# Chapter 4
# Deep Learning in Lexical Analysis and Parsing


**Wanxiang Che and Yue Zhang**

**Abstract** Lexical analysis and parsing tasks model the deeper properties of the words and their relationships to each other. The commonly used techniques involve word segmentation, part-of-speech tagging and parsing. A typical characteristic of such tasks is that the outputs are structured. Two types of methods are usually used to solve these structured prediction tasks: graph-based methods and transition-based methods. Graph-based methods differentiate output structures based on their characteristics directly, while transition-based methods transform output construction processes into state transition processes, differentiating sequences of transition actions. Neural network models have been successfully used for both graph-based and transition-based structured prediction. In this chapter, we give a review of applying deep learning in lexical analysis and parsing, and compare with traditional statistical methods.

## 4.1 Background

The properties of a word include its syntactic word categories (also known as **part of speech**, POS), **morphologies**, and so on (Manning and Schütze 1999). Obtaining these information is also known as **lexical analysis**. For languages like Chinese, Japanese, and Korean that do not separate words with whitespace, lexical analysis also includes the task of **word segmentation**, i.e., splitting a sequence of characters into words. Even in English, although whitespace is a strong clue for word boundaries, it is neither necessary nor sufficient. For example, in some situations, we might wish to treat *New York* as a single word. This is regarded as a **named entity recognition** (NER) problem (Shaalan 2014). On the other hand, punctuation marks are always adjacent to words. We also need to judge whether to segment them or not.

W. Che (✉)
Harbin Institute of Technology, Harbin, Heilongjiang, China
e-mail: wxche@ir.hit.edu.cn

Y. Zhang
Singapore University of Technology and Design, Singapore, Singapore
e-mail: yue_zhang@sutd.edu.sg



L. Deng and Y. Liu (eds.), *Deep Learning in Natural Language Processing*, https://doi.org/10.1007/978-981-10-5209-5_4

For languages like English, this is often called **tokenization** which is more a matter of convention than a serious research problem.

Once we know some properties of words, we may be interested in the relationships between them. The **parsing** task is to find and label words (or sequences of words) that are related to each other compositionally or recursively (Jurafsky and Martin 2009). There are two commonly used parses: **phrase-structure** (or **constituency**) parsing and **dependency** parsing.

All of these tasks can be regarded as structured prediction problems which is a term for supervised machine learning, i.e., the outputs are structured and influenced each other. Traditionally, huge amounts of human-designed handcrafted features are fed into a linear classifier to predict a score for each decision unit and then combine all of these scores together with satisfying some structured constraints. With the help of deep learning, we can employ an end-to-end learning paradigm which does not need costly feature engineering. The technology can even find more implicit features which are difficult to be designed by humans. Nowadays, deep learning has dominated these natural language processing tasks.

However, because of the pervasive problem of **ambiguity**, none of these tasks is trivial to predict. Some ambiguities may not even be noticed by humans.

This chapter is organized as follows. We will first select some typical tasks as examples to see where these ambiguities come from (Sect. 4.2). Then, we will review two typical structured prediction methods (Sect. 4.3): graph-based method (Sect. 4.3.1) and transition-based method (Sect. 4.3.2). Sections 4.4 and 4.5 are devoted to neural networks for graph-based and transition-based methods respectively. The chapter closes with a conclusion (Sect. 4.6).

## 4.2 Typical Lexical Analysis and Parsing Tasks

A natural language processing (lexical analysis and parsing here) pipeline usually includes three stages: word segmentation, POS tagging, and syntactic parsing.

### *4.2.1 Word Segmentation*

As mentioned above, some languages, such as Chinese, are written in contiguous characters (Wong et al. 2009). Even though there are dictionaries to list all words, we cannot simply match words in a sequence of characters because ambiguity exists. For example, a Chinese sentence

- yanshouyibashoujiguanle (Shouyi Yan turned off the mobile phone)

can match words

- yanshouyi (Shouyi Yan)/ba (NA)/shouji (mobile phone)/guan (turn off)/le (NA)

which is a correct word segmentation result. However,

- yanshou (strictly)/yibashou (leader)/jiguan (office)/le (NA)
- yanshou (strictly)/yiba (handful)/shouji (mobile phone)/guan (turn off)/le (NA)
- yanshouyi (Shouyi Yan)/bashou (handle)/jiguan (office)/le (NA)

are also valid matching results but the sentence becomes meaningless with the segmentations. Obviously, the word matching method cannot distinguish which segmentation result is better than others. We need some kinds of scoring functions to assess the results.

### 4.2.2 POS Tagging

POS tagging is one of the most basic tasks in NLP, and it is useful in many natural language applications.[1] For example, the word *loves* can be a noun (plural of love) or a verb (third person present form of love). We can determine that *loves* is a verb but not noun in the following sentence.

- *The boy loves a girl*

The determent can be made independently without knowing the tags assigned to other words. Better POS taggers, however, take the word tags into consideration, because the tags of nearby a word can help to disambiguate its POS tag. In the example above, the following determiner *a* can help to indicate that *loves* is a verb.

Therefore, the complete POS tagging output of above sentence is a tag sequence, for example.

- *D N V D N*

(here we use *D* for a determiner, *N* for noun, and *V* for verb). The tag sequence has the same length as the input sentence, and therefore specifies a single tag for each word in the sentence (in this example *D* for *the*, *N* for *boy*, *V* for *loves*, and so on). Usually, the output of POS tagging can be written into a tagged sentence, where each word in the sentence is annotated with its corresponding POS tag, i.e., *The/D boy/N loves/V a/D girl/N*.

Like word segmentation, some sentences may have different meanings, if they are assigned with different POS tag sequences. For instance, two interpretations are possible for the sentence "Teacher strikes idle kids", depending on the POS assignments of the words in the sentence,

### 4.2.3 Syntactic Parsing

Phrase structures are very often constrained to correspond to the derivations of context-free grammars (CFGs) (Carnie 2012). In such a derivation, each phrase that

[1] https://en.wikipedia.org/wiki/Part-of-speech_tagging.

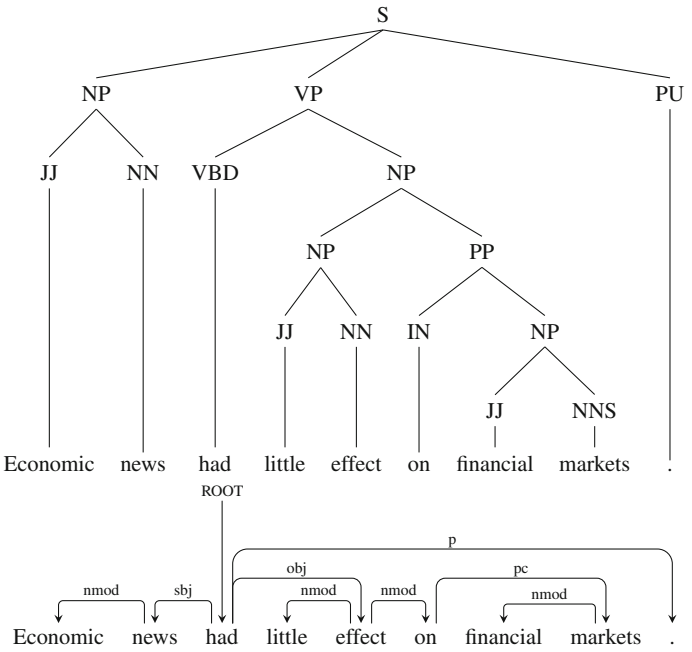



**Fig. 4.1** Constituent tree (above) versus dependency tree (below)

is longer than one word is made of a sequence of non-overlapping "child" phrases or words, such that the children "cover" the yield of the parent phrase.

Another syntactic structure widely used in NLP is **dependency parse** tree (Kbler et al. 2009). A dependency parse is a directed tree where the words are vertices. Edges (also know as arcs) correspond to syntactic relations between two words and may be labeled with relation types. One extra pseudo word is the root of the tree, and each other word has a single in-bound edge from its syntactic head. For example, Fig. 4.1 shows constituent and dependency trees for the sentence, *Economic news had little effect on financial markets*.[2]

Dependency parsing can be classified into two categories: **projective** parsing (if there are no crossing arcs in the trees) and **non-projective** parsing (if there are crossing arcs in the trees). English and Chinese structures are predominantly projective.

A primary reason for using dependency structures instead of more informative constituent structures is that they are usually easier to be understood. For example, in Fig. 4.1, it is hard to point out that the *news* is the subject of *had* from the constituent structure, while the dependency structure can clearly indicate this relation between the two words. In addition, dependency structures are more amenable to annotators who have good knowledge of the target domain but lack deep linguistic knowledge.

[2]From Joakim Nivre's tutorial at COLING-ACL, Sydney 2006.

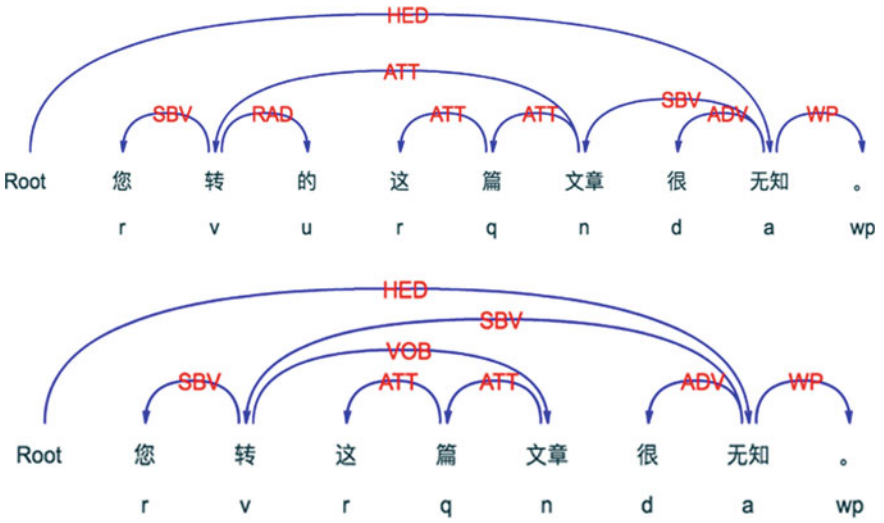



**Fig. 4.2** The processing results of LTP

Syntactic parsing can provide useful structural information for applications. For example, the following two Chinese sentences "nin zhuan zhe pian wen zhang hen wu zhi" (*You are ignorant to retweet the article*) and "nin zhuan **de** zhe pian wen zhang hen wu zhi" (*The article you retweeted is ignorant*) have completely different meanings, although the second one only has an additional word "de", which is a possessive particle in Chinese. The main difference between the two sentences is that they have different subjects.

Dependency parsing can tell us this syntactic information directly. One example is LTP (Language Technology Platform)[3] developed by HIT (Harbin Institute of Technology), which provides a Chinese NLP preprocessing pipeline that includes Chinese word segmentation, POS tagging, dependency parsing, and so on. The LTP's processing results of the above two sentences are shown in Fig. 4.2. From these results, we can easily know that subjects of the two sentences are *wenzhang* (article) and *zhuan* (retweet) respectively. Many applications, such as sentiment analysis, can take advantage of these syntactic information. Although the sentiment of the two sentences can be easily determined by the polarity word *wuzhi* (ignorant), it is difficult to identify its targets or aspects if we do not know their syntactic structures.

[3] http://www.ltp.ai.

### *4.2.4 Structured Predication*

These different natural language processing tasks can fall into three types of **structured prediction** problems (Smith 2011):

- Sequence segmentation
- Sequence labeling
- Parsing.

#### 4.2.4.1 Sequence Segmentation

Sequence segmentation is the problem of breaking a sequence into contiguous parts called segments. More formally, if the input is $\mathbf{x} = x_1, \ldots, x_n$, then a segmentation can be written as $\langle x_1, \ldots, x_{y_1} \rangle, \langle x_{y_1+1}, \ldots, x_{y_2} \rangle, \ldots, \langle x_{y_m+1}, \ldots, x_n \rangle$, and the output is $\mathbf{y} = y_1, \ldots, y_m$ which corresponds to the segmental points, where $\forall i \in \{1, \ldots, m\}, 1 \leq y_i \leq n$.

Besides word segmentation, there exist other sequence segmentation problems such as **sentence segmentation** (breaking a piece of string into sentences which is an important postprocessing stage for speech transcription) and **chunking** (also known as **shallow parsing** to find important phrases from sentences, such as noun phrases).

#### 4.2.4.2 Sequence Labeling

Sequence labeling (also named as **tagging**) is the problem of assigning a corresponding label or tag for each item of an input sequence. More formally, if the input sequence is $\mathbf{x} = x_1, \ldots, x_n$, then the output tag sequence is $\mathbf{y} = y_1, \ldots, y_n$, where each input $x_i$ has a single output tag $y_i$.

POS tagging is perhaps the most classical, and most famous, example of this type of problem, where $x_i$ is a word in a sentence, and $y_i$ is its corresponding POS tag.

Besides POS tagging, many NLP tasks can be mapped to sequence labeling problems such as **named entity recognition** (locating and classifying named entities in text into predefined categories such as the names of persons, locations, and organizations). For this problem, the input is again a sentence. The output is the sentence with entity boundaries tags. We assume there are three possible entity types: PER, LOC, and ORG. Then for input sentence

- *Rachel Holt, Uber's regional general manager for U.S. and Canada, said in a statement provided to CNNTech.*[4]

the output of named entity recognition can be

[4] http://money.cnn.com/2017/04/14/technology/uber-financials/.

- *Rachel/B-PER Holt/I-PER,/O Uber/B-ORG's/O regional/O general/O manager/O for/O U.S./B-LOC and/O Canada/B-LOC , /O said/O in/O a/O statement/O provided/O to/O CNNTech/B-ORG. /O*

where each word in the sentence is either tagged as being the beginning of a particular entity type, B-XXX (e.g., the tag B-PER corresponds to words that are the first word in a person), as being the inside of a particular entity type, I-XXX (e.g., the tag I-PER corresponds to words that are part of a person name, but are not the first word), or otherwise (the tag O, i.e., not an entity).

Once this mapping has been performed on training examples, we can train a tagging model on these training examples. Given a new test sentence we can then predict a sequence of tags by the model, and then it is straightforward to identify the entities from the tagged sequence.

The above sequence segmentation problems can even be transformed into sequence tagging problems by designing proper tag sets. For Chinese word segmentation as an example, each character in a sentence can be annotated with either tag B (beginning of a word) or I (inside a word) (Xue 2003).

The purpose of transforming a sequence segmentation problem into a sequence labeling problem is that the latter is much easier to be modeled and decoded. For example, we will introduce a traditional popular sequence labeling model, conditional random field (CRF), in Sect. 4.3.1.1.

#### 4.2.4.3 Parsing Algorithms

In general, we use **parsing** to denote all kinds of algorithms converting sentences to syntactic structures. As mentioned in Sect. 4.2.3, there are two popular syntactic paring representations, phrase-structure (or named as constituency) parsing and dependency parsing.

For constituent parsing, in general, a **grammar** is used to derive syntactic structures. In brief, a grammar consists of a set of rules, each corresponding to a derivation step that is possible to take under particular conditions. Context-free grammars (CFGs) are most frequently used in constituency parsing (Booth 1969). The parsing is viewed as choosing the maximum-scoring derivation from a grammar.

**Graph-based** and **transition-based** methods are currently two dominant dependency parsing algorithms (Kbler et al. 2009). Graph-based dependency parsing can be formalized as finding the maximum spanning tree (MST) from a directed graph with vertices (words) and edges (dependency arcs between two words). A transition-based dependency parsing algorithm can be formalized as a transition system consisting of a set of states and a set of transition actions. The transition system begins in start state and transitions are iteratively followed until a terminal state is reached. The common critical problem for graph-based and transition-based dependency parsing is how to calculate the score of a dependency arc or a transition action. We will introduce the two methods in detail at Sects. 4.3.1.2 and 4.3.2.1 respectively.

## 4.3 Structured Prediction Methods

In this section, we will introduce two types of state-of-the-art structured prediction methods: **graph-based** and **transition-based** respectively. Most deep learning algorithms for structured prediction problems are also derived from these methods.

### *4.3.1 Graph-Based Methods*

The graph-based structured prediction methods differentiate output structures based on their characteristics directly. The **conditional random fields** (CRFs) are typical graph-based methods, which aim to maximize the **probability** of the correct output structure. The graph-based methods can also be applied to dependency parsing, where the aim changes to maximize the **score** of the correct output structure. Next, we will introduce these two methods in detail.

#### 4.3.1.1 Conditional Random Fields

Conditional random fields, strictly speaking, are a variant of undirected graphical models (also called Markov random fields or Markov networks) in which some random variables are observed and others are modeled probabilistically. CRFs were introduced by Lafferty et al. (2001) for sequence labeling. They are also known as linear-chain CRFs. It has been the de facto method for sequence labeling problems before deep learning.

The CRFs define the distribution over label sequences $\mathbf{y} = y_1, \ldots, y_n$, given an observed sequence $\mathbf{x} = x_1, \ldots, x_n$, by a special case of log-linear models:

$$p(\mathbf{y}|\mathbf{x}) = \frac{\exp \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y_{i-1}, y_i, i)}{\sum_{\mathbf{y}' \in \mathscr{Y}(\mathbf{x})} \exp \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{f}(\mathbf{x}, y'_{i-1}, y'_i, i)}, \tag{4.1}$$

where $\mathscr{Y}(\mathbf{x})$ is a set of all possible label sequences; $\mathbf{f}(\mathbf{x}, y_{i-1}, y_i, i)$ is the feature function that extracts a feature vector from position $i$ of sequence $\mathbf{x}$, which can include the labels at the current position $y_i$ and at the previous position $y_{i-1}$.

The attraction of the CRFs is that it permits the inclusion of any (local) features. For example, in POS tagging, the features can be word-tag pairs, pairs of adjacent tags, spelling features, such as whether the word starts with a capital letter or contains a digit, and prefix or suffix features. These features may be dependent, but the CRFs permit over-lapping features and learn to balance their effect on prediction against the other features. The reason why we name these features as local features is that we assume the label $y_i$ only depends on $y_{i-1}$, but longer history. This is also named as (first order) Markov assumption.

The general Viterbi algorithm, a kind of dynamic programming algorithm, can be applied for decoding with CRFs. Then the first-order gradient-based (such as gradient descent) or second-order (such as L-BFGS) optimization methods can be used to learn proper parameters to maximize conditional probability in Eq. (4.1).

Besides sequence labeling problems, CRFs have been generalized in many ways for other structured prediction problems. For example, Sarawagi and Cohen (2004) proposed the semi-CRF model for sequence segmentation problems. In semi-CRF, the conditional probability of a semi-Markov chain on the input sequence is explicitly modeled, whose each state corresponds to a subsequence of input units. However, to achieve good segmentation performance, conventional semi-CRF models require carefully handcrafted features to represent the segment. Generally, these feature functions fall into two types: (1) the CRF style features which represent input unit-level information such as the specific words at a particular position; (2) the semi-CRF style features which represent segment-level information such as the length of the segment.

Hall et al. (2014) proposed a CRF-based constituency parsing model, where the features factor over anchored rules of a small backbone grammar, such as basic span features (first word, last word, and length of the span), span context features (the words immediately preceding or following the span), split point features (words at the split point inside the span), and span shape features (for each word in the span, indicating whether that word begins with a capital letter, lowercase letter, digit, or punctuation mark). The CKY algorithm[5] can be used to find the tree with maximum probabilities given learned parameters.

#### 4.3.1.2 Graph-Based Dependency Parsing

Consider a directed graph with vertices $V$ and edges $E$. Let $s(u, v)$ denote the score of an edge from vertex $u$ to vertex $v$. A directed spanning tree is a subset of edges $E' \subset E$ such that all vertices have exactly one incoming arc in $E$, except the root vertex (which has none), and such that $E'$ contains no cycles. Let $\mathscr{T}(E)$ denote the set of all possible directed spanning trees for $E$. The total score of a spanning tree $E'$ is the sum of the scores of edges in $E'$. The maximum spanning tree (MST) is defined by

$$\max_{E' \in \mathscr{T}(E)} \sum_{s(u,v) \in E'} s(u, v). \tag{4.2}$$

Then the (unlabeled) dependency parsing decoding problem can be reduced to the maximum spanning tree problem if we view words in a sentence as vertices and edges as dependency arcs, where $u$ is often named as a head (or parent) and $v$ as a modifier (or child).

It is straightforward to extend this approach to labeled dependency parsing, if we have multiple edges from $u$ to $v$, one associated with each label. The same algorithm

[5] https://en.wikipedia.org/wiki/CYK_algorithm.

applies. The most widely used decoding algorithm for the MST problem is the Eisner algorithm (Eisner 1996) for projective parsing and Chu-Liu-Edmonds algorithm (Chu and Liu 1965; Edmonds 1967) for non-projective parsing.

Here, we introduce the basic graph-based method, which is called the first-order model. The first-order graph-based model makes a strong independence assumption: the arcs in a tree are independent from each other. In other words, the score of an arc is not affected by other arcs. This method is also called the **arc-factorization** method.

So, the critical problem is, given an input sentence, how to determine the score $s(u, v)$ of each candidate arc. Traditionally, discriminative models were used which represent an arc with a feature vector extracted with feature function $\mathbf{f}(u, v)$. Then, the score of the arc is the dot product of a feature weight vector $\mathbf{w}$ and $\mathbf{f}$, i.e., $s(u, v) = \mathbf{w} \cdot \mathbf{f}(u, v)$.

Then how to define $\mathbf{f}(u, v)$ and how to learn optimizing parameters $\mathbf{w}$?

**Feature Definition**

The choice of features is central to the performance of a dependency parsing model. For each possible arc, the following features are readily considered:

- for each word involved, the surface form, its lemma, its POS, and any shape, spelling, or morphological features;
- words involved include the head, the modifier, context words on either side of the head and modifier, words in between the head and modifier;
- the length of the arc (number of words between the head and modifier), its direction, and (if the parse is to be labeled) the syntactic relation type.

Besides these atomic features, all kinds of combination features and back-off features can also be extracted.

**Parameter Learning**

Online structured learning algorithms such as the averaged perceptron (AP) (Freund and Schapire 1999; Collins 2002), online passive-aggressive algorithms (PA) (Crammer et al. 2006), or margin infused relaxed algorithm (MIRA) (Crammer and Singer 2003; McDonald 2006) are commonly used for learning parameters $\mathbf{w}$ in graph-based dependency parsing.

### *4.3.2 Transition-Based Methods*

Different from graph-based methods, which differentiate structural outputs directly, a transition-based method can be formalized as a transition system consisting of a set of states $S$ (possibly infinite), including a start state $s_0 \in S$ and a set of terminal states $S_t \in S$, and a set of transition actions $T$ (Nivre 2008). The transition system begins in $s_0$ and transitions are iteratively followed until a terminal state is reached. Figure 4.3 shows a simple finite state transducer, where the start state is $s_0$, and the terminal states include $s_6$, $s_7$, $s_8$, $s_{14}$, $s_{15}$, $s_{16}$, $s_{17}$ and $s_{18}$. The goal of a transition-based structured prediction model is to differentiate sequences of transition actions

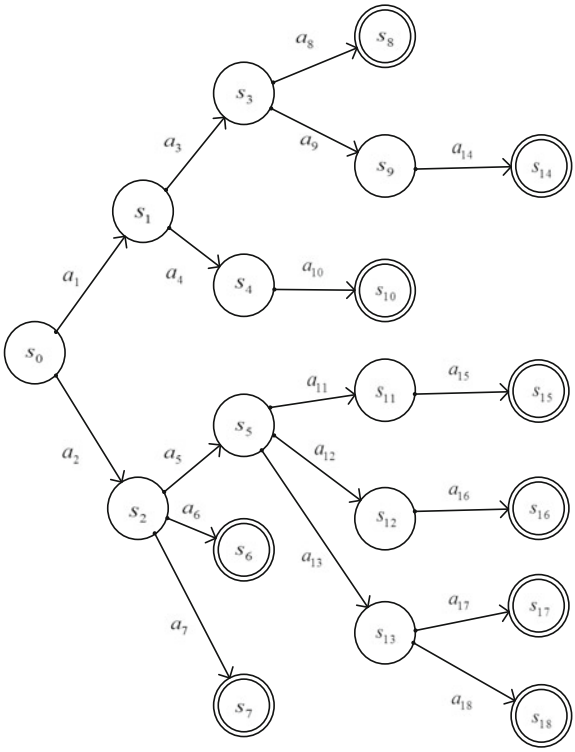



**Fig. 4.3** Transition-based method for structured prediction

that lead to the terminal states, so that those that correspond to the correct output state are scored higher.

#### 4.3.2.1 Transition-Based Dependency Parsing

The arc-standard transition system (Nivre 2008) is widely used for projective dependency parsing. In this system, each state corresponds to a stack $\sigma$ containing partially built subtrees, a buffer $\beta$ of as-yet-unprocessed words, and a set of dependency arcs $A$. The transition actions are shown as deductive rules in Fig. 4.4. A transition sequence for the sentence

- $Economic_1$ $news_2$ $had_3$ $little_4$ $effect_5$ $on_6$ $financial_7$ $markets_{8.9}$

in Fig. 4.1 generated by the arc-standard algorithm is presented in Table 4.1.

In a greedy parser, the decision about what to do in state $s \in S$ is made by a classifier. Training the classifier is accomplished by considering gold-standard trees in the training section of a treebank, from which we can derive canonical gold-standard sequences (oracle sequences) of transition state and action pairs.

Information that can be obtained from a state $s = \langle \sigma, \beta, A \rangle$ includes:

**Fig. 4.4** Transition actions in the deduction system (Nivre 2008)

Start state $([\text{ROOT}], [0 \ldots, n], \emptyset)$

$\textsc{LeftArc}_l$ ($\text{LA}_l$) $\dfrac{([\sigma \mid s_1, s_0], \beta, A)}{([\sigma \mid s_0], \beta, A \cup \{s_1 \overset{l}{\leftarrow} s_0\})}$

$\textsc{RightArc}_l$ ($\text{RA}_l$) $\dfrac{([\sigma \mid s_1, s_0], \beta, A)}{([\sigma \mid s_1], \beta, A \cup \{s_1 \overset{l}{\rightarrow} s_0\})}$

$\textsc{Shift}$ (SH) $\dfrac{(\sigma, [b \mid \beta], A)}{([\sigma \mid b], \beta, A)}$

Terminal state $([\text{ROOT}], [\,], A)$

**Table 4.1** Transitions by the arc-standard algorithm

| State | Action | $\sigma$ | $\beta$ | $A$ |
|---|---|---|---|---|
| 0 | Initialization | [0] | [1, . . . , 9] | $\emptyset$ |
| 1 | SH | [0, 1] | [2, . . . , 9] | |
| 2 | SH | [0, 1, 2] | [3, . . . , 9] | |
| 3 | $\text{LA}_{nmod}$ | [0, 2] | [3, . . . , 9] | $A \cup \{1 \overset{nmod}{\longleftarrow} 2\}$ |
| 4 | SH | [0, 2, 3] | [4, . . . , 9] | |
| 5 | $\text{LA}_{sbj}$ | [0, 3] | [4, . . . , 9] | $A \cup \{2 \overset{sbj}{\longleftarrow} 3\}$ |
| 6 | SH | [0, 3, 4] | [5, . . . , 9] | |
| 7 | SH | [0, 3, 4, 5] | [6, . . . , 9] | |
| 8 | $\text{LA}_{nmod}$ | [0, 3, 5] | [6, . . . , 9] | $A \cup \{4 \overset{nmod}{\longleftarrow} 5\}$ |
| 9 | SH | [0, 3, 5, 6] | [7, . . . , 9] | |
| 10 | SH | [0, 3, 5, 6, 7] | [8, 9] | |
| 11 | SH | [0, 3, 5, 6, 7, 8] | [9] | |
| 12 | $\text{LA}_{nmod}$ | [0, 3, 5, 6, 8] | [9] | $A \cup \{7 \overset{nmod}{\longleftarrow} 8\}$ |
| 13 | $\text{RA}_{pc}$ | [0, 3, 5, 6] | [9] | $A \cup \{6 \overset{pc}{\rightarrow} 8\}$ |
| 14 | $\text{RA}_{nmod}$ | [0, 3, 5] | [9] | $A \cup \{5 \overset{nmod}{\longrightarrow} 6\}$ |
| 15 | $\text{RA}_{obj}$ | [0, 3] | [9] | $A \cup \{3 \overset{obj}{\longrightarrow} 5\}$ |
| 16 | SH | [0, 3, 9] | [ ] | |
| 17 | $\text{RA}_{p}$ | [0, 3] | [ ] | $A \cup \{3 \overset{p}{\rightarrow} 9\}$ |
| 18 | $\text{RA}_{root}$ | [0] | [ ] | $A \cup \{0 \overset{root}{\longrightarrow} 3\}$ |

- all the words and their corresponding POS tags;
- the head of a word and its label from partial parsed dependency arcs $A$;
- the position of a word on the stack $\sigma$ and buffer $\beta$.

For example, Zhang and Nivre (2011) proposed 72 feature templates which include 26 baseline and 46 new feature templates. The baseline features mainly describe the words and POS tags at top of stack and buffer and their combination. The new features are: direction and distance between a pair of head and modifier; the number of modifiers to a given head; higher order partial parsed dependency arcs; the set of

unique dependency labels from the modifiers of the top word in the stack and buffer. Finally, these new features boost about 1.5% UAS (unlabeled attachment score).

We usually use the term "feature engineering" to describe the need of the amount of linguistic expertise that has gone into designing features for various linguistic structured prediction tasks.

NLP researchers tend to adopt the strategy of incorporating as many features as they can think of into learning and allowing the parameter estimation method to determine which features are helpful and which should be ignored. Perhaps because of the heavy-tailed nature of linguistic phenomena and the continued growth in computational power available to researchers, the current consensus seems to be that more features are always welcome in an NLP model, especially in frameworks like log-linear models that can incorporate them.

To reduce error propagation in greedy transition-based algorithms, beam search decoding with global normalization is usually applied and large margin training with early update (Collins and Roark 2004) is used for learning from inexact search.

#### 4.3.2.2 Transition-Based Sequence Labeling and Segmentation

Besides dependency parsing, the transition-based framework can be applied to most structured prediction tasks in NLP, to which a mapping can be found between structured outputs and state transition sequences. Take sequence labeling for example. The output can be constructed by incrementally assigning labels to each input from left to right. In this setting, the state is a pair $(\sigma, \beta)$, where $\sigma$ represents a partially labeled sequence and $\beta$ represents a queue of unlabeled words. With the start state being ([ ], *input*) and the terminal states being (*output*, [ ]), each action advances a state by assigning a particular label on the front of $\beta$.

Sequence segmentation, such as word segmentation is a second example, for which a transition system can process input characters incrementally from left to right. A state takes the form $(\sigma, \beta)$, where $\sigma$ is a partially segmented word sequence and $\beta$ is a queue of next incoming characters. In the start state, $\sigma$ is empty and $\beta$ consists of the full input sentence. In any terminal state, $\sigma$ contains a full segmented sequence and $\beta$ is empty. Each transition action advances the current state by processing the next incoming character, either separating (SEP) it at the beginning of a new word or appending (APP) it to the end of the last word in the partially segmented sequence. A gold-standard state transition sequence for the sentence "wo xi huan du shu (I like reading)" is shown in Table 4.2.

#### 4.3.2.3 Advantages of Transition-Based Methods

Transition-based methods do not reduce structural ambiguity—the search space does not shrink in size for a given structured prediction task when the solution changes from a graph-based model to a transition-based model. The only difference is that structural ambiguities are transformed into ambiguities between different transition

**Table 4.2** Gold state transition sequence for word segmentation

| State | $\sigma$ | $\beta$ | Next action |
|---|---|---|---|
| 0 | [] | [wo, xi, huan, du, shu] | SEP |
| 1 | [wo (I)] | [xi, huan, du, shu] | SEP |
| 2 | [wo (I), xi] | [huan, du, shu] | APP |
| 3 | [wo (I), xihuan (like)] | [du, shu] | SEP |
| 4 | [wo (I), xihuan (like), dushu (reading)] | [] | APP |

actions at each state. A question that naturally arises is why transition-based methods have attracted significant research attention.

The main answer lies in the features that can be utilized by transition-based models, or the information that is made available for ambiguity resolution. Traditional graph-based methods are typically constrained by efficiency of exact inference, which limits the range of features to use. For example, to train CRF models (Lafferty et al. 2001), it is necessary to efficiently estimate the marginal probabilities of small cliques, the sizes of which are decided by feature ranges. To allow efficient training, CRF models assume low-order Markov properties of their features. As a second example, CKY parsing (Collins 1997) requires that the features are constrained to local grammar rules, so that a tolerable polynomial dynamic program can be used to find the highest scored parse tree among an exponential of search candidates.

In contrast, early work on transition-based methods employ greedy local models (Yamada and Matsumoto 2003; Sagae and Lavie 2005; Nivre 2003), and are typically regarded as a very fast alternative to graph-based systems, running in linear time with regard to the input size. Thanks to the use of arbitrary nonlocal features, their accuracies are not far behind the state-of-the-art models. Since global training has been utilized for training sequences of actions (Zhang and Clark 2011b), fast and accurate transition-based models were made, which gives the state-of-the-art accuracies for tasks such as CCG parsing (Zhang and Clark 2011a; Xu et al. 2014), natural language synthesis (Liu et al. 2015; Liu and Zhang 2015; Puduppully et al. 2016), dependency parsing (Zhang and Clark 2008b; Zhang and Nivre 2011; Choi and Palmer 2011) and constituent parsing (Zhang and Clark 2009; Zhu et al. 2013). Take constituent parsing for example, ZPar (Zhu et al. 2013) gives competitive accuracies to Berkeley parser (Petrov et al. 2006), yet runs 15 times faster.

The efficiency advantage of transition-based systems further allows joint structured problems with highly complex search spaces to be exploited. Examples include joint word segmentation and POS tagging (Zhang and Clark 2010), joint segmentation, POS tagging and chunking (Lyu et al. 2016), joint POS tagging and parsing (Bohnet and Nivre 2012; Wang and Xue 2014), joint word segmentation, POS tagging and parsing (Hatori et al. 2012; Zhang et al. 2013, 2014), joint segmentation and normalization for microblog (Qian et al. 2015), joint morphological generation and text linearization (Song et al. 2014), and joint entity and relation extraction (Li and Ji 2014; Li et al. 2016).

## 4.4 Neural Graph-Based Methods

### *4.4.1 Neural Conditional Random Fields*

Collobert and Weston (2008) was the first work to utilize deep learning for sequence labeling problems. This was almost the earliest work successfully using deep learning for addressing natural language processing tasks. They not only embedded words into a $d$-dimensional vector, but also embedded some additional features. Then words and corresponding features in a window were fed into an MLP (multiple layer perceptron) to predict a tag. Word-level log-likelihood, each word in a sentence being considered independently, was used as the training criterion. As mentioned above, there is often a correlation between the tag of a word in a sentence and its neighboring tags. Therefore, in their updated work (Collobert et al. 2011), tag transition scores were added in their sentence-level log-likelihood model. In fact, the model is the same with the CRF models except that the conventional CRF models use a linear model instead of a nonlinear neural network.

While, limited by Markov assumption, the CRF models can only make use of local features. It leads to the long-term dependency between tags cannot be modeled, which sometimes is important in many natural language processing tasks. Theoretically, recurrent neural networks (RNNs) can model arbitrarily sized sequence into fixed-size vectors without resorting to the Markov assumption. Then the output vector is used for further prediction. For example, it can be used to predict the conditional probability of a POS tag given an entire previous word sequence.

In more detail, RNNs are defined recursively, by means of a function taking as input a previous state vector and an input vector and returning a new state vector. So, intuitively, RNNs can be thought of as very deep feedforward networks, with shared parameters across different layers. The gradients then include repeated multiplication of the weight matrix, making it very likely for the values to vanish or explode. The gradient exploding problem has a simple but very effective solution: clipping the gradients if their norm exceeds a given threshold. While the gradient vanishing problem is much more complicated. The gating mechanism, such as the long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) and the gated recurrent unit (GRU) (Cho et al. 2014), can solve it more or less.

A natural extension of RNN is a bidirectional RNN (Graves 2008) (BiRNN, such as BiLSTM and BiGRU). In sequence labeling problems, predicting a tag not only depends on the previous words, but also depends on the successive words, which cannot be seen in a standard RNN. Therefore, BiRNN use two RNNs (forward and backward RNN) to represent the word sequences before and behind the current word. Then, the forward and backward states of the current word are concatenated together as input to predict the probability of a tag.

In addition, RNNs can be stacked in layers, where the inputs of an RNN are the outputs of the RNN below it. Such layered architectures are often called deep RNNs. Deep RNNs have shown power in many problems, such as semantic role labeling

(SRL) with sequence labeling method (Zhou and Xu 2015, https://www.aclweb.org/anthology/P/P17/P17-1044.bib).

Although RNNs have been successfully applied in many sequence labeling problems, they do not explicitly model the dependency between output tags like CRFs. Therefore, the transition score matrix between any tags can also be added to form a sentence-level log-likelihood model usually named as RNN-CRF model where RNN can also be LSTM, BiLSTM, GRU, BiGRU, and so on.

Like conventional CRFs, the neural CRFs can also be extend to handle the sequence segmentation problems. For example, Liu et al. (2016) proposed a neural semi-CRF, which used a segmental recurrent neural network (SRNN) to represent a segment by composing input units with an RNN. At the same time, additional segment-level representation using segment embedding is also regarded as inputs which encodes the entire segment explicitly. Finally, they achieve the state-of-the-art Chinese word segmentation performance.

Durrett and Klein (2015) extended their CRF phrase-structure parsing (Hall et al. 2014) to neural one. In their neural CRF parsing, instead of linear potential functions based on sparse features, they use nonlinear potentials computed via a feedforward neural network. The other components, such as decoding, are unchanged from the conventional CRF parsing. Finally, they achieve the state-of-the-art phrase-structure parsing performance.

### *4.4.2 Neural Graph-Based Dependency Parsing*

Conventional graph-based models rely heavily on an enormous number of handcrafted features, which brings about serious problems. First, a mass of features could put the models in the risk of overfitting, especially in the combinational features capturing interactions between head and modifier could easily explode the feature space. In addition, feature design requires domain expertise, which means useful features are likely to be neglected due to a lack of domain knowledge.

To ease the problem of feature engineering, some recent works propose some general and effective neural network models for graph-based dependency parsing.

#### 4.4.2.1 Multiple Layer Perceptron

Pei et al. (2015) used an MLP (multiple layer perceptron) model to score an edge. Instead of using millions of features as in conventional models, they only use atomic features such as word unigrams and POS tag unigrams, which are less likely to be sparse. Then these atomic features are transformed into their corresponding distributed representations (feature embeddings or feature vector) and push into MLP. Feature combinations are automatically learned with novel *tanh*−*cub* activation function at the hidden layer, thus alleviating the heavy burden of feature engineering in conventional graph-based models.

The distributed representation can discover useful new features that have never been used in conventional parsers. For instance, context information of the dependency edge ($h$, $m$), such as words between $h$ and $m$, has been widely believed to be useful in graph-based models. However, in conventional methods, the complete context cannot be used as features directly because of the data sparseness problem. Therefore, they are usually backed off to low-order representation such as bigrams and trigrams.

Pei et al. (2015) proposed to use distributed representation of the context. They simply average all word embeddings in a context to represent it. The method can not only effectively use every word in the context, but also can capture semantic information behind context, because similar words have similar embeddings.

At last, max-margin criterion is used to train the model. The training object is that the highest scoring tree is the correct one and its score will be larger up to a margin to other possible tree. The structured margin loss is defined as the number of word with an incorrect head and edge label in the predicted tree.

#### 4.4.2.2 Convolutional Neural Networks

Pei et al. (2015) simply average embeddings in context to represent them, which ignore the word position information and cannot assign different weights for different words or phrases. Zhang et al. (2016b) introduce convolutional neural networks (CNN) to compute the representation of a sentence. Then use the representation to help scoring an edge. While the pooling regimes make CNN invariant to shifting, that is CNN ignore the position of words which is very important for dependency parsing. In order to overcome the problem, Zhang et al. (2016b) input the relative positions between a word and a head or modifier to CNN. Another difference from Pei et al. (2015) is that they utilize the probabilistic treatment for training: calculating the gradients according to probabilistic criteria. The probabilistic criteria can be viewed as a soft version of the max-margin criteria, and all the possible factors are considered when calculating gradients for the probabilistic way, while only wrongly predicted factors have nonzero subgradients for max-margin training.

#### 4.4.2.3 Recurrent Neural Networks

Theoretically, recurrent neural networks (RNN) can model sequences with arbitrary length which is sensitive to the relative positions of words in the sequence. As an improvement of conventional RNN, LSTM can better represent a sequence. The BiLSTM (bidirectional LSTM) particularly excels at representing words in the sequence together with their contexts, capturing the word and an "infinite" window around it. Therefore, Kiperwasser and Goldberg (2016) represent each word by its BiLSTM hidden layer output, and use the concatenation of the head and modifier words' representation as the features, which is then passed to a nonlinear scoring function (MLP). To speed up parsing, Kiperwasser and Goldberg (2016) proposed a two-stage strategy.

First, they predict the unlabeled structure using the method given above, and then predict the label of each resulting edge. The labeling of an edge is performed using the same feature representation as above fed into a different MLP predictor. Finally, the max-margin criterion is used to train the model, i.e., let the correct tree is scored above incorrect ones with a margin.

Wang and Chang (2016) also use BiLSTM to represent the head and modifier words. Moreover, they introduce some additional features, such as distance between the two words and context like Pei et al. (2015). Different from Pei et al. (2015), they utilize LSTM-Minus to represent a context, in which distributed representation of a context is learned by using subtraction between LSTM hidden vectors. The similar idea was also been used by Cross and Huang (2016) for transition-based constituent parsing.

All above work contact the distributed representation of head and modifier words outputted by LSTM as input to MLP to calculate the score of a potential dependency edge. Borrowing the idea from Luong et al. (2015), Dozat and Manning (2016) used a bilinear transformation between representation of the head and modifier words to calculate the score. While, they also notice that there are two disadvantages of using the representation directly. The first is that they contain much more information than is necessary for calculating the score, because they are recurrent, they also contain information needed for calculating scores elsewhere in the sequence. Training on the entire vector then means training on superfluous information, which is likely to lead to overfitting. The second disadvantage is that the representation $\mathbf{r}_i$ consists of the concatenation of the left recurrent state $\overleftarrow{r_i}$ and the right recurrent state $\overrightarrow{r_i}$, meaning using by itself in the bilinear transformation keeps the features learned by the two LSTMs distinct; ideally we would like the model to learn features composed from both. Dozat and Manning (2016) address both of these issues simultaneously by first applying (distinct) MLP functions with a smaller hidden size to the two recurrent states $\mathbf{r}_i$ and $\mathbf{r}_j$ before the bilinear operation. This allows the model to combine the two recurrent states together while also reducing the dimensionality. Another change to the bilinear scoring mechanism is to add a linear transformation of the head word representation to scoring function, which captures the prior probability of a word taking any dependent. They name the new method as *biaffine* transformation. Their model is a two-stage one with additional dependency relation classification stage. The biaffine transformation scoring function again is used to predict a label for each dependency edge. Finally, they achieve the state-of-the-art performance on English Penn Treebank test set.

## 4.5 Neural Transition-Based Methods

### *4.5.1 Greedy Shift-Reduce Dependency Parsing*

The outputs of dependency parsing are syntactic trees, which is a typical structure as sequences are. Graph-based dependency parsers score elements in dependency graphs, such as labels and sibling labels. In contrast, transition-based dependency parsers utilize shift-reduce actions to construct outputs incrementally. Seminal work use statistical models such as SVM to make greedy local decisions on the actions to take, as exemplified by MaltParser (Nivre 2003). Such greedy parsing processes can be illustrated in Table 4.1. At each step, the context, or parser configuration, I can be abstracted in Fig. 4.5, where the stack $\sigma$ contains partially processed words $s_0$, $s_1$, from the top, and the buffer $\beta$ contains the incoming words $q_0, q_1$, from the sentence. The task of a greedy local parser is to find the next parsing action given the current configuration, where an example set of actions is shown in Sect. 4.3.2.

MaltParser works by extracting features from the top nodes of $\sigma$ and the front words of $\beta$. For example, the form and POS of $s_0$, $s_1$, $q_0$ and $q_1$ are all used as binary discrete features. In addition, the forms, POS, and dependency arc labels of dependents of $s_0$, $s_1$ and other nodes on $\sigma$ can be used as additional features. Here, the dependency arc label of a word refers to the label of the arc between the word and the word it modifies. Given a parser configuration, all such features are extracted and fed to an SVM classifier, the output of which is a shift-reduce actions over a set of valid actions.

Chen and Manning (2014) built a neural network alternative of MaltParser, the structure of which is shown in Fig. 4.6a. Similar to MaltParser, features are extracted from the top of $\sigma$ and the front of $\beta$ given a parser configuration, and then used for predicting the next shift-reduce action to take. Chen and Manning (2014) follow Zhang and Nivre (2011) in defining the range of word, POS and label features. On the other hand, different from using discrete indicator features, embeddings are used to represent words, POS and arc labels. As shown in Fig. 4.6a, a neural network consisting of three layers is used to predict the next action given the input features. In the input layer, word, POS, and arc label embeddings from the context are concatenated. The hidden layer takes the resulting input vector, and apply a linear transformation before a cube activation function:

$$h = (Wx + b)^3.$$

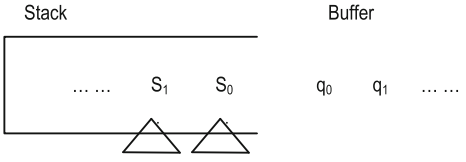
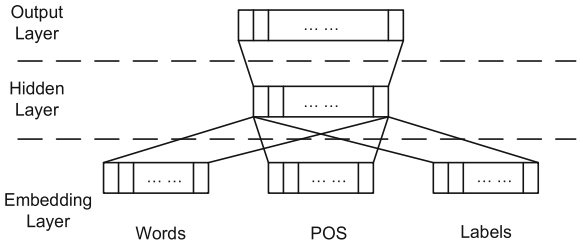
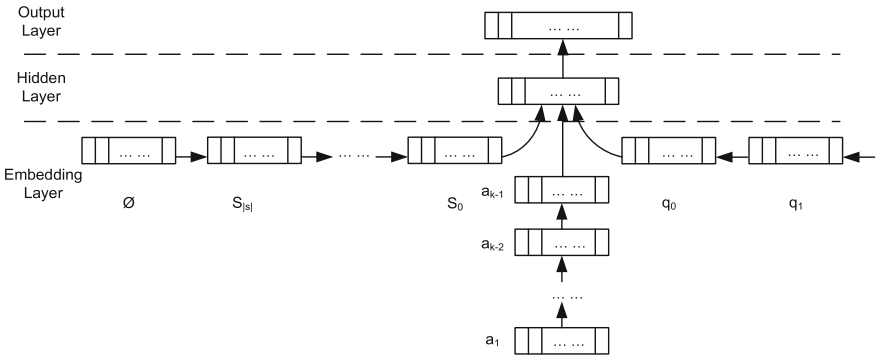



**Fig. 4.5** Context of shift-reduce dependency parsing




(a) Chen and Manning (2014)




(b) Dyer et al. (2015)

**Fig. 4.6** Two greedy parsers

The motivation behind using a cube function as the nonlinear activation function instead of the standard *sigmoid* and *tanh* functions is that it can achieve arbitrary combination of three elements in the input layer, which has been traditionally defined manually in statistical parsing models. This method empirically worked better than alternative activation functions. Finally, the hidden layer is passed as input to a standard *softmax* layer to choose the action.

The parser of Chen and Manning (2014) outperformed MaltParser significantly on several benchmarks. The main reasons are twofold. First, the use of word embeddings allows syntactic and semantic information of words to be learned from large raw data via unsupervised pretaining, which increases the robustness of the model. Second, the hidden layer achieves the effect of complex feature combinations, which is done manually in statistical models. For example, a combined feature can be $s_0wq_0p$, which captures the form of $s_0$ and the POS of $q_0$ simultaneously. This can be a strong indicator of certain actions to take. However, such combinations can be exponentially many, which requires significant manual efforts in feature engineering. In addition, they can be highly sparse if more than two features are combined into one feature. Such sparsity can cause issues in both accuracies and speeds, since they can result in a statistical model with tens of millions of binary indicator features. In contrast,

the neural model of Chen and Manning (2014) is compact and less sparse, making it strong in rendering contexts while less subject to overfitting.

The dense input feature representations of Chen and Manning (2014) are highly different from the manual feature templates of traditional statistical parsers, the former being real-valued and low-dimensional, while the latter being binary 0/1 valued and high-dimensional. Intuitively, they should capture different aspects of the same input sentences. Inspired by this observation, Zhang and Zhang (2015) built an extension of Chen and Manning (2014)'s parser, integrating traditional indicator features by concatenating a large sparse feature vector to the hidden vector of Chen and Manning (2014), before feeding it to the *softmax* classification layer. This combination can be regarded as an integration of decades of human labor in feature engineering, and the strong but relatively less interpretable power of automatic feature combination using neural network models. The results are much higher compared to the baseline Chen and Manning (2014) parser, showing that indicator features and neural features are indeed complimentary in this case.

Similar to the observation of Xu et al. (2015) over the super tagger of Lewis and Steedman (2014), Kiperwasser and Goldberg (2016) found the use of local context of Chen and Manning (2014) a potential limitation of their model. To address this issue, they extracted nonlocal features by using LSTMs over the input word and POS features of each word, resulting in a sequence of hidden vector representations for input words. Compared with the feature vectors of Chen and Manning (2014), these hidden feature vectors contain nonlocal sentential information. Kiperwasser and Goldberg (2016) utilized bidirectional LSTMs over the input word sequence, and stacked two LSTM layers to derive hidden vectors. Stack and buffer features are extracted from the corresponding hidden layer vectors, before being used for action classification. This method showed large accuracy improvements over Chen and Manning (2014), demonstrating the power of LSTM in collecting global information.

As shown in Fig. 4.6b, Dyer et al. (2015) took a different method to address the lack of nonlocal features in Chen and Manning (2014)'s model, using LSTMs to represent the stack $\sigma$, the buffer $\beta$ and the sequence of actions that have already been take. In particular, words on the stack are modeled left to right, recurrently, while words on the buffer are modeled right-to-left. The action history is modeled recurrently in temporal order. Since the stack is dynamic, it is possible for words to be popped off the top of it. In this case, Dyer et al. (2015) use a "stack LSTM" structure to model the dynamics, recording the current top of stack with a pointer. When a word is pushed on top of $s_0$, the word and the hidden state of the stack LSTM for $s_0$ are used to advance the recurrent state, resulting in a new hidden vector for the new word, which becomes $s_0$, and $s_0$ becomes $s_1$ after the pushing step. In the reverse direction, if $s_0$ is popped off the stack, the top pointer is updated, moving from the hidden state of $s_0$ to that of $s_1$ of the stack LSTM, with $s_1$ becoming $s_0$ after the action. By using the hidden states of the top of $\sigma$, the front of $\beta$ and the last action to represent the parser configuration, Dyer et al. (2015) obtained large improvements over the model of Chen and Manning (2014).

Dyer et al. (2015) represented input words with a retrained embedding, a randomly initialized but fine-tuned embedding and the embedding of their POS.

Ballesteros et al. (2015) extended the model of Dyer et al. (2015), by further using an LSTM to model the character sequence in each word. They experimented with multilingual data and observed consistently strong results. Further along this direction, Ballesteros et al. (2016) address the issue of inconsistence between action histories during training and during testing, by simulating testing scenarios during training, where the history of actions is predicted by the model rather than gold-standard action sequences, when a specific action is predicted. This idea is similar to the idea of scheduled sampling by Bengio et al. (2015).

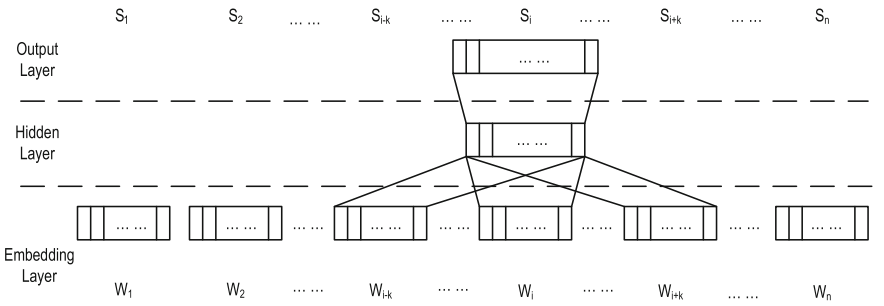
### *4.5.2 Greedy Sequence Labeling*

Given an input sentence, a greedy local sequence labeler works incrementally, assigning a label to each input word by making a local decision, and treating the assignment of labels as classification tasks. Strictly speaking, this form of sequence labeler can be regarded as either graph-based or transition-based, since each label assignment can be regarded as either disambiguating the graph structure ambiguities or transition action ambiguities. Here, we classify greedy local sequence labeling as transition-based due to the following reason. Graph-based sequence labeling models typically disambiguate whole sequences of labels as a single graph by making Markov assumptions on output labels, so that exact inference is feasible using the Viterbi algorithm. Such constraints imply that features can only be extracted over local label sequences, such as second-order and third-order transmission features. In contrast, transition-based sequence labeling models do not impose Markov properties on the outputs, and therefore typically extract highly nonlocal features. In consequence, they typically use greedy search or beam search algorithms for inference. All the examples below are greedy algorithms, and some use highly nonlocal features.

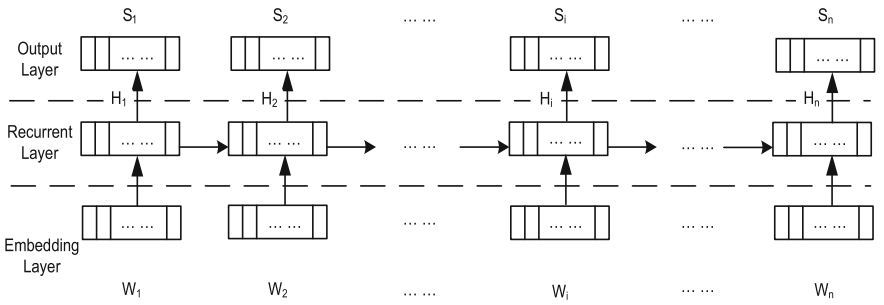
A strand of work has been done using neural models for CCG super tagging, which is a more challenging tasks compared to POS tagging. CCG is a lightly lexicalized grammar, where much syntactic information is conveyed in lexical categories, namely supertags in CCG parsing. Compared with shallow syntactic labels such as POS, super tags contain rich syntactic information, and also denote predicate-argument structures. There are over 1000 super tags that frequently occur in treebanks, which makes super tagging a challenging task.

Traditional statistical models for CCG super tagging employ CRF (Clark and Curran 2007) where features for each label are extracted over a word window context, and POS information is used as crucial features. This makes POS tagging a necessary preprocessing step before super tagging, thus making it possible for POS tagging errors to negatively affect super tagging quality.

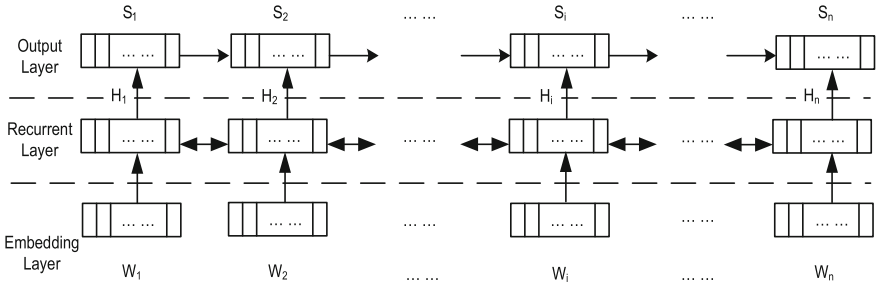
Lewis and Steedman (2014) investigated a simple neural model for CCG super tagging, the structure of which is shown in Fig. 4.7a. In particular, given an input sentence, a three-layer neural network is used to assign super tags to each word. The first (bottom) layer is an embedding layer, which maps each word into its embedding form. In addition, a few binary-valued discrete features are concatenated to the




(a) Feed forward network




(b) Recurrent network with independent labels




(c) Recurrent network with chained labels

**Fig. 4.7** Neural models for CCG supertagging

embedding vector, which include the two-letter suffix of the word, and a binary indicator whether the word is capitalized. The second layer is a hidden layer for feature integration. For a given word $w_i$, a context window of word $w_{i-k}, w_i, w_{i+k}$ is used for feature extraction. Augmented input embeddings from each word in the context window are concatenated, and fed to the hidden layer, which uses a *tanh* activation function to achieve nonlinear feature combination. The final (top) layer is a *softmax* classification function, which assigns probabilities to all possible output labels.

This simple model worked surprisingly well, leading to better parsing accuracies for both in-domain data and cross-domain data compared to the CRF baseline tagger. Being a greedy model, it also runs significantly faster compared to a neural CRF alternative, while giving comparable accuracies. The success can be attributed to the power of neural network models in automatically deriving features, which makes POS tagging unnecessary. In addition, word embeddings can be retrained over large raw data, thereby alleviating the issue of feature sparsity in baseline discrete models, allowing better cross-domain tagging.

The context window of Lewis and Steedman (2014) follows the work of Collobert and Weston (2008), which is local and comparable to the context window of CRF (Clark and Curran 2007). On the other hand, recurrent neural networks have been used to extract nonlocal features from the whole sequence, achieving better accuracies for a range of NLP tasks. Motivated by this observation, Xu et al. (2015) extended the method of Lewis and Steedman (2014), by replacing the window-based hidden layer with a recurrent neural network layer (Elman 1990). The structure of this model is shown in Fig. 4.7b.

In particular, the input layer of Xu et al. (2015) is identical to the input layer of Lewis and Steedman (2014), where a word embedding is concatenated with two-character suffix and capitalization features. The hidden layers are defined by an Elman recurrent neural network, which recurrently computes the hidden state for $w_i$ using the previous hidden state $h_{i-1}$ and the current embedding layer of $w_i$. A *sigmoid* activation function is used to achieve nonlinearity. Finally, the same form of output layers is used to label each word locally.

Compared with the method of Lewis and Steedman (2014), the RNN method gives improved accuracies for both super tagging and subsequent CCG parsing using a standard parser model. In addition, the RNN super tagging also gives better 1-best super tagging accuracy compared to the CRF method of Clark and Curran (2007), while the NN method of Lewis and Steedman did not achieve. The main reason is the use of recurrent neural network structure, which models unbounded history context for the labeling of a word.

Lewis and Steedman (2014) made further improvements to the model of Xu et al. (2015) by using LSTMs to replace the Elman RNN structure in the hidden layer. In particular, a bidirectional LSTM is used to derive the hidden features $h_1, h_2, h_n$ given the embedding layer. The input representations are also adjusted slightly, where the discrete components are discarded, and the 1- to 4-letter prefixes and suffixes of each word are represented with embedding vectors, and concatenated to the embeddings of words as input features. Thanks to these changes, the final model gives much improved accuracies for both super tagging and subsequent CCG parsing. In addition, by using tri-training techniques, the results are further raised, reaching 94.7% F1 on 1-best tagging.

The models of Xu et al. (2015) and Lewis and Steedman (2014) consider nonlocal dependencies between words in the input, yet does not capture nonlocal dependencies between output labels. In this respect, they are less expressive compared with the CRF model of Clark and Curran (2007), which considers the dependencies between three consecutive labels. To address this issue, Vaswani et al. (2016) leverage LSTM

on the output label sequence also, by considering the label history $s_1$, $s_2$, $s_{i-1}$ when the word $w_i$ is labeled. The model structure is shown in Fig. 4.7c.

The input layer of this model uses the same representations as Lewis and Steedman (2014), and the hidden layer is similar to that of Lewis and Steedman (2014). In the output layer, the classification of each label $s_i$ is based on both the corresponding hidden layer vector $h_i$ and the previous label sequence, represented by the hidden states $h^s_{i-1}$ of a label LSTM. The label LSTM is unidirectional, where each state $h^s_i$ is derived from its previous state $h^s_{i-1}$ and the previous label $s_i$. To further improve the accuracies, scheduled sampling (Bengio et al. 2015) is used to find training data that are more similar to test cases. During training, the history label sequence $s_1$, $s_2$, $s_{i-1}$ for labeling $s_i$ is sampled by choosing the predicted supertag at each position with a sampling probability $p$. This way, the model can learn better how to assign a correct label even if errors are made in the history during test time.

Vaswani et al. (2016) showed that by adding the output label LSTM, the accuracies can be slightly improved if scheduled sampling is applied, but decreases compared with the greedy local output model of Lewis and Steedman (2014) without scheduled sampling. This shows the usefulness of scheduled sampling, which avoids overfitting to gold label sequences and consequent tossing of test data robustness.

### *4.5.3 Globally Optimized Models*

Greedy local neural models have demonstrated their advantage over their statistical counterparts by leveraging word embeddings to alleviate sparseness, and using deep neural networks to learn nonlocal features. Syntactic and semantic information over the whole sentence has been utilized for structured prediction, and nonlocal dependencies over labels are also modeled. On the other hand, the training of such models is local, and hence can potentially lead to label bias, since the optimal sequence of actions does not always contain locally topical actions. Globally optimized models, which have been the dominant approach for statistical NLP, have been applied to neural models also.

Such models typically apply beam search (in Algorithm 1), where an agenda is used to keep the $B$ highest scored sequences of actions at each step. The beam search process for arc-eager dependency parsing is shown in Fig. 4.8. Here the blue circle illustrates the gold-standard sequence of actions. As shown in Fig. 4.8, at some steps, the gold-standard state may not be the highest scored in the agenda. In case of local search, such situation leads to search errors. For beam search, however, it is possible for the decoder to recover the gold-standard state in subsequent stages as the highest scored item in the agenda.

The beam search algorithm for transition-based structured prediction is formally shown in Algorithm 1. Initially, the agenda contains only the start state in the state transition system. At each step, all items in the agenda are expanded by applying all possible transition actions, leading to a set of new states. From these states, the highest scored $B$ are selected, and used as agenda items for the next step. Such process

**Algorithm 1** The generic beam search algorithm

```
1: function BEAM- SEARCH(problem, agenda, candidates, B)
2:     candidates ← {StartItem(problem)}
3:     agenda ← Clear(agenda)
4:     loop
5:         for each candidate ∈ candidates do
6:             agenda ← Insert(Expand(candidate, problem), agenda)
7:         end for
8:         best ← Top(agena)
9:         if GoalTest(problem, best) then
10:            return best
11:        end if
12:        candiates ← Top − B(agenda, B)
13:        agenda ← Clear(agenda)
14:    end loop
15: end function
```

repeats until terminal states have been reached, and the highest scored state in the agenda is taken as the output. Similar to greedy search, the beam search algorithm has a linear time complexity with respect to the action sequence length.

The items in the agenda are ranked using their global scores, which are the total scores of all transition actions in the sequence. Different from greedy local models, the training objective of globally optimized models is to different full sequences of actions based on their global scores. There are two general training approaches, with one being to maximize the likelihood of gold-standard sequences of actions, other being to maximize the score margin between the gold-standard sequence of

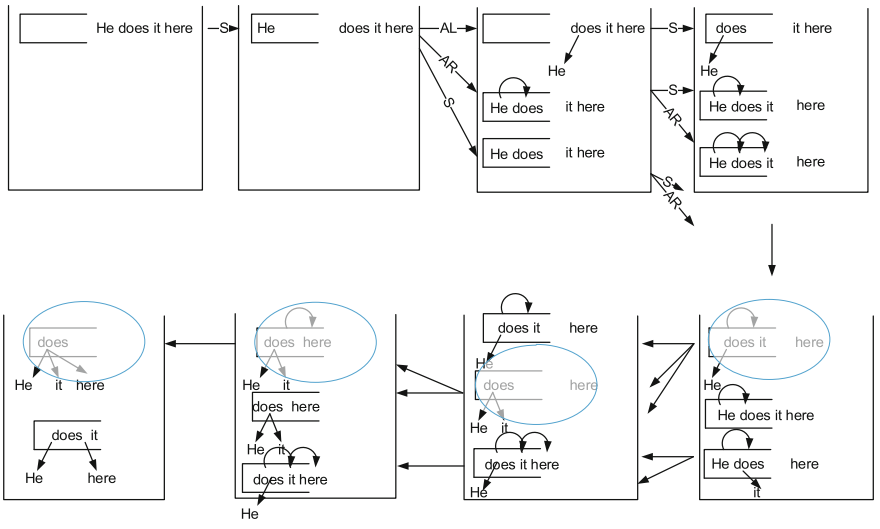



**Fig. 4.8** Parsing process given state transition system with beam search

action and non-gold-standard sequences of actions. Other training objectives are occasionally used, as will be shown later.

Following Zhang and Clark (2011b), most globally optimized models regard training as beam search optimization, where negative training examples are sampled by the beam search process itself, and used together with the gold-standard positive example to update the model. Here we use Zhang and Clark (2011b) as one example to illustrate the training method. Online learning is used, where an initial model is applied to decode the training examples. During the decoding of each sample, the gold-standard sequence of actions is available. The same beam search algorithm above is used, as in test cases. At any step, if the gold-standard sequence of actions falls out of the agenda, a search error is unavoidable. At this situation, search is stopped, and the model is updated by using the gold-standard sequence of actions till this step as the positive example, and the current highest scored sequence of actions in the beam as a negative example. Zhang and Clark (2011b) used a statistical model, where model parameters are updated using the perceptron algorithm of Collins (2002). The early stopping of beam search is known as *early update* (Collins and Roark 2004). In the case where the gold-standard sequence of action remains in the agenda until decoding finishes, the training algorithm checks if it is the highest scoring in the last step. If so, the current training sample is finished without parameter update; otherwise the current highest scored sequence of actions in the beam is taken as a negative example to update parameters. The same process can repeat over the training examples for multiple iterations, and the final model is used for testing.

We discuss a strand of work using global training for neural transition-based structured prediction below, categorized by their training objectives.

#### 4.5.3.1 Large Margin Methods

The large margin objective maximizes the score difference between gold-standard output structures and incorrect output structures; it has been used by discrete structured prediction methods such as the structured perceptron (Collins 2002) and MIRA (Crammer and Singer 2003). The ideal large margin training objective should ensure that the gold-standard structure is scored higher than all incorrect structures by a certain margin. However, for structured prediction tasks, the number of incorrect structures can be exponentially many, hence making the exact objective intractable in most cases. The perceptron approximates this objective by making model adjustments for the most violated margin, and has theoretical guarantee of convergence in training. In particular, given the gold-standard structure as a positive example, and the max-violation incorrect structure as a negative example, the perceptron algorithm adjusts model parameters by adding the feature vector of the positive example to the model, and subtracting the feature vector of the negative example from the model parameter vector. By repeating this procedure for all training examples, the model converges to scoring gold-standard structures higher than incorrect structures. The perceptron algorithm finds a negative example for each gold-standard training example, such that the violation of the ideal score margin is the largest. This typi-

cally implies the searching for a highest scored incorrect output, or one that ranks the highest by considering both its current model score and it deviation from the gold-standard. In the latter case, the structured dilation is the cost of the incorrect output, where outputs with similar structures to the gold-standard have less cost. By considering not only the model score but also the cost, this training objective allows model scores to differentiate not only gold-standard and incorrect structures, but also between different incorrect structures by their similarity to the correct structure.

With neural networks, the training objective translates to maximizing the score difference between a given positive example and a corresponding negative example. This objective is typically achieved by taking the derivative of the score difference with respect to all model parameters, updating model parameters using gradient-based methods such as AdaGrad (Duchi et al. 2011).

Zhang et al. (2016a) used such a large margin objective for transition-based word segmentation. As shown in Sect. 1.1, a state for this task can be encoded in a a pair $s = (\sigma, \beta)$, where $\sigma$ contains a list of recognized words, and $\beta$ contains the list of next incoming characters. Zhang et al. (2016a) use a word LSTM to represent $\sigma$, and a bidirectional character LSTM to represent $\beta$. In addition, following Dyer et al. (2015), they also use an LSTM to represent the sequence of actions that have been taken. Given a state $s$, the three LSTM context representations are integrated and used to score SEP and APP actions. Formally, given a state $s$, the score of action $a$ can be denoted as $f(s, a)$, where $f$ is the network model. As a global model, Zhang et al. (2016a) calculate the score of a sequence of actions for ranking the state they lead to, where

$$score(s_k) = \sum_{i=1}^{k} f(s_{i-1}, a_i).$$

Following Zhang and Clark (2011b), online learning with early update is used. Each training example is decoded using beam search, until the gold-standard sequence of transition actions fall out of beam, or does not rank highest by a score margin after decoding finishes. Here the margin between the gold-standard structure and an incorrect structure is defined by the number of incorrect actions $\Delta$, weighted by a factor $\eta$. Therefore, given a state after $k$ actions, the corresponding loss function for training the network is defined as follows:

$$L(s_k) = \max\left(score(s_k) - score(s_k^g) + \eta\Delta(s_k, s_k^g), 0\right),$$

where $s_k^g$ is the corresponding gold-standard structure after $k$ transitions.

During training, Zhang et al. (2016a) use the current model score $score(s_k)$ plus $\Delta(s_k, s_k^g)$ to rank states in the agenda, so that structural differences are considered for finding the maximum violation. Given this ranking, a negative example can be chosen in the early update and final update cases. Model parameters are updated according to the less function between $s_k$ and $s_k^g$ above. Since $score(s_k)$ is the sum of all action scores, the loss is evenly distributed to each action. In practice, back-propagation is used to train the network, where the derivative of the lost function is taken with respect

to model parameters via the network $f(s_{i-1}, a_i)$ for $i \in [1..k]$. Since each action $a_i$ shares the same representation layers as described earlier, their losses accumulate for model parameter updates. AdaGrad is used to change the model.

Cai and Zhao (2016) adopted a very similar neural model for word segmentation. Both the models of Zhang et al. (2016a) and Cai and Zhao (2016) can be regarded as extensions of the method of Zhang and Clark (2007) using neural network. On the other hand, the scoring function of Cai and Zhao (2016) is different from that of Zhang et al. (2016a), Cai and Zhao (2016) also uses beam search, segmenting a sentence incrementally. But their incremental steps are based on words, rather than characters. They used multiple beams to store partial segmentation outputs containing the same numbers of characters, which is similar to Zhang and Clark (2008a). As a result, constraints to the word size must be used to ensure linear time complexity. For training, exactly the same large margin objective is taken.

A slightly different large margin objective is used by Watanabe and Sumita (2015) for constituent parsing. They adopt the transition system of Sagae et al. (2005) and Zhang and Clark (2009), where a state can be defined as a pair $(\sigma, \beta)$, similar to the dependency parsing case in Sect. 1.1. Here $\sigma$ contains partially constructed constituent trees, and $\beta$ contains next incoming words. A set of transition actions including SHIFT, REDUCE and UNARY are used to consume input words and construct output structures. Interested readers can refer to (Sagae and Lavie 2005) and (Zhang and Clark 2009) for more details on the state transition system.

Watanabe and Sumita (2015) represent $\sigma$ using a stack LSTM structure, which dynamically change, and is similar to that of Dyer et al. (2015). $\beta$ is represented using a standard LSTM. Given this context representation, the score of a next action $a$ can be denoted as $f(s, a)$, where $s$ represents the current state and $f$ is the network structure. Similar to the case of Zhang et al. (2016a), the score of a state $s_k$ is the sum of all actions that lead to the state, as shown in Fig. 4.9:

$$score(s_k) = \sum_{i=1}^{k} f(s_{i-1}, a_i)$$

Similar to Zhang et al. (2016a), beam search is used to find the highest scored state over all structures. For training, however, max-violation update is used instead of early update (Huang et al. 2012), where the negative example is chosen by running beam search until the terminal state is reached, and then finding the intermediate state that gives the largest violation of the score margin between gold-standard and incorrect structures. Update is executed at the max-violation step. In addition, rather than using the maximum violation state as the negative example, all incorrect states in the beam are used as negative examples to enlarge the sample space, and the training objective is defined to minimize the loss:

$$L = \max\left(\mathbf{E}_{s_k \in A} score(s_k) - score(s_k^g + 1)\right).$$

Here $A$ represents the agenda, and the expectation $\mathbf{E}_{s_k \in A} score(s_k)$ is calculated based on probabilities of each $s_k$ in the agenda using model scores:

$$p(s_k) = \frac{\exp(score(s_k))}{\sum_{s_k \in A} \exp(score(s_k))}.$$

#### 4.5.3.2 Maximum Likelihood Methods

Maximum likelihood objectives for neural structured prediction are inspired by log-linear models. In particular, given the score of an output $y$ $score(y)$, a log-linear model calculates its probability as

$$p(y) = \frac{\exp(score(y))}{\sum_{y \in Y} \exp(score(y))},$$

where $Y$ represents the set of all outputs. When $y$ is a structure, this log-linear model becomes CRF under certain constraints.

A line of work investigate a similar objective by assuming the structured score calculation in Fig. 4.9 for transition-based models, where the score for a state $s_k$ is calculated as

$$score(s_k) = \sum_{i=1}^{k} f(s_{i-1}, a_i).$$

The definition of $f$ and $a$ are the same as in the previous section. Given this score calculation, the probability of the state $s_k$ is

$$p(s_k) = \frac{\exp(score(s_k))}{\sum_{s_k \in S} \exp(score(s_k))},$$

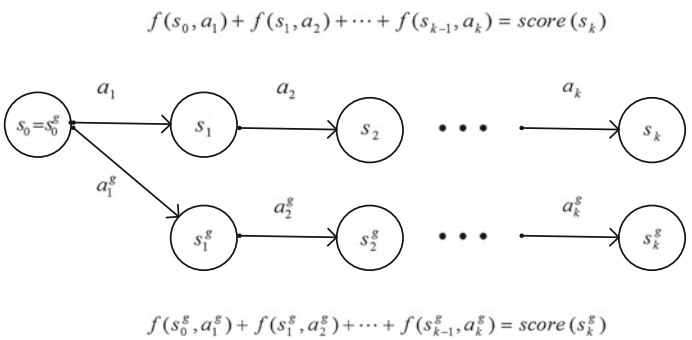



**Fig. 4.9** Structured score calculation

where $S$ denotes all possible states after $k$ transition actions. Apparently, the number of states in $S$ grows exponentially with $k$, as the number of structures they contain. As a result, it is difficult to estimate the denominator for maximum likelihood training, as in the case of CRF. For CRF, the issue is solved by imposing constraints on feature locality, so that marginal probabilities of features can be used to estimate the partition function. For transition-based models, however, such feature locality is nonexistent.

Zhou et al. (2015) first addressed this issue by using all states in the agenda to approximate $S$ during beam search. They perform beam search and online learning, using early update in the same way as Zhang et al. (2016a). On the other hand, during each update, rather than calculating a score margin between positive and negative examples, Zhou et al. (2015) maximize the approximated likelihood of the gold-standard stage $s_g$, where

$$p(s_g) = \frac{\exp(score(s_g))}{\sum_{s_k \in A} \exp(score(s_k))},$$

where $A$ represents the agenda, as in the last section.

This method uses the probability mass of states in the agenda to approximate the partition function, and hence is referred to as beam contrastive learning by Zhou et al. (2015). Zhou et al. (2015) applied the training objective to the task of transition-based dependency parsing, achieving better results compared to Zhang and Nivre (2011).

Andor et al. (2016) applied this method to more structured prediction tasks, including part-speech-tagging. They also obtained significantly better results than Zhou et al. (2015) by using a better baseline method and doing more thorough hyperparameter search. In addition, Andor et al. (2016) gave a theoretical justification that the globally normalized model outperforms locally trained baselines.

#### 4.5.3.3 Maximum Expected F1

Another training objective that has been tried is maximum F1, which Xu et al. (2016) used for transition-based CCG parsing (Zhang and Clark 2011a). In particular, Xu et al. (2016) use beam search to find the highest scored state, where the score of each state is given by the calculation method of Fig. 4.9. Given state $s_k$, the score is calculated as

$$score(s_k) = \sum_{i=1}^{k} g(s_{i-1}, a_i).$$

Here the function $g$ represents a network model, and $a$ represents a transition action. The difference between the network function $g$ of Xu et al. (2016) and the network function of all aforementioned methods is that $g$ uses a *softmax* layer to normalize the output actions, while $f$ does not use nonlinear activation functions over scores of different actions given a state.

The training objective of Xu et al. (2016) is

$$E_{s_k \in A} F1(s_k) = \sum_{s_k \in A} p(s_k) F1(s_k),$$

where $A$ denotes the beam after parsing finishes, and $F1(s_k)$ denotes the F1 score of $s_k$ as evaluated by standard metrics against the gold-standard structure.

Xu et al. (2016) calculates $p(s_k)$ using

$$p(s_k) = \frac{\exp(score(s_k))}{\sum_{s_k \in A} \exp(score(s_k))}$$

which is consistent to all aforementioned methods.

## 4.6 Summary

In this chapter, we provided an overview on the application of deep learning to lexical analysis and parsing, two standard tasks in NLP, and compare the deep learning approach with traditional statistical methods.

First, we introduced the definitions of lexical analysis and parsing. They model structured properties of words and their relationships to each other. The commonly used techniques in these tasks include word segmentation, part-of-speech tagging, and parsing. The most important characteristic of lexical analysis and parsing is that the outputs are structured.

Then, we introduced two types of traditional methods usually used to solve these structured prediction tasks: graph-based methods and transition-based methods. Graph-based methods exploit output structures based on their characteristics directly, while transition-based methods transform the output construction processes into state transition processes, and subsequently process sequences of transition actions.

Finally, we in this chapter introduced methods using neural network and deep learning models in both graph-based and transition-based structured prediction.

While recent advances have shown that neural network models can be used effectively to augment or replace statistical models in the traditional graph-based and transition-based frameworks for lexical analysis and parsing, they have begun to illustrate the strong representation power of neural networks which can go beyond the function of mere modeling. For example, in the traditional statistical modeling approach, it has been commonly understood that local training leads to weaknesses such as label bias (Lafferty et al. 2001). However, the model and method described in (Dozat and Manning 2016) achieve state-of-the-art accuracy results using a neural model that factors out single dependency arcs as training objectives, without globally training the probabilities of a dependency tree. This suggests that structural correlations between output labels can be obtained by the strong representation of word sequences using LSTMs. The future direction for lexical analysis and parsing in NLP will likely be a unification between well-established research on structured learning and the emerging power of deep learning.

## References


Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., et al. (2016). Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 2442–2452). Berlin, Germany: Association for Computational Linguistics.

Ballesteros, M., Dyer, C., & Smith, N. A. (2015). Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 349–359). Lisbon, Portugal: Association for Computational Linguistics.

Ballesteros, M., Goldberg, Y., Dyer, C., & Smith, N. A. (2016). Training with exploration improves a greedy stack LSTM parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 2005–2010). Austin, Texas: Association for Computational Linguistics.

Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems, NIPS'15* (pp. 1171–1179). Cambridge, MA, USA: MIT Press.

Bohnet, B. & Nivre, J. (2012). A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (pp. 1455–1465). Jeju Island, Korea: Association for Computational Linguistics.

Booth, T. L. (1969). Probabilistic representation of formal languages. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, *00*, 74–81.

Cai, D., & Zhao, H. (2016). Neural word segmentation learning for Chinese. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 409–420). Berlin, Germany: Association for Computational Linguistics.

Carnie, A. (2012). *Syntax: A Generative Introduction* (3rd ed.). New York: Wiley-Blackwell.

Chen, D., & Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP-2014*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1724–1734). Doha, Qatar: Association for Computational Linguistics.

Choi, J. D., & Palmer, M. (2011). Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (pp. 687–692). Portland, Oregon, USA: Association for Computational Linguistics.

Chu, Y., & Liu, T. (1965). On the shortest arborescence of a directed graph. *Scientia Sinica*, *14*, 1396–1400.

Clark, S., & Curran, J. R. (2007). Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, *33*(4), 493–552.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics* (pp. 16–23). Madrid, Spain: Association for Computational Linguistics.

Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing* (pp. 1–8). Association for Computational Linguistics.

Collins, M., & Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume* (pp. 111–118). Barcelona, Spain.

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning, ICML '08* (pp. 160–167). New York, NY, USA: ACM.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*, 2493–2537.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, *7*, 551–585.

Crammer, K., & Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, *3*, 951–991.

Cross, J., & Huang, L. (2016). Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 1–11). Austin, Texas: Association for Computational Linguistics.

Dozat, T., & Manning, C. D. (2016). Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

Durrett, G., & Klein, D. (2015). Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Vol. 1: Long Papers, pp. 302–312). Beijing, China: Association for Computational Linguistics.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., & Smith, N. A. (2015). Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Vol. 1: Long Papers, pp. 334–343). Beijing, China: Association for Computational Linguistics.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, *71B*, 233–240.

Eisner, J. (1996). Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics* (pp. 79–86). Santa Cruz, California, USA: Association for Computational Linguistics.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*(3), 277–296.

Graves, A. (2008). *Supervised sequence labelling with recurrent neural networks*. Ph.D. thesis, Technical University Munich.

Hall, D., Durrett, G., & Klein, D. (2014). Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 228–237). Baltimore, MD: Association for Computational Linguistics.

Hatori, J., Matsuzaki, T., Miyao, Y., & Tsujii, J. (2012). Incremental joint approach to word segmentation, pos tagging, and dependency parsing in Chinese. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 1045–1053), Jeju Island, Korea: Association for Computational Linguistics.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Huang, L., Fayong, S., & Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 142–151). Montréal, Canada: Association for Computational Linguistics.

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing* (2nd ed.). Upper Saddle River, NJ, USA: Prentice-Hall Inc.

Kbler, S., McDonald, R., & Nivre, J. (2009). Dependency parsing. *Synthesis Lectures on Human Language Technologies*, *2*(1), 1–127.

Kiperwasser, E., & Goldberg, Y. (2016). Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, *4*, 313–327.

Lafferty, J. D., McCallum, A., & Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01* (pp. 282–289), San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Lewis, M., & Steedman, M. (2014). A* CCG parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 990–1000). Doha, Qatar: Association for Computational Linguistics.

Li, F., Zhang, Y., Zhang, M., & Ji, D. (2016). Joint models for extracting adverse drug events from biomedical text. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016* (pp. 2838–2844). New York, NY, USA, 9–15 July 2016.

Li, Q., & Ji, H. (2014). Incremental joint extraction of entity mentions and relations. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 402–412). Baltimore, MD: Association for Computational Linguistics.

Liu, J., & Zhang, Y. (2015). An empirical comparison between n-gram and syntactic language models for word ordering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 369–378). Lisbon, Portugal: Association for Computational Linguistics.

Liu, Y., Che, W., Guo, J., Qin, B., & Liu, T. (2016). Exploring segment representations for neural segmentation models. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016* (pp. 2880–2886). New York, NY, USA, 9–15 July 2016.

Liu, Y., Zhang, Y., Che, W., & Qin, B. (2015). Transition-based syntactic linearization. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 113–122). Denver, Colorado: Association for Computational Linguistics.

Luong, T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1412–1421). Lisbon, Portugal: Association for Computational Linguistics.

Lyu, C., Zhang, Y., & Ji, D. (2016). Joint word segmentation, pos-tagging and syntactic chunking. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16* (pp. 3007–3014). AAAI Press.

Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press.

McDonald, R. (2006). *Discriminative learning spanning tree algorithm for dependency parsing*. PhD thesis, University of Pennsylvania.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)* (pp. 149–160).

Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, *34*(4), 513–554.

Pei, W., Ge, T., & Chang, B. (2015). An effective neural network model for graph-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Vol. 1: Long Papers, pp. 313–322), Beijing, China: Association for Computational Linguistics.

Petrov, S., Barrett, L., Thibaux, R., & Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics* (pp. 433–440), Sydney, Australia: Association for Computational Linguistics.

Puduppully, R., Zhang, Y., & Shrivastava, M. (2016). Transition-based syntactic linearization with lookahead features. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 488–493). San Diego, CA: Association for Computational Linguistics.

Qian, T., Zhang, Y., Zhang, M., Ren, Y., & Ji, D. (2015). A transition-based model for joint segmentation, pos-tagging and normalization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1837–1846), Lisbon, Portugal: Association for Computational Linguistics.

Sagae, K., & Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology, Parsing '05* (pp. 125–132). Stroudsburg, PA, USA: Association for Computational Linguistics.

Sagae, K., Lavie, A., & MacWhinney, B. (2005). Automatic measurement of syntactic development in child language. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)* (pp. 197–204). Ann Arbor, MI: Association for Computational Linguistics.

Sarawagi, S., & Cohen, W. W. (2004). Semi-Markov conditional random fields for information extraction. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in neural information processing systems 17* (pp. 1185–1192). Cambridge: MIT Press.

Shaalan, K. (2014). A survey of arabic named entity recognition and classification. *Computational Linguistics*, *40*(2), 469–510.

Smith, N. A. (2011). *Linguistic structure prediction*. Morgan and Claypool: Synthesis Lectures on Human Language Technologies.

Song, L., Zhang, Y., Song, K., & Liu, Q. (2014). Joint morphological generation and syntactic linearization. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI'14* (pp. 1522–1528). AAAI Press.

Vaswani, A., Bisk, Y., Sagae, K., & Musa, R. (2016). Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 232–237). San Diego, CA: Association for Computational Linguistics.

Wang, W., & Chang, B. (2016). Graph-based dependency parsing with bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 2306–2315). Berlin, Germany: Association for Computational Linguistics.

Wang, Z., & Xue, N. (2014). Joint pos tagging and transition-based constituent parsing in Chinese with non-local features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 733–742). Baltimore, MD: Association for Computational Linguistics.

Watanabe, T., & Sumita, E. (2015). Transition-based neural constituent parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Vol. 1: Long Papers, pp. 1169–1179). Beijing, China: Association for Computational Linguistics.

Wong, K.-F., Li, W., Xu, R., & Zhang, Z.-s., (2009). Introduction to Chinese natural language processing. *Synthesis Lectures on Human Language Technologies, 2*(1), 1–148.

Xu, W., Auli, M., & Clark, S. (2015). CCG supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Vol. 2: Short Papers, pp. 250–255). Beijing, China: Association for Computational Linguistics.

Xu, W., Auli, M., & Clark, S. (2016). Expected f-measure training for shift-reduce parsing with recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 210–220). San Diego, CA: Association for Computational Linguistics.

Xu, W., Clark, S., & Zhang, Y. (2014). Shift-reduce CCG parsing with a dependency model. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers).

Xue, N. (2003). Chinese word segmentation as character tagging. *International Journal of Computational Linguistics and Chinese Language Processing*, *8*, 29–48.

Yamada, H., & Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *In Proceedings of IWPT* (pp. 195–206).

Zhang, M., & Zhang, Y. (2015). Combining discrete and continuous features for deterministic transition-based dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1316–1321). Lisbon, Portugal: Association for Computational Linguistics.

Zhang, M., Zhang, Y., Che, W., & Liu, T. (2013). Chinese parsing exploiting characters. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 125–134). Sofia, Bulgaria: Association for Computational Linguistics.

Zhang, M., Zhang, Y., Che, W., & Liu, T. (2014). Character-level Chinese dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 1326–1336). Baltimore, MD: Association for Computational Linguistics.

Zhang, M., Zhang, Y., & Fu, G. (2016a). Transition-based neural word segmentation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 421–431), Berlin, Germany: Association for Computational Linguistics.

Zhang, Y., & Clark, S. (2007). Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics* (pp. 840–847), Prague, Czech Republic: Association for Computational Linguistics.

Zhang, Y., & Clark, S. (2008a). Joint word segmentation and POS tagging using a single perceptron. In *Proceedings of ACL-08: HLT* (pp. 888–896). Columbus, OH: Association for Computational Linguistics.

Zhang, Y., & Clark, S. (2008b). A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing* (pp. 562–571), Honolulu, HI: Association for Computational Linguistics.

Zhang, Y., & Clark, S. (2009). Transition-based parsing of the Chinese Treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies, IWPT '09* (pp. 162–171). Stroudsburg, PA, USA: Association for Computational Linguistics.

Zhang, Y., & Clark, S. (2010). A fast decoder for joint word segmentation and POS-tagging using a single discriminative model. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing* (pp. 843–852). Cambridge, MA: Association for Computational Linguistics.

Zhang, Y., & Clark, S. (2011a). Shift-reduce CCG parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (pp. 683–692). Portland, OR, USA: Association for Computational Linguistics.

Zhang, Y., & Clark, S. (2011b). Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics, 37*(1).

Zhang, Y., & Nivre, J. (2011). Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (pp. 188–193). Portland, OR, USA: Association for Computational Linguistics.

Zhang, Z., Zhao, H., & Qin, L. (2016b). Probabilistic graph-based dependency parsing with convolutional neural network. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 1382–1392), Berlin, Germany: Association for Computational Linguistics.

Zhou, H., Zhang, Y., Huang, S., & Chen, J. (2015). A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Vol. 1: Long Papers, pp. 1213–1222), Beijing, China: Association for Computational Linguistics.

Zhou, J., & Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Vol. 1: Long Papers, pp. 1127–1137), Beijing, China: Association for Computational Linguistics.

Zhu, M., Zhang, Y., Chen, W., Zhang, M., & Zhu, J. (2013). Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (Vol. 1: Long Papers, pp. 434–443), Sofia, Bulgaria: Association for Computational Linguistics.