MIND

这次介绍的论文是来自阿里的一篇文章Multi-Interest Network with Dynamic Routing for Recommendation at Tmall(下面简称MIND)，主要作用是学习**user embedding**，以用作**召回。**"the matching stage is responsible for retrieving thousands of candidate items that are relevant to user interests."

摘要

本篇文章是用于在**召回(matching)**阶段来建模用户兴趣(根据用户的历史行为来做兴趣的embedding)，**用来解决单个向量不能很好的表达用户多兴趣**的问题。

回忆一下以前对于用户embedding的工作：

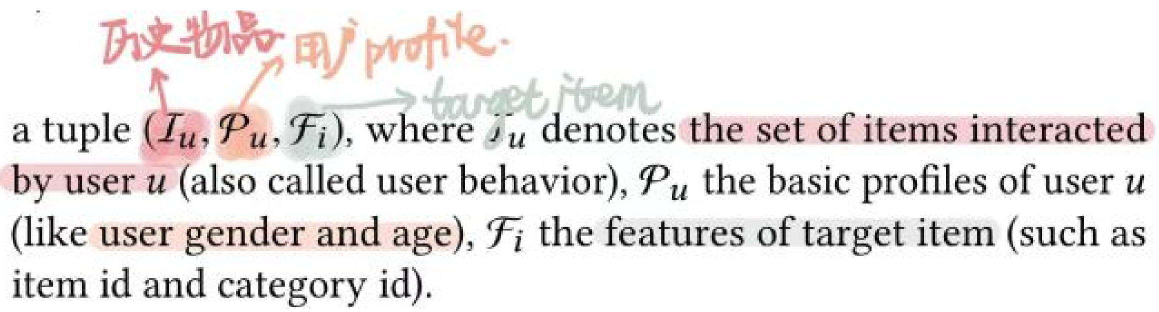
- 协同过滤的隐向量(分解user-item交互矩阵)。缺点：sparsity problem, 计算资源耗费大;
- 深度网络中，用**单一向量**表示用户embedding（如之前所有交互过商品的mean-pooling）。缺点：bottleneck, 不能表示多样的用户兴趣。
- DIN模型使用attention来赋予不同历史商品不同的权重。缺点：计算极为**耗时**（因为粒度太细了，每次来一个item，user embedding都会发生变化），只能用于ranking(千级数据排序)，不能用于matching(亿级数据召回)，因为无法做到把item embedding离线存好、然后用ANN检索到和user相似的item。

本文提出的MIND可以**对一个用户输出多个embedding**,但是是比DIN更加粗粒度的**聚类**，每个embedding代表一类用户兴趣cluster。

3. METHOD

3.1 Problem Formalization

首先定义三元组：

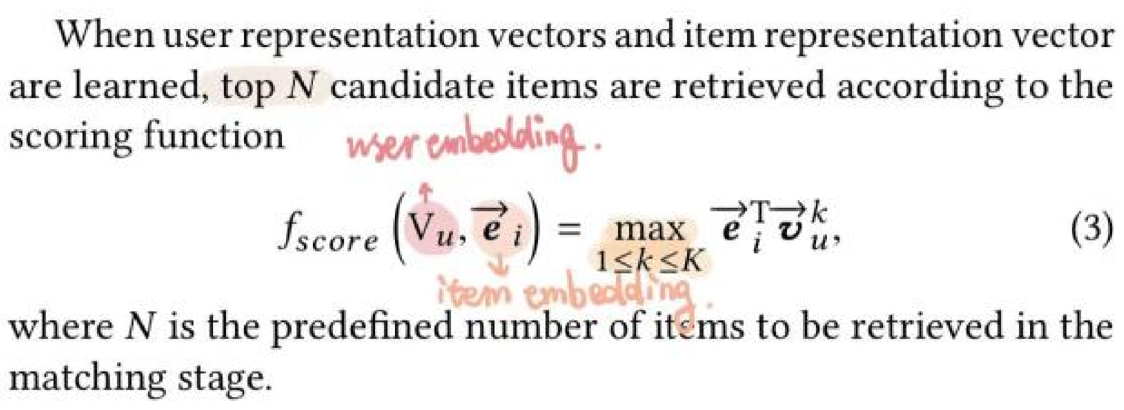

历史物品
用户profile
target item

a tuple $(\mathcal{I}_u, \mathcal{P}_u, \mathcal{F}_i)$, where $\mathcal{I}_u$ denotes the set of items interacted by user u (also called user behavior), $\mathcal{P}_u$ the basic profiles of user u (like user gender and age), $\mathcal{F}_i$ the features of target item (such as item id and category id).

MIND的任务就是要根据用户的基础特征(gender,age...)以及用户历史行为来**计算她的embedding。注意这里的embedding是K个向量，它们代表着K个不同类的兴趣**。一个用户需要用向量组{v1,...vk}表示。

同时，对于亿级的商品，也对每个商品做embedding，为ei. $e_i = f_{item}(\mathbb{F}_i)$

召回阶段其实就是从这亿级商品中，选择top N(N=1000+)个和用户最为相关的商品。其相关性度量如下,即item embedding和K个user embedding的最大相似度（Maxsim）：



When user representation vectors and item representation vector are learned, top N candidate items are retrieved according to the scoring function

$$f_{score}\left(\overset{\text{user embedding.}}{V_u}, \underset{\text{item embedding}}{\vec{e}_i}\right) = \max_{1\le k\le K} \vec{e}_i^{\,T} \vec{v}_u^{\,k}, \quad (3)$$

where N is the predefined number of items to be retrieved in the matching stage.

MIND模型网络结构:

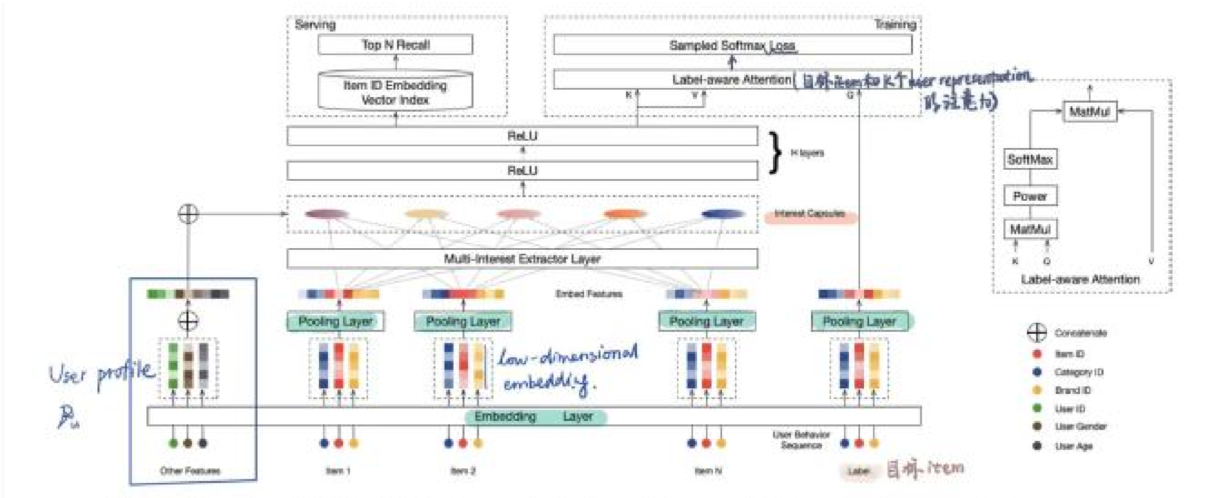
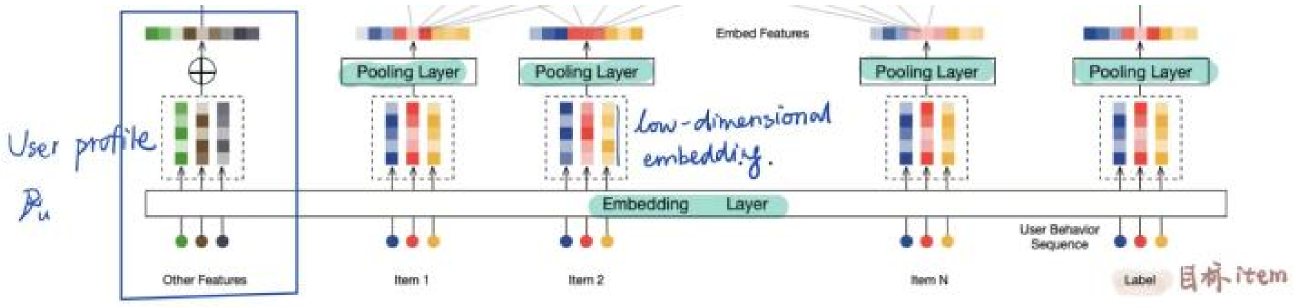


Figure 2: Overview of MIND. MIND takes user behaviors with user profile features as inputs, and outputs user representation vectors for item retrieval in the matching stage of recommendation. Id features from input layer are transformed into embeddings through the embedding layer, and embeddings of each item are further averaged by a pooling layer. User behavior embeddings are fed into the multi-interest extractor layer, which produces interest capsules. By concatenating interest capsules with user profile embedding and transforming the concatenated capsules by several ReLU layers, user representation vectors are obtained. During training, an extra label-aware attention layer is introduced to guide the training process. At serving, the multiple user representation vectors are used to retrieve items through an approximate nearest neighbor lookup approach.

3.2 Embedding & Pooling Layer

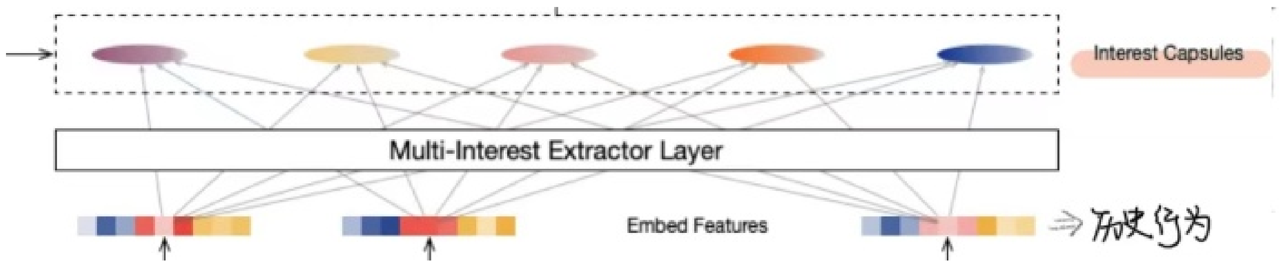
Embedding & Pooling 层对应图中的最底层:



其中user profile的embedding做concat, 所有item的各个特征embedding做avg-pooling，得到一个item的embedding。

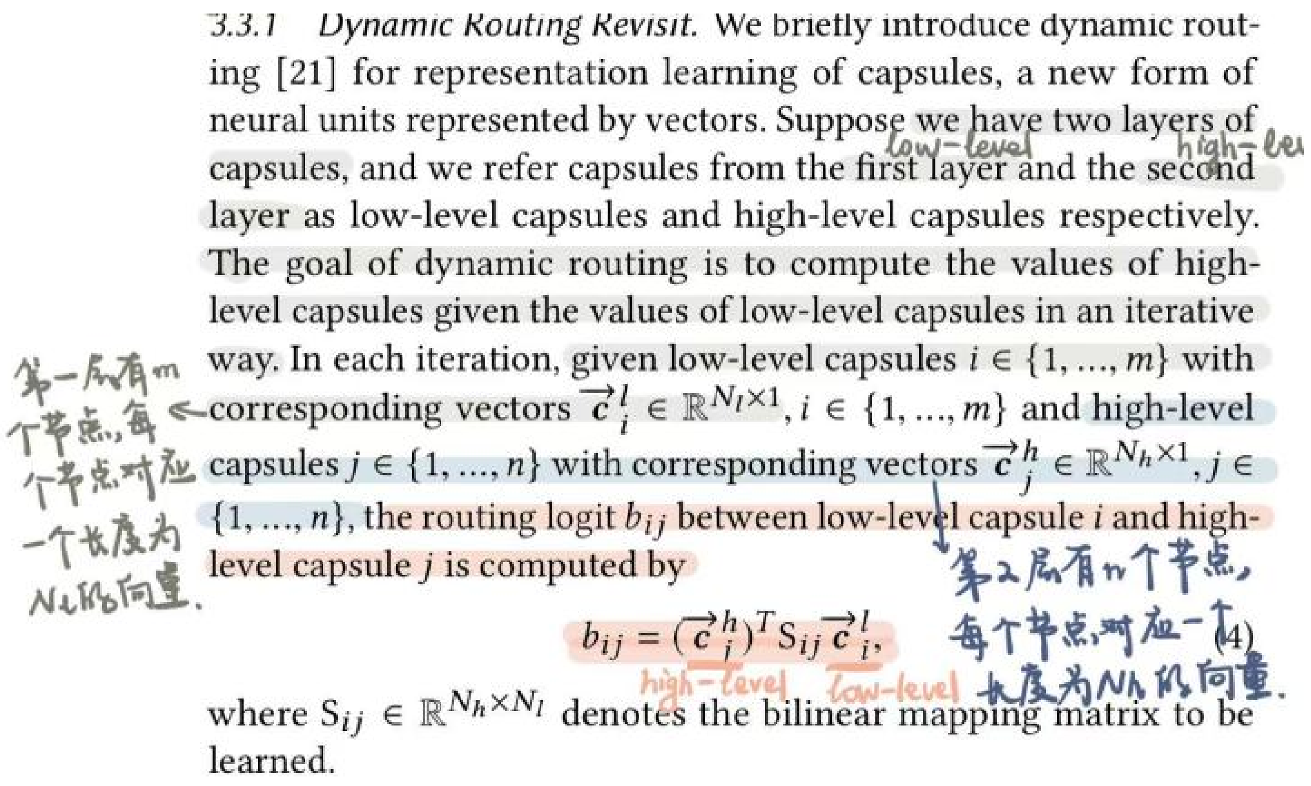
3.3 Multi-Interest Extract Layer

这部分介绍Multi-interest Extract Layer，这是用来给历史行为进行**聚类**，并且对每一个聚类生成一个embedding。这里的方法来源于dynamic routing for representation learning in capsule network, 因此先介绍**动态路由(dynamic routing)**。



3.3.1 动态路由

对于一层网络，第一层有m个节点，每个节点对应着一个长度为 N_l 的向量；第二层有n个节点，每个节点对应一个长度为 N_h 的向量。动态路由试图根据第一层的节点向量，通过迭代的方法，得到第二层的节点向量：



3.3.1 Dynamic Routing Revisit. We briefly introduce dynamic routing [21] for representation learning of capsules, a new form of neural units represented by vectors. Suppose we have two layers of capsules, and we refer capsules from the first layer and the second layer as low-level capsules and high-level capsules respectively. The goal of dynamic routing is to compute the values of high-level capsules given the values of low-level capsules in an iterative way. In each iteration, given low-level capsules $i \in \{1, ..., m\}$ with corresponding vectors $\vec{c}_i^{\,l} \in \mathbb{R}^{N_l \times 1}, i \in \{1, ..., m\}$ and high-level capsules $j \in \{1, ..., n\}$ with corresponding vectors $\vec{c}_j^{\,h} \in \mathbb{R}^{N_h \times 1}, j \in \{1, ..., n\}$, the routing logit b_{ij} between low-level capsule i and high-level capsule j is computed by

$$b_{ij} = (\vec{c}_j^{\,h})^T S_{ij} \vec{c}_i^{\,l}, \tag{4}$$

where $S_{ij} \in \mathbb{R}^{N_h \times N_l}$ denotes the bilinear mapping matrix to be learned.

low-level
high-lev
第一层有m个节点，每个节点对应一个长度为N_l的向量.
第2层有n个节点，每个节点对应一个长度为N_h的向量.
high-level
low-level

实际上，每轮迭代都是把底层的向量c_i先经过MLP降维($N_l \rightarrow N_h$维)，然后通过加权求和得到上层的一个向量c_j。

这个过程主要分为三步：

- 计算权重 w_{ij}. 首先，我们先来计算b_{ij}, 即从底层向量c_i到上层向量c_j的权重。$b_{ij} = (\vec{c_j} S_{ij} \vec{c_i})$, 是一个标量。然后，对底层的m个向量对应的$b_{ij}$求softmax：

$$w_{ij} = \frac{\exp b_{ij}}{\sum_{k=1}^{m} \exp b_{ik}}.$$

- 第二步，进行加权平均。把底层的m个向量以权重w_{ij}进行加权求和，得到上层向量z_j.

$$\vec{z}_j^{\,h} = \sum_{i=1}^{m} w_{ij} S_{ij} \vec{c}_i^{\,l},$$

low-level

- 第三步，把z_j进行归一化("squash")，得到c_j：

更新cj

$$\vec{c}_j^{\,h} = squash(\vec{z}_j^{\,h}) = \frac{\left\|\vec{z}_j^{\,h}\right\|^2}{1+\left\|\vec{z}_j^{\,h}\right\|^2} \frac{\vec{z}_j^{\,h}}{\left\|\vec{z}_j^{\,h}\right\|}.$$

迭代三次达到收敛。

3.3.2 B2I (Behavior to Interest) 动态路由

针对本文描述的情景，对原始的动态路由方法进行3方面改进：

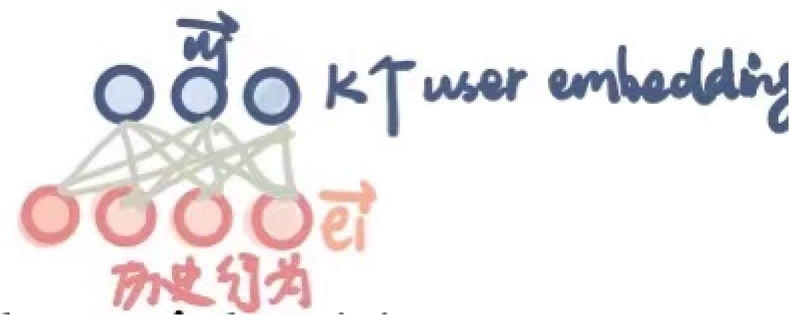
（1）mapping矩阵S对于所有用户、所有节点都**【共享】**

Shared bilinear mapping matrix. We use fixed bilinear mapping matrix S instead of a separate bilinear mapping matrix for each pair of low-level capsules and high-level capsules in original dynamic routing due to two considerations. On the one hand, user behaviors are of variable-length, ranging from dozens to hundreds for Tmall users, thus the use of fixed bilinear mapping matrix is generalizable. On the other hand, we hope interest capsules lie in the same vector space, but different bilinear mapping matrice would map interest capsules into different vector spaces. Thus, the routing logit is calculated by

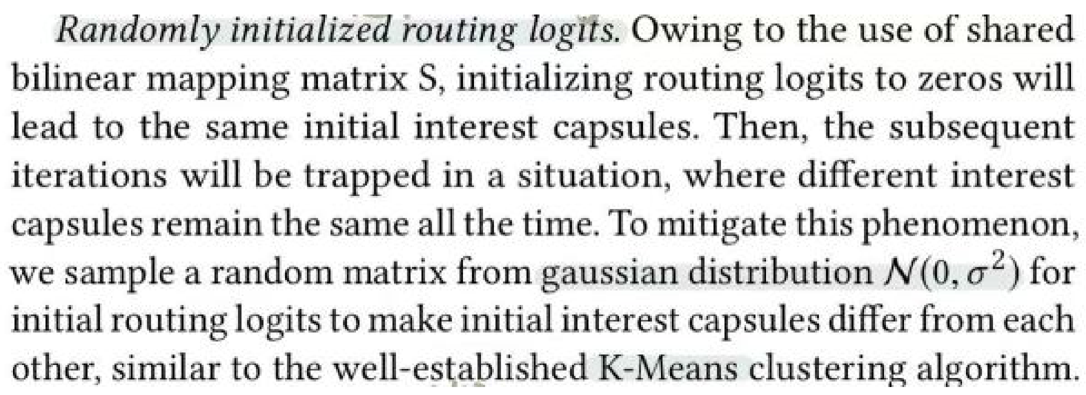
user表征 历史物品 Bo. 历史物品. 用户有K个embedding K个聚类)

$$b_{ij} = \vec{u}_j^{\,T} S \vec{e}_i, \qquad i \in \mathcal{I}_u, j \in \{1, \ldots, K\}, \tag{8}$$

where $\vec{e}_i \in \mathbb{R}^d$ denotes the embedding of behavior item i, $\vec{u}_j \in \mathbb{R}^d$ the vector of interest capsule j. The bilinear mapping matrix $S \in \mathbb{R}^{d \times d}$ is shared across each pair of behavior capsules and interest capsules.

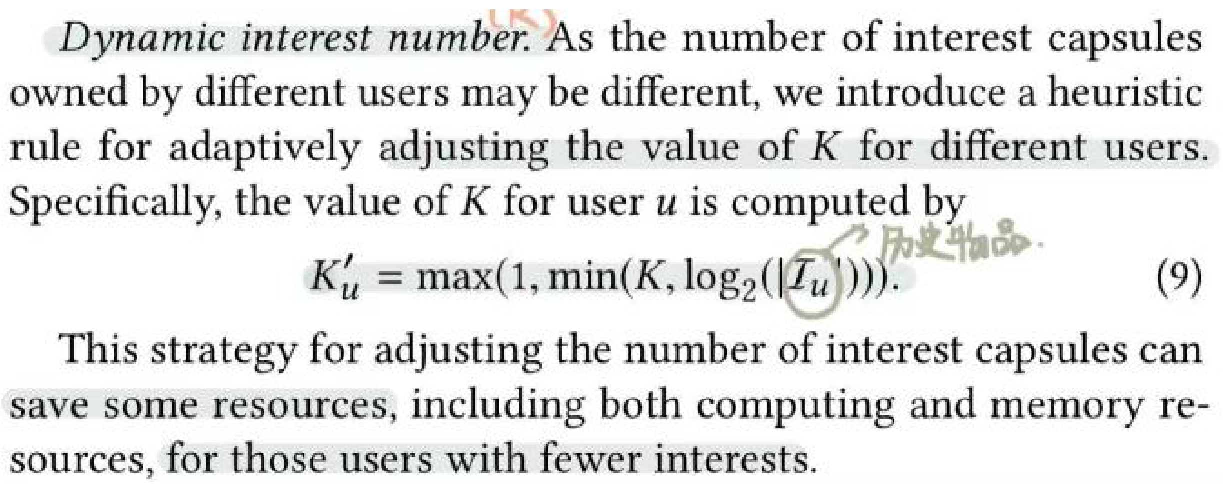


（2）随机初始化，而不是初始化为全零



Randomly initialized routing logits. Owing to the use of shared bilinear mapping matrix S, initializing routing logits to zeros will lead to the same initial interest capsules. Then, the subsequent iterations will be trapped in a situation, where different interest capsules remain the same all the time. To mitigate this phenomenon, we sample a random matrix from gaussian distribution $\mathcal{N}(0, \sigma^2)$ for initial routing logits to make initial interest capsules differ from each other, similar to the well-established K-Means clustering algorithm.

（3）自适应调整用户兴趣类簇个数K。如果历史物品个数太少，应该将K调小。



Dynamic interest number. (K) As the number of interest capsules owned by different users may be different, we introduce a heuristic rule for adaptively adjusting the value of K for different users. Specifically, the value of K for user u is computed by

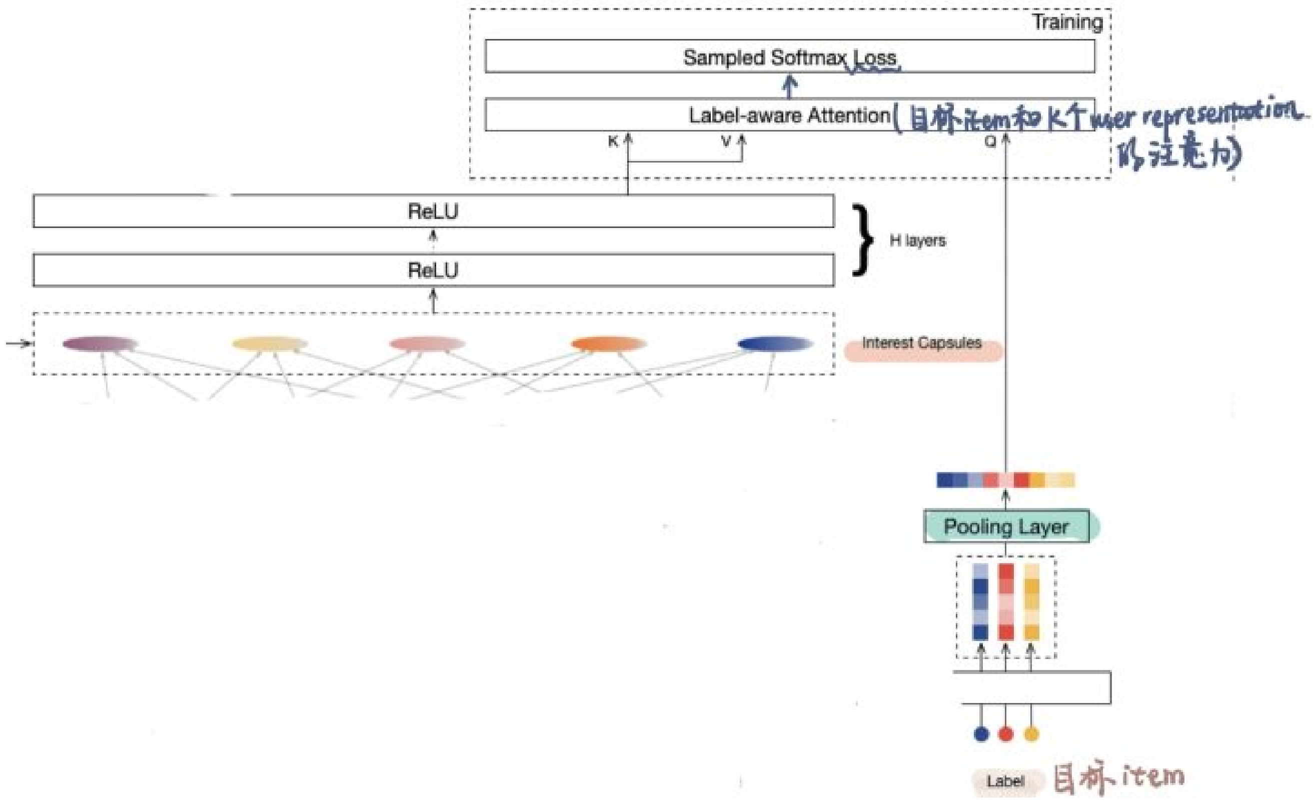
历史物品.

$$K'_u = \max(1, \min(K, \log_2(|\mathcal{I}_u|))). \tag{9}$$

This strategy for adjusting the number of interest capsules can save some resources, including both computing and memory resources, for those users with fewer interests.

3.4 Label-aware Attention Layer

label-aware attention layer对应图中的这个部分：



在训练阶段，使用label-aware attention来**计算target item和K个用户兴趣embedding的相关性**，赋予不同的权重。具体地，就是把target item作为Query，把K个兴趣聚类capsule作为Key和Value，求self-attention，这样我们就得到了user K个兴趣聚类更好的加权融合。这样，我们就得到了user端的embedding表示，这就是user塔的输出。item塔的输出就是target item embedding。

3.5 Training & Serving

1）训练阶段：

用户u购买/观看商品i的概率为：

item i as

$$\Pr(i|u) = \Pr\left(\vec{e}_i | \vec{v}_u\right) = \frac{\exp\left(\vec{v}_u^{\mathrm{T}} \vec{e}_i\right)}{\underbrace{\sum_{j \in \mathcal{I}}}_{\text{所有物品.}} \exp\left(\vec{v}_u^{\mathrm{T}} \vec{e}_j\right)}. \tag{10}$$

Then, the overall objective function for training MIND is

损失函数：

$$L = -\sum_{(u,i) \in \mathcal{D}} \log \Pr(i|u), \tag{11}$$

正样本集

由于分母要计算所有的商品，这实在是太多了。所以，我们采用类似word2vec中negative sampling的思想，抽取一些**负样本**，采用sampled softmax方法（和Youtube双塔一样）。同时使用Adam优化器。

2）serving阶段

训练之后，整个模型（除了Label-aware attention部分）都用于计算user embedding。我们把user的行为序列和profile都feed进模型，得到他的K个兴趣聚类capsule。然后，对于这每一个兴趣聚类embedding，都从item池中召回N个最接近的item，作为candidate。

当user有了新的行为，线上过来的时候他的兴趣聚类表征就会相应的发生变化，所以MIND可以进行实时的个性化召回。

4. Experiments

公开数据集：Amazon Books

工业数据集：TmallData, 200万用户行为

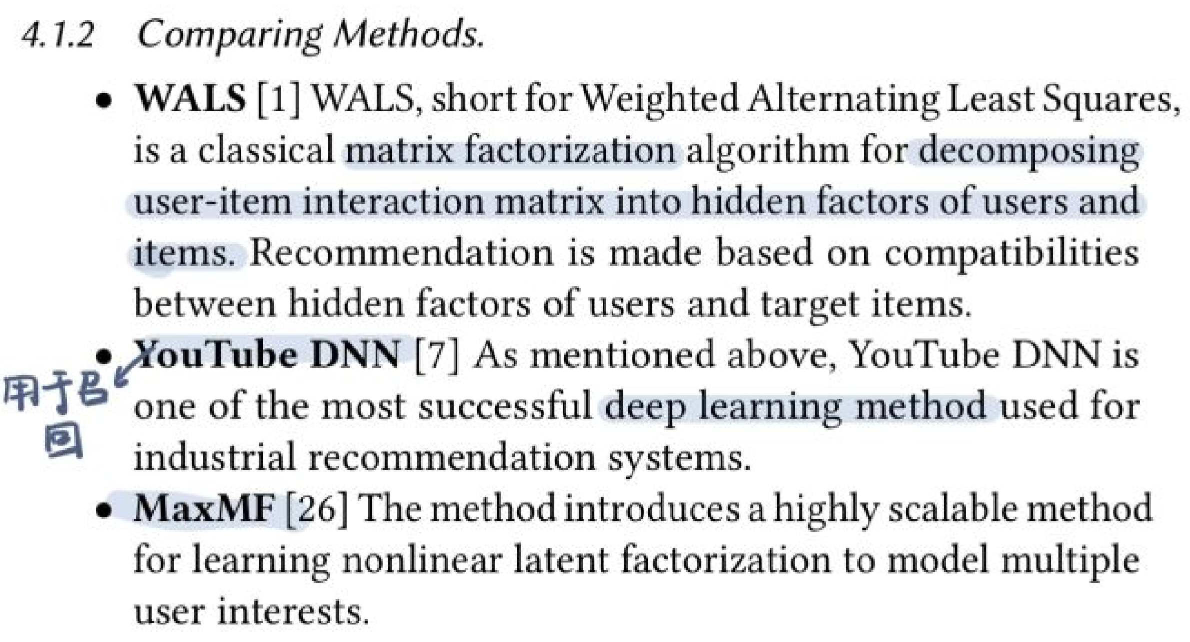
选择next-item prediction problem来进行召回，用hit-rate来判断召回算法的好坏：

算法选出的topN

$$HitRate@N = \frac{\sum_{(u,i)\in\mathcal{D}_{test}} I(\text{target item occurs in top } N)}{|\mathcal{D}_{test}|} \qquad (12)$$

实际的交互物品.

Baseline：



4.1.2 *Comparing Methods.*

- **WALS** [1] WALS, short for Weighted Alternating Least Squares, is a classical matrix factorization algorithm for decomposing user-item interaction matrix into hidden factors of users and items. Recommendation is made based on compatibilities between hidden factors of users and target items.
- **YouTube DNN** [7] As mentioned above, YouTube DNN is one of the most successful deep learning method used for industrial recommendation systems.
- **MaxMF** [26] The method introduces a highly scalable method for learning nonlinear latent factorization to model multiple user interests.

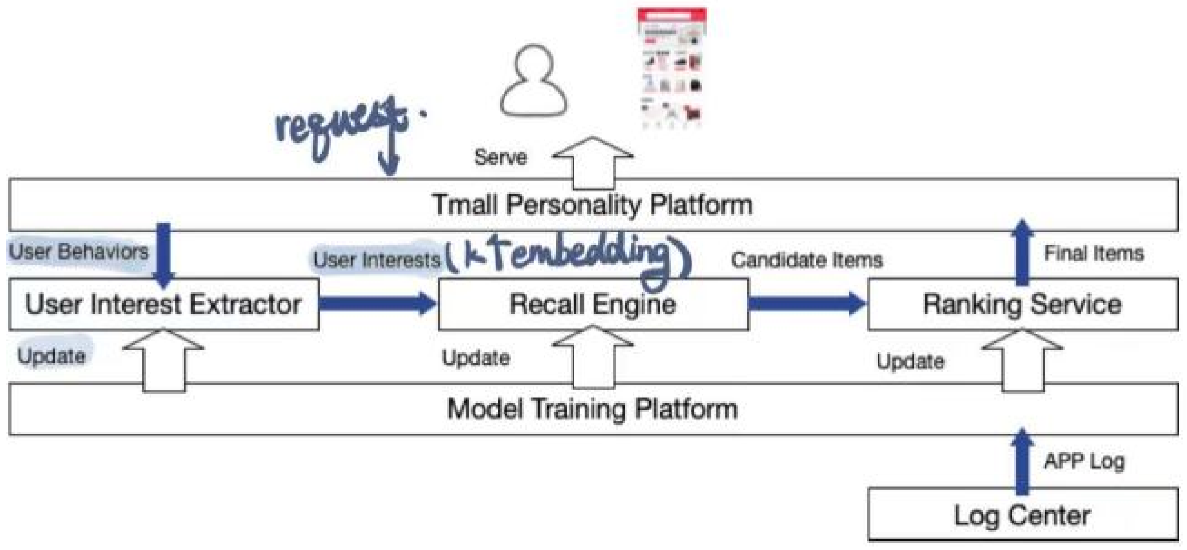
用于召回

Online Experiment：

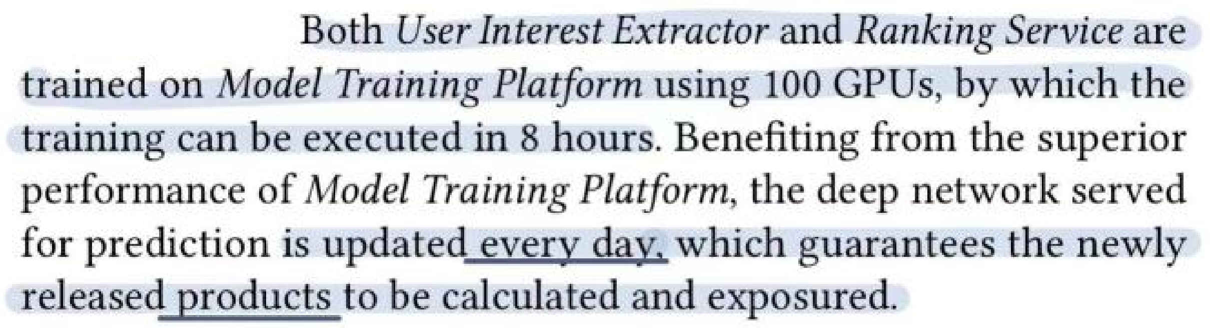
Baseline 选择item-based CF & Youtube DNN. 使用A/B test, 使用同样的ranking策略，最后比较CTR。

5. System Depolyment

用户打开app时，向Tmall Personality Platform提一个request。于是Tmall Personality Platform去找到该用户的bahaviors，经过User Interest Extractor提取出K个用户兴趣embedding。用此召回1000+个candidate items。经过ranking进行排序，取top-100展示。



Model Training Platform:



Both *User Interest Extractor* and *Ranking Service* are trained on *Model Training Platform* using 100 GPUs, by which the training can be executed in 8 hours. Benefiting from the superior performance of *Model Training Platform*, the deep network served for prediction is updated every day, which guarantees the newly released products to be calculated and exposured.

【总结】

MIND可以看作是Youtube DNN的多兴趣优化，当K = 1时，MIND退化为Youtube DNN。

MIND的思想和ColBERT也十分类似，都是用多向量来表示query，然后求query和document表示的MaxSim，来得到query和document的相似度。毕竟，把一个query/document压缩成一个向量会造成信息的损失。