写在前面

YouTubeDNN模型是2016年的一篇文章，虽然离着现在有些久远， 但这篇文章无疑是工业界论文的典范， 完全是从工业界的角度去思考如何去做好一个推荐系统，并且处处是YouTube工程师留给我们的宝贵经验， 由于这两天用到了这个模型，今天也正好重温了下这篇文章，所以借着这个机会也整理出来吧， 王喆老师都称这篇文章是"神文"， 可见其不一般处。

今天读完之后， 给我的最大感觉，首先是从工程的角度去剖析了整个推荐系统，讲到了推荐系统中最重要的两大模块: 召回和排序， 这篇论文对初学者非常友好，之前的论文模型是看不到这么全面的系统的，总有一种管中规豹的感觉，看不到全局，容易着相。 其次就是这篇文章给出了很多优化推荐系统中的工程性经验， 不管是召回还是排序上，都有很多的套路或者trick，比如召回方面的"example age", "负采样"，"非对称消费，防止泄露"，排序方面的特征工程，加权逻辑回归等， 这些东西至今也都非常的实用，所以这也是这篇文章厉害的地方。

本篇文章依然是以paper为主线， 先剖析paper里面的每个细节，当然我这里也参考了其他大佬写的文章，王喆老师的几篇文章写的都很好，链接我也放在了下面，建议也看看。然后就是如何用YouTubeDNN模型，代码复现部分，由于时间比较短，自己先不复现了，调deepmatch的包跑起来，然后在新闻推荐数据集上进行了一些实验， 尝试了论文里面讲述的一些方法，这里主要是把deepmatch的YouTubeDNN模型怎么使用，以及我整个实验过程的所思所想给整理下， 因为这个模型结构本质上并不是很复杂(三四层的全连接网络)，就不自己在实现一遍啦， 一些工程经验或者思想，我觉得才是这篇文章的精华部分。

引言与推荐系统的漏斗范式

引言部分

本篇论文是工程性论文(之前的DIN也是偏工程实践的论文)， 行文风格上以实际应用为主， 我们知道YouTube是全球性的视频网站， 所以这篇文章主要讲述了YouTube视频推荐系统的基本架构以及细节，以及各种处理tricks。

在Introduction部分， 作者首先说了在工业上的YouTube视频推荐系统主要面临的三大挑战:

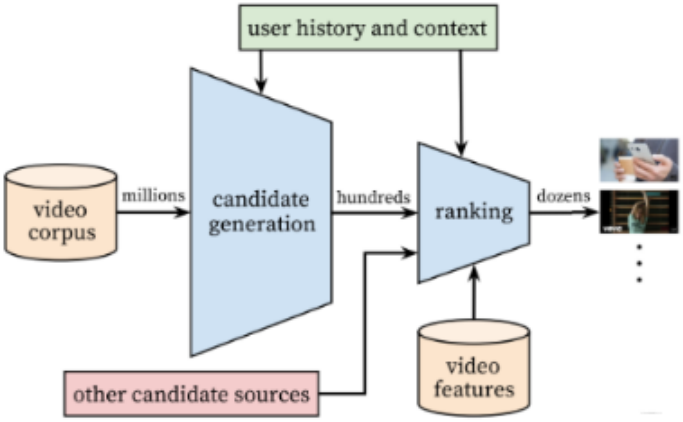
1. Scale(规模): 视频数量非常庞大，大规模数据下需要分布式学习算法以及高效的线上服务系统，文中体现这一点的是召回模型线下训练的时候，采用了负采样的思路，线上服务的时候，采用了hash映射，然后近邻检索的方式来满足实时性的需求， 这个之前我整理过faiss包和annoy包的使用， 感兴趣的可以看看。 其实，再拔高一层，我们推荐系统的整体架构呈漏斗范式，也是为了保证能从大规模情景下实时推荐。
2. Freshness(新鲜度): YouTube上的视频是一个动态的， 用户实时上传，且实时访问， 那么这时候， 最新的视频往往就容易博得用户的眼球， 用户一般都比较喜欢看比较新的视频， 而不管是不是真和用户相关(这个感觉和新闻比较类似呀)， 这时候，就需要模型有建模新上传内容以及用户最新发生的行为能力。为了让模型学习到用户对新视频有偏好， 后面策略里面加了一个"example age"作为体现。我们说的"探索与利用"中的探索，其实也是对新鲜度的把握。
3. Noise(噪声): 由于数据的稀疏和不可见的其他原因， 数据里面的噪声非常之多，这时候，就需要让这个推荐系统变得鲁棒，怎么鲁棒呢？ 这个涉及到召回和排序两块，召回上需要考虑更多实际因素，比如非对称消费特性，高活用户因素，时间因素，序列因素等，并采取了相应的措施， 而排序上做更加细致的特征工程， 尽量的刻画出用户兴趣以及视频的特征 优化训练目标，使用加权的逻辑回归等。而召回和排序模型上，都采用了深度神经网络，通过特征的相互交叉，有了更强大的建模能力， 相比于之前用的MF(矩阵分解)， 建模能力上有了很大的提升， 这些都有助于帮助减少噪声， 使得推荐结果更加准确。

所以从文章整体逻辑上看， 后面的各个细节，其实都是围绕着挑战展开的，找到当前推荐面临的问题，就得想办法解决问题，所以这篇文章的行文逻辑也是非常清晰的。

知道了挑战， 那么下面就看看YouTubeDNN的整体推荐系统架构。

YouTubeDNN推荐系统架构

整个推荐架构图如下， 这个算是比较原始的漏斗结构了:



这篇文章之所以写的好， 是给了我们一个看推荐系统的宏观视角， 这个系统主要是两大部分组成: 召回和排序。召回的目的是根据用户部分特征，从海量物品库，快速找到小部分用户潜在感兴趣的物品交给精排，重点强调快，精排主要是融入更多特征，使用复杂模型，来做个性化推荐，强调准。

而对于这两块的具体描述， 论文里面也给出了解释， 我这里简单基于我目前的理解扩展下主流方法:

1. 召回侧

> The candidate generation network takes events from the user's YouTube activity history as input and retrieves a small subset (hundreds) of videos from a large corpus. These

召回侧模型的输入一般是用户的点击历史， 因为我们认为这些历史能更好的代表用户的兴趣， 另外还有一些人口统计学特征，比如性别，年龄，地域等， 都可以作为召回侧模型的输入。 而最终模型的输出，就是与该用户相关的一个候选视频集合， 量级的话一般是几百。
召回侧， 目前根据我的理解，大致上有两大类召回方式，一类是策略规则，一类是监督模型+embedding，其中策略规则，往往和真实场景有关，比如热度，历史重定向等等，不同的场景会有不同的召回方式，这种属于"特异性"知识。
后面的模型+embedding思路是一种"普适"方法，我上面图里面梳理出了目前给用户和物品打embedding的主流方法， 这些方法大致成几个系列，比如FM系列(FM,FFM等)， 用户行为序列，基于图和知识图谱系列，经典双塔系列等，这些方法看似很多很复杂，其实本质上还是给用户或者是物品打embedding而已，只不过考虑的角度方式不同。 这里的YouTubeDNN召回模型，也是这里的一种方式而已。

2. 精排侧

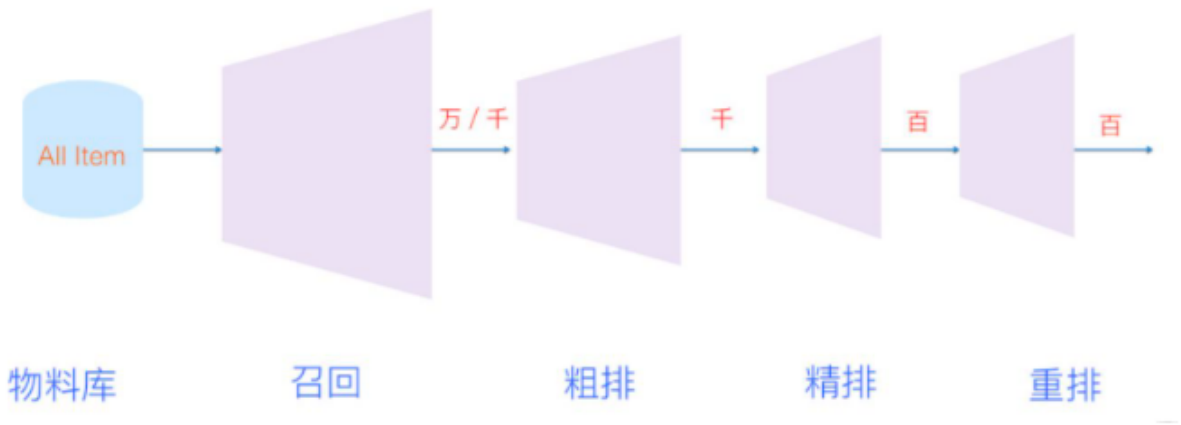
> Presenting a few "best" recommendations in a list requires a fine-level representation to distinguish relative importance among candidates with high recall. The ranking network accomplishes this task by assigning a score to each video according to a desired objective function using a rich set of features describing the video and user. The highest scoring

给视频打分
根据一个精心设计的目标函数 使用丰富的描述视频和用户的特征集合

召回那边对于每个用户， 给出了几百个比较相关的候选视频， 把几百万的规模降到了几百， 当然，召回那边利用的特征信息有限，并不能很好的刻画用户和视频特点，所以， 在精排侧，主要是想利用更多

> 的用户，视频特征，刻画特点更加准确些，从这几百个里面选出几个或者十几个推荐给用户。 而涉及到准， 主要的发力点一般有三个：特征工程， 模型设计以及训练方法。 这三个发力点文章几乎都有所涉及， 除了模式设计有点审时度势之外，特征工程以及训练方法的处理上非常漂亮，具体的后面再整理。精排侧，这一块的大致发展趋势，从ctr预估到多目标， 而模型演化上，从人工特征工程到特征工程自动化。主要是三大块， CTR预估主要分为了传统的LR，FM大家族，以及后面自动特征交叉的DNN家族，而多目标优化，目前是很多大公司的研究现状，更是未来的一大发展趋势，如何能让模型在各个目标上面的学习都能"游刃有余"是一件非常具有挑战的事情，毕竟不同的目标可能会互相冲突，互相影响，所以这里的研究热点又可以拆分成网络结构演化以及loss设计优化等， 而网络结构演化中，又可以再一次细分。 当然这每个模型或者技术几乎都有对应paper，我们依然可以通过读paper的方式，把这些关键技术学习到。

这两阶段的方法， 就能保证我们从大规模视频库中实时推荐， 又能保证个性化，吸引用户。 当然，随着时间的发展， 可能数据量非常非常大了， 此时召回结果规模精排依然无法处理，所以现在一般还会在召回和精排之间，加一个粗排进一步筛选作为过渡， 而随着场景越来越复杂， 精排产生的结果也不是直接给到用户，而是会再后面加一个重排后处理下，这篇paper里面其实也简单的提了下这种思想，在排序那块会整理到。 所以如今的漏斗， 也变得长了些。



论文里面还提到了对模型的评估方面， 线下评估的时候，主要是采用一些常用的评估指标(精确率，召回率，排序损失或者auc这种)， 但是最终看算法和模型的有效性， 是通过A/B实验， 在A/B实验中会观察用户真实行为，比如点击率， 观看时长， 留存率这种， 这些才是我们终极目标， 而有时候， A/B实验的结果和线下我们用的这些指标并不总是相关， 这也是推荐系统这个场景的复杂性。 我们往往也会用一些策略，比如修改模型的优化目标，损失函数这种， 让线下的这个目标尽量的和A/B衡量的这种指标相关性大一些。 当然，这块又是属于业务场景问题了，不在整理范畴之中。 但2016年，竟然就提出了这种方式， 所以我觉得，作为小白的我们， 想了解工业上的推荐系统， 这篇paper是不二之选。

OK， 从宏观的大视角看完了漏斗型的推荐架构，我们就详细看看YouTube视频推荐架构里面召回和排序模块的模型到底长啥样子？ 为啥要设计成这个样子？ 为了应对实际中出现的挑战，又有哪些策略？

YouTubeDNN的召回模型细节剖析

上面说过， 召回模型的目的是在大量YouTube视频中检索出数百个和用户相关的视频来。

这个问题，我们可以看成一个多分类的问题，即用户在某一个时刻点击了某个视频， 可以建模成输入一个用户向量， 从海量视频中预测出被点击的那个视频的概率。

换成比较准确的数学语言描述， 在时刻t下， 用户U在背景C下对每个视频i的观看行为建模成下面的公式：

$$ P\left(w_{t}=i \mid U, C\right)=\frac{e^{v_{i} u}}{\sum_{j \in V} e^{v_{j} u}} $$

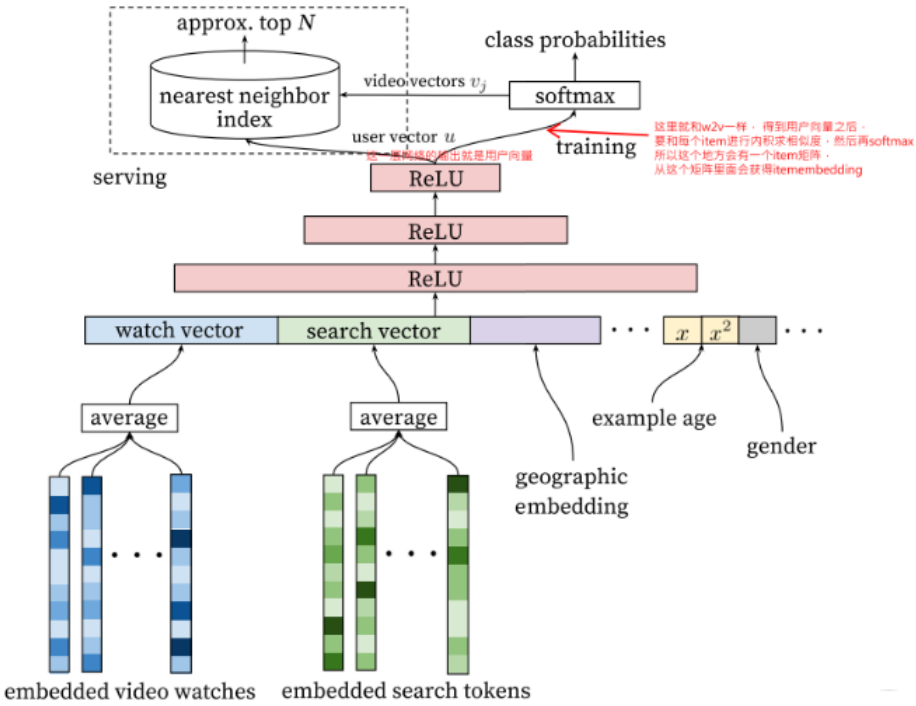
这里的u表示用户向量， 这里的v表示视频向量， 两者的维度都是N， 召回模型的任务，就是通过用户的历史点击和山下文特征， 去学习最终的用户表示向量u以及视频i的表示向量v_i， 不过这俩还有个区别是v_i本身就是模型参数， 而u是神经网络的输出(函数输出)，是输入与模型参数的计算结果。

> 解释下这个公式， 为啥要写成这个样子，其实是word2vec那边借鉴过来的，$e^{ (v_{i} u)}$表示的是当前用户向量u与当前视频v_i的相似程度，e只是放大这个相似程度而已， 不用管。 为啥这个就能表示相似程度呢？ 因为两个向量的点积运算的含义就是可以衡量两个向量的相似程度， 两个向量越相似， 点积就会越大。 所以这个应该解释明白了。 再看分母$\sum_{j \in V} e^{v_{j} u}$, 这个显然是用户向量$u$与所有视频$v$的一个相似程度求和。 那么两者一除， 依然是代表了用户u与输出的视频v_i的相似程度，只不过归一化到了0-1之间， 毕竟我们知道概率是0-1之间的， 这就是为啥这个概率是右边形式的原因。 因为右边公式表示了用户u与输出的视频v_i的相似程度， 并且这个相似程度已经归一化到了0-1之间， 我们给定u希望输出v_i的概率越大，因为这样，当前的视频v_i和当前用户u更加相关，正好对应着点击行为不是吗？

那么，这个召回模型到底长啥样子呢？

召回模型结构

召回模型的结构如下:



这个模型结构呢，相比之前的模型， 比较简单，就是一个DNN。

它的输入主要是用户侧的特征，包括用户观看的历史video序列， 用户搜索的历史tokens， 然后就是用户的人文特征，比如地理位置， 性别，年龄这些。 这些特征处理上，和之前那些模型的也比较类似，

- 用户历史序列，历史搜索tokens这种序列性的特征: 一般长这样`[item_id5, item_id2, item_id3, ...]`， 这种id特征是高维稀疏，首先会通过一个embedding层，转成低维稠密的embedding特征，即历史序列里面的每个id都会对应一个embedding向量， 这样历史序列就变成了多个embedding向量的形式， 这些向量一般会进行融合，常见的是average pooling，即每一维求平均得到一个最终向量来表示用户的历史兴趣或搜索兴趣。

> 这里值的一提的是这里的embedding向量得到的方式， 论文中作者这里说是通过word2vec方法计算的， 关于word2vec，这里就不过多解释，也就是每个item事先通过w2v方式算好了的embedding，直接作为了输入，然后进行pooling融合。
>
> 除了这种算好embedding方式之外，还可以过embedding层，跟上面的DNN一起训练，这些都是常规操作，之前整理的精排模型里面大都是用这种方式。

论文里面使用了用户最近的50次观看历史，用户最近50次搜索历史token， embedding维度是256维，采用的average pooling。 当然，这里还可以把item的类别信息也隐射到embedding， 与前面的concat起来。

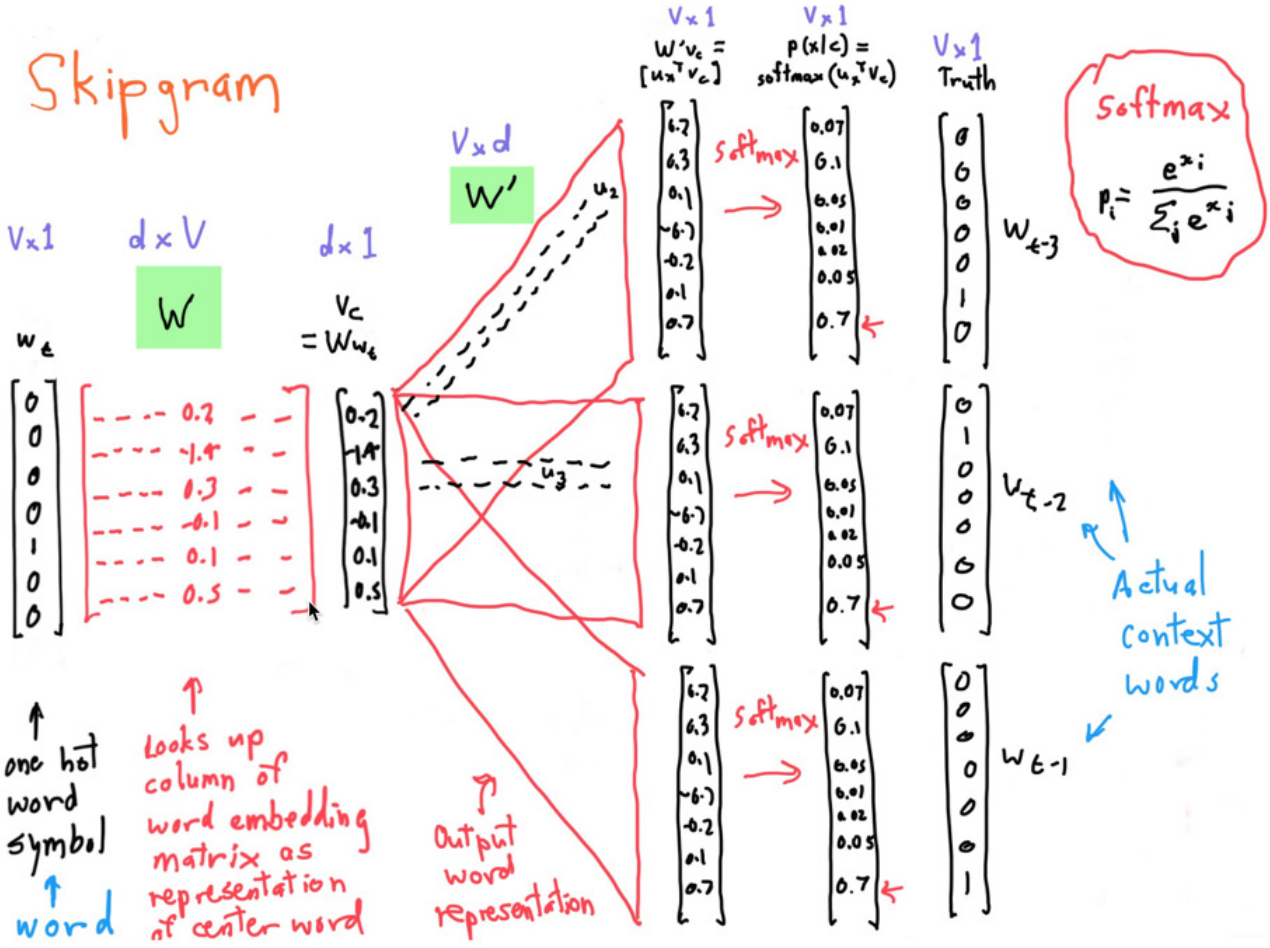
- 用户人文特征， 这种特征处理方式就是离散型的依然是labelEncoder，然后embedding转成低维稠密，而连续型特征，一般是先归一化操作，然后直接输入，当然有的也通过分桶，转成离散特征，这里不过多整理，特征工程做的事情了。 当然，这里还有一波操作值得注意，就是连续型特征除了用了x本身，还用了x^2，$logx$这种， 可以加入更多非线性，增加模型表达能力。
  这些特征对新用户的推荐会比较有帮助，常见的用户的地理位置， 设备， 性别，年龄等。

- 这里一个比较特色的特征是example age，这个特征后面需要单独整理。

这些特征处理好了之后，拼接起来，就成了一个非常长的向量，然后就是过DNN，这里用了一个三层的DNN，得到了输出， 这个输出也是向量。

Ok，到这里平淡无奇， 前向传播也大致上快说完了， 还差最后一步。 最后这一步，就是做多分类问题，然后求损失，这就是training那边做的事情。 但是在详细说这个之前， 我想先简单回忆下word2vec里面的skip-gram Model， 这个模型，如果回忆起来，这里理解起来就非常的简单了。

这里只需要看一张图即可， 这个来自cs231N公开课PPT， 我之前整理w2v的时候用到的，详细内容可看我这篇博客， 这里的思想其实也是从w2v那边过来的。

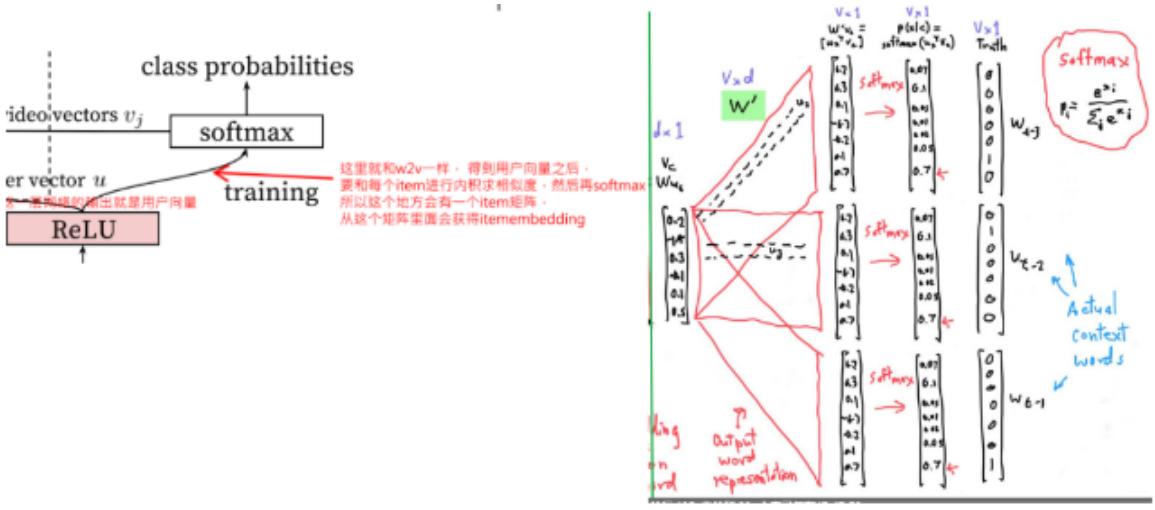


skip-gram的原理咱这里就不整理了， 这里就只看这张图，这其实就是w2v训练的一种方式，当然是最原始的。word2vec的核心思想呢？ 就是共现频率高的词相关性越大，所以skip-gram采用中心词预测上下文词的方式去训练词向量，模型的输入是中心词，做样本采用滑动窗口的形式，和这里序列其实差不多，窗口滑动一次就能得到一个序列[word1, word2, ...wordn]， 而这个序列里面呢？ 就会有中心词(比如中间那个)， 两边向量的是上下文词。 如果我们输入中心词之后，模型能预测上下文词的概率大，那说明这个模型就能解决词相关性问题了。

> 一开始， 我们的中心单词w_t就是one-hot的表示形式，也就是在词典中的位置，这里的形状是$V\times1$， V表示词库里面有V个单词， 这里的W长上面那样， 是一个$d\times V$的矩阵，d表示的是词嵌入的维度， 那么用$W*w_t$（矩阵乘法）就会得到中心词的词向量表示v_c， 大小是$d\times1$。这个就是中心词的embedding向量。 其实就是中心词过了一个embedding层得到了它的embedding向量。
> 然后就是v_c和上下文矩阵W'相乘， 这里的W'是$V\times d$的一个矩阵， 每一行代表每个单词作为上下文的时候的词向量表示， 也就是u_w， 每一列是词嵌入的维度。 这样通过$W'*v_c$就会得到一个$V\times 1$的向量，这个表示的就是中心单词w_t与每个单词的相似程度。
> 最后，我们通过softmax操作把这个相似程度转成概率， 选择概率最大的index输出。

这就是这个模型的前向传播过程。

有了这个过程， 再理解YouTubeDNN顶部就非常容易了， 我单独截出来:



只看这里的这个过程， 其实就是上面skip-gram过程， 不一样的是右边这个中心词向量v_c是直接过了一个embedding层得到的，而左边这个用户向量u是用户的各种特征先拼接成一个大的向量，然后过了一个DNN降维。 训练方式上，这两个也是一模一样的，无非就是左边的召回模型，多了几层全连接而已。

> 这样，也就很容易的理解，模型训练好了之后，用户向量和item向量到底在哪里取了吧。
>
> - 用户向量，其实就是全连接的DNN网络的输出向量，其实即使没有全连接，原始的用户各个特征拼接起来的那个长向量也能用，不过维度可能太大了，所以DNN在这里的作用一个是特征交叉，另一个还有降维的功效。
> - item向量: 这个其实和skip-gram那个一样，每个item其实是用两个embedding向量的，比如skip-gram那里就有一个作为中心词时候的embedding矩阵W和作为上下文词时候的embedding矩阵W'， 一般取的时候会取前面那个W作为每个词的词向量。 这里其实一个道理，只不过这里最前面那个item向量矩阵，是通过了w2v的方式训练好了直接作为的输入，如果不事先计算好，对应的是embedding层得到的那个矩阵。 后面的item向量矩阵，就是这里得到用户向量之后，后面进行softmax之前的这个矩阵， **YouTubeDNN最终是从这个矩阵里面拿item向量。**

这就是知识串联的魅力，其实熟悉了word2vec， 这个召回模型理解非常简单。

这其实就是这个模型训练阶段最原始的剖析，实际训练的时候，依然是采用了优化方法， 这个和word2vec也是一样，采用了负采样的方式(当然实现细节上有区别)，因为视频的数量太大，每次做多分类，最终那个概率分母上的加和就非常可怕了，所以就把多分类问题转成了多个二分类的问题。 也就是不用全部的视频，而是随机选择出了一些没点的视频， 标记为0， 点了的视频标记为1， 这样就成了二分类的问题。 关于负样本采样原理， 我之前也整理了一篇博客

> 负类基于样本分布抽取而来。负采样是针对类别数很多情况下的常用方法。当然，负样本的选择也是有讲究的，详细的看这篇文章, 我后面实验主要用了下面两种
>
> - 展示数据随机选择负例
> - 随机负例与热门打压

5. 随机负例+热门打压。随机选择，但是越是流行的Item，越大概率会被选择作为负例。例如word2vec中，某物料成为正样本的概率 $P_{pos} = (\sqrt{\frac{z(w_i)}{0.001}} + 1)\frac{0.001}{z(w_i)}$ ，其中 $z(w_i)$ 是第i个物料的曝光或点击占比，理解为降低高展item成为正样本的概率。某物料成为负样本的概率 $P_{neg} = \frac{n(w_i)^\alpha}{\sum_j n(w_j)^\alpha}$ 。其中n(w)是第i个物料的出现次数，而 α 一般取0.75，理解为增加高展item成为负样本的概率。

这样整个召回模型训练部分的"基本操作"就基本整理完了。关于细节部分，后面代码里面会描述下， 但是在训练召回模型过程中，还有一些经验性的知识也非常重要。 下面重点整理一下。

训练数据的选取和生成

模型训练的时候， 为了计算更加高效，采用了负采样的方法， 但正负样本的选取，以及训练样本的来源， 还有一些注意事项。

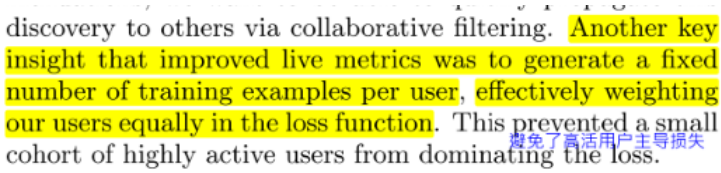
首先，训练样本来源于全部的YouTube观看记录，而不仅仅是被推荐的观看记录

> Training examples are generated from all YouTube watches (even those embedded on other sites) rather than just watches on the recommendations we produce. Otherwise, it would

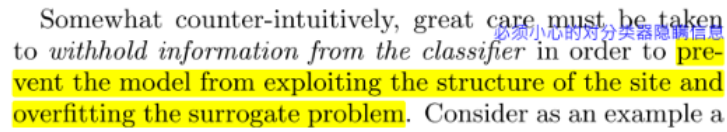
否则对于新视频会难以被曝光，会使最终推荐结果有偏；同时系统也会采集用户从其他渠道观看的视频，从而可以快速应用到协同过滤中；

其次， 是训练数据来源于用户的隐式数据， 且**用户看完了的视频作为正样本**， 注意这里是看完了， 有一定的时长限制， 而不是仅仅曝光点击，有可能有误点的。 而负样本，是从视频库里面随机选取，或者在曝光过的里面随机选取用户没看过的作为负样本。

==这里的一个经验==是**训练数据中对于每个用户选取相同的样本数， 保证用户在损失函数等权重**， 因为这样可以减少高度活跃用户对于loss的影响。可以改进线上A/B测试的效果。



这里的==另一个经验==是**避免让模型知道不该知道的信息**



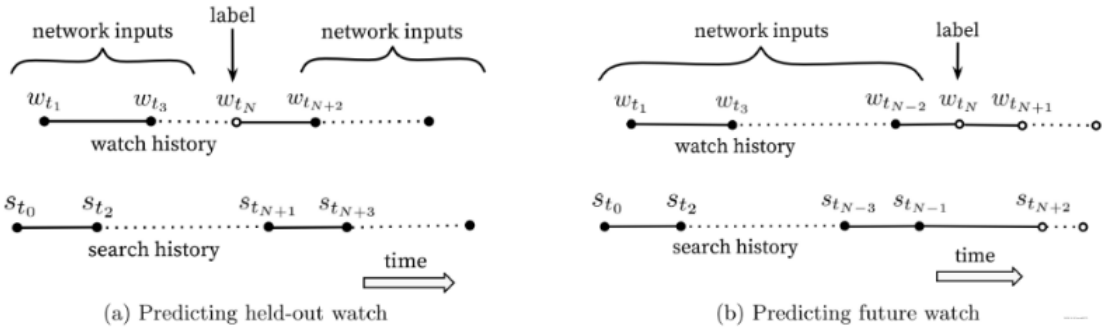
这里作者举了一个例子是如果模型知道用户最后的行为是搜索了"Taylor Swift"， 那么模型可能会倾向于推荐搜索页面搜"Taylor Swift"时搜索的视频， 这个不是推荐模型期望的行为。 解法方法是**扔掉时序信息**， 历史搜索tokens随机打乱， 使用无序的搜索tokens来表示搜索queryies(average pooling)。

> 基于这个例子就把时序信息扔掉理由挺勉强的，解决这种特殊场景的信息泄露会有更针对性的方法，比如把搜索query与搜索结果行为绑定让它们不可分。 感觉时序信息还是挺重要的， 有专门针对时序信息建模的研究。

在生成样本的时候， 如果我们的用户比较少，行为比较少， 是不足以训练一个较好的召回模型，此时一个用户的历史观看序列，可以采用滑动窗口的形式生成多个训练样本， 比如一个用户的历史观看记录是"abcdef"， 那么采用滑动窗口， 可以是abc预测d, bcd预测e, cde预测f，这样一个用户就能生成3条训练样本。 后面实验里面也是这么做的。 但这时候一定要注意一点，就是**信息泄露**，这个也是和word2vec的cbow不一样的地方。

论文中上面这种滑动制作样本的方式依据是用户的"asymmetric co-watch probabilities(非对称观看概率)"，即一般情况下，用户开始浏览范围较广， 之后浏览范围逐渐变窄。

下图中的w_{tN}表示当前样本， 原来的做法是它前后的用户行为都可以用来产生特征行为输入(word2vec的CBOW做样本的方法)。 而作者担心这一点会导致信息泄露， 模型**不该知道的信息是未来的用户行为**， 所以作者的做法是只使用更早时间的用户行为来产生特征， 这个也是目前通用的做法。 两种方法的对比如下:



(a) Predicting held-out watch (b) Predicting future watch

(a)是许多协同过滤会采取的方法，利用全局的观看信息作为输入（包括时间节点N前，N后的观看），这种方法忽略了观看序列的不对称性，而本文中采取(b)所示的方法，只把历史信息当作输入，用历史来预测未来

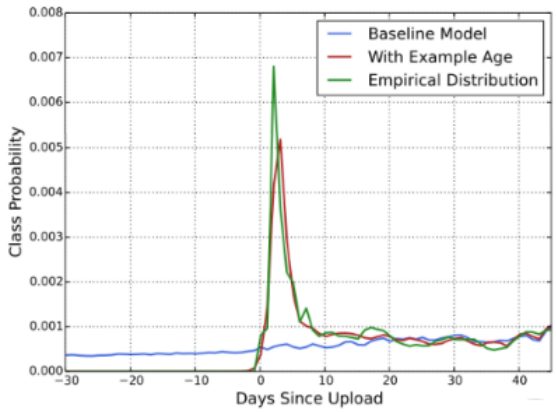
> In contrast, we "rollback" a user's history by choosing a random watch and only input actions the user took before the held-out label watch (5b).

模型的测试集， 往往也是用户最近一次观看行为， 后面的实验中，把用户最后一次点击放到了测试集里面去。这样可以防止信息穿越。

数据集的细节和tricks基本上说完， 更细的东西，就得通过代码去解释了。 接下来， 再聊聊作者加入的非常有意思的一个特征，叫做example age。

"Example Age"特征

这个特征我想单独拿出来说，是因为这个是和场景比较相关的特征，也是作者的经验传授。 我们知道，视频有明显的生命周期，例如刚上传的视频比之后更受欢迎，也就是用户往往喜欢看最新的东西，而不管它是不是和用户相关，所以视频的流行度随着时间的分布是高度非稳态变化的(下面图中的绿色曲线)



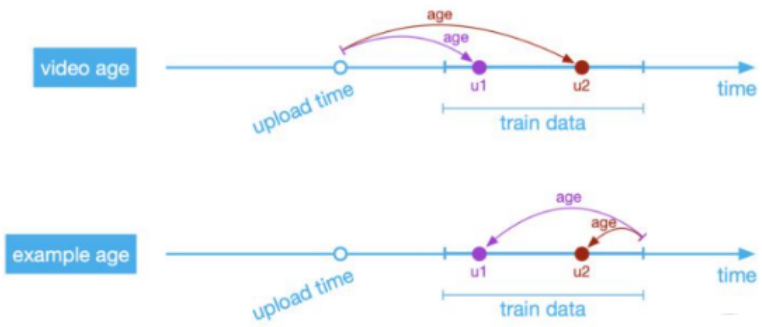
但是我们模型训练的时候，是基于历史数据训练的(历史观看记录的平均)，所以模型对播放某个视频预测值的期望会倾向于其在训练数据时间内的平均播放概率(平均热度)， 上图中蓝色线。但如上面绿色线，实际上该视频在训练数据时间窗口内热度很可能不均匀， 用户本身就喜欢新上传的内容。 所以，为了让模型学习到用户这种对新颖内容的bias， 作者引入了"example age"这个特征来捕捉视频的生命周期。

"example age"定义为$t_{max}-t$， 其中t_{max}是训练数据中所有样本的时间最大值(有的文章说是当前时间，但我总觉得还是选取的训练数据所在时间段的右端点时间比较合适，就比如我用的数据集， 最晚时间是

2021年7月的，总不能用现在的时间吧)， 而t为当前样本的时间。**线上预测时， 直接把example age全部设为0或一个小的负值，这样就不依赖于各个视频的上传时间了。**

> 其实这个操作， 现在常用的是位置上的除偏， 比如商品推荐的时候， 用户往往喜欢点击最上面位置的商品或广告， 但这个bias模型依然是不知道， 为了让模型学习到这个东西， 也可以把商品或者广告的位置信息做成一个feature， 训练的时候告诉模型。 而线上推理的那些商品， 这个feature也都用一样的。 异曲同工的意思有没有。
>
> 那么这样的操作为啥会work呢？ example age这个我理解，是有了这个特征， 就可以把某视频的热度分布信息传递给模型了， 比如某个example age时间段该视频播放较多， 而另外的时间段播放较少， 这样模型就能发现用户的这种新颖偏好， 消除热度偏见。
>
> 这个地方看了一些文章写说， 这样做有利于让模型推新热内容， 总感觉不是很通。 我这里理解是类似让模型消除位置偏见那样， 这里消除一种热度偏见。
>
> 我理解是这样，假设没有这样一个example age特征表示视频新颖信息，或者一个位置特征表示商品的位置信息，那模型训练的样本，可能是用户点击了这个item，就是正样本， 但此时有可能是用户真的喜欢这个item， 也有可能是因为一些bias， 比如用户本身喜欢新颖， 用户本身喜欢点击上面位置的item等， 但模型推理的时候，都会误认为是用户真的喜欢这个item。 所以，为了让模型了解到可能是存在后面这种bias， 我们就把item的新颖信息， item的位置信息等做成特征， 在模型训练的时候就告诉模型，用户点了这个东西可能是它比较新或者位置比较靠上面等，这样模型在训练的时候， 就了解到了这些bias，等到模型在线推理的时候呢， 我们把这些bias特征都弄成一样的，这样每个样品在模型看来，就没有了新颖信息和位置信息bias(一视同仁了)，只能靠着相关性去推理， 这样才能推到用户真正感兴趣的东西吧。
>
> 而有些文章记录的， 能够推荐更热门的视频啥的， 我很大一个疑问就是推理的时候，不是把example age用0表示吗？ 模型应该不知道这些视频哪个新不新吧。 当然，这是我自己的看法，感兴趣的可以帮我解答下呀。

`example age`这个特征到这里还没完， 原来加入这种时间bias的传统方法是使用`video age`， 即一个video上传到样本生成的这段时间跨度， 这么说可能有些懵， 看个图吧， 原来这是两个东西:



王喆老师那篇文章里面也谈到了这两种理解， 对于某个视频的不同样本，其实这两种定义是等价的，因为他们的和是一个常数。 $$ t_{\text {video age }}+t_{\text {example age }}=\text { Const } $$ 详细证明可以看参考的第三篇文章。但`example age`的定义有下面两点好处:

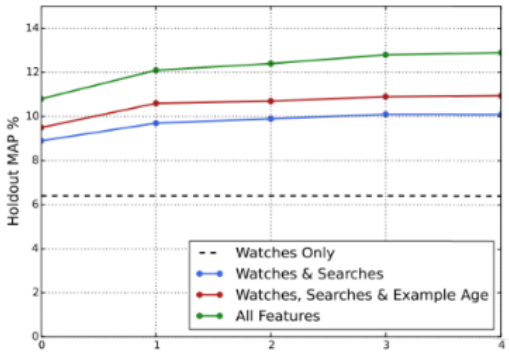
1. 线上预测时`example age`是常数值， 所有item可以设置成统一的， 但如果是`video age`的话，这个根每个视频的上传时间有关， 那这样在计算用户向量的时候，就依赖每个候选item了。 而统一的这个好处就是用户向量只需要计算一次。

2. 对不同的视频，对应的`example age`所在范围一致， 只依赖训练数据选取的时间跨度，便于归一化操作。

实验结果

这里就简单过下就好， 作者这里主要验证了下DNN的结构对推荐效果的影响，对于DNN的层级，作者尝试了0~4层， 实验结果是**层数越多越好， 但4层之后提升很有限， 层数越多训练越困难**

- Depth 1: 256 ReLU
- Depth 2: 512 ReLU → 256 ReLU
- Depth 3: 1024 ReLU → 512 ReLU → 256 ReLU
- Depth 4: 2048 ReLU → 1024 ReLU → 512 ReLU → 256 ReLU



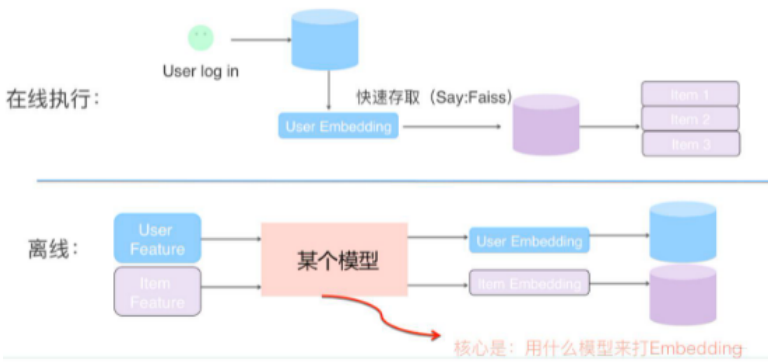
作者这里还启发了一个事情， 从"双塔"的角度再看YouTubeDNN召回模型， 这里的DNN个结构，其实就是一个用户塔， 输入用户的特征，最终通过DNN，编码出了用户的embedding向量。

而得到用户embedding向量到后面做softmax那块，不是说了会经过一个item embedding矩阵吗？ 其实这个矩阵也可以用一个item塔来实现， 和用户embedding计算的方式类似， 首先各个item通过一个物品塔(输入是item 特征， 输出是item embedding)，这样其实也能得到每个item的embedding，然后做多分类或者是二分类等。 所以**YouTubeDNN召回模型本质上还是双塔结构**， 只不过上面图里面值体现了用户塔。 我看deepmatch包里面实现的时候， 用户特征和item特征分开输入的， 感觉应该就是实现了个双塔。源码倒是没看， 等看了之后再确认。

线上服务

线上服务的时候， YouTube采用了一种最近邻搜索的方法去完成topK推荐，这其实是工程与学术trade-off的结果， model serving过程中对几百万个候选集一一跑模型显然不现实， 所以通过召回模型得到用户和video的embedding之后， 用最近邻搜索的效率会快很多。

我们甚至不用把任何model inference的过程搬上服务器，只需要把user embedding和video embedding存到redis或者内存中就好了。like this:



在线上，可以根据用户兴趣Embedding，采用类似Faiss等高效Embedding检索工具，快速找出和用户兴趣匹配的物品， 高效embedding检索工具， 我目前接触到了两个，一个是Faiss， 一个是annoy， 关于这两个工具的使用， 我也整理了两篇文章:

- annoy(快速近邻向量搜索包)学习小记
- Faiss(Facebook开源的高效相似搜索库)学习小记

之前写新闻推荐比赛的时候用过Faiss， 这次实验中使用的是annoy工具包。

另外多整理一点：

> 我们做线上召回的时候， 其实可以有两种:
>
> 1. item_2_item: 因为我们有了所有item的embedding了， 那么就可以进行物品与物品之间相似度计算，每个物品得到近似的K个， 这时候，就和协同过滤原理一样， 之间通过用户观看过的历史item，就能进行相似召回了， 工程实现上，一般会每个item建立一个相似度倒排表
> 2. user_2_item: 将item用faiss或者annoy组织成index，然后用user embedding去查相近item

基于Deepmatch包YouTubeDNN的使用方法

由于时间原因， 我这里并没有自己写代码复现YouTubeDNN模型，这个结构也比较简单， 几层的DNN，自己再写一遍剖析架构也没有啥意思， 所以就采用浅梦大佬写的deepmatch包， 直接用到了自己的数据集上做了实验。 关于Deepmatch源码， 还是看deepmatch项目， 这里主要是整理下YouTubeDNN如何用。

项目里面其实给出了如何使用YouTubeDNN，采用的是movielens数据集， 见这里

我这里就基于我做实验用的新闻推荐数据集， 把代码的主要逻辑过一遍。

数据集

实验用的数据集是新闻推荐的一个数据集，是做func-rec项目时候一个伙伴分享的，来自于某个推荐比赛，因为这个数据集是来自工业上的真实数据，所以使用起来比之前用的movielens数据集可尝试的东西多一些，并且原数据有8个多G，总共3个文件: 用户画像，文章画像， 点击日志，用户数量100多万，6000多万次点击， 文章规模是几百，数据量也比较丰富，所以后面就打算采用这个统一的数据集， 重新做实验，对比目前GitHub上的各个模型。关于数据集每个文件详细描述，后面会更新到GitHub项目。

这里只整理我目前的使用过程， 由于有8个多G的数据，我这边没法直接跑，所以对数据进行了采样， 采样方法写成了一个jupyter文件。 主要包括：

1. 分块读取数据， 无法一下子读入内存
2. 对于每块数据，基于一些筛选规则进行记录的删除，比如只用了后7天的数据， 删除了一些文章不在物料池的数据， 删除不合法的点击记录(曝光时间大于文章上传时间)， 删除没有历史点击的用户，删除观看时间低于3s的视频， 删除历史点击序列太短和太长的用户记录
3. 删除完之后重新保存一份新数据集，大约3个G，然后再从这里面随机采样了20000用户进行了后面实验

通过上面的一波操作， 我的小本子就能跑起来了，当然可能数据比较少，最终训练的YouTubeDNN效果并不是很好。详细看后面GitHub的: 点击日志数据集初步处理与采样.ipynb

简单数据预处理

这个也是写成了一个笔记本， 主要是看了下采样后的数据，序列长度分布等，由于上面做了一些规整化，这里有毛病的数据不是太多，并没有太多处理， 但是用户数据里面的年龄，性别源数据是给出了多种可能， 每个可能有概率值，我这里选出了概率最大的那个，然后简单填充了缺失。

最后把能用到的用户画像和文章画像统一拼接到了点击日志数据，又保存了一份。 作为YouTubeDNN模型的使用数据， 其他模型我也打算使用这份数据了。

详见EDA与数据预处理.ipynb

YouTubeDNN召回

这里就需要解释下一些代码了， 首先拿到采样的数据集，我们先划分下训练集和测试集:

- 测试集: 每个用户的最后一次点击记录
- 训练集: 每个用户除最后一次点击的所有点击记录

这个具体代码就不在这里写了。

```
user_click_hist_df, user_click_last_df = get_hist_and_last_click(click_df)
```

这么划分的依据，就是保证不能发生数据穿越，拿最后的测试，不能让模型看到。

接下来，就是YouTubeDNN模型的召回，从构造数据集 -> 训练模型 -> 产生召回结果，我写到了一个函数里面去。

```
def youtubednn_recall(data, topk=200, embedding_dim=8, his_seq_maxlen=50,
negsample=0,
                      batch_size=64, epochs=1, verbose=1, validation_split=0.0):
    """通过YouTubeDNN模型，计算用户向量和文章向量
    param: data: 用户日志数据
    topk: 对于每个用户，召回多少篇文章
    """
    user_id_raw = data[['user_id']].drop_duplicates('user_id')
    doc_id_raw = data[['article_id']].drop_duplicates('article_id')

    # 类别数据编码
    base_features = ['user_id', 'article_id', 'city', 'age', 'gender']
    feature_max_idx = {}
    for f in base_features:
        lbe = LabelEncoder()
        data[f] = lbe.fit_transform(data[f])
        feature_max_idx[f] = data[f].max() + 1

    # 构建用户id词典和doc的id词典，方便从用户idx找到原始的id
    user_id_enc = data[['user_id']].drop_duplicates('user_id')
    doc_id_enc = data[['article_id']].drop_duplicates('article_id')
    user_idx_2_rawid = dict(zip(user_id_enc['user_id'], user_id_raw['user_id']))
    doc_idx_2_rawid = dict(zip(doc_id_enc['article_id'],
doc_id_raw['article_id']))
```

```
    # 保存下每篇文章的被点击数量， 方便后面高热文章的打压
    doc_clicked_count_df = data.groupby('article_id')['click'].apply(lambda x:
x.count()).reset_index()
    doc_clicked_count_dict = dict(zip(doc_clicked_count_df['article_id'],
doc_clicked_count_df['click']))

    train_set, test_set = gen_data_set(data, doc_clicked_count_dict, negsample,
control_users=True)

    # 构造youtubeDNN模型的输入
    train_model_input, train_label = gen_model_input(train_set, his_seq_maxlen)
    test_model_input, test_label = gen_model_input(test_set, his_seq_maxlen)

    # 构建模型并完成训练
    model = train_youtube_model(train_model_input, train_label, embedding_dim,
feature_max_idx, his_seq_maxlen, batch_size, epochs, verbose, validation_split)

    # 获得用户embedding和doc的embedding， 并进行保存
    user_embs, doc_embs = get_embeddings(model, test_model_input,
user_idx_2_rawid, doc_idx_2_rawid)

    # 对每个用户，拿到召回结果并返回回来
    user_recall_doc_dict = get_youtube_recall_res(user_embs, doc_embs,
user_idx_2_rawid, doc_idx_2_rawid, topk)

    return user_recall_doc_dict
```

这里面说一下主要逻辑，主要是下面几步:

1. 用户id和文章id我们要先建立索引-原始id的字典，因为我们模型里面是要把id转成embedding，模型的表示形式会是{索引: embedding}的形式， 如果我们想得到原始id，必须先建立起映射来
2. 把类别特征进行label Encoder， 模型输入需要， embedding层需要，这是构建词典常规操作， 这里要记录下每个特征特征值的个数，建词典索引的时候用到，得知道词典大小
3. 保存了下每篇文章被点击数量， 方便后面对高热文章实施打压
4. 构建数据集

   ```
   rain_set, test_set = gen_data_set(data, doc_clicked_count_dict, negsample,
   control_users=True)
   ```

   这个需要解释下， 虽然我们上面有了一个训练集，但是这个东西是不能直接作为模型输入的， 第一个原因是正样本太少，样本数量不足，我们得需要滑动窗口，每个用户再滑动构造一些，第二个是不满足deepmatch实现的模型输入格式，所以gen_data_set这个函数，是用deepmatch YouTubeDNN的第一个范式，基本上得按照这个来，只不过我加了一些策略上的尝试:

   ```
   def gen_data_set(click_data, doc_clicked_count_dict, negsample,
   control_users=False):
   ```

```
    """构造youtubeDNN的数据集"""
    # 按照曝光时间排序
    click_data.sort_values("expo_time", inplace=True)
    item_ids = click_data['article_id'].unique()

    train_set, test_set = [], []
    for user_id, hist_click in tqdm(click_data.groupby('user_id')):
        # 这里按照expo_date分开，每一天用滑动窗口滑，可能相关性更高些,另外，这样序列不会太长，因为eda发现有点击1111个的
        #for expo_date, hist_click in hist_date_click.groupby('expo_date'):
        # 用户当天的点击历史id
        pos_list = hist_click['article_id'].tolist()
        user_control_flag = True

        if control_users:
            user_samples_cou = 0

        # 过长的序列截断
        if len(pos_list) > 50:
            pos_list = pos_list[-50:]

        if negsample > 0:
            neg_list = gen_neg_sample_candiate(pos_list, item_ids, doc_clicked_count_dict, negsample, methods='multinomial')

        # 只有1个的也截断 去掉，当然我之前做了处理，这里没有这种情况了
        if len(pos_list) < 2:
            continue
        else:
            # 序列至少是2
            for i in range(1, len(pos_list)):
                hist = pos_list[:i]
                # 这里采用打压热门item策略，降低高展item成为正样本的概率
                freq_i = doc_clicked_count_dict[pos_list[i]] / (np.sum(list(doc_clicked_count_dict.values())))
                p_posi = (np.sqrt(freq_i/0.001)+1)*(0.001/freq_i)

                # p_posi=0.3  表示该item_i成为正样本的概率是0.3，
                if user_control_flag and i != len(pos_list) - 1:
                    if random.random() > (1-p_posi):
                        row = [user_id, hist[::-1], pos_list[i], hist_click.iloc[0]['city'], hist_click.iloc[0]['age'], hist_click.iloc[0]['gender'], hist_click.iloc[i]['example_age'], 1, len(hist[::-1])]
                        train_set.append(row)

                        for negi in range(negsample):
                            row = [user_id, hist[::-1], neg_list[i*negsample+negi], hist_click.iloc[0]['city'], hist_click.iloc[0]['age'], hist_click.iloc[0]['gender'], hist_click.iloc[i]['example_age'], 0, len(hist[::-1])]
                            train_set.append(row)

                        if control_users:
                            user_samples_cou += 1
```

```
                                # 每个用户序列最长是50， 即每个用户正样本个数最多是50
个，如果每个用户训练样本数量到了30个，训练集不能加这个用户了
                                if user_samples_cou > 30:
                                    user_samples_cou = False

                            # 整个序列加入到test_set， 注意，这里一定每个用户只有一个最长序
列，相当于测试集数目等于用户个数
                            elif i == len(pos_list) - 1:
                                row = [user_id, hist[::-1], pos_list[i],
hist_click.iloc[0]['city'], hist_click.iloc[0]['age'], hist_click.iloc[0]
['gender'], 0, 0, len(hist[::-1])]
                                test_set.append(row)


    random.shuffle(train_set)
    random.shuffle(test_set)

    return train_set, test_set
```

关键代码逻辑是首先点击数据按照时间戳排序，然后按照用户分组，对于每个用户的历史点击， 采用滑动窗口的形式，边滑动边构造样本， 第一个注意的地方，是每滑动一次生成一条正样本的时候， 要加入一定比例的负样本进去， 第二个注意最后一整条序列要放到test_set里面。

我这里面加入的一些策略，负样本候选集生成我单独写成一个函数，因为尝试了随机采样和打压热门item采样两种方式， 可以通过methods参数选择。 另外一个就是正样本里面也按照热门实现了打压， 减少高热item成为正样本概率，增加高热item成为负样本概率。 还加了一个控制用户样本数量的参数，去保证每个用户生成一样多的样本数量，打压下高活用户。

5. 构造模型输入 这个也是调包的定式操作，必须按照这个写法来:

```
def gen_model_input(train_set, his_seq_max_len):
    """构造模型的输入"""
    # row: [user_id, hist_list, cur_doc_id, city, age, gender, label,
hist_len]
    train_uid = np.array([row[0] for row in train_set])
    train_hist_seq = [row[1] for row in train_set]
    train_iid = np.array([row[2] for row in train_set])
    train_u_city = np.array([row[3] for row in train_set])
    train_u_age = np.array([row[4] for row in train_set])
    train_u_gender = np.array([row[5] for row in train_set])
    train_u_example_age = np.array([row[6] for row in train_set])
    train_label = np.array([row[7] for row in train_set])
    train_hist_len = np.array([row[8] for row in train_set])

    train_seq_pad = pad_sequences(train_hist_seq, maxlen=his_seq_max_len,
padding='post', truncating='post', value=0)
    train_model_input = {
        "user_id": train_uid,
        "click_doc_id": train_iid,
        "hist_doc_ids": train_seq_pad,
        "hist_len": train_hist_len,
```

```
        "u_city": train_u_city,
        "u_age": train_u_age,
        "u_gender": train_u_gender,
        "u_example_age":train_u_example_age
    }
    return train_model_input, train_label
```

上面构造数据集的时候，是把每个特征加入到了二维数组里面去， 这里得告诉模型，每一个维度是啥特征数据。如果相加特征，首先构造数据集的时候，得把数据加入到数组中， 然后在这个函数里面再指定新加入的特征是啥。 下面的那个词典， 是为了把数据输入和模型的Input层给对应起来，通过字典键进行标识。

6. 训练YouTubeDNN 这一块也是定式， 在建模型事情，要把特征封装起来，告诉模型哪些是离散特征，哪些是连续特征， 模型要为这些特征建立不同的Input层，处理方式是不一样的

```
def train_youtube_model(train_model_input, train_label, embedding_dim,
feature_max_idx, his_seq_maxlen, batch_size, epochs, verbose,
validation_split):
    """构建youtubednn并完成训练"""
    # 特征封装
    user_feature_columns = [
        SparseFeat('user_id', feature_max_idx['user_id'], embedding_dim),
        VarLenSparseFeat(SparseFeat('hist_doc_ids',
feature_max_idx['article_id'], embedding_dim,

embedding_name="click_doc_id"), his_seq_maxlen, 'mean', 'hist_len'),

        SparseFeat('u_city', feature_max_idx['city'], embedding_dim),
        SparseFeat('u_age', feature_max_idx['age'], embedding_dim),
        SparseFeat('u_gender', feature_max_idx['gender'], embedding_dim),
        DenseFeat('u_example_age', 1,)
    ]
    doc_feature_columns = [
        SparseFeat('click_doc_id', feature_max_idx['article_id'],
embedding_dim)
        # 这里后面也可以把文章的类别画像特征加入
    ]

    # 定义模型
    model = YoutubeDNN(user_feature_columns, doc_feature_columns,
num_sampled=5, user_dnn_hidden_units=(64, embedding_dim))

    # 模型编译
    model.compile(optimizer="adam", loss=sampledsoftmaxloss)

    # 模型训练，这里可以定义验证集的比例，如果设置为0的话就是全量数据直接进行训练
    history = model.fit(train_model_input, train_label,
batch_size=batch_size, epochs=epochs, verbose=verbose,
validation_split=validation_split)

    return model
```

然后就是建模型，编译训练即可。这块就非常简单了，当然模型方面有些参数，可以了解下，另外一个注意点，就是这里用户特征和item特征进行了分开， 这其实和双塔模式很像， 用户特征最后编码成用户向量， item特征最后编码成item向量。

7. 获得用户向量和item向量 模型训练完之后，就能从模型里面拿用户向量和item向量， 我这里单独写了一个函数:

```
 获取用户embedding和文章embedding
def get_embeddings(model, test_model_input, user_idx_2_rawid,
doc_idx_2_rawid, save_path='embedding/'):
    doc_model_input =
{'click_doc_id':np.array(list(doc_idx_2_rawid.keys()))}

    user_embedding_model = Model(inputs=model.user_input,
outputs=model.user_embedding)
    doc_embedding_model = Model(inputs=model.item_input,
outputs=model.item_embedding)

    # 保存当前的item_embedding 和 user_embedding 排序的时候可能能够用到，但是需
要注意保存的时候需要和原始的id对应
    user_embs = user_embedding_model.predict(test_model_input, batch_size=2
** 12)
    doc_embs = doc_embedding_model.predict(doc_model_input, batch_size=2 **
12)
    # embedding保存之前归一化一下
    user_embs = user_embs / np.linalg.norm(user_embs, axis=1, keepdims=True)
    doc_embs = doc_embs / np.linalg.norm(doc_embs, axis=1, keepdims=True)

    # 将Embedding转换成字典的形式方便查询
    raw_user_id_emb_dict = {user_idx_2_rawid[k]: \
                                v for k, v in zip(user_idx_2_rawid.keys(),
user_embs)}
    raw_doc_id_emb_dict = {doc_idx_2_rawid[k]: \
                                v for k, v in zip(doc_idx_2_rawid.keys(),
doc_embs)}
    # 将Embedding保存到本地
    pickle.dump(raw_user_id_emb_dict, open(save_path +
'user_youtube_emb.pkl', 'wb'))
    pickle.dump(raw_doc_id_emb_dict, open(save_path + 'doc_youtube_emb.pkl',
'wb'))

    # 读取
    #user_embs_dict = pickle.load(open('embedding/user_youtube_emb.pkl',
'rb'))
    #doc_embs_dict = pickle.load(open('embedding/doc_youtube_emb.pkl',
'rb'))
    return user_embs, doc_embs
```

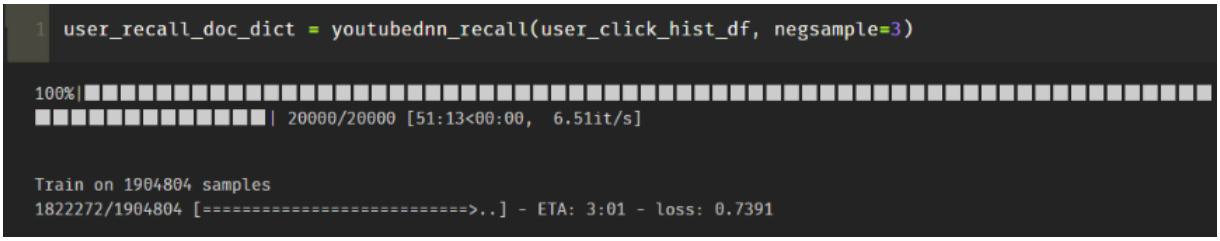
获取embedding的这两行代码是固定操作， 下面做了一些归一化操作，以及把索引转成了原始id的形式。

8. 向量最近邻检索，为每个用户召回相似item

```
def get_youtube_recall_res(user_embs, doc_embs, user_idx_2_rawid,
doc_idx_2_rawid, topk):
    """近邻检索，这里用annoy tree"""
    # 把doc_embs构建成索引树
    f = user_embs.shape[1]
    t = AnnoyIndex(f, 'angular')
    for i, v in enumerate(doc_embs):
        t.add_item(i, v)
    t.build(10)
    # 可以保存该索引树 t.save('annoy.ann')

    # 每个用户向量， 返回最近的TopK个item
    user_recall_items_dict = collections.defaultdict(dict)
    for i, u in enumerate(user_embs):
        recall_doc_scores = t.get_nns_by_vector(u, topk,
include_distances=True)
        # recall_doc_scores是(([doc_idx], [scores]))， 这里需要转成原始doc的id
        raw_doc_scores = list(recall_doc_scores)
        raw_doc_scores[0] = [doc_idx_2_rawid[i] for i in raw_doc_scores[0]]
        # 转换成实际用户id
        try:
            user_recall_items_dict[user_idx_2_rawid[i]] =
dict(zip(*raw_doc_scores))
        except:
            continue

    # 默认是分数从小到大排的序， 这里要从大到小
    user_recall_items_dict = {k: sorted(v.items(), key=lambda x: x[1],
reverse=True) for k, v in user_recall_items_dict.items()}

    # 保存一份
    pickle.dump(user_recall_items_dict, open('youtube_u2i_dict.pkl', 'wb'))

    return user_recall_items_dict
```

用了用户embedding和item向量，就可以通过这个函数进行检索， 这块主要是annoy包做近邻检索的固定格式， 检索完毕，为用户生成最相似的200个候选item。

以上，就是使用YouTubeDNN做召回的整个流程。 效果如下:



这个字典长这样:

```
user_recall_doc_dict

用户id    item id    相似分数
{17340: [(464361541, 0.4042508602142334),
  (465125301, 0.40399086475372314),
  (465015032, 0.40353378653526306),
  (466486538, 0.40345317125320435),
  (466781719, 0.4032151997089386),
  (466653789, 0.402744859457016),
  (466510622, 0.40270328521728516),
  (465158006, 0.4024966061115265),
  (466360139, 0.4021632671356201),
```

接下来就是评估模型的效果，这里我采用了简单的HR@N计算的， 具体代码看GitHub吧， 结果如下:

```
metrics_recall(user_recall_doc_dict, user_click_last_df, topk=200)

topk:  50  :  hit_num:  20 hit_rate:  0.001 user_num :  20000
topk:  100  :  hit_num:  48 hit_rate:  0.0024 user_num :  20000
topk:  150  :  hit_num:  88 hit_rate:  0.0044 user_num :  20000
topk:  200  :  hit_num:  107 hit_rate:  0.00535 user_num :  20000
```

结果不怎么样啊，唉， 难道是数据量太少了？ 总归是跑起来且能用了。

详细代码见尾部GitHub链接吧， 硬件设施到位的可以尝试多用一些数据试试看哈哈。

YouTubeDNN新闻推荐数据集的实验记录

这块就比较简单了，简单的整理下我用上面代码做个的实验，尝试了论文里面的几个点，记录下:

1. 负采样方式上，尝试了随机负采样和打压高热item两种方式， 从我的实验结果上来看， 带打压的效果略好一点点

1. 打压高热item， w2v的那种负采样方式

```
metrics_recall(user_recall_doc_dict, user_click_last_df, topk=200)

topk:  50  :  hit_num:  20 hit_rate:  0.001 user_num :  20000
topk:  100  :  hit_num:  48 hit_rate:  0.0024 user_num :  20000
topk:  150  :  hit_num:  88 hit_rate:  0.0044 user_num :  20000
topk:  200  :  hit_num:  107 hit_rate:  0.00535 user_num :  20000
```

1. 所有观看的视频随机负采样

```
metrics_recall(user_recall_doc_dict, user_click_last_df, topk=200)

topk:  50  :  hit_num:  25 hit_rate:  0.00125 user_num :  20000
topk:  100  :  hit_num:  45 hit_rate:  0.00225 user_num :  20000
topk:  150  :  hit_num:  69 hit_rate:  0.00345 user_num :  20000
topk:  200  :  hit_num:  104 hit_rate:  0.0052 user_num :  20000
```

2. 特征上， 尝试原论文给出的example age的方式，做一个样本的年龄特征出来 这个年龄样本，我是用的训练集的最大时间减去曝光的时间，然后转成小时间隔算的，而测试集里面的统一用0表示， 但效果好差。 看好多文章说这个时间单位是个坑，不知道是小时，分钟，另外这个特征我只做了简单归一化，感觉应该需要做归一化

特征上， 尝试原论文给出的example age的方式，做一个样本的年龄特征出来

```
topk:  50  :  hit_num:  22 hit_rate:  0.0011 user_num :  20000
topk:  100  :  hit_num:  43 hit_rate:  0.00215 user_num :  20000
topk:  150  :  hit_num:  55 hit_rate:  0.00275 user_num :  20000
topk:  200  :  hit_num:  71 hit_rate:  0.00355 user_num :  20000
```

这个效果并不是很好， 可能是因为单位没有把握住？ 小时，还是分钟还是啥?

3. 尝试了控制用户数量，即每个用户的样本数量保持一样，效果比上面略差

```
metrics_recall(user_recall_doc_dict, user_click_last_df, topk=200)

topk:  50  :  hit_num:  22 hit_rate:  0.0011 user_num :  20000
topk:  100  :  hit_num:  43 hit_rate:  0.00215 user_num :  20000
topk:  150  :  hit_num:  53 hit_rate:  0.00265 user_num :  20000
topk:  200  :  hit_num:  68 hit_rate:  0.0034 user_num :  20000
```

4. 开始模型评估，我尝试用最后一天的，而不是最后一次点击的， 感觉效果不如最后一次点击作为测试集效果好

当然，上面实验并没有太大说服力，第一个是我采样的数据量太少，模型本身训练的不怎么样，第二个这些策略相差的并不是很大， 可能有偶然性。

并且我这边做一次实验，要花费好长时间，探索就先到这里吧， example age那个确实是个迷， 其他的感觉起来， 打压高活效果要比不打压要好。

另外要记录下学习小tricks:

> 跑一次这样的实验，我这边一般会花费两个小时左右的时间， 而这个时间在做实验之前，一定要做规划才能好好的利用起来， 比如，我计划明天上午要开始尝试各种策略做实验， 今天晚上的todo里面，就要记录好， 我会尝试哪些策略，记录一个表， 调整策略，跑模型的时候，我这段空档要干什么事情，todo里面都要记录好，比如我这段空档就是解读这篇paper，写完这篇博客，基本上是所有实验做完，我这篇博客也差不多写完，正好，哈哈
>
> 这个空档利用，一定要提前在todo里面写好，而不是跑模型的时候再想，这个时候往往啥也干不下去，并且还会时不时的看模型跑，或者盯着进度条发呆，那这段时间就有些浪费了呀，即使这段时间不学习，看个久违的电视剧， 久违的书，或者keep下不香吗哈哈， 但得提前规划。
>
> 可能每个人习惯不一样，对于我，是这样哈，所以记录下 😉

总结

由于这篇文章里面的工程经验太多啦，我前面介绍的时候，可能涉及到知识的一些扩展补充，把经验整理的比较凌乱，这里再统一整理下， 这些也都是工业界常用的一些经验了:

召回部分:

1. 训练数据的样本来源应该是全部物料， 而不仅仅是被推荐的物料，否则对于新物料难以曝光
2. 训练数据中对于每个用户选取相同的样本数， 保证用户在损失函数等权重， 这个虽然不一定非得这么做，但考虑打压高活用户或者是高活item的影响还是必须的
3. 序列无序化: 用户的最近一次搜索与搜索之后的播放行为有很强关联，为了避免信息泄露，将搜索行为顺序打乱。
4. 训练数据构造: 预测接下来播放而不是用传统cbow中的两侧预测中间的考虑是可以防止信息泄露，并且可以学习到用户的非对称视频消费模式
5. 召回模型中，类似word2vec，video 有input embedding和output embedding两组embedding，并不是共享的， input embedding论文里面是用w2v事先训练好的， 其实也可以用embedding层联合训练
6. 召回模型的用户embedding来自网络输出， 而video的embedding往往用后面output处的
7. 使用 `example age` 特征处理 time bias，这样线上检索时可以预先计算好用户向量

参考资料:

- 重读Youtube深度学习推荐系统论文

- YouTube深度学习推荐系统的十大工程问题
- 你真的读懂了Youtube DNN推荐论文吗
- 推荐系统经典论文(二)】YouTube DNN
- 张俊林-推荐技术发展趋势与召回模型
- 揭开YouTube深度推荐系统模型Serving之谜
- Deep Neural Networks for YouTube Recommendations YouTubeDNN推荐召回与排序