7. Dependency Parsing

LING-581-Natural Language Processing 1

Instructor: Hakyung Sung

September 16, 2025

*Acknowledgment: These course slides are based on materials from CS224N @ Stanford University

Table of contents

Review

Review

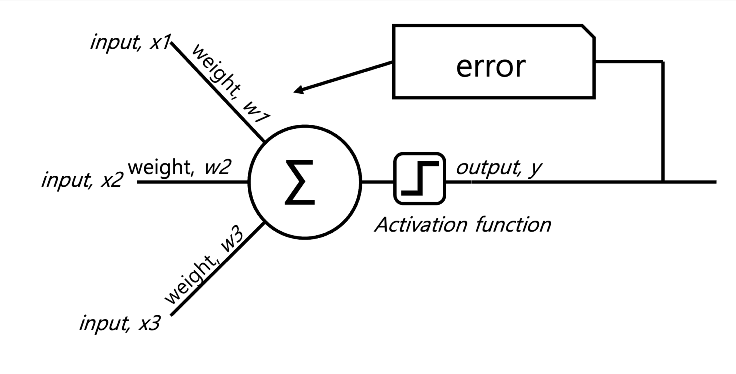
- GloVe
- Artificial neural network
- Perceptrons
- MLP
- Gradient descendant and loss function
- Backpropagation

Review: GloVe

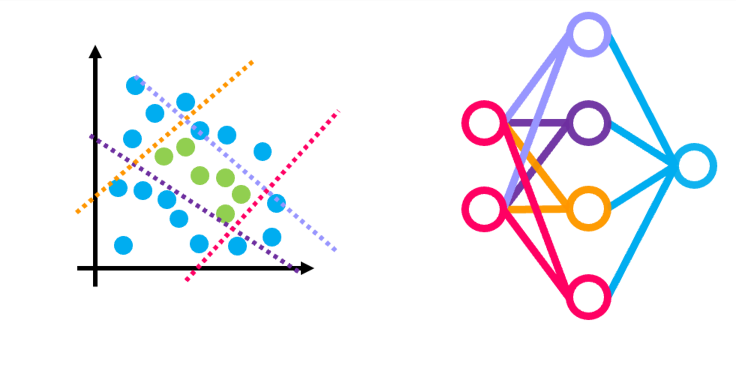
Find word vectors $\vec{w}_{\text{ice}}$, $\vec{w}_{\text{steam}}$ such that:

$$(\vec{w}_{\text{ice}} - \vec{w}_{\text{steam}}) \cdot \vec{w}_x \approx \log \frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$$

Review: Perceptron

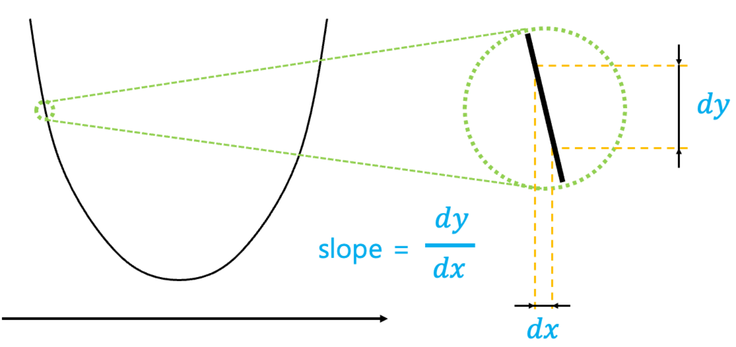


Review: MLP



- activation function
- optimization
- algorithm

Review: Gradient descent



Review: Gradient descent

Gradient descent learning rule:

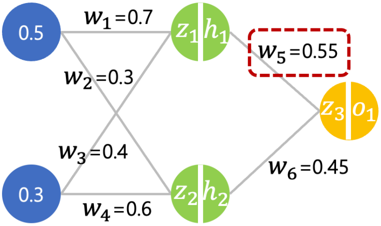
$$w_{\text{new}} = w_{\text{current}} - \eta \cdot \frac{\partial L}{\partial w}$$

- η: learning rate
- L: loss function
- $\frac{\partial L}{\partial w}$: gradient of the loss function with respect to weight

Review: Learning rule

1. **Feedforward**: compute outputs
2. **Loss calculation**: evaluate error
3. **Backpropagation**: propagate errors backward

Review: Backpropagation



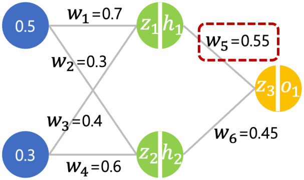
Gradient descent learning rule:

$$w_{\text{new}} = w_{\text{current}} - \eta \cdot \frac{\partial L}{\partial w}$$

- L: loss function
- $\frac{\partial L}{\partial w}$: gradient of the loss function w.r.t. weight

Review: Chain rule

Since this derivative cannot be computed directly, we apply the chain rule.



$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Lesson plan

Lesson plan

- Syntactic structure: Consistency and dependency
- Dependency grammar and treebanks
- Dependency parsing
- Transition-based dependency parsing
- Neural dependency parsing

Syntactic structure

Understanding linguistic structure

- So far, we have focused on how NLP deals with word meanings.

Understanding linguistic structure

- So far, we have focused on how NLP deals with word meanings.
- But natural language understanding goes beyond individual words.

Understanding linguistic structure

- So far, we have focused on how NLP deals with word meanings.
- But natural language understanding goes beyond individual words.
- Linguistic structure is equally important for capturing how words combine together to create meaning.

Understanding linguistic structure

A grammar is the system of rules that defines how linguistic structures are formed and how words relate to each other within a sentence.

Understanding linguistic structure

A grammar is the system of rules that defines how linguistic structures are formed and how words relate to each other within a sentence.

1. Part of Speech (POS)

Understanding linguistic structure

A grammar is the system of rules that defines how linguistic structures are formed and how words relate to each other within a sentence.

1. Part of Speech (POS)
2. Dependency grammar

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.
- POS is also called lexical category.

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.
- POS is also called lexical category.
- Main POS categories (in English):

1. Part-of-Speech (POS)

- A word’s POS determines how it fits into a sentence.
- POS is also called lexical category.
- Main POS categories (in English):
  - `N___`: reindeer, game, government

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.
- POS is also called lexical category.
- Main POS categories (in English):
  - `N___`: reindeer, game, government
  - `V___`: play, run, believe

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.
- POS is also called lexical category.
- Main POS categories (in English):
    - `N___`: reindeer, game, government
    - `V___`: play, run, believe
    - `A___`: fun, beautiful

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.
- POS is also called lexical category.
- Main POS categories (in English):
  - N___: reindeer, game, government
  - V___: play, run, believe
  - A___: fun, beautiful
  - A___: well, heavily

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.
- POS is also called lexical category.
- Main POS categories (in English):
  - N___: reindeer, game, government
  - V___: play, run, believe
  - A___: fun, beautiful
  - A___: well, heavily
  - P___: on, into

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.
- POS is also called lexical category.
- Main POS categories (in English):
    - `N___`: reindeer, game, government
    - `V___`: play, run, believe
    - `A___`: fun, beautiful
    - `A___`: well, heavily
    - `P___`: on, into
    - `A___/D___`: a, the, some

1. Part-of-Speech (POS)

- A word's POS determines how it fits into a sentence.
- POS is also called lexical category.
- Main POS categories (in English):
    - `N___`: reindeer, game, government
    - `V___`: play, run, believe
    - `A___`: fun, beautiful
    - `A___`: well, heavily
    - `P___`: on, into
    - `A___/D___`: a, the, some
    - `C___`: and, or

How do we identify POS?

- Examples:

How do we identify POS?

- Examples:
    - *This car is very interesting.*

How do we identify POS?

- Examples:
  - *This car is very interesting.*
  - *This car mooked fast.*

How do we identify POS?

- Examples:
  - *This car is very interesting.*
  - *This car mooked fast.*
  - *This nony car mooked fast.*

How do we identify POS?

- Examples:
  - *This car is very interesting.*
  - *This car mooked fast.*
  - *This nony car mooked fast.*
- We usually identify POS by:

How do we identify POS?

- Examples:
  - *This car is very interesting.*
  - *This car mooked fast.*
  - *This nony car mooked fast.*
- We usually identify POS by:
  - **Morphology:** how a word changes form (e.g., verbs mark tense: *play* → *played*, sometimes irregularly: *go* → *went*)

How do we identify POS?

- Examples:
  - *This car is very interesting.*
  - *This car mooked fast.*
  - *This nony car mooked fast.*
- We usually identify POS by:
  - **Morphology:** how a word changes form (e.g., verbs mark tense: *play* → *played*, sometimes irregularly: *go* → *went*)
  - **Distribution:** where a word appears in a sentence (e.g., nouns after articles, verbs after subjects)

2. From words to phrases

- Words combine into constituents based on POS:

2. From words to phrases

- Words combine into constituents based on POS:
  - **the reindeer** = article + noun = noun phrase

- Words combine into constituents based on POS:
    - **the reindeer** = article + noun = noun phrase
    - **play games** = verb + noun phrase = verb phrase

2. From words to phrases

- Words combine into constituents based on POS:
  - **the reindeer** = article + noun = noun phrase
  - **play games** = verb + noun phrase = verb phrase
- Constituents combine based on **phrasal category**:

2. From words to phrases

- Words combine into constituents based on POS:
  - **the reindeer** = article + noun = noun phrase
  - **play games** = verb + noun phrase = verb phrase
- Constituents combine based on **phrasal category**:
  - **Noun Phrase** + **Verb Phrase** = Sentence

Structure over meaning

- Chomsky (1957): *“Colorless green ideas sleep furiously”*

Structure over meaning

- Chomsky (1957): *“Colorless green ideas sleep furiously”*
- Nonsensical meaning, but:

Structure over meaning

- Chomsky (1957): *"Colorless green ideas sleep furiously"*
- Nonsensical meaning, but:
    - Correct lexical and phrasal categories

Structure over meaning

- Chomsky (1957): *“Colorless green ideas sleep furiously”*
- Nonsensical meaning, but:
    - Correct lexical and phrasal categories
    - Grammatically well-formed

Structure over meaning

- Chomsky (1957): *"Colorless green ideas sleep furiously"*
- Nonsensical meaning, but:
    - Correct lexical and phrasal categories
    - Grammatically well-formed
- Syntax is about **structure**, not always meaning.

Understanding linguistic structure 1: Constituency

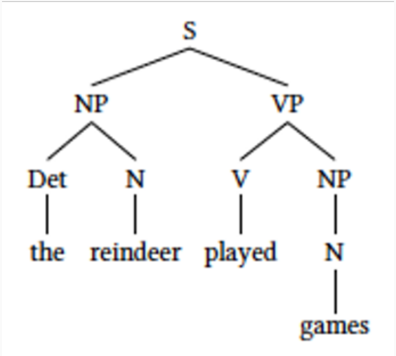
- Constituency grammar

Lexicon:

N → *reindeer, dragon, lunch, game, evening, morning*
V(trans) → *play, eat*
V(intrans) → *run, swim, dance*
Adj → *fun, beautiful, interesting*
Det → *the, a, some, many*
P → *for, in, to, at*

Phrase structure rules:

S → NP VP
VP → V(trans) NP
VP → V(intrans)
NP → Det (A*) N
NP → N
NP → NP PP
PP → P NP

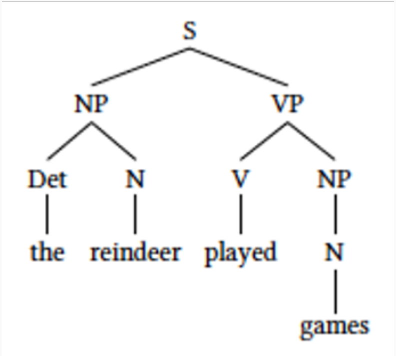


Understanding linguistic structure 1: Constituency

- Constituency grammar
- A linguistic theory that analyzes sentences as nested constituents (e.g., noun phrases, verb phrases).

Lexicon:
N → *reindeer, dragon, lunch, game, evening, morning*
V(trans) → *play, eat*
V(intrans) → *run, swim, dance*
Adj → *fun, beautiful, interesting*
Det → *the, a, some, many*
P → *for, in, to, at*

Phrase structure rules:
S → NP VP
VP → V(trans) NP
VP → V(intrans)
NP → Det (A*) N
NP → N
NP → NP PP
PP → P NP

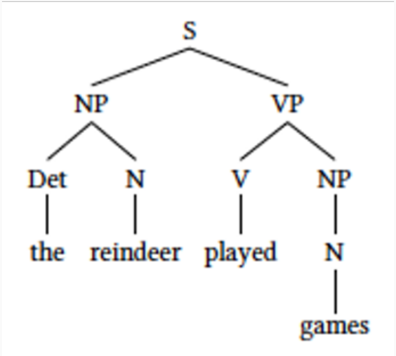


Understanding linguistic structure 1: Constituency

- Constituency grammar
- A linguistic theory that analyzes sentences as nested constituents (e.g., noun phrases, verb phrases).
- Also known as phrase structure grammar

Lexicon:
N → *reindeer, dragon, lunch, game, evening, morning*
V(trans) → *play, eat*
V(intrans) → *run, swim, dance*
Adj → *fun, beautiful, interesting*
Det → *the, a, some, many*
P → *for, in, to, at*

Phrase structure rules:
S → NP VP
VP → V(trans) NP
VP → V(intrans)
NP → Det (A*) N
NP → N
NP → NP PP
PP → P NP



Frameworks for analyzing grammar

- Linguists formalize sentence structure using grammar frameworks:
    - Phrase Structure Grammar

Frameworks for analyzing grammar

- Linguists formalize sentence structure using grammar frameworks:
    - Phrase Structure Grammar (linguistics)

Frameworks for analyzing grammar

- Linguists formalize sentence structure using grammar frameworks:
    - Phrase Structure Grammar (linguistics)
    - Dependency Grammar (widely used in NLP)

Understanding linguistic structure 2: Dependency

- Dependency grammar

Understanding linguistic structure 2: Dependency

- Dependency grammar
- It postulates that syntactic structure consists of relationships between lexical items, normally binary asymmetric relations ("arrows") called **dependencies**.

Understanding linguistic structure 2: Dependency

- Dependency grammar
- It postulates that syntactic structure consists of relationships between lexical items, normally binary asymmetric relations ("arrows") called **dependencies**.
- Dependency structure shows which words depend on (modify, attach to, or are arguments of) which other words.

Why do we need dependency structure?

- Humans communicate complex ideas by composing words together into bigger units into convey complex meanings.

Why do we need dependency structure?

- Humans communicate complex ideas by composing words together into bigger units into convey complex meanings.
- Readers/Listeners/NLP models need to work out what modifies (attaches to) what.

Why do we need dependency structure?

- Humans communicate complex ideas by composing words together into bigger units into convey complex meanings.
- Readers/Listeners/NLP models need to work out what modifies (attaches to) what.
- e.g., *I saw the man with the telescope*

More example: Prepositional phrase attachment ambiguity

San Jose cops kill man with knife

Text Paper Translate Listen Close

San Jose cops kill man with knife

Ex-college football player, 23, shot 9 times allegedly charged police at fiancee's home

By Hamed Aleaziz and Vivian Ho

A man fatally shot by San Jose police officers while allegedly charging at them with a knife was a 23-year-old former football player at De Anza College in Cupertino who was distraught and depressed, his family said Thursday.

Police officials said two officers opened fire Wednesday afternoon on Phillip Watkins outside his fiancee's home because they feared for their lives. The officers had been drawn to the home, officials said, by a 911 call reporting an armed home invasion that, it turned out, had been made by Watkins himself.

But the mother of Watkins' fiancee, who also lives in the home on the 1300 block of Sherman Street, said she witnessed the shooting and described it as excessive. Faye Buchanan said the confrontation happened shortly after she called a suicide intervention hotline in hopes of getting Watkins medical help.

Watkins' 911 call came in at 5:01 p.m., said Sgt. Heather Randol, a San Jose police spokeswoman. "The caller stated there was a male breaking into his home armed with a knife," Randol said. "The caller also stated he was locked in an upstairs bedroom with his children and requested help from police."

She said Watkins was on the sidewalk in front of the home when two officers got there. He was holding a knife with a 4-inch blade and ran toward the officers in a threatening manner, Randol said.

"Both officers ordered the suspect to stop and drop the knife," Randol said. "The suspect continued to charge the officers with the knife in his hand. Both officers, fearing for their safety and defense of their life, fired at the suspect."

On the police radio, one officer said, "We have a male with a knife. He's walking toward us."

"Shots fired! Shots fired!" an officer said moments later.

A short time later, an officer reported, "Male is down. Knife's still in hand."

Buchanan said she had been prompted to call the

Shoot continues on D8

Back Continue

More example: Prepositional phrase attachment ambiguity

BBC

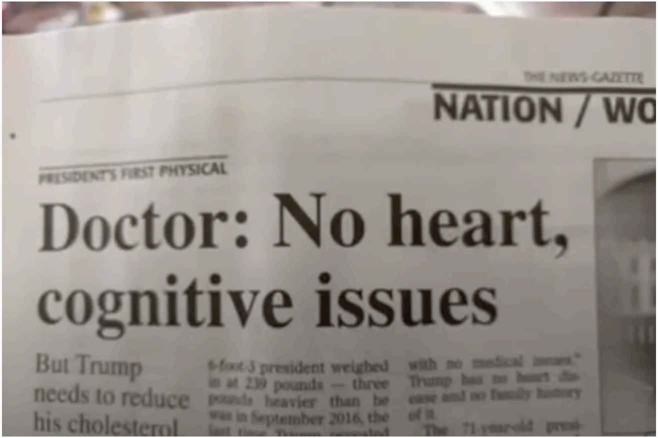
Scientists count whales from space

Hannah Cubaynes: "Boats and planes can't go everywhere, but satellites can" UK scientists have demonstrated the practicality of counting whales from space.

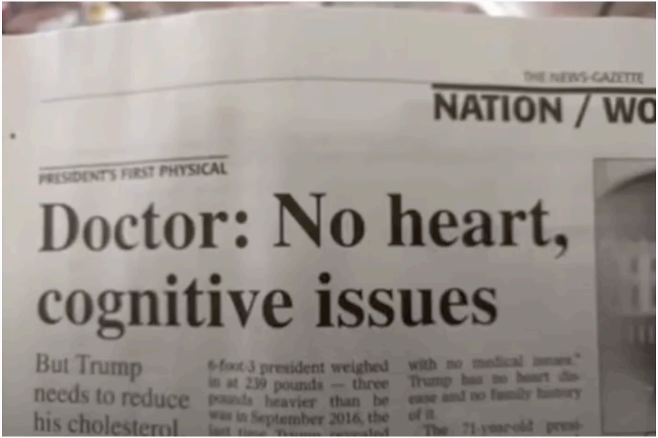
Oct 31, 2018

Sourced from: *https://www.bbc.com/news/science-environment-46046264*

More example: Coordination phrase attachment ambiguity



More example: Coordination phrase attachment ambiguity



- No heart, no cognitive issues?
- No heart, but cognitive issues?

More example: Adjectival/Adverbial modifier ambiguity

numbers, including some that featured a bucket and bells brigade [illegible] buckets and trash cans with drums sticks and hammer mallets. PHOTO BY JENNIFER STULTZ

MENTORING DAY

Students get first hand job experience

By Gale Rose
grose@pratttribune.com

Eager students invaded businesses all over Pratt Tuesday, October 24 as they looked for future job opportunities on Disability Mentoring Day.

The 97 students from 12 schools fanned out across Pratt and got first hand experience what it would be like to work at those 40 businesses. They asked questions and got some hands on experience with various operations.

Paola Luna of Pratt High School, Gina Patton of Kingman High School and America Fernandez of St. John chose the Main Street Small Animal Veterinarian Clinic for their business. Students got a tour of the facility, learned what happens in an examination, got to handle various animals and watched a snake eat a mouse.

Luna said she was interested in animal health and wanted to know more about caring for hurt animals. Patton likes all kinds of animals and said she learned a lot from the experience. Watching the snake eat the mouse impressed her the most.

Fernandez wants to become a veterinarian and enjoyed learning everything that veterinarians

SEE MENTORING, 6

More example: Verb phrase (VP) attachment ambiguity


theguardian

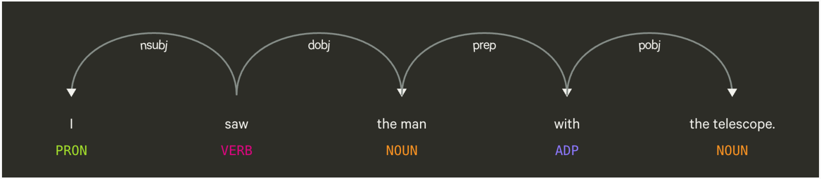
home › world › americas | asia | all

Rio de Janeiro

Mutilated body washes up on Rio beach to be used for Olympics beach volleyball

Dependency paths help extract semantic interpretation

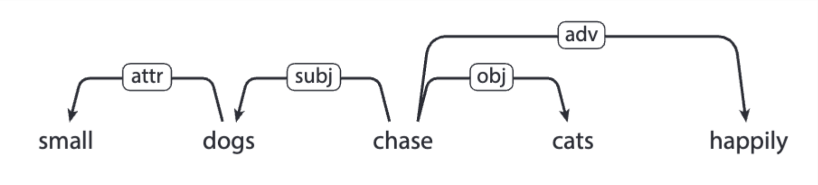
Coming back to the example:
I saw the man with the telescope.



Dependency grammar

Dependency grammar and dependency structure

Dependency grammar shows that syntactic structure (of a sentence) consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called dependencies.

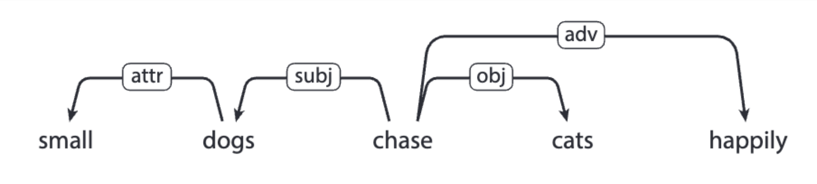


Sourced from: De Marneffe, M. C., & Nivre, J. (2019). *Dependency grammar*. *Annual Review of Linguistics*, 5(1), 197–218. Figure 1.

- The arrows are commonly *typed* with the name of grammatical relations (subject, prepositional object, adverb, etc.)

Dependency grammar and dependency structure

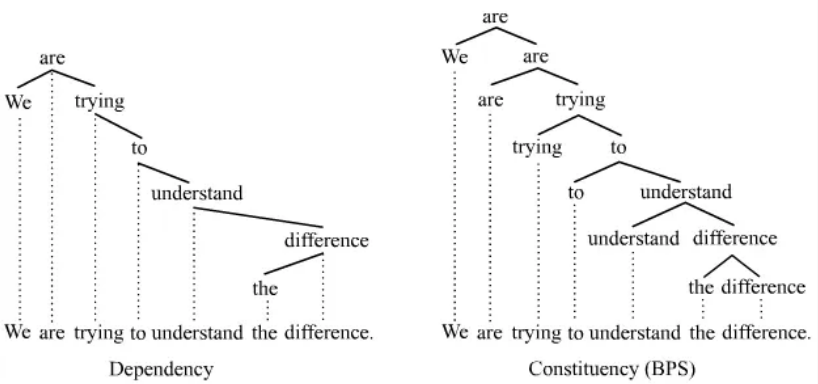
Dependency grammar shows that syntactic structure (of a sentence) consists of relations between lexical items, normally binary asymmetric relations ("arrows") called dependencies.



Sourced from: De Marneffe, M. C., & Nivre, J. (2019). *Dependency grammar*. *Annual Review of Linguistics*, 5(1), 197–218. Figure 1.

- The arrows are commonly *typed* with the name of grammatical relations (subject, prepositional object, adverb, etc.)
- Usually, dependencies form a tree (a connected acyclic, single-root graph)

Dependency grammar vs. Constituency parsing



Short history of dependency grammar/parsing

- The idea of dependency structures goes back a long way
  - Pāṇini was an ancient Indian grammarian, active around the 4th to 6th century BCE, who authored the Aṣṭādhyāyī (”Eight Chapters”), a formal system that systematically describes the grammar of Classical Sanskrit.
  - Basic approach to 1st millennium Arabic grammarians
- Constituency/CFG is a new-frangled invention
  - 20th centry invention (R. S. Wells, 1947; Chomsky, 1953, etc.)
- Modern dependency work is often sourced to Lucien Tesnière (1959)
  - Was dominant approach in “East” in 20th century (Russia, China, …)
  - Good gor free-er word order, inflected languages
- Used in some of the earliest parsers in NLP, even in the US:
  - David Hays, one of the founders of U.S. computational linguistics, built early dependency parsers (Hays, 1962) and published on dependency gramamr in *Language*

The arise of annotated data & Universal Dependencies tree

What is a treebank? An annotated corpus that includes syntactic or morphological structure, often in the form of parse trees.

The arise of annotated data & Universal Dependencies tree

What is a treebank**?** An annotated corpus that includes syntactic or morphological structure, often in the form of parse trees.

Milestones in treebank development:

- *Brown corpus* (1967): First general-purpose corpus; part-of-speech (PoS) tagged in 1979

The arise of annotated data & Universal Dependencies tree

What is a treebank? An annotated corpus that includes syntactic or morphological structure, often in the form of parse trees.

Milestones in treebank development:

- *Brown corpus* (1967): First general-purpose corpus; part-of-speech (PoS) tagged in 1979
- *Lancaster-IBM treebank* (Late 1980s): One of the first syntactically annotated corpora

The arise of annotated data & Universal Dependencies tree

What is a treebank? An annotated corpus that includes syntactic or morphological structure, often in the form of parse trees.

Milestones in treebank development:

- *Brown corpus* (1967): First general-purpose corpus; part-of-speech (PoS) tagged in 1979
- *Lancaster-IBM treebank* (Late 1980s): One of the first syntactically annotated corpora
- *The Penn treebank* (Marcus et al., 1993): Influential constituency-based treebank for English

The arise of annotated data & Universal Dependencies tree

What is a treebank? An annotated corpus that includes syntactic or morphological structure, often in the form of parse trees.

Milestones in treebank development:

- *Brown corpus* (1967): First general-purpose corpus; part-of-speech (PoS) tagged in 1979
- *Lancaster-IBM treebank* (Late 1980s): One of the first syntactically annotated corpora
- *The Penn treebank* (Marcus et al., 1993): Influential constituency-based treebank for English
- ***Universal Dependencies (UD)***: A multilingual, cross-linguistically consistent treebank project using dependency grammar

The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

But a treebank gives us many things:

- Reusability of the labor

The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

But a treebank gives us many things:

- Reusability of the labor
    - Many parser, POS taggers, and built on it

The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

But a treebank gives us many things:

- Reusability of the labor
    - Many parser, POS taggers, and built on it
    - Valuable resource for linguistics

The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

But a treebank gives us many things:

- Reusability of the labor
  - Many parser, POS taggers, and built on it
  - Valuable resource for linguistics
- Broad coverage, not just a few intuitions

The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

But a treebank gives us many things:

- Reusability of the labor
  - Many parser, POS taggers, and built on it
  - Valuable resource for linguistics
- Broad coverage, not just a few intuitions
- Frequencies and distributional information

The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

But a treebank gives us many things:

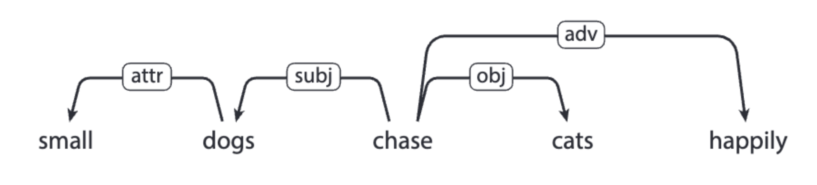
- Reusability of the labor
  - Many parser, POS taggers, and built on it
  - Valuable resource for linguistics
- Broad coverage, not just a few intuitions
- Frequencies and distributional information
- A way to evaluate NLP systems (work as a benchmark for empirical science)

Dependency parsing

Sources of information for dependency parsing

How do we build a parser, once we get the dependency information?

What are the sources of information for dependency parsing?



Sources of information for dependency parsing

1. Bilexical affinities
   - Which word pairs typically attach? (e.g., *eat* $\rightarrow$ *pizza*)
   - Use word–word statistics or embeddings to score candidate arcs.

Sources of information for dependency parsing

1. Bilexical affinities
   - Which word pairs typically attach? (e.g., *eat* $\rightarrow$ *pizza*)
   - Use word–word statistics or embeddings to score candidate arcs.
2. Dependency distance
   - Shorter arcs are preferred; long spans are penalized.
   - Implement via absolute token distance or distance buckets.

Sources of information for dependency parsing

1. **Bilexical affinities**
   - Which word pairs typically attach? (e.g., *eat* $\rightarrow$ *pizza*)
   - Use word–word statistics or embeddings to score candidate arcs.
2. **Dependency distance**
   - Shorter arcs are preferred; long spans are penalized.
   - Implement via absolute token distance or distance buckets.
3. **Intervening material**
   - Penalize arcs that span intervening verbs or punctuation.
   - Verbs and commas often mark clause/phrase boundaries.

Sources of information for dependency parsing

1. **Bilexical affinities**
   - Which word pairs typically attach? (e.g., *eat* $\rightarrow$ *pizza*)
   - Use word–word statistics or embeddings to score candidate arcs.
2. **Dependency distance**
   - Shorter arcs are preferred; long spans are penalized.
   - Implement via absolute token distance or distance buckets.
3. **Intervening material**
   - Penalize arcs that span intervening verbs or punctuation.
   - Verbs and commas often mark clause/phrase boundaries.
4. **Valency of heads**
   - Heads have typical patterns (e.g., V: subject on the left, object on the right; P: one object to the right).
   - Track how many left/right dependents are already attached to avoid overfilling a head.

Methods of dependency parsing

There are several ways (including dynamic programming, graph algorithms, etc.) but we'll focus on **greedy transition-based parsing** (Nivre, 2003).

1. Greedy transition-based parsing

Idea:

- **Stack:** Words that are being processed (“currently thinking about these”)

Idea:

- **Stack:** Words that are being processed ("currently thinking about these")
- **Buffer:** Words yet to be processed ("still to come")

1. Greedy transition-based parsing

Idea:

- **Stack:** Words that are being processed (“currently thinking about these”)
- **Buffer:** Words yet to be processed (“still to come”)
- **Transitions:** Actions that manipulate the stack and build syntactic relationships

Idea:

- **Stack:** Words that are being processed ("currently thinking about these")
- **Buffer:** Words yet to be processed ("still to come")
- **Transitions:** Actions that manipulate the stack and build syntactic relationships
- **Regulation (ROOT):**

1. Greedy transition-based parsing

Idea:

- **Stack:** Words that are being processed ("currently thinking about these")
- **Buffer:** Words yet to be processed ("still to come")
- **Transitions:** Actions that manipulate the stack and build syntactic relationships
- **Regulation (ROOT):**
    - ROOT can never be a dependent.

1. Greedy transition-based parsing

Idea:

- **Stack:** Words that are being processed (“currently thinking about these”)
- **Buffer:** Words yet to be processed (“still to come”)
- **Transitions:** Actions that manipulate the stack and build syntactic relationships
- **Regulation (ROOT):**
  - ROOT can never be a dependent.
  - Exactly one root arc per sentence: root(ROOT, sentential head).

1. Greedy transition-based parsing

Formal definition:

- σ: Stack, β: Buffer, A: Set of dependency arcs
- Initial state: $\sigma = [\text{ROOT}], \beta = [w_1, ..., w_n], A = \emptyset$
- Goal: Build all arcs and finish when $\sigma = [w], \beta = \emptyset$

Transitions (can choose one of three actions):

- **Shift:** $(\sigma, w_i|\beta, A) \Rightarrow (\sigma|w_i, \beta, A)$
- **Left-Arc$_r$:** $(\sigma|w_i|w_j, \beta, A) \Rightarrow (\sigma|w_j, \beta, A \cup \{r(w_j, w_i)\})$
- **Right-Arc$_r$:** $(\sigma|w_i|w_j, \beta, A) \Rightarrow (\sigma|w_i, \beta, A \cup \{r(w_i, w_j)\})$

Greedy transition-based parsing: Example

Sentence: *I saw him*

Initial State: Stack = [ROOT], Buffer = [I, saw, him], Arcs = {}

Step	Stack	Buffer	Transition	New Arc
1	[ROOT]	[I, saw, him]	SHIFT	—
2	[ROOT, I]	[saw, him]	SHIFT	—
3	[ROOT, I, saw]	[him]	LEFT-ARC	saw $\rightarrow$ I (subj)
4	[ROOT, saw]	[him]	SHIFT	—
5	[ROOT, saw, him]	[ ]	RIGHT-ARC	saw $\rightarrow$ him (obj)
6	[ROOT, saw]	[ ]	RIGHT-ARC	ROOT $\rightarrow$ saw (root)

MaltParser (Nivre and Hall, 2005)

- **Problem:** How do we choose the next parsing action?

MaltParser (Nivre and Hall, 2005)

- **Problem:** How do we choose the next parsing action?
    - **Answer:** Stand back — I know machine learning! Train a classifier that learns to predict the best transition at each step in a greedy dependency parser.

MaltParser (Nivre and Hall, 2005)

- **Problem:** How do we choose the next parsing action?
    - **Answer:** Stand back — I know machine learning! Train a classifier that learns to predict the best transition at each step in a greedy dependency parser.
- Each transition is predicted by a multi-class classifier (e.g., softmax or perceptron) over the set of legal moves.

MaltParser (Nivre and Hall, 2005)

- **Problem:** How do we choose the next parsing action?
    - **Answer:** Stand back — I know machine learning! Train a classifier that learns to predict the best transition at each step in a greedy dependency parser.
- Each transition is predicted by a multi-class classifier (e.g., softmax or perceptron) over the set of legal moves.
    - **Trained features:** Top word on the stack (and its POS tag), First word in the buffer (and its POS tag), Arc history, etc.

MaltParser (Nivre and Hall, 2005)

- There is **no search** in the simplest form — because it uses a **greedy algorithm**

MaltParser (Nivre and Hall, 2005)

- There is **no search** in the simplest form — because it uses a **greedy algorithm**
  - At each step, the parser selects the single best-scoring action and commits to it immediately.

MaltParser (Nivre and Hall, 2005)

- There is **no search** in the simplest form — because it uses a **greedy algorithm**
  - At each step, the parser selects the single best-scoring action and commits to it immediately.
  - No backtracking or consideration of alternatives.

MaltParser (Nivre and Hall, 2005)

- There is **no search** in the simplest form — because it uses a **greedy algorithm**
  - At each step, the parser selects the single best-scoring action and commits to it immediately.
  - No backtracking or consideration of alternatives.
- But you can profitably use **beam search** for better accuracy (at the cost of speed):

MaltParser (Nivre and Hall, 2005)

- There is **no search** in the simplest form — because it uses a **greedy algorithm**
  - At each step, the parser selects the single best-scoring action and commits to it immediately.
  - No backtracking or consideration of alternatives.
- But you can profitably use **beam search** for better accuracy (at the cost of speed):
  - Keep the top k highest-scoring partial parses at each step (beam width = k)

MaltParser (Nivre and Hall, 2005)

- There is **no search** in the simplest form — because it uses a **greedy algorithm**
  - At each step, the parser selects the single best-scoring action and commits to it immediately.
  - No backtracking or consideration of alternatives.
- But you can profitably use **beam search** for better accuracy (at the cost of speed):
  - Keep the top k highest-scoring partial parses at each step (beam width = k)
  - Allows recovery from early mistakes by exploring multiple promising paths.

MaltParser (Nivre and Hall, 2005)

- There is **no search** in the simplest form — because it uses a **greedy algorithm**
  - At each step, the parser selects the single best-scoring action and commits to it immediately.
  - No backtracking or consideration of alternatives.
- But you can profitably use **beam search** for better accuracy (at the cost of speed):
  - Keep the top k highest-scoring partial parses at each step (beam width = k)
  - Allows recovery from early mistakes by exploring multiple promising paths.
- The model's accuracy is fractionally below the state of the art in dependency parsing, but it provides very fast linear time parsing, with high accuracy.

Evaluation

Gold Standard: Hand-annotated syntactic structure used for evaluating parser output.

Evaluation

Gold Standard: Hand-annotated syntactic structure used for evaluating parser output.

Metrics: (1) **UAS (Unlabeled Attachment Score):** Correct head only; (2) **LAS (Labeled Attachment Score):** Correct head and label

Evaluation

Gold Standard: Hand-annotated syntactic structure used for evaluating parser output.

Metrics: (1) **UAS (Unlabeled Attachment Score):** Correct head only; (2) **LAS (Labeled Attachment Score):** Correct head and label

Example:

Word	Gold Head	Gold Label	Pred Head	Pred Label
She	2	*nsubj*	2	*nsubj*
likes	0	*root*	0	*root*
chocolate	2	*obj*	2	*nmod*
very	4	*advmod*	4	*advmod*
much	2	*advmod*	4	*advmod*

Evaluation

Gold Standard: Hand-annotated syntactic structure used for evaluating parser output.

Metrics: (1) **UAS (Unlabeled Attachment Score):** Correct head only; (2) **LAS (Labeled Attachment Score):** Correct head and label

Example:

Word	Gold Head	Gold Label	Pred Head	Pred Label
She	2	*nsubj*	2	*nsubj*
likes	0	*root*	0	*root*
chocolate	2	*obj*	2	*nmod*
very	4	*advmod*	4	*advmod*
much	2	*advmod*	4	*advmod*

Evaluation:

- **Total dependencies:** 5
- **Correct heads (UAS):** 4 → **UAS = 4/5 = 80%**
- **Correct heads + labels (LAS):** 2 → **LAS = 2/5 = 40%**

Neural dependency parsing

How do we gain from a neural dependency parser?

Indicator features revisited

- *Sparsity*: handcrafted feature templates generate very high-dimensional but rarely observed indicators (cf. one-hot encoding)

Neural approach: Dense and compact Representations

How do we gain from a neural dependency parser?

Indicator features revisited

- *Sparsity*: handcrafted feature templates generate very high-dimensional but rarely observed indicators (cf. one-hot encoding)
- *Incomplete coverage*: cannot anticipate every useful combination of word, POS, or context

Neural approach: Dense and compact Representations

How do we gain from a neural dependency parser?

Indicator features revisited

- *Sparsity*: handcrafted feature templates generate very high-dimensional but rarely observed indicators (cf. one-hot encoding)
- *Incomplete coverage*: cannot anticipate every useful combination of word, POS, or context
- *Engineering cost*: manual feature design and extraction pipelines add development overhead

Neural approach: Dense and compact Representations

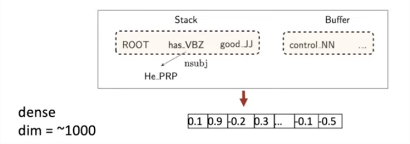
How do we gain from a neural dependency parser?

Indicator features revisited

- *Sparsity*: handcrafted feature templates generate very high-dimensional but rarely observed indicators (cf. one-hot encoding)
- *Incomplete coverage*: cannot anticipate every useful combination of word, POS, or context
- *Engineering cost*: manual feature design and extraction pipelines add development overhead
- *Runtime overhead*: expensive lookups and feature-template evaluations slow parsing

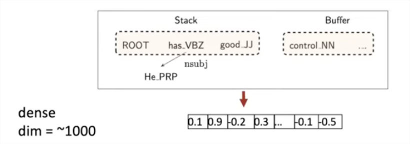
Neural approach: Dense and compact Representations

Neural approach: Dense and compact Representations



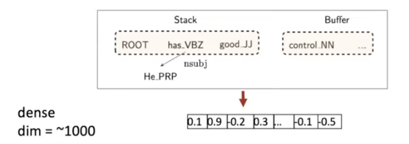
- Exactly the same parser configuration is used (e.g., top elements of the stack, front elements of the buffer, and relevant dependency arcs);

Neural approach: Dense and compact Representations



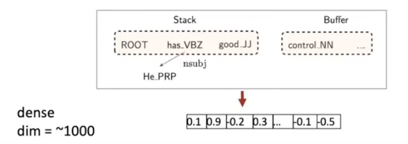
- Exactly the same parser configuration is used (e.g., top elements of the stack, front elements of the buffer, and relevant dependency arcs);
  – *Instead of hand-crafted binary features,* ***we summarize these elements into a single continuous "configuration vector."***

Neural approach: Dense and compact Representations



- Exactly the same parser configuration is used (e.g., top elements of the stack, front elements of the buffer, and relevant dependency arcs);
  – *Instead of hand-crafted binary features,* ***we summarize these elements into a single continuous "configuration vector."***
- Neural approach: the model *learns* this dense configuration automatically

Neural approach: Dense and compact Representations



- Exactly the same parser configuration is used (e.g., top elements of the stack, front elements of the buffer, and relevant dependency arcs);
  – *Instead of hand-crafted binary features, **we summarize these elements into a single continuous "configuration vector."***
- Neural approach: the model *learns* this dense configuration automatically
  – *Embedding layers map words, POS tags, and arc labels into low-dimensional vectors, which are concatenated to represent the parser state.*

A neural dependency parser (Chen & Manning, 2014)

- Review: *Distributed representations*
  - Represent each **word** as a d-dimensional dense vector (i.e., word embedding)

A neural dependency parser (Chen & Manning, 2014)

- Review: *Distributed representations*
  - Represent each **word** as a d-dimensional dense vector (i.e., word embedding)
    - Similar words are expected to have close vectors

A neural dependency parser (Chen & Manning, 2014)

- Review: *Distributed representations*
  - Represent each **word** as a d-dimensional dense vector (i.e., word embedding)
    - Similar words are expected to have close vectors
  - Meanwhile, POS and dependency labels are also represented as d-dimensional vectors

A neural dependency parser (Chen & Manning, 2014)

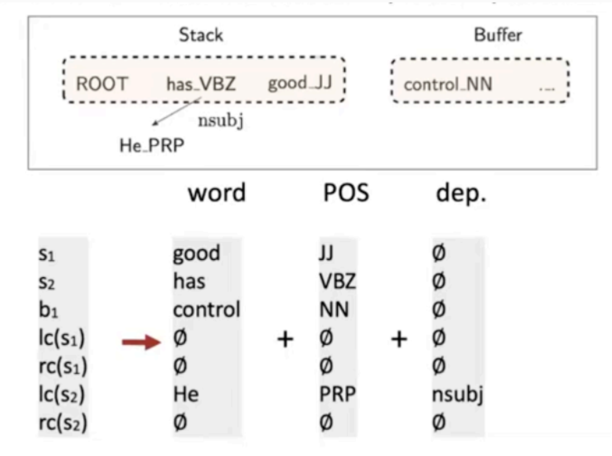
- Review: *Distributed representations*
  - Represent each **word** as a d-dimensional dense vector (i.e., word embedding)
    - Similar words are expected to have close vectors
  - Meanwhile, POS and dependency labels are also represented as d-dimensional vectors
    - The similar discrete sets also exhibit many semantical similarities.

A neural dependency parser (Chen & Manning, 2014)

- Review: *Distributed representations*
  - Represent each **word** as a d-dimensional dense vector (i.e., word embedding)
    - Similar words are expected to have close vectors
  - Meanwhile, POS and dependency labels are also represented as d-dimensional vectors
    - The similar discrete sets also exhibit many semantical similarities.
    - e.g., NNS (plural noun) should be close to NN (singular noun); nummod (numerical modifier) should be close to amod (adjective modifier).

Extracting tokens and vector representations from configuration

We can extract a set of tokens based on stack/buffer positions



A **concatenation** of the vector representation of all these is the neural representation of configuration.

Deep learning classifiers are non-linear classifiers

- A **softmax classifier** assigns classes $y \in C$ based on inputs $x \in \mathbb{R}^d$ via
$$p(y \mid x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^{C} \exp(W_c \cdot x)}.$$
- We train the weight matrix $W \in \mathbb{R}^{C \times d}$ by minimizing the negative log-likelihood (i.e., cross entropy loss):

Review: Neural networks are more powerful

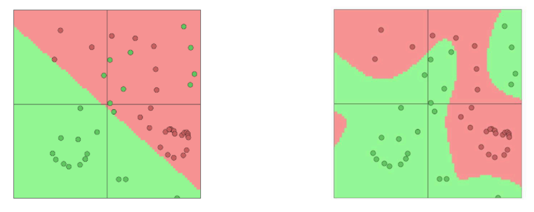
- Traditional ML classifiers (Naïve Bayes, SVMs, logistic regression, softmax) only produce **linear decision boundaries**.

Review: Neural networks are more powerful

- Traditional ML classifiers (Naïve Bayes, SVMs, logistic regression, softmax) only produce **linear decision boundaries**.
- Review: Neural networks (with multiple hidden layers) can learn much more complex, **nonlinear decision boundaries**.

Review: Neural networks are more powerful

- Traditional ML classifiers (Naïve Bayes, SVMs, logistic regression, softmax) only produce **linear decision boundaries**.
- Review: Neural networks (with multiple hidden layers) can learn much more complex, **nonlinear decision boundaries**.
- In the original input space, the boundary may look nonlinear. But after the hidden layers transform the data, the final softmax layer only needs a simple **linear classifier** to separate the classes.



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

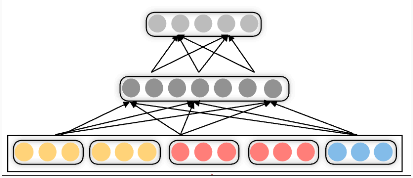
Simple feed-forward neural network multi-class classifier

Model architecture

Input: $x = [\dots, \mathrm{embed}(w_{i-1}), \mathrm{embed}(w_i), \mathrm{embed}(w_{i+1}), \dots]$

Hidden: $h = \mathrm{ReLU}(Wx + b_1)$

Output: $y = \mathrm{softmax}(Uh + b_2)$

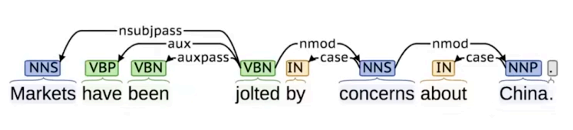


Training objective (cross-entropy loss) back-propagated

$$\mathcal{L} = -\sum_i \log p(y^{(i)} \mid x^{(i)})$$

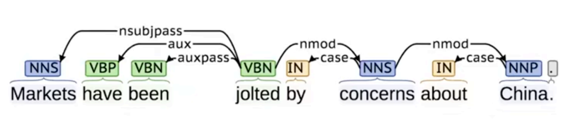
Dependency parsing for sentence structure

- Chen and Manning (2014) showed that neural networks can accurately determine the structure of sentences, supporting meaning interpretation.



Dependency parsing for sentence structure

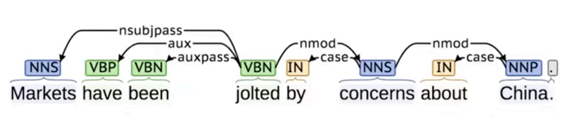
- Chen and Manning (2014) showed that neural networks can accurately determine the structure of sentences, supporting meaning interpretation.



- It was the first simple, successful neural dependency parser.

Dependency parsing for sentence structure

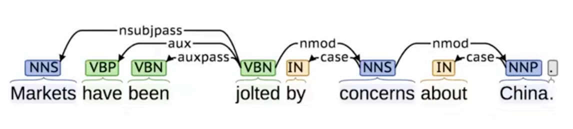
- Chen and Manning (2014) showed that neural networks can accurately determine the structure of sentences, supporting meaning interpretation.



- It was the first simple, successful neural dependency parser.
- The dense representations (and non-linear classifier) let it outperform other greedy parsers in both accuracy and speed.

Dependency parsing for sentence structure

- Chen and Manning (2014) showed that neural networks can accurately determine the structure of sentences, supporting meaning interpretation.



- It was the first simple, successful neural dependency parser.
- The dense representations (and non-linear classifier) let it outperform other greedy parsers in both accuracy and speed.
- This work was further developed and improved by others.

Further developments

This work was further developed and improved by others, including in particular at Google.

Graph-based dependency parsers

- Compute a score for every possible dependency (choice of head) for each word

Graph-based dependency parsers

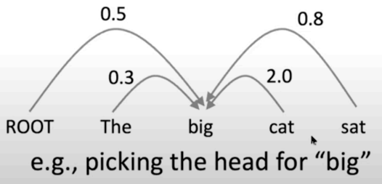
- Compute a score for every possible dependency (choice of head) for each word
  - Doing this well requires more than just knowing two words

Graph-based dependency parsers

- Compute a score for every possible dependency (choice of head) for each word
    - Doing this well requires more than just knowing two words
    - We need good “contextual” representations of each word token

Graph-based dependency parsers

- Compute a score for every possible dependency (choice of head) for each word
  - Doing this well requires more than just knowing two words
  - We need good “contextual” representations of each word token
- Repeat the same process for each other word; find the best parse



A neural graph-based dependency parser

- Dozat and Manning (2017); Dozat, Qi, and Manning (2017) - This paper revived interest in graph-based dependency parsing in a neural world

A neural graph-based dependency parser

- Dozat and Manning (2017); Dozat, Qi, and Manning (2017) - This paper revived interest in graph-based dependency parsing in a neural world
    - Designed a new scoring model (i.e., biaffine) for neural dependency parsing

A neural graph-based dependency parser

- Dozat and Manning (2017); Dozat, Qi, and Manning (2017) - This paper revived interest in graph-based dependency parsing in a neural world
  - Designed a new scoring model (i.e., biaffine) for neural dependency parsing
- Really great results!

Method	UAS	LAS (PTB WSJ SD 3.3
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79
Dozat & Manning 2017	**95.74**	**94.08**

A neural graph-based dependency parser

- Dozat and Manning (2017); Dozat, Qi, and Manning (2017) - This paper revived interest in graph-based dependency parsing in a neural world
  - Designed a new scoring model (i.e., biaffine) for neural dependency parsing
- Really great results!

Method	UAS	LAS (PTB WSJ SD 3.3
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79
Dozat & Manning 2017	**95.74**	**94.08**

- But, slower than the simple neural transition-based parsers.

Wrap-up

Wrap-up

- Syntactic structure: Consistency and dependency
- Dependency grammar and treebanks
- Dependency parsing
- Transition-based dependency parsing
- Neural dependency parsing

on Thursday

We will think about how to train a dependency parser on the provided training data and generate prediction for the test set.