Leature Note: Part I

关键词：自然语言处理(NLP)，词向量，SVD，Skip-gram. 连续词袋（CBOW），负采样（Negative Sampling），层次（Hierarchical）Softmax，Word2Vec。

这一系列的文档通过引入NLP的概念和当今NLP面临的问题作为开端。之后我们会讨论通过数值向量来表示词的概念。最后我们会讨论一下设计词向量的流行方法。

1. NLP的介绍

首先我们来从宏观层面上来讨论什么是NLP。

1.1 NLP中最特别的是什么？

人类的自然语言中，最特别的是什么。人类语言是一个专门为了传递意思而搭建的系统。它不是像那些有实际形体的表示，因此NLP和计算机视觉或者是其他机器学习任务很不一样。

大多数词语对于语言外的实体而言只是一个符号：

例如，词汇“rocket”表示火箭的概念，也可以指火箭的实例。有一些例外，当我们用词语和字母来表达信号，例如"Whooompaa"。基于此，语言符号可以表示若干种形态：声音，手势，书写等等。它们以连续的信号传输到大脑中。

1.2 任务举例

在NLP中，从语音处理到语义解析和话语处理中，有许多不同级别的任务。NLP的任务是能够设计算法，从而让计算机“理解”自然语言，进而来执行一些任务。不同的任务，难度是不一样的。

简单难度：

拼写纠正、关键词检索、同义词检索。

中等难度：

从网站、文档中抽取信息等

困难难度：

机器翻译（例如：中英翻译）、语义分析（查询语句的含义）、指代问题（“他”和“它”在文档中指什么）、问答（例如：回答竞猜问题）。

1.3 词语的表达

在NLP任务中，最常见的是如何将词语表达，作为我们的模型输入。大多数早期的NLP工作中，我们都没有把单词看作是最小的原子符号。为了在大多数NLP任务中表现出色，我们首先需要能够描述单词的相似性与差异性。有了词向量，我们可以很简单地将这种能力编码进向量中（例如用Jaccard、Cosine、Euclidean距离等）。

2. 词向量

虽然在英语中，大约有1.3千万个词汇，但是它们之间难道都是没有联系的吗？例如“feline”和“cat”，“hotel”和“motel”？我认为它们不是没有联系。因此，我们想要将英语词汇编码成某种向量，使得每个词汇在“词汇空间”中代表了一个点。这有很多重要的原因，但是最直观的原因是，可能存在有一个N维的空间（N远小于1.3千万），可以充分地将我们的语言表达的所有意思都装进去。每个维度会装进一部分我们传递的信息。例如，语义可能会表达时态（过去式、现在式、将来式），数量（单数和复数）和性别（男性和女性）。

我们首先来看第一个词向量，也是最简单的一个：独热向量（**one-hot vector**）。将每个单词表达成$\mathbb{R}^{|V|\times 1}$的向量。某个单词在英语单词的排序中的位置（index）为i，则这个单词的词向量，在第i个位置的元素值为1，其余的地方都为0。按照这个概念，$|V|$就是我们单词表的单词数量。这种词向量，会将单词表达成以下形式。

$$w^{aardvark} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{a} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

可以看到，我们将每个单词表示成了完全独立的个体。正如我们之前讨论到，这种单词的表达没有办法直接表达出单词相似性的概念。例如$(w^{hotel})^T w^{motel} = (w^{hotel})^T w^{cat} = 0$。

因此我们尝试降低空间的维度，找到一个维度小于$\mathbb{R}^{|V|\times 1}$子空间，从而可以找到单词间的关系。

3. 基于奇异值分解（SVD）的方法

利用这种方法找到的词嵌入（也叫词向量），我们首先遍历整个数据集，将词语共同出现的次数累计起来，形成一个矩阵X。然后对矩阵X进行SVD分解，得到USV^T的分解形式。我们用矩阵U的行，作为单词的词嵌入。接下来讨论一下X矩阵的几种选择。

3.1 词-文档矩阵

我们做一个大胆的猜想，相关的词语会出现在同一篇文档中。例如，”银行“、”债券“、”股票“、“钱”等词语会出现在一起，而“银行”、“章鱼”，“香蕉”和“曲棍球”可能就不会一起出现。我们利用这个事实，来构造一个词-文档矩阵。做法是：循环检索成千上万的文档，对于每个出现在文档j中的单词i，我们把它加到X_{ij}中。显然这是一个很大的矩阵$\mathbb{R}^{|V|\times M}$，其中M表示文档的数量。也许我们可以做得比这个更好。

3.2 基于窗口的共现矩阵

和3.1同样的逻辑也被运用到这，矩阵X表示的是单词的共现矩阵，因此这是一个关联矩阵（affinity matrix）。这个方法中，我们计算，给定一个感兴趣的单词，在这个单词周围的一个特定大小的窗口内，每个单词出现的次数。我们计算预料中的所有单词出现的次数。下面展示一个例子。预料中只包含3个句子，窗口大小为1：

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

计数矩阵的结果是

$$
X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array}
\begin{array}{c}
\begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\
\begin{bmatrix}
0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}
\end{array}
$$

3.3 将SVD方法运用到共现矩阵上

在矩阵X上运用SVD分解，观察奇异值（分解后的S矩阵的对角线）。基于期望解释的方差百分比$\frac{\sum_{i=1}^{k}\sigma_i}{\sum_{i=1}^{|V|}\sigma_i}$，选择在某个位置$k$截断。我们认为子矩阵$U_{1:|V|,1:k}$作为我们的词嵌入矩阵。因此这就是单词表中的每个单词的k维表示。

下面是在共现矩阵X上使用SVD分解的过程

$$
\begin{array}{c} |V| \\ |V| \end{array}
\begin{bmatrix} \\ X \\ \\ \end{bmatrix}
=
\begin{array}{c} |V| \\ |V| \end{array}
\begin{bmatrix} | & | & \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix}
\begin{array}{c} |V| \\ |V| \end{array}
\begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}
\begin{array}{c} |V| \\ |V| \end{array}
\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix}
$$

通过选择前k个奇异值对应的奇异向量，达到降维的效果。

$$
\begin{array}{c} |V| \\ |V| \end{array}
\begin{bmatrix} \\ \hat{X} \\ \\ \end{bmatrix}
=
\begin{array}{c} k \\ |V| \end{array}
\begin{bmatrix} | & | & \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix}
\begin{array}{c} k \\ k \end{array}
\begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}
\begin{array}{c} |V| \\ k \end{array}
\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix}
$$

这两种方法产生的词向量充分地把语义和语法进行了编码，但是也有很多其他的问题。

1. 矩阵的维度经常变（新的单词经常会加入，并且语料的大小也会变化）
2. 矩阵会异常稀疏，因为大多数单词没有一起出现
3. 矩阵的维度很高（大约$10^6 \times 10^6$）
4. 训练的时候是二阶的开销（在进行SVD分解时）
5. 在处理矩阵X时，需要引入一些手段，来解决词频非常不均等的情况。

以下是一些处理上述问题的方法

1. 忽略类似于“the”，“he”，“has”等词汇
2. 使用一个不等权的窗口，例如，基于文档中，单词间的数量，对共现的计数进行加权。

3. 使用pearson相关系数，不使用原始的计数，而是让负计数强行设置为0。

下一节中，迭代优化的方法，会让上面提到的问题，非常优雅的被解决。

4. 基于迭代的方法—Word2vec

来看一个新方法。这不再是计算和存储巨大的数据集的全局信息（可能上亿条句子），我们建立一个模型，使得每次迭代，都能够学到东西，最终能够在给定上下文的时候，把词语的概率编码出来。

这是一个设计模型的想法，模型的参数就是词语的向量。然后，按照一定的目标来训练模型。在运行模型的每个迭代中，衡量错误率，并且采用一个更新模型的方法，这个方法会对产生误差的模型参数进行惩罚。因此，我们学习了词向量。这个想法非常古老，可以追溯到1986年，我们称之为“后向传播”误差【见Rumelhart et al., 1988】。模型越简单，训练速度越快。

有几种方法已经尝试过了【Collobert et al., 2011】，首先要把单词转化为向量。对每个特殊的任务（命名实体识别NER，词性标注POS），它们不仅训练模型的参数，同时训练向量，以达到很好的表现。其他有趣的论文有【Bengio et al., 2003】。

在这节课中，我们会举一个简单的，最近的概率方法【Mikolov et al., 2013】：word2vec。Word2vec是一个软件包，包括：

两个算法：连续词袋（CBOW）和Skip-gram。其中，CBOW以词向量的形式，通过将词语作为中心词，通过该词语的上下文来预测中心词。Skip-gram则相反，它则是从中心词出发，去预测它的上下文的概率分布。

两种训练方法：负采样（negative sampling）和层次softmax（hierarchical softmax）。负采样方法定义目标的方法是对负例子进行采样。层次softmax是采用高效的树结构来定义目标，从而计算每个单词的概率。

4.1 语言模型（单元Unigrams、二元Bigrams等）

首先，我们需要产生一个模型，可以给一系列词语计算出概率。举个例子："The cat jumped over the puddle".

一个好的语言模型可以给这个句子高的概率，因为这基本上是一句合格的句子，无论是从语法上还是语义上。相似的，句子“stock boil fish is toy”应该会被给到很低的概率，因为它没有任何意义。数学上，我们可以用下面这个式子来表示句子的概率

$$P(w_1, w_2, \ldots, w_n)$$

我们采用一元语言模型的方法，把计算句子的概率拆分成，计算每个词语的概率，认为每个词语是独立的。

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i)$$

然而，我们知道这么做有一点可笑，因为我们知道下一个词语出现的概率是高度依赖前面的词语序列的。这个简单的句子可能真的会被打上很高的分数。因此我们会让句子的概率取决于一对词语的概率，即当前词和下一个词。我们称之为二元模型，表示为

$$P(w_1, w_2, \ldots, w_n) = \prod_{i}^{n} P(w_i|w_{i-1})$$

同样，这也有一点简单，因为我们只考虑了词语本身和它的邻居，而不是整个句子。而通过整个句子来计算概率，可以让我们考虑的很多。注意到，利用上下文尺寸为1构成的词与词的矩阵，我们基本上可以计算出词与词之间的概率。但是，这需要计算并且保存数据集全局的信息。

现在，我们可以理解，给定一个词语序列的概率，我们怎么看待它。接下来，我们看一些，可以学习（计算）出这些概率的模型。

4.2 连续词袋模型（CBOW）

一个方法是，把"The", "cat", "over", "the", "puddle"看成是"jumped"上下文，通过上下文能够预测，或者生成"jumped"。这类方法称为CBOW模型。

深入讨论CBOW模型，首先我们预设一些已知的参数。首先我们把模型中已知的参数表示为独热词向量。输入的上下文独热向量表示为$x^{(c)}$。CBOW的模型的输出记做$y^{(c)}$。因为我们只有一个输出，因此记做y，即中心词的输出。现在来定义一下未知的模型餐素。

产生两个矩阵$\mathcal{V} \in \mathbb{R}^{n \times |V|}$和$\mathcal{U} \in \mathbb{R}^{|V|} \times n$。$n$可以是任意尺寸，表示嵌入空间的尺寸。$\mathcal{V}$是输入词矩阵，第$i$列表示词语$w_i$的$n$维词嵌入向量。我们定义这个$n \times 1$的向量为$v_i$。相似的，$\mathcal{U}$矩阵是输出矩阵。第$j$行向量是$w_j$的$n$维词嵌入向量，定义为$u_j$。注意到，对每个词语$w_j$，我们会学习两组向量（输入向量$v_i$和输出向量$u_i$）。

我们把这个模型分解为以下3步。

1. 为每个词语产生独热向量，尺寸为m：$(x^{(c-m}, \ldots, x^{(c-1}, x^{(c+1}, \ldots, x^{(c+m} \in \mathbb{R}^{|V|})$
2. 获得上下文的词嵌入矩阵$(v_{c-m} = \mathcal{V}x^{(c-m)}, v_{c-m+1} = \mathcal{V}x^{c-m+1}, \ldots, v_{c+m} = \mathcal{V}x^{(c+m)} \in \mathbb{R}^{|V|})$
3. 把这些词嵌入向量平均起来$\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\ldots+v_{c+m}}{2m} \in \mathbb{R}^n$。
4. 产生一个打分向量$z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$。越相似的向量，点乘的结果越大。它会迫使相似的词语靠的更近，从而得高分。
5. 把得分转换为概率$\hat{y} = \mathrm{softmax}(z) \in \mathbb{R}^{|V|}$。
6. 我们希望产生的概率$\hat{y} \in \mathbb{R}^{|V|}$能够和真实的概率$y \in \mathbb{R}^{|V|}$尽可能匹配，这是真实的词语的独热向量。

现在，我们理解了，假如我们有矩阵$\mathcal{V}$和$\mathcal{U}$的情况，模型是怎么工作的。那么，我们怎样训练这两个矩阵。我们需要建立一个目标函数。当我们试图从一个真实的概率分布中，去学习一个概率时，我们从信息论中，选出一个描述两个分布的距离的指标。这里，我们使用一个流行的描述距离/损失的指标，叫做交叉熵$H(\hat{y}, y)$。

在离散的情况下，选择交叉熵的原因可以由下面的损失函数中表达出来。

$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log\left(\hat{y}_j\right)$$

其中，y是一个独热向量。因此上面的损失可以简化成

$$H(\hat{y}, y) = -y_i \log(\hat{y}_i)$$

在这个表达式中，c是正确的单词的词向量中，元素为1的下标。我们可以认为当$\hat{y_c} = 1$时，我们的预测时完美的。我们可以计算$H\hat{y}, y = -1\log(1) = 0$。因此，对于一个完美的预测，我们没有承受损失的惩罚。现在我们考虑相反的场景，如果我们预测的不好，给出了$\hat{y}_c = 0.01$的结果，像之前一样，我们计算出损失为$H(\hat{y}, y) = -1\log(0.01) \approx 4.605$。因此我们可以看出交叉熵可以给我们提供一个好的衡量距离的指标。因此我们可以将优化的目标表示成

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

我们用随机梯度下降法来更新所有相关词语的词向量u_c和v_j。

4.3 Skip-Gram模型

另外一个方法是，给定中心词“jumped”，模型要能够预测或产生出周围的词语"The", "cat", "over", "the", "puddle"。我们把单词"jumped"称为上下文。并且称呼这类模型为Skip-Gram模型。

接下来我们讨论Skip-Gram模型。初始设置是一样的，只不过我们互换了在CBOW中x和y的位置，即在CBOW中的x是在Skip-Gram的y。输入的中心词的独热向量表示为x（没有下标的原因是它只有一个词语）。输出为$y^{(j)}$。和CBOW一样，我们定义矩阵$\mathcal{V}$和$\mathcal{U}$。

我们将这个模型按照以下步骤分解

1. 产生输入的中心词独热向量$x\in\mathbb{R}^{|V|}$
2. 产生中心词的词嵌入向量$v_c=\mathcal{V}x\in\mathbb{R}^n$
3. 产生打分矩阵$z=U_{lv}$
4. 将打分矩阵转换为概率$\hat{y}=\text{softmax}(z)$。注意到$y^{(c-m)},\ldots,y^{(c-1)},y^{(c+1)},\ldots,y^{(c+m)}$是每个上下文单词的概率。
5. 我们希望产生的概率向量能够和真实的独热概率向量$y^{(c-m)},\ldots,y^{(c-1)},y^{(c+1)},\ldots,y^{(c+m)}$匹配上。

像在CBOW中，我们需要产生一个目标函数来评估模型。一个关键的不同点在于，我们引入了朴素贝叶斯的假设来分解概率。如果你之前没有看过朴素贝叶斯，那么可以简单的理解为，这是一个很强（简单）的条件独立假设。换句话说，给定一个中心词，所有输出的周围词语是完全独立的。

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

有了这个目标函数，我们可以计算对于未知变量的梯度，在每次迭代中，通过随机梯度下降法来更新它们。

注意到

$$J = -\sum_{j=0,j\neq m}^{2m} \log P\left(u_{c-m+j}|v_c\right)$$
$$= \sum_{j=0,j\neq m}^{2m} H\left(\hat{y}, y_{c-m+j}\right)$$

其中$H\left(\hat{y}, y_{c-m+j}\right)$是概率向量$\hat{y}$和独热向量$y_{c-m+j}$只见的交叉熵。

4.4 负采样

再看一遍目标函数，注意到对$|V|$求和的计算量很大。每次更新或者是计算目标函数，我们都要执行复杂度为$O(|V|)$的运算，这会造成百万级别的计算开销。一个简单的想法是，我们可以近似求解。

每个训练步骤中，我们不采用再整个单词表上做循环的操作，而是仅仅采样出一些负的例子。我们从一个噪声分布$P_n(w)$中采样，这个噪声分布的概率和单词表出现的频率的顺序是一致的。r为了强调引入负采样机制后的问题，我们必须将以下内容进行更新。

目标函数、梯度、更新规则。

MIKOLOV ET AL.在论文Distributed Representations of Words and Phrases and their Compositionality中介绍了负采样。当在Skip-Gram模型中运用负采样，事实上优化的目标就变了。考虑到单词和上下文的组合(w, c)。这个组合是来自于训练数据的吗？我们定义$P(D = 1|w, c)$表示这个组合来自于语料数据的概率。相对应的，$P(D = 0|w, c)$表示组合(w, c)不是来自于语料数据的概率。首先，我们利用sigmoid函数来对概率$P(D = 1|w, c)$建模：

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

现在，我们建立了一个新的目标函数，如果语料库中出现的单词与上下文组合，我们试图最大化在$P(D = 1|w, c)$。相反，如果单词与上下文组合没有出现在语料库中，我们想要最大化$P(D = 0|w, c)$。我们对这两个概率简单使用最大似然的方法来（参数θ在这里的参数是$\mathcal{V}$和$\mathcal{U}$）

$$\begin{aligned}
\theta &= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D = 1|w, c, \theta) \prod_{(w,c)\in \tilde{D}} P(D = 0|w, c, \theta) \\
&= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D = 1|w, c, \theta) \prod_{(w,c)\in \overline{D}} (1 - P(D = 1|w, c, \theta)) \\
&= \operatorname{argmax} \sum_{\theta} \log P(D = 1|w, c, \theta) + \sum_{(w,\mathcal{C})\in \tilde{D}} \log(1 - P(D = 1|w, c, \theta)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1 + \exp\left(-u_w^T v_c\right)} + \sum_{(w,c)\in \tilde{D}} \log\left(1 - \frac{1}{1 + \exp\left(-u_w^T v_c\right)}\right) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1 + \exp\left(-u_w^T v_c\right)} + \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1 + \exp\left(u_w^T v_c\right)}\right)
\end{aligned}$$

最大化似然函数和最小化负对数似然是一样的，因此

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1+\exp\left(-u_{iw}^T v_c\right)} - \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1+\exp\left(u_w^T v_c\right)}\right)$$

其中，$\tilde{D}$是“错误”或“负”的语料，例如有"stock boil fish is toy"这样的句子。不正常的句子出现的概率比较低。我们可以通过随机在词语集中采样，从而生成数据集$\tilde{D}$。

对于Skip-Gram，在给定了中心单词c的情况下，观察到上下文单词$c-m+j$的目标函数就更新为

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

对于CBOW，给定了上下文单词向量$\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\ldots+v_{c+m}}{2m}$，观察中心词向量$u_c$的目标函数更新为

$$-\log\sigma\left(u_c^T \cdot \hat{v}\right) - \sum_{k=1}^{K} \log\sigma\left(-\tilde{u}_k^T \cdot \hat{v}\right)$$

在上面的表达式中，$\{\tilde{u}_k|k=1\ldots K\}$是从$P_n(w)$中采样出来的。我们来讨论一下$P_n(w)$长什么样。有很多关于如何才能使得近似得最好的讨论。在许多研究中，似乎是单元模型的$3/4$次方可以做到最好。为什么是$3/4$，这里有一个例子也许对理解有帮助。

$$\text{is: } 0.9^{3/4} = 0.92$$
$$\text{Constitution: } 0.09^{3/4} = 0.16$$
$$\text{bombastic: } 0.01^{3/4} = 0.032$$

"Bombastic"被采样到的概率和原来比起来多了 3 倍，而"is"只是多了小数点的零头。

4.5 层次softmax

Mikolov et al.同样在论文中描述了层次softmax，比普通的softmax要快上许多。实践中，层次softmax对不频繁的单词表现更好，而负采样在频繁的单词和低纬度的情况下表现更好。

层次softmax使用了二叉树来代表单词表中的所有单词。每个叶子是一个单词。从根节点到叶子节点的路径是唯一的。在这个模型中，没有输出词语的表示形式。相反，每个图中的节点（除了根和叶）和模型要学习的向量都有关系。

在这个模型中，在给定向量w_i的情况下，词语w的概率$P(w|w_i)$，等于从根结点开始随机游走，走到词语w对应的叶子节点的概率。这种结构的主要有点是在计算概率的时候的复杂度是$O(\log(|V|))$，这对应了路径的长度。

引入一些标记，$L(w)$是从根结点到叶子节点w路径的节点数量。例如，在图 4 中$L(w_2)$等于 3。将$n(w,i)$记作通往w的路径上的第i个节点，用$v_{n(w,i)}$表示该节点的向量。因此$n(w,1)$表示根结点，$n(w,L(w))$是w的父节点。对于每一个内部的节点n，我们随机选择一个它的子节点，记作$ch(n)$（例如，总是取左节点）。我们可以计算概率

$$P(w|w_i) = \prod_{j=1}^{L(w)-1} \sigma\left(\left[n(w,j+1) = \text{ch}(n(w,j))\right] \cdot v_{n(w,j)}^T v_{w_i}\right)$$

其中

$$[x] = \begin{cases} 1 \text{ if } x \text{ is true} \\ -1 \text{ otherwise} \end{cases}$$

$\sigma(\cdot)$是sigmoid函数。

这个公式的信息量很大，我们仔细来看看。

首先，我们基于从根结点$n(w,1)$到叶子节点w的路径的形状，来计算点乘。如果假设$ch(n)$总是代表节点n的左节点，那么当路径往左走，$[n(w,j+1)=\mathrm{ch}(n(w,j))]$会返回`1`，否则返回`-1`。

$[n(w,j+1)=\mathrm{ch}(n(w,j))]$也起到了归一化的作用。在节点$n$，如果把往左和往右的概率加起来，对任何的$v_n^T v_{w_i}$，都有

$$\sigma\left(v_n^T v_{w_i}\right)+\sigma\left(-v_n^T v_{w_i}\right)=1$$

归一化同时保证了$\sum_{w=1}^{|V|} P(w|w_i)=1$，这和原始的softmax有一样的性质。

最后，我们利用点乘，比较了输入的向量v_{w_i}和每个内部的节点的向量$v_{n(w,j)}^T$的相似度。下面用一个例子介绍整个过程。

以图 4 中的w_2为例，我们从根结点要进行两次左转，然后一次右转，从而到达w_2，因此

$$\begin{aligned} P\left(w_2|w_i\right) &= p\left(n\left(w_2,1\right),\text{ left }\right)\cdot p\left(n\left(w_2,2\right),\text{ left }\right)\cdot p\left(n\left(w_2,3\right),\text{ right }\right) \\ &= \sigma\left(v_{n(w_2,1)}^T v_{w_i}\right)\cdot\sigma\left(v_{n(w_2,2)}^T v_{w_i}\right)\cdot\sigma\left(-v_{n(w_2,3)}^T v_{w_i}\right) \end{aligned}$$

在训练模型的过程中，我们的目标依旧是最小化负对数似然函数$-\log P(w|w_i)$。不过，这里不再是对每个输出词语的向量进行更新，而是更新从根结点到叶子节点路径中经过的每个节点的向量。

这个方法的速度取决于二叉树构建的方法和词语被分配到叶子的方式。Mikolov et al.使用了霍夫曼（Huffman）二叉树，这棵树的构建方式是根据词频，频率越高，路径越短。

附录

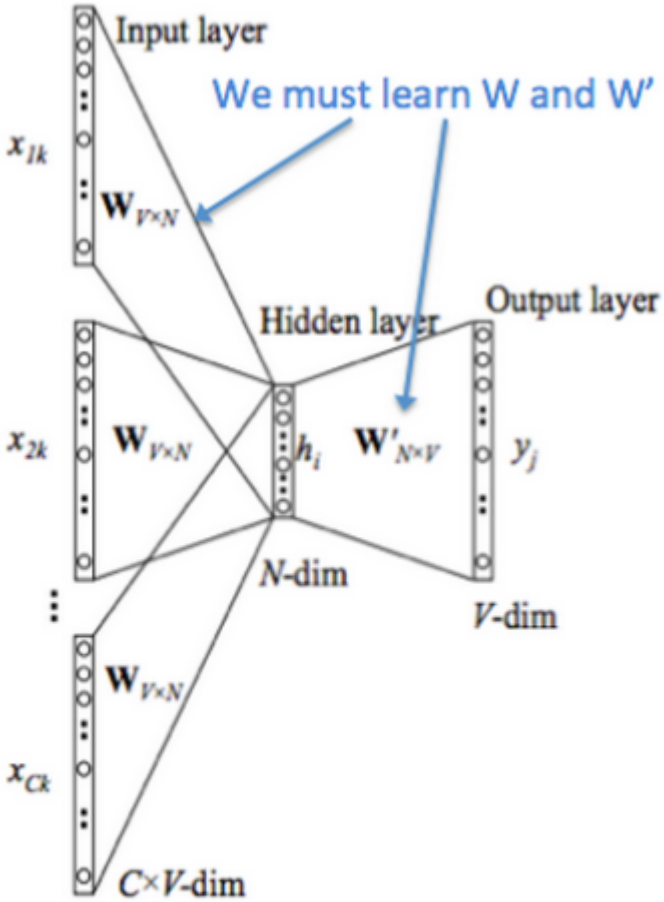

Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

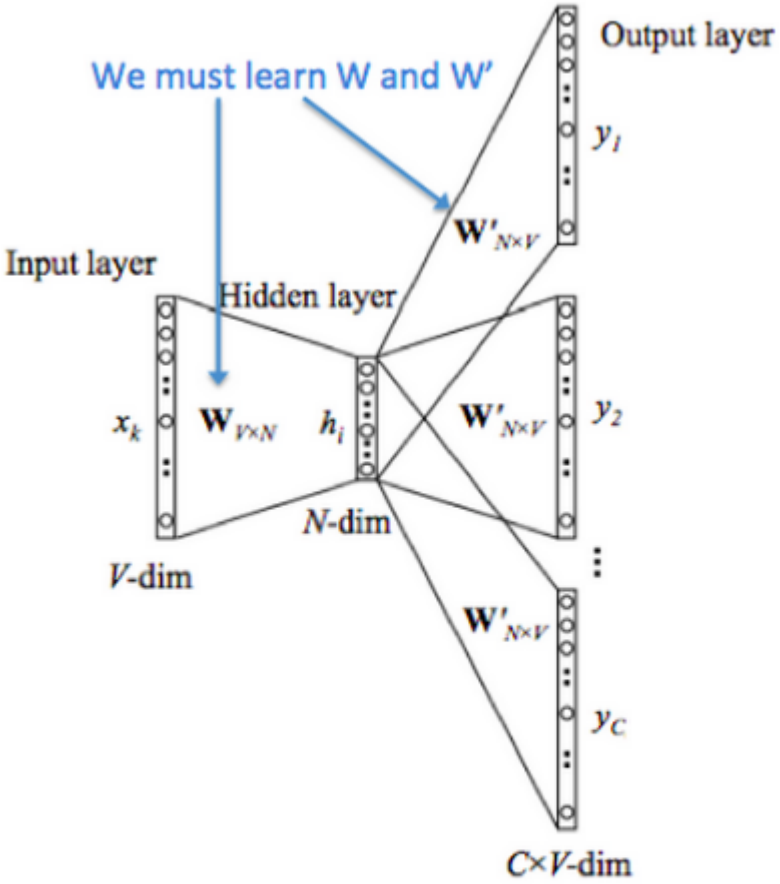

Figure 2: This image demonstrates how Skip-Gram works and how we must learn the transfer matrices

The **sigmoid** function

$\sigma(x) = \frac{1}{1+e^{-x}}$

is the 1D version of the softmax and can be used to model a probability

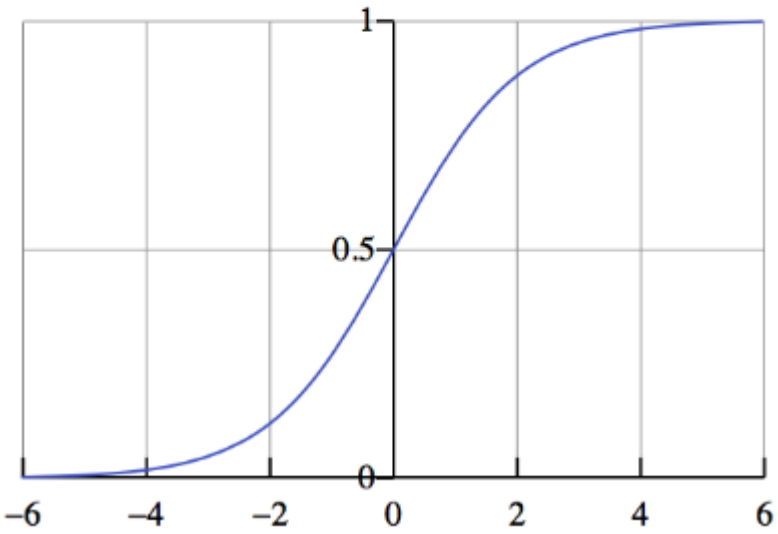

Figure 3: Sigmoid function

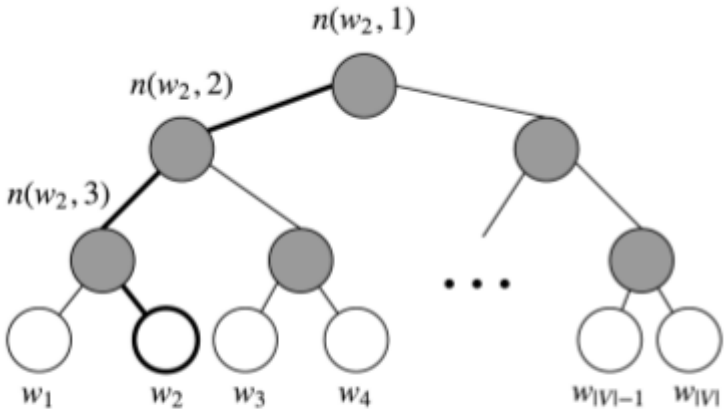

Figure 4: Binary tree for Hierarchical softmax