Graph Embedding 系列之 DeepWalk

Dorming 暮云瑟瑟 2020-03-10

1. 什么是 Graph Embedding？

> Embedding 拥有很多优点：得到的向量表达维度更低，并且表达了实体间内在的关系。下面以商品推荐为例，进一步解释。假设我们有千万级别的商品，我们通常使用 One-Hot 编码数字化地代表，则能够得到千万个向量，每个向量代表一种商品，并且每个向量都拥有千万级别的维度。向量只有一位是 1，其他位均为 0，信息量十分"稀疏"。此时，任何商品之间的距离都是一样的。并且，由于深度学习的特点以及工程方面的原因，深度学习并不善于处理稀疏特征的向量。相反， Embedding 之后，千万级别的维度可以缩小到自定义的维度大小（例如 1000）。变成向量之后的商品，可以直接通过计算向量相似度，寻找相似的商品，直接推荐给客户。

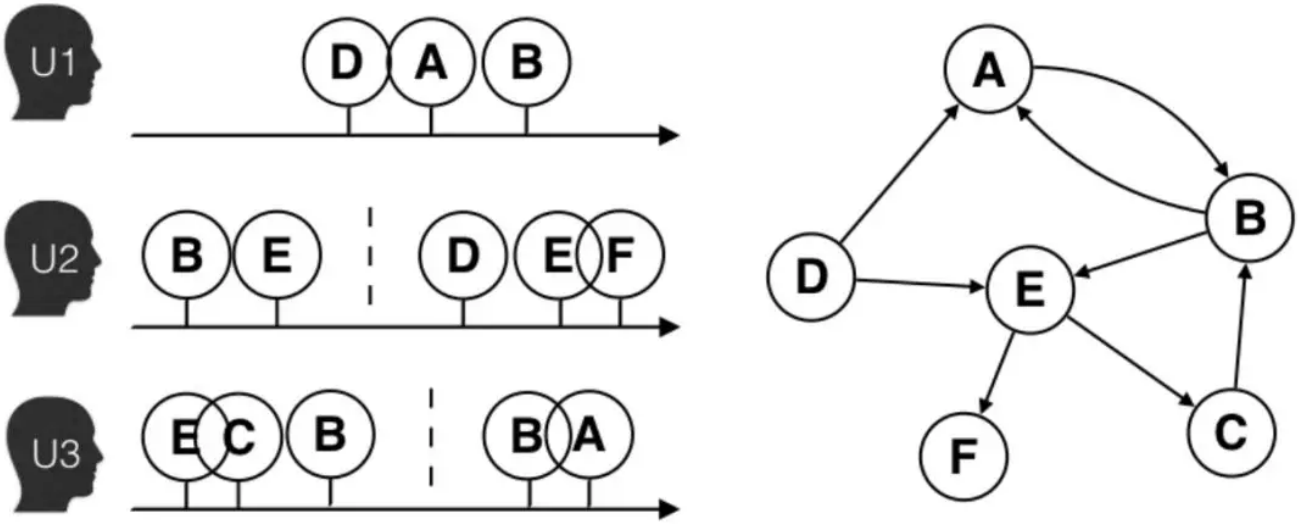
这些优点带来的好处是显著的，Embedding 主要的三个应用方向：

- 在深度学习网络中使用 Embedding 层，将高维稀疏特征向量转换成低维稠密特征向量，从而减少后续模型参数量，后续模型可以是深度学习模型，或者传统的机器学习模型；
- 作为预训练技术，直接使用别人训练好的 Embedding 向量，与其他特征向量一同输入后续模型进行训练，例如 Word2Vec；
- 通过计算用户和物品的 Embedding 相似度，Embedding 可以直接作为推荐系统或计算广告系统的召回层或者召回方法之一，例如 Youtube 推荐系统。

> Graph Embedding 是将图中的节点以低维、稠密、实值向量的形式进行表示，要求在原始图中相邻的的节点其在低维表达空间的表示也相近。图表示学习可以认为是图神经网络学习的基础任务。目前， Graph Embedding 已经是推荐系统、计算广告领域非常流行的做法，并且在实践后取得了非常不错的线上效果。

为什么能有这样的效果呢？

Word2Vec 通过序列（sequence）式的样本：句子，学习单词的真实含义。仿照 Word2Vec 思想而生的 Item2Vec 也通过商品的组合去生成商品的 Embedding，这里商品的组合也是序列式的，我们可以称他们为 Sequence Embedding 。然而，在更多场景下，数据对象之间更多以图/网络的结构呈现。例如下图，由淘宝用户行为数据生成的物品关系图（ Item Graph ）：

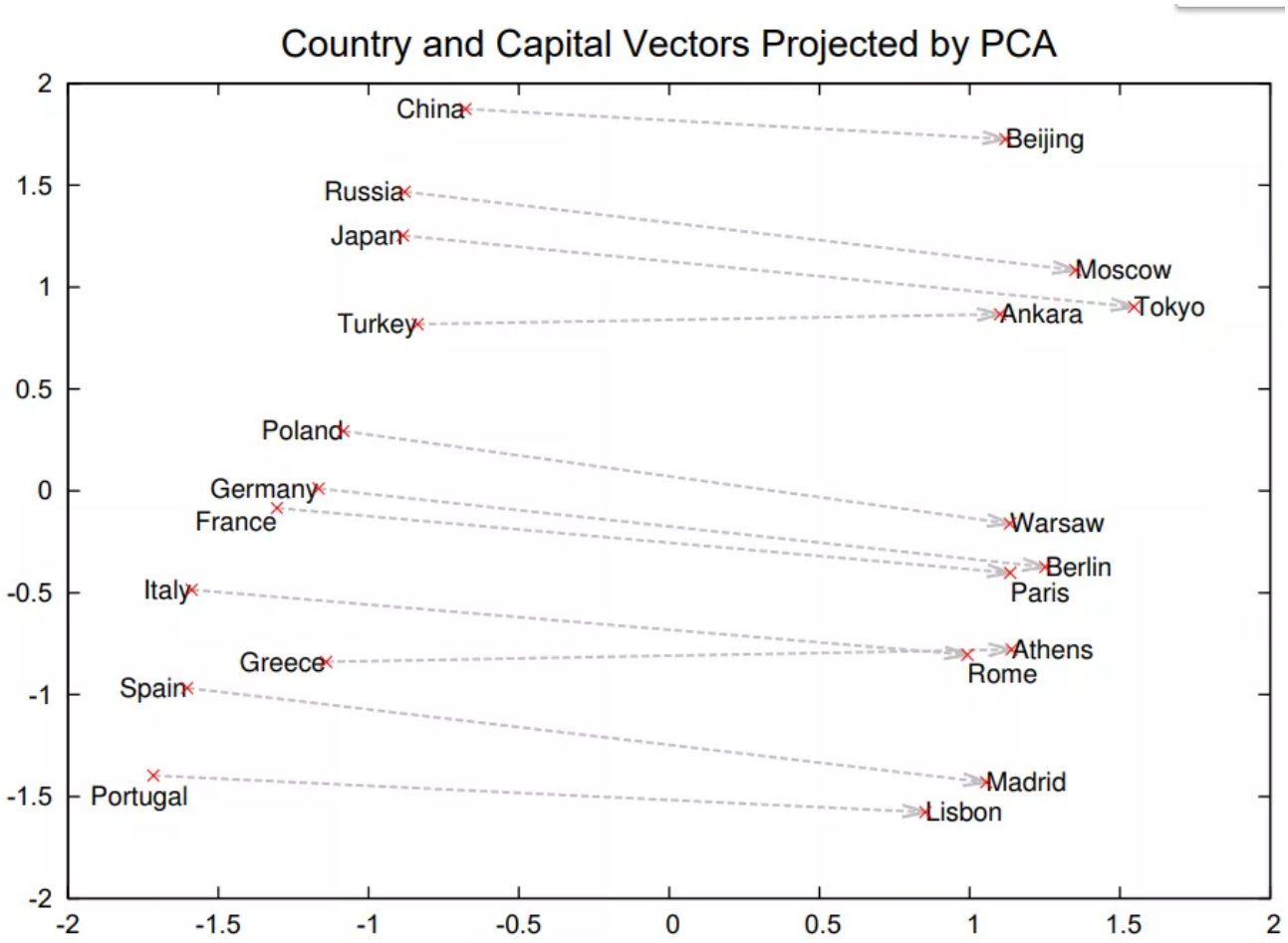


(a) Users' behavior sequences. (b) Item graph construction.

从上图的例子中，我们已经能触碰到一些 `Graph Embedding` 的本质。 `Graph Embedding` 之所以能好于 `Sequence Embedding` ，是因为 `Graph Embedding` 能够生成一些~~不存在的序列~~。例如，上图数据中没有这样的用户行为数据：**B-E-F**、**D-E-C**等等。但是在物品关系图中，我们有机会生成这样的序列。

2.Word2Vec

> 神经概率语言模型（Neural Probabilistic Language Model）中词的表示是向量形式、面向语义的。两个语义相似的词对应的向量也是相似的，具体反映在夹角或距离上。甚至一些语义相似的二元词组中的词语对应的向量做线性减法之后得到的向量依然是相似的。如下图所示：

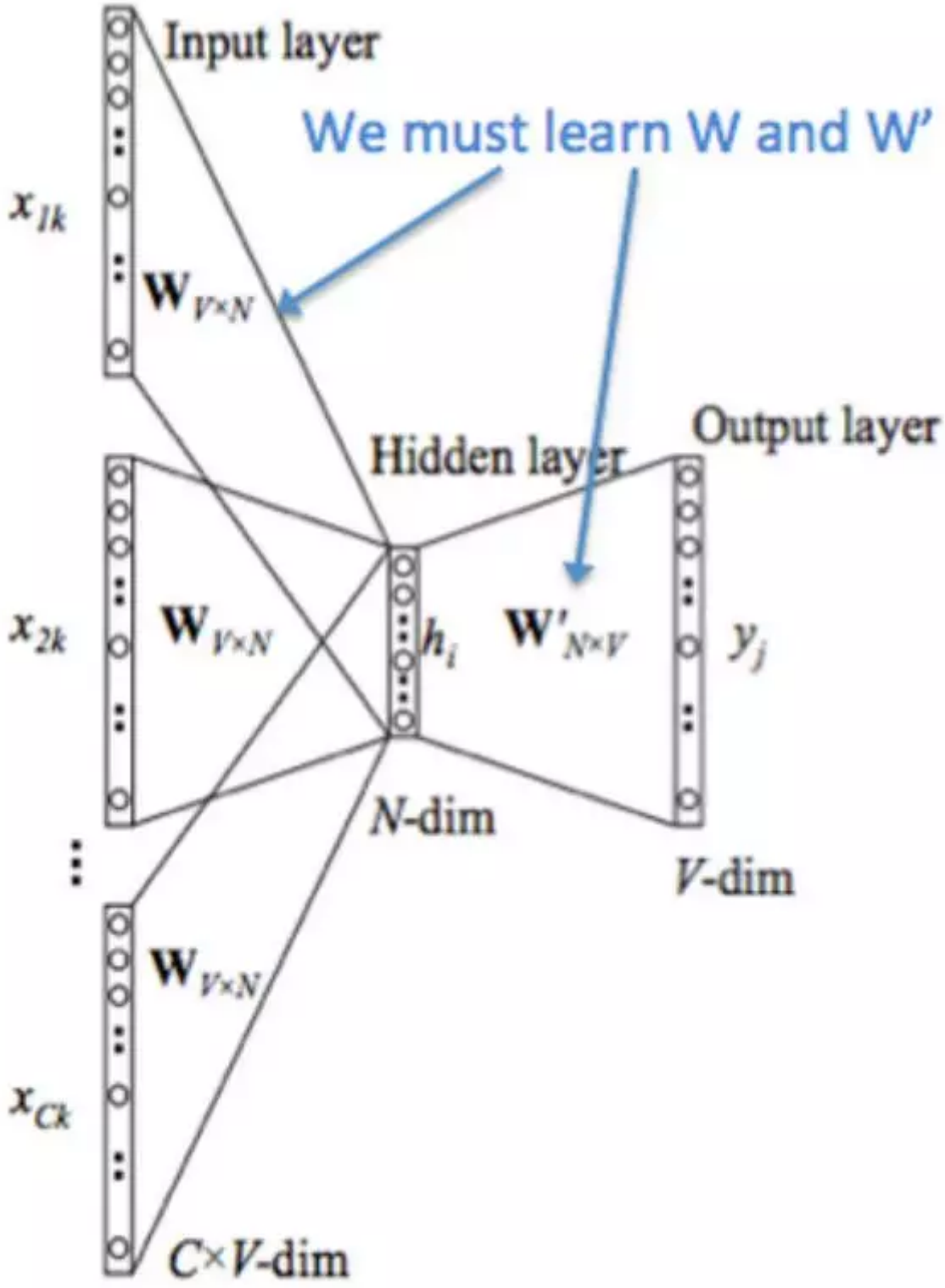


国家及其首都城市的1000维Skip-gram向量的二维PCA投影。该图展示了Skip-gram模型自动组织概念并隐式学习概念之间关系的能力，因为在训练期间，我们没有提供关于首都城市含义的任何监督信息

> word2vec 作为神经概率语言模型的输入，其本身其实是神经概率模型的副产品，是为了通过神经网络学习某个语言模型而产生的中间结果。具体来说，某个语言模型指的是 `CBOW` 和 `Skip-gram` 。具体学习过程会用到两个降低复杂度的近似方法—— `Hierarchical Softmax` 或 `Negative Sampling` 。两个模型乘以两种方法，一共有四种实现。为了更好地理解 `Deep Walk` 算法，word2vec 中有一些细节值得关注。首先，我们来阐述一下 `word2vec` 如何将 corpus 的 `one-hot向量` （模型的输入）转换成低维稠密词向量（模型的中间产物，更具体来说是输入权重矩阵）。

2.1 CBOW Model

CBOM 模型的全称为 Continuous Bag-of-Words Model。先上模型结构图：



让我们更详细地讨论上面的 CBOW 模型.首先,我们设置已知的参数。假设我们模型中的已知参数是由一个 `one-hot` 向量表示的句子。输入的 `one-hot` 向量我们表示为 $x^{(c)}$,输出向量表示为 $y^{(c)}$。因为在 cbow 模型中, 我们只有一个输出，因此我们可以把这个需要预测中心词的 `one-hot` 向量直接表示为 y。接下来我们来定义我们模型中未知的参数。

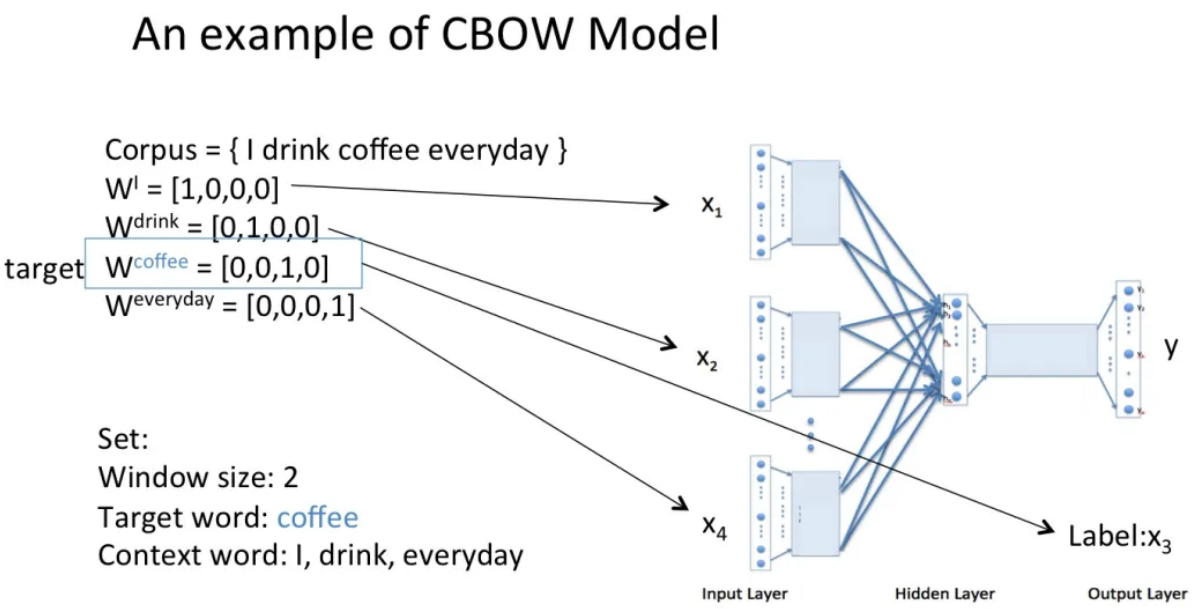
我们定义两个矩阵，$\mathcal{V} \in \mathbb{R}^{n\times|V|}$ 和 $\mathcal{U} \in \mathbb{R}^{|V|\times n}$。其中，$n$ 为任意大小，它定义了嵌入空间的大小.。$\mathcal{V}$ 是输入单词矩阵，$\mathcal{V}$ 的第 i 列是第 i 个单词 w_i 对应的 n 维大小的词嵌入向量。我们把这个 $n \times 1$ 大小的向量定义为 v_i。相似地，$\mathcal{U}$ 是输出单词矩阵，u_j 表示 $\mathcal{U}$ 的第 j 行，表示单词 w_j 的输出向量表示(output vector representation of word w_j)。

模型的流程可以简述如下：

- 对于滑动窗口为 m 的上下文，我们产生的 one-hot 词向量表示为: $x^{(c-m)},\ldots,x^{(c-1)},x^{(c+1)},\ldots,x^{(c+m)}$
- 通过输入单词矩阵，我们可以得到上下文的嵌入词向量表示为: $v_{c-m}=\mathcal{V}x^{(c-m)},v_{c-m+1}=\mathcal{V}x^{(c-m+1)},\ldots,v_{c+m}=\mathcal{V}x^{(c+m)}$
- 对这些得到的词嵌入向量进行平均得到：$\hat{v}=\frac{v_{c-m}+v_{c-m+1}+\ldots+v_{c+m}}{2m}$
- 产生一个得分向量: $z=\mathcal{U}\hat{v}$
- 把这个得分向量进行 softmax 归一化得到预测概率：$\hat{y}=softmax(z)$
- 把估计得到的 $\hat{y}$ 和真正的 one-hot 的概率向量 y 做交叉熵损失。

CBOW 模型流程举例[1]

假设我们现在的 Corpus 是这一个简单的只有四个单词的 document：{ I drink coffee everyday }我们选 coffee 作为中心词， window size 设为 2 也就是说，我们要根据单词 I，drink 和 everyday 来预测一个单词，并且我们希望这个单词是 coffee 。



An example of CBOW Model

Corpus = { I drink coffee everyday }

Initialize:

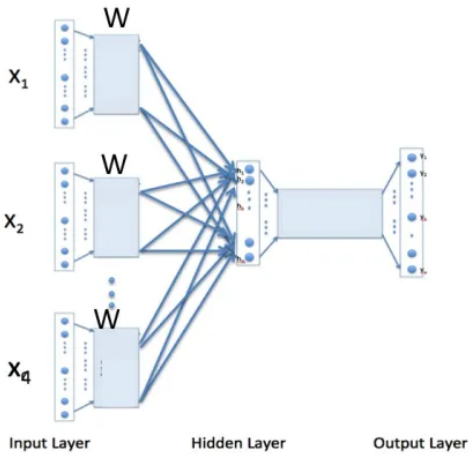
$$W = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 1 & 2 \\ -1 & 1 & 1 & 1 \end{bmatrix}$$

Ex:

$W^{drink} = [0,1,0,0]$

$$Wx_2 = v_2$$

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 1 & 2 \\ -1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$



Continuous bag-of-words (Mikolov et al., 2013)

An example of CBOW Model

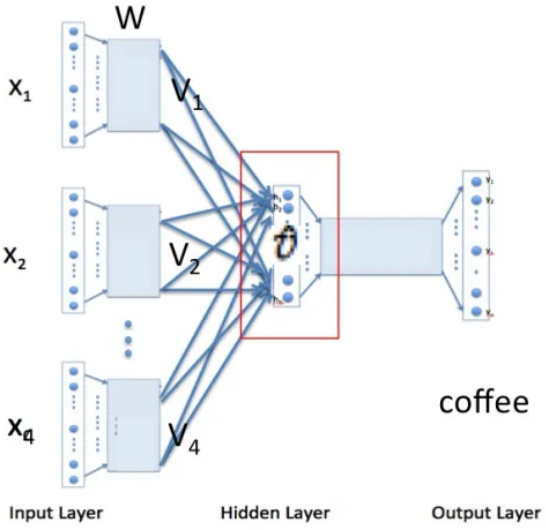
Corpus = { I drink coffee everyday }

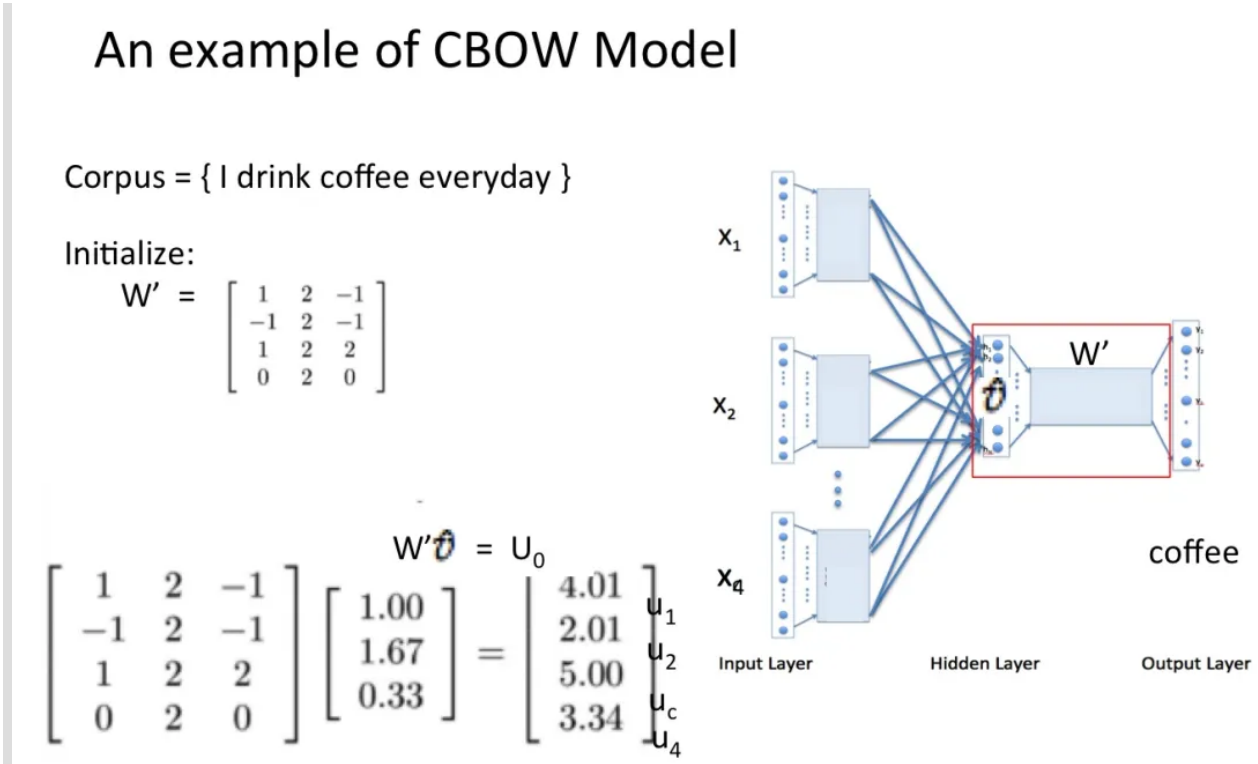
Initialize:

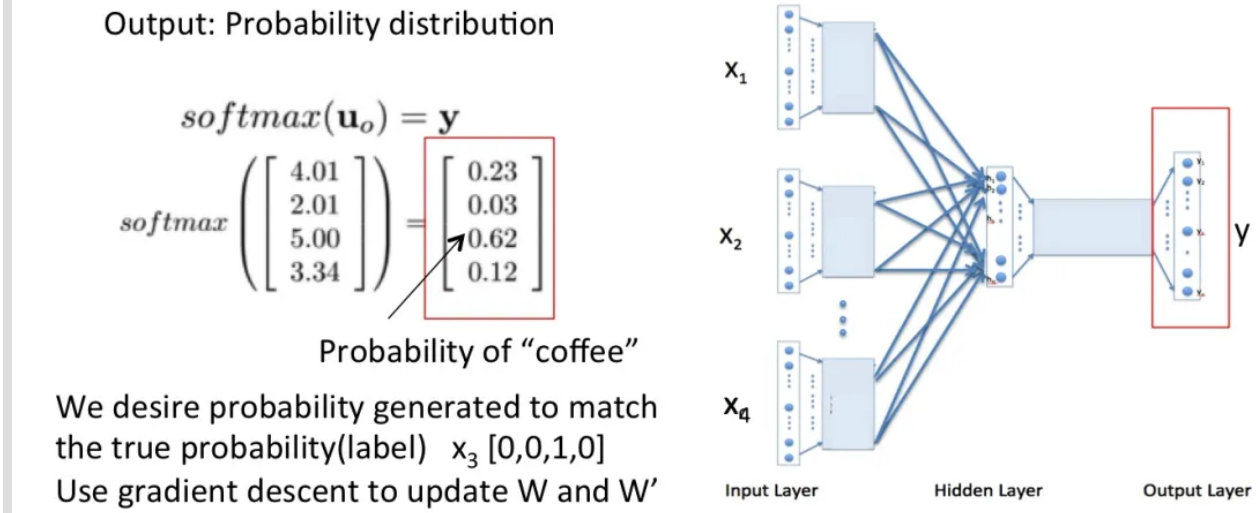
$$W = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 1 & 2 \\ -1 & 1 & 1 & 1 \end{bmatrix}$$

$$\frac{V_1 + V_2 + V_4}{3} = \hat{v}$$

$$\frac{1}{3}\left(\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1.67 \\ 0.33 \end{bmatrix}$$

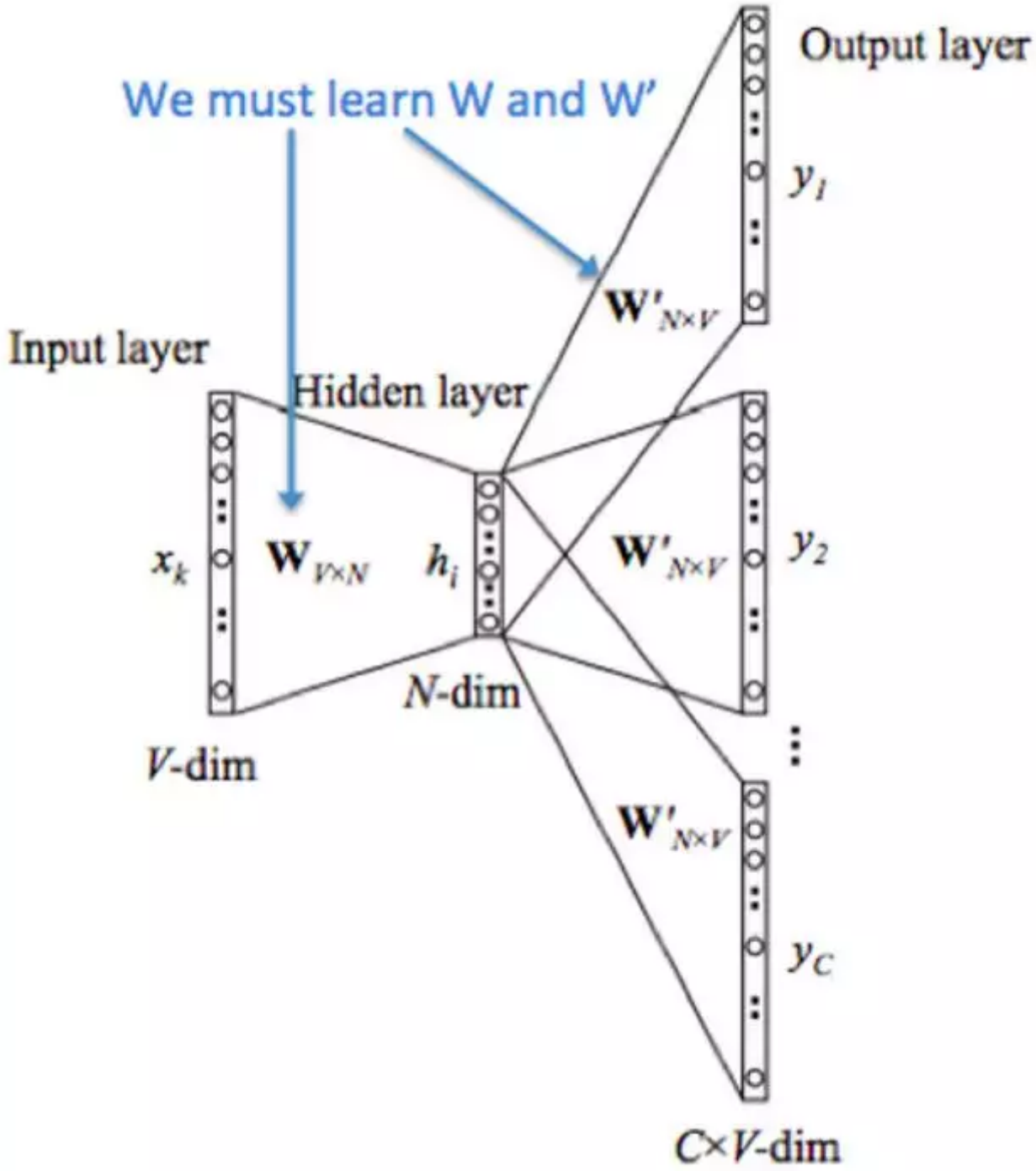






假设我们此时得到的概率分布已经达到了设定的迭代次数，那么现在我们训练出来的*look up table*应该为矩阵 W。即，任何一个单词的*one-hot*表示乘以这个矩阵都将得到自己的*word embedding*。

2.2 Skip-Gram Model



模型对于给定一个中心词，能够预测它周围的单词。如对于这样的一个句子{ `The` , `cat` , `jumped` , `over` , `the` , `puddle` }, 对于给定的中心词 `jumped` , Skip-Gram 能够预测或者生成其周围的单词 `the` , `cat` , `over` , `the` , `paddle` 。保持和 CBOW 相似的定义，则 Skip-Gram 的算法流程可以描述如下：

- 产生一个 `one-hot` 的输入向量 x，（上下文只有一个单词）
- 通过输入单词矩阵得到词嵌入向量表示：$v_c = \mathcal{V}x$
- 因为只有一个输入的单词，所以没有 averaging，只需要设置 $\hat{v}_c = v_c$

- 使用输出单词矩阵 $u = \mathcal{U}v_c$ 生成 $2m$ 个得分向量，$u_{c-m}, \ldots, u_{c-1}, u_{c+1}, \ldots, u_{c+m}$
- 把每一个得分向量都转化为概率，$y = softmax(u)$
- 把估计得到 $2m$ 个概率向量 $\hat{y}^{(c-m)}, \ldots, \hat{y}^{(c-1)}, y^{(c+1)}, \ldots, \hat{y}^{(c+m)}$ 和真正的 `one-hot` 的概率向量 $y^{(c-m)}, \ldots, y^{(c-1)}, y^{(c+1)}, \ldots, y^{(c+m)}$ 做交叉熵损失。

我们假设在给定中心词的条件下，所有的输出词都是完全独立的(a strong (naive) conditional independence assumption)。在此条件下， `Skip-Gram` 的目标函数可以写成如下的形式：

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

利用这个目标函数，我们可以计算出相关于未知参数的梯度，并在每次迭代中通过随机梯度下降算法更新它们。

2.3 Negative Sampling

观察上面的目标函数的形式，注意到 $|V|$ 是非常大的，会带来很高的时间复杂度($\mathcal{O}((|V|))$)。因此，一个简单的想法是我们想要能够近似这一项。对于每个训练步骤，我们不需要遍历整个词汇表，我们只需要采样几个负例! 我们从一个概率分布与其与词汇表中词频的顺序相匹配的噪声分布 $P_n(w)$ 噪声分布中采样。考虑一个中心词和其上下文的单词对 (w, c)，我们用 $P(D = 1|w, c)$ 来表示这个单词对 (w, c) 来自语料库的概率，相应地，$P(D = 0|w, c)$ 表示这个单词对 (w, c) 不是来自语料库的概率。首先使用 sigmoid 函数实例化，

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

现在，我们来构建了一个新的目标函数，该函数试图最大化一个单词和上下文在语料库数据中出现的概率(如果它确实是)，以及最大化一个单词和上下文不在语料库数据中出现的概率(如果它确实不是)。我们对这两个概率采用了一种简单的极大似然方法。这里我们用 θ 表示模型的参数，在我们的模型中它实际是 $\mathcal{V}$ 和 $\mathcal{U}$。

$$
\begin{aligned}
\theta &= \arg\max_{\theta} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta) \\
&= \arg\max_{\theta} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta)) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta)) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1-\frac{1}{1+\exp(-u_w^T v_c)}) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(u_w^T v_c)})
\end{aligned}
$$

注意到 $\tilde{D}$ 是*假的语料库*(false or negative corpus)。不自然的句子发生的可能性很低。我们可以从词库(word bank)中随机采样生成这种 $\tilde{D}$。我们新的目标函数可以写为：

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

在上述的公式中，$\{\tilde{u}_k|k=1,\dots,K\}$ 是从 $P_n(w)$ 中采样得到的。任何采样算法都应该保证频次越高的样本越容易被采样出来。基本的思路是对于长度为 1 的线段，根据词语的词频将其公平地分配给每个词语：

$$
len(w) = \frac{conuter(w)}{\sum_{u\in\mathcal{D}} conuter(u)}
$$

counter 表示单词的词频。具体实现时， `word2vec` 对词频取了 0.75 次幂， 即

$$
len(w) = \frac{[counter(w)]^{0.75}}{\sum_{u\in\mathcal{D}} [counter(u)]^{0.75}}
$$

这个幂实际上是一种“平滑”策略，能够让低频词多一些出场机会，高频词贡献一些出场机会，劫富济贫。

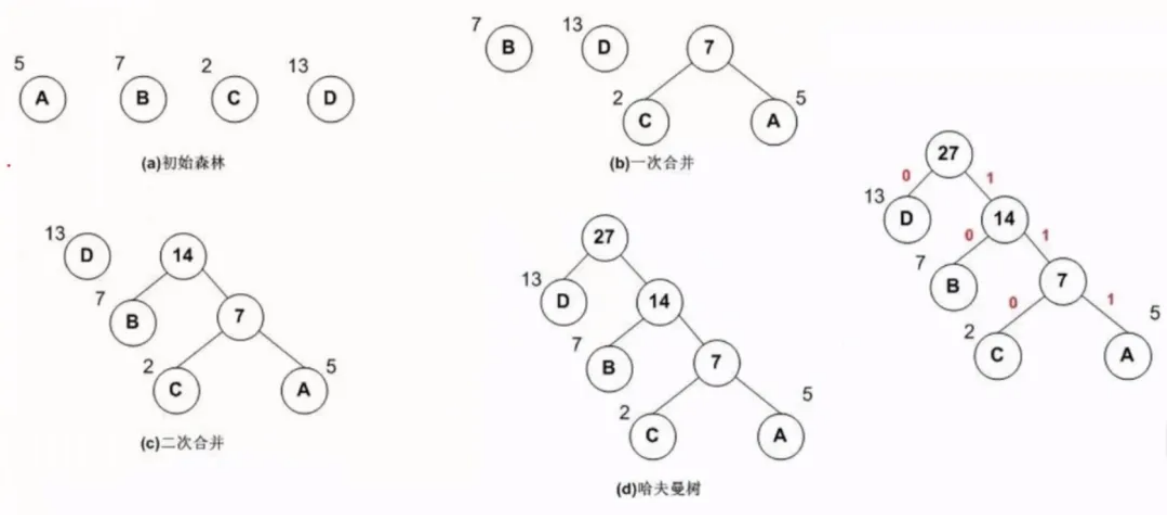
2.4 Hierarchical Softmax

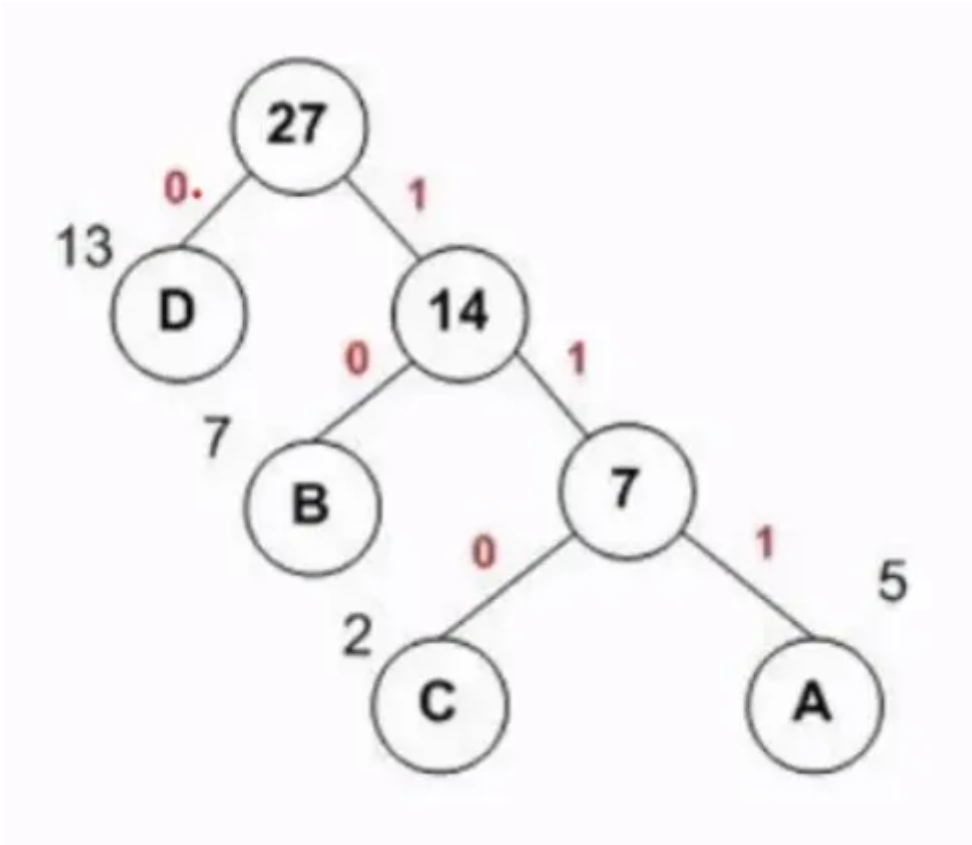
假设 `CBOW` 的优化目标可以构造如下:

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

显然根据上式，计算配分函数(归一化因子,红色部分)是很昂贵的，因此我们将使用 `Hierarchical Softmax` 对条件概率进行因式分解。首先,我们需要构造一个二叉树，非叶子节点相当于

一个神经元（感知机，我认为逻辑斯谛回归就是感知机的输出,代入 $\sigma(x)=\frac{1}{1+e^{-x}}$。），二分类决策输出 1 或 0，分别代表向下左转或向下右转；每个叶子节点代表语料库中的一个词语，于是每个词语都可以被 01 唯一地编码，并且其编码序列对应一个事件序列，于是我们可以计算条件概率 $p(u_c|\hat{v})$。

> 哈夫曼树(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。 word2vec 训练的时候按照词频将每个词语 Huffman 编码，由于 Huffman 编码中词频越高的词语对应的编码越短。所以越高频的词语在 Hierarchical Softmax 过程中经过的二分类节点就越少，整体计算量就更少了。举一个栗子: 有 A B C D 四个词，数字表示词频，构造过程如下:
>
> 
>
>
> 哈夫曼树编码为左子树为 0，右子树为 1，



那么 D 编码为 0 ， B 编码为 10 ， C 编码为 110 ， A 编码为 111 。我们将每一个单词都分配到二叉树的叶子上，将预测问题转化为在层次树中最大化特定路径的概率。在开始计算之前，我们首先定义一些符号:

> - p^c：从从根结点出发到达中心词 w_c 对应叶子结点的路径
> - l^c：路径中包含的节点个数
> - $p_1^c, p_2^c, \ldots, p_{l^c}^c$：路径 p^c 中的各个节点
> - $\{d_2^c, d_3^c, \ldots, d_{l^c}^c\}$：中心词 w_c 的编码， d_j^c 表示路径中第 j 个节点对应的编码（根节点无编码）
> - $\theta_1^c, \theta_2^c, \ldots, \theta_{l^c-1}^c \in \mathbb{R}^m$：路径中非叶节点对应的参数

于是可以给出中心词 w_c 的条件概率:

$$
\begin{aligned}
&p(w_c|w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}) \\
&= p(w_c|\hat{v}) \\
&= \prod_{j=2}^{l^c} p(d_j^c|\hat{v}, \theta_{j-1}^c)
\end{aligned}
$$

其中， $\hat{v}$ 表示投影层的输出。这是个简单明了的式子，从根节点到叶节点经过了 $l^c - 1$ 个节点，编码从下标 2 开始（根节点无编码），对应的参数下标从 1 开始（根节点为 1）。

其中，每一项是一个 Logistic Regression ：

$$
p(d_j^c|\hat{v}, \theta_{j-1}^c) = \begin{cases} \sigma(\hat{v}^T\theta_{j-1}^c), & d_j^c = 0 \\ 1 - \sigma(\hat{v}^T\theta_{j-1}^c), & d_j^c = 1 \end{cases}
$$

考虑到 d_j^c 只有 0 和 1 两种取值，我们可以用指数形式方便地将其写到一起:

$$
p(d_j^c|\hat{v}, \theta_{j-1}^c) = [\sigma(\hat{v}^T\theta_{j-1}^c)]^{1-d_j^c} \cdot [1 - \sigma(\hat{v}^T\theta_{j-1}^c)]^{d_j^c}
$$

我们的目标函数取对数似然:

$$
\mathcal{L} = \sum_{w_c \in \mathcal{D}} \log p(w_c|\hat{v})
$$

将 $p(w_c|\hat{v})$ 代入上式，有:

$$
\begin{aligned}
\mathcal{L} &= \sum_{w_c \in \mathcal{D}} \log \prod_{j=2}^{l^c} \{[\sigma(\hat{v}^T\theta_{j-1}^c)]^{1-d_j^c} \cdot [1 - \sigma(\hat{v}^T\theta_{j-1}^c)]^{d_j^c}\} \\
&= \sum_{w_c \in \mathcal{D}} \sum_{j=2}^{l^c} \{(1 - d_j^c) \cdot \log[\sigma(\hat{v}^T\theta_{j-1}^c)] + d_j^c \cdot \log[1 - \sigma(\hat{v}^T\theta_{j-1}^c)]\}
\end{aligned}
$$

我们把每一项简记为：

$$\mathcal{L}(w,j)=(1-d_j^c)\cdot\log[\sigma(\hat{v}^T\theta_{j-1}^c)]+d_j^c\cdot\log[1-\sigma(\hat{v}^T\theta_{j-1}^c)]$$

最大化对数似然函数需要分别最大化每一项即可（这应该是一种近似，最大化某一项不一定使整体增大，具体收敛的证明还不清楚）。怎么最大化每一项呢？先求函数对每个变量的偏导数，对每一个样本，代入偏导数表达式得到函数在该维度的增长梯度，然后让对应参数加上这个梯度，函数在这个维度上就增长了。

每一项有两个参数，一个是每个节点的参数 θ_{j-1}^c ，另一个是输出层的输入 $\hat{v}$ ，我们分别对其求偏导数：

$$\frac{\partial\mathcal{L}(w,j)}{\partial\theta_{j-1}^c}=\frac{\partial}{\partial\theta_{j-1}^c}\{(1-d_j^c)\cdot\log[\sigma(\hat{v}^T\theta_{j-1}^c)]+d_j^c\cdot\log[1-\sigma(\hat{v}^T\theta_{j-1}^c)]\}$$

因为 `sigmoid` 函数的导数有个很棒的形式：

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

于是代入上上式得到：

$$(1-d_j^c)[1-\sigma(\hat{v}^T\theta_{j-1}^c)]\hat{v}-d_j^c\sigma(\hat{v}^T\theta_{j-1}^c)\hat{v}$$

合并同类项得到:

$$\left[1-d_j^c-\sigma(\hat{v}^T\theta_{j-1}^c)\right]\hat{v}$$

于是 θ_{j-1}^c 更新表达式为：

$$\theta_{j-1}^c:=\theta_{j-1}^c+\eta\left[1-d_j^c-\sigma(\hat{v}^T\theta_{j-1}^c)\right]\hat{v}$$

其中，η 是学习率。同样地，对 $\hat{v}$ 求偏导数，注意到公式 10 中 $\hat{v}$ 和 θ_{j-1}^c 是对称的，所有直接将 θ_{j-1}^c 的偏导数中的 θ_{j-1}^c 替换为 $\hat{v}$ ，得到关于 θ_{j-1}^c 的偏导数：

$$\frac{\mathcal{L}(w,j)}{\partial\hat{v}}=\left[1-d_j^c-\sigma(\hat{v}^T\theta_{j-1}^c)\right]\theta_{j-1}^c$$

于是 $\hat{v}$ 的更新表达式也得到了。

不过 $\hat{v}$ 是上下文的词向量的和的平均(average)，不是上下文单个词的词向量。怎么把这个更新量应用到单个词的词向量上去呢？ `word2vec` 采取的是直接将 $\hat{v}$ 的更新量整个应用到每个单词的词向量上去：

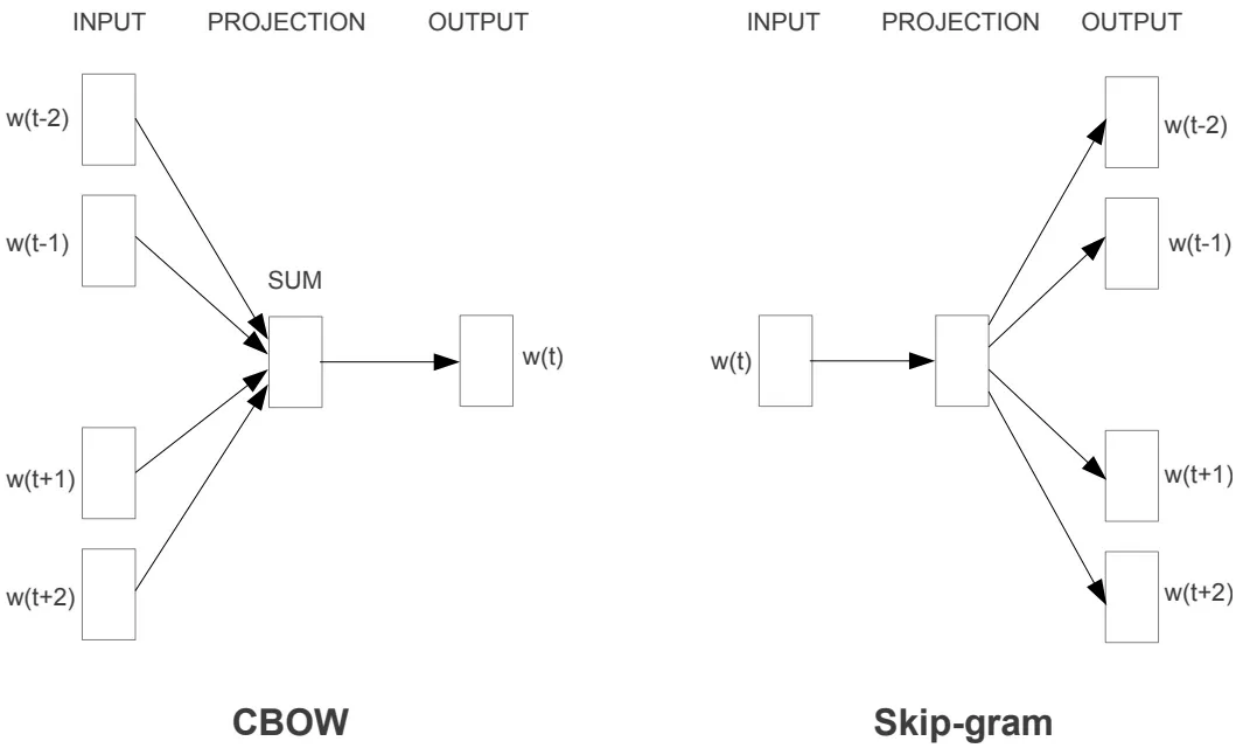
$$v(\tilde{w}) := v(\tilde{w}) + \eta \sum_{j=2}^{l^c} \frac{\partial \mathcal{L}(w, j)}{\partial \hat{v}}$$

$$\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \ldots + v_{c+m}}{2m}$$

$$\tilde{w} \in Context(w)$$

其中，v($\tilde{w}$)代表上下文中某一个单词的词向量。

`Skip-gram` 只是逆转了 `CBOW` 的 因果关系 而已，即**已知当前词语，预测上下文**。二者的关系可以用下图表示:



上图与 `CBOW` 的两个不同在于

> - 输入层不再是多个词向量，而是一个词向量
> - 投影层其实什么事情都没干，直接将输入层的词向量传递给输出层

回顾 `Skip-Gram` 的优化目标

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

于是语言模型的概率可以写为:

$$
\prod_{j=0,j\neq m}^{2m} P(u_{c-m+j}|v_c) = \prod_{u\in Context(w)} p(u|w)
$$

其中，u 表示 w_c 的上下文中的一个词语。注意这是一个词袋模型，所以每个 u 是无序的，或者说，互相独立的。在 `Hierarchical Softmax` 思想下，每个 u 都可以编码为一条 `01` 路径：

$$
p(u|w) = \prod_{j=2}^{l^c} p(d_j^u|v_c;\theta_{j-1}^u)
$$

类似地，每一项都是如下简写：

$$
p(d_j^u|v_c;\theta_{j-1}^u) = \left[\sigma(v_c^T\theta_{j-1}^u)\right]^{1-d_j^u}\cdot\left[1-\sigma(v_c^T\theta_{j-1}^u)\right]^{d_j^u}
$$

代入得到目标函数 $\mathcal{L}$：

$$
\sum_{w_c\in\mathcal{D}}\log\prod_{u\in Context(w)}\prod_{j=2}^{l^u}\{\left[\sigma(v_c^T\theta_{j-1}^u)\right]^{1-d_j^u}\cdot\left[1-\sigma(v_c^T\theta_{j-1}^u)\right]^{d_j^u}\}
$$

$$
\sum_{w_c\in\mathcal{D}}\log\sum_{u\in Context(w)}\sum_{j=2}^{l^u}\{(1-d_j^u)\cdot\log\left[\sigma(v_c^T\theta_{j-1}^u)\right]+d_j^u\cdot\left[1-\sigma(v_c^T\theta_{j-1}^u)\right]\}
$$

类似于 `CBOW` 的做法，将每一项简记为：

$$
\mathcal{L}(w_c,u,\theta_{j-1}^u) = (1-d_j^u)\cdot\log\left[\sigma(v_c^T\theta_{j-1}^u)\right]+d_j^u\cdot\left[1-\sigma(v_c^T\theta_{j-1}^u)\right]
$$

然上式对比 `CBOW` 多了一个 u，但给定训练实例（一中心词 w_c 和它的上下文 u），u 也是固定的。所以上式其实依然只有两个变量 v_c 和 θ_{j-1}^u，对 θ_{j-1}^u 求偏导数得到:

$$
\frac{\mathcal{L}(w_c,u,\theta_{j-1}^u)}{\partial\theta_{j-1}^u} = \left[1-d_j^u-\sigma(v_c^T\theta_{j-1}^u)\right]v_c
$$

于是得到 θ_{j-1}^u 的更新表达式:

$$
\theta_{j-1}^u := \theta_{j-1}^u + \eta\left[1-d_j^u-\sigma(v_c^T\theta_{j-1}^u)\right]v_c
$$

同理利用对称性得到对 v_c 的偏导数:

$$\frac{\mathcal{L}(w_c,u,\theta_{j-1}^u)}{\partial v_c}=\left[1-d_j^u-\sigma(v_c^T\theta_{j-1}^u)\right]\theta_{j-1}^u$$

于是得到 v_c 的更新表达式:

$$\begin{aligned}v(c)&:=v(c)+\eta\sum_{u\in Context(w)}\sum_{j=1}^{l^u}\frac{\partial\mathcal{L}(w_c,u,\theta_{j-1}^u)}{\partial v_c}\\&=\eta\sum_{u\in Context(w)}\sum_{j=1}^{l^u}\left[1-d_j^u-\sigma(v_c^T\theta_{j-1}^u)\right]\theta_{j-1}^u\end{aligned}$$

训练的伪代码如下:

$\mathbf{e}=0$

$\mathbf{FOR}\ u\in Context(w)\ \ \mathbf{DO}$

{

  $\mathbf{FOR}\ j=2:l^u\ \ \mathbf{DO}$

  {

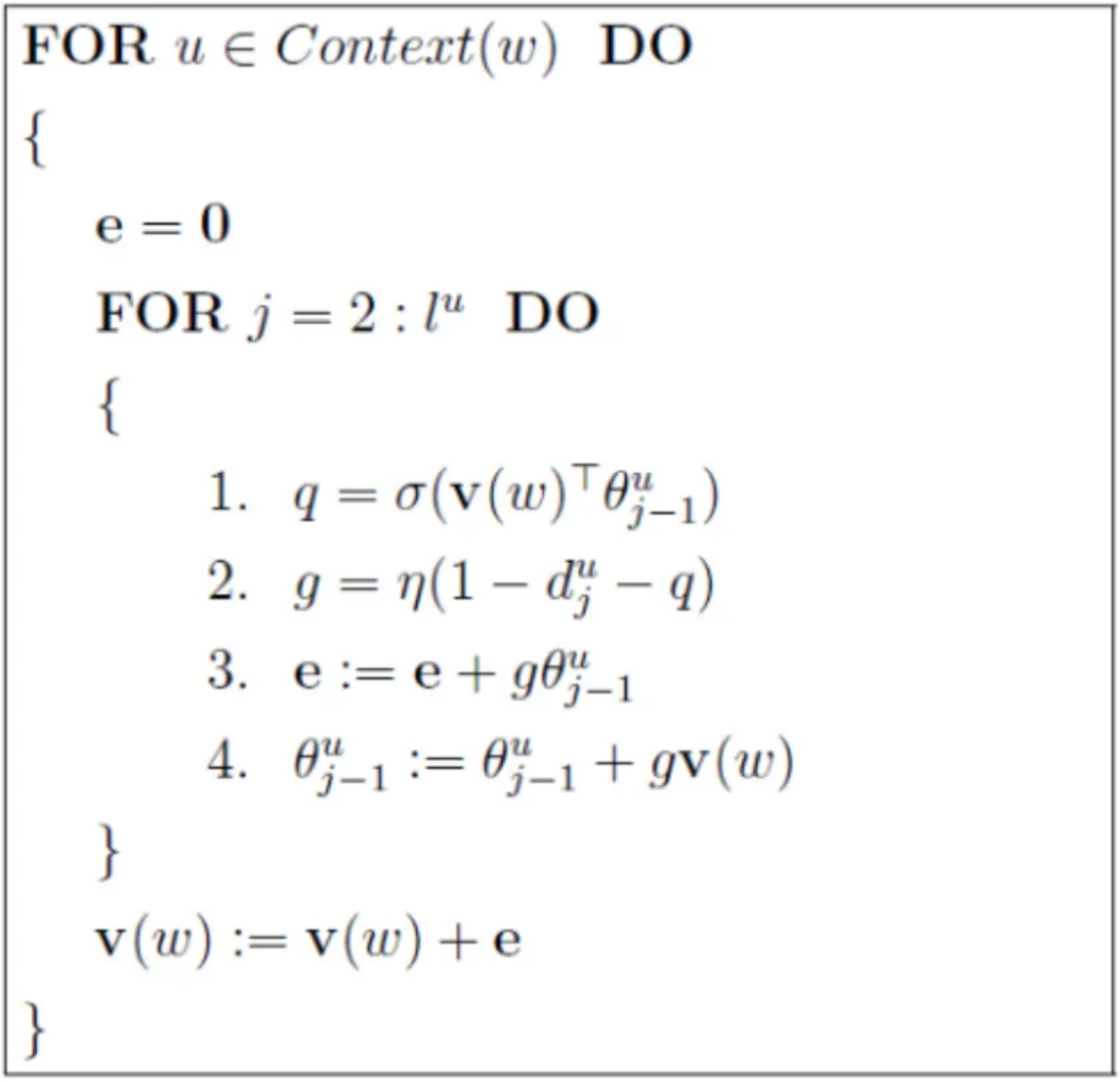
    1. $q=\sigma(\mathbf{v}(w)^\top\theta_{j-1}^u)$
    2. $g=\eta(1-d_j^u-q)$
    3. $\mathbf{e}:=\mathbf{e}+g\theta_{j-1}^u$
    4. $\theta_{j-1}^u:=\theta_{j-1}^u+g\mathbf{v}(w)$

  }

}

$\mathbf{v}(w):=\mathbf{v}(w)+\mathbf{e}$

`word2vec` 源码中并没有等 θ_{j-1}^u 更新完再更新 v_c，而是即时地更新，换句话说,也就是每一个上下文单词完成更新时，都对中心词完成一次更新:



3. DeepWalk

> 本文提出了一种网络嵌入的方法叫 DeepWalk ，它的输入是一张图或者网络，输出为网络中顶点的向量表示。 DeepWalk 通过截断随机游走(truncated random walk)学习出一个网络的社会表示(social representation)，在网络标注顶点很少的情况也能得到比较好的效果。并且该方法还具有可扩展的优点，能够适应网络的变化。简而言之， DeepWalk 就是 Random Walk 与 Skip-gram 的组合。

DeepWalk 只需要一个语料库和一个词汇表 $\mathcal{V}$。DeepWalk 将一组短截短的随机游走(random walks)视为它自己的语料库,把图顶点作为它自己的词汇表($\mathcal{V} = V$)。虽然在训练前知道图顶点 V 和随机游走中顶点的频率分布是有益的，但是不是必需的。

该算法由两个主要部分组成: 随机游走生成器 和一个 更新程序 。随机游走生成器从一个图 G 中，均匀地采样一个随机顶点 v_i 作为随机游走 $\mathcal{W}_{v_i}$ 的根。

Random Walk 从所访问的最后一个顶点的邻居中均匀采样，直到达到最大长度。虽然我们在实验中设置了随机游走的长度是固定的，但是对于相同长度的随机走没有限制。这些游走可能会有重启(restarts)(即传送回到它们根的概率)，但是我们的初步结果没有显示使用重启有任何好处。在实践中,我们的实现指定随机游走的数量为 γ，游走的长度为 t。

Algorithm 1 DeepWalk(G, w, d, γ, t)

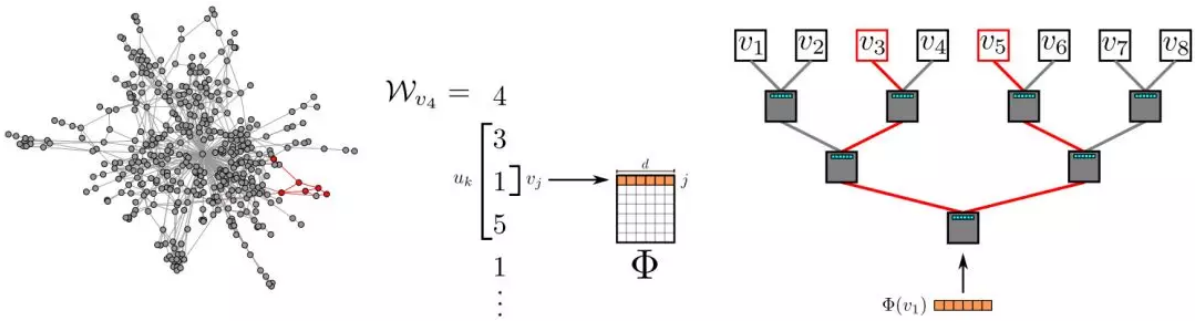
Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

1: Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
2: Build a binary Tree T from V
3: **for** $i = 0$ to γ **do**
4: $\quad \mathcal{O} = \text{Shuffle}(V)$
5: $\quad$ **for each** $v_i \in \mathcal{O}$ **do**
6: $\quad\quad \mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
7: $\quad\quad \text{SkipGram}(\Phi, \mathcal{W}_{v_i}, w)$
8: $\quad$ **end for**
9: **end for**

算法 1 的 3-9 行展示了我们算法的核心。外层循环指定了随机游走的次数 γ，我们认为每次迭代都是对数据进行一次“传递”，并在此传递过程中对每个顶点进行一次采样。在每次遍历的开始，我们都会生成一个随机的遍历顶点的顺序。这不是严格要求，但众所周知，这有助于加快收敛的随机梯度下降。

在内部循环中，我们遍历图上的所有顶点。对于每一个顶点，我们产生一个随机游走 $|\mathcal{W}_{v_i}| = t$ ($\mathcal{W}_{v_i}$ 表示以 v_i 为根节点生成的一条随机游走路径)，然后利用这个随机游走更新我么的表示。我们使用 Skip-Gram 算法来更新我们的表示矩阵 Φ。

一个 Deep WALK 的概览如下。我们在随机游走 $\mathcal{W}_{v_4}$ 中取一个 $2w+1$ 的滑动窗口，把中心顶点 v_1 映射成它的表示 $\Phi(v_1)$。 Hierarchical Softmax 将 $Pr(v_3|\Phi(v_1)$ 和 $Pr(v_5|\Phi(v_1)$ 根据二叉树上从根节点到达 v_3 和 v_5 的路径因子化分解成一个序列的概率分布。通过最大化 v_1 和它的上下文 $\{v_3, v_5\}$ 的共现概率更新 Φ。



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

参考文献

[1] CBOW 模型: https://www.zhihu.com/question/44832436/answer/266068967

[2] 训练的伪代码: https://www.hankcs.com/nlp/word2vec.html#respond

[3] DeepWalk: http://www.perozzi.net/publications/14_kdd_deepwalk.pdf

End

End