自然语言处理导论

张奇 桂韬 黄萱菁

2022 年 12 月 12 日

.

数学符号

数与数组

α	标量
$\boldsymbol{\alpha}$	向量
$\boldsymbol{A}$	矩阵
$\mathbf{A}$	张量
$\boldsymbol{I}_n$	n 行 n 列单位矩阵
$\boldsymbol{v}_w$	单词 w 的分布式向量表示
$\boldsymbol{e}_w$	单词 w 的独热向量表示：[0,0,...,1,0,...0]，w 下标处元素为 1

索引

α_i	向量 $\boldsymbol{\alpha}$ 中索引 i 处的元素
α_{-i}	向量 $\boldsymbol{\alpha}$ 中除索引 i 之外的元素
$w_{i:j}$	序列 w 中从第 i 个元素到第 j 个元素组成的片段或子序列
A_{ij}	矩阵 $\boldsymbol{A}$ 中第 i 行、第 j 列处的元素
$\boldsymbol{A}_{i:}$	矩阵 $\boldsymbol{A}$ 中第 i 行
$\boldsymbol{A}_{:j}$	矩阵 $\boldsymbol{A}$ 中第 j 列
$\boldsymbol{A}_{ijk}$	三维张量 $\mathbf{A}$ 中索引为 (i,j,k) 处元素
$\mathbf{A}_{::i}$	三维张量 $\mathbf{A}$ 中的一个二维切片

集合

$\mathbb{A}$	集合
$\mathbb{R}$	实数集合
$0, 1$	含 0 和 1 的二值集合
$0, 1, ..., n$	含 0 和 n 的正整数的集合
$[a, b]$	a 到 b 的实数闭区间
$(a, b]$	a 到 b 的实数左开右闭区间

线性代数

$\boldsymbol{A}^\top$	矩阵 $\boldsymbol{A}$ 的转置
$\boldsymbol{A} \odot \boldsymbol{B}$	矩阵 $\boldsymbol{A}$ 与矩阵 $\boldsymbol{B}$ 的 Hardamard 乘积
det$\boldsymbol{A}^\top$	矩阵 $\boldsymbol{A}$ 的行列式
$[\boldsymbol{x}; \boldsymbol{y}]$	向量 $\boldsymbol{x}$ 与 $\boldsymbol{y}$ 的拼接
$[\boldsymbol{U}; \boldsymbol{V}]$	矩阵 $\boldsymbol{A}$ 与 $\boldsymbol{V}$ 沿行向量拼接
$\boldsymbol{x} \cdot \boldsymbol{y}$ 或 $\boldsymbol{x}^\top \boldsymbol{y}$	向量 $\boldsymbol{x}$ 与 $\boldsymbol{y}$ 的点积

微积分

$\frac{\mathrm{d}y}{\mathrm{d}x}$	y 对 x 的导数
$\frac{\partial y}{\partial x}$	y 对 x 的偏导数
$\bigtriangledown_{\boldsymbol{x}} y$	y 对向量 $\boldsymbol{x}$ 的梯度
$\bigtriangledown_{\boldsymbol{X}} y$	y 对矩阵 $\boldsymbol{X}$ 的梯度
$\bigtriangledown_{\mathbf{X}} y$	y 对张量 $\mathbf{X}$ 的梯度

概率与信息论

$a \perp b$	随机变量 a 与 b 独立
$a \perp b \mid c$	随机变量 a 与 b 关于 c 条件独立
$P(a)$	离散变量概率分布
$p(a)$	连续变量概率分布
$a \sim P$	随机变量 a 服从分布 P
$\mathbb{E}_{x \sim P}[f(x)]$ 或 $\mathbb{E}[f(x)]$	$f(x)$ 在分布 $P(x)$ 下的期望
Var$(f(x))$	$f(x)$ 在分布 $P(x)$ 下的方差
Cov$(f(x), g(x))$	$f(x)$ 与 $g(x)$ 在分布 $P(x)$ 下的协方差
$H(f(x))$	随机变量 x 的信息熵
$D_{KL}(P \parallel Q)$	概率分布 P 与 Q 的 KL 散度
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	均值为 $\boldsymbol{\mu}$、协方差为 $\boldsymbol{\Sigma}$ 的高斯分布

数据与概率分布

$\mathbb{X}$	数据集
$\boldsymbol{x}^{(i)}$	数据集中第 i 个样本（输入）
$\boldsymbol{y}^{(i)}$ 或 $y^{(i)}$	第 i 个样本 $\boldsymbol{x}^{(i)}$ 的标签（输出）

函数

$f : \mathcal{A} \longrightarrow \mathcal{B}$	由定义域 $\mathcal{A}$ 到值域 $\mathcal{B}$ 的函数（映射）f
$f \circ g$	f 与 g 的复合函数
$f(\boldsymbol{x}; \boldsymbol{\theta})$	由参数 $\boldsymbol{\theta}$ 定义的关于 $\boldsymbol{x}$ 的函数（也可以直接写作 $f(\boldsymbol{x})$，省略 $\boldsymbol{\theta}$）
$\log x$	x 的自然对数函数
$\sigma(x)$	Sigmoid 函数 $\dfrac{1}{1+exp(-x)}$
$\|\|\boldsymbol{x}\|\|_p$	$\boldsymbol{x}$ 的 L^p 范数
$\|\|\boldsymbol{x}\|\|$	$\boldsymbol{x}$ 的 L^2 范数
$\mathbf{1}^{\text{condition}}$	条件指示函数：如果 condition 为真，则值为 1；否则值为 0

本书中常用写法

- 给定词表 $\mathbb{V}$，其大小为 $|\mathbb{V}|$
- 序列 $x = x_1, x_2, ..., x_n$ 中第 i 个单词 x_i 的词向量 $\boldsymbol{v}_{x_i}$
- 损失函数 $\mathcal{L}$ 为负对数似然函数：$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{(x,y)} \log P(y|x_1...x_n)$
- 算法的空间复杂度为 $\mathcal{O}(mn)$

目 录

3. 句法分析

任何人类语言都具备构造无限数量句子的能力[1]，可以通过增加形容词、副词、关系小句、介词短语等方法把任意的句子进一步的进行创造。因此，无法将一种语言按照词典的方式将所有句子存储起来。但是，通过语言学研究发现，句子并非词语的随意组合，而是按照一定规则结合起来的离散单位组成[2]。**句法**（Syntax）就是研究这些自然语言中不同成分组成句子的方式以及支配句子结构并决定句子是否成立的规则。句法反映了句子中词、词序以及层级结构之间的关系。通过句法规则，可以将词语组合成短语，将短语组合成句子，或者确定某个句子是否符合某一语言正确的词序。句法规则还阐释了词的分组与其意义的相关性，并在一定程度上解释了语言的无限性。句法分析具有非常重要的作用，是自然语言处理中的经典问题。

本章首先介绍语言学中句法基本概念，在此基础上以介绍成分句法分析算法、依存句法分析算法以及常用的句法分析语料库。

3.1 句法概述

句法是现代语言学研究中的重要课题，有大量的句法理论（syntactic theory）相关研究。句法理论在很多语法理论研究中也都是占据了主要部分。每种自然语言都可以看作由正确句子构成集合。通常一种自然语言中的包含的词和语素可以达到几万甚至几十万，而句子的长度可以超过 100 个单词，甚至可以认为是无限长，因此不可能采用穷举的方式将一个语言中所有正确的句子保存起来。语法就是指自然语言中句子、短语以及词等语法单位的语法结构与语法意义的规律[3]。根据语法就可以判断不同成分组成句子的方式以及决定句子是否成立。语法学（Garmmmar）的外延相关广泛，甚至有些语言学家认为音系学和语义学都包含在语法学中。在本书中我们采用比较狭义的语法学定义，既认为语言学不包含音系学和语义学，因此狭义的语法学的研究基本等同于句法学。语言学家自 19 世纪 50 年代以来，构建了大量表达明确并且形式化的语法理论，对自然语言句法分析提供了理论支撑。

语法理论在构建时，一个重要的问题是该理论是基于成分关系还是基于依存关系。如果一种语法理论基于成分关系，那么该理论就属于**成分语法**（Constituent Grammar），也称短语结构语法（Phrase Structure Grammar）。成分语法主要包含：范畴语法（Categorical Grammar）、词汇功能语

法（Lexical Functional Grammar）、最简方案（the Minimalist Program）等。如果一个语法理论基于依存关系，那么该理论就属于**依存语法**（Dependency Grammar）。依存语法主要包含：文本-意义理论（Meaning Text Theory）、词格理论（Lexicase）、功能生成描述理论（Functional Generative Description）等。

本节将分别针对成分语法和依存语法理论进行简单介绍。

3.1.1 成分语法理论概述

乔姆斯基（Chomsky）在其 1957 年发表的《句法结构》[4] 中奠定了成分语法的基础。**成分**（Constituent）又称短语结构，是指一个句子内部的结构成分。成分可以独立存在，或者可以用代词替代，又或者可以在句子中不同位置移动。

例如：他正在写一本小说。

“一本小说”是一个成分

在此基础上，根据不同成分之间是否可以进行相互替代而不会影响句子语法正确性，可以进一步的将成分进行分类，某一类短语就属于一个**句法范畴**。比如“一本小说”、“一所大学”等都属于一个句法范畴：名词短语（None Phrase，NP）。句法范畴不仅仅包含名词短语（NP）、动词短语（VP）、介词短语（PP）等短语范畴，也包含名词（N）、动词（V）、形容词（Adj）等词汇范畴。除此之外还包含功能范畴（包括：冠词、助动词等）。

句法范畴之间不是完全对等的，而是具有层级关系。例如：一个句子可以由一个名词短语和一个动词短语组成，一个名词短语可以由一个限定词和一个名词组成，一个动词短语又可以由一个动词和一个名词短语组成。我们可以定义**短语结构规则**（Phrase Structure Rules）对句法范畴间的关系进行形式化描述：

(1) S -> NP VP
(2) NP -> Det N
(3) VP -> V NP

短语结构规则又称改写规则或重写规则，通常可以用 $X \rightarrow YZW \cdots$ 表示，其中 X 表示短语名称，“$\rightarrow$”表示“改写为”，“$YZW \cdots$”定义了短语 X 的结构，如果 YZW 是短语，则需要构造出它们的规则。

成分语法就是由句法范畴以及短语结构规则定义的语法。由于短语结构规则具有递归性，可以使短语和句子无限循环组合。这也说明了语言的创造性和无限性。如图3.1和图3.2所示，一个句子根据成分语法分析得到的层级结构，可以使用成分语法树进行表示。

由于成分语法局限于表层结构分析，不能彻底解决句法和语义问题，因此存在非连续成分、结构歧义等问题。如图3.3和图3.4所示，对于句子“The boy saw the man with telescope”有两种可能的合法的句法结构树，不同的树结构对应不同的语义。第一种句法树表示“男孩使用望远镜看到了这个男人”，第二种句法树表示“男孩看到了一个拿着望远镜的男人”。

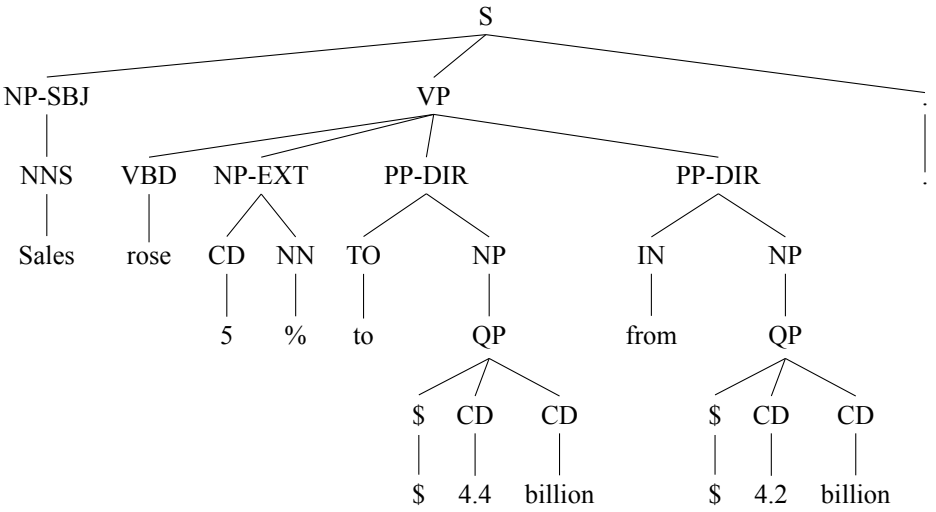


图 3.1　Penn Treebank 3.0 成分句法树样例

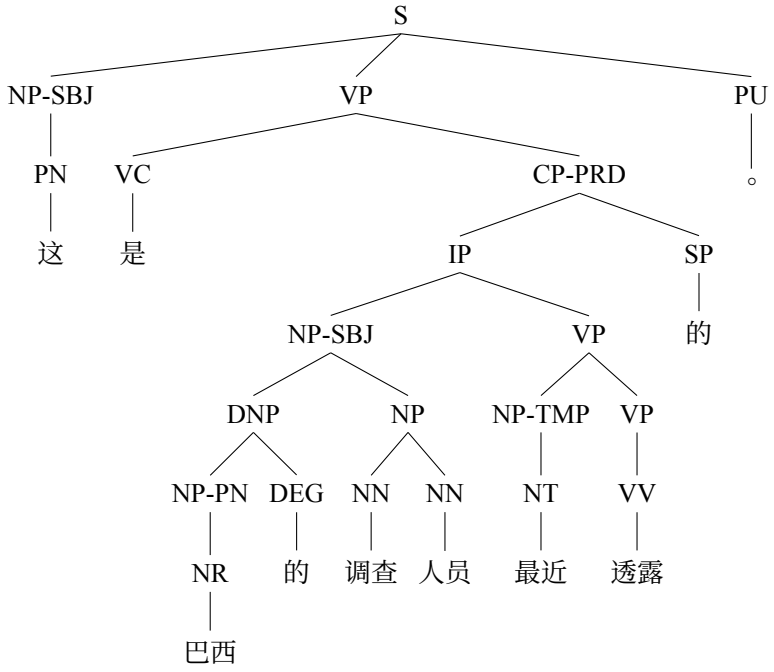


图 3.2　Chinese Treebank 7.0 成分句法树样例

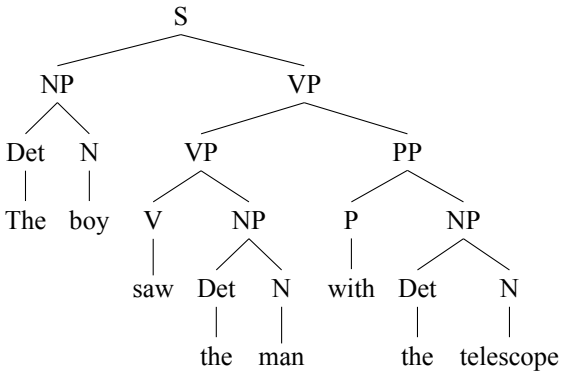


图 3.3 句子 The boy saw the man with telescope 的第一种成分句法树

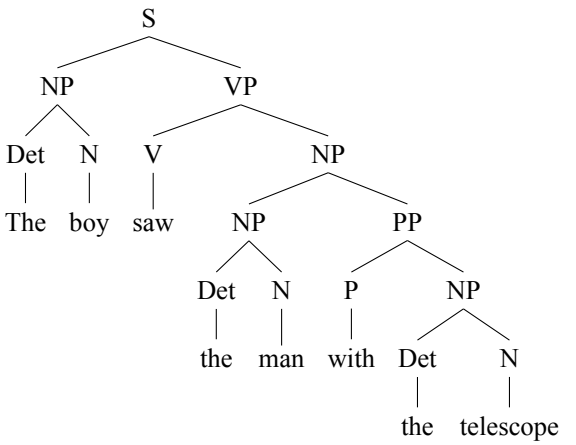


图 3.4 句子 The boy saw the man with telescope 的第二种成分句法树

3.1.2 依存语法理论概述

吕西安·泰尼埃（Lucien Tesnière）于 1959 年发表的《句法结构分析》奠定了句法以依存关系研究的基础[5]。在基于依存关系的语法中，句子中的每个成分对应句法结构中的唯一一个节点。两个成分之间的依存关系是二元的非对称关系，具有方向性，一个成分是中心语，另一个成分依附于中心语存在，关系从中心语成分指向依存成分。中心成分称为**中心词或支配者**（governor，regent，head），依存成分也称为**修饰词或从属者**（modifier，subordinate，dependency）。例如：“读书”中“读”是中心语，“书”依存于“读”。有多种方式可以用于表示依存关系，如下图所示：

两个单词之间是否存在依存关系？单词之间谁处于支配地为？谁处于从属地为？建立这些词与词之间关系的依据是什么？很多依存句法理论从不同方面对上述问题进行了回答。配价（Valency）

图 3.5 “读书”的依存关系表示实例

理论是其中最为经典的论著之一。这里“价”的概念是由泰尼埃从化学中的“化合价”的概念引入语言学研究。价是词语的一个属性，表示某个词语与其他词结合的能力。配价模式 (Valency pattern) 则是描述了某一个具有特定意义的词的出现语境，以及当一个词出现在一个特定的模式下时，还有哪些词语会出现在这个模式下及其语义角色。通过对不同词类的支配和被支配能力（配价）以及词类间依存关系类型的定义，就可以得出某种具体的依存句法。

利用配价理论，可以将汉语里的动词 V 根据其价数，即属于几价动词，分为以下四类：

(1) **零价动词**不强制与某个行动元关联的动词，例如：地震、刮风、下雨、下雪……
(2) **一价动词**强制与一个行动元关联的动词，例如：病、醉、休息、咳嗽、游泳
(3) **二价动词**强制与两个行动元关联的动词，例如：爱、采、参观、讨论、改良……
(4) **三价动词**强制与三个行动元关联的动词，例如：给、送、告诉、退还、赔偿……

有一种特殊的关系是并列关系，构成这种关系的元素之间不存在支配与被支配关系。但是为了能够使得与其他的关系统一，也通过并列关系标记将其纳入句法树中，通常将第一个词作为中心语，其余的词作为该词的从属成分。这种情况下，在树结构上表现出来的层级关系是一种伪层级。

词语间的依存关系还可以根据语法关系定义为不同的类型，Carroll 等人[6] 将依存关系细分为了 20 种，并给出了关系之间的层级结构。Marneffe 等人[7] 在上述工作的基础上对依存关系进行了进一步的细化，定义了 48 种依存关系，主要分为论元依存关系和修饰语依存关系两大类。图3.6给了一个包含依存关系类型的句法树的样例。

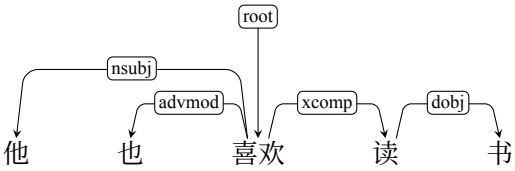


图 3.6 句子“他也喜欢读书”的依存句法树

通常依存句法树引入 *root* 做为句子或者单棵树的主要支配者，是树的根节点。一般情况下动词是 *root* 节点的直接从属成分，其余节点应该直接或者间接依存于动词节点。没有动词的句子除外，但是应存在一个句子成分做为 *root* 的从属成分。依存句法树中的每个节点只能有一个支配者，

但是可以有多个从属者。一个符合语法的句子中，所有节点应该是相连的，不允许存在游离在外的节点。

依存语法中根据其依存成分与中心语或姐妹成分在语序上的关系，可以分为符合投射性原则和违反投射性原则两类。在依存层级中如果每个依存成分与其中心或姐妹成分相邻，那么该依存层级就是符合投射性原则（projectivity）；如果依存成分的相邻成分使其与中心语分类那么就是违反投射性原则的。

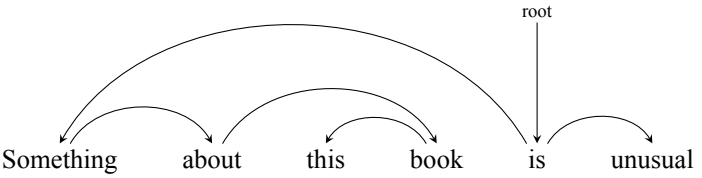


图 3.7　符合投射原则依存句法树样例

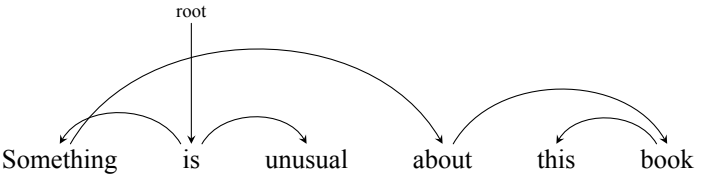


图 3.8　违反投射原则依存句法树样例

图3.7中没有任何两线交叉，因此是投射性的。图3.8中成分“about this book”与其中心语不相邻，因此存在交叉线，违反了投射原则。这种现象在依存句法中通常称为远距离依赖（long-distance dependencies）。

3.2 成分句法分析

成分句法分析（Constituency Parsing）就是对给定句子根据成分语法中制定的规则构建其所对应的结构树的过程。由第3.1节中关于成分语法的基本概念和组成部分的介绍可以知道，成分语法主要包含句法范畴和短语结构规则两个部分。句法范畴又包含短语范畴、词汇范畴和功能范畴。短语结构规则通常可以由 $X \rightarrow YZW \cdots$ 形式进行表示。成分语法与数学系统中的上下文无关文法（Context-Free Grammar，简称 CFG）组成非常类似，因此上下文无关文法是目前最常用的模拟英语以及其他语言的成分语法的数学系统。上下文无关文法 G 包含以下四个部分：

终结符 $\sum$（Terminal Symbols）：与语言中词汇对应的符号，用 u, v, w 等小写罗马字母表示；
例如：上海，walk

非终结符 N（Non-terminals）：对应词组等聚集类或概括性符号，用 A,B,C 等大写字母表示；

例如：NP（名词短语），VP（动词短语），N（名词），介词（P）

初始符 S（Start symbol）：语法中指定的初始符，通常用 S 表示；

规则 R（rules）：对应语法中的短语结构规则，从初始符号开始可以造出合法句子的规则集合，在上下文无关语法中，一个规则的左边是一个单独的非终结符，右边是一个或者多个终结符或非终结符组成的有序序列 $(\sum \cup N)*$，用 α, β, γ 等小写希腊字母表示 $(\sum \cup N)*$；

例如：S→ NP VP，NP→ Det Adj N，Det → the

对一个句子进行句法分析的过程可以看做对于一个句子搜索所有可能的路径空间，从中发现正确的句法树的过程。搜索过程受到两个约束限制，一个是句子本身，另外一个是语法。根据成分句法树的定义，叶子节点一定是句子中的单词，中间节点与其子节点需要符合语法定义。这两种约束也产生了大多数分析算法所采用的搜索策略：自底向上（Bottom-up）和自顶向下（Top-down）。

由于句法结构具有歧义，因此句法分析中最重要的工作之一也是如何消除歧义。成分语法中的结构歧义主要有两种：**附着歧义**（Attachment ambiguity）以及**并列连接歧义**（Coordination ambiguity）。图3.3和图3.3所给出的关于句子“The boy saw the man with telescope”的两种分析结果就是附着歧义的一个例子，其核心就是“with telescope”附着于动词 saw 还是名词短语 the man。并列连接歧义也是常见的结构歧义之一，例如，短语“重要政策和措施”中“重要”可以修饰“政策和措施”整体，也可以仅修饰“政策”，就可以表示为两种层级结构，“[重要 [政策和措施]]”和“[重要政策] 和 [措施]”。但是由于结构歧义通常伴随语义上的变化，因此句法结构歧义消除通常需要依赖统计知识、语义知识或语用知识才能更好的完成。

3.2.1 基于上下文无关文法的成分句法分析

在给定上下文无关文法 G 的情况下，对于给定的句子 $W = \{w_1, w_2, \cdots, w_n\}$，输出其对应的句法结构树，通常有两大类搜索方法：自顶向下和自底向上。自顶向下搜索试图从根节点 S 出发，搜索语法中的所有规则直到叶子节点，并行构造所有的可能的树。自底向上的方法是从输入的单词开始，每次是用语法规则，直到成功构造了以初始符 S 为根的树。针对这两大类算法，本节中将分别介绍 CYK 分析算法和移进-规约分析算法。

1. CYK 成分句法分析算法

CYK 算法（CYK 是 Cocke–Younger–Kasami 的缩写，有时也称为 CKY）是由 John Cocke、Daniel Younger 以及 Tadao Kasami 分别独立提出的基于动态规划思想的自底向上语法分析算法[8–10]。CYK 算法要求所使用的语法必须符合**乔姆斯基范式**（Chomsky Normal From，CNF），其语法规则被限制为只具有 $A \to BC$ 或 $A \to w$ 这种形式。由于任何上下文无关语法都可以转换为相应的 CNF 语法，因此这种限制并不会对表达能力造成损失。但是这种限制使得基于表的分析算法变得简单和非常易于实现。

根据 CNF 语法形式，句法树的叶子节点为单词，单词的父节点为词性符号，在词性符号层之

上每一个非终结符都有两个儿子节点。因此 CYK 算法采用了二维矩阵对整个树结构进行编码。对于一个长度为 n 的句子，构造一个 $(n+1)\times(n+1)$ 的二维矩阵 $\boldsymbol{T}$，矩阵主对角线以下全部为 0，主对角线上的元素由输入句子的终结符号 (单词) 构成，主对角线以上的元素 T_{ij} 包含由文法 G 的非终结符构成的集合，这个集合表示输入句子中横跨在位置 i 到 j 之间的单词的组成成分。输入句子中索引从 0 开始，索引位于输入句子的单词之间，也可以看成单词之间的间隔指针（例如：$_0$ 她 $_1$ 喜欢 $_2$ 跳 $_3$ 芭蕾 $_4$ ）。

CYK 算法按照平行于主对角线的方向，逐层向上填写矩阵中的各个元素 T_{ij}。如果存在一个正整数 k（$i+1\leqslant k\leqslant j-1$），在文法规则集中具有产生式 $A\to BC$ 并且 $B\in T_{ik}, C\in T_{kj}$，那么将 A 合并到矩阵表的 T_{ij} 中。为了方便根据矩阵 T 输出句法分析结果，在矩阵 T 每个单元 T_{ij} 中不仅记录非终结符的集合，还要记录推导路径。逐层计算完成后，判断一个句子由文法 G 所产生的充要条件是：$T_{0n}=S$。具体过程如算法3.1所示。

代码 3.1: CYK 句法分析算法

输入: 语法信息 G 和句子 $w_1w_2\cdots w_n$
输出: 句法树矩阵 T
// 初始化
for $i=1$ **to** n **do**
  $T_{ii}=w_i$; // 主对角线上依次放入单词 w_i ;
  foreach $A|A\to w_i$ **do**
    $T_{(i-1)i}=T_{(i-1)i}\cup A,(A\to w_i\in G)$; // 依次放入单词 w_i 的所有词性;
  end
end
// 平行于主对角线逐层计算
for $d=2$ **to** n **do**
  for $i=0$ **to** $n-d$ **do**
    j = i+d ;
    for $k=i+1$ **to** $j-1$ **do**
      if $A\to BC\in G$ and $B\in T_{ik}$ and $C\in T_{kj}$ **then**
        $T_{ij}=T_{ij}\cup\{A,(k,B,C)\}$; // (k,B,C) 用于保存推导路径;
      end
    end
  end
end
return T

以下通过一个实例来说明 CYK 算法的具体流程。给定如下符合乔姆斯基范式的文法 G 如下:

非终结符号集合：$N = \{S, N, P, V, VP\}$
终结符号集合：$\sum = \{$ 她，喜欢，跳，芭蕾 $\}$
规则集合：$R = \{$ (1) S -> P VP;　　(2) VP -> V V;　　(3) VP -> VP N;
(4) P -> 她;　　(5) V -> 喜欢;　　(6) V -> 读;　　(7) N -> 芭蕾 $\}$

图3.9给出了 CYK 算法分析句子“她喜欢跳芭蕾”的过程。首先初始化矩阵主对角线上的单词以及单词所对应的词性。在此基础上进行第二层 T_{02}, T_{13}, T_{24} 元素的计算，通过规则集合中 VP->V V 推导规则，可以得到在 T_{13} 添加 VP 以及相应的路径信息。针对第三层 T_{03}, T_{14} 元素，虽然针对 T_{03} 元素，可以根据推导规则 S->P VP 添加非终结符 S，但是由于非终结符 S 是只能出现在 T_{0n} 中，因此 T_{03} 不添加 S。T_{14} 元素中根据通过规则 VP->VP N 添加 VP 信息。最后一层 T_{04} 根据 S->P VP 规则，添加 S 及其路径信息并完成分析。

2. 移进-规约成分句法分析算法

移进-规约成分句法分析算法的基本思想是从左到右扫描输入的包含单词词性对的句子，使用堆栈和一系列的移进（Shift）和规约（Reduce）操作序列构建句法树[11]。

算法初始时堆栈 S 为空，队列 Q 中包含整个句子所有单词。最终通过一系列的操作，在算法结束时堆栈 S 中包含一个完整的句法树，队列 Q 为空。所采用的操作包含以下四个：

移进（Shift）： 将非空队列 Q 最左端的单词移入堆栈 S 中。

规约（Reduce）： 根据推导规则，将堆栈 S 中的最顶端的若干元素移出（根据推导规则右侧所包含非终结符数量，在堆栈中移除相应数量元素），然后将利用推导规则产生的新结构压入堆栈中。

接受（Accept）： 队列中所有单词都已被移到堆栈中，并且堆栈中只剩下一个由非终结符 S 为根的树，分析成功。

拒绝（Reject）： 队列中所有单词都已被移到堆栈中，但是堆栈中并非只有一个以非终结符 S 为根的树，并且无法继续规约，分析失败。

如何根据当前堆栈 S 和队列 Q 中的状态，选择下一步的操作是移进-规约分析算法中最重要的部分。由于移进和规约操作并不是完全互斥的，在很多状态下两种操作都可以选择，这就造成了移进规约冲突（Shift Reduce Conflict）。在自然语言这种非确定性语法下，这种冲突不可避免。在基于规则和策略的移进-规约分析方法中，当有多种规约可能时分析算法需要选择其中某个，当移进和规约都可以时也需要选择执行哪个动作。分析算法可以通过设置执行策略来解决这些冲突，例如采用深度优先搜索策略，只有在不能规约时才移进，有多种规约可能时选择最长的文法规则等策略。由于移进和规约操作不可撤销，因此即便输入的句子是符合语法的情况下，由于策略选择的问题，也可能分析失败，堆栈中不能规约到只有 S 为根的树的情况。为了解决这个问题也可以采用回溯策略，当分析失败时回溯顺次尝试不同选择。

为了方便理解移进-规约分析算法，以下使用在上节介绍 CYK 算法时相同的文法 G，也以“她喜欢跳芭蕾”句子为例，介绍移进-规约算法的分析具体过程，如图3.10和3.11所示。在初始化时，队

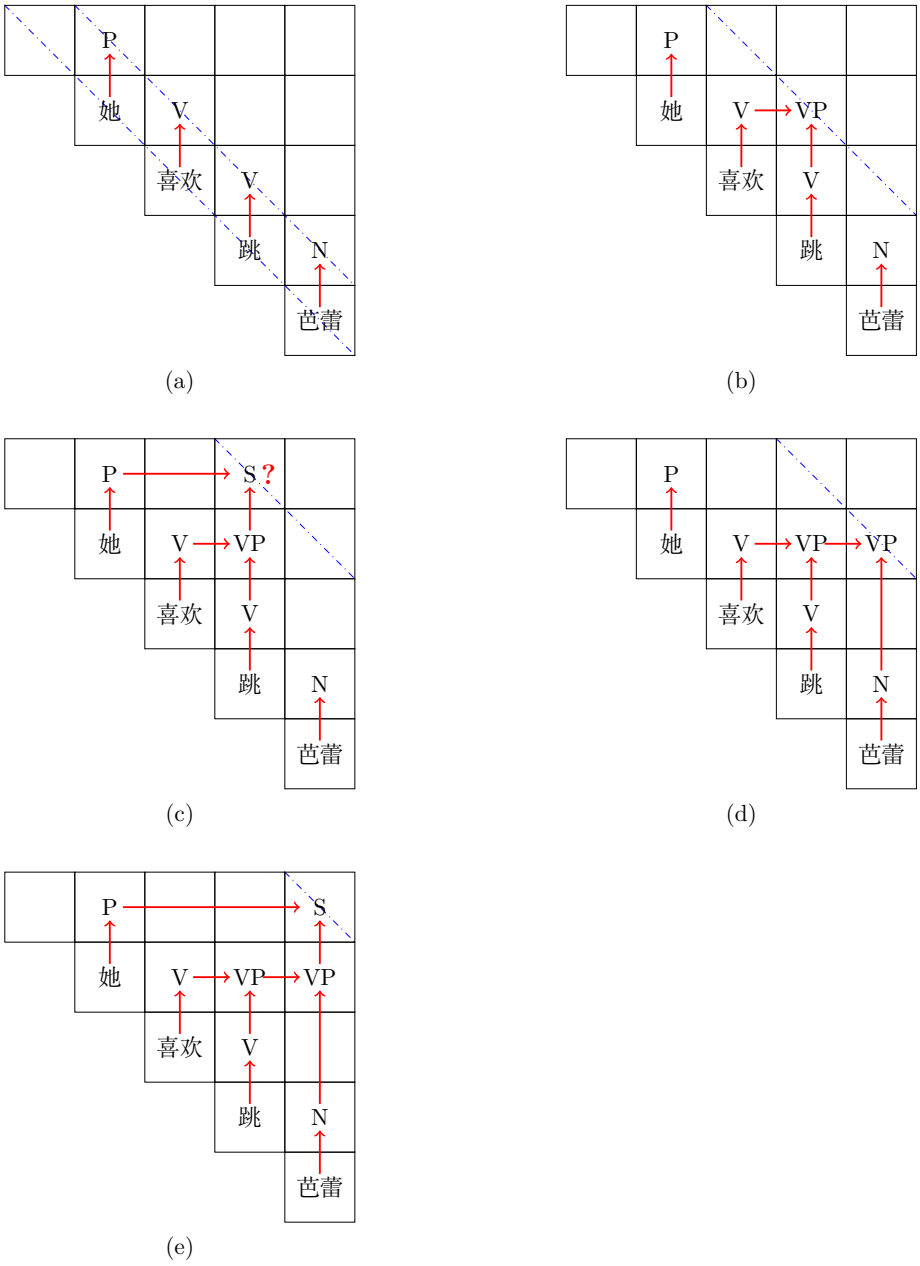


图 3.9 CYK 算法分析实例

列 Q 中保存句子中的所有单词，堆栈 S 为空。第 1 步进行移进操作，将句子的第一个单词“她”压入堆栈中。第 2 步利用文法 G 中的“P-> 她”规则生成子树，并将以 P 为根节点的子树压入堆栈中。第 3-6 步依次移进“喜欢”和“跳”并利用文法规则生成子树。接下来可以进行移进操作将句子中“芭蕾”压入栈顶，也可以利用文法“VP -> V V”进行规约操作，面临了移进规约冲突问题。在这里我们采用深度有限搜索策略，选择规约操作，并将生成的以 VP 为顶点的子树压入栈顶。第 8 步采用移进操作，将最后一个单词压入栈顶。此后通过第 9-11 步一系列的规约操作，最终生成的以 S 为根节点句法树，并保存在栈中。此时队列为空，最后执行接受操作，分析成功。

	堆栈	动作	队列
0.	∅	∅	她 喜欢 跳 芭蕾
1.	她	shift	喜欢 跳 芭蕾
2.	P | 她	reduce	喜欢 跳 芭蕾
3.	P 喜欢 | 她	shift	跳 芭蕾
4.	P V | | 她 喜欢	reduce	跳 芭蕾
5.	P V 跳 | | 她 喜欢	shift	芭蕾
6.	P V V | | | 她 喜欢 跳	reduce	芭蕾
7.	P VP | / \\ 她 V V | | 喜欢 跳	reduce	芭蕾
8.	P VP 芭蕾 | / \\ 她 V V | | 喜欢 跳	shift	∅

图 3.10 移进规约成分句法分析算法分析过程实例

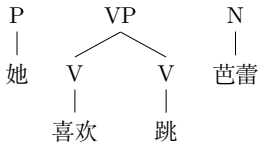
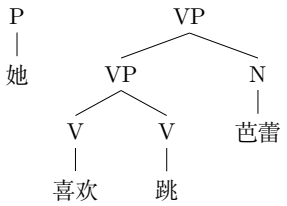
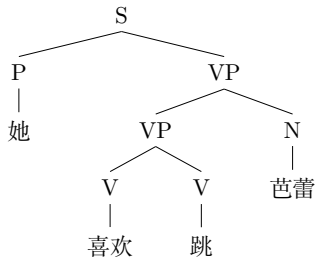
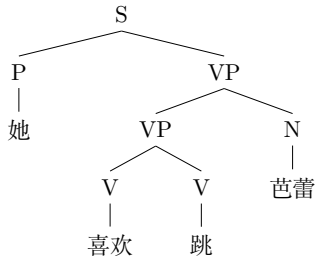
	堆栈	动作	队列
9.		reduce	∅
10.		reduce	∅
11.		reduce	∅
12.		accept	∅

图 3.11　移进规约成分句法分析算法分析过程实例（续）

3.2.2 基于概率上下文无关文法的成分句法分析

上下文无关文法虽然比较简单且容易理解，但是对于句子结构歧义无法很好的处理。比如句子“He eat soup with spoon”具有两种可能的句法结构树，并且两种树结构都符合语法（如图3.12所示)。使用上下文无关文法很难从这两种句法树中进行选择。基于概率上下文无关文法（Probabilistic Context-Free Grammar，PCFG）的句法分析则可以结合规则方法和统计方法，在一定程度上解决歧义问题。

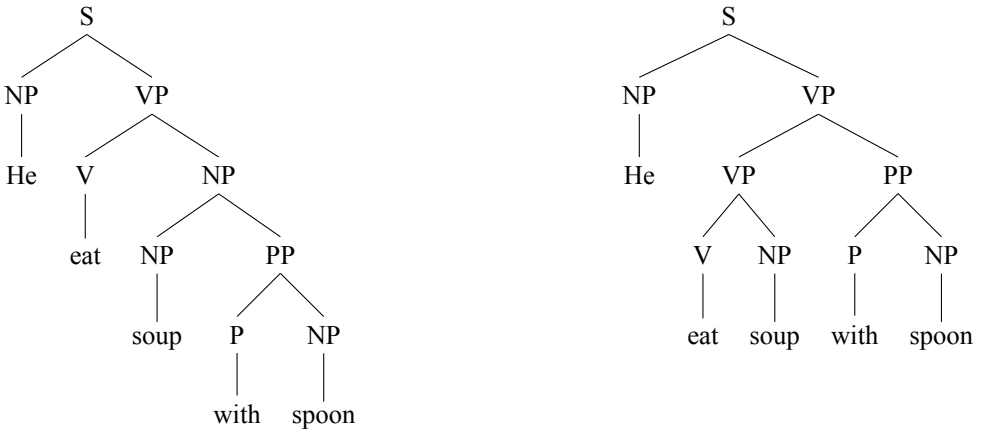


图 3.12 句子“He eat soup with spoon”的成分语法树

PCFG 是 CFG 的扩展，因此 PCFG 的文法也是由终结符集合 $\sum$、非终结符集合 N、初始符 S 以及规则集合 R 组成。只是在 CFG 的基础上对每条规则增加了概率，其规则用如下形式表示：

$$A \to \alpha, p$$

其中 A 为非终结符，α 为终结符，p 为 A 推导出 α 的概率，即 $p = P(A \to \alpha)$，该概率分布要满足如下条件：

$$\sum_{\alpha} P(A \to \alpha) = 1 \tag{3.1}$$

也就是说，每一个非终结符展开得到的概率之和为 1。

由于 PCFG 中每个规则中包含了概率信息 $P(A \to \alpha)$，因此可以根据一个句子及其句法分析树计算特定句法分析树的概率、句子的概率以及句子片段的概率。利用特定分析树的概率可以用于消除分析树的歧义。接下来面临的问题就是如何利用 PCFG 构建句子的最佳树结构以及如何得到规则的概率参数。在本节中，将针对上述问题进行介绍。

1. PCFG 句法分析树概率计算

句法分析树概率计算是指在给定 PCFG 文法 G、句子 $W = w_1 w_2 \cdots w_n$ 以及句法树 T 的情况下，计算句法分析树概率 $P(T)$。为了简化句法树概率计算，句法树概率计算还要应用以下三个独立假设：

(1) 位置不变性（Place in-variance）：子树的概率不依赖于该子树所在的位置；
(2) 上下文无关性（Context-free）：子树的概率不依赖于子树以外的单词；
(3) 祖先无关性（Ancestor-free）：子树的概率不依赖于子树的祖先节点。

基于上述独立性假设，一个特定句法树 T 的概率定义为该句法树 T 中用来得到句子 W 所使用的 m 个规则的概率乘积：

$$P(T) = \prod_{i=1}^{m} P(A_i \rightarrow \alpha) \tag{3.2}$$

假设给定如下 PCFG 文法：

G(S):	S → NP VP	1.0	NP → He	0.3
	VP → VP PP	0.8	NP → soup	0.3
	VP → V NP	0.2	NP → spoon	0.2
	NP → NP PP	0.2	V → eat	1.0
	PP → P NP	1.0	P → with	1.0

针对对于句子“He eat soup with spoon”的两种可能的句法结构包含概率的形式如图3.13所示。

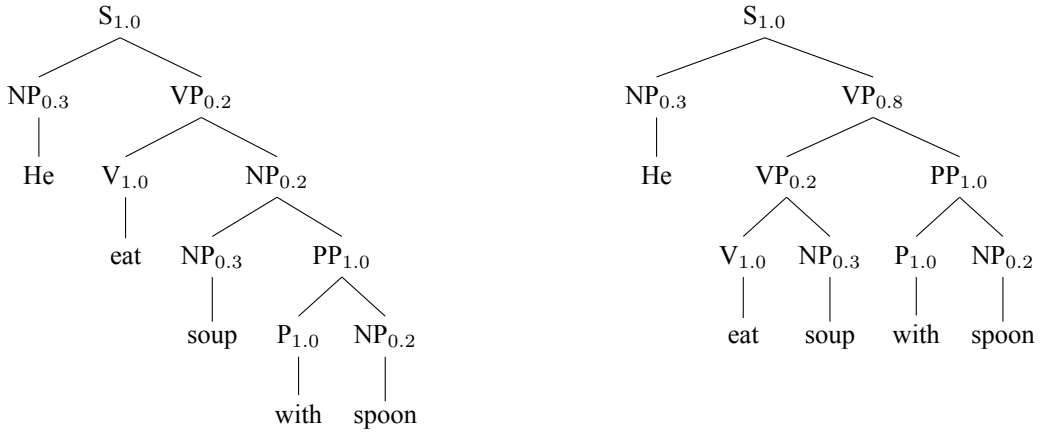


图 3.13　句子 He eat soup with spoon 标注推导规则概率的成分语法树

根据句法树概率计算公式3.2，图3.13所示的两个句法树的概率分别为：

$$\begin{aligned}P(T_{Left}) =& P(S \to NP\ VP) \times P(NP \to He) \times P(VP \to V\ NP) \times \\ & P(V \to eat) \times P(NP \to NP\ PP) \times P(NP \to soup) \times \\ & P(PP \to P\ NP) \times P(P \to with) \times P(NP \to spoon) \\ =& 1.0 \times 0.3 \times 0.2 \times 1.0 \times 0.2 \times 0.3 \times 1.0 \times 1.0 \times 0.2 = 0.00072\end{aligned}$$

$$\begin{aligned}P(T_{Right}) =& P(S \to NP\ VP) \times P(NP \to He) \times P(VP \to VP\ PP) \times \\ & P(VP \to V\ NP) \times P(V \to eat) \times P(NP \to soup) \times \\ & P(PP \to P\ NP) \times P(P \to with) \times P(NP \to spoon) \\ =& 1.0 \times 0.3 \times 0.8 \times 0.2 \times 1.0 \times 1.0 \times 0.3 \times 1.0 \times 0.2 = 0.00288\end{aligned}$$

由此可以选择图3.13右侧树结构，从而解决针对句子“He eat soup with spoon”的句法树歧义问题。

2. PCFG 的句子概率计算

句子概率计算是指在给定 PCFG 文法 G 的情况下，计算给定句子 W 的概率 $P(W|G)$。在第二章中我们介绍了 N 元语言模型，由于 N 元语言模型仅考虑上下文中相邻的单词，缺乏对远距离单词以及句法信息的考虑。基于 PCFG 文法的句子概率计算则可以将这些信息引入概率计算。$P(W|G)$ 表示句子 W 所有的句法分析树的概率之和。可以采用**内向算法**（inside algorithm）或**外向算法**（outside algorithm）通过动态规划算法计算得到句子的概率。

采用内向算法，首先定义内变量 $a_{ij}(A)$ 为非终结符 A 推导出 W 中子串 $w_i w_{i+1} \cdots w_j$ 的概率，即：

$$\alpha_{ij}(A) = P(A \to w_i w_{i+1} \cdots w_j) \tag{3.3}$$

句子 W 的概率则相应的记为 $a_{1n}(S)$，即：

$$P(W|G) = P(S \to W|G) = \alpha_{1n}(S) \tag{3.4}$$

$alpha_{ij}(A)$ 可通过如下递归公式计算得到：

$$\alpha_{ii}(A) = P(A \to w_i) \tag{3.5}$$

$$\alpha_{ij}(A) = \sum_{B,C} \sum_{i \leqslant k \leqslant j} P(A \to BC) \alpha_{ik}(B) \alpha_{(k+1)j}(C) \tag{3.6}$$

上述公式表示当 $i \neq j$ 时，子串 $w_i w_{i+1} \cdots w_j$ 可以被切分成两端：$w_i \cdots w_k$ 和 $w_{k+1} \cdots w_j$，前半部分 $w_i \cdots w_k$ 是由非终结符 B 推导出，后半部分 $w_{k+1} \cdots w_j$ 是由非终结符 C 推导出，即 $A \to BC \to w_i \cdots w_k w_{k+1} \cdots w_j$，这一推导过程的概率为 $P(A \to BC) \alpha_{ik}(B) \alpha_{(k+1)j}(C)$。具体过程如

算法3.2所示。

代码 3.2: 面向 PCFG 句子概率求解的内向算法

输入: PCFG G(S) 和句子 $w_1 w_2 \cdots w_n$
输出: $\alpha_{ij}(A), i \leqslant i \leqslant j \leqslant n$
for $i = 1$ to n **do**
  $\alpha_{ii}(A) = P(A \to w_i), 1 \leqslant i \leqslant n;$
end
for $j = 1$ to n **do**
  for $i = 1$ to $n - j$ **do**
    $\alpha_{i(i+j)}(A) = \sum_{B,C} \sum_{i \leqslant k \leqslant i+j-1} P(A \to BC)\alpha_{ik}(B)\alpha_{(k+1)(i+j)}(C);$
  end
end
return α

计算给定句子 W 的概率 $P(W|G)$ 也可以采用**外向算法（outside algorithm）**利用动态规划方法得到。外向变量 $\beta_{ij}(A)$ 定义为初始非终结符 S 在推导出句子 W 的过程中产生符号串 $w_1 w_2 \cdots w_{i-1}\, A\, w_{j+1}$ 的概率（$A \to w_i \cdots w_j$）：

$$\beta_{ij}(A) = P(S \to w_1 w_2 \cdots w_{i-1} A w_{j+1} \cdots w_n) \tag{3.7}$$

$\beta_{ij}(A)$ 可通过如下递归公式计算得到：

$$\beta_{1n}(A) = \begin{cases} 1, A = S \\ 0, A \neq S \end{cases} \tag{3.8}$$

$$\begin{aligned} \beta_{ij}(A) \quad = \quad & \sum_{B,C} \sum_{k \geqslant j} P(B \to AC)\alpha_{(j+1)k}(C)\beta_{ik}(B) \\ & + \sum_{B,D} \sum_{k \leqslant i} P(B \to DA)\alpha_{k(i-1)}(D)\beta_{kj}(B) \end{aligned} \tag{3.9}$$

上述公式表示当 $i = 1, j = n$ 时，如果 $A = S$，那么 $\beta_{1n}(A) = P(S \to W)$ 按照 PCFG 的定义 $P(S \to W) = 1$，即 $\beta_{1n}(A) = 1$。如果 $A \neq S$，同样按照 PCFG 的定义以及规范语法中不存在 $S \to A$ 的推导规则，则 $\beta_{1n}(A) = 0$。当 $i \neq 1$ 或 $j \neq n$ 时，如果在 S 推导出 W 的过程中出现了符号串 $w_1 w_2 \cdots w_{i-1}\, A\, w_{j+1} \cdots w_n$，那么根据上下文无关语法的性质一定存在 $B \to AC$ 或者 $B \to DA$ 的规则。因此构造上述递推公式。具体过程如算法3.3所示。

代码 3.3: 面向 PCFG 句子概率求解的外向算法

输入: PCFG G(S) 和句子 $w_1 w_2 \cdots w_n$

输出: $\beta_{ij}(A), i \leqslant i \leqslant j \leqslant n$

// 初始化

$$\beta_{1n}(A) = \begin{cases} 1, A = S \\ 0, A \neq S \end{cases};$$

for $j = n-1$ to 0 **do**

  for $i = 1$ to $n-j$ **do**

$$\beta_{ij}(A) = \sum_{B,C} \sum_{k \geqslant j} P(B \to AC) \alpha_{(j+1)k}(C) \beta_{ik}(B) + \sum_{B,D} \sum_{k \leqslant i} P(B \to DA) \alpha_{k(i-1)}(D) \beta_{kj}(B);$$

  end

end

return β

3. PCFG 的最佳树结构求解

最佳树结构求解是指对于给定句子 $W = \{w_1, w_2, \cdots, w_n\}$ 和 PCFG 文法 G，求解该句子的最佳树结构，即如何选择句法结构树使得其概率最大:

$$\hat{t} = \arg\max_{t \in \mathcal{T}_G(W)} P(t|W, G) \tag{3.10}$$

$\mathcal{T}_G(W)$ 表示句子 W 所有符合文法 G 的句法树。该问题可以通过利用基于概率的 CYK 算法进行。与 CYK 算法一样，概率 CYK 算法也要求所使用的语法必须符合乔姆斯基范式（CNF），其规则被限制为 $A \to BC$ 或 $A \to w$ 两种形式。与 CYK 算法一样，概率 CYK 算法也将输入的句子表示为单词之间带有索引号的句子形式。

例如：$_0$ He $_1$ eat $_2$ soup $_3$ with $_4$ spoon $_5$。

针对句子长度为 n 的句子，概率 CYK 算法同样也使用 $(n+1) \times (n+1)$ 的矩阵 $\boldsymbol{T}$ 的上三角形部分进行存储。T_{ij} 表示输入句子中横跨在位置 i 到 j 之间的单词的组成成分，包含由文法 G 的非终结符构成的集合以及以该非终结符为根的句法树的概率。为了方便期间这里我们使用 $T_{ij,A}$ 表示矩阵 T_{ij} 保存的非终结符集合中 A 的概率。算法3.4给出了概率 CYK 算法的基本流程。

以下通过一个实例来说明使用 CYK 算法求解 PCFG 的最佳树结构具体流程。给定如下符合乔姆斯基范式的文法 G 如下:

非终结符号集合：$N = \{S, P, V, NP, PP, VP\}$

终结符号集合：$\sum = \{$ eat, he, soup, spoon, with$\}$

规则集合：$R = \{$ (1) S -> NP VP 1.0; (2) VP -> VP PP 0.8; (3) VP -> V NP 0.2;

代码 3.4: 概率 CYK 句法分析算法

输入: 语法信息 G 和句子 $w_1 w_2 \cdots w_n$
输出: 句法树矩阵 T
// 初始化
for $i = 1$ **to** n **do**
  $T_{ii} = w_i$; // 主对角线上依次放入单词 w_i;
  foreach $A|A \to w_i$ **do**
    $T_{(i-1)i} = T_{(i-1)i} \cup (A, p), (A \to w_i, p \in G)$; // 依次放入单词 w_i 的所有词性及其概率;
  end
end
// 平行于主对角线逐层计算
for $d = 2$ **to** n **do**
  for $i = 0$ **to** $n - d$ **do**
    j = i+d;
    for $k = i + 1$ **to** $j - 1$ **do**
      if $(A \to BC, p) \in G$ and $T_{ik,B} > 0$ and $T_{kj,C} > 0$ **then**
        if $T_{ij,A} < p \times T_{ik,B} \times T_{kj,C}$ **then**
          $T_{ij} = T_{ij} \cup \{A, p \times T_{ik,B} \times T_{kj,C}, (k, B, C)\}$; // (k,B,C) 用于保存推导路径;
        end
      end
    end
  end
end
return T

(4) NP -> NP PP 0.2; (5) PP -> P NP 1.0 (6) NP -> He 0.3 ;
(7) NP -> soup 0.3; (8) NP -> spoon 0.2 ; (9) V -> eat 1.0;
(10) P -> with 1.0 }

图3.14和图3.15给出了概率 CYK 算法分析句子“He eat soup with spoon ”的过程。首先初始化矩阵主对角线上的单词以及单词所对应的词性和概率。之后进行 $T_{02}, T_{13}, T_{24}, T_{35}$ 元素的计算，通过规则集合中 VP->V NP 0.2 的推导规则，可以得到在 T_{13} 添加 VP 以及相应的路径信息，其概率为 $0.2 \times 1.0 \times 0.3 = 0.06$。根据 PP->P NP 1.0 的推导规则，在 T_{35} 添加 PP 以及相应的路径信息，其概率为 $1.0 \times 1.0 \times 0.2 = 0.2$。依次逐层进行分析，在对 $T_{1,6}$ 进行分析时，分别根据 VP->VP PP 0.8 和 VP->V NP 0.2 两个不同的推导规则得到 VP_1，其概率为 $0.8 \times 0.06 \times 0.2 = 0.0096$，$\text{VP}_2$ 的

概率为 $0.2 \times 1.0 \times 0.012 = 0.0024$。据此在 T_{05} 节点可以根据 S->NP VP 1.0，得到两个不同分析结果 S_1 的概率为 $1.0 \times 0.3 \times 0.0096 = 0.00288$ 和 S_2 的概率为 $1.0 \times 0.3 \times 0.0024 = 0.00072$。根据不同所得到的不同分析树的概率大小，选择最终结果。

通过上述例子我们可以看到，根据构建好的 PCFG 文法，利用概率 CYK 分析等最佳树结构求解算法，可以在一定程度上对句法分析中的歧义进行很好的处理，根据概率选择可能性较大的树结构。

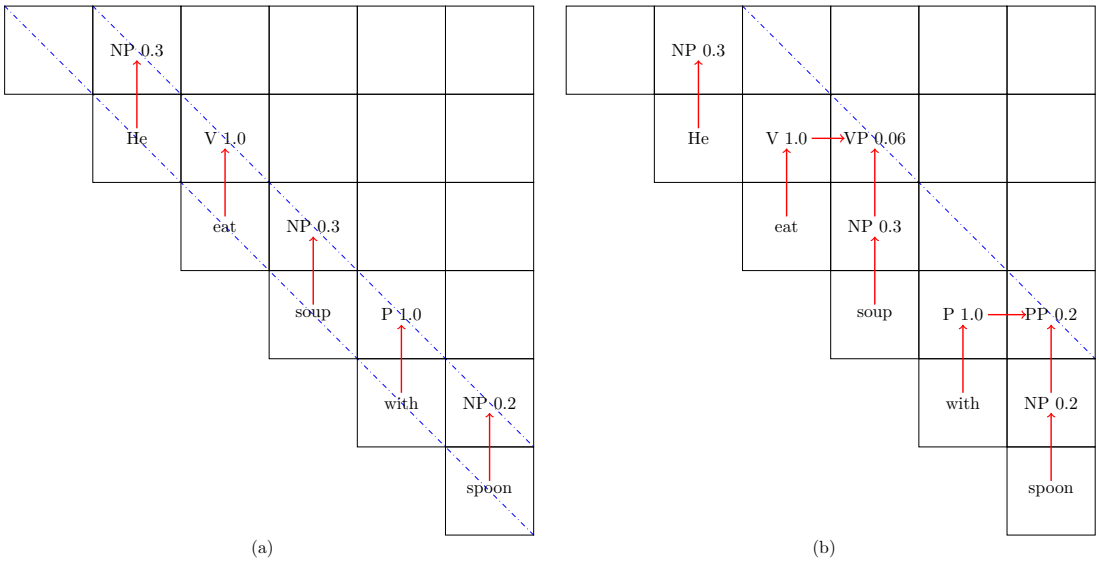


图 3.14　使用 CYK 算法求解 PCFG 的最佳树结构分析实例

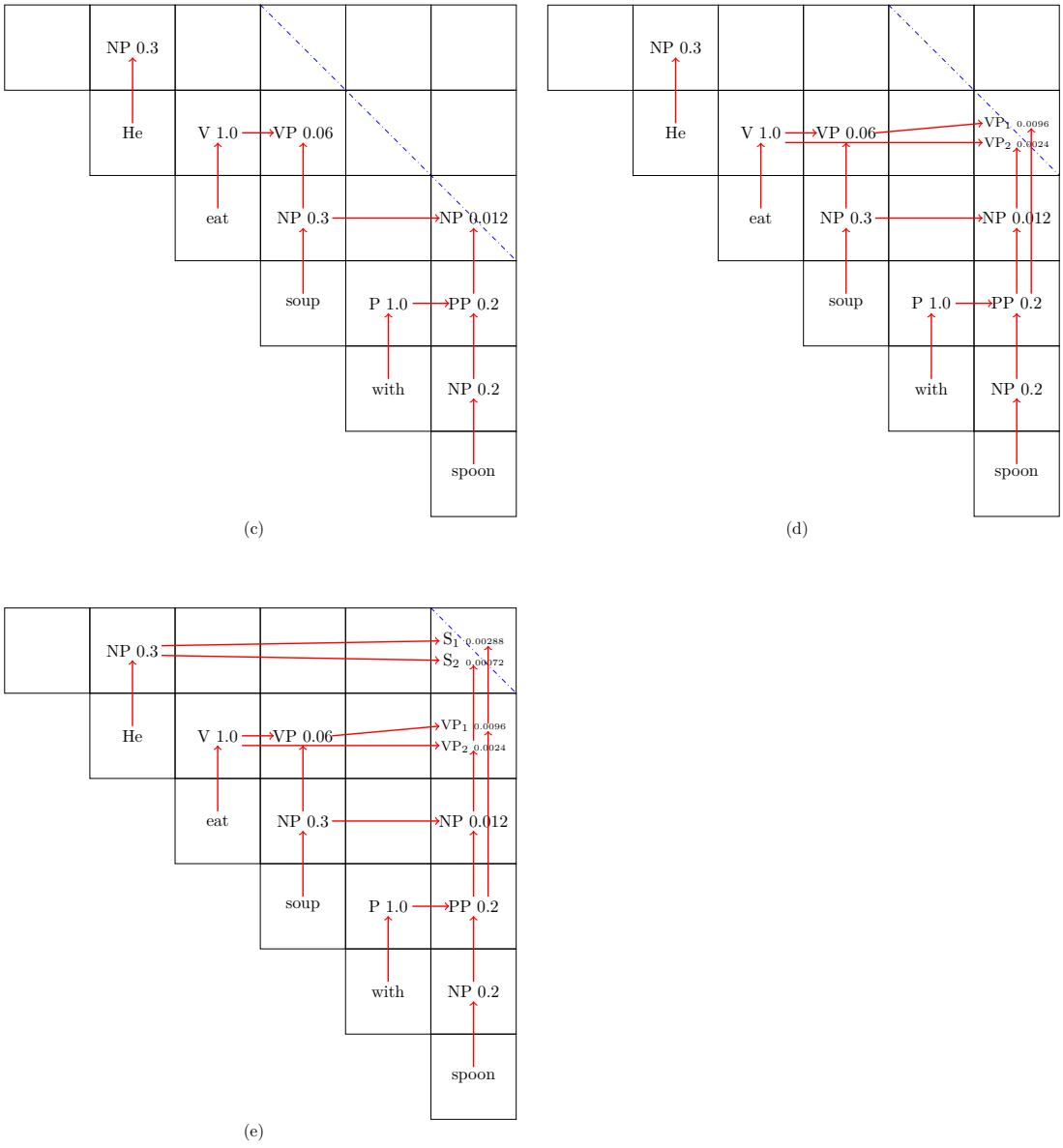


图 3.15　使用 CYK 算法求解 PCFG 的最佳树结构分析实例 (续)

4. PCFG 的模型参数学习

模型参数学习是指给定 CFG 文法 G 下，如何学习 G 中规则的概率参数构建 PCFG 文法。有两种常见的方法用于学习语法规则概率：1）有大规模句法树标注数据（树库）；2）有大规模的没有标注句法树的句子。

在有大规模标注句法树标注的情况下，可以基于最大似然估计，统计非终结符的出现次数进行概率参数估计：

$$P(A \to \alpha) = \frac{Count(A \to \alpha)}{\sum_{\lambda} Count(A \to \lambda)} = \frac{Count(A \to \alpha)}{Count(A)} \tag{3.11}$$

$Count(A \to \alpha)$ 是指规则 $A \to \alpha$ 在整个树库中出现的次数，$Count(A)$ 是指在树库中非终结符 A 出现的次数。

在仅有大规模无标记句子的情况下，也可以通过期望最大化算法（Expectation M，EM）估计规则的概率参数。基本思想是首先对给定的 CFG 文法 G 中的每个规则，在满足归一化条件的情况下随机赋值概率值，构造文法 G_0。在此基础上利用 G_0 对所有句子进行分析，计算出每条规则使用次数的期望值。之后利用期望次数根据最大似然估计原理，得到文法 G 的概率参数更新，记为 G_1。循环执行该过程直到 G 的概率参数收敛于最大似然估计值。利用当前的文法 G_i 估算每条规则出现的期望值是期望步（Expectation Step，E-步骤），重新估算概率得到 G_{i+1} 的步骤是最大化步（Maximization Step，M-步骤）。具体的计算公式可以参考[12, 13]。

3.2.3 成分句法分析评价方法

成分句法分析算法的性能评价，目前通常采用的是方法是 PARSEVAL 方法[14]。PARSEVAL 方法主要包含以下三个指标：

(1) 标记精度（Labeled Precision，LP）：句法器分析结果中正确的短语结构所占比例，即分析结果与标准句法树中短语匹配的个数占分析器所有输出结果中短语个数的比例，具体计算公式如下：

$$\text{LP} = \frac{\text{分析结果中正确的短语个数}}{\text{分析结果中短语总数}} \times 100\% \tag{3.12}$$

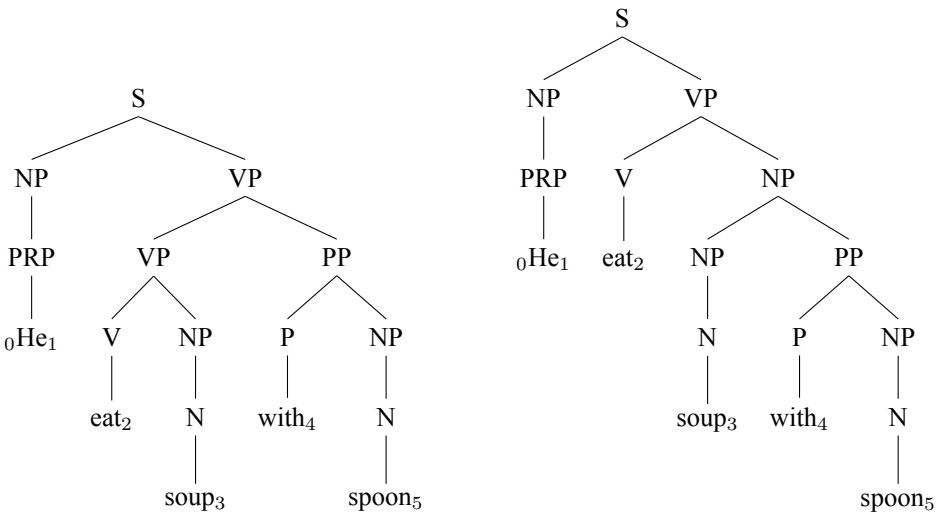
(2) 标记召回率（Labeled Recall，LR）：分析结果中正确的短语结构占标准句法树中短语总数的比例，具体计算公式：

$$\text{LR} = \frac{\text{分析结果中正确的短语个数}}{\text{标准结果中短语总数}} \times 100\% \tag{3.13}$$

(3) 交叉括号数（Crossing brackets，CBs）：一颗成分句法树中所包含的与标准句法树中边界相交叉的短语个数。

此外，通常还会根据标记精度（LP）和标记召回率（LR），利用与标准 F1 值计算公式一致的方法得到标记 F1 值，综合 LP 和 LR 得分。

以句子“He eat soup with spoon”的正确答案和某成分句法分析器结果为例，如图3.16所示，对上述评测指标进行解释。



(a) 正确结果　　(b) 算法分析结果

图 3.16　成分句法分析结果评测样例

为了方便评测，分析树中除词性之外的非终结符节点通常会增加起始位置和结束位置的方式进行表示：XP(起始位置，结束位置)。其中 XP 表示短语名称，例如：动词短语 VP，名词短语 NP 等。根据这种表示形式，图3.16所示的两棵树的结果表示为：

(a) 正确结果：S(0,5)，NP(0,1)，VP(1,5)，VP(1,3)，NP(2,3)，PP(3,5)，NP(4,5)

(b) 算法分析结果：S(0,5)，NP(0,1)，VP(1,5)，NP(2,5)，NP(2,3)，PP(3,5)，NP(4,5)

根据公式3.12、公式和公式可以计算得到该句法分析算法的如下评估指标：

$$\mathrm{LP} = \frac{6}{7} \times 100\% = 85.7\%$$

$$\mathrm{LR} = \frac{6}{7} \times 100\% = 85.7\%$$

算法分析结果中的 NP(2,5) 与正确结果的 PP(3,5) 和 VP(1,3) 都发生了边界交叉，因此交叉括号 CBs 值为 2。

3.3 依存句法分析

依存句法分析（Dependency Parsing）是依据依存语法理论，分析输入句子得到其依存句法结构

树。依存句法理论的基本假设是句法结构由单词和单词之间的依存关系组成。依存关系具有方向性，从中心语成分指向依存成分。依存关系根据中心成分和依存成分之间的关系又可以被划定义为不同的依存关系类型。依存句法结构使用依存图（Dependency Graph）进行表示。如图3.17所示。

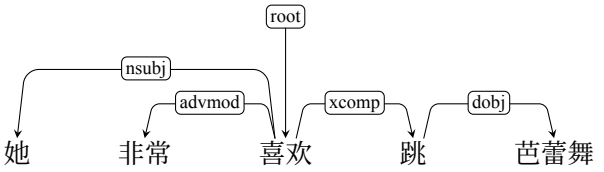


图 3.17　句子“她非常喜欢跳芭蕾舞”的依存句法树

在本节中，使用 $S = \{w_0, w_1, \cdots w_n\}$ 表示输入的句子，其中 $w_0 = root$ 表示虚拟根节点，$w_1 \ldots w_n$ 为输入句子中的 n 个单词。$R = \{r_0, r_1, \ldots, r_m\}$ 表示依存关系类型集合，$r \in R$ 表示在句子中单词之间的依存关系，也叫做边标签（arc label）。例如在图3.17中，“喜欢”和“她”之间的依存关系类型 $r = nsubj$。依存图 $G = (V, A)$ 是一个有标记的有向图，顶点 V 和边 A 针对句子 S 和依存关系集合符合以下要求：

(1) $V \subseteq \{w_0, w_1, \cdots w_n\}$

(2) $R \subseteq V \times R \times V$

(3) 如果 $(w_i, r, w_j) \in A$，那么 $(w_i, r, w_j) \notin A, \forall r' \neq r$

边 (w_i, r, w_j) 表示以 w_i 为头 w_j 为尾边标记为 r 的边，其表示以 w_i 为中心词，w_j 为修饰词，类型为 r 的依存关系。依存图 G 表示了一组句子 W 中的单词之间的有标记的依存关系。依存图中的节点集合 V 通常包含句子中的所有单词，针对句子 S 通常用 $V_S = \{w_0, w_1, \cdots w_n\}$ 表示。

根据上述定义图3.17可以表示为：

(1) $G = (V, A)$

(2) $V = V_W =$ {ROOT, 她, 非常, 喜欢, 跳, 芭蕾舞 }

(3) $A =$ {(ROOT, root, 喜欢), (喜欢, nsubj, 她), (喜欢, advmod, 非常), (喜欢, xcomp, 跳), (跳, dobj, 芭蕾舞) }

如果依存图 $G = (V, A)$ 对于输入句子 S 和关系集合 R，是一个从 w_0 出发的有向树并且包含句子中的所有单词，那么这依存图 G 就成为*形式良好的依存图*（*well-formed dependency graph*），也称为**依存树（Dependency Tree）**。对于输入句子 W 和关系集合 R，所有形式良好的依存图集合记为 $\mathcal{G}_S$。依存句法分析就是对输入句子 W 根据一定的原则从 $\mathcal{G}_S$ 中选择得分最高的依存句法树。

3.3.1 基于图的依存句法分析

基于图的依存句法分析核心是构造评分函数对句子 S 所有依存句法树 $G = (V, A) \in \mathcal{G}_W$ 进行评分。这个评分代表了一个句法树做为句子分析正确结果的可能性。不同的基于图的分析方法采

用不同的假设来计算该得分。基于图的依存句法分析算法通常将对依存句法树的 $G=(V,A)$ 的评分转化为对其树上的边的评分：

$$\text{score}(G_S)=\sum_{(w_i,r,w_j)\in A}\lambda_{(w_i,r,w_j)} \tag{3.14}$$

$\boldsymbol{\lambda}$ 表示所有边的评分，$\lambda_{(w_i,r,w_j)}$ 表示边 (w_i,r,w_j) 的得分，其具体计算方法将在本节后续部分进行详细介绍。基于图得分计算，可以将基于图的依存句法分析形式化表示为：

$$\hat{G}=\underset{G=(V,A)\in\mathcal{G}_S}{\arg\max}\sum_{(w_i,r,w_j)\in A}\lambda_{(w_i,r,w_j)} \tag{3.15}$$

可以证明在对依存句法树不考虑投射性（Projectivity）的情况下，对于输入句子 S 的依存句法分析问题等价于基于边评分 $\lambda_{(w_i,r,w_j)}$ 的图 G_S 的最大生成树（maximum spanning tree）寻找问题[15]。此外，可以定义 $\lambda_{(w_i,w_j)}=\max\limits_r \lambda_{(w_i,r,w_j)}$，并利用 $\lambda_{(w_i,w_j)}$ 作为边的权重构建图 G'。通过图 G' 得到的最大生成树 $\hat{G}'$，可以证明与通过 $\lambda_{(w_i,r,w_j)}$ 构建得到的最大生成树 $\hat{G}$ 相同[15]。由此，可以将包含依存关系类别的分析转换为无依存关系类别的分析，进一步降低分析算法的复杂性。

需要注意的是，利用最大生成树算法得到的依存句法树不具备投射性。针对具有投射性要求的依存句法树，可以利用其与上下文无关语法之间的强相关性，利用基于 CYK 算法等上下文无关语法分析算法进行依存句法树分析。在本节中我们将针对上述两种情况分别进行介绍。

1. 非投射性依存句法分析方法

由上节的介绍我们可以知道，非投射性的依存句法分析等价于图的最大生成树寻找问题。朱-刘/埃德蒙兹算法（Chu-Liu/Edmonds）方法[16, 17] 是一种常见的带权有向图最小/大生成树寻找算法，因此也常被应用于非投射性依存句法分析。朱-刘/埃德蒙兹算法的核心是采用贪心的思想对边进行选择，利用递归方法来实现整个过程。

朱-刘/埃德蒙兹算法用于非投射性依存句法分析的输入是待分析句子 $S=\{w_0,w_1,\cdots,w_n\}$ 和边之间权重 $\lambda_{(w_i,w_j)}\in\boldsymbol{\lambda}$。根据定义，$w_0$ 是句子的虚拟根节点，依存句法树中不存在指向 w_0 的边，因此设置所有 $\lambda_{(w_i,w_0)}=-\infty$。分析算法首先根据句子中的单词和权重组成有向图 $G_S=(V_S,A_S)$，$V_S=\{w_0,w_1,\ldots,w_n\}$，$A_S=\{(w_i,w_j)|\forall w_i,w_j\in V_S\}$。朱-刘/埃德蒙兹算法的第一步是针对图 G 中每个顶点选择入边权重最大的边构建子图 $G'=(V_S,A')$。如果该子图中没有环，那么该子图就是图 G 的最大生成树。否则，说明图 G' 中至少包含一个环。那么选择其中任意一个环 C，其边集合为 A_C，将环 C 用一个节点 w_c 来代表，图 G_C 中包含所有在图 G 中但是不在环 C 中的节点，以及 w_C，G_C 的边通过以下规则构建：

(1) 对于所有不在环 C 的顶点 w_j，如果存在一个边 (w_i,w_j)，w_i 在环 C 中，那么就添加边 (w_c,w_j) 到 G_C 中，定义 $ep(w_c,w_j)=\underset{w_i\in C}{\arg\max}\,\lambda_{(w_i,w_j)}$，用于记录环 C 中相对应的顶点，相应的修改

(w_c, w_j) 的边权重 $\lambda_{(w_c,w_j)} = \lambda_{(ep(w_c,w_j),w_j)}$；

(2) 对于所有不在环 C 的顶点 w_i，如果存在一个边 (w_i, w_j)，w_j 在环 C 中，那么添加边 (w_i, w_c) 到 G_C 中，定义 $ep(w_i, w_c) = \arg\max_{w_j \in C} \left[\lambda_{(w_i,w_j)} - \lambda_{(a(w_j),w_j)}\right]$，相应的修改 (w_i, w_c) 的边权重 $\lambda_{(w_i,w_c)} = \lambda_{(w_i,ep(w_i,w_c))} - \lambda_{a(ep(w_i,w_c)),(ep(w_i,w_c))} + \sum_{w \in C} \lambda_{(a(w),w)}$；

(3) 对于所有边 (w_i, w_j)，其顶点 w_i 和 w_j 都不在环 C 中，直接添加该边到图 G' 中，保持原有权重不变。

将图 G_C 作为输入递归调用上述算法得到其最大生成树 $G = (V, A)$。之后根据所返回的最大生成树信息对原始图信息进行修正，移除环 C，并且将 w_c 所指向的节点的实际边还原，返回输入图所对应的最大生成树。朱-刘/埃德蒙兹算法的伪代码如算法3.5所示。

2. 投射性依存句法分析方法

设置虚拟根节点在句子首位的情况下，投射性依存句法分析树等价于嵌套依存句法分析树（Nested Dependency Trees）。因此投射性依存句法分析与上下文无关语法具有非常强的关系，很多用于上下文无关语法分析算法也可以应用于投射性依存句法分析。在 4.2 节中介绍了 CYK 算法用于 CFG 和 PCFG 的句法分析，本节中将介绍基于 CYK 算法改进的依存句法分析算法。

首先定义动态规划表 C[s][t][i] ($s \leqslant i \leqslant t$)，表示投射性句法树以单词 w_i 为根节点覆盖从单词 w_s 到单词 w_t 的句子片段的最高得分。由此可以得到 C[0][n][0] 对于输入句子 $S = w_0, w_1, \ldots, w_n$ 的依存句法树的最高得分。任何以 w_i 为根节点覆盖 w_s 到 w_t 的投射性句法树都是由更小的子树构成。同时更大的子树是通过相邻的子树由内向外添加依存关系得到。因此对于 C[s][t][i] 可以构造如下递推公式：

$$C[s][t][i] = \max_{s \leqslant k \leqslant t, s \leqslant j \leqslant t} \begin{cases} C[s][k][i] + C[k+1][t][j] + \lambda(w_i, w_j), & \text{如果} j > i \\ C[s][k][j] + C[k+1][t][i] + \lambda(w_i, w_j), & \text{如果} j < i \end{cases} \tag{3.16}$$

图3.18给出了合并过程示意图。

由于 C[s][t][i] 中仅保存了子树的最高得分，但是没有保存子树的结构。因此在实际构建过程中，还需要增加树结构矩阵 A[s][t][i]，用于保存依存句法树结构。通过公式3.16计算得到最优的 k 和 j 之后，A[s][t][i] 按照如下公式记录树结构：

$$A[s][t][i] = \begin{cases} A[s][k][i] \cup A[k+1][t][j] \cup (w_i, w_j), & \text{如果} j > i \\ A[s][k][j] \cup A[k+1][t][i], & \text{如果} j < i \end{cases} \tag{3.17}$$

针对句子 S 得到的依存句法树 $G = (V, A[0][n][0])$ 组成。

代码 3.5: 朱-刘/埃德蒙兹算法求解依存句法分析树

输入: 图 $G=(V,A)$，$\lambda_{(w_i,w_j)} \in \lambda$

输出: 最大生成树图 G

Function Chu-Liu-Edmonds(G,λ) **:**

$A' = \{(w_i, w_j) | w_j \in V, w_i = \arg\max_{w_i} \lambda_{(w_i,w_j)}\}$;

$G' = (V, A')$;

if G' 中没有环 **then**

  return G';

end

A'_C= 图 G' 中任意一个环 C 的边集合;

V'_C= 环 C 顶点集合;

$V_C = V \cup \{w_c\} - V'_C$;

foreach $w_j \in V - V'_C : \exists_{w_i \in V'_C}(w_i, w_j) \in A$ **do**

  $A_C = A_C \cup \{(w_c, w_j)\}$;

  $ep(w_c, w_j) = \arg\max_{w_i \in C} \lambda_{(w_i,w_j)}$;

  $\lambda_{(w_c,w_j)} = \lambda_{(ep(w_c,w_j),w_j)}$;

end

foreach $w_i \in V - V'_C : \exists_{w_j \in V'_C}(w_i, w_j) \in A$ **do**

  $A_C = A_C \cup \{(w_i, w_c)\}$;

  $ep(w_i, w_c) = \arg\max_{w_j \in C} \left[\lambda_{(w_i,w_j)} - \lambda_{(a(w_j),w_j)}\right]$;

  $\lambda_{(w_i,w_c)} = \lambda_{(w_i,ep(w_i,w_c))} - \lambda_{a(ep(w_i,w_c)),(ep(w_i,w_c))} + \sum_{w \in C} \lambda_{(a(w),w)}$;

end

foreach $w_i \in V - V'_C : \exists_{w_j \in V - V'_C}(w_i, w_j) \in A$ **do**

  $A_C = A_C \cup \{(w_c, w_j)\}$;

end

G_C=(V_C, A_C);

$G = (A, V)$ = Chu-Liu-Edmonds(G_C,λ);

寻找 A 中指向 w_c 的边 (w_i, w_c)，w_j=$ep(w_i, w_c)$;

寻找环 C 中指向 w_j 的边 (w_k, w_j);

寻找 A 中所有以 w_c 为头节点的边 (w_c, w_l);

A=$A \cup (ep(w_c, w_l), wl)_{forall(w_c,w_l) \in A} \cup A_c \cup (w_i, w_j) - (w_k, w_j)$;

return G;

return

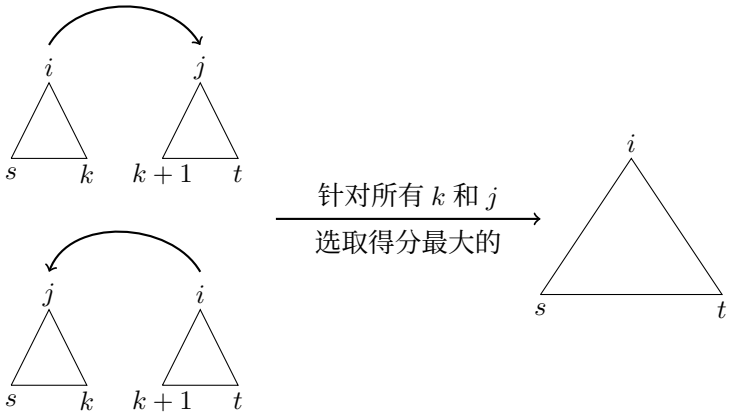


图 3.18 面向依存句法分析的 CYK 算法递推样例

3. 边评分模型学习方法

边评分模型可以使用基于高维特征向量的线性函数进行建模。使用 $f(w_i, r, w_j) \in \mathbb{R}^m$ 表示特征函数，其输出是高维特征向量，$\boldsymbol{w}$ 是 $f(\cdot)$ 的对应的权重向量。边评分参数 $\lambda_{(w_i,r,w_j)}$ 可以表示为：

$$\lambda_{(w_i,r,w_j)} = \boldsymbol{w} \cdot f(w_i, r, w_j) \tag{3.18}$$

$f(\cdot)$ 包含了边和输入句子 S 中各类型相关特征，例如：

- w_i= 喜欢
- w_j= 跳
- w_i 的词性 =V
- w_j 的词性 =V
- r 的依存关系类型 =xcomp
- w_{i-1} 的词性 =ADV
- w_{j+1} 的词性 =N
- w_i 和 w_j 之间距离 =1

这些特征还可以组合为更复杂的类型例如：

- w_i 的词性 =V & w_j 的词性 =V & w_i= 喜欢
- w_i= 喜欢 & w_j= 跳 & w_i 和 w_j 之间距离 =1

类别特征通常也会通过二值化操作转换为 0/1 特征，具有 m 个类别的特征通常会转化为 m 个 0/1 特征，表示某个类别出现或不出现。从上述特征的构成我们可以看到，特征向量维度通常会非常高，但是对于一个边来说特征向量非常稀疏，仅有非常小部分的特征不是 0。因此可以利用稀疏性进行表示和计算。

基于特征向量的线性函数建模的假设下，对于输入句子的依存句法分析转换为了如下问题：

$$\hat{G} = \underset{G=(V,A)\in\mathcal{G}_S}{\arg\max} \sum_{(w_i,r,w_j)\in A} \lambda_{(w_i,r,w_j)} = \underset{G=(V,A)\in\mathcal{G}_S}{\arg\max} \sum_{(w_i,r,w_j)\in A} \boldsymbol{w}\cdot f(w_i,r,w_j) \tag{3.19}$$

训练语料用 $D=\{(S_d,G_d)\}_{d=1}^{|D|}$ 表示，其中 S_d 表示输入句子，所对应的正确的依存句法树用 G_d 表示。针对输入句子可以利用特征函数 $\mathbf{f}(\cdot)$ 构造句子中所有单词对和根据不同依存关系类型输出特征向量。边评分模型的学习就转换为了权重向量 $\mathbf{w}$ 的学习问题。该问题可以采用感知器算法（Perceptron Algorithm）完成。感知器算法是一种典型的基于推理（Inference-based Learning）的在线学习方法，也称为错误驱动（Error-driven）的学习算法，伪代码如算法3.6所示。

代码 3.6: 感知器算法学习参数向量 $\mathbf{w}$

输入: 训练语料 $D=\{(S_d,G_d)\}_{d=1}^{|D|}$
输出: 权重向量 $\mathbf{w}$
Function `Perceptron(`D`)` **:**
  $\boldsymbol{w}$=0;
  for $n:1\ldots N$ **do**
    for $d:1\ldots|D|$ **do**
      $G'=\underset{G=(V,A)\in\mathcal{G}_{S_d}}{\arg\max} \sum_{(w_i,r,w_j)\in A} \boldsymbol{w}\cdot f(w_i,r,w_j)$;
      if $G\neq G'$ **then**
        $\boldsymbol{w}$=$\boldsymbol{w}+\sum_{(w_i,r,w_j)\in A_d} f(w_i,r,w_j)-\sum_{(w_i,r,w_j)\in A'} f(w_i,r,w_j)$;
      end
    end
  end
  return $\boldsymbol{w}$
return

感知器算法每次仅考虑一个训练样本，利用当前的权重向量 $\mathbf{w}$ 和依存树构建算法针对当前样本 S_d 构建依存句法树 G'。如果 G' 与标准标注 G_d 不同，则通过增加正确依存树所对应的边的特征向量并减去不正确的句法树分析结果的边更新当前的权重向量 $\mathbf{w}$。最大生成树依存句法分析器（Maximum Spanning Tree Dependency Parser, MSTParser）[18] 采用了上述方法，并在上述基础上增加了最大间隔（Max Margin）目标函数并引入了边缘注入松弛算法（Margin-infused relaxed algorithm, MIRA），进一步提升了所学习得到的权重向量的泛化能力。

3.3.2 基于神经网络的图依存句法分析

基于图的依存句法分析主要包含边评分模型和句法树生成算法两个部分组成。其中边评分模型对于分析效果具有决定性的影响。传统的分析方法依赖人工设计的数百万甚至数千万特征，模型的泛化能力不高，很容易出现过拟合问题。此外，由于特征函数中使用大量的单词关联信息，也很容易使得特征空间巨大。神经网络方法可以在一定程度上缓解上述问题，同时也不需要人工设计特征函数。在本节中，将介绍三种基于神经网络图依存句法分析算法。

1. 基于前馈神经网络的方法

Pei 等人[19] 提出的依存句法分析器采用最大生成树方法构造依存句法树。但是在边评分模型方面，他们提出了仅利用最基本信息作为输入的神经网络方法。公式3.14给出了基于图的依存句法分析的树得分计算方法，将树的得分转换为了树上边的得分。传统方法基于公式3.18将边得分转换为特征设计和对应权重学习问题。Pei 等人提出的算法的输入仅为包括单个单词、单个单词词性、单词之间距离等在内的基本信息，来计算边 (h, m) 的得分 $\lambda_{(h,m)}$，h 表示依存关系的中心词（head），m 表示该依存关系中的修饰词（modifier）。所采用的神经网络结构如图3.19所示。

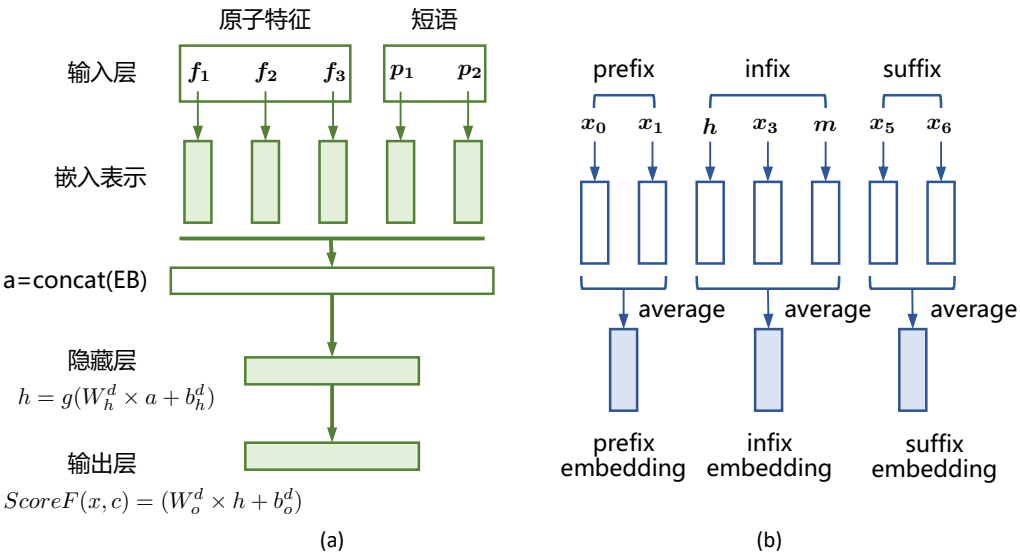


图 3.19 基于神经网络的边评分模型网络结构图[19]

模型输入包含两大部分：原子特征（Atomic Features）和短语结构（Phrases）。原子特征部分使用单个词和单个词性。通过查找嵌入矩阵 $\boldsymbol{M} = \mathbb{R}^{d \times |\mathcal{D}|}$ 转换为相应的嵌入（Embedding）表示。$|\mathcal{D}|$ 是特征字典大小，d 是特征嵌入表示的维度。为了更好的建模依存关系的上下文信息，根

据所要预测中心词和修饰词在句子中的位置，将句子切分为前缀（prefix）、中缀（infix）和后缀（suffix）。如图3.19的右边部分所示。采用平均池化（Average Pooling）的方法构建前缀嵌入表示（prefix embedding）、中缀嵌入表示（infix embedding）和后缀嵌入表示（suffix embedding）等成短语嵌入（Phrase Embedding），并与原子特征一起作为模型输入。

模型第二层将原子特征的嵌入表示和短语嵌入合并到单个向量 $\boldsymbol{a}$。第三层隐藏层利用激活函数 tanh-cube 学习线性变换后的向量 $\boldsymbol{a}$ 多个特征的综合。

$$\boldsymbol{h} = g(\boldsymbol{W}_h^d \boldsymbol{a} + \boldsymbol{b}_h^d) \tag{3.20}$$

$$g(l) = \tanh(l^3 + l) \tag{3.21}$$

模型最后一层输出层利用线性变换得到边评分函数：

$$\text{Score}F(x, c) = \boldsymbol{W}_o^d \boldsymbol{h} + \boldsymbol{b}_o^d \tag{3.22}$$

通过最大间隔标准进行模型参数 $\{\boldsymbol{W}_h^d, \boldsymbol{b}_h^d, \boldsymbol{W}_o^d, \boldsymbol{b}_o^d, \boldsymbol{M}\}$ 进行训练。

2. 基于双仿射变换的方法

Dozat 和 Manning 提出的 Deep Biaffine Parser[20]，也是采用最大生成树方法构造依存句法树，在 Pei 等人[19] 工作的基础上引入了双向长短时记忆网络（BiLSTM），更好的建模句子上下文信息。同时也考虑到直接使用 BiLSTM 的每个时刻隐藏状态序列作为单词表示可能带来的信息过多，容易造成过拟合并需要更多数量的训练语料的问题，提出了双仿射变换的方法。Deep Biaffine Parser 的边评分模型神经网络结构如图3.20所示。

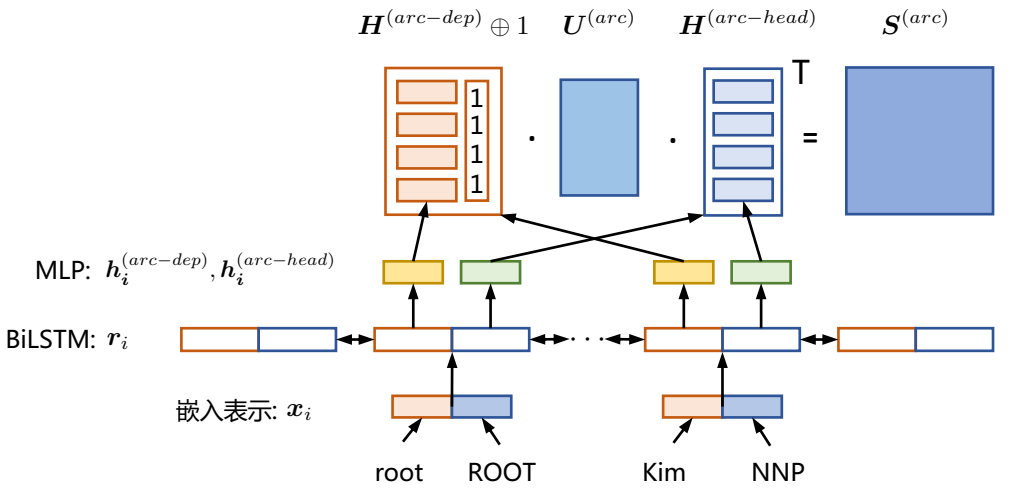


图 3.20 Deep Biaffine Parser 边评分模型神经网络结构图[20]

句子中所有单词的词嵌入和对应的词性嵌入合并作为双向长短时记忆网络的输入，记为 $\boldsymbol{x}_i = \boldsymbol{v}_i^{word} \oplus \boldsymbol{v}_i^{POS}$。所有单词经过 BiLSTM 层计算后得到每个时刻的输出记为 $\boldsymbol{r}_i$。接下来，使用多层感知器（MLP）对 $\boldsymbol{r}_i$ 用于中心词和修饰词的不同，分别进行两种不同的降维 $\boldsymbol{h}_i^{(arc-head)}$ 和 $\boldsymbol{h}_i^{(arc-dep)}$：

$$\boldsymbol{h}_i^{(arc-head)} = \mathbf{MAP}^{(arc-head)}(\boldsymbol{r}_i) \tag{3.23}$$

$$\boldsymbol{h}_i^{(arc-dep)} = \mathbf{MAP}^{(arc-dep)}(\boldsymbol{r}_i) \tag{3.24}$$

所有时刻的 $\boldsymbol{h}_i^{(arc-head)}$ 和 $\boldsymbol{h}_i^{(arc-dep)}$ 分别合并组成矩阵 $\boldsymbol{H}^{(arc-head)}$ 和 $\boldsymbol{H}^{(arc-dep)}$。之后利用不定类别双仿射分类器（Variable-class Biaffine Classifier）对 $\boldsymbol{H}^{(arc-dep)}$ 额外拼接了一个单位向量，利用矩阵 $\boldsymbol{U}^{(arc)}$ 进行仿射变换，得到边评分矩阵：

$$\boldsymbol{S}^{(arc)} = \boldsymbol{H}^{(arc-dep)} \oplus \mathbf{1} \cdot \boldsymbol{U}^{(arc)} \cdot \boldsymbol{H}^{(arc-head)\top} \tag{3.25}$$

由于依存关系类别数目是确定的，针对类别的分类问题，可以采用下述公式计算单词 w_i 作为修饰词 w_{y_i} 做为中心词的依存关系类别得分：

$$\boldsymbol{s}_i^{(label)} = \boldsymbol{r}_{y_i}^{\top} \boldsymbol{U}^{(1)} \boldsymbol{r}_i + (\boldsymbol{r}_{y_i} \oplus \boldsymbol{r}_i)^{\top} \boldsymbol{U}^{(2)} + \boldsymbol{b} \tag{3.26}$$

3. 基于图神经网络的方法

Ji 等人同样采用了在对边评分的基础上利用贪心或者最大生成树等算法构建依存句法生成树的框架，但是试图将更多的结构化信息引入到结点表示，构造了基于图神经网络的依存句法分析

算法（Graph-based Dependency Parsing with Graph Neural Networks，GNNDP）[21]，算法的网络架构如图3.21所示。GNNDP 从表层的单词序列和深层的树结构两个角度进行结点编码。

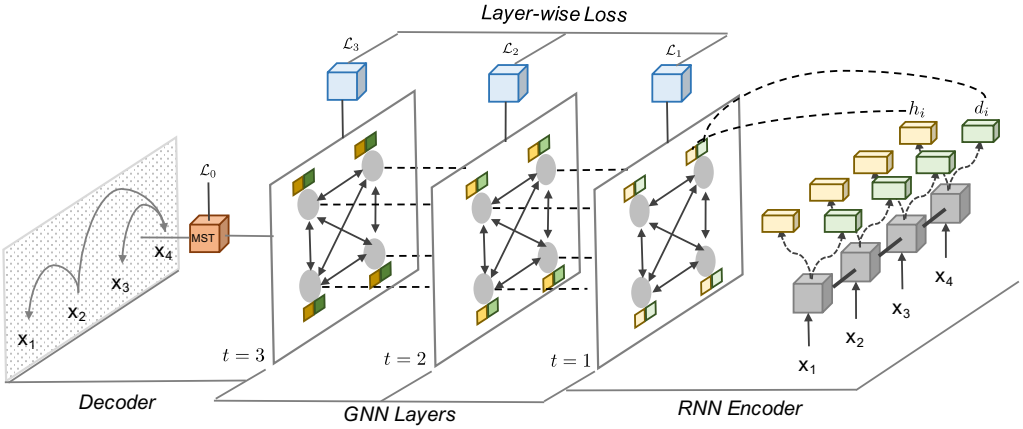


图 3.21　基于图神经网络的依存句法分析网络结构图[21]

单词序列表示使用双向长短时记忆网络（BiLSTM）编码单词序列。在每个句子位置 i，一个前向 LSTM 链（参数为 $\boldsymbol{\theta}^f$）通过收集从句子开头到当前位置 i 的信息计算出隐向量 $\boldsymbol{c}_i^f$；同样，一个后向 LSTM 链（$\boldsymbol{\theta}^b$）收集从句子结尾到位置 i 的信息 $\boldsymbol{c}_i^b$:

$$\boldsymbol{c}_i^f = \text{LSTM}\left(\boldsymbol{x}_i, \boldsymbol{c}_{i-1}^f; \boldsymbol{\theta}^f\right) \tag{3.27}$$

$$\boldsymbol{c}_i^b = \text{LSTM}\left(\boldsymbol{x}_i, \boldsymbol{c}_{i+1}^b; \boldsymbol{\theta}^b\right) \tag{3.28}$$

$$\boldsymbol{c}_i = \boldsymbol{c}_i^f \oplus \boldsymbol{c}_i^b \tag{3.29}$$

其中 $\boldsymbol{x}_i$ 是 LSTM 单元的输入，由三部分拼接而成：随机初始化的单词嵌入 $\boldsymbol{e}_{w_i}$，使用 Glove 的预训练单词嵌入 $\boldsymbol{e}'_{w_i}$ 以及随机初始化的词性标签嵌入 $\boldsymbol{e}_{pos_i}$。归一化双线性函数基于结点表征来定义得分函数 σ。由于依存边有方向性，因此采用两个多层感知器来生成不同的向量从而区分这两种角色[20]:

$$\boldsymbol{h}_i = \text{MLP}_h\left(\boldsymbol{c}_i\right) \tag{3.30}$$

$$\boldsymbol{d}_i = \text{MLP}_d\left(\boldsymbol{c}_i\right) \tag{3.31}$$

$$\begin{aligned}\sigma(i,j) &= \text{Softmax}_i\left(\boldsymbol{h}_i^\top \boldsymbol{A}\boldsymbol{d}_j + \boldsymbol{b}_1^\top \boldsymbol{h}_i + \boldsymbol{b}_2^\top \boldsymbol{d}_j\right) \\ &\triangleq P(i \mid j)\end{aligned} \tag{3.32}$$

其中 $\boldsymbol{A}, \boldsymbol{b}_1, \boldsymbol{b}_2$ 是可训练模型参数，输出的得分实际上也是依存边（w_i 支配 w_j）的概率。

树结构表示采用基于图神经网络（Graph Neural Networks, GNNs）进行建模。首先，我们以图注意力网络（Graph Attention Networks, GATs）为例图神经网络介绍的一般框架[22]。给定有向图 G，图神经网络是一个多层网络。它在每一层通过聚合其邻居的信息来维护一组结点的表示。对于结点 i，用 $\mathcal{N}(i)$ 表示 G 中结点 i 的邻居，用 $\boldsymbol{v}_i^t$ 表示结点 i 在图神经网络第 t 层的隐向量。$\boldsymbol{v}_i^t$ 的计算如下：

$$\boldsymbol{v}_i^t = g\left(W \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^t \boldsymbol{v}_j^{t-1} + B\boldsymbol{v}_i^{t-1}\right). \tag{3.33}$$

其中 g 是一个非线性激活函数 LeakyReLU，$\boldsymbol{W}$ 和 $\boldsymbol{B}$ 是参数矩阵。边权 α_{ij}^t 表示邻居结点 j 在构建 $\boldsymbol{v}_i^t$ 时的贡献值。可以看到，图神经网络自然地捕捉到了多跳（即高阶）关系。以前两层为例，对于第二层的每个结点 i，$\boldsymbol{v}_i^2$ 包含其 1 跳邻居 $\boldsymbol{v}_j^1$ 的信息。由于 $\boldsymbol{v}_j^1$ 已经在第一层编码了它自己的 1 跳邻居，$\boldsymbol{v}_i^2$ 实际上编码了它的 2 跳邻居的信息。因此，图神经网络可以有助于编码高阶结构特征。

GNNDP 将图神经网络应用到依存句法分析任务上。如图 3.21所示，解析任务需要在图 G 上同时处理支配词表征 $\boldsymbol{h}_i$ 和从属词表征 $\boldsymbol{d}_i$，而不是为每个结点编码一个向量。此外，为了近似精确的高阶解析，解析器需要每个 GNNs 网络层有关于句子的具体含义。因此，GNNDP 采用完全图（即所有结点都是连接的），并用依存边条件概率（公式 3.32）设置边权。

$$\alpha_{ij}^t = \sigma^t(i,j) = P^t(i \mid j) \tag{3.34}$$

GNNDP 关注三类高阶信息，即祖父母、孙子和兄弟关系，如图 3.22所示，其中灰色的阴影表示哪些结点表征已经存在于一阶特征中。橙色阴影表示在高阶特征中应该包括哪些结点表征。需要调整一般的 GNNs 更新公式，将它们适当地编码到结点表示中。首先，为了纳入祖父母的信息(图 3.22(a))，使得 $\sigma^t(j,i)$ 不仅考虑一跳父子关系 (j,i)，还考虑两跳的 i 的祖先结点（用 k 表示）。同样，为了在 $\sigma^t(j,i)$ 中对两跳的 j 的孙子结点进行编码（也用 k 表示），需要聚合邻居结点的从属词表示。

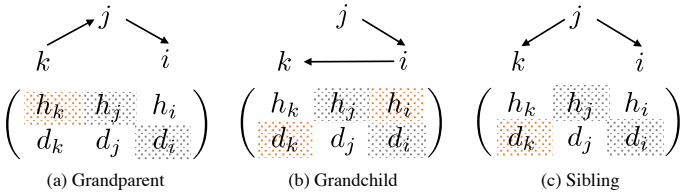


图 3.22　三种类型的高阶信息集成在依存边 (j,i) 中

GNNDP 采用以下协议:

$$\begin{cases} \boldsymbol{h}_i^t = g\left(W_1 \sum_{j\in\mathcal{N}(i)} \alpha_{ji}^t \boldsymbol{h}_j^{t-1} + B_1 \boldsymbol{h}_i^{t-1}\right) \\ \boldsymbol{d}_i^t = g\left(W_2 \sum_{j\in\mathcal{N}(i)} \alpha_{ij}^t \boldsymbol{d}_j^{t-1} + B_2 \boldsymbol{d}_i^{t-1}\right) \end{cases} \tag{3.35}$$

为了纳入 i 的兄弟结点（同样用 k 表示）的信息（图 3.22(c)），$\boldsymbol{h}_j^t$ 的更新涉及到 $\boldsymbol{d}_k^{t-1}$，这是第二个更新协议:

$$\begin{cases} \boldsymbol{h}_i^t = g\left(W_1 \sum_{j\in\mathcal{N}(i)} \alpha_{ij}^t \boldsymbol{d}_j^{t-1} + B_1 \boldsymbol{h}_i^{t-1}\right) \\ \boldsymbol{d}_i^t = g\left(W_2 \sum_{j\in\mathcal{N}(i)} \alpha_{ji}^t \boldsymbol{h}_j^{t-1} + B_2 \boldsymbol{d}_i^{t-1}\right) \end{cases} \tag{3.36}$$

将公式3.35和3.36整合到一个更新中，以统一的方式处理三种高阶信息。与一般的 GNNs 相比，上述的更新方式是为解析任务定制的。最后，除了默认的同步设置，GNNDP 还研究了异步更新版本，最终的更新协议如下:

$$\begin{cases} \boldsymbol{h}_i^{t-\frac{1}{2}} = g\left(W_1 \sum_{j\in\mathcal{N}(i)} \left(\alpha_{ji}^t \boldsymbol{h}_j^{t-1} + \alpha_{ij}^t \boldsymbol{d}_j^{t-1}\right) + B_1 \boldsymbol{h}_i^{t-1}\right) \\ \boldsymbol{d}_i^t = g\left(W_2 \sum_{i\in\mathcal{N}(i)} \left(\alpha_{ij}^t \boldsymbol{h}_j^{t-\frac{1}{2}} + \alpha_{ji}^t \boldsymbol{d}_j^{t-1}\right) + B_2 \boldsymbol{d}_i^{t-1}\right) \end{cases} \tag{3.37}$$

训练目标由两部分组成。第一部分是 GNNs 层后面有一个解码器（用 τ 表示），它将对树结构（使用 $P^\tau(i|j)$）和边标签（使用 $P(r|i,j)$）进行解码。为了预测边标签，使用用另一个 MLP 来衡量依存边 (i,j) 具有关系 r 的可能性。其交叉熵损失函数定义如下:

$$\mathcal{L}_0 = -\frac{1}{n} \sum_{(i,j,r)\in T} \left(\log P^\tau(i \mid j) + \log P(r \mid i,j)\right) \tag{3.38}$$

第二个部分是每个 GNNs 层的 $P^t(i|j)$ 的准确率进行监督（只对树结构进行监督，忽略边的标签）。每层交叉熵损失函数定义如下:

$$\mathcal{L}' = \sum_{t=1}^{\tau} \mathcal{L}_t = \sum_{t=1}^{\tau} -\frac{1}{n} \sum_{(i,j,r)\in T} \log P^t(i \mid j) \tag{3.39}$$

最终目标是最小化两者的加权组合 $L = \lambda_1 \mathcal{L}_0 + \lambda_2 \mathcal{L}'$。训练完成后，解析器在所有的依存边得分上使用最大生成树算法形成一棵有效的依存树。

3.3.3 基于转移的依存句法分析

转移系统（Transition system）包含状态（state 或 configuration）集合以及状态之间的转移动作（transition）集合。有限状态自动机就是属于转移系统的最简单的一种，但是应用于依存句法分析的转移系统更为复杂。标准弧（arc-Standard）转移系统[23] 是其中最常用的投射性依存句法分析转

移系统之一。按照标准弧转移系统定义，状态 $c = (\sigma, \beta, A)$ 由三个部分组织，σ 表示已处理的单词的堆栈，β 表示尚未处理的单词的缓冲器，A 表示已经构建的依存关系边集合。对于给定的句子 $S = w_0, w_1, \ldots, w_n$，其初始状态 $c_0(S) = ([w_0]_\sigma, [w_1, \ldots, w_n]_\beta, [\varnothing]_A)$，$(\sigma, []_\beta, A)$ 则对应终止状态的形式。标准弧转移系统中转移动作包含以下三类：

(1) **左弧**（LA_r）：$([\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_j|\beta, A \cup \{(w_j, r, w_i)\})$，$w_i$ 是堆栈 σ 最上面的单词，w_j 是缓冲器 β 中最前面的单词，将 w_i 从堆栈中弹出并增加从 w_j 到 w_i 关系类型为 r 的依存关系。前置条件是 $i \neq 0$ 并且 $\neg\exists w_k \exists r'[(w_k, r', w_i) \in A]$。

(2) **右弧**（RA_r）：$(\sigma|w_i, w_j|\beta, A) \Rightarrow (\sigma, w_i|\beta, A \cup \{(w_i, r, w_j)\})$，$w_i$ 是堆栈 σ 最上面的单词，w_j 是缓冲器 β 中最前面的单词，将 w_i 从堆栈中弹出，将缓冲器中最前面的单词 w_j 替换为 w_i，增加从 w_i 到 w_j 关系类型为 r 的依存关系。前置条件是 $\neg\exists w_k \exists r'[(w_k, r', w_j) \in A]$。

(3) **移进**（SH）：$(\sigma, w_i|\beta, A) \Rightarrow (\sigma|w_i, \beta, A)$，从缓冲器中移除最前面的单词 w_i，并压入堆栈中。

针对一个句子的转移动作序列（transition sequence）可以对应状态序列 $C_{0,m} = (c_0, c_1, \ldots, c_m)$。$c_0$ 表示起始状态，$c_0(S)$ 表示针对句子 S 的起始状态，c_m 是终止状态。转换动作为 $t \in \mathcal{T}$ 使得状态 c_{i-1} 转换到状态 c_i 的，使用 $c_i = t(c_{i-1})$ 表示。针对句子“她非常喜欢跳芭蕾”的依存句法分析转移动作序列如表3.1所示。

表 3.1　依存句法分析转移动作序列样例

	σ	β	A
	([ROOT],	[她, ...],	∅)
SH⇒	([ROOT, 她],	[非常, ...],	∅)
SH⇒	([ROOT, 她, 非常],	[喜欢, ...],	∅)
LA_{advmod} ⇒	([ROOT, 她],	[喜欢, ...],	A_1=(喜欢, advmod, 非常)
LA_{nsubj} ⇒	([ROOT],	[喜欢, ...],	A_2=A_1∪(喜欢, nsubj, 她)
SH⇒	([ROOT, 喜欢],	[跳, ...],	A_2)
SH⇒	([ROOT, 喜欢，跳],	[芭蕾舞],	A_2)
RA_{dobj} ⇒	([ROOT，喜欢],	[跳],	A_3=A_2∪(跳, dobj, 芭蕾舞)
RA_{xcomp} ⇒	([ROOT],	[喜欢],	A_4=A_3∪(喜欢, xcomp, 跳)
RA_{root} ⇒	([],	[ROOT],	A_5=A_4∪(ROOT, root, 喜欢)
SH⇒	([ROOT],	[],	A_5)

假设存在函数 o，可以根据当前的状态 c 正确的确定下一步的转移动作 t，即 $o(c) = t$。那么整个句法分析的过程就可以使用非常简单的贪心算法完成，伪代码如算法3.7所示。针对输入句子 S，首先，构造初始状态 $c_0(S)$，调用函数 o 得到下一步转移动作 $t = o(c)$。之后，利用根据转移动作得到下一个状态 $c = t(c)$。如此循环直到达到终止状态形式 $(\sigma, []_\beta, A)$。

转移动作预测函数可以使用分类器进行建模。构造一个分类器 $f(c)$，输入为状态 c，输出为其

代码 3.7: 基于转移动作函数 o 的依存句法分析贪心算法

$c = c_0(S)$;
while c 不是终止状态形式 $(\sigma, []_\beta, A)$ **do**
  $t = o(c)$;
  $c = t(c)$;
end
return G_c

所对应的标准转移动作 $o(c)$。由此，基于转移的依存句法分析问题转化为了典型的机器学习问题。如果采用有监督机器学习算法，那么需要解决如下三个基本问题：1）如何表示状态 c；2）如何构造训练语料；3）如何选择和训练分类器。

针对状态 c 的表示问题，可以采用从堆栈 σ、缓存器 β 以及边集合 A 中对特定位置的单词、单词词性、树结构等抽取信息构造特征向量的方法。Maltparser[23] 针对标准弧转移系统利用如表3.2所示的信息构造针对状态 c 的特征向量。$\beta[0]$ 表示缓冲器中最前面的单词，$\sigma[0]$ 表示堆栈最顶端的单词，$ld(x)$ 表示 x 最左面的修饰词，$rd(x)$ 表示 x 最右面的修饰词。利用这些特征模板，可以构造状态的高维向量表示。使用 $\mathbf{f}(c)$ 表示特征函数，其输出是状态 c 的特征向量。

表 3.2 Maltparser[23] 针对标准弧转移系统采用的特征向量

	单词	词元	粗粒度词性	细粒度词性	词形特征	依存关系类型
$\beta[0]$	×	×	×	×	×	
$\beta[1]$	×			×		
$\beta[2]$				×		
$\beta[3]$				×		
$ld(\beta[0])$						×
$rd(\beta[0])$						×
$\sigma[0]$	×	×	×	×	×	
$\sigma[1]$				×		
$ld(\sigma[0])$						×
$rd(\sigma[0])$						×

依存句法树库 $\mathbb{D} = \{(S_d, G_d)|_{d=0}^{|\mathbb{D}|}\}$ 是由大规模的句子 S_d 以及所对应的正确的依存句法树 G_d 组成。但是用于转移动作预测的分类器所需的训练数据集 $\mathbb{D}'$ 需要由状态的特征表示 $\mathbf{f}(c)$ 和对应的正确的转移动作 t 组成，$\mathbb{D}' = \{(\mathbf{f}(c_d), t_d)|_{d=0}^{|\mathbb{D}'|}\}$。因此，需要将原始依存句法树库 $\mathbb{D}$ 中每个句子和对应依存句法树 (S_d, G_d) 转换为转移序列 $C_{0,m}^d = (c_0^d, c_1^d, \ldots, c_m^d)$。其中每个非终结状态 $c_i^d \in C_{o,m}^d$，

都可以根据如下方式得到其对应的转移动作 $t_i^d = o(c_i^d)$:

$$o(c=(\sigma,\beta,A)) = \begin{cases} LA_r & \text{如果}(\beta[0], r, \sigma[0]) \in A_d \\ RA_r & \text{如果}(\sigma[0], r, \beta[0]) \in A_d \text{ 并且} \\ & \text{如果}(\beta[0], r', w) \in A_d \text{那么}(\beta[0], r', w) \in A \\ SH & \text{其他情况} \end{cases} \tag{3.40}$$

基于上述公式，可以根据非终结状态 c_i^d 和对应的转移动作 t_i^d，添加实例 $(\mathbf{f}(c_i^d), t_i^d)$ 到训练数据集合 $\mathbb{D}'$ 中。

利用训练数据集合 $\mathbb{D}' = \{(\mathbf{f}(c_d), t_d)_{d=0}^{|D'|}\}$ 可以选用各类型的分类器，已经成功应用于该任务的分类器包括：SVM[24]、基于记忆的学习[25]、最大熵[26] 等。相关的有监督分类算法在机器学习相关书籍和本书其他章节中也多有介绍，这里就不再赘述。

3.3.4 基于神经网络的转移依存句法分析

基于转移的依存句法分析通过定义转移系统，将依存句法分析转换为了根据当前状态 $c = (\sigma, \beta, A)$ 预测转移动作 t 的分类问题。上节中介绍的传统方法依赖特征工程，需要人工构建特征模板，构建状态的特征向量表示，在此基础上利用分类模型构建转移动作预测模型。神经网络方法不需要人工设计特征函数，并且可以在一定程度上缓解人工设计特征所带来的泛华能力不足的问题。在本节中，将介绍两种基于神经网络转移依存句法分析算法。

1. 基于前馈神经网络的方法

Chen 和 Manning 基于 MaltPraser 的基本思想，提出了将当前状态 c 中的堆栈 σ 和缓冲器 β 利用神经网络提取特征，并预测下一步的转移动作的方法[27]，其神经网络结构如图3.23所示。

神经网络结构由三层组成：输入层、隐藏层和输出层。输入层是由单词嵌入、词性嵌入以及边类别嵌入。单词嵌入表示（word embedding）由 d 维向量 $\boldsymbol{e}_i^w \in \mathbb{R}^d$ 表示，整个单词嵌入矩阵使用 $\boldsymbol{E}^w \in \mathbb{R}^{d \times N_w}$ 表示，其中 N_w 表示词典中单词数量。同样的使用 $\boldsymbol{e}_i^t$ 和 $\boldsymbol{e}_j^l$ 表示第 i 个词性标签的嵌入和第 j 个边类别的嵌入表示。对应的词性标签嵌入矩阵使用 $\boldsymbol{E}^t \in \mathbb{R}^{d \times N_t}$ 表示，其中 N_t 表示词性数量，边类别标签嵌入矩阵使用 $\boldsymbol{E}^l \in \mathbb{R}^{d \times N_l}$ 表示，其中 N_l 表示边类型数量。

单词嵌入、词性嵌入以及边类别嵌入根据设定的上下文分别连接起来构成 $\boldsymbol{x}^w, \boldsymbol{x}^t, \boldsymbol{x}^l$。$\boldsymbol{x}^w$ 是由堆栈最顶端的 3 个单词和缓冲器中最前面的 3 个单词：$\sigma[0], \sigma[1], \sigma[2], \beta[0], \beta[1], \beta[2]$，以及堆栈中最顶端的 2 个单词的最左、次左、最右和次右的儿子节点：$lc_1(\sigma[i]), lc_2(\sigma[i]), rc_1(\sigma[i]), rc_2(\sigma[i]), i = 1, 2$,再加上堆栈中最顶端的 2 个单词的最左的孙子节点和最右的孙子节点：$lc_1(lc_1(\sigma[i])), rc_1(rc_1(\sigma[i])), i = 1, 2$，共 18 个单词嵌入连接组成。$\boldsymbol{x}^t$ 是由组成 $\boldsymbol{x}_w$ 的 18 个单词的词性嵌入组成。$\boldsymbol{x}^l$ 是 $\boldsymbol{x}^w$ 中除了堆栈和缓冲器的 6 个单词之外的其他 12 个单词对应的边类别嵌入连接组成。

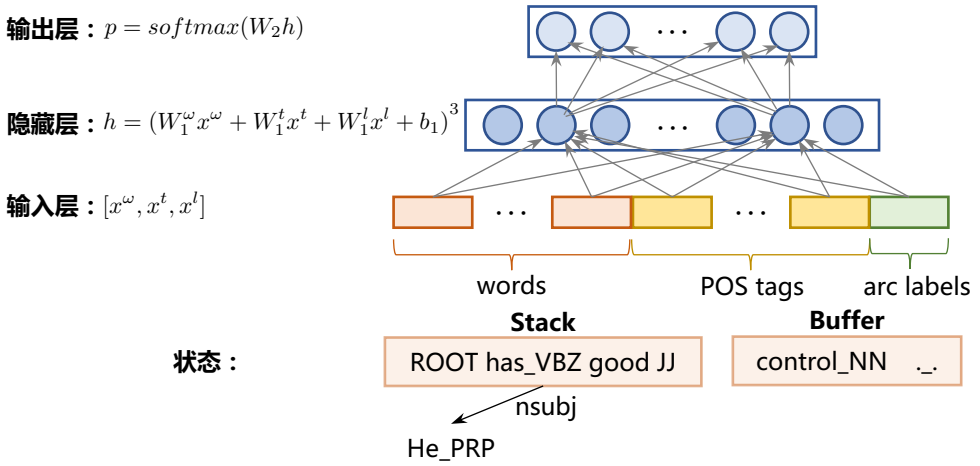


图 3.23　基于神经网络的依存句法分析算法神经网络结构图[27]

隐藏层为了有效的融合单词、词性以及边类别信息，采用立方激活函数结合线性变换：

$$\boldsymbol{h} = \left(\boldsymbol{W}_1^w \boldsymbol{x}^w + \boldsymbol{W}_1^t \boldsymbol{x}^t + \boldsymbol{W}_1^l \boldsymbol{x}^l + \boldsymbol{b}_1\right)^3 \tag{3.41}$$

其中，$\boldsymbol{W}_1^w \in \mathbb{R}^{d_h \times (d \cdot n_w)}$，$\boldsymbol{W}_1^t \in \mathbb{R}^{d_h \times (d \cdot n_t)}$，$\boldsymbol{W}_1^l \in \mathbb{R}^{d_h \times (d \cdot n_l)}$，$\boldsymbol{b}_1 \in \mathbb{R}^{d_h}$，$n_w$ 是 $\boldsymbol{x}^w$ 中单词的数量，n_t 是 $\boldsymbol{x}^t$ 中词性的数量，n_l 是 $\boldsymbol{x}^l$ 中边标签的数量，d_h 是隐藏单元 $\boldsymbol{h}$ 的维度。

输出层则是采用标准的 softmax 函数决定转移动作 $P = \text{softmax}(\boldsymbol{W}_2 h)$，其中 $\boldsymbol{W}_2 \in \mathbb{R}^{|\mathcal{T}| \times d_h}$，$\mathcal{T}$ 是转移动作集合。

最后，可以根据上节所介绍的通过依存句法树库构造训练语料 $\{(c_i, t_i)\}_{i=1}^m$，其中 c_i 是状态，$t_i \in \mathcal{T}$ 是对应的转移动作。利用交叉熵损失做为训练目标：

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_i \log P(t_i|c_i) + \frac{\lambda}{2}||\boldsymbol{\theta}||^2 \tag{3.42}$$

利用优化算法学习参数集合 $\boldsymbol{\theta} = \{\boldsymbol{W}_1^w, \boldsymbol{W}_1^t, \boldsymbol{W}_1^l, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{E}^w, \boldsymbol{E}^t, \boldsymbol{E}^l\}$。

2. 基于堆栈长短时记忆网络的方法

Dyer 等人针对基于转移的依存句法分析中如何对堆栈表示的问题提出了一种堆栈长短时记忆网络（Stack-LSTM）[28]。不同于标准长短时记忆网络的序列是从左到右的顺序，基于转移的依据句法分析中需要使用堆栈完成句法分析，因此堆栈长短时记忆网络针对堆栈的压栈（push）和出栈（pop）操作，增加了“栈指针”（stack pointer）来扩充 LSTM。针对出栈操作，只是将堆栈指针移动到栈顶的前一个元素。针对压栈操作，需要添加对应的输入 $\boldsymbol{x}_t$，长短时记忆网络的内部状

态计算 $\boldsymbol{c}_t$ 所依赖的前一个状态的 $\boldsymbol{c}_{t-1}$ 和 $\boldsymbol{h}_{t-1}$ 都指向栈指针所指向的位置。堆栈长短时记忆网络压栈和出栈操作样例如图3.24所示。Stack-LSTM 在原始从左至右 LSTM 的基础上，增加了堆栈指针（图中标记为 TOP）。图3.24给出了三种状态：带有单个元素的堆栈 (左侧)、堆栈执行出栈操作结果 (中部)，以及此后堆栈使用压栈操作的结果 (右侧)。最底层代表堆栈的内容，是 Stack-LSTM 的输入，中间层是 LSTM 单元，上层是输出层。使用堆栈指针指向的位置做为输出表示。通过上述操作我们可以看到，在执行出栈操作时，Stack-LSTM 中间层 LSTM 单元并不发生改变，仅是修改堆栈指针 TOP 的位置。在执行压栈操作时，新增加的元素增加在最右端，并与前一个 TOP 指针指向的 LSTM 单元连接。

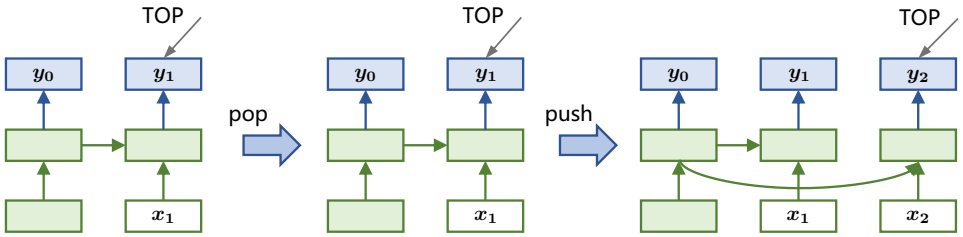


图 3.24　堆栈长短时记忆网络压栈和出栈操作样例[28]

分析算法采用标准弧转移系统，状态 $c = (\sigma, \beta, A)$ 包含已处理的单词的堆栈（S），尚未处理的单词的缓冲器（B），以及分析算法所执行的转移动作历史（A）。利用三个堆栈长短时记忆网络分别对状态中堆栈、缓存器以及转移动作历史进行表示，其神经网络架构如图3.25所示。

在 t 时刻，句法分析器状态的表示 $\boldsymbol{p}_t$ 根据三个堆栈长短时记忆网络的输出经过线性变换和非线性 ReLU 函数转换后得到。$\boldsymbol{p}_t$ 具体计算公式如下：

$$\boldsymbol{p}_t = \max\{\boldsymbol{0}, \boldsymbol{W}[\boldsymbol{s}_t, \boldsymbol{b}_t, \boldsymbol{a}_t] + \boldsymbol{d}\} \tag{3.43}$$

由此可以得到在 t 时刻在每个转移动作的概率：

$$P(z_t|\boldsymbol{p}_t) = \frac{\exp \boldsymbol{g}_{z_t}^{\top}\boldsymbol{p}_t + q_{z_t}}{\sum_{z' \in \mathcal{A}(S,B)} \exp \boldsymbol{g}_{z'}^{\top}\boldsymbol{p}_t + q_{z'}} \tag{3.44}$$

其中 $\boldsymbol{g}_z$ 表示转移动作 z 的向量表示，q_z 是转移动作 z 对应的偏置项，$\mathcal{A}(S, B)$ 是给定当前堆栈 S 和缓冲器 B 状态时所有可能的转移动作集合。根据每个转移动作的概率可以选择下一步的转移动作，从而完成依存句法分析。

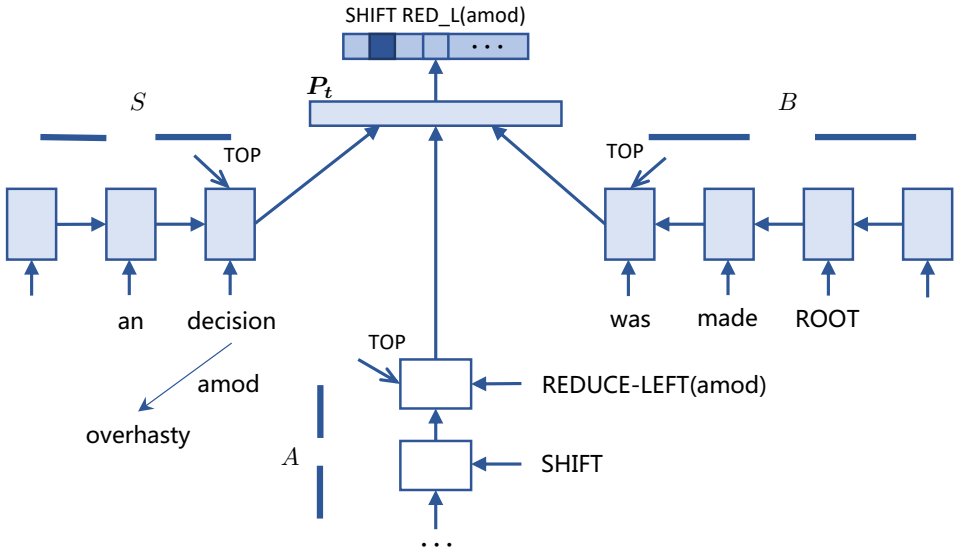


图 3.25　基于堆栈长短时记忆网络的状态表示神经网络结构图[28]

3.3.5 依存句法分析评价方法

依存句法分析算法的性能评价，目前较为常用的指标是由 Yamada 和 Matsumoto 在其利用支持向量机进行依存句法分析的论文中采用的三种指标[29]：

(1) **依存正确率**（Dependency Accuracy，简称 DA）：正确分析得到其中心词的非根节点词语个数占总非根节点词数的百分比；
(2) **根正确率**（Root Accuracy，简称 RA）：正确根节点的个数与句子个数的百分比；
(3) **完全匹配率**（Complete Match，简称 CM）：无标记依存结构完全正确的句子占句子总数的百分比；

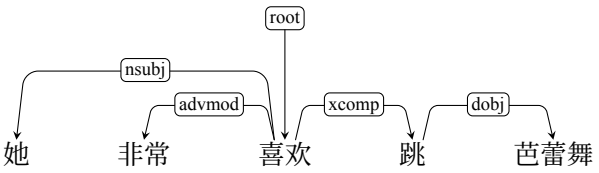
此外，如果考虑所有词的依存正确率，根据是否考虑依存标记，还可以使用**无标记依存正确率**（Unlabeled attachment score，简称 UAS）和**有标记依存正确率**（Labeled attachment score，简称 LAS）两个指标。UAS 表示正确分析得到其修饰词的词语个数占总词数的百分比。LAS 表示正确分析得到其修饰词以及依存关系的词语个数占总词数的百分比。

图3.26给出了依存句法分析样例，图3.26(a) 表示句子“他非常喜欢跳芭蕾舞”和正确分析结果，图3.26(b) 表示算法分析结果。根据上述评价指标定义，可以得到如下评价结果：

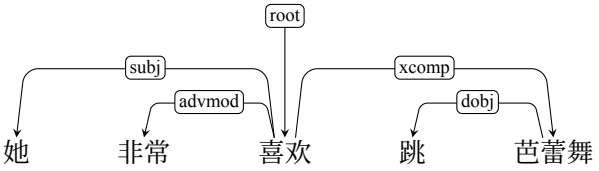
$$\text{依存正确率（DA）} = \frac{2}{4} \times 100\% = 50\%$$

$$\text{无标记依存正确率（UAS）} = \frac{3}{5} \times 100\% = 60\%$$

$$\text{有标记依存正确率（LAS）} = \frac{2}{5} \times 100\% = 40\%$$



(a) 正确结果



(b) 算法分析结果

图 3.26 依存句法分析结果评测样例

3.4 句法分析语料库

句法分析语料库也称为句法树库，包含大规模句子以及其对应句法树的集合。通过本章的介绍，可以知道很多句法分析方法都转换为了有监督机器学习算法，因此句法分析算法的训练和评测都依赖语料库的建设。常见句法分析语料库如表3.3所示。本节将介绍几种常见的句法分析语料库。

表 3.3 常见句法分析语料库汇总

语料库名称	单词数量	语法类型	语言
英语宾州树库（PTB）	117 万	成分语法	英文
通用依存树库（UD V2.0 CoNLL 2017）	281 万	依存语法	多语言
通用依存树库（UD V2.2 CoNLL 2018）	1714 万	依存语法	多语言
组合范畴语法树库（CCGBank）	116 万	组合范畴语法	英文
中文宾州树库 6.0（CTB 6.0）	78 万	成分语法	中文
中文宾州树库 7.0（CTB 7.0）	120 万	成分语法	中文
中文宾州树库 8.0（CTB 8.0）	162 万	成分语法	中文
中文宾州树库 9.0（CTB 9.0）	208 万	成分语法	中文
中文语义依存树库（SDP）	52 万	语义依存	中文

1. 英语宾州树库

英语宾州树库（English Penn Treebank，PTB）是最知名和最常用的短语结构句法树库之一。语言资源联盟（Linguistic Data Consortium，LDC）1999 年的发行版 Treebank-3 是 PTB 最常用的发行版本。Treebank-3 中包含四个部分：ATIS（Air Travel Information System 标注样例）、BROWN（布朗语料库标注）、SWBD（Switchboard 口语数据标注）、WSJ（华尔街日报标注）。WSJ 是英语宾州树库中最常用的部分，其原始数据来自于 1989 年华尔街日报文章，按照 PTB(V2) 的标注策略进行标注，使用括号表示树结构。WSJ 部分总计包含 49208 个句子，1173766 个单词，分成 25 个章（section），通常使用 0 到 18 章作为训练集合，19 到 21 章作为验证集合，22 到 24 章作为测试集合。

示例：

```
( (S
    (NP-SBJ (DT Both) (NNS distributions) )
    (VP (VBP are)
      (ADJP-PRD (JJ payable)
        (NP-TMP (NNP Dec.) (CD 4) )
        (PP (TO to)
          (NP
            (NP (JJ limited) (NNS partners) )
            (PP (IN of)
              (NP (NN record) ))
            (NP-TMP (NNP Nov.) (CD 3) )))))
    (. .) ))
```

2. 中文宾州树库

中文宾州树库（Chinese Penn Treebank，CTB）是目前最常用的大规模中文短语结构句法标注语料库之一。1998 年开始构建，Chinese Treebank 1.0 规模达到 10 万词语规模，其原始数据来源于新华社新闻报道文章。2016 年发布了最新的 Chinese Treebank 9.0 版本，包含中文新闻网站、政府文书、杂志文章、新闻群组、广播对话节目、博客等类不同来源的 3726 篇文章，共计 132076 个句子，2084387 个单词，3247331 个中文和外文字符。对这些文章中的句子进行了分词、词性标注和成分句法树标注。采用了英语宾州树库的括号结构对树结构进行表示。

示例：

```
( (IP (NP-SBJ (DNP (NP-PN (NR 北海市))
                   (DEG 的))
```

```
                (NP (NN 崛起 )))
    (PU ， )
    (VP (VC 是 )
        (NP-PRD (CP-APP (IP (IP-SBJ (LCP-TMP (NP (NT 近年 ))
                                             (LC 来 ))
                                    (NP-PN-SBJ (NR 广西 )
                                               (NN 壮族 )
                                               (NN 自治区 ))
                                    (VP (PP-DIR (P 对 )
                                                (NP (NN 外 )))
                                        (VP (VV 开放 ))))
                                (VP (VV 取得 )
                                    (NP-OBJ (ADJP (JJ 卓著 ))
                                            (NP (NN 成就 )))))
                        (DEC 的 ))
                (ADJP (JJ 重要 ))
                (NP (NN 标志 )
                    (NN 之一 ))))
    (PU 。 )) )
```

3. 通用依存树库

通用依存树库（Universal Dependencies，UD）是一个为多种语言开发的跨语言一致的依存句法树库项目。标注方案采用斯坦福通用依存标签[7, 30, 31]、Google 通用词性标签[32] 以及形态句法标签集[33]。目前共有超过 300 个贡献者提供了 100 多种语言上的 200 多个依存句法树库。树库中包含了原始句子、词语、词性、依存关系以及依存关系类型等信息。

示例：

1	同样	同样	ADJ	JJ	_	3	amod	_	SpaceAfter=No
2	的	的	PART	DEC	Case=Gen	1	case	_	SpaceAfter=No
3	手法	手法	NOUN	NN	_	5	nsubj:pass	_	SpaceAfter=No
4	被	被	VERB	BB	Voice=Pass	5	aux:pass	_	SpaceAfter=No
5	用	用	VERB	VV	_	0	root	_	SpaceAfter=No
6	于	于	VERB	VV	_	5	mark	_	SpaceAfter=No
7	现代	现代	NOUN	NN	_	10	nmod	_	SpaceAfter=No
8	的	的	PART	DEC	Case=Gen	7	case	_	SpaceAfter=No
9	摄影	摄影	NOUN	NN	_	10	nmod	_	SpaceAfter=No
10	技术	技术	NOUN	NN	_	5	obj	_	SpaceAfter=No
11	中	中	ADP	IN	_	10	acl	_	SpaceAfter=No
12	。	。	PUNCT	.	_	5	punct	_	SpaceAfter=No

4. 中文语义依存树库

中文语义依存树库（Chinese Semantic Dependency Parsing，SDP）是由哈尔滨工业大学车万翔教授在 SemEval-2016 发布。语料库包含包含从新闻中选取的 10068 个句子和从小学课文中选取的 14793 个句子。新闻句子平均长度是 31 个词，课文句子平均长度是 14 个词。与传统的依存句法树库不同，中文语义依存树库从语义角度构建依存关系，并依存结构扩展到符合汉语特点的有向无环图-汉语语义依存图结构[34]。定义了 45 个标签用来描述论元 (Argument) 之间的语义关系，19 个标签用来描述谓词 (Predicate) 之间的关系，以及 17 个标签用来提供更丰富的谓词描述。

示例：

1	他	他	PN	PN	_	2	Exp	_	_
2	是	是	VC	VC	_	0	Root	_	_
3	研究	研究	NN	NN	_	6	rAgt	_	_
4	机器人	机器人	NN	NN	_	3	Datv	_	_
5	的	的	DEG	DEG	_	3	mAux	_	_
6	专家	专家	NN	NN	_	2	Clas	_	_

3.5 延伸阅读

句法分析句法分析是句子结构和语义之间的桥梁，具有非常重要的作用，很多自然语言处理算法需要依赖句法分析结果，因此句法分析效果也直接影响到很多自然语言处理应用。句法分析是自然语言处理中长期关注的核心问题之一。

在本章中，句法分析任务限定在得到完整的句法分析树，重点介绍了基于有监督机器学习算法的句法分析方法。在实际应用中，有很多任务并不需要使用完整的句法分析树，仅依赖部分或者粗粒度的分析结果，这种句法分析称之为部分句法分析（Partial Parsing）又称浅层句法分析（Shallow

Parsing)。组块分析（Chunking）是最常用的浅层句法分析任务，目标是将句子分解为不重叠的片段，这些片段是由主要内容词的非递归短语组成，包括：名词短语、动词短语、形容词短语、介词短语等。

例如：句子 He eat soup with spoon.

组块分析结果：[$_{NP}$ He] [$_{VP}$ eat soup] [$_{PP}$ with spoon.]

组块分析通常也转换为分类问题，包括规则[35]、最大熵[36]、支撑向量机[37]、条件随机场[38]、卷积神经网[39] 等方法都针对组块分析任务开展了研究。

在成分句法分析方面，除了本章中所介绍算法之外，还有一些工作从如何对句法分析历史进行表示对子树评分[40]、基于神经网络的方法对子树和标签评分[41, 42]、将句法分析任务转换为机器翻译任务[43, 44]、利用特定的注意力机制[45, 46]、多语言预训练[47] 等方面开展了深入研究。此外针对大规模句法树标注困难的问题，一些研究工作引入了无标注数据，设计了多种半监督算法以降低对标注数据需求，包括基于期望最大化算法（Expectation Maximization）[48, 49]、自监督方法[50]、半监督重排序[51] 等。另外一些研究工作试图通过跨语言迁移方法，将标注资源丰富的汉语、英语语料库或者模型迁移到资源缺乏的语言中，采用了包括标注迁移[52]、结合机器翻译算法[53]、模型迁移等方法[54]。还有一些方法利用无标记数据，包括基于语言模型预训练[55]、递归自编码器（Recursive Auto-Encoders）[56]、标准化流 (Normalizing Flow)[57] 等方法。

在依存句法分析方面，针对如何减少大规模训练语料问题，很多工作从无监督方法、半监督方法以及迁移方法三个方面开展研究。在无监督方法方面，研究者们提出了基于有价的依存关系理论[58–60]，维特比期望最大化（Viterbi Expectation Maximization）[61]、声学线索[62]、条件随机场自编码器[63] 等方法。在半监督方法方面，研究者们针对无监督数据特征提取[64]、协同训练[65]、特征转换[66]、歧义感知的集成学习[67]、自监督训练[68]、弧因子变分自编码器（Arc-factored variational autoencoding）[69]、跨语言训练[70–72] 等方法开展了系列的工作。在迁移方法方面，提出了跨语言上下文表示对齐[73]、分布迁移[74]、跨语言 BERT 模型迁移[75] 等方法。此外，还有一些工作面向社会媒体非规范语言的依存句法分析[76]、针对混合语言句子的依存句法分析[77]、基于序列生成方法的依存句法分析[78]、基于二阶树条件随机场的依存句法分析[79] 等围绕依存句法分析的各个方面开展研究。

综上所述，句法分析在有监督算法、无监督算法、跨领域学习、跨语言迁移、高效解码等方面都有系统的长期研究。

3.6 习题

(1) 如何判断一个语法理论属于成分语法还是依存语法?

(2) 试举例说明什么是句法范畴以及句法范畴之间的层级关系。

(3) 除了本章中介绍的标准弧转移系统，还有什么转移系统可以应用于依存句法分析？试说明该种转移系统的优缺点。

(4) 基于转移的依存句法分析相较于基于图的依存句法分析有什么优缺点?

(5) 通常情况下对中文句子进行句法分析时，需要首先进行分词，并在基础上对词语进行词性标注，然后进行句法分析。这种流水线方法有什么优点和缺点？如何设计一种方法可以同时进行中文分词、词性标注和句法分析?

(6) 试比较几种开源句法分析器在常见数据集合上的性能。

参考文献

[1] Fromkin V, Rodman R, Hyams N. An introduction to language[M]. Cengage Learning, 2018.

[2] 维多利亚·弗罗姆金 (著), 罗伯特·罗德曼 (著), 王大惟 (译), 等. 语言引论 (第八版)[M]. 北京大学出版社, 2017.

[3] 崔应贤. 现代汉语语法学习与研究入门[M]. 清华大学出版社有限公司, 2004.

[4] Chomsky N. Syntactic structures[M]. De Gruyter Mouton, 2009.

[5] Lucien T. Eléments de syntaxe structurale[J]. Paris, Klincksieck, 1959:25.

[6] Carroll J, Briscoe T, Sanfilippo A. Parser evaluation: a survey and a new proposal[C]//Proceedings of the 1st International Conference on Language Resources and Evaluation: volume 32. Granada, 1998.

[7] De Marneffe M C, MacCartney B, Manning C D, et al. Generating typed dependency parses from phrase structure parses.[C]//Lrec: volume 6. 2006: 449-454.

[8] Cocke J. Programming languages and their compilers: Preliminary notes[M]. New York University, 1969.

[9] Younger D H. Recognition and parsing of context-free languages in time n3[J]. Information and control, 1967, 10(2):189-208.

[10] Kasami T. An efficient recognition and syntax-analysis algorithm for context-free languages[J]. Coordinated Science Laboratory Report no. R-257, 1966.

[11] Ullman J D. The theory of parsing, translation, and compiling[M]. Prentice-Hall, 1972.

[12] Lari K, Young S J. The estimation of stochastic context-free grammars using the inside-outside algorithm[J]. Computer speech & language, 1990, 4(1):35-56.

[13] Manning C, Schutze H. Foundations of statistical natural language processing[M]. MIT press, 1999.

[14] Black E, Abney S, Flickenger D, et al. Strzalkozskijl. 1991. a procedure for quantitatively comparing the syntactic coverage of english grammars[C]//Proceedings of the 4th DARPA Speech and Natural Language Workshop. 306-311.

[15] Kubler S, McDonald R, Nivre J. Dependency parsing[M]. Morgan & Claypool Publishers, 2009.

[16] Chu Y J. On the shortest arborescence of a directed graph[J]. Scientia Sinica, 1965, 14:1396-1400.

[17] Edmonds J. Optimum branchings[J]. Journal of Research of the National Bureau of Standards, B, 1967, 71:233-240.

[18] McDonald R, Pereira F, Ribarov K, et al. Non-projective dependency parsing using spanning tree algorithms[C]//Proceedings of human language technology conference and conference on empirical methods in natural language processing. 2005: 523-530.

[19] Pei W, Ge T, Chang B. An effective neural network model for graph-based dependency parsing[C]//Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2015: 313-322.

[20] Dozat T, Manning C D. Deep biaffine attention for neural dependency parsing[C/OL]//5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017. https://openreview.net/forum?id=Hk95PK9le.

[21] Ji T, Wu Y, Lan M. Graph-based dependency parsing with graph neural networks[C]//Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019: 2475-2485.

[22] Velivcković P, Cucurull G, Casanova A, et al. Graph attention networks[C]//International Conference on Learning Representations. 2018.

[23] Nivre J. Algorithms for deterministic incremental dependency parsing[J]. Computational Linguistics, 2008, 34(4):513-553.

[24] Nivre J, Hall J, Nilsson J. Maltparser: A data-driven parser-generator for dependency parsing.[C]//LREC: volume 6. 2006: 2216-2219.

[25] Nivre J, Hall J, Nilsson J. Memory-based dependency parsing[C]//Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004. 2004: 49-56.

[26] Che W, Li Z, Hu Y, et al. A cascaded syntactic and semantic dependency parsing system[C]//CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning. 2008: 238-242.

[27] Chen D, Manning C D. A fast and accurate dependency parser using neural networks[C]//Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014: 740-750.

[28] Dyer C, Ballesteros M, Ling W, et al. Transition-based dependency parsing with stack long short-term memory[C]//Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2015: 334-343.

[29] Yamada H, Matsumoto Y. Statistical dependency analysis with support vector machines[C]// Proceedings of the eighth international conference on parsing technologies. 2003: 195-206.

[30] De Marneffe M C, Manning C D. The stanford typed dependencies representation[C]//Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation. 2008: 1-8.

[31] De Marneffe M C, Dozat T, Silveira N, et al. Universal stanford dependencies: A cross-linguistic typology.[C]//LREC: volume 14. 2014: 4585-4592.

[32] Petrov S, Das D, McDonald R. A universal part-of-speech tagset[C]//Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12). 2012: 2089-2096.

[33] Zeman D. Reusable tagset conversion using tagset drivers[C]//Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08). 2008.

[34] Che W, Shao Y, Liu T, et al. Semeval-2016 task 9: Chinese semantic dependency parsing[C]// Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016). 2016: 1074-1080.

[35] Grover C, Tobin R. Rule-based chunking and reusability[C]//Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC' 06). 2006.

[36] Koeling R. Chunking with maximum entropy models[C]//Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop. 2000.

[37] Kudo T, Matsumoto Y. Chunking with support vector machines[C]//Second Meeting of the North American Chapter of the Association for Computational Linguistics. 2001.

[38] Sha F, Pereira F. Shallow parsing with conditional random fields[C]//Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics. 2003: 213-220.

[39] Collobert R, Weston J, Bottou L, et al. Natural language processing (almost) from scratch[J]. Journal of machine learning research, 2011, 12(ARTICLE):2493-2537.

[40] Henderson J. Inducing history representations for broad coverage statistical parsing[C]//Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics. 2003: 103-110.

[41] Stern M, Andreas J, Klein D. A minimal span-based neural constituency parser[C]//Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vancouver, Canada: Association for Computational Linguistics, 2017: 818-827.

[42] Gaddy D, Stern M, Klein D. What's going on in neural constituency parsers? an analysis[C]//Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). 2018: 999-1010.

[43] Vinyals O, Kaiser L, Koo T, et al. Grammar as a foreign language[J]. Advances in neural information processing systems, 2015, 28.

[44] Liu L, Zhu M, Shi S. Improving sequence-to-sequence constituency parsing[C]//Proceedings of the AAAI Conference on Artificial Intelligence: volume 32. 2018.

[45] Kitaev N, Klein D. Constituency parsing with a self-attentive encoder[C]//Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2018: 2676-2686.

[46] Tian Y, Song Y, Xia F, et al. Improving constituency parsing with span attention[C]//Findings of the Association for Computational Linguistics: EMNLP 2020. 2020: 1691-1703.

[47] Kitaev N, Cao S, Klein D. Multilingual constituency parsing with self-attention and pre-training[C]//Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019: 3499-3505.

[48] Matsuzaki T, Miyao Y, Tsujii J. Probabilistic cfg with latent annotations[C]//Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05). 2005: 75-82.

[49] Petrov S, Barrett L, Thibaux R, et al. Learning accurate, compact, and interpretable tree annotation [C]//Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics. 2006: 433-440.

[50] Huang Z, Harper M. Self-training pcfg grammars with latent annotations across languages[C]// Proceedings of the 2009 conference on empirical methods in natural language processing. 2009: 832-841.

[51] Liu J, Zhang Y. In-order transition-based constituent parsing[J]. Transactions of the Association for Computational Linguistics, 2017, 5:413-424.

[52] Goto I, Utiyama M, Sumita E, et al. Preordering using a target-language parser via cross-language syntactic projection for statistical machine translation[J]. ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP), 2015, 14(3):1-23.

[53] Tiedemann J, Agić vZ, Nivre J. Treebank translation for cross-lingual parser induction[C]// Proceedings of the Eighteenth Conference on Computational Natural Language Learning. 2014: 130-140.

[54] Zeman D, Resnik P. Cross-language parser adaptation between related languages[C]//Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages. 2008.

[55] Shen Y, Lin Z, Huang C w, et al. Neural language modeling by jointly learning syntax and lexicon [C]//International Conference on Learning Representations. 2018.

[56] Drozdov A, Verga P, Yadav M, et al. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders[C]//Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 2019: 1129-1141.

[57] Jin L, Doshi-Velez F, Miller T, et al. Unsupervised learning of pcfgs with normalizing flow[C]// Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019: 2442-2452.

[58] Klein D, Manning C D. Corpus-based induction of syntactic structure: Models of dependency and constituency[C]//Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL-04). 2004: 478-485.

[59] Headden III W P, Johnson M, McClosky D. Improving unsupervised dependency parsing with richer contexts and smoothing[C]//Proceedings of human language technologies: the 2009 annual conference of the North American chapter of the association for computational linguistics. 2009: 101-109.

[60] Spitkovsky V I, Alshawi H, Jurafsky D. Punctuation: Making a point in unsupervised dependency parsing[C]//Proceedings of the Fifteenth Conference on Computational Natural Language Learning. 2011: 19-28.

[61] Spitkovsky V I, Alshawi H, Jurafsky D, et al. Viterbi training improves unsupervised dependency parsing[C/OL]//Proceedings of the Fourteenth Conference on Computational Natural Language Learning. Uppsala, Sweden: Association for Computational Linguistics, 2010: 9-17. https://aclanthology.org/W10-2902.

[62] Pate J K, Goldwater S. Unsupervised dependency parsing with acoustic cues[J]. Transactions of the Association for Computational Linguistics, 2013, 1:63-74.

[63] Cai J, Jiang Y, Tu K. Crf autoencoder for unsupervised dependency parsing[C]//Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. 2017: 1638-1643.

[64] Koo T, Carreras X, Collins M. Simple semi-supervised dependency parsing[C/OL]//Proceedings of ACL-08: HLT. Columbus, Ohio: Association for Computational Linguistics, 2008: 595-603. https://aclanthology.org/P08-1068.

[65] Sogaard A, Rishoj C. Semi-supervised dependency parsing using generalized tri-training[C]//Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010). 2010: 1065-1073.

[66] Chen W, Zhang M, Zhang Y. Semi-supervised feature transformation for dependency parsing[C]//Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. 2013: 1303-1313.

[67] Li Z, Zhang M, Chen W. Ambiguity-aware ensemble training for semi-supervised dependency parsing[C]//Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2014: 457-467.

[68] Kiperwasser E, Goldberg Y. Semi-supervised dependency parsing using bilexical contextual features from auto-parsed data[C]//Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015: 1348-1353.

[69] Wang G, Tu K. Semi-supervised dependency parsing with arc-factored variational autoencoding[C]// Proceedings of the 28th International Conference on Computational Linguistics. 2020: 2485-2496.

[70] Ahmad W, Zhang Z, Ma X, et al. Cross-lingual dependency parsing with unlabeled auxiliary languages[C]//Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL). 2019: 372-382.

[71] Huang K H, Ahmad W, Peng N, et al. Improving zero-shot cross-lingual transfer learning via robust training[C]//Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 2021: 1684-1697.

[72] Ji T, Jiang Y, Wang T, et al. Word reordering for zero-shot cross-lingual structured prediction[C]// Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 2021: 4109-4120.

[73] Schuster T, Ram O, Barzilay R, et al. Cross-lingual alignment of contextual word embeddings, with applications to zero-shot dependency parsing[C]//Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 2019: 1599-1613.

[74] Ma X, Xia F. Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization[C]//Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2014: 1337-1348.

[75] Wang Y, Che W, Guo J, et al. Cross-lingual bert transformation for zero-shot dependency parsing [C]//Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2019: 5721-5727.

[76] Kong L, Schneider N, Swayamdipta S, et al. A dependency parser for tweets[C]//Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014: 1001-1012.

[77] Zhang M, Zhang Y, Fu G. Cross-lingual dependency parsing using code-mixed treebank[C]// Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2019: 997-1006.

[78] Li Z, Cai J, He S, et al. Seq2seq dependency parsing[C]//Proceedings of the 27th International Conference on Computational Linguistics. 2018: 3203-3214.

[79] Zhang Y, Li Z, Zhang M. Efficient second-order treecrf for neural dependency parsing[C]// Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020: 3295-3305.

索引