Word2vec是什么

- 我们通常是。

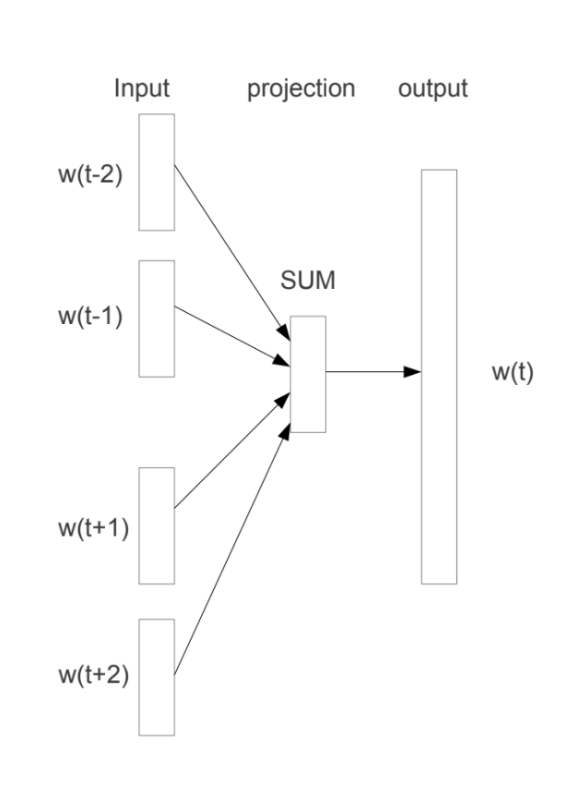
Word2vec是什么

- 是一种可以用于训练词向量的模型工具。
- 工具中包含两个模型：CBOW和Skip-gram，和两个近似训练的方法：Hierarchical Softmax和Negative Sampling。
- 常见的做法是，我们先用word2vec在公开数据集上预训练词向量，加载到自己的模型中，对词向量进行调整，调整成适合自己数据集的词向量。

Word2vec现实基础

- Word2vec很符合人类的认知行为：当遇到不认识词汇，我们会通过上下词，或上下字来推测出未知词汇的大致意思。例如：我吃的【蛤蜊】很美味。
- CBOW思想：其中“蛤蜊”为未知词汇，但是上下文中有“吃”“美味”，那么很容易可以推测出未知词汇是“食物”“美食”。
- Skip-gram思想：“蛤蜊”在词“吃”的附近，“蛤蜊”在词“美味”的附近，那么也容易推测出该未知词汇和“食物”相关。

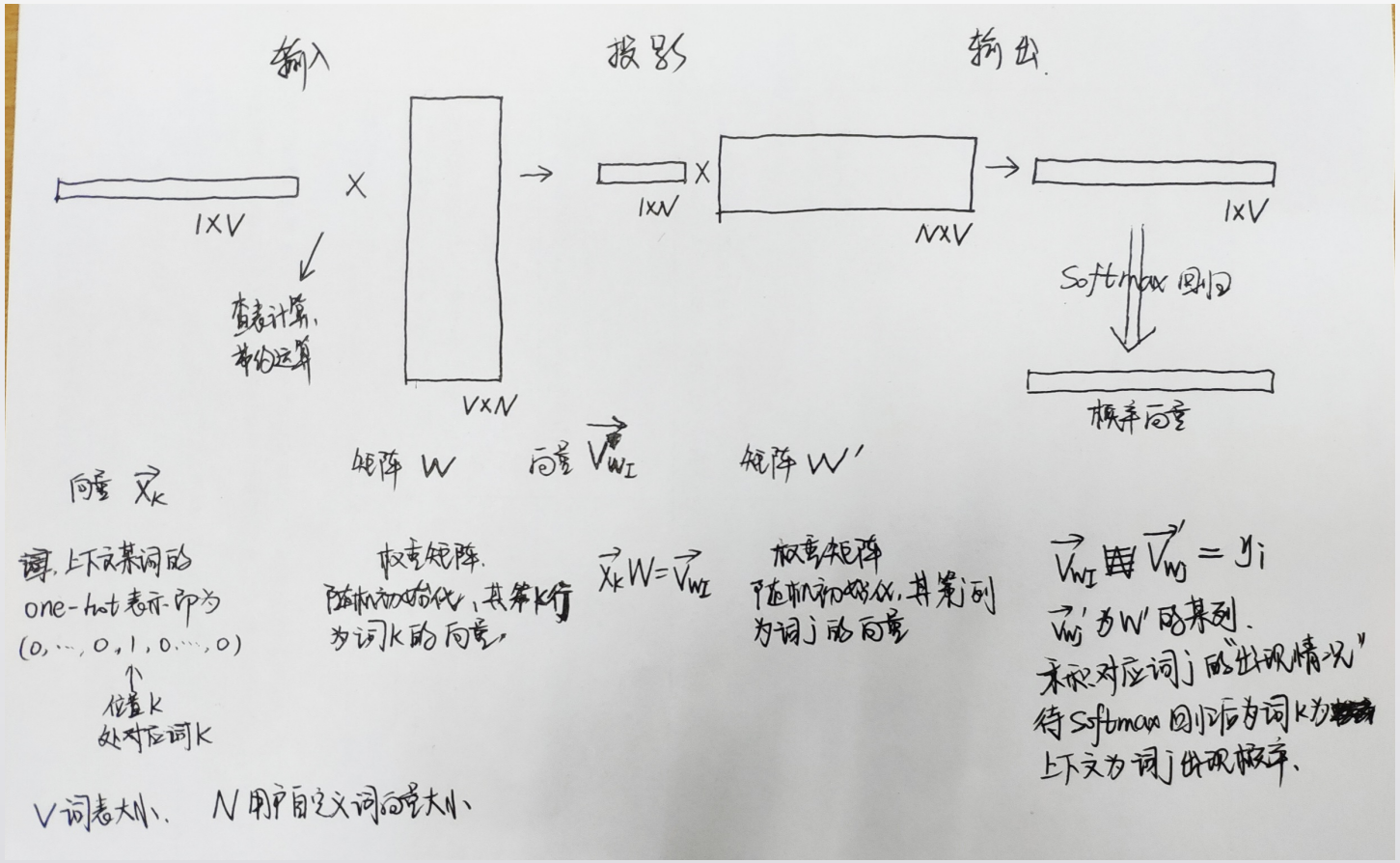
CBOW模型



- CBOW（Continuous Bag of Words）模型通过上下文预测中心词。
- 其中$w(t)$代表中心词的词向量。
- 输入层是背景词的one-hot向量。
- 投影层是将one-hot向量累加。
- 输出层经过Softmax输出一个概率向量，代表了每个在这些背景词下，每个词出现的概率大小。
- 图来自[1]。

Reference：[1]Learning Representations of Text using Neural Networks

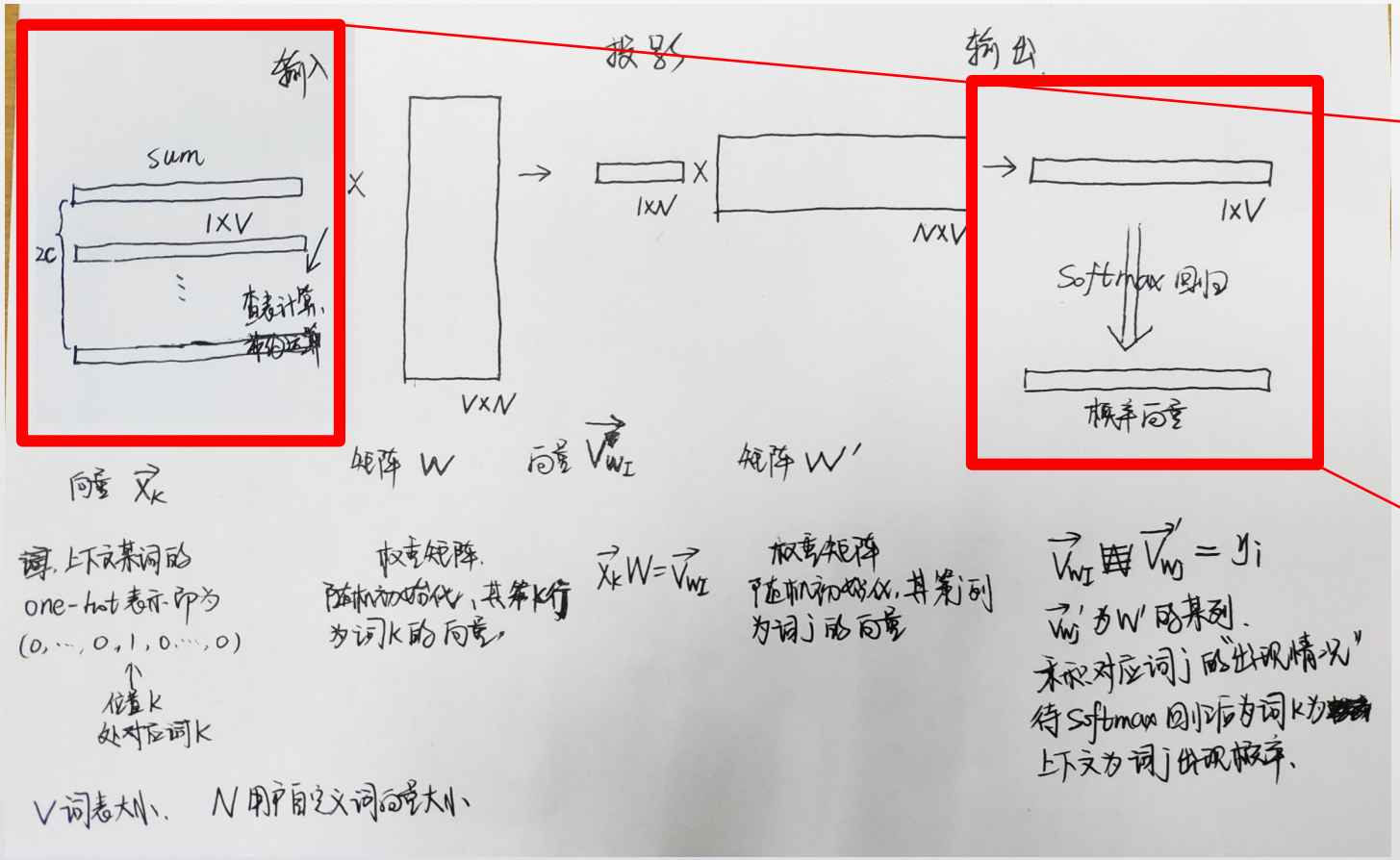
CBOW模型



- 左图表示了上下文中的一个词输入到CBOW模型后的计算过程。

Reference： [1]Learning Representations of Text using Neural Networks

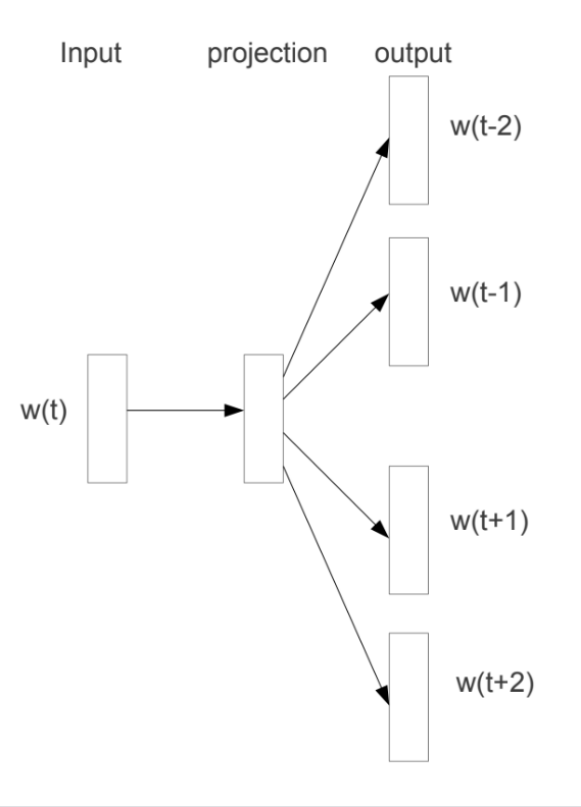
CBOW模型



- 左图表示了上下文中的$2c$个词，即窗口$window = c$，输入到CBOW模型后的计算过程。
- 红框内，计算Softmax与Skip-gram一样，会消耗大量时间资源。

Reference：[1]Learning Representations of Text using Neural Networks

Skip-gram模型



- Skip-gram模型通过中心词预测背景词。
- 其中$w(t)$代表中心词的词向量。
- 输入层是one-hot向量。
- 映射层是乘以权重矩阵后，得到中心词词向量。
- 输出层是乘以权重矩阵后，得到向量，通过Softmax函数，向量中每个值规范到[0,1]之间。
- 输出层使图来自[1]。

Reference： [1]Learning Representations of Text using Neural Networks

Skip-gram模型

- 目标函数：

maximiz $\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\neq 0}\log p(w_{t+j}|w_t)$

- 其中T是词典中词的总数；
- 背景词（被预测的词可能出现的位置）共有$2c$个，平均分布在中心词两侧；
- ω_t代表了中心词，ω_{t+j}代表了中心词某侧的第j个背景词；
- 取log是为了将微小变化放大，同时方便求梯度，因为对加法求偏导要比对乘法求偏导方便。

Reference：[1]Distributed Representations of Words and Phrases

Skip-gram模型

- 其中$p(\omega_{t+j}|\omega_t)$用如下Softmax函数：

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^{\top} v_{w_I}\right)}$$

- 其中v_ω、 v'_ω分别代表输入词、输出词的词向量；W代表了整个词表的词的数目。
- 从这一步可以看出，计算分母的计算量非常大，并且计算量与词表数目W成正比例。
- 以本PPT中演示的中文词数目为例，大约$0.55E9$有效词，除去重复词大约有$4E6$个词。

Reference： [1]Distributed Representations of Words and Phrases

Skip-gram模型

- Softmax中的求和计算量与词表数目W成正比例。
- 以本PPT中演示的中文词数目为例，

```
7678 2019-07-21 13:55:35,992:INFO:training on a 622244075 raw words (
     549932982 effective words) took 8037.5s, 68421 effective words/s
```

大约5.5亿有效词，除去重复词有4054594个词，计算这个加法会是十分消耗资源。

```
1  from gensim.models import word2vec
2  model = word2vec.Word2Vec.load('wiki_corpus_250_sg1_hs0.model')
3  print(model.corpus_count)
```

```
E:\Python\python.exe E:/Wiki/wiki/extracted1/AA/word2vec/800M/test_sim.py
4054594

Process finished with exit code 0
```

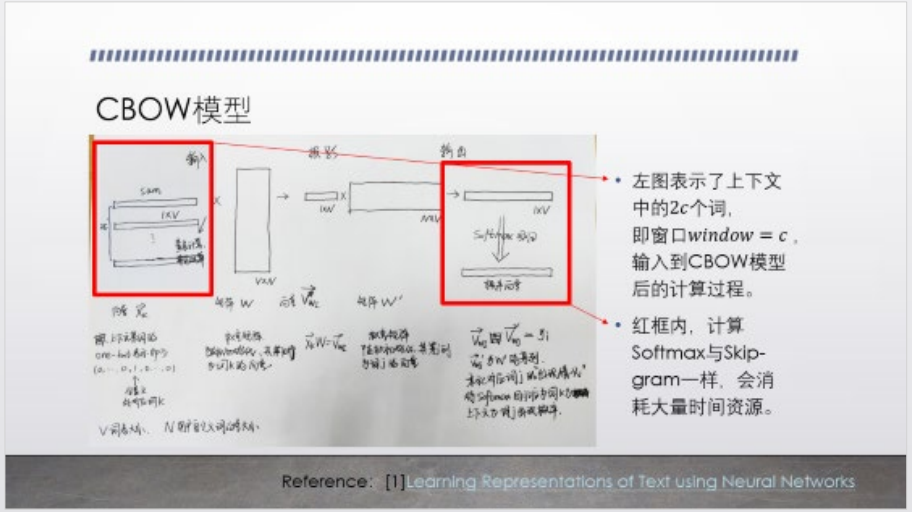
BP算法及随机梯度下降

- 思想：在上述两种方法中，可以看到每种方法都会产生两个权重矩阵，W和W'，其初值随机产生；观察输出值和真实值之间的error（误差）并计算其梯度；再在梯度的方向纠正误差矩阵的值；重新进入下一轮正向传播，也就是前面提到的传播过程。至此反向传播的过程描述完毕。
- 接下来以CBOW为例，进行BP算法下随机梯度下降推演：

Reference： [1]BP算法浅谈-2011-小例子[2]通俗理解BP[3]

BP算法及随机梯度下降

- CBOW模型（点击图片可跳转）。



- 随机的意思是根据中心词的one-hot来调整，显然这种调整可能会使得算法陷入局部最优解。
- 损失函数为：
- $E = -\log\left(p(\omega_o|\omega_I)\right)$

  $= -\boldsymbol{v}_{\omega_O}^T \cdot \boldsymbol{h} - log\sum_{j'=1}^{V} \exp(\boldsymbol{v}_{\omega_O}^T \cdot \boldsymbol{h})$
- 对其求导后有概率矩阵W, W'更新规则为：
- $\omega'^{(new)} = \omega'^{(old)}_{ij} - \eta \cdot (y_j - t_j) \cdot h_i$
- $\omega^{(new)} = \omega^{(old)}_{ij} - \eta \cdot \frac{1}{C} \cdot EH$
- 其中，η为学习率，由用户自定义，越大，则越快。

Reference：[1]BP算法浅谈-2011-小例子[2]通俗理解BP[3]

近似计算方法

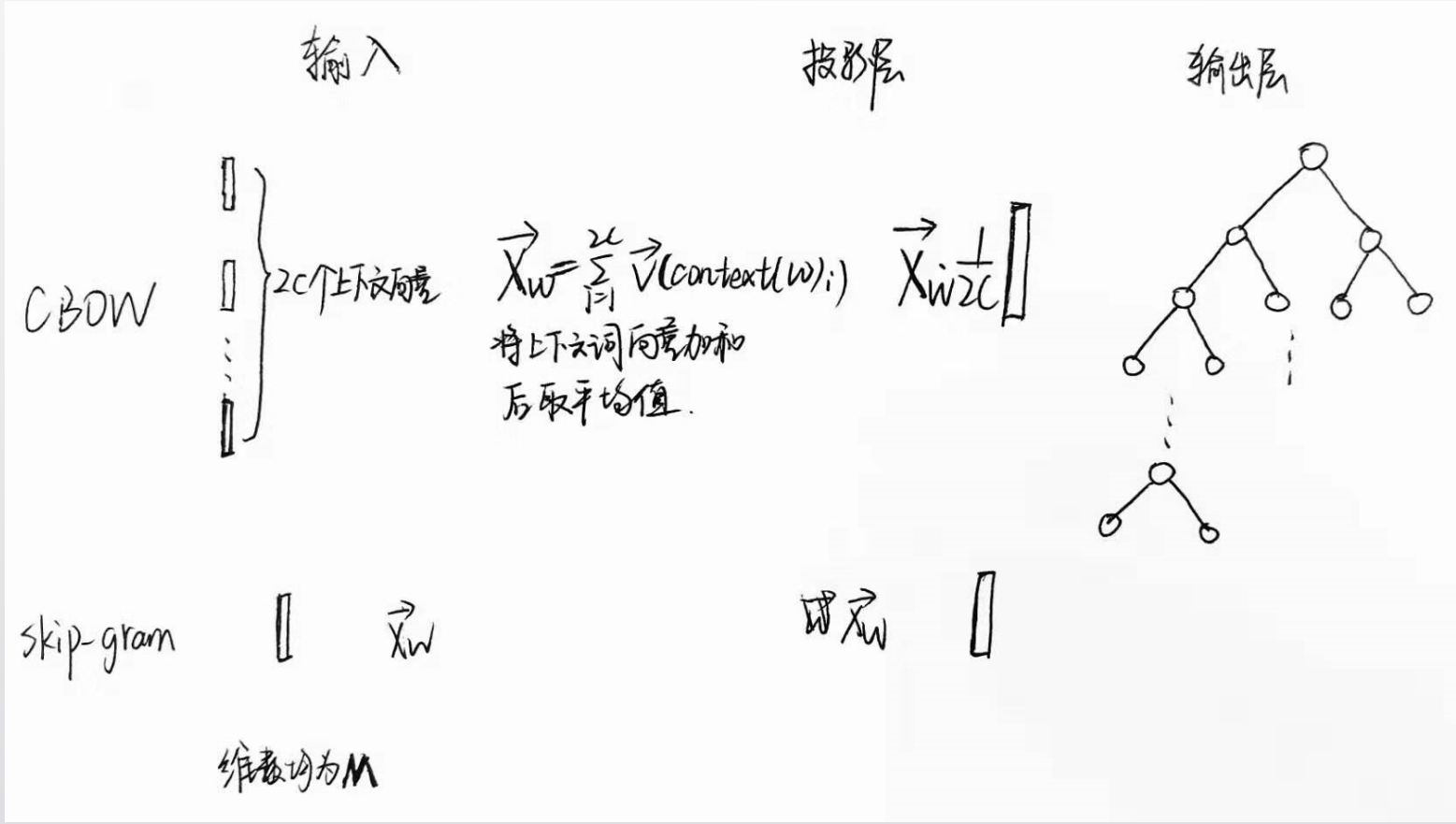
- 在Skip-gram和CBOW中，都要通过最后的Softmax进行归一化操作，来得到概率向量，并且方便之后的梯度求导。这已经可以作为一个简单的模型，而且这是word2vec的基础，这种模型也可以看作是神经概率语言模型[1]的简化。
- 由于在投影层到输出层的矩阵向量运算较大，以及输出层的Softmax归一化运算庞大，之后便有了word2vec工作中的两种近似方法：
  - Hierarchical Softmax 层次Softmax函数法（简记hs）
  - Negative Sampling 负采样方法（简记neg）

Reference： [1] 神经概率语言模型原论文

Hierarchical Softmax

- hs将权重矩阵W'以及Softmax函数取代，其中最主要的是取代了Softmax函数，从而使得模型的计算开销大大降低。
- hs的理论基础，是sigmoid函数，逻辑回归，Huffman树，梯度下降法。
- Hs虽然有Softmax函数，但就个人理解，其中其实没有用到Softmax函数，只不过通过Huffman树构造了新的向量编码，向量编码更短（相对于one-hot的长），在计算，迭代的时候效率更高，取代了W'和Softmax归一化的作用。

基于hs的CBOW模型和Skip-gram模型



- Skip-gram的输入层到投影层为恒等映射。

Reference: [1] 刘建平，基于hs的模型

基于hs的CBOW模型

- 接下来，计算$\boldsymbol{x}_\omega = \boldsymbol{x}_\omega / 2c$在投影层和输出层之间Huffman树中的过程。
- 先根据词库中每个词的词频排序，然后从小到大建Huffman树；然后对所有的叶子节点按Huffman编码，非叶子节点若是左子树代码1，右子树代码0，根节点不对应编码（为了方便后面讨论，定义左子树为负类，右子树为正类）。

Reference：[1] 刘建平，基于hs的模型

基于hs的CBOW模型

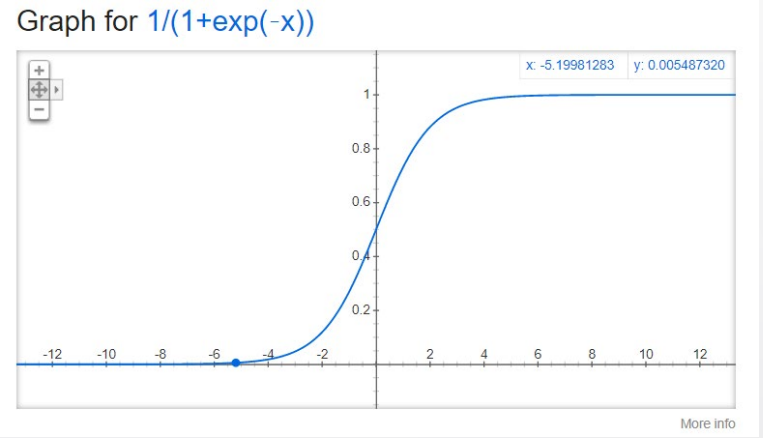
- 在上下文词$Context(\omega)$出现的情况下，中心词ω出现的概率如下：

$$p(\omega|Context(\omega)) = \prod_{j=2}^{l^\omega} p(d_j^\omega | x_\omega, \theta_{j-1}^\omega),$$

- 其中，l^ω为Huffman树从根部到中心词ω的路径中包含节点个数；
- d_j^ω为到中心词ω的路径中第j个节点的所对应的子树代码，$d_j^\omega \in \{0,1\}$；
- θ_{j-1}^ω为路径中第j个节点所对应的向量，不包括叶子节点；
- 综上：该式意思为从Huffman树的根部到达中心词ω的路径中，经过每个节点时，“左拐”或者“右拐”的概率乘积，即为从背景词$Context(\omega)$可以预测到中心词ω的概率大小。

Reference：[1] word2vec中数学原理解释

基于hs的CBOW模型



- 其中，$$p(d_j^\omega \mid x_\omega, \theta_{j-1}^\omega) = \begin{cases} \sigma(x_\omega^T \theta_{j-1}^\omega), d_j^\omega = 0; \\ 1 - \sigma(x_\omega^T \theta_{j-1}^\omega), d_j^\omega = 1 \end{cases},$$
- 该式意思是如果某非根节点是正类，那么用$\sigma(\Delta)$计算其概率；否则，用$1-\sigma(\Delta)$计算概率。其中$\sigma(\Delta)$代表了sigmoid的函数，这是比较常用的激活函数。

$$\sigma(x) = \frac{1}{1+e^{-x}}, \sigma'(x) = \sigma(x)[1-\sigma(x)],$$

- 该函数的图像如右上图所示。该激活函数中，无论是从左边走还是从右边走，都是要使$\Delta = x_\omega^T \theta_{j-1}^\omega$最大，因此我们沿着梯度上升方向，也就是不断加和求最优。

Reference：[1] 刘建平，基于hs的模型

基于hs的CBOW模型

- Sigmoid函数的另一个好用的性质：

$$[\log\sigma(x)]' = 1-\sigma(x), [\log(1-\sigma(x))]' = -\sigma(x),$$

- 为了方便计算，我们可以将$p(d_j^\omega | x_\omega, \theta_{j-1}^\omega)$由分段函数化成：

$$p(d_j^\omega \mid x_\omega, \theta_{j-1}^\omega) = \left[\sigma(x_\omega^T\theta_{j-1}^\omega)\right]^{1-d_j^\omega}\left[1-\sigma(x_\omega^T\theta_{j-1}^\omega)\right]^{d_j^\omega},$$

- 将其代入神经概率模型中常用的对数似然函数 $\mathcal{L} = \sum_{\omega\in C}\log p(\omega \mid Context(\omega))$,
- 之前提过 $p(\omega \mid Context(\omega)) = \prod_{j=2}^{l^\omega} p(d_j^\omega \mid x_\omega, \theta_{j-1}^\omega)$, 故得

$$\mathcal{L} = \sum_{\omega\in C}\log\prod_{j=2}^{l^\omega}\left[\sigma(x_\omega^T\theta_{j-1}^\omega)\right]^{1-d_j^\omega}\left[1-\sigma(x_\omega^T\theta_{j-1}^\omega)\right]^{d_j^\omega}$$

$$= \sum_{\omega\in C}\sum_{j=2}^{l^\omega}\left\{(1-d_j^\omega)\cdot\log\left[\sigma(x_\omega^T\theta_{j-1}^\omega)\right] + d_j^\omega\cdot\log\left[1-\sigma(x_\omega^T\theta_{j-1}^\omega)\right]\right\}$$

Reference：[1] 刘建平，基于hs的模型

基于hs的CBOW模型

- 令 $\mathcal{L}(\omega, j) = (1 - d_j^{\omega}) \cdot \log\left[\sigma(x_{\omega}^T \theta_{j-1}^{\omega})\right] + d_j^{\omega} \cdot \log\left[1 - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})\right]$
- 是基于hs的CBOW模型的目标函数，这里需要解释的是，由于是求$x_{\omega}^T, \theta_{j-1}^{\omega}$的最大值，因此梯度下降法可以被称作“梯度上升”用来提示求目标函数最大值。
- 对目标函数求求关于参数θ_{j-1}^{ω}的偏导:

$$\begin{aligned}\frac{\partial \mathcal{L}(\omega, j)}{\partial \theta_{j-1}^{\omega}} &= \frac{\partial}{\partial \theta_{j-1}^{\omega}}\left\{(1 - d_j^{\omega}) \cdot \log\left[\sigma(x_{\omega}^T \theta_{j-1}^{\omega})\right] + d_j^{\omega} \cdot \log\left[1 - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})\right]\right\} \\ &= (1 - d_j^{\omega})[1 - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})]x_{\omega} - d_j^{\omega}\sigma(x_{\omega}^T \theta_{j-1}^{\omega})x_{\omega} \\ &= [1 - d_j^{\omega} - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})]x_{\omega}\end{aligned}$$

- θ_{j-1}^{ω}的更新规则 $\theta_{j-1}^{\omega\ (new)} := \theta_{j-1}^{\omega\ (old)} + \eta[1 - d_j^{\omega} - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})]x_{\omega}$ ，其中η代表学习率。

Reference：[1] 刘建平，基于hs的模型

基于hs的CBOW模型

- 令 $\mathcal{L}(\omega, j) = (1 - d_j^{\omega}) \cdot \log\left[\sigma(x_{\omega}^T \theta_{j-1}^{\omega})\right] + d_j^{\omega} \cdot \log\left[1 - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})\right]$
- 是基于hs的CBOW模型的目标函数，这里需要解释的是，由于是求$x_{\omega}^T, \theta_{j-1}^{\omega}$的最大值，因此梯度下降法可以被称作“梯度上升”用来提示求目标函数最大值。
- 对目标函数求求关于参数θ_{j-1}^{ω}的偏导:

$$\begin{aligned}\frac{\partial \mathcal{L}(\omega, j)}{\partial \theta_{j-1}^{\omega}} &= \frac{\partial}{\partial \theta_{j-1}^{\omega}}\left\{(1 - d_j^{\omega}) \cdot \log\left[\sigma(x_{\omega}^T \theta_{j-1}^{\omega})\right] + d_j^{\omega} \cdot \log\left[1 - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})\right]\right\} \\ &= (1 - d_j^{\omega})[1 - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})]x_{\omega} - d_j^{\omega}\sigma(x_{\omega}^T \theta_{j-1}^{\omega})x_{\omega} \\ &= [1 - d_j^{\omega} - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})]x_{\omega}\end{aligned}$$

- θ_{j-1}^{ω}的更新规则 $\theta_{j-1}^{\omega\ (new)} := \theta_{j-1}^{\omega\ (old)} + \eta[1 - d_j^{\omega} - \sigma(x_{\omega}^T \theta_{j-1}^{\omega})]x_{\omega}$ ，其中η代表学习率。

Reference：[1] 刘建平，基于hs的模型

基于hs的CBOW模型

- 接下来可以对$\boldsymbol{x}_\omega$求偏导得，

$$\frac{\partial \mathcal{L}(\omega, j)}{\partial x_\omega} = [1 - d_j^\omega - \sigma(x_\omega^T \theta_{j-1}^\omega)]\theta_{j-1}^\omega$$

- 其中 $\boldsymbol{x}_\omega$ 为 $2c$ 个背景词（即上下文词）词向量的累加和，对背景词的词向量们更新，采用的方法是将路径上的每一步的贡献，累加到所有背景词词向量上，即

$$v(\tilde{\omega})^{(new)} := v(\tilde{\omega})^{(old)} + \eta \sum_{j=2}^{l^\omega} \frac{\partial \mathcal{L}(\omega, j)}{\partial x_\omega}, \tilde{\omega} \in Context(\omega)$$

- 到此，便将参数$x_\omega^T, \theta_{j-1}^\omega$的更新过程得到，即背景词的词向量更新，路径中非叶子节点的向量更新，根据ω的上下文找到ω的概率更大。

基于hs的Skip-gram模型

- 与之前不同这个模型是通过中心词预测上下文词。
- 根据hs过程的图，可以看到从输入到投影层是一个恒等映射，直接为中心词向量v_ω。
- 输出的过程和CBOW类似，区别在于：
- Skip-gram中的根节点为中心词词向量v_ω，经过Huffman树需要到达$Context(\omega)$词向量，而$Context(\omega)$有$2c$个，因此每到达一个就更新v_ω。然后到语料库的下一个词继续初始化新中心词ω的词向量v_ω。

Negative Sampling

- 这个方法是Mikolov在文[1]中提出的，这个方法用来提高词的训练速度并且改善了词向量的质量。
- 规定被预测的词为正样本，其他词为负样本。因此“负采样”的意思就是取部分负样本来更新参数。

Reference： [1] Distributed representations of words and phrases and their compositionality.

基于neg的CBOW模型

- CBOW是通过上下文词（背景词）$Context(\omega)$来预测中心词ω，因此对于已知的$2c$个 $Context(\omega)$，ω是一个正样本，其它词就是负样本，即：

$$L^{\omega}(\tilde{\omega}) = \begin{cases} 1, \tilde{\omega} = \omega; \\ 0, \tilde{\omega} \neq \omega \end{cases},$$

- 其中$\tilde{\omega} \in D$。D代表词库所有的词。此式说明正样本标签为1，负样本标签为0。
- 对于一组给定的正样本$(Context(\omega), \omega)$，我们需要最大化

$$g(\omega) = \prod_{u \in \{\omega\} \cup NEG(\omega)} p(u \mid Context(\omega))$$

- 在此式中，$NEG(\omega)$代表了ω的负采样样本，$NEG(\omega) \subset \mathrm{D}$。此式说明给定上下文，需要预测某个词，最大化从其样本集合中预测到ω的概率。

Reference：[1]基于Negative Sampling的模型[2]word2vec 中的数学原理详解[3]word2vec源码

基于neg的CBOW模型

- 这个采样集合$NEG(\omega)$通过负采样算法得出。
- 负采样算法，简单来讲就是将每个词映射到一个数轴上，这个数轴对应1亿单位的长度，词频越高的词所占的长度越长，低频词反之。然后采样时，系统随机生成一个1亿以内的随机整数，整数落到哪个词的范围内，该词即为负采样词。
- $g(\omega)$中的p为：

$$p(u \mid Context(\omega)) = [\sigma(x_\omega^T \theta^u)^{L^\omega(u)} \cdot (1-\sigma(x_\omega^T \theta^u))^{1-L^\omega(u)}]$$

- 其中θ^u为辅助向量（类比hs中的节点向量），和hs中的一样，通过sigmoid函数，对应二分类问题，如果为正样本，对应正类；负样本，对应负类。

Reference：[1]基于Negative Sampling的模型[2]word2vec 中的数学原理详解[3]word2vec源码

基于neg的CBOW模型

- 可以得到$g(\omega)$的表达式为

$$g(\omega)=\prod_{u\in\{\omega\}\cup NEG(\omega)}[\sigma(x_\omega^T\theta^u)^{L^\omega(u)}\cdot(1-\sigma(x_\omega^T\theta^u))^{1-L^\omega(u)}]$$

- 由于$u\in\{\omega\}$时和$u\in NEG(\omega)$时，概率不同，化简得

$$g(\omega)=\sigma(x_\omega^T\theta^\omega)\cdot\prod_{NEG(\omega)}(1-\sigma(x_\omega^T\theta^u))$$

- 注意到这个式子只是关于参数x_ω^T,θ的函数。$\sigma(\Delta)$代表，通过上下文词预测到的词为采样词的概率，采样词可能是目标词ω，也可能是$NEG(\omega)$中的某个负样本。如果是目标词我们希望$\sigma(x_\omega^T\theta^\omega)$越大越好；如果是负样本，我们希望$\sigma(x_\omega^T\theta^u)$越小越好。两者的变化都会导致$g(\omega)$的增大，与前面提到最大化此函数相符合。

Reference：[1]基于Negative Sampling的模型[2]word2vec 中的数学原理详解[3]word2vec源码

基于neg的CBOW模型

- 目标函数，也就是似然函数为

$$\mathcal{L} = \log G = \log \prod_{\omega \in C} g(\omega) = \sum_{\omega \in C} \log g(\omega)$$

$$= \sum_{\omega \in C} \log \prod_{u \in \{\omega\} \cup NEG(\omega)} [\sigma(x_\omega^T \theta^u)^{L^\omega(u)} \cdot (1 - \sigma(x_\omega^T \theta^u))^{1 - L^\omega(u)}]$$

$$= \sum_{\omega \in C} \sum_{u \in \{\omega\} \cup NEG(\omega)} [L^\omega(u) \cdot \log \sigma(x_\omega^T \theta^u) + (1 - L^\omega(u)) \log(1 - \sigma(x_\omega^T \theta^u))]$$

- 和hs一样，可以通过更新方括号内部，来一层层更新参数，令

$$\mathcal{L}(\omega, u) = L^\omega(u) \cdot \log \sigma(x_\omega^T \theta^u) + (1 - L^\omega(u)) \log(1 - \sigma(x_\omega^T \theta^u))$$

- 接下来的过程就是，分别对x_ω, θ^u求偏导，得到其更新规则为。

Reference：[1]基于Negative Sampling的模型[2]word2vec 中的数学原理详解[3]word2vec源码

基于neg的CBOW模型

- 更新规则为：

$$\theta^{u(new)} := \theta^{u(old)} + \eta[L^{\omega}(u) - \sigma(x_{\omega}^{T}\theta^{u})]x_{\omega},$$

$$v(\tilde{\omega})^{(new)} := v(\tilde{\omega})^{(old)} + \eta\sum_{u\in\{\omega\}\cup NEG(\omega)}[L^{\omega}(u) - \sigma(x_{\omega}^{T}\theta^{u})]\theta^{u},$$

- 可以看到，和hs下的CBOW的更新规则非常类似。

Reference：[1]基于Negative Sampling的模型[2]word2vec 中的数学原理详解[3]word2vec源码

基于neg的Skip-gram模型

- 基于前面的hs中Skip-gram和基于neg的CBOW，这个模型主要的不同由于从中心词预测上下文，因此在更新的时候，1.更新完每个采样词的向量后，2.再更新上下文的词，重复过程1、2，直到将$2c$个上下文的词更新完毕。

- 其实neg主要作用就是将hs中构建Huffman树的时间省略了，但是保留了Huffman树中非叶子节点的作用，也就是通过负采样的方法，找到一些负样本，这个寻找的过程是比较快的（相对于造树）。通过优化这些负样本的参数，从而达到优化输入词向量的目的。
- 其实，负采样非常依赖负采样算法，一个高效的算法可以使得采样效率和模型效果都会有很好的表现。

Reference：[1]基于Negative Sampling的模型[2]word2vec 中的数学原理详解[3]word2vec源码

两模型-两方法 训练过程对比

	Hierarchical Softmax			Negative Sampling		
	有效词（个）	训练时间（秒）	速度（个/秒）	有效词（个）	训练时间（秒）	速度（个/秒）
CBOW	549931947	2699.6	2699.6	549935065	1405.7	391229
Skip-gram	549928593	9869.3	5572	549931359	5361.9	102563

可以看到训练时间上来看，基于neg的CBOW最快，也是gensim中默认的训练模型。

模型优劣测试标准

1. 我们通常是通过将词向量用于某些任务中，用这些任务的衡量指标去衡量模型结果。
2. 通过相似度测试，测试英文模型有WordSim353[1]；中文有Peng Jin的中文近义词测试集合[2]。
3. 通过类比测试，有“国王-男人+女人=王后”[3]。

总的来讲，方法一，即根据需求进行测试是最为符合实际的一种。

Reference： [1]WordSim353[2]Evaluating Chinese Word Similarity[3]类比测试分析

类比测试（Analogy Test）：“国王+男人-女人”

- 模型：为了方便测试，随机截取57.5 MB的维基百科中文预料，采用CBOW模型，负采样法，*sample=0.001*。
- Python代码：

```
from gensim.models import word2vec

list = ['10', '100', '200', '300', '400', '500']
for i in list:
    model = word2vec.Word2Vec.load('wiki_corpus_'+ i +'.model')
    result = model.wv.most_similar(positive=[u'国王', u'女人'], negative=[u'男人'])
    j = 0
    for j in range(10):
        print("%s\t%.4f" % result[j])
    print("\n")
```

向量维数（Size）	10		100		200		300		400		500	
Topn=10	Word	Similarity	Word	Similarity	Word	Similarity	Word	Similarity	Word	Similarity	Word	Similarity
Result[0]	二世	0.9676	二世	0.8393	三世	0.8198	一世	0.8046	二世	0.8033	女王	0.8463
Result[1]	一世	0.9569	一世	0.8277	一世	0.8083	三世	0.7980	三世	0.7951	三世	0.8265
Result[2]	三世	0.9554	三世	0.8274	二世	0.8044	二世	0.7977	女王	0.7915	二世	0.8241
Result[3]	拿破仑	0.9547	五世	0.8038	公爵	0.7841	女王	0.7875	一世	0.7807	一世	0.8114
Result[4]	方济各	0.9525	公爵	0.8018	五世	0.7593	公爵	0.7766	公爵	0.7664	六世	0.7686
Result[5]	王朝	0.9419	女王	0.7996	亲王	0.7490	亲王	0.7652	拿破仑	0.7426	教宗	0.7681
Result[6]	阿道夫	0.9381	亲王	0.7911	女王	0.7442	五世	0.7526	亲王	0.7419	五世	0.7603
Result[7]	马木留	0.9366	教宗	0.7642	教宗	0.7435	伯爵	0.7520	教宗	0.7373	公爵	0.7553
Result[8]	沙皇	0.9328	伯爵	0.7637	四世	0.7431	王后	0.7455	五世	0.7302	伯爵	0.7552
Result[9]	教宗	0.9319	王室	0.7618	伯爵	0.7387	四世	0.7385	伯爵	0.7248	拿破仑	0.7510

从测试数据中可以看出，理想的词“女王”随着词向量维数Size的增大，其相似程度也越来越高，因此我们可以初步得出随着词向量维数增大，其准确性也会同步提高。

Reference：Game of Words（词语的加减游戏）

Secondary references

1. https://code.google.com/archive/p/word2vec/
2. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).
3. https://blog.csdn.net/u011734144/article/details/78668518
4. 介绍了二分类问题下逻辑回归时Softmax函利于理层次Softmax http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/
5. 激活函数、BP反向传播、梯度下降、softmax函数及其推导 https://blog.csdn.net/soundslow/article/details/78155281
6. word2vec源码，里面很计算和理论不是非常一致，需要再捋一遍 https://github.com/tmikolov/word2vec/blob/master/word2vec.c

Secondary references（CBOW）

1. 一个具体的小例子，解决了输入时求和的困惑
   https://blog.csdn.net/u011734144/article/details/78668518
2. 也是一个具体小例子，其中的$ativation(n)$应该是W和W'的行列词向量的乘积值。
   https://blog.csdn.net/weixin_40240670/article/details/81203428
3. 强化了自己对BP（反向传播）算法及随机梯度下降来学习权重的理解
   https://blog.csdn.net/u010665216/article/details/78724856
4. 博主自己写代码，明确了代码过程，还需多看（包括SG和huffman树）
   https://blog.csdn.net/u014595019/article/details/51943428

Secondary references（Skip-Gram）

- 标红字处还需要认真核验，加深理解。
- 感谢漆教授查看!