序列推荐

早期的做法：

- Embedding+MLP：不考虑顺序，只进行简单相加。虽然"看上去"简单相加好像是损失了很多信息，譬如说想象一个二维空间，如果把四个向量位置加权平均，那么最后的点会是这四个点的中点，是个"四不像"。但实际上，在高维空间是我们无法想象的，可以理解为每个点都是一个"小尖尖"上的点，那么平均之后不是"四不像"，而是"四都像"。其实，concat之后+MLP和平均+MLP的表征能力是相似的。见知乎问题：https://www.zhihu.com/question/483946894

MIMN (Multi-channel user Interest Memory Network)

随着我们引入更长的历史行为特征之后，会造成系统的**latency和storage cost**增加。所以，类似DIN, DIEN这样的序列推荐方法很难在工业界推行，因为随着行为序列变长，它们的消耗会大幅提升。但是，长序列肯定是可以提升AUC的。那么，怎么在latency和storage cost不增的前提下，尽量使用更长的行为序列呢？

文中作者说，

> Theoretically, the co-design solution of UIC and MIMN enables us to handle the user interest modeling with unlimited length of sequential behavior data.

但实际上，MIMN只能够解决**千级**的用户行为序列。如何解决更长的用户行为序列，我会在SIM中进行讲解。

1.1 实时CTR预测系统

在线上CTR预估系统中，CTR预估模块接收来自召回阶段的候选集之后，会实时的对该候选集中的候选广告进行预测**打分排序**，通常这一过程需要在一定时间内完成，通常是【10毫秒】。这个过程叫做Real-time prediction (RTP)

具体线上系统的结构如下图所示：

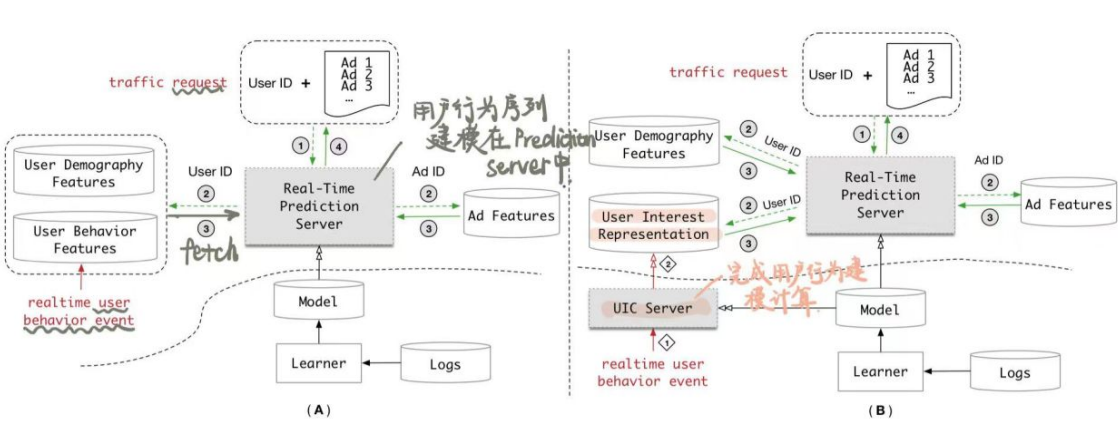


Figure 2: Illustration of Real-Time Prediction (RTP) system for CTR task. Typically it consists of three key components: feature management module, model management module and prediction server. (A) is the last version of our RTP system and (B) is the updated one with proposed UIC server. The key difference between system A and B is the calculation of user interest representation: (i) In A it is executed within prediction server w.r.t. request. (ii) In B it is executed separately in the UIC server w.r.t. realtime user behavior event. That is, it decoupled away and latency free w.r.t. traffic request.

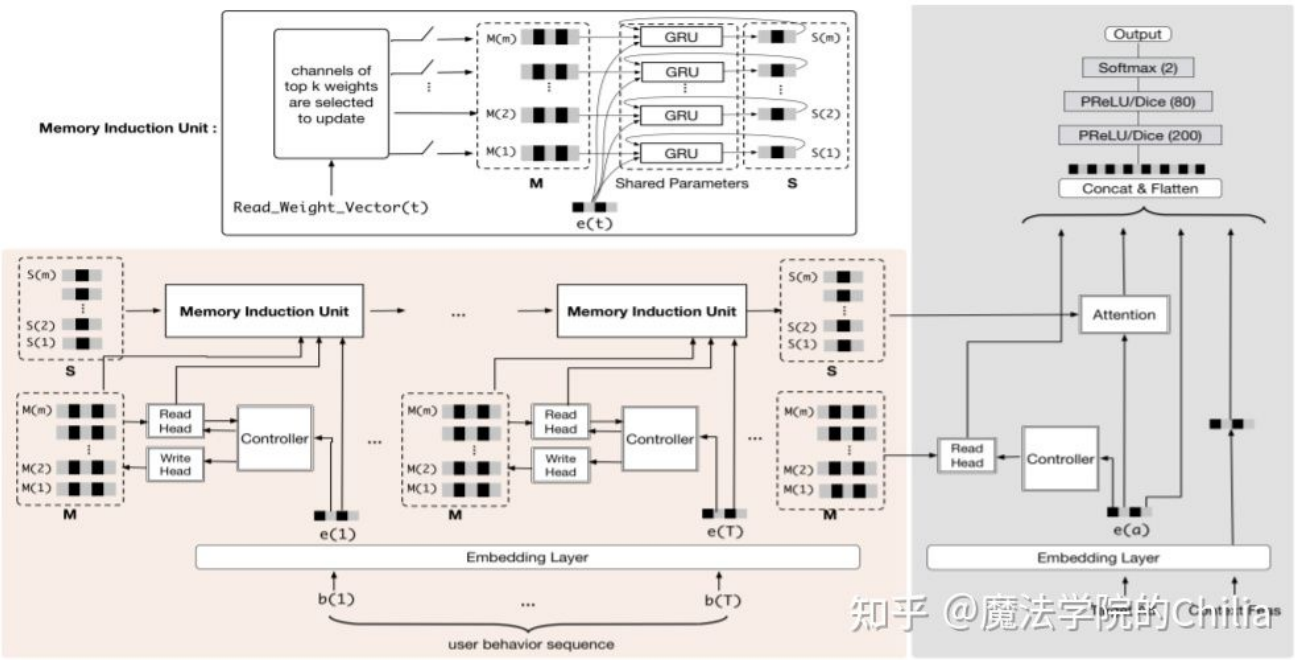
左侧为传统的线上实时CTR预估模块，用户行为序列建模是在**predict server**中完成。每次来一个query，都要去分布式数据库中找到用户的行为序列、然后拿到模型中进行建模。这样，用户的大量行为序列都得**存储**（在分布式数据库中，如TAIR），浪费空间；同时，**来了query才去现算**，浪费时间。实际上，使用DIEN来实时计算，其延迟(latency)和吞吐率(throughput)都已经达到RTP系统的极限了。

右侧为基于UIC的线上实时CTR预估模块，将资源消耗最大的**用户兴趣建模功能**单独解耦出来（最重要也是最庞大的部分就是建模丰富历史行为序列带来的user interest representation），设计成一个单独的模块**UIC(User Interest Center)**. UIC维护了一个用户**最新**的兴趣representation，是实时更新的。这样，每次来一个query的时候，我们就直接去UIC中去查询用户embedding就好了，而不用现去用复杂的GRU建模；**每次用户有了新的行为，就去增量更新embedding**，让我们的UIC模块处于最新状态。所以，UIC模块是latency-free的，因为我们在query来的时候不必用GRU等模型现去计算用户的超长序列表征，而是直接从UIC中去拿用户embedding就完事儿了。

1.2 离线MIMN模型

我们怎么能在UIC中建模千级的行为序列呢?

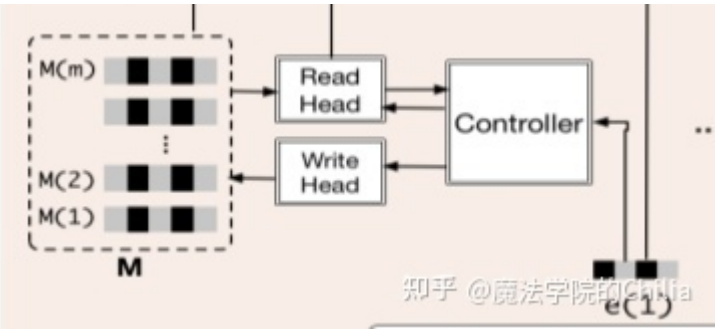
文章借鉴**神经图灵机（NTM,Neural Turing Machine）增量更新** 的思路，提出了一种全新的CTR预估模型**MIMN**（Multi-Channel User Interest Memory Network），整个系统的模型结构如下图所示：



左侧：用户行为序列的建模；右侧：传统Embedding+MLP。

1.2.1 神经图灵机 (Neural Turing Machine)

神经图灵机利用一个额外的**记忆网络**来存储长序列信息。在时间t，记忆网络可以表示为矩阵 M_t ，其包含m个memory slot $M_t(i), \{i = 1, \ldots, m\}$ ，NTM通过一个controller模块进行读写操作。这样，用户每来一个行为，就可以更新这个M矩阵，而不用把所有的用户行为序列都存储起来。



e(1)是第1时刻的用户行为embedding.

4.2.1.1 Memory Read

当输入第t个用户行为embedding向量e(t)，controller会生成一个用于寻址的read key k_t ，首先遍历全部的memory slot，生成一个**权重**向量 w_t^r ：

$$\mathbf{w}_t^r(i) = \frac{\exp\left(K\left(\mathbf{k}_t, \mathbf{M}_t(i)\right)\right)}{\sum_j^m \exp\left(K\left(\mathbf{k}_t, \mathbf{M}_t(j)\right)\right)}, \quad for \;\; i = 1, 2, \ldots m \qquad (1)$$

where

$$K\left(\mathbf{k}_t, \mathbf{M}_t(i)\right) = \frac{\mathbf{k}_t^T \mathbf{M}_t(i)}{\| \mathbf{k}_t \| \| \mathbf{M}_t(i) \|} \qquad (2)$$

最后得到一个**加权求和**的结果 r_t 作为输出即可:

$$\mathbf{r}_t = \sum_i^m w_t^r(i)\mathbf{M}_t(i). \qquad (3)$$

1.4.1.2 Memory Write

类似于memory read中的权重向量的计算（公式1），在memory write阶段会根据现在的用户行为$k(t)$先计算一个**权重**向量 w_t^w 。除此之外，还会生成两个向量，一个是add vector a_t ，另一个是erase vector e_t ，他们都是controller生成的并且他们控制着记忆网络的更新过程。记忆网络的更新过程如下所示：

$$\mathbf{M_t} = (1 - \mathbf{E_t}) \odot \mathbf{M_{t-1}} + \mathbf{A_t}$$

其中，

$$E_t = w_t^w \otimes e_t$$

$$A_t = w_t^w \otimes a_t$$

代表保留一些原来的记忆、增加一些新的记忆。

优化：Memory Utilization Regularization

传统的NTM存在memory利用分配不平衡的问题，这种问题在用户兴趣建模中尤为重要，因为热门的商品很容易出现在用户行为序列中，这部分商品会主导记忆网络的更新，为此文章提出一种名为memory utilization regularization。该策略的核心思想就是对不同的memory slot的**写权重向量**进行正则，来保证不同memory slot的利用率较为平均。实际上，就是增加了一个reg loss，**为不同slot写权重的方差。**

Memory Utilization Regularization. The idea behind memory utilization regularization strategy is to regularize the variance of write weight across different memory slots, pushing the memory utilization to be balanced. Let $\mathbf{g}_t = \sum_{c=1}^{t} \mathbf{w}_c^{\tilde{w}}$ be the accumulated update weight till t-th time step, where $\mathbf{w}_c^{\tilde{w}}$ represents the re-balanced write weight in c-th time-step. The re-balanced write weight $\mathbf{w}_t^{\tilde{w}}$ can be formulated as:

$$P_t = softmax(W_g \mathbf{g}_t) \tag{5}$$

$$\mathbf{w}_t^{\tilde{w}} = \mathbf{w}_t^{w} P_t. \tag{6}$$

$\mathbf{w}_t^{w}$ is the original write weight introduced in subsection 4.2, and $\mathbf{w}_t^{\tilde{w}}$ represents the new write weight for memory update. The weight transfer matrix P_t depends on (i) $\mathbf{g}_t$ which represents the accumulated utilization of each memory slot at t-th step, (ii) parameter matrix W_g which is learned by a regularization loss:

$$\mathbf{w}^{\tilde{w}} = \sum_{t=1}^{T} \mathbf{w}_t^{\tilde{w}}, \tag{7}$$

$$\mathbf{L}_{reg} = \lambda \sum_{i=1}^{m} \left(\mathbf{w}^{\tilde{w}}(i) - \frac{1}{m} \sum_{i=1}^{m} \mathbf{w}^{\tilde{w}}(i)\right)^2, \tag{8}$$

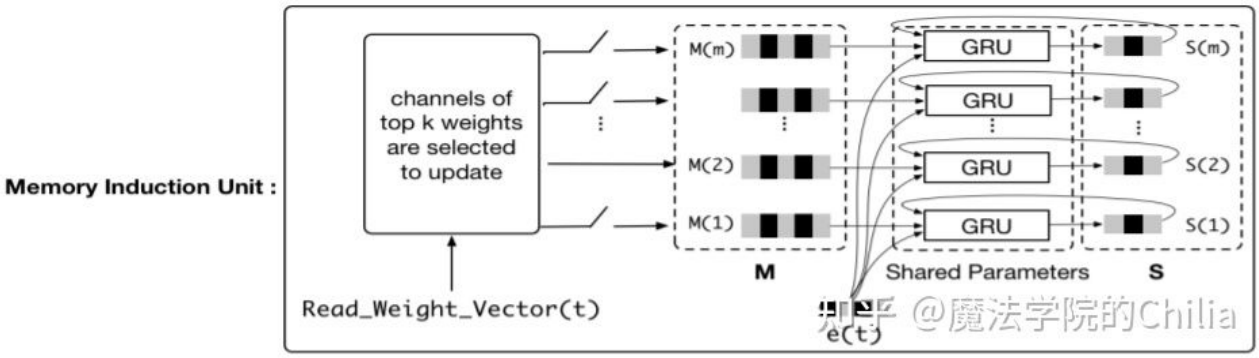
where m is the slot number of memory. L_{reg} helps to reduce the variance of update weight across different memory slots. Replacing $\mathbf{w}_t^{w}$ with $\mathbf{w}_t^{\tilde{w}}$, the update rates for all m slots tend be even. In that way, the utilization of all the memory slots are enhanced to be balanced. Utilization regularization can help memory tensor to store more information from source behavior data.

4.3 memory induction unit：Multi-channel

借鉴NTM的网络结构可以帮助有效构建用户**长**时间序列，但是无法进一步提取用户**高阶**的信息，比如在用户较长的历史浏览行为中，这一系列行为可以被认为是**多个channel**，具体可以表示为如下：

Figure 4: Multi-channel memory induction (MIU).

因此，文章提出**M**emory **I**nduction **U**nit来对用户**高阶兴趣**进行建模。



总共有m个GRU链，每个链有T个时间。

在MIU中包含一个额外的存储单元**S**，其包含m个memory slot，这里认为每一个memory slot都是一个用户interest **channel**。在时刻t，MIU首先选择top K的interest channel，然后针对第i个选中的channel，通过下式更新 $S_t(i)$

$$\mathrm{S}_t(i) = \mathrm{GRU}(\mathrm{S}_{t-1}(i), \mathrm{M}_t(i), e_t), \tag{9}$$

$M_t(i)$ 对应的是NTM在时刻t第i个memory slot向量， e_t 为用户行为embedding向量。也就是说**MIU从用户原始输入行为特征和存储在NTM中的信息中提取高阶兴趣信息**。值得一提的是为进一步减少参数量，**不同channel的GRU参数是共享的**.

4.4 Implementation for Online Serving

对于整个算法结构来说，在部署到线上的时候，主要的计算量都在网络的左侧部分（即用户行为兴趣建模），而右侧的Embedding+MLP的计算量则小很多。所以在线上部署的时候将左侧的网络部署到UIC server中，将右侧的网络部署到RTP(real-time prediction) server中。

在对左侧用户状态进行更新的时候，最新的memory state代表着用户的兴趣，并且可以存储到TAIR中用户**实时**CTR预估。当有一个用户行为事件发生的时候，UIC会重新计算用户兴趣的表征（而不会将原始的用户行为进行存储）并**增量更新**到TAIR中，所以长时间行为序列占用的系统空间从6T减少到了2.7T。同时文章指出MIMN+UIC这种系统结构不是适用任何场景下的，具体需要符合如下的要求：1、可以得到丰富的用户行为数据；2、用户实时行为的数量不能超过CTR预估的请求数量。