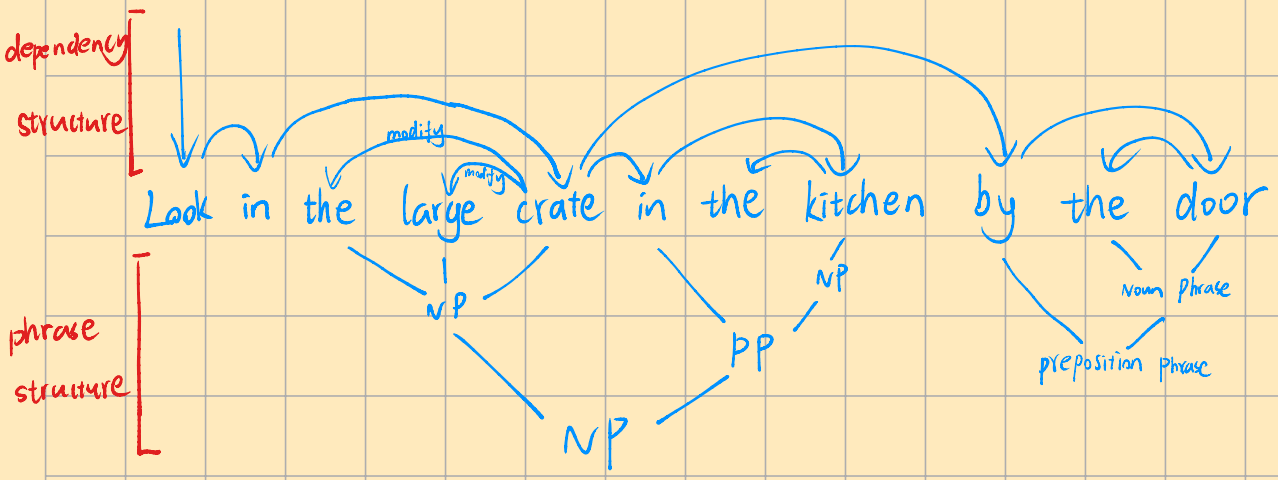
Linguistic Structure of human languages

2 views to think of it

- phrase structure which represented in terms of context-free grammars (CFG) — we've learned from Formal Languages and automata
- Dependency structure: find what's the headwork and then say which things modify the headword

example:



Why is sentence structure needed for communication

because we'll get a lot of ambiguities in languages

in English

Prepositional phrase attachment ambiguity

eg: Scientists count whales from space

- Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts

↳ the more pp you have, the more ambiguities will appear

Coordination scope ambiguity

eg: [[Shuttle veteran] and [longtime NASA executive]] Fred Gregory appointed to board

Adjectival / Adverbial Modifier Ambiguity

Verb Phrase attachment Ambiguity ....

in Chinese

eg:[咬死了][猎人的][狗]

in Japanese

eg、[古い[友達]に会った]

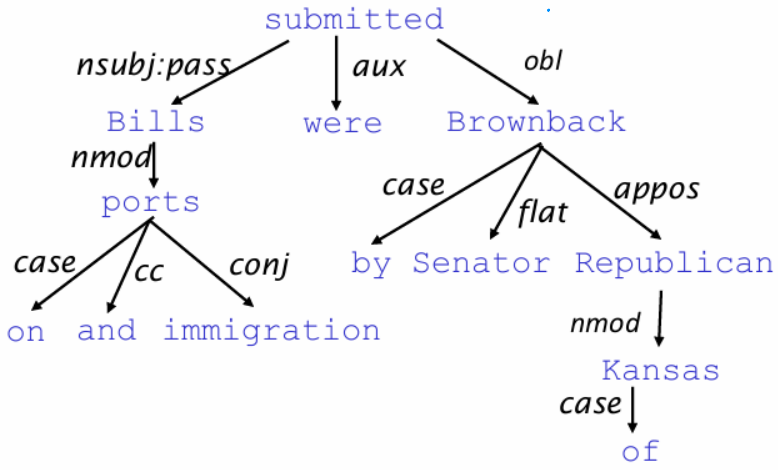
Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations ("arrows") called dependencies

An arrow connects a head with a dependent

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

无环的



Dependency Grammar/Parsing History

- The idea of dependency structure goes back a long way
  - To Pāṇini's grammar (c. 5th century BCE)
  - Basic approach of 1st millennium Arabic grammarians
- Constituency/context-free grammar is a new-fangled invention
  - 20th century invention (R.S. Wells, 1947; then Chomsky 1953, etc.)
- Modern dependency work is often sourced to Lucien Tesnière (1959)
  - Was dominant approach in "East" in 20th Century (Russia, China, …)
    - Good for free-er word order, inflected languages like Russian (or Latin!)
- 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 (first?) dependency parser (Hays 1962) and published on dependency grammar in *Language*

- Some people draw the arrows one way; some the other way!
  - Tesnière had them point from head to dependent – we follow that convention
- We usually add a fake ROOT so every word is a dependent of precisely 1 other node

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 parsers, part-of-speech taggers, etc. can be built on it
  - Valuable resource for linguistics
- Broad coverage, not just a few intuitions
- Frequencies and distributional information
- A way to evaluate NLP systems

Dependency Conditioning Preferences

What are the straightforward sources of information for dependency parsing?

1. Bilexical affinities — The dependency [discussion → issues] is plausible
2. Dependency distance — Most dependencies are between nearby words
3. Intervening material — Dependencies rarely span intervening verbs or punctuation
4. Valency of heads — How many dependents on which side are usual for a head?

Valency (价)

how many arguments they take

Argument (论元)

Programming / math : the input value to a function

Linguistics / Semantics / NLP : A semantic participant required by a predicate

Participants are called arguments of the predicate

predicate (谓词) and argument (论元) come from logic and linguistics

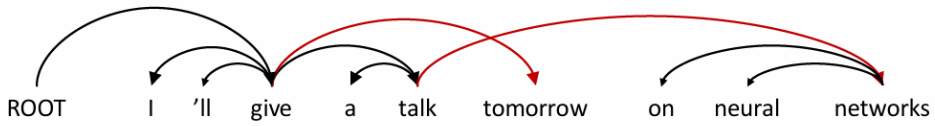
eg: Mary gave John a book

predicate

argument

Dependency Parsing

- A sentence is parsed by choosing for each word what other word (including ROOT) it is a dependent of
- Usually some constraints:
  - Only one word is a dependent of ROOT
  - Don't want cycles A → B, B → A
- This makes the dependencies a tree
- Final issue is whether arrows can cross (be non-projective) or not



Projectivity

- Definition of a projective parse: There are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words
- Dependencies corresponding to a CFG tree must be projective
  - I.e., by forming dependencies by taking 1 child of each category as head
- Most syntactic structure is projective like this, but dependency theory normally does allow non-projective structures to account for displaced constituents
  - You can't easily get the semantics of certain constructions right without these nonprojective dependencies

3. Methods of Dependency Parsing

1. Dynamic programming

   Eisner (1996) gives a clever algorithm with complexity $O(n^3)$, by producing parse items with heads at the ends rather than in the middle
2. Graph algorithms

   You create a Minimum Spanning Tree for a sentence

   McDonald et al.'s (2005) $O(n^2)$ MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)

   Neural graph-based parser: Dozat and Manning (2017) et seq. – very successful!
3. Constraint Satisfaction

   Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.
4. "Transition-based parsing" or "deterministic dependency parsing"

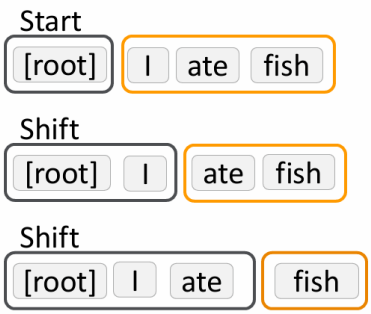
   Greedy choice of attachments guided by good machine learning classifiers

   E.g., MaltParser (Nivre et al. 2008). Has proven highly effective. And fast.

Arc-standard transition-based parser

(there are other transition schemes …)

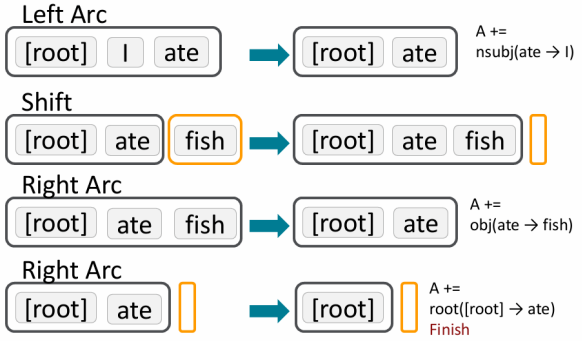
Analysis of "I ate fish"



Start: σ = [ROOT], β = w_1, …, w_n, A = ∅
1. Shift: σ, w_i|β, A → σ|w_i, β, A
2. Left-Arc, : σ|w_i|w_j, β, A → σ|w_j, β, A∪{r(w_j,w_i)}
3. Right-Arc, : σ|w_i|w_j, β, A → σ|w_i, β, A∪{r(w_i,w_j)}
Finish: σ = [w], β = ∅

Arc-standard transition-based parser

Analysis of "I ate fish"



Nota bene:
In this example I've at each step made the "correct" next transition.
But a parser has to work this out – by exploring or inferring!

A = { nsubj(ate → I), obj(ate → fish) root([root] → ate) }

43

finishing condition is that my buffer is empty and my stack

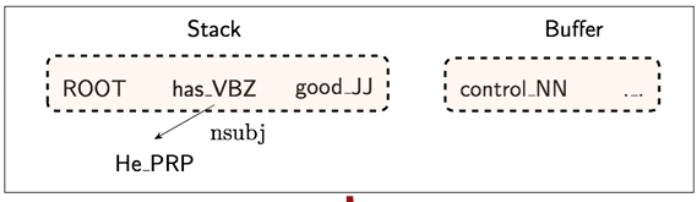
contains just the word root

MaltParser [Nivre and Hall 2005]

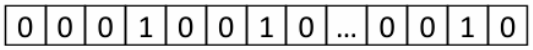
- We have left to explain how we choose the next action 🤷
  - Answer: Stand back, I know machine learning!
- Each action is predicted by a discriminative classifier (e.g., softmax classifier) over each legal move
  - Max of 3 untyped choices (max of |R| × 2 + 1 when typed)
  - Features: top of stack word, POS; first in buffer word, POS; etc.
- There is NO search (in the simplest form)
  - But you can profitably do a beam search if you wish (slower but better):
    - You keep *k* good parse prefixes at each time step
- 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 – great for parsing the web

The job of the machine learning to predict what is the right transition at each point in time

Conventional Feature Representation



binary, sparse
dim =10^6–10^7



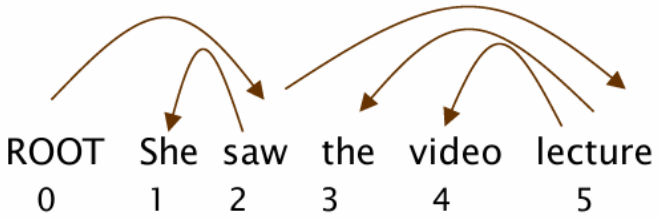
Feature templates: usually a combination of 1–3 elements from the configuration

Indicator features

$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$
$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

MaltParser: symbolic, feature-based machine learning system
Problem: it will end up with millions and millions of features

Evaluation of Dependency Parsing: (labeled) dependency accuracy



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

UAS = 4 / 5 = 80%

LAS = 2 / 5 = 40%

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

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

UAS: unlabeled Attachment Score

LAS: Labeled Attachment Score

They are used to measure the degree of match between a dependency tree predicted by an analyzer and the manually annotated "ground truth" (golden tree)

4. Why do we gain from a neural dependency parser? Indicator Features Revisited

Categorical features are:

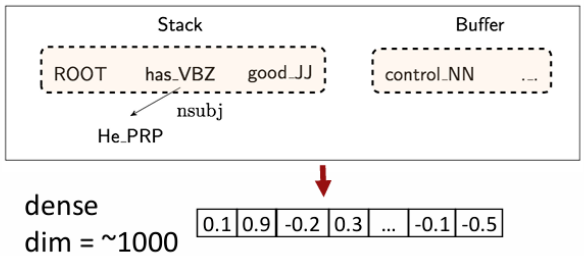
- Problem #1: sparse
- Problem #2: incomplete
- Problem #3: expensive to compute

More than 95% of parsing time is consumed by feature computation

$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$
$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$
$$lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$$
$$lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$$

Neural Approach:

learn a dense and compact feature representation



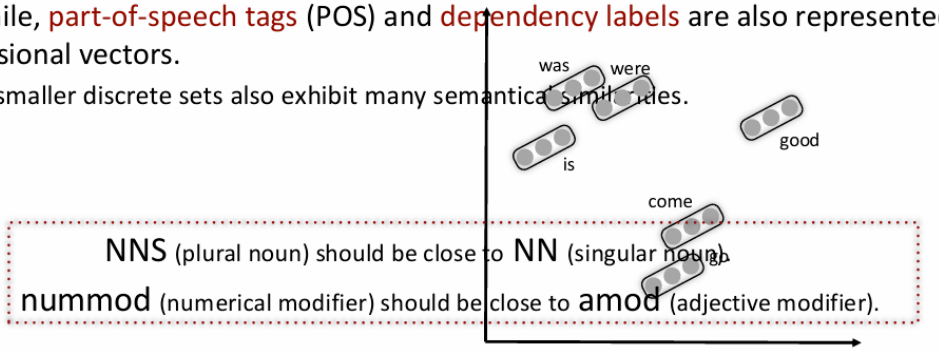
why sparse => don't see them often

why incomplete => some words and combinations you'd seen and some you just didn't see in the trainning data

A neural dependency parser [Chen and Manning 2014]

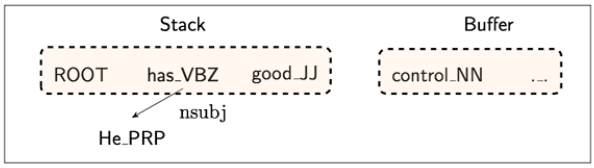
First win: Distributed Representations

- We represent each word as a *d*-dimensional dense vector (i.e., word embedding)
  - Similar words are expected to have close vectors.
- Meanwhile, part-of-speech tags (POS) and dependency labels are also represented as *d*-dimensional vectors.
  - The smaller discrete sets also exhibit many semantical similarities.



Extracting Tokens & vector representations from configuration

- We extract a set of tokens based on the stack / buffer positions:



	word	POS	dep.
s_1	good	JJ	Ø
s_2	has	VBZ	Ø
b_1	control	NN	Ø
$lc(s_1)$	Ø	Ø	Ø
$rc(s_1)$	Ø	Ø	Ø
$lc(s_2)$	He	PRP	nsubj
$rc(s_2)$	Ø	Ø	Ø

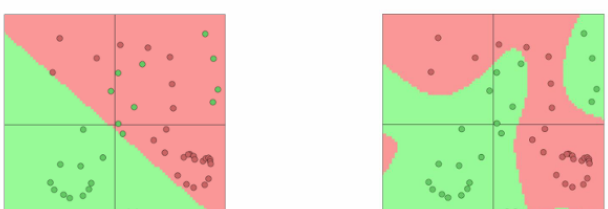
A concatenation of the vector representation of all these is the neural representation of a configuration

Second win: Deep Learning classifiers are non-linear classifiers

- A softmax classifier assigns classes $y \in C$ based on inputs $x \in \mathbb{R}^d$ via the probability:

$$p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$

- Traditional ML classifiers (including Naïve Bayes, SVMs, logistic regression and softmax classifier) are not very powerful classifiers: they only give linear decision boundaries
- But neural networks can use multiple layers to learn much more complex nonlinear decision boundaries



Neural Dependency Parser Model Architecture

(A simple feed-forward neural network multi-class classifier)

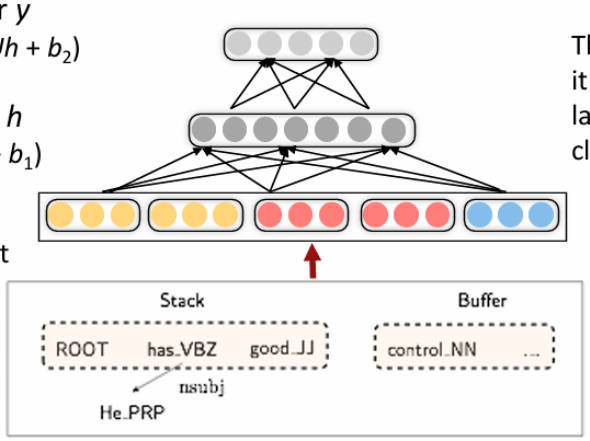
Log loss (cross-entropy error) will be back-propagated to the embeddings

Softmax probabilities → { Shift , Left-Arc$_r$, Right-Arc$_r$ }

Output layer y
$y = \text{softmax}(Uh + b_2)$

Hidden layer h
$h = \text{ReLU}(Wx + b_1)$

Input layer x
lookup + concat



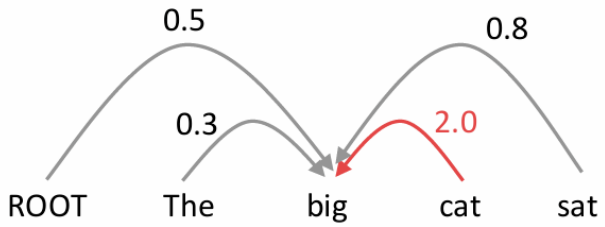
The hidden layer re-represents the input — it moves inputs around in an intermediate layer vector space—so it can be easily classified with a (linear) softmax

Wins:
Distributed representations!
Non-linear classifier!

Another parsers

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 the two words
  - We need **good "contextual" representations** of each word token, which we will develop in the coming lectures
- Repeat the same process for each other word; find the best parse (MST algorithm)



e.g., picking the head for "big"

Repeat to question itself which of these words (except the head word you are choosing now) is it most likely a dependent of eg: big to sat

then score the n-squared possible dependencies

and we want to form a tree, also want to find the minimum cost tree

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 biaffine scoring model for neural dependency parsing
    - Also crucially uses a neural sequence model, something we discuss later
- Really great results!
  - **But slower than the simple neural transition-based parsers**
    - There are n^2 possible dependencies in a sentence of length n

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**

2. 它的作用与价值

依赖解析器在传统NLP流水线中是一个**至关重要的“中间件”**，它的作用主要体现在：

A. 提供结构化的句法信息

它将线性的、序列化的文本，转换成了非线性的、结构化的树。这为下游任务提供了丰富的语言学特征。

B. 赋能下游NLP任务

在深度学习一统天下之前，许多核心NLP任务严重依赖依赖解析的结果作为特征输入：

- **信息抽取**：要找到“谁-做了什么-对谁”，可以寻找 `(nsubj, root, obj)` 这样的依赖三角。
  - 例如，从 “Apple acquired Beats.” 中，通过 `nsubj(acquired, Apple)` 和 `obj(acquired, Beats)` 可以轻松抽取出 `(Apple, acquired, Beats)`。
- **关系抽取**：判断两个实体之间的关系类型。
- **语义角色标注**：识别“谁在什么时候、什么地点、对谁、做了什么”。
- **情感分析**：通过分析依赖关系，可以更准确地找到观点词和评价对象。例如，明白 “not good” 中 `not` 是修饰 `good` 的。
- **机器翻译**：在翻译时调整语序以符合目标语言的语法结构。

C. 作为语言学理论的验证

它为计算语言学研究提供了一个强大的工具，用于验证和探索人类语言的语法理论。

3. 与现在模型的关系（演变与现状）

这是一个关键点。依赖解析器与现今模型的关系是 **“从台前主力到幕后功臣”** 的演变。

A. 传统范式（2018年之前）：管道式

在这种范式下，依赖解析器是一个独立的、显式的模块。

`原始文本 -> 分词 -> 词性标注 -> [依赖解析] -> 信息抽取/问答系统...`

问题：管道式系统存在错误传播问题。如果解析器出错，错误会一直传递到下游任务。

B. 现代范式（2018年之后）：端到端

随着BERT、GPT等**预训练Transformer模型**的出现，范式发生了根本性转变。

1. **隐式学习**：
   - 像BERT这样的大模型，通过在数十亿文本上进行预训练，已经**隐式地学习到了大量的语法知识。**
   - 它不需要被明确地告知什么是“主语”或“宾语”，但其内部的注意力机制已经能够捕捉到词与词之间强大的语法和语义关联。
   - **模型自己成了“黑箱语法专家”。**
2. **端到端学习**：
   - 现在，对于大多数任务（如情感分析、关系抽取），我们不再需要**显式地**先运行一个依赖解析器，再把结果作为特征输入。
   - 相反，我们直接将原始文本（或稍作处理）输入给大型语言模型，并通过微调，让模型**端到端地**学习从文本到最终答案的映射。
   - **依赖解析所承担的特征工程工作，被大模型强大的表示能力替代了。**