

Embedding

one-hot vector

喜欢 = [0 0 0 0 0 0 0 0 1 0 0 0]

喜爱 = [0 0 0 0 0 0 1 0 0 0 0 0]

讨厌 = [0 0 0 0 0 0 0 1 0 0 0 0 0]

dimension=| V |

缺点：

- 维度过大，表示和存储过于浪费， $|V|$ 个词每个 $R^{|V|}$ ($50000 \times 50000 \times 4$ bytes $\approx 954\text{MB}$)
- 存在稀疏性问题
- 词向量间不存在相似性度量

$$(w_{\text{喜欢}})^T w_{\text{喜爱}} = (w_{\text{喜欢}})^T w_{\text{讨厌}} = 0$$

But: 我们希望 喜欢 和 喜爱 有很强的相似性

Why similarity?

需要理解词的内在 meaning 而不是表面形式

Window Based methods

Distributional hypothesis: words that occur in the same contexts tend to have similar meanings

目标：

- 降低维度
- 构建 meaning

我们通过词的上下文信息 (context) 来构建词的 meaning，这里上下文可以简单理解为以词为中心的 window 内的单词，以一个小 corpus 为例

I enjoy flying.

I like NLP.

I like deep learning.

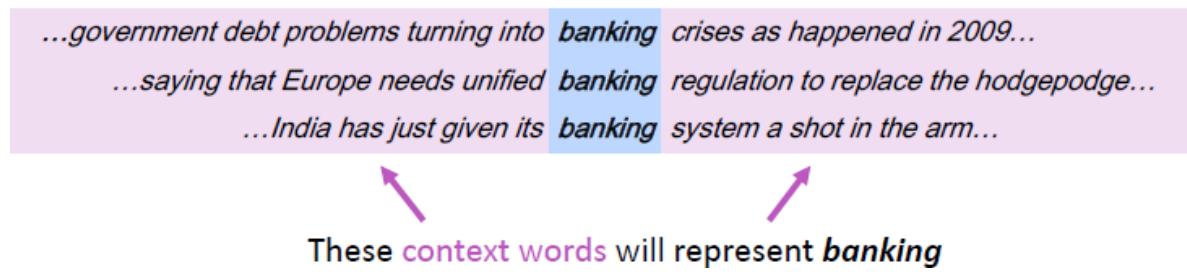
	<i>I</i>	<i>like</i>	<i>enjoy</i>	<i>deep</i>	<i>learning</i>	<i>NLP</i>	<i>flying</i>	.
<i>I</i>	0	2	1	0	0	0	0	0
<i>like</i>	2	0	0	1	0	1	0	0
<i>enjoy</i>	1	0	0	0	0	0	1	0
<i>deep</i>	0	1	0	0	1	0	0	0
<i>learning</i>	0	0	0	1	0	0	0	1
<i>NLP</i>	0	1	0	0	0	0	0	1
<i>flying</i>	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

缺点：

- 矩阵的维度变化相当频繁——新词的出现需要重新构建词共现矩阵 (cooccurrence matrix)
- sparse问题依然存在，没有考虑到某些词根本不会共同出现的情况
- 现有常用的解决方案
 - svd

word2vec

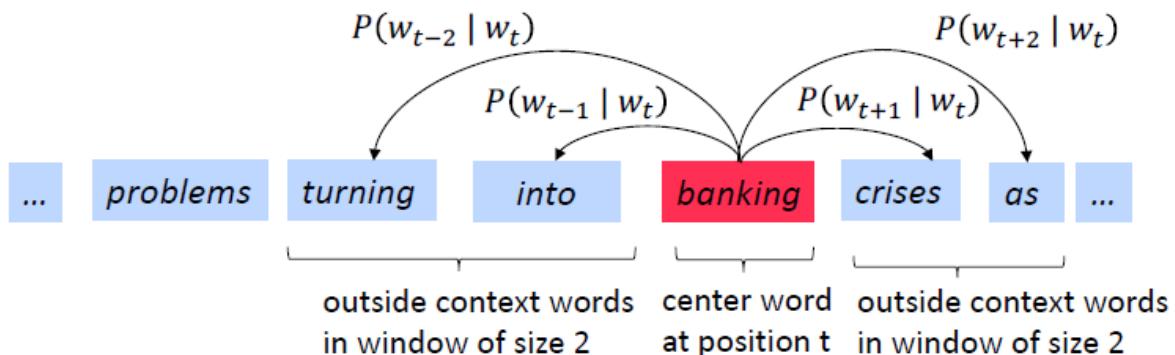
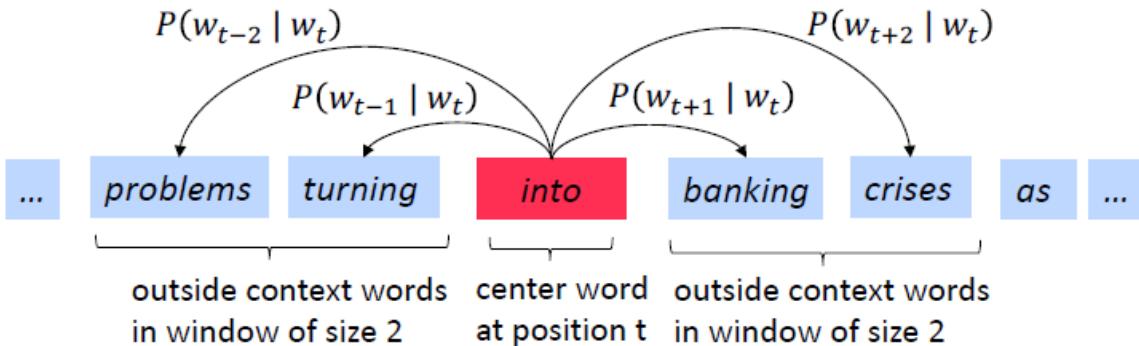
Distributional hypothesis: words that occur in the same contexts tend to have similar meanings



既然我们有上述假设，即一个词的 meaning 是由其上下文所决定的，那我们更近进一步

- 能否利用 **banking** (中心词, center word) 去预测上下文的词——Skip-grams (SG)
- 能否利用上下文的词去预测 **banking** (中心词)——Continuous Bag of Words (CBOW)

Skip-grams



数据集的构建过程如下

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

thou	shalt	not	make	a	machine	in	the	...
------	-------	-----	------	---	---------	----	-----	-----

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

最终我么想要的似然函数如下

$$L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j}|w_t; \theta) \quad (1.1)$$

转换成模型所需要的目标函数(负对数似然)

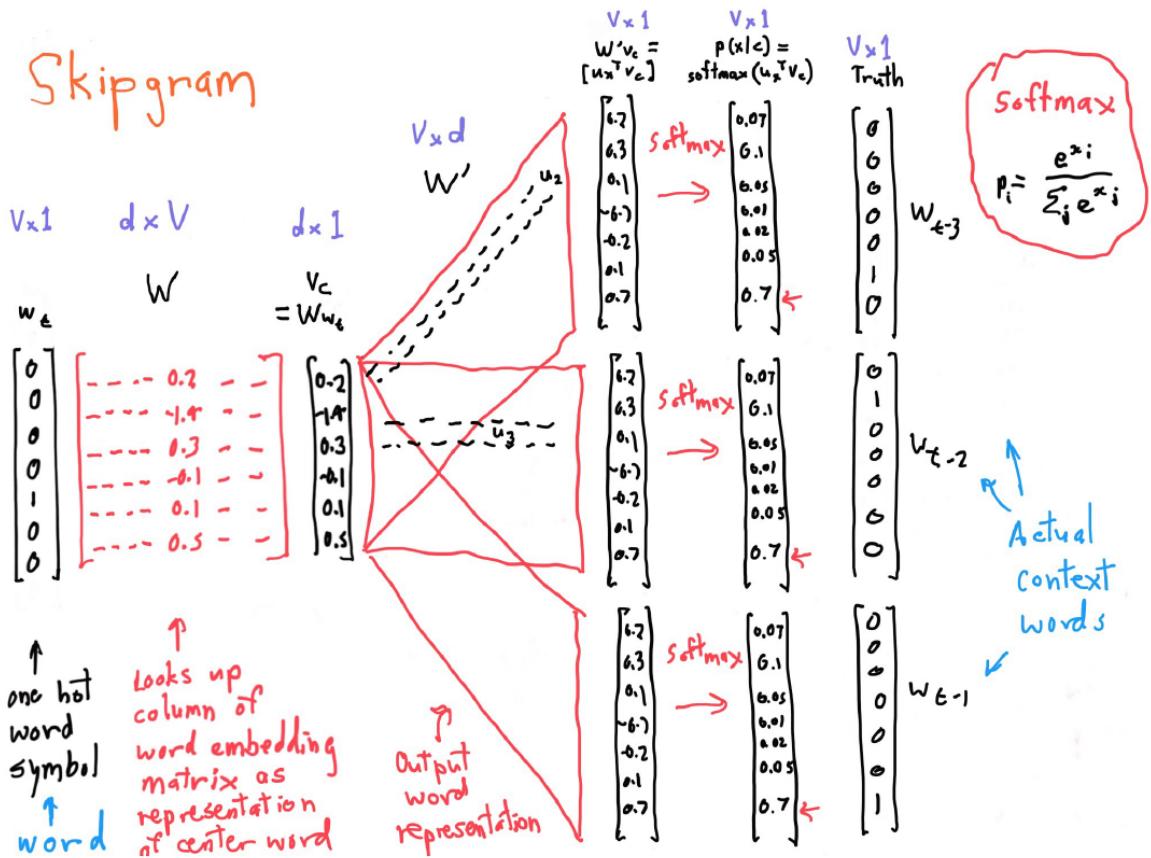
$$\begin{aligned} J(\theta) &= -\frac{1}{T} \log L(\theta) \\ &= -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m} \log P(w_{t+j}|w_t; \theta) \end{aligned}$$

其中最为关键的部分，是 P 的计算

$$p(o|c) = \frac{\exp(u_o^\top v_c)}{\sum_{w=1}^{|V|} \exp(u_w^\top v_c)} \quad (1.3)$$

对照下面的网络图，我们再来看看公式(1.3)

Skipgram



注意网络图的最后一层计算 loss 的时候我们选用交叉熵，和上面的负对数似然的关系如下

$$CE(\hat{y}, y) = - \sum_{j=1}^{|V|} y_j \log(\hat{y}_j)$$

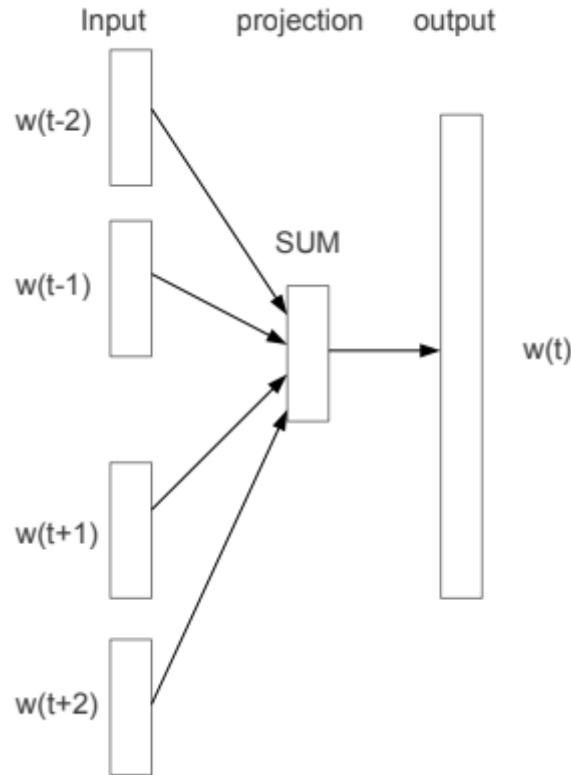
其中 \hat{y} 代表 softmax 后的概率， y 代表 gold label，是 one-hot 向量，所以上式可以进一步化简

$$CE(\hat{y}, y) = -y_i \log(\hat{y}_i)$$

从网络层面看

$$\begin{aligned} J(\theta) &= \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m} CE(\hat{y}_{t+j}, y_{t+j}) \\ &= \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m} -y_{t+j} \log(\hat{y}_{t+j}) \\ &= \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m} -1 \cdot \log(\hat{y}_{t+j}) \\ &= \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m} -\log P(w_{t+j}|w_t; \theta) \\ &= -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m} \log P(w_{t+j}|w_t; \theta) \end{aligned}$$

CBOW



相应的损失函数为

$$\begin{aligned}
 J(\theta) &= -\frac{1}{T} \sum_{t=1}^T \log P(w_c | w_{c-m}, \dots, w_{c-1}, w_{c+1}, \dots, w_{c+m}) \\
 &= -\frac{1}{T} \sum_{t=1}^T \log P(w_c | \hat{w}) \\
 &= -\frac{1}{T} \sum_{t=1}^T \log \frac{\exp(w_c^T \hat{w})}{\sum_{j=1}^{|V|} \exp(w_j^T \hat{w})}
 \end{aligned}$$

其中

$$\hat{w} = \frac{v_{w-m} + v_{w-m+1} + \dots + v_{w+1} + v_{w+m}}{2m}$$

但是，上述模型存在一个问题，从计算的角度来看，softmax 层的计算是极其费时的，特别是我们将对数据集中的每个训练样本执行一次（很容易执行数千万次）。我们需要一些 tricks 提高性能。由此，便有了以下两种改进算法

- Negative Sampling
- Hierarchical Softmax

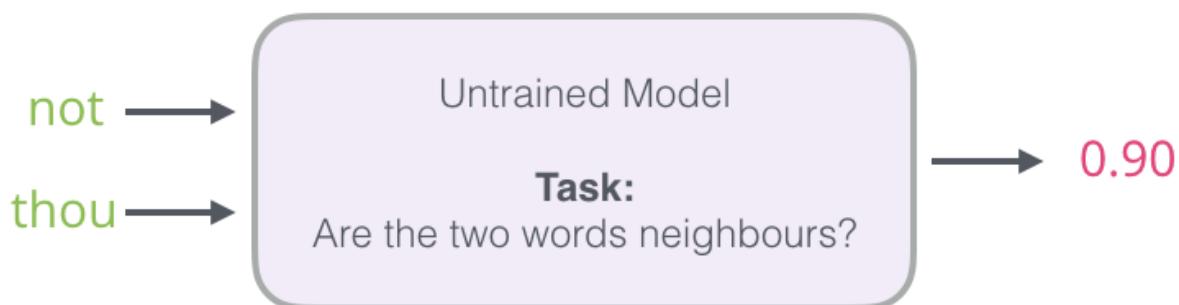
Negative Sampling

[注] 可以忘了之前的模型了，只需记得之前我们用点积考察了相似性

在之前的方法中，我们用 center word 去预测 context word，或者用 context word 去预测 center word，这近似可以看做是一个生成式任务，我们通过 word 去预测 word



现在我们想跳过 softmax 这种计算量很大的 operation，我们换一个思路，既然我们有两种 source 的 word，那我们索性直接预测两种 source 的 word 是否同属于上下文关系



这样的话问题就简单太多了，我们仅仅需要一个model，输入是两个词向量，输出是一个连续的 score 值，值的大小在一定程度上反应了两个词的相似程度，或者两个词是上下文关系的信任程度，位于区间 [0, 1]，很自然的，我们考虑使用 sigmoid 函数对 点积 的结果进行压缩

$$P(D = 1|w, c, \theta) = \sigma(v_c^T v_w) = \frac{1}{1 + e^{(-v_c^T v_w)}}$$

但是我们目前也仅仅只有之前收集好的，center word 和 context word 这种 pair 作为输入，这种输出的 gold label 毋庸置疑是 1.0，那我们的负样本从何而来，这就是 Negative Sampling 名称的由来

Pick randomly from vocabulary (random sampling)

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Word Count Probability

aardvark		
aarhus		
aaron		
taco		
thou		
zyzzyva		

Arrows point from the words "aaron" and "taco" in the table to their respective rows in the word count table.

具体做法：

对于每一个正样本，构造 K negative samples 辅助训练，负样本采样于分布 $P_n(w)$

$$P_n(w_i) = \frac{P(w_i)^{\frac{3}{4}}}{\sum_{j=0}^n \left(P(w_j)^{\frac{3}{4}}\right)}$$

其中 $P(w)$ 每个单词 w 的 Unigram Model

Why $\frac{3}{4}$?

- 提高低频词的采样概率，进行更充分的学习

$$\text{李: } 0.90^{\frac{3}{4}} = 0.924 \Rightarrow \frac{0.924}{0.924+0.164+0.032} = 0.825$$

$$\text{高: } 0.09^{\frac{3}{4}} = 0.164 \Rightarrow \frac{0.164}{0.924+0.164+0.032} = 0.146$$

$$\text{谯: } 0.01^{\frac{3}{4}} = 0.032 \Rightarrow \frac{0.032}{0.924+0.164+0.032} = 0.028$$

- 实验效果较好

所以 Negative Sampling 改进的 Skipgram 模型的输入数据大致如下

Skipgram					Negative Sampling		
shalt	not	make	a	machine	input word	output word	target
make		input	output		make	shalt	1
make			shalt		make	aaron	0
make		make	not		make	taco	0
make			a				
make			machine				

似然函数表达式

$$\begin{aligned}
 L(\theta) &= \prod_{(w,c) \in T} P(D = 1|w, c, \theta) \prod_{(w,c) \in F} P(D = 0|w, c, \theta) \\
 &= \prod_{(w,c) \in T} P(D = 1|w, c, \theta) \prod_{(w,c) \in F} (1 - P(D = 1|w, c, \theta)) \\
 &= \sum_{(w,c) \in T} \log P(D = 1|w, c, \theta) + \sum_{(w,c) \in F} \log(1 - P(D = 1|w, c, \theta)) \\
 &= \sum_{(w,c) \in T} \log \frac{1}{1 + \exp(-u_w^T v_c)} + \sum_{(w,c) \in F} \log \left(1 - \frac{1}{1 + \exp(u_w^T v_c)}\right) \\
 &= \sum_{(w,c) \in T} \log \frac{1}{1 + \exp(-u_w^T v_c)} + \sum_{(w,c) \in F} \log \left(\frac{1}{1 + \exp(u_w^T v_c)}\right)
 \end{aligned}$$

目标函数为

$$J = - \sum_{(w,c) \in T} \log \frac{1}{1 + \exp(-u_w^T v_c)} - \sum_{(w,c) \in F} \log \left(\frac{1}{1 + \exp(u_w^T v_c)}\right)$$

所以我们的目标彻底从预测问题转换成了二分类问题！

Hierarchical Softmax

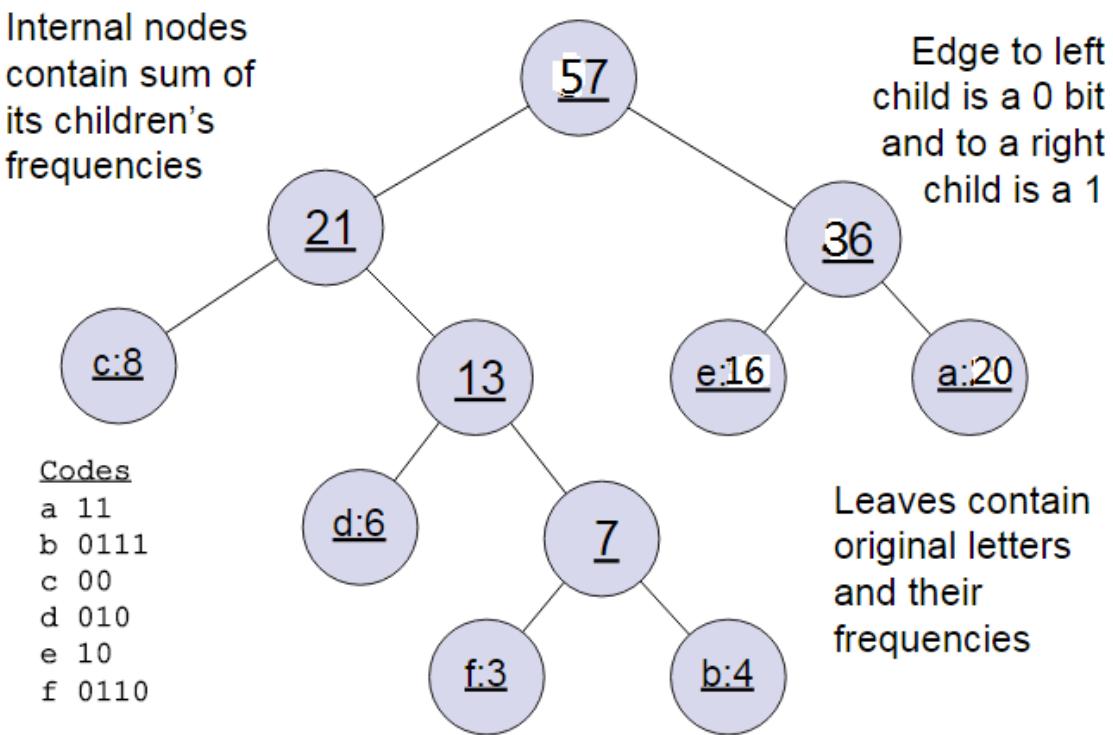
霍夫曼树(Huffman树)

输入：权值为 (w_1, w_2, \dots, w_n) 的 n 个节点

输出：霍夫曼树

- 1) 选择权值最小的两棵树进行合并，这两棵树分布作为新树的左右子树。新树的根节点权重为左右子树的根节点权重之和。
- 2) 将之前的根节点权值最小的两棵树从森林删除，并把新树加入森林。
- 3) 重复步骤 1) 和 2) 直到森林里只有一棵树为止。

e.g. (a,b,c,d,e,f) 6个节点，权值(20,4,8,6,16,3)

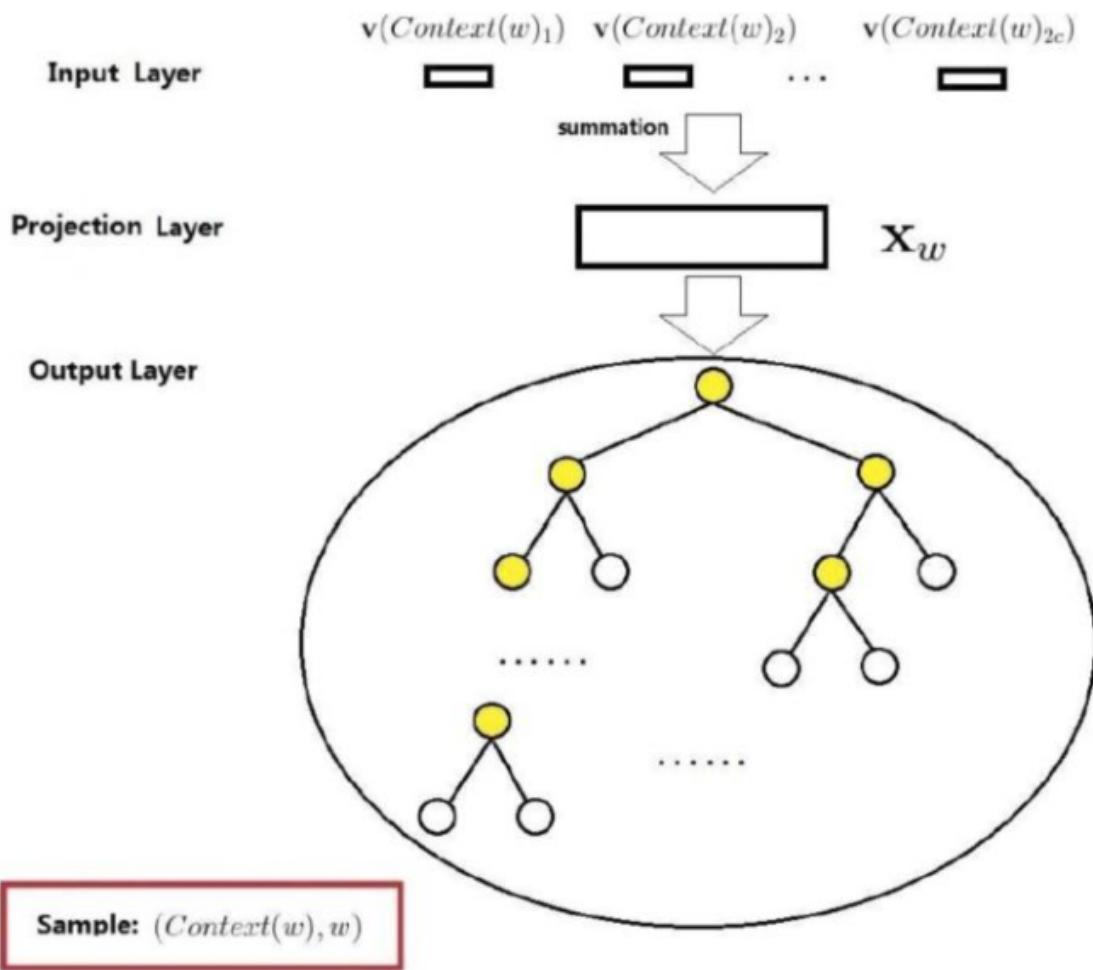


优势

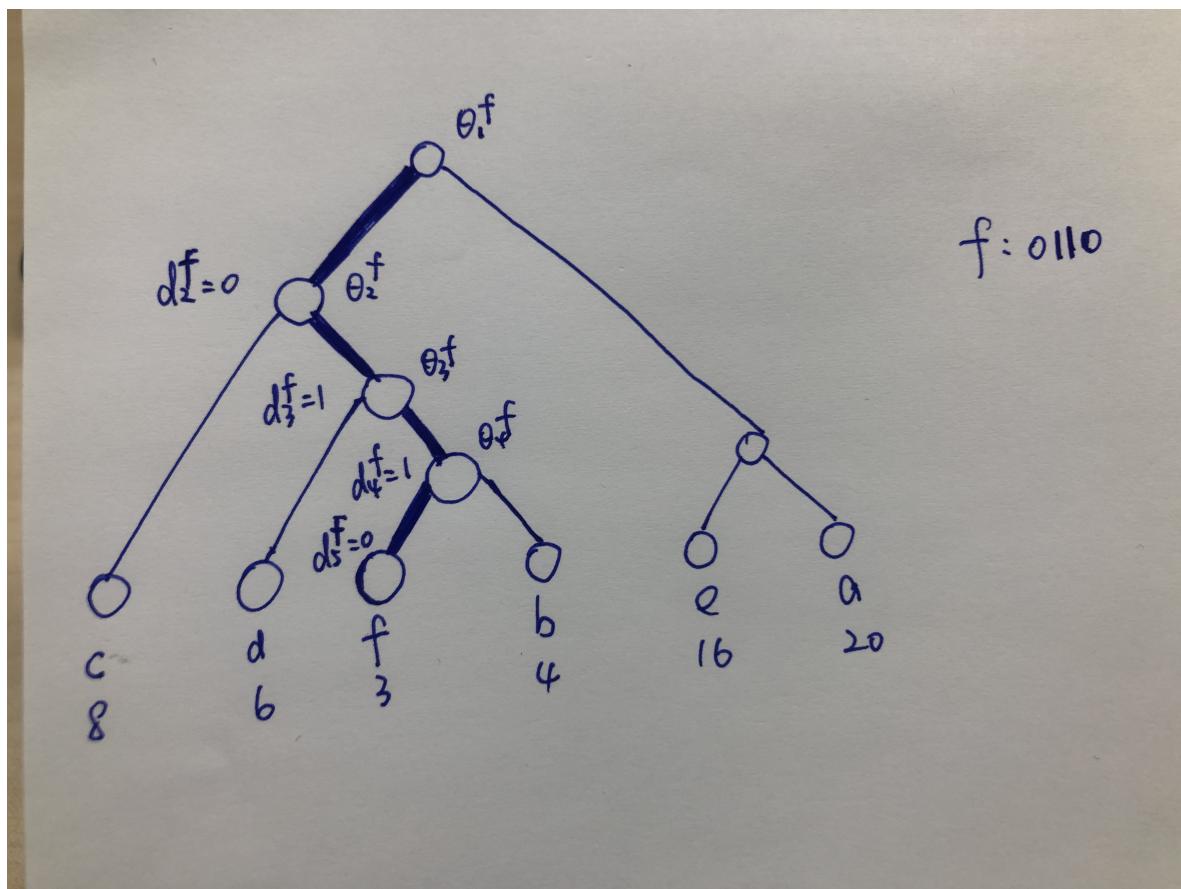
- 高权重节点编码值较短，而低权重值编码值较长
 - 常用的词拥有更短的编码
- 带权路径长度最短的二叉树
- 计算量 $O(\log(|V|))$

CBOW + Hierarchical Softmax

总体概览



我们 Zoom in



在开始计算之前，引入一些符号：

- l^w : 到达给定词的路径中包含结点的个数
 - $l^f = 5$
- $d_2^w, d_3^w, \dots, d_{l^w}^w \in \{0, 1\}$: 词 w 的编码，根节点无编码
 - $d_2^f = 0, d_3^f = 1, d_4^f = 1, d_5^f = 1$
- $\theta_1^w, \theta_2^w, \dots, \theta_{l^w-1}^w \in \mathbb{R}^m$: 路径中非叶节点对应的参数向量

经过的霍夫曼树某一个节点 j 的概率定义为

$$P(d_j^w | x_w, \theta_{j-1}^w) = \begin{cases} \sigma(x_w^T \theta_{j-1}^w) & d_j^w = 0 \\ 1 - \sigma(x_w^T \theta_{j-1}^w) & d_j^w = 1 \end{cases}$$

考虑到 d 只有0和1两种取值，我们可以用指数形式方便地将其写到一起：

$$p(d_j^w | x_w, \theta_{j-1}^w) = [\sigma(x_w^T \theta_{j-1}^w)]^{1-d_j^w} \cdot [1 - \sigma(x_w^T \theta_{j-1}^w)]^{d_j^w}$$

那么对于某一个目标输出词 w ，其最大似然为：

$$P(w|x_w) = \prod_{j=2}^{l_w} P(d_j^w | x_w, \theta_{j-1}^w) = \prod_{j=2}^{l_w} [\sigma(x_w^T \theta_{j-1}^w)]^{1-d_j^w} [1 - \sigma(x_w^T \theta_{j-1}^w)]^{d_j^w}$$

对于所有样本来说，总似然函数为

$$L(\theta) = \prod_{w \in C} \prod_{j=2}^{l_w} P(d_j^w | x_w, \theta_{j-1}^w)$$

最终的目标函数为

$$J = -\frac{1}{|C|} \sum_{w \in C} \sum_{j=2}^{l_w} \log(P(d_j^w | x_w, \theta_{j-1}^w))$$

That's All. Thanks!