第 3 章

自然语言处理的发展进程

本章重点介绍自然语言处理的发展历程，以及自然语言发展的趋势：规则→统计→深度学习。笔者就发展趋势方面利用时间轴的方法来介绍这一领域在发展过程中一些比较经典的方法和模型，从最初利用人工规则来处理自然语言的一些任务讲起，之后讲解如何“进化”到利用机器学习的方法来处理社会发展带来的更多复杂的任务，到发展到现在利用深度学习的方法应用在各种领域去处理各式各样的任务。

3.1 人工规则与自然语言处理

俗话说“巧妇难为无米之炊”。在机器学习中，数据和特征便是“米”，模型和算法则是“巧妇”。没有充足的数据、合适的特征，再强大的模型结构也无法得到满意的输出。正如一句业界经典所说，Garbage in，garbage out。对于一个机器学习问题，数据和特征往往决定了结果的上限，而模型、算法的选择及优化则在逐步接近这个上限，可见特征工程的重要性。在 1970 年以前，自然语言处理的研究主要分为两大阵营：一个是基于规则方法的符号派；另一个是采用概率方法的随机派。这一时期，虽然两种方法都取得了长足的发展，但由于当时多数学者注重研究推理和逻辑问题，所以可以说在当时基于规则的方法用来处理任务的比例更高。例如早在 20 世纪 50 年代初，Kleene 就研究了有限自动机和正则表达式。1956 年，Chomsky 又提出了上下文无关语法，并把它运用到了自然语言处理中，而且随着社会的发展，基于规则解决自然语言处理的问题也有长足的发展，例如词频、聚合度、自由度、编辑距离、主题和特征转换。下面就使用频率较高的特征处理方法（如词频、聚合度、自由度和编辑距离）做一个简要的介绍。

1. 词频

词频是多用于中文常用词分词的一种统计方法，这种方法旨在统计某个常用词在某个语境下或者某个数据集中出现的次数，从而判断这个词本身是否可以有独立成词的条件，即单独拿出这个词是否具有一定的含义。从人类语言学的角度来讲，能够具备成为词的要求的词语，一般在数据上会比较聚集地出现多次。

2. 聚合度

词频并不能作为判断是否具备独立成词条件的唯一标准，下面用一个例子引出更多元

化的标准来评估一个词语是否具备独立成词条件。例如，在一篇文章中用“电影院”成词这个例子来讲聚合度，笔者统计了在整个 2400 万字的数据中“电影”一词出现了 2774 次，出现的概率为 0.000 113，“院”字出现了 4797 次，出现的概率为 0.000 196 9，如果两者间真的毫无关系，则它们拼接在一起 P(电影院)的概率为 P(电影)×P(院)/2，但其实“电影院”一共出现 175 次，要远远高于两个词的概率的乘积，是 P(电影)×P(院)/2 的 600 多倍，还统计了“的”字出现的概率为 0.0166，并且文章中出现的“的电影”的真实概率 P(的电影)与 P(的)×P(电影)/2 很接近，所以表明“电影院”更可能是一个有意义的搭配，而“的电影”则更像是“的”和“电影”两个成分偶然拼接到一起的。通过这样的方式找到成词称为聚合度。计算过程及其举例如下：

(1) 计算当前词语 S 的在词语库中的出现概率 $P(S)$。

(2) 对词语 S 进行二分切法，切分出若干组词语组(SL，SR)，并分别计算每个词语的出现概率 $P(SL)$和 $P(SR)$。

(3) 对于切分出来的词语组计算聚合度 $\log\left(\frac{P(S)}{P(SL)\cdot P(SR)}\right)$，取最小值作为词语 S 的聚合度。其中，对得出的结果取对数是为了防止概率过低导致计算结果溢出，并把值域映射到了更加平滑的区间。

例如，“天气预报说周五会下雨”，使用 dop 表示聚合度，并令单字的聚合度为 0，则 dop(天)=0。

dop(天气)=P(天)×P(气)

dop(天气预)=P(天)×P(气预)+P(天气)×P(预)

dop(天气预报)=P(天)×P(气预报)+P(天气)×P(预报)+P(天气预)×P(报)

dop(天气预报说)=P(天)×P(气预报说)+P(天气)×P(预报说)+P(天气预)×P(报说)+P(天气预报)×P(说)

……

对词进行二切分，然后计算切分后的概率乘积，在这里除了每个二切分的概率乘积的和，其实也可以用另一种方法计算聚合度：“电影院”的聚合度则是 P(电影院)分别除以 P(电)×P(影院)和 P(电影)×P(院)所得的商的较小值，这样处理甚至会有更好的效果，因为用最小值来代表这个词的聚合度，更能有力地证明该词的成词性，如果该词的聚合度在最小的情况下都成词，则这个词肯定成词。

3. 自由度

只看文本片段的聚合度是不够的，还需要从整体看它在外部的表现。考虑“被子”，可以说“这被子”，“被子”是一个词语而且该词语的聚合度很高，而“这被子”并不是一个人类直观认为有意义的词语，但“这被子”的聚合度也很高，此时成词的标准就受到了挑战。

为此笔者引入自由度来解决此类问题，自由度的思想来源于信息熵，信息熵是一个定义事情信息量大小的单位。信息熵越高，含有的信息量越小，这件事情的不确定性也就越高；相反，信息熵越低，含有的信息量也就越大，则这件事情的确定性也就越高。算法人员可以

用信息熵来衡量一个文本片段的左邻字集合和右邻字集合有多随机。自由度的计算过程如下：

(1) 计算当前词语 S 的出现次数 N，则词语 S 的左边总共出现 N 个汉字。

(2) 对 N 个汉字的出现次数进行统计，计算词语 S 的左边每个字出现的概率。

(3) 根据信息熵公式计算左邻熵，同理计算右邻熵。其中，P_i 为词语 S 的左边每个字出现的概率。

(4) 信息熵越小对应着自由度越低，则该词语越稳定，因此选择信息熵最小的作为词语 S 的自由度。

$$E = -\sum_i P_i \log(P_i) \tag{3.1}$$

考虑这么一句话“吃葡萄不吐葡萄皮不吃葡萄倒吐葡萄皮”“葡萄”一词出现了 4 次，其中左邻字分别为 {吃，吐，吃，吐}，右邻字分别为 {不，皮，倒，皮} 。根据式(3.1)，“葡萄”一词的左邻字的信息熵为 $-(1/2)\times\log(1/2)-(1/2)\times\log(1/2)\approx 0.693$，它的右邻字的信息熵则为 $-(1/2)\times\log(1/2)-(1/4)\times\log(1/4)-(1/4)\times\log(1/4)\approx 1.04$。由此可见，在这个句子中，“葡萄”一词的右邻字更加丰富一些。

4. 编辑距离

编辑距离又称 Levenshtein 距离(莱文斯坦距离也叫作 Edit Distance)，指两个字符串之间，由一个转换成另一个所需的最少编辑操作次数，它们的编辑距离越大，字符串越不同。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符和删除一个字符。这个概念是由俄罗斯科学家 Vladimir Levenshtein 在 1965 年提出来的。它可以用来做 DNA 分析、拼字检测和抄袭识别等。总之，算法人员可以考虑使用编辑距离比较文本段的相似度。编辑操作只有 3 种：插入、删除和替换。例如有两个字符串，将其中一个字符串经过这 3 种操作之后，得到两个完全相同的字符串付出的代价是什么，是当前要讨论和计算的。

例如，有两个字符串 kitten 和 sitting，现在要将 kitten 转换成 sitting 可以进行如下一些操作：

kitten→sitten 将 k 替换成 s；

sitten→sittin 将 e 替换成 i；

sittin →sitting 添加 g。

在这里算法设置每经过一次编辑，也就是变化(插入、删除或替换)，花费的代价都是 1。

3.2　机器学习与自热语言处理

在机器学习中，利用一些规则，算法人员可以很好地使数据特征更加明显，使机器学习起来更加“轻松”，就如同有了很好的食材，或者说经过处理后的食材，可以被更好地处理成最好吃的食物，但是再好的食材如果厨师的厨艺很差，仍然会造成食材的浪费。模型和算法的应用对促进完成一些任务有很好的效果，可以说模型可以使数据的价值最大化。3.1 节

介绍了人们利用一些规则去处理数据，本节主要介绍一些传统算法的发展和应用。

3.2.1 词袋模型

词袋(BoW)模型是一种使用机器学习算法，也是数学中最简单的文本表示形式。该方法非常简单和灵活，可用于从文档中提取各种功能的各种方法。词袋是描述文档中单词出现的文本的一种表示形式。因为文档中的单词是以没有逻辑的顺序放置的，所以称为单词的“袋子”。该模型只关注文档中是否出现已知的单词，并不关注文档中出现的单词。

例如，以笔者之前看过的一部电影的评论作为例子：

评论 1：This movie is very scary and long

评论 2：This movie is not scary and slow

评论 3：This movie is spooky and good

首先根据以上 3 个评论中所有独特的单词来构建词汇表。词汇表由以下 11 个单词组成：This、movie、is、very、scary、and、long、not、slow、spooky 和 good。

将上述每个单词用 1 和 0 标记在上面的 3 个电影评论中。这将为 3 条评论提供 3 个向量，具体表示如表 3.1 所示。

表 3.1 句子转换为向量的表示

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	评论的 长度(单词数)
评论 1	1	1	1	1	1	1	1	1	0	0	0	7
评论 2	1	1	2	0	0	1	1	0	1	0	0	8
评论 3	1	1	1	0	0	0	1	0	0	1	1	6

评论 1 的向量：[1 1 1 1 1 1 1 1 0 0 0 0]

评论 2 的向量：[1 1 2 0 0 1 1 0 1 0 0]

评论 3 的向量：[1 1 1 0 0 0 1 1 0 0 1 1]

这是词袋模型背后的核心思想。使用词袋模型的缺点在于，当前可以有长度为 11 的向量，但是当遇到新句子时就会遇到问题：

第一，如果新句子包含新词，则词汇量将增加，因此向量的长度也将增加；第二，向量也将包含许多 0，从而导致稀疏矩阵(这是要避免的)；第三，不保留有关句子语法或文本中单词顺序的信息。

3.2.2 *n*-gram

在用谷歌或者百度搜索引擎时，输入一个或几个词，搜索框通常会以下拉菜单的形式给出几个备选，这些备选其实是在推测你想要搜索的那个词串。那么，原理是什么呢？也就输入“我们”的时候，后面的“都要好好的”“恋爱吧”等这些词语是怎么出来的，怎么排序的？实际上是根据语言模型得出的。假如使用二元语言模型预测下一个单词，则排序的过程如

图 3.1 所示。

图 3.1　n-gram 应用示例

P(“都要好好的”|“我们”)>P(“都在用力地活着是什么歌名”|“我们”)>P(“都要好好的电视剧免费观看”|“我们”)>……>P(“恋爱吧”|“我们”),数据的来源可以是用户搜索日志。

到底什么是 n-gram 呢? n-gram 是一个由 n 个连续单词组成的块,它的思想是一个单词出现的概率与它的 $n-1$ 个出现的词有关。也就是每个词依赖于第 $n-1$ 个词。下面是一些常见的术语及示例,可以帮助你更好地理解 n-gram 语言模型。

Unigrams: 一元文法,由一个单词组成的 token,例如,the、students、opened 和 their。

Bigrams: 二元文法,也叫一元马尔可夫链。由连续两个单词组成的 token,例如,the students、students opened 和 opened their。

Trigrams: 三元文法,由连续 3 个单词组成的 token,例如,the students opened 和 students opened their。

4-grams: 四元文法,由连续 4 个单词组成的 token,例如,the students opened their。

如何估计这些 n-gram 概率呢? 估计概率的一种直观方法叫作最大似然估计(MLE)。可以通过从正态语料库中获取计数,并将计数归一化,使其位于 0～1,从而得到 n-gram 模型参数的最大似然估计。

例如,要计算一个给定前一个单词为 x,后一个单词为 y 的 bigram 概率。计算 bigram $C(xy)$的计数,并通过共享第 1 个单词 x 的所有 bigram 的总和进行标准化。

$$P(X_n \mid X_{n-1}) = \frac{C(X_n X_{n-1})}{\sum XC(X_n X_{n-1})} \tag{3.2}$$

其中,分子为 bigram $C(xy)$在语料库中的计数,分母为前一个词 x,后一个词为任意词的 bigram 计数的总和。为了简单可以写成下面的形式:

$$P(X_n \mid X_{n-1}) = \frac{C(X_n X_{n-1})}{C(X_{n-1})} \tag{3.3}$$

这样就可以通过最大似然估计求得概率值，但是有个问题，在其他语料库中出现次数很多的句子可能在当前语料库中没有，所以很难进行泛化。n-gram 模型的稀疏性问题有以下几点。

(1) 如果要求的词没有在文本中出现，则分子的概率为 0。解决办法是添加一个很小的值给对应的词，这种方法叫作平滑，例如拉普拉斯平滑。这使词表中的每个单词都至少有很小的概率。

(2) 如果第 $n-1$ 个词没有出现在文本中，则分母的概率无法计算。解决办法是使用 water is so transparent that 替代，这种方法叫作后退，保证作为条件的分母概率值存在。(还有其他平滑技术)

(3) 概率是一个大于 0 小于 1 的数，随着相乘会变得很小，所以通常使用 log 的形式：$P_1P_2P_3P_4 = \exp(\log P_1 + \log P_2 + \log P_3 + \log P_4)$。

(4) 提高 n 的值会使稀疏性变得更糟糕，还会增加存储量，所以 n-gram 一般不会超过 5。

(5) 当 $n>2$ 时，例如 trigram，可能需要在头部添加两个 start-token，读者可自行验证效果。

3.2.3 频率与逆文档频率

TF-IDF 即术语频率-逆文档频率，是一种数字统计，反映单词对集合或语料库中文档的重要性。

术语频率(TF)用于衡量术语 t 在文档 d 中出现的频率：

$$\mathrm{TF}_{t,d} = \frac{n_{t,d}}{\text{文件中的术语数}} \tag{3.4}$$

其中，在分子中，n 是术语 t 出现在文档 d 中的次数，因此，每个文档和术语将具有其自己的 TF 值。笔者将再次使用在词袋模型中构建的相同词汇表来显示如何计算评论 2 的 TF：

评论 2：This movie is not scary and is slow

词汇：This、movie、is、very、scary、and、long、not、slow、spooky 和 good。评论 2 中的字数=8，单词 This 的 TF =(评论 2 中出现 This 的次数)/(评论 2 中的术语数)= 1/8。

同样有

TF(movie)=1/8

TF(is)=2/8=1/4

TF(very)=0/8=0

TF(scary)=1/8

TF(and)=1/8

TF(movie)=0/8

TF(not)＝1/8

TF(slow)＝1/8

TF(spooky)＝0/8＝0

TF(good)＝0/8＝0

所有术语和所有评论的术语频率如表 3.2 所示。

表 3.2　评论术语 TF

术　语	评论 1	评论 2	评论 3	TF(评论 1)	TF(评论 2)	TF(评论 3)
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

逆文档频率(IDF)用于衡量一个术语的重要性。算法人员需要 IDF 值，因为仅计算 TF 不足以理解单词的重要性，下面是计算 IDF 的公式：

$$\mathrm{IDF}_t = \log \frac{\text{文档数量}}{\text{包含单词'This'的文档数量}} \tag{3.5}$$

计算评论 2 中所有单词的 IDF 值：

IDF(This)＝log(文档数量/包含单词'This'的文档数量)＝log(3/3)＝log(1)＝0

同样：

IDF(movie)＝log(3/3)＝0

IDF(is)＝log(3/3)＝0

IDF(not)＝log(3/1)＝log(3)＝0.48

IDF(scary)＝log(3/2)＝0.18

IDF(and)＝log(3/3)＝0

IDF(slow)＝log(3/1)＝0.48

因此，整个词汇表的 IDF 值如表 3.3 所示。

表 3.3　评论术语 IDF

术　语	评　论　1	评　论　2	评　论　3	IDF
This	1	1	1	0.00
movie	1	1	1	0.00

续表

术　语	评　论　1	评　论　2	评　论　3	IDF
is	1	2	1	0.00
very	1	0	0	0.48
scary	1	1	0	0.18
and	1	1	1	0.00
long	1	0	0	0.48
not	0	1	0	0.48
slow	0	1	0	0.48
spooky	0	0	1	0.48
good	0	0	1	0.48

因此，读者可以看到像 is、This、and 等词被简化为 0，并且重要性不大，而 scary、long、good 等词更重要，具有较高的价值。

现在，算法可以为语料库中的每个单词计算 TF-IDF 分数。得分较高的单词更重要，得分较低的单词则不太重要：

TF-IDF(This，评论 2)＝TF(This，评论 2)×IDF(This)＝1/8×0＝0

TF-IDF(movie，评论 2)＝1/8×0＝0

TF-IDF(is，评论 2)＝1/4×0＝0

TF-IDF(not，评论 2)＝1/8×0.48＝0.06

TF-IDF(scary，评论 2)＝1/8×0.18＝0.023

TF-IDF(and，评论 2)＝1/8×0＝0

TF-IDF(slow，评论 2)＝1/8×0.48＝0.06

同样，针对所有评论计算所有单词的 TF-IDF 分数，如表 3.4 所示。

表 3.4　评论术语 TF-IDF

术　语	评论 1	评论 2	评论 3	IDF	TF-IDF（评论 1）	TF-IDF（评论 2）	TF-IDF（评论 3）
This	1	1	1	0.00	0.000	0.000	0.000
movie	1	1	1	0.00	0.000	0.000	0.000
is	1	2	1	0.00	0.000	0.000	0.000
very	1	0	0	0.48	0.068	0.000	0.000
scary	1	1	0	0.18	0.025	0.022	0.000
and	1	1	1	0.00	0.000	0.000	0.000
long	1	0	0	0.48	0.068	0.000	0.000
not	0	1	0	0.48	0.000	0.060	0.000
slow	0	1	0	0.48	0.000	0.060	0.000
spooky	0	0	1	0.48	0.000	0.000	0.080
good	0	0	1	0.48	0.000	0.000	0.080

总结一下本节所涉及的内容：词袋只创建一组向量，其中包含文档中单词出现的次数（审阅），而 TF-IDF 模型包含较重要单词和次重要单词的信息。词袋向量易于解释，但是，TF-IDF 通常在机器学习模型中表现更好。对于 n-gram 而言，利用前面的几个词来预测后面最有可能出现的几个词，效果也是很好的，直到今日在各个搜索引擎中的应用广泛。

3.3　深度学习与自然语言处理

深度学习目前虽然处于火热的发展阶段，但是不管是从理论方面来讲还是从实践方面来讲都有许多问题待解决。不过，我们处在一个大数据时代，而且随着计算资源的大幅提升，新模型、新理论的验证周期会更短。人工智能时代的开启必然很大程度地改变这个世界，不管是从交通、医疗、购物、军事等方面，还是涉及每个人生活的方方面面。或许我们处于最好的时代，也或许我们处于最不好的时代，但是未来无法预知，我们要做的是不断学习。本节将介绍在深度学习的发展过程中那些沉淀下来的经典模型，也是后面章节要讲的一些预训练模型的组成部分。

单词嵌入是文档词汇表最流行的表示形式之一。它能够最大程度地捕获文档中单词的上下文、语义及句法相似性，还有与其他单词的关系等。那么单词嵌入底是什么？广义上来讲，它们是特定单词在向量上的表示形式。话虽如此，但如何生成它们？更重要的是，它们如何捕获上下文？

Word2Vec 模型是使用浅层神经网络学习单词嵌入最流行的技术之一，它是由 Tomas Mikolov 于 2013 年开发的。虽然 Word2Vec 是浅层神经网络学习，但它是深度学习极其重要的组成部分，所以把这部分内容放在本节，接下来将介绍 Word2Vec 这一里程碑的模型体系结构和优化程序，它可用于从大型数据集中学习单词嵌入。通过 Word2Vec 学习的嵌入已被证明在各种下游自然语言处理任务上都是成功的。

考虑以下类似的句子：Have a good day 和 Have a great day。它们几乎没有不同的含义。如果构建一个详尽的词汇表（称其为 V），则其 $V=\{\text{Have,a,good,great,day}\}$。

现在，为 V 中的每个单词创建一个 One-Hot（单词独热编码向量）。单词独热编码向量的长度将等于 V 的大小（5）。除了索引中表示词汇表中相应单词的元素外，算法将有一个零向量。该特定元素将只有一个。下面的编码可以更好地说明这一点。

Have= [1,0,0,0,0]'; a = [0,1,0,0,0]'; good = [0,0,1,0,0]'; great = [0,0,0,1,0]'; day = [0,0,0,0,1]'('代表转置）。

尝试可视化这些编码，可以得到一个五维空间，其中每个单词占据一维，而与其余单词无关。这意味着 good 与 great 一样，这是不正确的。

算法的目标是使上下文相似的单词占据紧密的空间位置。在数学上，此类向量之间的角度的余弦值应接近 1，即角度接近 0，如图 3.2 所示。

这里是生成分布式表示，直观来看，笔者引入了一个单词对另一个单词的某种依赖性。在该词的上下文中的词将在这种依赖性中获得更大的权重。如前面提到的，在一个独热编

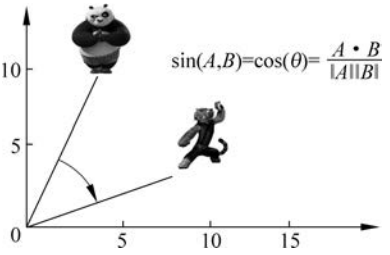

图 3.2 相似单词计算余旋度

码表示中，所有的字都是彼此独立的。

Word2Vec 是一种构造此类嵌入的方法。Word2Vec 的实现可以使用两种方法（都涉及神经网络）来获得：跳过语法（Skip-Gram）和通用单词袋（CBoW）。

CBoW 模型将每个单词的上下文作为输入，并尝试预测与上下文相对应的单词。例如 Have a great day。

假设输入神经网络的词为 great。需要注意，这里笔者尝试使用单个上下文输入单词 great 预测目标单词（day）。更进一步地，笔者使用输入字的一种独热编码，并与目标字的一种独热编码（day）相比，测量输出误差。在预测目标词的过程中，模型学习目标词的向量表示。详细的结构如图 3.3 所示。

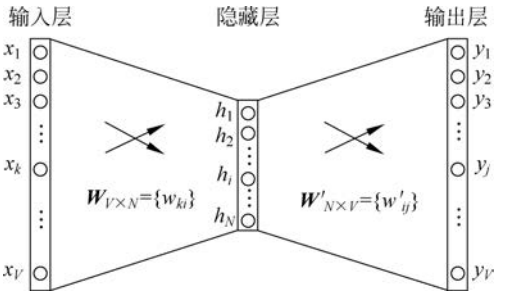

图 3.3 编码结构

其中，输入或上下文词是一个长度为 V 的独热编码向量。隐藏层包含 N 个神经元，输出也是 V 长度向量，其元素为 Softmax 值。隐藏层神经元仅将输入的加权总和复制到下一层。没有像函数 tanh 或 ReLU 这样的激活。唯一的非线性是输出层中的 Softmax 计算，但是，以上模型使用单个上下文词来预测目标。笔者可以使用多个上下文词来做同样的事情，如图 3.4 所示。

图 3.4 所示的模型采用 C 个上下文词。当 $\boldsymbol{W}_{v\times n}$ 用于计算隐藏层输入时，对这些上下文词 C 输入取平均值，因此，读者已经看到了如何使用上下文单词生成单词表示形式，但是，还有另一种方法可以做到这一点：使用目标词（为了生成其表示形式）来预测上下文，并在此过程中生成相应的表示形式。Skip-Gram 模型的变体可以做到这一点。

Skip-Gram 模型如图 3.5 所示。

看起来上下文 CBoW 模型刚刚被翻转，在某种程度上这样理解是对的。算法将目标词输入网络，该模型输出 C 个概率分布。这是什么意思？对于每个上下文位置，算法获得 C 个 V 维度的概率分布，每个单词都有一个。在这两种情况下，网络都使用反向传播进行学习。总体来讲，两者都有自己的优点和缺点。Skip-Gram 可以很好地处理少量数据，并且可以很好地代表稀有单词；CBoW 速度更快，对于更频繁的单词具有更好的表示。

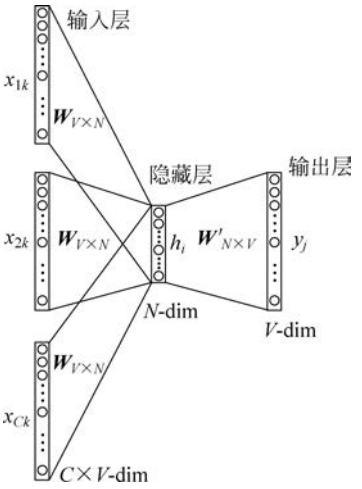

图 3.4　多个上下文词处理的示意图

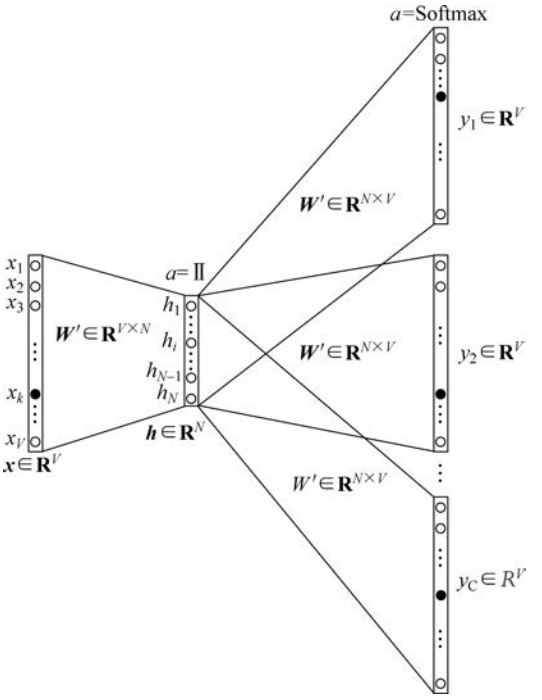

图 3.5　利用 Skip-Gram 生成表示形式结构图

3.4　小结

本章介绍了最初解决自然语言任务时利用的一些简单的规则，例如词频、聚合度、自由度和编辑距离等，发展到后面可以利用一些机器学习方法出色地完成一些相对简单的任务，不过随着社会的发展，算法面临的任务也越来越复杂。如今是大数据时代，机器学习的方法所带来的效果也遇到很多瓶颈，同时在社会快速发展的推动下，现如今的计算力和硬件水平也得到了快速发展，模型的验证周期也更短，这些都是深度学习模型快速发展的“催化剂”，同时本章还介绍了目前流行的浅层深度学习模型。