CS60075
Natural Language Processing
Autumn 2020

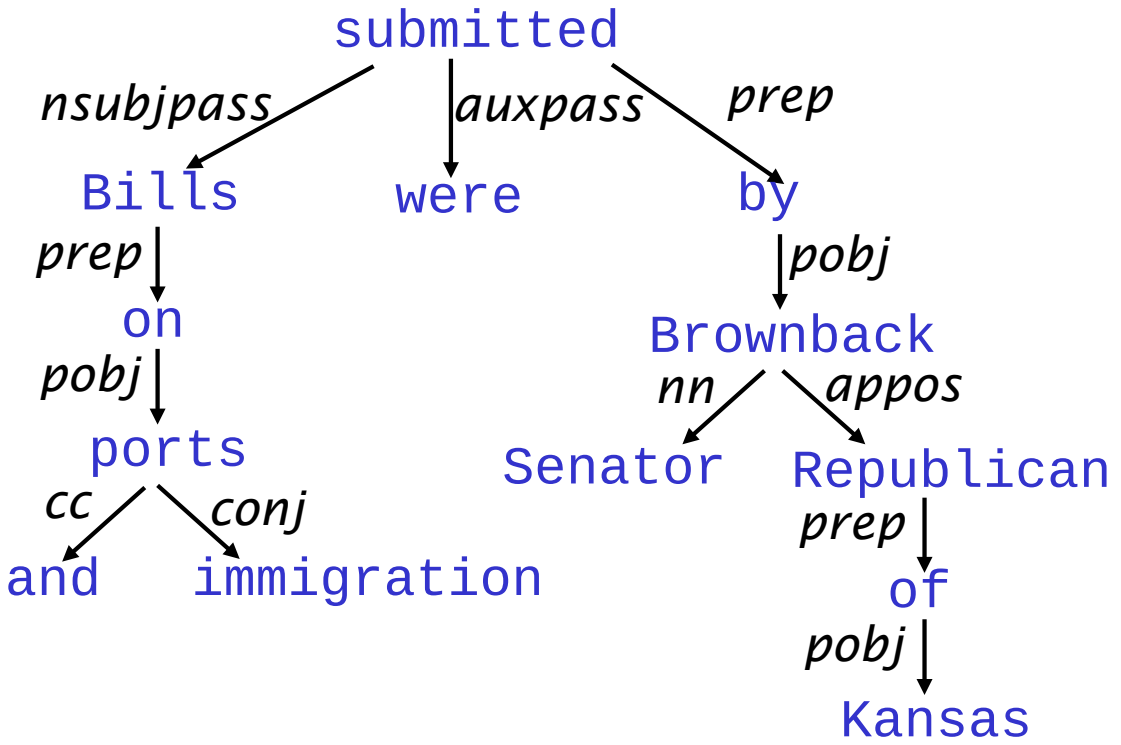
Module 5: Part D
Dependency Parsing

Oct 8 2020

Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations ("arrows") called dependencies

The arrows are commonly typed with the name of grammatical relations

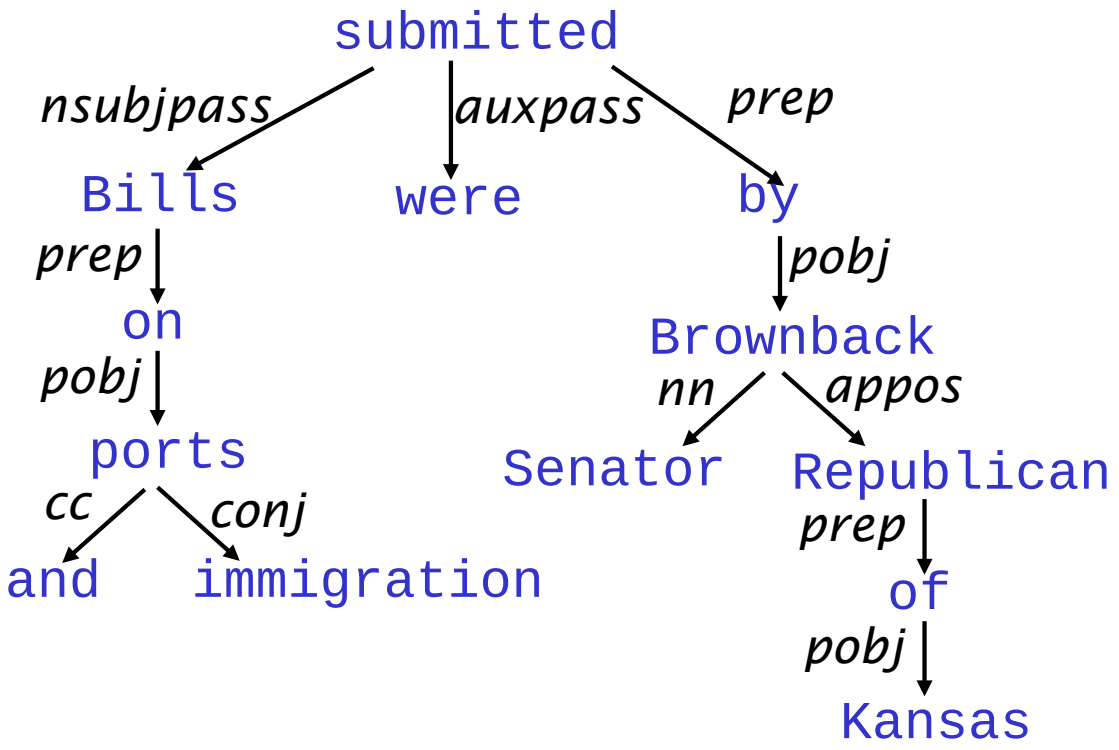


Dependency Grammar and Dependency Structure

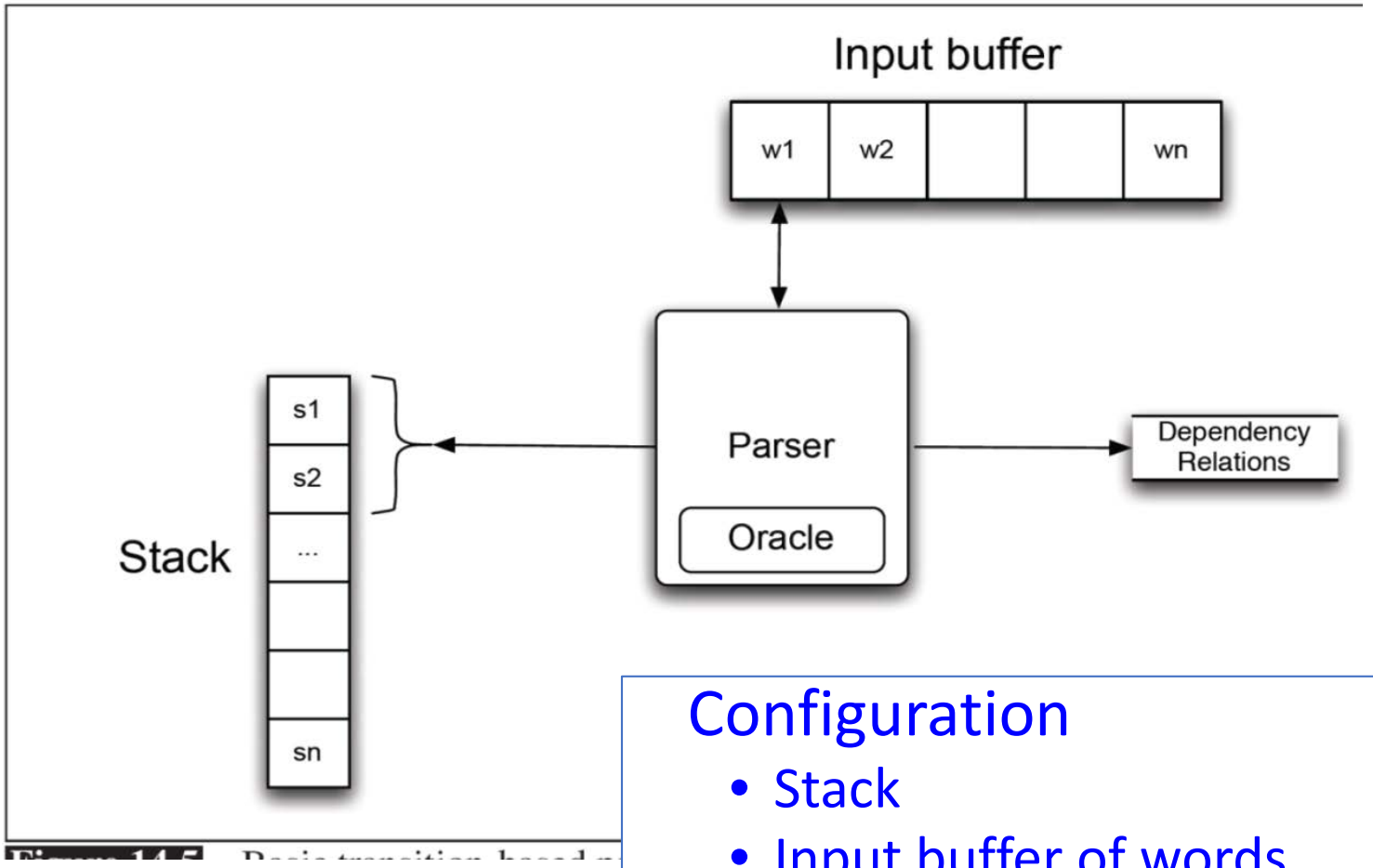
Dependency syntax postulates that syntactic structure consists of lexical items linked by binary asymmetric relations (“arrows”) called dependencies

The arrow connects a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate)

Usually, dependencies form a tree (connected, acyclic, single-head)



Transition-based dependency parsing



- The start state looks like this

  [[root], [word list], ()]

- A valid final state looks like
  - **[[root], [] (R)]**
  - R is the set of relations that we've discovered.
  - The [] Ian empty list represents the fact that all the words in the sentence are accounted for.

Configuration

- Stack
- Input buffer of words
- Set of dependency relations

Arc Standard Transition System

Defines 3 transition operators

- SHIFT
  - Remove word at head of input buffer
  - Push it on the stack
- LEFT-ARC
  - create head-dependent rel. between word at top of stack and 2ndword (under top)
  - remove 2ndword from stack
- RIGHT-ARC:
  - Create head-dependent rel. between word on 2ndword on stack and word on top
  - Remove word at top of stack

[[root], [I went home], ()]

→

[[root, I], [went home], ()]

[[root, I, went], [home], ()]

→

[[root, went], [home], (went, I)]

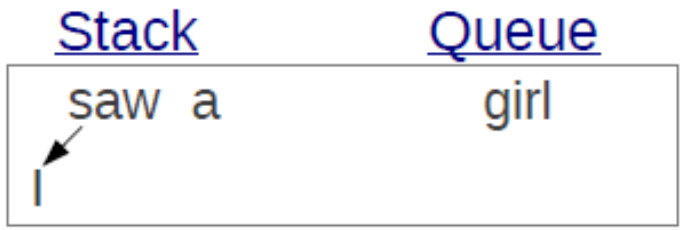
[[root, went, home], [], ()]

→

[[root, went], [], (went, home)]

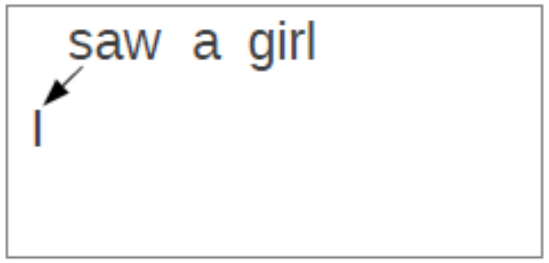
Classification for Shift-Reduce

- Given a state:

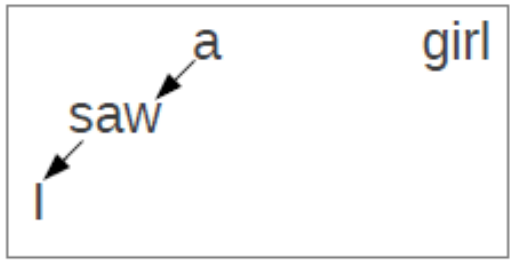


- Which action do we choose?

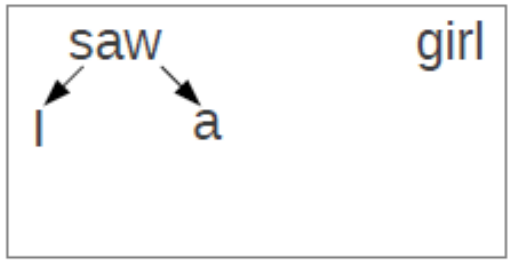
shift ?



r left ?



r right ?



- Correct actions → correct tree

Labeled Relations

- Label the relations like subject, direct object, indirect object, etc.

- Add new transitions
  - Essentially replace Left and Right with
    {Left, Right} X {all the relations of interest}

- So if there are 40 kinds of dependency relations
  - 81 transition operations possible

As 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 classifier over each legal move

  3 untyped choices; $|R| * 2 + 1$ when typed
- To apply ML in situations like this we have three problems
  1. Discovering features that are useful indicators of what to do in any situation
     - Characteristics of the state we're in
  2. Acquiring the necessary training data
     - Treebanks
  3. Training a classifier

Features

Compute features of the current configuration of the stack, buffer and A.

- Word (in stack, buffer, partial tree)
- POS of word
- Other features: Length of dependency arc

1. Part of speech of the top of the stack, POS of the third word in the remainder list, lemmas, last three letters
2. Head word of a word, number of relations already attached to a word, does the word already have a SUBJ relation, etc.

Algorithm

Given the current state (i.e., stack, buffer and A) predict the next action.

- Greedy classifier (no search involved)
  - At each stage ask the classifier to predict the next transition.
  - Select the best legal transition and apply it.

- O(N) in length of sentence.

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

- Input: Gold standard reference parse of a sentence
  - Contains the set of valid dependency relations R_P
- Simulate the operation of the parser by running the algorithm and relying on a new **training oracle** to give us correct transition operators for each successive configuration.
- Begin with a default initial configuration
  - Stack contains the ROOT
  - Buffer contains the list of words
  - Relations is empty.
- Compute correct sequence of “oracle” transitions
  - Use a “shortest stack” oracle which always prefers LeftArc over Shift.

Training Data

- Use the reference parse to guide the selection of operators as the parser steps through a sequence of configurations.
- Training oracle: At each stage it chooses as follows

1. LeftArc if the relation to be added is in the reference tree.
2. RightArc 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

LeftArc(r): *if* $(S_1 \mathrm{r} S_2) \in R_P$

RightArc(r): *if* $(S_2 \mathrm{r} S_1) \in R_P$ and $\forall\, r', w$ s.t. $(S_1 \mathrm{r}' w) \in R_P$ *then* $(S_1 \mathrm{r}' w) \in R_C$

Shift: *otherwise*

Book the flight through Travels

Root

Book the flight through Travels

Book the flight through Travels

Rina helps people in danger

Rina helps people in danger

Evaluation

- If we have a test set from a treebank and if we represent parses as a list of relations

(booked, I) (booked, flight) (flight, a) (flight, morning)

- **Unlabeled attachment score (UAS)** is the fraction of words (tokens) that are assigned the right head.
- **Labeled attachment score (LAS)** is the fraction of words assigned the right head with the right relation.

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:

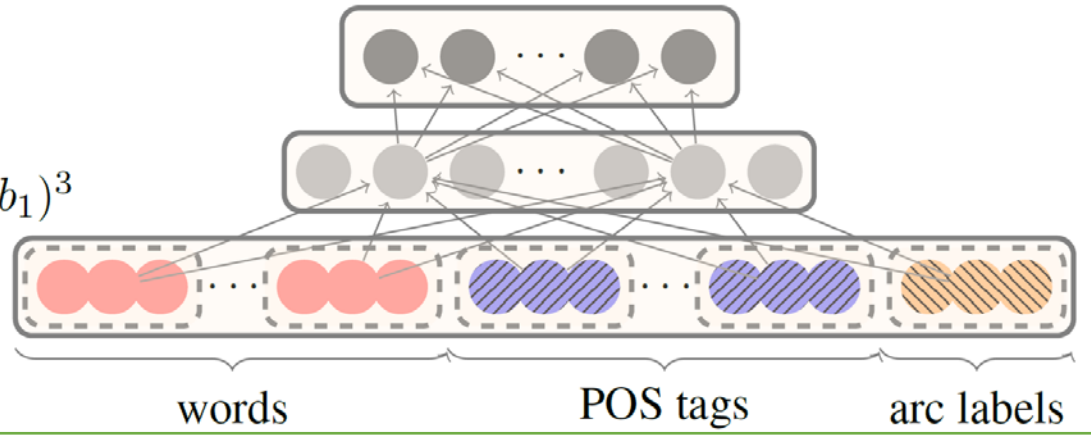
$$p = \texttt{softmax}(W_2 h)$$

Hidden layer:

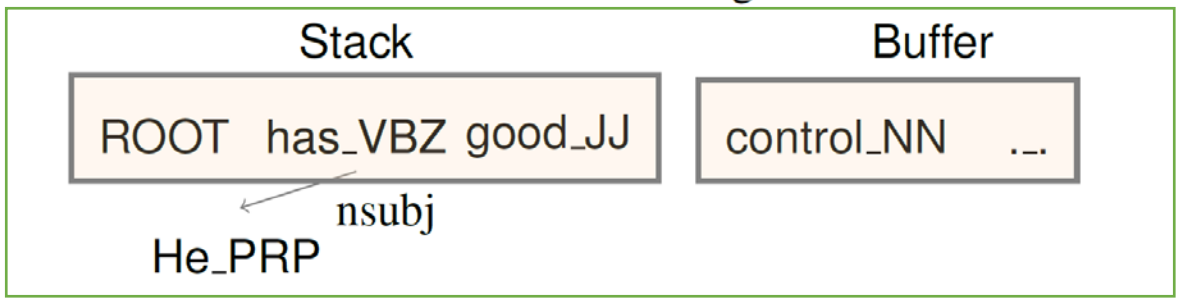
$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

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

Parse action classification

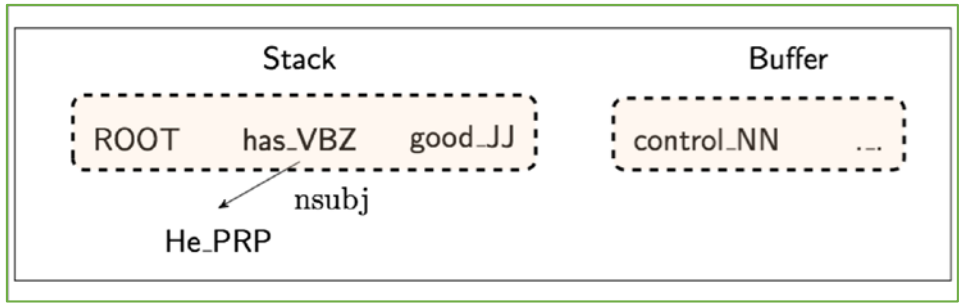


Configuration



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.