Department of Computing

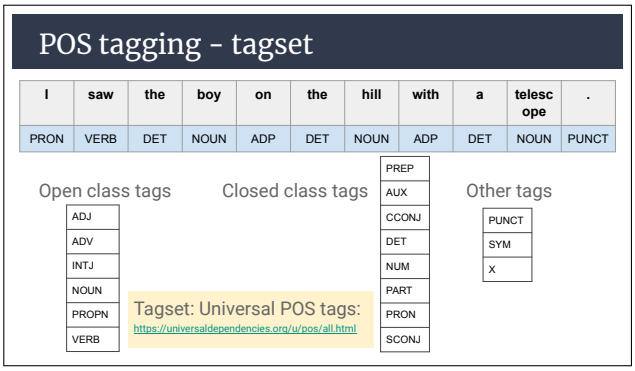
Imperial College of Science, Technology and Medicine

POS tagging

Author: Anton Zhitomirsky

Contents

1 Part Of Speech Tagging



- POS tries to find labels we're interested in (verb adjective, etc.)

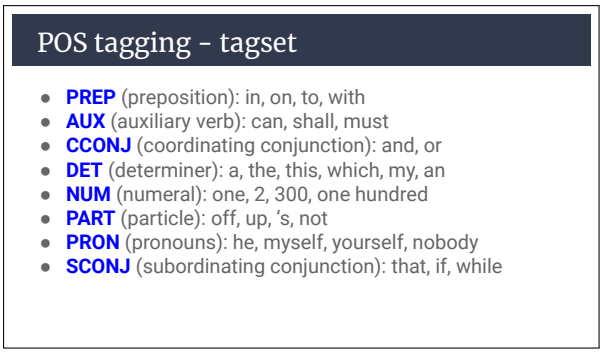
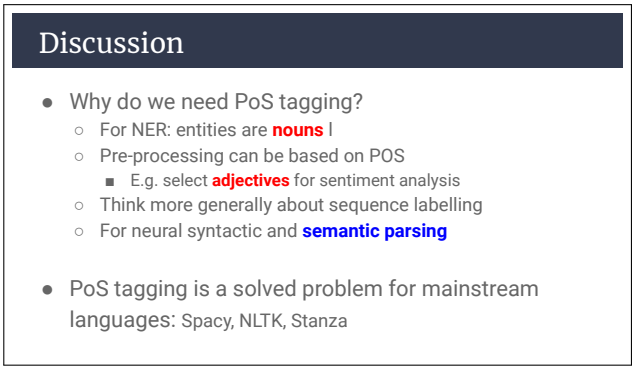


Figure 1: Some examples of attributes we may wish to tag

1.1 Why do we need POS tagging?



- it is a solved problem
- There are still tasks we may need to do at scale (e.g. a language filtering problem with many messages we may not be able to run a transformed based language model on all of them)

1.2 Baseline method

POS tagging - baseline method

- Naive approach
  - Assign each word its most frequent POS tag
  - Assign all unknown words the tag NOUN
- ~90% accuracy!
- There are exceptions....
- But frequency still plays a role
  - Probabilistic POS taggers

- NOUN is the most common tag, so we could just tag everything as a noun
- This is a baseline method, and we can do better than this

1.2.1 Ambiguities

POS ambiguities

back

The back/ADJ door

On my back/NOUN

Win the voters back/ADV

Promised to back/VERB the bill

POS ambiguities

Flies like a flower

Flies/**VERB**/**NOUN**

like/PREP/**ADV**/CONJ/NOUN/**VERB**

a/DET

flower/**NOUN**/VERB

1.3 Probabilistic POS tagging

Probabilistic POS tagging

- Use frequencies but take context into account
- Given a **sequence** of words $W = w_1, w_2, \ldots, w_n$
  - Estimate the **sequence** of POS tags $T = t_1, t_2, \ldots, t_n$
  - Compute $P(T|W)$

Instance **of *many-to-many*** classification

- Use the frequences but take the context into account.

Probabilistic POS tagging - generative

- Generative approach (Bayes):

$$P(T|W) = \frac{P(W|T)P(T)}{P(W)} = P(W|T)P(T)$$

LM of tags!

- Chain rule + **markov** assumption (e.g. bigram):

$$P(T) \approx P(t_1)P(t_2|t_1)P(t_3|t_2)\ldots P(t_n|t_{n-1})$$

- Each word depends only on its tag (not on previous words)

$$P(W|T) \approx P(w_1|t_1)P(w_2|t_2)\ldots P(w_n|t_n)$$

Like machine translation!

- The probability of a word sequence given the word sequence is 1 becuase we're not drawing $P(W)$ from a word distribution.
- We take the bigram model as assumption
- note above w is a word and t is a tag

Probabilistic POS tagging - generative

- Putting both together:

$$P(T|W) \approx P(t_1)P(t_2|t_1)\dots P(t_n|t_{n-1})P(w_1|t_1)P(w_2|t_2)\dots P(w_n|t_n)$$
$$\approx P(t_1)P(w_1|t_1)P(t_2|t_1)P(w_2|t_2)\dots P(t_n|t_{n-1})P(w_n|t_n)$$

where, as before $P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$

and $P(w_i|t_i) = \frac{C(w_i,t_i)}{C(t_i)}$

- as a baseline, we would a basic statistical count.
- The first says, how many times i encounter a tak t_i after i observe a tag t_{i-1} given that the total number of times I have ever seen the tag t_{i-1}.
- similarly for the second.

1.3.1 Example

Probabilistic POS tagging - generative

- For example, given a training corpus:

John/PROPN is/VERB expected/VERB to/PART race/VERB

This/DET is/VERB the/DET race/NOUN I/PRON wanted/VERB

Bring/VERB this/DET to/PART the/DET race/NOUN

- Compute the necessary probabilities

- Here is the training corpus

Probabilistic POS tagging - generative

$P(t_i|t_{i-1})$ $P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$

t_0

	PROPN	VERB	PART	NOUN	PRON	DET
<s> (3)	1/3	1/3	0	0	0	1/3
PROPN (1)	0	1/1	0	0	0	0
VERB (6)	0	1/6	1/6	0	0	2/6
PART (2)	0	1/2	0	0	0	1/2
NOUN (2)	0	0	0	0	1/2	0
PRON (1)	0	1/1	0	0	0	0
DET (4)	0	1/4	1/4	2/4	0	0

How to make the sum of each row **equal 1**?

- from the start symbol (we have 3 sentences above) we form a table.
- However, this doesn't add up to 1.

Probabilistic POS tagging - generative

$P(t_i|t_{i-1})$ $P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$

t_0

	PROPN	VERB	PART	NOUN	PRON	DET	</s>
<s> (3)	1/3	1/3	0	0	0	1/3	0
PROPN (1)	0	1/1	0	0	0	0	0
VERB (6)	0	1/6	1/6	0	0	2/6	2/6
PART (2)	0	1/2	0	0	0	1/2	0
NOUN (2)	0	0	0	0	1/2	0	1/2
PRON (1)	0	1/1	0	0	0	0	0
DET (4)	0	1/4	1/4	2/4	0	0	0

- Therefore, we add a terminating tag, which then fills the rest of the probability in.
- In the verb case, we have observed 6 verbs, yet 2 out of the 6 verbs have been used to finish the sentence.

Probabilistic POS tagging - generative

$P(w_i|t_i)$ $P(w_i|t_i) = \frac{C(w_i,t_i)}{C(t_i)}$

	john	is	expect	to	race	this	the	I	want	bring
PROPN (1)	1/1	0	0	0	0	0	0	0	0	0
VERB (6)	0	2/6	1/6	0	1/6	0	0	0	1/6	1/6
PART (2)	0	0	0	2/2	0	0	0	0	0	0
NOUN (2)	0	0	0	0	2/2	0	0	0	0	0
PRON (1)	0	0	0	0	0	0	0	1/1	0	0
DET (4)	0	0	0	0	0	2/4	2/4	0	0	0

- you do the same given for the word given the tag case.
- this is a simple counting baseline

We then apply the formula to get the posterior (the tag given the word sequence), and at each column we apply the max. This runs at lightning speed and is very parallelizeable, and gives a good baseline.

Probabilistic POS tagging - generative

$P(t_i|t_{i-1}) * P(w_i|t_i)$

- Tag the following test sentence (consider lemmas), left to right, taking the **max at every step to be the POS for that word**

	John	wants	to	race	this	race
PROPN						
VERB						
PART						
NOUN						
PRON						
DET						

Probabilistic POS tagging - generative

$P(t_i|t_{i-1}) * P(w_i|t_i)$

- Tag the following test sentence (consider lemmas), left to right, taking the **max at every step to be the POS for that word**

	John	wants	to	race	this	race
PROPN	1/3*1/1					
VERB	1/3*0					
PART	0*0					
NOUN	0*0					
PRON	0*0					
DET	1/3*0					

Probabilistic POS tagging

$P(t_i|t_{i-1}) * P(w_i|t_i)$

- Tag the following test sentence (consider lemmas), left to right, taking the **max at every step to be the POS for that word**

	John	wants	to	race	this	race
PROPN	1/3*1/1	0*0				
VERB		1/1*1/6				
PART		0*0				
NOUN		0*0				
PRON		0*0				
DET		0*0				

PROPN

Probabilistic POS tagging

$P(t_i|t_{i-1}) * P(w_i|t_i)$

- Tag the following test sentence (consider lemmas), left to right, taking the **max at every step to be the POS for that word**

	John	wants	to	race	this	race
PROPN	1/3*1/1		0*0			
VERB		1/1*1/6	1/6*0			
PART			1/6*2/2			
NOUN			0*0			
PRON			0*0			
DET			2/6*0			

PROPN VERB

Probabilistic POS tagging

$P(t_i|t_{i-1}) * P(w_i|t_i)$

- Tag the following test sentence (consider lemmas), left to right, taking the **max at every step to be the POS for that word**

	John	wants	to	race	this	race
PROPN	1/3*1/1			0*0		
VERB		1/1*1/6		1/2*1/6		
PART			1/6*2/2	0*0		
NOUN				0*2/2		
PRON				0*0		
DET				1/2*0		

PROPN VERB PART

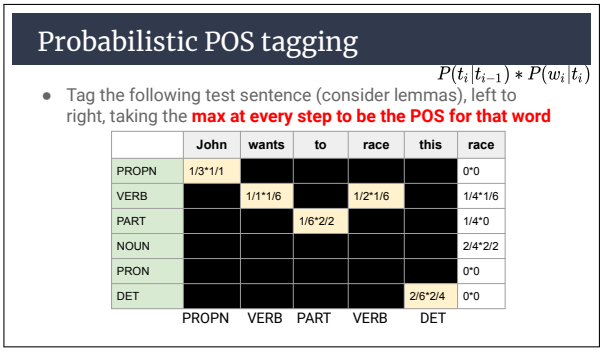
Probabilistic POS tagging

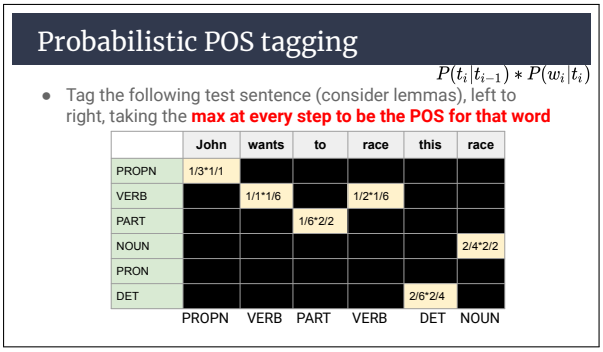
$P(t_i|t_{i-1}) * P(w_i|t_i)$

- Tag the following test sentence (consider lemmas), left to right, taking the **max at every step to be the POS for that word**

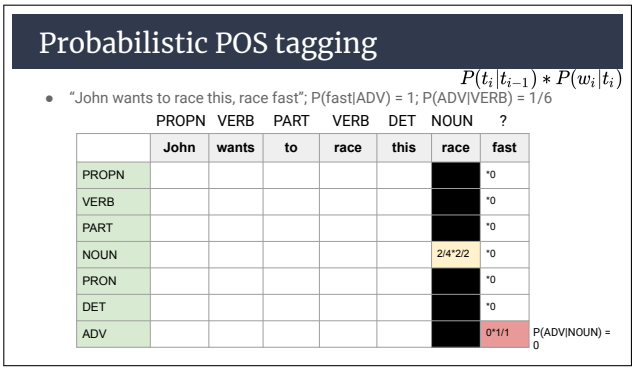
	John	wants	to	race	this	race
PROPN	1/3*1/1				0*0	
VERB		1/1*1/6		1/2*1/6	1/6*0	
PART			1/6*2/2		1/6*0	
NOUN					0*0	
PRON					0*0	
DET					2/6*2/4	

PROPN VERB PART VERB



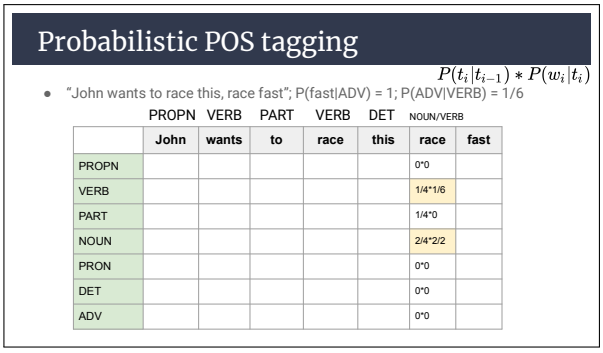


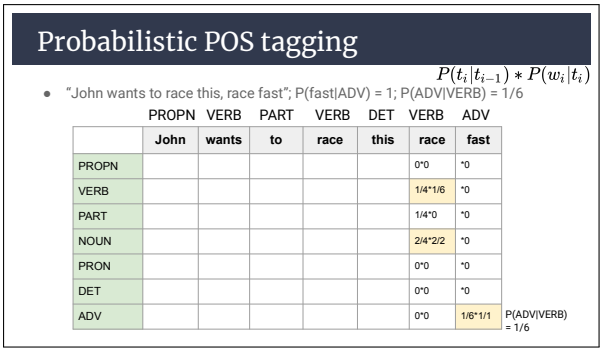
1.3.2 Example Problem

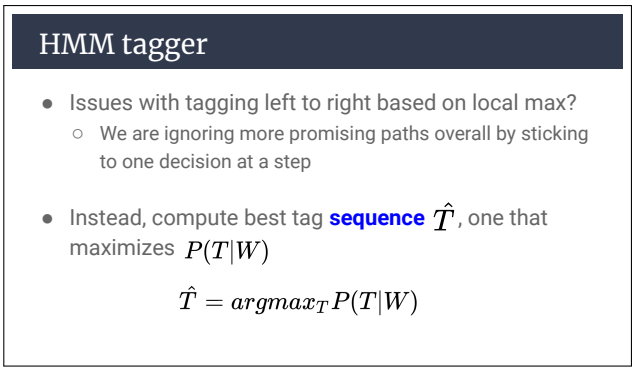


- If we extend this sentence with 'fast' we don't have a transition in the training sample.
- We have tagged this as a noun, even though it should be adverb. This is becuase it doesn't have a transition in the training data even though the only time we observed fast before, it was an adverb.
- However, we have never observed an adverb after a noun so we never transitioned from this point.

☞ Therfore, we aren't really considering the entire sequence, we're only considering local decisions by the greedy approach.







- We're ignoring more promising paths.
- We are therefore interested in maximising the posterior.

1.4 Hidden Markov Model Tagger

HMM tagger

- A Hidden Markov Model (**HMM**) allows inferring hidden states from observations:

$Q = q_1 q_2 \dots q_N$	A set of *N* states (tags)
$A = a_{11} a_{ij} \dots a_{NN}$	A transition probability matrix *A*, each a_{ij} representing the probability P of moving from state *i* to state *j*, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of T observations (words), each one drawn from a vocabulary V = $v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of observation likelihoods (emission probabilities), each expressing the probability of an observation o_t being generated from a state *i*
$\pi = \pi_1, \dots, \pi_N$	π_i is the probability that the chain will start in state *i* $\sum_{i=1}^{N} \pi_i = 1$

1. **Markov Chains**: Model probabilities of seqeunces of random variables (states)
2. **Hidden Markov Chains**: states are not given, but hidden. This means that words are observed, but the POS tags are hidden.

This allows us to inferr hidden states from observations.

HMM tagger

- Again: strong assumptions
  - **Markov**: to predict the future **tag** in the sequence, all that matters is the current state (bigrams of tags)
    - Can be extended to trigrams
  - **Independence**: the probability of an output observation (**word**) o_i depends only on the state that produced the observation q_i
    - Not on previous observations o_{i-1}

- we're not taking into account the FULL history. We can e.g. exapnd into trigrams.

HMM tagger

- Formulation is same as before:

$$P(T|W) \approx P(t_1)P(t_2|t_1)\dots P(t_n|t_{n-1})P(w_1|t_1)P(w_2|t_2)\dots P(w_n|t_n)$$
$$\approx P(t_1)P(w_1|t_1)P(t_2|t_1)P(w_2|t_2)\dots P(t_n|t_{n-1})P(w_n|t_n)$$

Emission probabilities Transition probabilities

- We use the same formula as before, only this time w is representing the emission probability (i.e. the probability of observing a word given a tag) and a transition probability (i.e. the probability of observing a tag given a previous tag).

Probabilistic POS tagging

$P(t_i|t_{i-1})$ $P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$

t_0

	PROPN	VERB	PART	NOUN	PRON	DET
<s> (3)	1/3	1/3	0	0	0	1/3
PROPN (1)	0	1/1	0	0	0	0
VERB (6)	0					2/6
PART (2)	0					1/2
NOUN (2)	0	0	0	0	1/2	0
PRON (1)	0	1/1	0	0	0	0
DET (4)	0	1/4	1/4	2/4	0	0

Transition probabilities

Probabilistic POS tagging

$P(w_i|t_i)$ $P(w_i|t_i) = \frac{C(w_i,t_i)}{C(t_i)}$

	john	is	expect	to	race	this	the	I	want	bring
PROPN (1)	1/1	0	0	0	0	0	0	0	0	0
VERB (6)	0	2/6	1/6	0	1/6	0	0	0	1/6	1/6
PART (2)	0	0						0	0	0
NOUN (2)	0	0						0	0	0
PRON (1)	0	0	0	0	0	0	0	1/1	0	0
DET (4)	0	0	0	0	0	2/4	2/4	0	0	0

Emission probabilities

These two tables are the **HMM** model

Figure 2: Tables are unchanged

HMM for POS tagging

- **Decoding/inference**: task of determining the **hidden state sequence** corresponding to the **sequence of observations**
  - Given as input an **HMM** model **λ** and a sequence of observations $O = o_1 o_2 \dots o_T$ (test case), find the most probable sequence of states $Q = q_1 q_2 \dots q_T$

$$\hat{T} = argmax_T P(T|W)$$
$$\approx argmax \prod_{i=1}^{N} P(w_i|t_i) P(t_i|t_{i-1})$$

emission transition

- Here, the hidden state is the tag, and the seqeunce of observations are words.
- As before, we're not doing this greedily, we want to maximise the posterior, but instead we are now working with emission and transition probabilities.

1.4.1 Viterbi algorithm

HMM for POS tagging

- **Viterbi**

$v_t(j)$ = Viterbi path **probability** ('t' = column & j = row), i.e. probability that the HMM is in ***state j*** *(present POS tag)* after seeing the ***first t observations*** *(past words for which lattice values have been calculated)* and passing through the most **probable state sequence** (previous POS tag) $q\_1 \dots .q\_t{-}1$

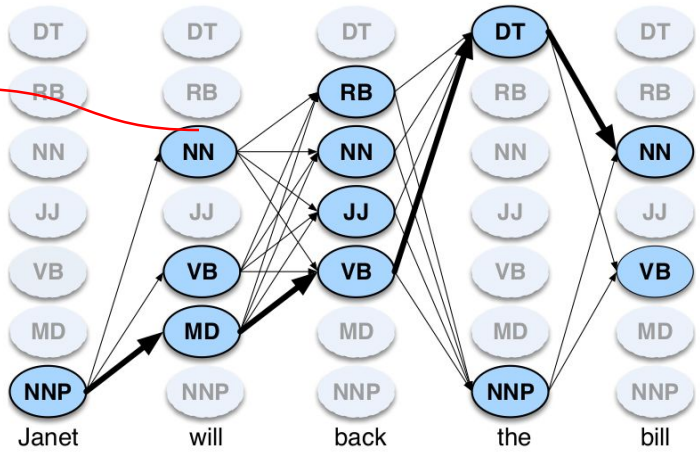


Figure from Jurafsky, D and Martin, J, "Speech and Language Processing," 2018

Figure 3: For a particular state, what is the porbability that the model is in state j, i.e. has that tag, after seeing the first p observations. Here, t is the time. and passing through the probable sequence.

HMM for POS tagging

- **Viterbi algorithm**:
  - Dynamic programming
  - First build a lattice/matrix
    - One column per observation and one row per state
    - Each node $v_t(j)$ is the **probability** that the HMM is in state *j* after seeing the first *t* observations and passing through the most probable state sequence $q_1, \dots, q_{t-1}$

HMM for POS tagging

- **Viterbi algorithm**:
  - The value of each $v_t(j)$ is computed by **recursively** taking the most probable path that could lead to this node:

$$v_t(j) = max_{q_1,\dots,q_{t-1}} P(q_1 \dots q_{t-1}, o_1 \dots o_t, q_t = j)$$

  - Probability of being in every state at time *t−1* is already computed, so Viterbi probability is simply the most **probable of the extensions of the paths** that lead to the current cell

Figure 4: Dynamic programming algorithm: we can reuse subprobelms.

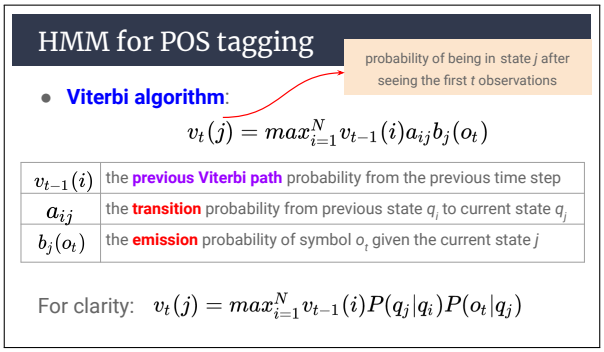
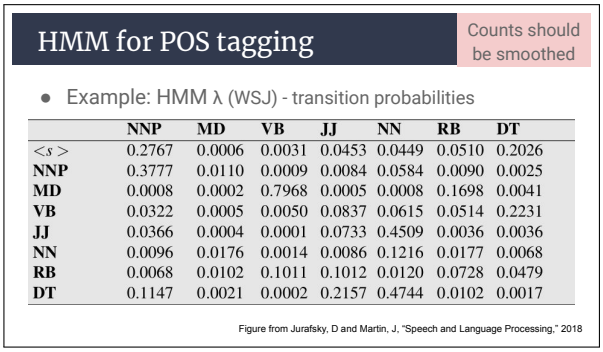


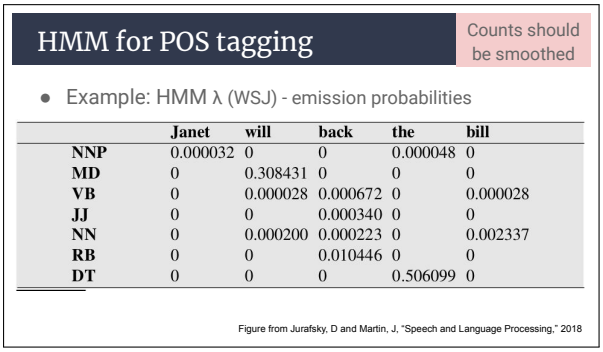
Figure 5: Recursive definition

1.4.2 Example



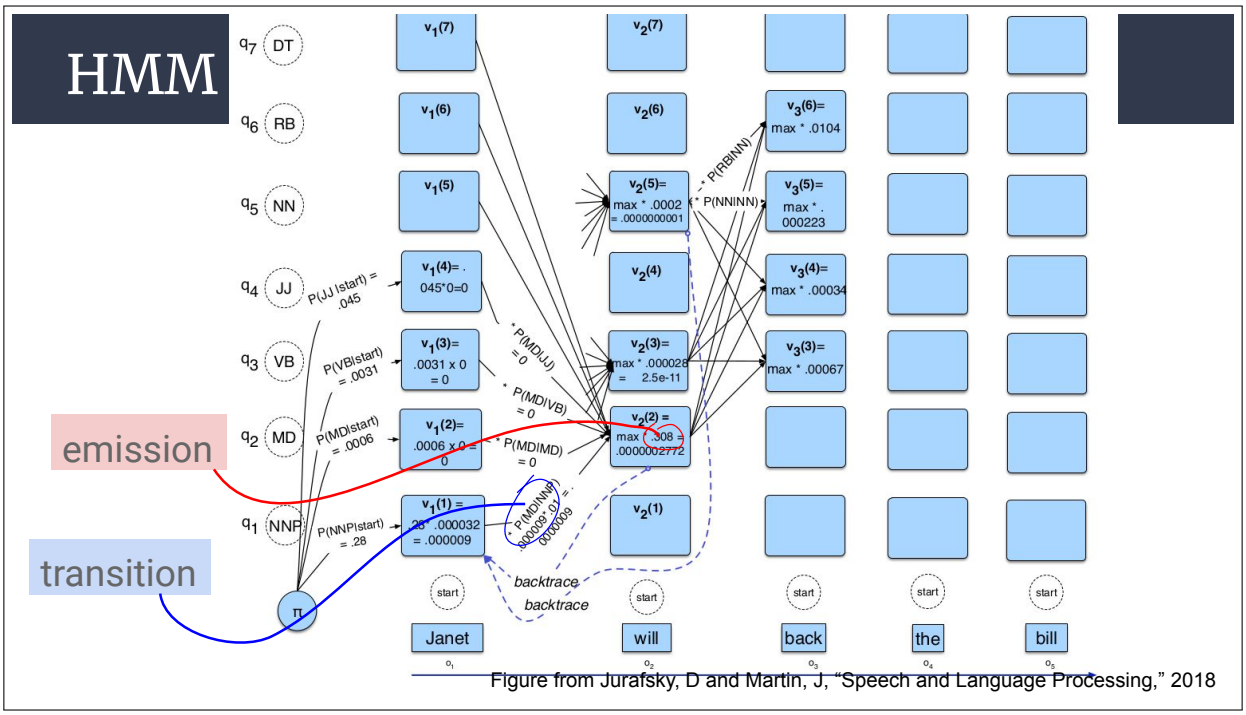
	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

(a) computed by counting



	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

(b) emission probabilities, e.g. we have never observed the word 'Janet' that is a verb

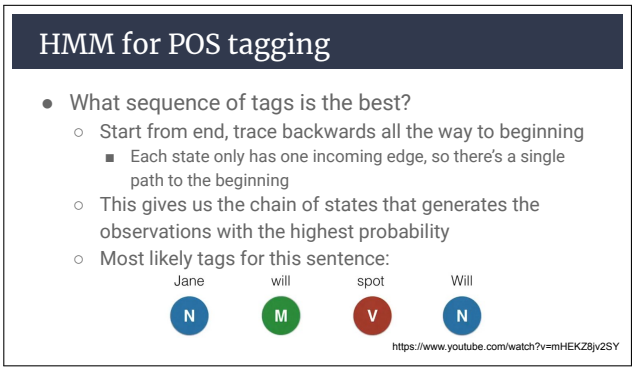


1. We have starting probabilities. e.g. what is the probability that we will transition into a noun phrase from the start token? It is given as .28 in the transition probabilities above.
2. Then we get to the blue box. Here, we compute the probability of the first word given the tag. e.g. the probability of the word 'Janet' given that it is a noun is 0.000032.

3. Then we transition from NNP to MD, and the probability of this transition is the probaility of Janet being an NNP multipled by MD following NNP which in the table above is 0.0110.

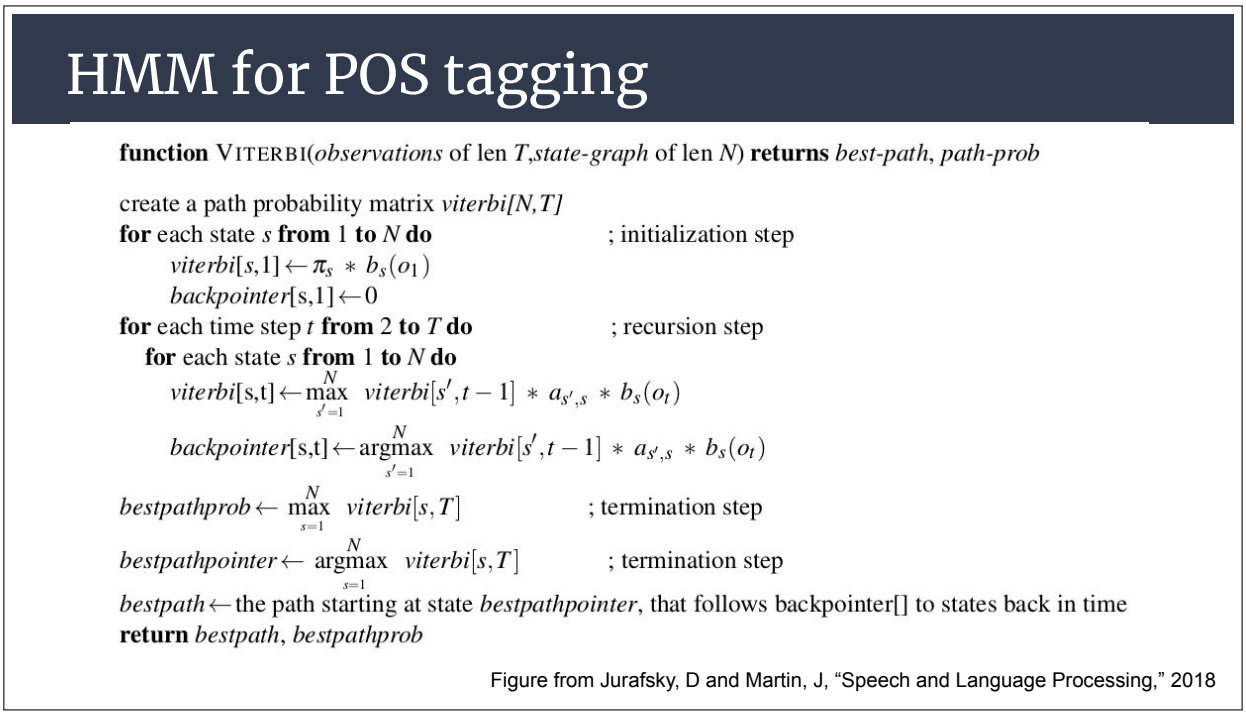
4. We then for each incoming array take the max and multiply it by the proabbility of will being an MD which is 0.308 from the emission table above.

We should realistically use log space instead of multiplying probabilities, but this is the general idea. This is becuase the probabilities are very small and multiplying them together will result in a very small number.



•

1.4.3 Pseudocode



1.4.4 Problem with HMM

HMM for POS tagging

- The number of possible paths grows exponentially with the length of the input
  - Viterbi's running time is $O(SN^2)$, where S is the length of the input and N is the number of states in the model
- Some tagsets are very large: 50 or so tags
  - **Beam search** as alternative decoding algorithm
    - At every step, only expand on top k most promising paths

- becuase we're considering every possible path.
- This is a problem becuase we're considering every possible path, and this is not scalable.
- We can use a beam search to limit the number of paths we consider.

HMM for POS tagging - beam search (k=2)

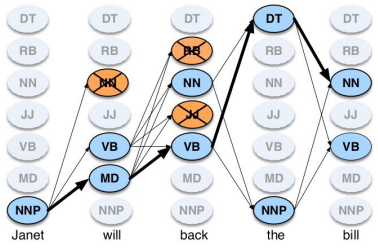


Figure from Jurafsky, D and Martin, J, "Speech and Language Processing," 2018

1.5 MEMM – Maximum entropy classifier

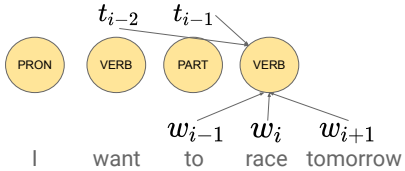
MEMM for POS tagging

- HMM is a generative model, powerful but limited in the features it can use
- **Alternative**: sequence version of logistic regression classifier - maximum entropy classifier (MEMM), a discriminative model to directly estimate posterior

$$\hat{T} = argmax_T P(T|W)$$
$$= argmax_T \prod P(t_i|w_i, t_{i-1})$$

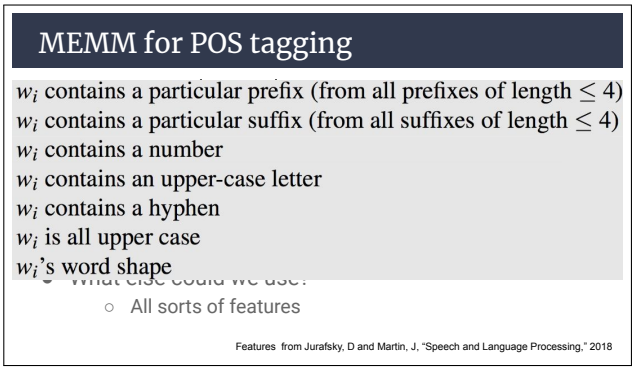
-

MEMM for POS tagging



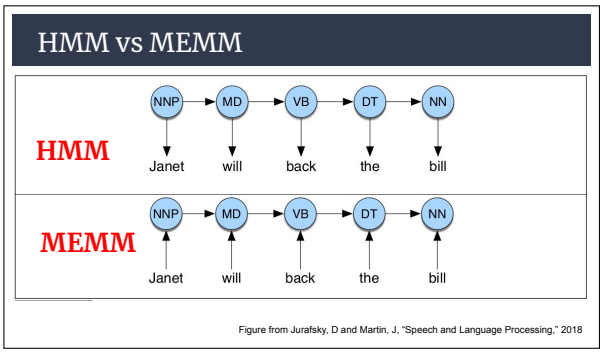
- What else could we use?
  - All sorts of features

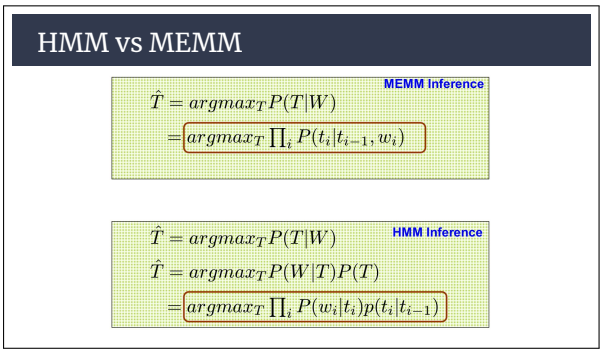
- instead of just using the immediate previous tag, we can use a window of words and previous n tags.

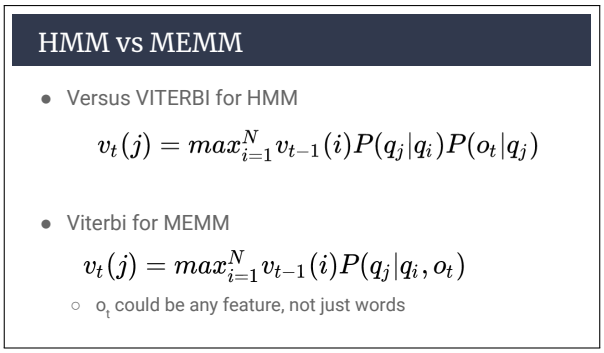


- we can use all sorts of features using this method
- if it contains an uppercase letter, e.g. in german, all nouns are capitalised, so this is a good feature to use.

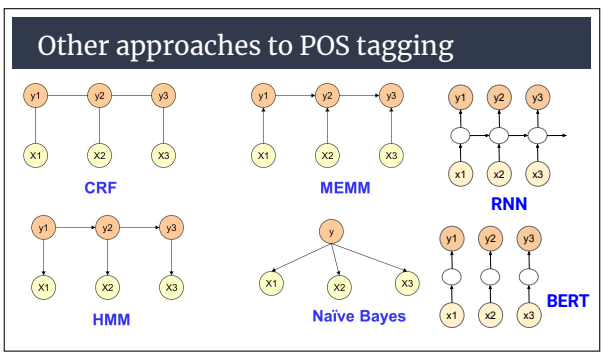
1.6 HMM vs MEMM







1.7 Other approaches



1.7.1 RNN for POS tagging

RNN for POS tagging

- RNN to assign a label from (small) tagset to each word in the sequence
  - **Inputs**: word embedding per word
  - **Outputs**: tag probabilities from a softmax layer over tagset
  - **RNN**: 1 input, 1 output, 1 hidden layer; *U*, *V* and *W* shared

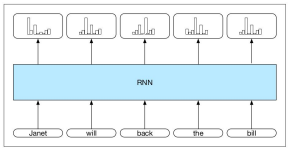


Figure from Jurafsky, D and Martin, J, "Speech and Language Processing," 2018

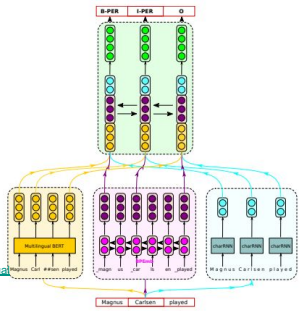
RNN for POS tagging

- **Training**
  - Cross-entropy loss over the tagset for each word
  - Sequence loss is the sum of loss for all words
- **Inference**
  - Run forward inference over the input sequence and select the most likely tag from the softmax at each step
  - Decision for each word in the sequence is taken independently from decision for other words - not optimising for **sequence** of tags

1.7.2 SOTA

SOTA

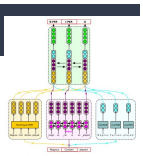
- Using Universal Dependencies (UD) as tagset for various languages
- Also used for Named Entity Recognition



Sequence Tagging with Contextual and Non-Contextual Subword Representations: A Multilingual Evaluation. Heinzerling† & Strube, 2019.

SOTA

- Ensemble of subword representations
- Input encoded using
  - Multilingual BERT (yellow, bottom left)
  - LSTM with BPEmb (pink, bottom middle)
  - Character-RNN (blue, bottom right)
- Meta-LSTM (green, center) combines the different encodings before classification (top)
- Horizontal arrows are bidirectional LSTMs.



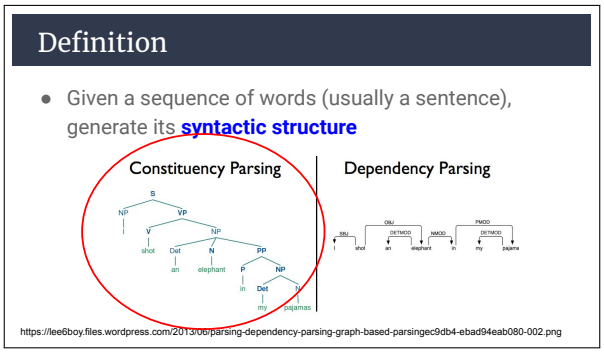
SOTA for POS tagging - high res langs.

Previous SOTA

Lang.	BiLSTM	Adv.	FastText	BPEmb	BPEmb +char	BPEmb +shape	BERT	BERT +char	BERT +char+BPEmb
Avg.	96.4	96.6	95.6	95.2	96.4	95.7	95.6	96.3	**96.8**
bg	98.0	98.5	97.7	97.8	98.5	97.9	98.0	98.5	**98.7**
cs	98.2	98.8	98.3	98.5	98.9	98.7	98.4	98.8	**99.0**
da	96.4	96.7	95.3	94.9	96.4	95.9	95.8	96.3	**97.2**
de	93.4	**94.4**	90.8	92.7	93.8	93.5	93.7	93.8	**94.4**
en	95.2	95.8	94.3	94.2	95.5	94.9	95.0	95.5	**96.1**
es	95.7	96.4	96.3	96.1	96.6	96.0	96.1	96.3	**96.8**
eu	95.5	94.7	94.6	94.3	**96.1**	94.8	93.4	95.0	96.0
fa	**97.5**	**97.5**	97.1	95.9	97.0	96.0	95.7	96.5	97.3
fi	**95.8**	95.4	92.8	92.8	94.4	93.5	92.1	93.8	94.3
fr	96.1	**96.6**	96.0	95.5	96.1	95.8	96.1	96.5	96.5
he	97.0	**97.4**	97.0	96.3	96.8	96.0	96.5	96.8	97.3
hi	97.1	97.2	97.1	96.9	97.2	96.9	96.3	96.8	**97.4**
hr	**96.8**	96.3	95.5	93.6	95.4	94.5	96.2	96.6	**96.8**
id	93.4	**94.0**	91.9	90.7	93.4	93.0	92.2	93.0	93.5
it	98.0	**98.1**	97.4	97.0	97.8	97.3	97.5	97.9	98.0
nl	93.3	93.1	90.0	91.7	93.2	92.5	91.5	92.6	93.3
no	98.0	98.1	97.4	97.0	98.2	97.8	97.5	98.0	**98.5**
pl	97.6	97.6	96.2	95.8	97.1	96.1	96.5	**97.7**	97.6
pt	97.9	98.1	97.3	96.3	97.7	97.2	97.5	97.8	98.1
sl	96.8	**98.1**	97.1	96.2	97.7	96.8	96.3	97.4	97.9
sv	96.7	96.7	96.7	95.3	96.7	95.7	96.2	97.1	**97.4**

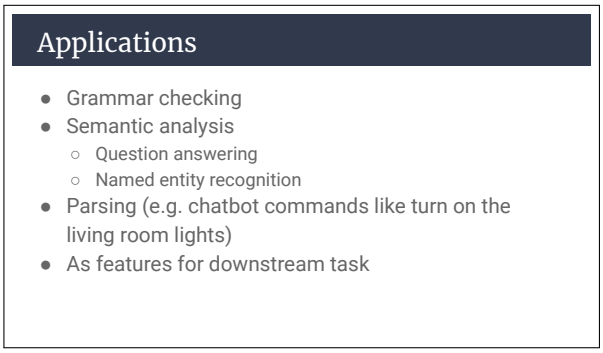
2 Constituence Parsing

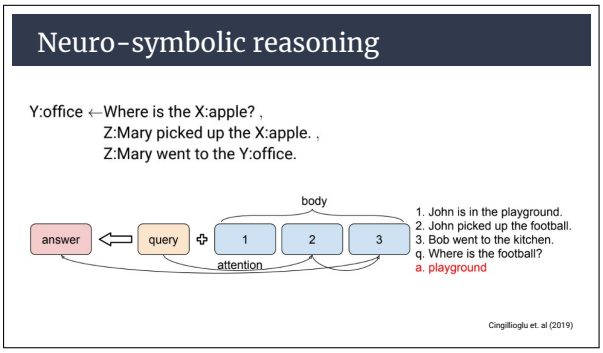
2.1 Introduction



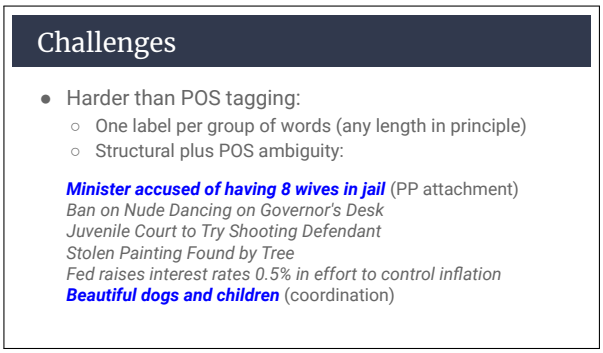
Constituency parsing is more so interested in the syntactic startucture of a sentence.

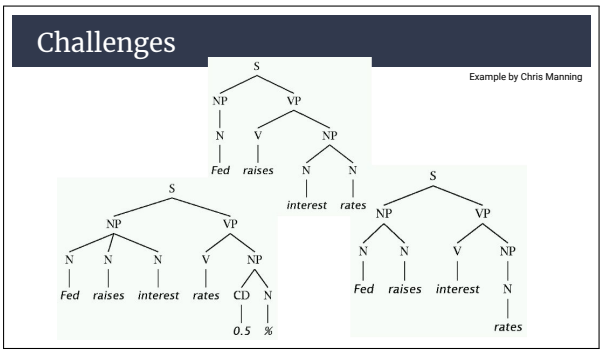
2.1.1 Applications





2.1.2 Challenges





2.1.3 Classical Parsing

Classical parsing

- Given a **grammar** and a **lexicon**

Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid the \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

Figure from Jurafsky, D and Martin, J, "Speech and Language Processing," 2018, ch 13

- we can parse language like we do a programming language

Constituency parsing

- Based on the idea of phrase structure
  - Words are grouped into constituents
- A constituent is a sequence of words that behaves as a **unit**, generally a phrase
  - There are tests for that, e.g. distribution: can I move phrases around?
    - John talked [to the children] [about drugs].
    - John talked [about drugs] [to the children].

- A constituent is a sqeunce of words that behaves as a unit, generally a phrase.

Classical parsing - more formally

Phrase structure grammar is a **context-free grammar** **G = (T, N, S, R)**, where:

- T is set of terminals
- N is set of nonterminals
- S is the start symbol (non-terminal)
- R is set of rules X $\rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals & nonterminals

A grammar G generates a language L, or L is recognised by G.

- we have a universe of grammar rules, and we can use these to parse the sentence.

Classical parsing

- To do this automatically
  - Use proof systems to prove parse trees from words
  - Search problem: all possible parse trees for string
    - Bottom-up
      - Words to grammar
    - Top-down
      - Grammar to words

- Automatically, we can serach all possible parse trees.
- We can do bottom up or top down.

2.2 The CKY Algorithm

The CKY algorithm

- Tests for possibilities to split the current sequence into **two** smaller sequences
- Grammar needs to be in Chomsky Normal Form (CNF)
  - Rules are of the form X →Y Z or X → w
  - Deterministic process to convert any CFG into CNF

VP → V NP PP ➡ VP → V NEW
NEW → NP PP

INF-VP → to VP ➡ INF-VP → TO VP
TO → to

- it tires to split the input into two parts, and then recursively parse the two parts.
- It has to be in Chomsky Normal Form; it means rules are only binary (two non binaries) or a non terminal that evaluates to a non-terminal.

The CKY algorithm

- Binarisation makes CKY very **efficient**
  - $O(n^3|G|)$: n is the length of the parsed string; |G| is the size of the CNF grammar G
  - Otherwise it would be exponential
- **Dynamic programming** algorithm to efficiently generate all possible parse trees bottom-up

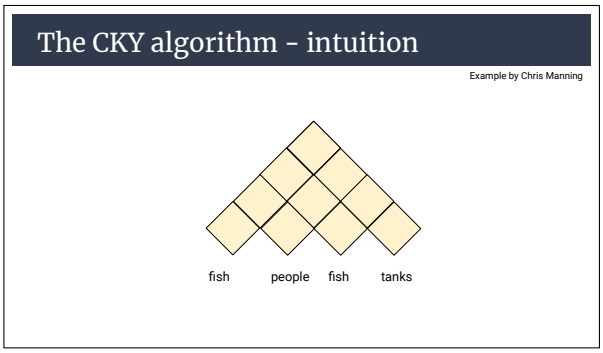
- We can use dynamic porgramming!

The CKY algorithm

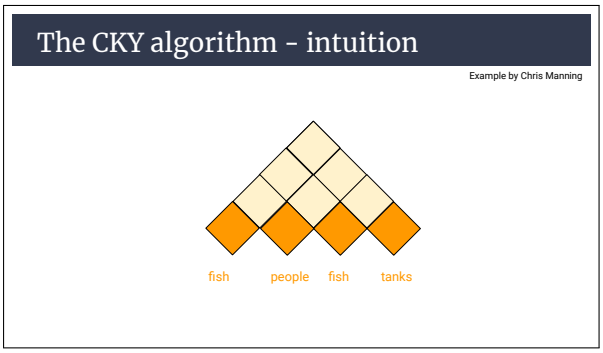
- Use a two-dimensional matrix n×n, where n is length of sentence, to encode the possible parses
- Use the upper-triangular portion of the matrix
- Each cell [i, j] in matrix contains the set of non-terminals that represent all the constituents that span positions i-j of the input
- The cell that represents the entire input resides in position [0,n] of the matrix (row x column)

- we start with a matrix, and use the uipper traingular portion of the matrix (since left-to-right parsing).

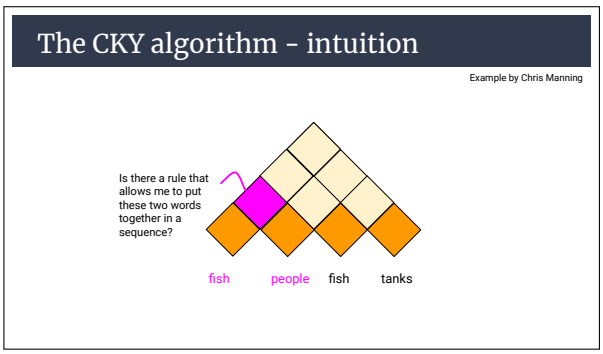
2.2.1 Intuition



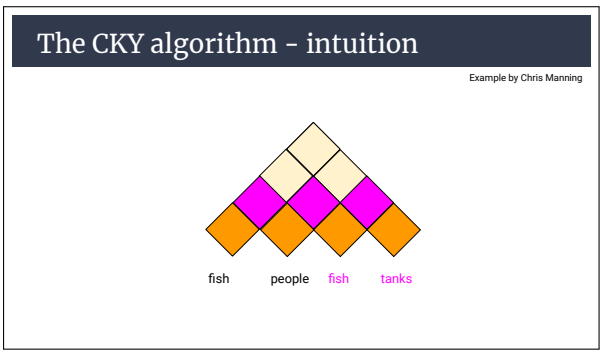
(o) we want to parse this short sentence



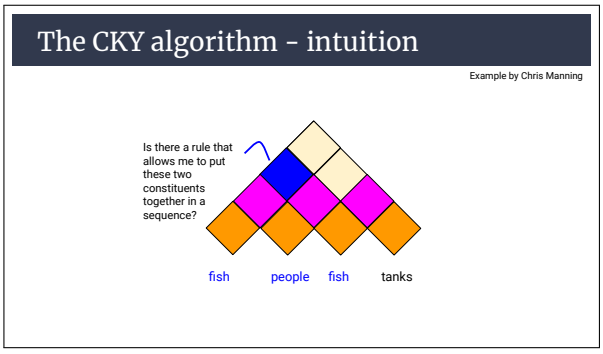
(p) we consider each individual word and see if it is a constituent or not.



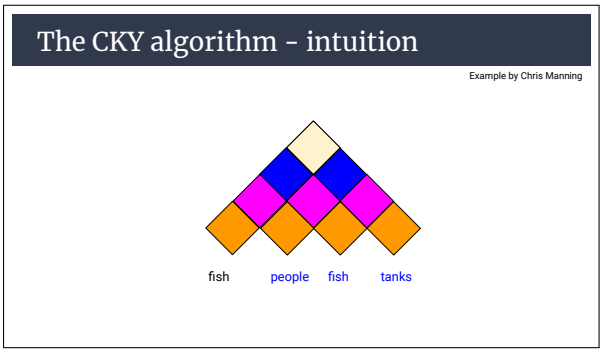
(q) Since constituents can be any length, we then ask ourselves if there is a rule that allows us to put these two words into a sqeunce



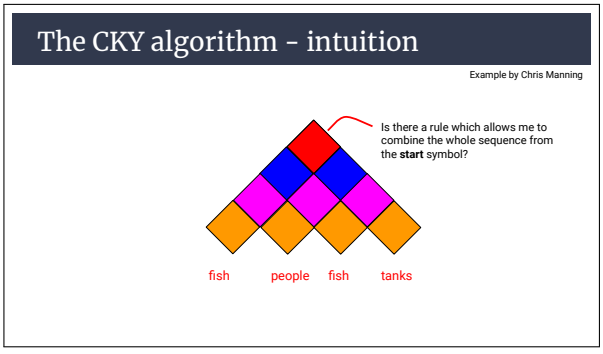
(r) We then do the same for people fish, and fish tanks



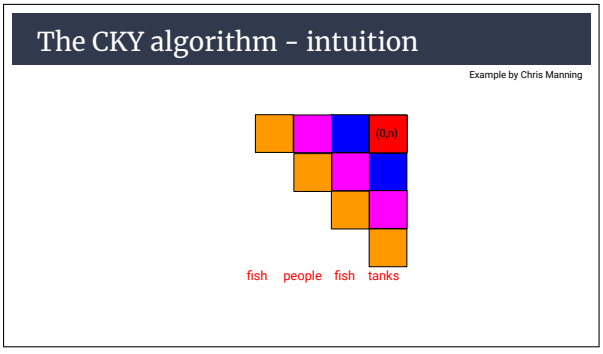
(s) We then consider constituents of level 3



(t) likewise here



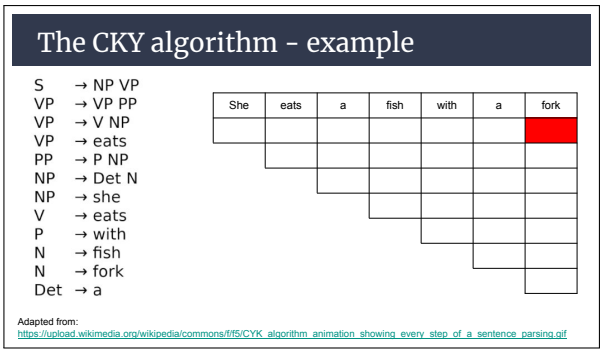
(u) you then consider is there a rule that allows you to combine the whole sequence from the start symbol



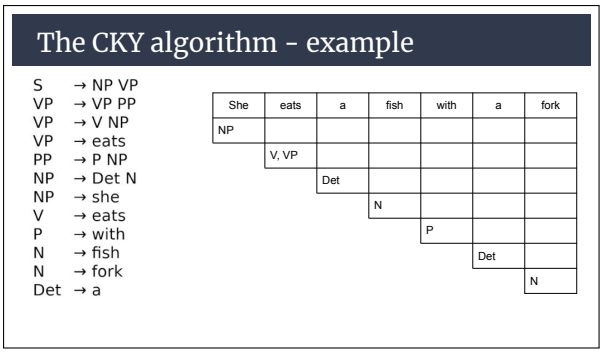
(v) Then, once flipped sideways this is the upper triangular of the matrix. then, the top right gives you the parse tree.

Figure 6: Intuition

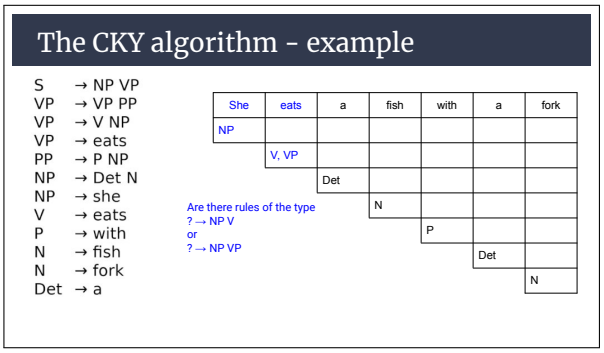
2.2.2 Example



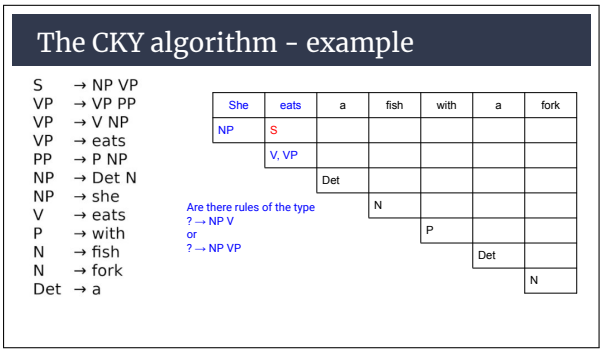
(a) we want to find the constituents of the sentence "she eats a fish with a fork"



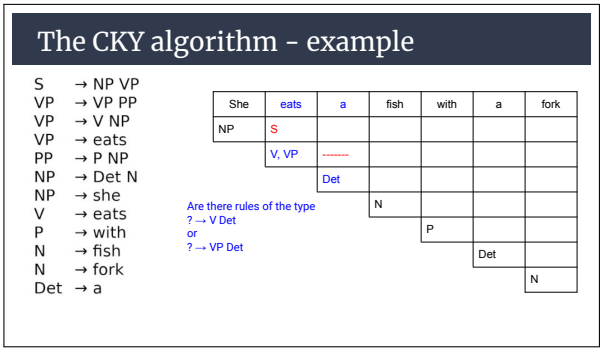
(b) consider each word on its own, does it form a constituent? E.g. 'she' attaches itself to a non-terminal NP. Furthermore, if more than two are applicable e.g. eats then we save both.



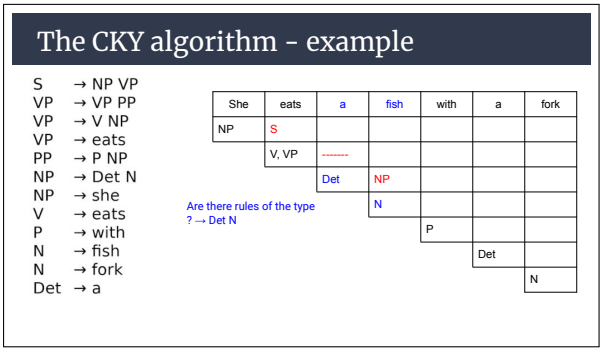
(c) consider the next level of length 2



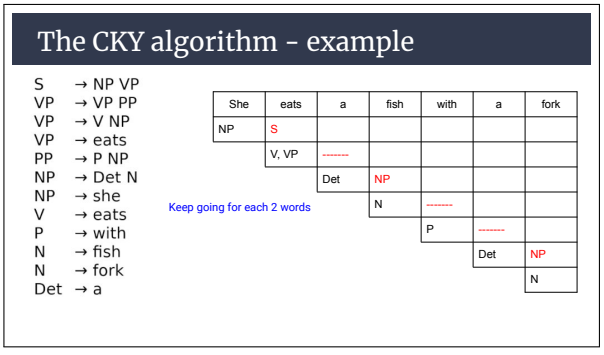
(d) this matches!



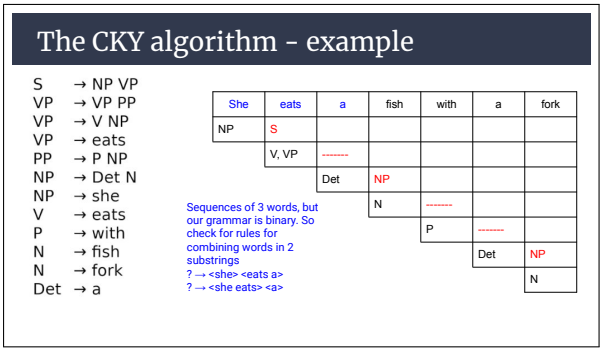
(e) same here, no rules apply



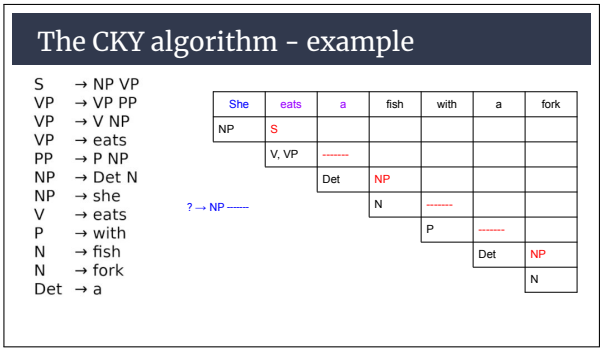
(f) we can apply one level of constituency here



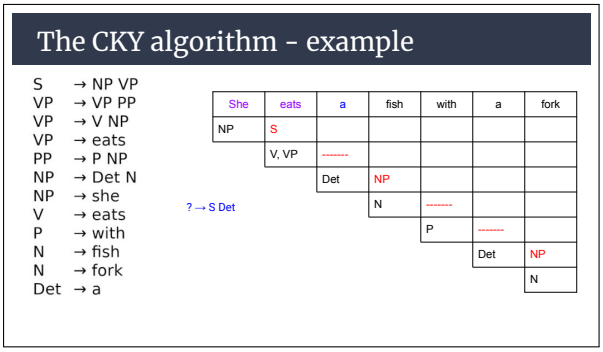
(g) We apply this to all similarly



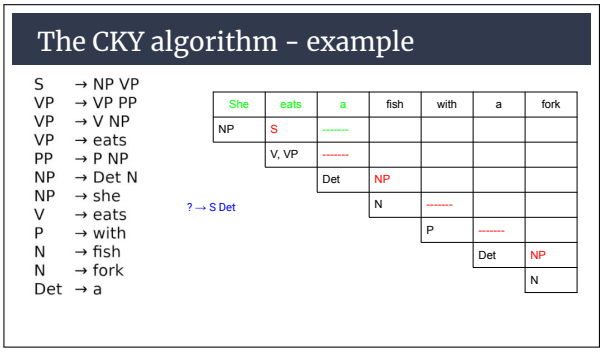
(h) now ask the qeustion, for the third level. However, we have applied Chmsky normal form which is binary. Therefore, look at all possible ways that we can form a binary constituent rule. therefore we consider the possible splits above



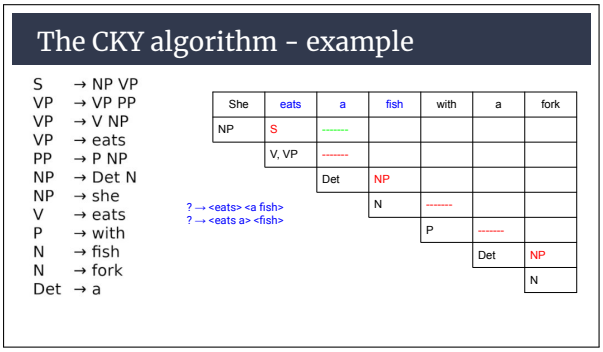
(i) For the first split, there is nothing that we can match to this



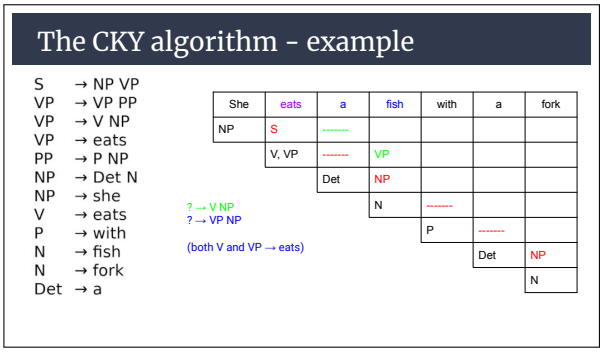
(j) fpr the second split, there is nothing also



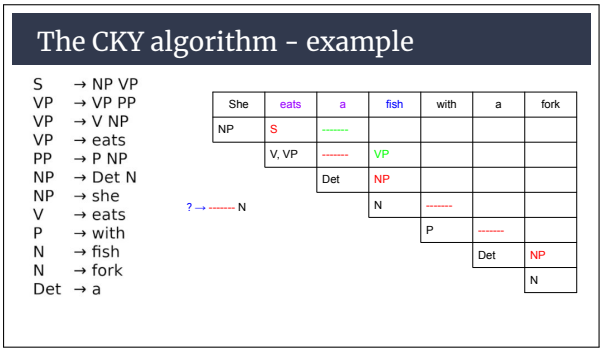
(k) therefore in total, we have a blank rule here also



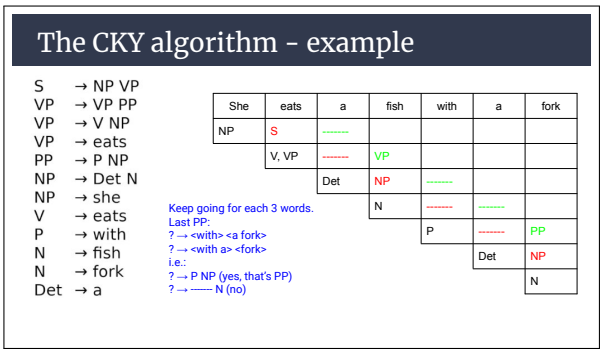
(l) we then move on to the next substring



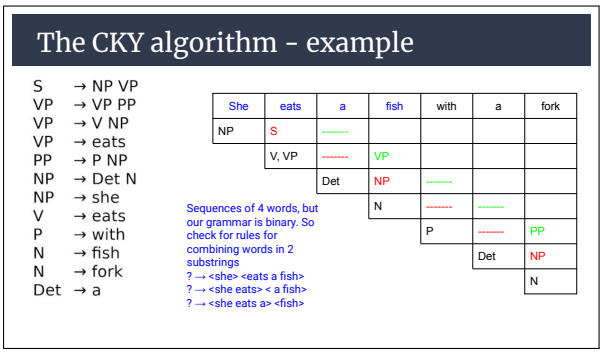
(m) for the first chunk, we have a match



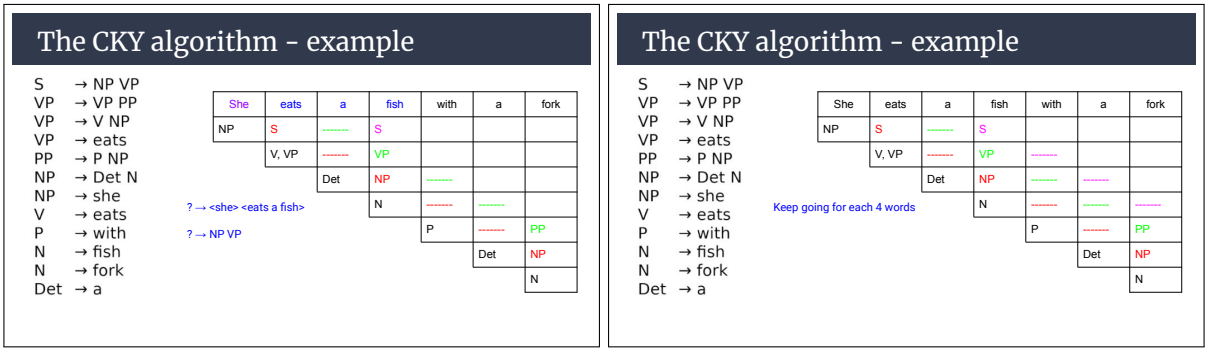
(n) for the second we do not. Therefore, in total, we only have one possible match



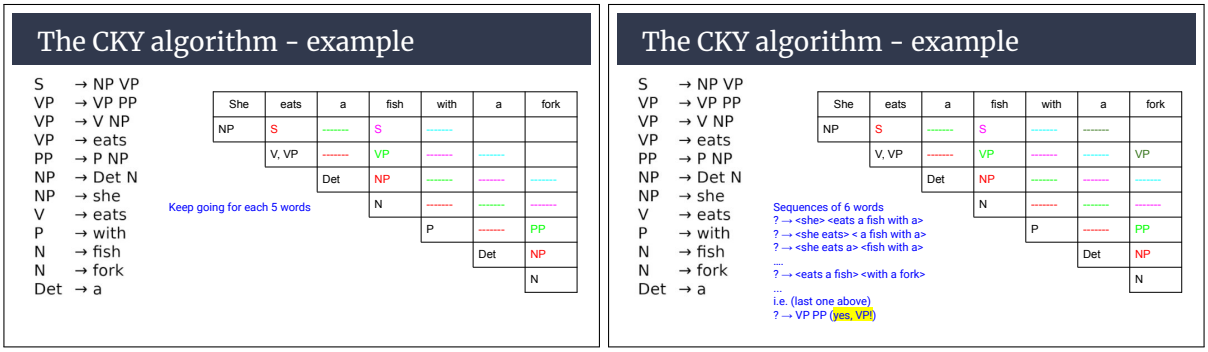
(o) We continue this for all 3 length substrings

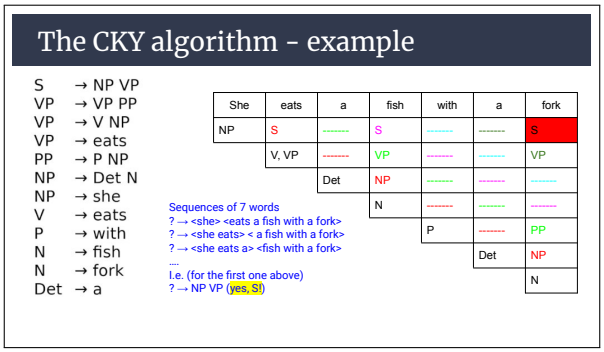


(p) We do the same for substring of 4, with all possible splits



(q) we find that for one of these, we have a match!

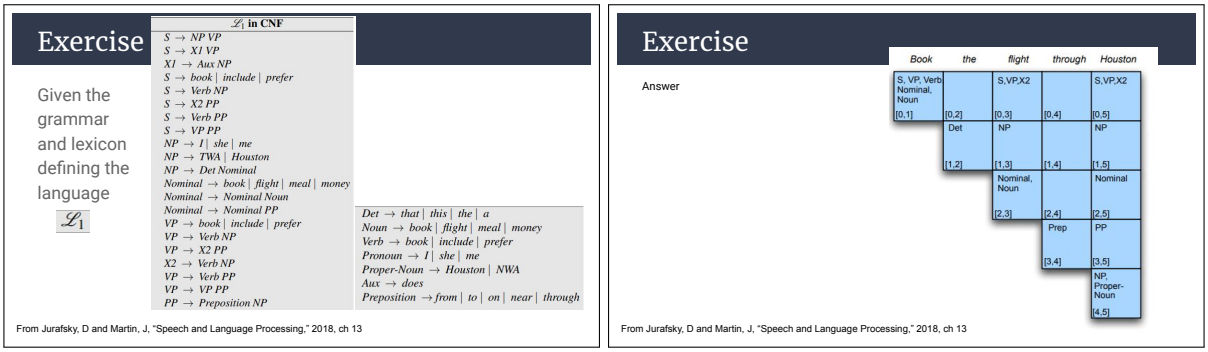




(u) We have created a constituency

Figure 6: Example part 3

2.2.3 Example 2



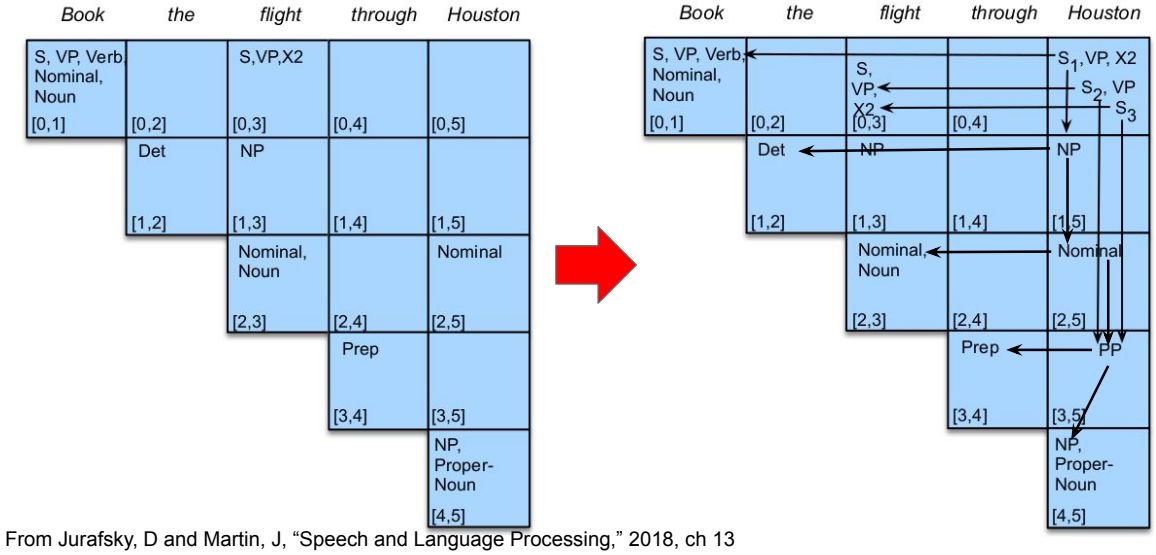
2.3 CKY for parsing

CKY for parsing

- So far: CKY for recognition
  - Fill out matrix such that [0,n] cell has symbol S
- To use CKY for parsing, need to
  1) Add **backpointers**: augment entries in matrix s.t. each non-terminal is paired with **pointers** to the matrix entries from which it was derived
  2) Allow multiple copies of the same non-terminal to be entered into the matrix to track multiple paths
- Choose S from cell [0, n] and recursively retrieve its component constituents from the matrix

- so far, this covers recognition
- however, for parsing (what is the actual structure?) we include pointers for back-pointers.
- This gives you the actual tree-structure

CKY for parsing



From Jurafsky, D and Martin, J, "Speech and Language Processing," 2018, ch 13

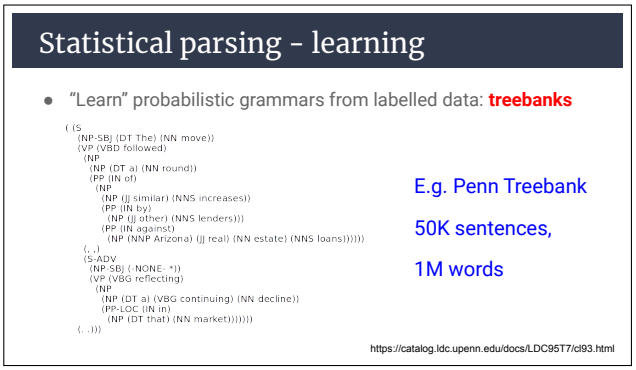
2.4 Statistical Parsing

Statistical parsing

- This does **not scale**: too many possible parse trees for comprehensive grammar
- Solution:
  - Find the most likely parse(s) via statistical parsing
  - Comprehensive grammars admit many parses for a sentence, but we can efficiently find the most likely parse

Statistical parsing

- this doesn't scale for comprehensive grammar
- Therefore, use statistical parsing finding the most probabble path as oppose to looking at all the possible passages that we have



- We get rules and probabilities from a tree-bank or corpus.
- Here the sentence is "The move followed a round of similar increases by other leders against Arizona real estate loans"
- this gives an insight into what the structure of english language is like.

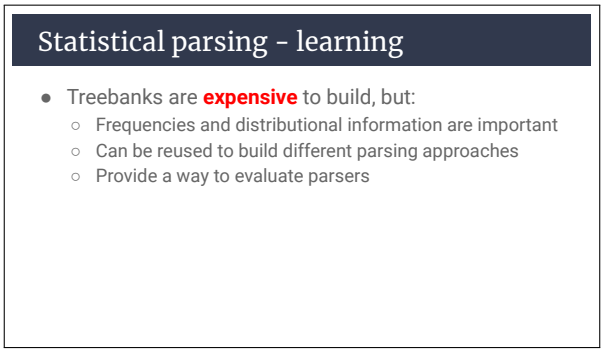
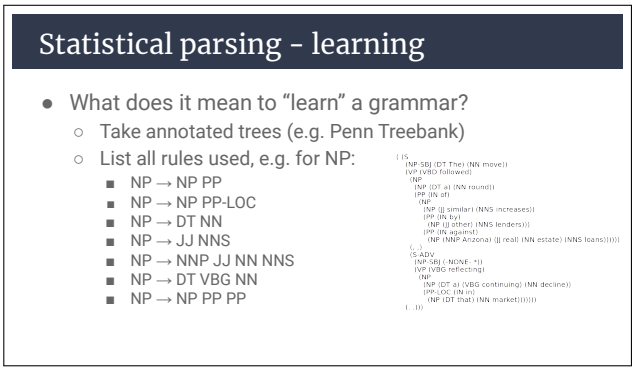
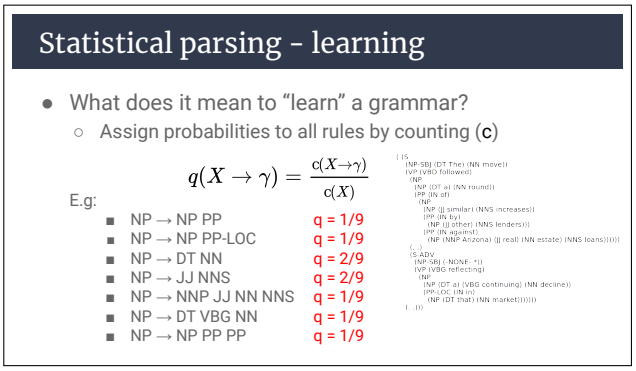


Figure 7



- We then take annotated grammars and list all the rules used.
- Here, there are way more non-terminals being used



- we then caluclate the proabbility of each rule being used.
- we find this probability by counting (how many times have i seen this rule, vs how many times i have seen this non-terminal in general?)

Statistical parsing - formally

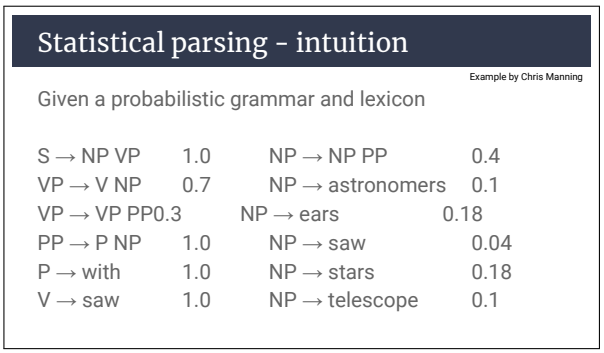
Probabilistic/stochastic phrase structure grammar is a context-free grammar **PCFG = (*T, N, S, R, q*)**, where:

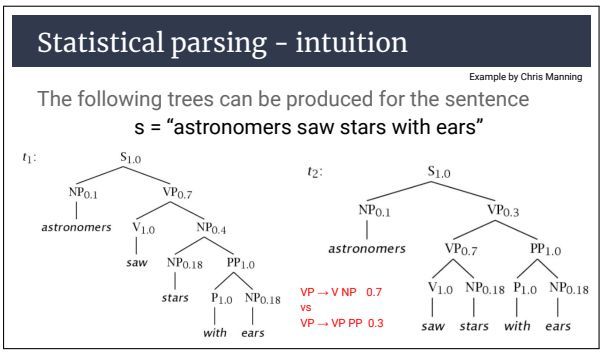
- *T* is set of terminals
- *N* is set of nonterminals
- *S* is the start symbol (non-terminal)
- *R* is set of rules $X \rightarrow \gamma$, where *X* is a nonterminal and γ is a sequence of terminals & nonterminals
- *q = P(R)* gives the probability of each rule

$$X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$$

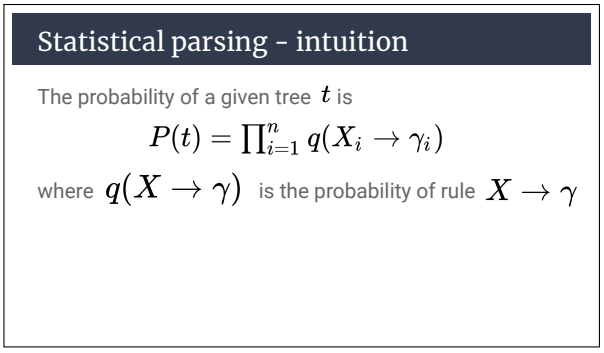
- From there create a probabilistic context free grammar.
- it gives a probability for each rule.

2.4.1 Intuition

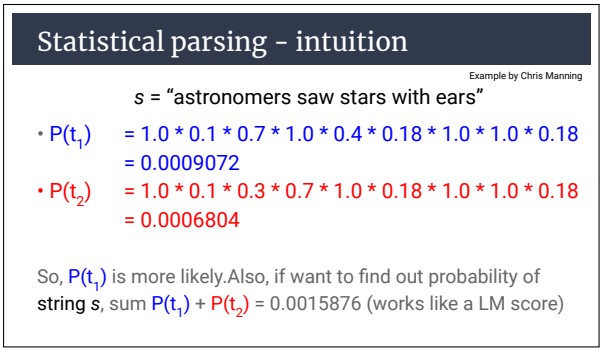




(b) This probabilities leads to different possible parse trees where both are valid, but less/more likely



(c) probability of a given tree is just the multiplication of each rule being used



(d) for the two parse trees, we conclude the first parse tree is more likely.

The probability of each of the trees is obtained by multiplying the probabilities of each of the rules used in the derivation
This works as a disambiguation method!
This brute-force approach (enumerating all options) does not scale...

Figure 8: Intuition

2.4.2 More Formally

Statistical parsing - formally

Probabilistic/stochastic phrase structure grammar is a context-free grammar **PCFG = (*T, N, S, R, q*)**, where:

- *T* is set of terminals
- *N* is set of nonterminals
- *S* is the start symbol (non-terminal)
- *R* is set of rules $X \rightarrow \gamma$, where *X* is a nonterminal and γ is a sequence of terminals & nonterminals
- *q = P(R)* gives the probability of each rule

$$X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$$

- As above

Statistical parsing - more formally

- Given all the possible parse trees τ for a given sentence *s*. The **string of words** *s* is called the **yield** of any parse tree over *s*
- **Out of all parse trees that yield *s***, the algorithm picks the parse tree *t* that is most probable given *s* ($t \in \tau(s)$):

$$\hat{t}(s) = \text{argmax}_{t \in \tau(s)} P(t|s)$$

- we are now looking at a parse tree, and we want to find the most likely parse-tree.
- each parse tree will have a probability

Statistical parsing - more formally

$$P(t, s) = \prod_{i=1}^{n} q(X_i \rightarrow \gamma_i)$$

- By definition $P(t, s) = P(t)P(s|t)$
- But $P(s|t) = 1$, since a parse tree includes all words in sentence, i.e.

$$P(t, s) = P(t)P(s|t) = P(t)$$

- Thus

$$\hat{t}(s) = \text{argmax}_{t \in \tau(s)} P(t)$$

- we expand this joint probability
- "we can do the marginal and say that it depends on the proability of the parse tree and then the proabbility of the sentence given the parse tree. This is equal to 1 because we are considering all possible pass trees; we don't consider parse trees that don't yield the sentence s."

2.5 The CKY algorithm for PCFG

The CKY algorithm for PCFG

- Same dynamic programming algorithm, but now to find most likely parse tree $\hat{t}(s) = \text{argmax}_{t \in \tau(s)} P(t)$
- Useful for
  - **Recognition**: does this sentence belong to the language?
  - **Parsing**: give me a possible derivation
  - **Disambiguation**: give me the best derivation
- All in polynomial time

- all done in polynomial time.

The CKY algorithm for PCFG

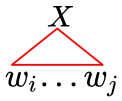
- Grammar needs to be in CNF, as before
  - Rules are of the form X →Y Z or X → w
  - Modify probabilities s.t. the probability of each parse remains the same under the new CNF grammar. E.g.:
    VP → Vt NP PP 0.2 ➡ VP → New PP 0.2
    New → Vt NP 1.0
- Build matrix as with CKY before
- Other **unary rules** can be included, e.g. NP → N, as long as they don't lead to loops

- this is the same algorith, but this time its for the probabilisitc context free grammar.
- We have to be careful that the probabilities don't change when introducing subrules

The CKY algorithm for PCFG

This is a hierarchical process:

- *n* is the number of words in sentence
- $w_{1\,n} = w_1 \dots w_n$ = the word sequence from *1* to *n*
- $w_{i\,j}$ = the subsequence $w_i \dots w_j$
- $X_{i\,j}$ = the nonterminal X dominating $w_i \dots w_j$



The CKY algorithm for PCFG

- Define the dynamic table

$\pi[i, j, X]$ = maximum probability of a constituent with non-terminal X spanning words $i \dots j$ inclusive

$i = 1 \dots n$ $\quad i <= j$ $\quad j = 1 \dots n$ $\quad X \in N$

- With the goal to calculate

$max_{t \in \tau(s)} p(t) = \pi[1, n, S]$

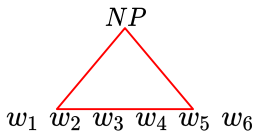
Highest scoring parse tree for entire sentence

- now we're looking at the dynamic table with three elements.

The CKY algorithm for PCFG

- For example, say sentence is $w_1\ w_2\ w_3\ w_4\ w_5\ w_6$
- And we take $\pi[2, 5, NP]$

The parse tree with the highest probability for words 2-5, whose head is NP



- There may be multiple possible ways in which **NP** has a parse tree under it that spans **words 2-5**
- Every of those parse trees will have a probability (product of the rule probabilities in that parse tree)

- "if i have this dynamic table what is the highest probability for the span 2-5 whose head is NP"
- This is the same dynamic algorithm where we use previous probabilities to calculate the tree

The CKY algorithm for PCFG

- **Base case**: for all $i = 1 \ldots n, X \in N$

$$\pi[i, i, X] = q(X \rightarrow w_i)$$

Terminals / unary rules

where $q(X \rightarrow w_i) = 0$ if $X \rightarrow w_i$ is not in grammar

- **Recursive case**: for all $i = 1 \ldots n-1, j = (i+1) \ldots n, X \in N$

$$\pi[i, j, X] = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \rightarrow YZ) \times \pi[i, s, Y] \times \pi[s+1, j, Z])$$

Split point

Recursive because the on $\pi[i, j, X]$ calculate based on π for substrings i to s and s+1 to j

For the base case, this is simply a match on a rule on a single tag. In the recursive case: "it has non terminals Y and Z, for the span i and j what is the maximum probaility i can get for this tag, i have to consider every posisble rule that has the tag. here, s is the split"

The CKY algorithm for PCFG - example

$$\pi[i, j, X] = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \rightarrow YZ) \times \pi[i, s, Y] \times \pi[s+1, j, Z]$$

the dog saw the man with the telescope

1 2 3 4 5 6 7 8

$\pi[3, 8, VP]$

$q(VP \rightarrow Vt\ NP) \times \pi[3, 3, Vt] \times \pi[4, 8, NP]$

$q(VP \rightarrow Vt\ NP) \times \pi[3, 4, Vt] \times \pi[5, 8, NP]$

....

$q(VP \rightarrow Vt\ NP) \times \pi[3, 7, Vt] \times \pi[8, 8, NP]$

$q(VP \rightarrow VP\ PP) \times \pi[3, 3, VP] \times \pi[4, 8, PP]$

$q(VP \rightarrow VP\ PP) \times \pi[3, 7, VP] \times \pi[8, 8, PP]$

VP → Vt NP 0.4

VP → VP PP 0.6

$s \in \{3, 4, 5, 6, 7\}$

- Here, for the span 3-8 we apply the recursive case.

$$\pi[3, 7, VP] = \max_{s \in \{3 \ldots 8\}} (q(X \rightarrow YZ) \times \pi[3, s, Y] \times \pi[s+1, 8, Z])$$

We then expand s and find the maximum probability for the span 3-7 with the head VP

$$q(X \rightarrow YZ) \times \pi[3, 3, Y] \times \pi[4, 8, Z]$$
$$q(X \rightarrow YZ) \times \pi[3, 4, Y] \times \pi[5, 8, Z]$$
$$q(X \rightarrow YZ) \times \pi[3, 5, Y] \times \pi[6, 8, Z]$$
$$q(X \rightarrow YZ) \times \pi[3, 6, Y] \times \pi[7, 8, Z]$$
$$q(X \rightarrow YZ) \times \pi[3, 7, Y] \times \pi[8, 8, Z]$$

Here, π is the probability of the parse tree that we calculated in the previous steps

And $(q \rightarrow YZ)$ is the probability of the rule that we have seen in the treebank that matches the given substring

The CKY algorithm for PCFG

Pseudocode by Michael Collins

Input: a sentence $s = x_1 \dots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.

Initialization:

For all $i \in \{1 \dots n\}$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \to x_i) & \text{if } X \to x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- For $l = 1 \dots (n-1)$
  - For $i = 1 \dots (n-l)$
    - Set $j = i + l$
    - For all $X \in N$, calculate

$$\pi(i, j, X) = \max_{\substack{X \to YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \to YZ) \times \pi(i, s, Y) \times \pi(s+1, j, Z))$$

and

$$bp(i, j, X) = \arg \max_{\substack{X \to YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \to YZ) \times \pi(i, s, Y) \times \pi(s+1, j, Z))$$

We always keep the most probably parse tree. We also record the backpointer to maintain the actual tree used to parse that substring.

2.6 Evaluating Parsers

Evaluating parsers

- **Parseval** metrics: evaluate structure
  - How much of constituents in the **hypothesis parse tree** look like the constituents in a **gold-reference parse tree**
  - A constituent in hyp parse is labelled "correct" if there is a constituent in the ref parse with the same yield and LHS symbol
    - Only rules **from non-terminal to non-terminal**
  - Metrics are more **fine-grained** than full tree metrics, more robust to localised differences in hyp and ref parse trees

Evaluating parsers

Precision = # of correct constituents in hyp parse of *s* / # of total constituents in hyp parse of *s*

Recall = # of correct constituents in hyp parse of *s* / # of total constituents in ref parse of *s*

F-measure = 2*Precision*Recall / Precision + Recall

Evaluating parsers - example

Example by Philipp Koehn

Hyp: X:a, Y:b, Z: cd, W: abcd

Ref: X: a, Z: b, V: cd, Y: bcd, W: abcd

These are flattened constituents.

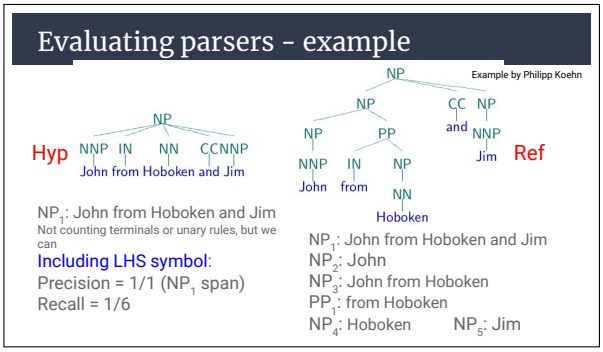
Discarding LHS symbol:

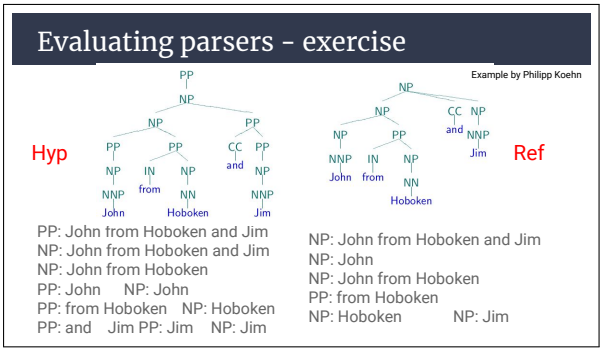
Precision = 4/4

Recall = %

F-measure = ...

- Here our output is 100% correct, however, we didn't predict one of the keys: V.

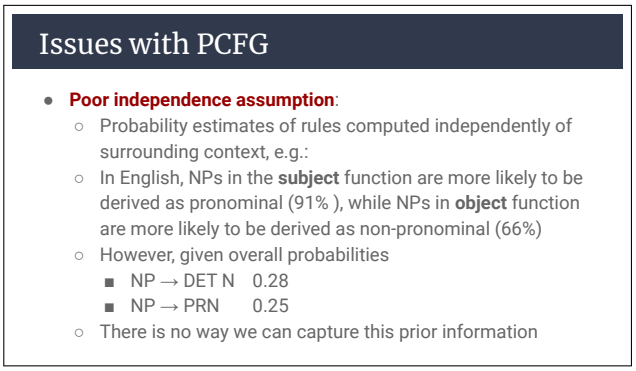




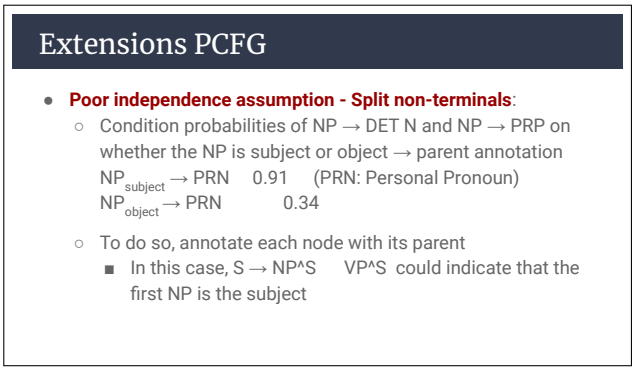
2.7 Issues with PCFG

2.7.1 Poor independence assumption

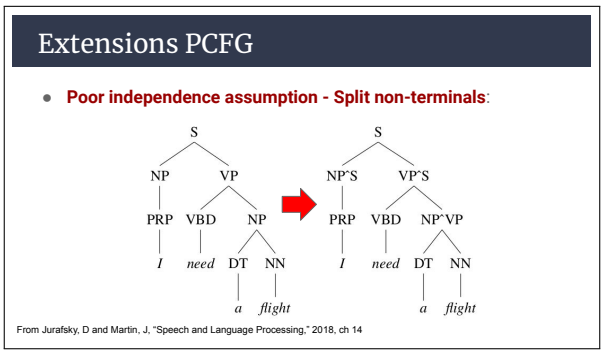
CFG rules impose an independence assumption on probabilities that leads to poor modelling of structural dependencies across the parse tree. Word is only dependent on its POS tag!



- e.g. “the boy jumped” the noun-phrase is “the boy” but in english it is more likley to be defined as pronominal (pronoun).
- with these rules, we cannot include this information

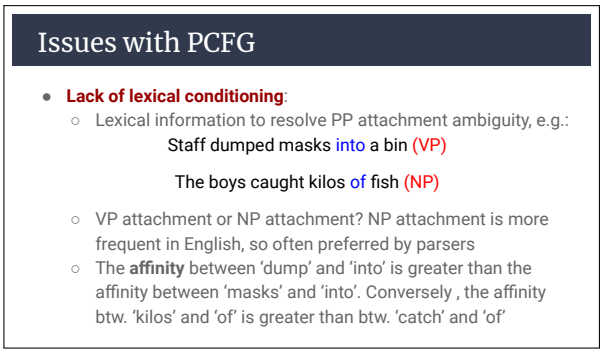


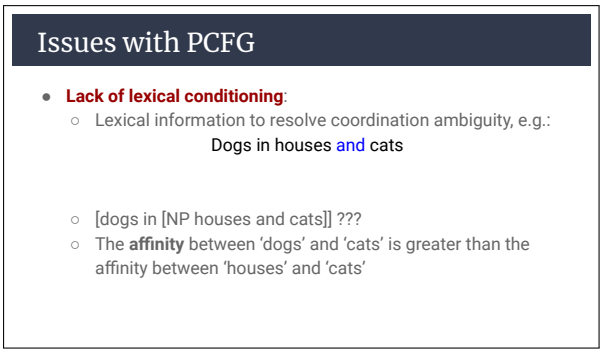
- we want to condition probabilities based on the context
- we can annotate each node with its parent.



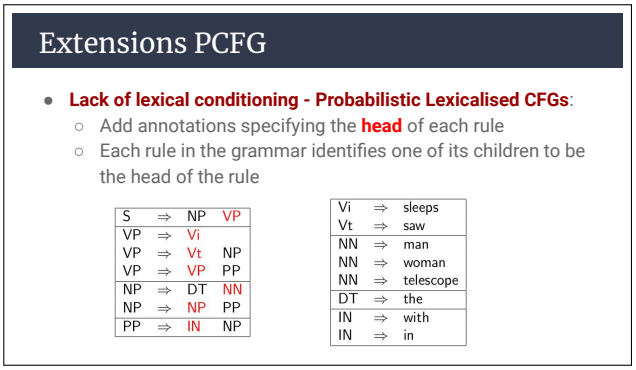
2.7.2 Lack of lexical conditioning

CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities

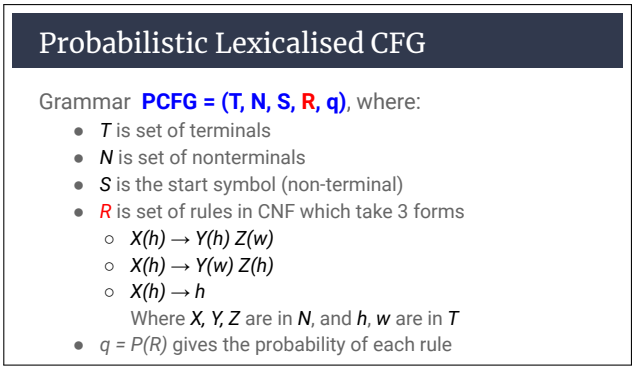




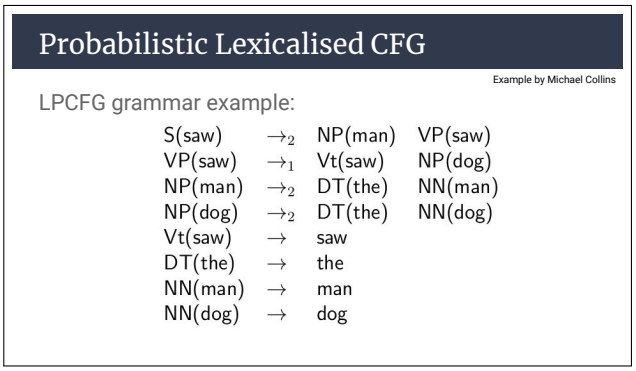
2.8 Probabilistic Lexicalised CFG



- create a head to the rule
- such that it is going to be the rule that identifies one of its children to the head.



- Formally, everything else is the same, but each rule has a head of the following form:



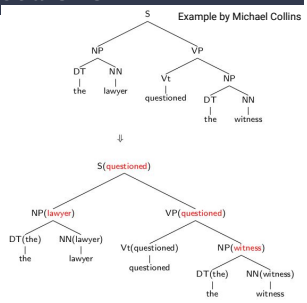
- This is a lexicalised CFG, instead of saying that a dog is a noun phrase, then we say the Noun phrase of the dog is dog.

Probabilistic Lexicalised CFG

- A constituent receives its headword from its headchild

S → NP VP (S receives from VP)
VP → Vt VP (VP receives from Vt)
NP → DT NN (NP receives from NN)

Example by Michael Collins



- that creates this transfomration, which brings the information in.

Probabilistic Lexicalised CFG

- **Head** is core linguistic concept, the central sub-constituent of each rule
- Treebanks not annotated for that, but can use rules to identify head

-

Probabilistic Lexicalised CFG

Example by Michael Collins

If the rule contains NN, NNS, or NNP: Choose the rightmost NN, NNS, or NNP Rule 1

Else If the rule contains an NP: Choose the leftmost NP Rule 2

Else If the rule contains a JJ: Choose the rightmost JJ Rule 3

Else If the rule contains a CD: Choose the rightmost CD Rule 4

Else Choose the rightmost child Rule 5

e.g.,

NP	⇒	DT	NNP	NN	Rule 1
NP	⇒	DT	NN	NNP	Rule 1
NP	⇒	NP	PP		Rule 2
NP	⇒	DT	JJ		Rule 3
NP	⇒	DT			Rule 5

Probabilistic Lexicalised CFG

Example by Michael Collins

If the rule contains Vi or Vt: Choose the leftmost Vi or Vt Rule 6

Else If the rule contains an VP: Choose the leftmost VP Rule 7

Else Choose the leftmost child

e.g.,

VP	⇒	Vt	NP	Rule 6
VP	⇒	VP	PP	Rule 7

Probabilistic Lexicalised CFG

Estimating the probabilities from treebank is similar:

- PCFG:

  $q(S \rightarrow NP\ VP)$

- LPCFG

  $q(S(saw) \rightarrow NP(man)\ VP(saw))$

Accuracy of PCFG on Penn Treebank = 72% →88%

3 Dependency Parsing

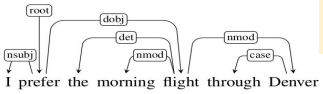
3.1 Dependency Parsing

In constituency parsing we group things, wheras now we try to see how the words modify or link together.

Dependency parsing

Figure from Jurafsky, D and Martin, J, "Speech and Language Processing," 2018, ch 15

- Connect words in sentence to indicate dependencies between them - much older linguistic theory
- Build around notion of having **heads** and **dependents**
- Arrows can be annotated by different types of dependencies
  - **Head** (governor), also called **argument**: origin
  - **Dependent**, also called **modifier**: destiny



From head to dependent

- the head is the arguemnt and depndent is the modifier
- Here, the 'morning' is modifying 'flight', and so is 'the'.

Dependency parsing

- There are versions without **typed dependencies**, just arcs

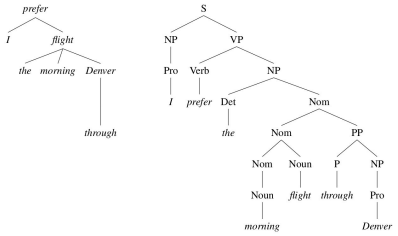
  - Untyped variant is simpler to build but less informative
- Can also represent as a tree

- There are versions without typed dependencies. Here, we see 'what is connected to what?'

Dependency parsing

Figure from Jurafsky, D and Martin, J, "Speech and Language Processing," 2018, ch 15

- Comparison to constituency parsing



- Here, 'prefer' is the root action that we're interested in and 'I' modifies that.
- The size is much smaller becuase the nodes are the words and we're tying to find the edges instead of inventing a new set of non-teriminals.

3.2 Differences when compared to Constituency Parsing

Dependency parsing

- Main **differences** to constituency parsing
  - No nodes corresponding to phrasal constituents or lexical categories
  - The internal structure consists only of directed relations between **pairs** of lexical items
  - These relationships allow directly encoding important information, e.g.:
    - The arguments of the verb *prefer* are directly linked to it in the dependency structure

Dependency parsing

- Main **advantages** in dependency parsing
  - Ability to deal with languages that are morphologically rich and have a relatively free word order. E.g. Czech location adverbs may occur before or after object:
    I caught the fish **here** vs I caught **here** the fish
  - Would have to represent two rules for each possible place of the adverb for constituency
  - Dependency approach: only one link; abstracts away from word order

Dependency parsing

- Main **advantages** in dependency parsing
  - Head-dependent relations provide approximation to semantic relationship between predicates and arguments
  - Can be directly used to solve problems such as co-reference resolution, question answering, etc.

3.3 Dependency Relations

Dependency relations (e.g. from universal dep.)

Figure from Jurafsky, D and Martin, J, "Speech and Language Processing," 2018, ch 15

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Dependency relations (e.g. from universal dep.)

Figure from Jurafsky, D and Martin, J, "Speech and Language Processing," 2018, ch 15

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Figure 9: what it may look like

3.4 Formally

Dependency formalisms - general case

- A dependency structure is a **directed graph** *G = (V, A)*
  - *V* = set of vertices (words, punctuation, ROOT)
  - *A* = set of ordered pairs of vertices (i.e. arcs)
- A dependency tree (directed graph):
  - Has a single ROOT node that has no incoming arcs
  - Each vertex has exactly one incoming arc (except for ROOT)
  - There's a unique path from ROOT to each vertex
  - There are no cycles A → B, B → A

Dependency formalisms - general case

- **This ensures the following properties**:
  - Dependency structure becomes a tree
  - Each word has a single head
  - The dependency tree is connected
  - There is a single ROOT from which a unique directed path follows to each word in sentence

3.5 Sources of information for Dependency Parsing

Dependency parsing -sources of info

- Distance between head and dependent
  - Mostly nearby words
- Intervening material
  - Dependencies don't cross over verbs or punctuation
- Valency of verbs
  - For a typical word, what kind of dependency it generally takes? E.g. A noun takes dependencies on the left (DET, JJ) but not on the right

•

Dependency parsing - two approaches

- Dynamic programming (cubic time, not very accurate)
- Shift-reduce (transition-based)
  - Predict from left-to-right
  - Fast (linear), but slightly less accurate
  - MaltParser
- Spanning tree (graph-based, constraint satisfaction)
  - Calculate full tree at once
  - Slightly more accurate, slower
  - MSTParser

•

3.6 Transition Based — Shift-reduce Parsing

Dependency parsing - transition-based

- Deterministic parsing, shift-reduce (MALT parser)
  - Greedy choice of attachment for each word in order, guided by ML classifier
  - Works very well in practice
  - **Linear time parsing!**

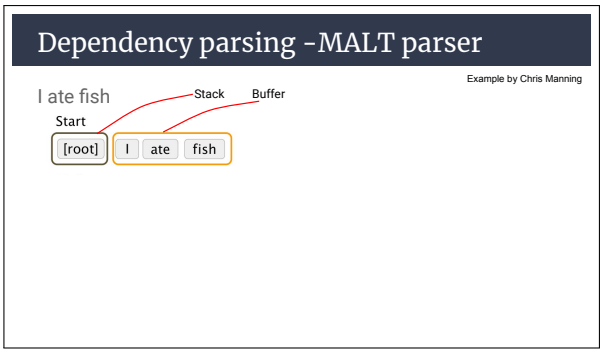
(c) greedy → linear time

Dependency parsing -MALT parser

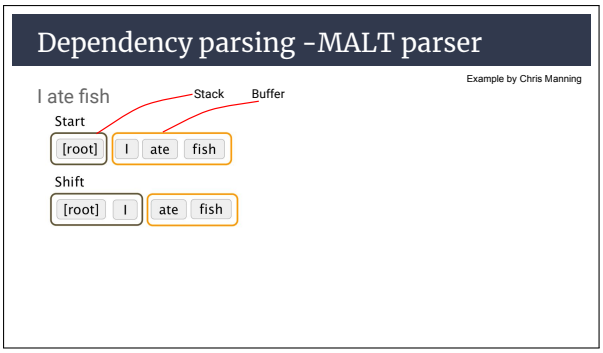
- Reads sentence word by word, left to right
- Greedy decision as to how to attach each word as it is read
- Sequence of actions bottom-up
- Formally, 3 data structures:
  - σ = **Stack**, which starts with ROOT
  - β = **Buffer**, which starts with all words in sentence
  - **A** = **Set of arcs**, which starts empty
- Set of actions
  - **Shift** / **left arc** / **right arc**
  - Optionally, set of **dep. labels** for left and right arc actions (~40)

(d) three structures and actions create a depedency tree

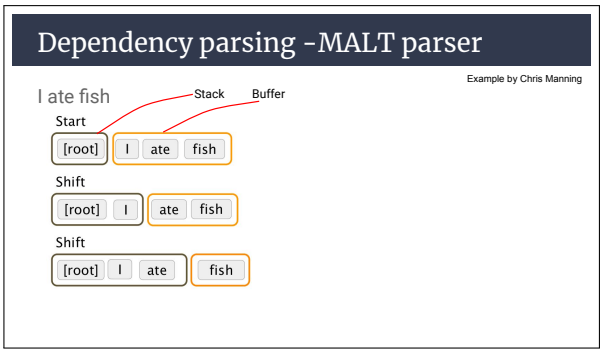
3.6.1 Dependency MALT Parser



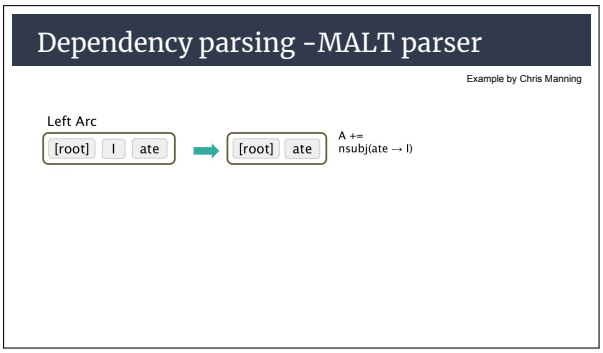
(e) we parse the sentence 'I ate fish'



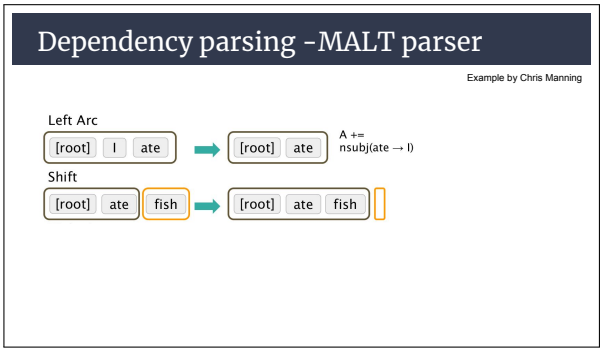
(f) an ML classifier might say 'shift'



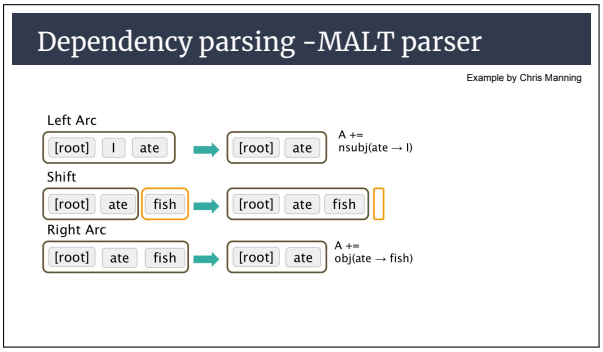
(g) again



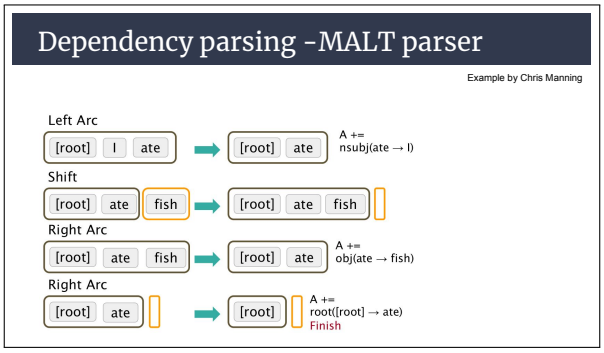
(h) create a left arc



(i) shift fish



(j) right arc and say fish is modifying ate with the object modifier label



(k) right arc and finish

Figure 10: Example

Dependency parsing -MALT parser

Pseudocode by Chris Manning

Start: $\sigma = [ROOT], \beta = w_1, \ldots, w_n, A = \emptyset$

1. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j, w_i)\}$
3. Right-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i, w_j)\}$

Finish: $\sigma = [w], \beta = \emptyset$

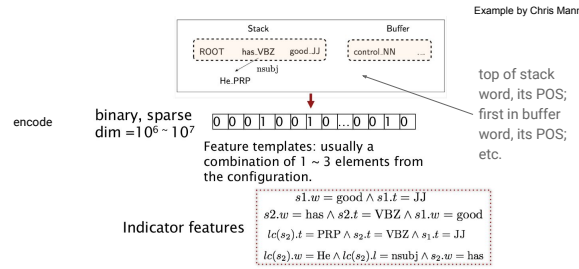
Dependency parsing -MALT parser

- How do we make the shift/reduce left/right decisions?
  - ML classifier
  - Each action is predicted by a **discriminative classifier** over each move
  - 3 classes for untyped dependencies: shift, left or right
  - 2*categories + 1 for typed dependencies
  - **Features**: top of stack word, its POS; first in buffer word, its POS; etc.
  - No beam-search in original version

- we just have an ML classifier with 3 classes.
- we can use as many features as we like

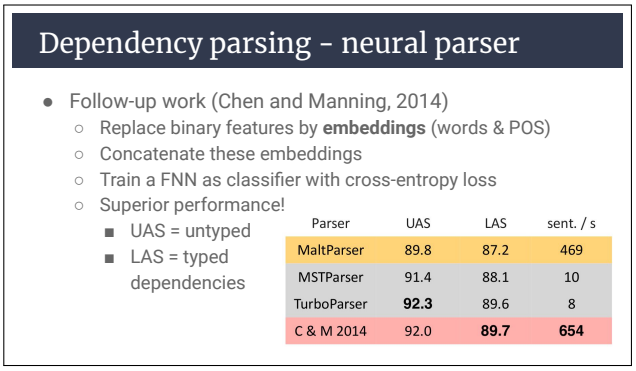
Dependency parsing -MALT parser

Example by Chris Manning

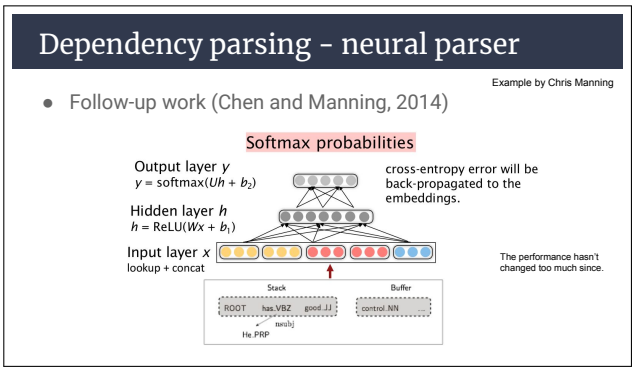


- take the current state of the parser, stack and buffer and one-hot encoded everything and encoded it.
- then pass the featuer vector onto a classifier
- it will say shift, left arc or right arc

3.6.2 Dependency neural parser

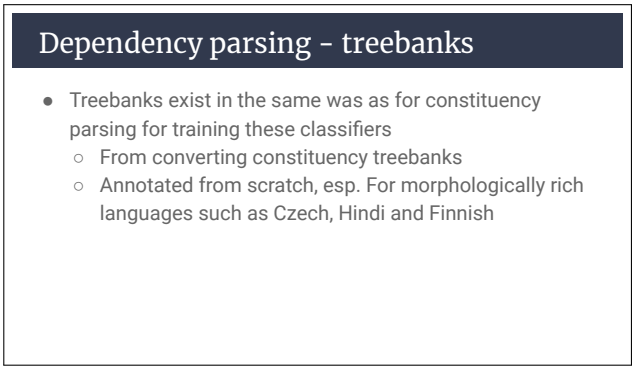


- we can improve on this, instead of using binary features and use embeddings.
- for ecah word you create an embedding, you concatenate it and you train an FNN



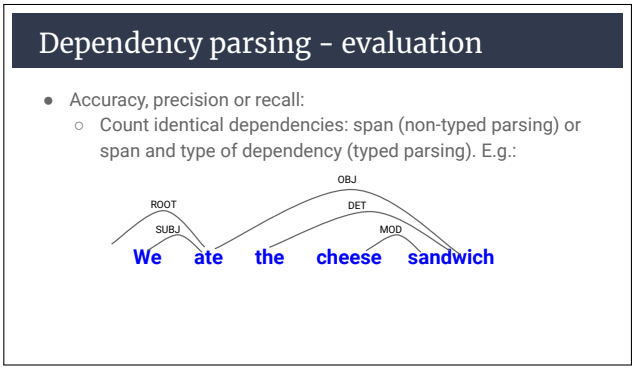
- example

3.6.3 Dependency Parsing TreeBank



- valuable resource

3.6.4 Dependency Parsing Evaluation



- count the identical dependencies

Dependency parsing - evaluation

- Accuracy, precision or recall:
  - Count identical dependencies: span (non-typed parsing) or span and type of dependency (typed parsing). E.g.:

Reference		Hypothesis	
(1,2) We	SUBJ	(1,2) We	SUBJ
(2,0) eat	ROOT	(2,0) eat	ROOT
(3,5) the	DET	(3,4) the	DET
(4,5) cheese	MOD	(4,2) cheese	OBJ
(5,2) sandwich	OBJ	(5,2) sandwich	PRED

- compare the reference and hypothesis.

Dependency parsing - evaluation

- Accuracy, precision or recall:
  - Count identical dependencies: span (non-typed parsing) or span and type of dependency (typed parsing). E.g.:

Reference		Hypothesis	
(1,2) We	SUBJ	(1,2) We	SUBJ
(2,0) eat	ROOT	(2,0) eat	ROOT
(3,5) the	DET	(3,4) the	DET
(4,5) cheese	MOD	(4,2) cheese	OBJ
(5,2) sandwich	OBJ	(5,2) sandwich	PRED

Accuracy = ⅖ = 40%

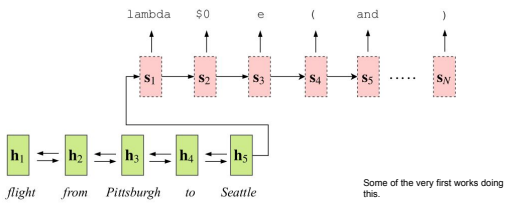
- only 2/5 match

3.7 Neural Parsing

3.7.1 simple approach

Neural parsing - simple approach

(Jia and Liang, 2016; Dong and Lapata, 2016)



Some of the very first works doing this.

- fianlly, people tried to flatten the dependency tree by passing it into the rnn and output something that is separated by brackets which is the flattened tree

Neural parsing - simple approach

- Parsing as translation
  - Linearise grammar from treebank: convert tree to bracketed representation, all in one line
  - Extract sentences from bracketed representation
  - Pair sentences and their linearised trees
  - No need to compute/represent probabilities - **learning**

Neural parsing - simple approach

- Train sequence to sequence model to *translate* from
  - Sentences to linearised tree; **brackets are tokens**!
  - Use, e.g. LSTM, transformers etc...
  - Attention helps
  - Train with cross-entropy
  - Evaluate as a translation (BLEU) or parsing tasks (parseval)

3.7.2 advanced approach

Neural parsing – advanced approaches

Table by Chris Manning

- Graph-based methods

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**

- UAS: Unlabeled attachment score
- LAS: Labeled attachment score

Not much more progress from there

4 Conclusion

Discussion

- Parsing is an important step for many applications
- Statistical models such as PCFGs allow for resolution of ambiguities
  - Lexicalisation and non-terminal splitting are required to effectively better resolve many ambiguities
- Current statistical/neural parsers are quite accurate
  - ~95% dependency; 97% constituency
  - Human-expert agreement: ~98%