Word2Vec详解

原创 越前浩波 浩波的笔记 9月1日

word2vec可以在百万数量级的词典和上亿的数据集上进行高效地训练；并且，该工具得到的训练结果——词向量（word embedding），可以很好地度量词与词之间的相似性。随着深度学习（Deep Learning）在自然语言处理中应用的普及，很多人误以为word2vec是一种深度学习算法。其实word2vec算法的背后是一个浅层神经网络。

另外需要强调的一点是，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。很多人以为word2vec指的是一个算法或模型，这也是一种谬误。接下来，本文将从统计语言模型出发，尽可能详细地介绍word2vec工具背后的算法模型的来龙去脉。

Statistical Language Model

在深入word2vec算法的细节之前，我们首先回顾一下自然语言处理中的一个基本问题：如何计算一段文本序列在某种语言下出现的概率？之所为称其为一个基本问题，是因为它在很多NLP任务中都扮演着重要的角色。例如，在机器翻译的问题中，如果我们知道了目标语言中每句话的概率，就可以从候选集合中挑选出最合理的句子做为翻译结果返回。

统计语言模型给出了这一类问题的一个基本解决框架。对于一段文本序列

$$S = w_1, w_2, \ldots, w_T$$

它的概率可以表示为：

$$P(S) = P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t|w_1, w_2, ..$$

即将序列的联合概率转化为一系列条件概率的乘积。问题变成了如何去预测这些给定previous words下的条件概率：

$$p(w_t|w_1, w_2, \ldots, w_{t-1})$$

由于其巨大的参数空间，这样一个原始的模型在实际中并没有什么用。我们更多的是采用其简化版本——Ngram模型：

$$p(w_t|w_1,w_2,\ldots,w_{t-1}) \approx p(w_t|w_{t-n+1},\ldots,w_{t-1})$$

常见的如bigram模型 (N=2) 和trigram模型 (N=3) 。事实上，由于模型复杂度和预测精度的限制，我们很少会考虑N>3的模型。我们可以用最大似然法去求解Ngram模型的参数——等价于去统计每个Ngram的条件词频。型进一步发展出了back-off trigram模型 (用低阶的bigram和unigram代替零概率的trigram) 和interpolated trigram模型（将条件概率表示为unigram、bigram、trigram三者的线性函数）。

Distributed Representation

不过，Ngram模型仍有其局限性。首先，由于参数空间的爆炸式增长，它无法处理更长程的context（N>3）。其次，它没有考虑词与词之间内在的联系性。例如，考虑"the cat is walking in the bedroom"这句话。如果我们在训练语料中看到了很多类似“the dog is walking in the bedroom”或是“the cat is running in the bedroom”这样的句子，那么，即使我们没有见过这句话，也可以从“cat”和“dog”（“walking”和“running”）之间的相似性，推测出这句话的概率。然而， Ngram模型做不到。

这是因为，Ngram本质上是将词当做一个个孤立的原子单元（atomic unit）去处理的。这种处理方式对应到数学上的形式是一个个离散的one-hot向量（除了一个词典索引的下标对应的方向上是1 ，其余方向上都是0）。例如，对于一个大小为5的词典：{"I", "love", "nature", "luaguage", "processing"}，“nature”对应的one-hot向量为：[0,0,1,0,0] 。显然，one-hot向量的维度等于词典的大小。这在动辄上万甚至百万词典的实际应用中，面临着巨大的维度灾难问题（The Curse of Dimensionality）

于是，人们就自然而然地想到，能否用一个连续的稠密向量去刻画一个word的特征呢？这样，我们不仅可以直接刻画词与词之间的相似度，还可以建立一个从向量到概率的平滑函数模型，使得相似的词向量可以映射到相近的概率空间上。这个稠密连续向量也被称为word的distributed representation。

事实上，这个概念在信息检索（Information Retrieval）领域早就已经被广泛地使用了。只不过，在IR领域里，这个概念被称为向量空间模型（Vector Space Model，以下简称VSM）。

VSM是基于一种Statistical Semantics Hypothesis[4]：语言的统计特征隐藏着语义的信息（Statistical pattern of human word usage can be used to figure out what people mean）。例如，两篇具有相似词分布的文档可以被认为是有着相近的主题。这个Hypothesis有很多衍生版本。其中，比较广为人知的两个版本是Bag of Words Hypothesis和Distributional Hypothesis。前者是说，一篇文档的词频（而不是词序）代表了文档的主题；后者是说，上下文环境相似的两个词有着相近的语义。后面我们会看到，word2vec算法也是基于Distributional的假设。

那么，VSM是如何将稀疏离散的one-hot词向量映射为稠密连续的Distributional Representation的呢？

简单来说，基于Bag of Words Hypothesis，我们可以构造一个term-document矩阵A：矩阵的行Ai,:: 对应着词典里的一个word；矩阵的列A:,j对应着训练语料里的一篇文档；矩阵里的元素Ai,j代表着wordwi在文档Dj中出现的次数（或频率）。那么，我们就可以提取行向量做为word的语义向量（不过，在实际应用中，我们更多的是用列向量做为文档的主题向量）。

类似地，我们可以基于Distributional Hypothesis构造一个word-context的矩阵。此时，矩阵的列变成了context里的word，矩阵的元素也变成了一个context窗口里word的共现次数。

注意，这两类矩阵的行向量所计算的相似度有着细微的差异：term-document矩阵会给经常出现在同一篇document里的两个word赋予更高的相似度；而word-context矩阵会给那些有着相同context的两个word赋予更高的相似度。后者相对于前者是一种更高阶的相似度，因此在传统的信息检索领域中得到了更加广泛的应用。

不过，这种co-occurrence矩阵仍然存在着数据稀疏性和维度灾难的问题。为此，人们提出了一系列对矩阵进行降维的方法（如LSI / LSA等）。这些方法大都是基于SVD的思想，将原始的稀疏矩阵分解为两个低秩矩阵乘积的形式。

Neural Network Language Model

接下来，让我们回到对统计语言模型的讨论。鉴于Ngram等模型的不足，2003年，Bengio等人发表了一篇开创性的文章：A neural probabilistic language model。在这篇文章里，他们总结出了一套用神经网络建立统计语言模型的框架（Neural Network Language Model，以下简称NNLM），并首次提出了word embedding的概念（虽然没

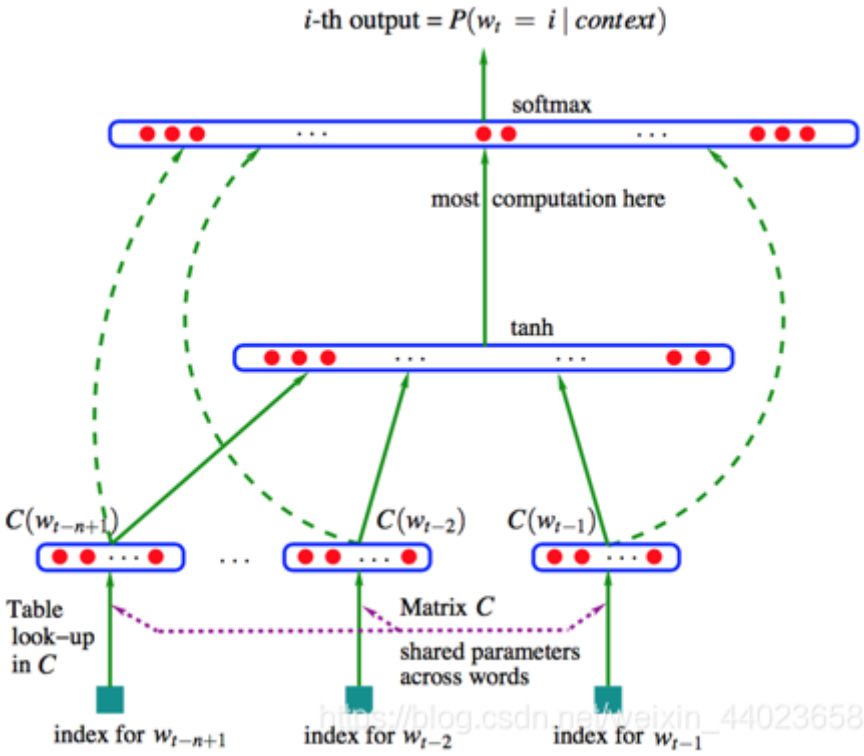
有叫这个名字），从而奠定了包括word2vec在内后续研究word representation learning的基础。

NNLM模型的基本思想可以概括如下：

- 假定词表中的每一个word都对应着一个连续的特征向量；
- 假定一个连续平滑的概率模型，输入一段词向量的序列，可以输出这段序列的联合概率；
- 同时学习词向量的权重和概率模型里的参数。

值得注意的一点是，这里的词向量也是要学习的参数。

在03年的论文里，Bengio等人采用了一个简单的前向反贵神经网络 $f(w_{t-n+1},\ldots,w_t)$ 来拟合一个词序列的条件概率 $p(w_t|w_1,w_2,\ldots,w_{t-1})$ 。整个模型的网络结构见下图:



我们可以将整个模型拆分成两部分加以理解：

- 首先是一个线性的Embedding层。它将输入的N-1个one-hot词向量，通过一个共享的D×V的矩阵C，映射为N-1个分布式的词向量（distributed vector）。其中，V是词典的大小，D是Embedding向量的维度（一个先验参数）。C矩阵里存储了要学习的word vector。
- 其次是一个简单的前向反馈神经网络g。它由一个tanh隐层和一个softmax输出层组成。通过将Embedding层输出的N-1个词向量映射为一个长度为V的概率分布向量，从而对词典中的word在输入context下的条件概率做出预估：

$$p(w_i|w_1,w_2,\ldots,w_{t-1}) \approx f(w_i,w_{t-1},\ldots,w_{t-n+1}) = g(w_i,C(w_{t-n+1}),\ldots,C(w_{t-1}))$$

我们可以通过最小化一个cross-entropy的正则化损失函数来调整模型的参数\theta：

$$L(\theta) = \frac{1}{T}\sum_t \log f(w_t, w_{t-1}, \ldots, w_{t-n+1}) + R(\theta)$$

其中，模型的参数θ包括了Embedding层矩阵C的元素，和前向反馈神经网络模型g里的权重。这是一个巨大的参数空间。不过，在用SGD学习更新模型的参数时，并不是所有的参数都需要调整（例如未在输入的context中出现的词对应的词向量）。计算的瓶颈主要是在softmax层的归一化函数上（需要对词典中所有的word计算一遍条件概率）。

然而，抛却复杂的参数空间，我们不禁要问，为什么这样一个简单的模型会取得巨大的成功呢？

仔细观察这个模型就会发现，它其实在同时解决两个问题：一个是统计语言模型里关注的条件概率 $p(w_t|\text{context})$ 的计算; 一个是向量空间模型里关注的词向量的表达。而这两个问题本质上并不独立。通过引入连续的词向量和平滑的概率模型，我们就可以在一个连续空间里对序列概率进行建模，从而从根本上缓解数据稀政性和维度灾难的问题。另一方面，以条件概率 $p(w_t|\text{context})$ 为学习目标去更新 词向量的权重, 具有更强的导向性, 同时也与VSM里的Distributional Hypothesis不谋而合。

在主角正式登场前，我们先看一下NNLM存在的几个问题。

一个问题是，同Ngram模型一样，NNLM模型只能处理定长的序列。在03年的论文里，Bengio等人将模型能够一次处理的序列长度N提高到了5，虽然相比bigram和trigram已经是很大的提升，但依然缺少灵活性。

因此，Mikolov等人在2010年提出了一种RNNLM模型[7]，用递归神经网络代替原始模型里的前向反馈神经网络，并将Embedding层与RNN里的隐藏层合并，从而解决了变长序列的问题。

另一个问题就比较严重了。NNLM的训练太慢了。即便是在百万量级的数据集上，即便是借助了40个CPU进行训练，NNLM也需要耗时数周才能给出一个稍微靠谱的解来。显然，对于现在动辄上千万甚至上亿的真实语料库，训练一个NNLM模型几乎是一个impossible mission。

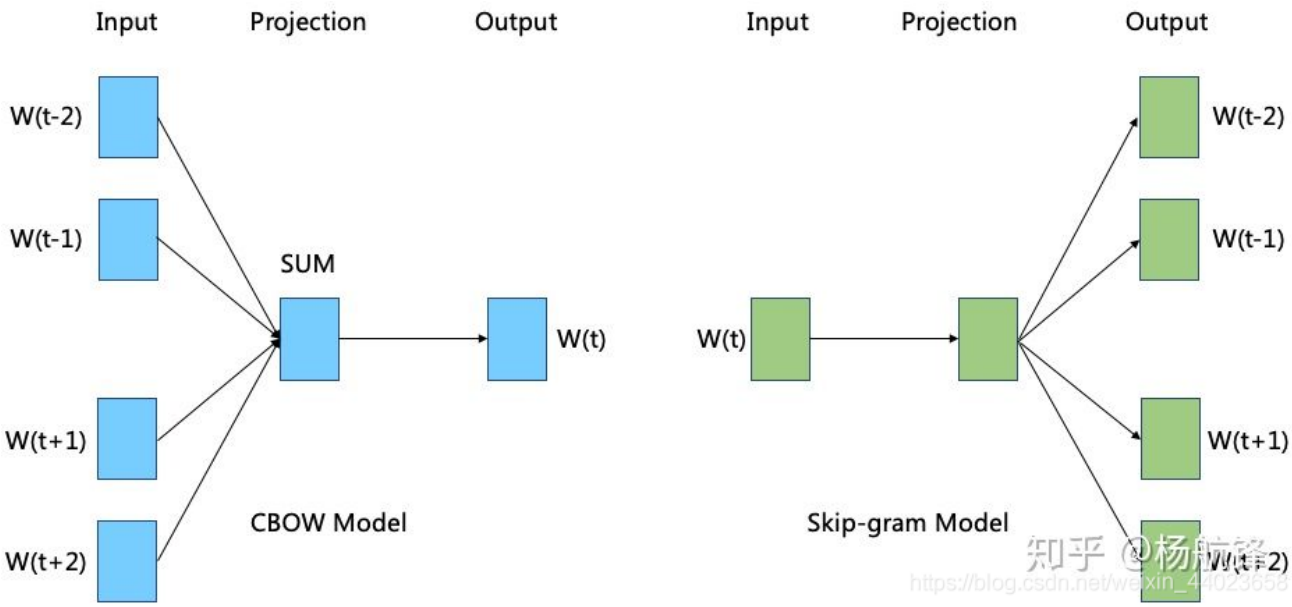
这时候，还是那个Mikolov站了出来。他注意到，原始的NNLM模型的训练其实可以拆分成两个步骤：

- 用一个简单模型训练出连续的词向量
- 基于词向量的表达，训练一个连续的Ngram神经网络模型。
- 而NNLM模型的计算瓶颈主要是在第二步。

如果我们只是想得到word的连续特征向量，是不是可以对第二步里的神经网络模型进行简化呢？

Mikolov是这么想的，也是这么做的。他在2013年一口气推出了两篇paper，并开源了一款计算词向量的工具——至此，word2vec横空出世，主角闪亮登场。

1 Word2Vec模型总述



Word2Vec 简单讲其实就是通过学习文本然后用词向量的方式表征词的语义信息，即通过 Embedding 把原先词所在空间映射到一个新的空间中去, 使得语义上相似的单词在该空间内 距离相近。以传统神经网络为基础的神经概率语言模型，缺点主要是计算量太大，集中体现在：隐 层和输出层之间的矩阵运算和输出层上的 Softmax 归一化运算上。因此 Word2Vec 就 是针对这两点来优化神经概率语言模型的。Word2Vec 中两个重要的模型是：CBOW 模 型和 Skip-gram 模型。对于这两个模型, Word2Vec 给出了两套框架，分别是基于 Hierarchical Softmax 和 Negative Sampling 来设计的，本文硫理的是第一种类型。

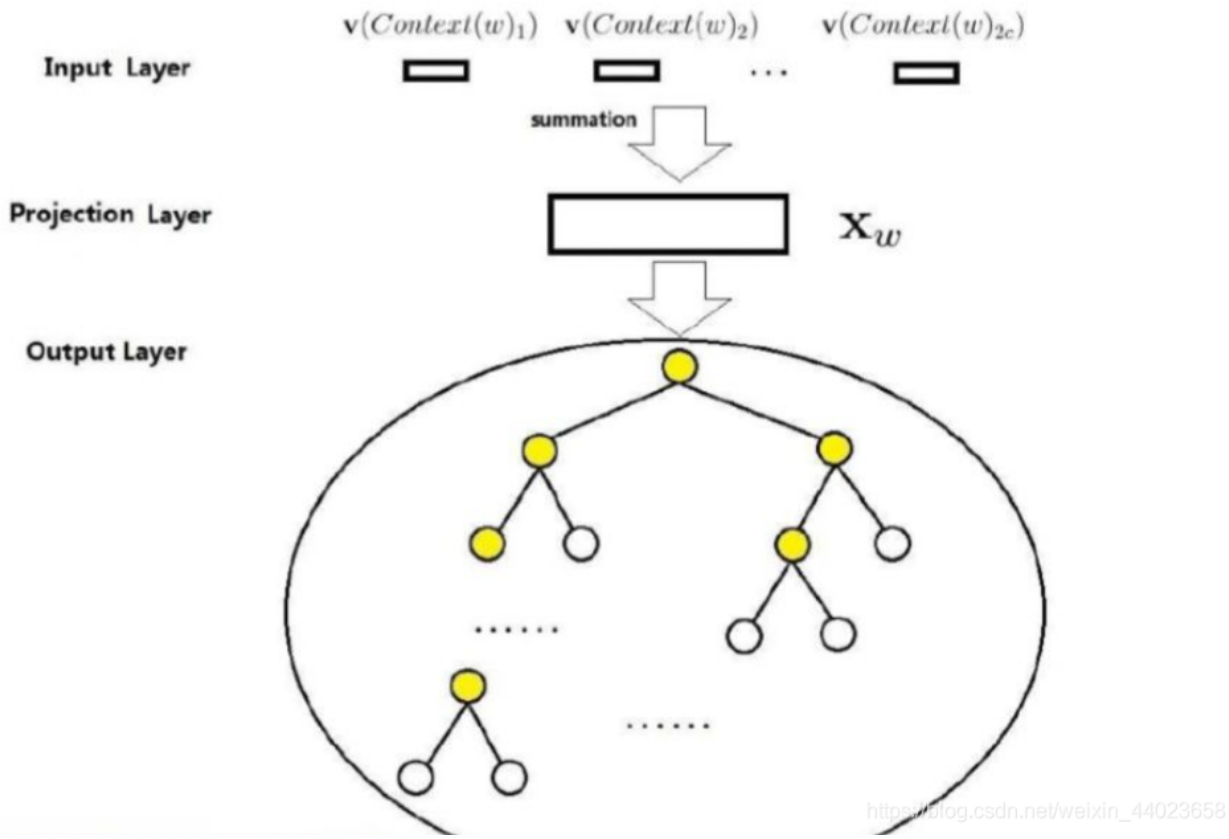
2 CBOW模型

2.1 基于 $HierarchicalSoftmax$ 模型的网络结构CBOW 模型的全称是 Continous bag-of-words，它包括三层结构分别是：输入层。投影层和输出层。

1. 输入层：包含 Context (w) 中 $2c$ 个词的词向量 $v(\text{Context}(w)_1), v(\text{Context}(w)_2), \cdots, v(\text{Context}(w)_{2c})$ 其中 $\forall v \in \mathbb{R}^n$， n 表示词向量的长度。

2. 投影层：将输入层的 $2c$ 个向量做求和累加处理，即

$$X_w = \sum_{i=1}^{2c} v(\text{Context}(w)_i)$$



Sample:$(\text{Context}(w), w)$

3. 输出层：输出层对应一颗 Huffman 树，它是以语料中出现过的词当叶子节点，以各词 在 语料库中出现的次数当权值构造而成。在这颗 Huffman 树中，叶子结点共 $N(=|\mathcal{D}|)$ 个 分别对应词典 \mathcal 中的词，非叶结点 $N-1$ 个 (上图中黄色的结点) 。

2.2 梯度的计算为了后续方便描述问题，首先对 $CBOW$ 模型中用到的符号做一个统一的说明：

$\boldsymbol{p}^w$：从根节点到出发到达 $\boldsymbol{w}$ 对应叶子结点的路径;$\cdot l^w$：路径 p^w 中包含节点的个数 $\cdot \{p_1^w, p_2^w, \cdots, p_{l^w}^w\}$：路径 p^w 中的 l^w 个结点, 其中 p_1^w 表示根结点, $p_{l^w}^w$ 表示词 w对应的结点;

·$\left\{d_2^w, d_3^w, \cdots, d_{l^w}^w\right\}$, 其中 $d_j^w \in \{0,1\}$: 词 w 对应的 Huffman 编码, 它由 l^w-1 位编码构成, d_j^w 表示路径 p^w 中第 j 个结点对应的编码（根结点不对应编码）

·$\left\{\theta_1^w, \theta_2^w, \cdots, \theta_{l^w-1}^w\right\}$, 其中 $\theta_j^w \in \mathbb{R}^n$: 路径 p^w 中非叶子结点对应的向量, θ_j^w 表示 路径 p^w 中第 j 个非叶子结点对应的向量。

所以 Hierarchical Softmax 的思想，即对于词典 D 中的任意词 $w, Huffman$ 树中必然存在唯一一条从根结点到词 w 对应叶子结点的路径 p^w 。路径 p^w 上存在 l^w-1 个分支, 将每个分支看作一次二分类，那么每一次分类就对应一个概率，最后将这些概率连乘得到 $p(w \mid \text{Context}(w))$

$$
\begin{aligned}
p(w \mid \text{Context}(w)) &= \prod_{j=2}^{l^w} p\left(d_j^w \mid X_w ; \theta_{j-1}^w\right) \\
p\left(d_j^w \mid X_w ; \theta_{j-1}^w\right) &= \begin{cases} \sigma\left(X_w^T \theta_{j-1}^w\right), & \text{if } d_j^w = 0 \\ 1-\sigma\left(X_w^T \theta_{j-1}^w\right), & \text{otherwise} \end{cases}
\end{aligned}
$$

其中 $\sigma(x)=\frac{1}{1+e^{-x}}$ 。通过对数极大似然化处理可得 $\boldsymbol{CBOW}$ 模型的目标函数为:

$$
\begin{aligned}
&\mathcal{L}=\sum_{w \in \mathcal{D}} \log \prod_{j=2}^{l^w}\left(\left[\sigma\left(X_w^T \theta_{j-1}^w\right)\right]^{1-d_j^w} \cdot\left[1-\sigma\left(X_w^T \theta_{j-1}^w\right)\right]^{} \right. \\
&=\sum_{w \in \mathcal{D}} \sum_{j=2}^{l^w}\left(\left(1-d_j^w\right) \cdot \log \left[\sigma\left(X_w^T \theta_{j-1}^w\right)\right]+d_j^w \cdot \log \left[1-\sigma\left(X_w^T \theta_{j-1}^w\right)\right]\right) \\
&=\sum_{w \in \mathcal{D}} \sum_{j=2}^{l^w} \Phi\left(\theta_{j-1}^w, X_w\right)
\end{aligned}
$$

Word 2Vec 极大化化目标函数使用的算法是是随机梯度上升法, 首先考虑 $\Phi\left(\theta_{j-1}^w, X_w\right)$ 关于 θ_{j-1}^w 的梯度计算:

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

于是, θ_{j-1}^w 的更新公式为:

$$
\theta_{j-1}^w := \theta_{j-1}^w+\eta\left(1-d_j^w-\sigma\left(X_w^T \theta_{j-1}^w\right)\right) X_w
$$

然后再考虑 $\Phi\left(\theta_{j-1}^w, X_w\right)$ 关于 X_w 的梯度计算:

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

如果观察到 $\Phi\left(\theta_{j-1}^w, X_w\right)$ 中 θ_{j-1}^w 和 X_w 具有对称性, 那么计算相应梯度会更方便。由于 X_w 表示的是 $\text{Context}(w)$ 中所有词向量的蛋加, 那么如何根据 $\nabla_{X_w}\Phi\left(\theta_{j-1}^w, X_w\right)$ 来更 新每一个分量 $v(\widetilde{w})$ 呢? $Word2Vec$ 中的做法非常的朴素, , 直接取

$$v(\widetilde{w}) := v(\widetilde{w}) + \eta\sum_{j=2}^{l^w}\frac{\partial\Phi\left(\theta_{j-1}^w, X_w\right)}{\partial X_w}, \widetilde{w} \in \text{C}$$

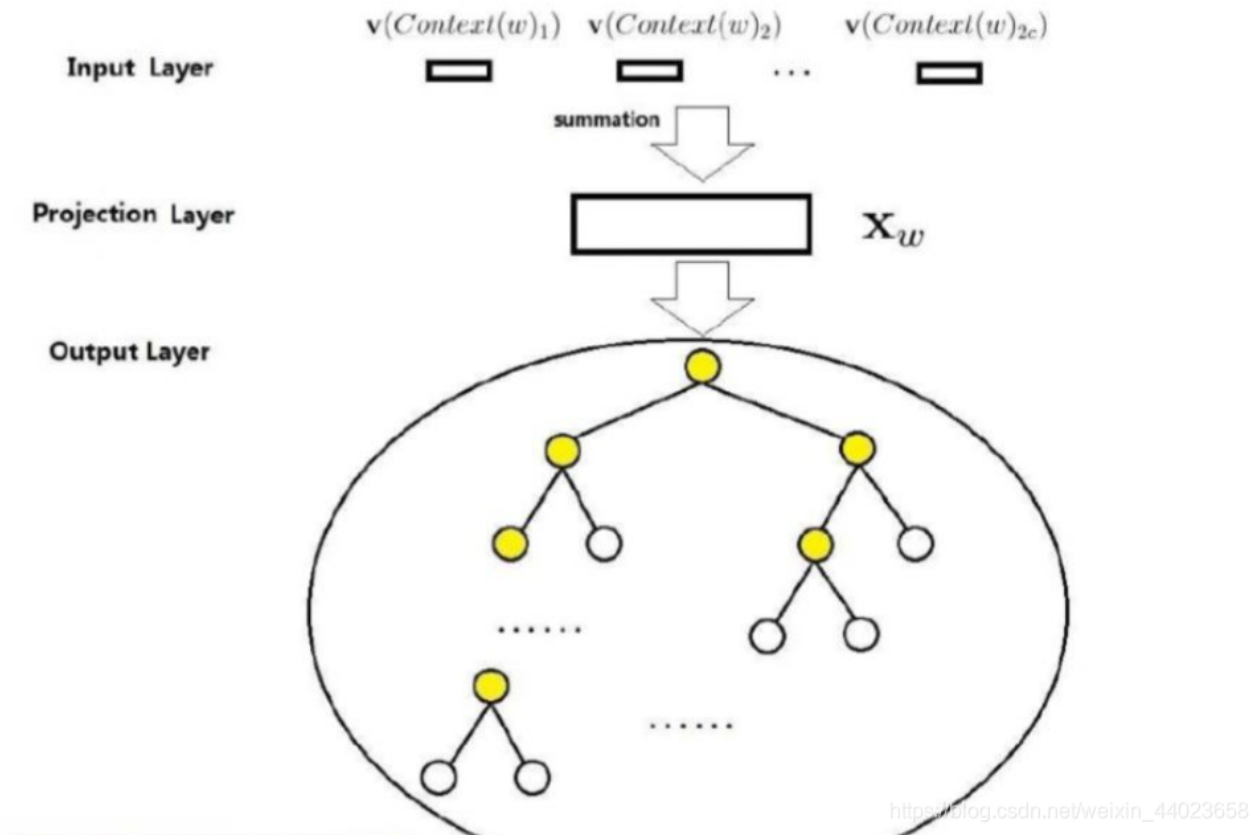
2.3 CBOW模型更新相关参数伪代码

1. $e=0, X_w=\sum_{u\in\text{Context}(w)} v(u)$
2. FOR$j=2: l^w DOq=\sigma\left(X_w^T\theta_{j-1}^w\right)g=\eta\left(1-d_j^w-q\right)e:=e+g\theta_{j-1}^w\circ\theta_{j-1}^w:=\theta_{j-1}^w+gX_w$
3. FOR $u\in$ Context (w) DO$\circ v(u):=v(u)+e$

3 Skip-gram模型

同 $\boldsymbol{CBOW}$ 模型一样, Skip-gram 模型的网络结构也包括三层结构分别是输入层、投影层和输出层:

1. 输入层：只含有当前样本的中心词 w 的词向量$v(w)\in\mathbb{R}$
2. 投影层：该层为恒等投影，其实这层可有可无，在这里只是为了方便和 CBOW 模型的网络结构做对比。



Sample:$(\text{Context}(w), w)$

3. 输出层：和 CBOW 模型一样，输出层也是一颗 Huffman 树。

3.2 梯度的计算

对于 Skip-gram 模型已知的是当前词 w, 需要对其上下文 Context (w) 中的词进行预 测，所以关键是条件概率函数 $p(\text{Context}(w)|w)$ 的构造， Skip-gram 模型中将其定义 为:

$$p(\text{ Context }(w)|w)=\prod_{u\in \text{Context}(w)} p(u|w)$$

上式中的 $p(u|w)$ 可以类比上节介绍的 Hierarchical Softmax 的思想, 因此可得:

$$\begin{aligned} p(u|w) &= \prod_{j=2}^{l^w} p\left(d_j^u|v(w);\theta_{j-1}^u\right) \\ &= \prod^{l^u} \left[\sigma\left(v(w)^T\theta_{j-1}^u\right)\right]^{1-d_j^u} \cdot \left[1-\sigma\left(v(\right.\right. \end{aligned}$$

通过对数极大似然化处理可得 Skip-gram 模型的目标函数为：

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{D}} \log \prod_{u\in \text{Contert}(w)} \prod_{j=2}^{t^u} \Big(\big[\sigma\big(v(w)^T\theta^u_{j-1}\big)\big]^{1-d^u_j} \cdot \Big[1-\sigma\big(v(w)^T\theta^u_{j-1}\big]^{d^u_j}\Big) \\ &= \sum_{w\in\mathcal{D}} \sum_{u\in \text{Context}(w)} \sum_{j=2}^{l^u} \big((1-d^u_j)\cdot \log\big[\sigma\big(v(w)^T\theta^u_{j-1}\big)\big] + d^u_j \cdot \log\big[1-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big]\big) \\ &= \sum_{w\in\mathcal{D}} \sum_{u\in \text{Context}(w)} \sum_{j=2}^{l^u} \mathcal{O}\big(\theta^u_{j-1}, v(w)\big)\end{aligned}$$

首先考虑 $\mathcal{O}\Big(\theta^u_{j-1}, v(w)\Big)$ 关于 θ^u_{j-1} 的梯度计算:

$$\begin{aligned}\frac{\partial \mathcal{O}\Big(\theta^u_{j-1}, v(w)\Big)}{\partial \theta^u_{j-1}} &= \frac{\partial}{\partial \theta^u_{j-1}} \big((1-d^u_j)\cdot \log\big[\sigma\big(v(w)^T\theta^u_{j-1}\big)\big] + d^u_j \cdot \log\big[1-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big]\big) \\ &= \big(1-d^u_j\big)\big[1-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big]v(w) - d^u_j\sigma\big(v(w)^T\theta^u_{j-1}\big)v(w) \\ &= \big(\big(1-d^u_j\big)\big[1-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big] - d^u_j\sigma\big(v(w)^T\theta^u_{j-1}\big)\big)v(w) \\ &= \big(1-d^u_j-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big)v(w)\end{aligned}$$

于是, θ^v_{j-1} 的更新公式为:

$$\theta^u_{j-1} := \theta^u_{j-1} + \eta\Big(1-d^u_j-\sigma\Big(v(w)^T\theta^u_{j-1}\Big)\Big)v(w)$$

然咸再考虑 $\mathcal{O}\Big(\theta^u_{j-1}, v(w)\Big)$ 对关于 $v(w)$ 的梯度计算 (亦可根据对称性直接得出)

$$\begin{aligned}\frac{\partial \mathcal{O}\Big(\theta^u_{j-1}, v(w)\Big)}{\partial v(w)} &= \frac{\partial}{\partial v(w)} \big(\big(1-d^u_j\big)\cdot \log\big[\sigma\big(v(w)^T\theta^u_{j-1}\big)\big] + d^u_j \cdot \log\big[1-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big]\big) \\ &= \big(1-d^u_j\big)\big[1-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big]\theta^u_{j-1} - d^u_j\sigma\big(v(w)^T\theta^u_{j-1}\big)\theta^u_{j-1} \\ &= \big(\big(1-d^u_j\big)\big[1-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big] - d^u_j\sigma\big(v(w)^T\theta^u_{j-1}\big)\big)\theta^u_{j-1} \\ &= \big(1-d^u_j-\sigma\big(v(w)^T\theta^u_{j-1}\big)\big)\theta^u_{j-1}\end{aligned}$$

于是, $v(w)$ 的更新公式为:

$$v(w) := v(w) + \eta \sum_{u\in Context(w)} \sum_{j=2}^{l^u} \frac{\partial \mathcal{O}\Big(\theta^u_{j-1}, v(w)\Big)}{\partial v(w)}$$

3.3 Skip-gram 模型更新相关参数伪代码•$e=0$・FOR $u \in$ Context (w) DO

$$
\begin{aligned}
&\text{FOR } j = 2 : l^u \text{ DO} \\
&\text{q} = \sigma\left(v(w)^T \theta_{j-1}^u\right) \\
&\text{g} = \eta\left(1 - d_j^u - q\right) \\
&\text{e} := e + g\theta_{j-1}^u \\
&\theta_{j-1}^u := \theta_{j-1}^u + gv(w) \\
&\text{o } v(w) := v(w) + e
\end{aligned}
$$

4 总结$Word2Vec$的基本功能就是把自然语言中的每一个词，表示成一个统一意义统一维度的词向量，因为只有把自然语言转化为向量的形式，才能在此之上构建相关的算法，至于向量中的每个维度具体是什么含义，得自己探索了~