Transition-based syntactic parsing

Transition-based syntactic parsing

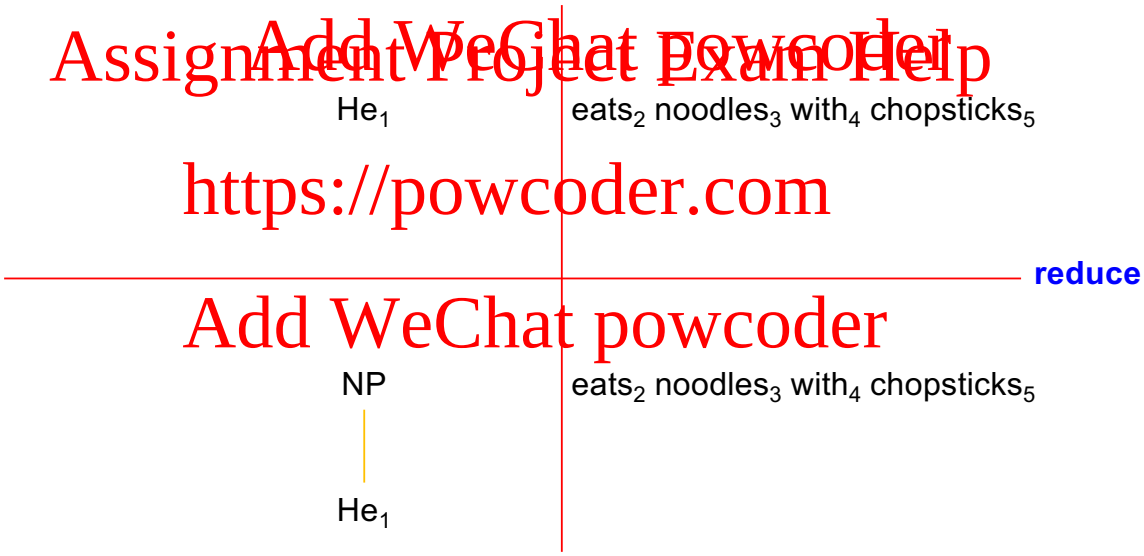
- Transition-based constituent parsing
- Transition-based dependency parsing
- Transition-based AMR parsing

Transition-based Constituent Parsing

- A transition-based constituent parsing model is a quadruple $C = (S, T, s_0, S_t)$ where:
  - S is a set of parser *states* or *configurations*,
  - T is a set of actions, e.g., *shift*, *reduce*,
  - s_0 is an initialization function that maps an input sentence into a unique *initial state*
  - $S_t \in S$ is a set of *terminal states*
- An action $t \in T$ is a transition function that transforms the current state into a new state
- A state $s \in S$ is defined as a tuple $s = (\alpha, \beta)$ where α is a *stack* that holds already constructed subtrees, and β is a *queue* which is used to store words that is yet to be processed.

Shift-Reduce: a Transition-based algorithm

Shift-Reduce: a Transition-based algorithm

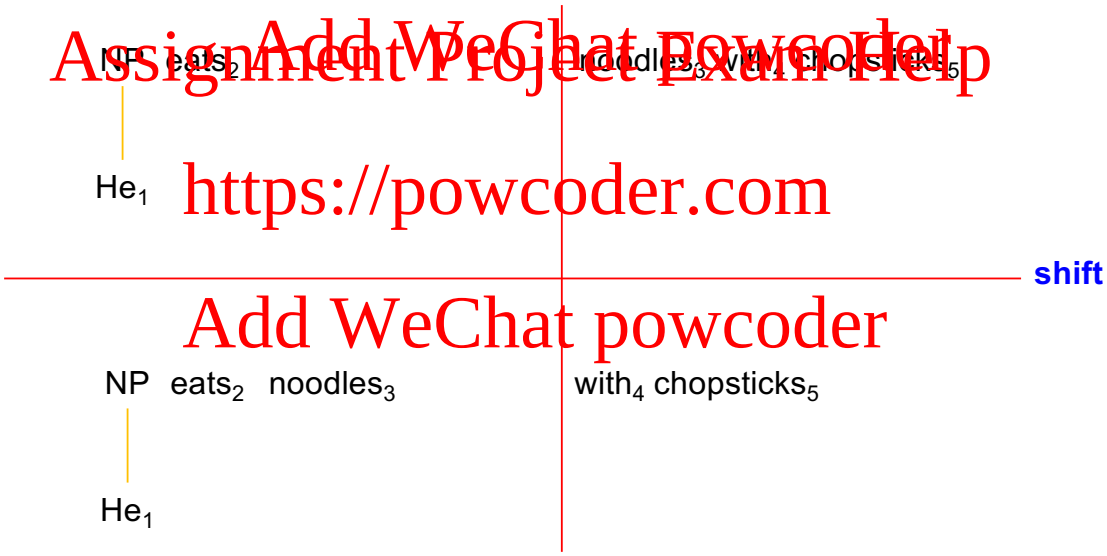


Shift-Reduce: a Transition-based algorithm

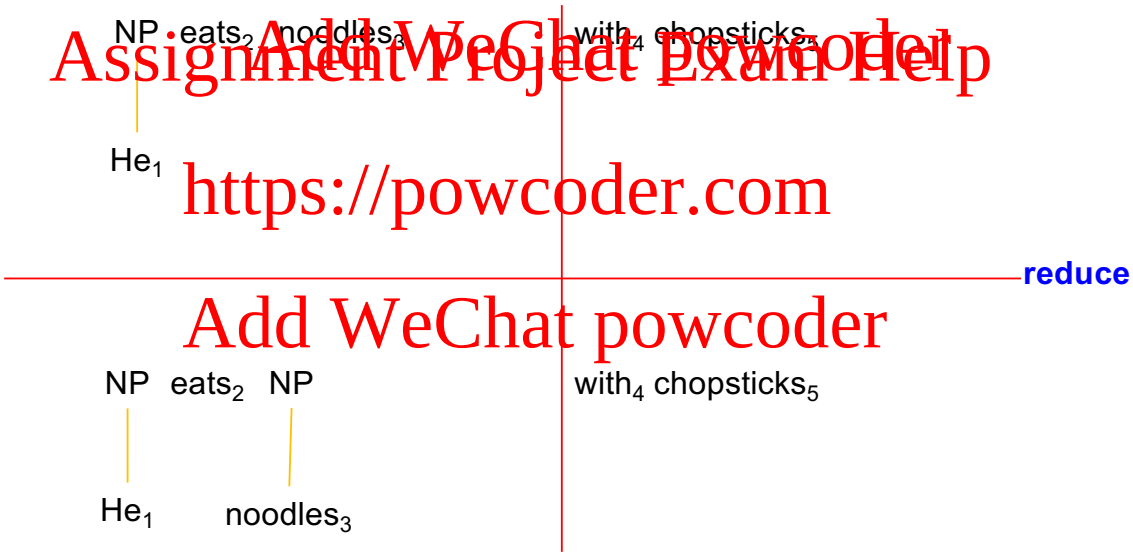
NP eats$_2$ noodles$_3$ with$_4$ chopsticks$_5$

He$_1$

Shift-Reduce: a Transition-based algorithm



Shift-Reduce: a Transition-based algorithm



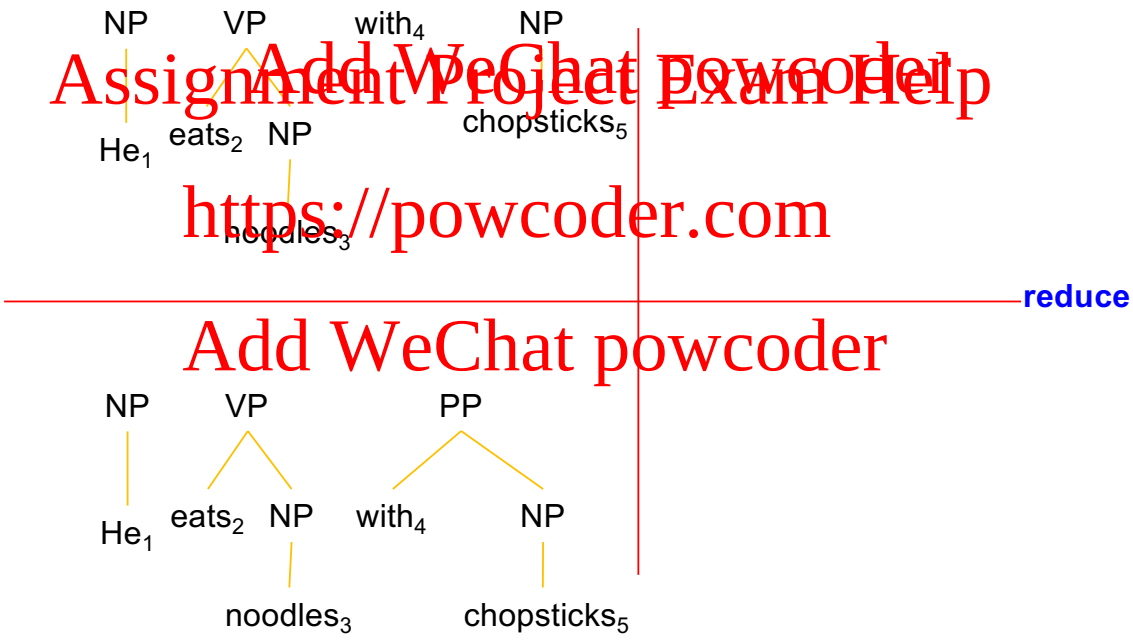
Shift-Reduce: a Transition-based algorithm

Shift-Reduce: a Transition-based algorithm

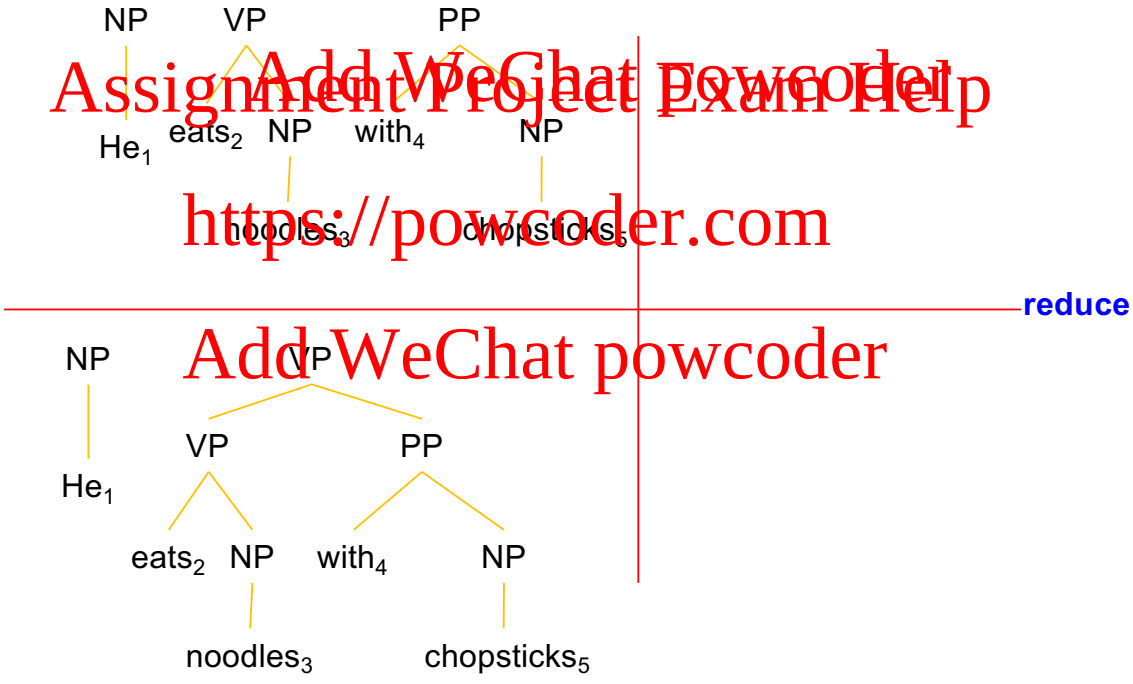
Shift-Reduce: a Transition-based algorithm

Shift-Reduce: a Transition-based algorithm

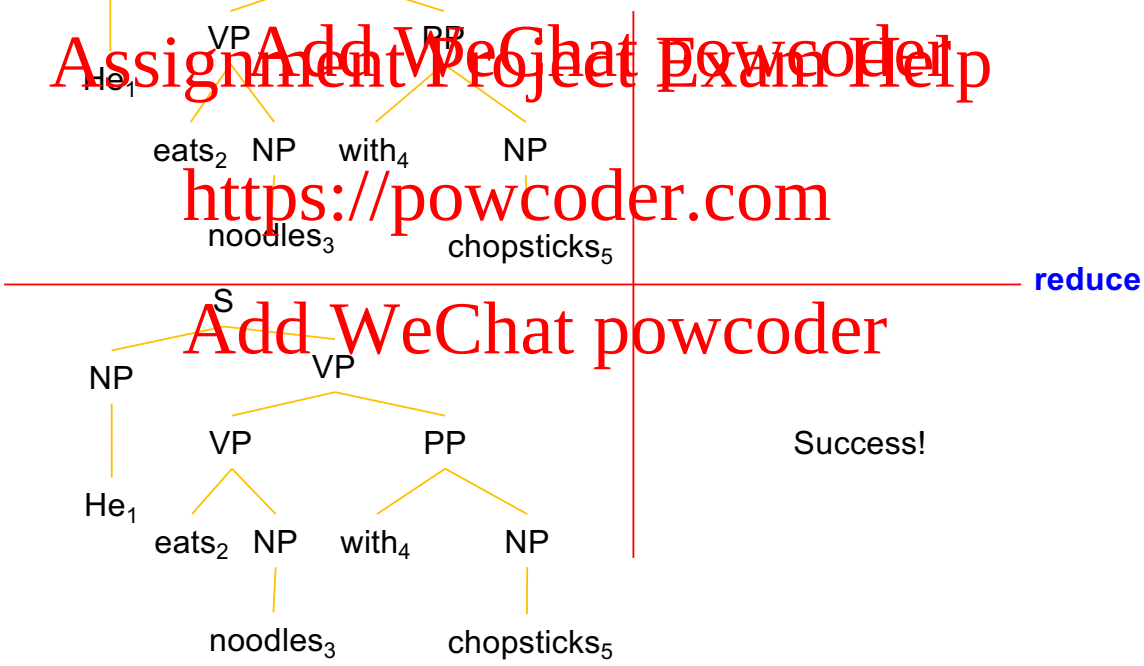
Shift-Reduce: a Transition-based algorithm



Shift-Reduce: a Transition-based algorithm



Shift-Reduce: a Transition-based algorithm



"Oracle"

- The oracle is a sequence of actions that lead to the correct parse of a sentence.
- When training a transition-based parsing model, we first map a gold parse tree onto an oracle sequence of actions
- We can learn a model by comparing the oracle to predicted action sequences and update the parameters of the model.

The Perceptron learning algorithm

1: ***Input***: Training examples (x_i, y_i)
2: ***Initialization***: Set $\boldsymbol{\theta} = \mathbf{0}$
3: **for** $t \leftarrow 1, T$ **do**
4: **for** $i \leftarrow 1, N$ **do**
5: $z_i \leftarrow \text{argmax}_{z \in GEN(x_i)} \boldsymbol{f}(x_i, z) \cdot \boldsymbol{\theta}$
6: **if** $z_i \neq y_i$ **then**
7: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{f}(x_i, y_i) - \boldsymbol{f}(x_i, z_i)$
8: ***Output***: Parameters $\boldsymbol{\theta}$

Lexcialized transition-based parsing actions

- Each action $t \in T$ is a transition action that transforms a state into a new state.
  - SHIFT(SH): remove the first word-POS pair from β, and pushes it onto the top of σ;
  - REDUCE-UNARY-X(ru-x): pop the top subtree from σ, construct a new unary node labeled with X for the subtree, and then push the new subtree back onto σ. The head of the new subtree is inherited from the child.
  - REDUCE-BINARY-L/R-X (rl/rl-x): pop the top two subtrees from σ, combine them into a new tree with a node labeled with X, then push the new subtree back onto σ. The left (L) and right (R) versions of the action indicate whether the head of the new subtree is inherited from its left or right child.
- A parsing state $s \in S$ is defined as a tuple $s = (\sigma, \beta)$, where σ is a stack that is maintained to hold the partial parsing structures that are already constructed and β is a queue used to store unprocessed input (typically word-POS tag pairs).

Updating feature weights

$$\nabla_\theta \left(\begin{bmatrix} p_0tc = N - NP \sim shift \\ p_0tc = N - NP \sim reduce \\ p_0wc = noodles - NP \sim shift \\ p_0wc = noodles - NP \sim reduce \\ p_1tc = V - V \sim shift \\ p_1tc = V - V \sim reduce \\ p_1wc = eats - V \sim shift \\ p_1wc = eats - V \sim reduce \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Notes: The feature $p_0tc = N - NP$ predicts a "shift" action when the oracle action should be "reduce".

Transition-based parsing features

Type	Feature Templates
unigrams	p_0tc, p_0wc, p_1tc, p_1wc, p_2tc
	p_2wc, p_3tc, p_3wc, q_0wt, q_1wt
	q_2wt, q_3wt, $p_{0l}wc$, $p_{0r}wc$
	$p_{0u}wc$, $p_{1l}wc$, $p_{1r}wc$, $p_{1u}wc$
bigrams	p_0wp_1w, p_0wp_1c, p_0cp_1w, p_0cp_1c
	p_0wq_0w, p_0wq_0t, p_0cq_0w, p_0cq_0t
	q_0wq_1w, q_0wq_1t, q_0tq_1w, q_0tq_1t
	p_1wq_0w, p_1wq_0t, p_1cq_0w, p_1cq_0t
trigrams	$p_0cs_1cs_2c$, $p_0wp_1cp_2c$, $p_0cp_1wq_0t$
	$p_0cp_1cp_2w$, $p_0cp_1cq_0t$, $p_0wp_1cq_0t$
	$p_0cp_1wq_0t$, $p_0cp_1cq_0w$

Baseline features, where p_i represents the i_{th} subtree in the stack σ and q_i denotes the i_{th} item in the queue β. w refers to the head word, t refers to the head POS, and c refers to the constituent label. p_{il} and p_{ir} refer to the left and right child for a binary subtree p_i, and p_{iu} refers to the child of a unary subtree p_i.

Feature vector

feature	count	feature	count
$p_0 tc$=N-NP˜shift	0	$p_0 tc$=N-NP˜reduce	1
$p_0 wc$=noodles-NP˜shift	0	$p_0 wc$=noodles-NP˜reduce	1
$p_1 tc$=V-V˜shift	0	$p_1 tc$=V-V˜reduce	1
$p_1 wc$=eats-V˜shift	0	$p_1 wc$=eats-V˜reduce	1
$p_{0u} wc$=noodles-N˜shift	0	$p_{0u} wc$=noodles-N˜reduce	1
$q_0 wt$=with-P˜shift	0	$q_0 wt$=with-P˜reduce	1
$q_1 wt$=chopsticks-N˜shift	0	$q_1 wt$=chopsticks-N˜reduce	1
...	...	...	...

Notes: Feature count for one configuration. The total count for a sentence will be a sum over all configurations in the derivation of the syntactic structure of the sentence

Beam Search

Input: A POS-tagged sentence, beam size k.
Output: A constituent parse tree

1: $beam_0 \leftarrow \{s_0\}$ ▷ initialization
2: $i \leftarrow 0$ ▷ step index
3: **loop**

4: $P \leftarrow \{\}$ ▷ a priority queue

5: **while** $beam_i$ is not empty **do**
6: $s \leftarrow \text{POP}(beam_i)$

7: **for** all possible $t \in T$ **do**
8: $s_{new} \leftarrow$ apply t to s

9: score s_{new}
10: insert s_{new} into P
11: $beam_{i+1} \leftarrow k$ best states of P
12: $s_{best} \leftarrow$ best state in $beam_{i+1}$
13: **if** $s_{best} \in S_t$ **then**
14: **return** s_{best}
15: $i \leftarrow i + 1$

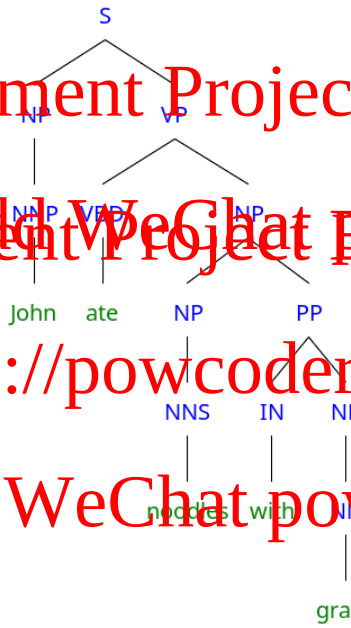
CFG based parsing vs transition-based parsing

- A transition-based parser scores the actions while a PCFG based parsing model scores the rules
- It's customary to use the beam search algorithm in transition-based parsing
- The transition-based approach can be easily applied to dependency parsing as well as graph-based semantic parsing
- Learning for transition-based parsing can be with done with basically any type of classifier, including neural network models

Constituent parsing as sequence-to-sequence learning

- When you have a hammer, everything looks like a nail. Can we apply sequence-to-sequence modeling to syntactic parsing?
- Our input is the raw sentence, which is a sequence, but our output is a constituent tree. How do we make the tree a sequence?
- We need to linearize the tree in such a way that it is fully reversible and can be mapped back to a tree.

Tree linearization



(S (NP NNP)NP (VP VBD (NP (NP NNS)NP (PP IN (NP NNS)NP)PP)NP)VP)S

Alternative linearizations possible, e.g., a sequence of shift reduction actions.

Tricks and tips

- Normalizing POS tags: replacing all POS tags with XX to further reduce the size of the output vocabulary
- Input inversing: Inversing the input sentence but not the output trees helps. Similar effects have been shown in MT: inversing the source sentence but not the target sentence
- There is no guarantee that the output will be a well-formed tree structure. If the model is well trained, most of the trees will be well-formed, but you need to do some post-processing or checking to make sure the trees are well-formed, with matching parentheses, etc.
- In practical systems, LSTM-RNNs are used rather than vanilla RNNs, which are easier to explain but don't work as well in practice

BLEU: a metric for measuring similarity of sequences

- **BLEU** stands for Bilingual Evaluation Understudy, and it is the most popular MT evaluation metric.

- N-gram overlap between machine translation output and reference translation

- Compute precision for n-grams of size 1 to 4
- Add brevity penalty (for too short translations)

$$\text{BLEU} = \min\left(1, \exp\left(1 - \frac{\textit{reference-length}}{\textit{output-length}}\right)\right) \left(\prod_{i=1}^{4} \textit{precision}_i\right)^{\frac{1}{4}}$$

- In logarithms: $\exp \frac{1}{N} \sum_{n=1}^{N} \log P_n$
- Typically computed over the entire corpus, not single sentences

Example

Metric	System A	System B
precision (1gram)	3/6	6/6
precision (2gram)	1/5	4/5
precision (3gram)	0/4	2/4
precision (4gram)	0/3	1/3
brevity penalty	6/7	6/7
BLEU	0%	52%

Fine prints of BLEU

- To avoid computing $\log 0$, all precisions are smoothed to ensure that they are positive.

- Each n-gram in the reference can be used at most once, so that *to to to to to to* does not achieve $P_1 = 1$ against the reference *to be or not to be*

- **Brevity penality** is applied to translations that are shorter than the reference.

- Normalization often performed on reference and hypothesis translations to get better match.