Dependency Parsing

THE UNIVERSITY OF UTAH

Outline

Two formalisms for syntactic structure: Phrase structure and dependencies

Two algorithms for dependency parsing

- Transition based dependency parsing
- Graph based dependency parsing

Evaluating dependencies

Dependency parsing

- Input: Sentence, tokenized + a dummy ROOT word
- Output: A dependency tree
  - Each word in the sentence is a node
  - Every word (except ROOT) should have an incoming edge indicating its head word
  - Only one word should be a dependent of ROOT
  - There are no cycles

Dependency parsing

- Input: Sentence, tokenized + a dummy ROOT word
- Output: A dependency tree
  - Each word in the sentence is a node
  - Every word (except ROOT) should have an incoming edge indicating its head word
  - Only one word should be a dependent of ROOT
  - There are no cycles
- Dependency theory also allows arrows to cross
  - Trees no arrows cross are called projective
  - Otherwise, they are called non-projective

Projective parse tree: No crossing dependency arcs when the words are laid out in their linear order, with all arcs above the word

Two families of parsing algorithms

Transition-based parsing

Graph based parsing

Two families of parsing algorithms

Transition-based parsing

- A generalization of the idea of shift-reduce parsing
- Greedily build attachments, using classifiers to decide which attachments to perform next
- Before neural networks: MaltParser (Nivre et al 2008)
- After neural networks: Chen and Manning (2014), Kipperwaser and Goldberg (2017)

Graph based parsing

Two families of parsing algorithms

Transition-based parsing

- A generalization of the idea of shift-reduce parsing
- Greedily build attachments, using classifiers to decide which attachments to perform next
- Before neural networks: MaltParser (Nivre et al 2008)
- After neural networks: Chen and Manning (2014), Kipperwaser and Goldberg (2017)

Graph based parsing

- Score all possible pairs of dependencies using a classifier
- Use a minimum spanning tree algorithm to find the best labeled tree
- Before neural networks: MSTParser (McDonald et al, 2005)
- With neural networks: Dozat and Manning (2017)

Two families of parsing algorithms

Transition-based parsing

- A generalization of the idea of shift-reduce parsing
- Greedily build attachments, using classifiers to decide which attachments to perform next
- Before neural networks: MaltParser (Nivre et al 2008)
- After neural networks: Chen and Manning (2014), Kipperwaser and Goldberg (2017)

Graph based parsing

- Score all possible pairs of dependencies using a classifier
- Use a minimum spanning tree algorithm to find the best labeled tree
- Before neural networks: MSTParser (McDonald et al, 2005)
- With neural networks: Dozat and Manning (2017)

Other algorithms exist as well. E.g. Eisner’s algorithm is a dynamic programming approach

Outline

Two formalisms for syntactic structure: Phrase structure and dependencies

Two algorithms for dependency parsing

- Transition based dependency parsing
- Graph based dependency parsing

Evaluating dependencies

Transition based parsing

- What is transition based parsing?
- The arc-standard transition system
- An example
- Greedy parsing algorithm
- Model building
- Practical concerns

Transition based parsing

- What is transition based parsing?
- The arc-standard transition system
- An example
- Greedy parsing algorithm
- Model building
- Practical concerns

Transition based parsing

Simple greedy discriminative parser that executes a sequence of *actions* that update the *parse state*

Transition based parsing

Simple greedy discriminative parser that executes a sequence of *actions* that update the *parse state*

Parse state

1. A buffer that consists of the input words
2. A stack whose top elements represent the next words that will be connected with a dependency edge
3. A set of all dependency edges that have been created so far

Transition based parsing

Simple greedy discriminative parser that executes a sequence of *actions* that update the *parse state*

Parse state	Actions
1. A buffer that consists of the input words	Operate on a parse state to produce the next state
2. A stack whose top elements represent the next words that will be connected with a dependency edge	Behave like shift and reduce in a shift-reduce parser
	Shift moves a word from the buffer to the stack
3. A set of all dependency edges that have been created so far	Different kinds of reduce actions that produce dependency edges

Transition based parsing

Simple greedy discriminative parser that executes a sequence of *actions* that update the *parse state*

Parse state

1. A buffer that consists of the input words
2. A stack whose
   the next word
   with a depen
3. A set of all de
   been created so far

Actions

Operate on a parse state to produce the next state

-reduce parser

o the stack

produce

dependency edges

There are different kinds of transition systems whose behavior is defined by the set of actions.

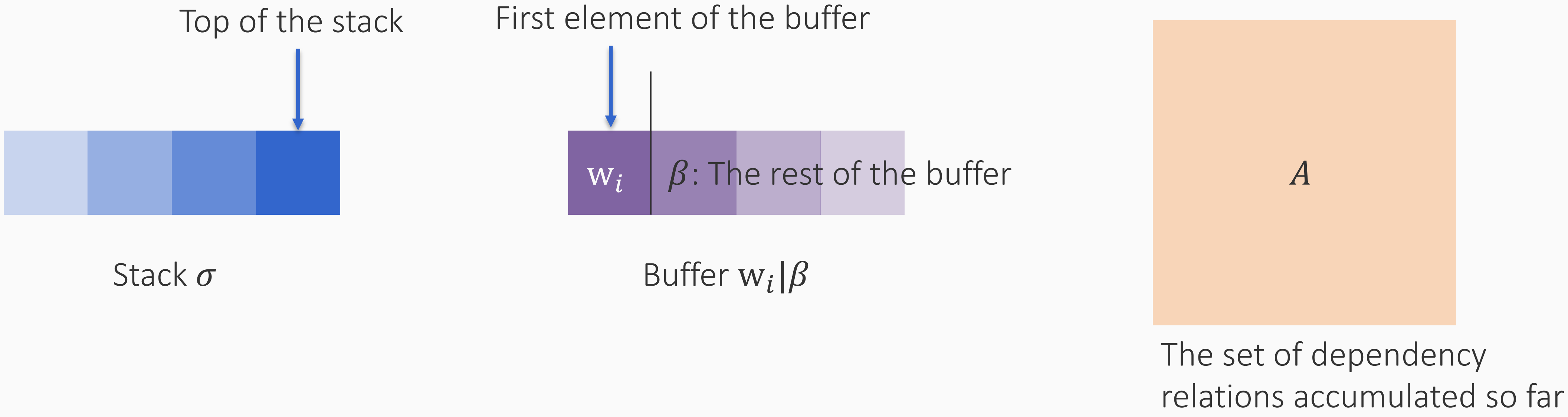
We will look at the Arc-Standard transition system which has three actions: shift, left-arc and right-arc

Transition based parsing

- What is transition based parsing?
- The arc-standard transition system
- An example
- Greedy parsing algorithm
- Model building
- Practical concerns

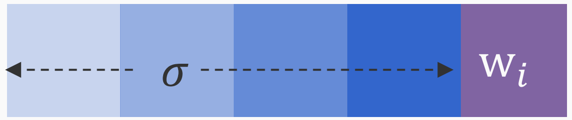
1. Shift

$\sigma, w_i \mid \beta, A$

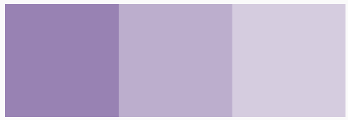


1. Shift

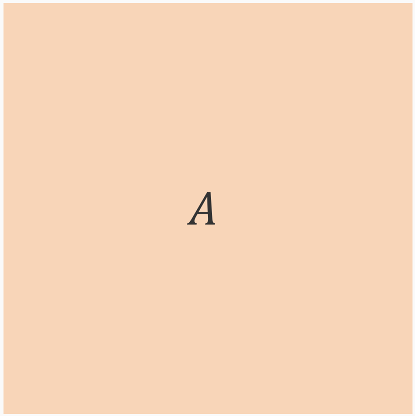
$$\sigma, w_i \mid \beta, A \rightarrow \sigma \mid w_i, \beta, A$$



Stack $\sigma|w_i$



Buffer β



The set of dependency relations accumulated so far

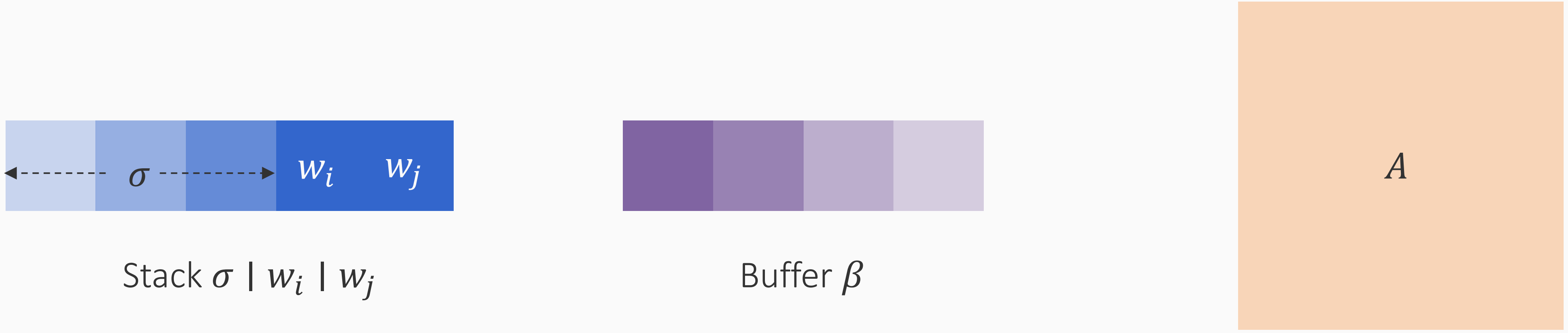
Add the first element of the buffer to the top of the stack

Remove the first element of the buffer

Keep the dependency relations unchanged

2. Left−arc$_r$

$\sigma \mid w_i \mid w_j, \beta, A$



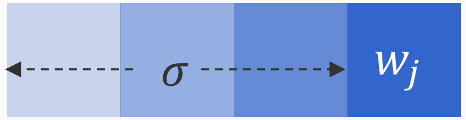
Stack $\sigma \mid w_i \mid w_j$

Buffer β

The set of dependency relations accumulated so far

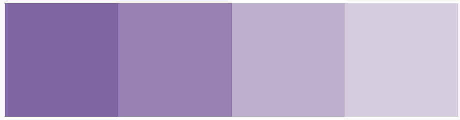
2. Left—arc$_r$

$$\sigma \mid w_i \mid w_j, \beta, A \rightarrow \sigma \mid w_j, \beta, A \cup \{r(w_j, w_i)\}$$



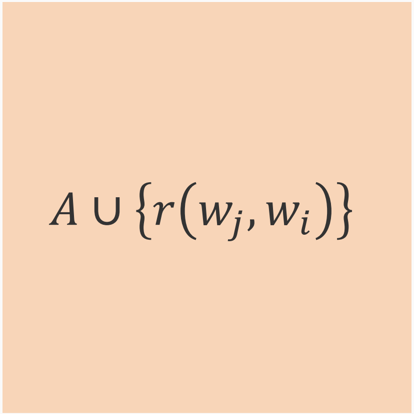
Stack $\sigma \mid w_j$

Remove the top two elements of the stack



Buffer β

Keep the buffer unchanged

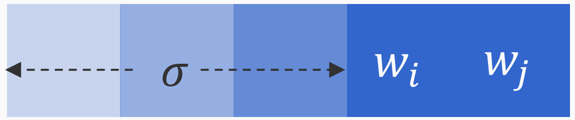


The set of dependency relations accumulated so far

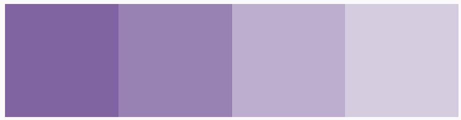
1. Add an edge from w_j to w_i with label r
2. Push w_j back on the stack

3. Right—arc$_r$

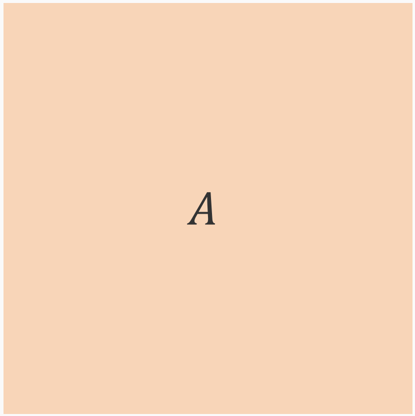
$\sigma \mid w_i \mid w_j, \beta, A$



Stack $\sigma \mid w_i \mid w_j$



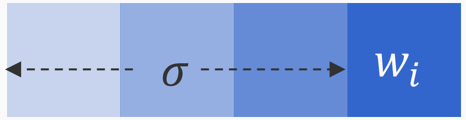
Buffer β



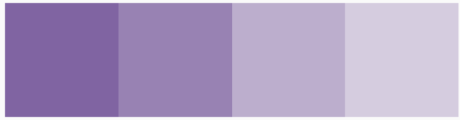
The set of dependency relations accumulated so far

3. Right—arc$_r$

$$\sigma \mid w_i \mid w_j, \beta, A \rightarrow \sigma \mid w_i, \beta, A \cup \{r(w_i, w_j)\}$$



Stack $\sigma \mid w_i$



Buffer β

$A \cup \{r(w_i, w_j)\}$

The set of dependency relations accumulated so far

Remove the top two elements of the stack

Keep the buffer unchanged

1. Add an edge from w_i to w_j with label r
2. Push w_i back on the stack

Transition based parsing

- What is transition based parsing?
- The arc-standard transition system
- An example
- Greedy parsing algorithm
- Model building
- Practical concerns

An example

Step	Stack	Buffer	Dependencies
0	root	The tabby cat scratched the couch	empty

To start things off:

- place all the words in the buffer.
- The stack contains only root.
- The set of dependencies is empty

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	`shift`

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	

The first element of the buffer moves to the stack

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	

The first element of the buffer moves to the stack

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The	scratched the couch	empty	

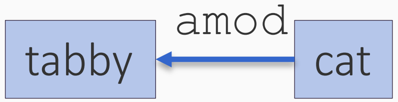
Take the top two elements of the stack

tabby cat

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The	scratched the couch	empty	

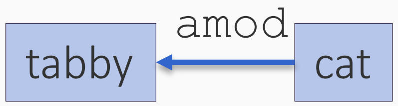
Add an edge to that goes to the left with the appropriate label



An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The	scratched the couch	amod(cat, tabby)	

Record that edge in the set of dependencies so far



An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The cat	scratched the couch	amod(cat, tabby)	Left-arc det

Place the top element of the stack from before back on the stack

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The cat	scratched the couch	amod(cat, tabby)	Left-arc det
5	root	scratched the couch	amod(cat, tabby)	

The cat

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The cat	scratched the couch	amod(cat, tabby)	Left-arc det
5	root	scratched the couch	amod(cat, tabby)	

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The cat	scratched the couch	amod(cat, tabby)	Left-arc det
5	root	scratched the couch	amod(cat, tabby) det(cat, the)	

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The cat	scratched the couch	amod(cat, tabby)	Left-arc det
5	root cat	scratched the couch	amod(cat, tabby) det(cat, the)	

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The cat	scratched the couch	amod(cat, tabby)	Left-arc det
5	root cat	scratched the couch	amod(cat, tabby) det(cat, the)	shift

An example

Step	Stack	Buffer	Dependencies	Next action
0	root	The tabby cat scratched the couch	empty	shift
1	root The	tabby cat scratched the couch	empty	shift
2	root The tabby	cat scratched the couch	empty	shift
3	root The tabby cat	scratched the couch	empty	Left-arc amod
4	root The cat	scratched the couch	amod(cat, tabby)	Left-arc det
5	root cat	scratched the couch	amod(cat, tabby) det(cat, the)	shift
6	root cat scratched	the couch	amod(cat, tabby) det(cat, the)	

An example

Step	Stack	Buffer	Dependencies	Next action
6	root, cat, scratched	the, couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`

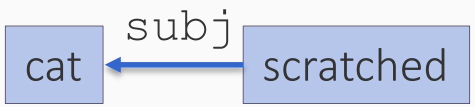
An example

Step	Stack	Buffer	Dependencies	Next action
6	root cat scratched	the couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root	the couch	amod(cat, tabby) det(cat, the)	

cat   scratched

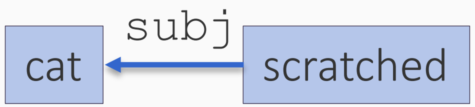
An example

Step	Stack	Buffer	Dependencies	Next action
6	root, cat, scratched	the, couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root	the, couch	amod(cat, tabby) det(cat, the)	



An example

Step	Stack	Buffer	Dependencies	Next action
6	root, cat, scratched	the, couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root	the, couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	



An example

Step	Stack	Buffer	Dependencies	Next action
6	root | cat | scratched	the | couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root | scratched	the | couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	

An example

Step	Stack	Buffer	Dependencies	Next action
6	root | cat | scratched	the | couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root | scratched	the | couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`

An example

Step	Stack	Buffer	Dependencies	Next action
6	root, cat, scratched	the, couch	amod(cat, tabby) det(cat, the)	Left-arc subj
7	root, scratched	the, couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	shift
8	root, scratched, the	couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	

An example

Step	Stack	Buffer	Dependencies	Next action
6	root cat scratched	the couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root scratched	the couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
8	root scratched the	couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`

An example

Step	Stack	Buffer	Dependencies	Next action
6	root, cat, scratched	the, couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root, scratched	the, couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
8	root, scratched, the	couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
9	root, scratched, the, couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat)	

An example

Step	Stack	Buffer	Dependencies	Next action
6	root | cat | scratched	the | couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root | scratched	the | couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
8	root | scratched | the	couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
9	root | scratched | the | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`Left-arc det`

An example

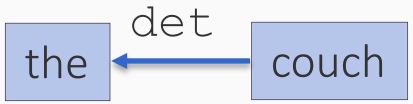
Step	Stack	Buffer	Dependencies	Next action
6	root | cat | scratched	the | couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root | scratched	the | couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
8	root | scratched | the	couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
9	root | scratched | the | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`Left-arc det`
10	root | scratched the couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat)	

An example

Step	Stack	Buffer	Dependencies	Next action
6	root, cat, scratched	the, couch	amod(cat, tabby) det(cat, the)	Left-arc subj
7	root, scratched	the, couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	shift
8	root, scratched, the	couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	shift
9	root, scratched, the, couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat)	Left-arc det
10	root, scratched	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat)	

An example

Step	Stack	Buffer	Dependencies	Next action
6	root, cat, scratched	the, couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root, scratched	the, couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
8	root, scratched, the	couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
9	root, scratched, the, couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`Left-arc det`
10	root, scratched	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	



An example

Step	Stack	Buffer	Dependencies	Next action
6	root | cat | scratched	the | couch	amod(cat, tabby) det(cat, the)	`Left-arc subj`
7	root | scratched	the | couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
8	root | scratched | the	couch	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`shift`
9	root | scratched | the | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat)	`Left-arc det`
10	root | scratched | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	

An example

Step	Stack	Buffer	Dependencies	Next action
10	root | scratched | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	

An example

Step	Stack	Buffer	Dependencies	Next action
10	root | scratched | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	`Right-arc obj`

An example

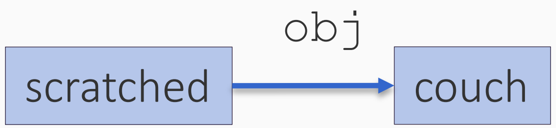
Step	Stack	Buffer	Dependencies	Next action
10	root | scratched | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	`Right-arc obj`
11	root	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	

scratched couch

Take the top two elements of the stack

An example

Step	Stack	Buffer	Dependencies	Next action
10	root | scratched | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	`Right-arc obj`
11	root	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch)	



Add an edge going to the right
with the appropriate label

An example

Step	Stack	Buffer	Dependencies	Next action
10	root | scratched | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	`Right-arc obj`
11	root | scratched	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch)	

Record the new edge in the set of dependencies and place the *second* element back on the stack

An example

Step	Stack	Buffer	Dependencies	Next action
10	root | scratched | couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	`Right-arc obj`
11	root | scratched	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch)	`Right-arc root`

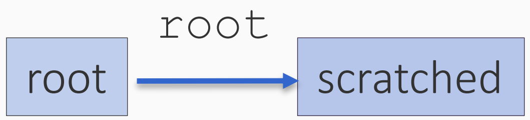
An example

Step	Stack	Buffer	Dependencies	Next action
10	root, scratched, couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	`Right-arc obj`
11	root, scratched	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch)	`Right-arc root`
11		empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch)	

root   scratched

An example

Step	Stack	Buffer	Dependencies	Next action
10	root, scratched, couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	Right-arc obj
11	root, scratched	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch)	Right-arc root
11	empty	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch) root(root, scratched)	



An example

Step	Stack	Buffer	Dependencies	Next action
10	root, scratched, couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	Right-arc obj
11	root, scratched	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch)	Right-arc root
11	root	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch) root(root, scratched)	

An example

Step	Stack	Buffer	Dependencies	Next action
10	root, scratched, couch	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the)	`Right-arc obj`
11	root, scratched	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch)	`Right-arc root`
11	root	empty	amod(cat, tabby) det(cat, the) subj(scratched, cat) det(couch, the) obj(scratched, couch) root(root, scratched)	

Stop when the stack contains only root and the buffer is empty

Transition based parsing

- What is transition based parsing?
- The arc-standard transition system
- An example
- Greedy parsing algorithm
- Model building
- Practical concerns

The parsing algorithm

Input: A tokenized sentence

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:

3. Return $state$

The parsing algorithm

Input: A tokenized sentence

Stack

1. Set $state$ $\leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:

3. Return $state$

The parsing algorithm

Input: A tokenized sentence

Stack Buffer

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:

3. Return $state$

The parsing algorithm

Input: A tokenized sentence

Stack — Buffer — Dependencies

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:

3. Return $state$

The parsing algorithm

Input: A tokenized sentence

Stack   Buffer   Dependencies

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:

The stack has only root,
and the buffer is empty

3. Return $state$

The parsing algorithm

Input: A tokenized sentence

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:
   1. Choose $action \leftarrow NextAction(state)$
   2. Set $state \leftarrow Apply(state, action)$

3. Return $state$

The parsing algorithm

Input: A tokenized sentence

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:
   1. Choose $action \leftarrow NextAction(state)$
   2. Set $state \leftarrow Apply(state, action)$

3. Return $state$

Action can be one of shift, labeled left-arc or labeled right-arc.

If the dependency formalism has L labels, then this action will be one of 1 + 2L possible options

The parsing algorithm

Input: A tokenized sentence

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:
   1. Choose $action \leftarrow NextAction(state)$
   2. Set $state \leftarrow Apply(state, action)$

3. Return $state$

Action can be one of shift, labeled left-arc or labeled right-arc.

If the dependency formalism has L labels, then this action will be one of 1 + 2L possible options

Typically, a classifier over the action set

The parsing algorithm

Input: A tokenized sentence

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:
   1. Choose $action \leftarrow NextAction(state)$
   2. Set $state \leftarrow Apply(state, action)$

3. Return $state$

Action can be one of shift, labeled left-arc or labeled right-arc.

If the dependency formalism has L labels, then this action will be one of 1 + 2L possible options

Typically, a classifier over the action set

This is a greedy algorithm. Once it takes an action, it does not back track.

Transition based parsing

- What is transition based parsing?
- The arc-standard transition system
- An example
- Greedy parsing algorithm
- Model building
- Practical concerns

The parsing algorithm

Input: A tokenized sentence

1. Set $state \leftarrow \{[root], [words], []\}$

2. While $state$ is not the final state:
   1. Choose $action \leftarrow NextAction(state)$
   2. Set $state \leftarrow Apply(state, action)$

3. Return $state$

The parser behavior is defined by its how it chooses the action to take at each state.

This can be framed as a multi-class classification problem

Choosing the next action

Given a parse state, what action should be taken next?

Input: The entire parse state, i.e., the stack, buffer and dependencies so far

Output: Shift, Left-arc$_r$ or Right-arc$_r$ for different dependency labels r

How would you approach this modeling problem?

The action classifier

A multiclass classifier, whose inputs is features from the state, and the label space is the set of all possible actions.

Key design choices:

- What features?
- What classifier?

Featurizing the stack, the buffer and dependencies

Stack		
root	cat	scratched

Buffer	
the	couch

Dependencies

amod(cat, tabby)
det(cat, the)

What information can we get from such a configuration?

- Words in the stack and the buffer: Here, `cat`, `scratched` from the stack and `the`, `couch` from the buffer
- Any properties of these words such as parts of speech (assuming that this is available)
- The positions of words on the stack and the buffer. E.g. `cat` is at position 2 on the stack
- Previously generated children of the words on the stack. Here, we know from the existing dependencies that `cat` has two children—`tabby` and `the`—with labels `amod` and `det` respectively

All this information could contribute to features for this configuration

Indicator features versus embedding features

The typical featurizing strategy: Take the top 1-3 words from the stack and the buffer and extract features from them.

Pre-neural era: Indicator features. E.g. a one-hot vector representing the fact that "second element of the stack = cat, POS of second element on stack = Noun, first element on stack = scratched"

- Sparse, very high dimensional features
- Feature computation can be slow

Neural era:

- All the words are represented by word embeddings
- POS tags and other information like dependency labels (if they are used) can also be represented as embeddings that will get trained along the way
- All these vectors can be combined by concatenating them

The action classifier

A multiclass classifier, whose inputs is features from the state, and the label space is the set of all possible actions.

Key design choices:

- What features? For neural models: embeddings of words, POS tags, dependency labels
- What classifier?

What classifier?

The model can be any neural network architecture provided the final layer is assigns a probability or score to actions. For example, a two-layer neural network

What classifier?

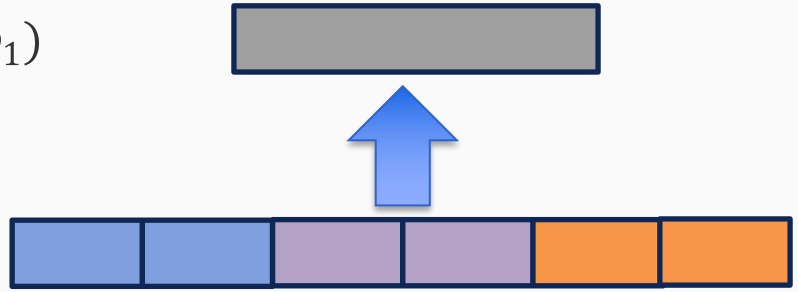
The model can be any neural network architecture provided the final layer is assigns a probability or score to actions. For example, a two-layer neural network

The input layer x consisting of concatenated embeddings from the stack, buffer, dependencies, etc

What classifier?

The model can be any neural network architecture provided the final layer is assigns a probability or score to actions. For example, a two-layer neural network

A hidden layer, e.g. $\mathrm{h} = ReLU(W_1 x + b_1)$



The input layer x consisting of concatenated embeddings from the stack, buffer, dependencies, etc

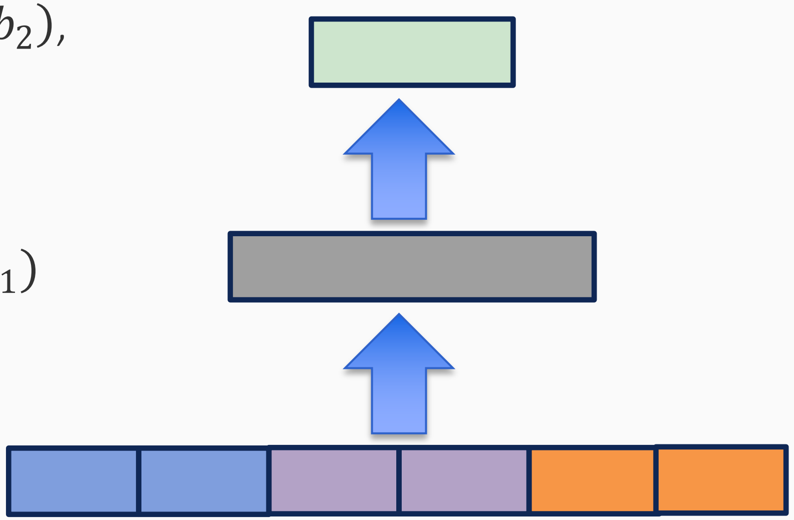
What classifier?

The model can be any neural network architecture provided the final layer is assigns a probability or score to actions. For example, a two-layer neural network

The output layer, e.g. $softmax(W_2h + b_2)$, that produces probabilities over actions

A hidden layer, e.g. $\mathrm{h} = ReLU(W_1x + b_1)$

The input layer x consisting of concatenated embeddings from the stack, buffer, dependencies, etc



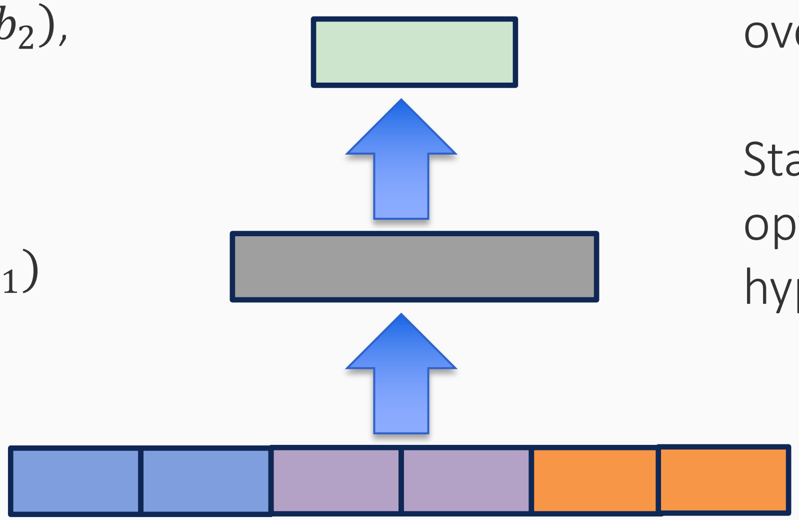
What classifier?

The model can be any neural network architecture provided the final layer is assigns a probability or score to actions. For example, a two-layer neural network

The output layer, e.g. $softmax(W_2h + b_2)$, that produces probabilities over actions

A hidden layer, e.g. $\mathrm{h} = ReLU(W_1x + b_1)$

The input layer x consisting of concatenated embeddings from the stack, buffer, dependencies, etc



Training such a neural network will involve minimizing cross-entropy loss over a training set

Standard training considerations apply: optimizers, learning rates, batch sizes, hyper-parameter selection, etc.

Training the model with a treebank

Treebanks contain `(sentence, tree)` pairs

But to train a model that maps parse states to actions, we need training data with `(configuration, action)` pairs

Before any training is done, we need to convert the trees in the treebank to the form our model knows about

- This requires us to first use a training oracle that looks at parse configuration and a reference tree and decides what action to take next
- See Jurafsky and Martin's book chapter for details

Transition based parsing

- What is transition based parsing?
- The arc-standard transition system
- An example
- Greedy parsing algorithm
- Model building
- Practical concerns

Odds and ends

- We saw the Arc-Standard transition system (defined by the three kinds of actions). There are other transition systems
- Arc-standard transition parsing produces projective trees. How to address this?
  - Do nothing, lose accuracy points on any non-projective trees in the data
  - Change the dependency formalism to one that is only projective
  - Use some sort of post-processing to fix edges that ought to be non-projective
  - Change the transition system to include additional actions to handle non-projectivity
  - Use a graph-based parser where this does not matter

Transition-based dependency parsing: Summary

- Transition based parsing: The parse tree is constructed by applying actions to configurations
  - We saw the arc-standard transition system, but there are others
- The modeling question: What action to pick for the current configuration?
  - Previously: Linear models
  - More recently: Neural models. Can use any embedding—word2vec, Glove type static embeddings, or also more modern embeddings generated by models like BERT
- Greedy parsing algorithm:
  - The model produces scores/probabilities over the next action. The algorithm we saw greedily selects the action predicted by the model at each step. Could lead to error propagation
  - Beam search is an alternative approach: Instead of greedily picking the next transition, keep a *beam* of size *k*. The beam represents the *k* best sequences of actions so far
  - We will see beam search in a later lecture