Syntactic parsing approaches

- Grammar-based approach with CKY decoding
  - PCFG, a generative approach that extends the Naive Bayes Model
    - Lexicalization, parent annotation
  - Discrminative approaches: linear and neural models
    - Perceptron and CRF training with discrete features
    - Neural models
- Transition-based approach: the shift-reduce algorithm with greedy or beam search
  - Linear models with discrete features – Perceptron, Conditional Random fields
  - Non-linear (neural) models
- Thinking out of the box: a sequence-to-sequence approach to syntactic parsing

Learning PCFGs

Parameters in probabilistic context-free grammars can be estimated by relative frequency, as with HMMs:

$$\psi(X \to \alpha) = \log P(X \to \alpha)$$

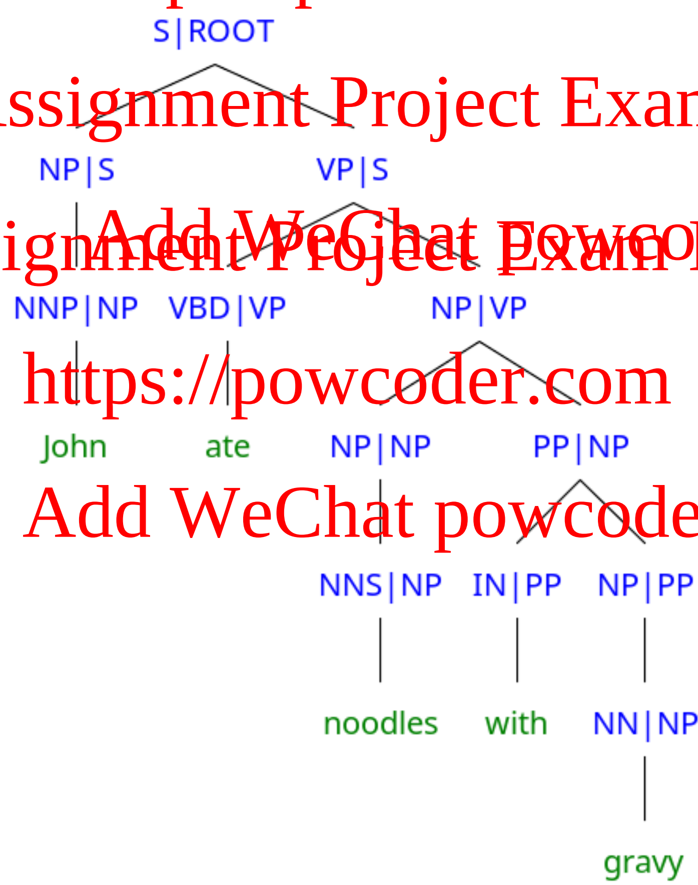
$$\hat{P}(X \to \alpha) = \frac{count(X \to \alpha)}{count(X)}$$

E.g., the probability of the production NP $\to$ DET NN is the corpus count of this production, divided by the count of the non-terminal NP. This applies to terminals as well.

Grammar Refinement

- Grammars extracted from treebanks (e.g., the Penn TreeBank) are often sensitive to ambiguities in the parses, even with the weighted productions
- There are various attempt to augment with the vanilla PCFG with more expressive productions
  - Parent annotation
  - Lexicalization

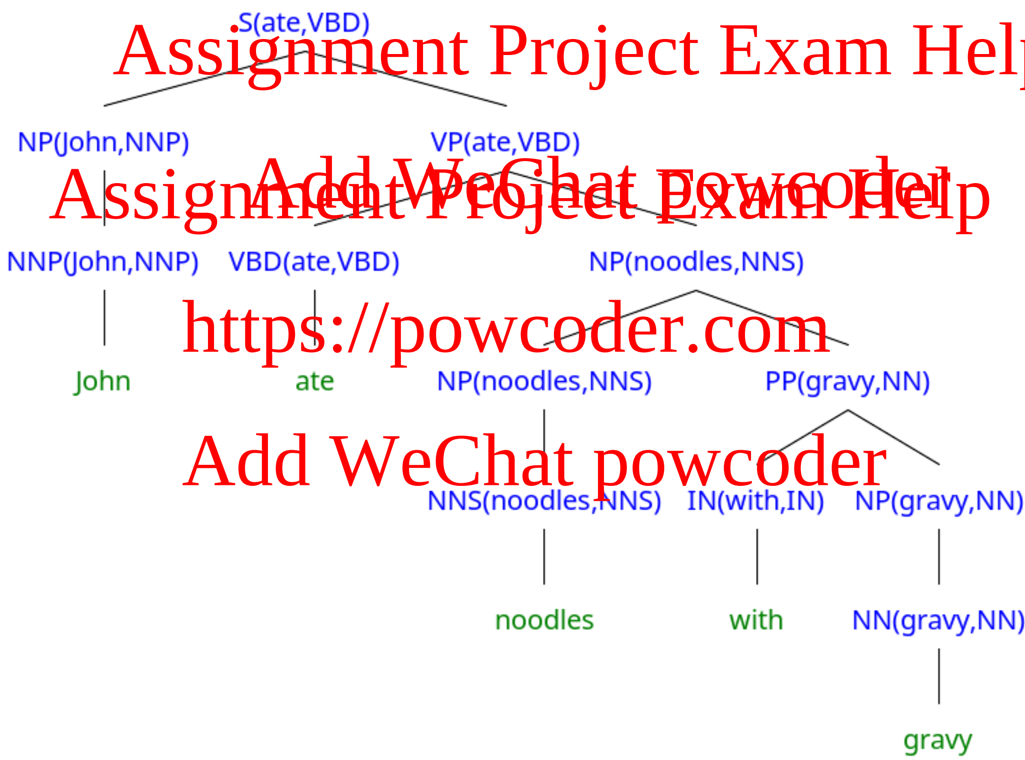
Parent annotation



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Lexicalized CFGs

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Discriminative approaches with discrete features

- The scores for each production can be computed as an inner product of features and their weights,

$$\psi(X \to \alpha, (i, j, k)) = \boldsymbol{\theta} \cdot \boldsymbol{f}(X, \alpha, (i, j, k), \boldsymbol{w})$$

Assignment Project Exam Help

  where the feature vector f is a function of the left-hand side X, the right-hand side α, the anchor indices (i, j, k), and the input $\boldsymbol{w}$.

https://powcoder.com
Add WeChat powcoder

- The basic feature $\boldsymbol{f}(X, \alpha, (i, j, k)) = \{(X, \alpha)\}$ encodes only the identify of the production itself and is therefore as expressive as PCFG trained discriminatively.
- Other features include the words in the beginning and at the end of the span w_i, w_{j+1}, the word at the split point w_{k+1}, etc.

Perceptron training

- Perceptron training for parsing is very similar to that of sequence labeling
- The feature vector for a sentence-tree pair decomposes to the sum of local features

$$\boldsymbol{f}(\tau, \boldsymbol{w}^{(i)}) = \sum_{(X \to \alpha, (i,j,k)) \in \tau} \boldsymbol{f}(X, \alpha, (i,j,k), \boldsymbol{w}^{(i)})$$

- Find the tree with the highest score based on the current model

$$\hat{\tau} = \operatorname*{argmax}_{\tau \in \mathcal{T}(\boldsymbol{w})} \boldsymbol{\theta} \cdot \boldsymbol{f}(\tau, \boldsymbol{w}^{(i)})$$

- Update the feature weights

$$\boldsymbol{\theta} \leftarrow \boldsymbol{f}(\tau^{(i)}, \boldsymbol{w}^{(i)}) - \boldsymbol{f}(\hat{\tau}, \boldsymbol{w}^{(i)})$$

CRF parsing

- The score of a derivation $\Psi(\tau)$ can be converted into a probability by normalizing over all possible derivations:

$$P(\tau|\boldsymbol{w}) = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\boldsymbol{w})} \exp \Psi(\tau')}$$

- Using this probability, a WCFG can be trained by maximizing the conditional log-likelihood of a labeled corpus.

CRF training

- Just as in logistic regression and the conditional random field over sequences, the gradient of the conditional log-likelihood is the difference between the observed and expected counts of each feature.
- The expectation $E_{\tau|\boldsymbol{w}}[\boldsymbol{f}(\tau, \boldsymbol{w}^{(i)}); \boldsymbol{\theta}]$ requires summing over all possible parses, and computing the marginal probabilities of anchored productions, $P(X \to \alpha, (i, j, k)|\boldsymbol{w})$.
- In CRF sequence labeling, marginal probabilities of over tag bigrams are computed by the two-pass **forward-backward algorithm**. The analogue for context-free grammars is the **inside-outside** algorithm, in which marginal probabilities are computed from terms generated by an upward and downward pass over the parsing chart.

Neural context-free grammars

- Neural networks can be applied to parsing by representing each span with a dense numerical vector. For example, the anchor (i, j, k) and sentence $\boldsymbol{w}$ can be associated with a column vector:

$$\boldsymbol{v}_{(i,j,k)} = [\boldsymbol{u}_{w_{i-1}}; \boldsymbol{u}_{w_i}; \boldsymbol{u}_{w_{j-1}}; \boldsymbol{u}_{w_j}; \boldsymbol{u}_{w_{k-1}}; \boldsymbol{u}_{w_k}]$$

- The vector can be fed into a feed-forward neural net:

$$\tilde{\boldsymbol{v}}_{(i,j,k)} = \text{FeedForward}(\boldsymbol{v}_{(i,j,k)})$$

- The score of a constituent can be computed with a weight matrix

$$\psi(X \to \alpha, (i, j, k)) = \tilde{\boldsymbol{v}}_{(i,j,k)}^{\top} \boldsymbol{\Theta} \boldsymbol{f}(X \to \alpha)$$

Parsing with the Transformer-based encoder-decoder framework

- Using the contextualized embeddings trained with Transformer to compute the vectorial representation for constituents, and then efficiently search for the syntactic tree with the highest score with the CKY algorithm.

Score the candidate trees and search for the optimal one by the model

- Assign a real-valued score $s(T)$ to each tree T, which decomposes as

Assignment Project Exam Help

$$s(T) = \sum_{(i,j,l) \in T} s(i,j,l)$$

https://powcoder.com

Add WeChat powcoder

  where $s(i,j,l)$ is a real-valued score for a constituent between position i and j with the label l

- Given the scores of constituent, the model-optimal tree can be found with the CKY algorithm.

Train the model with a max-margin objective

- Given the correct tree $T\star$, the model is trained to satisfy the margin constraints

$$s(T\star) \geq s(T) + \Delta(T, T\star)$$

for all trees T by minimizing the hinge loss

$$max\left(0, \max_{T \neq T\star}[s(T) + \Delta(T, T\star)] - s(T\star)\right)$$

- Δ is the Hamming loss on labeled spans, and the tree that violates the most constraints is selected for purposes of updating parameters.

Encoder

- The encoder portion of our model is split into two parts:
  - A word-based portion that assigns a context-aware vector representation $\boldsymbol{y}_t$ to each sentence with Transformer (self-attention followed by position-wise feedforward neural network)

    - The input is the sum of a word embedding, position embedding, and POS embedding
  - A chart portion that combines the vectors $\boldsymbol{y}_t$ to generate the scores for each span $s(i, j, l)$.

    - Span score:

$$s(i, j, \cdot) = \Theta_2 \text{ReLU}(\text{LayerNorm}(\Theta_1 \boldsymbol{v} + \boldsymbol{b}_1) + \boldsymbol{b}_2$$

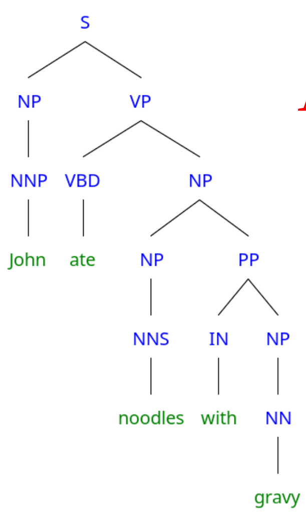
    - The input vector $\boldsymbol{v}$ combines the word-based vectors:

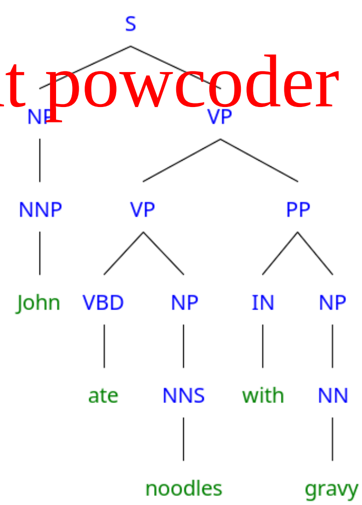
$$\boldsymbol{v} = [\overleftarrow{\boldsymbol{y}}_j - \overleftarrow{\boldsymbol{y}}_i ; \overrightarrow{\boldsymbol{y}}_{j+1} - \overrightarrow{\boldsymbol{y}}_{i+1}]$$

    where $\overleftarrow{\boldsymbol{y}}_t$ and $\overrightarrow{\boldsymbol{y}}_t$ are the first and second half of the $\boldsymbol{y}_t$ respectively

Parser evaluation

- **Precision**: the fraction of constituents in the system parse that match a constituent in the reference parse.
- **Recall**: the fraction of constituents in the reference parse that match a constituent in the system parse.
- **labeled** vs **unlabeded** precision and recall: In labeled precision and recall, the system must also match the phrase type for each constituent; in unlabeled precision and recall, it is only required to match the constituent structure.





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 quaduple $C = (S, T, s_0, S_t)$ 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

	stack	queue	
current state		He_1 $eats_2$ $noodles_3$ $with_4$ $chopsticks_5$	
			shift
new state	He_1	$eats_2$ $noodles_3$ $with_4$ $chopsticks_5$	

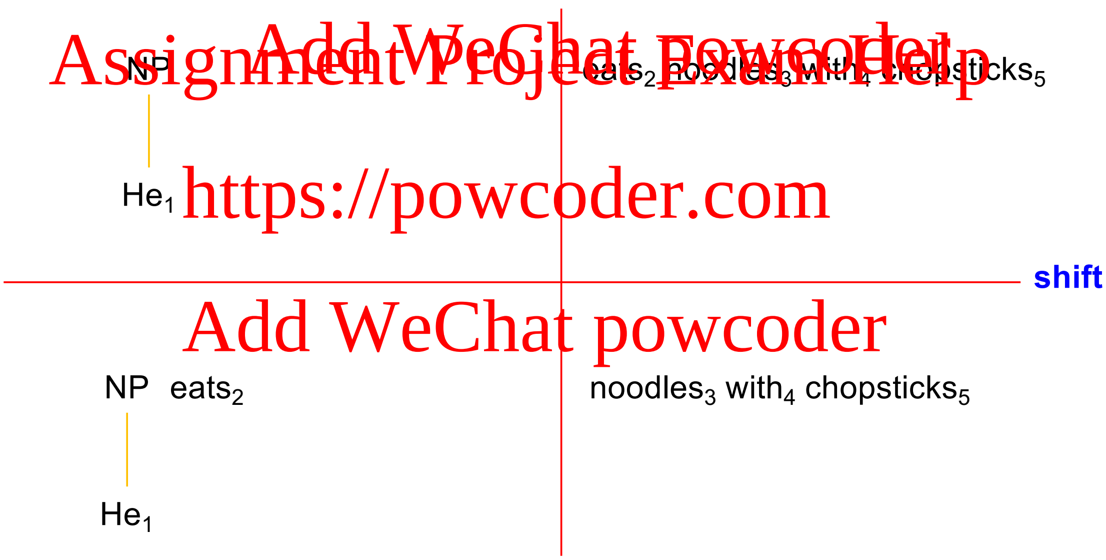
Shift-Reduce: a Transition-based algorithm

He_1	eats_2 noodles_3 with_4 chopsticks_5

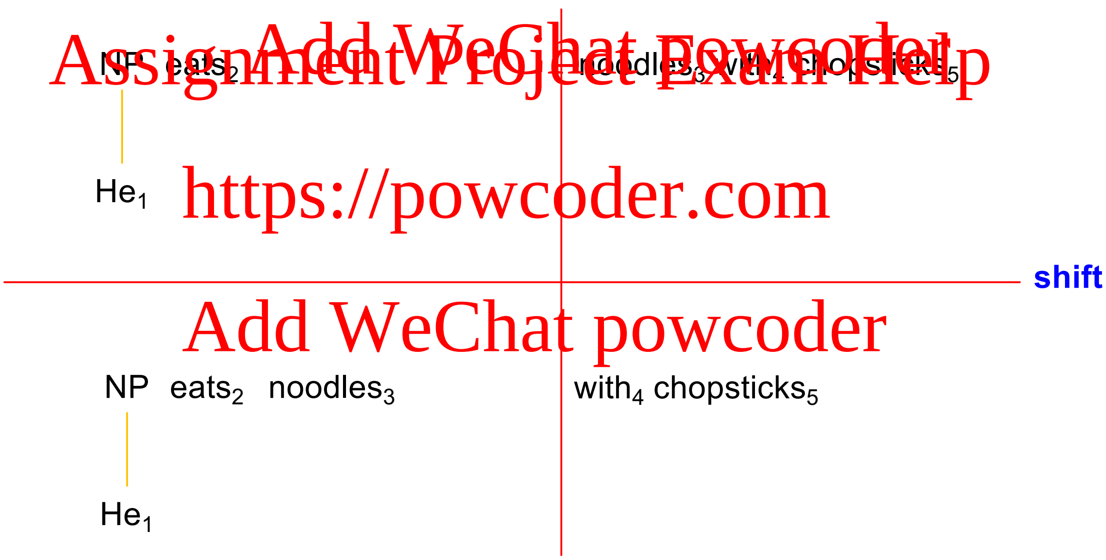
reduce

NP	eats_2 noodles_3 with_4 chopsticks_5
He_1	

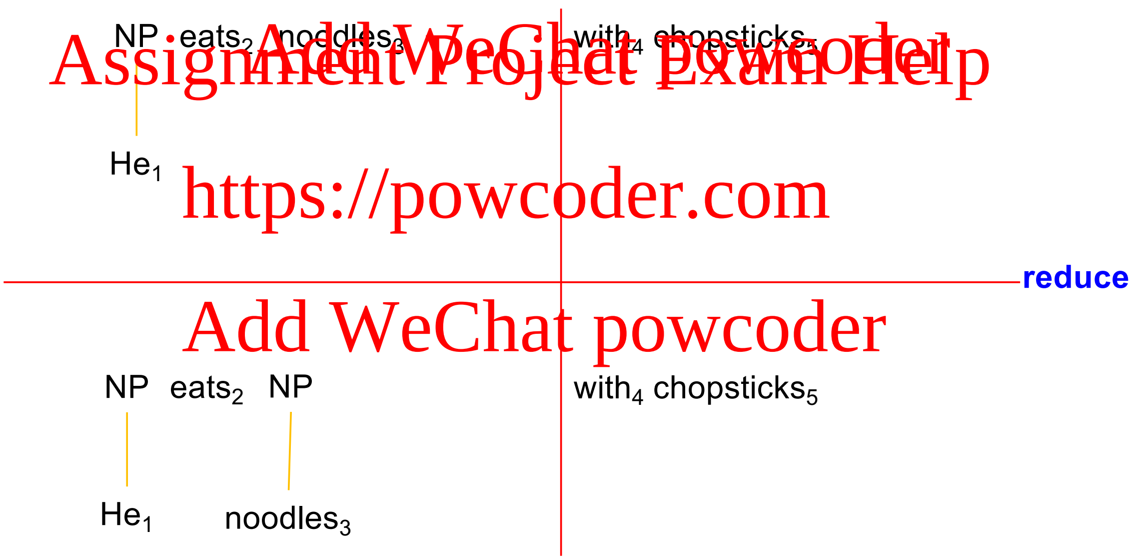
Shift-Reduce: a Transition-based algorithm



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

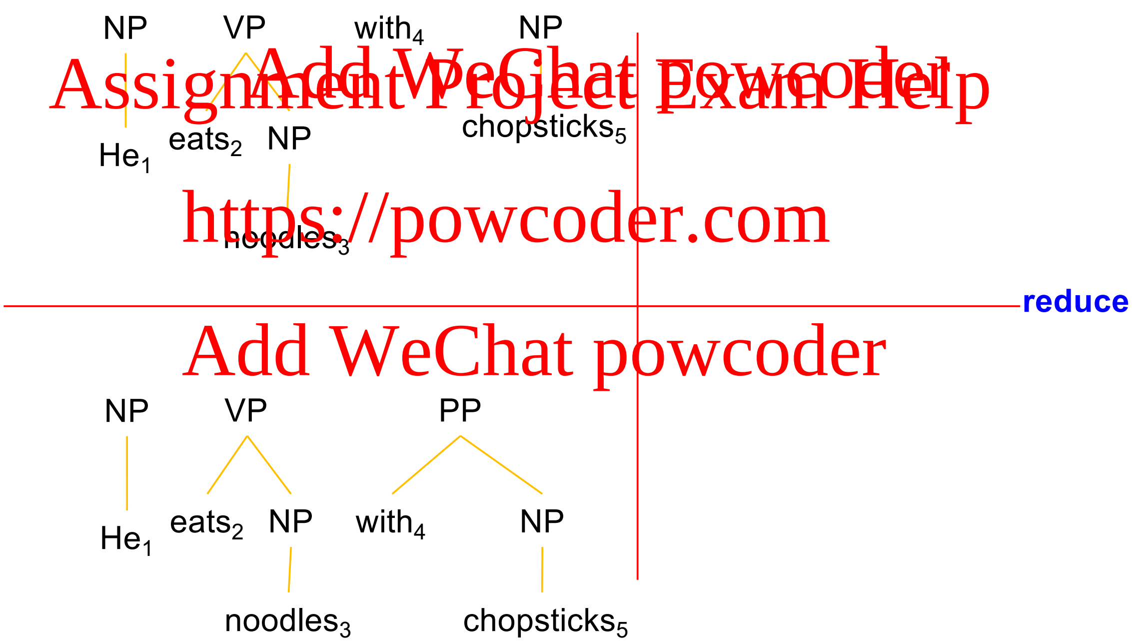
Shift-Reduce: a Transition-based algorithm

Assignment Project Exam Help

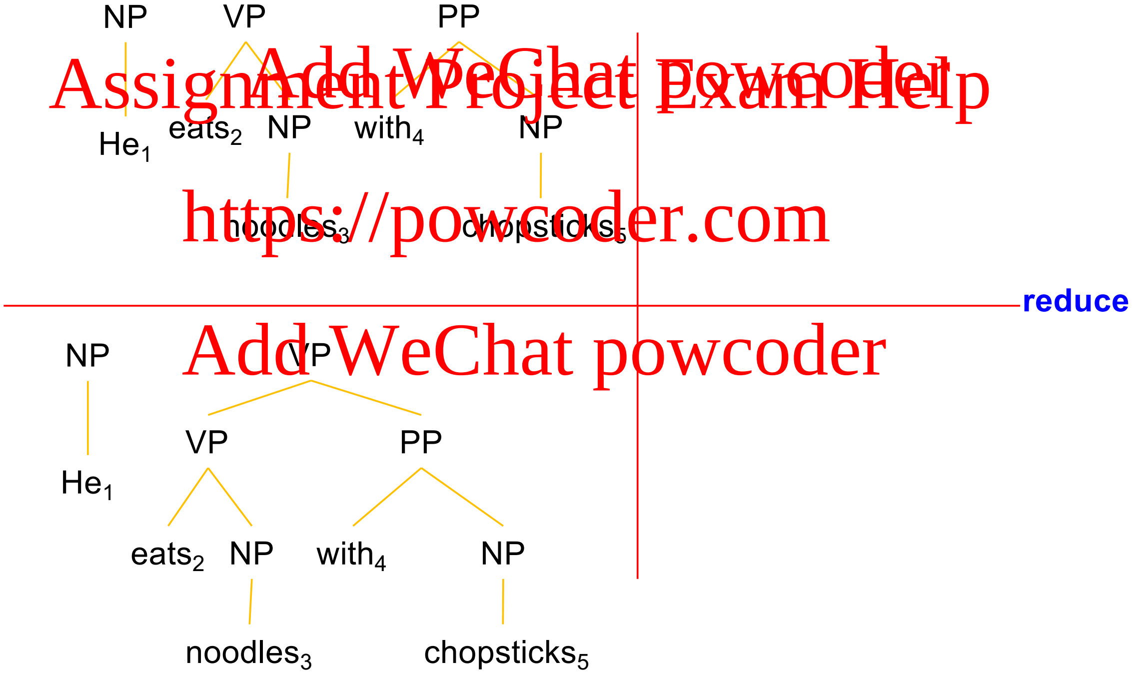
https://powcoder.com

Add WeChat powcoder

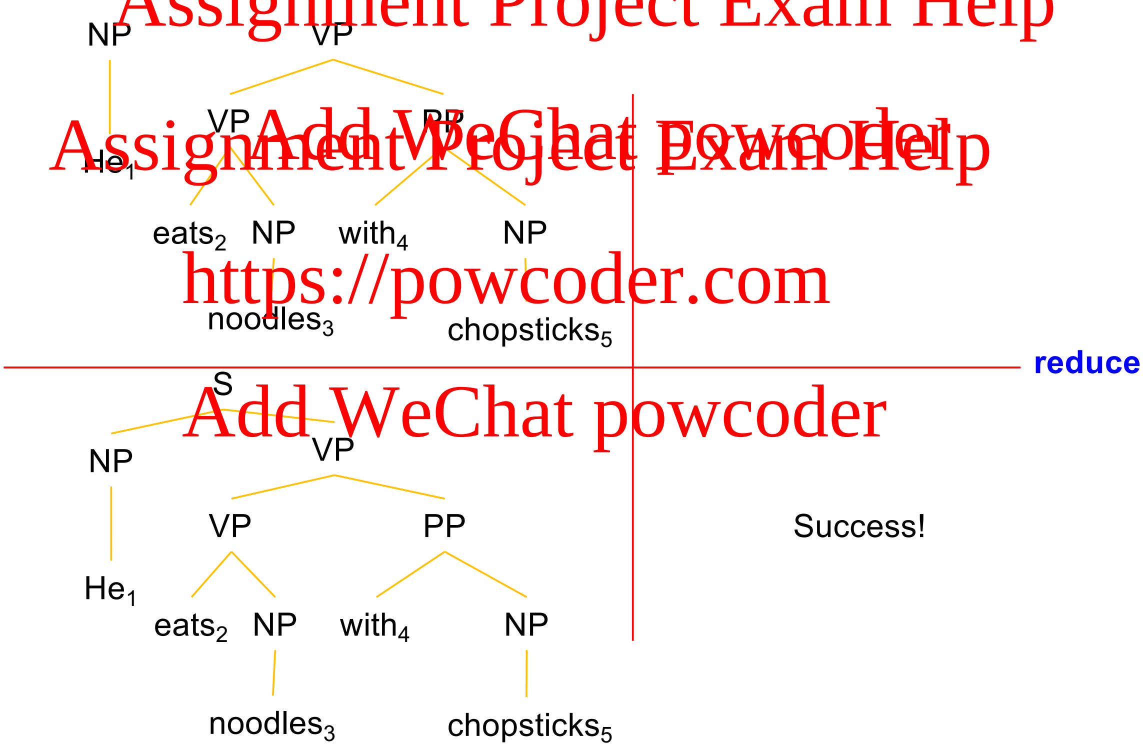
Shift-Reduce: a Transition-based algorithm



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

Input: Training examples (x_i, y_i)
Initialization: Set $\boldsymbol{\theta} = 0$
for $t \leftarrow 1, T$ **do**
  for $i \leftarrow 1, N$ **do**
    $z_i \leftarrow \operatorname{argmax}_{z \in GEN(x_i)} \boldsymbol{f}(x_i, z) \cdot \boldsymbol{\theta}$
    if $z_i \neq y_i$ **then**
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{f}(x_i, y_i) - \boldsymbol{f}(x_i, z_i)$
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).

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Updating feature weights

$$\nabla_{\theta}\left(\begin{bmatrix} p_0 tc = N - NP \sim shift \\ p_0 tc = N - NP \sim reduce \\ p_0 wc = noodles - NP \sim shift \\ p_0 wc = noodles - NP \sim reduce \\ p_1 tc = V - V \sim shift \\ p_1 tc = V - V \sim reduce \\ p_1 wc = eats - V \sim shift \\ p_1 wc = eats - V \sim reduce \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Notes: The feature $p_0 tc = N - NP$ 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	$p_0cp_1cp_2c$, $p_0wp_1cp_2c$, $p_0cp_1wq_0t$ $p_0cp_1cp_2w$, $p_0cp_1cq_0t$, $p_0wp_1cq_0t$ $p_0cp_1wq_0t$, $p_0cp_1cq_0w$

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

$beam_0 \leftarrow \{s_0\}$ ▷ initialization
$i \leftarrow 0$ ▷ step index
loop
    $P \leftarrow \{\}$ ▷ a priority queue
    while $beam_i$ is not empty **do**
        $s \leftarrow \text{POP}(beam_i)$
        for all possible $t \in T$ **do**
            $s_{new} \leftarrow$ apply t to s
            score s_{new}
            insert s_{new} into P
    $beam_{i+1} \leftarrow k$ best states of P
    $s_{best} \leftarrow$ best state in $beam_{i+1}$
    if $s_{best} \in S_t$ **then**
        return s_{best}
    $i \leftarrow i + 1$

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