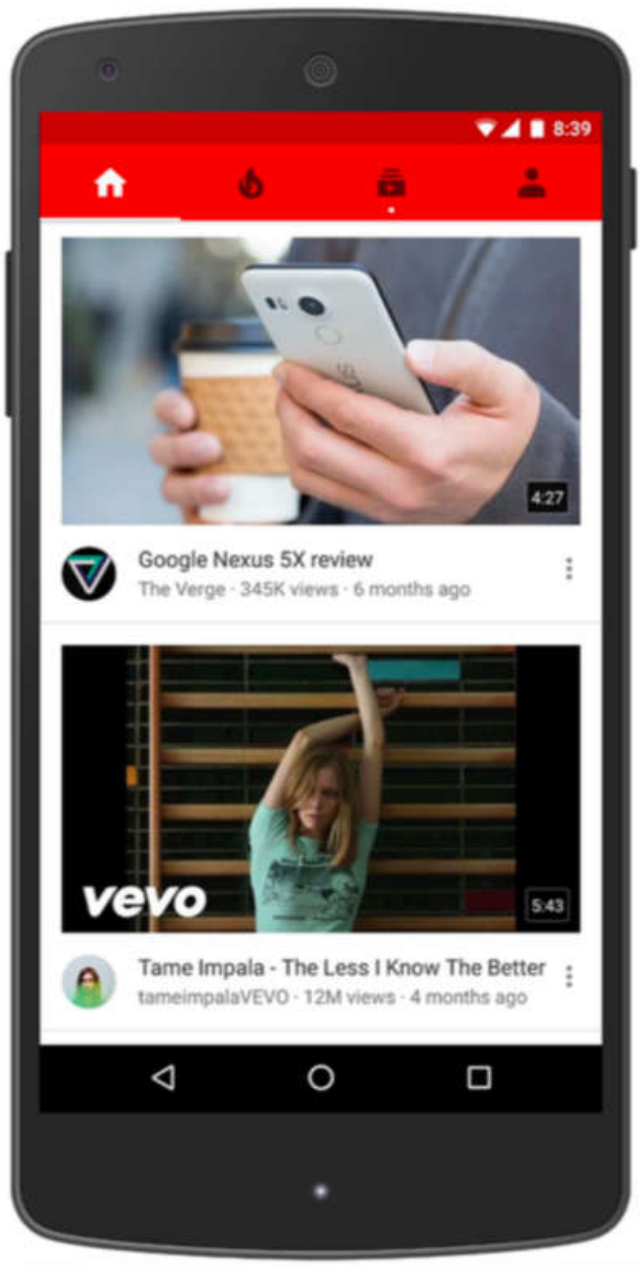
关于'Deep Neural Networks for YouTube Recommendations'的一些思考和实现

2018-06-26 | 深度学习

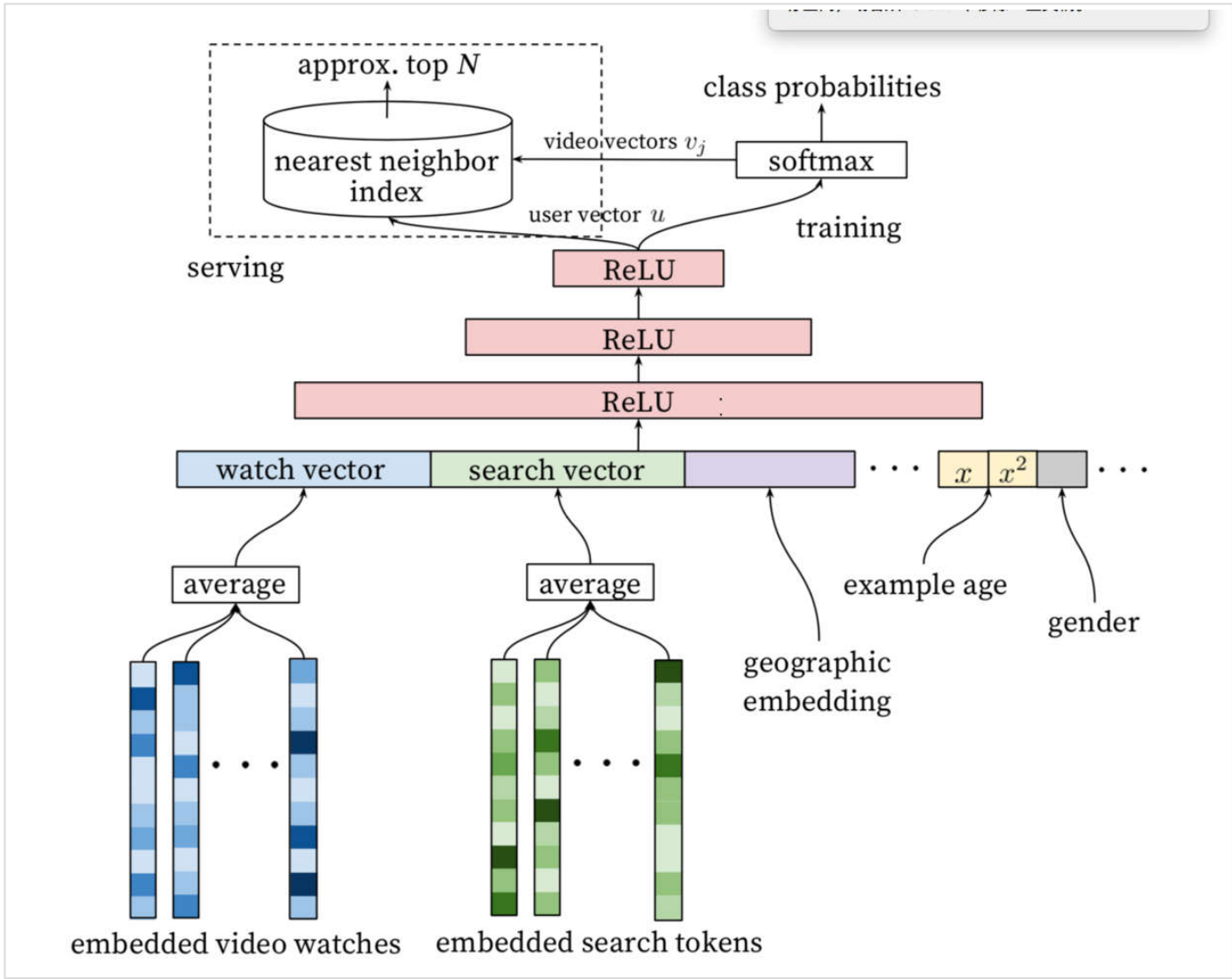
4,104 字 | 15 分钟



论文 Deep Neural Networks for YouTube Recommendations 来自google的YouTube团队，发表在16年9月的RecSys会议。我想应该很多人都读过，之前参与了公司的推荐系统优化的项目，本来想从各大搜索引擎中寻找到现成的分析，但是出人意料的一无所获。Github上的代码实现也出奇的少以及不清晰，所以就借着这个机会和大家分享一下自己做的过程中的一些理论心得、工程坑、代码实现等等。

本文基于大家对Deep Neural Networks for YouTube Recommendations已经完成通读的基础上，不会做细致的论文解析，只会涉及到自己实现过程中的一些总结，如果没有论文了解，会非常不易理解。

系统概览



上面这张图可以说是比较详细的涵盖了基础框架部分，整体的模型的优点我就不详述了，包括规模容纳的程度大啊、鲁棒性好啊、实时性优秀啊、延展性好啊等等，网上很多水字数的文章很多，我们主要总结几个愿论文上的亮点和实际去做的时候需要注意的地方：

- DNN网络可以怎么改
- 负采样的“避坑”
- example age有没有必要构造
- user feature的选择方向
- attention 机制的引入
- video vectors的深坑
- 实时化的选择

整体上来说，G厂的这套算法基于的是两个部分：matching+ranking，这个的也给我们带来了更大的工作量，在做的时候，分成两个部分，我们在实际处理的时候，通过recall rate来判断matching部分的好坏，通过NDCG来判断排序部分的好坏。总体如下：

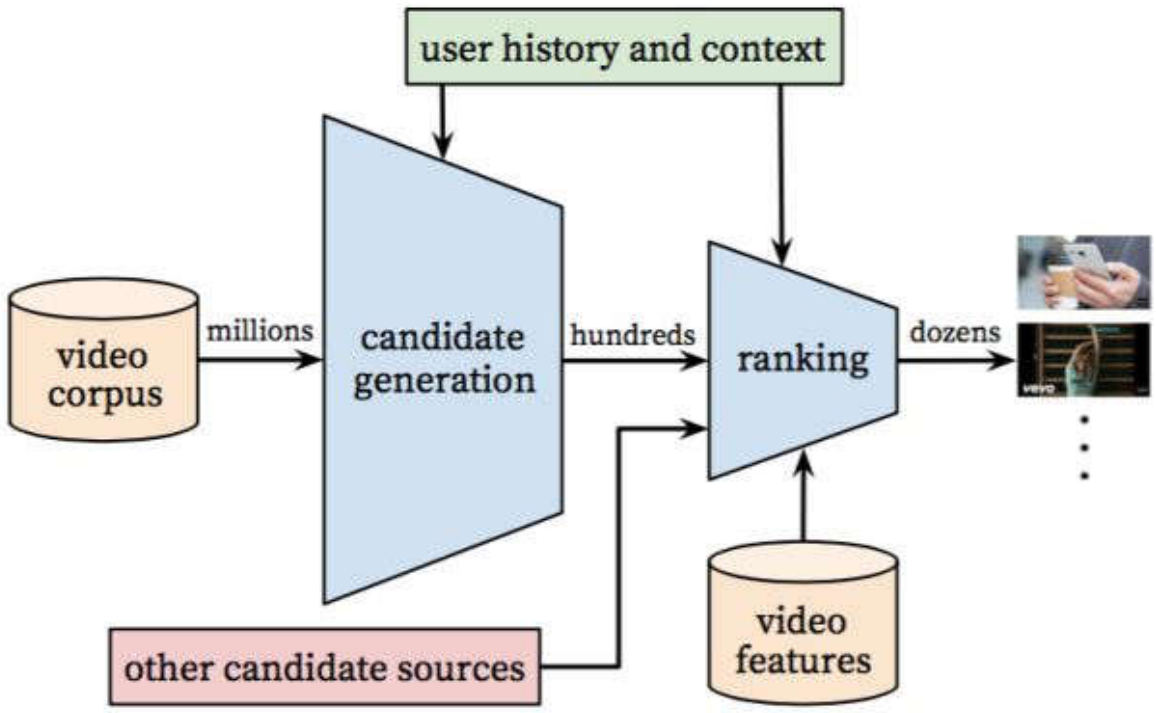


Figure 2: Recommendation system architecture demonstrating the "funnel" where candidate videos are retrieved and ranked before presenting only a few to the user.

candidate generation就是我们matching的模块，目的是把百万级的商品、视频筛选出百级、千级的可排序的量级；再通过ranking模块，选出十位数的展示商品、视频作为最后的推送内容。之所以把推荐系统划分成Matching和Ranking两个阶段，主要是从性能方面考虑的。Matching阶段面临的是百万级，而Ranking阶段的算法则非常消耗资源，不可能对所有目标都算一遍，而且就算算了，其中大部分在Ranking阶段排名也很低，也是浪费计算资源。

Matching & Ranking Problems

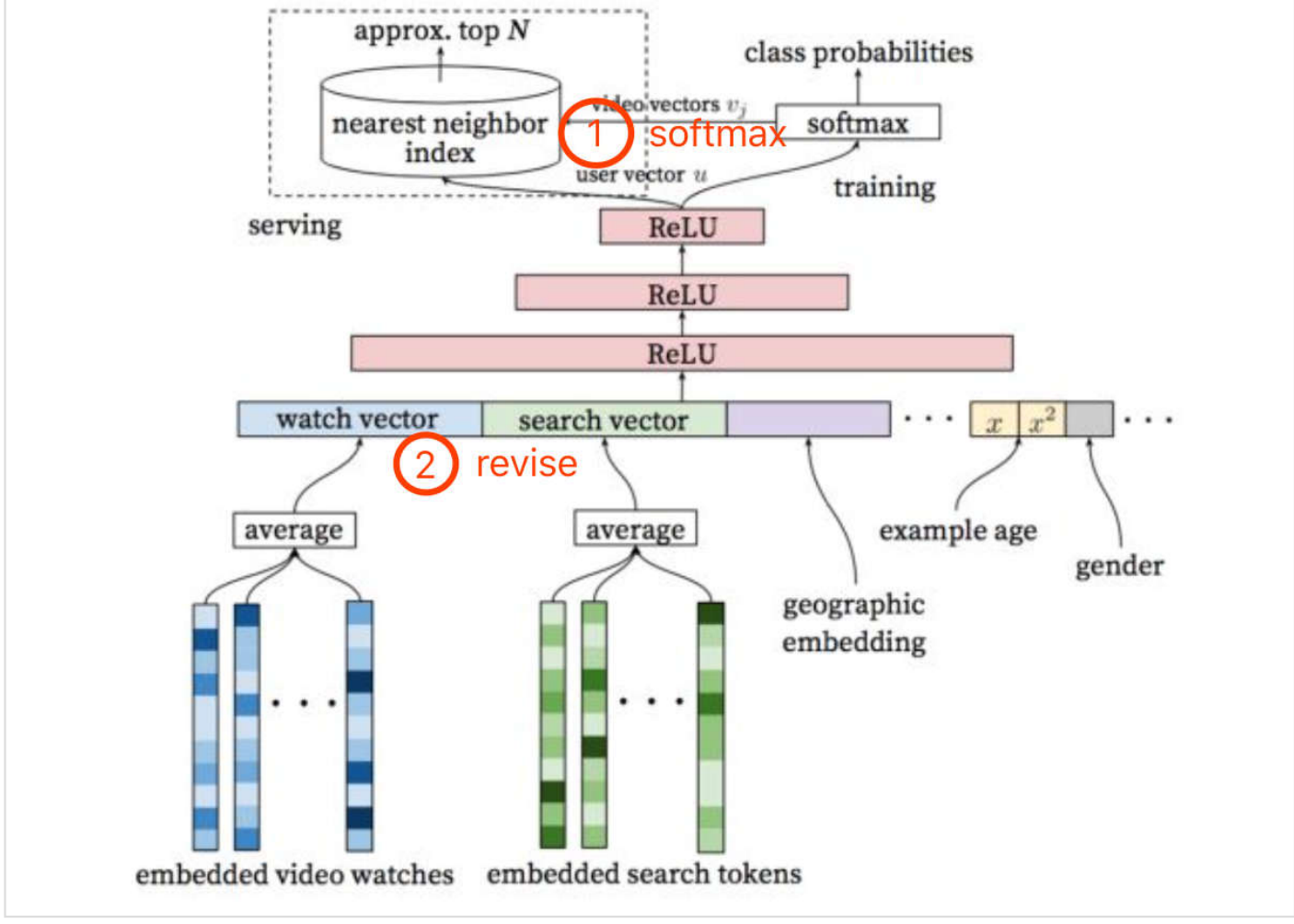
首先，我们都知道，G厂给出的这个解决方案用的就是基于DNN的超大规模多分类思想，即在时刻t，为用户U（上下文信息C）在视频库V中精准的预测出视频i的类别（每个具体的视频视为一个类别，i即为一个类别），用数学公式表达如下：

$$P(w_t = i|U, C) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

很显然上式为一个softmax多分类器的形式。向量u是信息的高纬“embedding”，而向量v则是视频 j 的embedding向量，通过u与v的点积大小来判断用户与对应视频的匹配程度。所以DNN的目标就是在用户信息和上下文信息为输入条件下学习视频的embedding向量v，从而得到用户的向量u。

说完基本思想，让我们看看实际的效果对比：

DNN网络可以怎么改：softmax及revise的考虑



如我图中两处红色标记，论文中虽然给出了模型的整体流程，但是却没有指明，1处的video vectors需要单独的embedding还是沿用最下方的embedded video watches里面的已经embedding好的结果，我们称之为softmax问题；2处论文没有提及一个问题，就是在固定好历史watch的长度，比如过去20次浏览的video，可能存在部分用户所有的历史浏览video数量都不足20次，在average的时候，是应该除以固定长度（比如上述例子中的20）还是选择除以用户真实的浏览video数量，我们称之为revise问题。

根据我们的数据实测，效果对比如下：

网络	epoch	rate	allhit	allclick
softmax+revise	3	0.15%	189	129246
softmax+norevise	3	0.12%	161	129246
nosoft+revise	3	1.46%	1882	129246
nosoft+norevise	3	1.04%	1350	129246

`nosoft`:沿用最下方的`embedded video watches`里面的已经`embedding`好的结果

`revise`:除以用户真实的浏览`video`数量

我们尝试的去探求原因发现，nosoft比softmax好的原因在于user vector是由最下方的embedded video watches里面的已经embedding好的结果进行多次FC传递得来的，如果新增一个video embedded vector 的话，和FC传递得到的u vector的点积的意义就难以解释；revise比norevise好的原因是，实际在yoho!buy的购物场景下，用户的点击历史比较我们实际选取的window size要短不少，如果所有的用户都除以固定长度的话，大量的用户history click average的vector大小接近于0。

DNN网络可以怎么改：神经元死亡及网络的内部构造

这是一个异常恶心还有没什么好方法的问题，在刚开始做的时候，我们遇到了一个常见的问题，神经元批量死亡的问题。在增加了batch normalization、clip_by_global_norm和exponential_decay learning rate 后有所缓解。

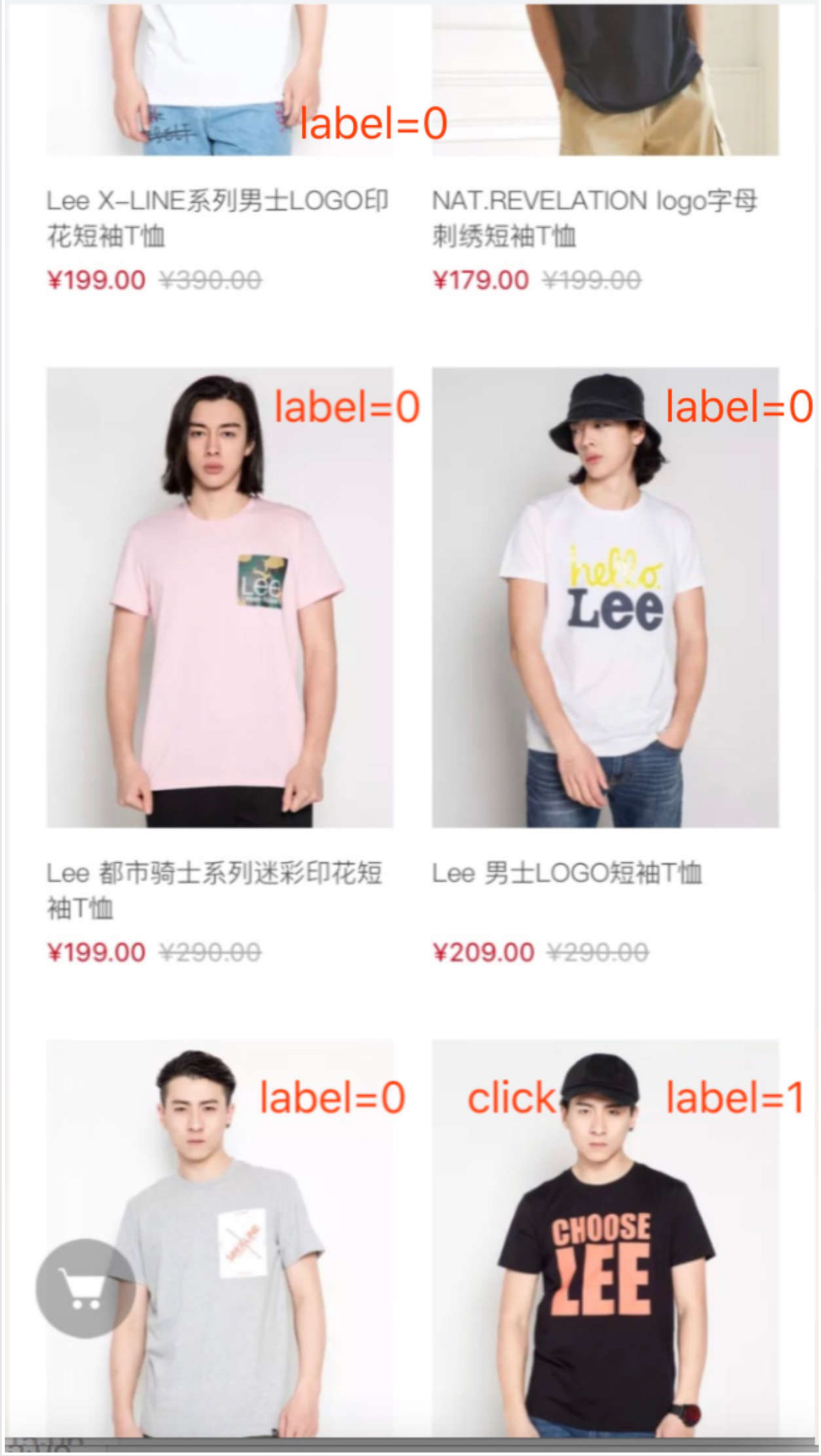
网络结构的变化比较常规，对比场景的激活函数，参考了论文中推荐的深度、节点数，效果对比如下：

model	network	click_rate	uid_rate
RNN	-	0.03	0.2359
nosoft_revise	relu+relu+leakyrelu+leakyrelu	0.0033	-
nosoft_revise	relu+relu+leakyrelu+leakyrelu	0.0005	-
nosoft_revise	relu+relu+leakyrelu	0.0157	-
nosoft_revise	relu+relu+leakyrelu+leakyrelu	0.0153	0.2124
nosoft_revise	relu+relu+leakyrelu+leakyrelu	0.0173	0.2220
nosoft_revise	relu+relu+leakyrelu+leakyrelu	0.0147	0.2044

虽然我们看到增加网络的深度（3–>4）一定程度上会提高模型的命中率，增加leakyrelu的一层网络也可以有些许的提升，但是总的来说，对模型没有啥大的影响，所以在之后的实际模型中，我们选择了原论文中relu+relu+relu，1024+512+256的框架。

负采样的“避坑”

我们都知道，算法写起来小半天就可以搞定，但是前期的数据处理要搞个小半个月都不一定能出来。作为爱省事的我，为了快速实现算法，没有重视负采样的部分，采取了列表页点击为label=1，未点击为label=0的方式，详情如下：



看上去没什么问题，省略了从全量样本中抽样作为负样本的复杂过程，实际上，我把代码狂改了n边效果也一直维持在1.57%，可以说是没有任何提升，在此过程期间，我还是了拿用户的尾次点击（last_record）进行训练，拿了有较多行为的用户的尾次点击（change_last_record）进行训练，效果很感人：

data_format	model	network	click_rate	uid_rate	location
base_data	**nosoft_revise**	**relu+relu+leakyrelu(elu)**	**0.0028**	-	-
base_data	nosoft_revise	leakyrelu+leakyrelu+leakyrelu	0.0026	-	-
base_data	nosoft_revise	relu+relu+relu	0.0015	-	-
base_data	**nosoft_revise**	**relu+relu+leakyrelu+leakyrelu**	**0.0033**	-	-
base_data	nosoft_revise	relu+relu+relu+leakyrelu	0.0010	-	-
last_record	nosoft_revise	relu+relu+leakyrelu	0.0054	-	-
change_last_record	**nosoft_revise**	**relu+relu+leakyrelu**	**0.0157**	-	-

在我孤注一掷一致，选择按照原论文中说的，每次label=0的我不拿展现给用户但是用户没有点击的商品，而是随机从全量商品中抽取用户没有点击过的商品作为label=0的商品后，奇迹发生了：

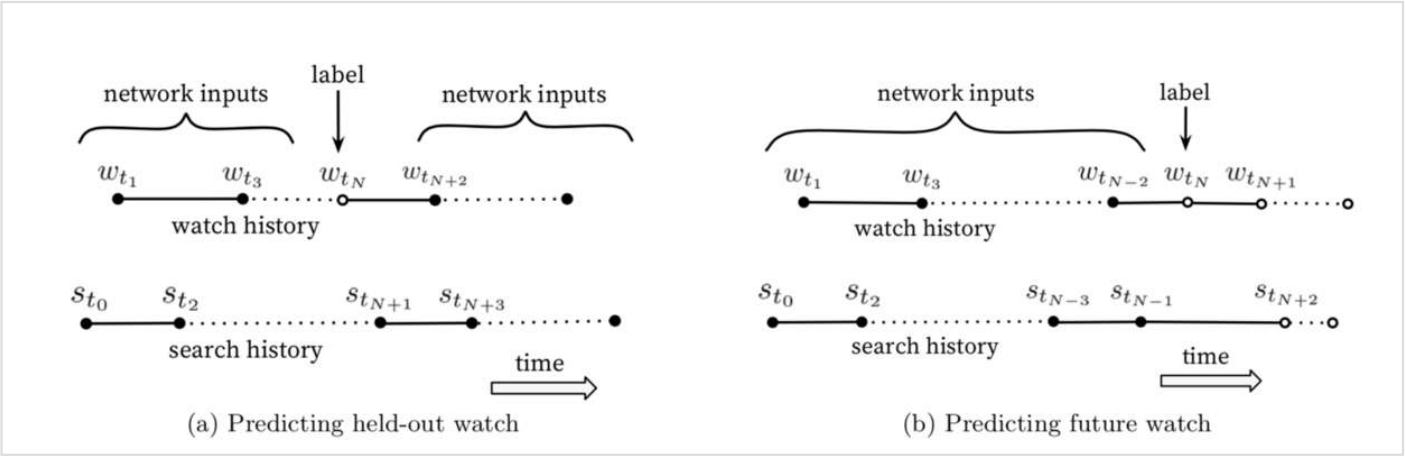
指标	base
hit	9912
click	358674
rate	2.8%
hit uid	6972
uid rate	20.4%
top 1	1.0%
top 5	2.9%
top 10	4.5%
top 20	7.0%

事后我仔细分析了原因：

a.在当次展现的情况下，虽然用户只点击了click商品，其他商品没有点击，但是很多用户在后续浏览的时候未click的商品也在其他非列表页的地方进行click，我实际上将用户感兴趣的商品误标记为了负样本

b.后来我咨询看了论文，也发现了原论文中也提及到，展现商品极有可能为热门商品，虽然该商品该用户未点击，但是我们不能降低热门商品的权重（通过label=0的方式），实际上最后的数据也证明了这一点

c.“偷窥未来的行为”，如下图，原论文中指出input构造时候不能拿还未发生的点击，只能拿label=1产生时之前的所有历史点击作为input；同理，在构造label=0的时候，只能拿在label=0的时候已经上架的商品，由于训练时间的拉长，不能偷窥label=1发生时还未上架的商品作为label=0的负样本



example age有没有必要构造

首先，先稍微解释一下我对example age的概念的理解。所有的训练数据其实都是历史数据，距离当前的时刻都过去了一段时间，站在当前来看，距离当前原因的数据，对当前的影响应该是越小的。就比如1年前我买了白色的铅笔，对我现在需要不需要再买一支黑色的钢笔的影响是微乎其微的。而example age其实就是给了每一条数据一个权重，引用一下原论文的描述 `In (5b), the example age is expressed as tmax – tN where tmax is the maximum observed time in the training data` ，我这边采取了(tmax – tN)/tmax的赋权方式：

指标	base	exampe age
hit	9912	9702
click	358674	358674
rate	2.8%	2.7%
hit uid	6972	6972
uid rate	20.4%	20.3%
top 1	1.0%	1.0%
top 5	2.9%	2.8%
top 10	4.5%	4.6%
top 20	7.0%	7.0%

很悲催的是，直观的离线训练数据并没有给出很直观的效果提升，但是由于评估机制的问题（我们后面会说到），我会在实际上线 做abtest的时候重新验证我的这个example age的点，但是可以肯定的是，理论和逻辑上，给样本数据进行权重的更改，是一个可以深挖的点，对线上的鲁棒性的增强是正向的。

user feature的选择方向

很不幸的是，在这一块的提升，确实没有论文中说的那么好，对于整个网络的贡献，以我做的实际项目的结果来说，history click embedded item > history click embedded brand > history click embedded sort > user info > example age > others。不过，因为时间、数据质量、数据的真实性的原因，可能作为原始input的数据构造的就没有那么好。这边主要和大家说两个点：

1.topic数据

原论文中在第四节的RANKING中指出:

```
We observe that the most important signals are those that describe a user’s previous interaction with the item itself and other similar items, matching others’ experience in ranking ads
```

论文中还举出了比如用户前一天的每个频道（topic）的浏览视频个数，最后一次浏览距今时间，其实说白了就是强调了过去的行为汇总对未来的预测的作用，认为过去的行为贯穿了整体的用户点击轨迹。

除此之外，G厂大佬还认为一些用户排序性质的描述特征对后面的ranking部分的提高也是蛮重要的，这边还举出了用户视频评分的例子，更多的内容大家可以自己去看一下原论文的部分，应该都会有自己的体会。

回到我们的项目，因为yoho!buy是电商，我类比着做了用户每个类目（裤子、衣服、鞋子...）的历史浏览点击购买次数、最后一次点击距今时长等等的topic信息，提升不是很明显。但是在大家做G厂这边论文，准确率陷入困境的时候，可以尝试一下这边的思路。

指标	base	add_basic
rate	2.8%	3.6%
uid rate	20.4%	26.5%
top 1	1.0%	1.1%
top 5	2.9%	4.0%
top 10	4.5%	6.1%
top 20	7.0%	9.2%

2.query infomation

相比于论文中的user information的添加，在实际模型测试中，我们发现，query的information的部分有更多的”遐想”。

原论文中点名指出user language and video language 做为basic info的重要性，这边给出的提升也是相对于user info有明显的增长的：

指标	base	add_user_info4&query_info
rate	2.8%	3.7%
uid rate	20.4%	26.4%
top 1	1.0%	1.1%
top 5	2.9%	3.7%
top 10	4.5%	6.2%
top 20	7.0%	9.3%

有提升也自然有该部分的缺点：

1.语言模型的处理复杂，耗时久

在该部分的处理中，我强行拖着隔壁组的nlp博士和我一起搞了一周，每天都加班的搞去做数据清理，句法分析，语句树解析。如果需要让一个常规做推荐的人去弄，会有各种各样的坑，而且耗时还久

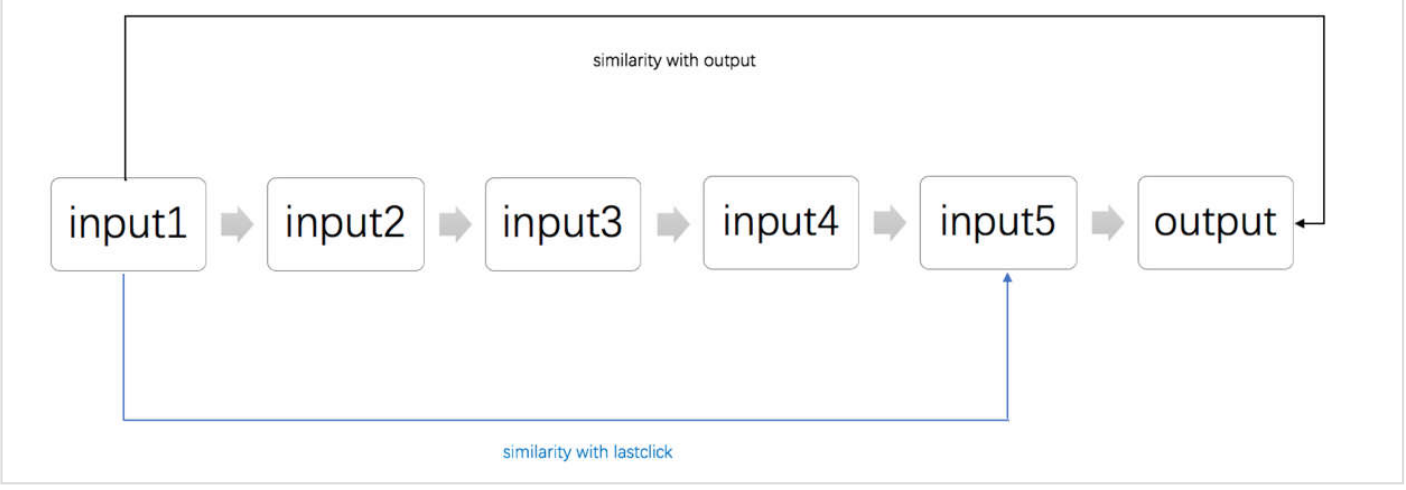
2.语言新增问题

商品的标题这类的文本处理还好，毕竟每日更新的数据存在一个可控的范围，但是用户搜索内容的变化是巨大的，粗略估测一下，一周时间间隔后，原提纯文本数据和新提纯文本数据的交集覆盖率不到78%，这意味着要重复的做nlp工作

attention 机制的引入

attention 机制的引入是我老大的硬性需求，我这边也就做了下，如果不了解attention 机制的朋友，可以阅读以下这边文章：Attention model。

我通俗的解释一下，不准确但是方便理解，Attention model就是让你每一个input与你的output计算一个similarity，再通过这些similarities给出每个input的权重。但是，很明显，我们离线训练还好，既有input也有output，但是线上预测的时候，就没有output了，所以，我们采取了lastclick替代的方式：



指标	base	add attention
rate	2.8%	**3.8%**
uid rate	20.4%	27.2%
top 1	1.0%	1.5%
top 5	2.9%	4.7%
top 10	4.5%	7.1%
top 20	7.0%	10.3%

不得不说，老祖宗传下来的东西确实有独到之处，但是在提升了近1pp的rate代价之下，会有一个让人头疼的问题耗时。因为每一个input的weight需要和output进行一次相似度计算，而且后续还要对计算出的相似度进行处理，原本只需要6-7小时训练完的模型，在我加了3层Multihead Attention后被拖到了一天。数据量还只采样了一半，确实需要斟酌带来的提升与投入的成本之间的平衡问题。

video vectors的深坑

G厂一句话，我们测断腿。这句话不是瞎说的，大家应该还记得一开始我给出的那张图，在最上面有一行不是很明显的小字：video vectors。G厂的大佬们既没有说这些video vectors该怎么构造，也没有说video vectors需不需

要变动，留下了一个乐趣点让大家体验。

刚开始我很傻的用了我们最开始的embedded item作为video vectors，与模型FC出来的user vectors进行点击，计算top items。我来来回回测了一个月，老命都快改没了，最后提升rate到4pp。然而RNN随便跑跑就能到达3pp，我说很不服气的，所以拉着同事一起脑洞了一下，我们之前做图片相似度匹配的时候，喜欢把图片的向量拆成颜色+款式+性别，所以我们就借用了一下，改成了embedded item + embedded brand + embedded sort作为video vectors，历史总是给我们惊喜，效果上一下子就能大到5.2pp左右，这个点的提升应该是得来的最意外的，建议大家在用的时候考虑一下。

指标	base	change user vectors
rate	2.8%	5.17%
uid rate	20.4%	33.33%
top 1	1.0%	2.35%
top 5	2.9%	6.87%
top 10	4.5%	9.86%
top 20	7.0%	13.79%

实时化的选择

实时部署上，我们用了tensor flow serving，没什么好说的，给一下关键代码，大家看下自己仿一下就行，一般自己做做demo不需要，企业级上线才需要，企业级上线的那些大佬可能也比我有更多想法，所以就不展开了。

```
部署及用python作为Client进行调用的测试：
#1.编译服务
bazel build //tensorflow_serving/model_servers:tensorflow_model_server
#2.启动服务
bazel-bin/tensorflow_serving/model_servers/tensorflow_model_server --port=9005 --model_name=
#3.编译文件
bazel build //tensorflow_serving/test:test_client
#4.注销报错的包
注销：/Data/muc/serving/bazel-bin/tensorflow_serving/test/test_client.runfiles/org_tensorflo
参考：https://github.com/tensorflow/serving/issues/421
#5.运行
bazel-bin/tensorflow_serving/test/test_client --server=localhost:9005
```

相关的问题，有大佬已经梳理好了，自取其他可选的一些参数设置：tensorflow serving 参数设置。

还有一些评估技巧，模型之间的对比技巧，这边就不细讲了，可借鉴的意义也不大。

总结

虽然早就读过这篇文章，但是实现之后，发现新收获仍然不少。我特别赞成清淞的一句话:’对于普通的学术论文，重要的是提供一些新的点子，而对于类似google这种工业界发布的paper，特别是带有practical lessons的paper，很值得精读。’
G厂的这个推荐代码和attention model的代码之前是准备放GitHub的，想想还是算了。一是之前也放过很多此代码，也没什么反馈，二是这两个代码自己写也不是很难，可以作为练手项目。

鸣谢

以上我个人在Yoho!Buy团队在实践中的一点总结，不代表公司的任何言论，仅仅是我个人的观点。最后感谢项目推进过程中所有合作方和项目组同学的付出和努力，感谢各个团队各位老大们的支持！溜了溜了。

论文解析

‹ tf.nn.embedding_lookup

Tensorflow在GPU下的Poolallocator Message ›