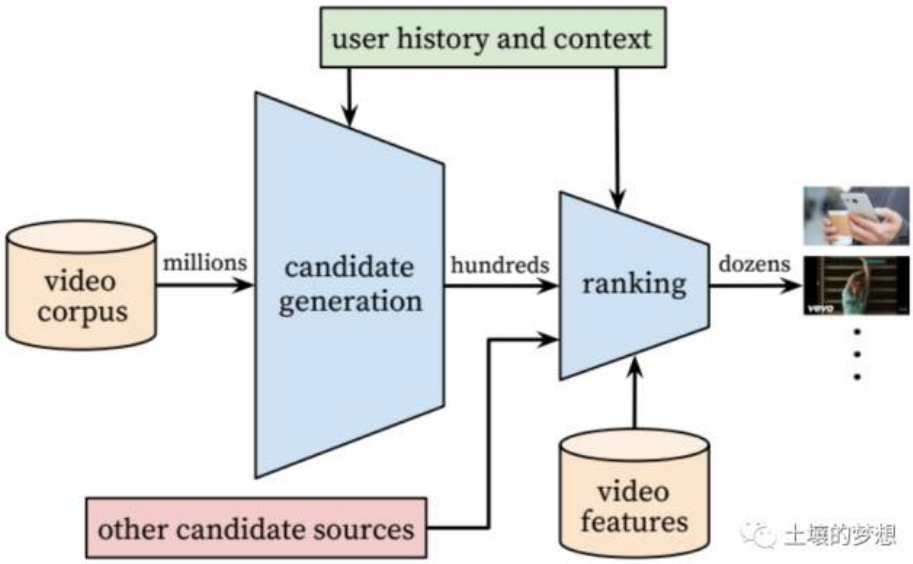
带你玩转word2vec（一）

庄子齐 土壤的梦想 2017-09-29

Hi, there.

按上次说的，今天分享一下自然语言处理领域的一些经验和看法。以下便是我个人的一些看法和观点，望多指正。

主题是word2vec的来源，因为我们现在采用的算法架构为“**海选Candidate Generation -> 排序Ranking**”，来完成从上百万（因为规模原因，目前数据量不太大）物品item库里面选择最可能适合某个用户的top N个物品的个性化推荐，这里在Candidate Generation部分由很多引擎共同完成“召回”这部分任务，目标是较准确且又不失多样性的找到先选出一部分候选项送到排序部分。如图：



这里，我们会用到一个在NLP（自然语言处理）领域内的词向量模型word2vec，因为大家发现word2vec是一个包含语义的**近义词模型**，而对候选集的召回过程可以采用事物的相似性（与近义词类比发现场景很像）来完成，因此自然而然的word2vec就开始成为了一个海选引擎。

既然word2vec是来自自然语言处理领域的词向量模型，那首先还是需要先从NLP中“词的向量化”任务开始聊聊。

说说人类的语言，比如中文句子由词为粒度来组成，它不比声音频率和图片像素那种原始的特征，词是一个高度抽象的高级特征，一个词背后代表的意义其实非常多。

词之所以要向量化是因为需要把一个抽象的东西给数字化，这样才能处理，向量化的过程就是想一个办法给每个词在一个高维空间中选择一个位置（坐标点），这个空间的维度即我们用来表示词向量的维度一样大。

接下来会按照提纲来走一路：

一．词向量化的演进路线

二．NNLM神经网络语言模型的诞生

三．Word2vec做了哪些针对性优化

四．Word2vec模型怎么增量学习

五．带你使用Word2vec扩展到其他任务

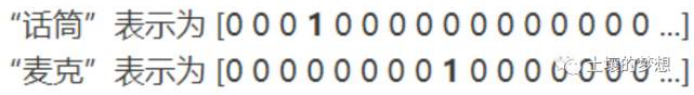
六．Word2vec做召回引擎

一．词向量的进化

One-hot 向量表示法

这是一种简单的办法来表示每个词乃至每个特征，如果我们的词典里有1000个词汇，那么给每个词汇一个index，这个就是词的向量one-hot表示：

比如“话筒”和“麦克”分别在词典中的位置是4和9，那么one-hot向量化则为下面这种方式来完成。



如果把所得到的词向量全部画在一个空间坐标系中间去，那么这个空间的维度将会大到爆炸（假设词典里面有20万个词汇，那我们的向量已经大得内存存不下了），并且想象一下也能发现，这种表示方式非常浪费空间一个词要霸占一个坐标轴，而且空间中词汇的位置之间没有语义关系。

TF-IDF 向量表示法

这是之前在搜索领域常用的一种检索召回排序的方式，简单提一下，tf表示词频，idf表示文档逆向频率，比如一个词在一篇文章中出现的次数越多，而且这个词出现在其他文章的次数越少，则表示这个词对这篇文章来说是一个很重要的特征。最后这个词的tf-idf=f(tf, idf)可以用一个函数来表示，对于不同的工具包这个f(tf,idf)还不一样，但大同小异，有点还加上了其他的一些指标。那么一句话就相当于一篇短文章，这句话同样是由很多词组成了，只不过这里我们将one-hot的词序列乘上了每个词对于这篇短文章的tf-idf值，从而可以将一句话映射成带权重的one-hot表示：

我=[0,0,0,1,0,.....] ->[0, 0, 0, 0.36, 0,.....]

是=[0,1,0,0,0,.....] ->[0, 0.001, 0, 0, 0,.....]

中国人=[0,0,0,0,..0,1,0...] ->[0, 0, 0, 0,.. 0, 0.05, 0...]

之所以产生这种演变的一个直接原因就是基于仿生学的神经网络方向的人工智能科学发展迅速，并有超过其他AI分支方向的趋势，而神经网络的理论是基于高维向量空间的映射来的，跟传统算法不同的是，我们的系统依旧是输入到输出，只是系统函数并非用传统可书写的方程去表示，而是用空间之间的映射关系表示，也就是网络结构参数即系统函数。

为什么从隐式马尔科夫到循环神经网络是一个进步？从概率统计模型到函数表示的网络结构，要求的都是将抽象特征第一步映射到高维空间中去，所以特征的向量化一直在演进。

二．NNLM-神经网络语言模型

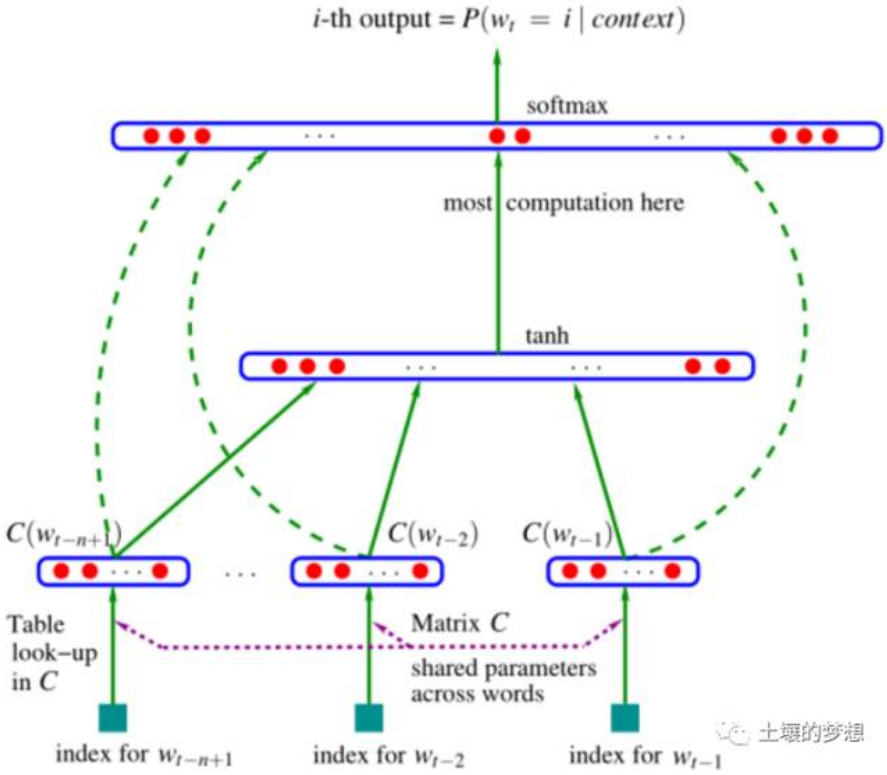
即便用TF-IDF的方式可以对词进行带权重的向量化，但是我们还是没有摆脱以下问题：

-浪费空间，内存消耗大

-向量本身在空间中的位置没有任何含义

因此我们希望词向量本身的每个维度的值或者说其空间坐标位置能带有一些实际的物理意义。

在进入word2vec之前，我们先来看看它的前身，NNLM。



下面来解释一下上面这个NNLM（neural network language model）在做什么：

首先这个模型是一个极端多分类模型（extreme multi-classfication），任务就是“我们需要在给定一些词之后准确的预测出下一个词是什么”。比如“今天天气不错，我们可以出去看（电影）”“我爱（你）”。

1. 看到输入的index for Wt了吗？首先我们会把所有需要被考虑的词汇放在一个大词典dictionary中，然后像one-hot那样给每个词一个索引index，即标记它在词典的第几个位置，词典里面假设包含N个词的话那这里每个词的one-hot向量都是长度为N的矩阵。而这里的Wt即表示当前的词，至于Wt-k则表示当前词之前的k个词。

2. 关键的地方便是这里的Matrix C矩阵，这个是一个非常庞大的矩阵N*D的矩阵，D是我们希望把这个one-hot向量压缩到多大的维度，其实在做的是一个降维操作，该操作业界也有一个称呼叫word embedding，得到的新词向量从sparse vector变成了dense vector。Matrix C矩阵即完成一个one-hot向量经过查询变成一个dense vector的过程。

3. 那么后续的一个隐层和最后的softmax多分类层是在干嘛呢，当然是去试图构造一个更加复杂的deep learning网络来加强特征提取的能力了。这样，我们把每个词的one-hot向量作为输入，查询到Matrix C中自己对于的稠密向量后，在第一隐层首尾拼接成一个长长的一维数组(完成了一个向量化后的句子的一次空间映射)，在softmax层我们又进行了一次空间映射，这里得到的向量空间可就很大了，这里的输出层之所以叫极端多分类层，是因为它实际上的节点数（长度）与我们的词典大小N是等大的，可想最后把短句窗口（以窗口为单位在句子上右移）预测出来的下一个词的向量表示成一个这么高维度空间中的一个坐标点。

4.通过后向传播回传误差实现了上述模型的学习收敛过程，我们最终可以得到一个“前文来预测下一个词”的语言模型，这个模型可以判断一句话是不是人说的话，但是可以看见它的计算量是非常庞大的。首先从倒数第二层到输出层可以数数大概有多少万个全连接W，以及每个节点使用softmax函数做归一化的时候要进行多少次e^x指数运算。

5.在我们进行完长时间的计算之后，我们可以得到一个不那么精确但有一定预测功能的语言模型（来判断一句话是不是人话的模型），而**其衍生物就是之前提到的第一网络层Matrix C，这样我们将C导出来就得到了我们需要的词向量**，如果有人问，为什么是Matrix C，自己再想想看，是不是？

不过这里的词向量矩阵包含N行长度为D的一维数组来表示N个词在空间维度为D的空间中的位置，而采用**NNLM我们解决了之前提到的两个one-hot和TF-IDF向量没法解决的问题：**

-**维度爆炸问题**：通过word embedding压缩降维到了一个稠密向量，从几万维变成了一两百维。

-**位置物理意义**：现在它的坐标对应的位置是有物理意义的，即有语义关系，因为它们通过学习使得向量之前产生了联系，因此坐标间有语义和近义词关系。如“我们喜欢猫”和“我们喜欢狗”中，猫和狗两个词在学习结束后必定在位置上是比较相近的。

三．Word2vec做了哪些针对性优化呢？

既然我们已经有了NNLM为什么还需要word2vec？因为有两个比较重要的遗留问题：

-在输出层我们不是有提到过**极端多分类问题**吗？参数规模是非常庞大的，这样要得到收敛的模型需要很长时间。

-在**第一隐层**不是有提到过将查询到的稠密向量首尾相接拼成一个非常长的一维数组吗？这个地方也是非常高维的一个空间。

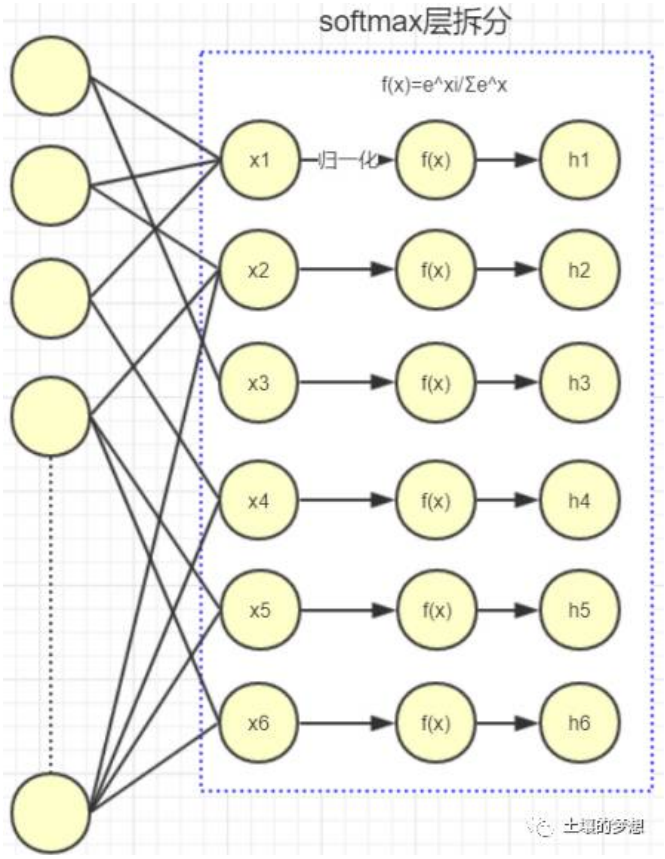
因此word2vec算法是针对自己想要的目标对NNLM做了修改：

-针对第一个问题，它把**softmax做了一个简化以求更快的训练速度**，两种方式：负采样的softmax和层次化树结构的softmax。

-针对第二个问题，它把**输入层得到的稠密向量直接累加在了一起**，以求更简单的网络结构。

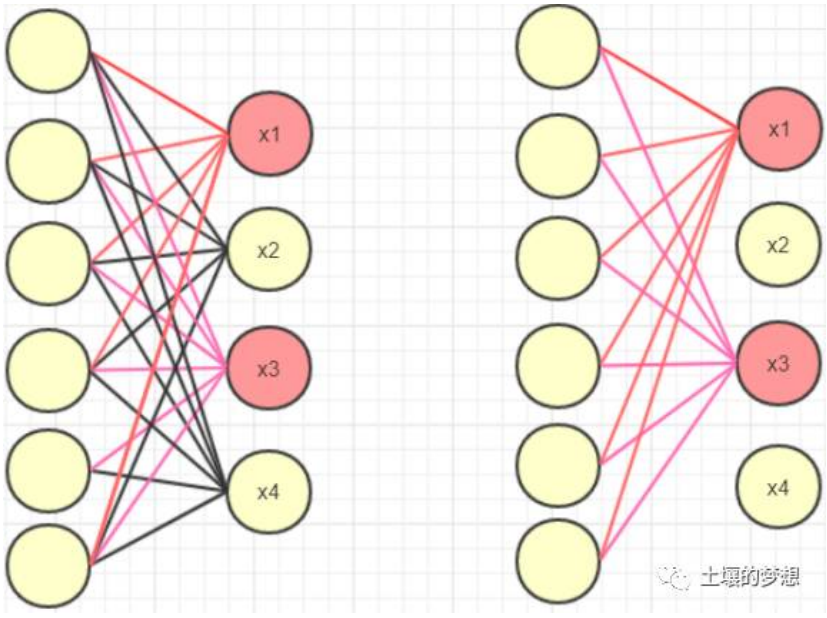
关于负采样的softmax的理解：

这个概念在网上很难找到一个令人满意的解释，所以我来解释一下。Softmax层作为最后一个多分类输出层在做的如蓝色框内部分，将上一层加权得到的节点值用f(xi)=exp(xi)/Σexp(x)得到归一化后的hi作，而h1+h2+...hn=1，也就是说它将接收到的特征值用softmax函数归一化为每个类别的一个概率，而每个类别的概率和为1。



这里可以看到，如果输出的类别非常多的话，计算量和节点之间的连接数规模都会非常大，那就会严重影响运算效率。

而negative sampling的方式是一种折衷的加速方式，我们只抽取了红色的节点进行误差回传，如下图所示：



因为我们**训练模型的目的是得到一个符合映射关系的目标函数**，在这里使用误差后向传播来学习的时候，我们原本是回传所有节点x1, x2, x3, x4对应的误差（误差由损失函数来定义，这里不做展开），在回传过程中每个连接节点的边W（即图中的线）都是带有权重的，它们的权重会被逐步修正（增减）来缩小误差来逼近最后的目标函数。**为了加快收敛的速度，我们可以按一个抽样的法则抽取其中的某些节点来进行误差回传（而不是回传所有节点误差），这种技术即negative sampling 在softmax的应用**。我们这样做可以省略掉很多计算量，假设我们有一百万个分类作为输出，那我们采用negative sampling可以只抽取中间1个标准答案对应类的那个节点+999个其他类对应的节点作为错误的答案，这样计算规模直接

缩小了1000倍。但是，代价也是有的，那就是会损失精确度，因为我们每次回传的误差中并没有包含所有错误答案对应的误差，而只是抽取了其中的一部分。

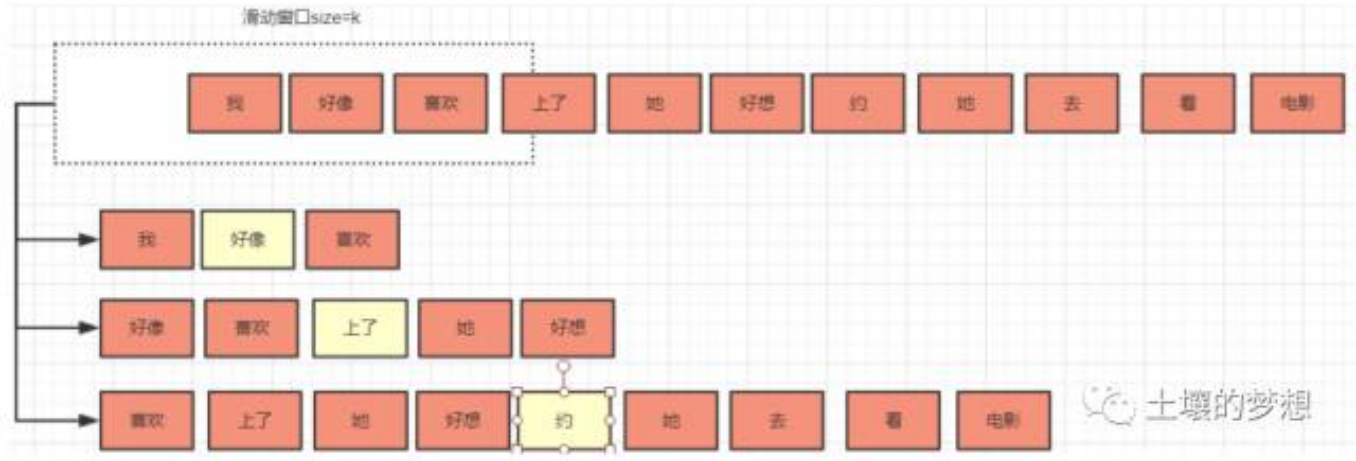
这是一个取**最优解的通用哲学问题**，以后碰到类似的情况可以直接以此类推来预判：在算法或规则最优秀和无误的情况下，我们永远只能在效率和效果中去寻找一个折衷的较优解，去符合我们当前的需要，大部分时候并没有一个客观的最优解。（比如经常会需要在准确率和召回率中取一个折衷）

层次化softmax的理解：

大家应该知道二分搜索和二叉树对于查询来说是非常快速的，这里就是利用了这个思想。接下来就以层次化softmax为标准来进行原理的简析。

模型的原理简析：

首先在建立一个模型之前需要搞清楚输入和目标。在这里，输入是一串词序列表示的上下文“[我]，[好像]，[x]，[上了]，[你]”，输出是预测出未知的中心词[x]是什么。当然，我们人一看就知道是什么，x=[喜欢]，[爱]之类的都是可能的，因为我们已经学了几十年语文了，这任务是小case，那么对于word2vec而言，它如何完成的？



如图所示，一句人类的自然语言被滑窗提取出了好多组“上下文+中心词”的样本，只要设置好窗口采样规则，它会自己把自己组成很多对输入和输出，它是一个非监督学习的任务，因为你并不需要对语料做任务预先的人工分类标签标记。

我不要贴代码和公式，那样没有必要，就大致用语言来描述一下流程：

1>文本预处理

我们需要根据语料库先建立的是一本词典，因为我们的语料库（用来训练的文字样本）中有一些废词、干扰项，并且为了控制规模，我们需要将十分罕见的干扰词给去掉。我们还需要去计数，统计每个词的词频（出现次数），至于如何分词，我推荐使用ansj分词器来完成。

2>huffman树的建立

这里有两个知识点，一个是通信领域内的**无前缀编码**（这个编码方式是源于信息在物理层都是01比特流，而在接收端解码的过程中我们为了避免解码歧义，需要用到无前缀编码技术来保证比特流解释的唯一性），二是计算机领域的**霍夫曼树**（一棵最优二叉树）。

这一步是紧接着词频统计结果来的，huffman树在这里的目标是建立效率最高的编码表去表示语料库，即我们希望频率高的词，比如“我们”用01这种短编码来表示，而“符拉迪沃斯托克”这种低频词用

010101101这种长编码来表示，这样我们的语料进行编码后得到的长度就会缩短，结合无前缀编码后又不丢失信息（对这一块感兴趣还可以百度一下熵的概念）。

为什么我们需要在word2vec里面用到这种树呢？

顾名思义，我们这里对softmax采取的优化加速方式是层次化，即我们在输出的时候其实把定位softmax输出层N个节点变成了定位寻找huffman树的叶节点的过程，每个叶节点里面存的便是原来输出层的词汇，但是采用二叉树搜索的方式搜索次数降为了logN，加速就在这里完成，而霍夫曼01编码的意义就是表示这个词的路径，0为左子树，1为右子树。霍夫曼编码树又进一步缩短了高频词的路径（因为它会被频繁定位到），而延长了低频词的路径，从而进一步加速。

3>词汇的初始化和数字化

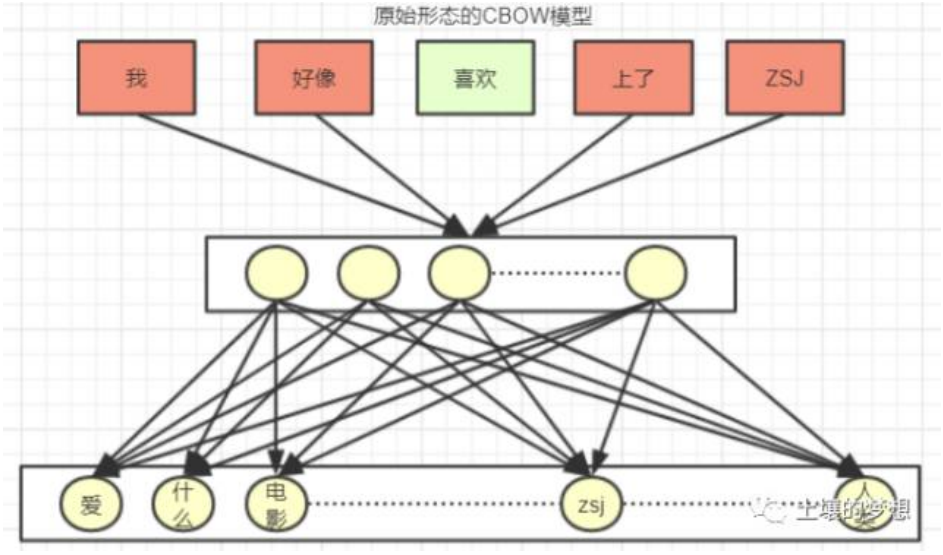
我们的目标是把一个抽象的词汇“中国”表示成一个稠密向量[0.01, -0.16, 0.34,.....]，第一我们需要知道这个向量的长度，即该向量所处高维空间的维度D。我们需要给每个词在空间中找一个初始的位置，目前的做法就是给每个词一个随机的位置（随机初始化）。**这里会碰到一个问题就是：该把词向量定为多少维度？**这个又回到了之前的那个哲学问题“**efficency-performance balance**”（效率-效果权衡），高维度意味着更复杂、消耗更多空间和更多计算时间（服务时实时计算），但低维度又会带来区分度低的问题，因为最后我们得到的word2vec是一个近义词词典，用余弦相似度来表示词之间语义的相似性，取值为[-1,1]。如果维度越低，它们重叠的可能性越大，意思是语义不相近的词它们的差异度不会特别明显，如果维度太高我们实时计算的代价又会增高，所以建议根据语料库词典大小来，词汇量越大给越高的维度。

4>构造word2vec神经网络

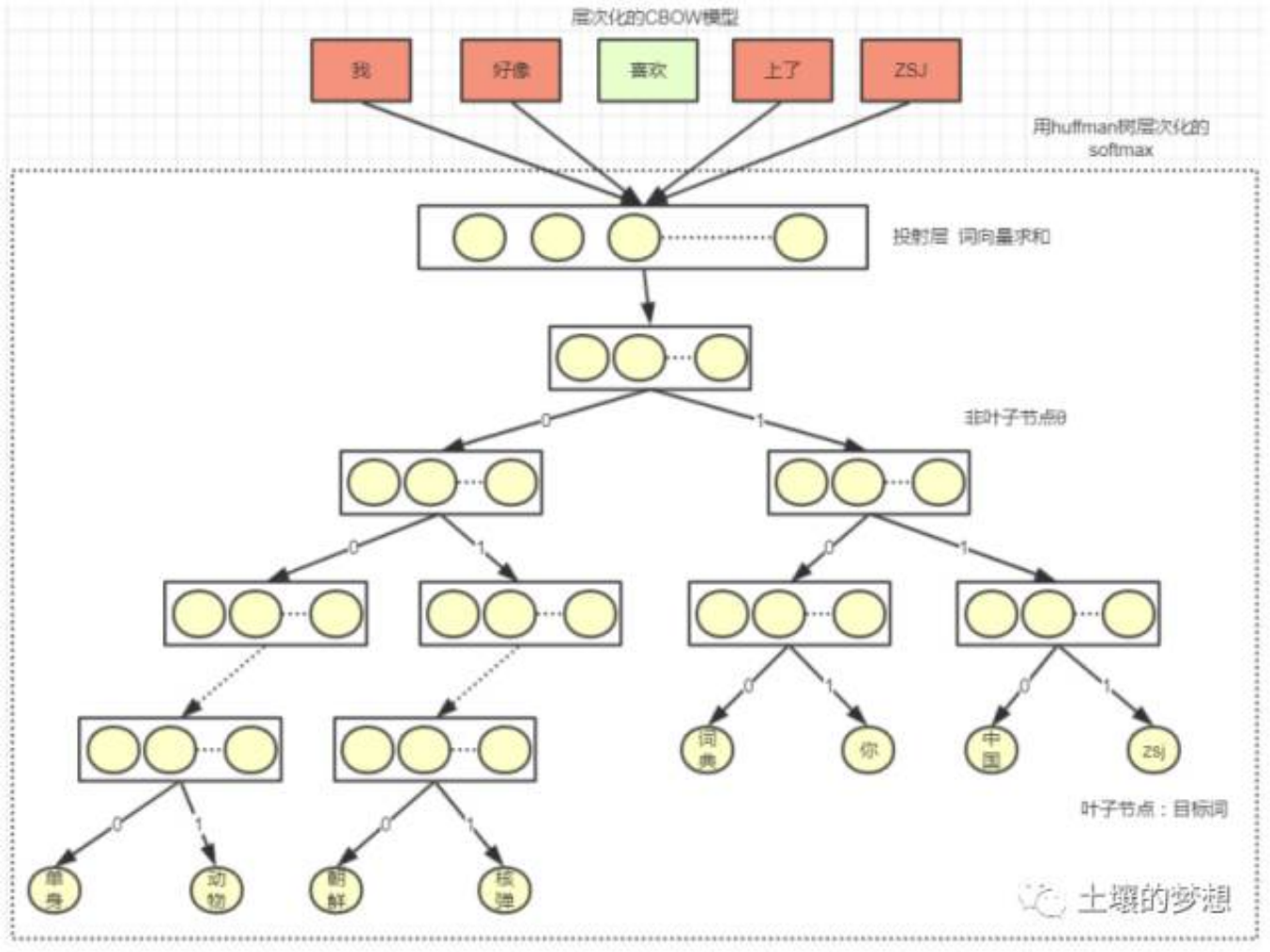
有人说这是一个**浅层的神经网络**，因为它实际上是由一个输入层（模型中叫投射层）通过一层全连接就到了softmax输出层，而hierarchical或者negative sampling改造的softmax层是为了加速，但并没有加深网络的层次，所以在我看来word2vec归根到底是一个无隐层的神经网络。

这个地方有两种，一种是叫CBOW（连续词袋）模型，另一种叫skip-gram模型，它们的区别还在于：

CBOW中我们是根据上下文（前和后的k个，而不再像NNLM里面采用前k个）来推测中心词，而skip-gram我们是根据中心词来推测它的上下文，上图，网图实在太丑了，我只好重新画了一个。



而下面这个是改进后的CBOW-层次softmax：



看上面这个例子：“我好像喜欢上了zsj”，分词结果为“[我]，[好像],[喜欢]，[上了],[zsj]”，这里zsj是一个人名，这样[喜欢]成为了中心词也就是预测目标，而其他词则成成为了它的上下文。

只有叶节点里面才保存了目标词，比如这里的“zsj”对应的路径编码是111，“你”对应的路径编码是101，每一个非叶子节点里面保存的都是一组与与词向量同长度的参数组 θ 向量对应的一个逻辑回归单元，非叶子节点逻辑回归单元的任务是“择路”——选择下一步向左还是向右，我们知道逻辑回归的输出p范围是0~1，我们认为输出p表示的是向左的概率，则向右的概率是1-p。

5>模型学习词向量

到上面为止，我们已经把初始网络结构搭建好了，但是词向量都是随机给的位置，现在我们需要让word2vec模型跑起来，并且学习到词典中每次词的稠密词向量dense word vector，并使得这些向量能比较完美的符合我们这个预测模型：

1.继续上面所说的，拿滑动窗口扫描一个句子获取到一个“上下文+中心词”的训练样本

2.输入的上下文中所有的词获取到自己当前状态的词向量，并做单纯的累加得到与词向量等长的投射层向量X

3.现在我们希望这个向量X进入到huffman树中间去循迹，从而找到它的中心词，只要每一个非叶子节点方向都选对了，那就可以正确找到中心词

4.开始循迹，如何保证每个抉择节点都选对呢？之前不是提到每个非叶抉择节点中保存了与向量X等长的 θ 参数吗？我们把X与路径上遇到的所有节点的 θ 参数做内积，再进行逻辑回归函数运算得到一个0~1之间的概率，我们规定它表示向左的概率，这样就可以知道向量X在这个抉择节点的是选择更倾向于向左还是向右。

同时注意到，在我们构建huffman树的时候，每个叶节点中就保存了中心词，以及它的01编码，这个01编码又能回溯从根出发到它的正确路径，所以，我们完全不用害怕在训练节点循迹过程中哪一步走错了会影响下一步，因为我们已经知道了该样本的“中心词”，所以我们直接拿X与该中心词对应的所有父节点中的 θ 去逻辑回归就好了。

目标函数是什么？

如果这一组“上下文”能够正确的找到“中心词”，那么每一非叶节点选择路径的逻辑回归运算都要对才行，于是目标函数就成了该路径对应的节点上逻辑回归选对的联合概率要最大化，对于整体语料来说，那就是所有的样本都要满足循迹正确的联合概率要最大。

5.开始学习，如何更新模型得到最终需要的词向量？

这个地方如果感兴趣，可以去github上下载一份word2vec的源代码。

这里的训练方式是先将 θ 作为未知数，将X作为已知数去更新 θ ，然后再反过来把X当作未知数把 θ 当作已知数去更新X，巧妙的使用的这种交替迭代更新的方式，而X是上下文的那些词汇的求和向量，因此使用梯度下降法，我们对所有有贡献的上下文词，统一进行一次梯度下降更新。

6.学习类算法本身并没有终点，我们认为当前效果还不错了就让它停止，而word2vec的默认学习模式就是把语料中所有的句子过一遍，越往后走学习率逐步递减（小的学习率迭代多了照样带来大的改变）。当结束之后，我们可以获取到词典内所有词的向量表示，用hashmap保存起来即可。

关于word2vec的注意事项：

以下都是我总结的一些点，有些在网络暂时找不到答案。

1.关于随机性：

如果有接触过的人细心肯定会发现，对于同一份语料，采用同样的参数，单单只是重复做一次词向量模型的学习，但是所得到的所有词向量与上一次大相近庭，为什么？

关于这个问题我也上知乎和一些机构网站问过，但是没有得到有效回复，那还是我来解释一下吧。由于神经网络的随机性，这个随机性体现在词向量的随机初始化、滑动窗口的随机大小，高频词的随机丢弃、如果采用negative sampling softmax的结构还会有随机的负采样，我想很多初学者肯定和我当时一样，以为词向量具备不变性，其实你每次训练的一组词向量都是唯一的一组空间分布，不可能重复（如果设置相同的随机种子倒是可以）。

也许有人会问，如果做到动态扩展呢？如果我很多其他地方是依赖之前的一组词向量的值，岂不是每次为了添加新词我所有的依赖都需要重新跟着一起更新吗？这样更新成本势必会增大哇。这里其实是可以做到的，包括很多框架也提供了增量学习的接口，如果我们要自己去理解和实现的话也并非难事，本人实践后发现是可行的，思路如下：固定之前已有的词向量参与前向计算但不参与值的更新（即固定不变），对包含新词的语料进行多次迭代并更新新词，如此便可得到增量学习的效果。这个思路在迁移学习中用得非常多——迁移学习关注的是我们如何最大程度的去复用之前花大代价（长时间、大样本）训练的模型（不限于神经网络），并应用到一个类似的新任务，比如我们拿训练好的动物图像识别模型去迁移到商品识别上。

2.关于一词多义的困惑：

看看这个词“苹果”，我们在“水果繁多种类，其中有苹果、香蕉...”中它表示一种水果，在“有大量果粉云集在苹果专卖店前排队等待新机到货...”中它表示一种手机，然而在word2vec中它并不会区分，从而

会造成局部的混乱，也就是苹果的近义词可能不会如我们所想的那样。

3.局部上下文的限制

上下文是什么？是只一个事物发生的先验环境，这里是一个词出现时周围的词，但是由于算法复杂性的限制，word2vec只会考虑到局部窗口内、至多一个句子内的上下文，无法直接考虑到句子之间的上下文，而句子与句子之间的语义联系，所以在利用其近义词查找能力的同时，要认识到其召回的作用范围。

4.关于**word2vec**的本质和适用范围：

新手总是觉得很多东西就是万能的，会盲目的去进行尝试，大忌。你必须了解其机制才行，就好像玩一些策略游戏一样，取胜的关键是去迎合机制而非蛮干。这里需要强调一下的是，word2vec以及后续衍生的一些算法，是近义词模型，近义词模型是代表什么？代表不精确，代表相似。所以如果有精度要求高的请绕道，而在推荐系统里面，这也是为什么它用来做召回引擎的原因。

5.word2vec依然是弱语义词袋模型

Word2vec依然是词袋模型，从字面就能看到“CBOW-continuous bag of words”，翻译过来就是连续词袋模型。它的语义关系属于弱语义关系，这还归功于其局部滑动窗口，把常在一起的词给包装在了一个样本里，这样才形成了弱语义，但为什么弱？因为你把滑动窗口里的上下文随机交换位置打乱语序，它的结果不会改变（因为上下文被累加在一起的时候就已经消除了先后顺序的信息）。

有人认为向量同向的为近义词，那反向的就为反义词了？如果明白机制，就不会浪费时间去尝试了。答案是当然不会是反义词，因为模型学习目标里面只有确定中心词是什么，目标函数并没有包含反义词这个概念。

关于开通大纲提到的后面三点，由于时间关系，留在下次在讨论：

-Word2vec模型怎么增量学习

-带你使用Word2vec扩展到其他任务

-Word2vec做召回引擎

今天就到这里，国庆快乐，呀哈呼咿哈哈哈~!

to be continued...