Alibaba Group 阿里巴巴集团 | 阿里技术

从虚拟世界走进现实应用

强化学习在阿里的技术演进与业务创新

Reinforcement Learning Beyond Games:
To Make a Difference in Alibaba

Alibaba Group
阿里巴巴集团

阿里技术

扫一扫二维码图案，关注我吧

「阿里技术」微信公众号

「阿里技术」官方微博

钉钉扫描二维码，或者钉钉搜索群号：21716284 ，与我们一起交流

主编　　达卿　　仁重

作者　　（排名不分先后）

搜索事业部	哲予	削觚	萱然	益风
	席奈	龙楚	云志	士雍
	晨松	劲忞	冷江	
阿里妈妈事业部	探微	庙算	亮博	红桦
计算平台事业部	临在	岑鸣		
智能服务事业部	海青			

致谢　　技术战略部

（以上均为阿里花名）

序

当前的机器学习算法大致可以分为有监督的学习、无监督的学习和强化学习（Reinforcement Learning）等。强化学习和其他学习方法不同之处在于强化学习是智能系统从环境到行为映射的学习，以使奖励信号函数值最大。如果智能体的某个行为策略导致环境正的奖赏，那么智能体以后产生这个行为策略的趋势便会加强。强化学习是最接近于自然界动物学习的本质的一种学习范式。然而强化学习从提出到现在，也差不多有半个世纪左右，它的应用场景仍很有限，规模大一点的问题就会出现维数爆炸，难于计算，所以往往看到的例子都是相对简化的场景。

最近因为与深度学习结合，解决海量数据的泛化问题，取得了让人印象深刻的成果。包括 DeepMind 的自动学习玩 ATARI 游戏，以及 AlphaGo 在围棋大赛中战胜世界冠军等，其背后的强大武器就是深度强化学习技术。相对于 DeepMind 和学术界看重强化学习的前沿研究，阿里巴巴则将重点放在推动强化学习技术输出及商业应用。在阿里移动电商平台中，人机交互的便捷，碎片化使用的普遍性，页面切换的串行化，用户轨迹的可跟踪性等都要求我们的系统能够对变幻莫测的用户行为以及瞬息万变的外部环境进行完整地建模。平台作为信息的载体，需要在与消费者的互动过程中，根据对消费者（环境）的理解，及时调整提供信息（商品、客服机器人的回答、路径选择等）的策略，从而最大化过程累积收益（消费者在平台上的使用体验）。基于监督学习方式的信息提供手段，缺少有效的探索能力，系统倾向于给消费者推送曾经发生过行为的信息单元（商品、店铺或问题答案）。而强化学习作为一种有效的基于用户与系统交互过程建模和最大化过程累积收益的学习方法，在一些阿里具体的业务场景中进行了很好的实践并得到大规模应用。

在搜索场景中，阿里巴巴对用户的浏览购买行为进行 MDP 建模，在搜索实时学习和实时决策计算体系之上，实现了基于强化学习的排序策略决策模型，

从而使得淘宝搜索的智能化进化至新的高度。双 11 桶测试效果表明，算法指标取得了近 20% 的大幅提升。

在推荐场景中，阿里巴巴使用了深度强化学习与自适应在线学习，通过持续机器学习和模型优化建立决策引擎，对海量用户行为以及百亿级商品特征进行实时分析，帮助每一个用户迅速发现宝贝，提高人和商品的配对效率，算法效果指标提升了 10% 20%。

在智能客服中，如阿里小蜜这类的客服机器人，作为投放引擎的 agent，需要有决策能力。这个决策不是基于单一节点的直接收益来确定，而是一个较为长期的人机交互的过程，把消费者与平台的互动看成是一个马尔可夫决策过程，运用强化学习框架，建立一个消费者与系统互动的回路系统，而系统的决策是建立在最大化过程收益上，来达到一个系统与用户的动态平衡。

在广告系统中，如果广告主能够根据每一条流量的价值进行单独出价，广告主便可以在各自的高价值流量上提高出价，而在普通流量上降低出价，如此容易获得较好的 ROI，与此同时平台也能够提升广告与访客间的匹配效率。阿里巴巴实现了基于强化学习的智能调价技术，对于来到广告位的每一个访客，根据他们的当前状态去决定如何操作调价，给他们展现特定的广告，引导他们的状态向我们希望的方向上做一步转移，在双 11 实测表明，CTR，RPM 和 GMV 均得到了大幅提升。

当然，强化学习在阿里巴巴内部的实践远不止此，鉴于篇幅限制，这本电子书只介绍了其中的一部分。未来深度强化学习的发展必定是理论探索和应用实践的双链路持续深入。希望这本电子书能抛砖引玉，给工业界和学术界带来一些输入，共同推进深度强化学习的更大发展。

阿里巴巴 研究员 青峰

2018 年 01 月于杭州

目　　录

第一章　基于强化学习的实时搜索排序策略调控

1.1　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做出针对性的排序，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）、把用户看作环境（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) 用户每次请求 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于成交的 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1.1比较直观地展示了的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。本文接下来将对我们具体的方案进行详细介绍。

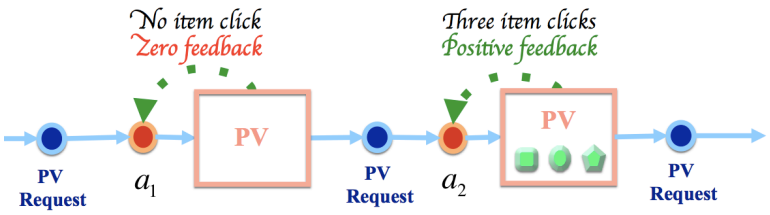


图 1.1: 搜索的序列决策模型

1.2 问题建模

1.2.1 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 [33]。一般地，MDP 可以由一个四元组 <S, A, R, T> 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) $R : S \times A \times S \to \mathbb{R}$ 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) $T : S \times A \times S \to [0, 1]$ 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。

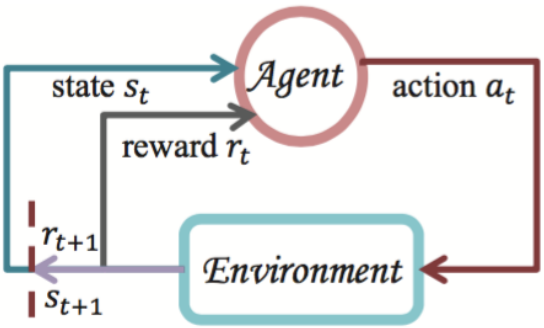


图 1.2: 强化学习 agent 和环境交互

在 MDP 中，Agent 和环境之间的交互过程可如图1.2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 π^*，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

$$\pi^* = argmax_\pi \mathbb{E}_\pi\{\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s\}, \forall s \in S, \forall t \geq 0. \tag{1.1}$$

在这里，$\pi : S \times A \to [0, 1]$ 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 π 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）

$$V^*(s) = \max_\pi \mathbb{E}_\pi\{\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s\}, \forall s \in S, \forall t \geq 0 \tag{1.2}$$

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

$$Q^*(s, a) = \max_\pi \mathbb{E}_\pi\{\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s, a_t = a\}, \forall s \in S, \forall a \in A, \forall t \geq 0 \tag{1.3}$$

来学习最优策略 π^*。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）方法对最优值函数进行逼近，从而得到最优策略 π^*。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 θ 下，任意状态 s 的值 $V(s)$ 可以表达为

$$V_\theta(s) = f_\theta(\phi(s)). \tag{1.4}$$

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 θ 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。

1.2.2 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商品，并且较少点击他不感兴趣的商品。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

$$s = (price_1, cvr_1, sale_1, ..., price_n, cvr_n, sale_n). \tag{1.5}$$

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

$$s = (price_1, cvr_1, sale_1, ..., price_n, cvr_n, sale_n, power, item, shop). \tag{1.6}$$

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。

1.2.3　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。

1.3　算法设计

1.3.1　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 $\mu = (\mu_1, \mu_2, ..., \mu_m)$ 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 μ 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分为输出。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

$$\mu_\theta(s) = (\mu_\theta^1(s), \mu_\theta^2(s), ..., \mu_\theta^m(s)). \tag{1.7}$$

其中，$\theta = (\theta_1, \theta_2, \ldots, \theta_m)$ 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

$$\mu_\theta^i = \frac{C_i \exp(\theta_i^\top \phi(s))}{\sum_{j=1}^m \exp(\theta_j^\top \phi(s))}. \tag{1.8}$$

在这里，$\phi(s)$ 为状态 s 的特征向量，$\theta_1, \theta_2, ..., \theta_m$ 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）。

1.3.2 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（参考对 V^* 和 Q^* 的定义）。实际上，我们可以用一个更一般的形式来表达这一目标，

即

$$\begin{aligned} J(\mu_\theta) &= \int_S \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_0(s') T(s', \mu_\theta(s'), s) R(s, \mu_\theta(s)) \, \mathrm{d}s' \, \mathrm{d}s \\ &= \int_S \rho^\mu(s) R(s, \mu_\theta) \, \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^\mu}[R(s, \mu_\theta(s))]. \end{aligned} \tag{1.9}$$

其中，$\rho^\mu(s) = \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_0(s') T(s', \mu_\theta(s'), s) \, \mathrm{d}s'$ 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 μ_θ 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 θ 的梯度，让 θ 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 θ 的梯度为

$$\begin{aligned} \nabla_\theta J(\mu_\theta) &= \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|a = \mu_\theta(s) \, \mathrm{d}s \\ &= \mathbb{E}_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|a = \mu_\theta(s)]. \end{aligned} \tag{1.10}$$

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

$$\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|a = \mu_\theta(s). \tag{1.11}$$

在这个公式中，α_θ 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：

$$Q^\mu(s, a) \approx Q^w(s, a) = \phi(s, a)^\top w. \tag{1.12}$$

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

$$\nabla_a Q^\mu(s,a) \approx \nabla_a Q^w(s,a) = \nabla_a (a^\top \nabla_\theta \mu_\theta(s))^\top w = \nabla_\theta \mu_\theta(s)^\top w. \tag{1.13}$$

因此，策略函数的参数向量 θ 的更新公式可以写为

$$\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s) (\nabla_\theta \mu_\theta(s)^\top w). \tag{1.14}$$

1.3.3 值函数的学习

在更新策略函数 μ_θ 的参数向量 θ 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：

$$\begin{aligned} \delta_{t+1} &= r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \\ &= r_t + w_t^\top (\gamma \phi(s_{t+1}, \mu_\theta(s_{t+1}) - \phi(s_t, a_t)) \\ w_{t+1} &= w_t + \alpha_w \delta_{t+1} \phi(s_t, a_t) \\ &= w_t + \alpha_w \delta_{t+1} (a_t^\top \nabla_\theta \mu_\theta(s_t)). \end{aligned} \tag{1.15}$$

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，δ_{t+1} 被称作差分误差（Temporal-Difference Error），α_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中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

$$Q(s,a) = A^w(s,a) + V^v(s) = (a - \mu_\theta(s))^\top \nabla_\theta \mu_\theta(s)^\top w + \phi(s)^\top v. \tag{1.16}$$

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 θ、w 和 v 的更新方式总结如下：

$$\begin{aligned}
\delta_{t+1} &= r_t + \gamma Q(s_{t+1}, \mu_\theta(s_{t+1})) - Q(s_t, a_t) \\
&= r_t + \gamma \phi(s_{t+1})^\top v_t - ((a_t - \mu_\theta(s_t))^\top \nabla_\theta \mu_\theta(s_t)^\top w_t + \phi(s_t)^\top v_t) \\
\theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t)(\nabla_\theta \mu_\theta(s_t)^\top w_t) \\
w_{t+1} &= w_t + \alpha_w \delta_{t+1} \phi(s_t, a_t) = w_t + \alpha_w \delta_{t+1}(a_t^\top \nabla_\theta \mu_\theta(s_t)) \\
v_{t+1} &= v_t + \alpha_v \delta_{t+1} \phi(s_t)
\end{aligned} \tag{1.17}$$

1.4　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了验证。具体地，我们采用了上文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，而这是在实际中我们必须要考虑的问题。

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，我们也可以认为的对 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

$$R(s,a,s') = R_0(s,a,s') + \Phi(s). \tag{1.18}$$

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

$$\Phi(s) = \sum_{i=1}^{K} \mathbb{ML}(i|\mu_\theta(s)). \tag{1.19}$$

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，$\mathbb{ML}(i|\mu_\theta(s))$ 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，$\Phi(s)$ 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 $\mathbb{ML}(i|\mu_\theta(s))$ 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 $x_i = (x_i^1, x_i^2, ..., x_i^m)$，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

$$\mathbb{ML} = {p_i}^{y_i}(1-p_i)^{1-y_i} = (\frac{1}{1+\exp(-x_i^\top \mu_\theta(s))})^{y_i}(\frac{1}{1+\exp(x_i^\top \mu_\theta(s))})^{1-y_i}. \tag{1.20}$$

为简化计算，我们对 $\mathbb{L}$ 取对数，得到对数似然概率

$$\mathbb{ML}_{log}(i|\mu_\theta(s)) = y_i x_i^\top \mu_\theta(s) - \ln(1+\exp(x_i^\top \mu_\theta(s))). \tag{1.21}$$

将 PV 中所有商品的对数似然概率综合起来，则有

$$\Phi(s) = \sum_{i=1}^{K} y_i x_i^\top \mu_\theta(s) - \ln(1+\exp(x_i^\top \mu_\theta(s))). \tag{1.22}$$

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

$$\Phi_{clk}(s) = \sum_{i=1}^{K} y_i^c x_i^\top \mu_\theta(s) - \ln(1 + \exp(x_i^\top \mu_\theta(s))). \tag{1.23}$$

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

$$\Phi_{pay}(s) = \sum_{i=1}^{K} y_i^p x_i^\top \mu_\theta(s) - \ln(1 + \exp(x_i^\top \mu_\theta(s))) + \ln\ \text{Price}_i. \tag{1.24}$$

其中，y_i^p 和 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。

1.5　实验效果

在双 11 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。

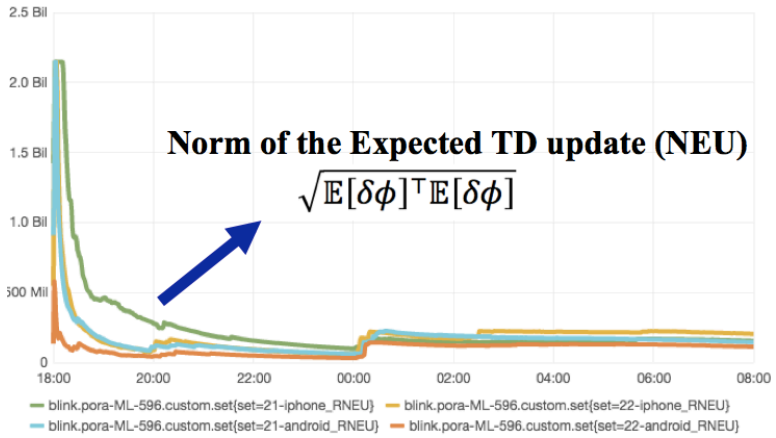


图 1.3: Norm of the Expected TD Update

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 11 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。

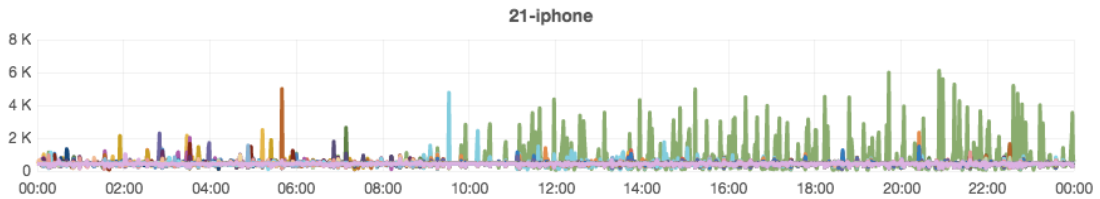


图 1.4: 每个维度的排序权重分在一天内的变化（iPhone）

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

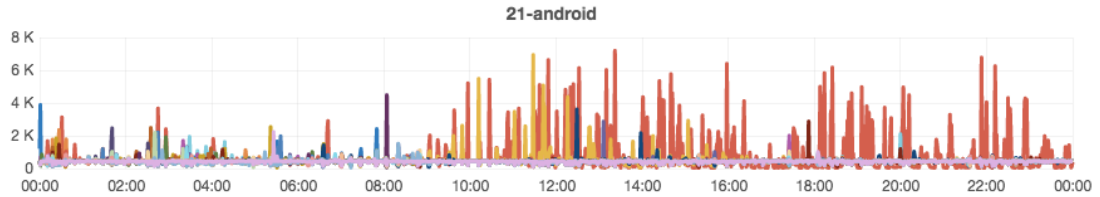


图 1.5: 每个维度的排序权重分在一天内的变化（Android）

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 方面的提升高于其他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。

1.6　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 $X=[x_1,x_2,...,x_n]$ 是 n 个宝贝的归一化的特征分向量，$Y=[y_1,y_2,...,y_n]$ 是对应的点击、成交的 label。具体地：

$$L(a,X,Y)=\frac{1}{n}\sum_{i}^{n}y_i\log(\sigma(a^Tx_i))+(1-y_i)\log(1-\sigma(a^Tx_i)) \tag{1.25}$$

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

$$\nabla_{\theta^\mu} J = -\frac{1}{N}\sum_i^N \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} + \lambda \nabla_a L(a,X,Y)|_{a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \tag{1.26}$$

大致的整体框架如图1.6所示。

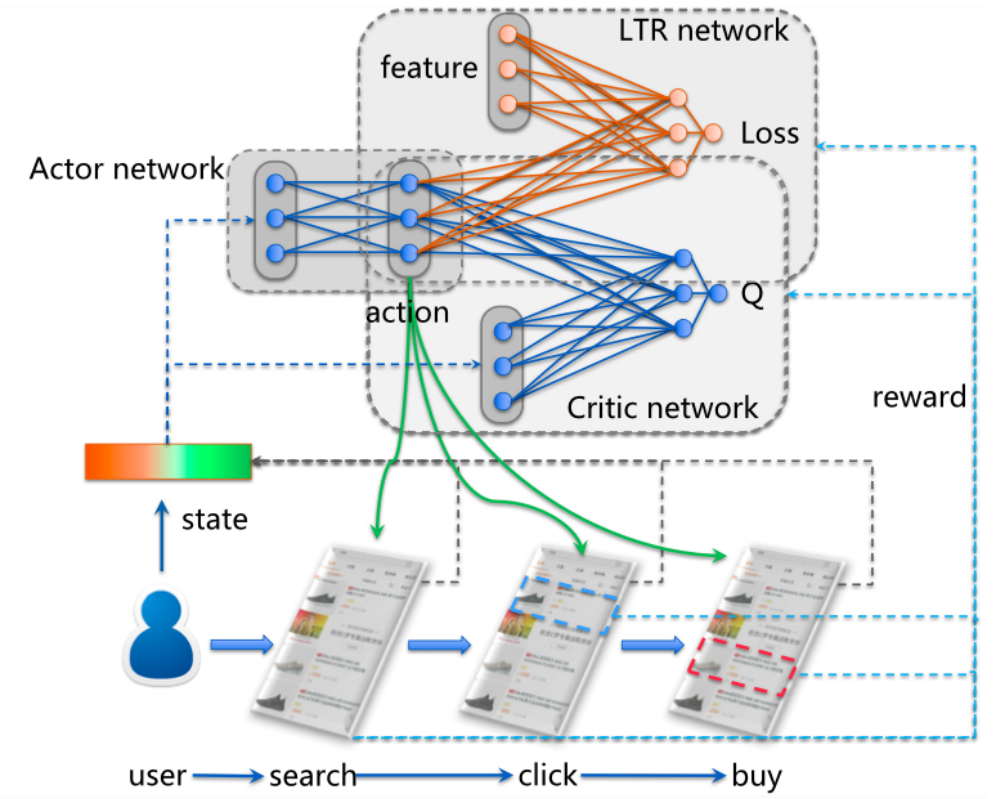


图 1.6: 监督学习和强化学习的多任务学习网络

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。

1.7　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（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 θ，使得 Agent 从环境获得的奖赏信号在不同的 θ 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。因此我们可以将 $J(\mu_\theta)$ 重写为

$$\begin{aligned}\bar{J}(\mu_\theta) &= \int_S \int_S \sum_{t=1}^{\infty} \gamma^{t-1} p_0(s') T(s', \mu_\theta(s'), s) R_\theta(s, \mu_\theta(s)) \, \mathrm{d}s' \, \mathrm{d}s \\ &= \int_S \rho^\mu(s) R_\theta(s, \mu_\theta) \, \mathrm{d}s.\end{aligned} \tag{1.27}$$

其中，R_θ 为 Agent 的策略参数 θ 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。

第二章　延迟奖赏在搜索排序场景中的作用分析

2.1　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。

2.2 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2.1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（$a_1, a_2, a_3, ...$）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2.1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2.2进行描述。在图2.2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

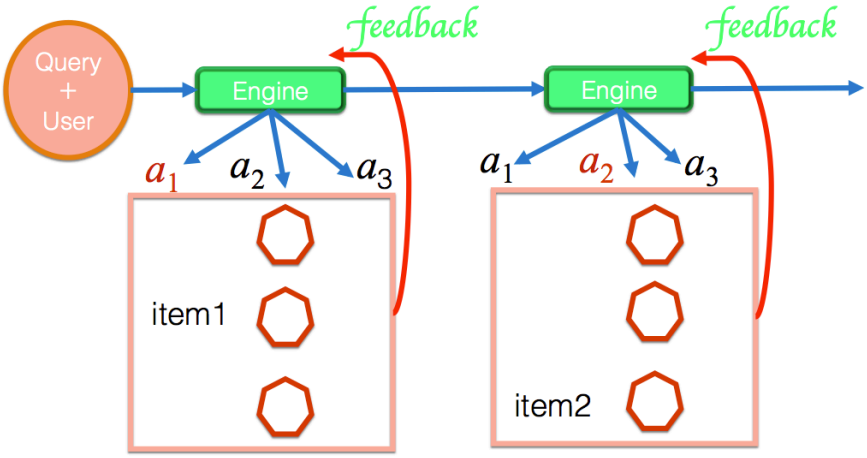


图 2.1: 搜索引擎与用户交互示意图

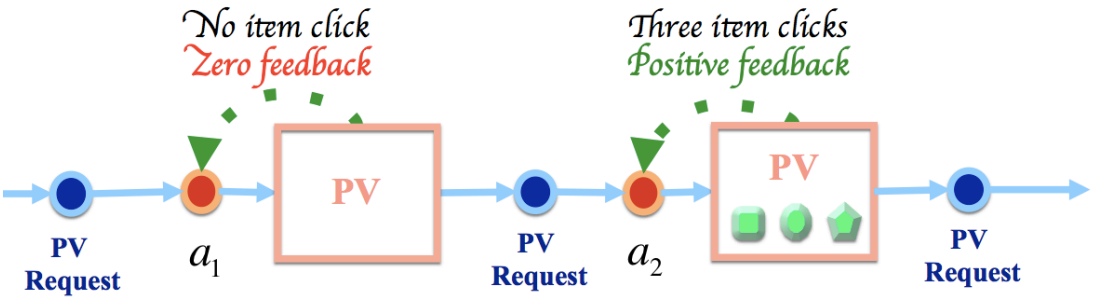


图 2.2: 搜索引擎 Agent 决策过程示意图

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。

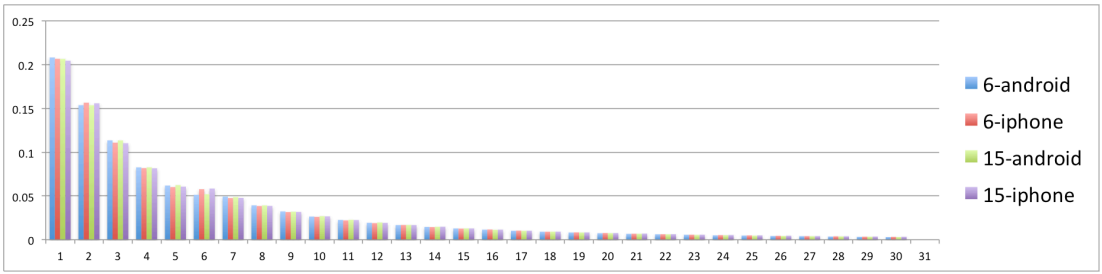


图 2.3: 全类目数据下所有成交 Episode 的长度分布情况

2.3 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 $n = 1, 2, ..., 30$ 的 Episode 占总体成交 Episode 的比例，结果如图2.3所示。

从图2.3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2.4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2.4、2.5、2.6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

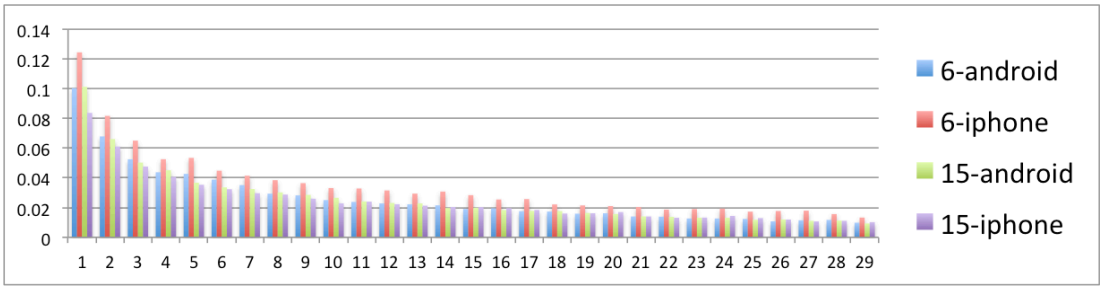


图 2.4: 类目 A 下所有成交 Episode 的长度分布情况

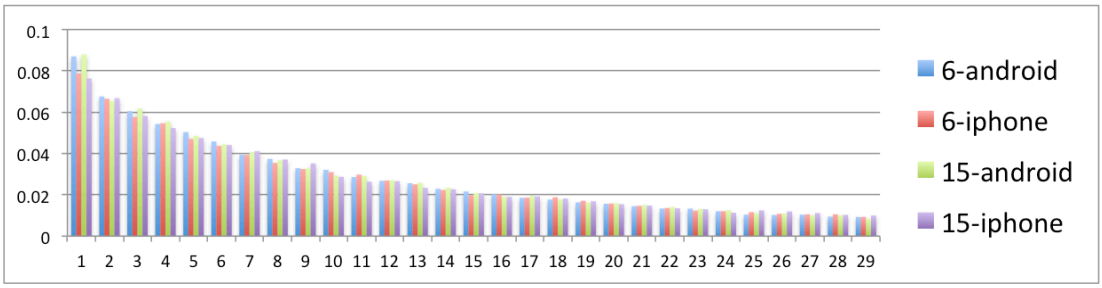


图 2.5: 类目 B 下所有成交 Episode 的长度分布情况

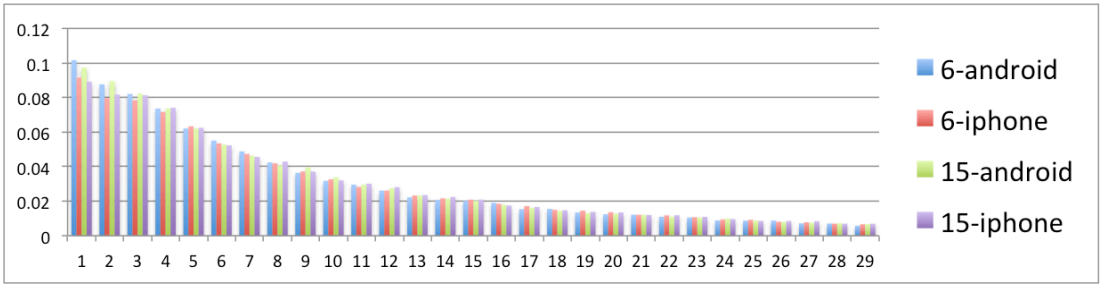


图 2.6: 类目 C 下所有成交 Episode 的长度分布情况

以将一次搜索会话过程用图2.7的抽象示意图来描述。如图所示，垂直方向的箭

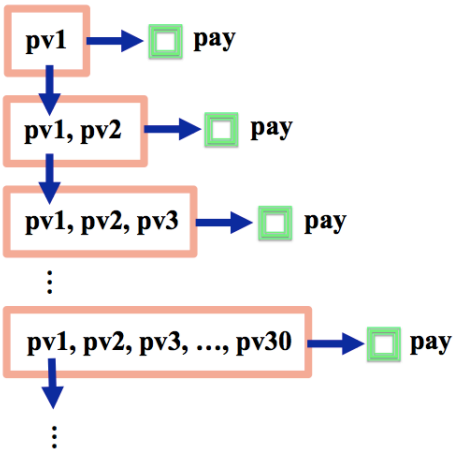


图 2.7: 仅考虑成交和翻页的搜索会话图示

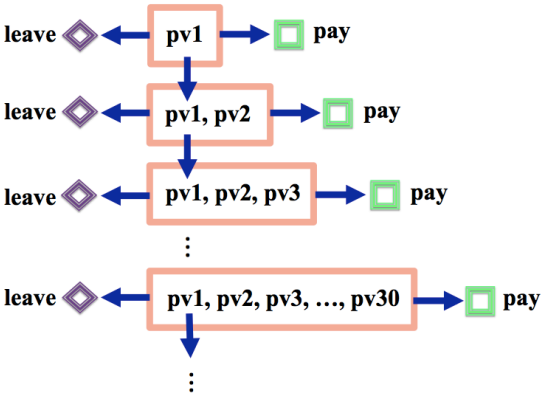


图 2.8: 考虑成交、翻页和用户离开的搜索会话图示

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2.8中的更一般的示意图。

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。

2.4　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 $(\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_K)$。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 $(0 \leq t \leq T)$

- $\mathcal{S} = \mathcal{H}_C \bigcup \mathcal{H}_B \bigcup \mathcal{H}_L$ 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

- $\mathcal{R} : \mathcal{H}_C \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ 为奖赏函数

- $\mathcal{P} : \mathcal{H}_C \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ 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

$$\mathcal{P}(C(h_t), a, s') = \begin{cases} b(h_{t+1}) & \text{if } s' = B(h_{t+1}), \\ l(h_{t+1}) & \text{if } s' = L(h_{t+1}), \\ c(h_{t+1}) & \text{if } s' = C(h_{t+1}), \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

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

$$\mathcal{R}(C(h_t), a, s') = \begin{cases} m(h_{t+1}) & \text{if } s' = B(h_{t+1}), \\ 0 & \text{otherwise}, \end{cases} \tag{2.2}$$

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

2.5 理论分析

2.5.1 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 $s_0, a_0, s_1, a_1, s_2, ..., s_{t-1}, a_{t-1}, s_t$，都有如下等式成立：

$$\Pr(s_t|s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}) = \Pr(s_t|s_{t-1}, a_{t-1}). \tag{2.3}$$

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 $s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}, s_t$，都有等式 $\Pr(s_t|s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}) = \Pr(s_t|s_{t-1}, a_{t-1})$ 成立即可。

除了状态 s_t 以外，序列 $s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}, s_t$ 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 $C(h_0), a_0, C(h_1), a_1, ..., C(h_{t-1}), a_{t-1}, s_t$。需注意的是，对于任意时间步 t' $(0 < t' < t)$，都有

$$h_{t'} = (h_{t'-1}, \mathcal{L}_K(\mathcal{D}_{t'-1}, a_{t'-1})), \tag{2.4}$$

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：

$$\begin{aligned}
&\Pr(s_t|s_0, a_0, s_1, a_1, ..., s_{t-1}, a_{t-1}) \\
=&\Pr(s_t|C(h_0), a_0, C(h_1), a_1, ..., C(h_{t-1}), a_{t-1}) \\
=&\Pr(s_t|h_1, h_2, ..., h_{t-1}, C(h_{t-1}), a_{t-1}) \\
=&\Pr(s_t|h_{t-1}, C(h_{t-1}), a_{t-1}) \\
=&\Pr(s_t|C(h_{t-1}), a_{t-1}) \\
=&\Pr(s_t|s_{t-1}, a_{t-1}).
\end{aligned} \tag{2.5}$$

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。

2.5.2　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 γ 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 $\pi : \mathcal{S} \to \mathcal{A}$，记每个时间步 $t (0 \le t \le T)$ 对应的 item page history 为 π by h_t^π，我们把在策略 π 下能够访问的所有状态都展示在图2.9中。

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^π、c_t^π、b_t^π 和 m_t^π。

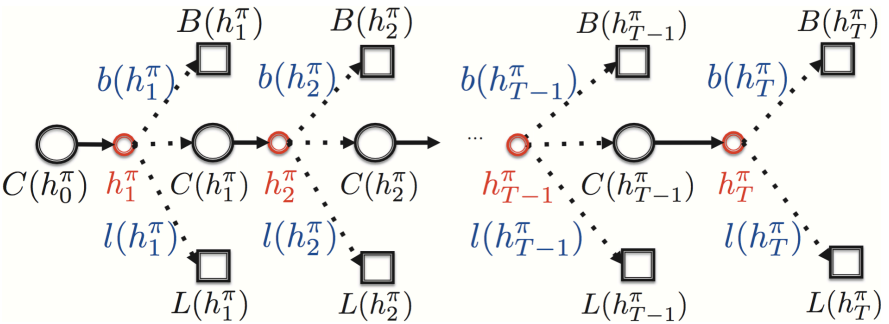


图 2.9: Agent 策略 π 下能够访问 SSMDP 中的所有状态

不失一般性，我们设 SSMDP $\mathcal{M}$ 的折扣率为 γ $(0 \leq \gamma \leq 1)$。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^π 的 state value 为

$$\begin{aligned} V_\gamma^\pi(C_t^\pi) &= \mathbb{E}^\pi\Big\{\sum_{k=1}^{T-t}\gamma^{k-1}r_{t+k}\Big|C_t^\pi\Big\} \\ &= \mathbb{E}^\pi\big\{r_{t+1}+\gamma r_{t+2}+\cdots+\gamma^{T-t-1}r_T\big|C_t^\pi\big\}. \end{aligned} \tag{2.6}$$

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}^π 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 π 下的期望值为 $\mathbb{E}^\pi\big\{r_{t+k}\big\} = b_{t+k}^\pi m_{t+k}^\pi$。在这里，$m_{t+k}^\pi = m(h_{t+k}^\pi)$ 为 item page history h_{t+k}^π 的成交价格期望。由于 $V_\gamma^\pi(C_t^\pi)$ 表达的是在 C_t^π 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^π 到达 item page history h_{t+k}^π 的概率考虑进来。记从状态 C_t^π 到达 h_{t+k}^π 的概率为 $\Pr(C_t^\pi \to h_{t+k}^\pi)$，根据状态转移函数的定义可得

$$\Pr(C_t^\pi \to h_{t+k}^\pi) = \begin{cases} 1.0 & k = 1, \\ \Pi_{j=1}^{k-1} c_{t+j}^\pi & 1 < k \leq T-t. \end{cases} \tag{2.7}$$

从状态 C_t^π 到 item page history h_{t+1}^π 的概率为 1 是因为 h_{t+1}^π 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：

$$\begin{aligned} V_\gamma^\pi(C_t^\pi) &= \mathbb{E}^\pi\{r_{t+1}|C_t^\pi\} + \gamma\mathbb{E}^\pi\{r_{t+2}|C_t^\pi\} + \cdots + \gamma^{T-t-1}\mathbb{E}^\pi\{r_T|C_t^\pi\} \\ &= \sum_{k=1}^{T-t}\gamma^{k-1}\Pr(C_t^\pi \to h_{t+k}^\pi)b_{t+k}^\pi m_{t+k}^\pi \\ &= b_{t+1}^\pi m_{t+1}^\pi + \gamma c_{t+1}^\pi b_{t+2}^\pi m_{t+2}^\pi + \cdots + \gamma^{T-t-1}\big(\Pi_{j=1}^{T-t-1}c_{t+j}^\pi\big)b_T^\pi m_T^\pi \\ &= b_{t+1}^\pi m_{t+1}^\pi + \sum_{k=2}^{T-t}\gamma^{k-1}\Big(\big(\Pi_{j=1}^{k-1}c_{t+j}^\pi\big)b_{t+k}^\pi m_{t+k}^\pi\Big). \end{aligned} \tag{2.8}$$

根据图2.9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 π 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

$$\begin{aligned} \mathbb{E}_{gmv}^\pi &= b_1^\pi m_1^\pi + c_1^\pi b_2^\pi m_2^\pi + \cdots + \big(\Pi_{k=1}^{T}c_k^\pi\big)b_T^\pi m_T^\pi \\ &= b_1^\pi m_1^\pi + \sum_{k=2}^{T}\big(\Pi_{j=1}^{k-1}c_j^\pi\big)b_k^\pi m_k^\pi. \end{aligned} \tag{2.9}$$

通过比较 $\mathbb{E}_{gmv}$ 和 V_γ^π，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_γ^π 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 $\pi : \mathcal{S} \to \mathcal{A}$ 和折扣率 γ ($0 \le \gamma \le 1$)，都有式子 $V_\gamma^\pi(C(h_0)) \le \mathbb{E}_{gmv}^\pi$ 成立，其中 V_γ^π 为 agent 在策略 π 和折扣率 γ 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 π 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 $\sum_{k=2}^{T}(1-\gamma^{k-1})\big(\Pi_{j=1}^{k-1}c_j^\pi\big)b_k^\pi m_k^\pi$ 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）做出的反应，这天然决定了搜索排序问题的 sequential decision-making 本质。

2.6 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 Dynammic Programming 方法，在折扣率 γ 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2.10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。

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2.11所示。可以看到，Actor-Critic 方法在不同折扣率下的 GMV 指标与图2.10的结果一致，而图右边的值函

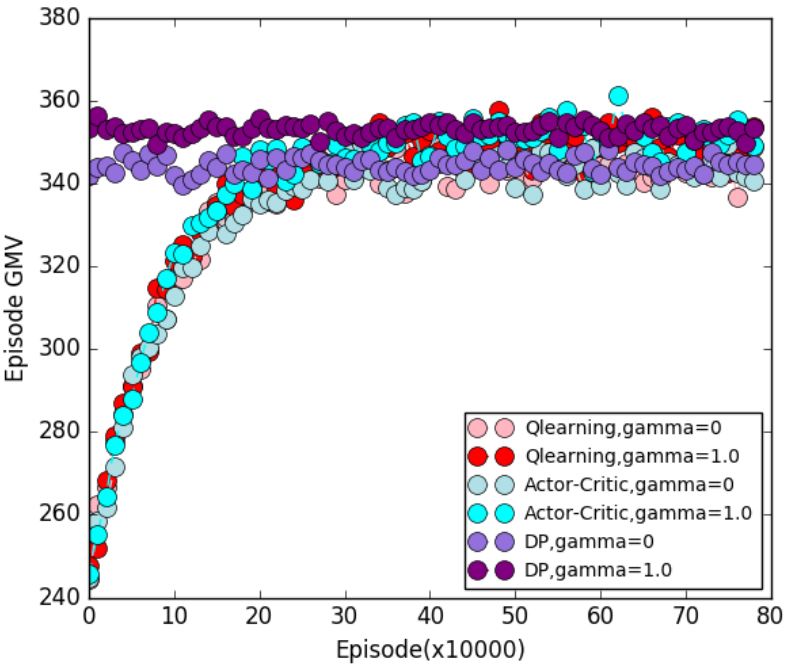


图 2.10:　$\gamma = 0$ 和 $\gamma = 1$ 情况下测试算法的 GMV 曲线

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 γ 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。

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转化率之类的目标，结论可能会不同，但都可以采用与本文类似的方法进行分析。

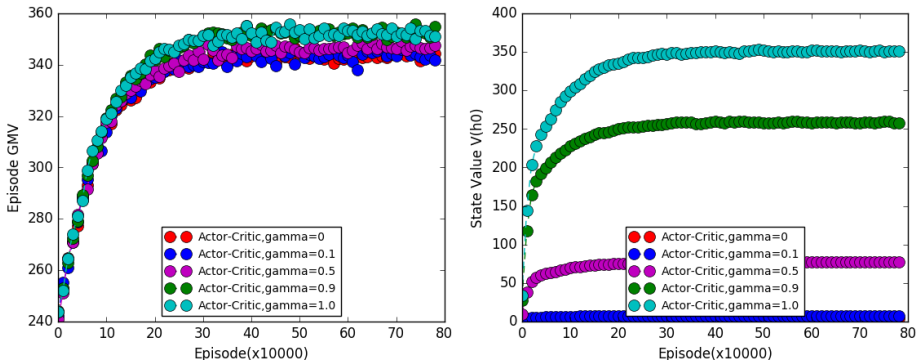


图 2.11: Actor-Critic 方法在各个折扣率设置下的 GMV 曲线和值函数曲线

第三章　基于多智能体强化学习的多场景联合优化

3.1　背景

淘宝平台下有非常多的子场景，例如搜索、推荐、广告；每个子场景又有非常多细分，例如搜索包括默认排序、店铺内搜索、店铺搜索等；推荐内有猜你喜欢、今日推荐、每日好店等。基于数据驱动的机器学习和优化技术目前大量的应用于这些场景中，并已经取得的不错的效果——在单场景内的 A/B 测试上，点击率、转化率、成交额、单价都能看到显著提升。

然而，目前各个场景之间目前是完全独立优化的，这样会带来几点比较严重的问题：

a. 用户在淘宝上购物会经常在多个场景之间切换，例如：从主搜索到猜你喜欢，从猜你喜欢到店铺内。不同场景的商品排序仅考虑自身，会导致用户的购物体验是不连贯或者雷同的。例如：从冰箱的详情页进入店铺，却展示手机；各个场景都展现趋同，都包含太多的 U2I（点击或成交过的商品）。

b. 多场景之间是博弈（竞争）关系，期望每个场景的提升带来整体提升这一点是无法保证的。很有可能一个场景的提升会导致其他场景的下降，更可怕的是某个场景带来的提升甚至小于其他场景更大的下降。这并非是不可能的，那么这种情况下，单场景的 A/B 测试就显得没那么有意义，单场景的优化也会存在明显的问题。因为这一点尤为重要，因此我们举一个更简单易懂的例

子（如图3.1）。一个 1000 米长的沙滩上有 2 个饮料摊 A 和 B，沙滩上均分分布者很多游客，他们一般会找更近的饮料摊去买饮料。最开始 A 和 B 分别在沙滩 250 米和 750 米的位置，此时沙滩左边的人会去 A 买，右边的人去 B 买。然后 A 发现，自己往右边移动的时候，会有更多的用户（A/B 测试的结论），因此 A 会右移，同样 B 会左移。A 和 B 各自“优化”下去，最后会都在沙滩中间的位置，从博弈论的角度，到了一个均衡点。然而，最后“优化”得到的位置是不如初始位置的，因为会有很多游客会因为太远而放弃买饮料。这种情况下，2 个饮料摊各自优化的结果反而是不如不优化的。

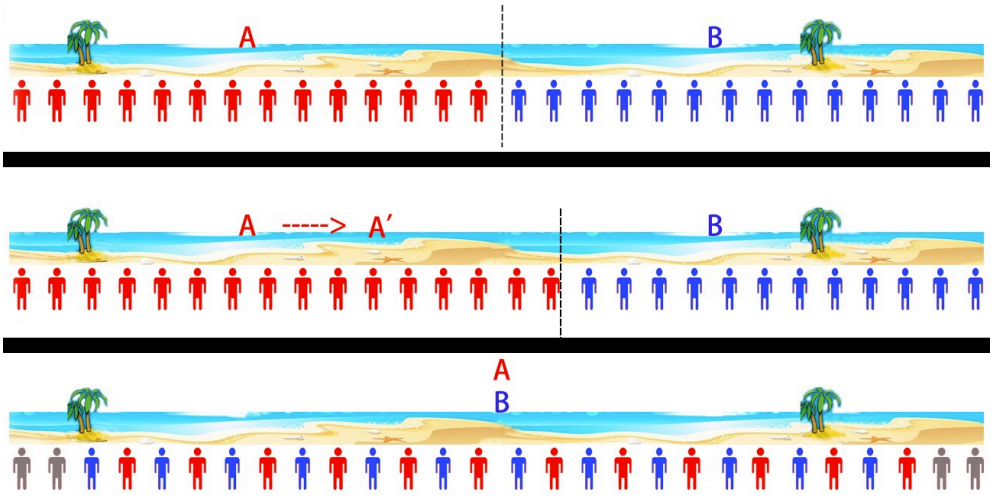


图 3.1: 一片长沙滩上 2 个零食店关于选择开店位置的博弈过程。上图表示初始位置，中间图表示博弈过程，下方图表示均衡状态。红色的用户表示愿意在 A 购买，蓝色的用户表示愿意在 B 购买，灰色用户表示太远而偏向于放弃购买

多场景问题实际并不止存在于淘宝上，实际上目前比较大型的平台或者无线 APP 都不止一个场景。即使不谈 Yahoo，Sina 等综合性网站，像 Baidu、Google 等功能比较单一、集中的应用，也会有若干场景（如网页、咨询、地图等）。那么这些平台或应用都会面临类似的问题。

综上，研究大型在线平台上的多子场景联合优化，无论从淘宝平台的应用上，还是从科研的角度，都具有重要意义。

为了解决上述的问题，本文提出一个多场景联合排序算法，旨在提升整体指标。我们将多场景的排序问题看成一个完全合作的、部分可观测的多智能体序列决策问题，利用 Multi-Agent Reinforcement Learning 的方法来尝试着对问题进行建模。该模型以各个场景为 Agent，让各个场景不同的排序策略共享同一个目标，同时在一个场景的排序结果会考虑该用户在其他场景的行为和反馈。这样使得各个场景的排序策略由独立转变为合作与共赢。由于我们想要使用用户在所有场景的行为，而 DRQN 中的 RNN 网络可以记住历史信息，同时利用 DPG 对连续状态与连续动作空间进行探索，因此我们算法取名 MA-RDPG(Multi-Agent Recurrent Deterministic Policy Gradient)。

本章内容，带来的核心进步主要有以下三点:

- 我们将多场景联合优化（排序）的问题，建模成一个完全合作、部分可观测的多智能体序列决策问题。
- 我们提出了一个全新的、基础的多智能体强化学习模型，叫作 Multi-Agent Recurrent Deterministic Policy Gradient。该模型能使多个智能体（多个场景）合作以取得全局的最佳效果。
- 我们将算法应用在了淘宝的线上环境，测试表明我们的模型能显著提升淘宝的整体指标。

3.2 问题建模

3.2.1 相关背景简介

排序学习（Learning to rank, L2R [25]）被广泛应用于在线的排序系统中。排序学习最基本的思想是最优的排序策略是能通过大量的训练样本被学习得到。每条训练样本包含一个搜索词（query）以及改搜索词下展示的商品序列，排序策略将每个商品的多个特征映射成一个排序分值。排序策略的参数能够通过多种不用的方式学习得到，例如基于 point-wise [9, 21] 的优化，基于 pair-wise [2, 29] 的优化和基于 list-wise methods [3, 5] 的优化算法。

深度递归 Q 网络：在实际的应用场景中，环境中的状态可能是只能被部分观测的。因此强化学习中的智能体不能观测到全部的状态，这种设定被称为“部分可观测”。深度递归 Q 网络（Deep Recurrent Q-Networks, DRQN）被提出来，通过递归编码之前的观测，解决这种部分观测的问题。DRQN 在当前行为之前，先使用递归神经网络编码之前的状态，取代 Q 网络中消除“状态-行为函数”$Q(s_t, a_t)$，它会估计 $Q(h_{t-1}, o_t, a_t)$，其中 h_{t-1} 表示 RNN 中的隐状态，通过之前的观测 $o_1, o_2, \ldots, o_{t-1}$ 综合而来。递归神经网络必须使用这个函数去更新它的隐状态 $h_t = g(h_{t-1}, o_t)$，其中 g 是一个非线性函数。

在**多智能体强化学习（Multi-Agent Reinforcement Learning，MARL）** [4, 24, 13, 28] 中，存在一组自主的、可交互的智能体，他们处于一个相同的环境。每个智能体会观测到我们各自的信息，然后根据自身的策略函数，决定一个当前行为。不同智能体智能通过完全合作、完全竞争或者混合方式相处。完全合作下，所有智能体共享一个相同目标；完全竞争下，不同智能体的目标是相反的；混合模型则介于前两者之间。

3.2.2 建模方法

问题描述

我们将任务设计成一个完全合作、部分可观测、智能体序列决策问题。更具体的：

多智能体，在系统中，每一个子场景都拥有自己的排序策略。每个智能体产出一个排序策略，同时学习自己的策略函数，该函数将自己的状态映射到一个行为上。

序列决策，用户时序的与系统进行交互，因此智能体的行为也是序列性的。在每一个时间点上，智能体通过返回一个商品序列给用户，完成一次用户与场景的交互。当前的决策会对未来接下来的决策产生影响。

完全合作，所有的智能体共同优化一个相同的目标值。更进一步，每个智能体会发送消息给其他智能体来进行通信，整体的收益、目标通过一个综合的裁判来评判。

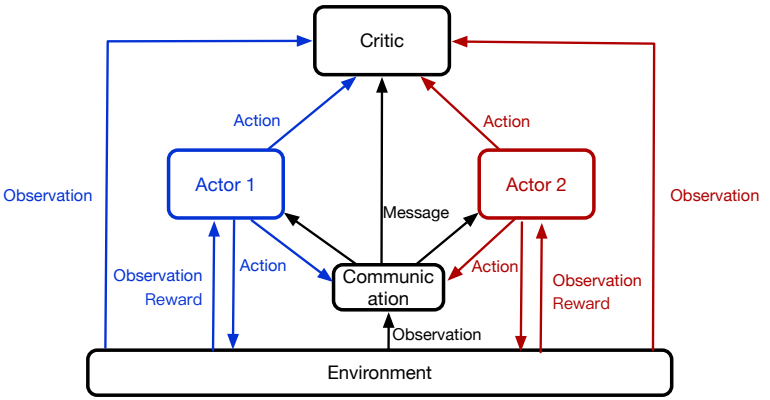


图 3.2: 模型的整体结构。模型有一个综合的、全局的“裁判”来评价整体的收益。一个通信模块用来生成消息，消息可以被多个智能体共享。每条消息编码了一个智能体的历史观测和行为，被用于逼近全局的环境状态。每个智能体网络接收它独自的观测以及受到的信息，同时独立产出一个行为。

部分观测，每个智能体智能观测一部分环境，同时能接受其他智能体发送的信息。

模型

下面我们会详细的介绍我们提出的多智能体递归确定策略梯度法，来解决上述完全合作、部分可观测、智能体序列决策问题。

整体模型。

图3.2展示了我们模型的整体结构。为了简单起见，我们仅考虑 2 个智能体的情况，每个智能体表示一个能自我优化的场景和策略。收到深度策略梯度法 (DDPG [22]) 的启发，我们的模型同样基于 actor-critic 方法 [17]。我们设计了 3 个重要的模块让多智能之间能有协同与合作，分别是一个整体的、全局的“裁判”，独立的智能体，以及通信机制。

全局的裁判会维护一个行为-值函数，该函数表示，在当前状态下，进行一个行为时未来整体收益的期望。每个智能体维护一个行为网络，将状态确定的映射到一个唯一的行为上。每个智能体的决策行为会在它的场景内进行优化。

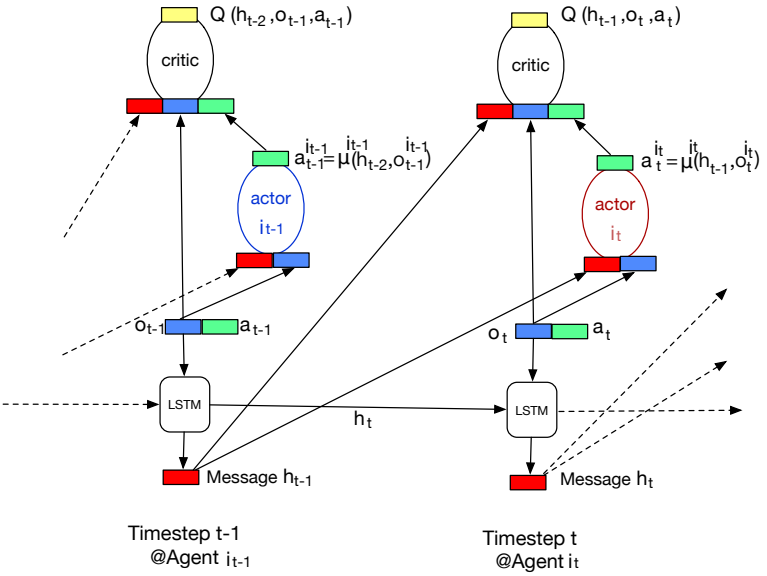


图 3.3: MA-RDPG 算法的详细结构。中心的裁判会模拟“行为-值”函数 $Q(h_{t-1}, o_t, a_t)$，它表示当接受到信息 h_{t-1} 和观测 o_t 时，采取行为 a_t 会获得的整体收益。每个智能体才产生一个确定的行为，基于函数 $a^i_t = \mu^i(h_{t-1}, o^i_t)$。信息会被通信模块更新，基于观测 o_t 和行为 a_t。红色表示信息，蓝色表示观测，绿色表示行为。

我们设计了一种基于 LSTM [12] 的消息机制。LSTM 是一种递归神经网络，他能将所有智能体的全部观测和行为编码成一个消息向量。该消息向量会被发送到不同的智能体，以此形成协作。由于这种机制的存在，每个智能体的决策并不只是基于自己的状态以及之前行为，同时会考虑其他智能体的状态与行为。这种通信能让每个智能体去模拟全局的环境状态，让他们进行更准确的决策。

模型细节。

一个经典的强化学习问题中，会存在一个形如 $(o_1, r_1, a_1, \cdots, a_{t-1}, o_t, r_t)$ 的历史的行为序列，其中 $o/r/a$ 分别表示观测、收益以及行为。正如之前提及的，我们问题中的环境是部分可观测的，这也就是说状态 s_t 代表的是过往的

经验，即 $s_t = f(o_1, r_1, a_1, \cdots, a_{t-1}, o_t, r_t)$[1]我们考虑的是一个 N 个智能体的问题，$\{A^1, A^2, \ldots, A^N\}$，每个智能体对应一个特征的优化场景（例如排序、推荐等）。在这种多智能体的设定下，环境的状态 (s_t) 是全局的，被多个智能体共享；但是观测 $(o_t = (o_t^1, o_t^2, \cdots, o_t^N))$，行为 $(a_t = (a_t^1, a_t^2, \cdots, a_t^N))$，记忆短期收益 $(r_t = (r(s_t, a_t^1), r(s_t, a_t^2), \cdots, r(s_t, a_t^N)))$ 都是独自拥有的。

更具体的来说，每个智能体 A^i 会根据自己的策略 $\mu^i(s_t)$ 和状态 s_t 进行每次决策行为 a_t^i，然后会从环境中得到一个暂时收益 $r_t^i = r(s_t, a_t^i)$，同时状态会从 s_t 更新为 s_{t+1}。在我们的任务中，多个智能体会协同合作，期望达到整体的最大收益。我们有一个全局的“行为-值”函数（critic）$Q(s_t, a_t^1, a_t^2, \cdots, a_t^N)$ 去预估整体的全局收益，在当前状态去采取行为 $(a_t^1, a_t^2, \cdots, a_t^N)$ 时。我们同时又一个全局的状态表示，每个智能体在获得本地观测后，会执行一个本地行为。因此，我们的模型属于一种 actor-critic 强化学习，包含一个中心的 critic 以及多个独立的 actor（每个代表一个智能体）。

如图3.3所示，在时间点 t，智能体 A^{i_t} 从环境中接受当前的观测 $o_t^{i_t}$。环境的全局状态被所有智能体共享，不止依赖于各个智能体的历史状态和行为，同时也依赖当前观测 o_t，换句话说，$s_t = f(o_1, a_1, \cdots, a_{t-1}, o_t)$。为了达到这个目的，我们设计了一个通信模块，该模块使用 LSTM 来编码之前的观测以及行为，编码得到的是一个向量形式的信息。通过智能体之间的通信交流，整体的状态可以近似为 $s_t \approx \{h_{t-1}, o_t\}$，这是因为信息 h_{t-1} 已经包含了所有之前的观测和行为。每个智能体 A^{i_t} 选择一个行为 $a_t^{i_t} = \mu^{i_t}(s_t) \approx \mu^{i_t}(h_{t-1}, o_t^{i_t})$，目标是最大化整体的未来收益，该收益通过一个中心的 critic$Q(s_t, a_t^1, a_t^2, \cdots, a_t^N)$ 来评价。需要注意，在每个时间点，$o_t = (o_t^1, o_t^2, \cdots, o_t^N)$ 是有所有的观测组合得到的。

通信模块。我们设计了一个通信模块，目的是让智能体之间能通过相互传递信息以更好的合作。信息包含了一个智能体本地的历史观测和行为。在时间点 t，智能体 A^{i_t} 接收到一个观测 $o_t^{i_t}$ 和一个来自环境的消息 h_{t-1}。通信模块会根据之前的消息 h_{t-1} 以及当前的观测 o_t 生成一个新的消息 h_t，通过这个消息，该智能体能达成与其他合作智能体的信息共享。如图 3.4，我们使用一个 LSTM

[1]在一个完全可观测的环境中, $s_t = f(o_t)$。

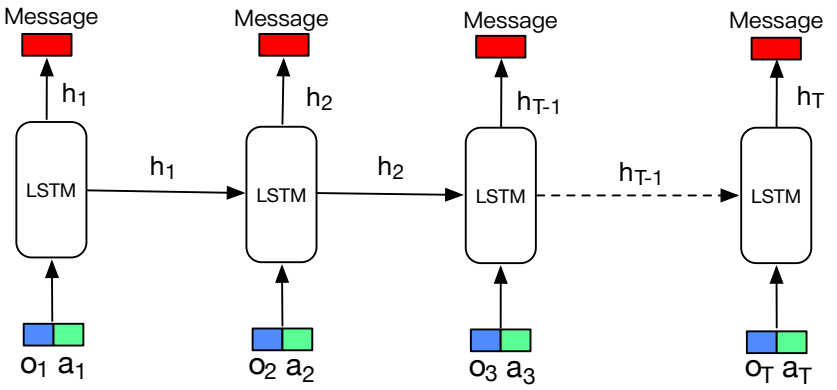


图 3.4: 通信模块。之前的观测 (o_t) 以及行为 (a_t) 都会被作为输入进入 LSTM 网络，隐状态 (h_{t-1}) 被作为消息发送到其他智能体。需要注意，o_t, a_t 是向量。

结构来达到这个目的，更形式化的，通信模块工作模式如下：

$$h_{t-1} = LSTM(h_{t-2}, [o_{t-1}; a_{t-1}]; \psi) \tag{3.1}$$

注意，o_t 和 a_t 由所有智能体的观测和行为组成，同时每个 a_t^i 也是一个实值向量。

受益于信息 h_{t-1}，每个单独的智能体能够去近似得到全局的环境状态，$s_t \approx \{h_{t-1}, o_t\}$。这解决了每个智能体智能接受本地的观测 o_t^i，却不能得到全局的状态 s_t 的问题。

Private Actor。每个智能体（agent）都是一个独立的“演员”（actor），它们会接受本地的观测以及共享的信息，然后最初一个决策。由于我们考虑的是连续行为的强化学习问题，我们定义行为是一个实质向量 $a^i = (w_1^i, \ldots, w_{N^i}^i), a^i \in \mathbb{R}^{N^i}$。因此，每个行为是一个 N^i 维向量，每一个维度都是一个实数值。这个向量会被当作排序的参数控制搜索排序。

由于这是一个连续行为类型，类似的工作常见于控制问题中 [32, 22, 11]。受相关工作的启发，我们使用一个确定策略的方法，而不是随机策略的方式。每个智能体的 actor 对应函数 $\mu^i(s_t; \theta^i)$，其中参数是 θ^i，改函数将一个状态确定的映射到一个行为上。在时刻 t，智能体 A^{i_t} 根据 actor 网络决定自己的行为：

$$a_t^{i_t} = \mu^{i_t}(s_t; \theta^{i_t}) \approx \mu^{i_t}(h_{t-1}, o_t^{i_t}; \theta^{i_t}) \tag{3.2}$$

其中 $s_t \approx \{h_{t-1}, o_t\}$，如之前描述，表示通信模块。因此，actor 的行为同时依赖于信息 h_{t-1} 和自己当前的观测 observation $o_t^{i_t}$.

Centralized Critic。和 DDPG 算法一样，我们设计了一个评价（critic）网络来拟合“行为-值”函数，该网络用来近似未来整体收益的期望。因为所有智能体可以共享一个目标，我们使用一个全局的 critic 函数 $Q(s_t, a_t^1, a_t^2, \cdots, a_t^N; \phi)$ 来拟合未来的整体收益，当全部的智能体在状态 $s_t \approx \{h_{t-1}, o_t\}$ 时采取行为 $a_t = \{a_t^1, \ldots, a_t^N\}$。

以上公式在所有智能体都是活动的状态下，是基础而有效的。在我们的设定下[2]，因此只会有一个智能体 A^{i_t} 在时刻 t 是活跃的，此刻 $o_t = \{o_t^{i_t}\}$，$a_t = \{a_t^{i_t}\}$。于是，我们需要简化“行为-值”函数为 $Q(h_{t-1}, o_t, a_t; \phi)$，并且行为函数为 $\mu^{i_t}(h_{t-1}, o_t; \theta^{i_t})$。

模型训练

centralized critic 网络 $Q(h_{t-1}, o_t, a_t; \phi)$ 我们和 Q-learning 一样 [38] 使用 Bellman 公式训练。我们最小化以下 loss 函数：

$$L(\phi) = \mathbb{E}_{h_{t-1}, o_t}[(Q(h_{t-1}, o_t, a_t; \phi) - y_t)^2] \tag{3.3}$$

其中

$$y_t = r_t + \gamma Q(h_t, o_{t+1}, \mu^{i_{t+1}}(h_t, o_{t+1}); \phi) \tag{3.4}$$

private actor 网络的更新则基于最大化整体的期望。假设在时刻 t，A^{i_t} 是活跃的，那么目标函数是：

$$J(\theta^{i_t}) = \mathbb{E}_{h_{t-1}, o_t}[Q(h_{t-1}, o_t, a; \phi)|_{a=\mu^{i_t}(h_{t-1}, o_t; \theta^{i_t})}] \tag{3.5}$$

根据链式法则，每个 actor 的参数梯度可以表示如下：

$$\begin{aligned} &\nabla_{\theta^{i_t}} J(\theta^{i_t}) \\ &\approx \mathbb{E}_{h_{t-1}.o_t}[\nabla_{\theta^{i_t}} Q^{i_t}(h_{t-1}, o_t, a; \phi)|_{a=\mu^{i_t}(h_{t-1}, o_t; \theta^{i_t})}] \\ &= \mathbb{E}_{h_{t-1}, o_t}[\nabla_a Q^{i_t}(h_{t-1}, o_t, a; \phi)|_{a=\mu^{i_t}(h_{t-1}, o_t)} \nabla_{\theta^{i_t}} \mu^{i_t}(h_{t-1}, o_t; \theta^{i_t})] \end{aligned} \tag{3.6}$$

[2]因为一个用户同一时刻只会存在于一个场景中

通信模块训练的目标是最小化以下函数：

$$
\begin{aligned}
& L(\psi) \\
& = \mathbb{E}_{h_{t-1},o_t}[(Q(h_{t-1},o_t,a_t;\phi) - y_t)^2|_{h_{t-1}=LSTM(h_{t-2},[o_{t-1};a_{t-1}];\psi)}] \\
& - \mathbb{E}_{h_{t-1},o_t}[Q(h_{t-1},o_t,a_t;\phi)|_{h_{t-1}=LSTM(h_{t-2},[o_{t-1};a_{t-1}];\psi)}]
\end{aligned} \tag{3.7}
$$

整体的训练流程见 1。我们使用一个 replay buffer [22] 来存储每个智能体与环境的交互，并使用 minibatch 的方式更新。在每个训练时刻，我们选出一组 minibatch，然后并行的训练他们，同时更新 actor 网络和 critic 网络。

3.3 应用

以上章节描述了一个通用的多智能体强化学习框架，能解决多场景协作优化的问题。这一章节，我们着重描述这个算法在淘宝上的应用。更具体的，我们在 2 个相关的排序场景下的应用。

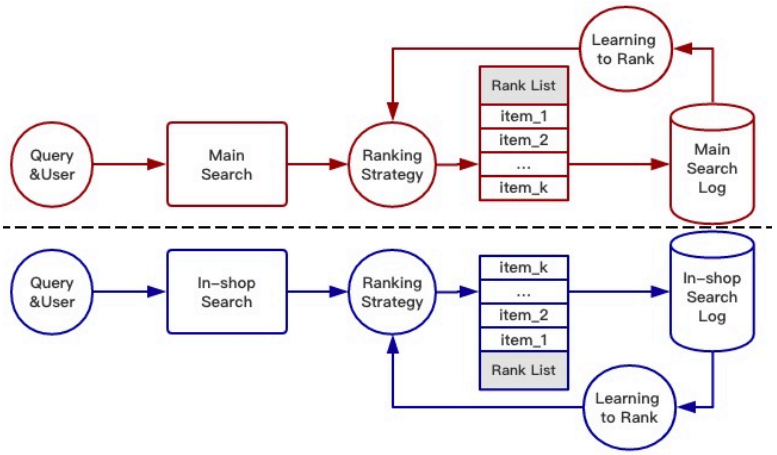
首先我们会给出一个简单的淘宝电商平台整体介绍，然后我们会详细介绍 MA-DRPG 算法在淘宝的应用。

3.3.1 搜索与电商平台

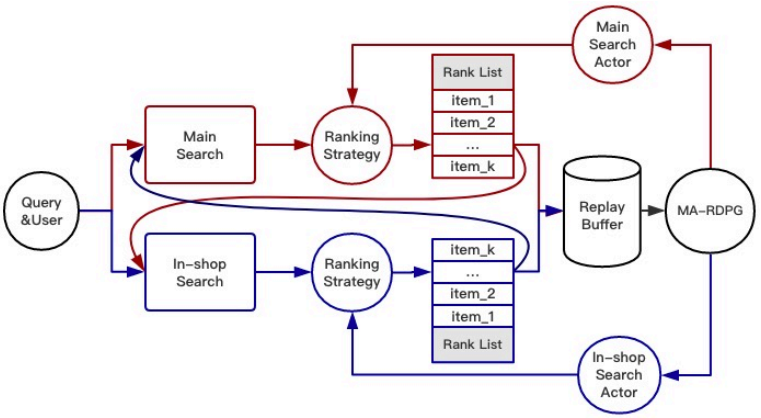
在一个电商平台上，往往都存在多个不同的排序场景，每个场景都会有自己独立的排序策略。特别的，我们选择了淘宝上 2 个最重要的搜索排序场景来应用 MA-DRPG 算法：主搜索和店铺内搜索。下面会简单的独立介绍：

主搜索是淘宝电商平台的最大入口，当用户在淘宝主页面上的搜索框上输入一个请求词时，主搜索会返回相关的商品。主搜索会对淘宝上各个业务线上的商品进行综合排序，每秒会收到大于 40,000 个搜索请求，每天会有 35 亿以上的页面展示次数和 15 亿以上的点击，1 亿以上的用户。

店铺内搜索对一个特定店铺的商品进行排序。这总排序一般发生在一个用户进入到一个特定的店铺的时候，例如“Nike 官方旗舰店”等。在这种排序下，用



(a) 2 个搜索引擎独立优化



(b) 使用 MA-RDPG，2 个搜索引擎联合优化

图 3.5: 独立优化与协同优化的对比.

户可以输入一个关键词，也可以补输入任何关键词。在一天中，会有超过 5000 万用户使用店铺内搜索，并有超过 6 亿的点击和 15 亿的页面展示数。

用户会频繁的在 2 种场景下切换：当一个用户在主搜索上看到一条漂亮的连衣裙时，她会进入对应的店铺看看有没更好看的；当一个用户在店铺内搜索时衣服，也许会觉得商品太少，进而到主搜索去找更多的衣服，从而又进入其

他的店铺。我们的一个简单统计发现，会有 25.46% 的用户会从主搜索进入店铺内搜索，而会有 9.12% 的用户在使用店铺搜索后又使用主搜索。

在已有的模型中 [8, 15, 10]，不同场景中的排序策略会独立的优化，优化的目标也仅仅考虑自身而不管其他的场景。图 Figure 3.5(a) 描述了传统的多场景优化方式。图中红色上部分表示主搜索，下部蓝色部分表示店铺搜索。2 个搜索引擎是独立优化的。

3.3.2 多排序场景协同优化

图 3.5(b) 描述了主搜索和店铺内搜索的联合优化方式。不同于 2 个场景的独立优化，MA-RDPG 对 2 个智能体协同建模，主搜索和店铺内搜索各自学习自身场景的排序策略权重。不同场景之间的协同，通过下面 2 种方式：首先，我们拥有一个共同的全局优化目标；其次，他们会产生并广播自己的排序策略。为了让说明更加清晰，我们将描述一些将 MA-DRPG 用于淘宝上的核心概念。

环境（Environment）。环境是在线的电商系统。它的状态会随着 2 个排序场景的策略变化而变化，它会对每个排序策略给出相应的收益。

智能体（Agents）。有 2 个智能体，分别是搜索排序和店铺内排序。在每一个时刻，一个搜索引擎会根据自己的排序策略给用户返回一个商品列表。2 个智能体共同优化一个整体的淘宝平台收益，例如 GMV。

状态（States）。如前文提到，状态是部分可观测的。每个排序场景智能观测到当前场景下的信息，例如用户的数据（年龄、性别、购买力等）、用户点击的商品（价格、销量等）、搜索词的类型等。一个 52 维的向量被用来表示一个状态。如 MA-DRPG 算法中所说，完全的状态向量除了包括本地的观测，还包括全局的消息。

行为（Actions）。每个智能体需要在用户搜索时输出一个排序的商品列表，因此我们将每个智能体的行为定义为一组 ranking feature 的权重。计算排序分数，是将一个商品的特征值与对应的特征权重做内积；改变行为，意思则是变更排序特征的权重。在主搜索中，我们行为的维度是 7，而在店铺内搜索中，我们行为的维度是 3。

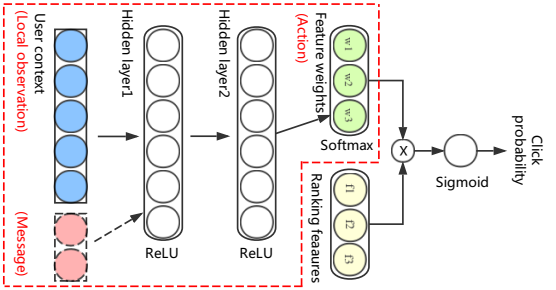


图 3.6: Actor 网络。红色虚线的部分输出一个实数值的排序向量（绿色），蓝色和红色分别表示本地观测和接收消息。

每个智能体有它独自的策略函数，图3.6给出了 actor 网络的结构。网络结构是一个 3 层的感知机，前 2 层使用 ReLu 作为激活函数，而最后一层使用 softmax 作为激活函数。网络的输入是本地观测以及收到的消息，而输出是一组排序的特征权重。

收益（Reward）。在我们的系统中，设计的收益不只是成交行为，其他行为也被考虑到。这样，我们能更多的使用用户的反馈行为特征。

如果产生了一个成交行为，会得到一个等于成交价格的收益；如果一个点击发生了，会得到一个等于 1 的收益；如果没有任何点击或者成交，会得到一个值为-1 的收益；如果用户直接离开了搜索，那么会得到一个值为-5 的收益。

3.4　实验

为了验证 MA-RDPG 算法的效果，我们将算法应用到了淘宝的在线环境——主搜索和店铺内搜索中。

3.4.1 实验设置

训练流程。训练流程如图3.5(b) 所示。我们的训练过程基于淘宝的实时在线训练系统，首先系统会实时的获取用户的行为日志，为 MA-RDPG 算法提供训练样本；然后这些样本存储在一个 replay buffer 中；最后，更新模型，并将更新后的模型应用于线上。这个流程不断反复，因此这个在线模型是在线动态更新的，从而捕捉到用户的行为变化。

参数设置。对每个智能体（搜索场景），本地的观测都是一个 52 维的向量，行为分别对应 7 维和 3 维的向量。由于通信模块和评价网络都需要各个不同场景的行为，因此为简单，我们使用一个长度为 10 的向量（0 补充空的部分）作为 LSTM 和评价网络的输入。

对通信模块来说，输入是一个 $52+7+3=62$ 维的向量，同时输出是一个 10 维的向量。网络结构如图3.4。在 actor 网络，输入维度是 $52+7+3=62$，网络隐层的大小分别是 32/32/7（3）。前 2 层的激活函数使用的 ReLU，最后一层的激活函数是 Softmax。网络的结构如图3.6。

评价网络有 2 个隐层，每层的神经元个数是 32，使用 ReLU 作为激活函数。

Bellman 公式中的收益衰减系数设为 $\gamma=0.9$。在我们的实验中，我们使用 RMSProp 来学习网络的参数；学习率则是使用的 10^{-3} 和 10^{-5}，分别对应 actor 网络和 critic 网络。我们使用的 replay buffer 的大小是 10^4，每个 minibatch 的大小是 100。

3.4.2 对比基准

排序算法的对比基准如下：

经验权重 (EW)。这种算法中，主搜索和店铺内搜索的排序权重是人工根据经验确定的。

Learning to Rank (L2R)。在这种算法中，特征的权重使用一种基于 point-wise 的 L2R 算法学习得到。算法使用一个同样的结构的神经网络，如图3.6所示，但是输入不包含收到的消息。这个网络通过用户的反馈行为学习得到。

EW，L2R，MA-RDPG3 个算法主要的不同在于产出排序特征权重的方式。

在 MA-RDPG 算法中，排序的特征权重使用 actor 网络产出，一些有代表性的特征在表 3.1中列出。

基于上述算法，我们将 MA-RDPG 与 3 中独立优化的方法进行对比：1) EW+L2R; 2) L2R+EW; 3) L2R+L2R。加号左边表示主搜索使用的排序算法，右边表示店铺内搜索使用的排序算法。

3.4.3 实验结果

实验对比指标

我们与上述基准算法对比，报告提升的相对百分比，基准是主搜和店铺内搜索都是用 EW 算法。使用的指标，GMV gap 定义为 $\frac{(GMV(x)-GMV(y))}{GMV(y)}$，对了进行一次公平的对比，我们将算法使用标准的 A/B 测试进行，3% 的用户被选作测试组，实验进行 10 天左右。我们同时提供了每个场景的实验效果，以便我们分析 2 个场景之间的关联。

实验分析

实验的结果被列在表 3.2中，从中我们能得到以下结论：一，我们的 MA-RDPG 算法的在线效果明显优于其他算法。更具体的，MA-RDPG 从整体收益来看是比 L2R+L2R 算法更加有效的；L2R+L2R 算法是淘宝在线使用的算法，效果一直比较优秀，但是该算法只考虑各自场景本身，而没有考虑场景之间的协同合作。这说明，不同场景之间的合作是确实能提升提升 GMV 的。

二，使用 MA-RDPG 后，店铺内搜索 GMV 的提升非常明显，同时主搜索有一定的提升。这种现象产生的主要原因是有更多的用户是从主搜索到店铺内搜索的，而不是反过来。因此，从这 2 个场景的合作中，店铺内搜索能够获得更大的收益。

三，L2R+EW 的实验结果进一步证明了不同场景之间是需要进行合作的，因为我们可以比较明显的看到，在单独优化主搜索的时候，店铺内搜索的指标是会被损伤的。

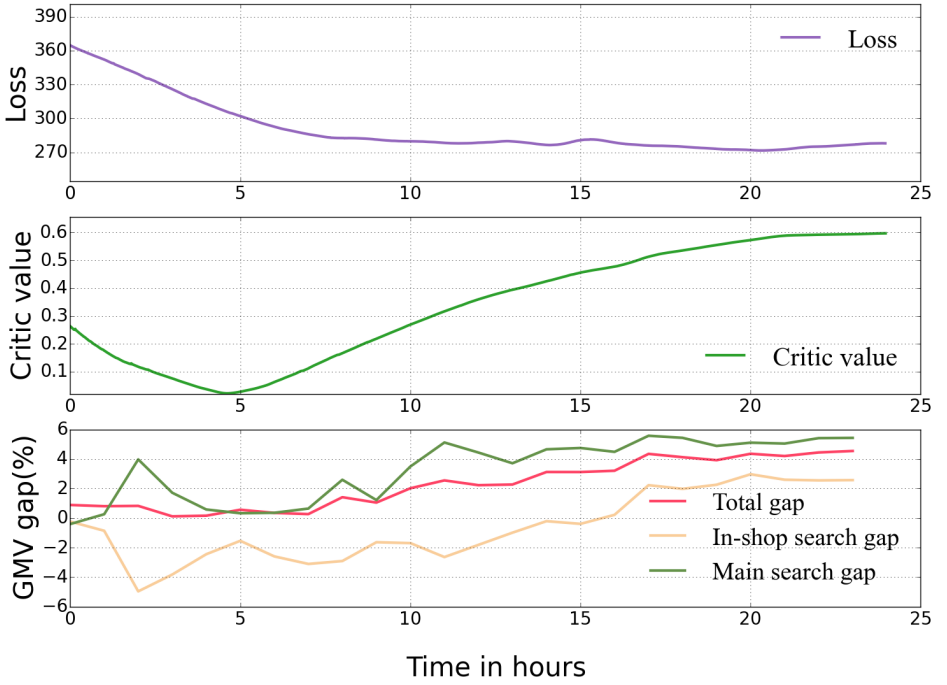


图 3.7: 上/中: critic/actor 网络的学习过程。下: GMV 的在线提升随时间的变化.

图3.7的下面，我们可以看到 MA-RDPG 对在线 GMV 的提升对比时间的变化。我们可以看到算法对 GMV 的提升是连续而稳定的。

行为分析

如之前提到，每个智能体都是使用的连续行为，因此我们可以分析不同搜索场景下，行为随时间的变化，如图3.8所示。因为每个行为都是是个实质的向量，因此我们曲线中画的是不同维度上行为的平均值。

上面一张子图画的是主搜索上行为随时间变化的曲线。**Action_1** 拥有最大的权重，它对应的 feature 是 CTR 预估分。这表示，CTR 预估是我们学习得到的最重要的特征，这与我们对排序的认识也是一致的。**Action_6** 是第二重要的

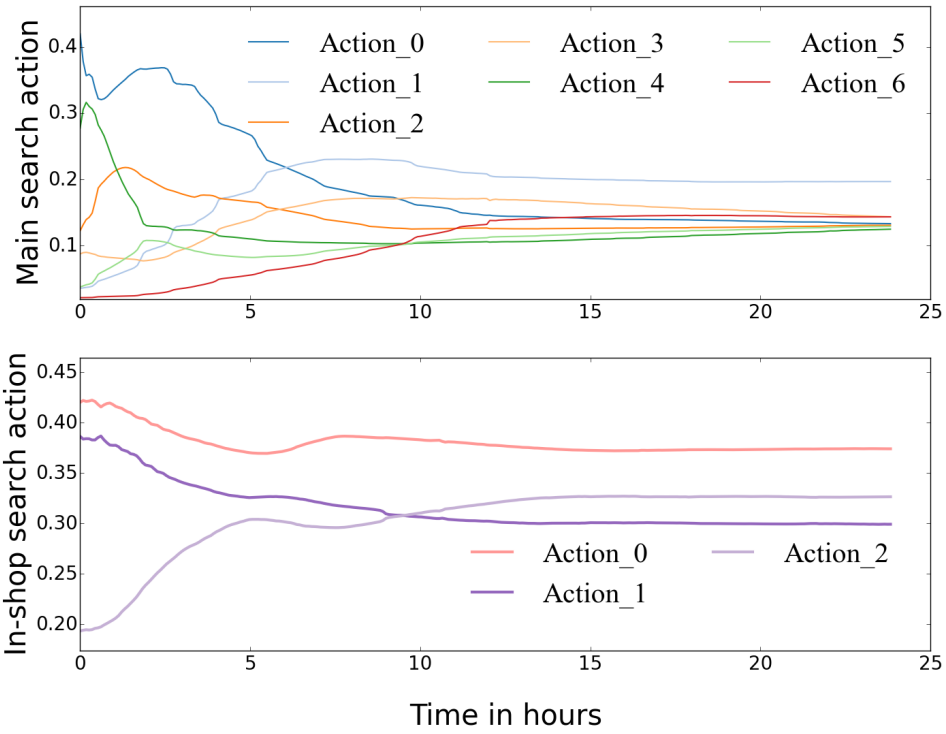


图 3.8: 主搜和店铺内搜索的 action 均值变化情况。

特征，它表示的是商品对应店铺的热度。在 L2R 模型中，我们会看到这并不是一个重要的特征，但是在我们的实验中，它表现得出乎我们意料的重要。这主要是因为通过这个特征，主搜索能直接将更多的用户流量引导到店铺内搜索中，从何实现合作。

下面一张子图描述的是店铺内搜索的行为随时间变化。**Action_0** 是最重要的特征，它表示的是一个商品的销量；这表示在一个店铺内，热销的商品往往会更容易达成成交。

尽管不同行为的分布一开始会变化的非常大，但是在训练 15 小时的时候，就趋于平稳。这与图3.7上也是一致的。

3.4.4 在线示例

在这一小节我们进一步分析了一些典型的例子，说明 MA-RDPG 是怎么让主搜索和店铺内搜索合作起来的。考虑到在线系统的变化太多，我们这里集中分析一些典型情况，对比 MA-RDPG 和 L2R+L2R 算法的排序结果。

第一个例子反映的是主搜索怎么去帮助店铺内搜索，从而得到更多的整体收益。我们假设这样一个场景：一个强购买力的女性用户点击了很多高价而低转化的商品，然后搜索了一个关键词“连衣裙”。不同算法的结果展示在图3.9(a)中。显然，MA-RDPG 更容易返回一些大店中的高价而低销量的商品，这能让用户更容易进到店铺中。对比 L2R+L2R 算法，MA-RDPG 能从一个更加全局的角度进行排序，它不仅考虑了当前的短期点击和成交，同时会考虑潜在的会在店铺内进行的成交。

第二个例子，我们选的情景是一个男士想买一个冰箱。他首先在主搜搜索了“冰箱”，然后点击进了其中一个商品，通过该商品进入了店铺；这个店铺是一个非常大型的家电卖家。图3.9(b) 中分别是店铺内搜索 MA-RDPG 和 L2R+L2R 的排序结果对比。从图中可以看到，MA-RDPG 算法更容易排出冰箱，而 L2R+L2R 展示的则更加发散。这主要是因为店铺内搜索的时候，会接受到主搜索传递的信息，因此会基于主搜的排序结果产生更合理的排序。

3.5 总结与展望

随着 AI 技术的发展，越来越多的新技术涌现，我们的排序也经历了从规则到浅层监督学习，到深度学习、强化学习等智能排序算法的转变。新的问题与挑战，迫使我们不断地开拓与进取。从一个场景的优化，到现在尝试着联合两个场景一起优化，这只是联合优化的一小步，现在的做法也比较简单，甚至还存在着非常多的不足和缺陷，还有更多更复杂的问题需要我们去克服去解决。我们相信在面对未来越来越复杂的电商排序场景下，平台需要的不是各场景之间的互搏与内耗，而是协同与合作，我们期待更多更高级的多场景联合优化的方法涌现，为平台创造更大的价值。

Algorithm 1: MA-RDPG

Input: The environment

Output: $\theta = \{\theta^1, \ldots, \theta^N\}$

Initialize the parameters $\theta = \{\theta^1, \ldots, \theta^N\}$ for the N actor networks and ϕ for the centralized critic network;

Initialized the replay buffer R;

foreach *training step* e **do**

  for *i = 1 to M* **do**

    h_0 = initial message, $t = 1$;

    while $t < T$ *and* $o_t \neq terminal$ **do**

      Select the action $a_t = \mu^{i_t}(h_{t-1}, o_t)$ for the active agent i_t;

      Receive reward r_t and the new observation o_{t+1};

      Generate the message $h_t = LSTM(h_{t-1}, [o_t; a_t])$;

      $t = t + 1$;

    end

    Store episode $\{h_0, o_1, a_1, r_1, h_1, o_2, r_2, h3, o3, \ldots\}$ in R;

  end

  Sample a random minibatch of episodes B from replay buffer R ;

  foreach *episode in* B **do**

    for *t = T downto 1* **do**

      Update the critic by minimizing the loss:

      $L(\phi) = (Q(h_{t-1}, o_t, a_t; \phi) - y_t)^2$, where

      $y_t = r_t + \gamma Q(h_t, o_{t+1}, \mu^{i_{t+1}}(h_t, o_{t+1}); \phi)$;

      Update the i_t-th actor by maximizing the critic:

      $J(\theta^{i_t}) = Q(h_{t-1}, o_t, a; \phi)|_{a=\mu^{i_t}(h_{t-1}, o_t; \theta^{i_t})}$;

      Update the communication component.;

    end

  end

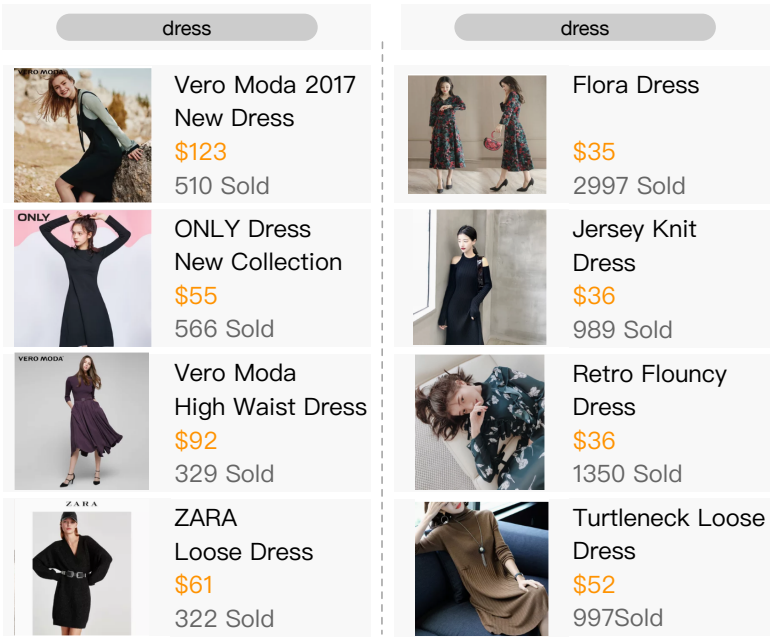
end

表 3.1: Examples of Ranking Features

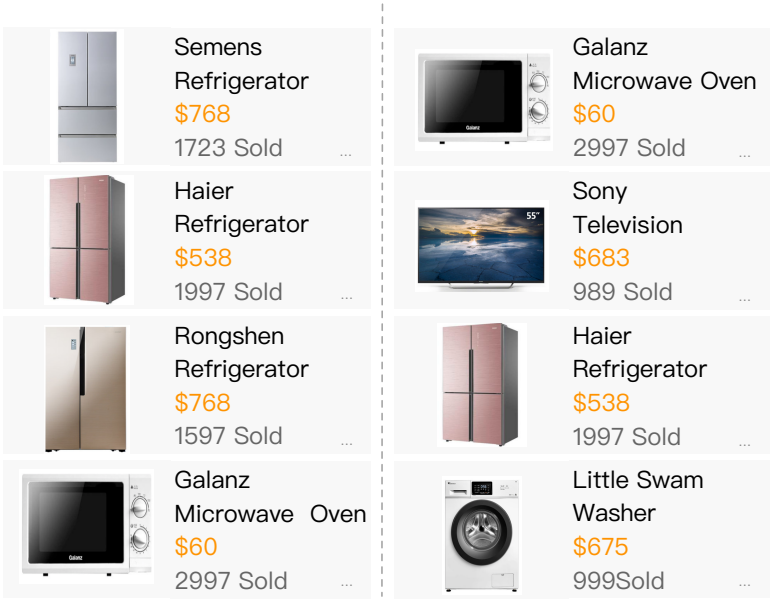
Scenario	Feature Name	Description
Main Search	Click Through Rate	An CTR estimation using logistic regression, considering features of users, items and their interactions
	Rating Score	Average user ratings on a certain item
	Shop Popularity	Popularity of the item shop
In-shop Search	Latest Collection	Whether an item is the latest collection or new arrivals of the shop
	Sales Volume	Sales volume of an in-shop item

表 3.2: GMV gap evaluated on an online E-commerce platform. A+B means algorithm A is deployed for the main search and B for the in-shop search. The values are the relative growth ratio of GMV compared with the EW+EW setting.

day	EW + L2R			L2R + EW			L2R + L2R			MA-RDPG		
	main	in-shop	total	main	in-shop	total	main	in-shop	total	main	in-shop	total
1	0.04%	1.78%	0.58%	5.07%	-1.49%	3.04%	5.22%	0.78%	3.84%	5.37%	2.39%	4.45%
2	0.01%	1.98%	0.62%	4.96%	-0.86%	3.16%	4.82%	1.02%	3.64%	5.54%	2.53%	4.61%
3	0.08%	2.11%	0.71%	4.82%	-1.39%	2.89%	5.02%	0.89%	3.74%	5.29%	2.83%	4.53%
4	0.09%	1.89%	0.64%	5.12%	-1.07%	3.20%	5.19%	0.52%	3.74%	5.60%	2.67%	4.69%
5	-0.08%	2.24%	0.64%	4.88%	-1.15%	3.01%	4.77%	0.93%	3.58%	5.29%	2.50%	4.43%
6	0.14%	2.23%	0.79%	5.07%	-0.94%	3.21%	4.86%	0.82%	3.61%	5.59%	2.37%	4.59%
7	-0.06%	2.12%	0.62%	5.21%	-1.32%	3.19%	5.14%	1.16%	3.91%	5.30%	2.69%	4.49%
avg.	0.03%	2.05%	0.66%	5.02%	-1.17%	3.09%	5.00%	0.87%	3.72%	5.43%	2.57%	4.54%



(a) Main search results.



(b) In-shop search results.

图 3.9: 搜索结果对比

第四章　强化学习在淘宝锦囊推荐系统中的应用

4.1　背景

4.1.1　淘宝锦囊

在手淘的搜索中，当用户输入 query 进行搜索之后，一方面有适合他的商品展现出来，另一方面，如何更好地理解用户意图，为其推荐更合适的关键词进行细分查找，引导用户到他想找的商品，也是一件非常重要的事情。因此在手淘搜索场景下，我们以“锦囊”这种产品形态来承载对用户意图的理解和细分。同时锦囊的内容也分了不同的类型。目前，我们已经设计了两万多种类型的锦囊，如“细选”,“相关搜索”, 以及和商品 PID 相关的“袖长”,“裙长”等类型的锦囊。比如当用户输入“连衣裙”这一并不是特别精确的 query 后，我们可能提供“适用年龄”类型锦囊，而用户在锦囊中点击“18-24 岁”来精细化自己的需求，从而更快地找到想要的商品，提升购物的满意度，下图 4.1展示了锦囊的典型形态：

4.1.2　锦囊的类型调控

锦囊存在的目的是为了给用户提供良好的导购功能，我们不希望给用户带来过多的打扰。锦囊的原理是在当前的搜索结果页 (SRP) 中插入一个内容框，提供一定数目的候选词，或者是一个交互式的问答，用户一旦点击，就会将原先

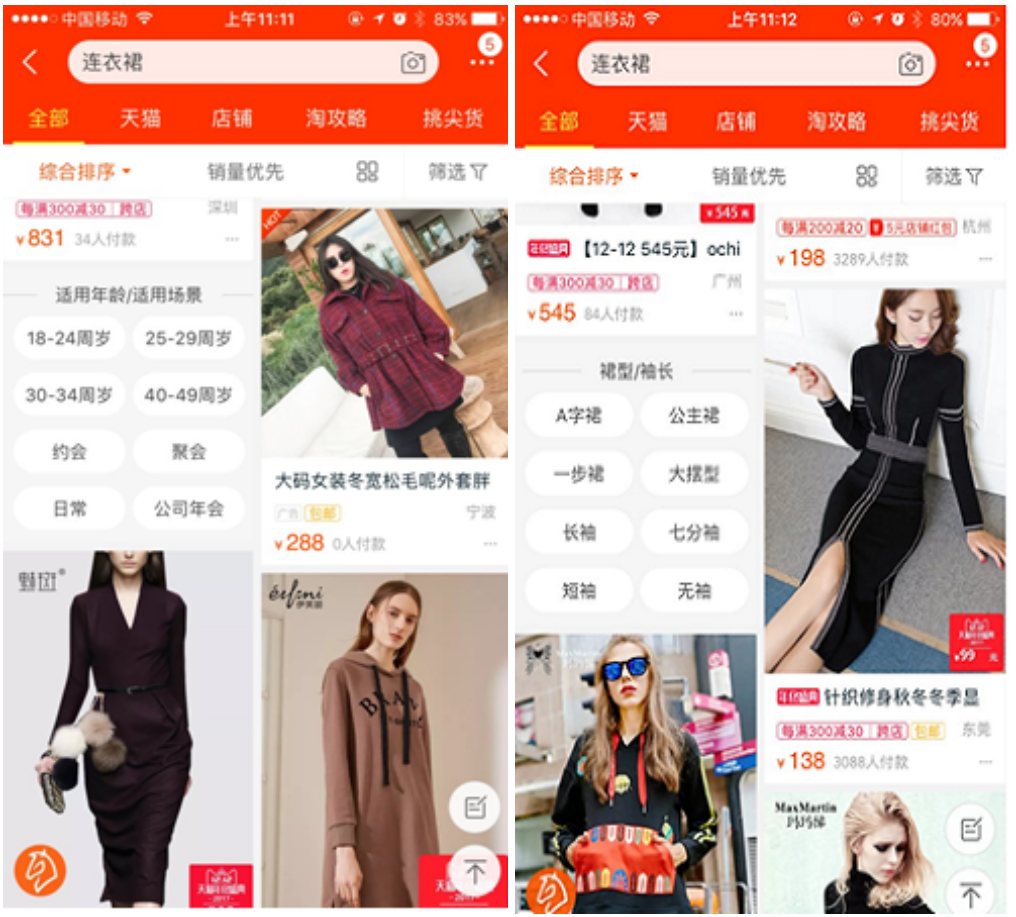


图 4.1: 淘宝上锦囊的典型形态

的 query 和锦囊进行结合，重新搜索从而得到新的 SRP，如下图 4.2所示。因此，如何根据用户的历史行为，以及当前的实时状态，给其推荐合适的锦囊，就变得很重要。这个工作的目标就是在不增加锦囊的展示 PV(page-view) 基础上，提升锦囊的 CTR，以及使用用户数，即表明给用户提供了更加需要的锦囊，方便了用户的购物过程。

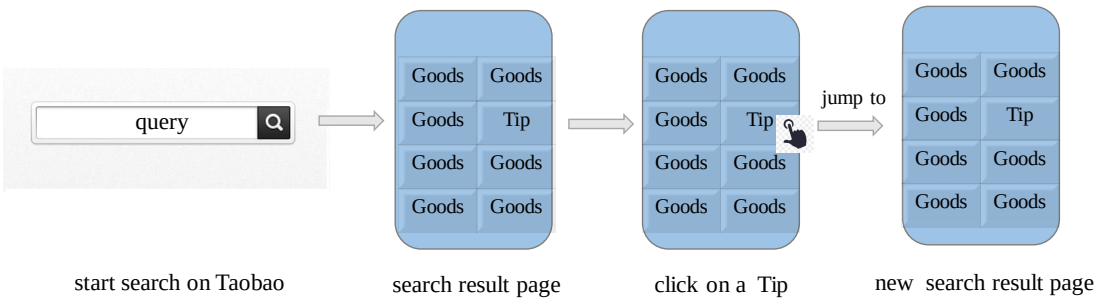


图 4.2: 淘宝上锦囊被点击后的生效逻辑

4.1.3 工作摘要

本工作主要是针对淘宝的锦囊产品的候选类型的实时推荐。学习方法上，由于两万多个候选动作的存在，我们基于 deep double Q-learning [35] 进行适应性改变，将动作 ID 进行 embedding, 再进行 Q 值的学习 [19]。然而，由于我们的电商情形和传统的强化学习应用的游戏等问题的存在很大不同：1. 用户分布变化较大，导致了策略评估不准；2. 用户群体变化很大，导致了很差的策略可能被认为是好策略的问题。我们又提出了两种方法来缓解：1. 采用 stratified sampling replay 取代原始的 random replay；2. 利用基准数据的实时变化进行约减。并结合整个系统进行了相关实验，最终 CTR，UV 等指标均达到了相关预期。

4.2 系统框架及问题建模

4.2.1 系统框架

考虑到系统的安全性以及实现的可能性，我们把锦囊类型的推荐系统分割成两个部分，主要包括：1. 学习模块；2. 推荐实际生效模块。其中，学习模块的功能是实时的获取用户的浏览，点击日志，用户特征，锦囊特征等，然后从中获取得到我们训练需要的 s, a, s', r 等，并进行训练，并更新网络参数。同时，学习

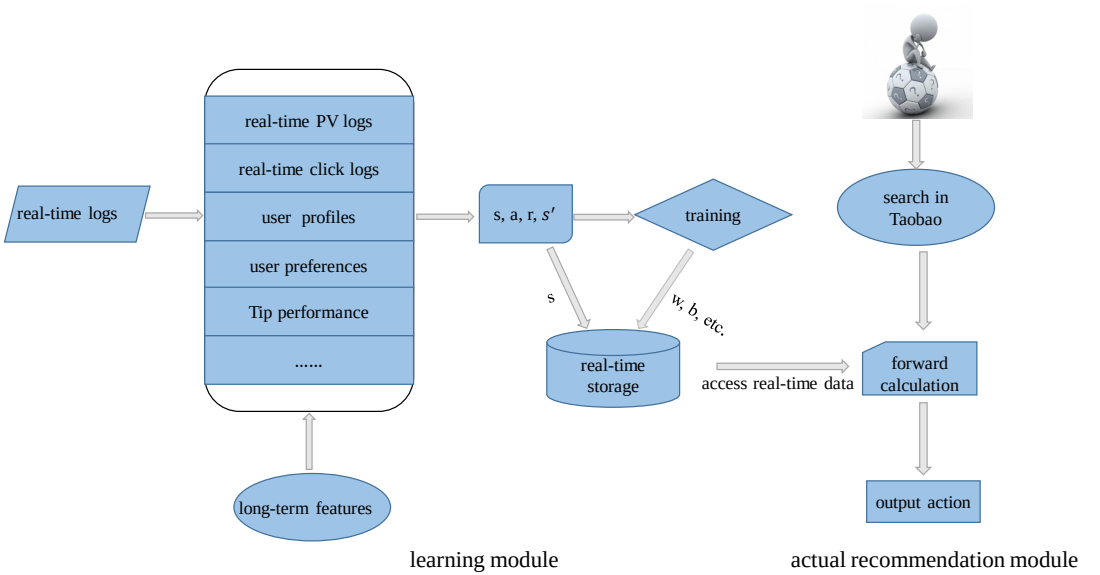


图 4.3: 锦囊推荐系统的框架.

模块中得到的实时状态 s, 以及网络参数会放到实时的存储中 (iGraph)。而实际生效的推荐模块，则一方面根据用户 ID 去获取相应的 state 等信息另一方面则去获取最新的网络参数，然后在生效模块中就可以进行网络的前向计算，得到相应的结果，最终进行锦囊类型的推荐。图 4.3 展示了我们的系统的框架：

4.2.2 问题建模

考虑到用户在淘宝平台上的行为具有序列决策性，符合强化学习中延迟反馈的设定，并且强化学习能很好地捕捉实时变化，因此我们决定用强化学习来求解问题。强化学习通常用马尔科夫决策过程 (MDP) 来建模，在我们的问题上，我们建模成如下的形式 [33, 34]：

状态

状态应当能够代表用户的长期和当前的特征，以及对商品和锦囊的一些偏好。因此，我们首先添加用户的一些特征，如性别、年龄、购买力、偏好等。然后，还添加了 query 的相关特征，如当前 query 的类型、不同类型的用户整体和当前用户的偏好情况。此外，状态中还包含了用户当前的行为、页面编号、查看和点击的商品的特征等信息。

动作

我们的目标是学习一种策略，它可以决定在当前页面上显示哪种类型的锦囊，所以我们就直接将锦囊类型作为动作。现在有超过二万种类型的锦囊，动作空间非常大，直接采用原始 DQN 网络最后一层输出各个动作的 Q 值的形式没办法继续使用，需要进行一定的改变。

奖赏函数

首先，我们想推荐给用户的是用户最需要的，最有可能去点击的锦囊。当点击发生时，应该给予正的奖赏，当点击不发生时，情况则正好相反。同时，考虑到锦囊作为一种导购性质的产品，用户在前面页数点击锦囊比在靠后面点击时应该更有价值，这说明推荐的锦囊更加被用户需要。因此，在奖赏的设计中也应考虑点击发生的时间。奖赏函数如下所示：

$$r_1 = I * (1 + \alpha * e^{-x}) \tag{4.1}$$

其中当点击发生时 I 为 1 否则为 0，x 为页数，而 α 则是一个系数。

然后，我们又考虑到不同用户的习惯。有些用户习惯于点击锦囊，而某些用户很少去点击。如果说一个很少有点击行为的用户选择了点击我们提供的锦囊，可以认为该锦囊是更有价值的，也就是应该给予更多的奖赏，因此我们有：

$$r_2 = I * e^{-y} \tag{4.2}$$

其中，y 表示的是在最近的 100 次 PV 中用户点击锦囊的次数。最终我们将以上进行结合得到：

$$r = r_1 + \beta * r_2 \tag{4.3}$$

β 是一个 0 至 1 之间的系数。

4.3　算法及模型设计

强化学习中，主要有两大类学习方法，分别是基于值函数 (value-based) 的以及基于策略 (policy-based) 的方法。考虑到我们的问题有两万多个离散的候选动作，以及问题的应用情形，我们采用了前者作为方案 [33]。

4.3.1　主体框架

基于值函数的方法中，最有名的就是 DeepMind 的 DQN 系列的方法了，DQN 在 Atari 上取得了非常好的表现。DQN 中每一轮迭代的 loss function [26]：

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right] \tag{4.4}$$

其中：

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{4.5}$$

θ^- 表示的是一个独立的目标网络 (*target network*) 的参数。除此之外，*experience replay* 也被采用来提升 DQN 的效果。Agent 从很多的 episodes 中获得 experiences $t_i = (s_i, a_i, r_i, s_{i+1})$，从而形成了一个 buffer，$\mathcal{D} = t_1, t_2, ...t_i$。然后从这个 buffer 中随机采样来获得样本进行网络的训练。则其 loss function 可表达为：

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'\sim u(\mathcal{D})}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right] \tag{4.6}$$

然而，由于 Q-learning 中存在对值函数估计的过于乐观的问题，又有了将 double Q-learning 引入的 deep double Q-learning [35]. 主要是将其中的 target 更换

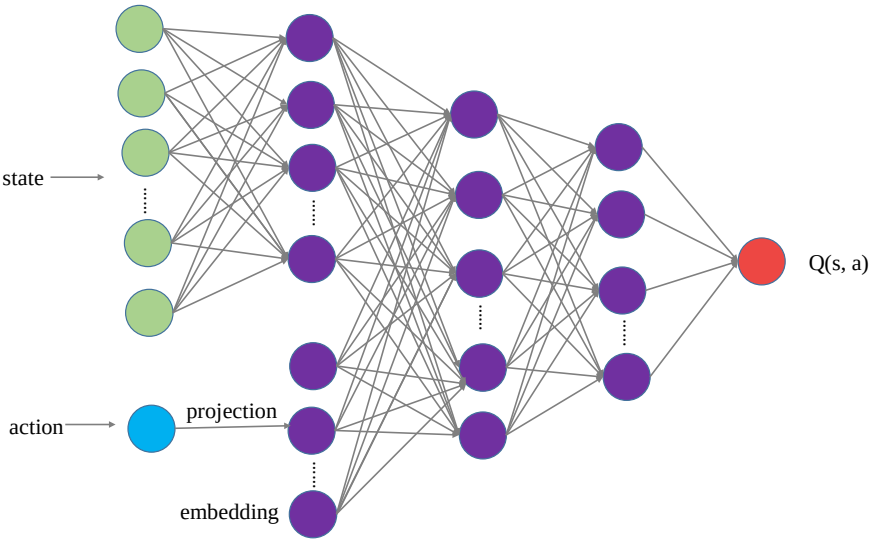


图 4.4: 网络结构.

为如下形式：

$$y_i^{DDQN} = r + \gamma Q(s', \underset{a'}{\text{argmax}}\, Q(s', a'; \theta_i); \theta^-) \tag{4.7}$$

而其余部分则保持和原 DQN 部分相同。

同时，因为我们的候选动作有两万多个，不能像一般的 DQN 网络一样，在最后一层里一次性得到所有动作的 Q 值，从而形成策略。我们采用的方法是，首先，给每一个候选的锦囊类型一个 ID。然后，将此 ID 与 state 过来得到的量进行 concat，这个结合的量继续送入网络进行计算。最终得到的就是对应的 $Q(s, a)$。网络的结构可以大概如下图 4.4所示：

4.3.2 分层采样池

通常，在 deep RL 中，都会采用 replay memory 来保存历史样本。从 nature DQN 开始，该方法已经取得了很大成功和效果。然而，在以往所遇到的问题中，环境的分布通常较为固定。但是在我们的问题中，Agent 是推荐系统，环境则是

广大的用户，策略则是向用户推荐何种类型的锦囊，而用户的分布通常变化非常快。用户分布的变化会带来一些问题，其中之一是由于不同类型的用户行为之间有很大习惯差异，当用户分布发生变化时，则可能导致错误的策略评估。考虑一个特殊情况，1 型用户更愿意点击 A 型锦囊，而目前 1 型用户占大多数，当系统已经学会了主要推荐的是 A 锦囊的策略时，2 型用户却成为平台上的多数用户。在这种情况下，显然会导致学习效果变差。

为了缓解这个问题，我们决定使用 *stratified sampling replay* 代替以前的 random replay。同样，仍然有一个 replay memory 用于保存以前的那些样本。然而，与 random replay 不同，我们会在某些指标上进行分层抽样，比如根据用户性别，年龄，购买力等，而不是随机进行采样，从而使得具有更小的方差。

其中，在分配各层样本数量时，有多种方案，我们主要尝试了 *Proportional Allocation* 以及 *Optimal Allocation* 两种方法并进行实验。其中，前者是按照各层在总体中占有的比例来进行同比例的采样。后者的分配目标则是需要使得最后的方差最小，当然，这里就需要先估计各层数据的均值，方差等。[27, 14]

4.3.3 基准约减

除了用户分布的巨大变化之外，整个用户群体在不同时间段的行为特性也会发生波动，可能用户在晚上就是比白天要更加愿意去点击。一种情景是，当系统推荐策略保持不变时，锦囊的点击率会突然增加，然而这可能仅仅是因为用户更愿意在那个时候去点击。这也显然导致了一个问题：当策略没有朝正确的方向更新时，却由于外部环境的变化，它被误认为取得了很好的学习表现，从而使得最终的学习效果并不好。

出于这个原因，我们提出了另一个方法：基准约减 (*Benchmark Elimination*)。首先，我们随机选择一些用户作为我们的基准用户。其次，以优化 CTR 为目标，用离线监督学习方法进行训练，得到一个模型，并将此固定模型应用于上述用户群体中。之后，对于这些用户，我们也实时记录用户的浏览、点击等行为，并用和上述相同的方案计算平均奖赏 $r_{benchmark}$。最后，在我们的推荐系统里的用户里也得到了奖赏 r_{origin} 之后，将两种 reward 之差作为最终的奖赏：

Algorithm 2: 锦囊类型推荐算法

Require:

$\mathcal{D}$: 空的 replay 池, N_r: replay 池的最大容量

θ: 初始的网络参数, θ^-: θ 的一份副本,

N_b: 训练的 batch size, N^-: target network 更新频率

r_b: 实时的 benchmark 奖赏.

for *episode e* ∈1,2,3,...M **do**

  for $t \in 0, 1, ...$ **do**

    从环境 $\mathcal{E}$ 获取 state s, action a, reward r_{origin}, next state $s^{'}$

    基准约减: $r = r_{orign} - r_{benchmark}$

    将 transition tuple $(s, a, r, s^{'})$ 加入到 $\mathcal{D}$

    if $|\mathcal{D}| \geq N_r$ **then**

      替换掉最久之前的的 tuple

    end if

    用分层采样的方法采样一个 mini batch：N_b tuples $(s, a, r, s^{'}) \sim SRS(\mathcal{D})$

    对每一个 N_b tuples 建立目标值函数

    定义 $a^{max}(s^{'}; \theta) = argmax_{a^{'}} Q(s^{'}, a^{'}; \theta)$

$$y_j = \begin{cases} r, & if\ s^{'}\ is\ terminal \\ r + \gamma Q(s^{'}, a^{max}(s^{'}; \theta); \theta^-), & otherwise. \end{cases} \tag{4.8}$$

    用如下 loss 进行梯度下降过程：$||y_j - Q(s, a; \theta)||^2$

    每间隔 N^- 步进行更新：$\theta^- \leftarrow \theta$

  end for

end for

$r = r_{origin} - r_{benchmark}$. 这种方法是为了抵消外部环境的自然变化带来的负面影响，使得推荐系统能够向着更加正确的方向来进行更新。

4.3.4 算法流程

如上所述，我们采用了强化学习的值函数估计的方法，并使用了深度神经网络来进行拟合。此外，为了解决用户分布和用户群体快速变化这两个问题，我们增加了分层采样 replay 和基准约减两个方法。算法流程如 2 所示。基准约减将在第 4 行进行，之后，这些 transitions 都被存储在 $\mathcal{D}$，并将按照一定的指标进行分层随机抽样 (SRS)。然后就可以用学习到的 Q 值来得到策略。

4.4 实验与总结

我们将上述算法应用到淘宝实际的锦囊类型推荐中进行了实验，最终的 CTR 和 UV 均比以往方法有所提升。一方面，我们对比了我们强化学习的方法和直接用一样的网络结果的单纯监督学习方法，发现强化学习方法有所增益。另一方面，我们也测试了提出的 stratified sampling relay 对比 random replay 的效果, 以及用了基准约减对比直接原始 reward 的表现，实验的结果证明了这两种方法对系统的效果均起到了很好的作用。

以往的强化学习方法更多的应用在游戏领域中，而电商环境具有更加明显的不确定性，方差更大。本文提出的方法也是试图探索方法来减弱这种影响。然而，笔者能力有限，在应对真实复杂的环境，以前的方法确实仍然存在太多问题。比如用户的特征并不完全在淘宝平台上展现，并不完全符合马尔科夫性等等，还需要更多的研究和尝试，以求取得更多的突破。

第五章　基于强化学习的引擎性能优化

5.1　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！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：

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

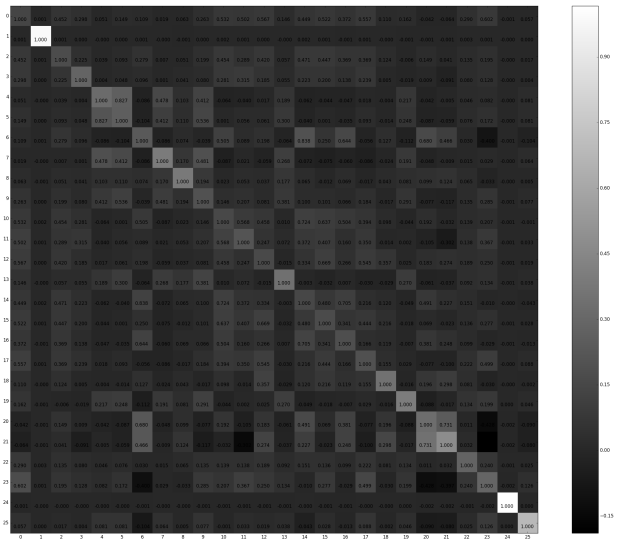


图 5.1: 排序因子的皮尔逊积矩相关系数矩阵

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 $< user, query >$ 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？

5.2　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 $s(d) = (x_1(d), x_2(d), ..., x_n(d)) \in R^n$。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

$$F(u,q,d) = f\big(x_1(d), x_2(d), ..., x_n(d)\big) \tag{5.1}$$

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，召回之后有 m 个商品待排序，我们可以使用全部因子集合 Ω 计算

$$F_o = [f\big(s(d_1)), f(s(d_2)), ..., f(s(d_m)\big)] \tag{5.2}$$

亦可取一个子集 $S \subset \Omega$，计算近似值

$$F_a = [f\big(\pi_S(s(d_1))), f(\pi_S(s(d_2))), ..., f(\pi_S(s(d_m)))\big)] \tag{5.3}$$

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

$$\min_{S \subset \Omega} \ D(F_o||F_a) + \lambda||S||_0 \tag{5.4}$$

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，其直觉含义是，在尽量用少的因子的情况下，尽可能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：

$$S_{u,q} = H(u,q|\theta) \tag{5.5}$$

转而去求解一个全局的模型参数 θ。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

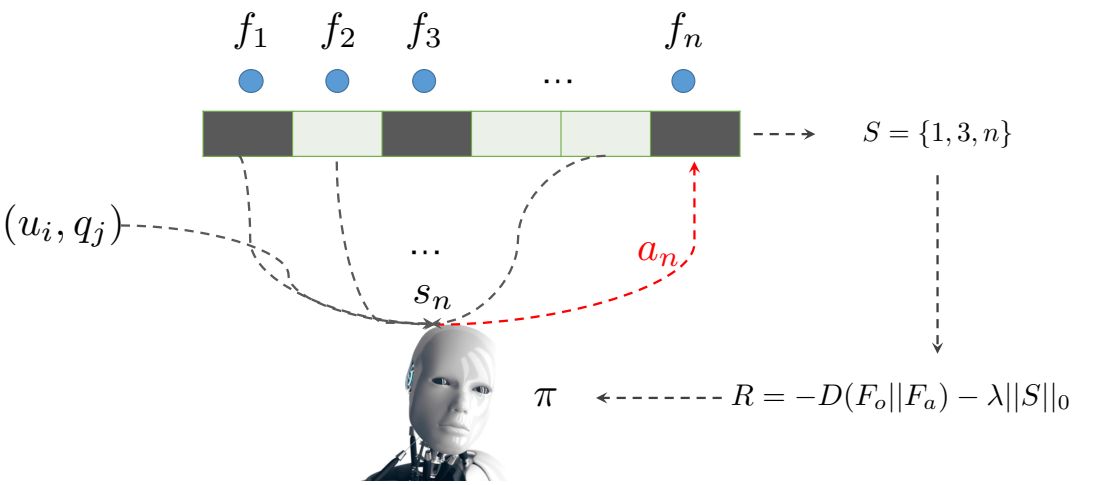


图 5.2: 因子选择的序列决策模型

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。

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。

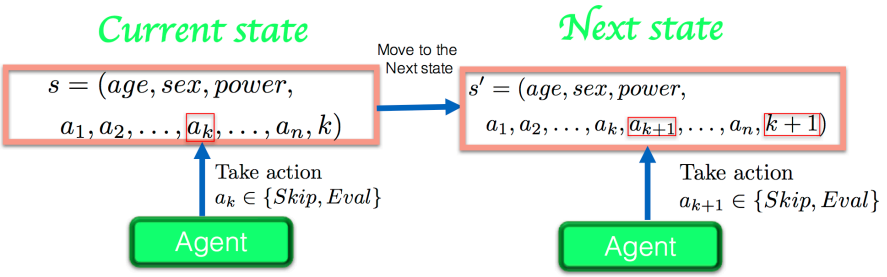


图 5.3: 状态转移建模

5.2.1 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：

$$s = (age, sex, power, a_1, a_2, \ldots, a_n, T) \tag{5.6}$$

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。

5.2.2 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。

5.2.3 状态转移函数

状态转移函数 T 我们的设计方中比较简单，如图5.3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。

5.2.4 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

$$b(a_k) = \begin{cases} 0 & \text{if } a_k = \text{Skip} \\ 1 & \text{if } a_k = \text{Eval} \end{cases} \tag{5.7}$$

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：

$$T(s_k, a_k) = \begin{cases} r_p & t < C \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

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：

$$\hat{R}(s_k, a_k) = \begin{cases} 0 & \text{if } a_k = \text{Skip} \\ c_k^{i,j} & \text{if } a_k = \text{Eval} \end{cases} \tag{5.9}$$

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高开销的 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：

$$\mathcal{R}(s_k, a_k) = -\hat{\mathcal{R}}(s_k, a_k) - \mathcal{T}(s_k, a_k). \tag{5.10}$$

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。

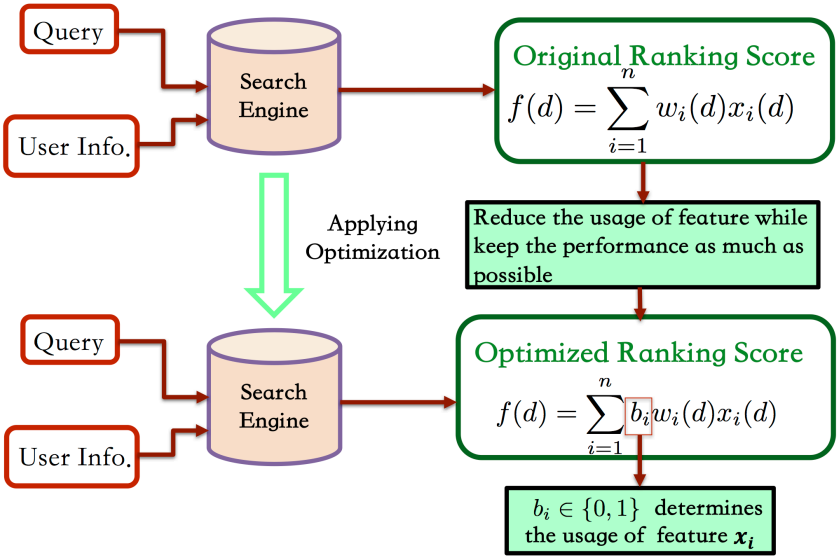


图 5.4: 系统框架图

5.3 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，具体设定见图5.4：

5.3.1 Loss Function

我们的 Loss 函数定义如下

$$\mathcal{J}(\theta) = \mathbb{E}_{q_i}\left[\mathcal{L}(b(\theta); q_i, w^{q_i})\right], \tag{5.11}$$

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：

$$\min_{\theta} \frac{1}{|\mathcal{Q}|} \sum_{i=1}^{|\mathcal{Q}|} \left(-\mathcal{J}_{\mathcal{L},\theta} \left(\tilde{\theta}; q_i, w_i \right) + \mu \left\| \tilde{\theta} \right\|_1 \right) \tag{5.12}$$

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

$$\mathcal{J}_{\mathcal{L},\theta} \left(\tilde{\theta}, p_i, w_i \right) = \int_{\tau} p_{\theta}(\tau) \mathcal{R}(\tau) \log \frac{p_{\tilde{\theta}}(\tau)}{p_{\theta}(\tau)} d\tau \tag{5.13}$$

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

$$p_{\theta}(\tau) = p_0(s_0) \prod_{m=0}^{M-1} p\left(s_{m+1} \mid s_m, a_m, q_i, w_i\right) \pi_{\tilde{\theta}}(a_m \mid s_m, q_i, w_i), \tag{5.14}$$

$$\mathcal{R}(\tau) = \frac{1}{M} \sum_{m=0}^{M} \gamma^m r_{m+1}, \tag{5.15}$$

5.3.2 Actor-crtitic 方法

我们采用目前流行的 Actor-critic 来优化上述的 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：

$$\mathcal{L}(\theta_c) = \left\| V_{\theta_c}^{\pi}(s_k) - r_{k+1} - \gamma V_{\theta_c}^{\pi}(s_{k+1}) \right\|^2. \tag{5.16}$$

其中 θ_c 是刻画 critic 网络的参数。我们的算法伪代码如下:

5.4 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：

Algorithm 1 Actor-critic training

Input training data set $\mathcal{D} = \{(\mathcal{A}_i, q_i, \boldsymbol{y}_i)\}_{i=1}^{N}$
initialize actor network params $\tilde{\boldsymbol{\theta}}$
initialize critic network params $\boldsymbol{\theta}_c$.
for each $(A_i, q_i, \boldsymbol{y}_i) \in \mathcal{D}$ (For each episode) **do**
  initialized the initial state $\boldsymbol{s}_0$ by the query q_i
  for $k = 1, \ldots, n$ (n is the number of features) **do**
    Taking action $\boldsymbol{a}_k \in$ {Skip,Keep} on feature x_k^i, observe $\boldsymbol{s_{k+1}}$, r_k
    Calculate actor loss by $-\mathcal{J}_{\mathcal{L},\theta}\left(\tilde{\theta}, p_i, \boldsymbol{w}_i\right) = -\sum_{t=0}^{k} p_{\theta}(\boldsymbol{\tau})\mathcal{R}(\boldsymbol{\tau}) \log \frac{p_{\tilde{\theta}}(\boldsymbol{\tau})}{p_{\theta}(\boldsymbol{\tau})} d\boldsymbol{\tau} - \mu \left\|\tilde{\boldsymbol{\theta}}\right\|_1$
    Update $\tilde{\boldsymbol{\theta}}$ by $RMSProp\left(\tilde{\boldsymbol{\theta}}, -\nabla_{\boldsymbol{\theta}_c}\mathcal{L}(\boldsymbol{\theta}_c)\right)$
    Calculate critic loss $\mathcal{L}(\boldsymbol{\theta}_c) = \left\|V_{\boldsymbol{\theta}_c}^{\pi}(\boldsymbol{s}_k) - r_{k+1} - \gamma V_{\boldsymbol{\theta}_c}^{\pi}(\boldsymbol{s}_{k+1})\right\|^2$
    Update $\boldsymbol{\theta}_c$ by $ADMA\left(\boldsymbol{\theta}_c, \nabla_{\boldsymbol{\theta}_c}\mathcal{L}(\boldsymbol{\theta}_c)\right)$
  end for
end for

图 5.5: 算法伪代码

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

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

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

图 5.6: 理论结果

5.5 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。

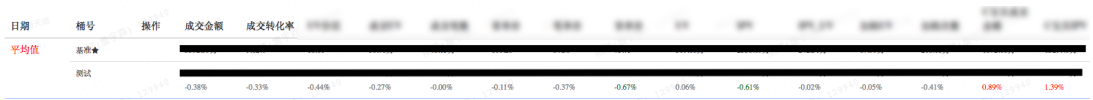


图 5.7: 实验结果

5.6　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。

第六章　基于强化学习分层流量调控

6.1　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。在满足一定的条件情况下，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，在我们这几乎无不例外的出现了。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

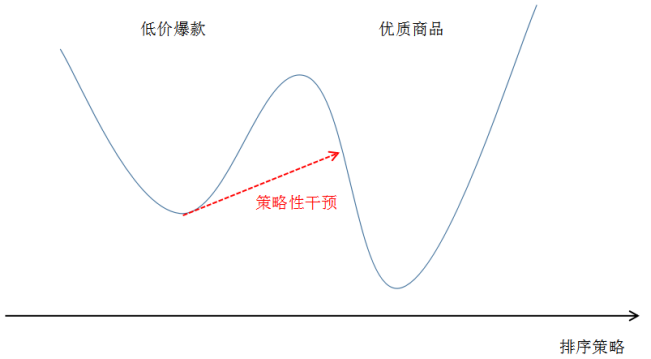


图 6.1: 局部最优和全局最优

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。

所以，这其实给我们算法同学带来 2 个问题：

(1) 如果尽可能的减少瞬时损失?

(2) 如何尽快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：

(1) 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

(2) 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作针对不同的状态，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。

6.2 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

$$a^k = L^k + (U^k - L_k)o^k, \forall k \in \{1, 2..., m\} \tag{6.1}$$

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 $p_i(\pi)$。与此同时，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，pay>cart>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 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

$$r(s,a) = \text{GMV}(n_{click}, n_{cart}, n_{buy}) + \sum_{i}^{m} \lambda_i \big(p_i(\pi) - p_i(\pi_{basic})\big) \tag{6.2}$$

6.2.1　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 actiona_k，我们假设其符合一个高斯分布，在

$$a^{k*} \in N(\mu_k, \sigma_k^2) \tag{6.3}$$

每次迭代的开始，我们从这个分布上采样 s 个样本，即 $\Omega_1, \Omega_2, ..., \Omega_s$，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

$$R(\Omega_1) = \frac{1}{N_1}\sum_{i}^{N_1} r_i(\Omega_1) \tag{6.4}$$

$$R(\Omega_2) = \frac{1}{N_2}\sum_{i}^{N_2} r_i(\Omega_2) \tag{6.5}$$

$$\cdots \tag{6.6}$$

$$R(\Omega_s) = \frac{1}{N_s}\sum_{i}^{N_s} r_i(\Omega_s) \tag{6.7}$$

然后我们对 $R(\Omega_1), R(\Omega_2), ..., R(\Omega_s)$ 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

$$\max_{\mu_k^*, \sigma_k^{2*}} \quad f(\mu_k^*, \sigma_k^{2*}) = \sum_{i \in D} \log N(\Omega_i | \mu_k^*, \sigma_k^{2*}) \tag{6.8}$$

实际上，上面的式子是有最优解的，μ_k^* 即 D 中所有样本的均值，σ_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

$$\mu_k \leftarrow \mu_k + \alpha \frac{\partial f}{\partial \mu_k} \tag{6.9}$$

$$\sigma_k \leftarrow \sigma_k + \alpha \frac{\partial f}{\partial \sigma_k} \tag{6.10}$$

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

$$L^k = \mu_k - 2\sigma_k \tag{6.11}$$

$$U^k = \mu_k + 2\sigma_k \tag{6.12}$$

$$\tag{6.13}$$

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：

图 6.2: 实验结果

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

6.3　实验效果

双 11 期间在 gmv 损失可控的情况下，目标商家流量占比大幅提升。

6.4　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。

第七章　风险商品流量调控

7.1　背景

7.1.1　为什么进行风险商品流量调控

风险商品长期以来都是淘宝平台面临的顽疾，如果不加以合理的控制，会严重威胁到平台在消费者心目中的形象。当前平台面临风险商品多，风险类型多样化，例如：

- 假货商品通过复制、仿制大牌商品，以低价的方式出售牟取暴利，危害极大，非常容易产生 PR 事件；
- 次品、瑕疵品等劣质商品，差评率和退款率高，严重影响平台的正品感知，损坏平台在买家心目中的形象；
- 大量的标题品牌堆砌、品牌标属不一致、图文不一的侵权商品是权利人关注的重点，被称为商品的霾；
- 未按淘宝规范要求及广告法发布的滥发商品，违规类型多、违规手段层出不穷，且商品量大。

传统的风险治理手段主要有两种：

1. 通过商品管理，将违规商品下架或删除，对卖家进行罚分、关店等，使用条件为确定性风险，投诉成立率低，而且还要考虑客满压力；

2. 流量处罚，治理手段主要为在搜索端对商品直接屏蔽或减固定的分数。优点是快速高效，但缺点为不考虑大盘情况，流量调节的粒度不够细致。

因此，我们亟需一套精准流量调控机制，在大盘稳定的前提下，有效降低风险商品的流量，提升平台洁净度。

7.1.2 为什么使用强化学习调控

在风险流量调控项目 1.0 中，我们基于人工设定的权重对风险商品进行调控，在一定程度上实现了风险商品流量调控的目的，但也存在一些明显问题：

1. 基于大盘整体的表现确定的全局权重，无法实现更细粒度的流量调控。不同风险状态下的 query，流量调控的权重也是相同的，其流量调控效果显然大打折扣；

2. 降权权重是固定的，无法随着环境的变化而动态调整。

因此，项目目标是采用更加智能的算法，将流量调控的粒度从全局细化到 query，并实现实时动态的权重寻优。最终在大盘稳定的前提下，获取更好的风险流量调控效果。建模思路：已知 query 当前的状态，选择一组排序 feature 的权重，能够最大程度的降低当前 query 下的风险商品流量，同时平衡平台收益。这是一个典型的序列决策问题，非常适合用强化学习的框架来求解。

7.2 基于强化学习的问题建模

7.2.1 状态空间的定义

我们的目标是在大盘稳定的前提下降低风险商品的流量，因此 state 的定义分为 query 的正向表现 (CTR/CVR/...) 以及 query 的负向风险程度，包含了 query 的离线特征和实时特征。

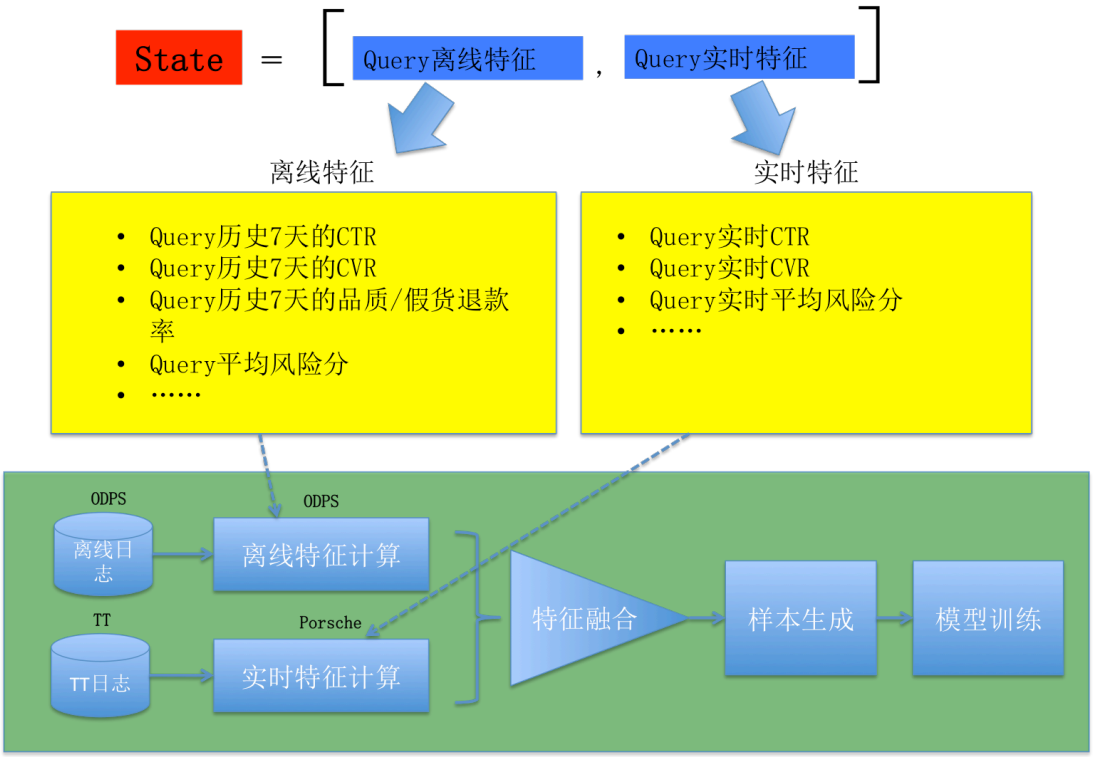


图 7.1: 状态的定义及计算流程

其中，query 的离线特征用于刻画 query 的长期特性，比如：历史 7 天的 CTR、CVR 等，可以反映 query 历史的成交效率高低；历史 7 天的品质退款率、假货退款率可以反映 query 下历史退款风险的大小。除了离线特征以外，我们还考虑了 query 的实时特征，这能让我们对 query 的短期状态变化进行快速感知，对环境变化做出及时的响应，包括 query 的实时风险分、实时 CTR、实时 CVR 等。在我们的定义中，query 的实时特征是通过对用户在 query 下长度为 T 的时间段内的行为数据累积的结果，之所以这样做是因为我们用到了 query 的 CTR、CVR 这样的特征，累积长度为 T 的时间可以有效地降低特征计算的偏差，在我们的实现中 T 取 15 分钟。

7.2.2 动作空间的定义

Action 定义为一组 rank feature 的权重向量，每一维代表一个 rank feature 的权重大小。我们使用的 rank feature 包括三个：

1. 商品的风险分；
2. 商品是否为高风险商品；
3. 商品的 GMV 分。

特征虽然不多，但却是我们实现调控的有力抓手。商品的风险分和是否高风险商品可以帮助我们有效调节风险商品的曝光程度，GMV 分可以帮助我们在降低风险流量的同时，平衡大盘的正向收益，其中 GMV 分计算时融合商品的炒信风险，确保正向的纯净。

7.2.3 奖赏函数的定义

奖赏函数设计时，同样会兼顾正负向收益，定义如下：
正向 reward：

$$R_p = \sum_i \beta_1 \text{clk}_i + \beta_2 \times \text{price}_i \times \text{ord}_i \tag{7.1}$$

负向 reward：

$$R_n = -\sum_i \text{rscore}_i \times (\gamma_1 \text{pv}_i + \gamma_2 \text{clk}_i + \gamma_3 \text{ord}_i) \times 1.5^{y_i} \tag{7.2}$$

总收益为 $R = \alpha_1 R_p + \alpha_2 R_n$
其中：

1. R_p 为正向收益，目的是平衡大盘的正向指标 (GMV、CTR)，主要包括两个部分，分别为点击和成交对应的正向收益，当用户点击或者购买商品时会产生对应的正向 reward；

2. R_n 为负向收益，目的是引导搜索排序尽可能的降低 query 下的风险商品流量，由三个部分组成，用户在 query 下的每次曝光、点击和成交都会根据对应商品的风险大小进行加权，进而计算得出负向 reward；

3. ord、clk、pv 分别表示 query 下的商品是否有成交、点击、展现，y 表示商品是否为高风险的商品，当一个商品为高风险商品时，我们会对其负向 reward 加权，引导搜索排序的结果尽可能的少展示这样的高风险商品，进而降低风险商品流量；

4. rscore 表示商品本身的风险分，α、β、γ 为调节因子，可以调节总收益中不同部分的重要程度，比如正向 reward 和负向 reward 的相对重要程度，展现、点击、成交的相对重要度等。

7.2.4 模型选择

在模型选择的问题上，我们调研了多种不同的算法：

1. 基于值函数的 Q-Learning、DQN 等，这些方法适用于离散动作空间，而我们场景下的 action 为 rank feature 的权重值，是一个连续变化的量，所以这些算法不适合于我们的问题，而且 DQN 算法在实际训练时也有收敛慢的缺点；

2. 经典的 Policy Gradient 算法虽然适用于连续动作输出的场景，但缺点是训练的过程太慢，因为算法必须在每一轮 Episode 结束后才能进行梯度的估计和策略的更新；

3. Actor-Critic 算法通过引入 critic 网络对每一步的 action 进行评价解决了必须在 Episode 结束后才能更新策略的问题，算法可以通过 step by step 的方式进行更新，但该算法的缺点是由于使用连续的样本更新模型，样本之间的相关性强而影响模型的收敛性；

4. Google DeepMind 团队把在 DQN 训练中取得成功的 Experience Replay 机制和 TargetNetwork 两个组件引入了 Actor-Critic 算法，极大的提高了模型训练的稳定性和收敛性，在很多复杂的连续动作控制任务上取得了非常好的效果。

我们的场景属于连续动作空间问题，所以我们最终选择了性能较好的 DDPG 模型作为我们方案的核心算法。整体的网络结构如下图所示：

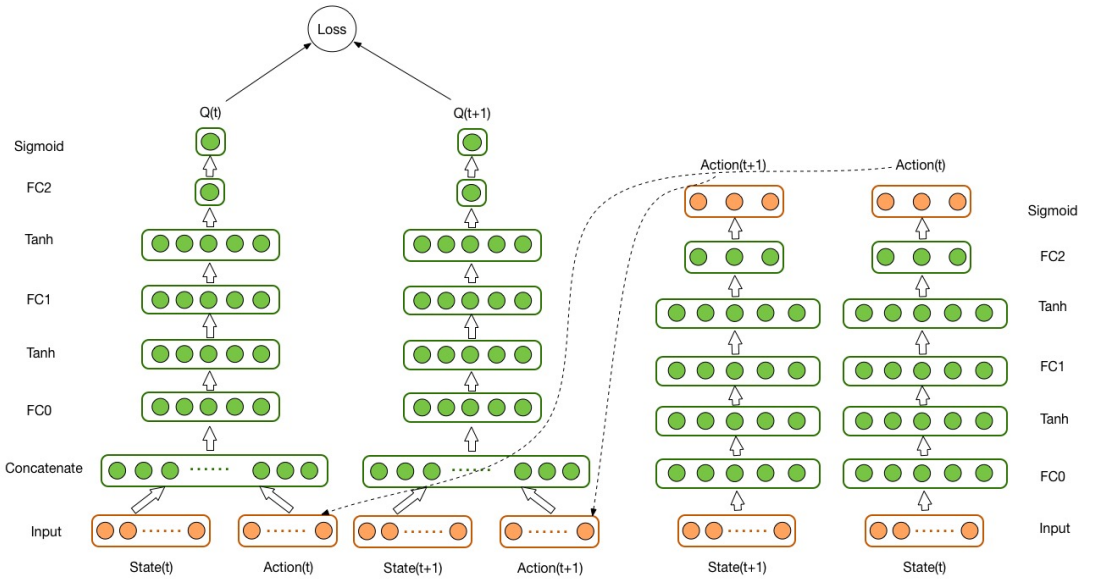


图 7.2: 模型网络结构

7.2.5　奖赏函数 scale

在实际的训练过程中我们发现，由于 reward 考虑了正负两个方面，正向 reward 的取值范围远大于负向 reward(因为正向的成交 reward 考虑了价格因素)，导致负向的风险因素几乎不起作用，为了解决这个问题，我们对原始的 reward 计算结果进行了 scale 的后处理，使正负 reward 具备可比较性。具体地，我们使

用 Ln 变换对 reward 进行了后处理，处理之后正负向 reward 的分布区间变的更具可比性。

正向 reward 的 scale：

$$R'_p = \frac{\alpha_1 \log(1 + R_p)}{\alpha_1 + \alpha_2} \tag{7.3}$$

负向 reward 的 scale：

$$R'_n = -\frac{\alpha_2 \log(1 - R_n)}{\alpha_1 + \alpha_2} \tag{7.4}$$

scale 后的总收益为 scale 后的正向 reward 和负向 reward 的和，即 $R' = R'_p + R'_n$。

7.3 流量调控系统架构

我们基于搜索工程同学研发的 Porsche 平台进行 DDPG 模型的训练、query 实时特征的计算、训练样本的生成等任务，训练后的模型定时推送到 IGpraph 上存储，虽然模型是实时训练的，但在具体实现时我们是每隔 10 分钟推送一次新模型，避免频繁写对 IGraph 造成压力。QP 负责读取模型，并根据当前 query 的特征预测 rank feature 的权重，rank 取到权重后计算最后的 rank 分，影响线上排序效果。

7.4 线上效果

双十一当天，在保证大盘稳定的前提下 (GMV 不降低，客单价不降低)，平台的假货退款率和品质退款率显著降低，高风险商品的流量和成交额显著下降。

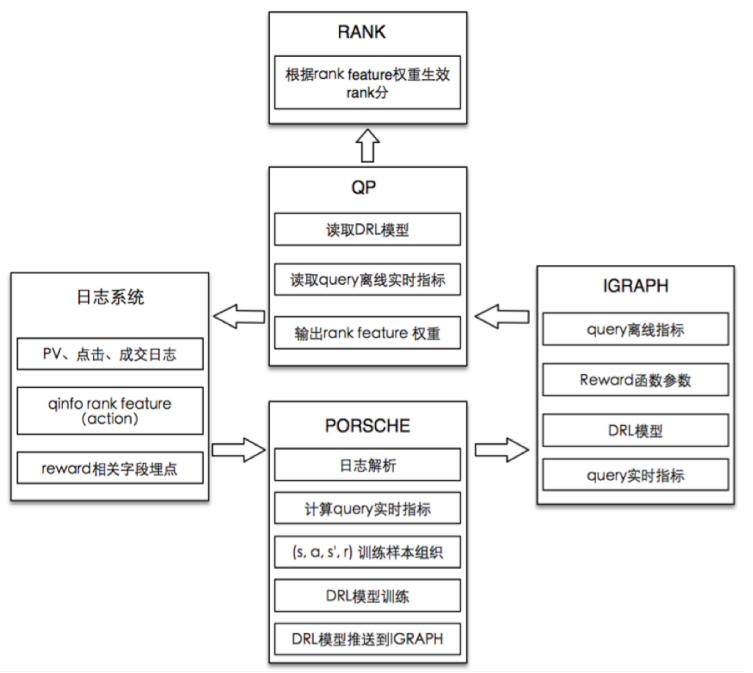


图 7.3: 强化学习流量调控系统整体架构

第八章　虚拟淘宝

8.1　背景

8.1.1　强化学习面临的问题

在某些场景下中应用强化学习（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。

8.1.2　虚拟淘宝

8.2　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。

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，

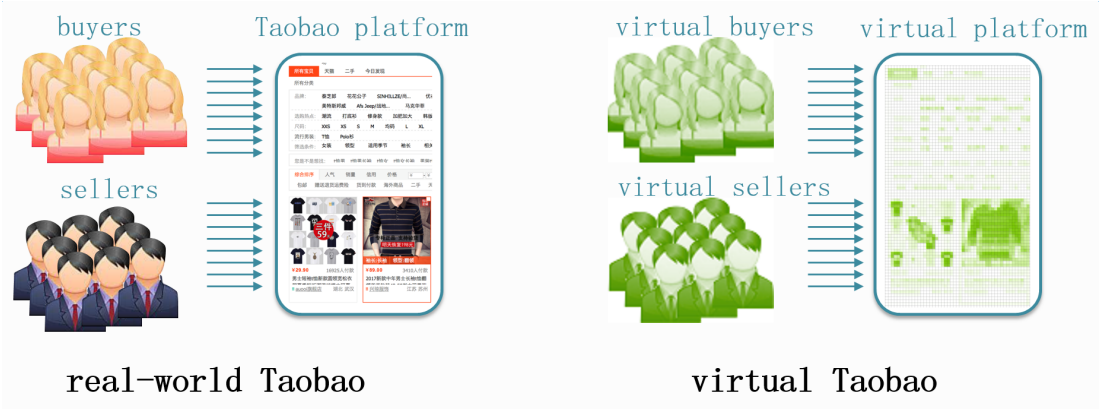


图 8.1: 真实淘宝和虚拟淘宝

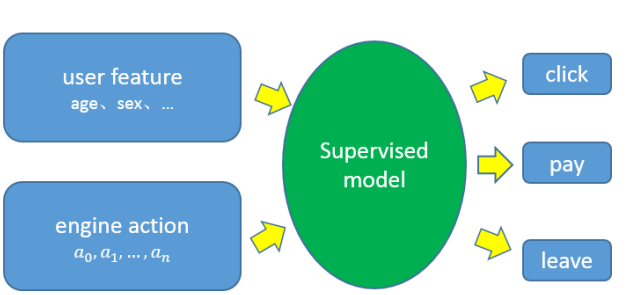


图 8.2: 监督学习方案

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。

8.3　学习用户意图：逆强化学习

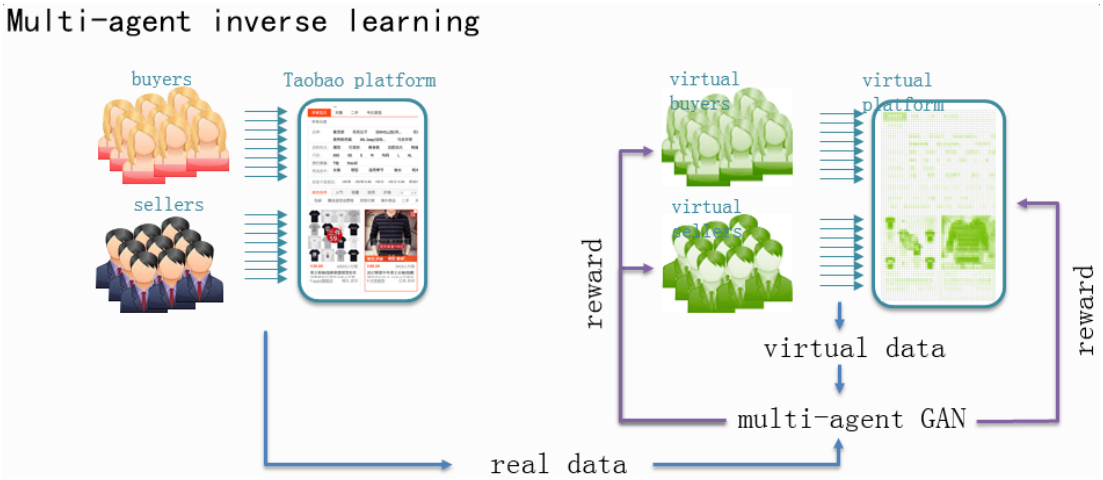


图 8.3: 多智能体逆强化学习

8.3.1 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。

8.3.2 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）$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbf{R}$ 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。

8.3.3 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]。

8.4 构建用户行为模拟器

8.4.1 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 $(x_1(d), x_2(d), ..., x_n(d))$，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）$\mathcal{M} =< S, A, P, R, \gamma >$

- **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** π　定义参数化策略 $\pi_\theta : \mathcal{S} \to \mathcal{A}$。

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 $\mathcal{M}_e =< S_e, A_e, P_e, R_e, \gamma_e >$ 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 $< u, w >$ 是否来自真实数据，用 D_w 的输出作为奖励。
- **策略** π_e 定义参数化策略 $\pi_{e\theta} : s \to a$。

模拟器框架如图 [8.4]：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。

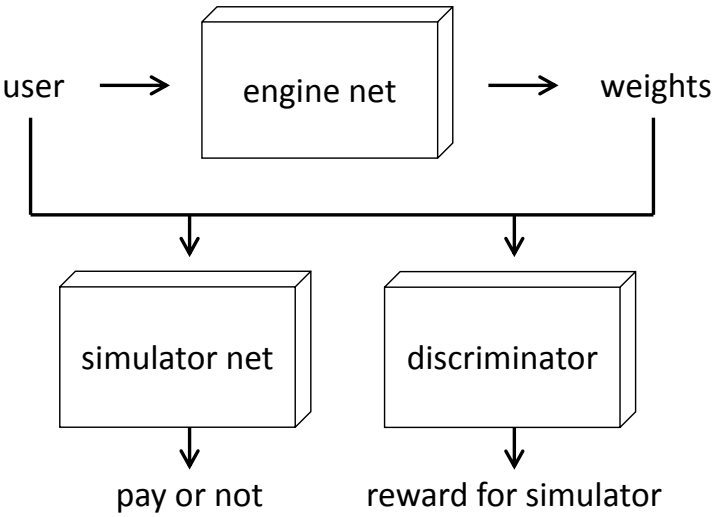


图 8.4: 模拟器网络结构

8.4.2　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：

- 初始化引擎 π 和模拟器策略 π_e 以及判别器函数 D_w。
- 循环执行，直到终止条件。
  - 用 π_e 和 π 采样出一系列轨迹 τ_i。
  - 用真实轨迹 τ 以及采样出的轨迹 τ_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 π_e，π。

Algorithm 1 LERD

Input: Expert trajectories τ, initial policy $\pi_\theta, \pi_{e\theta}$ and discriminator D_w.
for $i = 0, 1, 2...$ **do**
    Sample trajectories τ_i by $\pi_\theta, \pi_{e\theta}$
    Update discriminator D_w with the gradient.

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

    Update policy π_θ, $\pi_{e\theta}$ using some RL method with reward function D_w.
end for

图 8.5: LERD 算法

8.4.3 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。

第九章　组合优化视角下基于强化学习的精准定向广告 OCPC 业务优化

9.1　背景

在精准定向单品（按点击扣费）广告业务中，广告主会为广告设置一个固定出价，作用在指定的场景和定向类型下。如果广告主能够根据每一条流量的价值进行单独出价，可以带来两点好处

1. 广告主可以在各自的高价值（如点击、成交）流量上提高出价，而在普通流量上降低出价，如此容易获得较好的 ROI（投资回报率）；

2. 流量细分后，平台能够提升广告与访客间的匹配效率，体现为 CTR（点击率）、GMV （成交总额）等用户指标提升，而在大盘 ROI 不变的情况下，商业指标 RPM（千次展现收入）也能相应提升。

在单品广告业务里，广告主无法对单流量价值进行评估并实时出价，所以这个根据价值预估做智能调价的担子自然落到了广告平台方的肩膀上。2016 年起，我们在智能调价方面进行了深入的技术探索，相关成果 [39] 已发表。

本文是在已有的智能调价系统基础上的改进工作。智能调价系统的目标可以简单概括为：**在保障单广告主 ROI 的约束下，提升大盘 GMV、RPM**。我们认为，原系统在达成这个目标的解法上有以下几点可以持续改进

1. GMV、RPM 两指标的优化被分成单独的阶段，二者串联依赖于后阶段调整尽量保证前阶段排序不变，会带来效果损失；

2. 模型需要手动调参，限制了模型的复杂度和迭代效率；

3. 离线调参的评价标准是预估值，而非真实效果，所以离线最优参数很可能不是在线最优。

总结以上，我们本次改进的目标是，将之前的基于预估值反馈的、多目标分离优化的离线学习，变为基于线上真实效果反馈的、多目标联合优化的在线实时学习。强化学习技术显然和我们的需求很匹配。

9.2 问题建模

9.2.1 奖赏

背景中提到，智能调价系统的目标是在保障单广告主 ROI 约束的前提下，提升大盘 GMV、RPM。关于 ROI 约束，目前我们通过在单个流量上根据 CVR（预估转化率）和 HCVR（历史转化率）计算调价上界（公式 9.1）的方法来保障。如此，我们问题中的奖赏就是 RPM、GMV 两目标。

$$ocpc\_bid < \frac{cvr}{hcvr} bid \tag{9.1}$$

9.2.2 动作

智能调价系统中的动作是每个广告的调价比例，更确切地说是本次参竞广告的调价比例构成的向量。

图 9.1 展示了我们对于 OCPC 问题的理解。如果仅考虑优化 RPM、GMV，我们可以先将每个广告调价到 ROI 所允许的最高，然后为每个广告预估其展现奖赏 $RPM + \alpha GMV$，最后按照奖赏从高到低排序选出 a、b、c 三个广告进行

展现。然而，这种做法打破了广告按照 ECPM 排序的商业逻辑，也削弱了 OCPC 做调价的意义。

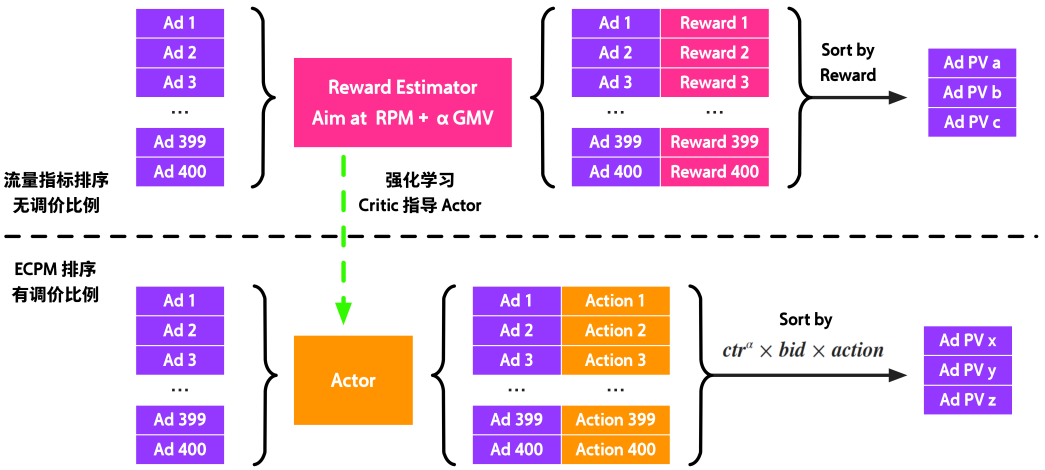


图 9.1: 以调价比例为动作的 OCPC 问题

在 OCPC 业务中，**我们的动作被限定为调价比例，排序公式被限定为** $ctr^{\alpha} \times bid \times action$，但流量优化目标仍旧是 $RPM + \alpha GMV$，所以我们要尽可能保证调价后按 ECPM 排序的 Top 3 广告 x、y、z 正是 a、b、c。在强化学习建模中，我们希望通过奖赏预估（Critic）指导动作生成网络（Actor）在每个广告上输出适当的调价比例达成两种排序间的统一。

9.2.3　状态定义

在强化学习中，状态首先要能够决策动作继而优化（长期）奖赏，其次得是合理转移着的。基于这两点，我们认为本次流量广告**候选集的全部打分**是我们需要的一类重要状态。接下来我们将分别说明

1. 候选集信息是必要的；
2. 候选集信息主要指各种打分；
3. 候选集打分是具有自然转移的用户状态。

候选集信息必要

在机器翻译任务中，我们使用 LSTM[12] 模型建模句子已生成部分的隐状态（hidden state），据此从全量词库中挑选一个词作为下个输出。在这个任务里，我们每次选词的候选集是固定的，即全量词库，所以我们并没有将全量词库信息建模到状态里。

然而，在电商的搜索、推荐和广告业务中，出于效率的考量，我们会首先通过召回系统将本次选品范围限定在比全量商品/广告库小很多的一个候选集上，然后在这个候选集上应用排序算法或 OCPC。**用户请求不同，候选集也在变，而正是因为候选集在变，我们在建模时就必须考虑候选集信息**。举一个简单的例子，要想估计不同人的通勤时间，收入水平、居住地点等都只是一般化的特征，如果我们能确切地知道每个人选择的交通方式，那这个信息显然可以帮助我们做更为精准的通勤时间预估。

候选集信息主要指各种打分

在电商任务中，候选集信息主要是候选集的全部打分。为说明这一点，我们不妨先把问题设定在最理想的环境下，有如下几点假设。

1. 强化学习中的折扣系数为 0，单个流量最优化就是全流量最优化；
2. 优化目标相关的全部因素都具备，比如优化目标是 RPM，我们有每个广告的预估 CTR 和 BID；
3. 所有的预估值都是准确的，如 CTR 和 BID 完全准确；
4. 从因素到优化目标的建模是准确的，如输入三个广告的顺序和相应的预估 CTR、BID 值，建模能给你算出准确的 RPM 收益（甚至已经考虑了三个广告的相互影响）。

在如此理想的环境下，我们不需要引入除候选集的全部打分之外的任何信息，只要穷举广告三元组即可。

把假设条件稍微放松，如预估值或者优化目标建模有瑕疵，我们可以利用强化学习主动探索和对标真实奖赏的特性进行修正。

只有当一些假设严重失真的时候，我们才需要引入候选集的全部打分以外的信息，比如

1. 折扣系数大于 0，这意味着单个流量最优化并非全流量最优化，而候选集的全部打分只能帮你做到单个流量最优化，所以引入额外信息是必然的；

2. 优化目标相关因素不完备和部分预估值很不准确其实有一定的重叠，它们都要求引入额外信息修正用户的点击、购买估计。

以上表述想说明的道理也可以用一个简单的例子类比。

开学初，老师说期末考题都在教材范围以内，吃透教材就能得到满分。

后来老师说，教材内容有错误，吃透教材 90 分还是有的，想得满分要同时参考教材勘误表。

再后来老师又说，期末考题不限于教材范围，光看教材最多考 70 分，想得满分要另外参考一本国外教材。

候选集的全部打分其实就是教材，教材（候选集的全部打分）是考试（决策）考高分（获得最优奖赏）的基础，其他资料（如用户最近的行为偏好）是教材的纠正或补充。

候选集打分是自然转移的状态

我们已经说明候选集信息必须引入，且候选集全部打分就是我们用于决策进而优化奖赏的一类重要信息，那么它是一个转移的状态么？答案是肯定的。

候选集的商品构成是召回系统根据用户最近状态挑选出来的，候选集的打分是排序模块根据用户最近的状态和广告自身信息计算出来的，所以候选集的全部打分完全可以被视为对当前用户状态的一种凝练。用户相邻两次到访业务场景，候选集的构成和打分都会发生变化，这就是一个自然的转移过程。

以上，我们主要从逻辑分析角度解释了我们对状态的选择，其实还可以从建模角度理解。我们可以将在动作网络中使用预估值作为状态类比成使用没有 End2End 训练的 Embedding，如图 9.2 所示。

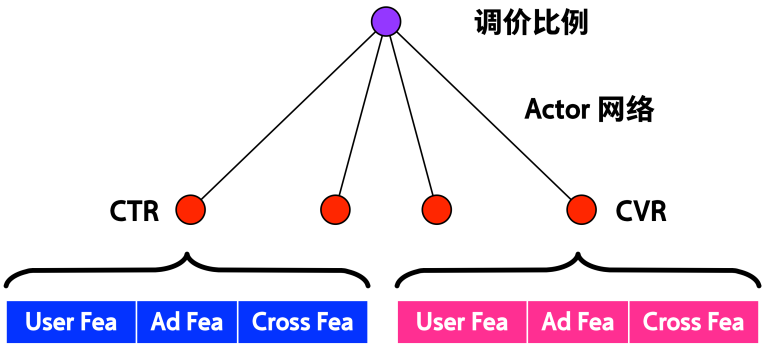


图 9.2: 预估值是动作网络的 Embedding 输入

9.3 建模粒度

在建模粒度这个问题上，我们关心的是，应该将候选集的全部打分整体输入一个网络，还是将每个广告的打分信息分别输入一个共享参数的网络。我们称前者为 session 粒度建模，称后者为 ad 粒度建模。

这个问题我们并不陌生，ad 粒度建模就是我们在 CTR 预估中的一般做法。那么，为什么我们几乎没有见到过 session 粒度建模的预估方案呢？一个重要原因是，用监督学习直接建模组合优化问题有很大的难度。

所谓组合优化问题，是指我们在电商环境下会面对的这样一类问题：**如何从一个大的候选集合中挑选一个子集，按照一定的顺序展现出来，实现流量指标（如 RPM、GMV）最优**。独立假设下的 ad 粒度预估排序是解决这类问题的一种高效、近似的手段。

基于上述认识，我们接下来从强化学习的动作生成和值函数预估角度思考这两种建模粒度。

ad 粒度建模中，动作是每个 ad 的调价比例，和 CTR 模型输出预估值类似。

session 粒度建模中，动作的含义应该是**某种区分方式**，这种方式能够将 3 个最优广告和其他 397 个广告分开。这类动作包括但不限于以下几种可能：

1. 直接输出哪 3 个是最优广告；
2. 输出 400 维奖赏预估值向量，排序取最优 3 个；
3. 输出 400 维调价比例向量，用调价比例和统一的公式给 400 个广告打分，排序取最优 3 个；
4. 输出一个权重，用这个权重和统一的公式给 400 个广告打分，排序取最优 3 个，如图 9.3。

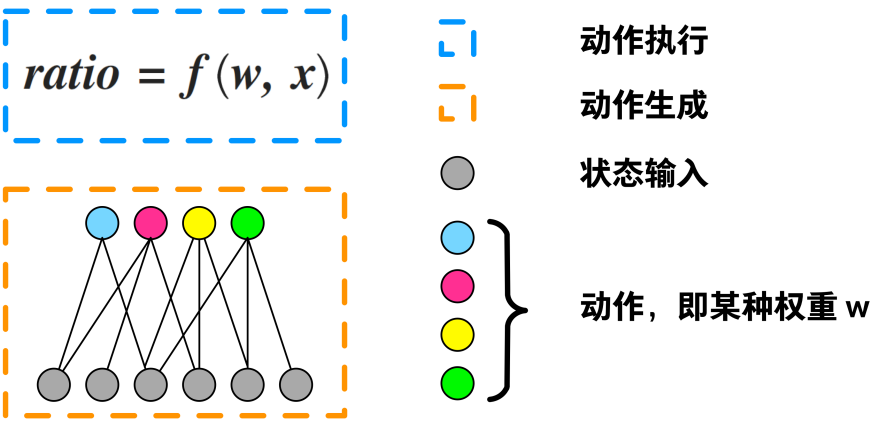


图 9.3: session 粒度建模举例

这四种动作网络的建模难度是一样大的，而第一种网络显然是想用监督学习直接建模组合优化问题。值函数预估上的分析与此类似，考虑到 session 粒度值函数预估实际上要完成两个任务：400 选 3 和 3 广告奖赏的预估。

除建模难度外，session 粒度建模还会导致动作生成（图 9.3 下半部橙色框）和动作执行（图 9.3 上半部蓝色框）两部分在优化上的耦合。强化学习并不会训练 f 函数，而 f 函数自身也有优化空间，如果我们调整了 f 或者 x，那么动作生成部分的状态输入或网络设计也很可能要随之变化。

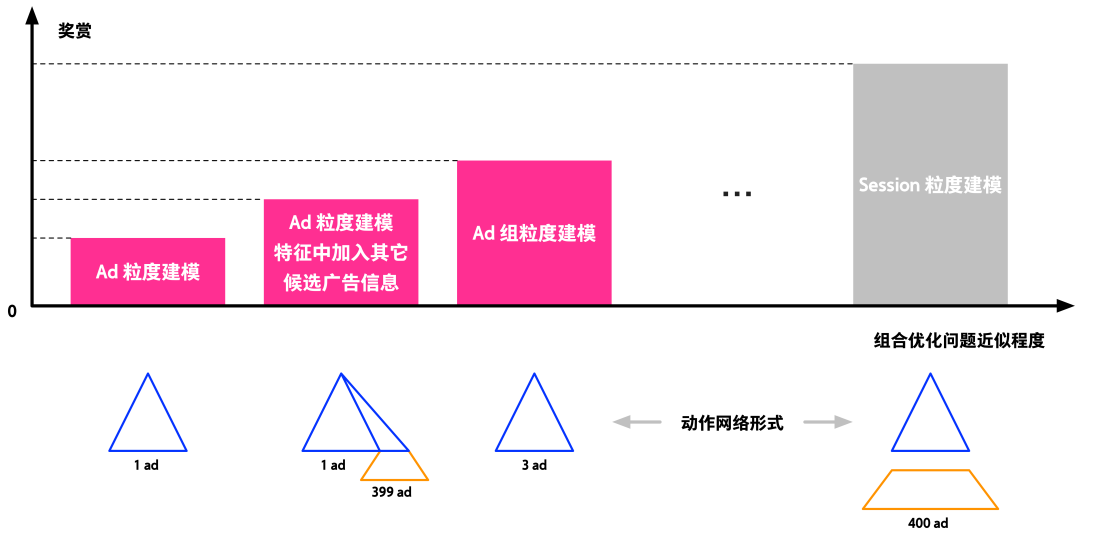


图 9.4: 组合优化问题的可持续迭代方法论

总的来说，**我们面对的问题本质上是组合优化问题，组合优化问题自身的难度不会随着建模形式的变化而自动消除，它必然栖身于我们系统的某个环节中**。session 粒度建模是将组合优化问题的难度放到了监督学习建模（值函数预估与动作生成）里，而 ad 粒度建模则将其放到了单广告调价（独立假设，近似组合优化）与强化学习探索（探索组合优化解空间）中。

其实在 ad 粒度和 session 粒度之间还有一个 ad 组合粒度，Group CTR 之类的任务就是尝试在组合粒度上做预估，以弥补独立预估假设的不足。无论是 ad 粒度，还是 ad 组合粒度，本质上都是从监督学习中剥离组合优化的难度，只是两者对组合优化问题的近似程度不同。图 9.4 展示了在组合优化问题上的一种可持续迭代的方法论：**一方面不断接近现有近似建模方案的效果天花板，另一方面让建模更接近原问题的组合优化本质**。

最后，考虑到我们的动作是每个 ad 的调价比例，特别是最终要为每个 ad 单独计算 ECPM 进行排序，所以我们选择 ad 粒度建模。

9.4　模型选择

模型选择方面，我们主要考虑三类方法：

1. 值函数估计方法，如 DQN[26]；
2. Likelihood Ratio 策略梯度方法，如 Actor-Critic[17]；
3. Pathwise Derivative 策略梯度方法，如 DDPG[22]。

DQN

DQN 是一种最直观的，将现有监督学习预估方法和强化学习结合，解决组合优化问题的方式，图 9.1 的上半部分其实就是 DQN 方案。在这种方案中

1. 强化学习的探索为监督学习解决训练样本探索不充分的问题；
2. 监督学习提供的预估 CTR、CVR 等可以帮助强化学习做更高效率的探索；
3. 强化学习的值函数预估对标真实奖赏；
4. 监督学习提供的预估 CTR、CVR 等可以作为强化学习值函数的输入。

DQN 方案可以应用于一般的搜索、推荐场景，但它并没有显式建模动作。我们在 9.2.2 一节提到，从广告的商业逻辑和 OCPC 自身的产品定位看，显式定义一个 Actor 网络是更合理的，所以我们重点考察 DDPG 和 Actor-Critic。

Actor-Critic 与 DDPG

Actor-Critic 和 DDPG 的最大差异已经体现在所属方法的命名中。Actor-Critic 属于 Likelihood Ratio 方法，这种方法让 Actor 学习好动作的经验、吸取差动作的教训，这些动作都是真实发生过的，值函数预估主要起到调节动作样本权重的作用。DDPG 属于 Pathwise Derivative 方法，这种方法 Actor 的监督信息来自值函数的梯度，真实历史动作通过服务于值函数的学习间接影响 Actor 网络。

在 OCPC 任务中，我们较为倾向于使用 Actor-Critic 方法，原因有二：

1. Actor-Critic 方法中的值函数预估主要起到调权的作用，所以我们可以做 ad 粒度建模，只对真实展现出来的广告组合做奖赏估计，然而 DDPG 里要用到值函数对动作的梯度，也就是要求做 session 粒度建模，我们已知这是在直接建模组合优化问题；

2. 奖赏预估，特别是其中的 GMV 预估尚不能保障预估值的精准性，用它做广告组合的优劣判别进而调整权重是可以的，但用它的梯度去指导 Actor 的训练可能会带来误差的传播。

9.5 探索学习

我们已经对奖赏、动作、状态、建模粒度和模型选择进行了深入讨论，接下来我们需要思考的问题只剩下三个：

1. 如何做有效的探索；

2. 如何评价每个探索；

3. 如何把好的探索记录下来。

问题 2 其实就是值函数（Q、V）预估，ad 粒度建模下相对简单，我们重点关注问题 1、3。

探索可以天马行空，我们可以任意扰动每个广告的调价比例，以改变展现出来的 3 个广告，关键是如何指导 Actor 网络把好的探索记下来。

探索方式主要有两种，**动作空间扰动**和**参数空间扰动**。前者指在 Actor 网络输出的动作上加噪声，后者指在 Actor 网络的参数上加噪声，使动作输出发生变化。在用 Actor-Critic 建模 OCPC 问题的前提下，动作空间扰动会有三方面不足：

1. 动作空间扰动要学习一个好的动作，不仅要包括每个展现广告的调价比例，还要包括会对本次展现广告构成威胁的那些广告的当前调价比例，但即便你把这些广告及相应的调价比例都送给 Actor 网络学习，也很可能无法保证这次好的动作探索能被复现；

2. 给 Actor 网络设定的动作标签并不稳定，因为未展现广告的调价比例只要小于等于当前比例就一定不会影响本次展现的广告，那么动作标签应该如何设定呢（一种可能的解法是将广告调价限制为二值，即要么调到最高，要么调到最低，这么做和 DQN 方案区别不大）；

3. 动作空间扰动并不考虑扰动后的动作是否可由当前状态和 Actor 网络结构学习到，更增大了无法复现本次动作的可能性。

参数空间扰动的好处与此相对：

1. 能够完整地复现本次动作，这个特性有助于记忆广告二价率；

2. 梯度来源不是动作标签的反向传播，而是参数的扰动量；

3. 能复现本次动作的参数是确定性存在的。

总的来说，二者的区别在于梯度来源不同，动作空间扰动靠标签的反向传播获得梯度，而参数空间扰动的梯度就是扰动量本身。参数空间扰动也有其自身局限性，可以说是**以牺牲一定程度的探索与学习的灵活性为代价，换得动作学习的稳定性**。

1. 探索灵活性：OCPC 中每个广告可在一定区间内任意调价的灵活性增加了动作学习的难度，参数空间扰动把这种难度消除了；

2. 学习灵活性：当无法给监督学习指定稳定可靠的标签供其做反向传播时，我们只能使用遗传算法类的优化方法。

9.6　业务实战

9.6.1　系统设计

目前，我们强化学习系统的总体设计如图 9.5 所示，这是我们在探索实践中不断迭代完善的结果。绿色虚线标出的是离线模拟流程，我们主要用来检验算

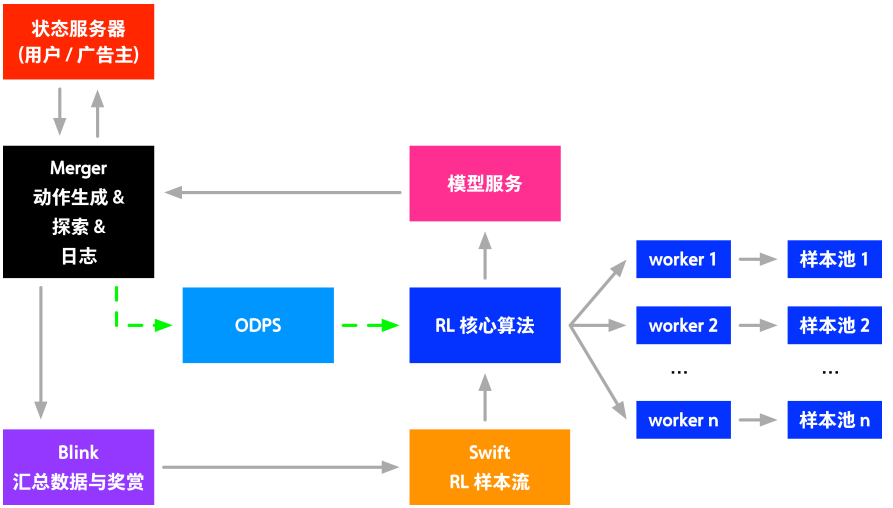


图 9.5: 在离线强化学习系统设计

法收敛性，定性测量状态输入或网络设计的有效性。灰色实线标出的是在线训练、服务流程，主要分以下几部分：

1. OCPC 主逻辑，负责动作生成与探索，并记录日志；
2. 状态服务，提供用户状态做流量端优化，提供广告主状态做广告主端优化；
3. 实时样本生成，Blink 分布式流处理做展现日志与奖赏信息的汇总，输出到 Swift 供下游算法逻辑订阅；
4. 分布式强化学习算法，执行 Actor-Critic 学习逻辑；
5. 模型服务，实时模型导出，供 OCPC 访问。

2017 年双十一、双十二期间，我们系统中的 RL 核心算法尚不是分布式 Actor-Critic，而是一种进化策略方法 CEM（Cross Entropy Method）[30]。在我们看来，CEM 是满足上文所有建模思考的一种极简实现方案，有以下一些优势：

1. CEM 通过参数空间扰动学习 Actor 网络，将蒙特卡洛采样真实奖赏替换成值函数预估，就构成了基本的 Actor-Critic 结构；

2. CEM 在单参数扰动上的奖赏聚合逻辑有利于降低奖赏的方差，挑选高奖赏（Elite）部分更新参数的做法其实是通过相互比较去判断探索的好坏，而 Actor-Critic 则是用基线（Baseline）或优势函数（Advantage Function）做到这一点；

3. 离线模拟显示，CEM 在我们的数据上能稳定收敛，目标值近乎单调上升；

4. CEM 工程实现代价低，其轻量特性便于快速迁移推广到其他需要策略学习的场景；

5. CEM 逻辑简单，可解释性强，有助于我们前期快速验证状态输入、网络结构及奖赏设计的有效性，**在 CEM 中无效的状态、网络或奖赏在更复杂模型中也大可能是发挥不出作用的**。

9.6.2　奖赏设计

我们要联合优化流量 RPM 与 GMV，但它们并不是完全独立的，甚至优化其中一个目标会对另一个目标带来负面影响。绘制出帕累托曲面做多目标决策目前还只能在离线模拟中完成，在线做法上，我们还是使用简单加权目标作为奖赏。这当中要考虑到两目标量纲、量级和数据分布的不同，因应用场景而异，就不详细展开了。下面我们分别介绍在 RPM、GMV 上的处理考量。

二阶扣费逻辑下，RPM 很难在单广告粒度上预估准确，考虑到二价率的学习，以及场景下较高的点击率，我们决定使用真实反馈做 RPM 奖赏。然而，分析 CEM 训练过程我们发现，一些参数扰动点的 RPM 奖赏之所以高，是因为它们的广告候选集整体出价就很高，被状态中的 BID 特征捕捉到，所以在线 RPM 提升主要是 PPC（单次点击扣费）提升带来的。我们希望系统性地消除不同广告候选集的出价偏差，于是提出以公式 9.2 作为奖赏，该式表达的是**系统在单位出价上的营收能力**。

$$\frac{\sum_{click} cost}{\sum_{pv} bid} \tag{9.2}$$

在我们的业务场景下，GMV 奖赏很稀疏，而且因为笔单价不同，奖赏值的波动还很大。不仅如此，GMV 奖赏的延迟时间还很久，也即用户从点击到购买需要较长时间。针对稀疏、波动和延迟的性质，我们有必要使用值函数预估的方法对 GMV 做估计。双十一期间，我们使用的是辅助 GMV 奖赏的方法，其实就是一种简单的值函数预估方法。具体做法是，我们线上有一个 GMV 最优策略桶，智能调价桶会先用这个 GMV 最优策略给自己的广告候选集排序，把广告的序折算为 CEM 算法中的 GMV 奖赏。

9.6.3 实验效果

双十一预热期，我们进行了十天完整的实验，基准桶无强化学习方法。十天累积效果，GMV 持平，RPM 提升 11%，其中 CTR 提升 4%，PPC 提升由两部分带来，分别是（保自身 ROI 的情况下）提高出价和提升二价率。

双十一零点，我们以前一天参数为初值重新训练。全天效果，GMV 提升 6%，几乎全部由 CTR 提升带来，RPM 提升 11%。在之后的 11 月 12 日，这种提升效果仍旧能够保持。

双十二前，我们将动作网络从线性模型升级为神经网络，并在状态中加入了广告定向类型，考虑到一些定向的兴趣指向和实时性可以帮助我们更好的建模用户购买意愿。双十二期间，我们进行了十二天完整的实验，基准桶是双十一强化学习最优桶。十二天累积效果，RPM 提升 1%，GMV 提升 4%，其中 CTR、CVR 的贡献占五成，笔单价贡献了另外一半。

9.7 总结与展望

现如今，强化学习在电商领域乃至整个工业界的应用方兴未艾，我们在 OCPC 业务上的实践只是大浪潮中的一朵浪花，能够将自己的探索历程和思考撰写成文与诸君共享是我们的荣幸。

随着思考与实践的深入，我们的一种感觉越发强烈，那就是多想想强化学习能给我们现有解决方案带来什么，比直接思考如何在业务上做强化学习建模

更有效。**强化学习是一个通用的问题解决框架，核心思想是 Trial & Error，它不是能替代我们思考业务本质的黑魔法，而是在我们认清问题本质的前提下帮我们优化解法的工具**。

具体到 OCPC 业务上，我们认为动作受限下的组合优化是问题的本质，这个组合优化难度不因强化学习的引入而消除，而探索是强化学习提供给我们的解决组合优化问题的可持续迭代方案。此外，值函数预估支持我们直接对标长期终局指标，策略梯度定理允许我们将系统中难于建模的部分黑盒化。总而言之，**强化学习要素的引入让我们现有的独立预估方案在解法上更适配原始问题的组合优化属性**。

在具体的算法迭代中，我们一直是小步快跑，希望通过严谨细致的实验对比，理清每一小步的收益。我们坚信，只有充分理解现有做法的成败原因，才能明确未来迭代的提升方向。

最后，结合经验与思考，我们认为 OCPC 强化学习下一阶段的重点研究方向主要有以下几点：

1. 状态设计：进一步引入有效状态作为候选集全部打分的修正或补充，特别是引入广告主状态做全流量 ROI 优化；

2. 探索效率：结合启发式方法不断提升组合优化问题空间的探索效率；

3. 值函数预估：进一步提升奖赏预估精度，并提高值函数预估样本利用率。

第十章　策略优化方法在搜索广告排序和竞价机制中的应用

10.1　业务背景

搜索广告业务是阿里巴巴电商体系下的最为重要的一个业务，在创造整个集团大部分营收的同时，也承担着重要的生态调节功能，是帮助商家成长的“快车道”和“名校”。随着大数据和算法越来越深刻的影响业务的发展，技术已经成为搜索营销业务的核心驱动力。

搜索广告的竞价和排序遵循下面的业务流程：广告上在竞价词上定义自己的出价，对于每个广告位，搜索广告引擎根据广告质量（包括广告的点击率、转化率等）和广告主的出价对候选广告集合进行排序，排名第一位的广告获得当前广告位的展示机会。阿里巴巴搜索广告业务采用用户点击扣费的收费模式，即当有用户点击广告时，系统才对广告对应的广告主进行扣费。从整个业务流程来看，每一次搜索广告的展示都牵扯到了广告商、用户和平台三房的利益。对于广告商来说，借助搜索广告的流量获取作用，通过竞价来提升商品的曝光率，从而提高商品的销量；对于用户来说，用户在平台中搜索自己感兴趣的商品，希望平台能够提供更具个性化的推荐结果，从而提高浏览的效率和体验；而对搜索广告平台来说，希望在保证广告商诉求和用户体验的前提下，不断提高自己的收益。而同时最大化这三方利益的关键就是选择合适的广告和合适的扣费标准。在我们的场景下，我们通过优化排序公式的设计来达到这样的目的。一方

面，排序公式决定了哪个广告会最终被展示出来；另一方面，我们的搜索广告平台采用二价扣费机制（Generalized Second Price，GSP）进行点击扣费计算，这种计费方式按照保证展示广告能够维持自己排序位置的最低出价来对广告商扣费。也就是说，排序公式也决定了广告最后的收费标准。

由于用户在淘宝搜索之后的浏览过程可以看作是一种于平台连续的交互过程，我们提出一种策略优化算法（policy optimization）来优化不同搜索场景下的排序公式，该算法可以针对某一目标或者多目标的组合进行实时地调优。具体来说，我们将策略优化问题描述成一个强化学习问题，利用强化学习序列优化的能力进行策略的优化。在模型的学习方面，提出了基于仿真系统的 Agent 初始化方法和基于 Evolution Strategy 在线学习方法。

10.2　广告排序和竞价的数学模型和优化方法

如前文所述，排序公式将直接影响广告主、用户和平台的收益，对于广告主来说，能够获取展示机会是商家进行商品的推广重要前提；对用户而言，展示高质量的广告将有助于提升用户的满意度；而对平台来说，用户是否点击和点击之后的扣费将决定平台最终的收益，而用户和广告商的满意度也恰恰是维护平台长期收益的前提。因此，一个好的排序公式无疑对三方都有着重要的作用。那么我们应该怎么进行排序公式的优化呢？一方面，搜索广告的优化具有场景相关性的特点，搜索广告和原生搜索结果同时展示在搜索结果流里，两者互为上下文，广告是否能吸引用户的眼球，并且在风格和内容上保证用户体验是优化的目标之一。从排序公式的场景相关性来看，可以把排序公式的学习定义如下：

$$a = A(s) \tag{10.1}$$

其中，a 表示对排序公式的参数化描述，s 为当前请求的上下文环境，即 $s =< \text{User}, \text{Query}, \text{Ads} >$。$Ads$ 表示当前的候选广告集合。另一方面，搜索广告的优化具有全局选优而非单点最优的特点，从上面提到的用户和平台的序列交

互过程来看，我们希望优化的是广告展示序列的收益最大化而非单点的收益最优。也就是说，对于公式10.1，在每个单点场景 s 会获得一个“奖励“r_s，我们希望的是在序列交互过程中总的奖励之和（$\sum_s r_s$）最大。

将排序公式的学习抽象成一个模型后，想要进行优化我们还需要回答两个问题：1. 优化目标是什么？2. 哪些因素能够影响优化目标，也就是在目标优化的过程中有哪些抓手可以解决问题。

首先是优化目标问题，搜索广告核心任务是给公司带来盈利，因此一个直观的目标就是提高 RPM（revenue per thousand impressions，即平均千次广告展示获得的总广告商扣费）。而且手淘体系是一个完整购物链路，保持长期的效益离不开参与者（用户和广告主）的参与，因此搜索广告在不断提高收入的同时要兼顾用户和广告主的诉求，从指标角度来说，表示成不断提高 RPM 兼顾 CTR（click-through-rate，广告点击率），CVR（conversion rate，转化率或者用户购买率）和 GMV（gross merchandise volume，电商货品总销量）等指标。

有了目标之后还需要找到优化目标的抓手从而实现目标优化工作。首先是用户是否点击或者购买，只有点击平台才能获得收入，只有购买广告主的推广诉求才能被最直接地表达出来。如上文所述，搜索广告结果在展示页面是和自然搜索（主搜）的结果混合排列的（搜索广告在每一页有固定的展示位置），这就使得用户对广告的响应（点击或者购买）除了受到广告本身质量的影响以外，还受到自然搜索广告结果的影响，因此搜索广告的效果需要考虑自然搜索的结果。当然从业务流程来看，策略端在进行广告打分排序时是拿不到自然搜索结果的，但是自然搜索结果仍然是由用户的 Query 和本身的特点召回的，具有相似的上下文环境。此外，用户对搜索广告和自然搜索结果的不同响应也是我们比较 ad 和自然搜索结果的重要手段。也就是说模型 $a = A(s)$ 需要感知上线文环境并给根据上下文特点给出相应的排序参数 a。当用户愿意点击我们的广告时，我们还能做什么呢？一个最直接的问题就是扣费，扣费的多少直接影响平台的收入情况，在 GSP 的扣费计算公式下，相邻两个广告的排序分的紧凑程度会对扣费产生影响，在个性化广告召回的算法下，不同的用户，不同的上下文环境召回的广告集合都是不同，如果能够感知候选广告集合的分布，我们就有可能预估不同排序参数下的扣费情况，从而对扣费进行调节。也就是说感知候选广

告分布，预估扣费也将是我们实现目标优化的一个手段。此外我们还需要考虑的是，用户的浏览是一个过程，用户对前一次 pv 展示的响应必然会对后面浏览产生影响，因此一个完整的优化需要考虑整个浏览过程，对策略端来说是不是考虑到用户的历史行为就可以实现一个完整浏览过程优化呢？不是的，因为策略端在每一次进行广告打分排序时都是一个博弈的过程，这里可以做一些简化和类比。我们假定候选广告集合不变，用户连续的浏览一个个的广告，那么从策略端来看这件事情就变成了策略端每次从广告集合中无重复地拿出广告展示给用户，并按照这个广告和排在第二位的广告的紧密程度进行扣费。这个过程就像田忌赛马，并不是说每次拿出质量分最高的，就能获得最好的结果。从这个角度看，排序公式优化就像是一个博弈过程，需要全局的统筹规划。

上一段我们分析了排序优化的三个可能的抓手，那么应该建立一个什么样的模型才能让抓手动起来呢？模型上我们选择了强化学习，主要从以下几个方面考虑：1. 强化学习是对 MDP 过程进行建模的，这一点和我们面向浏览过程的优化目标一致；2. 强化学习尤其是深度强化学习的研究和发展，为我们在复杂场景下的策略优化提供了理论上的保证；3. 强化学习是面向综合收益最大化的优化 $(V(s_t) = r_t + \eta \cdot V(s_{t+1}))$ 这和我们希望的长期目标最优化是一致的；4. 强化学习的奖励函数的设计是灵活的，可以是连续的、离散的或者不可导的。这一点和我们的优化目标可以进行很好的融合，比如我们面向 RPM 和 CTR 进行优化，则可以将奖励函数设计为 $r = rpm + \lambda \cdot ctr$。

10.3　面向广告商、用户和平台收益的排序公式设计

未来使排序公式对于广告商、用户和平台收益具有调控能力，我们设计如下所示的排序公式

$$\phi(s,a,ad) = \underbrace{f_{a_1}(CTR)\cdot bid}_{\text{platform}} + a_2 \cdot \underbrace{f_{a_3}(CTR,CVR)}_{\text{user}} + a_4 \cdot \underbrace{f_{a_5}(CVR,price)}_{\text{advertiser}} \quad (10.2)$$

其中 $a = a_i\ (i = 1, ..., 5)$ 表示排序公式的参数，bid 表示用户对广告 ad 的出

价，$price$ 表示广告对应商品的价格，CTR,CVR 为系统预测点击概率和转化概率。排序公式中的 f_{a1} 可以认为是平台的收入的期望值；f_{a2} 考虑了用户的点击概率和转化概率，主要用于描述用户的满意程度；f_{a3} 考虑了与购买相关的因素，表示了广告主可能的收益。此外 a_2, a_4 用于调节后两个因素的平衡关系。我们用 ad, ad' 表示排在相邻的两个位置之间的广告，则根据 GSP 的扣费计算方式，可以计算当前点击扣费为

$$click\_price = \frac{\phi(s, a, ad') - \left(a_2 \cdot f_{a_3}(CTR, CVR) + a_4 \cdot f_{a_5}(CVR, price)\right)}{f_{a_1}(CTR)} \tag{10.3}$$

10.4 系统简介

现有文献中的强化学习方法大多应用于虚拟场景，特别是游戏场景。对于像广告这种场景的应用，需要考虑的一个重要的问题就是初始化和探索过程（explore）对于平台效果的影响。因此，我们设计了如下图所示的系统架构。系统主要由三个模块构成：离线搜索广告仿真模块，离线强化学习模块和在线策略优化模块。离线仿真模块主要用于仿真不同策略函数的参数的影响，如计算不同策略下候选广告的排序结果，可能的用户行为和扣费情况。仿真模块的使用可以使系统在离线的情况下充分探索可能的策略，同时不损害线上用户的真实体验。离线强化学习模块主要根据仿真模块产生的结果学习最优的离线策略，完成策略模型的初始化工作。当然离线仿真不可能代表线上的真实环境，需要我们根据线上的真实反馈来调节策略模型，这部分工作主要由在线策略优化模块来完成。

10.4.1 离线仿真模块

利用离线仿真模块一方面可以为强化学习进行环境探索产生大量的训练样本，从而保证强化学习算法的有效性，另一方面可以避免环境探索对线上真实

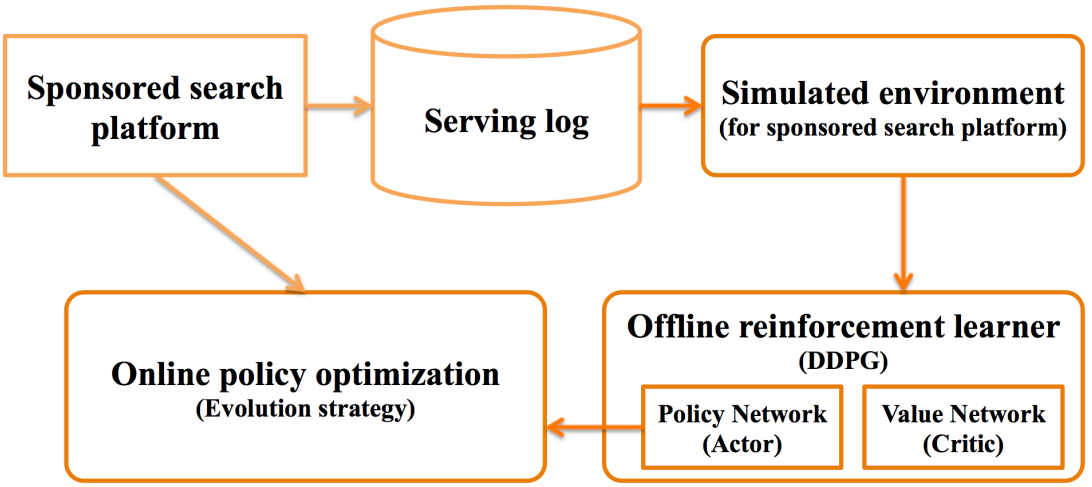


图 10.1: 策略优化系统框架

环境的影响，避免“不好”的策略对用户体验、平台收益的损失。排序结果会受到很多因素的影响，如广告主的预算，竞价价格的变动以及用户的偏好等因素，理想化的仿真系统相当于复制全套线上系统，而线上系统会受到广告商预算，用户分布等多维因素的影响，模拟难度很大。为了简化仿真系统，我们的仿真系统通过记录线上每次广告展示对应的上下文环境（user,query）和候选广告集合，在此基础上对不同策略函数参数进行仿真，得到新的用于展示的广告并计算相关的点击扣费 $click\_price$，并用 CTR，CVR 来预估用户的行为。在奖励函数的设计上，如果面向用户点击和平台收益进行优化，则 $reward$ 函数可以设计为：

$$r(s_t, a_t) = CTR \cdot click\_price + \delta \cdot CTR \tag{10.4}$$

其中，δ 是调节因子，用于调节点击率和扣费之间的平衡。

这里需要说明的是，算法预估的 $CTR(CVR)$ 和线上广告真实的点击率并不相等，因此为了使用仿真系统的用户响应估计更加贴近线上真实情况，系统会对 $CTR(CVR)$ 的结果进行标定 (Calibration), 并用标定的结果计算 $reward$。对于标定方法，使用 Isotonic regression method [1]。

10.4.2 离线强化学习进行排序策略模型初始化

在上面介绍的离线仿真模块基础上，我们定义强化学习相关的因素，包括状态 s, 动作 a 和奖励 $reward$ 以及状态转移 $s_t \rightarrow s_{t+1}$。对于状态 s，我们用用户请求时上下文作为状态的描述，可以包括 User,Query 和广告列表等相关的搜索上下文信息，包括用户的 profile 信息，用户的历史行为等。动作 a 即为排序函数中的参数 $a = \{a_i\}_{i=1}^{5}$。奖励函数由公式10.4定义的 $reward$ 的方式计算。我们假设用户在同一 query 下的浏览序列作为一次完成浏览过程，那么一个 Episode 就是用户从 page1 开始的浏览序列，状态之间的转移就是用户在不同 page 之前的迁移或者离开。

在强化学习模型的选择上，我们主要考虑两个因素，第一个是在个性化的广告系统中，前一页的展示结果和用户行为会对后续页的广告的打分（CTR,CVR）产生影响，这个是目前离线仿真系统无法仿真的（只能对每个展示机会相互独立的进行仿真），因此需要使用一种 off-policy 的强化学习模型；另一个是动作空间 $a \in A$ 是连续的，因此需要使用一种连续策略优化方法。考虑上述两个因素，我们使用 Deep Deterministic Policy Gradient (DDPG)[23] 模型进行离线强化学习。

DDPG 网络结构

本文使用的 DDPG 模型是基于 Actor-Critic 架构的，具体的网络结构如上图所示。因为输入的状态 s 特征都是 id 化的，因此所有的 id 特征会经过一个 embedding 层进行特征编码。激活函数选择 ELU 函数 [7]，实验中发现使用 Sigmoid 和 ReLU 作为激活函数时，当输出的动作 a 偏移中心位置较远时，产生的梯度值很小，收敛速度很慢或者不收敛。此外对于 critic 函数的输出，采用了 dueling architecture 的网络结构 [37]，即将输出的 $Q(s,a)$ 表示成 $V(s)+A(s,a)$，dueling 结构的使用可以使 critic 在学习过程中侧重能够获得更多奖励的动作。在实验中，由于数据的方差比较大，我们观察到 a 的更新可能会比较剧烈，为了保证输出的动作 a 在可控的范围，我们利用 clip method 方法对输出进行截断。

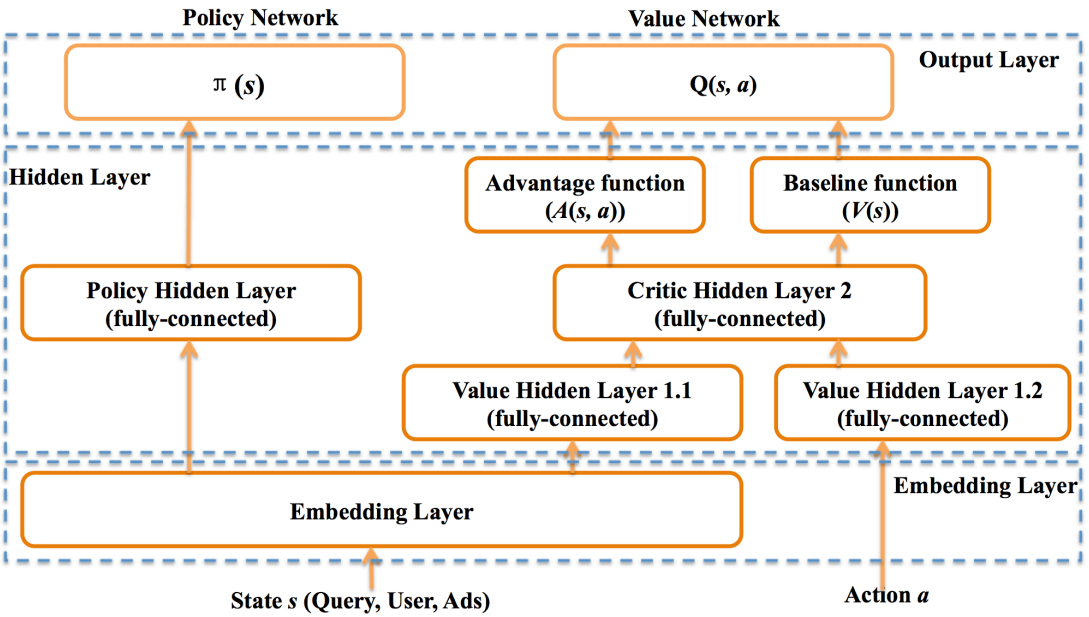


图 10.2: DDPG 网络结构

异步 DDPG 学习

在 DDPG 学习过程中，我们采用异步学习的方式进行，学习流程如上图所示，在仿真环境中，我们用不同策略的 agent 进行动作空间的探索，从而生成训练样本 $< s_t, a_t, r_t, s_{t+1} >$，不同的 worker 会计算网络梯度并将梯度计算结果发给参数服务器，参数服务器每 N 步进行网络参数的更新。异步更新算法如下图所示。

10.5　在线排序策略模型优化

尽管在策略优化学习的过程中，我们使用了离线仿真模型进行策略空间的探索并对预估结果进行了奖励标定 (calibration), 但是仿真的结果并不能代表用户的真实行为，因为用户的行为会受到其他环境因素的影响，而且正如在仿真

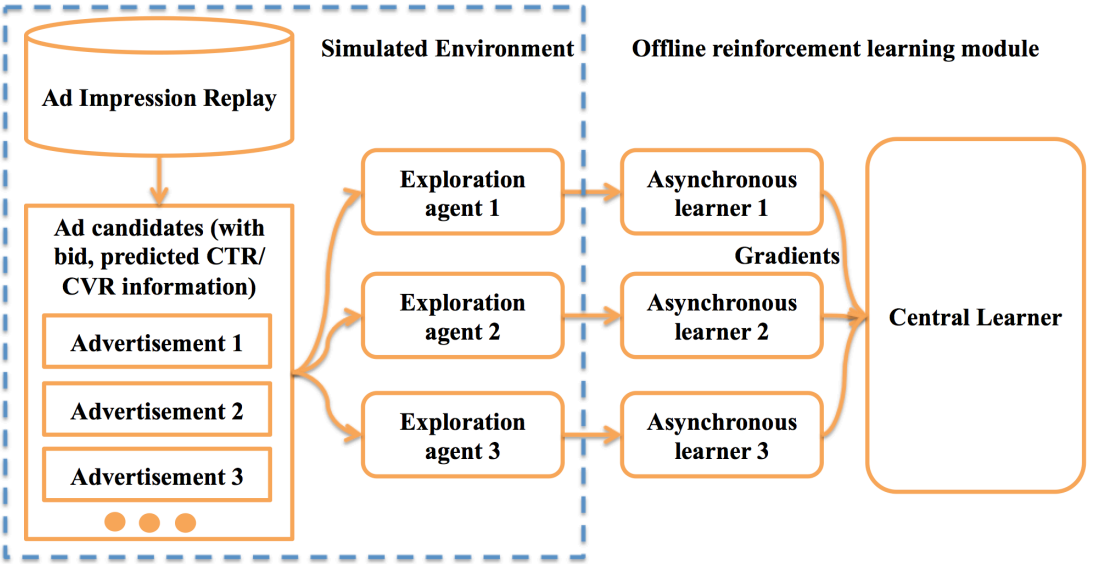


图 10.3: 离线 DDPG 学习流程框架

系统中提到的，仿真系统仍有很多因素没有考虑，如广告主的预算、出价等信息，仿真系统无法得到序列化得仿真结果。因此策略优化算法需要根据线上的真实反馈进行在线学习。

对于在线学习方法，我们利用 Evolution Strategy 方法 [31] 进行在线策略更新。对于给定的排序策略模型 $\pi_\theta(s_t)$，Evolution Strategy 通过执行以下两步进行策略的探索和模型的更新：（1）在模型参数空间 θ 加入高斯噪声产生探索动作 a;(2) 统计不同噪声下策略得到的 $reward$ 结果，并根据结果来更新网络参数。假设我们对参数空间进行 n 次扰动，产生扰动后的参数空间 $\Theta_\pi = \theta_\pi + \epsilon_1, \theta_\pi + \epsilon_2, ..., \theta_\pi + \epsilon_n$，对应的线上的实际奖励为 R_i，则参数的更新的方法为：

$$\theta'_\pi = \theta_\pi + \eta \frac{1}{n\sigma} \sum_{i=1}^{n} \overline{R_i} \epsilon_i \tag{10.5}$$

其中 η 表示学习率。使用 Evolution Strategy 进行模型参数更新，具有三点优

Algorithm 3: Asynchronous DDPG Learning

Input: Simulated transition tuple set $\mathcal{T}$ in the form $\psi =< s_t, a_t, r_t, s_{t+1} >$

Output: Strategy Network $\pi_{\theta_\pi}(s_t)$

Initialize critic network $Q_{\theta_Q}(s_t, a_t)$ with parameter θ_Q and actor network $\pi_{\theta_\pi}(s_t)$ with parameter θ_π;

Initialize target network Q', π' with weights $\theta_{Q'} \leftarrow \theta_Q$, $\theta_{\pi'} \leftarrow \theta_\pi$;

repeat

　Update network parameters θ_Q, $\theta_{Q'}$, θ_π and $\theta_{\pi'}$ from parameter server;

　Sampling subset $\Psi = \{\psi_1, \psi_2, ..., \psi_m\}$ from $\mathcal{T}$;

　For each ψ_i, calculate $Q^* = r_t + \gamma \cdot Q'(s_{t+1}, \pi'(s_t))$;

　Calculate critic loss $L = \sum_{\psi_i \in \Psi} \frac{1}{2} \cdot (Q^* - Q(s_t, a_t))^2$;

　Compute gradients of Q with respect to θ_Q by $\bigtriangledown_{\theta_Q} Q = \frac{\partial L}{\partial \theta_Q}$;

　Compute gradients of π with respect to θ_π by $\bigtriangledown_{\theta_\pi} \pi = \sum_{\psi_i \in \Psi} \frac{\partial Q(s_t, \pi(s_t))}{\partial \pi(s_t)} \cdot \frac{\partial \pi(s_t)}{\partial \theta_\pi} = \sum_{\psi_i \in \Psi} \frac{\partial A(s_t, \pi(s_t))}{\partial \pi(s_t)} \cdot \frac{\partial \pi(s_t)}{\partial \theta_\pi}$;

　Send gradients $\bigtriangledown_{\theta_Q} Q$ and $\bigtriangledown_{\theta_\pi} \pi$ to the parameter server;

　Update θ_Q and θ_π with $\bigtriangledown_{\theta_Q} Q$ and $\bigtriangledown_{\theta_\pi} \pi$ for each global N steps by gradients method;

　Update $\theta_{Q'}$ and $\theta_{\pi'}$ by $\theta_{Q'} \leftarrow \theta_{Q'} + (1-\tau)\theta_Q$, $\theta_{\pi'} \leftarrow \theta_{\pi'} + (1-\tau)\theta_\pi$;

until *Convergence*;

势。首先 Evolution Strategy 是一种 derivative-free 的更新方式，使用这种更新方式可以避免计算梯度带来的计算量；其次，在分布式 parameter-serving 框架下，每一个 worker 只需要把 reward 数值传给 parameter-server 即可，可以大幅度降低在线学习对网络带宽的需求；最后，这种方法可以以一个 episode 整体计算奖励，而不必考虑状态转移过程中奖励稀疏性对算法的影响，从而实现基于浏览序列的整体优化效果。

10.6 实验分析

我们通过实验验证以下问题，首先，模型是否能够收敛到最优解？其次，不同的网络架构和参数设计会对于模型收敛性有什么影响？最后，在线更新对于提高模型的线上效果的增益大概是多少？

针对第一、二个问题，我们采用简单的搜索上下文特征表示 s，只使用查询词 ID 来表示 s，在这种简单的表示情况下，通过在离线仿真平台上对排序函数参数集合 a 进行滑动窗口搜索，可以找到排序函数参数集合的最优值，对比从 DDPG 搜索到的最优值和滑动窗口得到的最优值，即可判断方法的收敛性。在图10.4和图10.5中，我们比较了不同模型配置情况下的训练收敛性。其中参数配置如表10.1所示，从结果中，我们可以发现以下结论：(1) dueling 通过将奖励函数 (value function) 和优势函数 (advantage function) 区分对待，明显地提升了模型的收敛性质；（2）由于数据的方差比较大，选取大的训练数据集合尺寸 (batch size) 对于收敛有正向作用；（3）使用衰减的学习率对于收敛有正向作用。

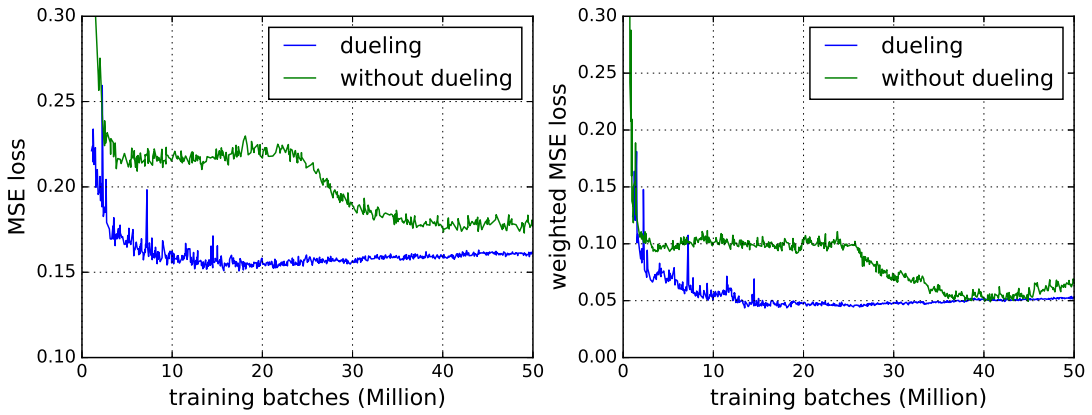


图 10.4: 使用 dueling 结构对于收敛性的影响

针对问题三，我们将 DDPG 学习到策略模型放到线上进行 2% 流量测试，并用 ES 进行策略更新。实验进行了 4 天，主要比较了 CTR,PPC 和 RPM 指标

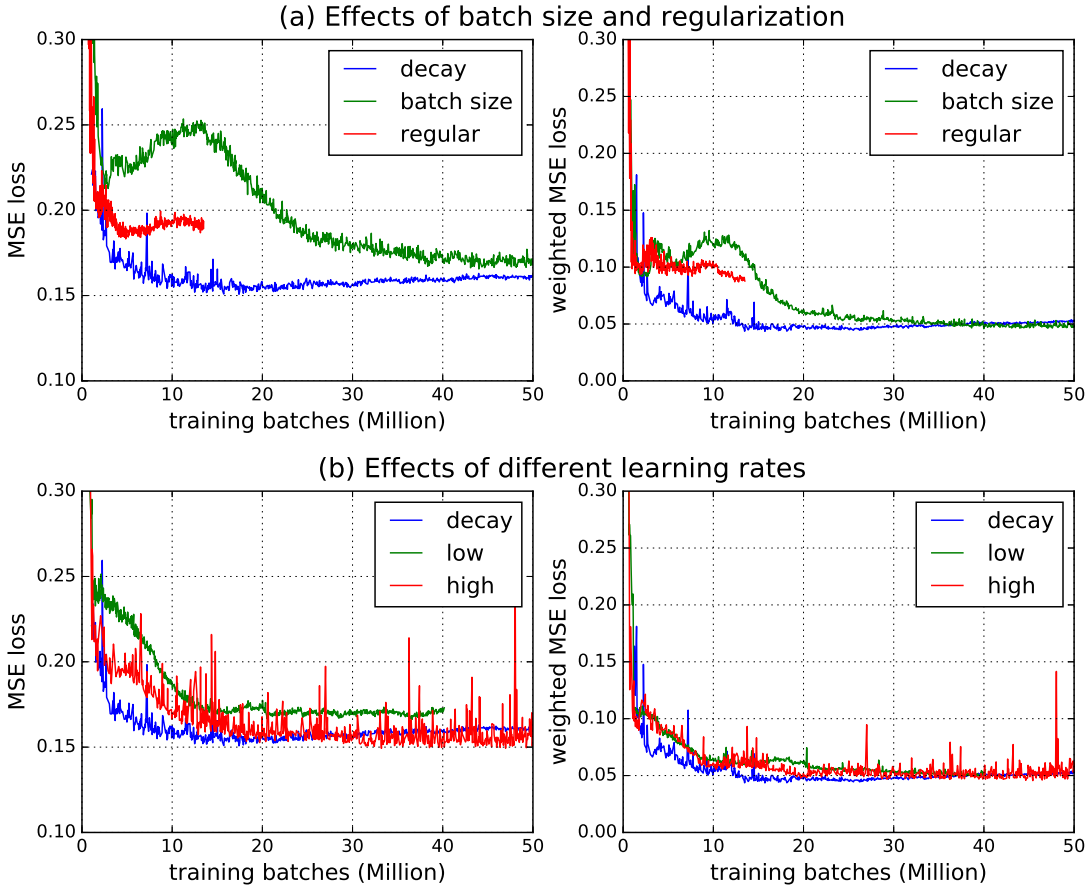


图 10.5: 使用不同的衰减因子，不同的批训练数据集合大小 (batch size) 对于收敛性的影响

的变化情况，实验结果如下图所示。从结果中，不难发现在线更新对于算法效果的正向作用。

表 10.1: 图10.5中的模型参数设置。

ID	Learning rate	Regularization	Batch size
decay	exponential decay	1.0e-5	50k
low	**1.0e-5**	1.0e-5	50k
high	**1.0e-4**	1.0e-5	50k
batch size	exponential decay	1.0e-5	**10k**
regular	exponential decay	**1.0e-3**	50k

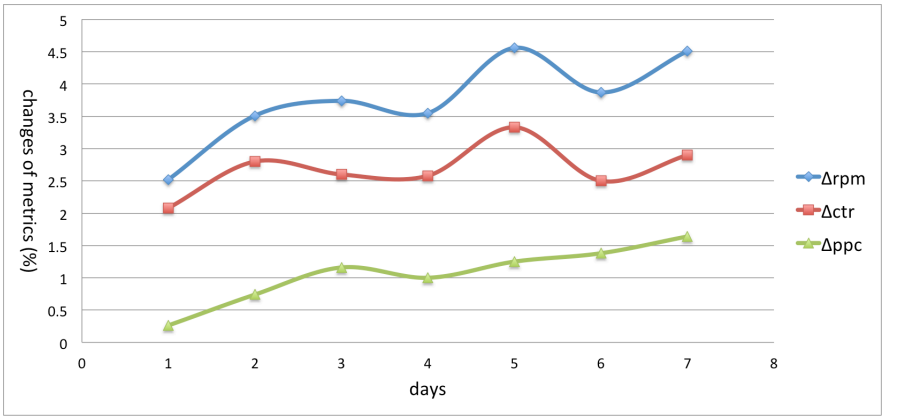


图 10.6: 在线 ES 效果变化趋势

10.7 总结

在本文中，在吸取了强化学习和遗传策略 (evolution strategy) 的优势的基础上，我们提出了一套在线上系统上现实可行的策略学习方案。通过建立离线仿真模型，我们完成了在不损失线上系统性能的前提下对于策略函数（动作函数）的最大化探索；由于线上线下数据的不一致性，直接使用线上数据来更新离线仿真数据中学习的策略函数会面临数据分布不一致等问题，通过使用遗传策略的方法，我们实现了方法的在线更新，并且在淘宝搜索广告系统上实现了效果的稳定增益。

第十一章　TaskBot – 阿里小蜜的任务型问答技术

11.1　背景和问题建模

在阿里小蜜里，除了 QA 问答和开发域聊天之外，还有一种任务型对话。这里的任务型问答是指由任务驱动的多轮对话，需要在对话中协助用户完成某个任务，比方说订机票，订酒店等。传统的任务型问答通常是由 slot filling 来做的 [20]，需要较多的人工模板和规则加上大量的训练语料组成。我们在阿里小蜜里面尝试了一个端到端可训练的 TaskBot 方案，基于强化学习和 Neural Belief Tracker，旨在可以快速搭建一个任务型的对话服务。

下面我们在订机票这个业务上，展示一个完整的任务型的多轮对话过程，如下图 11.1所示。这里系统会反问一些用户信息，比方说：请问您从哪里出发 ($where\_from$)，请问您要到哪里去 ($where\_to$)，还有最终下订单（$order$）这个动作。除此之外，还需要记录下目前获取到的 slot 状态 (V_{slots})，方便之后出订单。因此，这里涉及两个任务：

- Action policy: 系统如何给出合适的回复（反问或者出订单）；
- Belief tracker: 如何抽取 slot 状态。因为需要产出订单，所以在每轮对话中都需要抽取出当前用户给出的 slot 信息。

对于第一个，这是一个多轮对话，而且我们可以收集到用户的反馈（继续聊天，退出，下单等操作），所以我们尝试了深度强化学习来做这个事情。对于

	User	System Actions
1	我要订机票	Where_From
2	北京	Where_To
3	上海	What_Date
4	不对，我要天津走	What_Date
5	明天	Order
6	有其他时间吗	What_Date
7	后天呢	Order

Vslots
(None, None, None)
(京, None, None)
(京, 沪, None)
(津, 沪, None)
(津, 沪, 2017-03-24)
(津, 沪, 2017-03-24)
(津, 沪, 2017-03-25)

图 11.1: 订机票场景中需要的系统反馈和 slot 状态示例。

第二个，深度学习技术在 sequential labeling 里有非常不错的效果，自然的我们尝试了深度学习的方法。

11.2 模型设计

整体我们的系统结构分为了数据预处理层，以强化学习为中心的端到端的对话管理层，和任务生成层。其中数据预处理层包括常见的分词，实体抽取等模块，基于这些模块的输出接入以强化学习为中心的对话管理层。

其中，强化学习模块主要包括一下三个部分，*Intent network* 用来处理用户的输入,*Neural belief tracker* 记录 slot 信息, 和 *Policy network* 决定系统的 actions（反问哪个 slot，或者出 order）。

11.2.1 Intent Network

这里我们尝试了 RNN 和 CNN，在实验里我们发现 CNN 和 RNN 效果差不多，但是速度会快一倍，所以我们最后采用的 CNN 的方案。网络结构参考了这篇文章 [16]，具体结构如图11.2, 这里画的是单层的 CNN，我们也尝试了多层的

CNN，在订机票的任务里面差别不大。简单来说，我们用 CNN 学一个 sentence embedding 来表征用户的意图，这个信息作为后面的 policy 网络的输入。

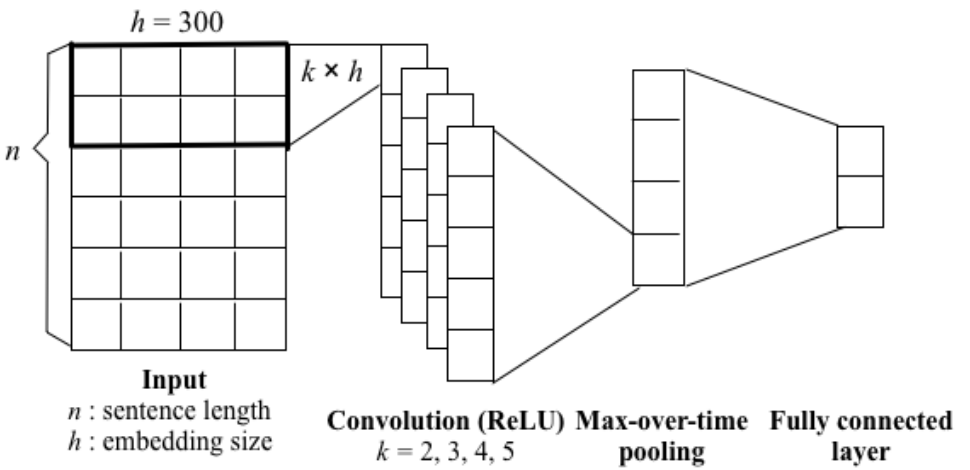


图 11.2: Intent Network.

11.2.2 Belief Tracker

Belief tracker，有时候也被称为 slot extraction，是用来 track 到当前回话为止的 slot 状态。我们尝试了两种技术方案。一个是用 pointer network (PtrNet) [36]，输出有 N*2 个（N 是 slot 数量），每个 slot 有个开始 Pointer 和结束 Pointer，都指向输入数据的某个位置。这样，PtrNet 可以直接产出 slot 信息。另外一个是 Sequence Labeling，模型用的是 BiLSTM-CRF [18]，需要标记出每个词是否属于某个 label。这里每个 slot S_i 有两种 labels:S-S_i 和 O-S_i，分别代表 S_i 的开始和中间位置。除了 slot 的 label 之外，还有一个 label O 代表空。实验对比下来，方案一模型比较简单，直接产出多个 slots 的结果，但是效果不如方案二。

11.2.3 Policy Network

强化学习部分我们选用的方案是 Policy Network，模型的 episode，reward，state，和 action 的定义如下。

Episode. 在订机票场景里，在一个用户和系统的会话里，如果系统第一次判断当前用户的意图为“购买机票”，这个就作为一个 episode 的开始，如果用户购买了机票或者退出会话，则认为 episode 结束。

Reward. 在这个场景里，获取用户反馈非常关键。我们有两种方式：第一，收集线上用户的反馈，比方说用户的下单，退出等行为；第二，初始化的时候，如果是没有训练过的模型直接上线学习的话，用户体验比较差。为了避免这个问题，我们使用预训练环境，让小二用预训练环境训练出一个效果相对可以的模型再上线。预训练环境主要需要获取两部分的反馈，一个是 action policy，另外一个是 belief traker。下图11.3是预训练环境的示例。其中 Action policy 部分用的是 Policy Gradient 方法更新模型，正反馈的 reward 为 1.0，负反馈为-1.0, Belief tracker 部分仅使用正反馈作为正例，出现错误需要小二标出正确的 slots。

图 11.3: 阿里小蜜任务型对话的预训练环境。

State. 我们主要考虑了 intent network 出来的 user question embeddings，当前抽取的 slot 状态，和历史的 slot 信息，之后接入一个全连接的神经网络，最后连

softmax 到各个 actions。

Action. 在订机票场景，action 空间是离散的，主要包括对各个 slot 的反问和 Order（下单）: 反问时间，反问出发地，反问目的地，和 Order。这里的 action 空间可以扩展，加入一些新的信息比方询问说多少个人同行，用户偏好等。

11.2.4 模型

我们最终的模型如图11.4，每个用户的问题都输入到 IntentNetwork 里得到问题的表示，然后和 BelifTracker 获取 slot 信息, 还有历史的 slot 信息合起来输入到 PolicyNetwork 里。假设当前用户问题为 q_i，上轮系统问题为 a_{i-1}，历史 slot

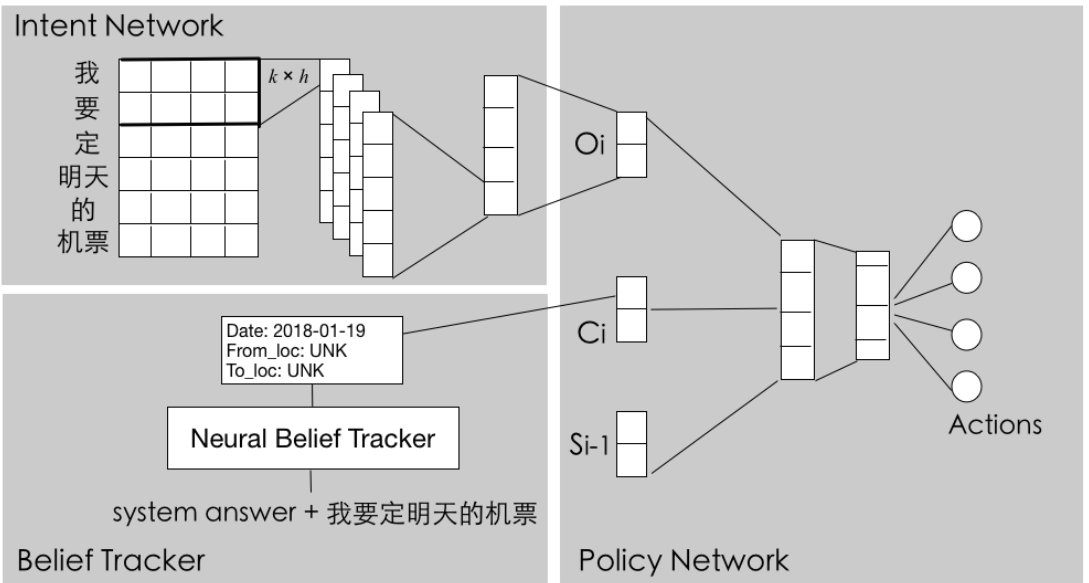


图 11.4: TaskBot 里的强化学习方案。

信息为 S_i，完整的网络定义如下。

$$\begin{aligned} O_i &= \text{IntentNet}(q_i), \\ C_i &= \text{BeliefTracker}(q_i, a_{i-1}), \\ X_i &= O_i \oplus C_i \oplus S_{i-1}, \\ H_i &= \text{FullyConnectedLayer}(X_i), \\ P(\cdot) &= \text{Softmax}(H_i). \end{aligned} \tag{11.1}$$

我们用的是 Policy Gradient 算法，具体来说是 REINFORCE 算法，为了使得训练更加稳定，这里可以加个 baseline 和 advantage function。实验中我们也尝试了 Deep Q Network 和 Actor-Critic，但是发现效果相对 REINFORCE 提升不大。

11.3 业务实战

首先我们对比了两种 belief tacker 的方案，效果如图12.1。在几个指标里，BiLSTM+CRF 的效果都会比 PtrNet 好，主要原因是前者对于序列的建模效果更好，BiLSTM 可以挖掘出当前词的 context 的信息，最后 CRF 层能有效的对标记序列进行建模。PtrNet 是端到端的 sequence to sequence 模型，但是相比之下对于序列的建模效果会差一点，前面学到的信息随着序列的增长会减弱。

	Acc	F1	Precision	Recall
PtrNet	0.968	0.850	0.844	0.856
BiLSTM+CRF	0.995	0.961	0.965	0.965

表 11.1: Belief tracker 效果对比.

下面我们对比不同的 DRL 配置下的效果如图11.3。在测试的时候发现，如果用户退出会话 (Quit) 给一个比较大的惩罚-1，模型很难学好。这个主要原因是，用户退出的原因比较多样化，有些不是因为系统回复的不好而退出的，如果这个时候给比较大的惩罚，会对正确的 actions 有影响。

Is Trainable	Reward			Discount Factor
	Click	Quit	Continue	
Yes	1	0	0	0.1∼0.9
Yes	1	-0.1	0	0.1∼0.9
No	1	-1	0	0.1∼0.9
Slow	1	0	0.1	0.1∼0.9
Slow	1	-0.1	0.1	0.1∼0.9
Slow	1	-1	0.1	0.1∼0.9

表 11.2: DRL 的参数设置。

11.4 总结

目前模型已经上线，大家可以在阿里小蜜里试用。在这个项目里，我们探索了基于深度强化学习（DRL）的任务型问答的方案，在订机票这个任务上走通了这个方案。据我们所知，这是第一个在工业界落地的基于 DRL 的任务型问答技术，后续我们准备把这个方案扩展到其他的任务上（充值，订电影票等），理想的情况下是我们通过预训练平台就可以把 action policy 和 slot extraction 两部分都学好。

第十二章　DRL 导购 – 阿里小蜜的多轮标签推荐技术

12.1　背景

阿里小蜜作为一个人工智能助理，除了可以解决用户的业务咨询类问题，缓解人工压力之外，也兼备了其他助理业务包括：查天气、买机票、充话费等，同时，还可以通过多轮会话完成一个导购助理的工作。借助小蜜多轮会话能力，如何一步一步的搜集用户购买意图，促成商品成交，是本文要解决的问题。本文旨在通过主动推荐引导的方式推荐商品标签，搜集用户意图，最终为用户推荐出满足他们需求的商品。

小蜜导购助理的目标可细分为以下几点：

- 借助多轮对话场景逐步理解用户购买意图；
- 形成通用的挖掘和上下位链路预测方案；
- 结合用户信息 CTR 预估，实现实时、个性化推荐，最终促成商品成交；
- 使用强化学习的方式完成在线增强式标签推荐。

最终的产品形态是标签基础上，结合图片和选项卡的方式，应用到多轮对话中，如下图12.1（红框圈定范围内）。

如图所示，这次交互总共分四轮，用户和系统的行为如下：

图 12.1: 阿里小蜜多轮标签推荐样例。

User	System
Ask：我要买笔记本电脑	展示品牌：ThinkPad，Dell，Apple, ···
Click：Apple	展示屏幕属性：17 英寸，11.6 英寸，···
Click：13.3 英寸	展示商品：Apple,13.3 英寸电脑
Click item, buy it (or quit)	End

表 12.1: 阿里小蜜多轮标签推荐中用户和系统的交互。

这里有几个难点，首先如何从海量商品数据里挖掘出有效的商品属性值和属性名这样的标签对，其次是基于多轮对话信息，如果给出实时和人性化的标签推荐，并且促成商品的点击和成交。前者是**商品标签挖掘**的任务，我们尝试了传统的数据挖掘方法；后者是**多轮标签推荐**的任务，我们用深度强化学习模型来促成 CTR 和成交量的提升。

12.2 算法框架

算法框架如图12.2，包含商品标签挖掘，推荐链路预测，和多轮标签推荐。

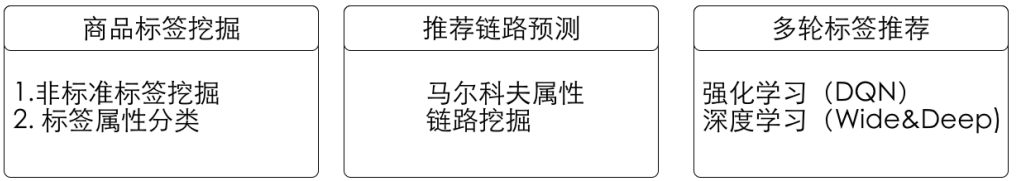


图 12.2: 阿里小蜜多轮标签推荐算法框架。

商品标签挖掘我们把标签定义为一个商品的属性值和属性名，比如说苹果这个标签的属性值是“品牌”，属性名是“苹果”。我们需要推荐的商品来自于淘宝，这里商品的量非常大，需要先对商品标签进行抽取。对于属性名，除了标准的名字例如“触屏”, “控油”等之外，我们挖掘了一些更加常用的口语化属性表示（比如: “触摸屏”, “皮肤比较油”, “千元以内”），用在多轮推荐中。训练语料来自淘宝主搜 query，用互信息、出现频率、左邻接熵、右邻接熵衡量一个词成词的概率。假设一个词是 a, b, 具体定义如下:

$$
\begin{aligned}
MI_{a,b} &= \log \frac{p(a,b)}{p(a)p(b)}, \\
H_{left} &= -\sum_{w} p(w,a,b|a,b) \log p(w,a,b|a,b), \\
H_{right} &= -\sum_{w} p(a,b,w|a,b) \log p(a,b,w|a,b).
\end{aligned}
$$

其中，互信息 MI 越大，表示词的内聚程度越大，成为候选属性的概率越大；左右邻接熵 H_{left} 和 H_{right} 则反映了一个文本片段的左邻字集合和右邻字集合的随机程度，邻熵越大，表明该词的左右边界越随机，成词的概率越大。

通过在大规模语料中计算限定长度的字组合的三个特征值，做归一化求和计算得分，选取得分较高的词作为我们的候选词。最后根据挖掘结果对原有 query 进行分词。该方法对性能要求较高，尤其是内存，因此可以通过 batch 的

方式分批进行。在阿里小蜜的计算环境中，我们将 batch 的大小限定在 100w，可最大限度的保证不会出现 OOM。最后整理所有 batch 各个维度的指标，做加和处理。

挖掘出标签之后，需要对同一属性的标签进行归类，目的是保证在每次推荐标签时，在同一个属性类别下，给用户多重选择。标准属性的归类沿用主搜已有的属性命名方法，非标准属性的归类任务转化为给每个非标属性打上属性名（例如:“皮肤比较油”对应的属性名是“肤质”)。我们使用主搜中的搜索点击数据，即每个 query 后用户点击的商品详情。具体操作流程如下:

- 对 query 分词，去掉停用词和品类词，剩下的作为候选属性;
- 使用候选属性匹配商品详情页中的属性 key-value pair，将匹配到的属性值对应的属性名与候选属性做关联 (映射过程采用模糊匹配：融合了编辑距离和词共现。例如:“皮肤比较油”会映射到商品详情页中会出现“肤质：油性皮肤”上，进而可以认为“皮肤比较油”的属性类别是“肤质”)

推荐链路预测拿到候选的标签之后，需要在用户的会话链路中，即用户的每轮会话，展示推荐的标签。这里我们根据主搜用户 query 中个属性自然顺序，计算属性之间的马尔科夫链路，具体计算方式如下。

$$
\begin{aligned}
&\text{开始:} \quad \hat{p} = \text{argmax}_{p_i}\, p(p_i|context), \\
&\text{第一轮:} \quad p(a_1|\hat{p}), \qquad e.g. \quad p(\text{苹果}|\text{笔记本}) \\
&\text{第二轮:} \quad p(a_2|a_1,\hat{p}), \qquad e.g. \quad p(\text{17 英寸}|\text{笔记本电脑，苹果})
\end{aligned}
$$

这里涉及的概率计算都是由对全网类目用户的 query（采用模糊匹配）和标签的点击数据统计而来的。每轮取最多前 50 个高频标签作为候选。

多轮标签推荐系统设计上，我们采用粗排加精排的方式，上面模块相当于是粗排，后续连到一个多轮标签推荐模块去对标签做精排。传统解决上面问题的思路是做 CTR 预估，根据标签的点击率来建模。然而这个方案有两个问题，第一，CTR 和 GMV 通常是分开优化，两者是非常相关的任务，考虑多目标优化会对两个任务都有帮助；第二，无法用到多轮交互的信息，不同的标签推荐序列会

对用户的是否成交有影响。基于这两点考量，我们用 DRL 对多轮对话进行建模，旨在通过多轮交互同时提升 CTR 和 GMV。

我们的问题可以看作是一个多轮链路决策的问题，agent 是我们的系统，agent 决定每一轮出什么标签（action），之后环境（用户）会给系统一个反馈，这里反馈可以是正向的（比如商品点击或者商品成交），也可以是负向的（退出或者用户表达不满等）。这个问题就成了，如何在多轮交互中获取最大的 reward，这就是典型的 RL 的问题。因此使用 RL 来做多轮导购，可以拟合出不同用户的推荐链路来最大化用户的点击和成交，即 CTR 和 GMV。

12.3 深度强化学习模型

深度强化学习是由深度学习和强化学习两者的结合，其中深度学习负责拟合给定的输入和输出的关系，强化学习负责把控学习方向。典型的方法有基于价值的深度强化学习、基于策略的深度强化学习、基于模型的深度强化学习。这三种不同类型的深度强化学习用深度神经网络替代了强化学习的不同部件。基于价值的深度强化学习本质上是一个 Q Learning 算法，目标是估计最优策略的 Q 值。DQN 是一个典型的基于价值的深度强化学习算法，它是由 DeepMind 在 2013 年在 NIPS 提出。DQN 算法的主要做法是 Experience Replay，其将系统探索环境得到的数据储存起来，然后，打破数据之间的关联性，通过随机采样样本更新深度神经网络的参数。实验下来我们发现 DQN 比一般的基于策略的深度强化学习效果好，所以我们采用了 DQN 作为我们的强化学习算法。

DQN 的核心思想是通过一个 value network $Q(\cdot)$ 来拟合当前 state s 和 action a 所能达到的累积的 rewards，优化的时候需要考虑所有未来 actions 所带来的 rewards。它的 loss 定义如下：

$$L(w) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w))^2] \tag{12.1}$$

其中 s' 和 a' 为下个 state 和 action，γ 为阻尼系数 (discount factor)，r 为当前的 reward。这里如果当前状态为一个 episode 结束的状态的话，$Q(s', a', w)$ 设为 0。下面我们详细讲述模型的 episode，state，action，和 reward 的定义。

12.3.1 强化学习模块

Episode. 在多轮标签推荐场景里，如果系统第一次判断当前用户有购买商品的意图，这个就作为一个 episode 的开始，如果用户购买了商品或者退出会话，则认为 episode 结束。

State. 状态主要考虑了三部分信息，用户的 profile，用户的 question，商品信息和 session 信息。这里除了用户的 profile 是个静态特征之外，后面三类特征都是会随着对话的进行改变的。

Action. 由于我们的 action space 非常大（所有的商品的属性名和属性值的组合），我们对 action 进行了参数化，每个 action 用它的属性名和属性值表示 $< p_i, v_i >$，两部分都是离散的 id。所以，action i 的向量表示为:$e_i \oplus e'_i$, 这里 e_i 和 e'_i 为神经网络学到的两部分各自的向量表示。

Reward. 我们主要优化的是 CTR 和 GMV 两个目标，其中 GMV 比较重要。所以我们定义 reward 的时候，购买的 reward 设为 1.0，点击的 reward 相对小一些，为 0.1。由于我们每轮是展示多个标签，只有点击或者购买的标签有正的 reward，其他部分 reward 都设为 0.0。下图12.3是多轮交互中的 reward 定义的示例。

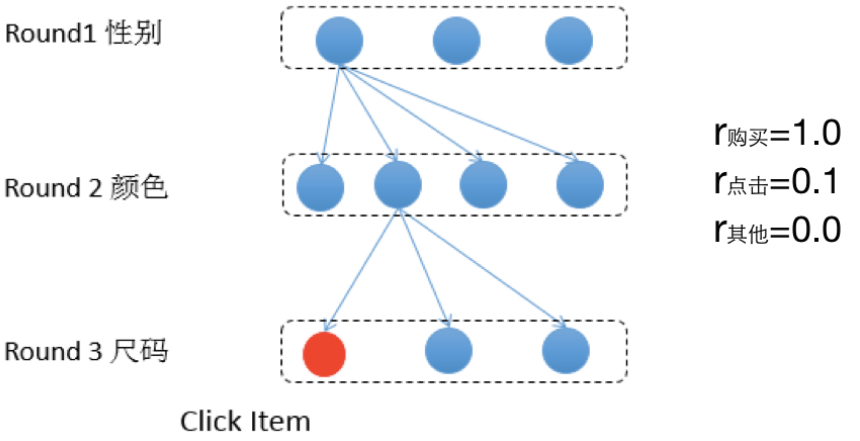
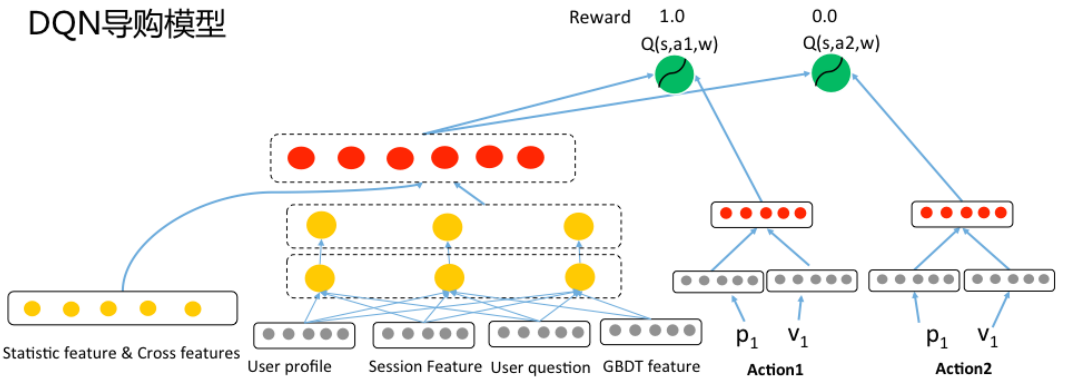


图 12.3: 多轮标签推荐的 reward 定义。

12.3.2 最终模型

模型上的设计我们主要参考了 Wide&Deep 模型 [6]，这个模型融合了记忆（memorization）和归纳（generalization）的两个过程，模型中的宽线性模型（Wide Model）用于记忆；深度神经网络模型（Deep Model）用于归纳。该模型不仅可以容纳一些统计类特征，也可以将 id 类（或者文本类）特征通过 DNN 做有效的归纳抽象，泛化性更强。



Statistic Feature	Cross Feature	User Profile	Session Feature	User Question	GBDT Feature	Action1 Feature	Action2 Feature
History PV, CTR etc.	Age&Product etc.	Age,Gender, Address, etc.	Click sequence, etc.	我想要一个笔记本 (I wanna buy a laptop)	Tree nodes	Property-Value pair	Property-Value pair

图 12.4: 多轮标签推荐的模型。

模型结构如图12.4, 是一个 regression 模型。模型主要是根据给定 state s，action a，拟合出一个 value $Q(s, a, w)$，这里的 w 是模型参数。建模的时候，wide 部分和 deep 部分分别处理不同的特征：其中 Wide 部分放了一些常用的统计特征和组合特征，Deep 部分放的是 state 信息，Deep 部分学到的表示和 Action 的表示会 concat 到一起，经过一个 MLP 再和 wide 部分一起输出。

12.4 业务实战

在离线数据上，我们对比了 DRL 和非 DRL 模型在 AUC，Recall 指标上的变化。如图12.4, 这里主要测试的是用户点击的指标，在对比的四个衡量标准里，DRL 模型都有了提升。这个说明了 DRL 能够学习到多轮对话的序列信息，从而在这个任务上取得了效果的提升。

	AUC	Recall1	Recall2	Recall5
非 DRL	0.94	0.86	0.95	0.98
DRL	0.96	0.87	0.97	0.99

表 12.2: 离线评测 DRL 和非 DRL 模型。

在线上对比，DRL 模型相比非 DRL 模型在商品点击成交量提升了 6%，相比纯统计方法提升 15%。双十一期间，DRL 模型产生数百万对话轮次，GMV 超过千万，商品点击成交转化率相比去年提高 54%。

12.5 总结和展望

我们尝试在阿里小蜜的场景中为用户提供一种新的导购助理。借助多轮对话场景，逐步理解用户的购买意图，实现实时、个性化推荐，最终促成商品成交。经过 2017 年双 11 的考验，我们的方案不仅承接住了流量洪峰的考验，同时也验证了技术方向的可行性。然而，导购助理还只是 DRL 在阿里小蜜场景下的初步尝试，不仅还有着很大的提升空间，而且在业务上也具有很强的扩展性。在接下来的时间内，除了继续增强导购助理的能力之外，我们也将探索在业务咨询场景下的问题补全等更多的实际问题。

参考文献

[1] Michael Best and Nilotpal Chakravarti. Active set algorithms for isotonic regression: a unifying framework. *Mathematical Programming*, (1-3):425–429, 1990.

[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *International Conference on Machine Learning*, pages 89–96, 2005.

[3] Christopher J Burges, Robert Ragno, and Quoc V Le. Learning to rank with nonsmooth cost functions. In *NIPS*, pages 193–200, 2007.

[4] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*, 2008.

[5] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, pages 129–136. ACM, 2007.

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. *CoRR*, abs/1606.07792, 2016.

[7] Djork-Arne Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). 2016.

[8] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *ACM Conference on Recommender Systems*, pages 191–198, 2016.

[9] Fredric C Gey. Inferring probability of relevance using the method of logistic regression. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 222–231. Springer-Verlag New York, Inc., 1994.

[10] Saurabh Gupta, Sayan Pathak, and Bivas Mitra. *Complementary Usage of Tips and Reviews for Location Recommendation in Yelp*. Springer International Publishing, 2015.

[11] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *NIPS*, pages 2944–2952, 2015.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] Junling Hu, Michael P Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pages 242–250, 1998.

[14] Guido W Imbens and Tony Lancaster. Efficient estimation and stratified sampling. *Journal of Econometrics*, 74(2):289–318, 1996.

[15] Krishnaram Kenthapadi, Krishnaram Kenthapadi, and Krishnaram Kenthapadi. Lijar: A system for job application redistribution towards efficient career marketplace. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1397–1406, 2017.

[16] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[17] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[18] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360, 2016.

[19] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[20] Feng-Lin Li, Minghui Qiu, Haiqing Chen, Xiongwei Wang, Xing Gao, Jun Huang, Juwei Ren, Zhongzhou Zhao, Weipeng Zhao, Lei Wang, Guwei Jin, and Wei Chu. *AliMe Assist: An Intelligent Assistant for Creating an Innovative E-commerce Experience*. 2017.

[21] Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2008.

[22] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[23] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of International Conference on Learning Representations*, pages 1–14, 2015.

[24] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pages 157–163, 1994.

[25] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[27] Jerzy Neyman. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, 97(4):558–625, 1934.

[28] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[29] Tao Qin, Xu-Dong Zhang, De-Sheng Wang, Tie-Yan Liu, Wei Lai, and Hang Li. Ranking with multiple hyperplanes. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 279–286. ACM, 2007.

[30] RY Rubinstein and DP Kroese. The cross-entropy method: A unified approach to combinatorial optimization, monte-carlo simulation and machine learning. 2004.

[31] Tim Salimans, Jonathan Ho, Xi Chen, and IIya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864*, 2017.

[32] Juan C Santamaría, Richard S Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163–217, 1997.

[33] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[34] Paul R Thie. *Markov decision processes*. Comap, Incorporated, 1983.

[35] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[36] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015.

[37] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of International Conference on Machine Learning*, 2015.

[38] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[39] Han Zhu, Junqi Jin, Chang Tan, Fei Pan, Yifan Zeng, Han Li, and Kun Gai. Optimized cost per click in taobao display advertising. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 2191–2200, New York, NY, USA, 2017. ACM.