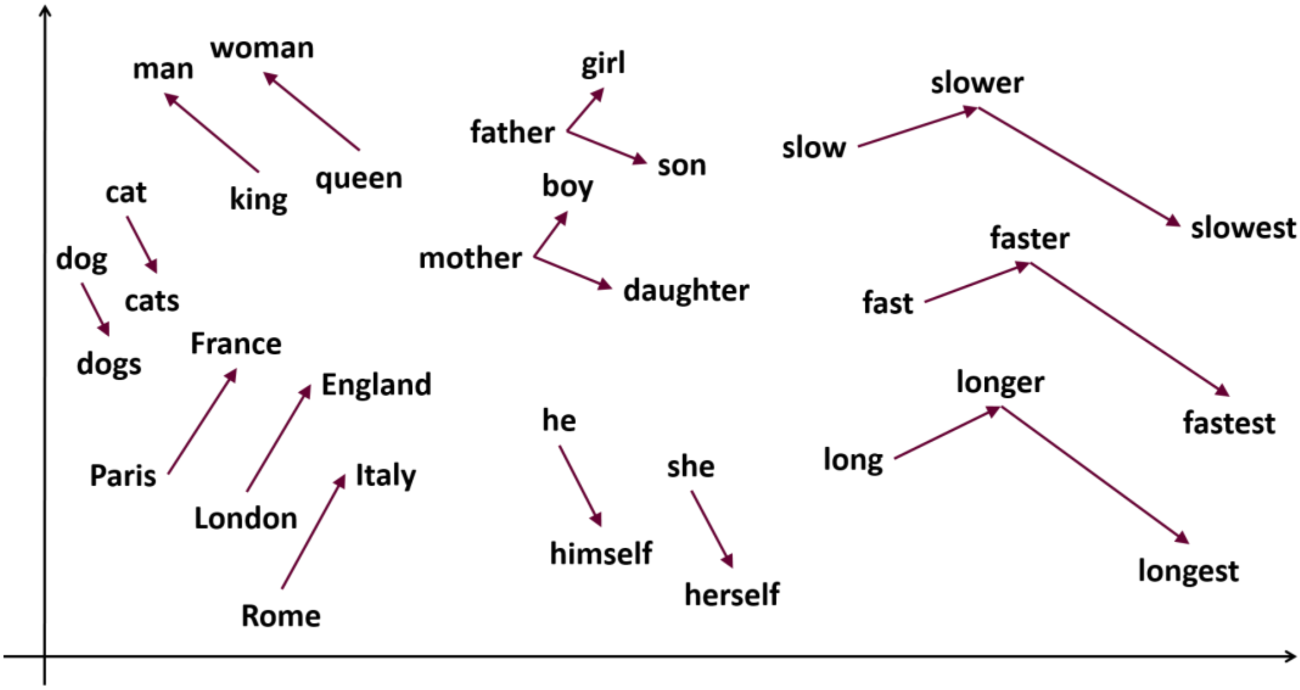
词嵌入Word Embedding

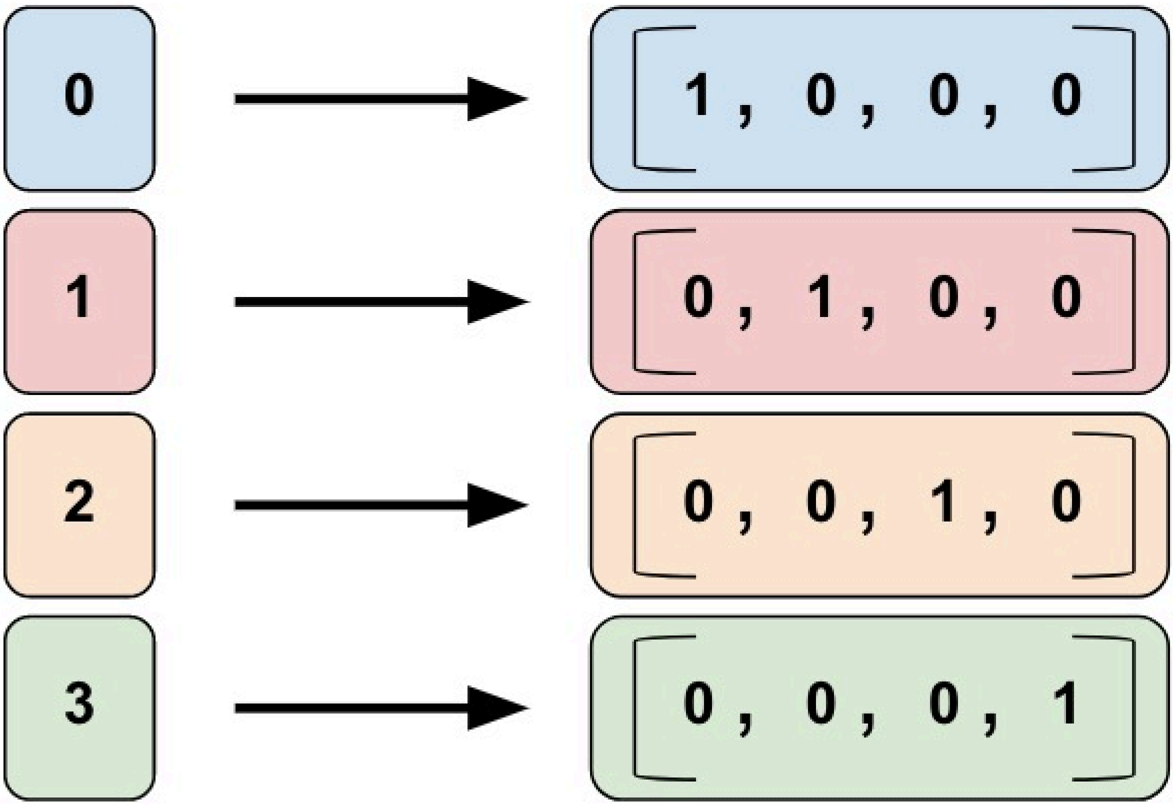
词的表示方法

语言是高度抽象的离散符号系统。为了能够使用神经网络来解决NLP任务，深度学习的第一步就是将离散的符号变成向量。把词语映射到语义空间中的一个点，是的相似的词相近不相似的的词较远，用向量来表示一个点，该向量为词向量。



one-hot向量

最简单的方法就是one-hot表示。



one-hot的问题是不满足我们前面的期望——相似的词的距离较近而不相似的较远。
one-hot是一个高维度的稀疏向量，我们希望用一个低维度的稠密向量来表示一个词。其中每一个维度都是一种语义，词义相近的向量距离近。

神经网络语言模型

N-Gram模型

N-Gram模型是一种基于统计语言模型计算的模型，基本思想为将文本里面的内容按照字节进行大小为N的滑动窗口操作，形成了长度是N的字节片段序列。

每一个字节片段都成为gram，对所有gram出现的频度进行统计，按照事先设定好的阈值进行过滤，形成**关键gram列表**，也就是这个文本的向量特征空间，列表中的每一种gram就是一个特征向量维度。
该模型基于这样一种假设，第N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，整句的概率就是各个词出现概率的乘积。这些概率可以通过直接从语料中统计N个词同时出现的次数得到。常用的是二元的Bi-Gram和三元的Tri-Gram。

给定词序列w1,…,wKw1,…,wK，语言模型会计算这个序列的概率，根据条件概率的定义，我们可以把联合概率分解为如下的条件概率：

$$P(w) = \prod_{k=1}^{K} P(w_k | w_{k-1}, \ldots, w_1)$$

实际的语言模型很难考虑特别长的历史，通常我们会限定当前词的概率值依赖与之前的N-1个词，在实际的应用中N的取值通常是2-5。

$$P(w) = \prod_{k=1}^{K} P(w_k|w_{k-1}, \ldots, w_{k-N+1})$$

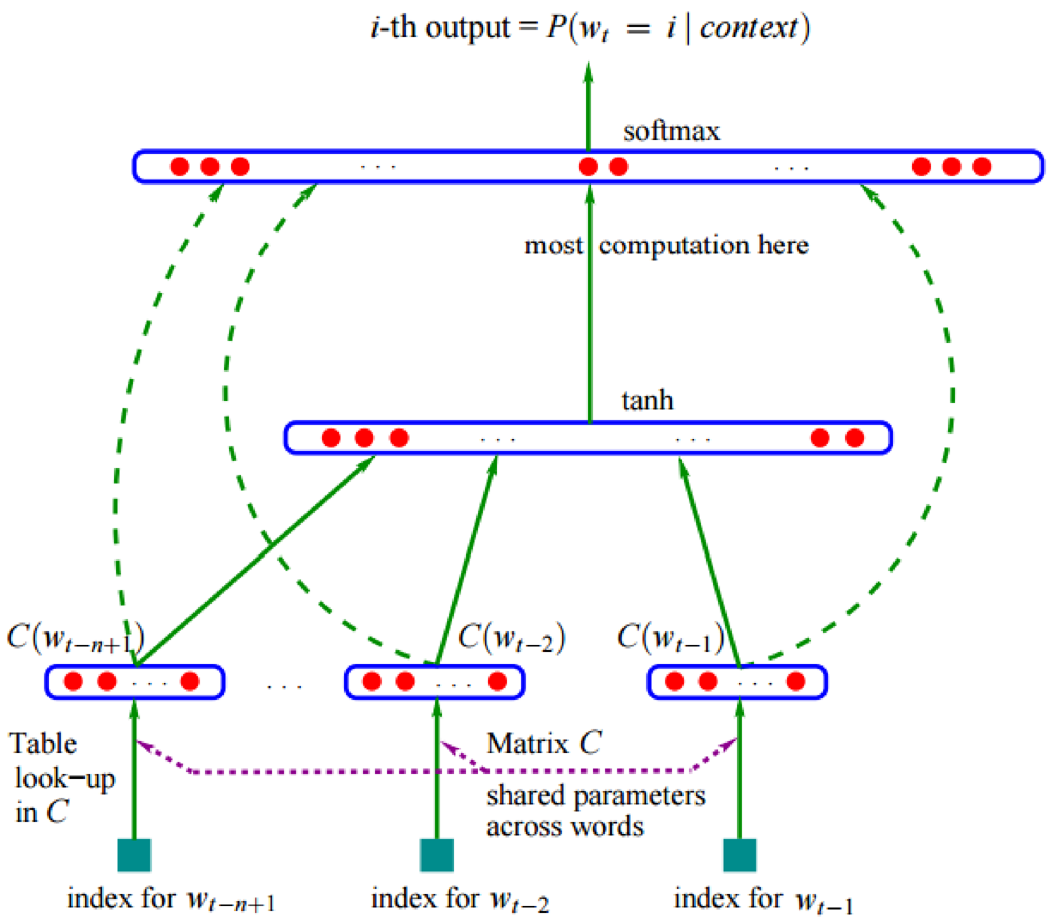
通常用困惑度(Perplexity)来衡量语言模型的好坏：

$$H = -\lim_{K\to\infty} \frac{1}{K} log_2 P(w_1, \ldots, w_K)$$
$$\approx \frac{1}{K} \sum_{k=1}^{K} log_2(P(w_k|w_{k-1}, \ldots, w_{k-N+1}))$$

N-Gram语言模型有两个比较大的问题。第一个就是N不能太大，否则需要存储的N-gram太多，因此它无法考虑长距离的依赖。

另外一个问题就是它的泛化能力差，因为它完全基于词的共现。

通过把一个词表示成一个低维稠密的向量就能解决这个问题，通过上下文，模型能够知道北京和上海经常出现在相似的上下文里，因此模型能用相似的向量来表示这两个不同的词。



图：神经网络语言模型

这个模型的输入是当前要预测的词，比如用前两个词预测当前词。模型首先用lookup table把一个词变成一个向量，然后把这两个词的向量拼接成一个大的向量，输入神经网络，最后使用softmax输出预测每个词的概率。

Lookup table等价于one-hot向量乘以Embedding矩阵。假设我们有3个词，词向量的维度是5维，那么Embedding矩阵就是(3, 5)的矩阵，比如：

$$\begin{bmatrix} 1.5 & 2.3 & -3.2 & 4.8 & 5.1 \\ 8.3 & 3.3 & 4.1 & -5.3 & 6.8 \\ 3.2 & -4.8 & 5.5 & 16 & -0.7 \end{bmatrix}$$

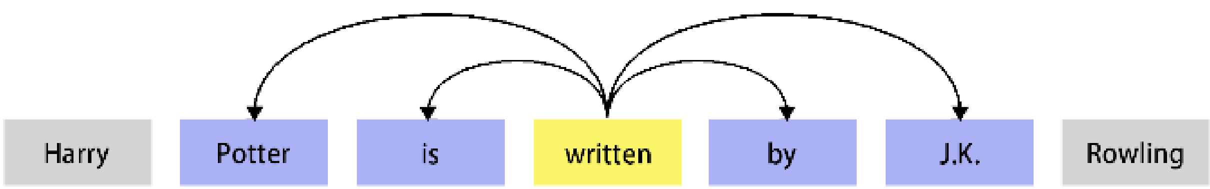
这个矩阵的每一行表示一个词的词向量，那么我们要获得第二个词的词向量，就可以用如下的向量矩阵乘法来提取：

$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1.5 & 2.3 & -3.2 & 4.8 & 5.1 \\ 8.3 & 3.3 & 4.1 & -5.3 & 6.8 \\ 3.2 & -4.8 & 5.5 & 16 & -0.7 \end{bmatrix} = \begin{bmatrix} 8.3 & 3.3 & 4.1 & -5.3 & 6.8 \end{bmatrix}$$

这个Embedding矩阵不是固定的，它也是神经网络的参数之一。通过语言模型的学习，我们就可以得到这个Embedding矩阵，从而得到词向量。

Word2Vec

Word2Vec的基本思想就是Distributional假设(hypothesis)：如果两个词的上下文相似，那么这两个词的语义就相似。上下文有很多粒度，比如文档的粒度，也就是一个词的上下文是所有与它出现在同一个文档中的词。也可以是较细的粒度，比如当前词前后固定大小的窗口。比如下图所示，written的上下文是前后个两个词，也就是”Portter is by J.K.”这4个词。



图：词的上下文

还有很多其它方法也可以利用上述假设学习词向量。所有通过Distributional假设学习到的(向量)表示都叫做Distributional表示(Representation)。

还有一个很像的术语叫Distributed表示(Representation)。它其实就是指的是用稠密的低维向量来表示一个词的语义，也就是把语义”分散”到不同的维度上。与之相对的通常是one-hot表

示，它的语义集中在高维的稀疏的某一维上。

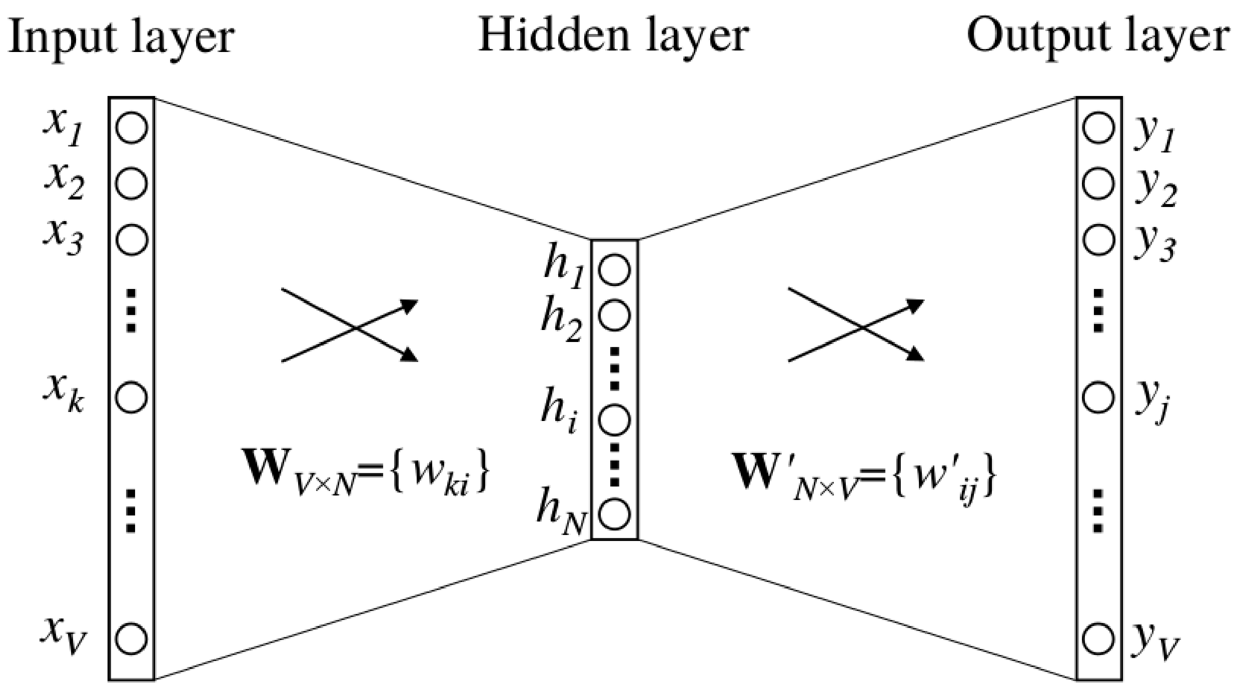
Word2Vec包含两个模型：CBOW（Continuous Bag-of-Word）词袋模型和SG（Skip-Gram）模型。

CBOW模型类似完形填空。将一个句子中的某个词扣掉，从全文所有的词语中挑选一个合适的词填入其中，通过计算每一个词的可能性来实现。

上下文（context）只有一个词

词典的大小是V(词的个数)，隐层的隐藏单元个数是N（词向量的长度为N）。输入层-隐层之间是全连接的神经网络。输入是one-hot的形式，即不需要计算矩阵乘法，仅仅需要提取出地k行的相关联即可。Word2Vec的隐层一般不使用激活函数。

$$h = W^T x = W^T_{(k,.)} \equiv W^T_{w_I} \quad (公式1)$$



图：上下文只有一个词的CBOW模型

输出层计算第j个词的得分：

$$u_j = v'^T_{w_j} h \quad (公式2)$$

为了计算得到概率，对所有的词的得分进行softmax：

$$p(w_j|w_I) = y_j = \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u_{j'})} \quad (公式3)$$

输入词向量V_w和输出词向量V'_w的内积越大，代表两个词越相似，则$P(W_j|W_i)$就较大。

接下是反向梯度计算。word2vec的损失函数是交叉熵损失，优化时最小化损失函数，因此取负对数似然，目标是让uj的得分远高于其他词的得分，从而使yj的概率趋近于1：

$$E = -\log y_j = -u_j + \log \sum_{j'=1}^{V} \exp(u_{j'})$$

其中：

- U_j是模型对目标词语未归一化的得分。
- y_j是模型对目次预测的归一化概率。
- V是词汇表的大小。

E是u_j的函数，对其求偏导如下：

- 对于目标词来说，第一项的导数是-1。第二项需要用到链式法则

$$\frac{\partial \log Z}{\partial u_j} = \frac{1}{Z} \cdot \frac{\partial Z}{\partial u_j}$$

，其结果为$exp(u_j)$。
- 对于非目标词来说，第一项的导数为0。第二项的导数为$exp(u_k)$。

$$\frac{\partial E}{\partial u_j} = -t_j + \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u_{j'})} = y_j - t_j \equiv e_j$$

- **总结**：
  当为目标词时，tj=1，非目标词时为0。

求出后采用梯度下降法更新参数：

$$w'^{(new)}_{ij} \leftarrow w'^{(old)}_{ij} - \eta \cdot e_j \cdot h_i$$

向量形式：

$$v'^{(new)}_{w_j} \leftarrow v'^{(old)}_{w_j} - \eta \cdot e_j \cdot h \ , \ for \ j = 1, 2, \dots, V$$

对于每一个训练数据，都需要更新所有V个词对应输出的词向量，计算量十分大。接下来计算E对隐层输出h的梯度：

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} \equiv EH_i$$

$$h_i = \sum_{k=1}^{V} x_k \cdot w_{ki}$$

因此可求出：

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = EH_i \cdot x_k$$

向量形式：

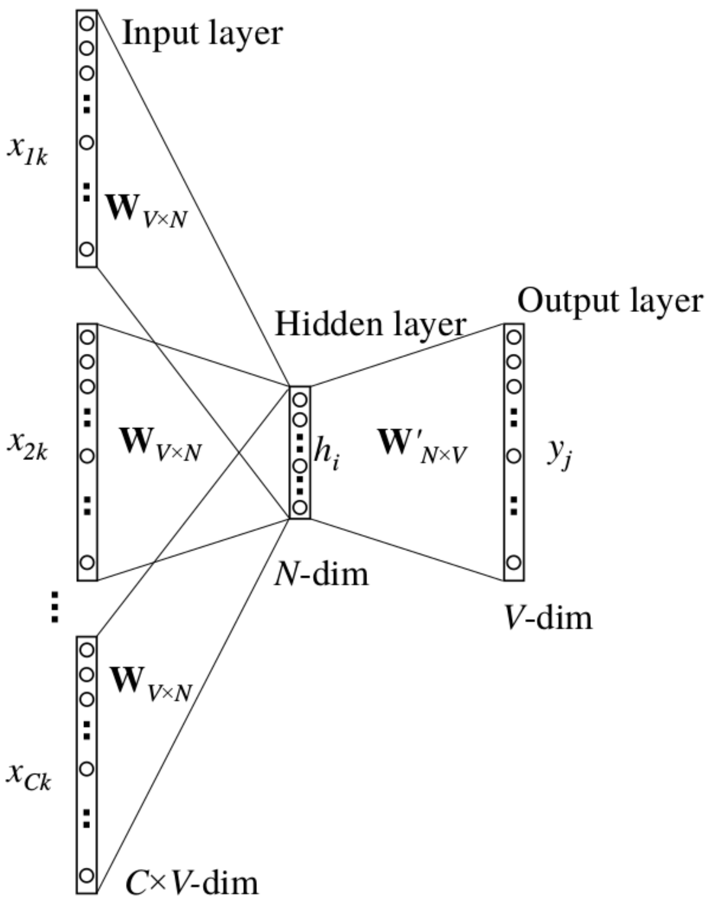
$$\frac{\partial E}{\partial W} = x \bigotimes EH^T$$

这是一个V*N的矩阵，但是x只有一个元素非零，因此对应的梯度也只有一行是非零的。我们只需要更新输入词向量对应的那一行 的参数：

$$v^{(new)}_{w_I} \leftarrow v^{(old)}_{w_I} - \eta EH^T$$

上下文（context）为多个词

用一个词周围的多个词来预测这个词。



图：CBOW模型

使用onehot来表示每一个词，使用最简单平均来输入到CBOW模型：

$$h = \frac{}{C} W^T (x_1 + x_2 + \ldots + x_C) = \frac{1}{C} (v_{w_1} + v_{w_2} + \ldots + v_{w_C})^T$$

计算出h后，之后所有计算都和上下文是一个词时相同，因此可以计算损失：

$$E = -log y_{j^*} = -u_{j^*} + log \sum_{j'=1}^{V} exp(u_{j'}) = -v'^{T}_{w_O} \cdot h + log \sum_{j'=1}^{V} exp(v'^{T}_{w_j} \cdot h)$$

更新输出向量的梯度更新公式不变：

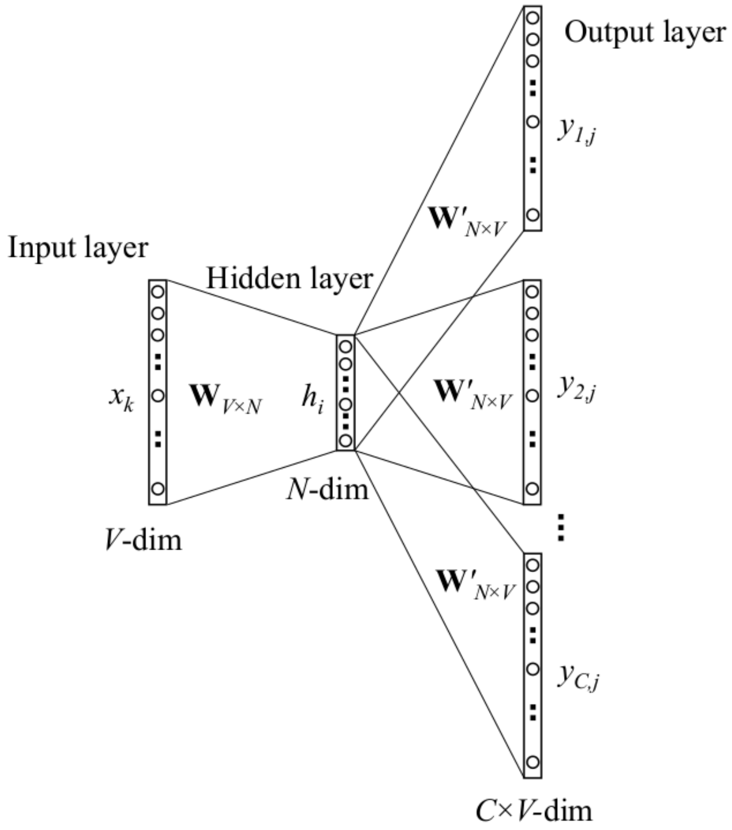
$$w'^{(new)}_{ij} \leftarrow w'^{(old)}_{ij} - \eta \cdot e_j \cdot h_i$$

输入向量的梯度更新稍微有点区别，因为计算h时进行了平均，计算梯度是也要乘以1/C：

$$v^{(new)}_{w_{I,c}} \leftarrow v^{(old)}_{w_{I,c}} - \frac{1}{C} \cdot \eta \cdot EH^T, \; for \; c = 1, 2, \ldots, C$$

Skip-Gram模型

Skip-Gram模型是用一个词来预测它的上下文。



图： Skio-Gram模型

用一个词来预测上下文的C个词相当于预测一个词，重复C词，预测公式为：

$$p(w_{c,j} = W_{O,c}|w_I) = y_{c,j} = \frac{exp(u_{c,j})}{\sum_{j'=1}^{V} exp(u_{c,j'})}$$

其中：

- W_I是输入。
- W_O, c是需要预测的第C个输出。
- u_c, j是第C个词为j的概率

概率为：

$$u_{c,j} = u_j = v'^T_{w_j} \cdot h, forc = 1, 2, \ldots, C$$

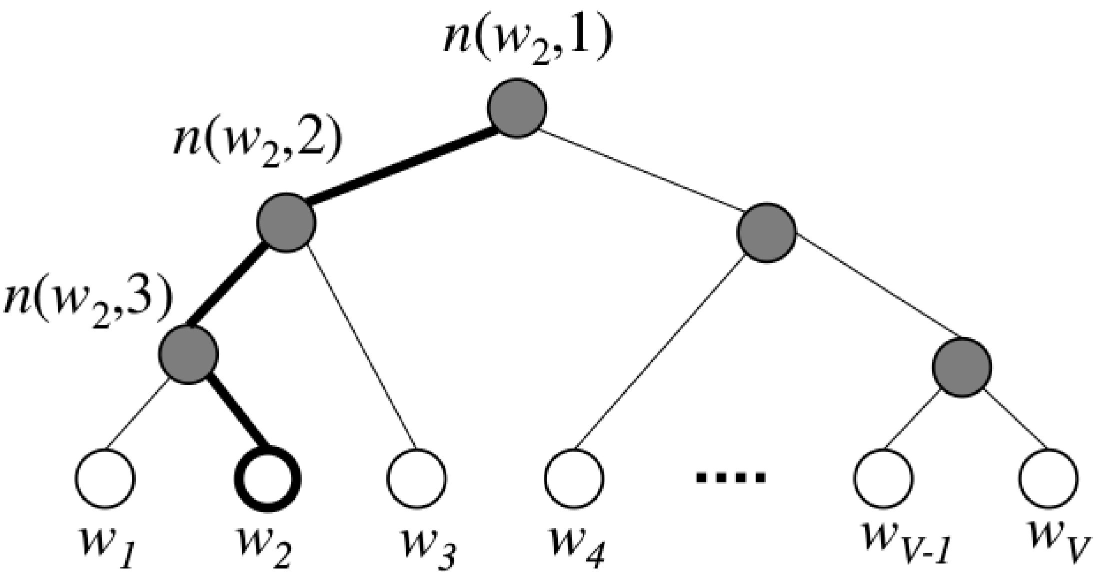
计算损失函数：

$$
\begin{aligned}
E &= -logp(w_{O,1}, w_{O,2}, \ldots, w_{O,C}|W_I) \\
&= -log\prod_{c=1}^{C}\frac{exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} exp(u_{j'})} \\
&= \sum_{c=1}^{C} u_{j_c^*} + C \cdot log\sum_{j'=1}^{V} exp(u_{j'})
\end{aligned}
$$

CBOW的梯度计算和Skip-Gram的梯度计算类似，Skip-Gram的E是多个次的损失求和。在实际计算中，可以把一次预测C个词分解成一次预测一个词然后重复C次。前者的C个词的forward是一次性计算出来的，然后用C个词的损失去计算梯度；后者是计算C次forward然后分别用对应的E计算backward。二者不完全相同，其中后者的计算效率比较低。

Hierarchical Softmax

传统softmax需要遍历整个词汇表，在词汇表较大的情况下效率低。Hierarchical softma通过二叉树结构来计算概率，将计算复杂度从O(V)降低到O(logV)。



词汇表中所有的词被组织成一颗二叉树（通常为哈夫曼树），每个词对应一个叶子节点。高频词的路径较短，低频词的路径较长。每一个非叶子节点都代表一个二分类器（通常为逻辑回归），用于决定向左（0）或向右（1）。

计算公式为：

$$p(w) = \prod_{j=1}^{L(w)-1} \sigma([\![n(w,j+1) = ch(n(w,j))]\!] \cdot v'^{T}_{n(w,j)} \cdot h)$$

其中：

- L(w)为道该叶子节点路径上节点的个数。
- n(w,j)为该路径上的第j个节点
- ch(n(w,j))为n(w,j)的左子树

另一种写法：

- 要计算词w的概率$P(w|w_I)$，从根节点出发走到对应w的叶子节点。
- 在每一个内部节点n计算向左或向右的概率：
  - 向左的概率：
    $$\sigma(\mathbf{v}_n^T \mathbf{h}) = \frac{1}{1 + \exp(-\mathbf{v}_n^T \mathbf{h})}$$
    $$1 - \sigma(\mathbf{v}_n^T \mathbf{h}) = \sigma(-\mathbf{v}_n^T \mathbf{h})$$
  - 向右的概率：
  - 其中：
    - h是当前上下文词的因曾表示（CBOW中是上下文词向量的平均，Skip-Gram是输入词向量）
    - v_n是节点n的向量表示，可训练参数。
- 词w的概率是所有路径上的概率的乘积：

$$P(w|w_I) = \prod_{n \in \mathrm{Path}(w)} P(\text{direction at } n|\mathbf{h})$$

损失函数计算：

$$E = -\log P(w|w_I)$$

$$E = -logp(w = w_O|w_I) \prod_{j=1}^{L(w)-1} \sigma([\![]\!] v'^{T}_{j} \cdot h)$$

计算梯度时仅更新节点向量v_n，而不是整个词汇表：

$$\frac{\partial E}{\partial v_j'} = \frac{\partial E}{\partial (v_j')^T h} \frac{\partial (v_j')^T h}{\partial v_j'}$$
$$= (\sigma((v_j')^T h) - t_j) \cdot h$$

更新节点向量：

$$v_j'^{(new)} \leftarrow v_j'^{(old)} - \eta(\sigma((v_j')^T h) - t_j) \cdot h, j = 1, 2, \ldots, L(w)$$

计算h的梯度：

$$\frac{\partial E}{\partial h} = \sum_{j=1}^{L(w)-1} \frac{\partial E}{\partial (v_j')^T h} \frac{\partial (v_j')^T h}{\partial h}$$
$$= \sum_{j=1}^{L(w)-1} (\sigma((v_j')^T h) - t_j) \cdot v_j' \equiv EH$$

Negative Sampling

Negative Sampling通过采样少量负样本来近似计算softmax的梯度，将计算效率降低到O(K)。

仅计算目标词和少量随机采样的负样本（如K=5个非目标词）。

目标函数改为二分类任务：

- 最大化目标词w_O的得分（正样本）。
- 最小化负样本w_N的得分（负样本）。

损失函数如下：

$$E = -\log\sigma(\mathbf{u}_{w_O}^T \mathbf{h}) - \sum_{i=1}^{K} \log\sigma(-\mathbf{u}_{w_i}^T \mathbf{h})$$

其中：

- σ(x)时sigmoid函数。
- 第一项：正样本w_O的得分$u_{w_O}^T h$尽可能大。
- 第二项：负样本w_N的得分$u_{w_i}^T h$尽可能小。

对正样本w_O梯度：

$$\frac{\partial E}{\partial \mathbf{u}_{w_O}} = \left[\sigma(\mathbf{u}_{w_O}^T \mathbf{h}) - 1\right]\mathbf{h}$$

对负样本的梯度：

$$\frac{\partial E}{\partial \mathbf{u}_{w_i}} = \sigma(\mathbf{u}_{w_i}^T \mathbf{h})\mathbf{h} \quad \text{（对每个负样本）}$$

对h的梯度:

$$\frac{\partial E}{\partial \mathbf{h}} = \left[\sigma(\mathbf{u}_{w_O}^T \mathbf{h}) - 1\right]\mathbf{u}_{w_O} + \sum_{i=1}^{K} \sigma(\mathbf{u}_{w_i}^T \mathbf{h})\mathbf{u}_{w_i}$$

负样本采样策略：

$$P(w) \propto \text{词频}(w)^{3/4}$$

正样本梯度更新：

$$\mathbf{u}_{w_O} \leftarrow \mathbf{u}_{w_O} - \eta\left[\sigma(\mathbf{u}_{w_O}^T \mathbf{h}) - 1\right]\mathbf{h}$$

负样本梯度更新：

$$\mathbf{u}_{w_i} \leftarrow \mathbf{u}_{w_i} - \eta\sigma(\mathbf{u}_{w_i}^T \mathbf{h})\mathbf{h}$$

Skip-Gram输入向量h更新:

$$\mathbf{h} \leftarrow \mathbf{h} - \eta\left(\left[\sigma(\mathbf{u}_{w_O}^T \mathbf{h}) - 1\right]\mathbf{u}_{w_O} + \sum_{i=1}^{K} \sigma(\mathbf{u}_{w_i}^T \mathbf{h})\mathbf{u}_{w_i}\right)$$