Dependency Parsing

Sudeshna Sarkar

22 AUG 2019

Transition Based Dependency Parser

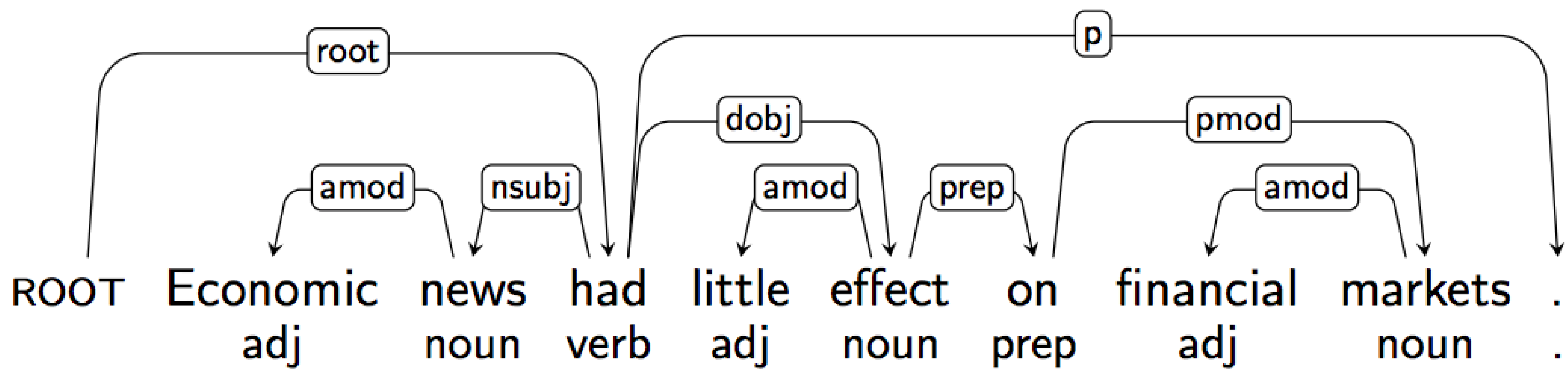
Start: σ = [ROOT], $\beta = w_1, \ldots, w_n$, A = Ø

1. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j,w_i)\}$
3. Right-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i,w_j)\}$

Finish: $\sigma = [w]$, β = Ø

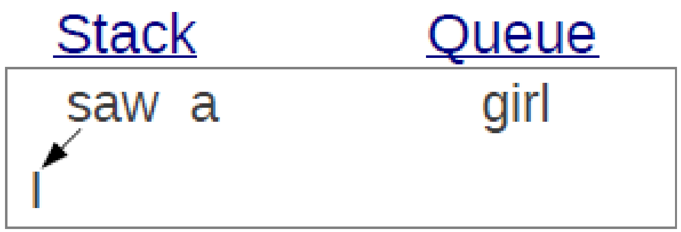
And on it goes until …

$[\text{ROOT}, \text{had}, .]_S \quad [\ ]_B$



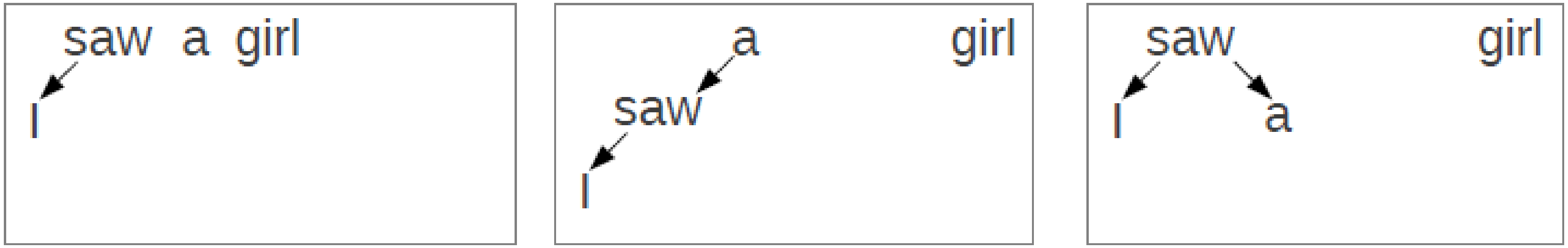
Classification for Shift-Reduce

- Given a state:



- Which action do we choose?

shift ? r left ? r right ?

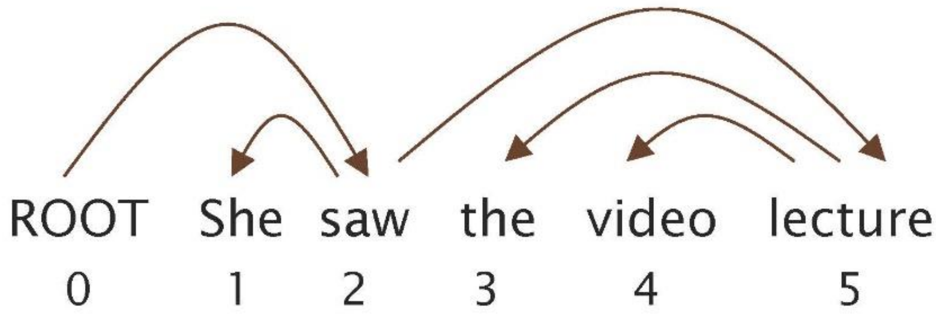


- Correct actions → correct tree

A supervised classification task

- Given the current state (i.e., stack, buffer and A) predict the next action.
- Can be viewed as a supervised learning problem.
- Each action is predicted by a discriminative classifier over each legal move
  - Max of 3 untyped choices; max of $|R| * 2 + 1$ when typed
- Features
  - Compute features of the current configuration of the stack, buffer and A.
  - Word in stack, POS of word, Word in buffer and POS of Word in buffer.
  - Other features: Length of dependency arc
- Greedy classifier (no search involved)
  - At each stage ask the classifier to predict the next transition.
- O(N) in length of sentence.

Evaluation of Dependency Parsing: (labeled) dependency accuracy



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

UAS = 4 / 5 = 80%

LAS = 2 / 5 = 40%

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed

1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Training

- Supervised ML methods require training material in the form of (input, output) pairs.
  - Treebanks associate sentences with their corresponding trees
  - We need parser states paired with their corresponding correct operators
  - But we do know the correct trees for each sentence

Training Data

- Automatically construct dependency parses from treebank.
- Compute correct sequence of "oracle" shift-reduce parse actions (transitions, t_i) at each step from gold-standard parse trees.
- Determine correct parse sequence by using a "shortest stack" oracle which always prefers LeftArc over Shift.
- At each stage it chooses using a case statement

    1. Left if the relation to be added is in the reference tree.
    2. Right if the resulting relation is in the correct tree AND if all the other outgoing relations associated with the word are already in the relation list.
    3. Otherwise shift

Stanford Neural Dependency Parser
(Chen and Manning, 2014)

- Train a neural net to choose the best shift-reduce parser action to take at each step.
- Uses features (words, POS tags, arc labels) extracted from the current stack, buffer, and arcs as context.

Neural Architecture

Softmax layer:

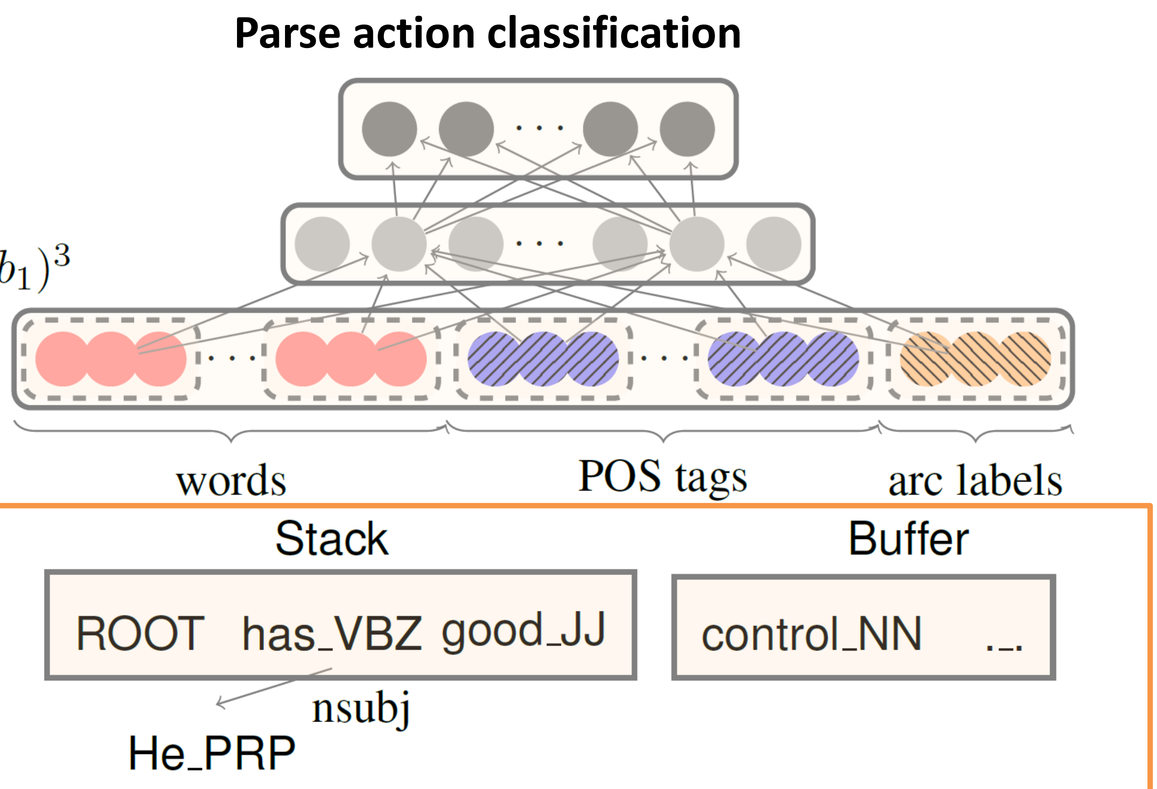
$$p = \texttt{softmax}(W_2 h)$$

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



State Representation

Extract a set of tokens from Stack/Buffer

	Word	POS	dep
S1	good	JJ	0
S2	has	VBZ	0
B1	control	NN	0
lc(S1)	0	0	0
rc(S1)	0	0	0
lc(S2)	He	PRP	nsubj
rc(S2)	0	0	0

Embeddings express similarities
POS: NN similar to NNS
deps: amod similar num

Concatenate their vector embeddings

Context Features Used

(rc = right-child, lc=left-child)

- The top 3 words on the stack and buffer: s_1; s_2; s_3; b_1; b_2; b_3;
- The first and second leftmost / rightmost children of the top two words on the stack: $lc_1(s_i)$; $rc_1(s_i)$; $lc_2(s_i)$; $rc_2(s_i)$, $i = 1; 2$.
- The leftmost-of-leftmost and rightmost-of-rightmost children of the top two words on the stack: $lc_1(lc_1(s_i))$; $rc_1(rc_1(s_i))$, $i = 1; 2$.
- Also include the POS tag and parent arc label (where available) for these same items.

Input Embeddings

- Instead of using one-hot input encodings, words and POS tags are “embedded” in a 50 dimensional set of input features.
- Embedding POS tags is unusual since there are relatively few; however, it allows similar tags (e.g. NN and NNS) to have similar embeddings and thereby behave similarly.

Cube Activation Function

- Alternative non-linear output function instead of sigmoid (softmax) or tanh.
- Allows modeling the product terms of $x_i x_j x_k$ for any three different input elements.
- Based on previous empirical results, capturing interactions of three elements seems important for shift-reduce dependency parsing.

Training Algorithm

- Training objective is to minimize the cross-entropy loss plus a L2-regularization term:

$$L(\theta) = -\sum \log p_{t_i} + \frac{\lambda}{2} \|\theta\|^2$$

- Initialize word embeddings to precomputed values such as Word2Vec.
- Use AdaGrad with dropout to compute model parameters that approximately minimize this objective.