本文描述了基于神经网络的统计语言模型，包括 NNLM，RNNLM，以及 word2vec 方法，CBOW，skip-gram 等。

一．n-gram 与 nnlm，rnnlm

1.1 n-gram 语言模型

在 LSA，LDA 等传统的语言模型中，是利用词在文章中出现频率的统计规律对文档进行建模。而 n-gram 语言模型是一种对句子建模的模型，它涉及到自然语言处理的一个关键问题，判断一个句子是否是自然语言，即是否符合语义和语法规律（semantic and syntactic）。句子 W 由一系列的词组成 w1、w2、w3……wn，我们假设每个词 wi 出现的可能性与前面 1 到 i-1 个词有关，那么句子 W 出现的概率可以建模为：

$P(W) = p(w_1)\, p(w_2|w_1)\, p(w_3|w_1,w_2)\ldots\ldots p(w_n|w_1,w_2,\ldots,w_{n-1})$

则我们需要知道 $P(w_i|w_1,w_2,\ldots,w_{i-1})$，但是由于 i 的长度没有限制，如果对大小为 V 的词表计算每一个词在其他所有词任意长度的排列组合下的条件概率，由于参数过多，计算量非常的大，而且在数据集有限的情况下这些条件的出现非常稀疏。观察到通常离 w_i 近的词与它的相关性最大，所以我们对 w_i 之前起作用的词的数量加以限制，假设前 n-1 个词对其有影响，这就是 n-gram 模型，当 n 等于 2 时：

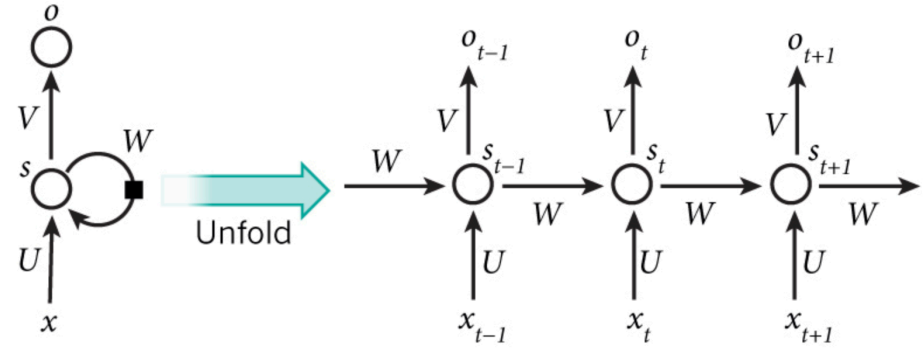
$P(W) = p(w_1)\, p(w_2|w_1)\, p(w_3|w_1)\ldots\ldots p(w_n|w_{n-1})$

这样我们只需要计算大小为 V*V 的 $P(w_i|w_{i-1})$矩阵，就可以得到句子的似然概率，利用贝叶斯公式：$P(w_i|w_{i-1}) = P(w_i,w_{i-1})\backslash P(w_{i-1})$。在 w_{i-1} 后面出现 w_i 的概率为这个两个词共同出现的联合概率除以 w_{i-1} 在所有单词中出现的概率。

基于 n-gram 模型得到的语言模型经常用 preplexity 进行评估。

1.2 循环神经网络语言模型（rnnlm）

在 nnlm 中，我们使用固定长度的上下文对当前词进行预测，然而该长度之外的单词对当前词仍存在影响，所以我们希望能找到一种能够处理任意长度上下文，可以保留历史输入的结构。循环神经网络是一种适用于序列数据的网络，包含一个输入层，一个隐藏层和一个输出层。



rnn 的计算方法如下：

$$o_t = g(Vs_t)$$

$$s_t = f(Ux_t + Ws_{t-1})$$

o_t 是输出层计算公式，s_t 是隐藏层计算公式。左图书 rnn 的基础结构，有图书 RNN 按时间展开的结果，rnn 的输出不只受当前输入 xt 影响，还受 xt 之前的

输入影响，通过隐藏层的状态传递过来。

具体用于单词序列的是，首先设置任意一个初始值作为第一个输入，然后希望输出第一个词 w1，然后将第一个词 w1 作为输入，希望输出 w2，以此类推。在 t 时刻，输入 wt 时，输出层使用 softmax 函数输出一个 V 维多分类结果，作为对 wt+1 的预测，此时的损失函数通过 wt+1 和 ot+1 的交叉熵误差函数求出。这个误差一边反向传播到输入层，一边沿着时间线传递到初始 t1 时刻。

作为语言模型，我们最小化交叉熵的同时，就达到了最小化 mean perplexity 的目的，因为 perplexity 就是交叉熵的指数形式。

1.3 神经网络语言模型（nnlm）

而近年来，随着深度学习的发展，以 word embedding 为代表的分布式表示（Distributed Representation）取得了很好的效果。其中 NNLM 是比较早的一篇。

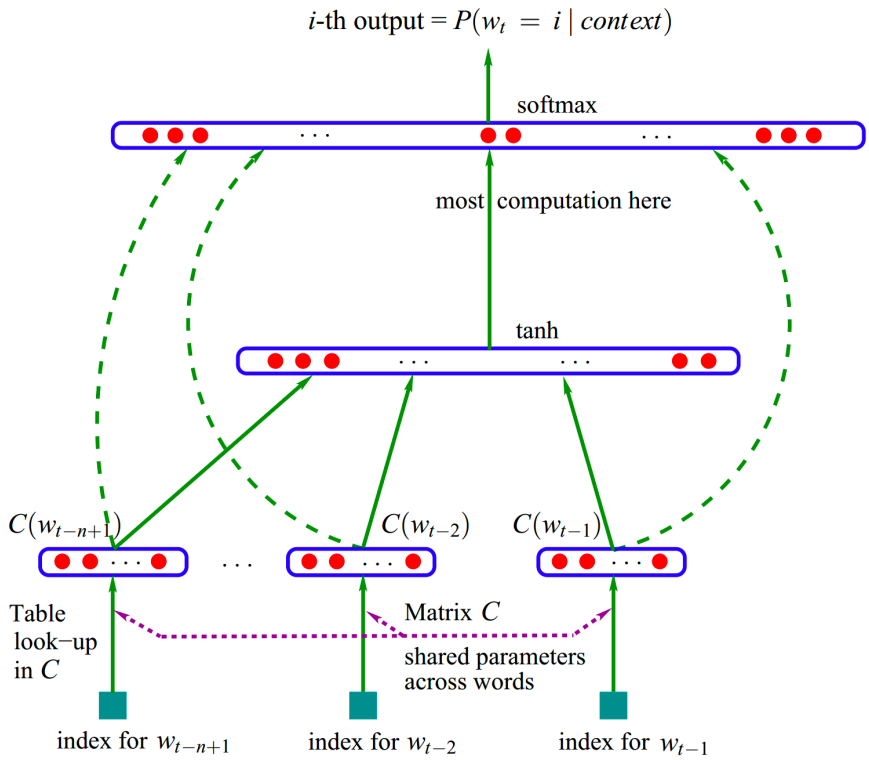
对连续的变量建模是一项相对简单的任务，使用混合高斯模型或者多层神经网络都可以完成建模；而在离散空间中，比如对长度为 10 的句子（其中每个词取自长度为 V 的词表）建模，每个词可以看做是一个 one-hot 表示的 V 维向量，这相当于要对离散随机变量的联合分布进行建模，其中某个变量任意的改变（比如从（1，0，0，0）变为（0，1，0，0）），都可能导致要要评估的联合概率的剧烈的变化，因为改变了其中的一个词，但是却无法得知这个词和原来的词的相似程度；而当随机变量的维度 V 很大时，观测到的句子的汉明距离大部分都很远，测试集中的句子可能和训练集中的所有句子看起来都非常不同，对这就是维度灾难问题。对这样的高维数据建模需要的参数数量很大，n-gram 模型希望通过限制建模每个词所使用的的上下文范围来减少参数的数量，而这也带来了新的问题。

n-gram 模型利用在训练集中词的组合的出现频率来进行建模，而当一个新的组合出现时，模型认为这个组合出现的概率为 0，这显然是不合理的，因为训练集的大小是有限的，也许该组合在一个更大的训练集中会出现。此外，同样受限于训练集大小的是，我们不能将 n 设置的非常大（参数的数量为 V^n），这就导致了该模型只利用了单词的很小的上下文内容。而由于每个词被当做独立的原子单元，作为词表 V 中的一个索引 ID，这个模型也无法衡量单词之间的相似性。

由于维度灾难出现是因为在离散空间中，随机变量 w_i 可能的取值很多（或者看作 V 维随机变量的维度很高），导致训练的参数过多，nnlm 提出，如果可以将每个单词映射到一个低维连续空间中去，然后利用擅长在连续空间中进行训练的神经网络，就可以有效的减少参数的数量，同时在这个连续空间中可以对单词之间的距离（相似性）进行计算。这种方法首先要将词表中每个单词映射到一个特征向量上去，然后利用单词特征向量表示出单词序列的联合分布，然后进行训练，同时学习单词的特征向量和联合概率分布中的参数。

模型的理论基础类似 n-gram，单词序列的联合概率分布用每个单词在它前面单词序列下的条件概率的乘积$\prod P(w_i|context(w_i))$，训练参数时只要最大训练集的似然概率就可以。这个模型的泛化性非常好，因为相似的词的特征向量会很接近，而 P(W)概率函数是一个关于 wi 特征向量连续的函数，这样当其中某个词的特征向量有一个微小的改变时，概率函数也会产生微小的改变。

NNLM 模型见下图：



这个模型由两部分构建P(wi|context(wi))：一个大小为 V*m 的 embedding 矩阵 C 将词表 V 中第 i 个单词映射到特征向量 C(i)∈R^m，一个由 nn 构建的函数 g 将 context(w_i)映射后的特征向量作为输入，输出 V 中每个词在 context(w_i)下的条件概率。

通过矩阵 C 映射过程如下：

$C(w_i) = w_i * C$

w_i 的前 n-1 个上下文单词经过映射后得到的向量进行拼接作为神经网络 g 的输入：$x = (C(w_{t-n+1}),\ldots,C(w_{t-1}))$，长度为(n-1)m

g 包含一个长度为 h 的隐藏层，使用 tanh 激活函数，和一个长度为 V 的输出层，输出层利用 softmax 函数得到在当前上下文下每个单词出现的概率：

$y = b+Wx+U \tanh(d + Hx)$

$$p(w_i = k) = \frac{e^{y_k}}{\sum_l e^{y_l}}$$

其中参数包括 b,W,U,d,H，其中 W 是一个从输入层直接连接到输出层的矩阵，可设置为 0。这些参数加上 embedding 矩阵 C 就是这个模型所有的参数，对数据库中所有长度为 n 的单词序列用这个网络进行预测，预测的误差反向传播，对所有参数进行随机梯度下降即可。

二．Word2vec

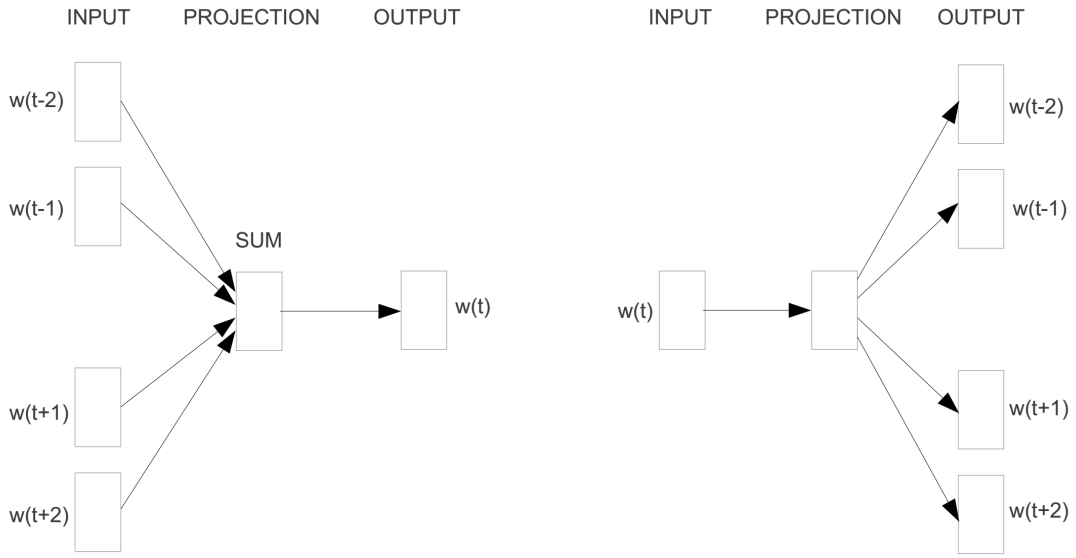
2.1 Skip gram 和 CBOW

在 nnlm 中，对每个长度为 n 的序列进行训练的复杂度为：n*m+n*m*h+h*V，其中最耗时的计算集中在隐藏层到输出层的计算 h*V，因为 V 作为词表的大小可能会非常大，存在一种词表的二叉树表示方法，使得我们可以在 log(V)的时间内计算出输出单元（见第三部分），此时模型的复杂度集中在 n*m*h，也就是从

embedding 层到隐藏层的计算。由于 nnlm 模型的高复杂度，使得它一直不能在大规模、大词表的数据集上成功训练，skip gram 和 CBOW 模型的提出是希望找到能够在这样的数据集上快速训练出高质量的 word embedding 的方法。

由于 nnlm 大部分的复杂度是由模型中的非线性隐藏层造成的，这篇文章提出去掉这个隐藏层来简化模型：continuous bag of words（CBOW）模型类似于 NNLM 模型，有三点区别：1. 移除了隐藏层 2. 所有单词共享了映射层（也就是下图中的到 SUM 的四根线是相同的变换），将所有的上下文单词映射到同一个点（他们的平均点），这样计算时就简化掉了窗口大小 n；nnlm 只是所有单词共享 embedding 矩阵，然后将上下文单词的特征向量进行拼接。3. 将 embedding 后的特征向量直接作为输入，而不是作为参数，反向传播的时候将误差传到输入层。

由于这个模型中，上下文中单词的顺序并不影响预测结果，相当于词袋模型，由于使用的是单词的连续分布式表示所以称之为 CBOW。本模型除了使用预测单词前的单词序列外，还使用了未来的单词。使用 Hierarchical softmax 的方法，复杂度为 n*m+m*log(V)



CBOW **Skip-gram**

skip-gram 模型类似 CBOW 模型，但是是反过来用当前词来预测它的上下文。由于使用上下文单词距离当前词的距离越远，两个词的相关性越小，所以该方法通过减小了使用远距离单词的可能性，从而降低了这些单词的权重：C 是可能使用单词的最远距离，在每次训练中，随机从（1;C）中取一个数字 R，然后使用当前单词的前后 2R 个上下文单词。该方法在复杂度为 C*(m+m*log(V))。需要注意的是，在上图中，单词特征向量从输入层到映射层不做变换。

注意到，nnlm 作为语言模型目的是通过该模型生成的句子的 perplexity 更小，word embedding 结果是作为一个副产品。本方法的目的在于产生高质量的 word embedding，希望 embedding 后的向量空间中存在一些语法和语义的性质，可以完成比如 vector(”King”) - vector(”Man”) + vector(”Woman”) = vector(”Queen”)的代数运算，本方法并不能作为语言模型来生成句子。

2.2 Hierarchical softmax and negative sampling

2.2.1 逻辑回归和哈夫曼编码

逻辑回归是一种用于分类问题的算法，此处只分析逻辑回归用于二分类问题的情况。二分类问题的数据集为$\{x_i,t_i\}$，其中 $i\in \{1,2,\ldots,n\}$, $t_i\in\{1,0\}$。

二分类问题实际上要找到一个模型，为每个数据 t_i 确定一个伯努利分布，以确定 $P(t_i = 1)$的概率值，这个值由 y_i 表示，表示 x_i 属于第一类的概率。这样 t_i 的概率密度函数为：

$$P(ti) = y_i^{ti}(1-y_i)^{1-ti} = \begin{cases} y_i & \text{if } t_i = 1 \\ 1-y_i & \text{if } t_i = 0 \end{cases}$$

y_i 是关于 x_i 的函数，在逻辑回归中，这个函数通过 logistic 函数表示：

$y = sigmoid(w x)$

这样，对于整个数据集，似然函数可以写成：

$$p(t_n) = \prod_{n=1}^{N} y_n^{t_n}\{1-y_n\}^{1-t_n}$$

负似然对数：

$$V = -\ln p(t_n) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$

可以看到这个 loss function 其实是交叉熵误差函数，利用这个 loss function 对 w 进行梯度下降即可。

哈弗曼编码是一种数据压缩的熵编码，给出现频率越高的单词越短的编码，以实现数据压缩，而编码的结果也体现了出现概率越大的单词携带的信息越少（log(1/p)）的信息论思想。

哈夫曼编码使用了哈弗曼树结构，哈夫曼树又称最优二叉树，是一种带权路径长度最短的二叉树。所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其到根结点的路径长度（若根结点为 0 层，叶结点到根结点的路径长度为叶结点的层数）。二叉树是每个结点最多有两个子树的有序树，两个子树称为左子树，右子树，两个子树顺序不能交换。

给定 n 个权值作为 n 个叶子节点，构造一棵二叉树使它的带权路径长度最小，就是 huffman 树。假设有 n 个权值，则构造出的哈夫曼树有 n 个叶子结点。n 个权值分别设为 w1、w2、…、wn，则哈夫曼树的构造规则为：

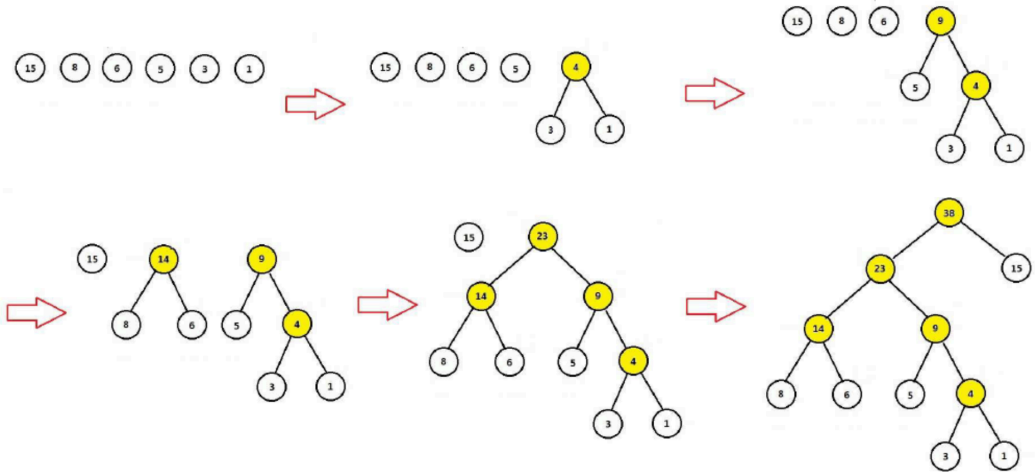
(1) 将 w1、w2、…，wn 看成是有 n 棵树的森林(每棵树仅有一个结点)；

(2) 在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点权值之和；

(3)从森林中删除选取的两棵树，并将新树加入森林；

(4)重复(2)、(3)步，直到森林中只剩一棵树为止，该树即为所求得的哈夫曼树。

例 2.1 假设 *2014* 年世界杯期间，从新浪微博中抓取了若干条与足球相关的微博，经统计，“我”、“喜欢”、“观看”、“巴西”、“足球”、“世界杯”这六个词出现的次数分别为 *15, 8, 6, 5, 3, 1*. 请以这 *6* 个词为叶子结点，以相应词频当权值，构造一棵 *Huffman* 树.



可见词频越大的词离根节点越近，将每棵树的左子结点编码为 1，右子结点编码为 0，那么我们就得到了一个哈夫曼编码，给词频越低的词越短的编码。

2.2.2 Hierarchical softmax

不论 CBOW 还是 skip-gram 的两个模型（以及最早的 NNLM），网络的最后一层都要进行 softmax 计算，这是因为这些模型都要输出 $p(w_i|x)$的结果，其中 x 为 $context(w_i)$或者 w_t（在 skip-gram 中，$w_i \in content(w_t)$）。计算单词 wi 在当前条件下的条件概率，在普通的 softmax 函数中要对 V 个输出结点进行计算，而 Hierachical softmax 是一种利用树结构更有效的计算 softmax 的方法，只需要对 log(V)个结点进行计算即可得到 softmax 结果。

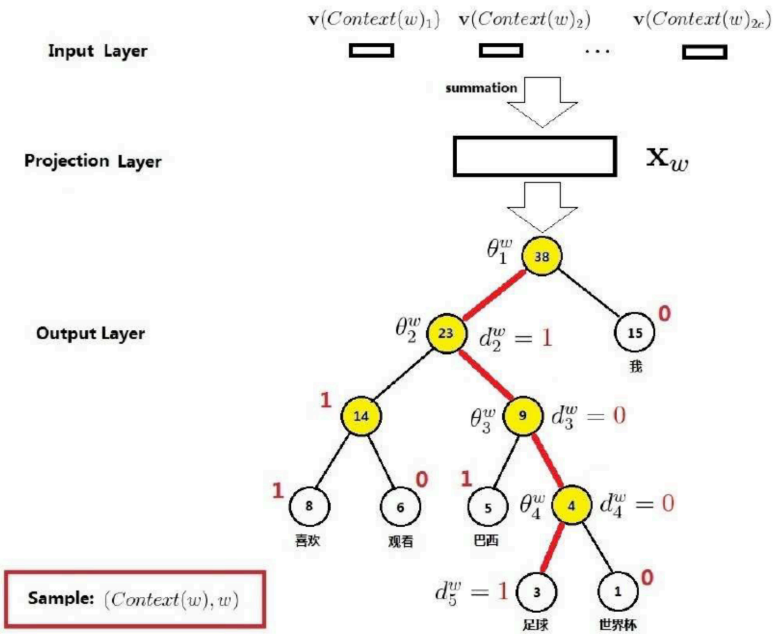
这种方法用 huffman 树组织单词，w_i 是某个叶子结点，要计算 $P(w_i|x)$，相当于做多次的二分类问题，每到一个结点，进行一次路径选择，选择左\右结点的概率用 sigmoid 函数表示（相当于逻辑回归）。如下图，足球的编码为 1001，那么 P(足球|x)等于每层概率的乘积：

1. $p(w|x_w,\theta_1)=\sigma(x_w^T\theta_1)$，向左走；
2. $p(w|x_w,\theta_2)=1-\sigma(x_w^T\theta_2)$，向右走；
3. $p(w|x_w,\theta_3)=1-\sigma(x_w^T\theta_3)$，向右走；
4. $p(w|x_w,\theta_4)=\sigma(x_w^T\theta_4)$，向左走。

其中 θ 是每个非叶子结点的 softmax 参数（不同于普通 softmax 每个单词对应一个参数），这样得到了 $P(w_i|x)$的结果：

$$p(w|Context(w))=\prod_{j=1}^{l_w}[\sigma(x_w^T\theta_j)]^{d_j^w}\cdot[1-\sigma(x_w^T\theta_j)]^{1-d_j^w}$$

l_w 是单词 w 所在路径长度，d_j 表示 w 在第 j 层的哈夫曼编码。有了该式，我们就可以对 word2vec 模型求参数的最大似然解。



2.2.3 negative sampling

在 2.2.2 中，我们指出 word2vec 模型都会落到计算 p(w|context(w))的任务上来，为此在模型的输出层进行了 softmax 的计算。而事实上，我们还可以把要预测的 one-hot 向量看作是 V 个二分类结果，其中 w 作为正样本，其他的所有 V-1 个词作为负样本。那么我们的目标可以变为训练模型使正样本出现的概率尽可能大（接近 1），同时使负样本出现的概率尽可能小（接近 0）。对给定正样本 w，我，希望最大化 g(w)= ∏p(u|context(w))，对 u∈V 求乘积，这里p(u|context(w))不代表在 V 个单词中 u 可能出现的概率，而是意味着 u 在该上下文条件下，是否会出现（伯努利分布）：$p(u|Context(w)) = \left[\sigma(\mathbf{x}_w^\top \theta^u)\right]^{L^w(u)} \cdot \left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right]^{1-L^w(u)}$，Lw(u)表示是否（1\0）w 的上下文条件下能得到 u。

我们不需要对所有的负样本进行计算，因为 V-1 个词中可能包含一部分潜在正样本（比如词义和 wi 相似的词），我们只从中取样一部分，这样也可以有效减小计算量，取样的负样本，记为 NEG(w)。则目标函数可以写作：

$$
\begin{aligned}
\mathcal{L} &= \log G = \log \prod_{w\in\mathcal{C}} g(w) = \sum_{w\in\mathcal{C}} \log g(w) \\
&= \sum_{w\in\mathcal{C}} \log \prod_{u\in\{w\}\cup NEG(w)} \left\{ \left[\sigma(\mathbf{x}_w^\top \theta^u)\right]^{L^w(u)} \cdot \left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right]^{1-L^w(u)} \right\} \\
&= \sum_{w\in\mathcal{C}} \sum_{u\in\{w\}\cup NEG(w)} \left\{ L^w(u)\cdot \log\left[\sigma(\mathbf{x}_w^\top \theta^u)\right] + \left[1-L^w(u)\right]\cdot \log\left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right] \right\}. \\
&= \sum_{w\in\mathcal{C}} \left\{ \log\left[\sigma(\mathbf{x}_w^\top \theta^w)\right] + \sum_{u\in NEG(w)} \log\left[1-\sigma(\mathbf{x}_w^\top \theta^u)\right] \right\} \\
&= \sum_{w\in\mathcal{C}} \left\{ \log\left[\sigma(\mathbf{x}_w^\top \theta^w)\right] + \sum_{u\in NEG(w)} \log\left[\sigma(-\mathbf{x}_w^\top \theta^u)\right] \right\} (
\end{aligned}
$$

注意，采样的时候一般设置词频高的单词被采样的概率大。利用这个目标函数对 θ 和 embedding 矩阵进行训练。