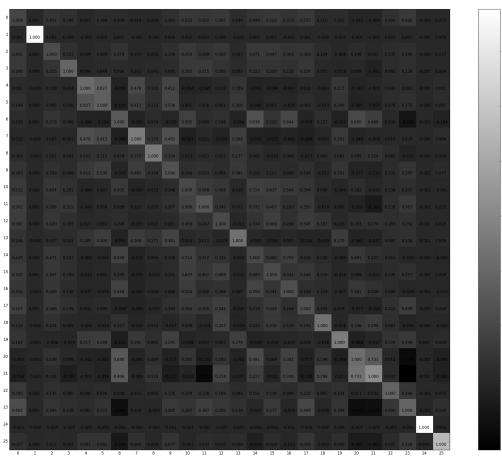


Figure 18: 排序因子的皮尔逊积矩相关系数矩阵

格子中颜色越白，对应的2个因子之间的相关性就越高，不难看出存在大量的相关因子对；另一方面，我们还发现，不同的上下文（context，这里指得是 $\langle user, query \rangle$ 对）下，商品的转化率也是差异很大，举个例子，高购买力的用户在长尾query上的查询，转化率通常要比平均高很多，因此我们猜想，在类似这样的流量下，是不是仅仅使用廉价的因子就够了？

## 问题建模

如上节所介绍，对于一次搜索请求 $(u, q)_i$ ，淘宝的排序引擎会依次计算待排序的每个文档 $d$ 的 $n$ 个排序因子，即 $s(d) = (x_1(d), x_2(d), \dots, x_n(d)) \in R^n$ 。当然，部分因子的计算会同时依赖 $u, q$ 和 $d$ ，但这一点对我们考虑的问题是透明的，所以不失一般性，我们用 $x_j(d)$ 来指代 $x_j(u, q, d)$ 。最后将这些因子输入到一个最终ranker进行总分的计算

$$F(u, q, d) = f(x_1(d), x_2(d), \dots, x_n(d)) \quad (37)$$

值得指出的是，这里并没有对 $f$ 的形式做任何假设，可以是一个线性模型，一个DNN网络，甚至是一个GBDT。

我们先考虑某一个特定context的优化，即对某个 $(u, q)$ 表示，召回和海选之后还有 $m$ 个商品待排序，我们可以使用全部的因子集合 $\Omega$ 计算总得分 $F_o = [f(s(d_1)), f(s(d_2)), \dots, f(s(d_m))]$ ，亦选取一个子集 $S \subset \Omega$ ，计算近似总得分 $F_a = [f(\pi_S(s(d_1))), f(\pi_S(s(d_2))), \dots, f(\pi_S(s(d_m)))]$ ，这里的 $\pi_S(\cdot)$ 指的是因子全集向子集的映射，于是我们的目标可以写成

$$\min_{S \subset \Omega} D(F_o || F_a) + \lambda ||S||_0 \quad (38)$$

这里 $D(p || q)$ 表示的是KL距离，而目标中的第二项表示是子集的大小，目标的直觉含义是，在尽量用少的因子的情况下，最大可能的逼近原先的排序函数 $F_o$ 。

然后，即使对单个context，Eq. (2) 都不是特别好解的问题，其本质上是一个optimal subset selection问题，可以证明是NP-Hard问题。换言之，我们要尝试对所有的 $(u, q)$ ，都要分别求解一个NP-Hard问题，这显然是不现实的。解决这个问题的第一步，是把最优子集的解在context特征层上进行泛化，即我们不直接求解子集，而是通过定义：

$$S_{u,q} = H(u, q | \theta) \quad (39)$$

转而去求解一个全局的模型参数 $\theta$ 。同时，在机器学习的多个子领域都有optimal subset selection的问题，例如feature selection, ensemble pruning等，其中不乏很多有效的启发式以及近似算法。受Google近年来的工作的启发（通过DRL求解TSP问题），我们将子集选取定义为一个最优决策序列，序列的奖赏，即reward，则可以定义为我们想要的metric，例如将我们的loss取反。由于reward可以在模拟环境中得到，因此可以通过离线的充分训练，让模型有机会探索到更优的解，同时通过策略梯度更新模型，直至收敛。

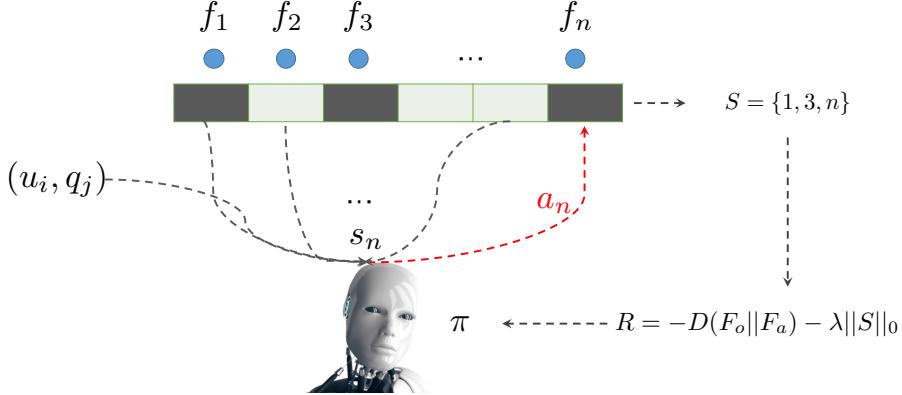


Figure 19: 因子选择的序列决策模型

状态的定义是一个关键环节，在我们的方案中，我们将排序模型作为环境相应Agent的动作请求，而Agent通过转移状态和reward来决定下一次的动作。因此，如何设计状态等MDP的关键环节成为方案的核心。我们将一步一步展开我们的设计。

### 状态定义

为了能够包含contexts的信息，我们在状态中引入了用过的特征（包括：年龄，性别，购买力），query特征（query行业，包含二级类目），记录前面步骤的决策的动作 $a_i$ 以及T当前决策的总步骤T。所以最终状态的定义如下：

$$s = (age, sex, power, a_1, a_2, \dots, a_n, T) \quad (40)$$

这样状态不仅包含了用户与query的上下文信息，同时包含了历史的决策信息。由于状态特征不同维度的尺度不一样，我们会将所有维度的特征值归一化到[0, 1]区间内，再进行后续处理。

### 动作空间设计

对于每个状态 $s$ ，动作 $a_i \in \{Skip, Eval\}$ ，其中 Eval代表feature  $x_i$ 被保留作为排序feature，反之，skip代表feature  $x_i$ 不被保留作为排序标准。

### 状态转移函数

状态转移函数 $T$ 我们的设计方中比较简单，如下图所示：

agent根据当前状态 $s$ ，做出决策，选择动作 $a_k$ ；这时将动作 $a_k$ 存储在 $s'$ 中，同时最后一维计数 $k + 1$ 。重复以上过程直到最后一维到达某个值。

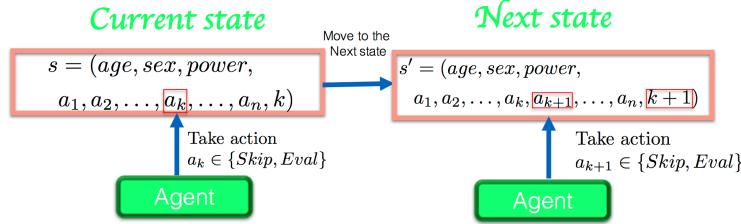


Figure 20: 状态转移建模

### 奖赏函数的设计

奖赏函数在我们的方案中处于核心位置，正是由于正确的奖赏函数设计，才使得强化学习算法在保证 ranking effectiveness，同时最大化的节省搜索引擎的性能开销。首先，我们定义一个 $\{Skip, Eval\}$  到 $\{0, 1\}$ 的 $b$ 函数

$$b(a_k) = \begin{cases} 0 & \text{if } a_k = \text{Skip} \\ 1 & \text{if } a_k = \text{Eval} \end{cases} \quad (41)$$

我们的主要目标是在保持排序的有效性（比如我们的方案是保持原有的序）的基础下，同时尽最大的可能减少 feature 的使用。所以我们的奖赏函数在鼓励减少使用 Feature 的同时，当排序结果太差的时候会给出一个惩罚 (penalty). 定义如下：

$$T(s_k, a_k) = \begin{cases} r_p & t < C \\ 0 & \text{otherwise.} \end{cases} \quad (42)$$

其中， $t = \sum_{j=1}^{n_i} \sum_{k=1, k \neq j}^{n_i} \mathbf{1}_{\pi_{opt}}(\pi_i^{\mathbb{I}}(j) > \pi_i^{\mathbb{I}}(k))$  定义了两个排序的 pairwise loss。如果这个 loss 太大，超过某个阈值  $C$ ，就会触发一个很大的惩罚  $r_p$ ，使得 agent 减少 drop feature 的数量。然后我们在定义没有没有 penalty 的奖赏函数：

$$\hat{R}(s_k, a_k) = \begin{cases} 0 & \text{if } a_k = \text{Skip} \\ c_k^{i,j} & \text{if } a_k = \text{Eval} \end{cases} \quad (43)$$

这里  $c_k^{i,j}$  的 feature  $x_k^{i,j}$  的计算开销函数。直觉上，这个设计能更多的倾向于跳过开销的 feature。最后我们把上面两个函数联合起来，就得到了我们想要的最终奖赏函数：

$$\mathcal{R}(s_k, a_k) = \hat{R}(s_k, a_k) + T(s_k, a_k). \quad (44)$$

这个函数设计既能达到节省性能开销，同时也能保证排序的有效性。

## 算法设计

直觉上来讲，在原有学习的基础我们的目的是学习一个 $b$ 向量，具体设定见下图：

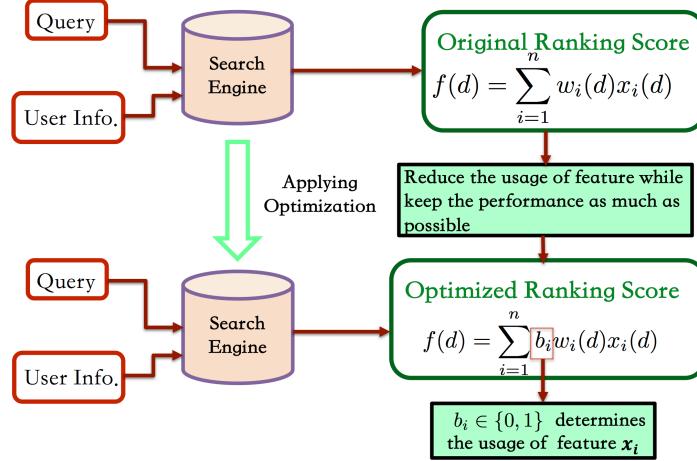


Figure 21: 系统框架图

### Loss Function

我们的Loss函数定义如下

$$\mathcal{J}(\theta) = \mathbb{E}_{q_i} [\mathcal{L}(b(\theta); q_i, w^{q_i})], \quad (45)$$

这里 $b(\theta)$ 是 $\theta \in \mathbb{R}^d$ 参数化的函数 $d > 0$ 是参数向量的维度.我们重新将奖赏函数写为：

$$\min_{\theta} \frac{1}{|\mathcal{Q}|} \sum_{i=1}^{|\mathcal{Q}|} \left( -\mathcal{J}_{\mathcal{L}, \theta} (\tilde{\theta}; q_i, w_i) + \mu \|\tilde{\theta}\|_1 \right) \quad (46)$$

这里 $\|\tilde{\theta}\|_1$ 是 $L_1$ 泛数,  $\mu$ 是正则参数, 然后

$$\mathcal{J}_{\mathcal{L}, \theta} (\tilde{\theta}, p_i, w_i) = \int_{\tau} p_{\theta}(\tau) \mathcal{R}(\tau) \log \frac{p_{\tilde{\theta}}(\tau)}{p_{\theta}(\tau)} d\tau \quad (47)$$

这里 $p_\theta(\tau)$ 是轨迹 $\tau$ 出现的概率， $\pi$ 是我们的策略函数，即

$$p_\theta(\tau) = p_0(s_0) \prod_{m=0}^{M-1} p(s_{m+1} | s_m, a_m, q_i, w_i) \pi_{\tilde{\theta}}(a_m | s_m, q_i, w_i), \quad (48)$$

$$\mathcal{R}(\tau) = \frac{1}{M} \sum_{m=0}^M \gamma^m r_{m+1}, \quad (49)$$

### Actor-critic 方法

---

#### Algorithm 1 Actor-critic training

---

```

1: Input training data set  $\mathcal{D} = \{(\mathcal{A}_i, q_i, \mathbf{y}_i)\}_{i=1}^N$ 
2: initialize actor network params  $\tilde{\theta}$ 
3: initialize critic network params  $\theta_c$ 
4: for each  $(A_i, q_i, \mathbf{y}_i) \in \mathcal{D}$  (For each episode) do
5:   initialized the initial state  $s_0$  by the query  $q_i$ 
6:   for  $k = 1, \dots, n$  ( $n$  is the number of features) do
7:     Taking action  $a_k \in \{\text{Skip}, \text{Keep}\}$  on feature  $x_k^i$ , observe  $s_{k+1}, r_k$ 
8:     Calculate actor loss by  $-\mathcal{J}_{\mathcal{L}, \theta}(\tilde{\theta}, p_i, w_i) = -\sum_{t=0}^k p_\theta(\tau) \mathcal{R}(\tau) \log \frac{p_\theta(\tau)}{p_{\tilde{\theta}}(\tau)} d\tau - \mu \|\tilde{\theta}\|_1$ 
9:     Update  $\tilde{\theta}$  by  $RMSPProp(\tilde{\theta}, -\nabla_{\theta_c} \mathcal{L}(\theta_c))$ 
10:    Calculate critic loss  $\mathcal{L}(\theta_c) = \|V_{\theta_c}^\pi(s_k) - r_{k+1} - \gamma V_{\theta_c}^\pi(s_{k+1})\|^2$ 
11:    Update  $\theta_c$  by  $ADMA(\theta_c, \nabla_{\theta_c} \mathcal{L}(\theta_c))$ 
12:   end for
13: end for

```

---

Figure 22: 算法伪代码

我们采用目前流行的Actor-critic 来优化上述的的loss function，其中我们利用一个policy网络来作为actor，然后利用一个参数化的网络来估计每个状态 $s_k$ 的值函数。因此，我们的critic网络的目标函数如下：

$$\mathcal{L}(\theta_c) = \|V_{\theta_c}^\pi(s_k) - r_{k+1} - \gamma V_{\theta_c}^\pi(s_{k+1})\|^2. \quad (50)$$

其中 $\theta_c$ 是刻画critic网络的参数。我们的算法伪代码如下：

### 理论分析

目前我们初步得到一下理论结果，更多的理论结果与分析将在以后的文章中陆续给出：

### 实验效果

基于强化学习端的训练主要是在实时计算平台完成，在引擎实时生效，日常测试成交转化和成交额均没有下跌（尽管在我们的设计中，是允许指标微跌的），同时节省了大约30%搜索引擎的耗时开销，在双十一当天，我们也上线测试，在全量已经减少精排数的基础之上，相比基准桶，再节省了20%的引擎性能开销。

**Theorem.** Suppose the optimal and optimized ranking permutations are  $\pi_{opt}$  and  $\pi_i$ , with a probability  $\delta > 0$ , the distance between  $\pi_{opt}$  and  $\pi_i$ :

$$d^{A_i}(\pi_i, \pi_{opt}) = \sum_{j=1}^{n_i} \sum_{k=1, k \neq j}^{n_i} \mathbf{1}_{\pi_{opt}}(\pi_i(j) > \pi_i(k))$$

is less than a small constant  $\epsilon > 0$ , where  $n_i$  is the size of the item set  $A_i$ ,  $\mathbf{1}_{\pi_{opt}}(\pi_i(j) > \pi_i(k)) = 1$  if  $\pi_{opt}(j) < \pi_{opt}(k)$  and  $\mathbf{1}_{\pi_{opt}}(\pi_i(j) > \pi_i(k)) = 0$  otherwise.

Figure 23: 理论结果



Figure 24: 实验结果

## 总结

我们将强化学习应用到了搜索引擎的性能优化上面，目前在业界据我们所知是首次应用。这位强化学习在工业界的应用提供了一种新的思路，也为业务压力越来越大的淘宝搜索提供了一种优化方案。目前我们已经取得了初步的实验结果，我们将继续优化我们的方法，希望能够为集团的其他基础设施也提供类似的优化方案。

## 基于强化学习分层流量调控

### 背景

福利经济学告诉我们，市场可以解决两大问题，效率和公平。在满足一定的条件下，通过市场机制可以实现帕累托最优，达到单独改变任何一个个体都不能实现更优的状态，以此实现效率的最优化。但效率最优往往是不够的，一个贫富差距巨大的社会仍然有可能是帕累托最优的，但是是一个极不稳定的状态，一个稳定的社会结构还需要考虑公平，福利经济学第二定理因此指出，通过改变个体之间禀赋的初始分配状态，仍然可以通过竞争性市场来达到帕累托有效配置，从而兼顾公平。

事实上，今天的淘宝俨然已经成为了一个规模不小的经济体，因此，社会经济学里面讨论的问题，在我们这几乎无一例外的出现了。早期的淘宝多数是通过效率优先的方式去优化商品展示的模式，从而产生了给消费者最初的刻板印象：低价爆款，这在当时是有一定的历史局限性而产生的结果，但肯定不是我们长期希望看到的情形。因为社会大环境在变化，人们的消费意识也在变化，如果我们不能同步跟上，甚至是超前布局的话，就有可能被竞争对手赶上，错失良机。因此有了我们近几年对品牌的经营，以至于现在再搜索“连衣裙”这样的词，也很难看到9块9包邮的商品，而这个在3年之前仍然很常见。而这里的品牌和客单等因素，是通过一系列的计划经济手段来进行干预的，类似于上文福

利经济学第二定理中的禀赋分配，依据的是全局的的观察和思考，很难而且也不可能通过一个局部的封闭系统（例如搜索的排序优化器）来实现。

因此，越来越多的运营和产品同学，鉴于以上的思考，提出了很多干预的分层，这里的分层指的是商品/商家类型的划分，可以从不同的维度来划分，比如，按照对平台重要性将天猫商家划分成A、B、C和D类商家；按照品牌影响力将商品划分为高调性和普通商品；按照价格将商品划分为高端、中等、低端商品等。而早期的算法同学对这些可能也不够重视，一个经典的做法即简单加权，这通常往往会带来效率上的损失，因此结果大多也是不了了之。但当我们认真审视这个问题的时候，我们其实可以预料，损失是必然的，因为一个纯粹的市场竞争会在当前的供需关系下逐步优化，达到一个局部最优，所以一旦这个局部最优点被一个大的扰动打破，其打破的瞬间必然是有效率损失的，但是其之后是有机会达到比之前的稳定点更优的地方。

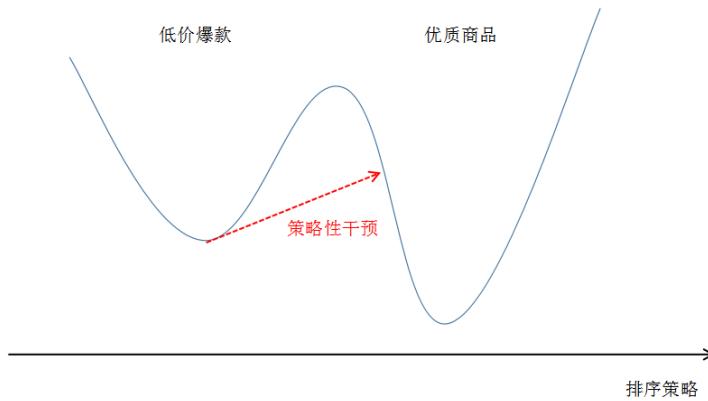


Figure 25: 局部最优和全局最优

所以，这其实给我们算法同学带来2个问题：

- 如果尽可能的减少瞬时损失？
- 如何尽快的到达新的有可能更优的局部最优点？

对应的解决方案也很自然：

- 进行个性化的干预，减少不必要的损失，例如干预的分层为物流时效，那么当时对物流不敏感而对销量更看重的那些用户，则没有必要进行很强的干预
- 通过更广泛，更smart的exploration，仍然以上面的例子，因为当前的整体排序没有考虑物流时效，所以我们的数据中就没有这样的属性，所以我们无法从监督学习来学习到类似更多次日达这样的商品被排到首页的效率会如何变化，这只能逐渐的“试”出来，再从之后的用户反馈中总结经验，是一个典型的“trial and error”的过程

所以当我们进一步抽象时，会发现这自然的定义了一个强化学习问题：个性化的干预可以看做针对不同的状态，所采取的动作不一样，而更广泛，更smart的exploration则对应着要将强化学习的搜索学习过程。

## 问题建模

我们把搜索行为看成是用户与搜索引擎交互的MDP，搜索引擎作为agent、观察到的用户和搜索词信息作为state、排序策略作为action（流量调控feature只是众多action中的一员）、用户反馈（pv/click/pay）作为reward，排序参数优化问题也可通过RL来求解。为了引入流量结构变化的影响，我们将分层流量占比的变化和用户行为反馈一起作为reward，具体地

- 搜索场景下的上下文通常包括query profile和user profile，其中query profile由query的物理属性（词性、词长度等）和淘宝属性（类目、行业等）组成，user profile由user长期的行为偏好、实时的状态和行为序列表示，因此我们将这些表示为state，记为 $s \in R^d$ 。
- 假设需要进行干预的信号有m个，把每个分层抽象成一个rank feature，如果商品属于该分层则score为1，否则为0；每个分层对应的weight组成action。一个常用的trick是，不直接输出action的绝对值，而是在神经网络的最后一个输出层，使用sigmoid（亦可使用tanh，二者是可以相互表示的），将actor网络的输出每一维限定在[0, 1]之内，即 $o \in [0, 1]^m$ ，然后再经过一个变换进行生效

$$a^k = L^k + (U^k - L_k)o^k, \forall k \in \{1, 2, \dots, m\} \quad (51)$$

这里 $U^k$ 和 $L_k$ 是第k维分层rank feature的权重的upper bound和low bound，一般来自于领域知识，但通常受限于经验的局限，因此我们尝试了一种新的方法进行自动赋值，将在下一节进行阐述。3. reward设计的第一要素即分层比例，即展示商品中属于分层商品占总商品的比例 $p_i(\pi)$ 。于此同时，由于在流量调控的同时需要兼顾效率，用户的行为反馈必须作为reward考虑的因素，反馈中考虑click、pay和cart行为，每种行为的影响因子不同， $\text{pay} > \text{cart} > \text{click}$ ，这里统一表示为每个PV中用户click、cart、pay的数量 $n_{click}, n_{cart}, n_{buy}$ 的一个函数，即 $\text{GMV}(n_{click}, n_{cart}, n_{buy})$ 。此外，分层比例需要设定对照组，举个例子，羊绒衫的搜索结果中高价商品比例明显高于毛衣，因为query本身已经体现出了价格差异，与流量调控的action并无关系，所以在计算实际的分层比例时，我们会将其原值减去同query在基准桶的分层比例 $p_i(\pi_{basic})$ ，即

$$r(s, a) = \text{GMV}(n_{click}, n_{cart}, n_{buy}) + \sum_i^m \lambda_i (p_i(\pi) - p_i(\pi_{basic})) \quad (52)$$

### Dynamic Action Boundary by CEM

上文的建模建立在PV粒度的奖赏，但是由于用户的行为的不确定性（这个不确定性一方面来自于用户的点击购买行为具有随机性，另一方面来自于我们对用户建模的不确定性），所以瞬时奖赏会有很大的variance，会对学习带来很大

的影响，所以此时如果在整个实数空间进行搜索的话，很有可能收敛不了。因此我们设计了upper bound和low bound，使得RL算法只需要在局部进行搜索，降低了学习的难度，但这又带来了2个新的问题：1. 如何确保upper bound和low bound的合理性？2. 如何防止选取了一个局部的最优区间？

针对以上2个问题，我们设计了一个通过Cross Entropy Method (CEM) 的方法来实时的动态更新action的upper bound和low bound，具体而言，我们不考虑state，考虑一个全局最优action $a_k$ ，我们假设其符合一个高斯分布，在

$$a^{k*} \in N(\mu_k, \sigma_k^2) \quad (53)$$

每次迭代的开始，我们从这个分布上采样 $s$ 个样本，即 $\Omega_1, \Omega_2, \dots, \Omega_s$ ，然后对这些action进行充分的投放，得到对应的action的一个充分置信的reward值，即

$$R(\Omega_1) = \frac{1}{N_1} \sum_i^{N_1} r_i(\Omega_1) \quad (54)$$

$$R(\Omega_2) = \frac{1}{N_2} \sum_i^{N_2} r_i(\Omega_2) \quad (55)$$

$$\dots \quad (56)$$

$$R(\Omega_s) = \frac{1}{N_s} \sum_i^{N_s} r_i(\Omega_s) \quad (57)$$

然后我们对 $R(\Omega_1), R(\Omega_2), \dots, R(\Omega_s)$ 进行排序，选取出top p的子集D，以最大化高斯分布产生这些样本的概率，即

$$\max_{\mu_k^*, \sigma_k^{2*}} f(\mu_k^*, \sigma_k^{2*}) = \sum_{i \in D} \log N(\Omega_i | \mu_k^*, \sigma_k^{2*}) \quad (58)$$

实际上，上面的式子是有最优解的， $\mu_k^*$ 即D中所有样本的均值， $\sigma_k^{2*}$ 则是所有样本的方差，但如果直接求解，则模型则会迭代过快，一方面会完全忘记之前迭代的信息，另一方面，因为这个会直接输出给上面的RL学习的actor使用，所以bound不能变化多块，否则RL很有可能不能及时跟上变化，因此，我们采用了缓慢更新的方法，即

$$\mu_k \leftarrow \mu_k + \alpha \frac{\partial f}{\partial \mu_k} \quad (59)$$

$$\sigma_k \leftarrow \sigma_k + \alpha \frac{\partial f}{\partial \sigma_k} \quad (60)$$

在更新之后，我们使用下面的方法赋值第k维action的upper bound和low bound，确保RL调节的action在一个全局较优的空间内

$$L^k = \mu_k - 2\sigma_k \quad (61)$$

$$U^k = \mu_k + 2\sigma_k \quad (62)$$

$$(63)$$

我们的RL实现选择了我们自己在AI4B中实现的DDPG，整体流程如下：

- 使用CEM选取初始upper bound和low bound
- 启动RL进行学习，于此同时，使用CEM动态调节upper bound和low bound

## 实验效果

双11期间在gmv损失可控的情况下，目标商家流量占比提升30%+。

Figure 26: 实验结果

## 总结与展望

本文的主要工作是基于强化学习的分层流量调控框架实现，在一小部分流量上探索分层调控策略对指标的影响，再结合探索策略的收益在剩余流量上精细化投放。作为流量结构调整的实施部分，框架本身还有很多需要改进的地方，在reward设计方面，不同分层流量的reward融合、分层流量reward与行为反馈reward的融合都是需要深入的方向；在探索策略设计方面，目前还是单个维度explore，效率较低，后面会尝试多个维度同时explore。另外，文章开头提到的如何评估流量结构变化的长期影响是一个更有价值的课题。

## 虚拟淘宝（联合研究项目）

### 背景

#### 强化学习面临的问题

在某些场景下应用强化学习（例如围棋游戏中的AlphaGo），进行策略探索的成本是非常低的。而在电商场景下，策略探索的成本会比较昂贵，一次策略评估可能需要一天并且差的策略往往对应着经济损失，这是在线应用强化学习遇到的一个普遍问题，限制了强化学习在真实场景下的应用。针对这个问题，我们和强化学习方面的知名专家，南京大学机器学习与数据挖掘研究所的俞扬副教授进行了深度合作，通过逆向建模环境，尝试构建了一个“淘宝模拟器”，在该模拟器上，策略探索的几乎没有成本，并且可以快速进行策略评估。而且在这样一个模拟器上，不仅可以对各种RL算法进行离线尝试，而且还可以进行各种生态模拟实验，辅助战略性决策。

## 虚拟淘宝

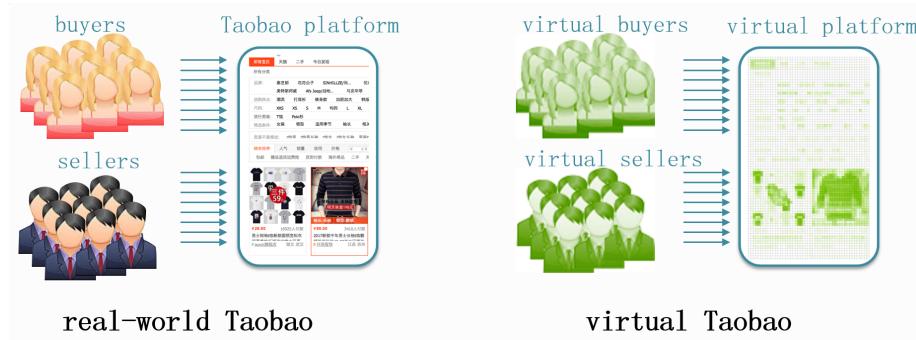


Figure 27: 真实淘宝和虚拟淘宝

## 学习用户行为：监督学习

模拟器的关键在于模拟用户的行为。传统的监督学习方法将用户的观察（observation）作为特征，用户的行为作为标签（label），试图在这些数据上训练得到用户的行为策略。

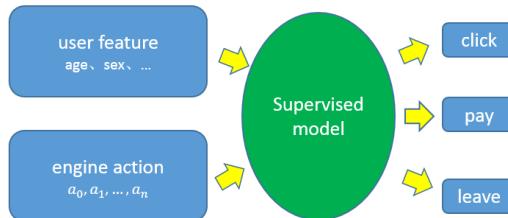


Figure 28: 监督学习方案

这种简单的方式不是很奏效，原因是数据分布高度依赖于当时的线上策略，导致数据会很不充分，这样由于方差漂移（covariate shift）带来的compounding error会使得算法失效。

## 学习用户意图：逆强化学习

### 逆强化学习概述

强化学习是求累积回报期望最大时的最优策略，在求解过程中立即回报是人为给定的。然而，在很多任务中，尤其是复杂的任务中，立即回报很难指定。那么如何获取即时回报呢？逆向强化学习的提出者Ng认为：专家在完成某项任务时，其决策往往是最优的或接近最优的，那么可以这样假设，当所有的策略所

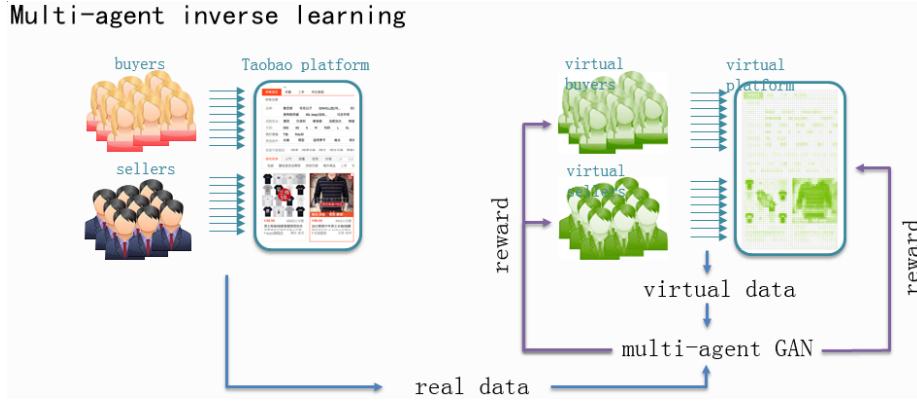


Figure 29: 多智能体逆强化学习

产生的累积回报期望都不比专家策略所产生的累积回报期望大时，强化学习所对应的回报函数就是根据示例学到的回报函数。简单地讲，逆向强化学习可以定义为从专家示例中学到回报函数。传统强化学习在很多复杂问题上难以学得较优策略，而逆强化学习通过专家策略，往往能够取得更好的效果。例如在预测司机行为以及规划机器人步态等问题，逆强化学习都取得了很好地效果。

### 学习用户意图

用户看到了商品，为什么会购买？我们假设，用户有一个购买商品的意图（intention），用户看到商品之后，用户本身的属性以及商品的一些属性使得用户有了购买的意图。我们用奖赏函数（reward function） $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ 表示用户的内在行为意图，其中 $\mathcal{S}$ 是用户的观察空间（包括了用户的特征以及用户看到的信息）， $\mathcal{A}$ 是用户的动作空间。那么，如何获得 $r(s, a)$ 呢。我们可以将淘宝用户视为“专家”，用逆强化学习方法，通过淘宝用户的历史行为，学出其内在的奖励函数。然后，利用强化学习方法，学习出用户的行为策略，即构建了用户行为模拟器。

### 生成对抗式模仿学习

如果给了专家历史数据，逆强化学习（IRL）能够方法能够学出专家的奖励函数，相对于行为克隆（behavior cloning）方法，该方法能够处理历史数据不够充分的问题。然而迭代使用RL方法使得IRL效率非常低。最近，理论表明显式地学习出奖励函数并非必要，可以直接学得专家策略，生成对抗式模仿学习在理论上等价于逆强化学习，并且效率更高 [Ho and Ermon, 2016]。

### 构建用户行为模拟器

#### 问题建模

淘宝搜索是建立在数十种排序因子之上的，对于用户的每一次搜索请求 $u = (user, query)$ ，淘宝的排序引擎会依次计算每个文档 $d$ 的 $n$ 个排序因子 $(x_1(d), x_2(d), \dots, x_n(d))$ ，

然后将这些因子进行加权求和得出最后的总分  $s(u, d) = \sum_{i=0}^n w_{i,u}x_i(d)$ , 用  $s(u, d)$  进行排序, 最后展示排名靠前的商品。排序因子的质量和排序权重  $w(u)$  的选择, 都决定着排序结果的好坏。这里, 我们重点关注排序权重的选择问题。

为了在淘宝搜索中应用RL方法, 首先需要为优化引擎策略建立马尔科夫决策过程 (MDP)  $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$

- 状态空间  $\mathcal{S}$  我们将用户的一些特征和查询信息  $u$  作为状态  $s$ , 用户特征我们选择了用户的性别、年龄、购买力, 查询信息我们只提取到查询行业的粒度。
- 动作空间  $\mathcal{A}$  我们将排序的权重  $w \in \mathbf{R}^n$  作为动作  $a$ 。
- 奖励函数  $r$  如果用户购买了商品, 我们会返回一个正的奖励, 否则返回 0。
- 策略  $\pi$  定义参数化策略  $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$ 。

智能体 (agent) 在状态  $s_t$  下作了动作  $a_t$  之后, 应该转移到哪一个状态? 即时的奖赏应该是多少? 在很多强化学习问题 (例如atari游戏, 围棋) 中, 这都不是问题, 因为我们与环境进行交互非常方便, 我们只需要做出一个动作, 然后等待环境反馈的结果就可以了, 并不会带来更多的开销。而在电商场景下, 与环境的交互是昂贵且耗时的, 大量探索式的交互是不切实际的。

我们希望用模仿学习的方法学习用户的意图, 也就是模拟线上环境。将环境, 也就是在线的用户, 视为专家 (expert)。我们专家历史数据 (每天交易产生的大量日志), 可以从中学得用户的策略作为我们的环境。注意到, 我们日志的量虽然很大, 但它是高度有偏的, 因为只有发生购买行为的pv才会被记录, 所以基于行为克隆的模仿学习是不适用的。为了区别与训练引擎策略的MDP过程  $\mathcal{M}$ , 我们用  $\mathcal{M}_e = \langle S_e, A_e, P_e, R_e, \gamma_e \rangle$  表示模拟环境时的MDP过程。

- 状态空间  $\mathcal{S}_e$  将提取的用户特征和引擎权重作为状态。
- 动作空间  $\mathcal{A}_e$  将用户购买行为作为环境的动作。
- 奖励函数  $r_e$  训练判别器  $D_w$ , 用来判断  $\langle u, w \rangle$  是否来自真实数据, 用  $D_w$  的输出作为奖励。
- 策略  $\pi_e$  定义参数化策略  $\pi_{e\theta} : s \rightarrow a$ 。

模拟器框架如图 [30]: 输入是用户的特征 (性别、年龄、购买力、query行业), 首先, 经过引擎网络, 输出引擎的动作, 然后, 用户特征以及引擎动作经过模拟器网络产生用户行为, 即是否购买。另外, 判别器网络会根据用户特征以及引擎权重给出奖励。

## 算法设计

我们在生成对抗模仿学习 (GAIL) 的框架下, 提出了LERD (Learn Environment with Restricted Data) 算法, 过程如下:

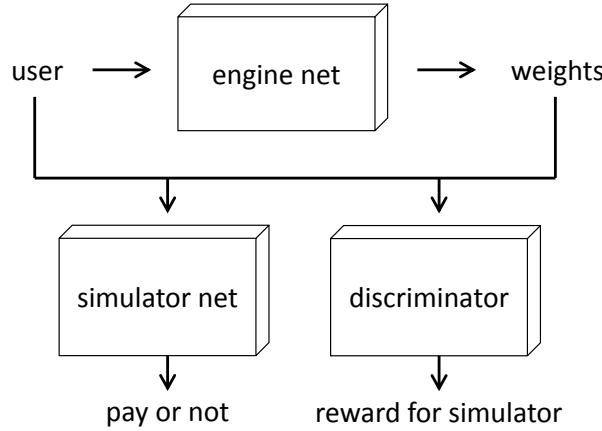


Figure 30: 模拟器网络结构

---

**Algorithm 1** LERD

---

1: **Input:** Expert trajectories  $\tau$ , initial policy  $\pi_\theta, \pi_{e\theta}$  and discriminator  $D_w$ .

2: **for**  $i = 0, 1, 2, \dots$  **do**

3:     Sample trajectories  $\tau_i$  by  $\pi_\theta, \pi_{e\theta}$

4:     Update discriminator  $D_w$  with the gradient.

$$-E_{\tau_i}[\nabla_w \log(D_w(s, a))] - E_\tau[\nabla_w \log(1 - D_w(s, a))]$$

5:     Update policy  $\pi_\theta, \pi_{e\theta}$  using some RL method with reward function  $D_w$ .

6: **end for**

---

Figure 31: LERD算法

- 初始化引擎  $\pi$  和模拟器策略  $\pi_e$  以及判别器函数  $D_w$ 。
- 循环执行，直到终止条件。
  - 用  $\pi_e$  和  $\pi$  采样出一系列轨迹  $\tau_i$ 。
  - 用真实轨迹  $\tau$  以及采样出的轨迹  $\tau_i$  更新判别器，即在如下的方向上更新  $D_w$

$$-E_{\tau_i}[\nabla_w \log(D_w(s, a))] - E_\tau[\nabla_w \log(1 - D_w(s, a))]$$

- 将  $D_w$  作为强化学习的奖励函数，用强化学习方法更新  $\pi_e, \pi$ 。

## 实验结果

初步的实验结果表明，相同的引擎策略（random策略），模拟器上模拟的购买率与真实购买率类似，同时在模拟器上训练TRPO算法，得到的策略在模拟环境上购买率有明显提升，这项工作的研究仍然在进行中。

## 参与人员

阿里巴巴 搜索事业部-AI技术及应用：胡裕靖、詹宇森、潘春香、笪庆、曾安祥

合作方 南京大学：侍竞成、陈士勇、俞扬（副教授）