词向量

- 自然语言理解的问题要转化为机器学习的问题，第一步肯定是要找一种方法把这些符号数学化，向量就是一种很好的形式。
- 对于NLP任务，首先要将这些向量表示成稠密、低维的实值向量,这是我们处理NLP任务的基础。

One-hot

NLP 中最直观，也是到目前为止最基础的词表示方法是 One-hot Representation，这种方法把每个词表示为一个很长的向量。这个向量的维度是词表大小，其中绝大多数元素为 0，只有一个维度的值为 1，这个维度就代表了当前的词。
举个栗子，
“话筒”表示为 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 …]T
“麦克”表示为 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 …]T
每个词都是茫茫 0 海中的一个 1。

问题：
1.“词汇鸿沟”现象：任意两个词之间都是孤立的。光从这两个向量中看不出两个词是否有关系。

2.维数高

3.矩阵稀疏

词-文档共现矩阵

syntagmatic models关注的是词与词的组合关系（combinatorial relations），强调的是有组合关系词会共现于同一个语境（text region），比如说同一个句子。举个非常简单的例子，现在有三篇文档——doc1: I love playing football. doc2: I love playing tennis. doc3: You love playing football. 那么现在可以建立一个词-文档共现矩阵，元素值代表词频。

	doc1	*doc2*	*doc3*
I	1	1	0
love	1	1	1
playing	1	1	1
football	1	0	1
tennis	0	1	0
you	0	0	1

Love，playing均同时出现，认为有较强的组合关系。Football与tennis均不同时出现，认为有较强的替换关系。

词-词共现矩阵

paradigmatic models关注的是词与词的替换关系（substitutional relations），强调的是具有替换关系的词拥有相似的上下文（context）而可以不同时出现。
窗口：所取的窗口大小为1，考虑的是中心词左边和右边各1个词。

	I	*love*	*playing*	*football*	*tennis*	*you*
I	0	2	0	0	0	0
love	2	0	3	0	0	1
playing	0	3	0	2	1	0
football	0	0	2	0	0	0
tennis	0	0	1	0	0	0
you	0	1	0	0	0	0

问题：
1.规模随着语料库词汇的增加而增加
2.非常高的维度：需要大量的存储
3.分类模型会遇到稀疏问题

方法：降维

svd分解

咱们一起来看个例子，假定我们有如下的3个句子，同时我们的窗口大小设定为1（把原始的句子分拆成一个一个的词）：

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

由此产生的计数矩阵如下：

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

Svd分解

然后我们对X做奇异值分解，观察观察奇异值（矩阵的对角元素），并根据我们期待保留的百分比来进行截断（只保留前k个维度）

$$\frac{\sum_{i=1}^{k}\sigma_i}{\sum_{i=1}^{|V|}\sigma_i}$$

然后我们把子矩阵U1:|V|,1:k视作我们的词嵌入矩阵。也就是说，对于词表中的每一个词，我们都用一个k维的向量来表达了。

$$\underset{|V|}{\begin{bmatrix} \\ \hat{X} \\ \\ \end{bmatrix}} = \underset{|V|}{\begin{bmatrix} | & | & \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix}} \underset{k}{\begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}} \underset{|V|}{\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix}}$$

http://www.cnblogs.com/LeftNotEasy/archive/2011/01/19/svd-and-applications.html

Svd分解

改进：
1.功能词(the, he, has)过于频繁，对语法有很大影响，解决办法是降低使用或完全忽略功能词
2.应用ramp window
3.用皮尔逊相关性（Pearson correlation）

缺陷：
1. 时间复杂度高
2. 对于新词或者新的文档很难及时更新
3. 相对于其他的DL模型，有着不同的学习框架

Word2vec

2013年，Google开源了一款用于词向量计算的工具——word2vec，word2vec是一个计算word vector的开源工具。当我们在说word2vec算法或模型的时候，其实指的是其背后用于计算word vector的CBoW模型和Skip-gram模型。当然还有其他更加复杂的原理。暂且不加讨论。
Nnlm是cbow和skip-gram的基础，我们先介绍nnlm。
Nnlm基于n-gram模型，假设第t个词的出现只与前面N-1个词相关，而与其它任何词都不相关。

$$P(\omega_t \mid \omega_1 \ldots \omega_{t-1}) \approx P(\omega_t \mid \omega_{t-(n-1)} \ldots \omega_{t-1})$$

Nnlm将语句中的一个词串的 $\omega_{t-(n-1)}^{t}$ 的目标词 ω_t 之前的n-1个词的词向量（由one-hot经矩阵C处理而来，设有m维）按首尾拼接得到一个长的列向量X，所谓输入层（共有（n-1）m个神经元）

NNLM

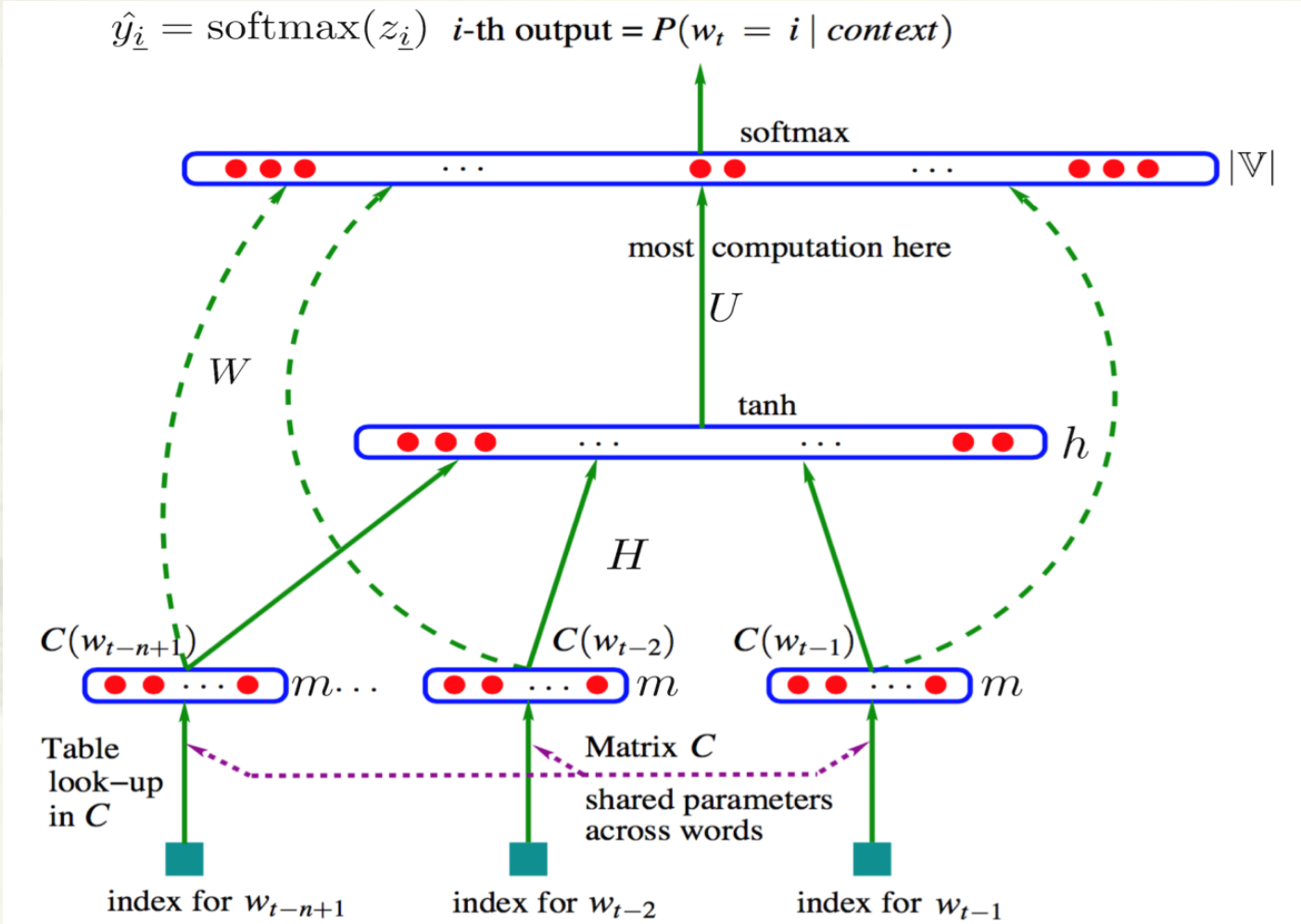
X经过权重矩阵 $H_{h\times m(n-1)}$ 来到隐含层（神经元数为h），利用激活函数激活后，在经过权重矩阵 $U_{|V|\times h}$ 来到输出层。神经元个数为|V|，再使用softmax函数将其归一化为概率。另外还存在一个从输入层直连输出层的一个权重矩阵 $W_{|V|\times(n-1)m}$，所以网络的输出如下，（隐含层与输出层增加了偏置）。

$$Z = U\tanh(HX + d) + b + WX$$

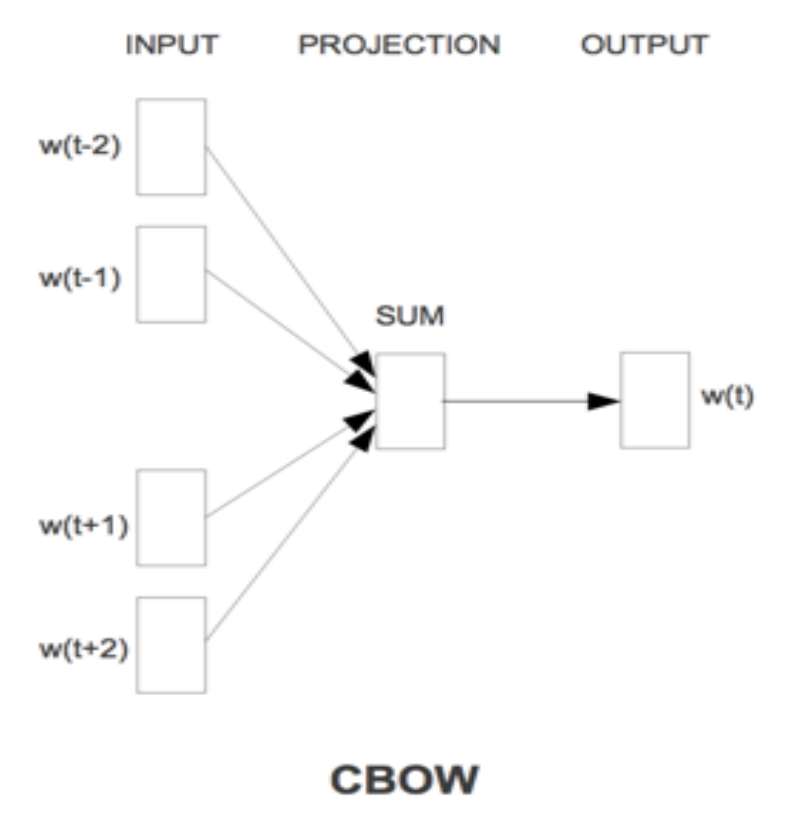
$$\hat{y}_i = P(\omega_i \mid \omega_{t-(n-1)}...\omega_{t-1}) = soft\max(z_i)$$

$$= \frac{e^{z_i}}{\sum_{k=1}^{|V|} e^{z_k}}$$

NNLM

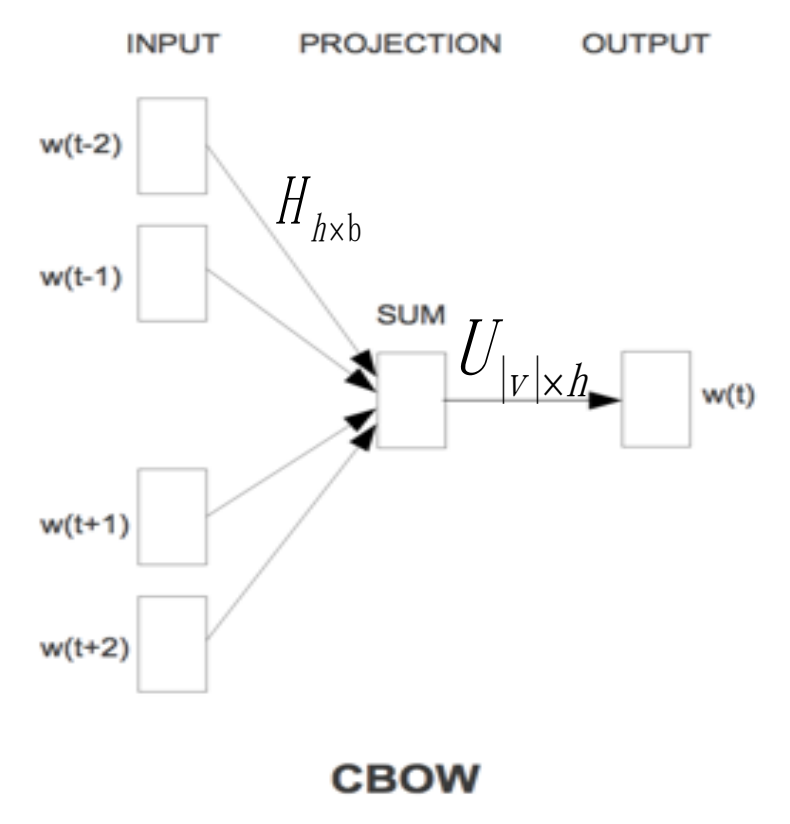

$\hat{y}_i = \text{softmax}(z_i)$ i-th output = $P(w_t = i \mid context)$
softmax
$|\mathbb{V}|$
most computation here
U
W
tanh
h
H
$C(w_{t-n+1})$
$C(w_{t-2})$
$C(w_{t-1})$
m
m
m
Table look-up in C
Matrix C
shared parameters across words
index for w_{t-n+1}
index for w_{t-2}
index for w_{t-1}

CBOW



1.移除前向神经网络中非线性的hidden layer，直接将中间层的embedding layer与输出层的softmax layer连接；
2.忽略上下文环境的序列信息：输入的所有词向量均汇总到同一个embedding layer，取均值；
3.预测中间词，将future words纳入上下文环境

CBOW



$$H_{h\times b}\omega_i = V_i$$

$$\hat{V} = \frac{1}{2m}\sum V_i$$

$$Z = U\hat{V}$$

$$\hat{y}_i = P(\omega_i \mid \omega_{t-m}...\omega_{t-1}, \omega_{t+1}...\omega_{t+m}) = soft\max(z_i)$$

$$= \frac{e^{z_i}}{\sum_{k=1}^{|V|} e^{z_k}}$$

CBOW

我们需要有一个目标函数。通常来说，当我们试图从已知概率学习一个新的概率时，最常见的是从信息论的角度寻找方法来评估两个概率分布的差距。其中广受好评又广泛应用的一个评估差异/损失的函数是交叉熵：

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

结合我们当下的例子，y只是一个one-hot向量，于是上面的损失函数就可以简化为：

$$L(\hat{y}, y) = -y_c \log(\hat{y}_c)$$

我们用c表示y这个one-hot向量取值为1的那个维度的下标。所以在我们预测为准确值的情况下y^c=1。于是损失为 −1 log(1) = 0。所以对于一个理想的预测值，因为预测得到的概率分布和真实概率分布完全一样，因此损失为0。现在让我们看一个相反的情况，也就是我们的预测结果非常不理想，此时y^c=0.01。计算得到的损失为−1 log(0.01) ≈ 4.605，损失非常大，原本这才是标准结果，可是你给了一个非常低的概率，因此会拿到一个非常大的loss 。可见交叉熵为我们提供了一个很好的衡量两个概率分布的差异的方法。于是我们最终的优化函数为：

CBOW

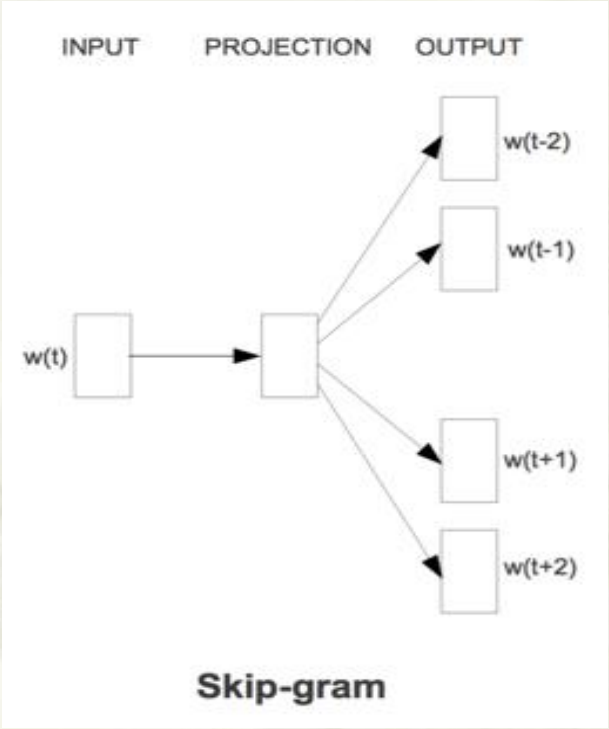
$$
\begin{aligned}
\text{minimize } J &= -\log P(w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}) \\
&= -\log P(u_c | \hat{v}) \\
&= -\log \frac{\exp(u_c^T \hat{v})}{\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})} \\
&= -u_c^T \hat{v} + \log \sum_{j=1}^{|V|} \exp(u_j^T \hat{v})
\end{aligned}
$$

u_c表示矩阵U第c行的列向量

CBOW

在已有目标函数的条件下，我们可以对整个模型运用梯度下降法进行更新，更新的参数为H，U，随着整个模型的迭代运行，vi也在不断更新，vi的维度相比原来的one-hot要小得多，所以最终我们也就得到了每个词向量。

Skip-gram



$$\text{minimize } J = -\log P(w_{c-m},\ldots,w_{c-1},w_{c+1},\ldots,w_{c+m}|w_c)$$

$$= -\log \prod_{j=0,j\neq m}^{2m} P(w_{c-m+j}|w_c)$$

$$= -\log \prod_{j=0,j\neq m}^{2m} P(u_{c-m+j}|v_c)$$

$$= -\log \prod_{j=0,j\neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|}\exp(u_k^T v_c)}$$

$$= -\sum_{j=0,j\neq m}^{2m} u_{c-m+j}^T v_c + 2m\log\sum_{k=1}^{|V|}\exp(u_k^T v_c)$$