# 第一章 概述

分布式词向量的思想有一个朴素的假设：词的某些语义信息（和某些语法信息）蕴含在词的上下文关系中，具体来说，具有相似上下文的词，大概率也有相似的语义（或语法特性）。所有的分布式词向量的生成方法都是基于这个假设。

分布式词向量的意义在于：相比于词的不含语法语义信息的one-hot向量，用具有语法语义信息的分布式且维度较低的向量作为词的特征，进而构建句子和文档的特征，然后输入机器学习模型，能够得到较好效果和效率。

用一句话来说明分布式词向量的意义，就是：基于上下文关系信息的词的语义表征。

无论使用什么方法生成词向量，都要基于语料库（corpus）和词典（vacabulary），语料库是大量文本数据，它由句子或者文档构成，每个句子或者文档都是词的序列；词典则是语料库中所有句子或文档中所有的词进行去重后得到的词的集合（对于中文语料库，要想得到词典，需要分词）。可见，词典由语料库决定。

# 第二章 静态词向量

第一节 MF（矩阵分解）方法

对稀疏、高维的词向量构成的矩阵进行SVD分解，然后选取保留大部分原始矩阵信息的前几维分量。这样就可以得到低维词向量。这是一个简单思路。

MF方法可以针对tf-idf向量构成的矩阵进行分解（LSA方法），也可以针对共现矩阵进行分解。下面仅介绍共现矩阵的矩阵分解。

共现矩阵指明了词典中任意两个词在语料库中同时出现（即相邻）的次数。

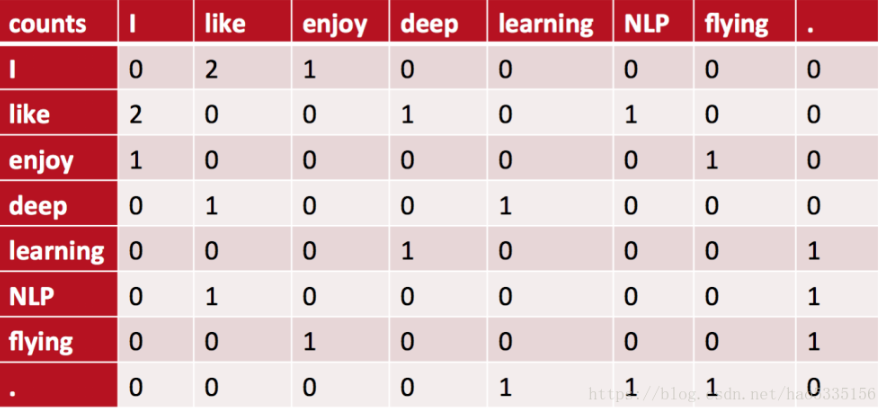
假设一个语料库中有三句话：

I like deep learning.

I like NLP.

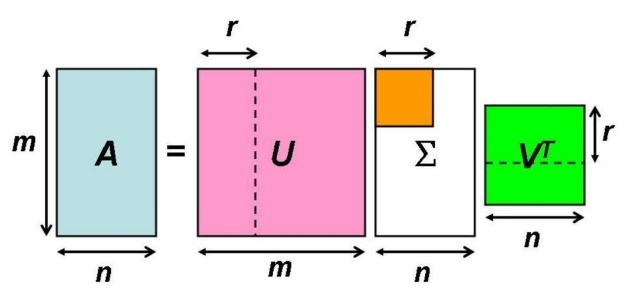
I enjoy flying.

则其共现矩阵为：



**图1 共现矩阵示意图**

共现矩阵中，每一列都可以看成是一个词的向量表示，向量的每个元素表示该词与其他词的共现次数，代表了一定的语义信息。但是这种词向量非常稀疏，其维度太高。因此通常使用SVD分解对共现矩阵进行分解并降维（人为指定降维后的向量长度），得到低维的分布式词向量。

****

**图2 SVD分解示意图**

共现矩阵MF方法存在的问题：

（1）SVD分解计算量太大；

（2）一些虚词如is、the、a等，它们与其它词的共现并没有包含语义信息，但是共现次数却很多，这影响语义表征。

第二节 NNLM

**1、NNLM简介**

NNLM（neural network language model）是Bengio等人在2003年提出的一种

模型，它是用神经网络的方式实现统计语言模型。在该模型中第一次提出了“分布式词向量”的概念，但分布式词向量只是该模型的一个副产品。

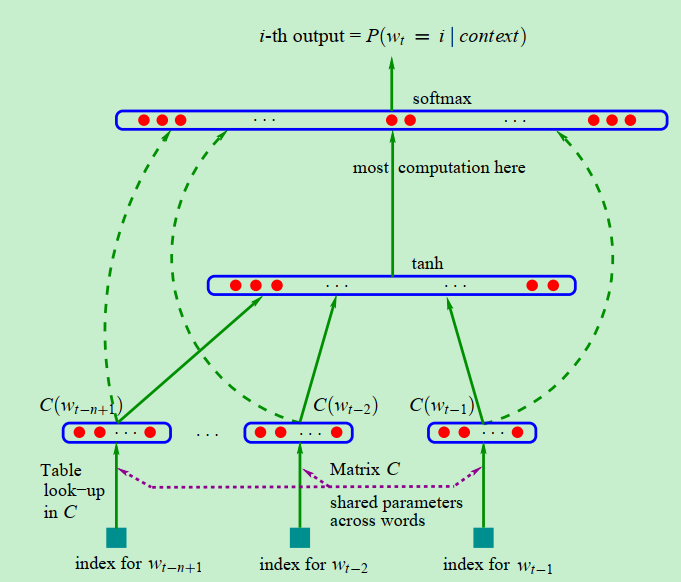
NNLM模型用上文的n-1个词的序列来预测第n个词更可能是哪个词。NNLM模型蕴含了词向量的概念（或者说，词向量是NNLM的副产品），它的隐层向量就相当于n-1词的词向量的拼接。

NNLM将语料库作为训练集，语料库中的句子或者文档就是训练样本。NNLM的目标是最大化似然概率，这个目标使得学习的词向量具有一定的语义信息（为什么）。

NNLM的架构具有很大的理论价值，但是它的softmax部分的计算代价太大。后续有很多模型都是基于NNLM架构，高效地实现词向量的生成，如LBL、RNNLM、C&W、CBOW与skip-gram、order等。

**2、NNLM的结构**

模型的结构如下：



**图2 NNLM结构图（图来自于Bengio的论文）**

NNLM输入是一个n-1个词构成的序列（通常每个词用one-hot向量作特征）；输出是第n个位置取各个词的概率。假设语料库的词典为V，则词典中每个词的one-hot向量的维数是N=|V|。下面分三部分介绍NNLM的结构：

（1）**输入和投影**：输入的n-1个词的one-hot向量，每个都经过投影矩阵C矩阵投影成m维的词向量（m是人为指定的隐层的维数），注意，所有输入的词共享矩阵C。显然，矩阵C的shape是|V|\*m。

实际上，由于输入是one-hot向量，所以投影矩阵C实际上起的是“look-up”的作用，也就是说，一个one-hot向量与C相乘的结果，就相当于取出one-hot向量中唯一取1的索引i，然后从C中查出第i行向量。因此，C矩阵实际上就相当于词典中所有词的词向量构成的矩阵。

（2）**隐层**：隐层向量由输入投影后的n-1个m维词向量拼接得到，因此隐层向量的shape是(n-1)\*m。然后隐层向量经过tanh激活函数，再由还原矩阵U将向量“还原”成|V|维向量。可见，还原矩阵U的shape是(n-1)\*m\*|V|。

（3）**还原和输出**：经还原矩阵U还原的向量，再经过softmax归一化成概率，用它来表示输入的n-1个词的后面一个词取词典中各个词的概率。

**3、NNLM的目标**

这里需要注意，NNLM的目标并不是最小化模型的输出和true label之间的误差，而是最大化用前n-1个词预测第n个词的似然概率：



NNLM训练好以后，不仅得到根据n-1个词构成的序列预测下一个词的概率，还得到了词向量。

第三节 word2vec

**1、简介**

word2vec是Google的一个开源工具，专门用来高效地训练词向量。它包括CBOW和skip-gram两个模型，并提出了一些高效计算的方法。

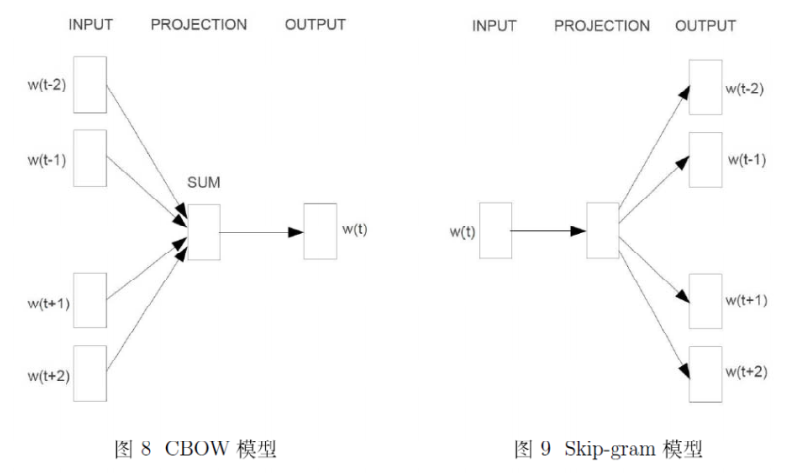
word2vec基本上也是遵从了NNLM的思想。但它为了高效地计算，对NNLM的模型结构进行了简化和改造：

（1）将隐层的tanh激活部分去掉，直接将投影的向量连到softmax输出；

（2）不仅考虑中心词的上文，还考虑其下文。

（3）不考虑上下文词的顺序，比如CBOW，它跟NNLM相比，不再将投影后的向量拼接，而是将这些向量加起来；

CBOW是用上下文词预测中心词，skip-gram是用中心词预测上下文词。它们的基本结构如下：



**图3 CBOW和skip-gram结构**

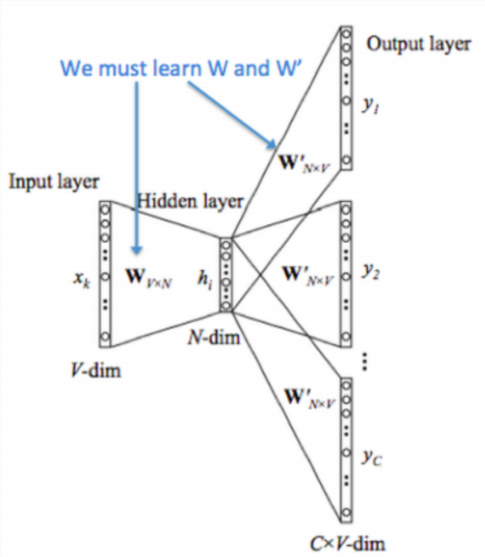
注意，CBOW模型的输入投影的处理与NNLM不同，它是将上下文词的向量投影后的结果加起来。但与NNLM相同的是，投影矩阵是被所有输入词共享的。

下面我们主要介绍skip-gram模型。

**2、skip-gram**

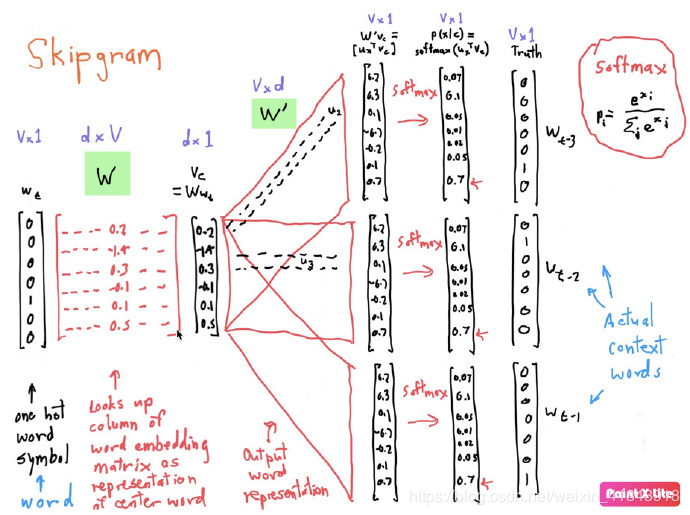
**（1）skip-gram的结构**

skip-gram的具体结构如下图：



**图4 skip-gram的具体结构**

一个更为生动的图如下：



**图5 skip-gram的具体结构**

skip-gram的结构可以类比NNLM的结构来理解。

参照图5，我们来描述一下skip-gram的结构：

其输入是一个词的one-hot向量（其维数为V），它作为中心词；该向量经过look-up矩阵W（d\*V维）的投影，得到隐层向量——输入词的词向量（d维），然后经过反投影矩阵W’得到C个维数为V的上下文词的输出词向量（注意，上面已经说了，skip-gram是不考虑不同上下文词的顺序的影响的，所以这些上下文词的输出词向量都一样），然后将这些上下文词向量经过softmax函数变成概率分布向量（显然，这些上下文词的概率分布向量也都一样），这个概率分布向量表示的就是输入词的周围取每个词的概率。

注意，W表示词典中所有词的中心词向量，W’则表示所有词的背景词向量——每个词都有一个“中心词向量”和一个“背景词向量（也就是上下文词向量）”（最终提取的一般是中心词向量）。

**（2）skip-gram的目标**

skip-gram的目标是基于语料库来定义的，具体来说，需要最大化用中心词预测上下文词的似然概率：

其中，T表示语料库的大小，c表示定义上下文词的滑窗大小，由如下的softmax来定义：

其中就是模型中的中心词，就是模型中的背景词（也就是上下文词），表示词w的中心词,从模型中的矩阵W中取出；表示词w的背景词，从模型中的矩阵W’中取出。

By the way，CBOW的目标函数是：

或者：

由于word2vec的目标是最大化用上下文词（中心词）预测中心词（上下文词）的概率，所以其学出来的词向量必然具有这样的性质：在词向量所在空间中（一般称为嵌入空间），具有相似语义的词的词向量距离较近，语义不相似的词的词向量距离较远。

**（3）skip-gram的训练**

上面介绍了skip-gram的结构和其目标。这里可能存在一个疑问：我们一般理解的神经网络模型，都是有监督的，模型输出需要跟true label作比较，将误差作为优化目标。但是skip-gram虽然也是一个神经网络模型，它有输入，有输出，但它并不是有监督的模型，尽管它确实有一个目标函数。这里需要灵活理解。

正因为skip-gram模型的这种特殊性，在训练时它实际上不需要完整地对每个输入计算出其输出。

根据目标函数，我们可以得出skip-gram的训练过程：使用梯度上升法，迭代地更新参数（即两个词向量矩阵），更新公式这里不再推导，但更新公式里需要计算语料库中每个词对其上下文词的条件概率。

可见，虽然模型的网络结构的输出是输入词对词库中的每个词的概率，但在训练过程中，对每一个样本（语料库中某个位置的一个词），我们不需要计算该词对所有词库中的词的条件概率，只需要计算该词对其上下文词的条件概率（但是在最基本的结构中，非上下文词的条件概率也作为副产品被计算出来了）。

另外，可以看出，对每一个输入的词，计算其对上下文词的概率时，要计算softmax，也就是需要对词典中所有词的背景词向量跟输入词的中心词向量求内积和指数，然后求和。这个计算量是非常大的。因此难以进行快速的训练。

注意，skip-gram模型本质上是使用语料库中词的上下文关系来学习词向量（强调：它只用了词的上下文关系的信息），因此输入的词的one-hot向量其实只是起到标记一下词的id的作用。这一点可以从模型中W矩阵其实只是起到look-up的作用看出。

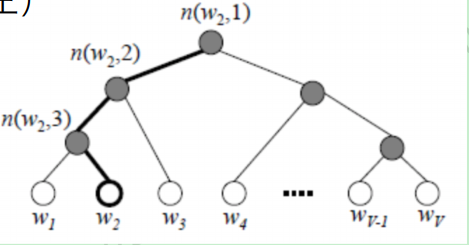
**3、模型改进**

**（1）hierarchical softmax**

分层softmax是一种用来简化计算NNLM中输出层直接计算softmax的技巧，因此同样适用于skip-gram和CBOW。它的根本思想是用递归的二分类来代替softmax的一次性多分类过程。具体来说，它基于词典中每个词的词频构建词典的Huffman树，用树结构（通常是Huffman树）代替softmax的输出层（实际上，从图5中看，是用树结构代替了W’映射+softmax层，树的节点上的向量起的作用就类似于W’里的每个词的背景词向量的作用），这样，计算每个条件概率时，平均只需计算次求和即可。

hierarchical softmax的思想其实来源于对softmax的本质的考察，它的本质其实就是多分类，就是计算中心词对每个词的概率，也就是由中心词预测上下文词属于每个类别的概率。那么自然地想到，这个计算概率的过程，可以用递归地二分类过程（这就形成一棵二叉树）对应的概率计算过程来代替。

下面我们详细地介绍一下hierarchical softmax。



**图6 词典的Huffman树**

首先，如何构建词典的Huffman树呢？如上图所示，Huffman树是基于词频构建的，所有的词都在叶子节点，词频越高的词越接近根节点。这样，每个词都有一个编码，编码对应于从根节点到该词的叶子节点的路径。

其次，如何根据Huffman树，计算一个条件概率？上面说道，这对应于一个二分类的过程。首先，我们已知中心词的词向量，然后从根节点出发，用logistic函数进行二分类，根据和节点上的参数值（是一个网络中的待定参数，与W和W’一样，需要训练得到）决定将输入的中心词对应的样本分到左子树的类还是分到右子树的类：

该概率表面的解释是分到正类的概率，同时它也是父节点相对于子节点的概率。

我们已经知道每个词（叶子节点）的编码，也就是能确定根节点到每个词的路径，因此要计算中心词（词向量为）对其一个背景词（在叶子节点上）的条件概率，就沿着相应的路径，一步一步地执行上述过程，然后再把路径上的分类概率都乘起来，就得到了中心词对该背景词的条件概率。可以证明，这样计算的条件概率是满足归一化条件的。

注意，这里的Huffman树对应二分类的过程，这确实有点抽象，不易理解。因为非叶子节点并不对应任何一个词，而且这里的“分类”也并不真是一个具体的分类过程。

另外，hierarchical softmax与softmax并不是等价的，它们计算出来的概率并不一定相等。hierarchical softmax虽然相比于softmax减小了计算量，但是也付出了代价。代价就是人为地增强了词与词之间的耦合性。因为不同的词可能拥有一部分相同的路径。这样一个词出现的概率会对其他词出现的概率产生影响。

（https://www.cnblogs.com/guoyaohua/p/9240336.html）

其实就相当于施加了一些先验约束。

当然，除了简单的Huffman树以外，还有其他的树结构，不同的树结构具有不同的计算效率。

**（2）NEG（负采样）**

负采样是对NCE（noise-contrastive-estimation）的简化算法，NCE算法基于这样的假设：好的模型可以通过logistic函数来区分正常数据和噪声。

负采样也是一种提升skip-gram模型效率的方法，它与hierarchical softmax不同，后者是改变了模型的输出层的结构，而前者是改变了模型的目标。

那么它是如何改变目标的呢？

负采样仍然是基于语料库来构建目标。对于语料库中每个位置的词w，我们取其上下文词，作为正样本；然后对每个正样本再取k个负样本，也就是在语料库中不是w的上下文词的词（负样本的选取也是个技术活，注意，负样本的采集是在语料库的全局进行采样的）。然后我们希望学习出来的向量具备这样的特性：能够使用logistic函数，将词向量作为样本特征，来区分正样本和负样本。仔细思考可知，这个目标跟softmax的目标本质上是一致的，但是负采样的目标更加简洁。具体来说，其本质目标就是想让学到的词向量位于这样一个空间中，在这个空间中语义不相似的词是线性可分的，且具有较大的区分度（认为对于一个给定的词，能作为其上下文的词跟不能作为其上下文的词在语义上是不相似的）。

对于词典中的每个词：



其中前一项是为了最大化正样本的logistic函数值（即概率值），这类似于最大似然的思想；后一项是为了最小化负样本的的logistic函数值（即概率值）。

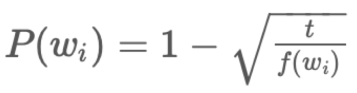
与原模型一样，整体的目标函数就是将语料库中每个位置的词的目标函数加起来。

可以看出，负采样目标与原模型的目标相比，对语料库中每个位置的词，不需要计算其词向量跟词典中所有词的词向量的内积，只需计算它的词向量跟正样本和采样出的负样本的词向量的内积。因此节省了时间。

（问题：既然负采样是在语料库全局进行采样，那么是否可以对一个词既做正采样，也做负采样，这样只需遍历词典，而不需遍历整个语料库了。）

**（3）高频词下采样**

这种技巧主要是为了提升低频词向量的准确性。它的做法是在训练时，依概率随机丢弃高频词：



其中t是阈值，是人为设定的参数，代表我们认为多大频率的词算是高频词。

第四节 fastText

fastText是Facebook于2016年开源的一个快速的文本分类的工具。它实现文本分类的功能，具有相当快的速度和较好的效果，且它也像word2vec一样，能够生成词向量。

fastText的模型结构与word2vec非常相似，但是它输入的不是语料库中的某个词（作为中心词）或者某几个词（作为上下文词），而是一个文档（词序列）；输出的不是target词的概率分布，而是文档的类别。

相应的，fastText的目标也不是最大似然，而是文档的分类。

fastText也使用one-hot作为词的特征，不同的是，它进一步将词分解成n-gram特征。

fastText是多分类问题，所以其输出层也采用了hierarchical softmax，用来加快计算。

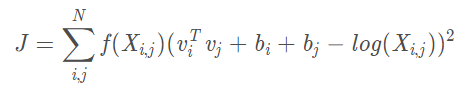
（问题：fastText的目标与word2vec不同，那么其训练出来的词向量的性质跟word2vec相比应该是有差异的，那么具体有什么差异？）

第五节 glove

glove（global vector）是2014年提出的一种训练词向量的方法，它基于共现矩阵的统计信息来学习词向量。

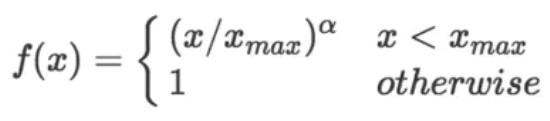
注意，glove不是一个网络结构模型。

设共现矩阵为X，其大小为N\*N（显然，N就是词典的大小），表示矩阵第i行第j列的值，也就是词典中第i个词与第j个词的共现次数，则glove的目标函数为：

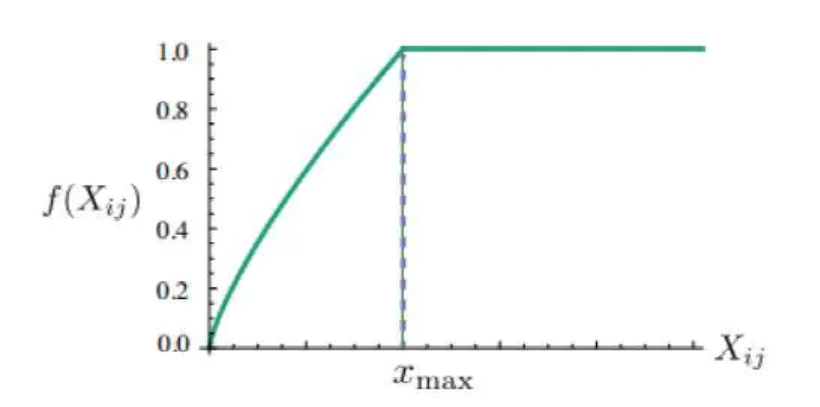


其中是第i个词的词向量，是第j个词的词向量，是第i个词的偏置，是第j个词的偏置。是对共现词对的加权，它对低频的共现词进行衰减，因为低频的共现词对包含的语义信息较少。

的定义如下：



当时，的图像为：



**图7 的图像**

glove方法是基于共现矩阵的，因此它学出来的词向量具有全局的语义信息。

这里的“全局语义信息”怎么理解呢？我们知道，共现矩阵中的每个值表示两个词在语料库中共现的次数，这是在整个语料库上统计得到的。而像word2vec这样的模型并没有先统计全局的信息然后再基于全局的统计信息构建目标，而是遍历语料库上各个位置的词与其周围词，构建目标函数。但是不能简单地认为word2vec就只是基于“局部”语义信息学习词向量，因为word2vec的目标函数是将语料库所有位置的词的条件概率乘起来作为似然，然后最大化这个似然。这样必然也考虑到了全局的信息。只不过不像基于共现矩阵的方法那样，更加明显地使用语义的全局统计信息。

因此，不能说glove由于基于全局语义统计信息，就比word2vec学到的词向量包含更多的全局语义信息。但是当然，既然glove和word2vec的目标不同，那么glove学到的词向量的特性必然也与word2vec不同（那具体是如何不同呢？）。

<https://www.jianshu.com/p/5e0443bd5efc>

<https://blog.csdn.net/linchuhai/article/details/97135612>

# 第三章 动态词向量

动态词向量是与静态词向量相对的概念。静态词向量是如第二章所述的方法获得的词向量，其对词典中每一个词给出一个固定的词向量，不随该词的语境不同而变化；而动态词向量则是同一个词在不同语境中有不同的表征。

动态词向量一般由预训练语言模型学习得到，但预训练语言模型的应用范式通常不是获得动态词向量，而是基于fine-tune方式应用到下游任务中。

关于预训练语言模型，我们将专门辟出一章来介绍。