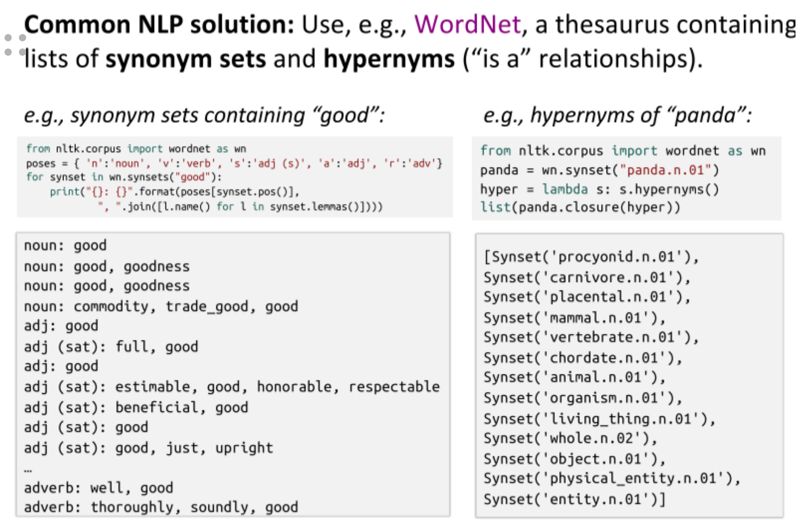
01 Word Vectors笔记

1. 如何表示词

1.1 解决方案如 WordNet

给出能根据一个词列出其如同义词、被包含词



使用 WordNet 得到近义词和被包含词

1.2 WordNet 类方案存在的问题

1. 列出的同义词忽略了细微差别，即比如列出的同义词只在特定上下文成立
2. 缺少新词，即不可能实时更新
3. 主观
4. 需要人工增加和更新
5. 不能给出数值上准确的相似度

1.3 离散符号 one-hot 表示词语存在的问题

1. 正交
2. 不存在天然的相似性衡量

1.4 如何计算相似性

1. 用WordNet的近义词列表来得到相似度，失败了，不可行
2. 用向量来encode词语自身的similarity

1.5 矩阵分解的方式

1.5.1 生成共现矩阵

如何生成共现矩阵

1. 由语料库中所有不重复单词构成矩阵 A 以存储不同单词的共现次数
2. 认为指定context window 大小 n，计算每个单词在指定window 大小与其context中单词的共现次数
3. 依次计算语料库中各单词对的共现次数

生成共现矩阵的例子

1. 假设语料库包含下面 3 句话

```Python
I enjoy flying.
I like NLP.
I like deep learning.
```

2. 构造语料库中所有不重复单词所构成的Dictionary

```Python
Dictionary: [ 'I', 'like', 'enjoy', 'deep', 'learning', 'NLP', 'flying', '.' ]
```

3. 假设上下文窗口context window大小为1，这意味着每个单词的上下问单词都是由左1个单词，右1个单词构成

4. 依次计算语料库中不重复单词构成的矩阵X中各单词与其左右1个单词在上线文窗口中的共现次数

```Python
I = enjoy(1 time), like(2 times)  # I和enjoy共现1次，I和like共现2次
like = I(2 times), NLP(1 time), deep(1 time)
enjoy = I (1 time), flying(1 times)
deep = like(1 time), learning(1 time)
learning = deep(1 time)
NLP = like(1 time)
flying = enjoy(1 time)
. = flying(1 times), NLP(1 time), learning(1 time)
```

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

The resulting counts matrix will then be:

$$X = \begin{array}{c|cccccccc} & I & like & enjoy & deep & learning & NLP & flying & . \\ \hline I & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ like & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ enjoy & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ deep & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ learning & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ NLP & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ flying & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ . & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}$$

共现矩阵

1.5.2 SVD 分解

- 由语料库中单词频次一个所有单词的共现矩阵X
- 对X进行SVD分解得到 $X = USV^T$
- 选择矩阵U的前k列得到k维词向量

这种方式**存在的问题**：

1. 由于语料库的词汇量经常变导致矩阵的维数也会经常变
2. 矩阵稀疏
3. 矩阵维数高
4. n方的训练复杂度(由于SVD)
5. 需要引入一些hacks来解决词频的严重不平衡

> Requires the incorporation of some hacks on X to account for the drastic imbalance in word frequency

解决方法：

1. 忽略一些常见单词，如the
2. 引入移动窗口
   Apply a ramp window – i.e. weight the co-occurrence count based on distance between the words in the document
3. 使用Person系数并且把负的计数置为0而不是采用原始计数(为什么会有负的计数？)

word2vec更为优雅地解决了上面的很多问题

1.5.3 truncate SVD

truncate SVD的策略是：

1. 对于对角矩阵上面的奇异值进行降序排序；
2. 在对角矩阵∑上面取前r个奇异值，相对应的在左右两边的奇异向量矩阵上面也取相对于的r列，最后分别得到了Σ_r，U_r， V_r。
3. 将 $A_r = U_r \Sigma_r V_r^T$ 作为最终矩阵分解的产物。

这样，通过SVD就可以成功的完成矩阵降维的过程。我们从矩阵分解来到了很相近的另外一个主题——降维，作为降维届的代表技术——PCA，而矩阵的奇异值分解能够直接得到矩阵通过PCA的投影空间。对于一个协变量矩阵 X 为 m × p（m为观测）的观测特征矩阵，计算一个 p × l 的矩阵 W 和一个 l × l 的对角矩阵 Λ，能够使得 $X^T X \approx W \Lambda W^T$。这样地近似可以将原先的观测矩阵投影到一个维度为 l 的空间从而达到降维。有一种精简的PCA算法计算的形式是通过直接近似一个低秩的协变量矩阵 $X \approx U_r \Sigma_r V_r^T$，然后乘上转置，最终因为左奇异向量矩阵正交而直接得到投影的向量空间。

$$X^{\mathrm{T}} X \approx (U_{\mathrm{r}}\ \Sigma_{\mathrm{r}}\ V_{\mathrm{r}}{}^{\mathrm{T}})^{\mathrm{T}}\,(U_{\mathrm{r}}\ \Sigma_{\mathrm{r}}\ V_{\mathrm{r}}{}^{\mathrm{T}}) = V_{\mathrm{r}}\ \Sigma_{\mathrm{r}}{}^{2}\ V_{\mathrm{r}}{}^{\mathrm{T}}$$

如上最终的对角矩阵就是新的投影空间。

但是truncate SVD有一些缺点（通过上述和PCA的论述其实可以得到这个缺点是传统降维技术共有的）：

- 实际的工业环境中，矩阵的维度是巨大的，并且数据往往是缺失，不准确的；当不准确的输入限制了输出的精确性时，仅仅依靠这些实际的数据只会白白浪费计算资源。
- 传统的降维技术是不支持并行计算的，如果你熟悉工业数据你就应该知道这一点有多么可怕。

2 word2vec 概述

word2vec是一个learning word vectoe的framework

假设相似的单词具有相似的上下文

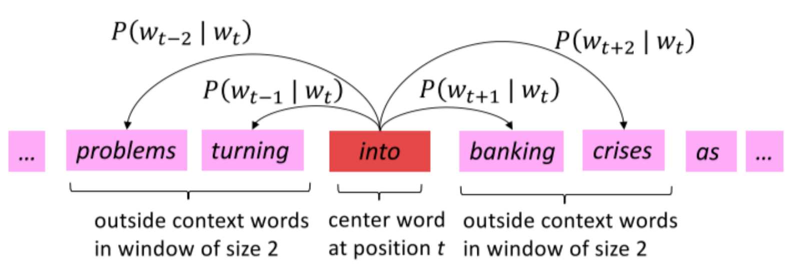
使用一个稠密的向量来表示每个词，在相同上下文出现的单词会表现出相似性。词向量也被称为word embedding或(netural) word representations，是一种分布式表示(distributed representation， 与one hot离散的表示相对应，强调低维、稠密)

2.1 idea

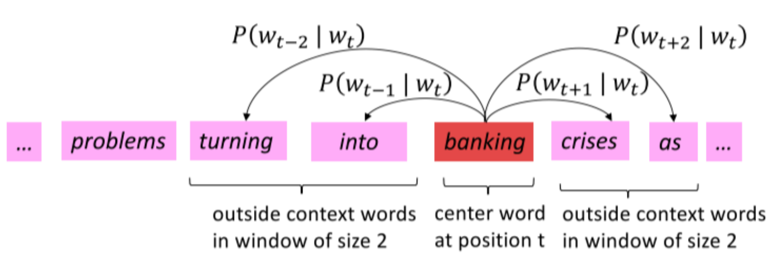
- 有大型的文本语料库
- 在固定词汇表中的每个单词都被表示为一个向量
- go through文本中的每个位置t，t有一个中心词c和上下文(outside)词o
- 使用c和o的word vector的similarity来计算p(o|c)，反之亦然
- 调整word vectors使得概率最大

2.2 计算

计算 $P(w_{t+j}|w_t)$ 的计算窗口和过程示例



计算 $P(w_{t+j}|w_t)$ 图1

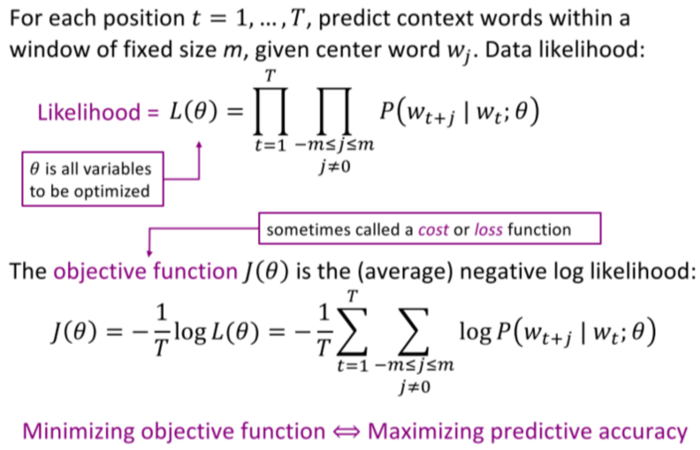


计算 $P(w_{t+j}|w_t)$ 图2

3 word2vec 目标函数

3.1 交叉熵损失

平均指的是对位置取平均，最小化目标函数等价于最大化预测精度，损失函数为 $J(\theta)$



word2vec 目标函数

上述损失函数其实是来自交叉熵损失，接下来首先说明为什么要采用交叉熵损失, 再证明为什么上述损失函数是来自交叉熵损失

为什么要采用 交叉熵损失？

交叉熵损失定义为 $H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log(\hat{y}_j)$ ，由于我们的y是one hot向量(sigmoid后的概率值)，所以可以简化为 $H(\hat{y}, y) = -y_i \log(\hat{y}_i)$ ，预测的很完美即 $\hat{y}$ =1则交叉熵损失为-1log(1) = 0；预测的很糟糕假设 $\hat{y}$ =0.01则交叉熵损失越接近为-1log(0.01)约等于4.605，这个变化趋势很符合我们对损失函数的要求

为什么 $J(\theta)$ 这种**条件概率的形式来自交叉熵损失**?

预测输出即 Sigmoid 函数的输出表征了当前样本标签为 1 的概率：

$$\hat{y} = P(y = 1|x)$$

反之，当前样本标签为 0 的概率就可以表达成：

$$1 - \hat{y} = P(y = 0|x)$$

如果我们从极大似然性的角度出发，把上面两种情况整合到一起:

$$P(y \mid x) = \hat{y}^y \cdot (1 - \hat{y})^{1-y}$$

取log并取负号：

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

推广:

$$L = -\sum_{i=1}^{N} y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

3.2 如何计算 $P(w_{t+j}|w_t; \theta)$

3.2.1 向量表示

对每个单词w使用两个向量：

- v_w 表示w是中心词
- u_w 表示w是上下文词

那么对于中心词c和上下文词o则有预测函数：

$P(o|c) = \frac{exp(u_o{}^T v_c)}{\sum_{w \in V} exp(u_w{}^T v_c)}$

说明：

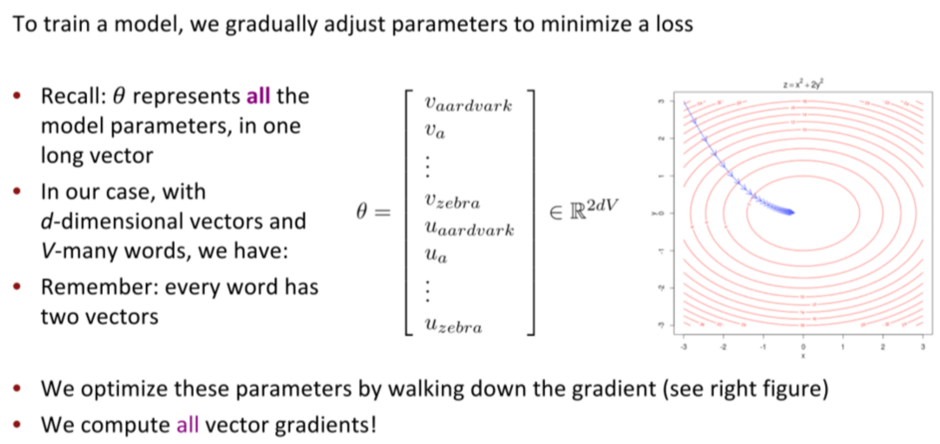
- 为什么 $w \in V$, V是什么?

  > V是corpus中的单词个数

- 整个公式看起来是一个softmax公式，指数来保证概率为正，分子越大相似度越大，分母的作用是归一化，值域是(0,1)的一维实数
- 为什么对一个单词用两个向量u和v来表示?
  因为两个向量表示在数学易于进行优化计算，而且这两个向量是假设相互独立的，不会耦合

3.2.2 参数维数

- d维向量，V个单词
- 每个单词有两个向量，中心和上下文向量，所以参数的维数是2dV

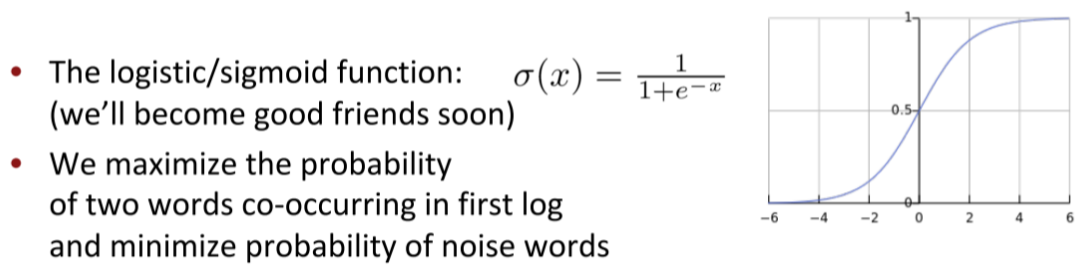


word2vec 参数维数

4 word2vec 梯度下降推导

4.1 sigmoid 函数

定义：



说明: 适用于二分类

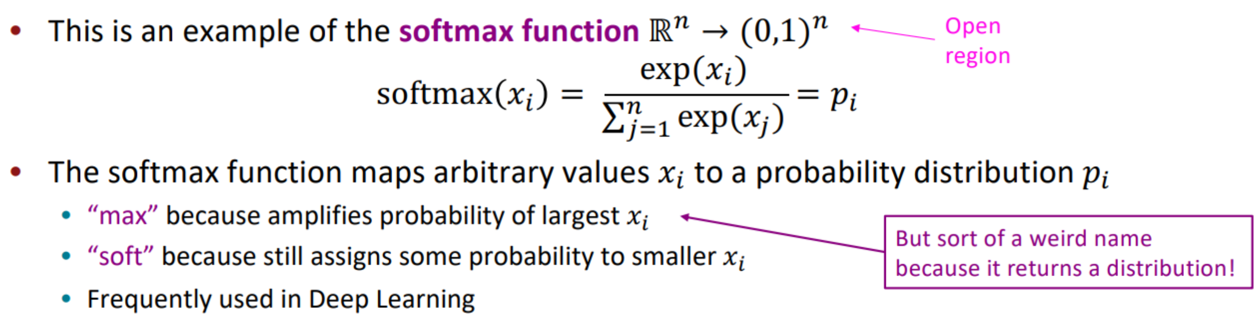
关于sigmoid函数的一个性质：

$$\sigma(-x) = 1 - \sigma(x)$$

4.1 softmax 函数

定义：各个输出节点的输出值范围映射到[0, 1]，并且约束各个输出节点的输出值的和为 1 的函数

说明：适用于多分类，可对条件概率建模，如 $P(o|c) = \frac{exp({u_o}^T v_c)}{\sum_{w=1}^{v} exp({u_w}^T v_c)}$



4.3 求条件概率偏导

已知 $logP(o|c) = log\frac{exp({u_o}^T v_c)}{\sum_{w=1}^{v} exp({u_w}^T v_c)}$

> a 对 b 的偏导数
> partial derivative a respect to b

对Vc求偏导(用到了链式法则)

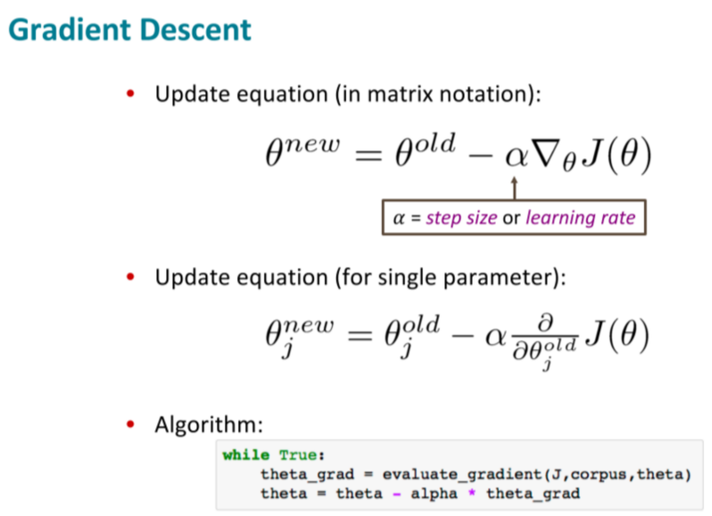
$$
\begin{aligned}
&\frac{\partial log\frac{exp(u_o{}^T v_c)}{\sum_{w=1}^{V} exp(u_w{}^T v_c)}}{\partial v_c} \\
&= \frac{\partial logexp(u_o{}^T v_c)}{\partial v_c} - \frac{\partial log\sum_{w=1}^{V} exp(u_w{}^T v_c)}{\partial v_c} \\
&= u_o - \frac{1}{\sum_{w=1}^{V} exp(u_w{}^T v_c)} \frac{\partial \sum_{x=1}^{V} exp(u_x{}^T v_c)}{\partial v_c} \\
&= u_o - \frac{1}{\sum_{w=1}^{V} exp(u_w{}^T v_c)} \sum_{x=1}^{V} exp(u_x{}^T v_c) u_x \\
&= u_o - \sum_{x=1}^{v} \frac{exp(u_x{}^T v_c) u_x}{\sum_{w=1}^{V} exp(u_w{}^T v_c)} \\
&= u_o - \sum_{x=1}^{v} P(x|c) u_x \\
&= observation - expectation
\end{aligned}
$$

> 矩阵求导
> 矩阵a的转置点乘b后对矩阵b求偏导的结果是矩阵a，因为矩阵a转置点乘矩阵b是X1Y1+X2Y2+...+XnYn,对矩阵b即[Y1,Y2,...Yn]求偏导，即每个位置的元素分别求偏导，最后仍然得到矩阵a

5 优化理论基础

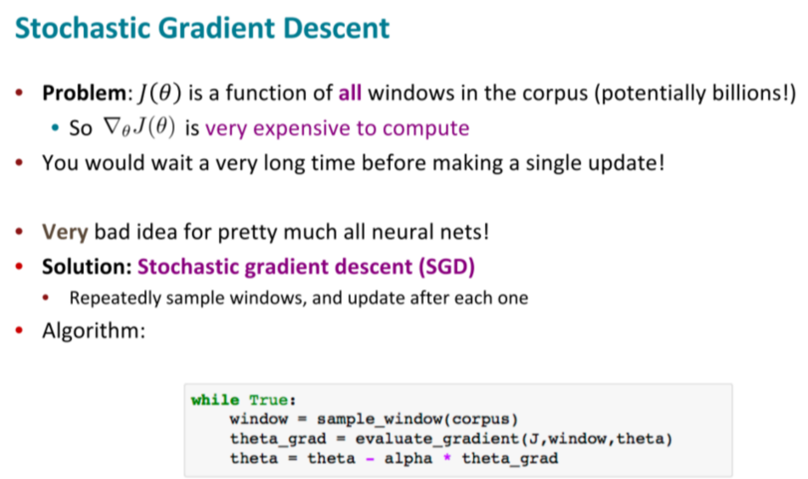
实际中由于语料库中单词量极大，为了考虑计算速度等，一般裁员stochastic gradient descent，即仅计算一个窗口内的梯度

5.1 Gradient Gescent



梯度下降图1

5.2 Stochastic Gradient Descent



梯度下降图2

说明: 神经网络中出现的大多数目标函数都是**非凸函数**，这就使得初始值非常重要，以免陷入局部最优，通常只要选取非常小的随机值就可以解决

6 word2vec 算法细节

6.1 算法流程

6.1.1 CBOW (Continuous Bag of Words)

思想与Skip-gram相反，根据周围词预测中心词

6.1.1.1 算法流程

1. 生成上下文的one-hot向量，大小为m(考虑左右对称，实际为2m个)，记作: ($x^{(c-m)}, \ldots, x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)} \in \mathbb{R}^{|V|}$)
   其中向量维度|V|x1是因为corpus词汇数量为V
2. 得到上下文的embedded word vector: ($v_{c-m} = \mathcal{V}x^{c-m}, v_{c-m+1} = \mathcal{V}x^{c-m+1}, \ldots v_{c+m} = \mathcal{V}x^{c+m} \in \mathbb{R}^n$)
   其中向量维度是nx1表示压缩后的维度，其中 $\mathcal{V}$ 的维度是nx|V|， x^{c-m} 的维度是Vx1
3. 将上一步得到的向量求平均 $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\ldots+v_{c+m}}{2m} \in \mathbb{R}^n$ ，即得到隐藏层向量
4. 生成一个score vector $z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$.由于相似向量的点积更大，所以(训练过程中)为了得到较高的score，相似的向量将会彼此之间更靠近
   向量 z 维度为|V|x1，，其中 $\mathcal{U}$ 的维度是|V|xn， $\hat{v}$ 的维度是nx1
5. 用softmax函数将score转化成概率 $\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$
6. 我们想要让我们生成的概率 $\hat{y} \in \mathbb{R}^{|V|}$ 和真实概率 $y \in \mathbb{R}^{|V|}$ 相吻合，后者刚好是one hot vector of the actual word

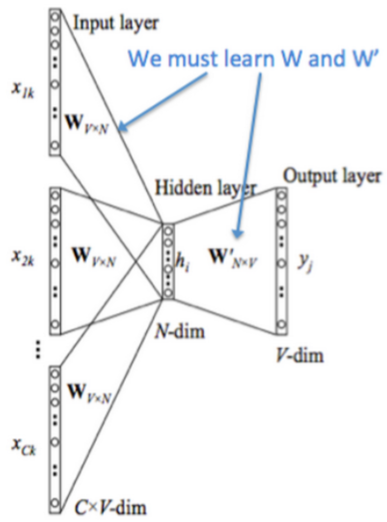


Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

CBOW示意图

说明：图片里的参数矩阵维度相当于是算法流程里转置的，看的时候注意，左边CxV中的C表示C个词

6.1.1.2 损失函数

损失函数:

$$\begin{aligned} \text{minimize } J &= -\log P\left(w_c \mid w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}\right) \\ &= -\log P\left(u_c \mid \hat{v}\right) \\ &= -\log \frac{\exp\left(u_c^T \hat{v}\right)}{\sum_{j=1}^{|V|} \exp\left(u_j^T \hat{v}\right)} \\ &= -u_c^T \hat{v} + \log \sum_{j=1}^{|V|} \exp\left(u_j^T \hat{v}\right) \end{aligned}$$

使用SGD来更新所有相关的word vector u 和 v

6.1.2 Skip-gram

论文：Distributed Representations of Words and Phrases and their Compositionality(Mikolov et al. 2013)

参数：T是需要遍历的窗口数或者时间步， θ 此处指u向量和v向量

损失函数：使第一个log项的单词的共现概率尽可能大，第二项的噪声的概率尽可能小

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta)$$

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)}\left[\log \sigma\left(-u_j^T v_c\right)\right]$$

算法流程:

6.1.2.1 算法流程

1. 生成中心词的one-hot向量 $x \in \mathbb{R}^{|V|}$
   其中向量维度|V|x1是因为corpus词汇数量为V
2. 得到中心词的embedded word vector: $v_c = \mathcal{V}x \in \mathbb{R}^n$
   其中向量维度是nx1表示压缩后的维度，其中 $\mathcal{V}$ 的维度是nx|V|， x 的维度是Vx1

3. 生成一个score vector $z=\mathcal{U}v_c$
   向量 z 维度为|V|x1，，其中 $\mathcal{U}$ 的维度是|V|xn， v_c 的维度是nx1
4. 用softmax函数将score vector转化成概率 $\hat{y}=softmax(z)\in\mathbb{R}^{|V|}$,记为 $\hat{y}^{(c-m)},\ldots,\hat{y}^{(c-1)},\hat{y}^{(c+1)},\ldots,\hat{y}^{(c+m)}$ ，是观测到的(observing)每个上下文的词的概率
5. 我们想要让我们生成的概率向量和真实概率 $y^{(c-m)},\ldots,y^{(c-1)},y^{(c+1)},\ldots,y^{(c+m)}$ 相吻合，实际表示为one hot vectors

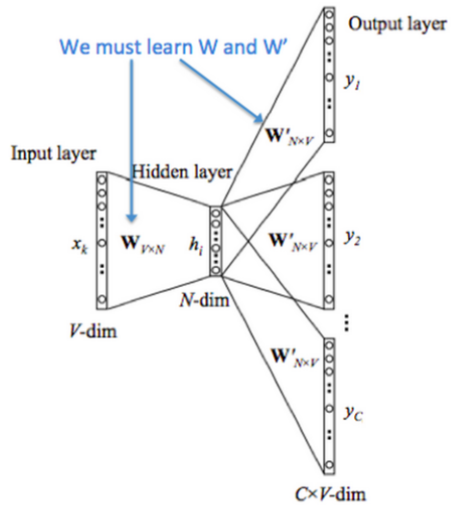


Figure 2: This image demonstrates how Skip-Gram works and how we must learn the transfer matrices

Skip-Gram示意图

说明：图片里的参数矩阵维度相当于是算法流程里转置的，看的时候注意

6.1.2.2 损失函数

和CBOW一样，需要找一个损失函数，一个关键的不同点是我们invoke朴素贝叶斯假设来break out概率，这是一个很strong(naive)的条件独立假设，即，给定中心词，假设假设所有输出词是完全独立的。损失函数：

$$\begin{aligned}
\text{minimize } J &= -\log P\left(w_c-m,\ldots,w_{c-1},w_{c+1},\ldots,w_{c+m}\mid w_c\right)\\
&= -\log\prod_{j=0,j\neq m}^{2m}P\left(w_{c-m+j}\mid w_c\right)\\
&= -\log\prod_{j=0,j\neq m}^{2m}P\left(u_{c-m+j}\mid v_c\right)\\
&= -log\prod_{j=0,j\neq m}^{2m}\frac{exp(u_{c-m+j}^Tv_c)}{\sum_{k=1}^{|V|}exp(u_k^Tv_c)}\\
&= -\sum_{j=0,j\neq m}^{2m}u_{c-m+j}^Tv_c+2m\log\sum_{k=1}^{|V|}exp(u_k^Tv_c)
\end{aligned}$$

损失函数其实是交叉熵损失：

$$\begin{aligned}
J &= -\sum_{j=0,j\neq m}^{2m}\log P\left(u_{c-m+j}\mid v_c\right)\\
&= \sum_{j=0,j\neq m}^{2m}H\left(\hat{y},y_{c-m+j}\right)
\end{aligned}$$

6.2 负采样(negetivate sampling)

CBOW和Skip-Gram的损失函数J的计算代价很高，因为softmax的归一化项是在对所有的|V| scores求和，一个简单的想法就是我们估计来取代它。

对每一步训练来说，我们不再在整个vocabulary上循环，而是只对一些负样本进行采样， We "sample" from a noise distribution ($P_n(w)$) whose probabilities match the ordering of the frequency of the vocabulary.接下来需要定义负采样的目标函数、梯度、更新参数的方式。

尽管负采样是基于Skip-Gram模型，但是它优化的目标函数和Skip-Gram模型不同。将词和它的上下文的pair记作 (w,c) , 这个pair来自训练数据吗？将 $P(D=1|w,c)$ 记作 (w,c) 来自corpus data的概率，相应的，将 $P(D=0\mid w,c)$ 记作 (w,c) 不来自corpus data的概率。首先，用sigmoid函数对 $P(D=1|w,c)$ 来建模：

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

接着，构建一个新的目标函数，作用是 (w,c) 在corpus data的条件下，最大化 (w,c) 在corpus data的概率，且 (w,c) 不在corpus data的条件下，最大化 (w,c) 不在corpus data的概率

> we build a new objective function that tries to maximize the probability of a word and context being in the corpus data if it indeed is, and maximize the probability of a word and context not
> being in the corpus data if it indeed is not

从最大似然的角度看这两种概率(θ 是模型参数，在我们的场景下是 $\mathcal{V}$ 和 $\mathcal{U}$):

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

最大似然和最小化负的log损失是一样的：

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

其中 $\tilde{D}$ 是 "false" or "negative" corpus里面可能包含"stock boil fish is toy"这样的句子，不自然的句子的出现概率应该很低，我们可以从词库中随机抽取来动态生成 $\tilde{D}$

> We can generate D˜ on the fly by randomly **sampling this negative** from the word bank

对skip-gram模型，我们新的(负采样)关于观测到的上下文单词 c −m+ j 对(given)中心词c的目标函数是：

$$-\log\sigma\left(u_{c-m+j}^T\cdot v_c\right)-\sum_{k=1}^{K}\log\sigma\left(-\tilde{u}_k^T\cdot v_c\right)$$

作为对比，普通的skip-gram的softmax损失函数是：

$$-u_{c-m+j}^T v_c+\log\sum_{k=1}^{|V|}\exp\left(u_k^T v_c\right)$$

对CBOW模型，我们新的(负采样)关于中心词 u_c 对(given)观测到的上下文单词 $\hat{v}=\frac{v_{c-m}+v_{c-m+1}+\ldots+v_{c+m}}{2m}$ 的目标函数是：

$$-\log\sigma\left(u_c^T\cdot\hat{v}\right)-\sum_{k=1}^{K}\log\sigma\left(-\tilde{u}_k^T\cdot\hat{v}\right)$$

作为对比，普通的CBOW的softmax损失函数是：

$$-u_c^T\hat{v}+\log\sum_{j=1}^{|V|}\exp\left(u_j^T\hat{v}\right)$$

在上面的公式中， $\{\tilde{u}_k\mid k=1\ldots K\}$ 是从 $P_n(w)$ 中采样的，那么 $P_n(w)$ 是什么呢？虽然有很多关于什么是最佳近似的讨论，但似乎效果最好的是指数为3/4的Unigram Model. 为什么是3/4呢？下面是一些可以帮助从直觉上理解的例子：

$$\begin{gathered}
is: 0.9^{3/4}=0.92 \\
Constitution: 0.09^{3/4}=0.16 \\
bombastic: 0.01^{3/4}=0.032
\end{gathered}$$

"Bombastic"(不常见词)现在被抽样的可能性增加了3倍，而"is"(常见词)只略微上升

6.3 层次化 softmax

6.3.1 Huffman Tree

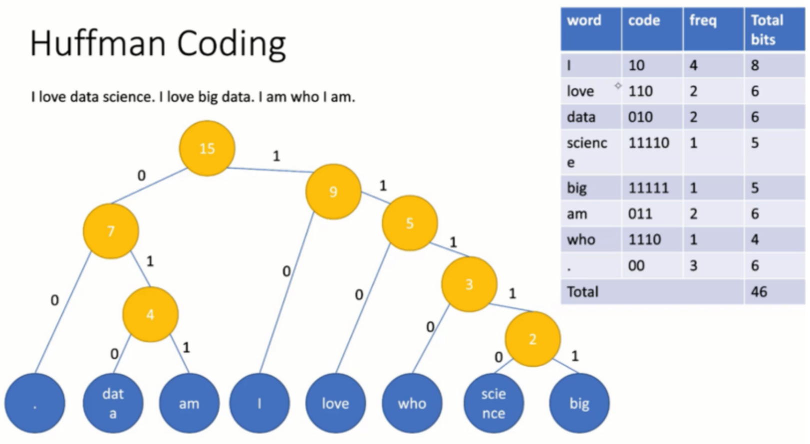
Huffman Tree是一种带权路径长度最短的二叉树，也称为**最优二叉树。**

6.3.1.2 Huffman Tree和普通二叉树的对比

左子树为0，右子树为1

code Tree

Huffman Tree



6.3.2 层次化 softmax

skip-gram论文中也提到将hierarchical softmax作为提升softmax归一化效率的选项，在实际中，hierarchical softmax倾向于在低频单词效果好，而负采样对高频单词和低维word vectors表现更好

层次化softmax用于**取代output layer(softmax classifier)**，因为原始的output layer中神经元的个数和corpus大小是一样的，即如果corpus很大，这个结构也是很大的

hierarchical softmax用二叉树来表示语料库中的所有词，数的每个叶子表示一个词，并且从根节点到叶子的路径是唯一的。在这个模型中，没有对单词的输出表示，而是图中的每个节点 (除了根节点和叶子节点) 都关联一个模型待学习的向量

在该模型中，单词 w 对向量 w_i 的的概率 $P(w|w_i)$ 等于与 w 对应从根开始到的叶节点结束的随机游走概率。这种方法计算概率的主要优点是相对于路径长度, 复杂度为 $O(log(|V|))$，对应于路径的长度

下面将用到的一些符号：

- $L(w)$ 表示从根节点到叶子节点 w 的路径上的所有节点的数量。比如下图中的 $L(w_2)$ 为4
- $n(w,i)$ 表示从根节点到叶子节点 w 这条路径上的第i个节点，与向量 $v_{n(w,i)}$ 关联。所以，$n(w,1)$ 是根节点， $n(w,L(w))$ 是w的父节点
- v_{w_i} 是hidden layer vector

对每个下图中灰色的内部节点n，我们任意选择一个它的子节点并将其称之为 $ch(n)$, 比如我们总选择左节点，计算得到的概率为：

$$P\left(w \mid w_i\right)=\prod_{j=1}^{L(w)-1} \sigma\left([n(w, j+1)=\operatorname{ch}(n(w, j))] \cdot v_{n(w, j)}^T v_{w_i}\right)$$

其中:

$$[x]=\left\{\begin{array}{l}1 \text { if } x \text { is true } \\ -1 \text { otherwise }\end{array}\right.$$

且 $\sigma(.)$ 表示sigmoid函数

这个公式该怎么理解呢?

首先，我们基于根节点($n(w,1)$)到叶子节点(w)的形状来计算乘积。如果我们假设 $ch(n)$ 总是 n 的左节点，那么式子 $[n(w,j+1)=ch(n(w,j))]$ 将会在路径往左史返回1，路径往右时返回-1.

再者，式子 $[n(w,j+1)=ch(n(w,j))]$ 起到了正则化的功能。在节点n，往左走的概率 $p(n,left)=\sigma\left(v_n^T v_{w_i}\right)$, 同理，往右走的概率 $p(n,right)=1-\sigma\left(v_n^T v_{w_i}\right)=\sigma\left(-v_n^T v_{w_i}\right)$ 如果我们对前往左右节点的概率求和，对于任何 $v_n^T v_{w_i}$, 有 $\sigma(v_n^T v_{w_i})+\sigma(-v_n^T v_{w_i})=1$

最后，对比输入向量 v_{w_i} 和每一个下图中灰色的内部节点向量 $v_{n(w,j)}^T$ 的点积的相似度.比如下图中的 w_2 , 从根节点出发，要向左走两步在向右走一步才能到达 w_2 ，所以有：,tongli

$$\begin{aligned} P\left(w_2 \mid w_i\right) &= p\left(n\left(w_2,1\right), \text{ left }\right)\cdot p\left(n\left(w_2,2\right), \text{ left }\right)\cdot p\left(n\left(w_2,3\right), \text{ right }\right) \\ &= \sigma\left(v_{n\left(w_2,1\right)}^T v_{w_i}\right)\cdot \sigma\left(v_{n\left(w_2,2\right)}^T v_{w_i}\right)\cdot \sigma\left(-v_{n\left(w_2,3\right)}^T v_{w_i}\right) \end{aligned}$$

为了训练模型，目标仍然是最小化负的对数似然 $-logP(w|w_i)$, 但是，我们不更新每个词的输出向量，而是更新二叉树中从根节点到叶节点的路径中的节点向量

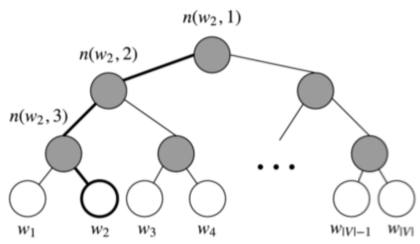


Figure 4: Binary tree for Hierarchical softmax

层次化softmax

7 参考

1. cs224n winter2021 notes
2. https://zhuanlan.zhihu.com/p/56139075