## Syntactic parsing approaches



- Grammar-based approach with CKY decoding
  - PCFG, a generative approach that extends the Naive Bayes Model
    - Lexicalization, parent annotation
  - Discriminative approaches: linear and ne
    - P
    - N
- Transition-based approach: the shift ith greedy or beam search
  - Linear models with discrete features – ional 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:



$$P(X \to \alpha) = \frac{c}{\text{———————}}$$



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. TreeBank) are often sensitive to ambiguiti even with the w
- There are vari with more expressive productions
  - Parent annotation
  - Lexicalization



FG

Parent annotation

# Lexicalized CFGs

## Discriminative approaches with discrete features



- The scores for eac
  product of features and their weights,

$$s(X \rightarrow \alpha, (i,j,k)) = \boldsymbol{w} \cdot \boldsymbol{f}(X \rightarrow \alpha, (i,j,k))$$

  where the feat and side $X$, the right-h or indices $(i,j,k)$, and the input $\boldsymbol{w}$.



- The basic feature $\boldsymbol{f}(X, \alpha, (i,j,k$ 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 trai
sequence labeling
- The feature vector for a sentence-tree pair decomposes to the sum of local features



$$\boldsymbol{f}(\tau, \boldsymbol{w}^{(i)}) = \qquad \boldsymbol{f} \qquad \boldsymbol{w}^{(i)})$$



- Find the tree with the highest score based o
model

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

- 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 c probability by normalizing over all possible d



$$\underline{\qquad\qquad}$$
$$\tau' \in$$



- Using this probability, a WCFG can be tra 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 ex
each feature
- The expectat
ming over all possible pars
ties of anchored pr
.
- In CRF sequence labeling, marginal pro
bigrams are computed by the two-pass
**-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 network
  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:



$$\boldsymbol{v} \qquad\qquad _{k-1}; \boldsymbol{u}_{w_k}]$$



- The vector ca



$$\tilde{v}_{(i,j,k)} = \text{Tree} \qquad \text{rd}($$

- 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 cont
  Transform
  constituent
  with the highest score with the CKY algori



for
ic tree

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



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



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

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

- 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





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

- $\Delta$ 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 porti
  - A word-based portion that assigns a context-aware vector representation $\boldsymbol{y}_t$ to each sentence with Transformer (self-attention followed by position-wi al network)
    - The input is the sum of a word embedding e
  - A chart p $\boldsymbol{y}_t$ to generate the scores fo
    - Span score:

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

    - 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 fra
  that match a const
- **Recall**: the fraction of constituents in the reference parse that match a constituent in the system parse.
- **labeled** vs **unlabeded** precision and r
  precision and recall, the system must also mat
  type for each co ll, it
  is only require

Assignment Project

Tra

## Transition-based syntactic parsing



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

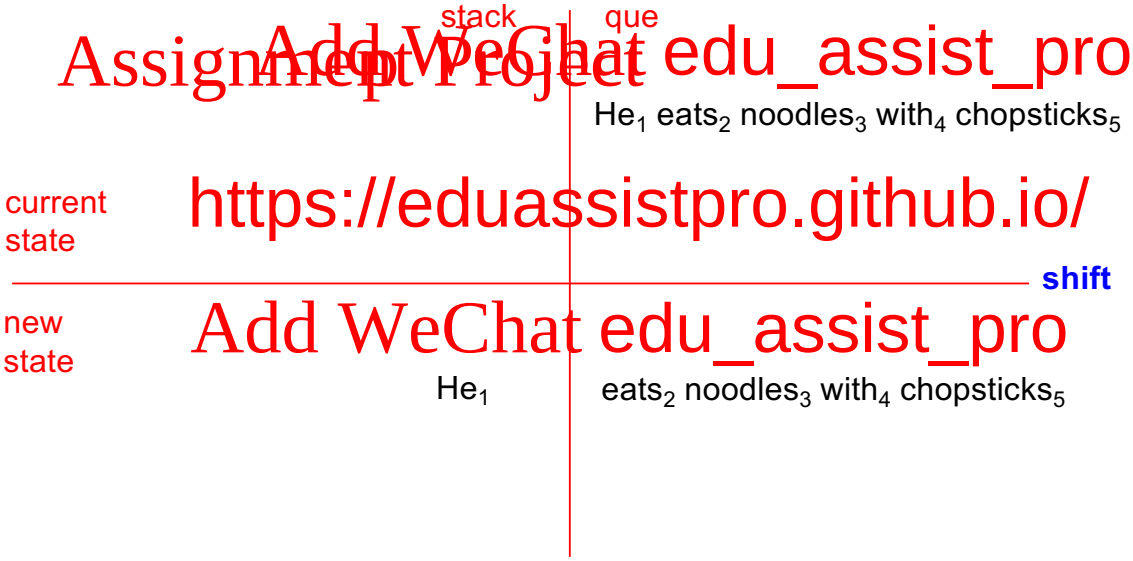
## Transition-based Constituent Parsing

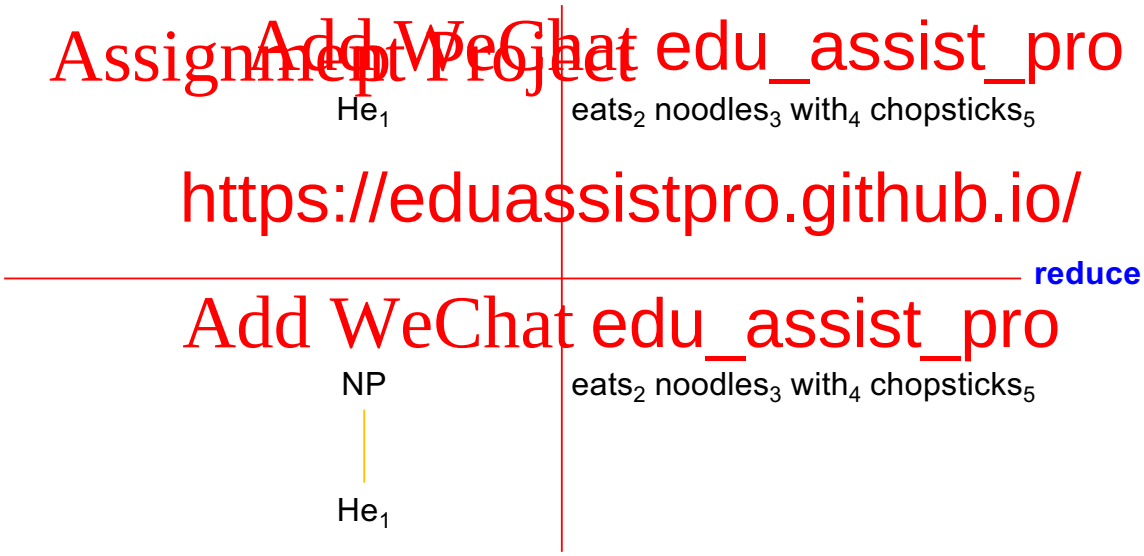


- A transition-based constituent parsing model is a quadruple $C = (S, T, s_0, S_t)$ where:
  - $S$ is a set of parser *states* or *con*
  - $T$ is a set of actions, e.g., *shift*,
  - $s_0$ is an in ce into a un
  - $S_t \in S$
- An action $t \in T$ is a transition funct s the current state into a new state
- A state $s \in S$ is defined as a tuple $s = (\alpha, \beta)$ where $\alpha$ is a *stack* that holds already constructed subtrees, and $\beta$ is a *queue* which is used to store words that is yet to be processed.

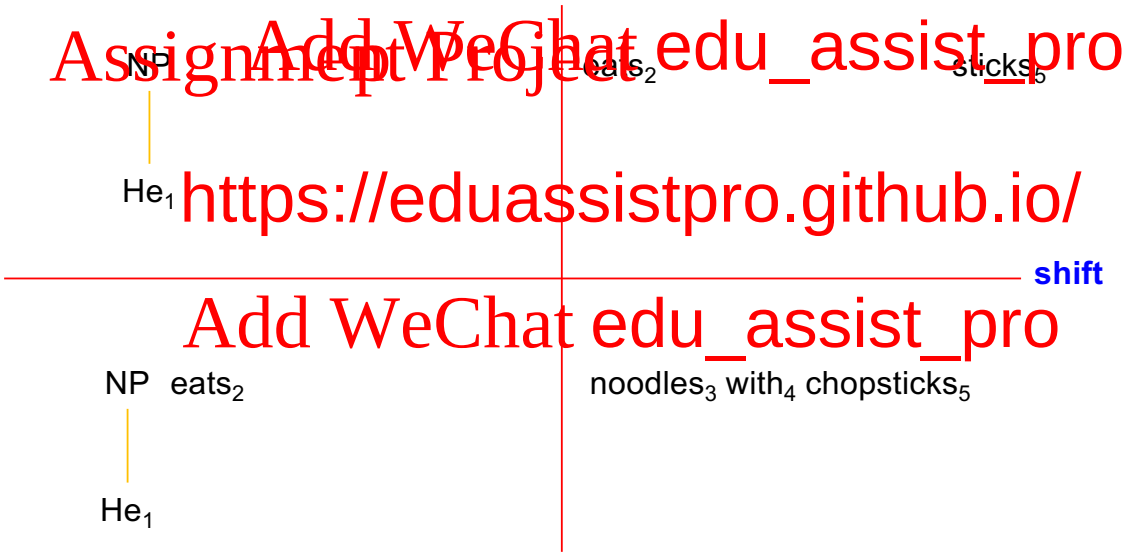
Shift-Reduce







Shift-Red gorithm



Shift-Redu



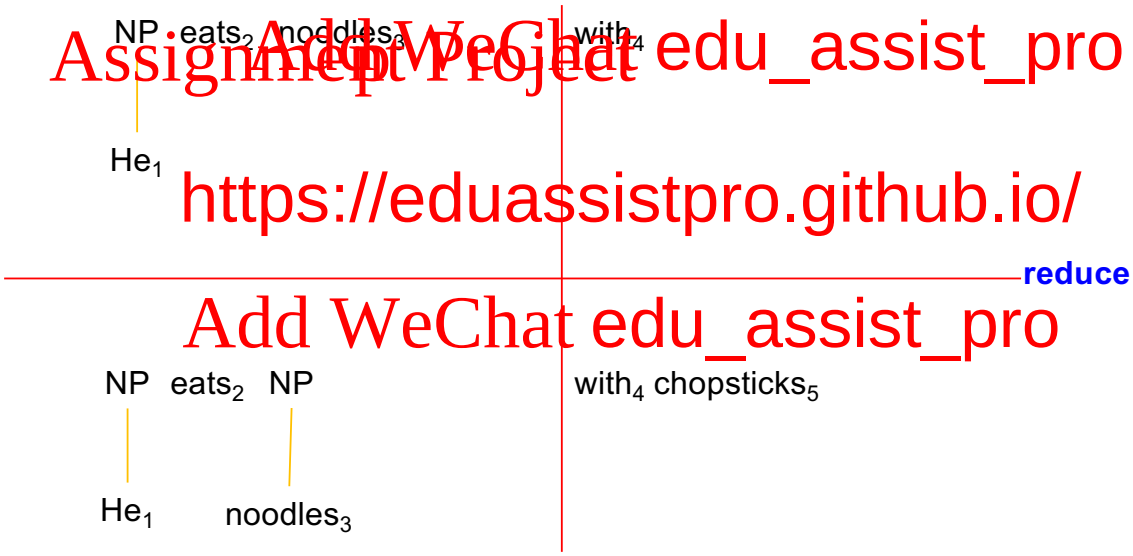
orithm



Shift-Redu https://eduassistpro.github.io/ orithm

Assignment Project Exam Help

Assignment Project Add WeChat edu_assist_pro

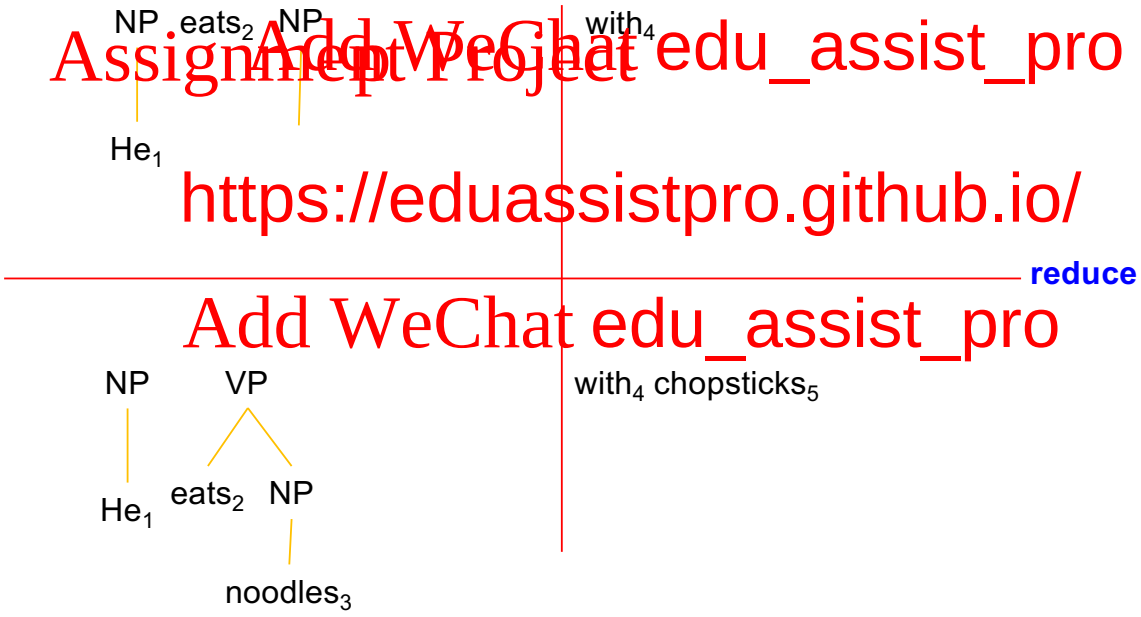



https://eduassistpro.github.io/

Add WeChat edu_assist_pro

Shift-Redu https://eduassistpro.github.io/ orithm

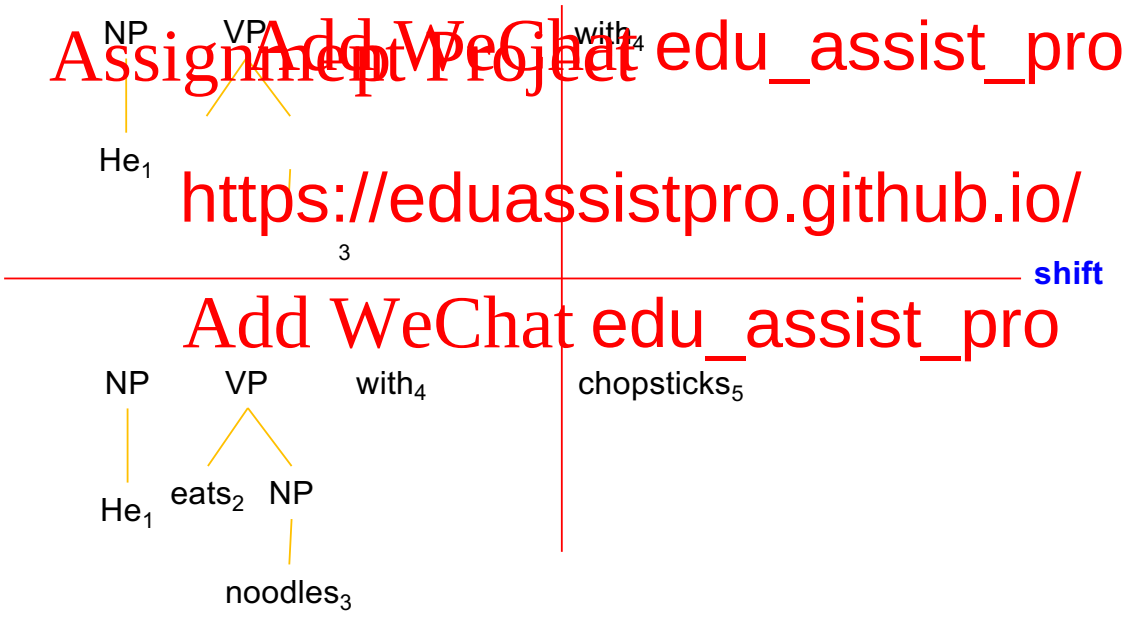




Shift-Redu https://eduassistpro.github.io/ orithm

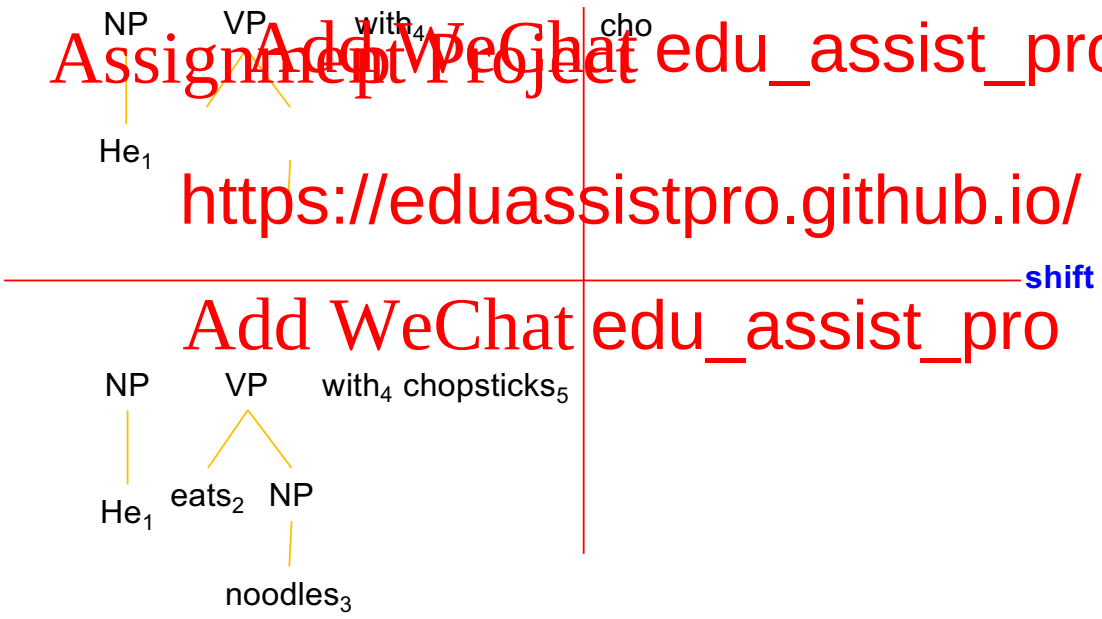
Assignment Project Exam Help

Assignment Project Add WeChat edu_assist_pro



Shift-Redu orithm



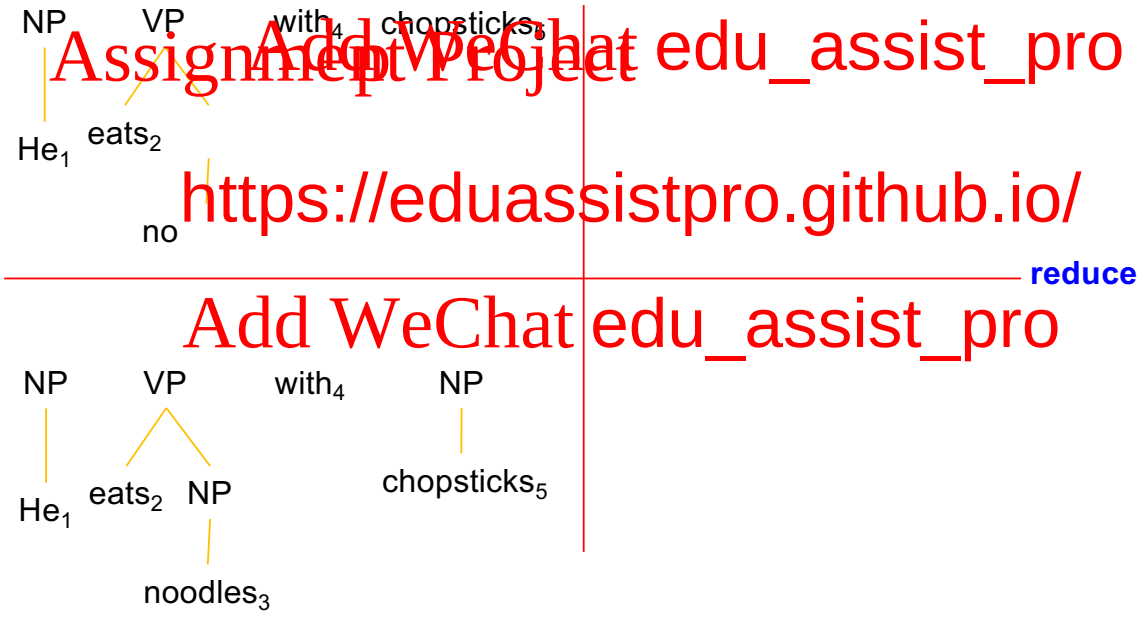


Shift-Redu



orithm



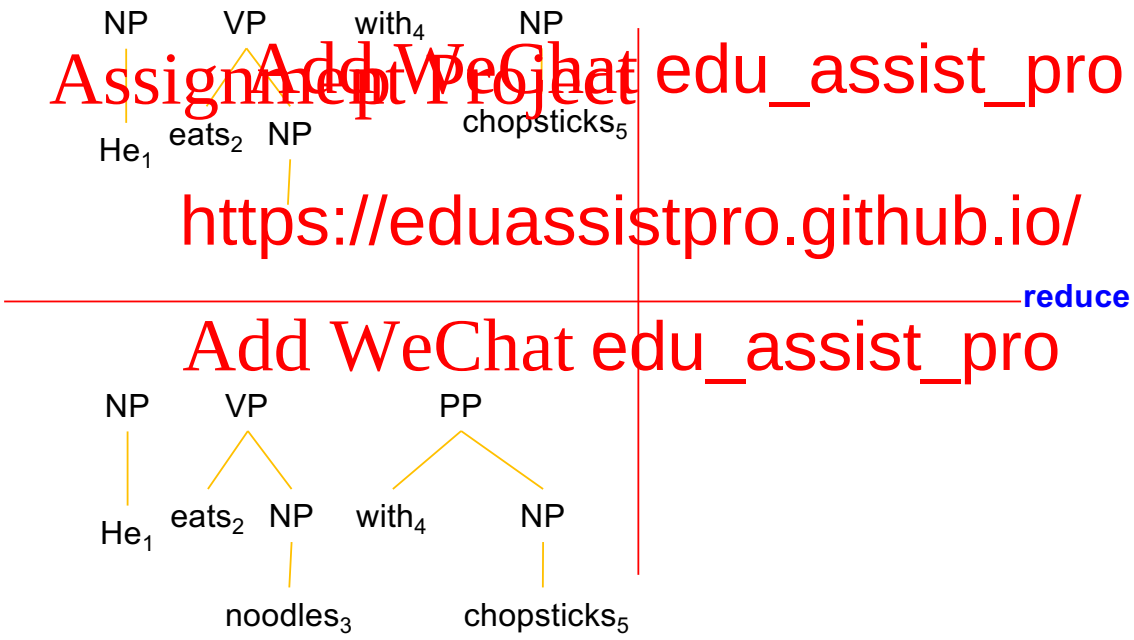


Shift-Redu



orithm



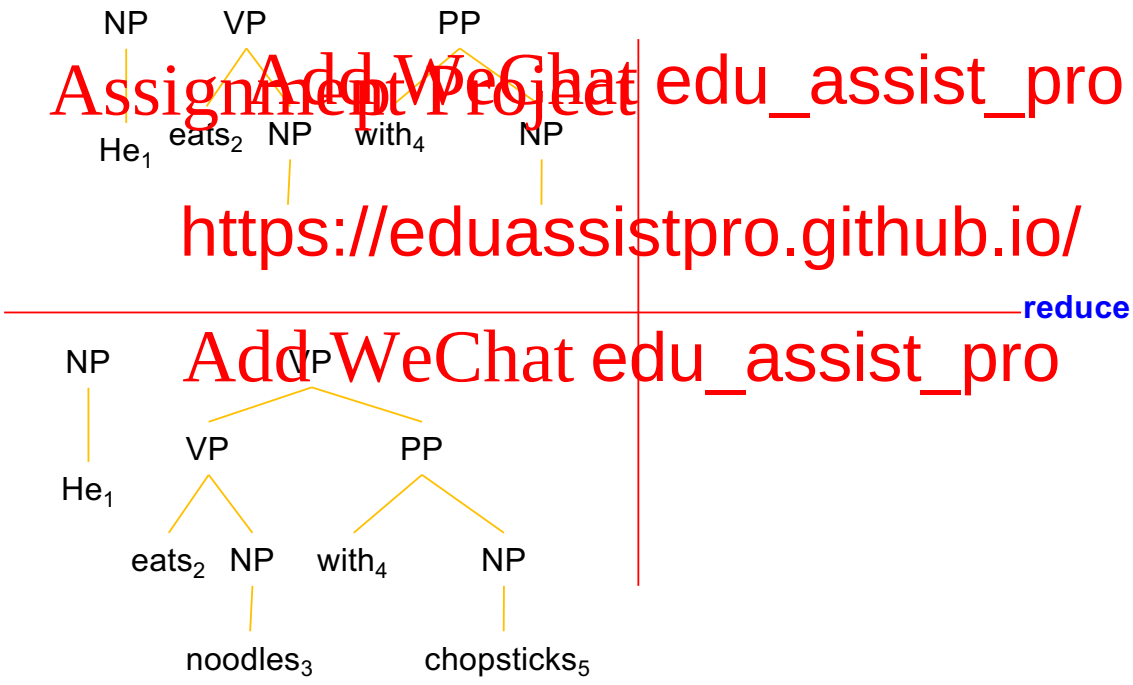


Shift-Redu



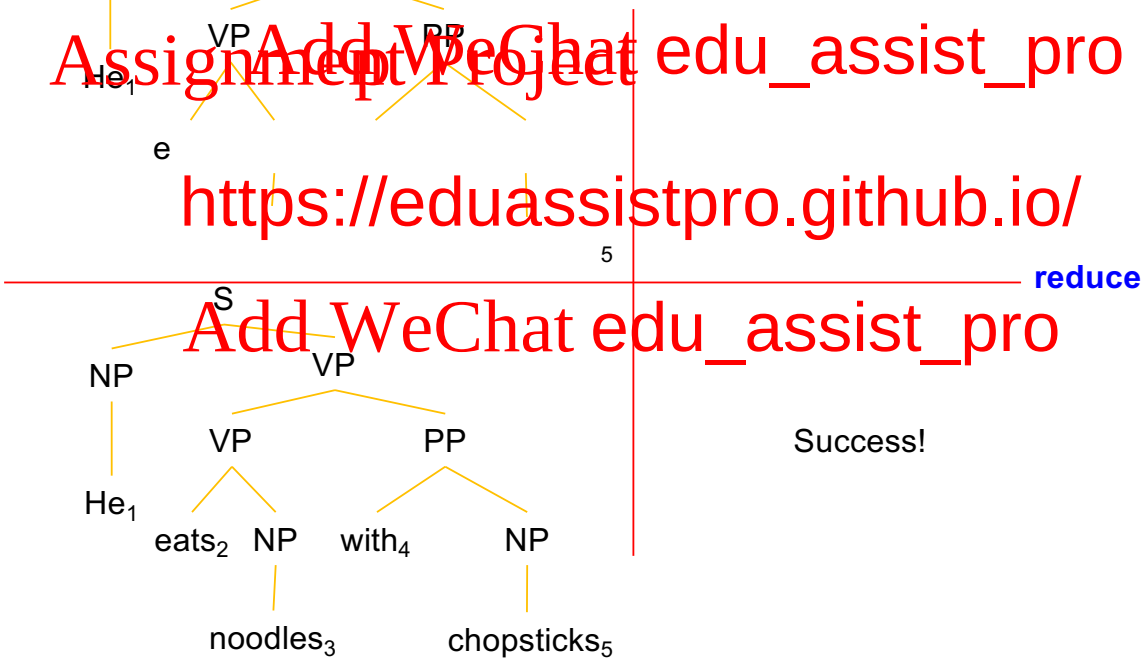
crithm





Shift-Redu rithm





## “Oracle”



- The oracle is a sequence of actions that lead to th parse of a sentence.
- When trainin t map a gold parse tre
- We can learn a model by comparing the ora action sequences and update the param

# The Perceptron learning algorithm



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

## Lexcialized transition-based parsing actions

- Each action $t$ s a state into a new state.
  - SHIFT(SH): remove the first word-POS pair from $\beta$, and pushes it onto the top of $\sigma$;
  - REDUCE-UNARY-X(RU-X): pop the to σ construct a new binary node labeled with ubtree, and then . The head of the new sub
  - REDU o subtrees from $\sigma$, combine them into a new tree w with $X$, then push the new subtree bac $\sigma$. 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 $\sigma$ is a stack that is maintained to hold the partial parsing structures that are already constructed and $\beta$ is a queue used to store unprocessed input (typically word-POS tag pairs).

## Updating feature weights



$$\nabla_\theta \left( \begin{bmatrix} p_0tc = N - NP \sim shift & 1 \\ p_0tc = N - NP \sim reduce & 0 \\ p_0wc = noodles \quad NP \quad shift & 0 \\ p_0wc = p_1t & 1 \\ p_1tc = V - V \sim reduce & 1 \\ p_1wc = eats - V \sim shift & 0 \\ p_1wc = eats - V \sim reduce & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

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 | $p_0 wc$, $p_3 t_0$, $p_3 wc$, $q_0 wt$, $q_1 wt$, $q_2 wt$, $q_3 wt$, $p_{0l} wc$, $p_{0r} wc$, $p_{0u} wc$, $p_{1l} wc$, $p_{1r}$ |
| bigrams | $p_0 wp_1 w$, $p_0 wp_1 c$, $p_0 cp_1 c$, $p_0 cq_0 t$, $p_0 wq_0 t$, $p_0 wq_0 w$, $p_1 wq_0 w$, $p_1 w$, $cq_0 t$, $p_0 cq_0 w$ |
| trigrams | $p_0 cp_1 cp_2 c$, $p_0 cp_1 cp_2 w$, $p_0 cp_1 wq_0 t$, $p_0 cp_1 cq_0 w$ |

Baseline features, where $p_i$ represents the $i_{th}$ subtree in the stack $\sigma$ and $q_i$ denotes the $i_{th}$ item in the queue $\beta$. $w$ 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 | $p_0$ P˜reduce | 1 |
| $p_1 tc$=V-V˜shift | 0 | $p_1$ e | 1 |
| $p_1 wc$=eats-V˜ | | -V˜reduce | 1 |
| $p_{0u} wc$=noodle | | dles-N˜reduce | 1 |
| $q_0 wt$=with-P˜shift | 0 | reduce | 1 |
| $q_1 wt$=chopsticks-N˜shift | 0 | cks-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 s
**Output:** A constituent parse tree

1: $beam_0 \leftarrow \{s_0\}$ ▷ initialization
2: $i \leftarrow 0$ ▷ step index
3: **loop**
4: $P \leftarrow \{\}$ priority queue
5: **while** *bea*
6: $s \leftarrow$
7: **for** all p
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 algorit transition-b
- The transitio dependency parsing as well as graph-ba g
- Learning for transition-based parsing basically any type of classifier, including neural network models