CS11-747 Neural Networks for NLP

Generating Trees or Graphs w/ Neural Networks

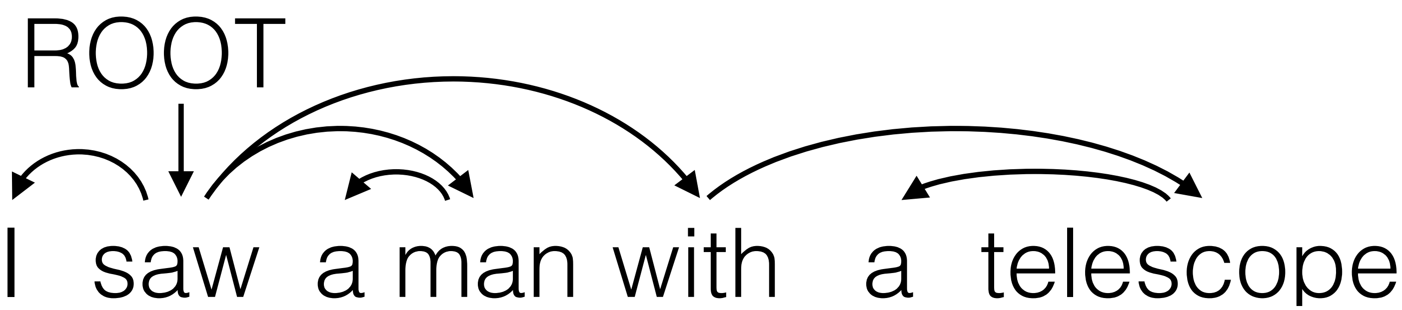
Graham Neubig

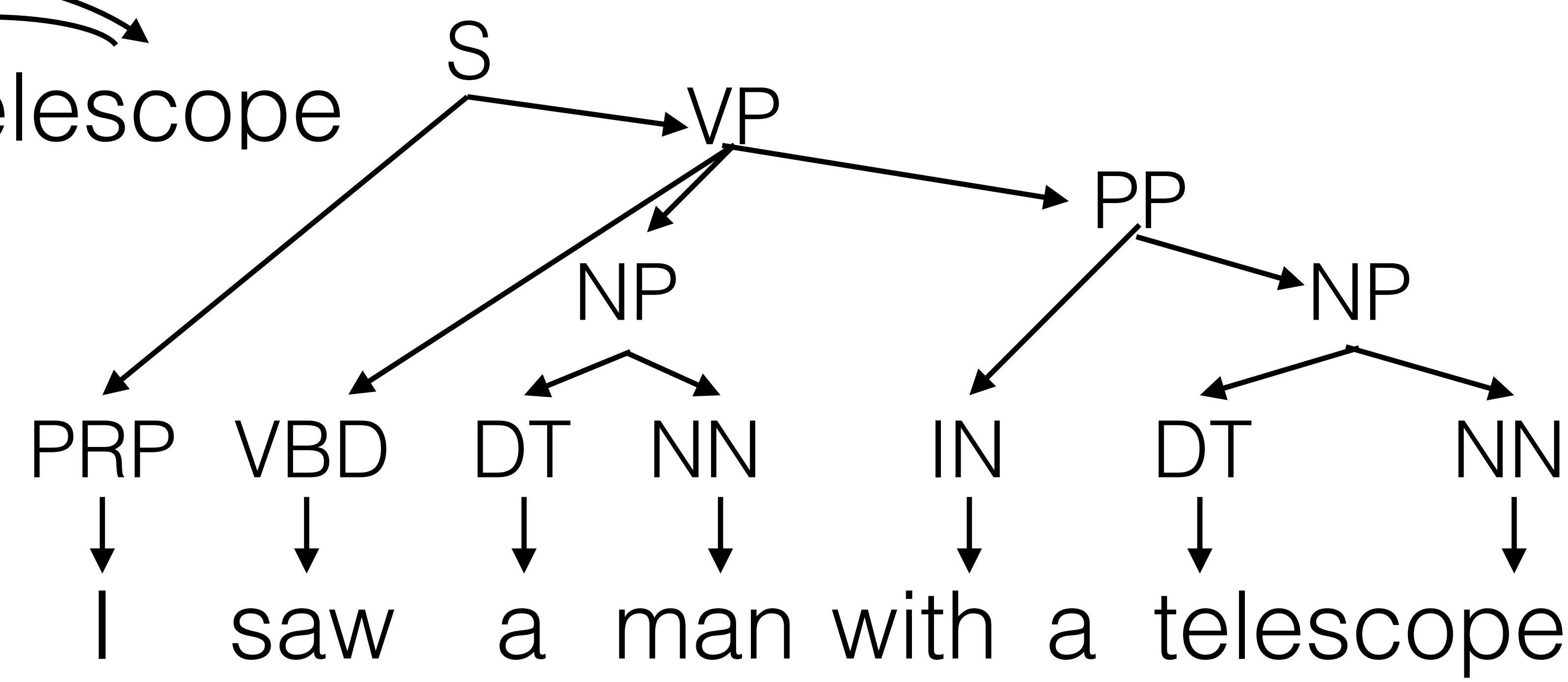
Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/nn4nlp2021/

Trees and Graphs in NLP

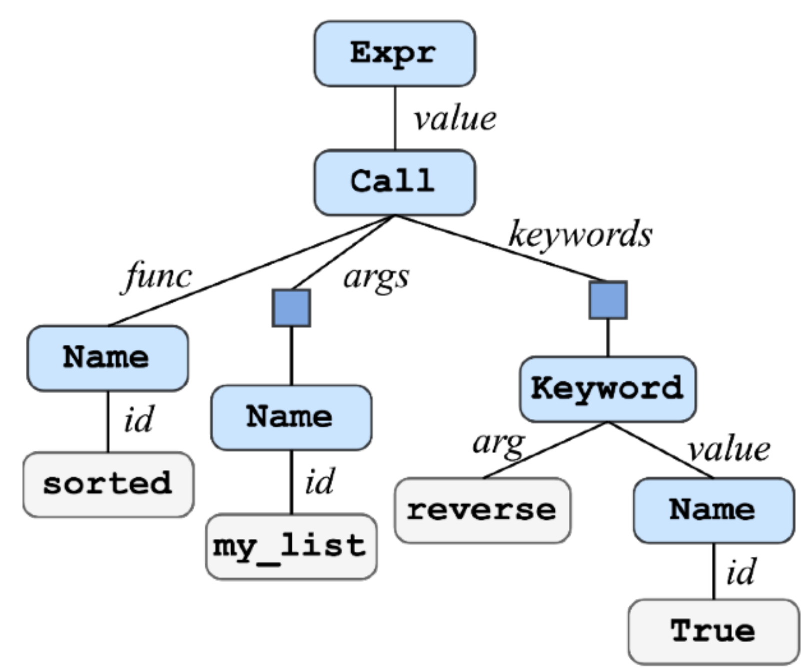
- **Syntactic Structure:**





- **Underlying Semantics:**

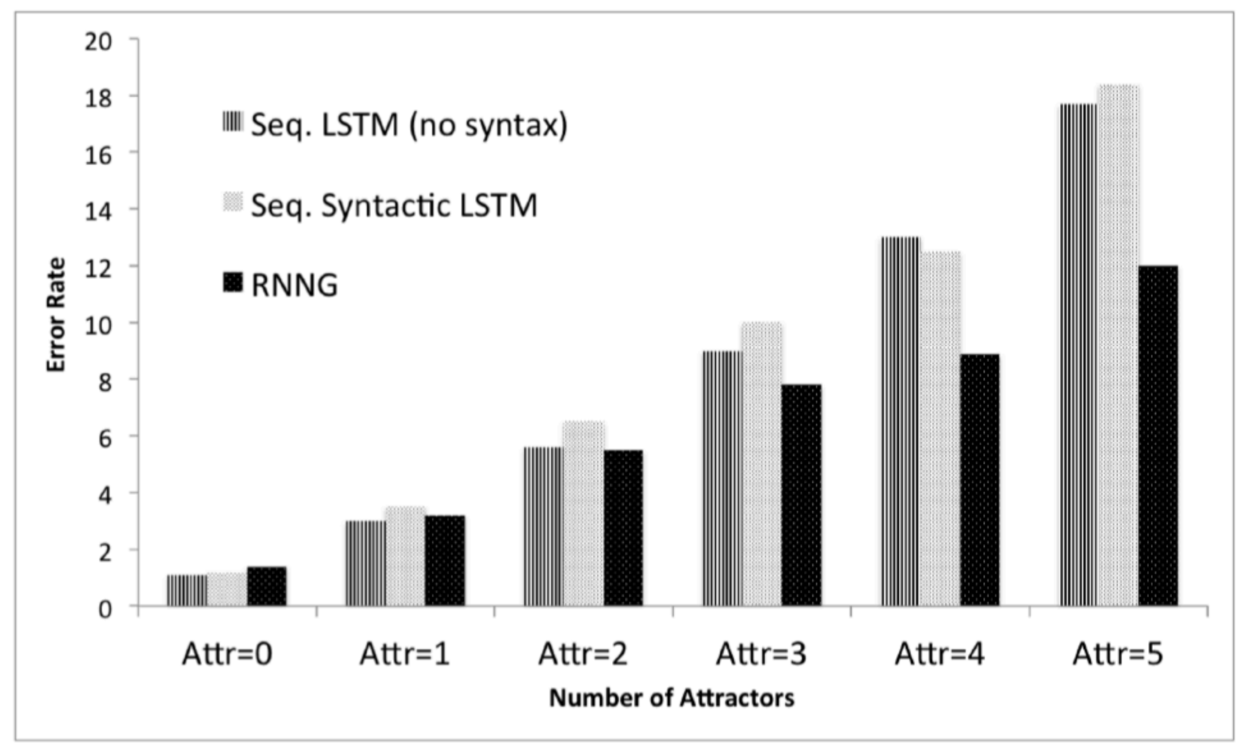
Sort my_list in descending order



Why Syntactic Structure?

- Regular models over word sequences do quite well

- But may not capture phenomena that inherently require structure, such as long-distance agreement e.g. Kuncoro et al (2018)



- Important for robustness, generalization

Why Semantic Structure?

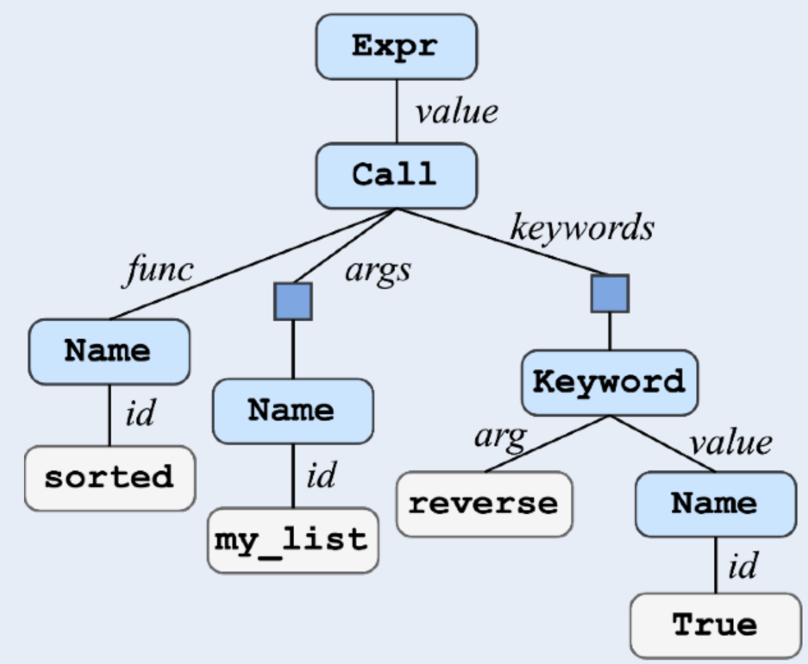
Natural Language Abstracted to Actionable Meaning

Sort my_list in descending order

```
sorted(my_list,
reverse=True)
```

Example: Python code generation

Structured Meaning Representations



Abstract Syntax Trees

- Executable programs
- Abstracted meaning representations

Parsing

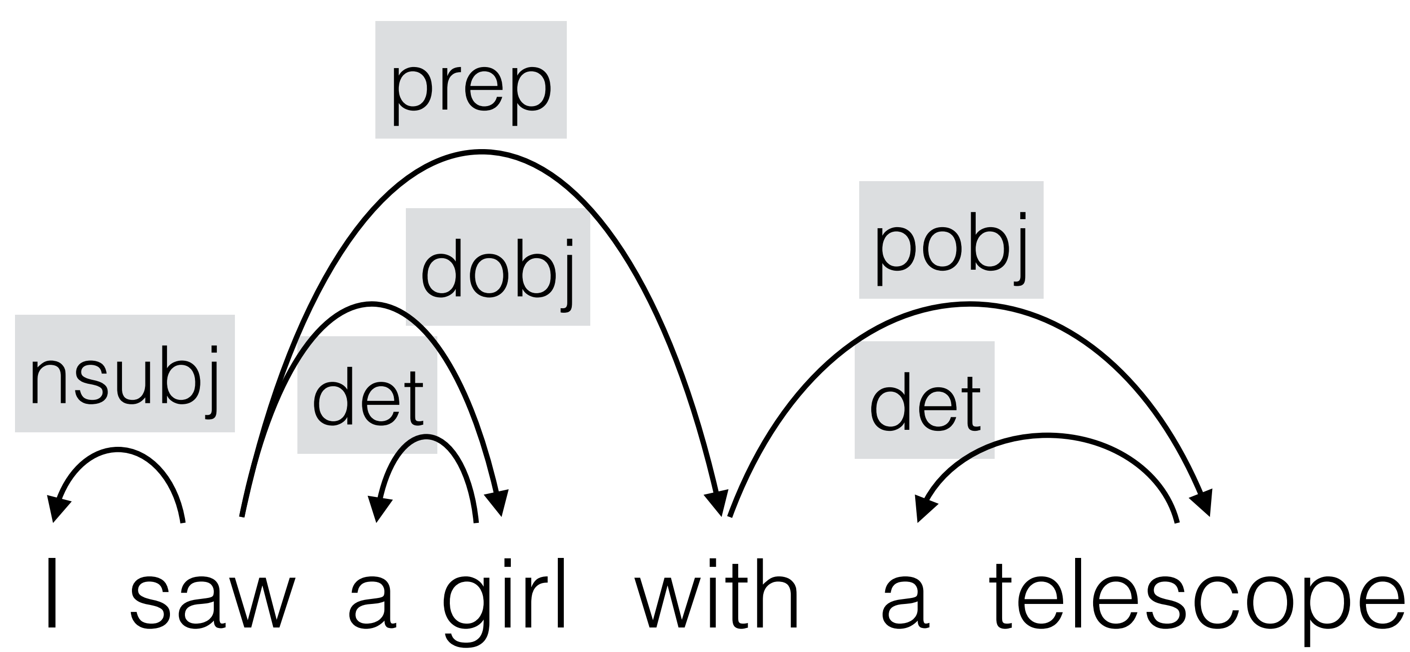
- Predicting structured outputs from input sentence
- **Transition-based models**
  - step through actions one-by-one until we have output
  - like history-based model for POS tagging
- **Graph-based models**
  - calculate probability of each edge/constituent, and perform some sort of dynamic programming
  - like linear CRF model for POS

Shift-reduce Dependency Parsing

Why Dependencies?

- Dependencies are often good for semantic tasks, as related words are close in the tree

- It is also possible to create labeled dependencies, that explicitly show the relationship between words

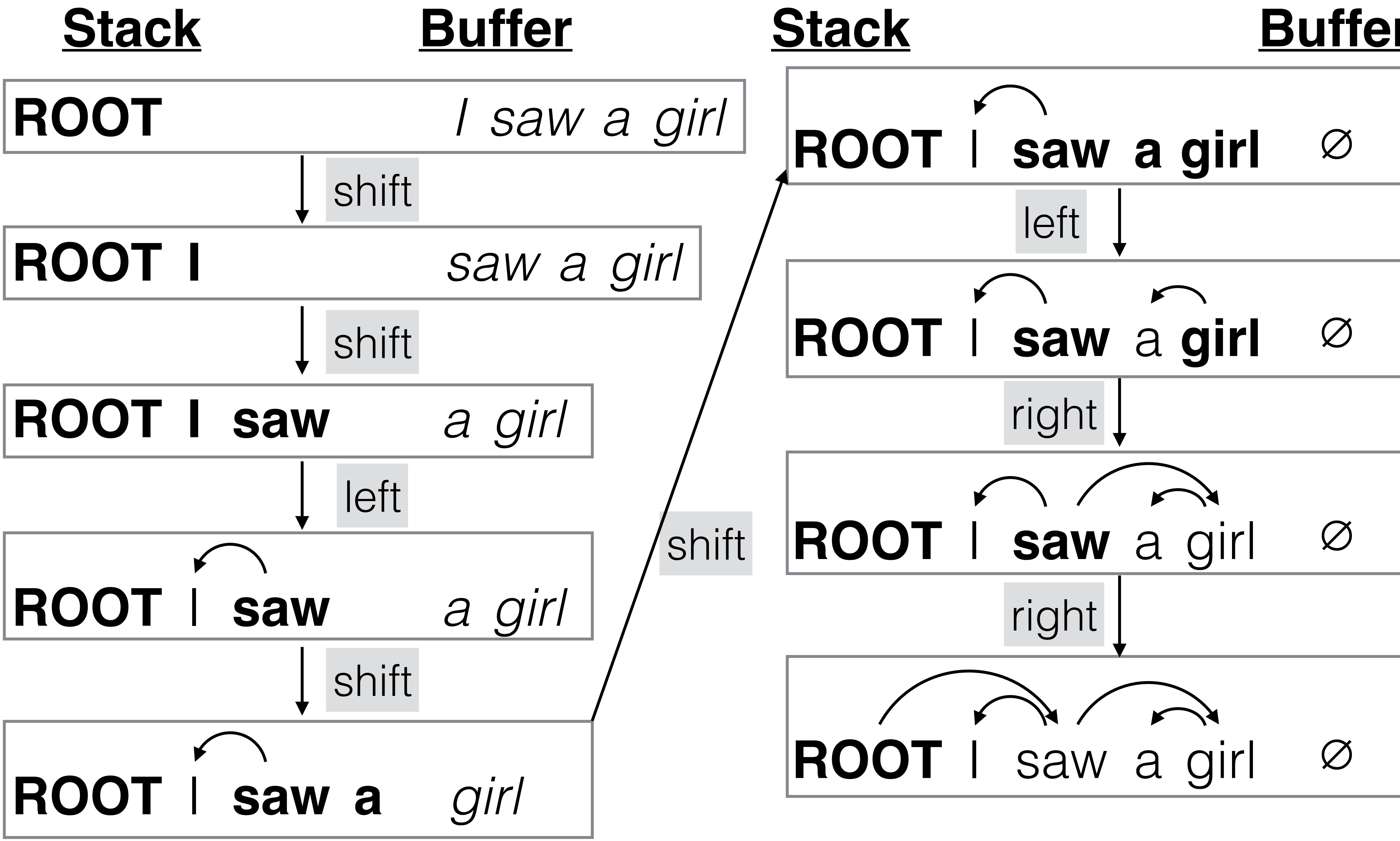


Arc Standard Shift-Reduce Parsing

(Yamada & Matsumoto 2003, Nivre 2003)

- Process words one-by-one left-to-right
- Two data structures
  - **Queue:** of unprocessed words
  - **Stack:** of partially processed words
- At each point choose
  - **shift:** move one word from queue to stack
  - **reduce left:** top word on stack is head of second word
  - **reduce right:** second word on stack is head of top word
- Learn how to choose each action with a classifier

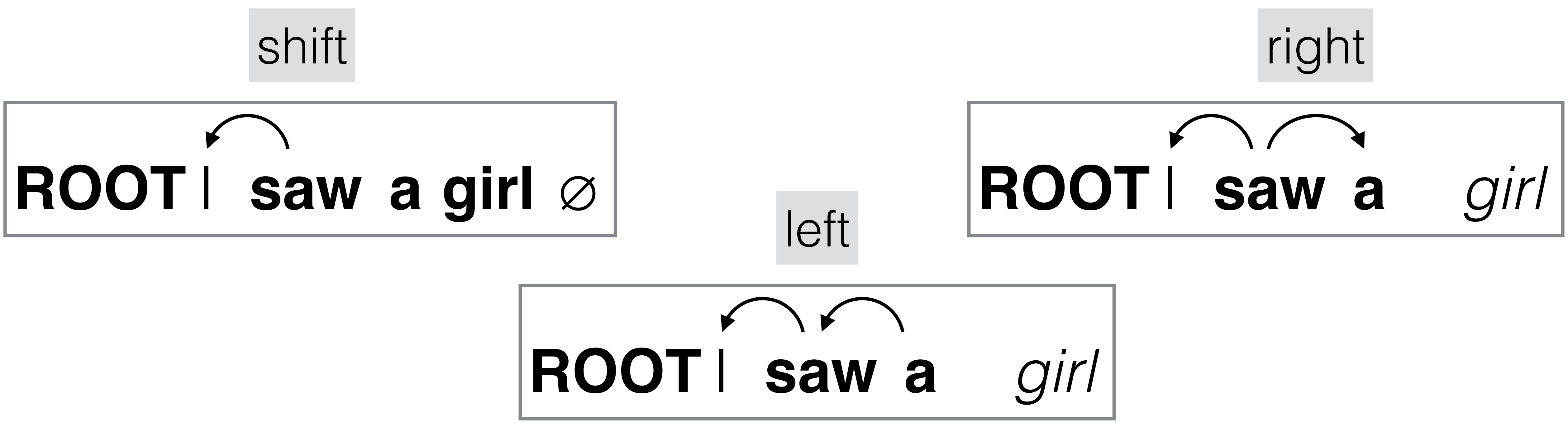
Shift Reduce Example



Classification for Shift-reduce

- Given a **configuration**

- Which **action** do we choose?



Making Classification Decisions

- Extract features from the configuration
  - what words are on the stack/buffer?
  - what are their POS tags?
  - what are their children?
- Feature combinations are important!
  - Second word on stack is verb **AND** first is noun: “right” action is likely
- Combination features used to be created manually (e.g. Zhang and Nivre 2011), now we can use neural nets!

A Feed-forward Neural Model for Shift-reduce Parsing

(Chen and Manning 2014)

- Extract non-combined features (embeddings)
- Let the neural net do the feature combination

Softmax layer:

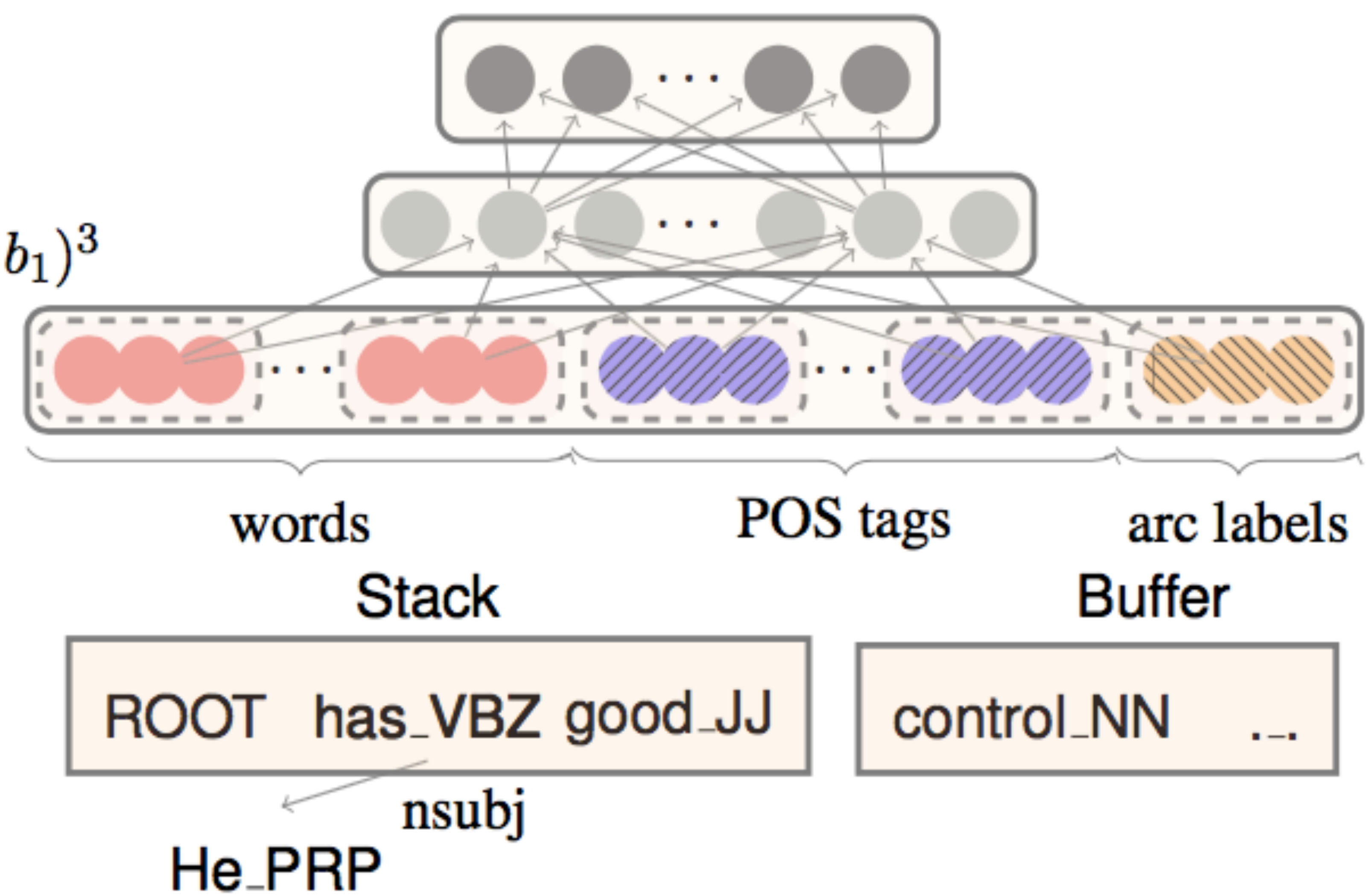
$$p = \mathtt{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

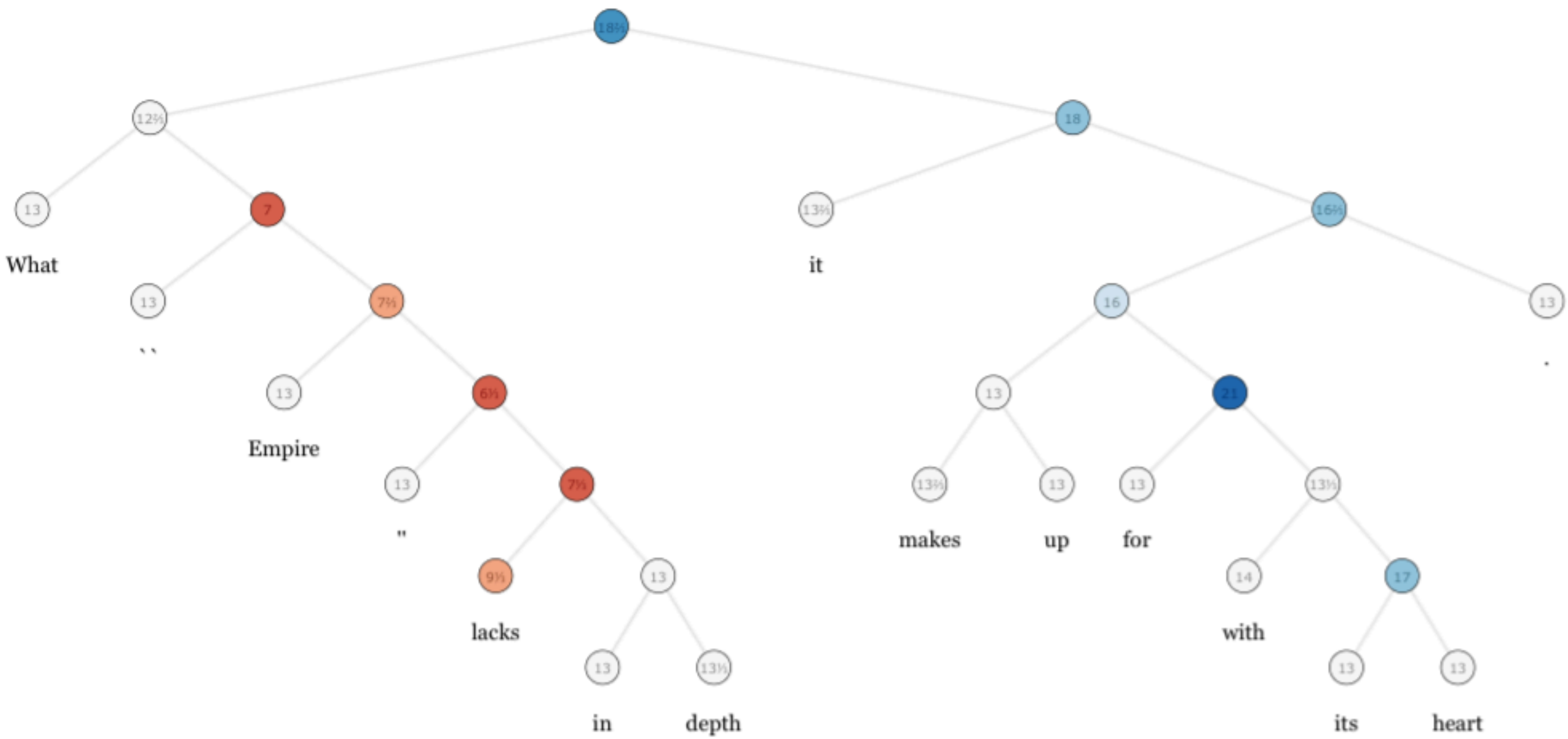
Input layer: $[x^w, x^t, x^l]$

Configuration



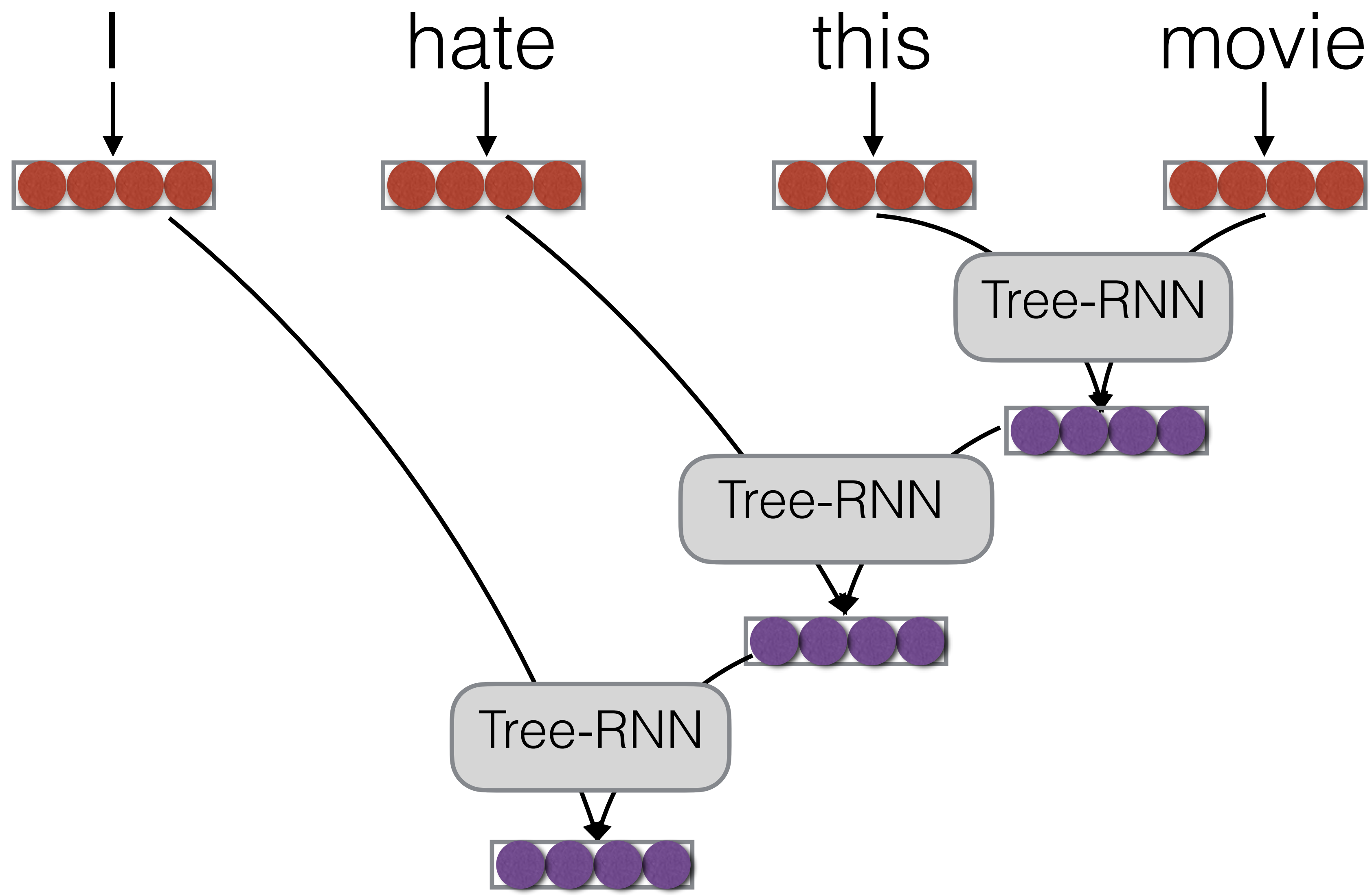
Using Tree Structure in NNs: Syntactic Composition

Why Tree Structure?



Recursive Neural Networks

(Socher et al. 2011)



$$\text{tree-rnn}(\boldsymbol{h}_1, \boldsymbol{h}_2) = \tanh(W[\boldsymbol{h}_1; \boldsymbol{h}_2] + \boldsymbol{b})$$

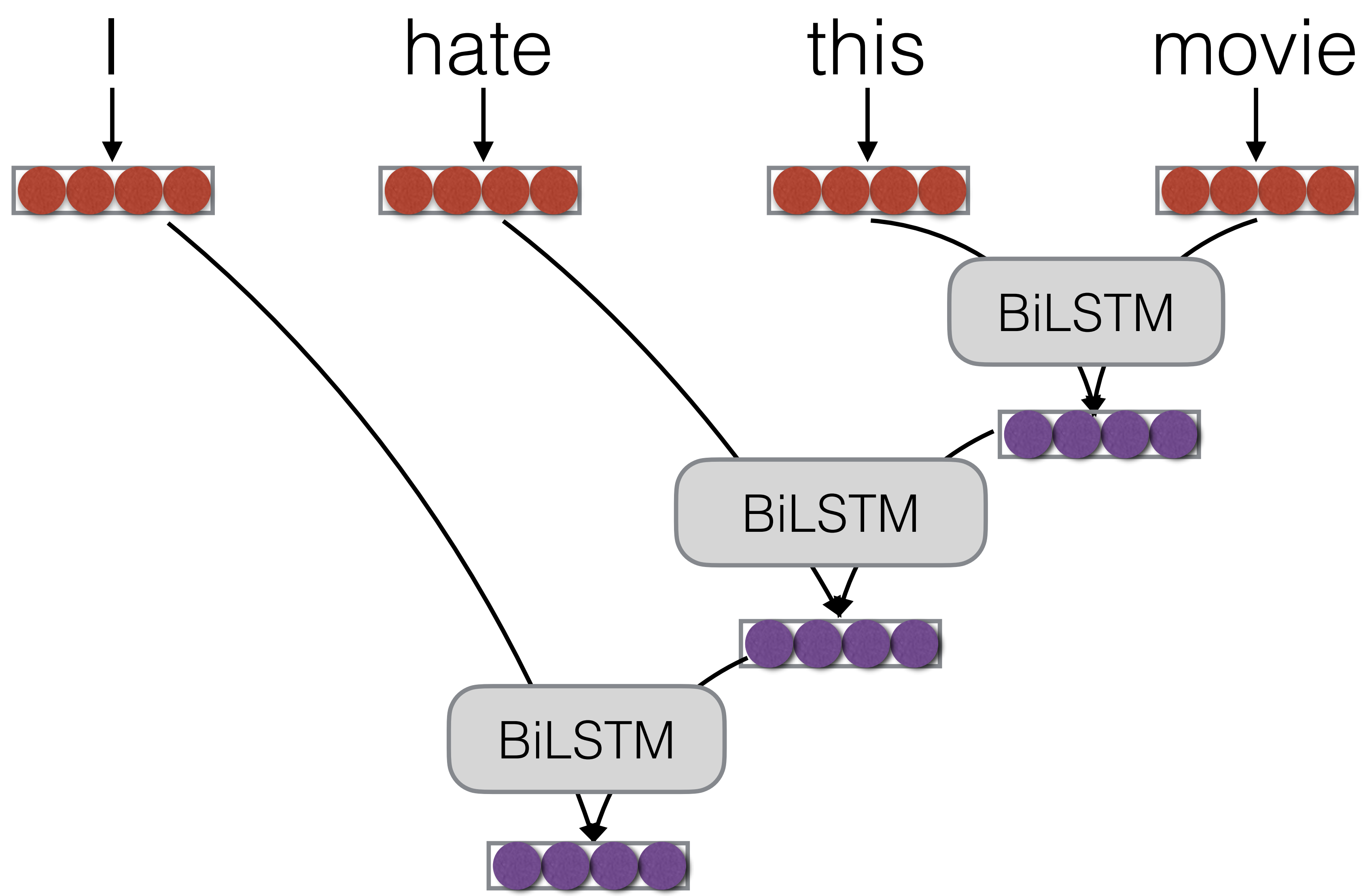
Can also parameterize by constituent type → different composition behavior for NP, VP, etc.

Tree-structured LSTM
(Tai et al. 2015)

- **Child Sum Tree-LSTM**
  - Parameters shared between all children (possibly based on grammatical label, etc.)
  - Forget gate value is different for each child → the network can learn to “ignore” children (e.g. give less weight to non-head nodes)
- **N-ary Tree-LSTM**
  - Different parameters for each child, up to N (like the Tree RNN)

Bi-LSTM Composition
(Dyer et al. 2015)

- Simply read in the constituents with a BiLSTM
- The model can learn its own composition function!



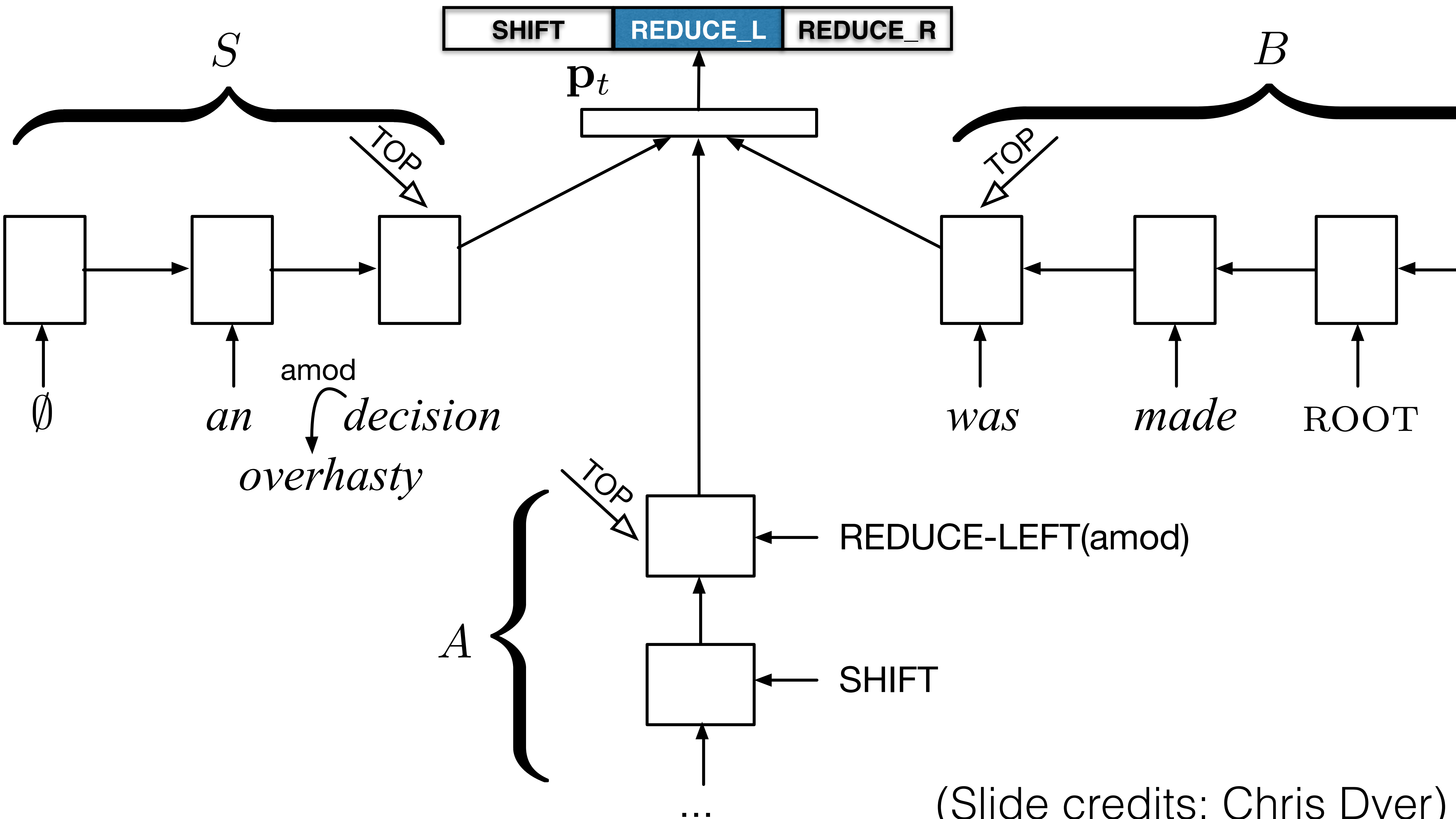
Let's Try it Out!

`tree-lstm.py`

Encoding Parsing Configurations w/ RNNs

- We don’t want to do feature engineering (why leftmost and rightmost grandchildren only?!)
- Can we encode all the information about the parse configuration with an RNN?
- Information we have: stack, buffer, past actions

Encoding Stack Configurations w/ RNNs

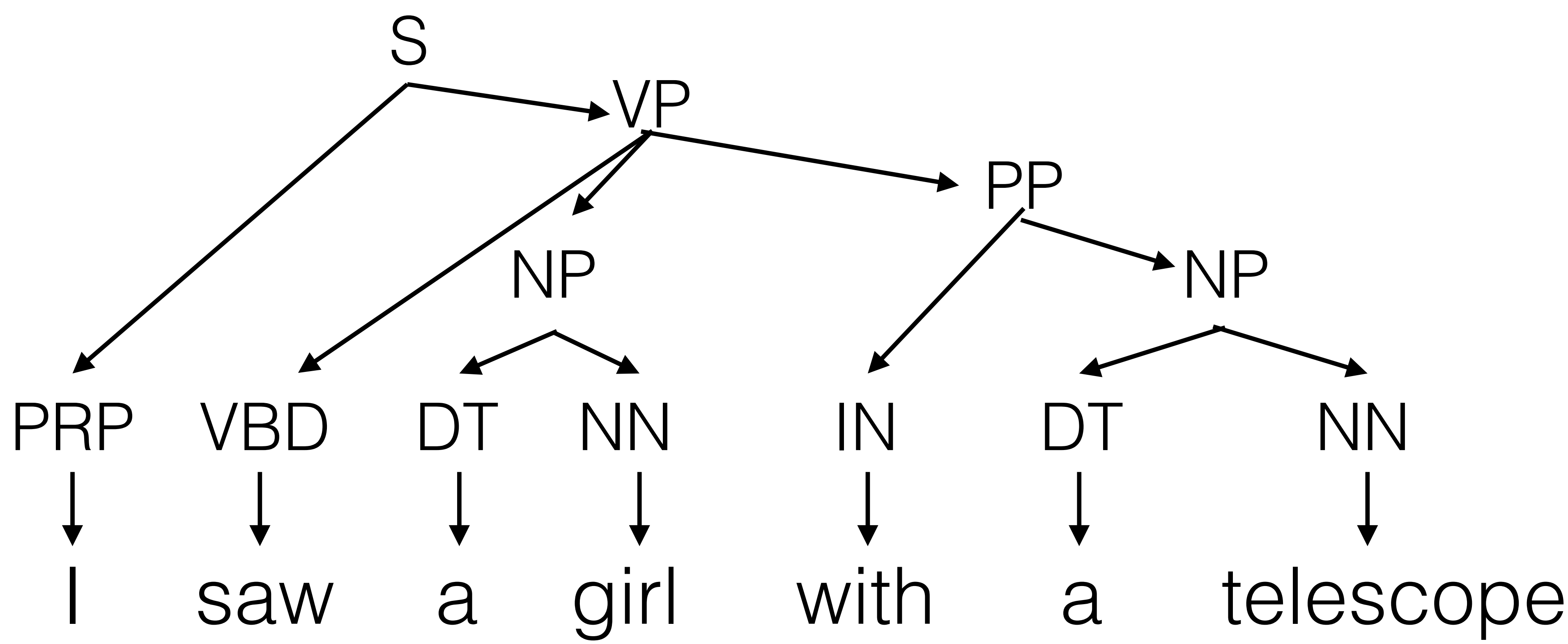


(Slide credits: Chris Dyer)

Dynamic Programming for Phrase Structure Parsing

Phrase Structure Parsing

- Models to calculate phrase structure

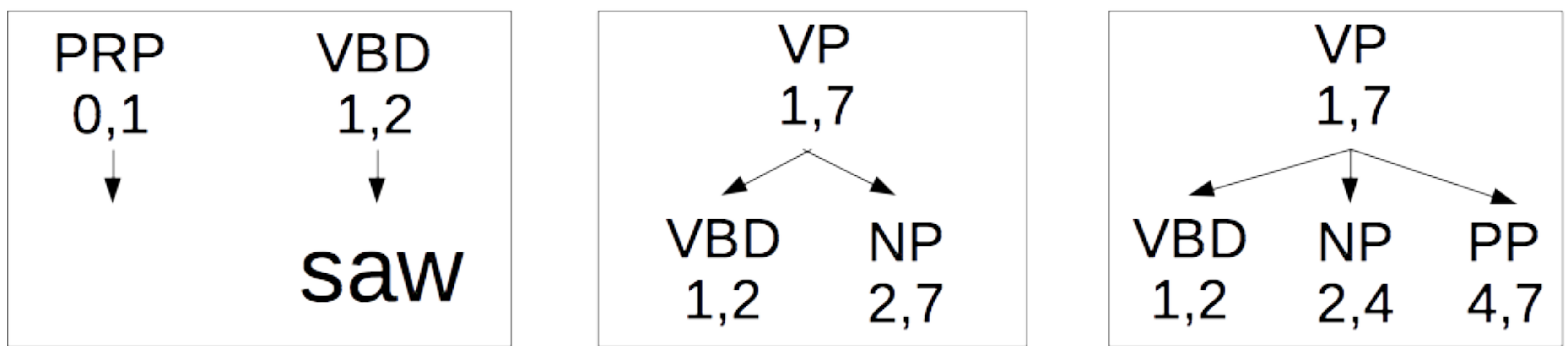


- Important insight: parsing is similar to tagging
  - Tagging is search in a **graph** for the best **path**
  - Parsing is search in a **hyper-graph** for the best **tree**

What is a Hyper-Graph?

- The “degree” of an edge is the number of children

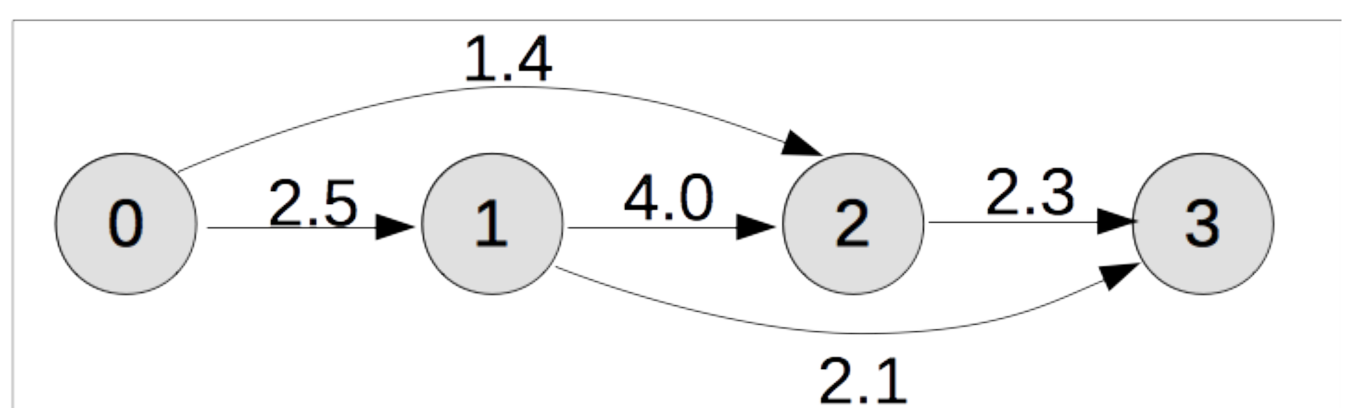
Degree 1 Degree 2 Degree 3



- The degree of a hypergraph is the maximum degree of its edges

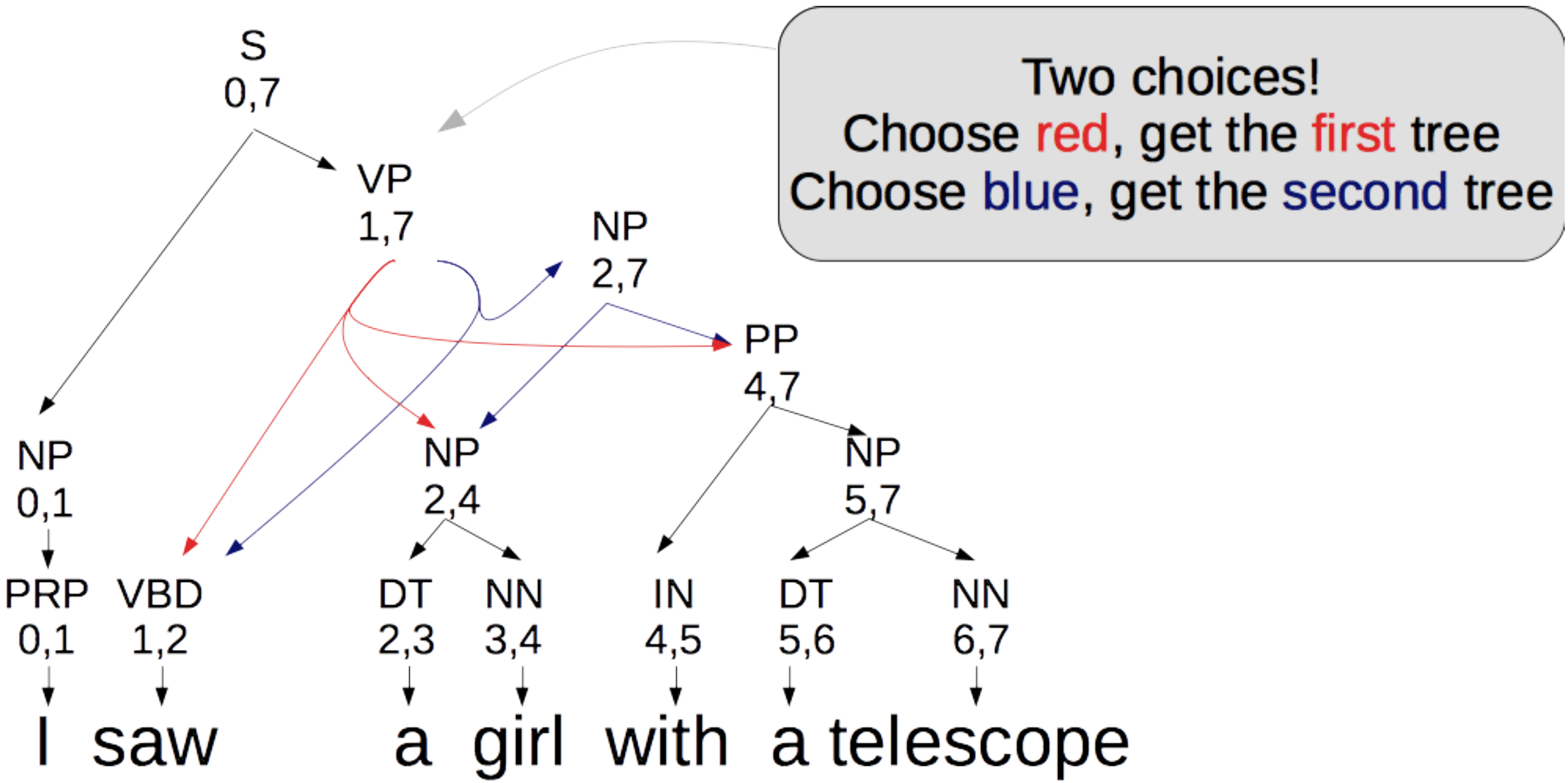
- A graph is a hypergraph of degree 1!

Example →



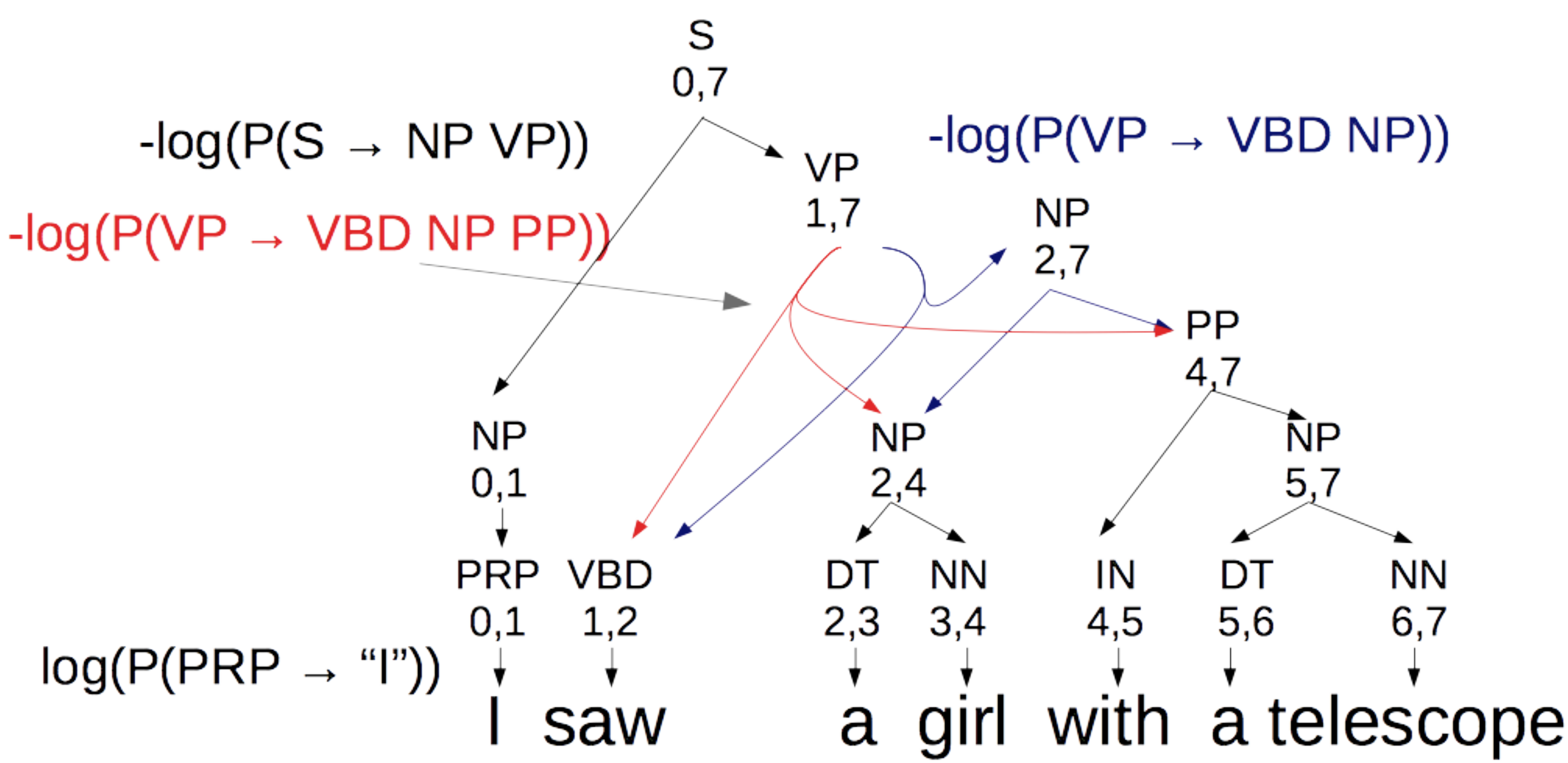
Tree Candidates as Hypergraphs

- With edges in one tree or another



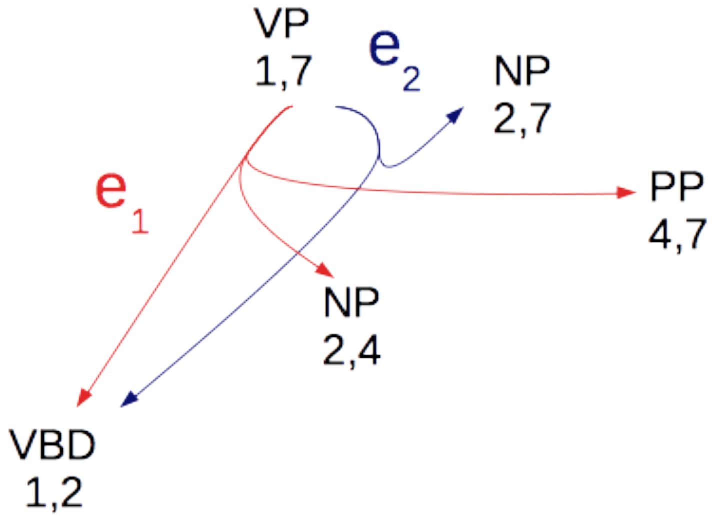
Weighted Hypergraphs

- Like graphs, can add weights to hypergraph edges
- Generally negative log probability of production



Hypergraph Search Example: CKY Algorithm

- Find the highest-scoring tree given a CFG grammar
- Create a hypergraph containing all candidates for a binarized grammar, do hypergraph search



score(e_1) =
 -log(P(VP → VBD NP PP)) +
 best_score[VBD1,2] +
 best_score[NP2,4] +
 best_score[NP2,7]

score(e_2) =
 -log(P(VP → VBD NP)) +
 best_score[VBD1,2] +
 best_score[VBD2,7]

best_edge[VB1,7] = $\text{argmin}_{e1,e2}$ score

best_score[VB1,7] =
 score(best_edge[VB1,7])

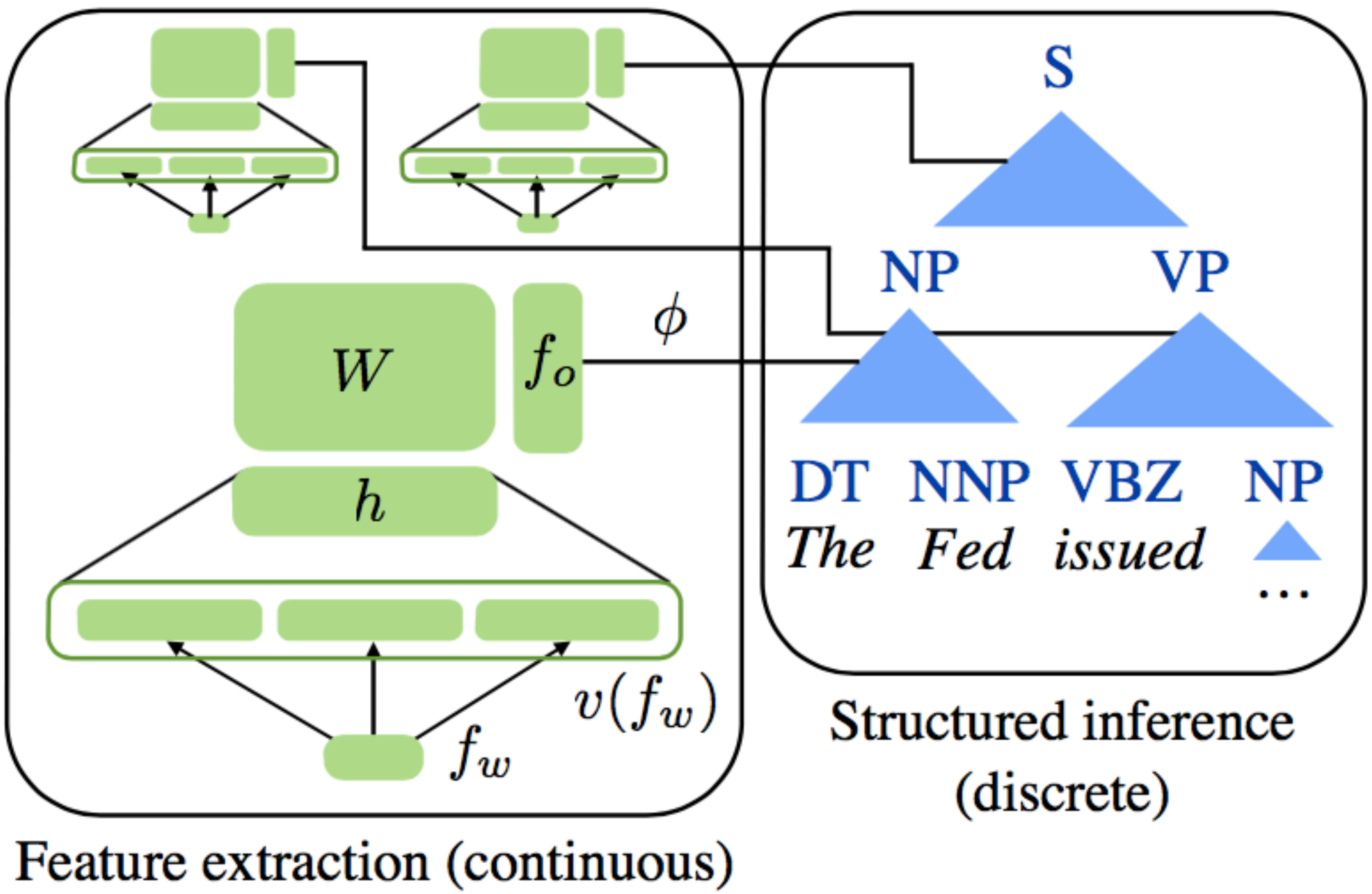
- Analogous to Viterbi algorithm, which is over graphs, but over hyper-graphs

Hypergraph Partition Function: Inside-outside Algorithm

- Find the marginal probability of each span given a CFG grammar
- Partition function us probability of the top span
- Same as CKY, except we logsumexp instead of max
- Analogous to forward-backward algorithm, but forward-backward is over graphs, inside-outside is over hyper-graphs

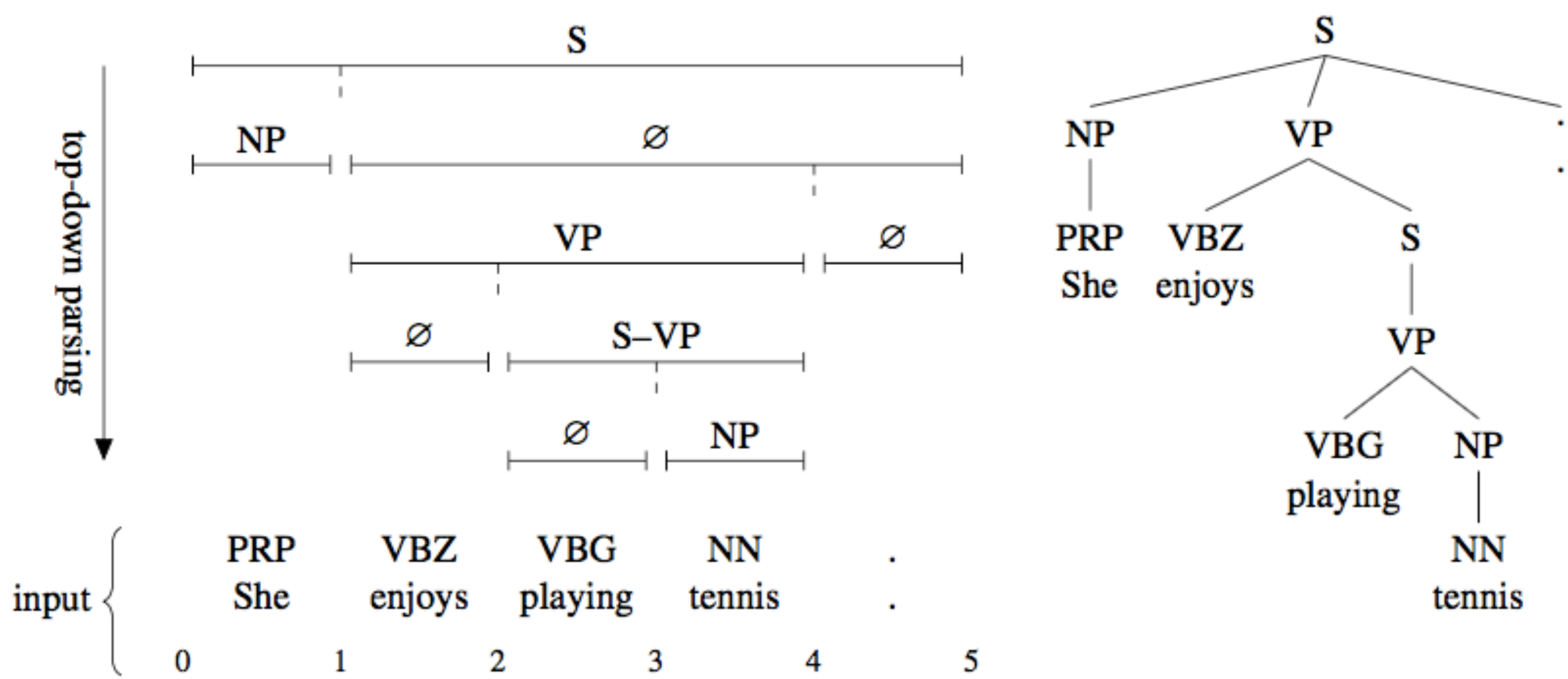
Neural CRF Parsing
(Durrett and Klein 2015)

- Predict score of each span using FFNN
- Do discrete structured inference using CKY, inside-outside



Span Labeling
(Stern et al. 2017)

- Simple idea: try to decide whether span is constituent in tree or not

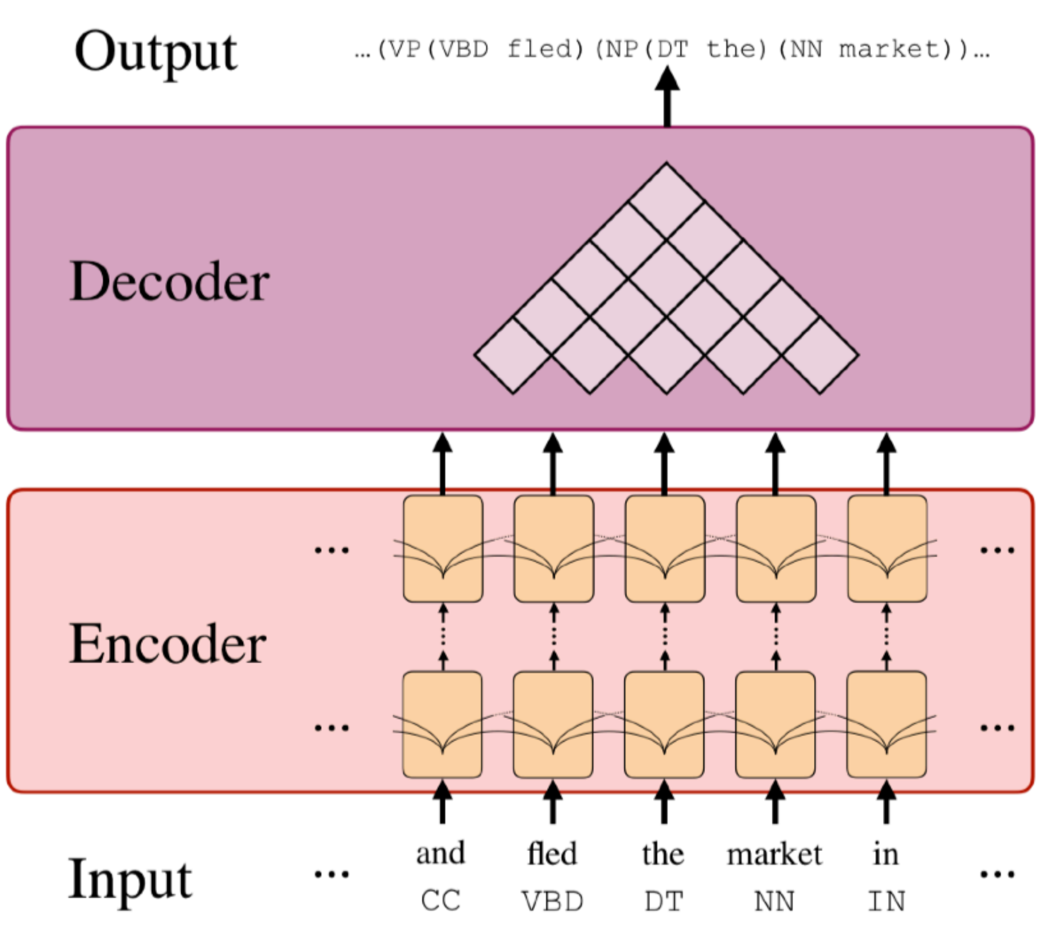


(a) Execution of the top-down parsing algorithm. (b) Output parse tree.

- Allows for various loss functions (local vs. structured), inference algorithms (CKY, top down)

Self-Attentional Encoding+Structured Inference (Kitaev et al. 2018)

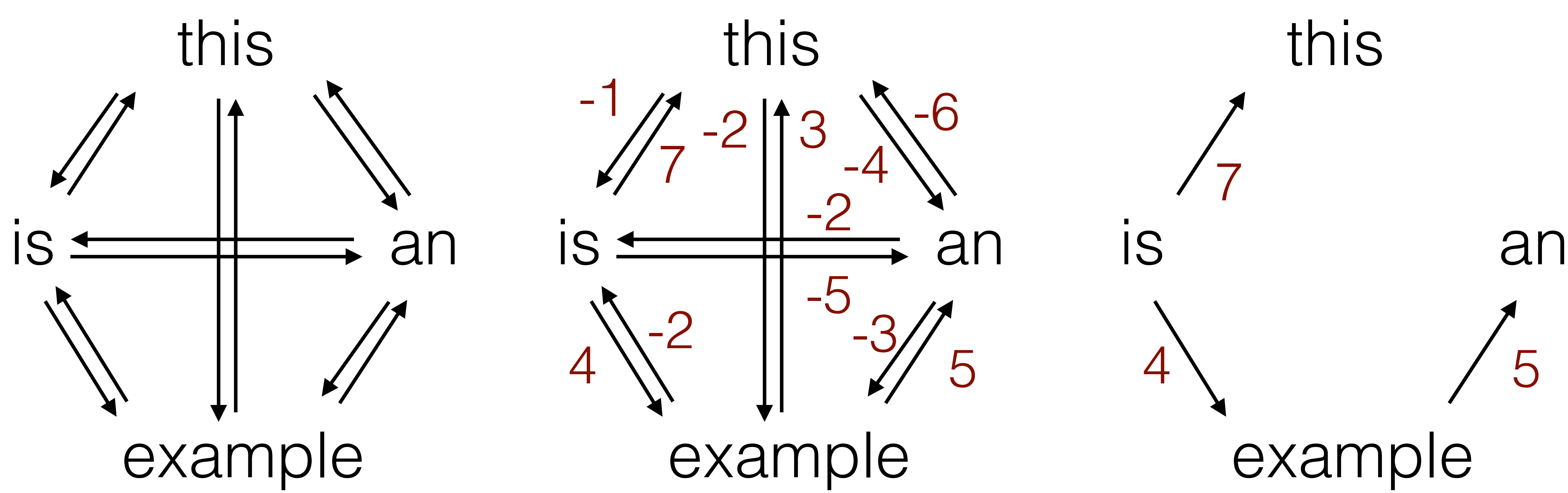
- Self-attention based encoding
- Structured margin-based inference
- Berkeley neural parser: https://github.com/nikitakit/self-attentive-parser



Neural Models for Graph-based Parsing

(First Order) Graph-based Dependency Parsing

- Express sentence as fully connected directed graph
- Score each edge independently
- Find maximal spanning tree

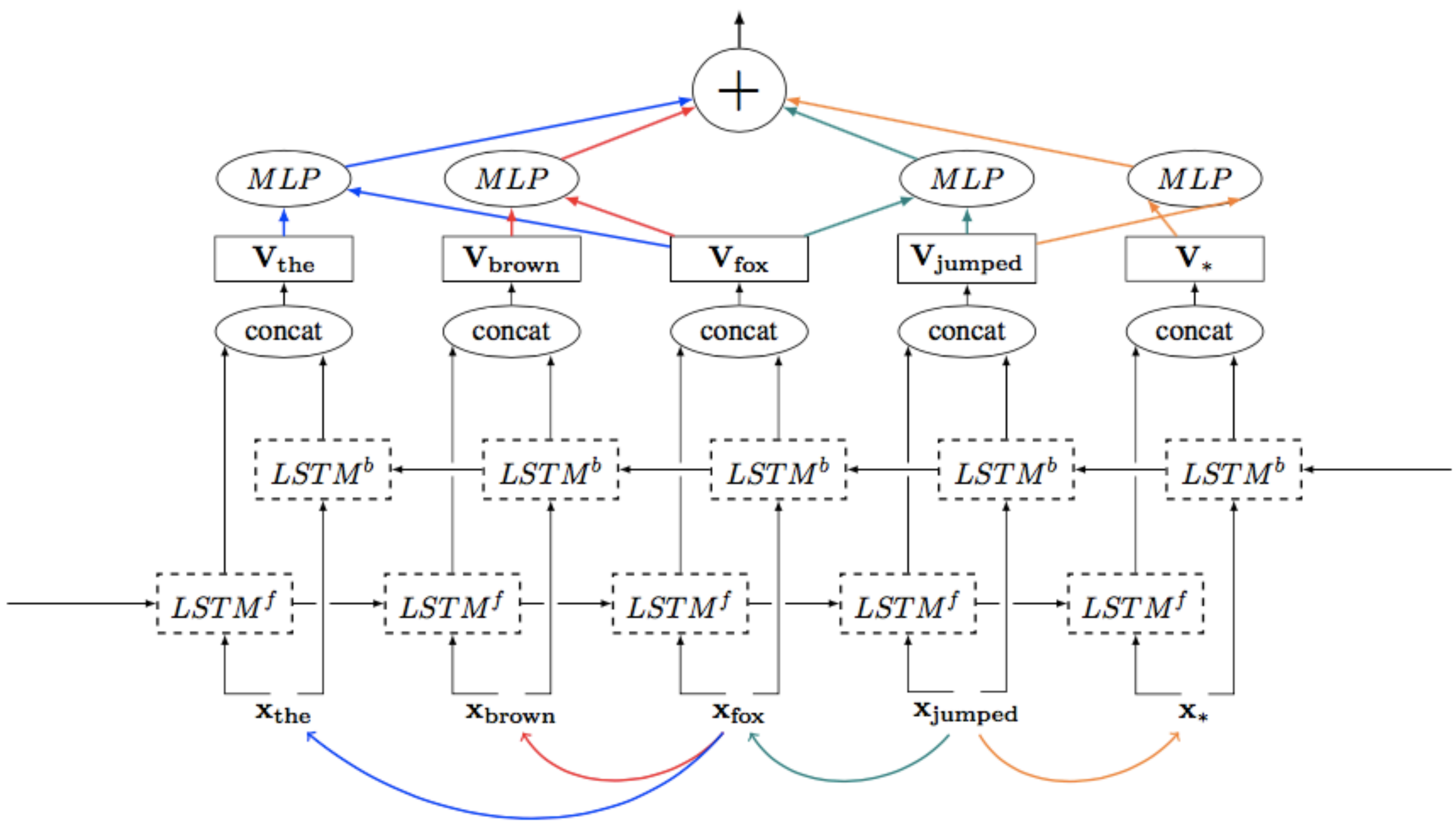


Chu-Liu-Edmonds
(Chu and Liu 1965, Edmonds 1967)

- We have a graph and want to find its spanning tree
- **Greedily select** the best incoming edge to each node (and subtract its score from all incoming edges)
- If there are cycles, select a cycle and **contract** it into a single node
- **Recursively call** the algorithm on the graph with the contracted node
- **Expand** the contracted node, deleting an edge appropriately

BiLSTM Feature Extractors

(Kipperwasser and Goldberg 2016)



- Simpler and better accuracy than manual extraction

BiAffine Classifier

(Dozat and Manning 2017)

$$\mathbf{h}_i^{(arc\text{-}dep)} = \mathbf{MLP}^{(arc\text{-}dep)}(\mathbf{r}_i)$$
$$\mathbf{h}_j^{(arc\text{-}head)} = \mathbf{MLP}^{(arc\text{-}head)}(\mathbf{r}_j)$$
$$\mathbf{s}_i^{(arc)} = H^{(arc\text{-}head)} U^{(1)} \mathbf{h}_i^{(arc\text{-}dep)} + H^{(arc\text{-}head)} \mathbf{u}^{(2)}$$

Learn specific representations for head/dependent for each word

Calculate score of each arc

- Just optimize the likelihood of the parent, no structured training
  - This is a local model, with global decoding using MST at the end
- Best results (with careful parameter tuning) on universal dependencies parsing task
- Implementation: https://github.com/XuezheMax/NeuroNLP2

Global Training

- Previously: margin-based global training, local probabilistic training
- What about global probabilistic models?

$$P(Y \mid X) = \frac{e^{\sum_{j=1}^{|Y|} S(y_j | X, y_1, \ldots, y_{j-1})}}{\sum_{\tilde{Y} \in V*} e^{\sum_{j=1}^{|\tilde{Y}|} S(\tilde{y}_j | X, \tilde{y}_1, \ldots, \tilde{y}_{j-1})}}$$

- Algorithms for calculating partition functions:
  - **Projective parsing:** Eisner algorithm is a bottom-up CKY-style algorithm for dependencies (Eisner et al. 1996)
  - **Non-projective parsing:** Matrix-tree theorem can compute marginals over directed graphs (Koo et al. 2007)
- Applied to neural models in Ma et al. (2017)

An Alternative:
Parse Reranking

An Alternative: Parse Reranking

- You have a nice model, but it's hard to implement a dynamic programming decoding algorithm
- Try reranking!
  - Generate with an easy-to-decode model
  - Rescore with your proposed model

Examples of Reranking

- Inside-outside recursive neural networks (Le and Zuidema 2014)
- Parsing as language modeling (Choe and Charniak 2016)
- Recurrent neural network grammars (Dyer et al. 2016)

A Word of Caution about Reranking! (Fried et al. 2017)

- Your reranking model got SOTA results, great!
- But, it might be an effect of model combination (which we know works very well)
    - The model generating the parses **prunes down the search space**
    - The reranking model chooses the best parse **only in that space**!

	Scoring models		
Candidates	RD	RG	RD + RG
RD	92.22	93.45	93.87
RG	90.24	89.55	90.53
RD ∪ RG	92.22	92.78	93.92

Questions?