Natural Language Processing (NLP) and Large Language Models (LLMs)
Lecture 5: Parsing

Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

WISE @ XMU

2025 年 3 月 20 日

Adapted in part from CS 224n

Overview of Linguistic Structure

- Linguistic structures: constituency grammar (成分结构) and dependency grammar (依存关系).
- We will mainly focus on dependency grammar.
- Why deep learning? Recommended reading:
  - A Fast and Accurate Dependency Parser using Neural Networks. Chen & Manning, 2014.

The Parsing Problem

- **Input:** A sentence $S = w_0 w_1 \cdots w_L$
- **Output:** A grammar tree graph
- **Training:** Given training data D, consisting of sentences annotated with grammar graphs, induce a parsing model M.
- **Parsing:** Using the learned model M, derive the optimal dependency graph for a new sentence.

Penn treebank: annotated dataset for dependency parsing

- The Penn treebank is publicly available syntactically annotated corpus.
- It is from the Wall Street Journal (50,000 sentences, 1 million words)
- It contains POS, dependency trees...

Phrase Structure

- **Words:** Cat; sleeps; sofa
- **Phrase:** The cat; sleeps peacefully; on the sofa
- **Bigger Phrase:**
  - The cat sleeps peacefully
  - On the cozy sofa
- To even bigger phrase; then sentence.

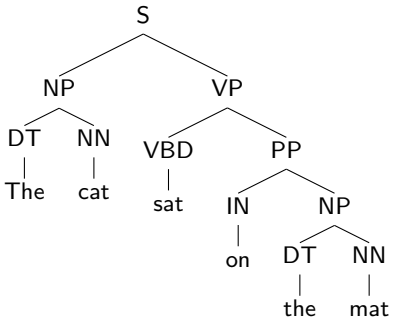
Constituency parsing

The cat sat on the mat

- The cat, sat, on the mat are all constituent.
- “cat sat on” is not a constituent.

Constituency parse tree

- Each constituent in the sentence is assigned a tag; The sentence can be represented as a grammatical tree.



Tags

- Phrase Tags:
  - NP: noun phrase (e.g., “the cat”)
  - VP: verb phrase (e.g., “sat on the mat”)
  - PP: prepositional phrase (e.g., “on the mat”)
- POS tags:
  - DT: Determiner (e.g., “the”)
  - NN: noun
  - IN: preposition
  - VBD: verb
  - etc.

Chart-based approach

- Lexical rules: POS of a word
- Phrasal eules: DT+NN = NP; IN + NP = PP

	0 (The)	1 (cat)	2 (sat)	3 (on)	4 (the)	5 (mat)
0	DT					
1		NN				
2			VBD			
3				IN		
4					DT	
5						NN

Chart-based approach

- Lexical rules: POS of a word
- Phrasal eules: DT+NN = NP; IN + NP = PP

	0 (The)	1 (cat)	2 (sat)	3 (on)	4 (the)	5 (mat)
0	DT	NP				
1		NN				
2			VBD			
3				IN		PP
4					DT	NP
5						NN

Chart-based Approach and Classification

- Consider a sequence of tokens $\mathbf{x} = (x_1, \ldots, x_L)$ and define a span as a continuous subsequence $\mathbf{x}_{i:j} = (x_i, \ldots, x_j)$.
- **Step 1**: Use binary classification to determine whether span $\mathbf{x}_{i:j}$ forms a constituent.
- **Step 2**: For spans classified as constituents, perform multiclass classification to assign one of the predefined tags.
- The classifiers can be trained using annotated corpora and foundation models such as BERT or GPT-2.
- Apply dynamic programming techniques for efficient traversal of spans.
- To enhance efficiency and avoid conflicting constituents (e.g., spans like [1,2,3] and [3,4,5] overlapping at token 3), traverse possible parse trees or, more precisely, minimum spanning trees rather than individual spans.

Transition-based constituent parsing

Input: *The hungry cat meows .*

	Stack	Buffer	Action
0		*The* | *hungry* | *cat* | *meows* | .	NT(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	NT(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .		REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)		

Figure is from Recurrent Neural Network Grammars

State Machines and Transitions

- A **state machine** defines a process for converting sentences into grammar trees.
- Composed of two key elements:
  - **States**: Consist of a *stack* (processed words) and a *buffer* (words to be processed).
  - **Transitions**: Actions that move the system from one state to another.

State Machines and Transitions

- **Learning**: Predict the next transition based on transition history.
- **Parsing**: Given a learned model, construct the optimal sequence of transitions for the input sentence.
- Most transition-based systems do not rely on a formal grammar.

Deep transition-based constituent parsing

- Input: Stack Buffer, previous action; Output: new actions
- Use deep neural networks.

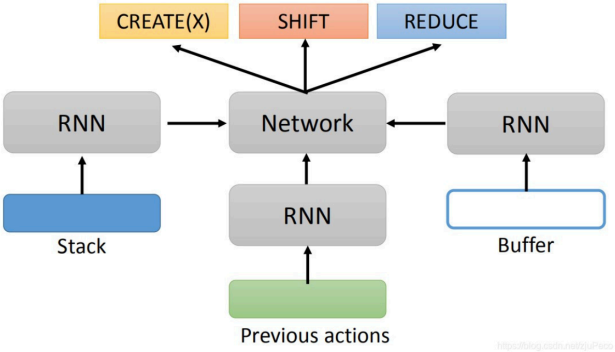


Figure is from https://speech.ee.ntu.edu.tw/~tlkagk/courses/DLHLP20/ParsingC%20(v2).pdf

Prepositional phrase attachment ambiguity

Scientists count whales from space

Scientists count whales from space

✓

Scientists count whales from space

✗

Figure is from CS224n

Ambiguity from constituent attachment

- Ambiguity arises when constituents can attach differently within a sentence.
- As sentence length (number of constituents) increases, possible attachments grow exponentially (a Catalan number $\frac{1}{n+1}C_{2n}^{n}$).

Coordination scope ambiguity

I invited the chefs and the waiters from the restaurant.

- I invited the chefs and the waiters from the restaurant.
- I invited the chefs and the waiters from the restaurant.

Relative clause （关系从句）attachment ambiguity

I know the friend of the girl who is wearing red clothes.

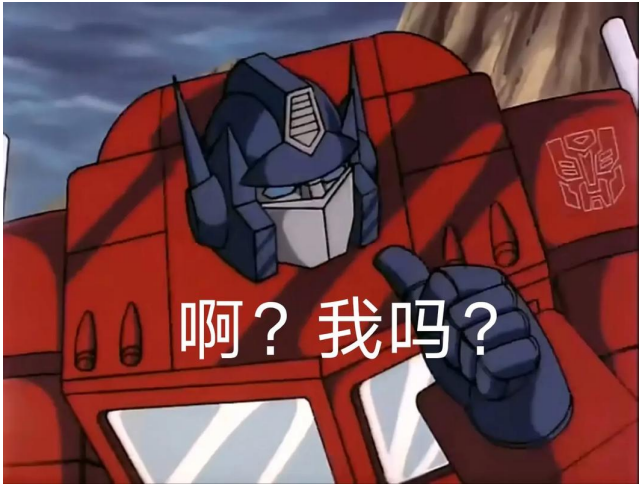
- I know the friend of the girl who is wearing red clothes.
- I know the friend of the girl who is wearing red clothes.

Adjectival/Adverbial modifier ambiguity

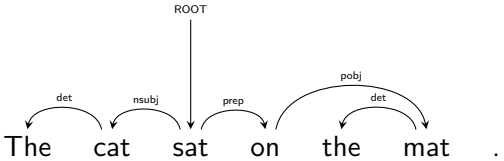
- Adjectival: She saw the man with the telescope.
- Adverbial: She almost drove her kids to school every day.

A Chinese example

- 新能源汽车人才能落户上海



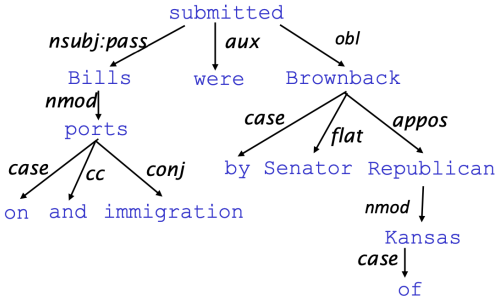
Dependency parsing



- nsubj: nominal subject (名词性主语); pobj: object of a preposition（介词宾语）
- sat $\rightarrow$ cat: sat is call a head (of cat); cat is called a dependent (of sat)

Dependency parsing

- Dependency parsing analyzes a sentence by breaking it down into a tree structure that represents the syntactic dependency structure between words (lexical items).



nmod:Nominal Modifier (名词修饰语); aux: auxiliary; obl: Oblique Nominal (间接名词修饰语); appos(同位语); flat (专有名词扁平关系); case (介词标记)....

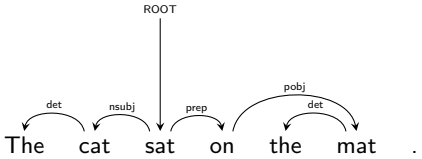
Figure is from CS 224n

Dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

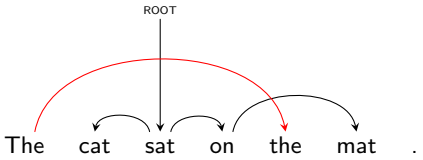
Figure is from Stanford Speech and Language Processing

Dependency Parsing



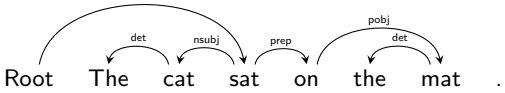
- Arrows indicate dependency relations: by convention, we draw them from head to dependent. (Some sources may use the reverse direction.)
- We introduce a fake root to ensure every word depends on one head.
- Only one word is related to ROOT
- No loop (sat $\rightarrow$ cat; cat $\rightarrow$ sat is not allowed)
- Those requirements make it a tree.

Projectivity



- A parse is projective if it contains no crossing dependency arcs.
- Constituency parses are always projective.
- However, dependency parsing typically allows for non-projective structures to represent displaced constituents more naturally.

Data and algorithms



- $S = w_0 w_1 \cdots w_L$, where $w_0 = \text{Root}$
- Dynamic programming and Minimum Spanning Tree also work for dependency parsing.
- We focus on the more effective and more efficient "Transition-based parsing"

Transition-Based Dependency Parsing

- Dependency parsing can be viewed as a state machine characterized by:
  - States: a stack $\mathcal{S}$, a buffer $\mathcal{B}$, and a set of dependency arcs $\mathcal{A}$.
  - Transitions: Actions that change the parser's state.
- Initialization:
  - Stack $\mathcal{S}$ initialized with [ROOT].
  - Buffer $\mathcal{B}$ initialized with the entire input sentence.
  - Dependency arc set $\mathcal{A}$ initialized as empty.
- Available actions:
  - Shift, Left-Arc, and Right-Arc.

Example: I am Groot

- $\mathcal{S}$: [ROOT], $\mathcal{B}$: I am Groot, $\mathcal{A}$: $\emptyset$.
- Shift $\Rightarrow$ $\mathcal{S}$: [ROOT, I], $\mathcal{B}$: am Groot, $\mathcal{A}$: $\emptyset$.
- Shift $\Rightarrow$ $\mathcal{S}$: [ROOT, I, am], $\mathcal{B}$: Groot, $\mathcal{A}$: $\emptyset$.
- Left-Arc$\Rightarrow$ $\mathcal{S}$: [ROOT, am], $\mathcal{B}$: Groot, $\mathcal{A}$: $\{\text{nsubj}(\text{am} \rightarrow \text{I})\}$.
- Shift$\Rightarrow$ $\mathcal{S}$: [ROOT, am, Groot], $\mathcal{B}$: $\emptyset$, $\mathcal{A}$: $\{\text{nsubj}(\text{am} \rightarrow \text{Is})\}$.
- Right-Arc $\Rightarrow$ $\mathcal{S}$: [ROOT, am], $\mathcal{B}$: $\emptyset$, $\mathcal{A}$: $\{\text{nsubj}(\text{am} \rightarrow \text{I}), \text{obj}(\text{am} \rightarrow \text{Groot})\}$.
- Right-Arc $\Rightarrow$ $\mathcal{S}$: [ROOT], $\mathcal{B}$: $\emptyset$, $\mathcal{A}$: $\{\text{nsubj}(\text{am} \rightarrow \text{I}), \text{obj}(\text{am} \rightarrow \text{Groot}), \text{root}(\text{ROOT} \rightarrow \text{am})\}$.

Basic greedy transition-based dependency parser

- Start: $\mathcal{S} = [\text{ROOT}], \mathcal{B} = w_1 \cdots w_L, \mathcal{A} = \emptyset$.
- Shift: $\mathcal{S}, w_i|\mathcal{B}, \mathcal{A} \quad \Rightarrow \quad \mathcal{S}|w_i, \mathcal{B}, \mathcal{A}$
- Left-Arc: $\mathcal{S}|w_i|w_j, \mathcal{B}, \mathcal{A} \quad \Rightarrow \quad \mathcal{S}|w_j, \mathcal{B}, \mathcal{A} \cup \{w_j \rightarrow w_i\}$
- Right-Arc: $\mathcal{S}|w_i|w_j, \mathcal{B}, \mathcal{A} \quad \Rightarrow \quad \mathcal{S}|w_i, \mathcal{B}, \mathcal{A} \cup \{w_i \rightarrow w_j\}$

Neural Greedy Dependency Parsing

- For a given sentence S, the input layer is a vector consists of three components $[\mathbf{x}^w, \mathbf{x}^t, \mathbf{x}^l]$:
  - Word representations (S_{word}): Concatenated embeddings of selected words at the top of the stack and buffer, forming a vector $\mathbf{x}^w$.
  - POS-tag embeddings (S_{tag}): Concatenated embeddings of POS-tags for selected words in S, forming a vector $\mathbf{x}^t$.
  - Arc-label embeddings (S_{label}): Concatenated embeddings of dependency arc-labels (e.g., `nsubj`) for selected words in S, forming a vector $\mathbf{x}^l$.
- The embeddings (denoted as matrices E^w, E^t, E^l) will also be updated as a trainable parameter.
- Initialization: E^w is pretrained by word2vec; E^t and E^l is randomly initialized.

Neural Network for Action Classification

- Concatenate embeddings into a single input vector:

$$\mathbf{x} = [\mathbf{x}^w, \mathbf{x}^t, \mathbf{x}^l]$$

- Apply a neural network classifier $f_{W,b}$ to predict the next parsing action:

$$f_{W,b}(\mathbf{x}) \in \mathbb{R}^d$$

  Here, d is the number of possible actions (e.g., Shift, Left-Arc, Right-Arc).
- Example action set:

$$f_{W,b}(\mathbf{x}) \in \{\text{Shift, Left-Arc, Right-Arc}\}$$

- Trainable parameters include:

$$W, \quad b, \quad E^w, \quad E^t, \quad E^l$$

Deep learning is powerful

- Extract embeddings not only for words but also for POS-tags and dependency labels.
- Significantly more powerful than linear classifiers.
- Endless wins for deep learning?

Evaluation of dependency parsing

True dependency			
position	word	head	label
1	the	2 (cat)	det
2	cat	3	nsubj
3	sat	0	root
4	on	3	prep
5	the	6	det
6	mat	4	pobj

Predicted dependency			
position	word	head	label
1	the	3	det
2	cat	3	pobj
3	sat	0	root
4	on	3	prep
5	the	6	det
6	mat	4	nsubj

- Unlabeled attachment scores (UAS): just count the correct heads (5/6 in this case).
- Labeled attachment scores (LAS): count both the correct heads and the labels (3/6 in this case).

Assignment 3 or 2-2: Dependency Parsing with PyTorch (TBD)

- Train a dependency parsing model using PyTorch.
- Tutorial guidance will be provided step-by-step (a "babysitting" style).
- Questions or concerns about difficulty? Let's discuss!

What's next

- RNN; sequence models; Attentions; pretrain; fine-tune; trustworthy LLMs.