文本表征学习

4. Word2vec

EE1513

宋彦

Word2vec大纲

- 概念和基础
- 连续词袋模型(CBOW) 和 Skip-gram (SG)
- 层级化Softmax和负采样
- 评价
- 实现的细节

面向词的神经网络表征模型

- 词是语言中最基本的单位
  - 基本的语义信息
- 可以在很多任务中灵活的使用
- 便于学习

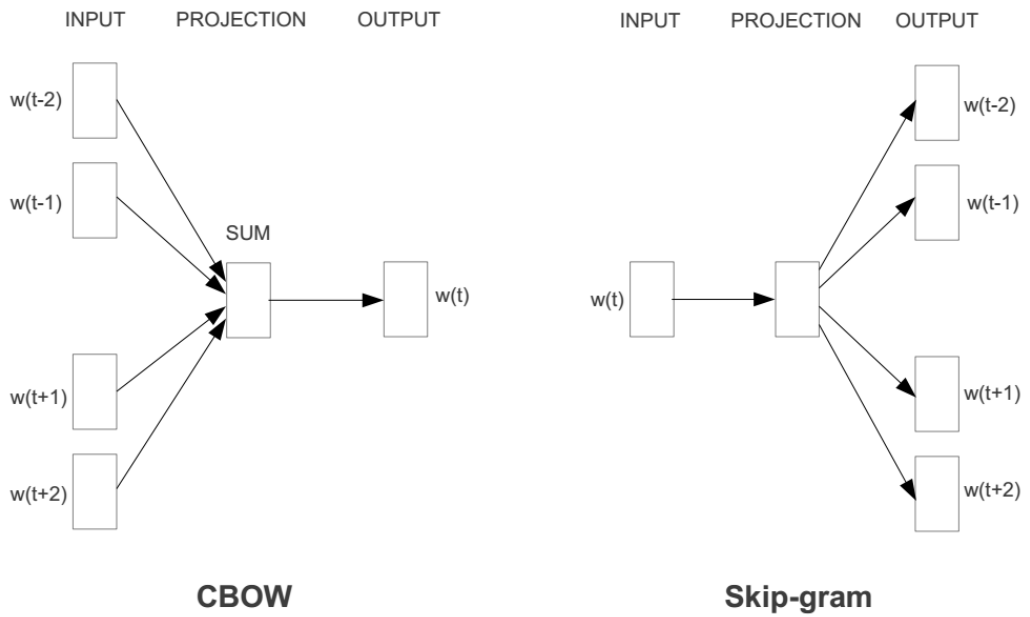
- 也有很多关于sub-words的研究

Word2vec的概念

- 核心观点：预测周围的词
- 动机：
  - 相似的词有详细的上下文
    - 高语义相关性词：“医生”-“医院”
    - 高句法相关性词：“上”-“下”
  - 我们可以从词之间的“共现”关系学习词的表征
- 特点：
  - 更快且可以轻松地将新的句子/文档或单词添加到词汇表中。
  - 通过间接矩阵分解的方式学习稠密表示（与GloVe相比）。

Word2vec的基础

- 两个基础模型:
  - 连续词袋(CBOW): 使用一个窗口中的词预测中心的目标词
  - Skip-gram (SG): 使用一个词预测周围的词



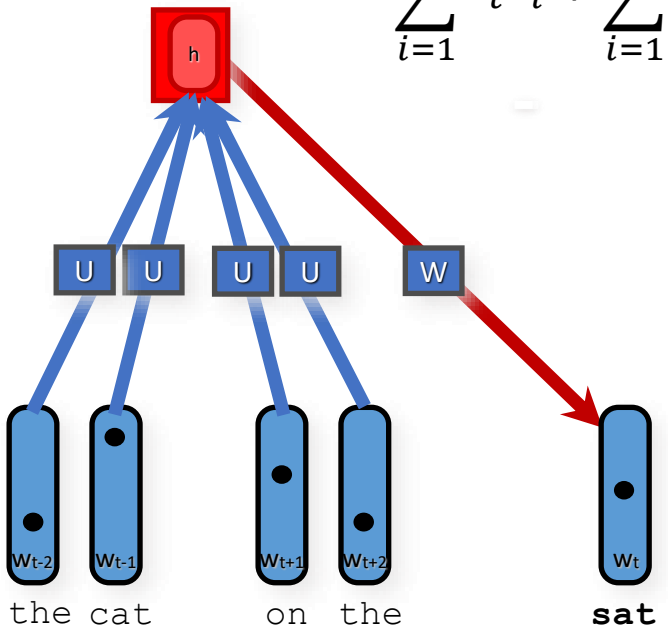
词袋模型(CBOW)

word embedding space $\Re^D$ in dimension D=100 to 300

Word embedding matrices

discrete word space {1, ..., V} V>100k words

Simple Sum $h = \sum_{i=1}^{c} z_{t-i} + \sum_{i=1}^{c} z_{t+i}$



$$o = W \cdot h$$

$$P(w_t | w_{t-c}^{t-1}, w_{t+1}^{t+c}) = \frac{\exp(o(w_t))}{\sum_v \exp(o(v))}$$

Extremely efficient estimation of **word embeddings** in matrix **U** **without a Language Model**.
Can be used as input to neural LM.
Enables much larger datasets, e.g., Google News (6B words, V=1M)

Complexity: $2C \times D + D \times V$

Complexity: $2C \times D + D \times \log(V)$ (**hierarchical softmax using tree factorization**)

[Mikolov et al, 2013a; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

CBOW

- 对于语料库中的每个单词，根据一个窗口选择上下文词。
  - 忽略窗口之外的上下文信息。
- 上下文词使用矩阵U映射到向量（嵌入）。
- 上下文词的向量逐元素相加。
  - 忽略上下文词的顺序和位置。
  - 所有上下文词被等同对待。
- 得到的向量通过W映射到输出空间。

$$h = \sum_{i=1}^{c} z_{t-i} + \sum_{i=1}^{c} z_{t+i}$$

$$o = W \cdot h$$

$$P(w_t | w_{t-c}^{t-1}, w_{t+1}^{t+c}) = \frac{\exp(o(w_t))}{\sum_v \exp(o(v))}$$

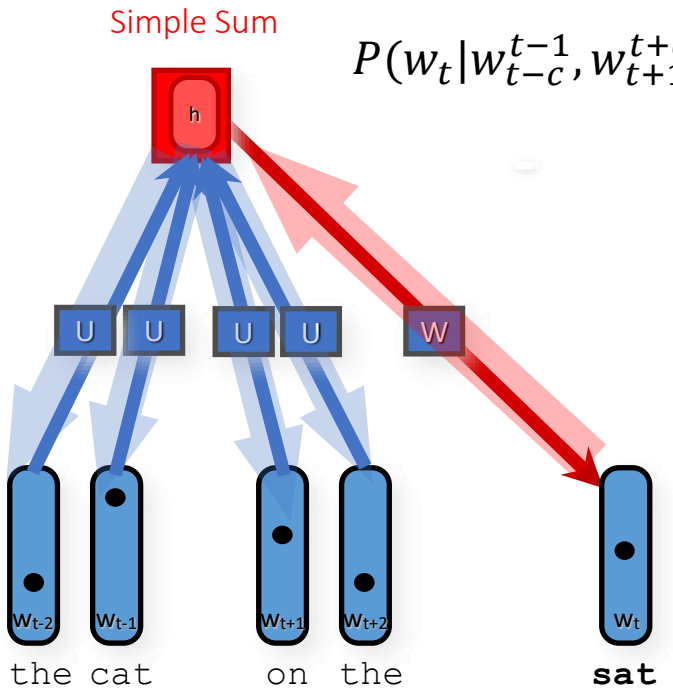
学习词向量

- U和W具有相同的大小，都是|V| x D。
  - V：词汇表大小
  - D：单词嵌入的维度
- 在训练中，U和W都会被更新。
- 相比W，U的更新更加强烈。
  - 我们有多个上下文词。
- U被用作单词的表示。

$$h = \sum_{i=1}^{c} z_{t-i} + \sum_{i=1}^{c} z_{t+i}$$

$$o = W \cdot h$$

$$P(w_t | w_{t-c}^{t-1}, w_{t+1}^{t+c}) = \frac{\exp(o(w_t))}{\sum_v \exp(o(v))}$$



CBOW vs. 神经网络语言模型（NLM）

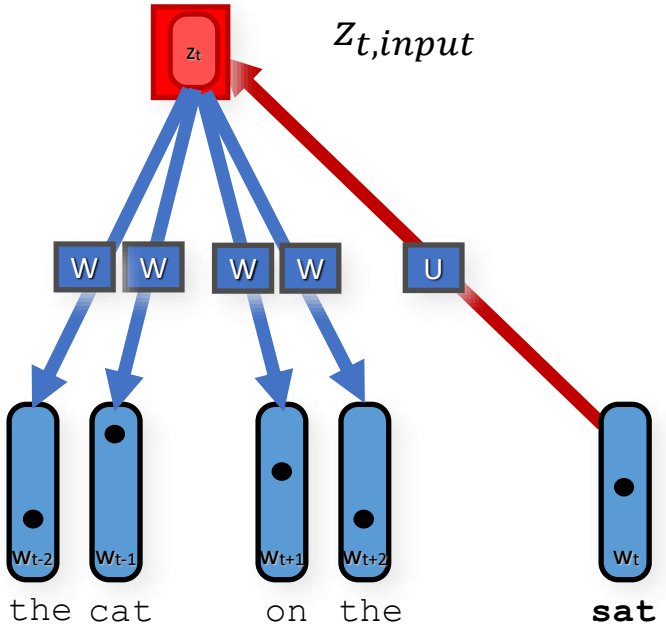
- 这两种方法有不同的目标：
  - CBOW用于训练词向量。
  - NLM用于语言建模。
- 输入方式也不同：
  - CBOW考虑左右两侧的上下文。
  - NLM只考虑左侧的上下文。
- 架构也不同：
  - CBOW：上下文词的嵌入进行求和。
  - NLM：滑动窗口中的单词嵌入进行串联。

Skip-gram (SG)

word embedding space $\Re^D$ in dimension D=100 to 1000

Word embedding matrices

discrete word space {1, ..., V} V>100k words



$$s_\theta(v,c) = z_{v,output}^T \cdot z_{t,input}$$

$$P(w_{t+c}|w_t) = \frac{\exp(s_\theta(w,c))}{\sum_v \exp(s_\theta(v,c))}$$

Extremely efficient estimation of word embeddings in matrix U without a Language Model.
Can be used as input to neural LM.
Enables much larger datasets, e.g., Google News (33B words, V=1M)

Complexity: $2C\times D + 2C\times D\times V$

Complexity: $2C\times D + 2C\times D\times \log(V)$ (hierarchical softmax using tree factorization)

Complexity: $2C\times D + 2C\times D\times (k+1)$ (negative sampling with k negative examples)

[Mikolov et al, 2013a, 2013b; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

Skip-gram算法

- 目标词通过矩阵U映射到其输入嵌入$z_{t,input}$
- 对于词汇表中的单词v，根据矩阵W找到其输出向量$z_{v,output}$。
- 计算$z_{t,input}$和$zz_{v,output}$的内积。
  - 较高的内积意味着单词v更有可能出现在目标词的上下文中，其中上下文也由一个窗口确定

$z_{t,input}$

$$s_\theta(v,c) = z_{v,output}^T \cdot z_{t,input}$$

$$P(w_{t+c}|w_t) = \frac{\exp(s_\theta(w,c))}{\sum_v \exp\big(s_\theta(v,c)\big)}$$

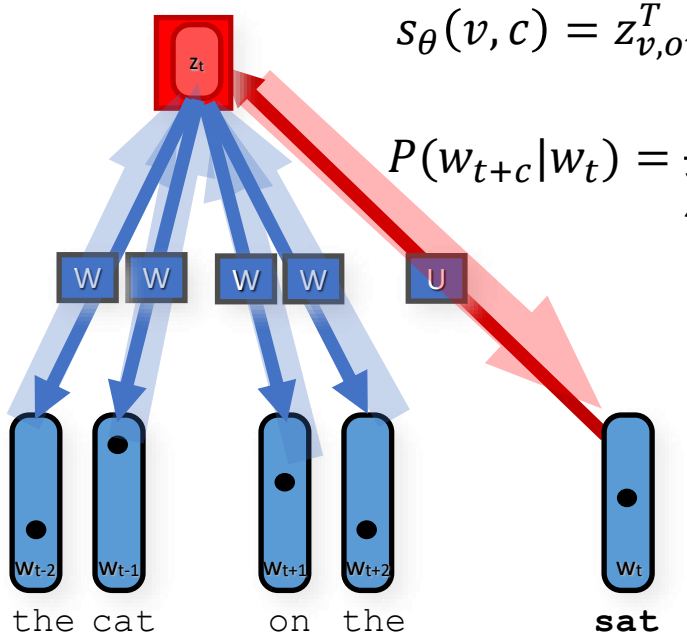
学习词向量

- U和W具有相同的大小，都是|V| x D。
  - V：词汇表大小
  - D：单词嵌入的维度
- 在训练中，U和W都会被更新。
- 相比W，U的更新更加强烈。
  - 我们需要使用相同的$z_{t,input}$来预测多个上下文词。
  - $z_{t,input}$会被多次更新。
- U被用作单词的表示。

$$z_{t,input}$$

$$s_\theta(v,c) = z_{v,output}^T \cdot z_{t,input}$$

$$P(w_{t+c}|w_t) = \frac{\exp(s_\theta(w,c))}{\sum_v \exp(s_\theta(v,c))}$$



Word2vec的进一步说明

- CBOW和SG都包含两个矩阵，即U和W，其中U用作单词嵌入。
- U和W具有相同的大小（|V| x D），这意味着：
  - W可以学习到很多上下文信息，但U中没有包含这些信息。
  - 计算整个词汇表上的概率并不高效。
- 单词具有不同的频率。
  - 频繁出现的单词的嵌入会比不常见单词的嵌入更新得更频繁。
  - 嵌入会过度拟合频繁出现的单词。

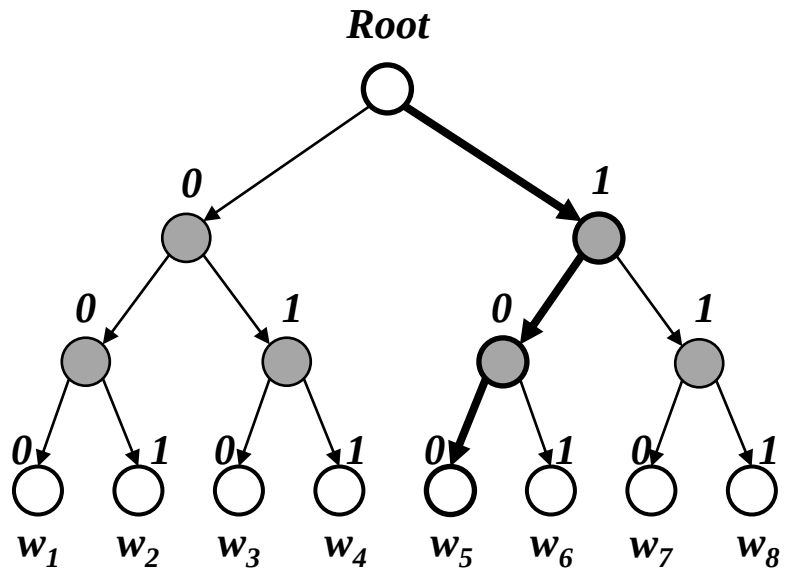
Word2vec的预测策略

- 层次Softmax（Hierarchical softmax, HS）
  - 多个连续的softmax经过一个二叉树结构
  - 对于具有相似频率模式的单词，能有效捕捉其信息

- 负采样（Negative sampling, NS）
  - 通过二元分类来区分一个单词是否在上下文中
  - 在训练数据庞大时效果很好（减少softmax操作）

层次Softmax

- 使用Huffman树来存储所有节点。
- 每个非叶节点与一个向量关联。
- 每个叶节点表示一个单词。
- 每个边都有一个标签（0或1）。

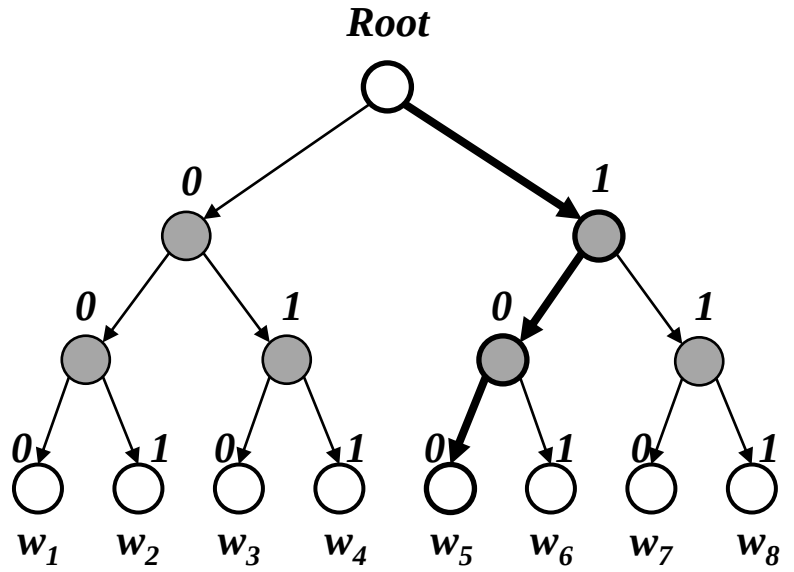


层次Softmax

- 对于每个叶节点，都存在一条独特的路径到根节点。

- 利用路径上边的标签，每个单词（叶节点）可以由一串0或1表示。
  - 例如， w_5 =1 0 0， w_3 =0 1 0。

- 预测单词等效于预测路径上的弧标签



Huffman 树

- Huffman编码：
  - 给定一个词汇表，包含V个单词 $w_1 \cdots w_V$ ，以及相应的词频（权重）$c_1 \cdots c_V$
  - 寻找一个使得期望编码长度最小的无重复前缀编码（即一组编码字）

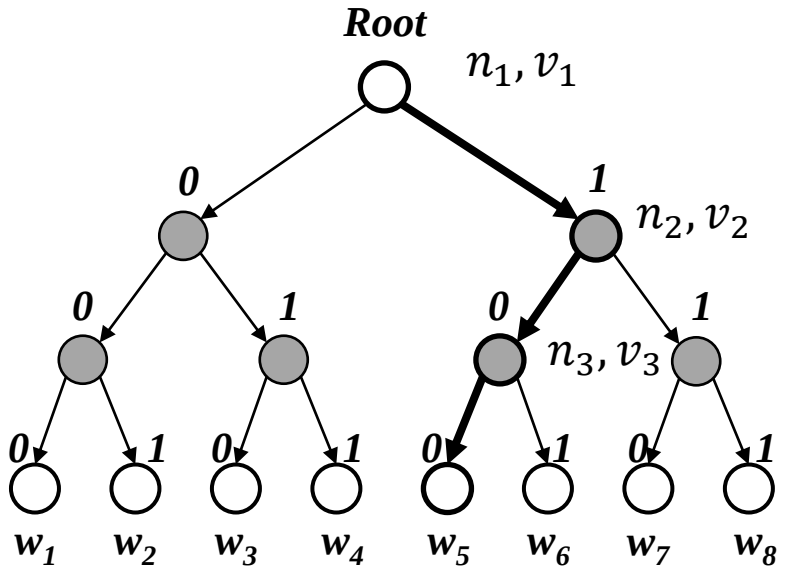
- 方法：
  - 构建一个Huffman树，每个单词的编码是从根节点到相应叶节点的路径

Huffman 树的构建方法

- 使用优先队列（priority queue），其中权重最低的节点具有最高优先级：
  - 为每个符号创建一个叶节点并将其添加到优先队列中。
  - 当队列中有多于一个节点时：
    - 从队列中移除两个具有最高优先级（即权重最低）的节点。
    - 创建一个新的内部节点，将这两个节点作为子节点，并且其权重等于这两个节点权重的和。
    - 将新节点添加到队列中。
  - 剩下的节点即为根节点，树构建完成。

层次Softmax

- 假设我们要预测的单词是w。
  - 在CBOW中，w是目标词。
  - 在SG中，w是上下文词。
- 等效于预测从根节点到w的路径上的标签。
- 我们将路径上的节点表示为$n_1,\cdots,n_d$，其中n_1是根节点，n_d是w对应的叶节点，d是w在树中的深度。



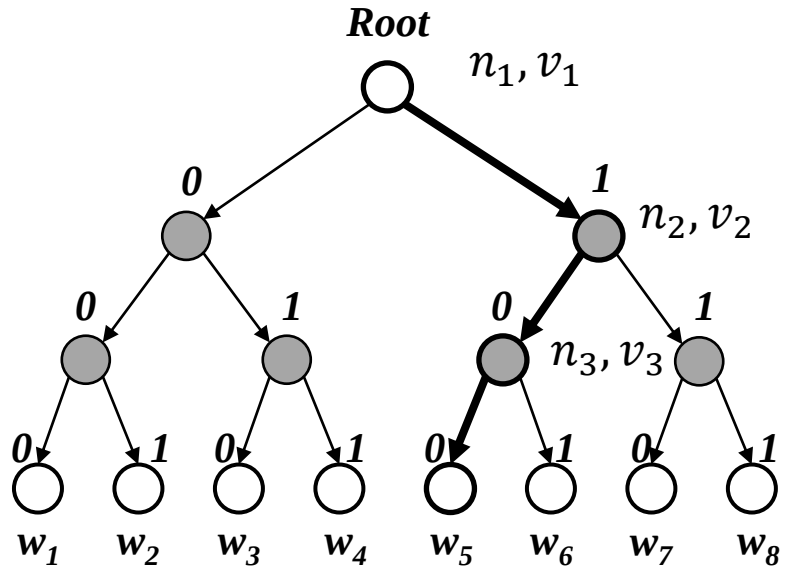
层次Softmax

- Word2vec模型需要预测路径上相邻节点n_i和n_{i+1}之间的标签$\hat{l}_i$。

$$P(\hat{l}_i = 1) = sigmoid(h \cdot v_i)$$

$$P(\hat{l}_i = 0) = 1 - P(\hat{l}_i = 1)$$

- 其中h是输入的隐藏向量
  - CBOW: h 是上下文词向量的和
  - SG: h 是目标词的向量
- v_i 是与 n_i 关联的可训练向量

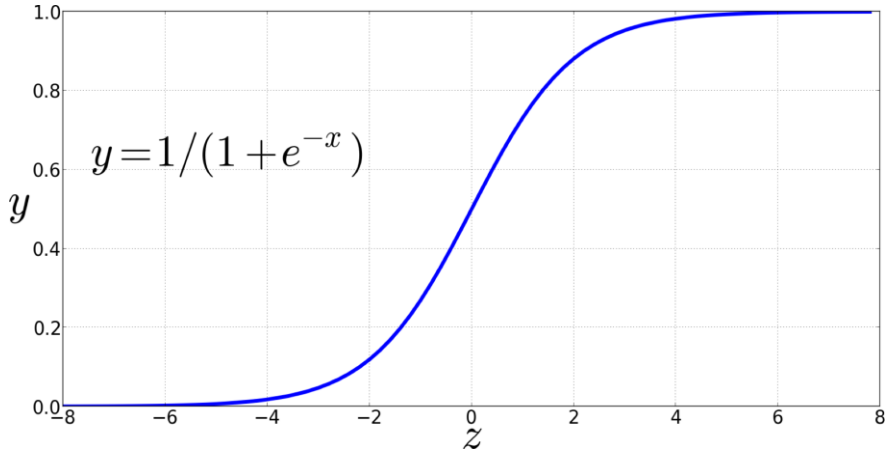


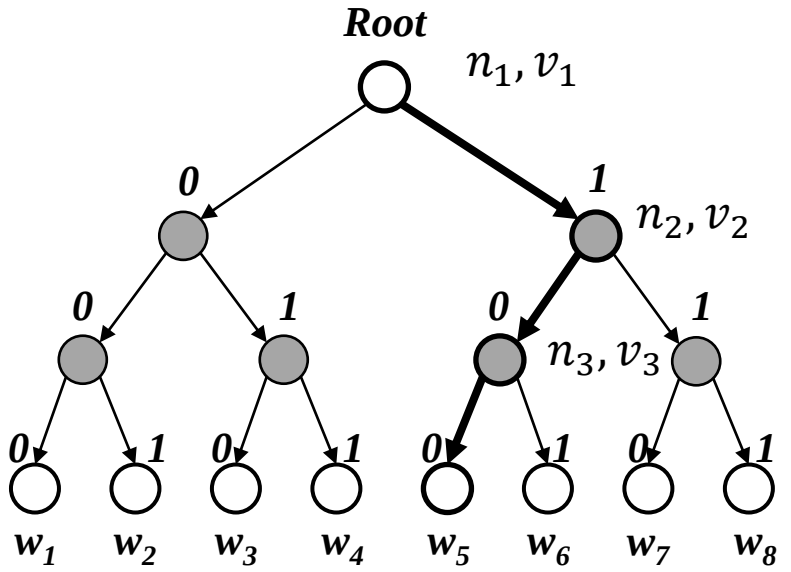
层次Softmax

- sigmoid 可以被看做是一种特殊的softmax

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$1 - sigmoid(x) = \frac{e^{-x}}{1+e^{-x}}$$



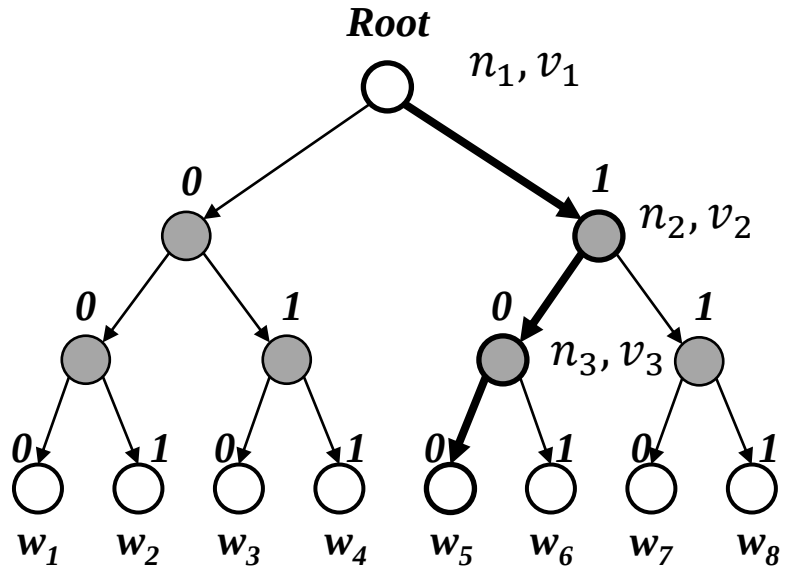


层次Softmax

- 全部的损失 J_θ 为

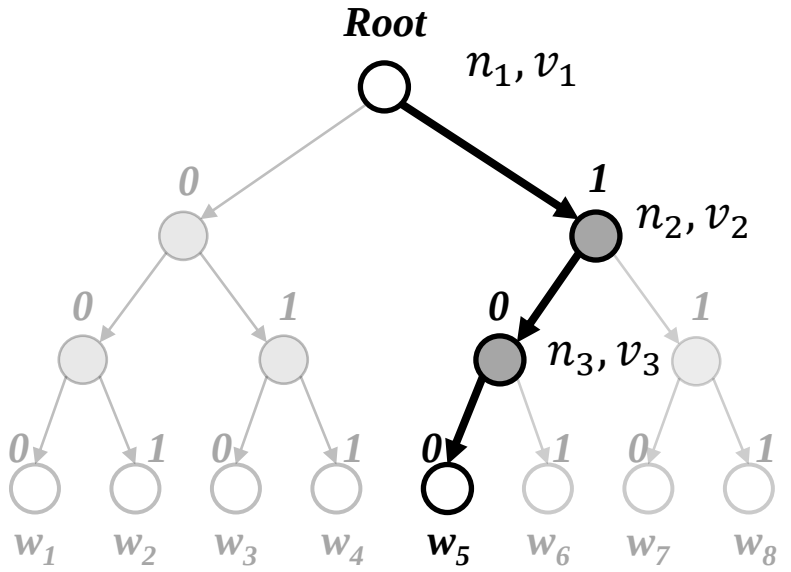
$$J_\theta = \sum_{i=1}^{d-1} L(l_i, \hat{l}_i)$$

- L 是损失函数
- l_i 是节点 n_i 到 n_{i+1} 之间边上的标签



层次Softmax

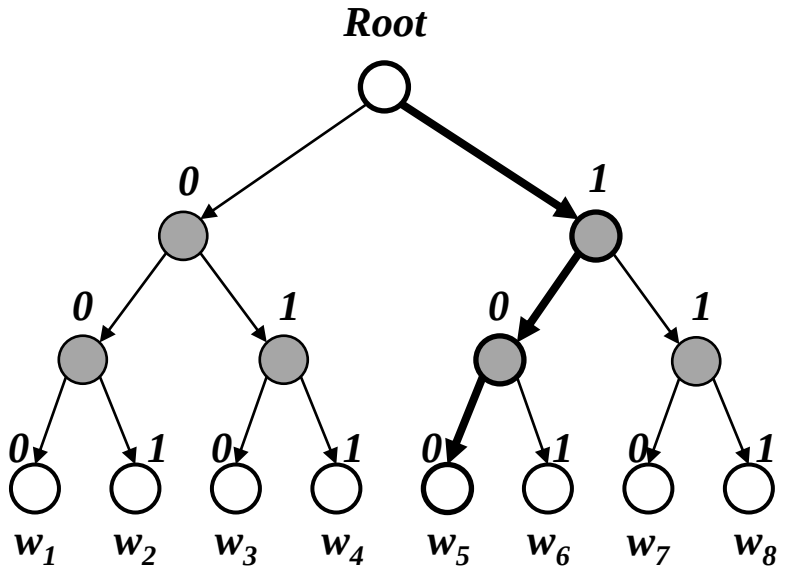
- 假设我们要预测的单词是w_5。
  - 在CBOW中， w_5是目标词。
  - 在SG中， w_5是上下文词。
- 等效于预测从根节点到w_5的路径上的标签，即1、0、0。



全部的损失: $J_\theta = L(l_1, \hat{l}_1) + L(l_2, \hat{l}_2) + L(l_3, \hat{l}_3)$

层次Softmax

- 较深的叶节点会导致更多的损失，从而导致更多的模型参数更新。
  - 例如，深度为4的单词会导致3次softmax计算
- Huffman树的特性导致频繁出现的单词具有较低的损失，而罕见的单词具有较高的损失。



层次Softmax总结

- 使用Huffman树，其中频繁出现的单词具有较低的深度，不常见的单词具有较高的深度。
- 将上下文/目标词预测定义为路径（从根节点到表示待预测单词的叶节点）的预测。
- 内积 $h \cdot v_i$ 平均计算次数是Huffman 树的平均深度即 $\log(V)$, 其中 V 是词表大小
  - 使得模型可以运行得更高效
- 高频词的loss比较低，可以解决过拟合问题

负采样

- 进一步将估计过程更改为一步二元分类：
  - 判断目标词是否"属于"当前上下文
- 不需要对数据进行实际的"预测”
  - 随机选择负例样本
  - 复杂度从 $d \times V$ 降低到$d \times 2$

负采样

- 训练句子 ：

... lemon, a tablespoon of apricot jam a pinch ...

c_1 c_2 t c_3 c_4

- 我们把上下文词/目标词的预测视为二分类问题
  - CBOW：给定上下文，判断词是否为中心词
    - 正例：给定 “tablespoon” “of” “jam” “a”，中心词是“apricot”
    - 反例：“tablespoon” “of” “jam” “a”，中心词不是“lemon”
  - SG：给定中心词，判断词是否为上下文词
    - 正例：给定 “apricot”，上下文词是“tablespoon”
    - 反例：给定 “apricot”，上下文词不是“pinch”

负采样

- 训练句子:

... lemon, a tablespoon of apricot jam a pinch ...

c_1 c_2 t c_3 c_4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

- 每个正例，随机采样 k 个负样本
- 使用噪音词
- 只要不是目标词的任意词

负采样

- 训练句子:

... lemon, a tablespoon of apricot jam a pinch ...

c_1 c_2 t c_3 c_4

k=2

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

负采样中的概率计算

- (t, c)：（目标，上下文）
- CBOW：

$$P(+|t, c) = sigmiod(h \cdot v_t)$$
$$P(-|t, c) = 1 - P(+|t, c)$$

h：上下文词向量之和

v_t：目标词的输出词向量

- SG：

$$P(+|t, c) = sigmiod(z_{c,output} \cdot z_{t,input})$$
$$P(-|t, c) = 1 - P(+|t, c)$$

$z_{c,output}$：上下文词的输出词向量

$z_{t,input}$：目标词的输入词向量

负采样的目标函数

- 最大化

$$\sum_{(t,c)\in+} \log P(+|t,c) + \sum_{(t,c)\in-} \log P(-|t,c)$$

- (t,c)：（目标，上下文）
- 最大化正训练数据中正例对的+标签，以及从负例数据中采样的负例对的−标签。

负采样：总结

- 从语料库中提取共现的词对作为正例
- 从不共现的词对中提取作为负例
- 通过逐渐调整所有词向量来训练模型，以提高分类器的性能，使其能够区分正例和负例

内部（intrinsic）评价

- 给定一个单词对的列表，测量每对单词之间的紧密程度，并将得分与预定义的得分进行比较
  - cosine
- 如何系统性地评价?
  - 准备一个人工标注的相似性/相关性数据集
    - 一个单词对的列表
  - 使用得到的词向量计算所有单词对之间的相似性
  - 计算人工标注与相似性之间的相关性

词相似度

- 余弦相似度是内部评价（intrinsic evaluation）
- 如何评价两组分数列表？
  - 皮尔逊相关系数
    Pearson’s correlation

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

  - 斯皮尔曼相关系数
    Spearman's correlation

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}.$$

词相似度

- 斯皮尔曼相关性
  Spearman's correlation

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}.$$

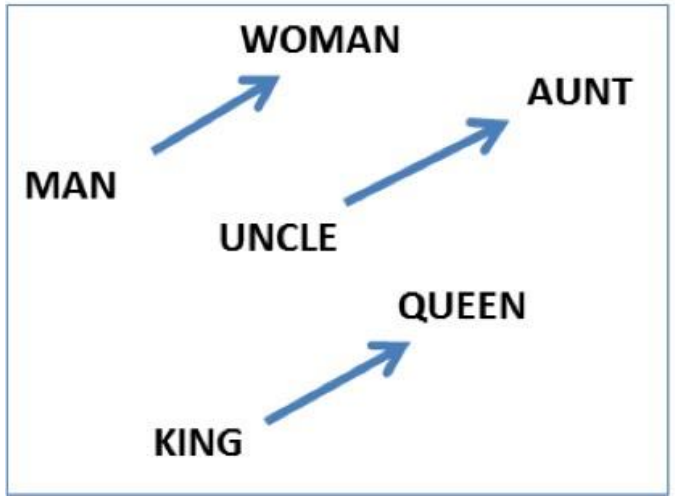
tiger cat	7.35	0.517977544
tiger tiger	10.00	1.0
plane car	5.77	0.217732013636
train car	6.31	0.197087634791
television radio	6.77	0.257970738766
media radio	7.42	0.212014745796
bread butter	6.19	0.7017263618
cucumber potato	5.92	0.396596853365
doctor nurse	7.00	0.382799131727
professor doctor	6.62	0.288006966845
student professor	6.81	0.1973114775
smart stupid	5.81	0.228546918905
wood forest	7.73	0.229380996826

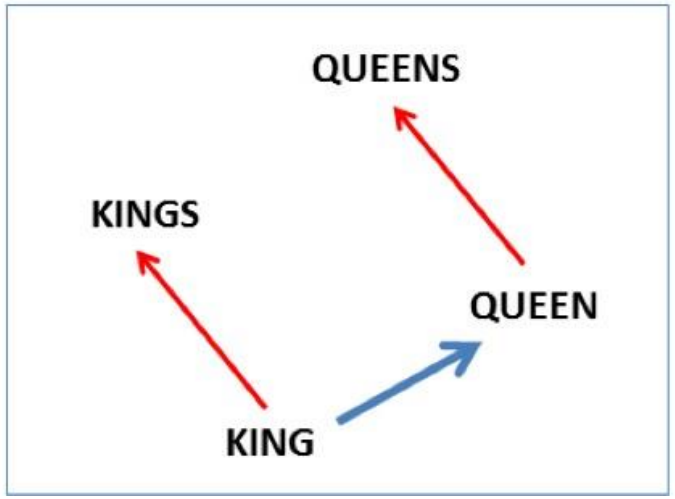
x y

类比评价

vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) ≈ vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) ≈ vector(‘Rome’)





Word2vec的总结

- 两种结构
  - 连续词袋(CBOW)：使用一个窗口中的上下文词预测中间词
  - Skip-gram (SG)：使用一个中心词预测上下文词.

- 两种预测策略
  - 层级化Softmax（HS）：预测词的哈夫曼编码
  - 负采样（NS）：把词预测任务转变为负采样训练样本和二分类任务

Word2vec代码的流程

- 加载数据
- 准备数据结构
- 训练(CBOW或者SG)
  - 如果是 HS
    - 通过层级路径预测
    - 更新参数
  - 如果是 NS
    - 随机选择负样本
    - 预测和更新
- 保存模型

https://code.google.com/archive/p/word2vec/source/default/source