3. Word2Vec

1．词嵌入

将无法计算的非结构化信息转化为可计算的结构化信息

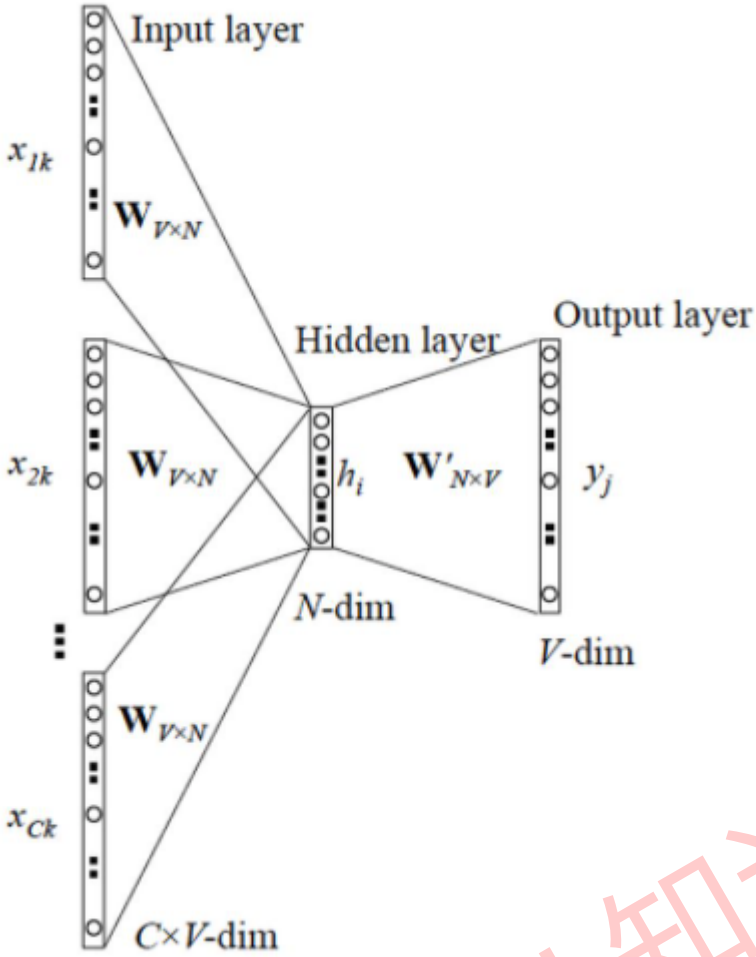
2．分布式假设

所谓分布式假设，用一句话可以表达：**相同上下文语境的词有似含义**。而由此引申出了word2vec、fastText，在此类词向量中，虽然其本质仍然是语言模型，但是它的目标并不是语言模型本身，而是词向量，其所作的一系列优化，都是为了更快更好的得到词向量。

3．word2vec的模型结构

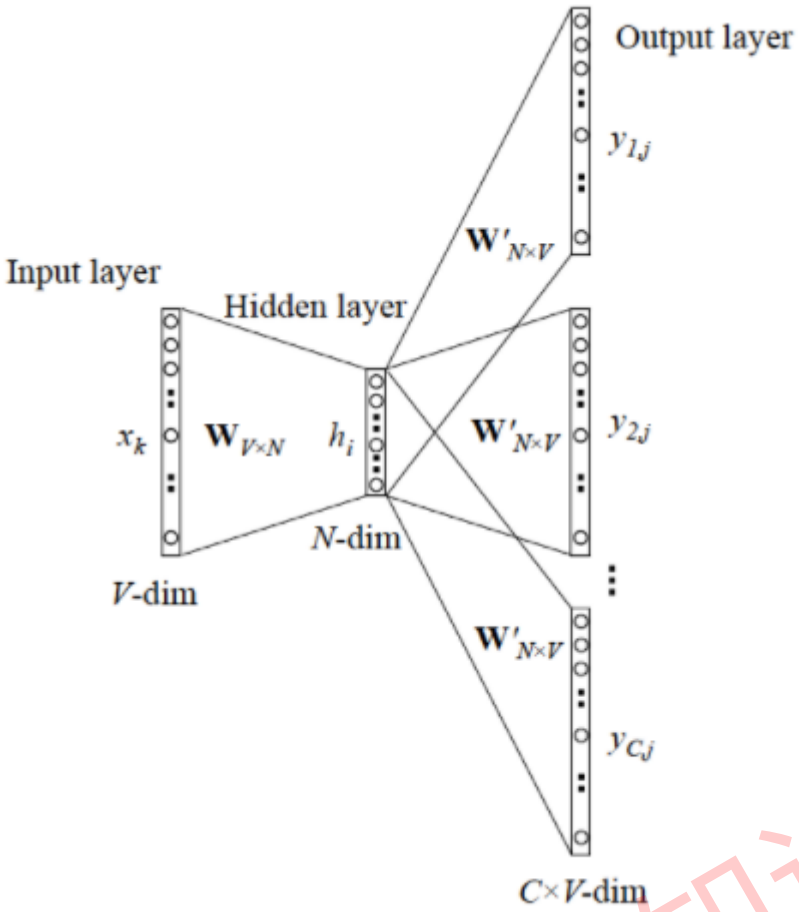
CBOW：

拿一个词语的上下文作为输入，来预测这个词语本身，则是『CBOW 模型』。在cbow方法中，是用周围词预测中心词，从而利用中心词的预测结果情况，使用GradientDesent方法，不断的去调整周围词的向量。当训练完成之后，每个词都会作为中心词，把周围词的词向量进行了调整，这样也就获得了整个文本里面所有词的词向量。要注意的是， cbow的对周围词的调整是统一的：求出的gradient的值会同样的作用到每个周围词的词向量当中去。



skip-gram：

用一个词语作为输入，来预测它周围的上下文，那这个模型叫做『Skip-gram 模型』。skip-gram是用中心词来预测周围的词。在skip-gram中，会利用周围的词的预测结果情况，使用GradientDecent来不断的调整中心词的词向量，最终所有的文本遍历完毕之后，也就得到了文本所有词的词向量。



4．Word2vec的训练任务

词向量一般不是模型训练的任务，而是为完成任务顺带得到的附属品。CBOW主要完成的任务是根据上下文预测中间词，而 Skip-gram 的任务是根据中间词预测上下文。

5．Word2vec中的CBOW模型

- 输入层：上下文单词的onehot. {假设单词向量空间dim为V，上下文单词个数为C}
- 所有onehot分别乘以共享的输入权重矩阵W. {V*N矩阵，N为自己设定的数，初始化权重矩阵W}*
- *所得的向量 {因为是onehot所以为向量} 相加求平均作为隐层向量, size为 1*N.
- 乘以输出权重矩阵W' {N*V*}
- *得到向量 {1*V} 激活函数处理得到V-dim概率分布 {PS: 因为是onehot嘛，其中的每一维都代表着一个单词}
- 概率最大的index所指示的单词为预测出的中间词（target word）与true label的onehot做比较，误差越小越好（根据误差更新权重矩阵）

所以，需要定义loss function（一般为交叉熵代价函数），采用梯度下降算法更新W和W'。训练完毕后，输入层的每个单词与矩阵W相乘得到的向量的就是我们想要的词向量（word embedding），这个矩阵（所有单词的word embedding）也叫做look up table（其实聪明的你已经看出来了，其实这个look up table就是矩阵W自身），也就是说，任何一个单词的onehot乘以这个矩阵都将得到自己的词向量。有了look up table就可以免去训练过程直接查表得到单词的词向量了。即任何一个单词的one-hot表示乘以这个矩阵都将得到自己的word embedding。

6．CBOW模型的目标函数

CBOW中的目标函数是使条件概率 $P(w|context(w))$ 最大化，其等价于：

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{C}} \log \prod_{j=2}^{l^w} \{[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{1-d_j^w} \cdot [1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{d_j^w}\} \\ &= \sum_{w\in\mathcal{C}} \sum_{j=2}^{l^w} \{(1-d_j^w)\cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]\}\end{aligned}$$

7．Word2vec中的Skip-gram模型

- 首先我们选句子中间的一个词作为我们的输入词，例如我们选取“dog”作为input word
- 有了input word以后，我们再定义一个叫做skip_window的参数，它代表着我们从当前input word的一侧（左边或右边）选取词的数量。如果我们设置skip_window=2，那么我们最终获得**窗口中的词（包括input word在内）**就是['The', 'dog'，'barked', 'at']。skip_window=2代表着选取左input word左侧2个词和右侧2个词进入我们的窗口，所以整个窗口大小span=2x2=4。
- 另一个参数叫num_skips，它代表着我们从整个窗口中选取多少个不同的词作为我们的output word，当skip_window=2，num_skips=2时，我们将会得到两组 (input word, output word) 形式的训练数据，即 ('dog', 'barked')，('dog', 'the')。
- 模型的输出概率代表着到我们词典中每个词有多大可能性跟input word同时出现

8．skip-gram模型的目标函数

Skip-gram中的目标函数是使条件概率 $P(context(w)|w)$ 最大化，其等价于：

$$\mathcal{L} = \sum_{w \in \mathcal{C}} \log \prod_{u \in Context(w)} \prod_{j=2}^{l^u} \{[\sigma(\mathbf{v}(w)^\top \theta^u_{j-1})]^{1-d^u_j} \cdot [1-\sigma(\mathbf{v}(w)^\top \theta^u_{j-1})]^{d^u_j}\}$$

$$= \sum_{w \in \mathcal{C}} \sum_{u \in Context(w)} \sum_{j=2}^{l^u} \{(1-d^u_j) \cdot \log[\sigma(\mathbf{v}(w)^\top \theta^u_{j-1})] + d^u_j \cdot \log[1 - \sigma(\mathbf{v}(w)^\top \theta^u_{j-1})]\}.$$

9．CBOW与skip-gram的输入分别是什么

- CBOW是multi-hot
- skip-gram是one-hot

10．隐层到输出层的权重矩阵作用

后面那个隐层到输出层的权重矩阵其实就是全连接层的权重矩阵，将词向量映射成词汇表维度一样的形式。这样才能输入到softmax中。然后通过梯度下降最小化交叉熵损失以拟合样本，训练模型。但是这个全连接层没有激活函数。

11．为什么使用softmax

模型使用交叉熵作为损失函数，而softmax能够使神经网络的前向传播结果变成一个概率分布，每个输出在0~1之间。并且softmax函数在反向传播更新参数时，计算非常简单。

12．Word2vec的两种优化方法

- hierarchical softmax：本质是把 N 分类问题变成 log(N)次二分类

根据单词出现频率构建好的huffman树，沿着路径从根节点到对应的叶子节点，一层一层的利用sigmoid函数做二分类，判断向左还是向右走，规定沿着左子树走，那么就是负类(霍夫曼树编码1)，沿着右子树走，那么就是正类(霍夫曼树编码0)。一路上的概率连乘，最终得到某个单词的输出概率。这样做的好处是：原先一个full softmax需要一次计算所有的词（n），而hierarchical softmax却只需要计算大约（即树根到该叶子节点的路径长度）个词log（n），大大减少了计算的复杂度。当然，如果中心词是一个很生僻的词，还是需要在霍夫曼树中向下走很久，这也是它的一个不可避免的缺点。

- negative sampling：本质是预测总体类别的一个子集

思想：**一个词在整篇文本中出现的频率越高，它出现在训练词周围的概率自然也相对较高**。分层softmax在每次循环迭代过程中依然要处理大量节点上

的更新运算，而负采样技术只需更新“输出向量”的一部分。负抽样的目的是为了最终输出的上下文单词（正样本）[基于训练样本的半监督学习]在采样过程中应该保留下来并更新，同时也需要采集部分负样本（非上下文单词）。通过负采样，在更新隐层到输出层的权重时，只需更负采样的单词，而不用更新词汇表所有单词，从而节省巨大计算量。

13．是否一定要用Huffman tree?

未必，比如用完全二叉树也能达到O(log(N))复杂度。Huffman 树是带权路径和最小的树。 Huffman tree 被证明是更高效、更节省内存的编码形式，所以相应的权重更新寻优也更快。 举个简单例子，高频词在Huffman tree中的节点深度比完全二叉树更浅，比如在Huffman tree中深度为3，完全二叉树中深度为5，则更新权重时，Huffmantree只需更新3个w，而完全二叉树要更新5个，当高频词频率很高时，算法效率高下立判。

14．负采样的样本是怎么选择的

使用 一元模型分布 (unigram distribution) 来选择 negative words，**一个单词被选作 negative sample 的概率跟它出现的频次有关，出现频次越高的单词越容易被选作negative words**。

$$P(w_i) = \frac{f(w_i)^{0.75}}{\sum_{j=0}^{n}(f(w_j)^{0.75}})$$

```
def sample(n, cnt):
    a = i = 0
    table = []
    prob = 0
    z = sum(num ** 0.75 for num in cnt) # denominator

    prob = cnt[i] ** 0.75 / z # cumulative probability
    for a in range(n):
        '''
        loop invariant:
            at any time, a <= prob * n,
            with the same i, largest a = prob * n,
                             smallest a = prob_old * n + 1,
                             largest a - smallest a + 1 = prob * n - prob_old * n,
            which is the expectation of count of i.
        '''
        table.append(i)
        if a > prob * n:
            i += 1
            prob += cnt[i] ** 0.75 / z

    return table
```

```
if __name__ == '__main__':
    from collections import Counter

    # count, aka frequency
    cnt = [1, 2, 3, 100, 15]
    # 0.75 power
    prob = [x ** 0.75 for x in cnt]
    prob = [x / sum(prob) for x in prob]
    print(prob)

    # sampling
    res = Counter(sample(50000, cnt))
    print([x / sum(res.values()) for x in res.values()])
```


15. 为什么要去优化

Word2vec 本质上是一个语言模型，它的输出节点数是 V 个，对应了 V 个词语，本质上是一个多分类问题，但实际当中，词语的个数非常非常多，带入softmax公式中会发现每一次计算都要把整个词典过一遍，计算量非常大。因此才有了negative sampling和分层SOFTMAX，用以减少计算量的同时期望达到近似效果。

16．CBOW与skip-gram哪个更快，为什么?

训练速度上 CBOW 应该会更快一点。因为CBOW每次会更新 context(w) 的词向量，cbow预测行为的次数跟整个文本的词数几乎是相等的，时间复杂度分别是 O(V)。在skip-gram中，因为每个词在作为中心词时，都要使用周围词进行预测一次，这样相当于比cbow的方法多进行了K次（假设K为窗口大小），因此时间的复杂度为O(KV)，训练时间要比cbow要长。

17．CBOW与skip-gram哪种模型对罕见词的处理能力更好?

在skip-gram当中，每个词都要受到周围的词的影响，**每个词在作为中心词的时候，都要进行K次的预测、调整。**因此， 当数据量较少，或者词为生僻词出现次数较少时， 这种多次的调整会使得词向量相对的更加准确。因为尽管cbow从另外一个角度来说，某个词也是会受到多次（主要看共现次数）周围词的影响，进行词向量的跳帧，但是他的调整是跟周围的词一起调整的，grad的值会平均分到该词上， 相当于该生僻词没有收到专门的训练，它只是沾了周围词的光而已。

因此，从更通俗的角度来说：**在skip-gram里面，每个词在作为中心词的时候，实际上是 1个学生 VS K个老师，K个老师（周围词）都会对学生（中心词）进行“专业”的训练，这样学生（中心词）的“能力”（向量结果）相对就会扎实（准确）一些，但是这样肯定会使用更长的时间；cbow是 1个老师 VS K个学生，K个学生（周围词）都会从老师（中心词）那里学习知识，但是老师（中心词）是一视同仁的，教给大家的一样的知识。至于你学到了多少，还要看下一轮（假如还在窗口内），或者以后的某一轮，你还有机会加入老师的课堂当中（再次出现作为周围词），跟着大家一起学习，然后进步一点。因此相对skip-gram，你的业务能力肯定没有人家强，但是对于整个训练营（训练过程）来说，这样肯定效率高，速度更快。**

18．word2vec和fasttext的异同

相同点：

- fasttext与CBOW结构相似，多加了字符级别的n-gram丰富词向量
- 都可以去做无监督学习
- 都可采用两种加速方法
- 损失函数都是交叉熵损失

不同点：

- 目标不同，fasttext是分类，word2vec是预测上下文或中心词
- fastText还可以进行有监督学习进行文本分类
- fastText引入N-gram，考虑词序特征
- fastText引入subword来处理长词，处理未登陆词问题
- fastText分层Softmax的叶子结点（类别）相对w2v（所有词汇）少很多，样本中标签多的类别被分配短的搜寻路径

19．文本表示方法

- 基于one-hot、tf-idf、textrank等的bag-of-words；
- 主题模型：LSA（SVD）、pLSA、LDA；
- 基于词向量的固定表征：word2vec、fastText、glove
- 基于词向量的动态表征：elmo、GPT、bert、XLnet

20．word2vec的优缺点

优点：

- 由于 Word2vec 会考虑上下文，跟之前的 Embedding 方法相比，效果要更好
- 比之前的 Embedding方法维度更少，所以速度更快
- 通用性很强，可以用在各种 NLP 任务中

缺点：

- 由于词和向量是一对一的关系，一词多义无法解决
- context 很小，没有使用全局的cooccur，所以实际上对cooccur的利用很少

- [ ] https://www.cnblogs.com/zhangyang520/p/10969975.html <https://www.cnblogs.com/zhangyang520/p/10969975.html>
- [ ] https://blog.csdn.net/wisimer/article/details/104688095/ <https://blog.csdn.net/wisimer/article/details/104688095/>
- [ ] https://zhuanlan.zhihu.com/p/26306795/ <https://zhuanlan.zhihu.com/p/26306795/>
- [ ] https://zhuanlan.zhihu.com/p/86680049 <https://zhuanlan.zhihu.com/p/86680049>
- [ ] https://zhuanlan.zhihu.com/p/37477611 <https://zhuanlan.zhihu.com/p/37477611>

https://zhuanlan.zhihu.com/p/56382372

<https://zhuanlan.zhihu.com/p/56382372>