

CSC485/2501 A1 Tutorial #1

Zhewei Sun



UNIVERSITY OF
TORONTO

Assignment 1

- Is now available!
- Asks you to implement a set of neural dependency parsers.
- Due on Fri. Oct. 6th, at 11:59 pm.

Assignment 1

- Note: Remote GPU access on teach.cs is currently unavailable due to an ongoing system upgrade.
- An announcement will be made on Piazza if this changes.
- Instructions on how to access GPUs via Slurm:
<https://www.teach.cs.toronto.edu/using-labs/remote-gpu-computing/>

Assignment 1

- Part 1: Transition-based dependency parser
- Part 2: Graph-based dependency parser

Assignment 1

- Part 1: Transition-based dependency parser
 - We will focus on this part today.
- Part 2: Graph-based dependency parser

Outline

- Dependency Parsing Example
 - Obtaining the necessary parsing steps for a dependency tree.
- Gap Degree Example
- Neural Dependency Parser
 - With PyTorch pointers 😊

Quiz

- Which of the following ***is not*** a well-formedness criterion for dependency graphs?
 - a) Every node has at most one dependent.
 - b) Every node has at most one head.
 - c) The graph is weakly connected.
 - d) The graph contains no cycles.

Quiz

- Which of the following ***is not*** a well-formedness criterion for dependency graphs?
 - a) Every node has at most one dependent.
 - b) Every node has at most one head.
 - c) The graph is weakly connected.
 - d) The graph contains no cycles.

Transition-based Parser - Review

- Dependency parser: Given a sentence, output a dependency parse tree.
- Three things to keep track of:
 1. A **stack of words** being processed.
 2. A **buffer of words** to be eventually pushed onto the stack.
 3. A **list of predicted dependencies** (i.e. arcs).

Transition-based Parser - Review

- Three possible operations:
 1. **SHIFT**: removes the first word from the buffer and pushes it onto the stack.
 2. **LEFT-ARC**: marks the second-from-top item (i.e., second-most recently added word) on the stack as a dependent of the first item and removes the second item from the stack.
 3. **RIGHT-ARC**: marks the top item (i.e., most recently added word) on the stack as a dependent of the second item and removes the first item from the stack.

SHIFT Operation

- Removes the first word from the buffer and pushes it onto the stack.
- Step T:
 - **Stack**: [ROOT, John, saw]; **Buffer**: [dogs, yesterday]
- Step T+1:
 - **Stack**: [ROOT, John, saw, dogs]; **Buffer**: [yesterday]
 - **Action**: SHIFT

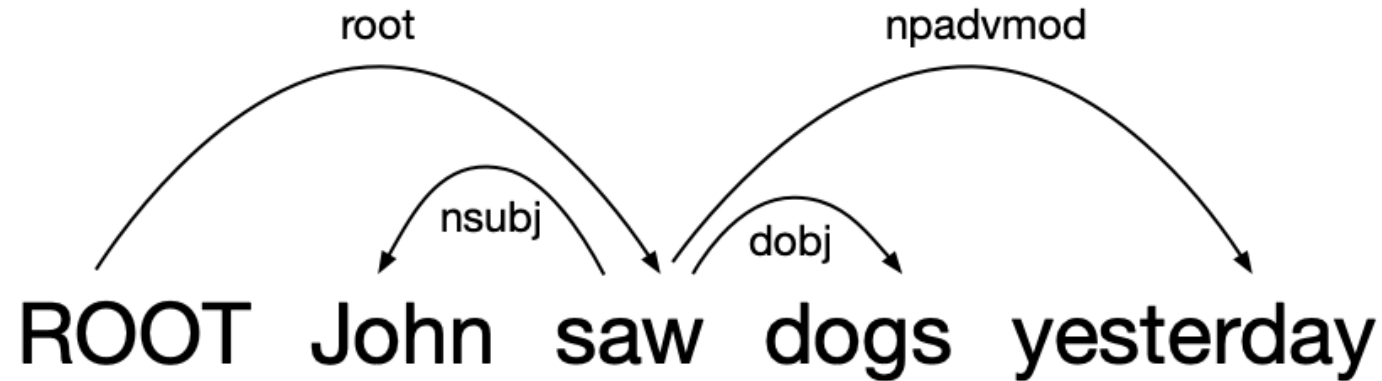
LEFT-ARC Operation

- Marks the second-from-top item (i.e., second-most recently added word) on the stack as a dependent of the first item and removes the second item from the stack.
- Step T:
 - **Stack**: [ROOT, John, saw]; **Buffer**: [dogs, yesterday]
- Step T+1:
 - **Stack**: [ROOT, saw]; **Buffer**: [dogs, yesterday]
 - **New Dependency**: saw → John, nsubj
 - **Action**: LEFT-ARC

RIGHT-ARC Operation

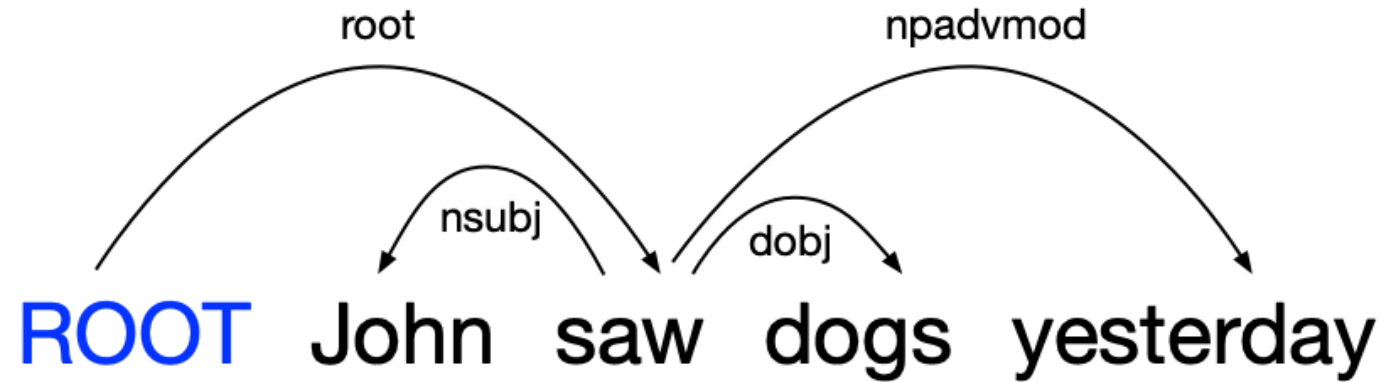
- Marks the top item (i.e., most recently added word) on the stack as a dependent of the second item and removes the first item from the stack.
- Step T:
 - **Stack**: [ROOT, saw, dogs]; **Buffer**: [yesterday]
- Step T+1:
 - **Stack**: [ROOT, saw]; **Buffer**: [yesterday]
 - **New Dependency**: saw → dogs, dobj
 - **Action**: RIGHT-ARC

Dependency Parsing Example



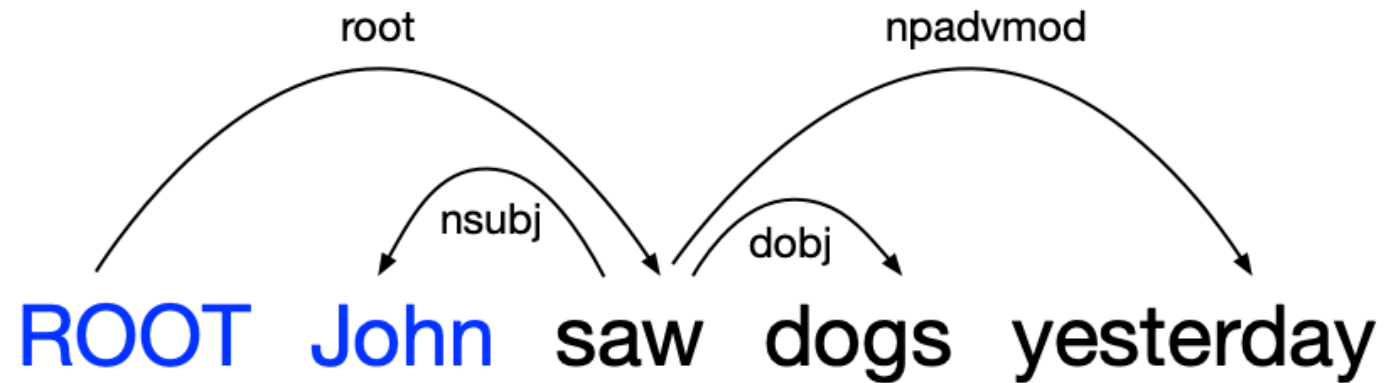
- Given a dependency tree, figure out the intermediate parsing steps.
- Check the top of your stack to see whether it is appropriate to create an arc.
- After creating an arc, record it, and then remove the dependent word from the stack.

Dependency Parsing Example



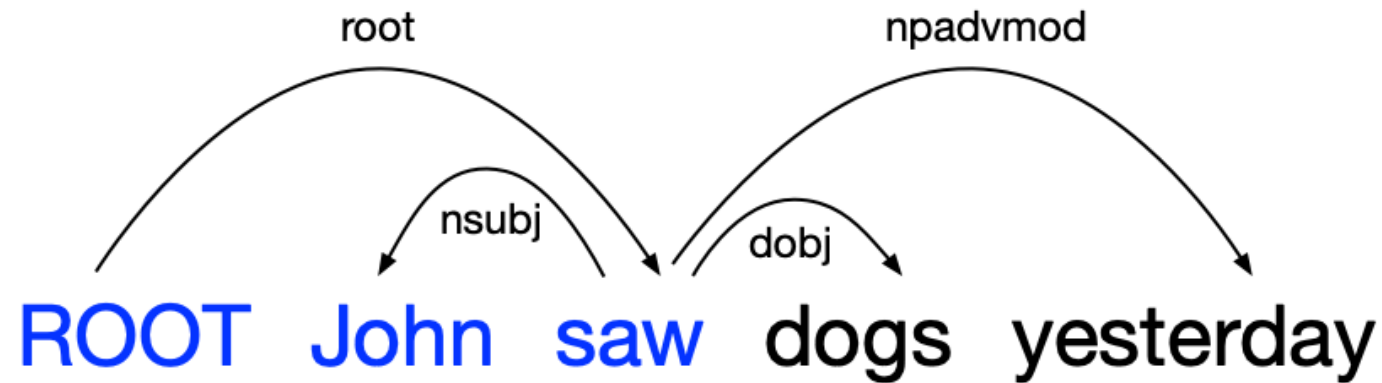
- Step 0:
 - **Stack:** [ROOT]; **Buffer:** [John, saw, dogs, yesterday]

Dependency Parsing Example



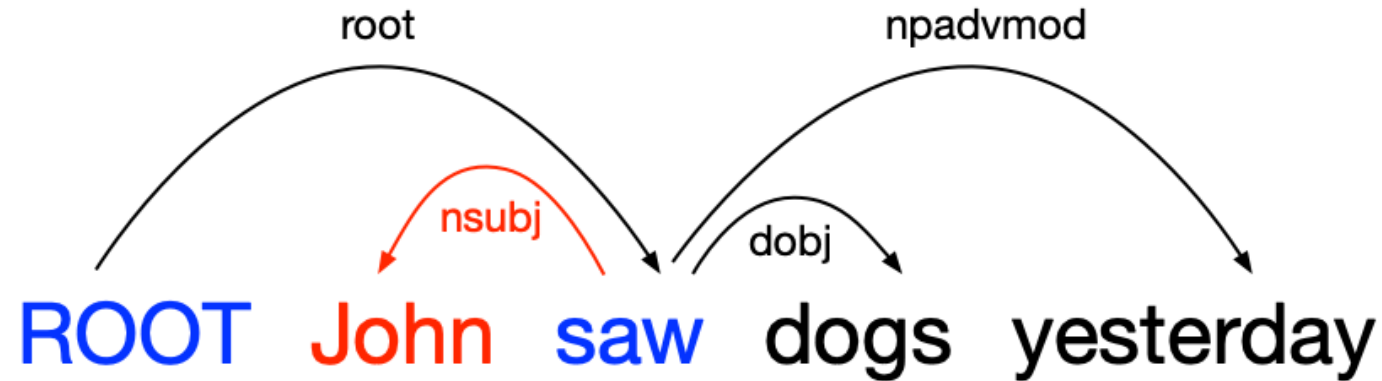
- Step 0:
 - **Stack**: [ROOT]; **Buffer**: [John, saw, dogs, yesterday]
- Step 1:
 - **Stack**: [ROOT, John]; **Buffer**: [saw, dogs, yesterday]
 - **New Dependency**: None
 - **Action**: SHIFT

Dependency Parsing Example



- From Step 1:
 - **Stack**: [ROOT, John]; **Buffer**: [saw, dogs, yesterday]
- Step 2:
 - **Stack**: [ROOT, John, saw]; **Buffer**: [dogs, yesterday]
 - **New Dependency**: None
 - **Action**: SHIFT

Dependency Parsing Example

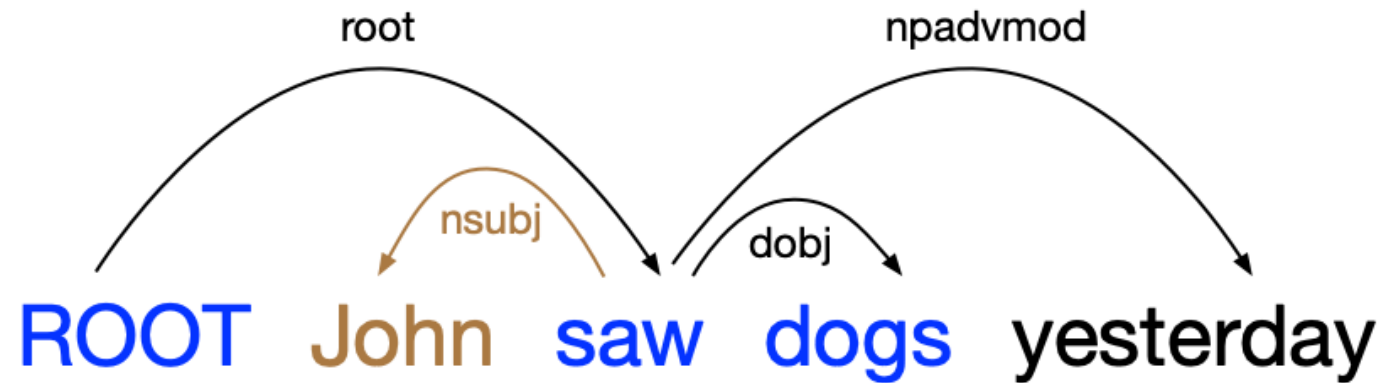


- From Step 2:
 - **Stack**: [ROOT, John, saw]; **Buffer**: [dogs, yesterday]
- Step 3:
 - **Stack**: [ROOT, saw]; **Buffer**: [dogs, yesterday]
 - **New Dependency**: saw → John, nsubj
 - **Action**: LEFT-ARC

For this assignment:

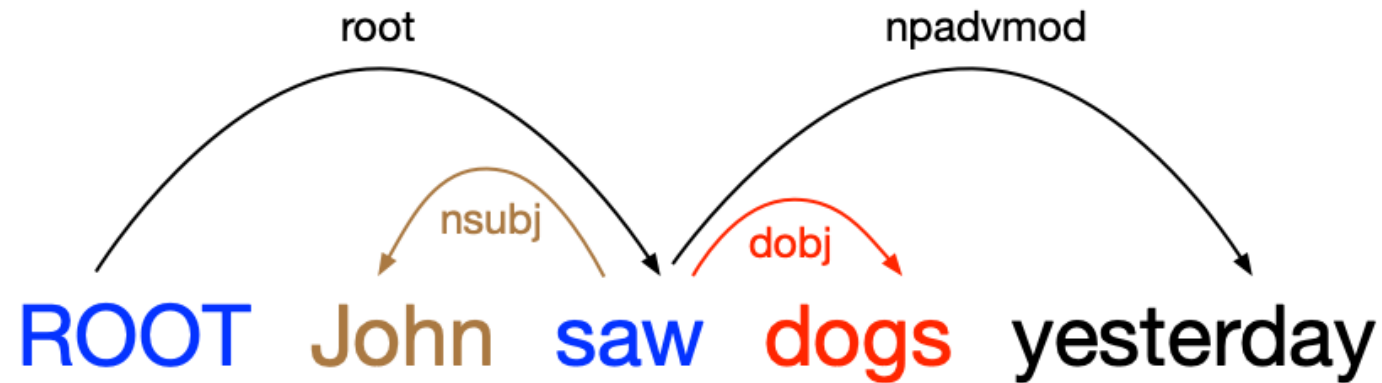
Choose LEFT-ARC over SHIFT when both are valid and generate the same tree.

Dependency Parsing Example



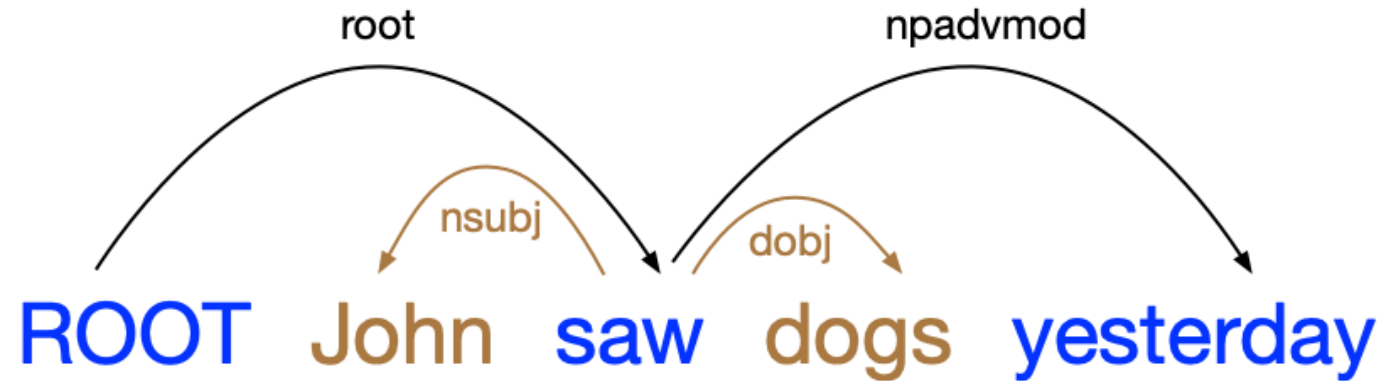
- From Step 3:
 - **Stack**: [ROOT, saw]; **Buffer**: [dogs, yesterday]
- Step 4:
 - **Stack**: [ROOT, saw, dogs]; **Buffer**: [yesterday]
 - **New Dependency**: None
 - **Action**: SHIFT

Dependency Parsing Example



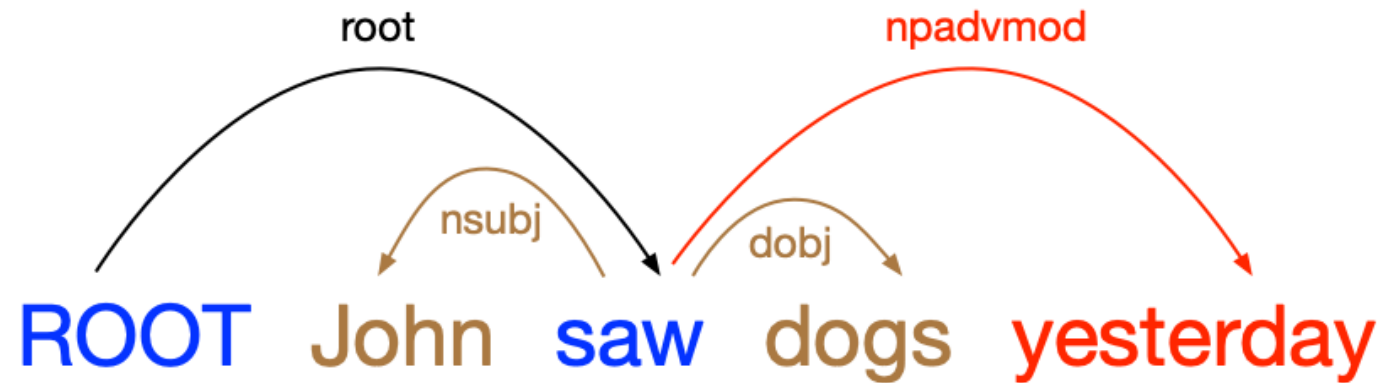
- From Step 4:
 - **Stack**: [ROOT, saw, dogs]; **Buffer**: [yesterday]
- Step 5:
 - **Stack**: [ROOT, saw]; **Buffer**: [yesterday]
 - **New Dependency**: saw → dogs, dobj
 - **Action**: RIGHT-ARC

Dependency Parsing Example



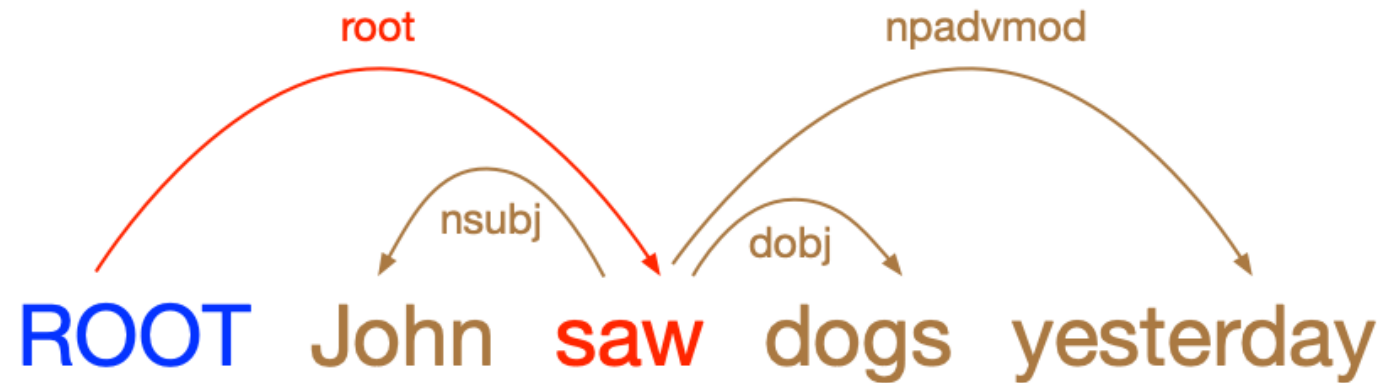
- From Step 5:
 - **Stack**: [ROOT, saw]; **Buffer**: [yesterday]
- Step 6:
 - **Stack**: [ROOT, saw, yesterday]; **Buffer**: []
 - **New Dependency**: None
 - **Action**: SHIFT

Dependency Parsing Example



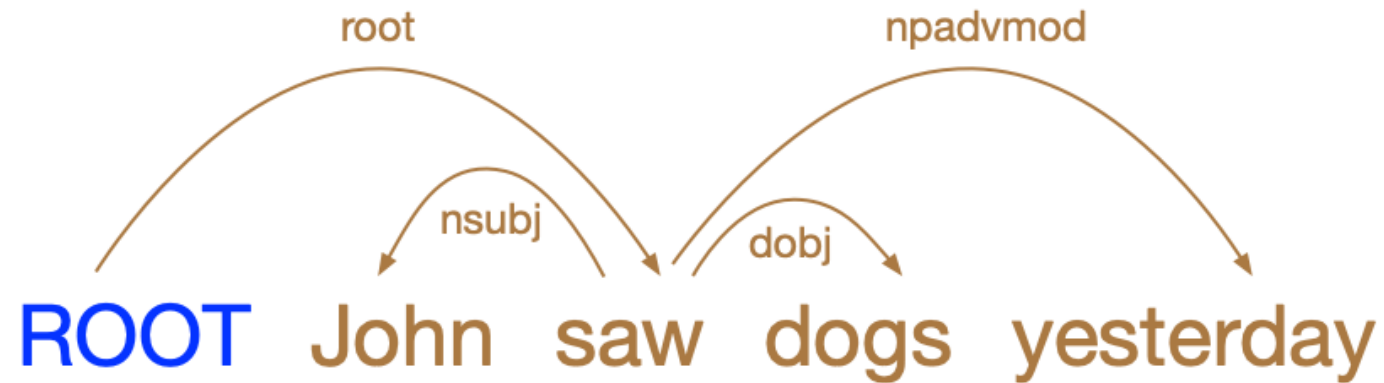
- From Step 6:
 - **Stack**: [ROOT, saw, yesterday]; **Buffer**: []
- Step 7:
 - **Stack**: [ROOT, saw]; **Buffer**: []
 - **New Dependency**: saw -> yesterday, npadvmod
 - **Action**: RIGHT-ARC

Dependency Parsing Example



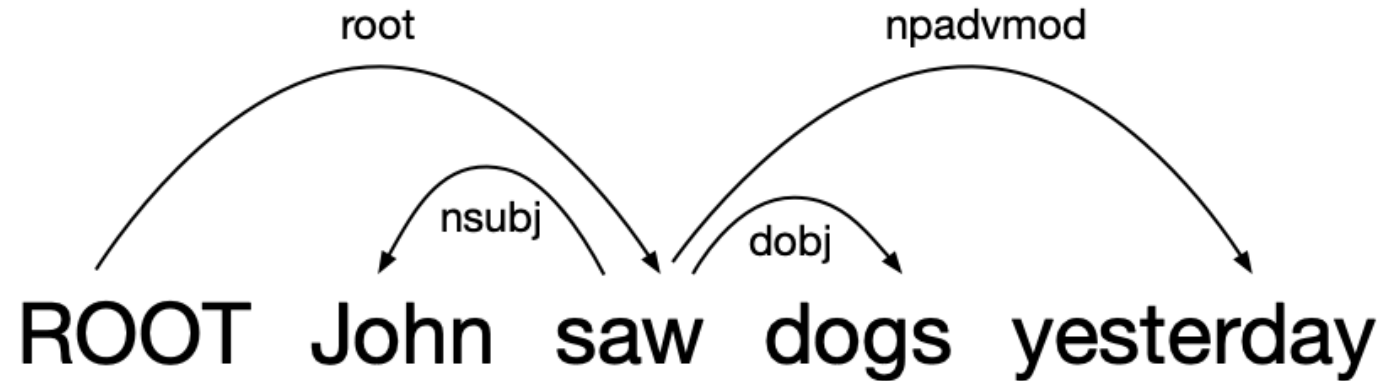
- From Step 7:
 - **Stack:** [ROOT, saw]; **Buffer:** []
- Step 8:
 - **Stack:** [ROOT]; **Buffer:** []
 - **New Dependency:** ROOT → saw, root
 - **Action:** RIGHT-ARC

Dependency Parsing Example



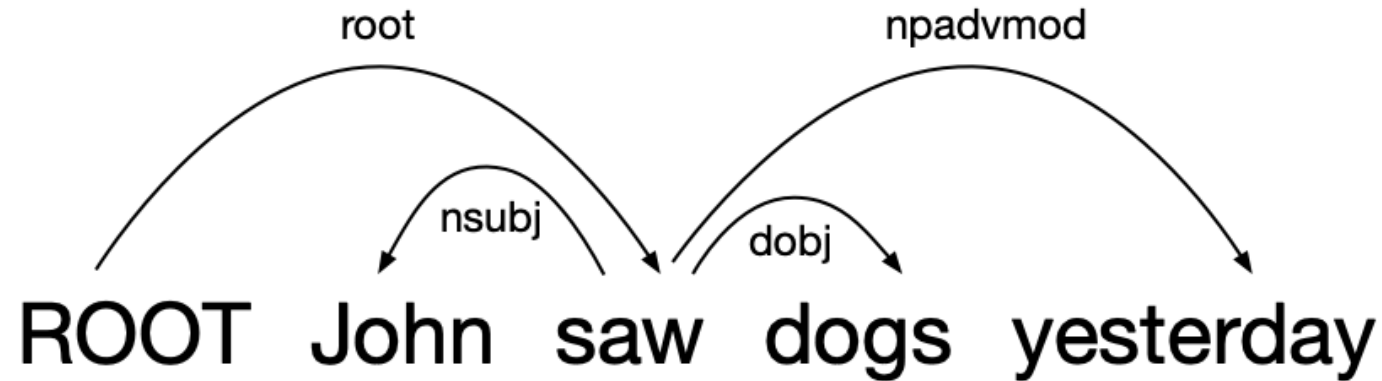
- We've figured out all the parsing steps!
- Similar exercise in the assignment.
- How to do this algorithmically? What are the conditions?

Gap Degree Example



- The **gap degree of a word** in a dependency tree is the least k for which the subsequence consisting of the word and its descendants (both direct and indirect) is entirely comprised of $k + 1$ maximally contiguous substrings. Equivalently, the gap degree of a word is the *number of gaps* in the subsequence formed by the word and all of its descendants, regardless of the *size* of the gaps.
- The **gap degree of a dependency tree** is the greatest gap degree of any word in the tree.

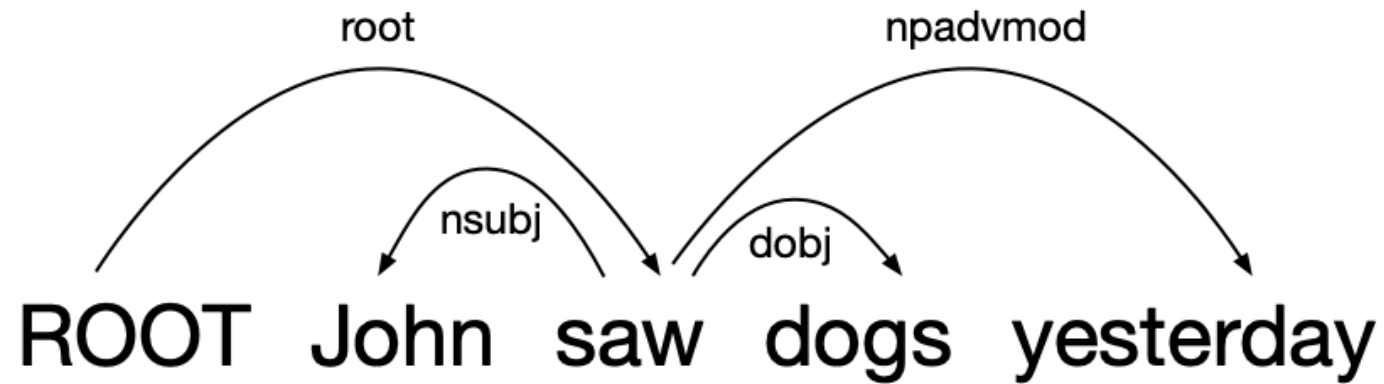
Gap Degree Example



- For each word, check the substring consisting itself and all its descendants:
 - ROOT: ROOT John saw dogs yesterday
 - John: John
 - saw: John saw dogs yesterday
 - dogs: dogs
 - yesterday: yesterday

All substrings are contiguous!
 $k = 0$

Neural Dependency Parser



- Now assume we don't have the dependency tree.

Neural Dependency Parser

ROOT John saw dogs yesterday

- Now assume we don't have the dependency tree.
- When do we need to make decisions when parsing?

Neural Dependency Parser

ROOT John saw dogs yesterday

- Suppose we have the following partial parse:
 - **Stack**: [ROOT, John, saw]; **Buffer**: [dogs, yesterday]
- Now we need to decide which transition to do next:
 - a) **SHIFT**: Shift dogs onto the stack
 - b) **LEFT-ARC**: create the arc: saw → john
 - c) **RIGHT-ARC**: create the arc john → saw

Neural Dependency Parser

ROOT John saw dogs yesterday

- Use a neural network to make a prediction at each parse step.
- Implement this in PyTorch, read the docs if you're not familiar:
 - <https://pytorch.org/docs/stable/index.html>

Neural Dependency Parser

ROOT John saw dogs yesterday

- Input: Word level features (e.g. word embeddings) for each word in the sentence.
 - *torch.nn.Embedding(size, shape)*
 - *torch.nn.Embedding.from_pretrained(...)*
 - Make sure you DON'T freeze the pre-trained embeddings!!

Neural Dependency Parser

ROOT John saw dogs yesterday

- Input: Word level features (e.g. word embeddings) for each word in the sentence.
- One linear (fully-connected) hidden layer.
 - `hidden_layer = torch.nn.Linear(input_size, output_size)`
 - To apply: `hidden_layer(features)`
- Also checkout `torch.nn.relu(...)` and `torch.nn.dropout(...)`

Neural Dependency Parser

ROOT John saw dogs yesterday

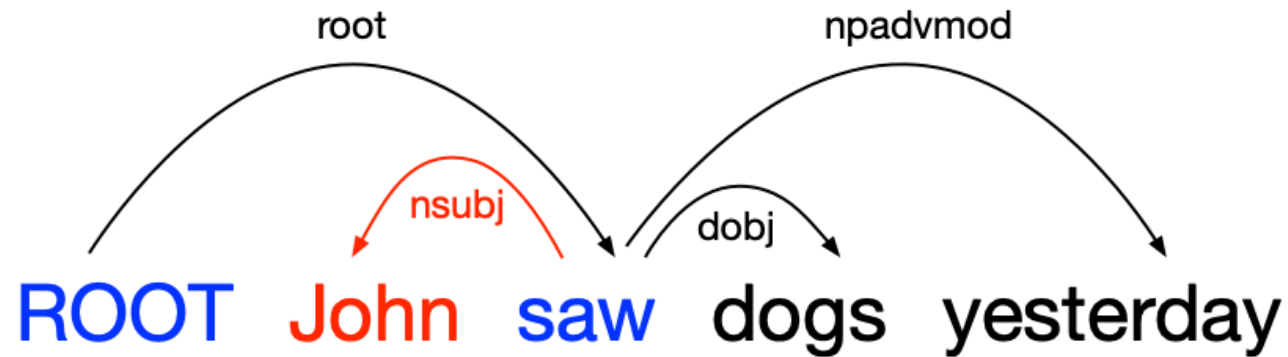
- Input: Word level features (e.g. word embeddings) for each word in the sentence.
- One linear (fully-connected) hidden layer.
- A softmax layer to obtain a probability distribution over transitions.
 - *torch.nn.CrossEntropyLoss / torch.nn.functional.CrossEntropy*

Neural Dependency Parser



- Suppose our neural network gives us an answer:
 - a) **SHIFT**: Shift dogs onto the stack
 - b) **LEFT-ARC**: create the arc: saw → john
 - c) **RIGHT-ARC**: create the arc john → saw
- How can we tell whether we have made the right choice?

Neural Dependency Parser



- How can we tell whether we have made the right choice?
 - Implement an "oracle" that peaks into the parsed tree and tells us the correct transition to make.
- Think about the first example we did in this tutorial.
 - How to make the process automatic?
 - What conditions need to be met to make a particular transition?

To be continued...

- The transition-based parser can only handle projective parse trees (think about why this is the case).
- Next time, we will look at graph-based dependency parsing, which accounts for non-projective trees.