Algorithms for NLP

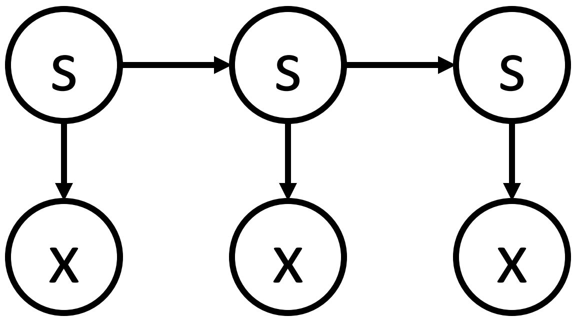
POS Tagging / Parsing I

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley

Speech Training

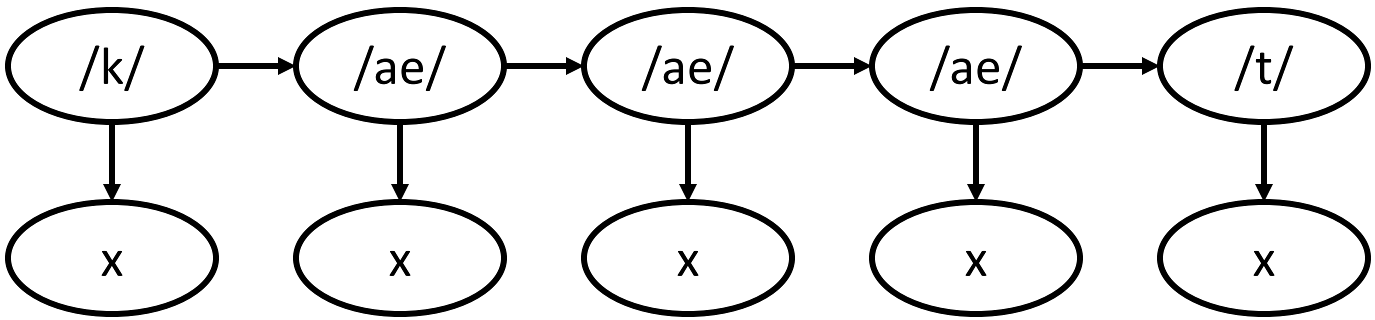
What Needs to be Learned?



- Emissions: P(x | phone class)
  - X is MFCC-valued
- Transitions: P(state | prev state)
  - If between words, this is P(word | history)
  - If inside words, this is P(advance | phone class)
  - (Really a hierarchical model)

Estimation from Aligned Data

- What if each time step was labeled with its (context-dependent sub) phone?



- Can estimate P(x|/ae/) as empirical mean and (co-)variance of x's with label /ae/

- Problem: Don't know alignment at the frame and phone level

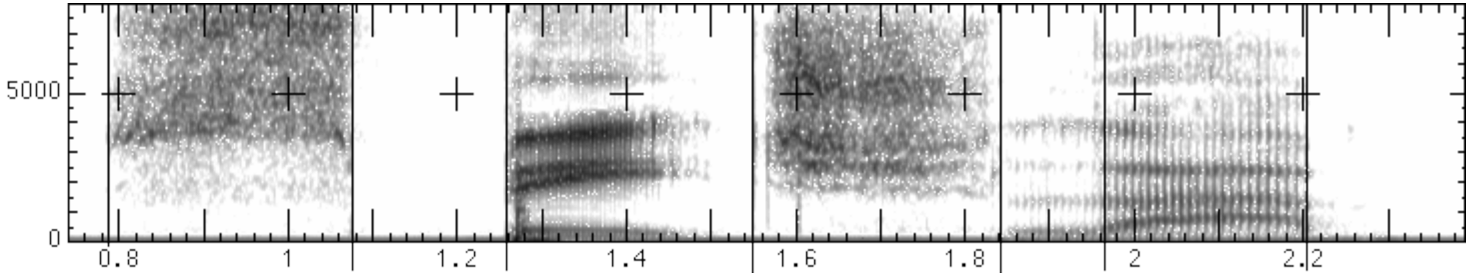
Forced Alignment

- What if the acoustic model P(x|phone) was known?
  - … and also the correct sequences of words / phones

- Can predict the best alignment of frames to phones

"speech lab"

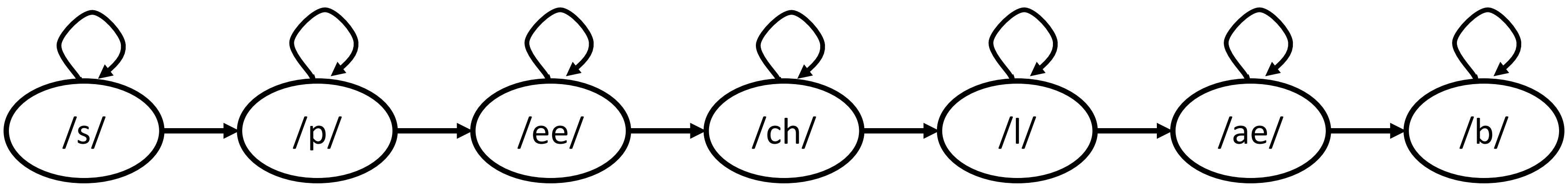
ssssssssppppeeeeeeetshshshshllllaeaeaebbbbb



- Called "forced alignment"

Forced Alignment

- Create a new state space that forces the hidden variables to transition through phones in the (known) order



- Still have uncertainty about durations

- In this HMM, all the parameters are known
  - Transitions determined by known utterance
  - Emissions assumed to be known
  - Minor detail: self-loop probabilities

- Just run Viterbi (or approximations) to get the best alignment

EM for Alignment

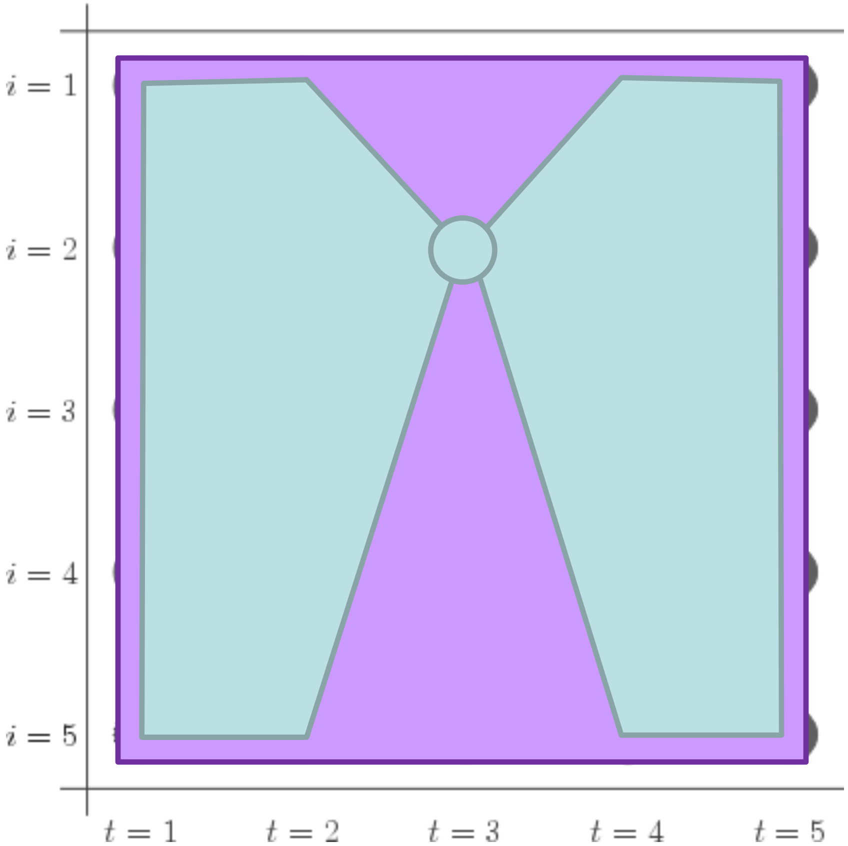
- Input: acoustic sequences with word-level transcriptions
- We don’t know either the emission model or the frame alignments
- Expectation Maximization (Hard EM for now)
  - Alternating optimization
  - Impute completions for unlabeled variables (here, the states at each time step)
  - Re-estimate model parameters (here, Gaussian means, variances, mixture ids)
  - Repeat
  - One of the earliest uses of EM!

Soft EM

- Hard EM uses the best single completion
  - Here, single best alignment
  - Not always representative
  - Certainly bad when your parameters are initialized and the alignments are all tied
  - Uses the count of various configurations (e.g. how many tokens of /ae/ have self-loops)

- What we'd really like is to know the fraction of paths that include a given completion
  - E.g. 0.32 of the paths align this frame to /p/, 0.21 align it to /ee/, etc.
  - Formally want to know the expected count of configurations
  - Key quantity: $P(s_t \mid x)$

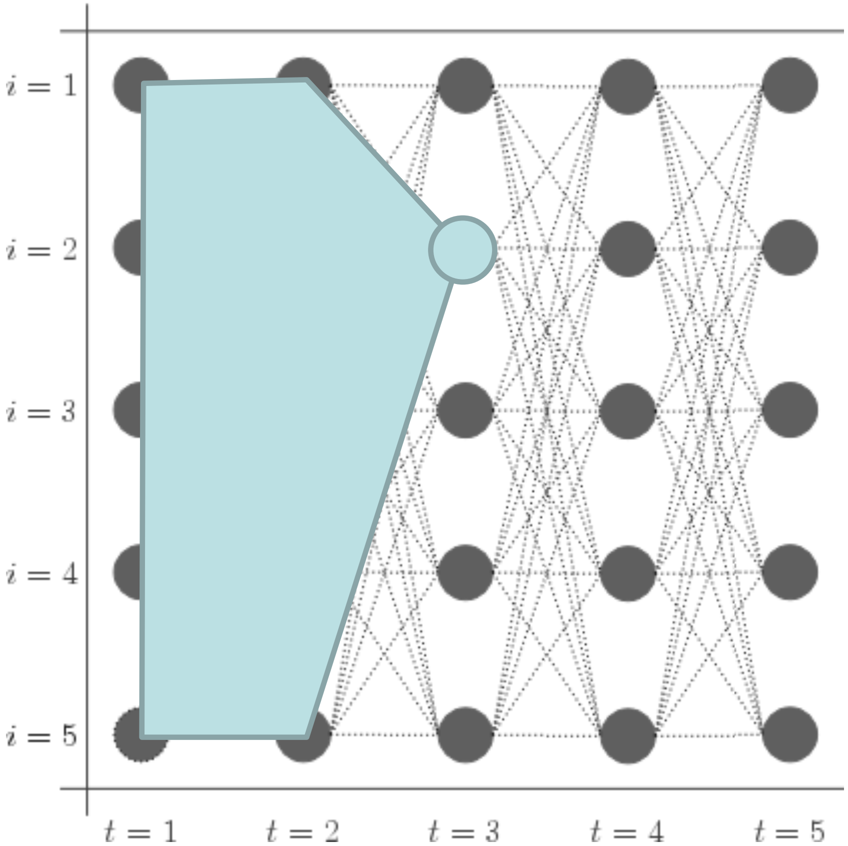
Computing Marginals



$$P(s_t|x) = \frac{P(s_t, x)}{P(x)}$$

$$= \frac{\text{sum of all paths through s at t}}{\text{sum of all paths}}$$

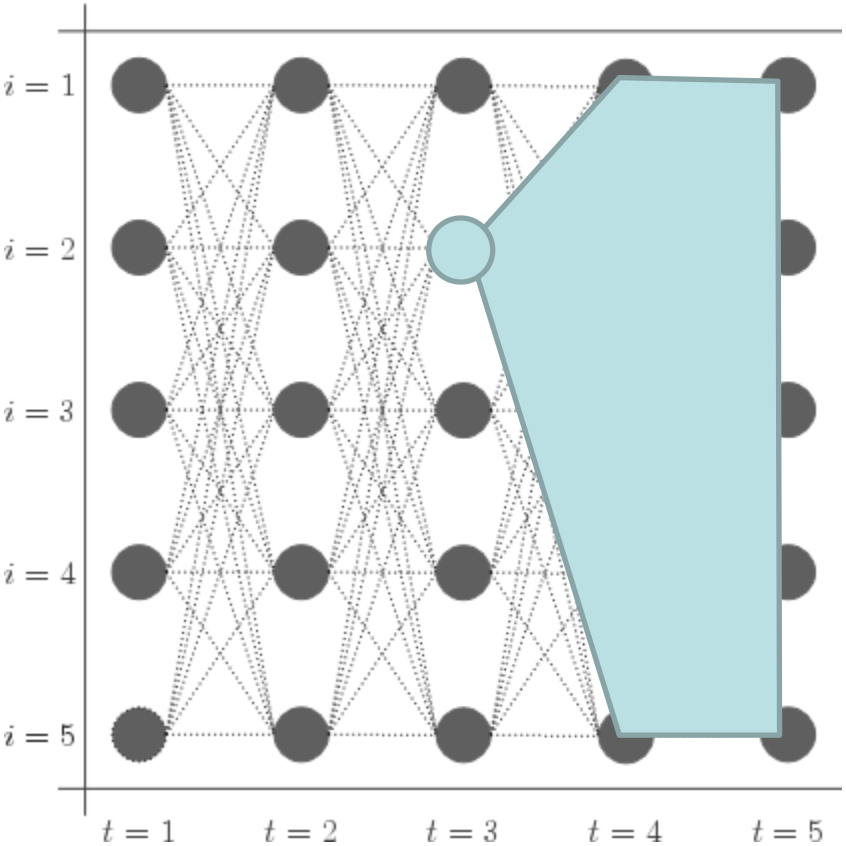
Forward Scores



$$v_t(s_t) = \max_{s_{t-1}} v_{t-1}(s_{t-1})\phi_t(s_{t-1}, s_t)$$

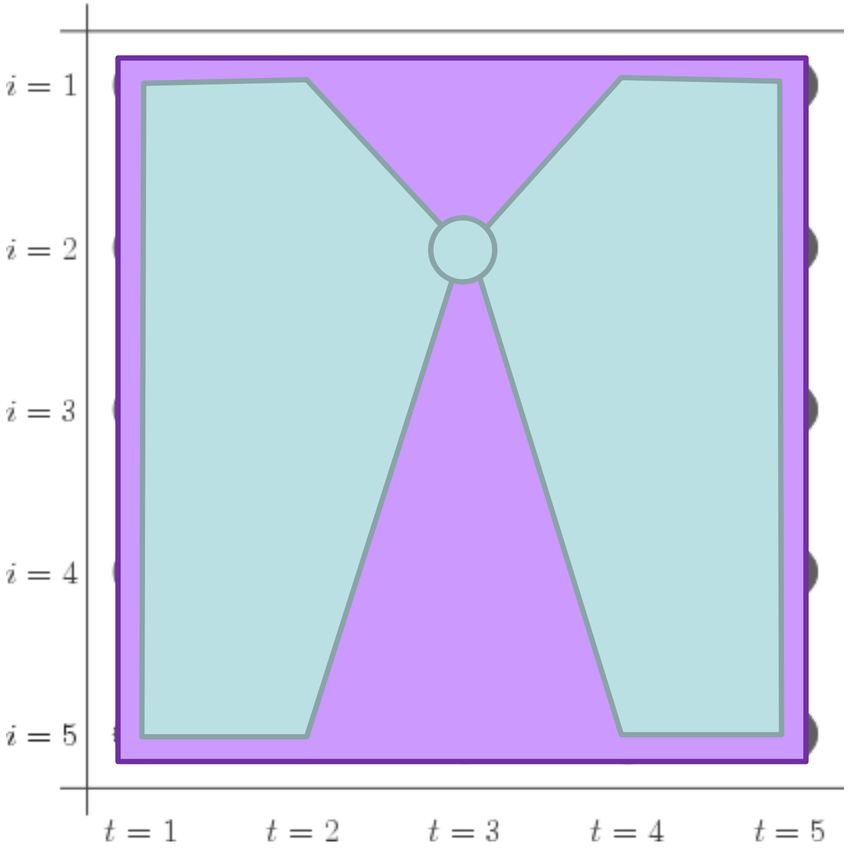
$$\alpha_t(s_t) = \sum_{s_{t-1}} \alpha_{t-1}(s_{t-1})\phi_t(s_{t-1}, s_t)$$

Backward Scores



$$\beta_t(s_t) = \sum_{s_{t+1}} \beta_{t+1}(s_{t+1})\phi_t(s_t, s_{t+1})$$

Total Scores



$$P(s_t, x) = \alpha_t(s_t)\beta_t(s_t)$$

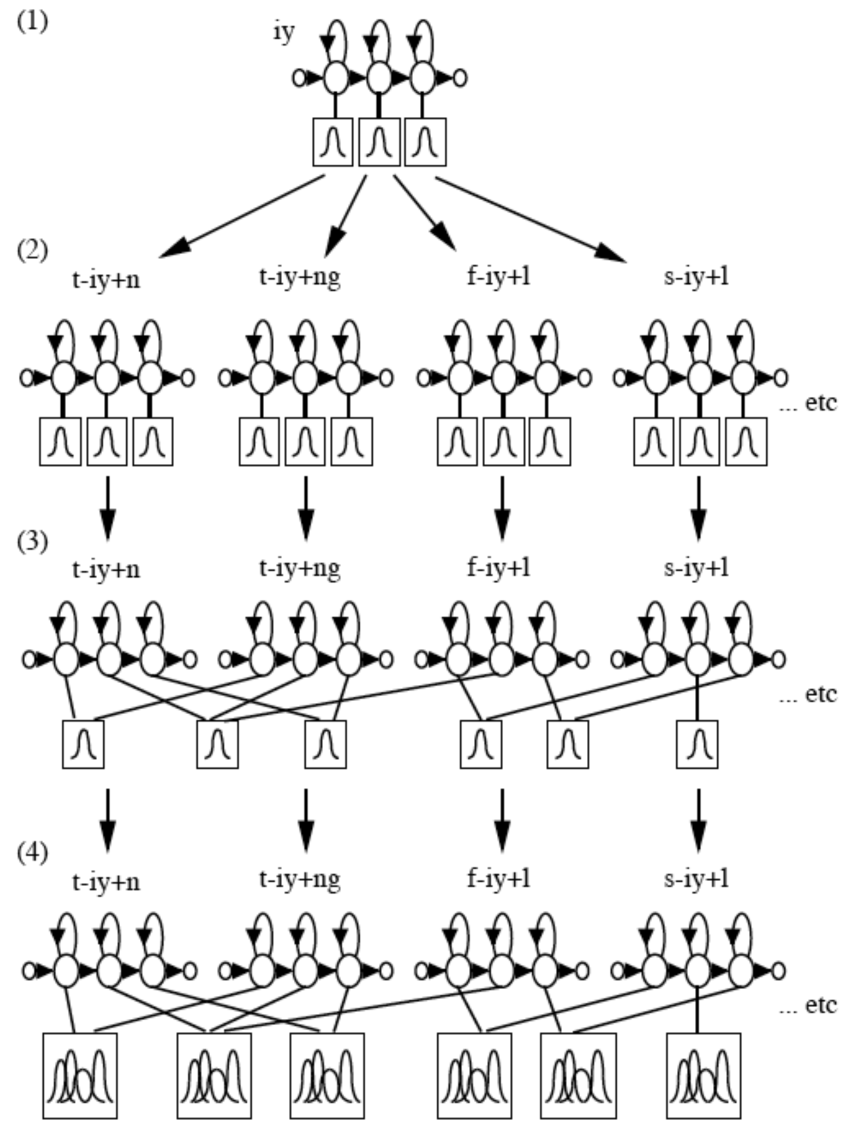
$$P(x) = \sum_{s_t} \alpha_t(s_t)\beta_t(s_t)$$
$$= \alpha_T(\text{stop})$$
$$= \beta_0(\text{start})$$

Fractional Counts

- Computing fractional (expected) counts
  - Compute forward / backward probabilities
  - For each position, compute marginal posteriors
  - Accumulate expectations
  - Re-estimate parameters (e.g. means, variances, self-loop probabilities) from ratios of these expected counts

Staged Training and State Tying

- Creating CD phones:
  - Start with monophone, do EM training
  - Clone Gaussians into triphones
  - Build decision tree and cluster Gaussians
  - Clone and train mixtures (GMMs)
- General idea:
  - Introduce complexity gradually
  - Interleave constraint with flexibility



Parts of Speech

Parts-of-Speech (English)

- One basic kind of linguistic structure: syntactic word classes

Open class (lexical) words

- Nouns
  - Proper: *IBM*, *Italy*
  - Common: *cat / cats*, *snow*
- Verbs
  - Main: *see*, *registered*
- Adjectives: *yellow*
- Adverbs: *slowly*
- Numbers: *122,312*, *one*
- … more

Closed class (functional)

- Determiners: *the some*
- Conjunctions: *and or*
- Pronouns: *he its*
- Verbs
  - Auxiliary: *can*, *had*
- Prepositions: *to with*
- Particles: *off up*
- … more

CC	conjunction, coordinating	and both but either or
CD	numeral, cardinal	mid-1890 nine-thirty 0.5 one
DT	determiner	a all an every no that the
EX	existential there	there
FW	foreign word	gemeinschaft hund ich jeux
IN	preposition or conjunction, subordinating	among whether out on by if
JJ	adjective or numeral, ordinal	third ill-mannered regrettable
JJR	adjective, comparative	braver cheaper taller
JJS	adjective, superlative	bravest cheapest tallest
MD	modal auxiliary	can may might will would
NN	noun, common, singular or mass	cabbage thermostat investment subhumanity
NNP	noun, proper, singular	Motown Cougar Yvette Liverpool
NNPS	noun, proper, plural	Americans Materials States
NNS	noun, common, plural	undergraduates bric-a-brac averages
POS	genitive marker	' 's
PRP	pronoun, personal	hers himself it we them
PRP$	pronoun, possessive	her his mine my our ours their thy your
RB	adverb	occasionally maddeningly adventurously
RBR	adverb, comparative	further gloomier heavier less-perfectly
RBS	adverb, superlative	best biggest nearest worst
RP	particle	aboard away back by on open through
TO	"to" as preposition or infinitive marker	to
UH	interjection	huh howdy uh whammo shucks heck
VB	verb, base form	ask bring fire see take
VBD	verb, past tense	pleaded swiped registered saw
VBG	verb, present participle or gerund	stirring focusing approaching erasing
VBN	verb, past participle	dilapidated imitated reunifed unsettled
VBP	verb, present tense, not 3rd person singular	twist appear comprise mold postpone
VBZ	verb, present tense, 3rd person singular	bases reconstructs marks uses
WDT	WH-determiner	that what whatever which whichever
WP	WH-pronoun	that what whatever which who whom
WP$	WH-pronoun, possessive	whose
WRB	Wh-adverb	however whenever where why

Part-of-Speech Ambiguity

- Words can have multiple parts of speech

VBD		VB			
VBN	VBZ	VBP	VBZ		
NNP	NNS	NN	NNS	CD	NN
Fed	raises	interest	rates	0.5	percent

Mrs./NNP Shaefer/NNP never/RB got/VBD **around/RP** to/TO joining/VBG

All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB **around/IN** the/DT corner/NN

Chateau/NNP Petrus/NNP costs/VBZ **around/RB** 250/CD

- Two basic sources of constraint:
  - Grammatical environment
  - Identity of the current word
- Many more possible features:
  - Suffixes, capitalization, name databases (gazetteers), etc…

Why POS Tagging?

- Useful in and of itself (more than you'd think)
  - Text-to-speech: record, lead
  - Lemmatization: saw[v] $\rightarrow$ see, saw[n] $\rightarrow$ saw
  - Quick-and-dirty NP-chunk detection: grep {JJ | NN}* {NN | NNS}

- Useful as a pre-processing step for parsing
  - Less tag ambiguity means fewer parses
  - However, some tag choices are better decided by parsers

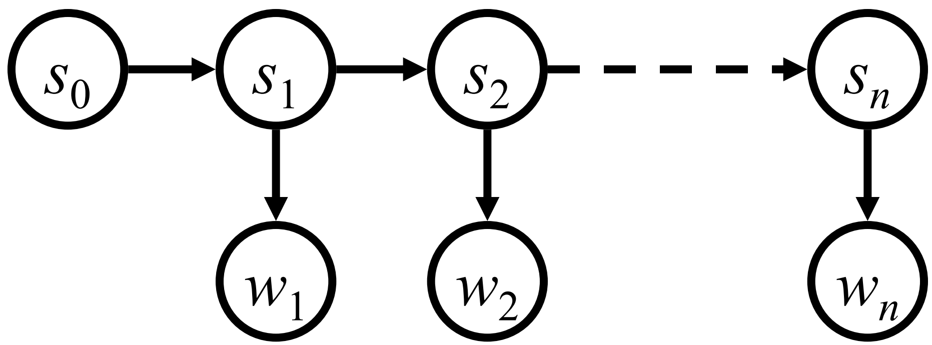
					IN		
DT	NNP	NN	VBD	VBN	RP	NN	NNS
The	Georgia	branch	had	taken	on	loan	commitments …

				VDN		
DT	NN	IN	NN	VBD	NNS	VBD
The	average	of	interbank	offered	rates	plummeted …

Part-of-Speech Tagging

Classic Solution: HMMs

- We want a model of sequences s and observations w

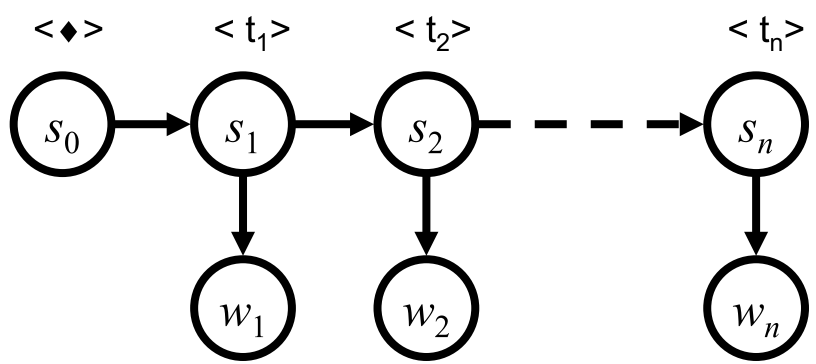


$$P(\mathbf{s}, \mathbf{w}) = \prod_i P(s_i|s_{i-1})P(w_i|s_i)$$

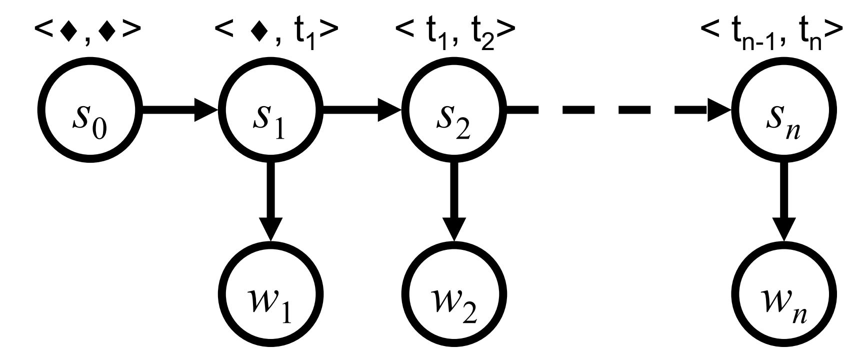
- Assumptions:
  - States are tag n-grams
  - Usually a dedicated start and end state / word
  - Tag/state sequence is generated by a markov model
  - Words are chosen independently, conditioned only on the tag/state
  - These are totally broken assumptions: why?

States

- States encode what is relevant about the past
- Transitions P(s|s') encode well-formed tag sequences
  - In a bigram tagger, states = tags



  - In a trigram tagger, states = tag pairs



Estimating Transitions

- Use standard smoothing methods to estimate transitions:

$$P(t_i \mid t_{i-1}, t_{i-2}) = \lambda_2 \hat{P}(t_i \mid t_{i-1}, t_{i-2}) + \lambda_1 \hat{P}(t_i \mid t_{i-1}) + (1 - \lambda_1 - \lambda_2)\hat{P}(t_i)$$

- Can get a lot fancier (e.g. KN smoothing) or use higher orders, but in this case it doesn't buy much

- One option: encode more into the state, e.g. whether the previous word was capitalized (Brants 00)

- BIG IDEA: The basic approach of state-splitting / refinement turns out to be very important in a range of tasks

Estimating Emissions

$$P(\mathbf{s}, \mathbf{w}) = \prod_i P(s_i | s_{i-1}) P(w_i | s_i)$$

- Emissions are trickier:
  - Words we've never seen before
  - Words which occur with tags we've never seen them with
  - One option: break out the fancy smoothing (e.g. KN, Good-Turing)
  - Issue: unknown words aren't black boxes:

    343,127.23 11-year Minteria reintroducibly

  - Basic solution: unknown words classes (affixes or shapes)

    $D^+,D^+.D^+$ D^+-x^+ Xx^+ x^+-"ly"

  - Common approach: Estimate P(t|w) and invert
  - [Brants 00] used a suffix trie as its (inverted) emission model

Disambiguation (Inference)

- Problem: find the most likely (Viterbi) sequence under the model

$$\mathbf{t}^* = \arg\max_{\mathbf{t}} \; P(\mathbf{t}|\mathbf{w})$$

- Given model parameters, we can score any tag sequence

<♦,♦>	<♦,NNP>	<NNP, VBZ>	<VBZ, NN>	<NN, NNS>	<NNS, CD>	<CD, NN>	<STOP>
	NNP	VBZ	NN	NNS	CD	NN	.
	Fed	raises	interest	rates	0.5	percent	.

$P(NNP|<♦,♦>)\ P(Fed|NNP)\ P(VBZ|<NNP,♦>)\ P(raises|VBZ)\ P(NN|VBZ,NNP)\ldots\ldots$

- In principle, we're done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence)

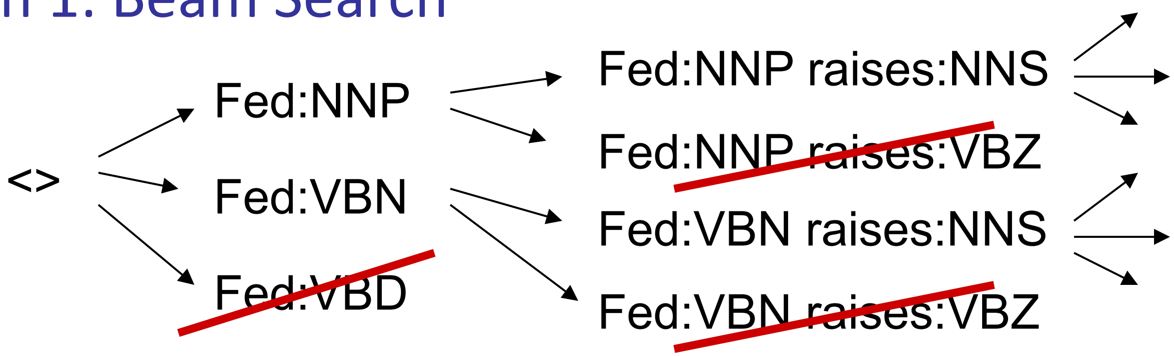
NNP VBZ NN NNS CD NN $\Rightarrow$ logP = -23

NNP NNS NN NNS CD NN $\Rightarrow$ logP = -29

NNP VBZ VB NNS CD NN $\Rightarrow$ logP = -27

Finding the Best Trajectory

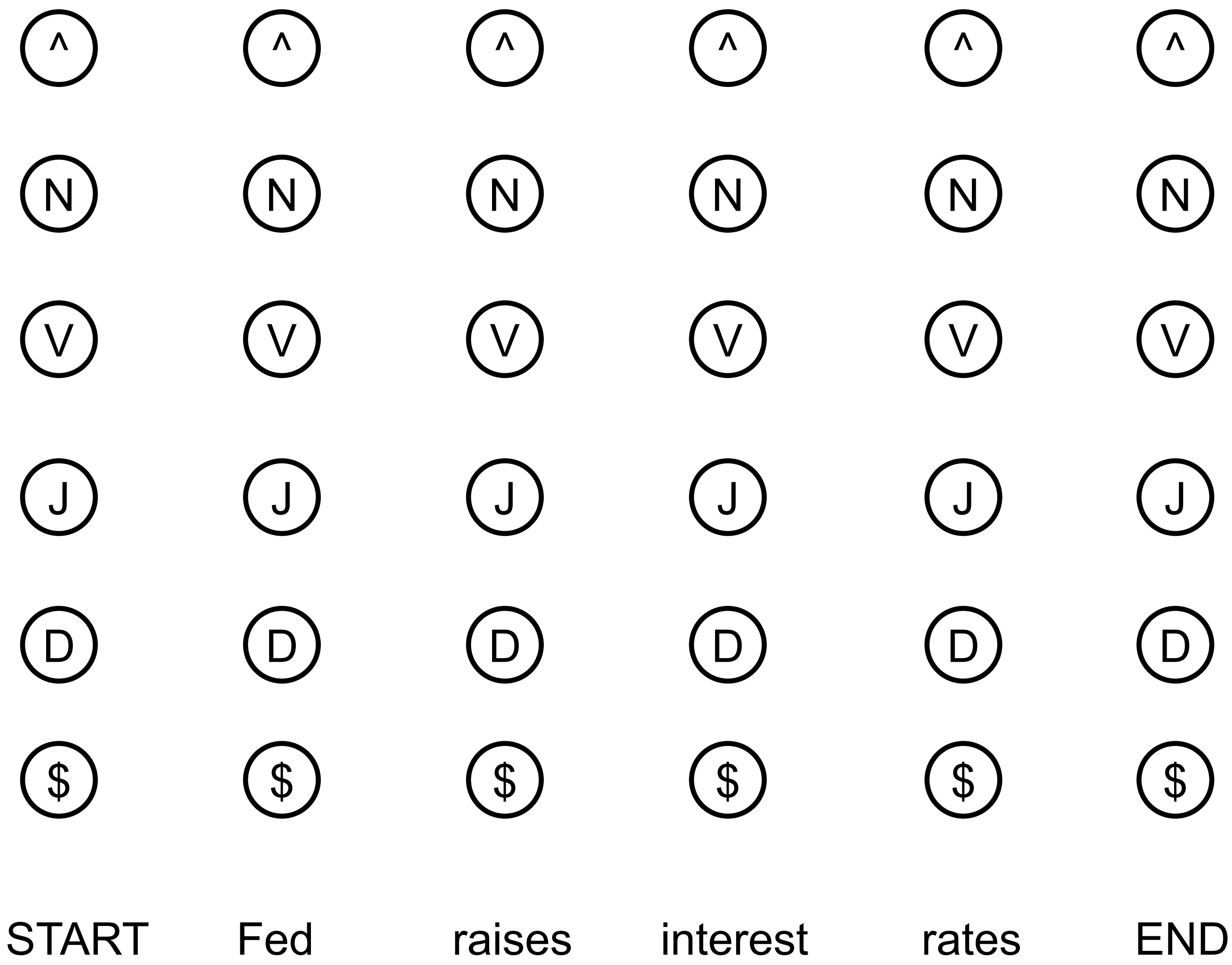
- Too many trajectories (state sequences) to list
- Option 1: Beam Search



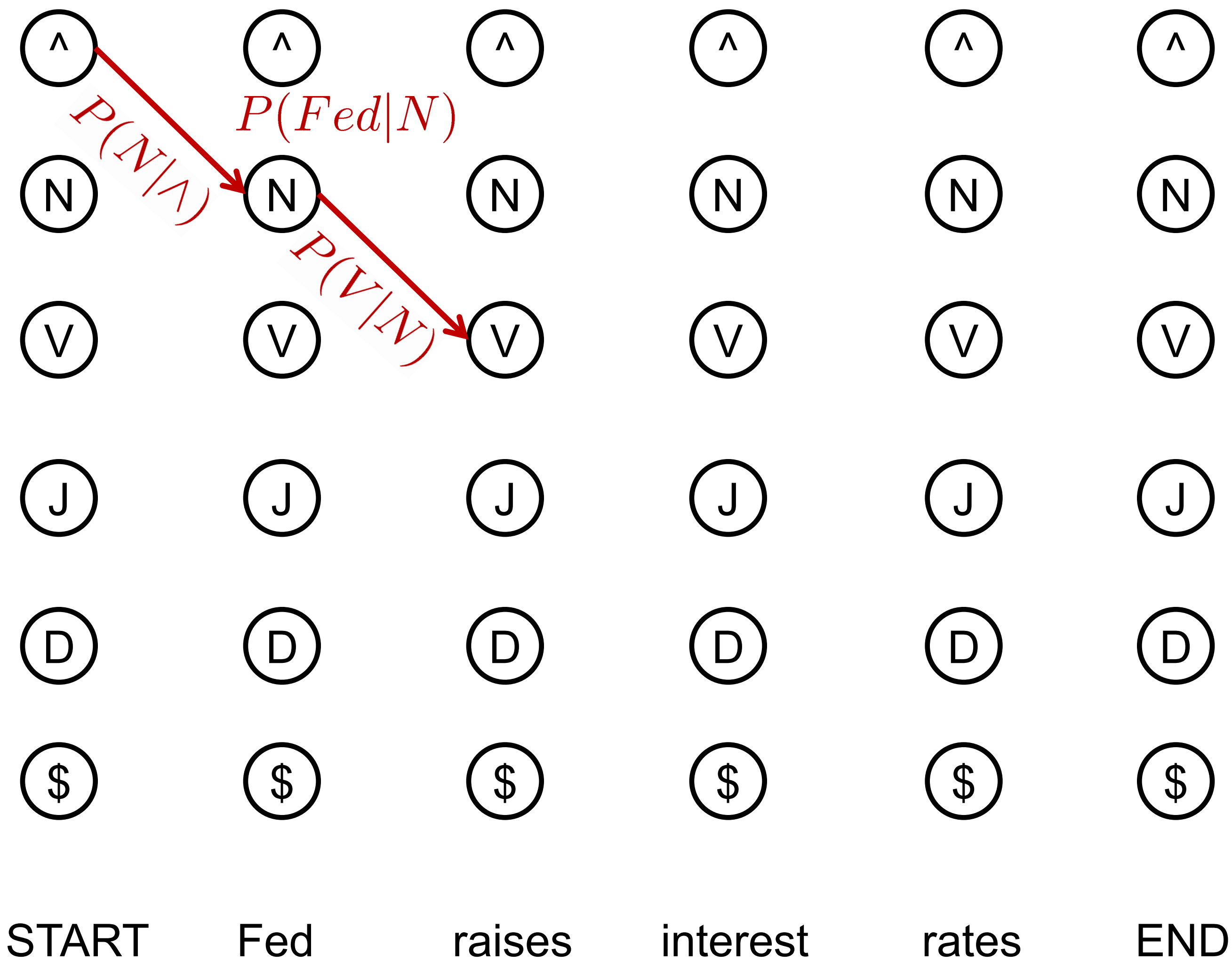
  - A beam is a set of partial hypotheses
  - Start with just the single empty trajectory
  - At each derivation step:
    - Consider all continuations of previous hypotheses
    - Discard most, keep top k, or those within a factor of the best

- Beam search works ok in practice
  - … but sometimes you want the optimal answer
  - … and you need optimal answers to validate your beam search
  - … and there’s usually a better option than naïve beams

The State Lattice / Trellis



The State Lattice / Trellis



The Viterbi Algorithm

- Dynamic program for computing

$$\delta_i(s) = \max_{s_0 \ldots s_{i-1} s} P(s_0 \ldots s_{i-1} s, w_1 \ldots w_{i-1})$$

  - The score of a best path up to position i ending in state s

$$\delta_0(s) = \begin{cases} 1 & if\ s = <\bullet,\bullet> \\ 0 & otherwise \end{cases}$$

$$\delta_i(s) = \max_{s'} P(s \mid s') P(w \mid s') \delta_{i-1}(s')$$

  - Also can store a backtrace (but no one does)

$$\psi_i(s) = \arg\max_{s'} P(s \mid s') P(w \mid s') \delta_{i-1}(s')$$

- Memoized solution
- Iterative solution

So How Well Does It Work?

- Choose the most common tag
  - 90.3% with a bad unknown word model
  - 93.7% with a good one

- TnT (Brants, 2000):
  - A carefully smoothed trigram tagger
  - Suffix trees for emissions
  - 96.7% on WSJ text (SOTA is 97+%)

- Noise in the data
  - Many errors in the training and test corpora

DT	NN	IN	NN	VBD	NNS	VBD	
The	average	of	interbank	offered	rates	plummeted	…

  - Probably about 2% guaranteed error from noise (on this data)

JJ	JJ	NN
chief	executive	officer
NN	JJ	NN
chief	executive	officer
JJ	NN	NN
chief	executive	officer
NN	NN	NN
chief	executive	officer

Overview: Accuracies

- Roadmap of (known / unknown) accuracies:
  - Most freq tag: ~90% / ~50%
  - Trigram HMM: ~95% / ~55%
  - TnT (HMM++): 96.2% / 86.0%
  - Maxent P(t|w): 93.7% / 82.6%
  - MEMM tagger: 96.9% / 86.9%
  - State-of-the-art: 97+% / 89+%
  - Upper bound: ~98%

Most errors on unknown words

Common Errors

- Common errors [from Toutanova & Manning 00]

	JJ	NN	NNP	NNPS	RB	RP	IN	VB	VBD	VBN	VBP	Total
JJ	0	**177**	**56**	0	**61**	2	5	10	15	**108**	0	488
NN	**244**	0	**103**	0	12	1	1	29	5	6	19	525
NNP	**107**	**106**	0	**132**	5	0	7	5	1	2	0	427
NNPS	1	0	**110**	0	0	0	0	0	0	0	0	142
RB	**72**	21	7	0	0	16	**138**	1	0	0	0	295
RP	0	0	0	0	**39**	0	**65**	0	0	0	0	104
IN	11	0	1	0	**169**	**103**	0	1	0	0	0	323
VB	17	**64**	9	0	2	0	1	0	4	7	**85**	189
VBD	10	5	3	0	0	0	0	3	0	**143**	2	166
VBN	**101**	3	3	0	0	0	0	3	**108**	0	1	221
VBP	5	34	3	1	1	0	2	**49**	6	3	0	104
Total	626	536	348	144	317	122	279	102	140	269	108	3651

NN/JJ NN
official knowledge

VBD RP/IN DT NN
made up the story

RB VBD/VBN NNS
recently sold shares

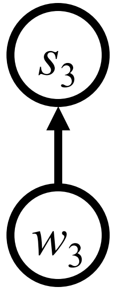
Richer Features

Better Features

- Can do surprisingly well just looking at a word by itself:

▪ Word	the: the $\rightarrow$ DT
▪ Lowercased word	Importantly: importantly $\rightarrow$ RB
▪ Prefixes	unfathomable: un- $\rightarrow$ JJ
▪ Suffixes	Surprisingly: -ly $\rightarrow$ RB
▪ Capitalization	Meridian: CAP $\rightarrow$ NNP
▪ Word shapes	35-year: d-x $\rightarrow$ JJ

- Then build a maxent (or whatever) model to predict tag
- Maxent P(t|w): 93.7% / 82.6%



Why Linear Context is Useful

- Lots of rich local information!

  RB
PRP VBD IN RB IN PRP VBD .
They left as soon as he arrived .

  - We could fix this with a feature that looked at the next word

  JJ
NNP NNS VBD VBN .
Intrinsic flaws remained undetected .

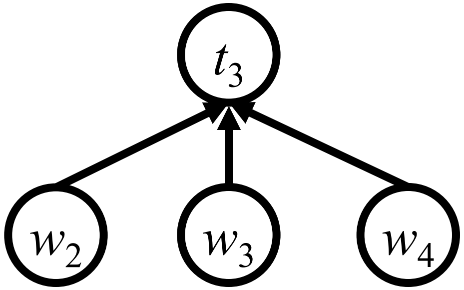
  - We could fix this by linking capitalized words to their lowercase versions

- Solution: discriminative sequence models (MEMMs, CRFs)

- Reality check:
  - Taggers are already pretty good on newswire text…
  - What the world needs is taggers that work on other text!

Sequence-Free Tagging?

- What about looking at a word and its environment, but no sequence information?



  - Add in previous / next word the __
  - Previous / next word shapes X __ X
  - Occurrence pattern features [X: x X occurs]
  - Crude entity detection __ ….. (Inc.|Co.)
  - Phrasal verb in sentence? put …… __
  - Conjunctions of these things

- All features except sequence: 96.6% / 86.8%
- Uses lots of features: > 200K
- Why isn't this the standard approach?

Named Entity Recognition

- Other sequence tasks use similar models

- Example: name entity recognition (NER)

PER PER O O O O O O ORG O O O O O LOC LOC O

Tim Boon has signed a contract extension with Leicestershire which will keep him at Grace Road .

Local Context

	Prev	Cur	Next
State	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

MEMM Taggers

- Idea: left-to-right local decisions, condition on previous tags and also entire input

$$P(\mathbf{t}|\mathbf{w}) = \prod_i P_{\mathsf{ME}}(t_i|\mathbf{w}, t_{i-1}, t_{i-2})$$

  - Train up $P(t_i|w,t_{i-1},t_{i-2})$ as a normal maxent model, then use to score sequences
  - This is referred to as an MEMM tagger [Ratnaparkhi 96]
  - Beam search effective! (Why?)
  - What about beam size 1?

- Subtle issues with local normalization (cf. Lafferty et al 01)

NER Features

Because of regularization term, the more common prefixes have larger weights even though entire-word features are more specific.

Local Context

	Prev	Cur	Next
State	Other	???	???
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

Feature Weights

Feature Type	**Feature**	**PERS**	**LOC**
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*<G*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Previous state	Other	-0.70	-0.92
Current signature	Xx	0.80	0.46
Prev state, cur sig	O-Xx	0.68	0.37
Prev-cur-next sig	x-Xx-Xx	-0.69	0.37
P. state - p-cur sig	O-x-Xx	-0.20	0.82
…			
Total:		**-0.58**	**2.68**

Decoding

- Decoding MEMM taggers:
  - Just like decoding HMMs, different local scores
  - Viterbi, beam search, posterior decoding
- Viterbi algorithm (HMMs):

$$\delta_i(s) = \arg\max_{s'} P(s|s')P(w_{i-1}|s')\delta_{i-1}(s')$$

- Viterbi algorithm (MEMMs):

$$\delta_i(s) = \arg\max_{s'} P(s|s', \mathbf{w})\delta_{i-1}(s')$$

- General:

$$\delta_i(s) = \arg\max_{s'} \phi_i(s', s)\delta_{i-1}(s')$$

Conditional Random Fields
(and Friends)

Maximum Entropy II

- Remember: maximum entropy objective

$$L(\mathbf{w}) = \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

- Problem: lots of features allow perfect fit to training set
- Regularization (compare to smoothing)

$$\max_{\mathbf{w}} \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right) - k||\mathbf{w}||^2$$

Derivative for Maximum Entropy

$$L(\mathbf{w}) = -k||\mathbf{w}||^2 + \sum_i \left(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y}^i) - \log \sum_{\mathbf{y}} \exp(\mathbf{w}^\top \mathbf{f}_i(\mathbf{y})) \right)$$

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}_n} = -2k\mathbf{w}_n + \sum_i \left(\mathbf{f}_i(\mathbf{y}^i)_n - \sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}_i)\mathbf{f}_i(\mathbf{y})_n \right)$$

Big weights are bad

Expected count of feature n in predicted candidates

Total count of feature n in correct candidates

Perceptron Review

[Collins 01]

Perceptron

- Linear model:

$$\text{score}(\mathbf{t}|\mathbf{w}) = \lambda^\top f(\mathbf{t}, \mathbf{w})$$

- ... that decompose along the sequence

$$= \lambda^\top \sum_i f(t_i, t_{i-1}, \mathbf{w}, i)$$

- ... allow us to predict with the Viterbi algorithm

$$\mathbf{t}^* = \arg\max_{\mathbf{t}} \ \text{score}(\mathbf{t}|\mathbf{w})$$

- ... which means we can train with the perceptron algorithm (or related updates, like MIRA)

Conditional Random Fields

- Make a maxent model over entire taggings
  - MEMM

$$P(\mathbf{t}|\mathbf{w}) = \prod_i \frac{1}{Z(i)} \exp\left(\lambda^\top f(t_i, t_{i-1}, \mathbf{w}, i)\right)$$

  - CRF

$$P(\mathbf{t}|\mathbf{w}) = \frac{1}{Z(\mathbf{w})} \exp\left(\lambda^\top f(\mathbf{t}, \mathbf{w})\right)$$

$$= \frac{1}{Z(\mathbf{w})} \exp\left(\lambda^\top \sum_i f(t_i, t_{i-1}, \mathbf{w}, i)\right)$$

$$= \frac{1}{Z(\mathbf{w})} \prod_i \phi_i(t_i, t_{i-1})$$

CRFs

- Like any maxent model, derivative is:

$$\frac{\partial L(\lambda)}{\partial \lambda} = \sum_k \left(\mathbf{f}_k(\mathbf{t}^k) - \sum_{\mathbf{t}} P(\mathbf{t}|\mathbf{w}_k)\mathbf{f}_k(\mathbf{t}) \right)$$

- So all we need is to be able to compute the expectation of each feature (for example the number of times the label pair *DT-NN* occurs, or the number of times *NN-interest* occurs) **under the model distribution**

- Critical quantity: counts of posterior marginals:

$$\text{count}(w, s) = \sum_{i:w_i=w} P(t_i = s|\mathbf{w})$$

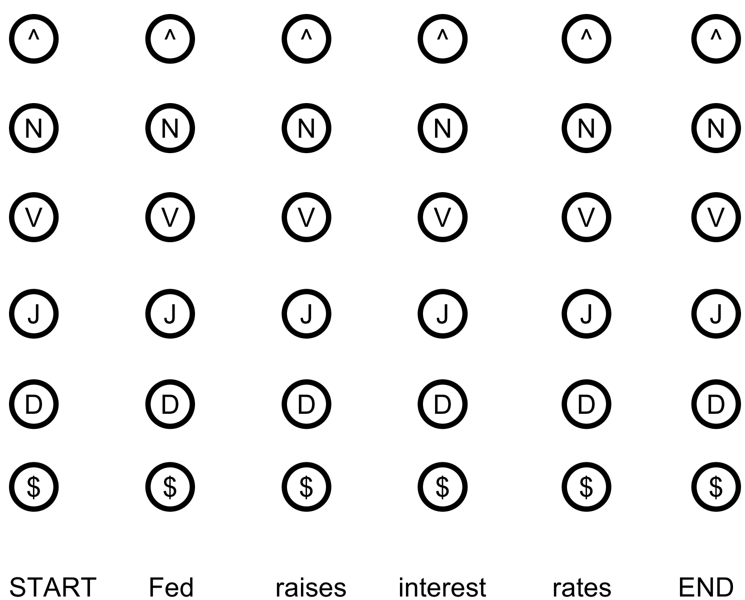
$$\text{count}(s \rightarrow s') = \sum_i P(t_{i-1} = s, t_i = s'|\mathbf{w})$$

Computing Posterior Marginals

- How many (expected) times is word w tagged with s?

$$\text{count}(w, s) = \sum_{i:w_i=w} P(t_i = s|\mathbf{w})$$

- How to compute that marginal?



$$\alpha_i(s) = \sum_{s'} \phi_i(s', s)\alpha_{i-1}(s')$$

$$\beta_i(s) = \sum_{s'} \phi_{i+1}(s, s')\beta_{i+1}(s')$$

$$P(t_i = s|\mathbf{w}) = \frac{\alpha_i(s)\beta_i(s)}{\alpha_N(\text{END})}$$

Global Discriminative Taggers

- Newer, higher-powered discriminative sequence models
  - CRFs (also perceptrons, M3Ns)
  - Do not decompose training into independent local regions
  - Can be deathly slow to train – require repeated inference on training set
- Differences tend not to be too important for POS tagging
- Differences more substantial on other sequence tasks
- However: one issue worth knowing about in local models
  - "Label bias" and other explaining away effects
  - MEMM taggers' local scores can be near one without having both good "transitions" and "emissions"
  - This means that often evidence doesn't flow properly
  - Why isn't this a big deal for POS tagging?
  - Also: in decoding, condition on predicted, not gold, histories

Transformation-Based Learning

- [Brill 95] presents a *transformation-based* tagger
  - Label the training set with most frequent tags

    DT MD VBD VBD .
    The can was rusted .

  - Add transformation rules which reduce training mistakes

    - MD $\rightarrow$ NN : DT __
    - VBD $\rightarrow$ VBN : VBD __ .

  - Stop when no transformations do sufficient good
  - Does this remind anyone of anything?

- Probably the most widely used tagger (esp. outside NLP)
- … but definitely not the most accurate: 96.6% / 82.0 %

Learned Transformations

- What gets learned? [from Brill 95]

	Change Tag		
#	From	To	Condition
1	NN	VB	Previous tag is *TO*
2	VBP	VB	One of the previous three tags is *MD*
3	NN	VB	One of the previous two tags is *MD*
4	VB	NN	One of the previous two tags is *DT*
5	VBD	VBN	One of the previous three tags is *VBZ*
6	VBN	VBD	Previous tag is *PRP*
7	VBN	VBD	Previous tag is *NNP*
8	VBD	VBN	Previous tag is *VBD*
9	VBP	VB	Previous tag is *TO*
10	POS	VBZ	Previous tag is *PRP*
11	VB	VBP	Previous tag is *NNS*
12	VBD	VBN	One of previous three tags is *VBP*
13	IN	WDT	One of next two tags is *VB*
14	VBD	VBN	One of previous two tags is *VB*
15	VB	VBP	Previous tag is *PRP*
16	IN	WDT	Next tag is *VBZ*
17	IN	DT	Next tag is *NN*
18	JJ	NNP	Next tag is *NNP*
19	IN	WDT	Next tag is *VBD*
20	JJR	RBR	Next tag is *JJ*

	Change Tag		
#	From	To	Condition
1	NN	NNS	Has suffix **-s**
2	NN	CD	Has character **.**
3	NN	JJ	Has character **-**
4	NN	VBN	Has suffix **-ed**
5	NN	VBG	Has suffix **-ing**
6	??	RB	Has suffix **-ly**
7	??	JJ	Adding suffix **-ly** results in a word.
8	NN	CD	The word **$** can appear to the left.
9	NN	JJ	Has suffix **-al**
10	NN	VB	The word **would** can appear to the left.
11	NN	CD	Has character **0**
12	NN	JJ	The word **be** can appear to the left.
13	NNS	JJ	Has suffix **-us**
14	NNS	VBZ	The word **it** can appear to the left.
15	NN	JJ	Has suffix **-ble**
16	NN	JJ	Has suffix **-ic**
17	NN	CD	Has character **1**
18	NNS	NN	Has suffix **-ss**
19	??	JJ	Deleting the prefix **un-** results in a word
20	NN	JJ	Has suffix **-ive**

EngCG Tagger

- English constraint grammar tagger
  - [Tapanainen and Voutilainen 94]
  - Something else you should know about
  - Hand-written and knowledge driven
  - “Don’t guess if you know” (general point about modeling more structure!)
  - Tag set doesn’t make all of the hard distinctions as the standard tag set (e.g. JJ/NN)
  - They get stellar accuracies: 99% on *their* tag set
  - Linguistic representation matters…
  - … but it’s easier to win when you make up the rules

```
walk
    walk <SV> <SVO> V SUBJUNCTIVE VFIN
    walk <SV> <SVO> V IMP VFIN
    walk <SV> <SVO> V INF
    walk <SV> <SVO> V PRES -SG3 VFIN
    walk N NOM SG
```

```
walk V-SUBJUNCTIVE V-IMP V-INF
     V-PRES-BASE N-NOM-SG
```

Domain Effects

- Accuracies degrade outside of domain
  - Up to triple error rate
  - Usually make the most errors on the things you care about in the domain (e.g. protein names)

- Open questions
  - How to effectively exploit unlabeled data from a new domain (what could we gain?)
  - How to best incorporate domain lexica in a principled way (e.g. UMLS specialist lexicon, ontologies)

Unsupervised Tagging

Unsupervised Tagging?

- AKA part-of-speech induction
- Task:
  - Raw sentences in
  - Tagged sentences out
- Obvious thing to do:
  - Start with a (mostly) uniform HMM
  - Run EM
  - Inspect results

EM for HMMs: Process

- Alternate between recomputing distributions over hidden variables (the tags) and reestimating parameters
- Crucial step: we want to tally up how many (fractional) counts of each kind of transition and emission we have under current params:

$$\text{count}(w, s) = \sum_{i: w_i = w} P(t_i = s | \mathbf{w})$$

$$\text{count}(s \to s') = \sum_i P(t_{i-1} = s, t_i = s' | \mathbf{w})$$

- Same quantities we needed to train a CRF!

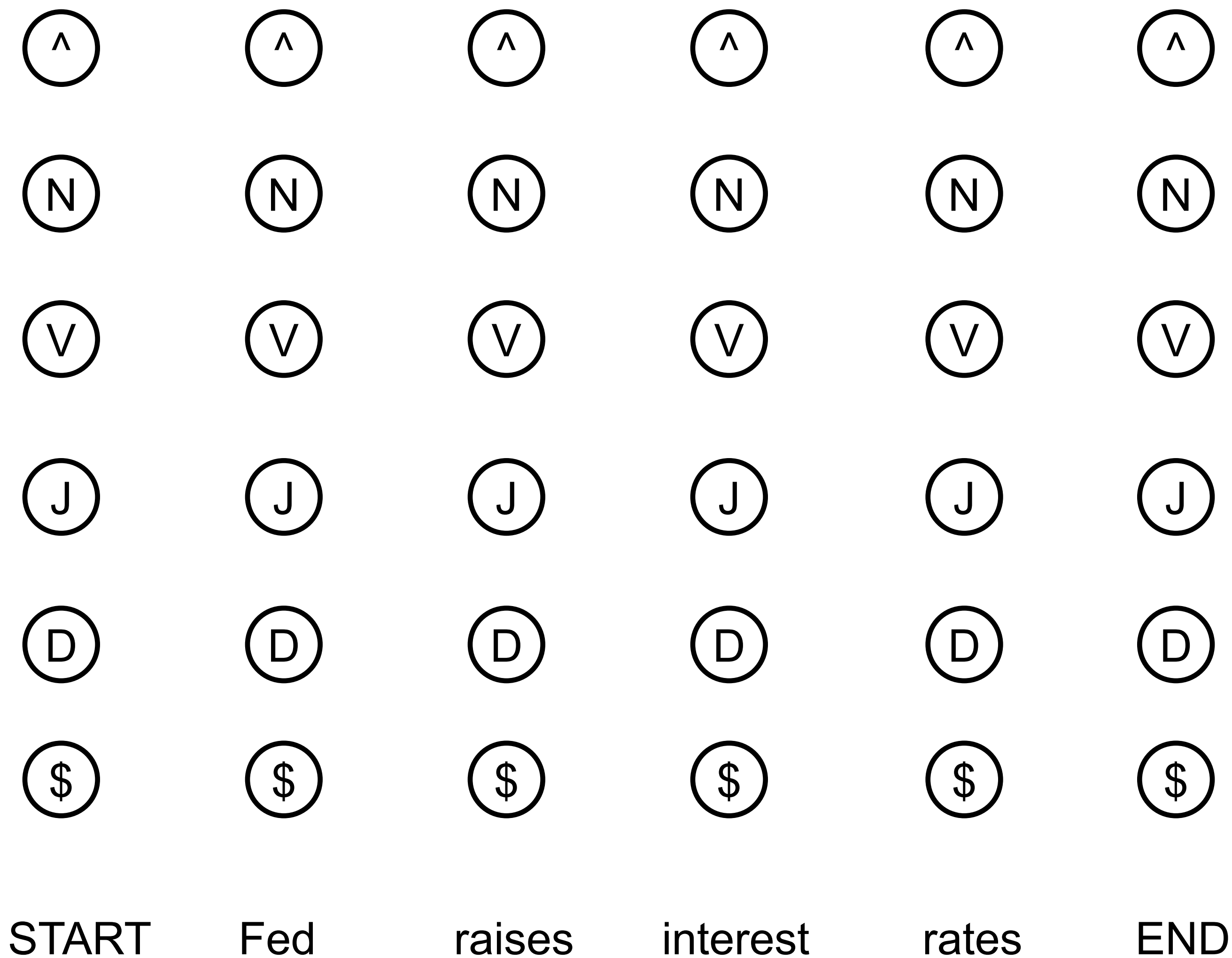
EM for HMMs: Quantities

- Total path values (correspond to probabilities here):

$$
\begin{aligned}
\alpha_i(s) &= P(w_0 \ldots w_i, s_i) \\
&= \sum_{s_{i-1}} P(s_i|s_{i-1}) P(w_i|s_i) \alpha_{i-1}(s_{i-1})
\end{aligned}
$$

$$
\begin{aligned}
\beta_i(s) &= P(w_i + 1 \ldots w_n|s_i) \\
&= \sum_{s_{i+1}} P(s_{i+1}|s_i) P(w_{i+1}|s_{i+1}) \beta_{i+1}(s_{i+1})
\end{aligned}
$$

The State Lattice / Trellis



EM for HMMs: Process

- From these quantities, can compute expected transitions:

$$\text{count}(s \to s') = \frac{\sum_i \alpha_i(s) P(s'|s) P(w_i|s) \beta_{i+1}(s')}{P(\mathbf{w})}$$

- And emissions:

$$\text{count}(w, s) = \frac{\sum_{i:w_i=w} \alpha_i(s) \beta_{i+1}(s)}{P(\mathbf{w})}$$

Merialdo: Setup

- Some (discouraging) experiments [Merialdo 94]

- Setup:
  - You know the set of allowable tags for each word
  - Fix k training examples to their true labels
    - Learn $P(w|t)$ on these examples
    - Learn $P(t|t_{-1},t_{-2})$ on these examples
  - On n examples, re-estimate with EM

- Note: we know allowed tags but not frequencies

Merialdo: Results

	Number of tagged sentences used for the initial model						
	0	100	2000	5000	10000	20000	all
Iter	Correct tags (% words) after ML on 1M words						
0	77.0	90.0	95.4	96.2	96.6	96.9	97.0
1	80.5	92.6	95.8	96.3	96.6	96.7	96.8
2	81.8	93.0	95.7	96.1	96.3	96.4	96.4
3	83.0	93.1	95.4	95.8	96.1	96.2	96.2
4	84.0	93.0	95.2	95.5	95.8	96.0	96.0
5	84.8	92.9	95.1	95.4	95.6	95.8	95.8
6	85.3	92.8	94.9	95.2	95.5	95.6	95.7
7	85.8	92.8	94.7	95.1	95.3	95.5	95.5
8	86.1	92.7	94.6	95.0	95.2	95.4	95.4
9	86.3	92.6	94.5	94.9	95.1	95.3	95.3
10	86.6	92.6	94.4	94.8	95.0	95.2	95.2

Distributional Clustering

♦ the president said that the downturn was over ♦

president	the __ of
president	the __ said
governor	the __ of
governor	the __ appointed
said	sources __ ♦
said	president __ that
reported	sources __ ♦

president
governor

the
a

said
reported

[Finch and Chater 92, Shuetze 93, many others]

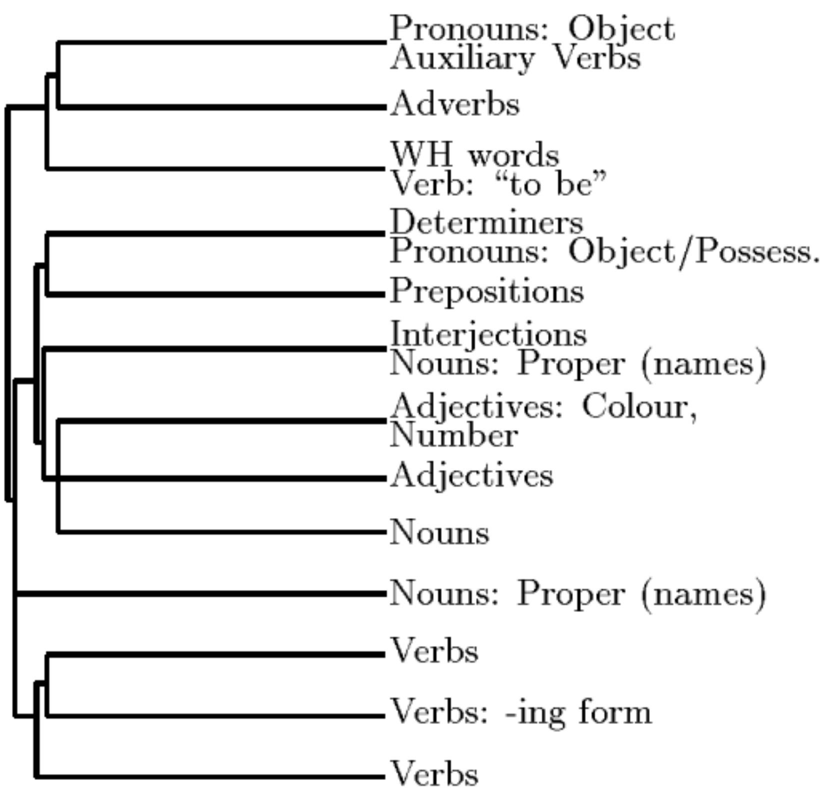
Distributional Clustering

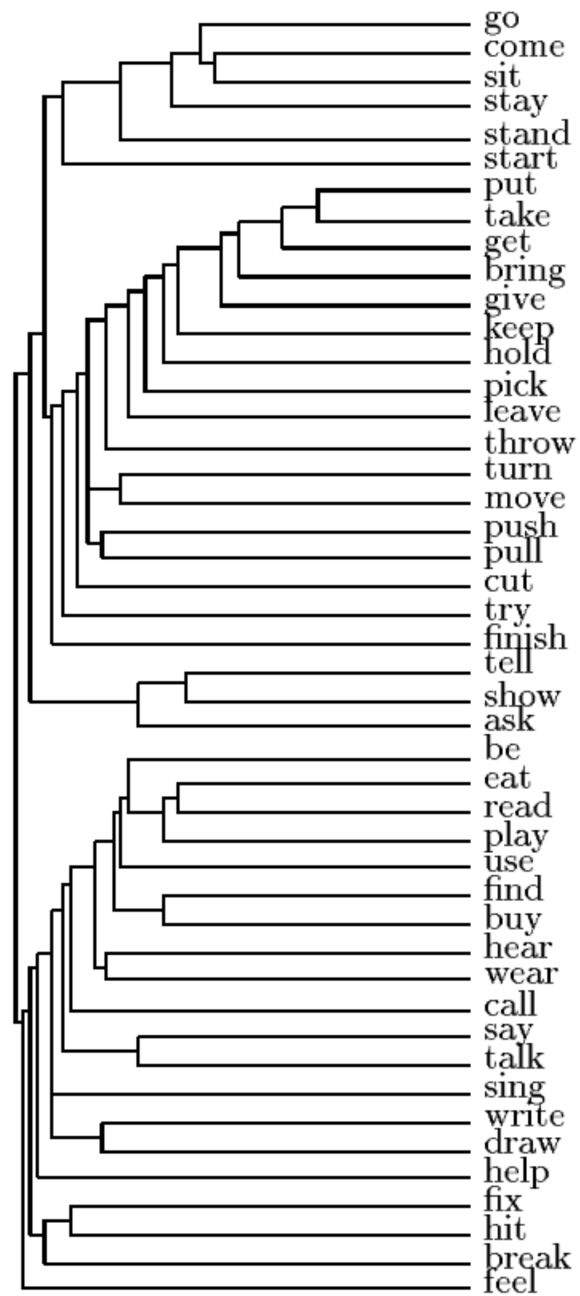
- Three main variants on the same idea:
  - Pairwise similarities and heuristic clustering
    - E.g. [Finch and Chater 92]
    - Produces dendrograms
  - Vector space methods
    - E.g. [Shuetze 93]
    - Models of ambiguity
  - Probabilistic methods
    - Various formulations, e.g. [Lee and Pereira 99]

Nearest Neighbors

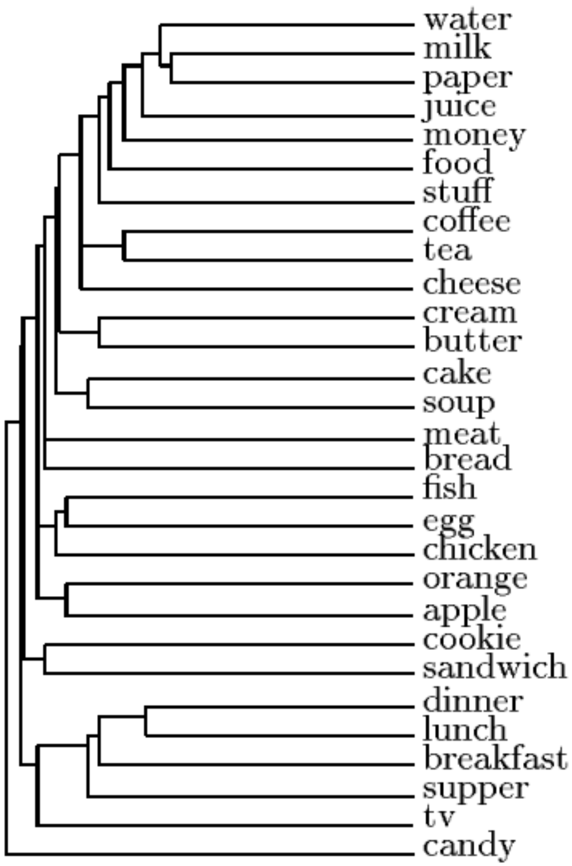
word	nearest neighbors
accompanied	submitted banned financed developed authorized headed canceled awarded barred
almost	virtually merely formally fully quite officially just nearly only less
causing	reflecting forcing providing creating producing becoming carrying particularly
classes	elections courses payments losses computers performances violations levels pictures
directors	professionals investigations materials competitors agreements papers transactions
goal	mood roof eye image tool song pool scene gap voice
japanese	chinese iraqi american western arab foreign european federal soviet indian
represent	reveal attend deliver reflect choose contain impose manage establish retain
think	believe wish know realize wonder assume feel say mean bet
york	angeles francisco sox rouge kong diego zone vegas inning layer
on	through in at over into with from for by across
must	might would could cannot will should can may does helps
they	we you i he she nobody who it everybody there

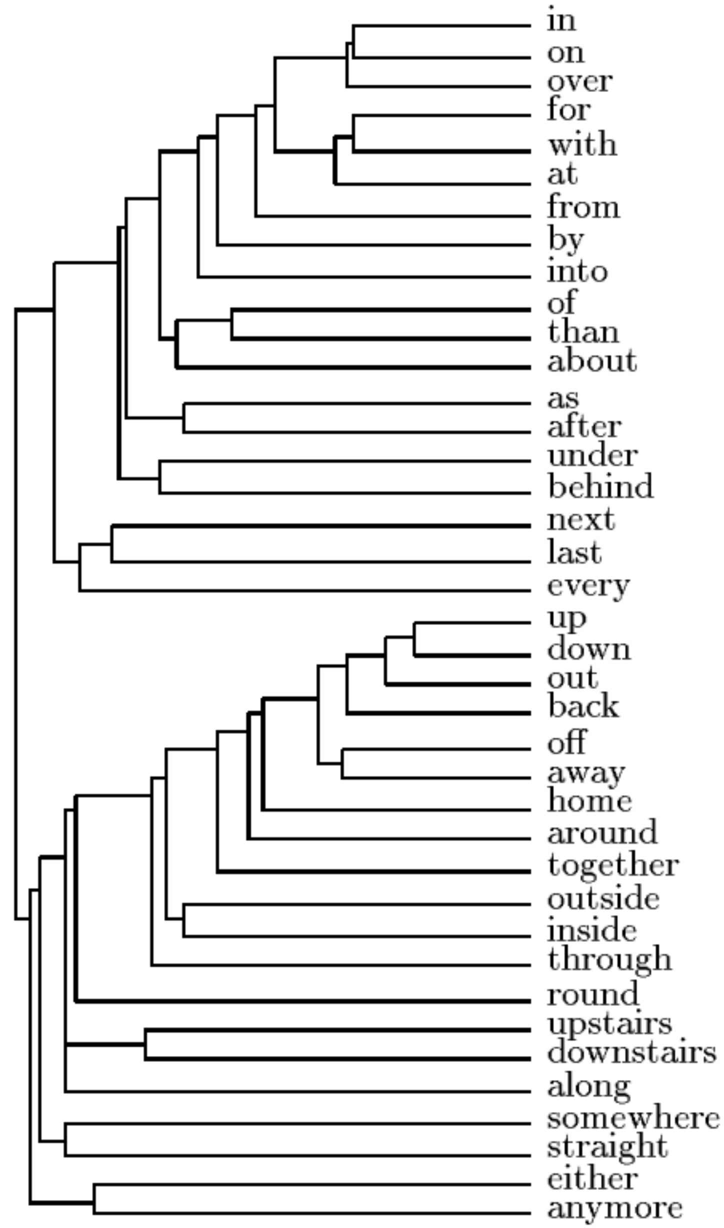
Dendrograms





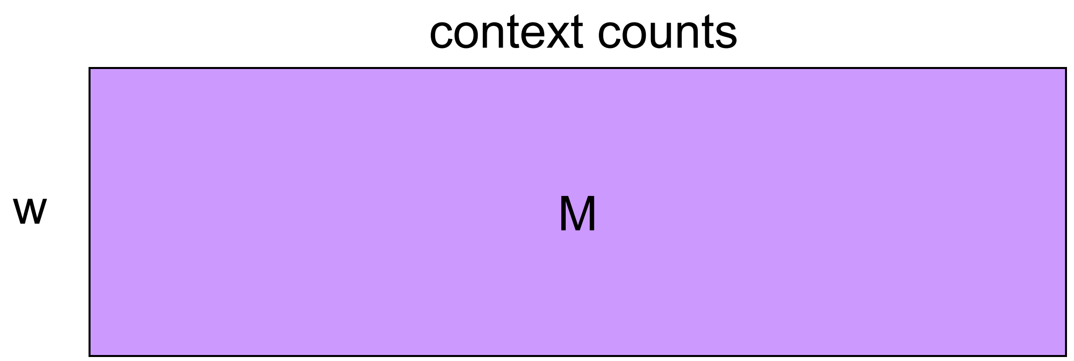
Dendrograms



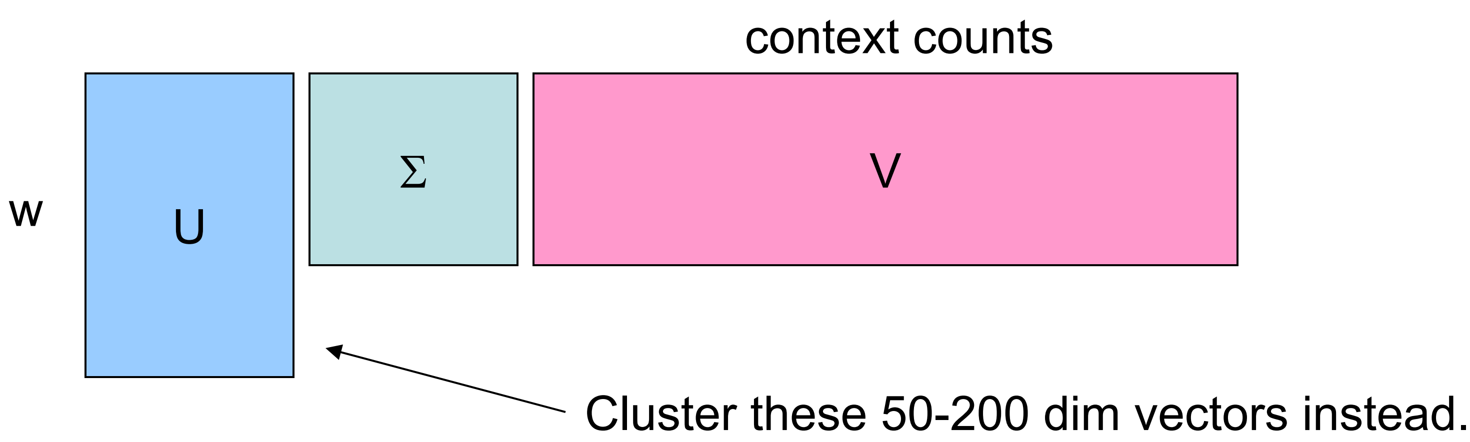


Vector Space Version

- [Shuetze 93] clusters words as points in R^n

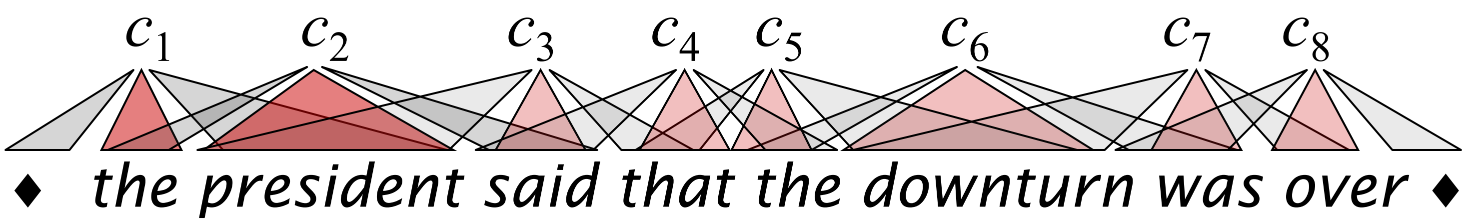


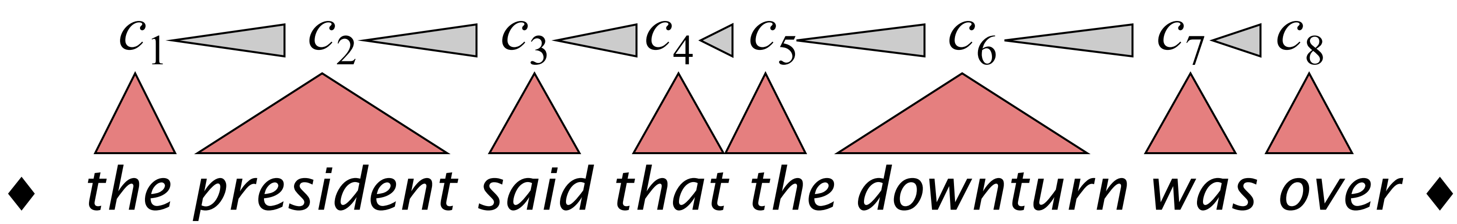
- Vectors too sparse, use SVD to reduce



A Probabilistic Version?

$$P(S,C) = \prod_i P(c_i)P(w_i \mid c_i)P(w_{i-1}, w_{i+1} \mid c_i)$$



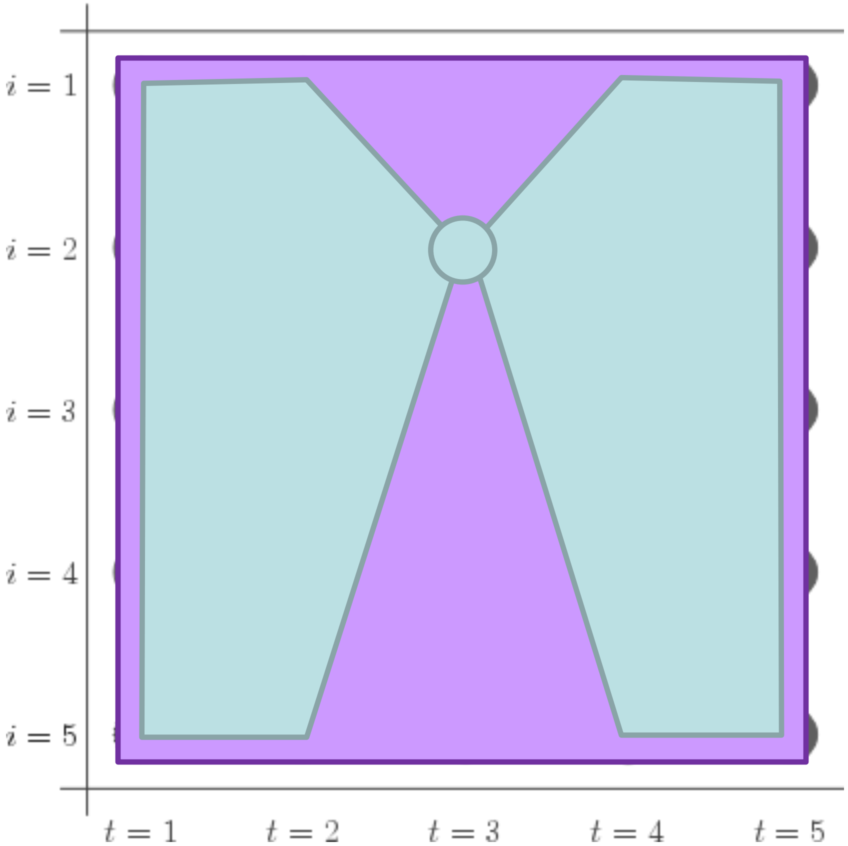


What Else?

- Various newer ideas:
  - Context distributional clustering [Clark 00]
  - Morphology-driven models [Clark 03]
  - Contrastive estimation [Smith and Eisner 05]
  - Feature-rich induction [Haghighi and Klein 06]

- Also:
  - What about ambiguous words?
  - Using wider context signatures has been used for learning synonyms (what's wrong with this approach?)
  - Can extend these ideas for grammar induction (later)

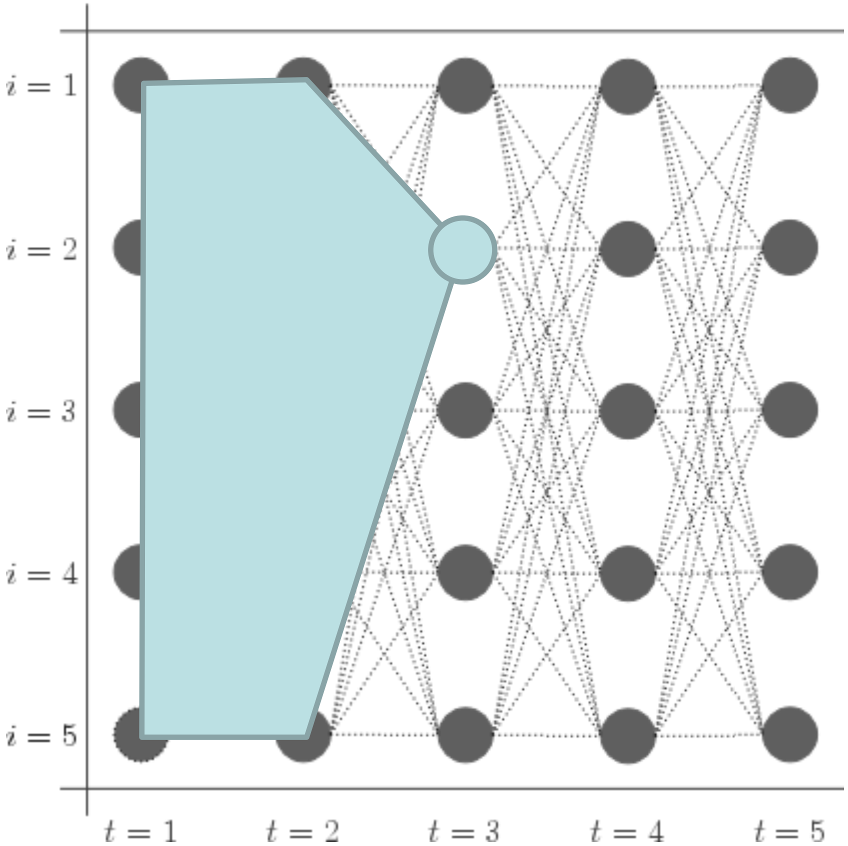
Computing Marginals



$$P(s_t|x) = \frac{P(s_t, x)}{P(x)}$$

$$= \frac{\text{sum of all paths through s at t}}{\text{sum of all paths}}$$

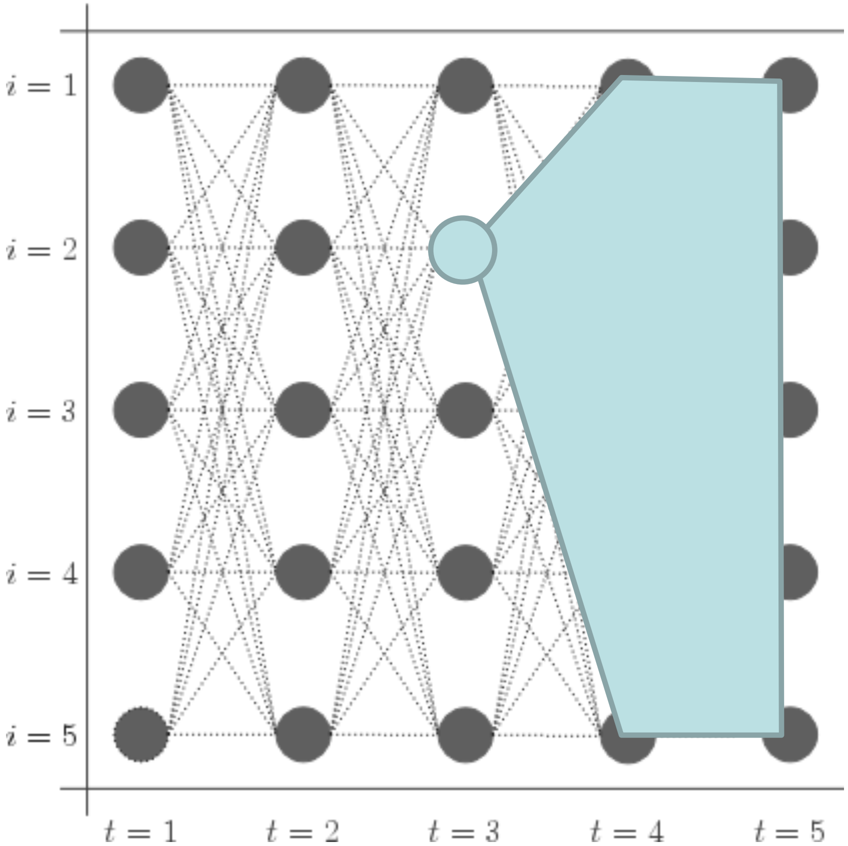
Forward Scores



$$v_t(s_t) = \max_{s_{t-1}} v_{t-1}(s_{t-1})\phi_t(s_{t-1}, s_t)$$

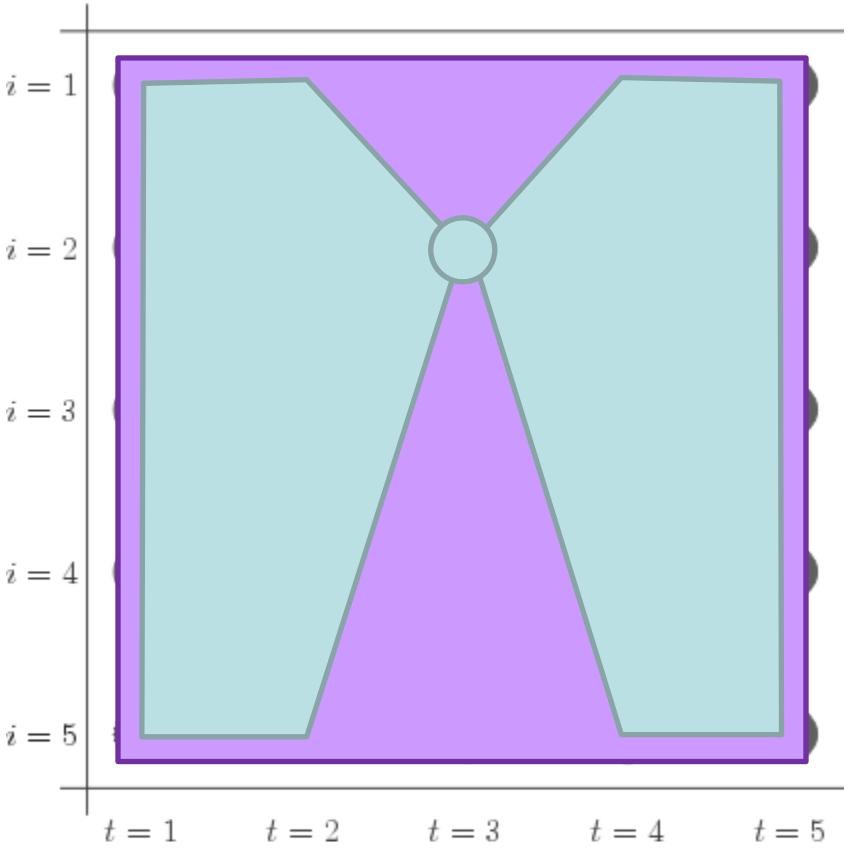
$$\alpha_t(s_t) = \sum_{s_{t-1}} \alpha_{t-1}(s_{t-1})\phi_t(s_{t-1}, s_t)$$

Backward Scores



$$\beta_t(s_t) = \sum_{s_{t+1}} \beta_{t+1}(s_{t+1}) \phi_t(s_t, s_{t+1})$$

Total Scores

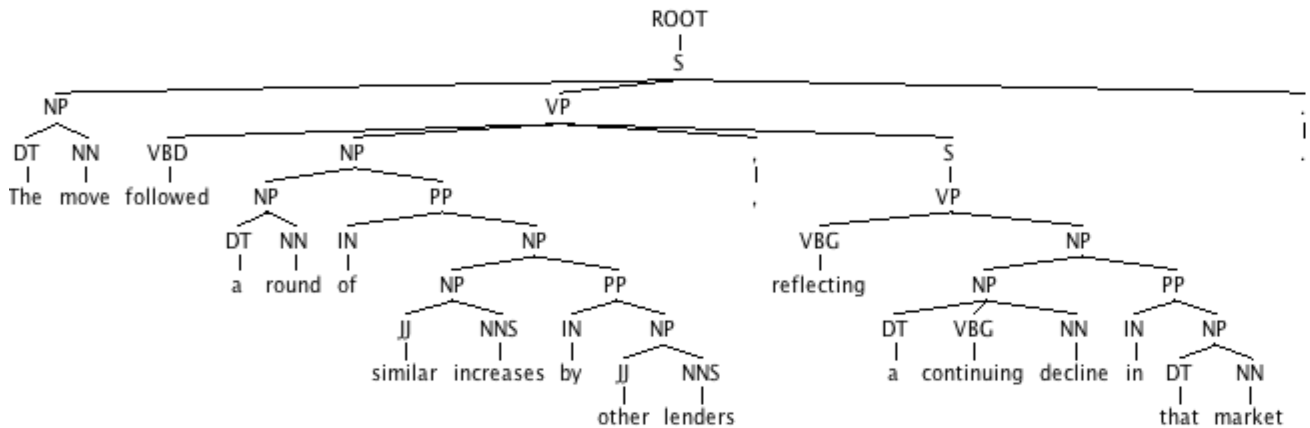


$$P(s_t, x) = \alpha_t(s_t)\beta_t(s_t)$$

$$P(x) = \sum_{s_t} \alpha_t(s_t)\beta_t(s_t)$$
$$= \alpha_T(\text{stop})$$
$$= \beta_0(\text{start})$$

Syntax

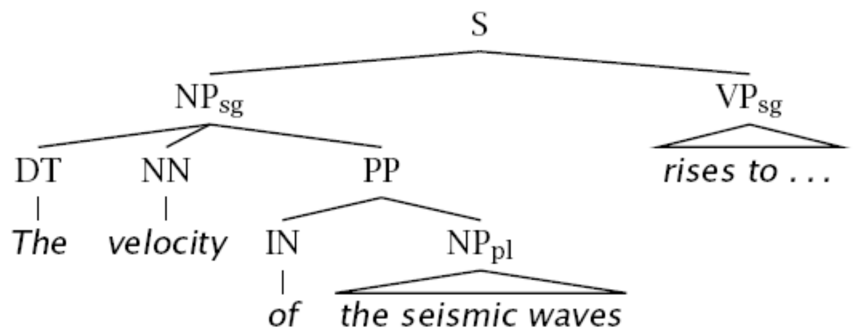
Parse Trees



The move followed a round of similar increases by other lenders, reflecting a continuing decline in that market

Phrase Structure Parsing

- Phrase structure parsing organizes syntax into *constituents* or *brackets*
- In general, this involves nested trees
- Linguists can, and do, argue about details
- Lots of ambiguity
- Not the only kind of syntax…

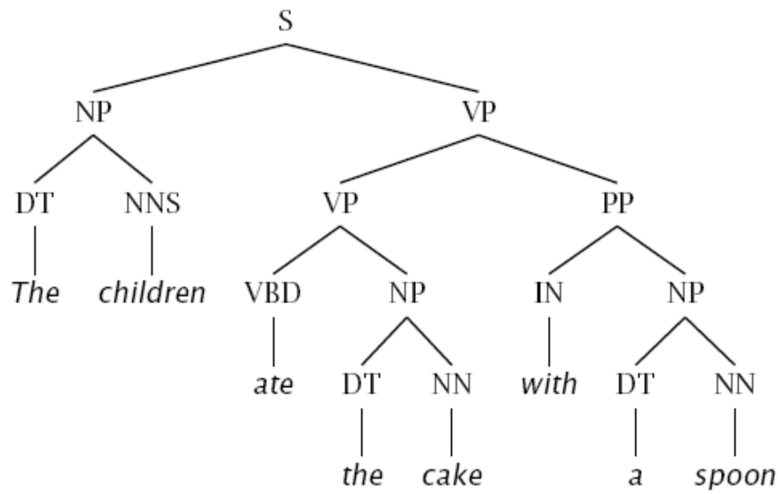


new art critics write reviews with computers

Constituency Tests

- How do we know what nodes go in the tree?

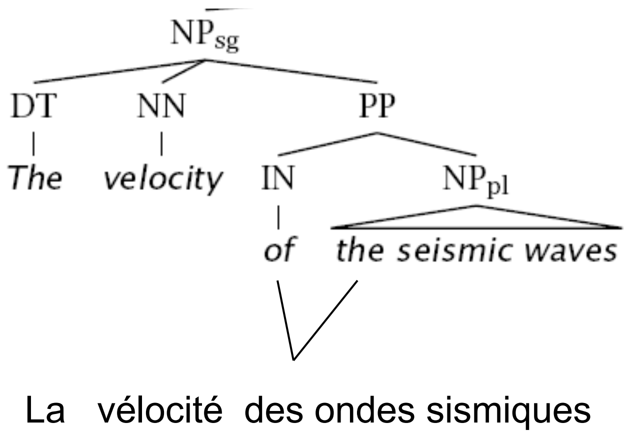
- Classic constituency tests:
  - Substitution by *proform*
  - Question answers
  - Semantic gounds
    - Coherence
    - Reference
    - Idioms
  - Dislocation
  - Conjunction



- Cross-linguistic arguments, too

Conflicting Tests

- Constituency isn't always clear
  - Units of transfer:
    - think about ~ penser à
    - talk about ~ hablar de
  - Phonological reduction:
    - I will go → I'll go
    - I want to go → I wanna go
    - a le centre → au centre
  - Coordination
    - He went to and came from the store.



Classical NLP: Parsing

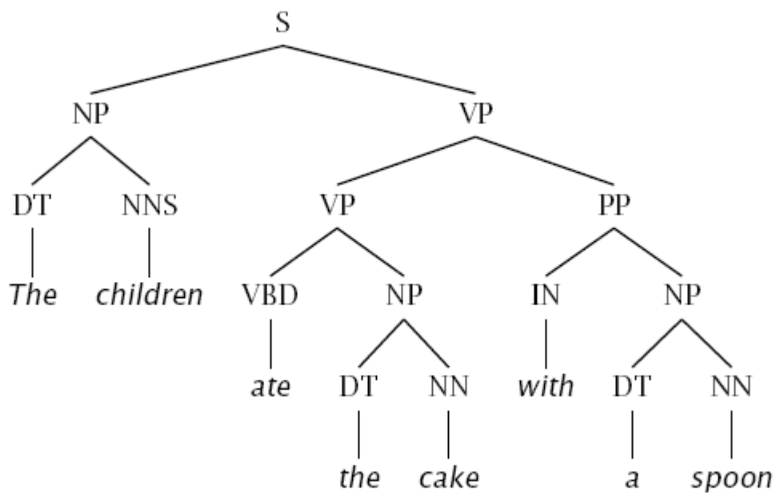
- Write symbolic or logical rules:

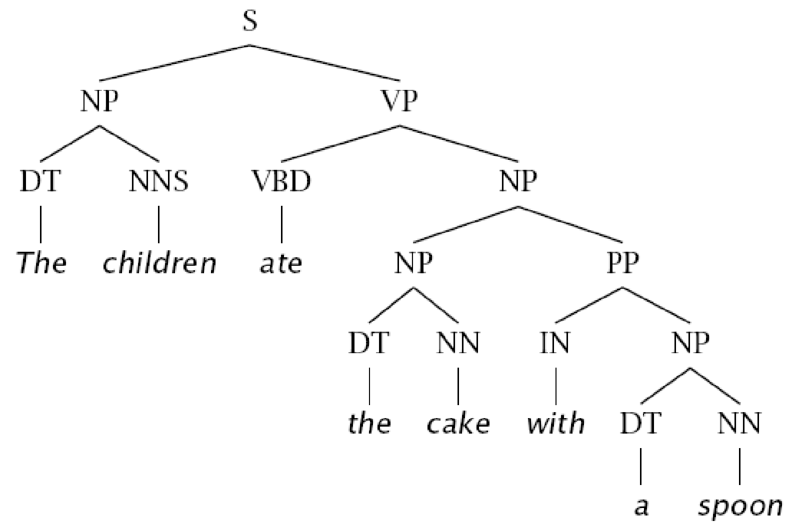
Grammar (CFG)		Lexicon
ROOT $\rightarrow$ S	NP $\rightarrow$ NP PP	NN $\rightarrow$ interest
S $\rightarrow$ NP VP	VP $\rightarrow$ VBP NP	NNS $\rightarrow$ raises
NP $\rightarrow$ DT NN	VP $\rightarrow$ VBP NP PP	VBP $\rightarrow$ interest
NP $\rightarrow$ NN NNS	PP $\rightarrow$ IN NP	VBZ $\rightarrow$ raises
		…

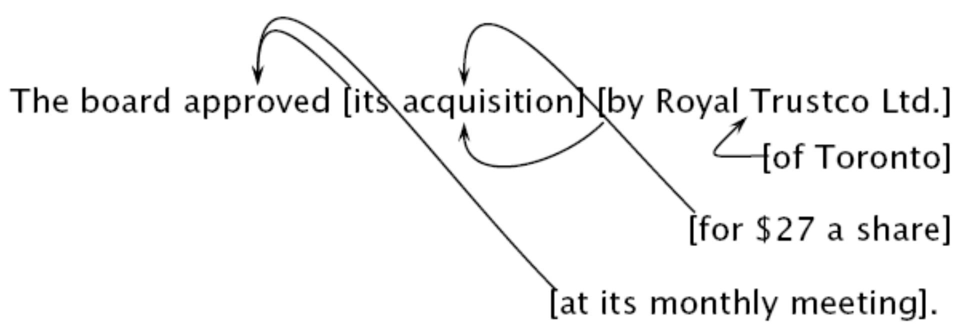
- Use deduction systems to prove parses from words
  - Minimal grammar on "Fed raises" sentence: 36 parses
  - Simple 10-rule grammar: 592 parses
  - Real-size grammar: many millions of parses
- This scaled very badly, didn't yield broad-coverage tools

Ambiguities

Ambiguities: PP Attachment







Attachments

- I cleaned the dishes from dinner
- I cleaned the dishes with detergent
- I cleaned the dishes in my pajamas
- I cleaned the dishes in the sink

Syntactic Ambiguities I

- Prepositional phrases:
  They cooked the beans in the pot on the stove with handles.
- Particle vs. preposition:
  The puppy tore up the staircase.
- Complement structures
  The tourists objected to the guide that they couldn't hear.
  She knows you like the back of her hand.
- Gerund vs. participial adjective
  Visiting relatives can be boring.
  Changing schedules frequently confused passengers.

Syntactic Ambiguities II

- Modifier scope within NPs
  impractical design requirements
  plastic cup holder

- Multiple gap constructions
  The chicken is ready to eat.
  The contractors are rich enough to sue.

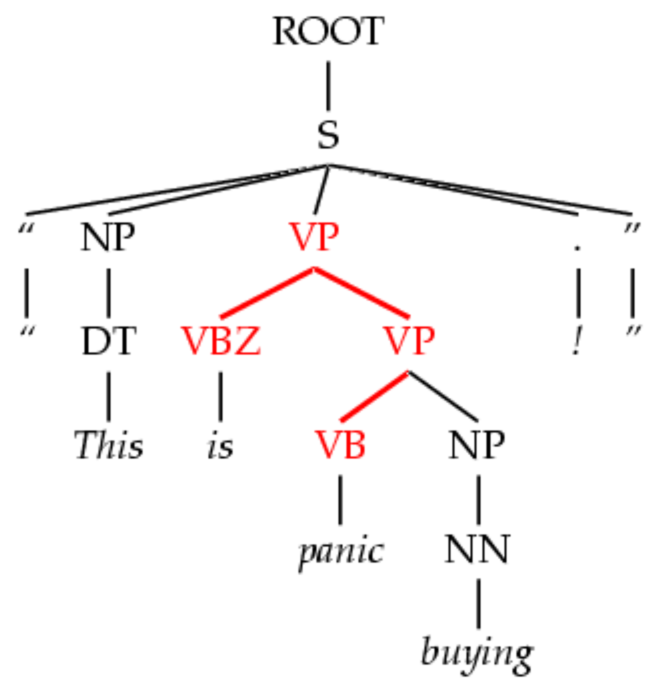
- Coordination scope:
  Small rats and mice can squeeze into holes or cracks in the wall.

Dark Ambiguities

- *Dark ambiguities*: most analyses are shockingly bad (meaning, they don't have an interpretation you can get your mind around)

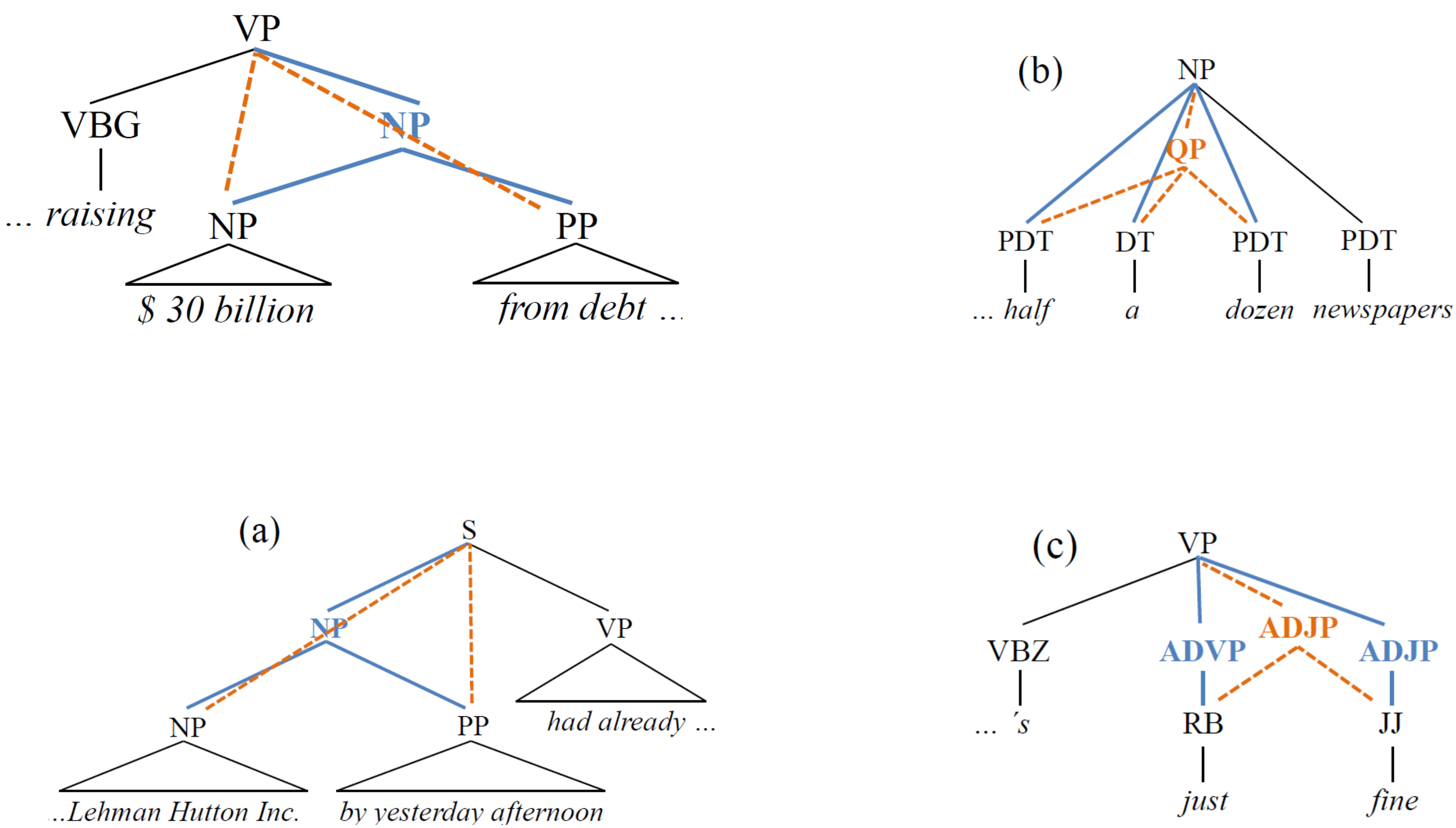
This analysis corresponds to the correct parse of

"This will panic buyers ! "



- Unknown words and new usages
- Solution: We need mechanisms to focus attention on the best ones, probabilistic techniques do this

Ambiguities as Trees



PCFGs

Probabilistic Context-Free Grammars

- A context-free grammar is a tuple <*N, T, S, R*>
  - *N* : the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - *T* : the set of terminals (the words)
  - *S* : the start symbol
    - Often written as ROOT or TOP
    - *Not* usually the sentence non-terminal S
  - *R* : the set of rules
    - Of the form $X \rightarrow Y_1\ Y_2 \ldots Y_k$, with $X, Y_i \in N$
    - Examples: $S \rightarrow NP\ VP$, $VP \rightarrow VP\ CC\ VP$
    - Also called rewrites, productions, or local trees

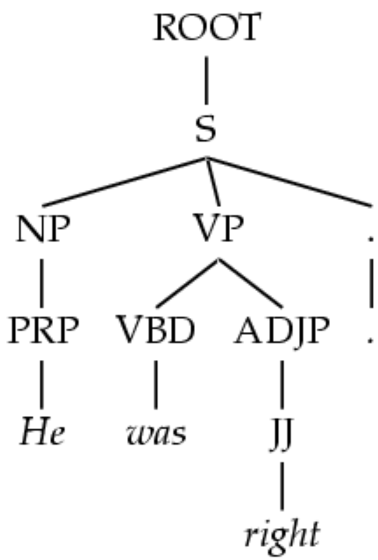
- A PCFG adds:
  - A top-down production probability per rule $P(Y_1\ Y_2 \ldots Y_k \mid X)$

Treebank Sentences

```
( (S (NP-SBJ The move)
     (VP followed
         (NP (NP a round)
             (PP of
                 (NP (NP similar increases)
                     (PP by
                         (NP other lenders))
                     (PP against
                         (NP Arizona real estate loans)))))
         ,
         (S-ADV (NP-SBJ *)
                (VP reflecting
                    (NP (NP a continuing decline)
                        (PP-LOC in
                                (NP that market))))))
     .))
```

Treebank Grammars

- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn't work well):

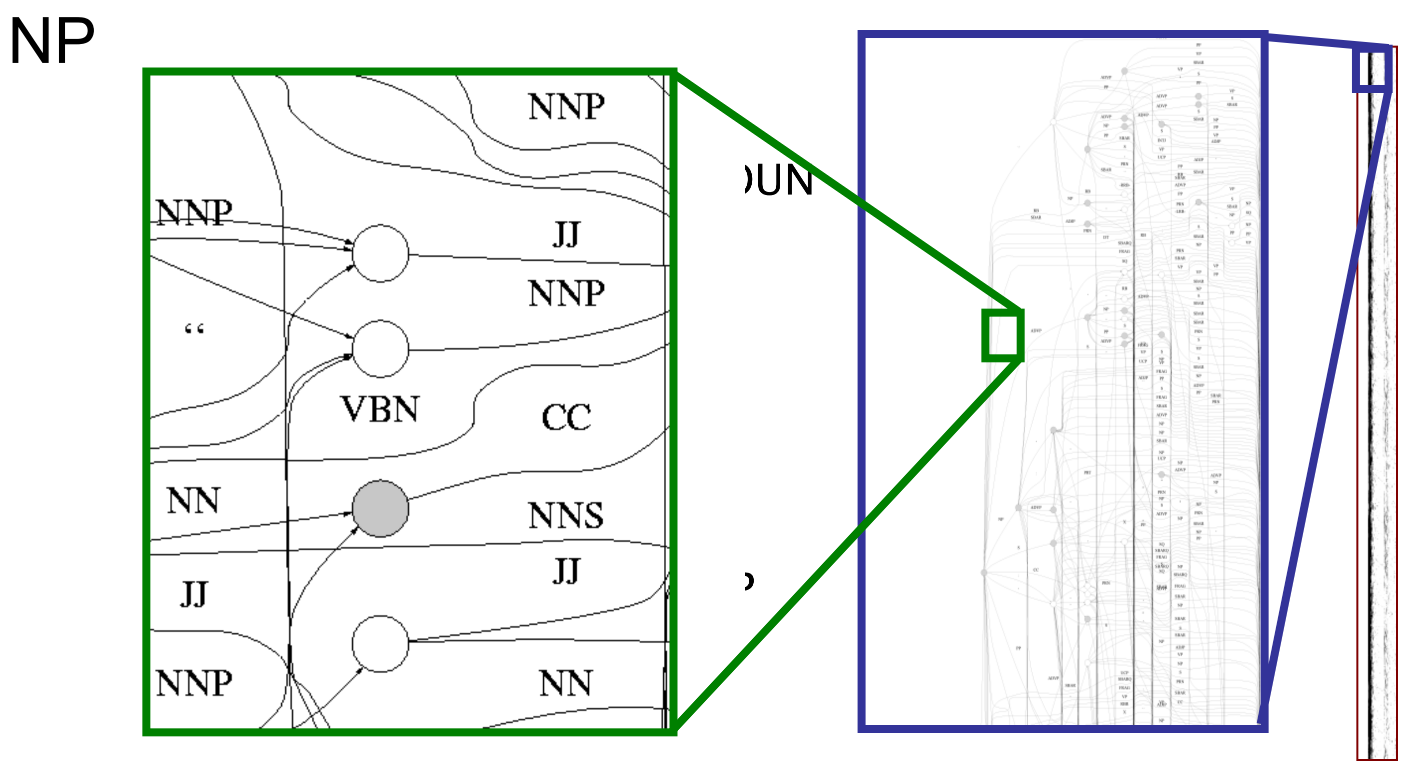


ROOT $\rightarrow$ S	1
S $\rightarrow$ NP VP .	1
NP $\rightarrow$ PRP	1
VP $\rightarrow$ VBD ADJP	1
…..	

- Better results by enriching the grammar (e.g., lexicalization).
- Can also get state-of-the-art parsers without lexicalization.

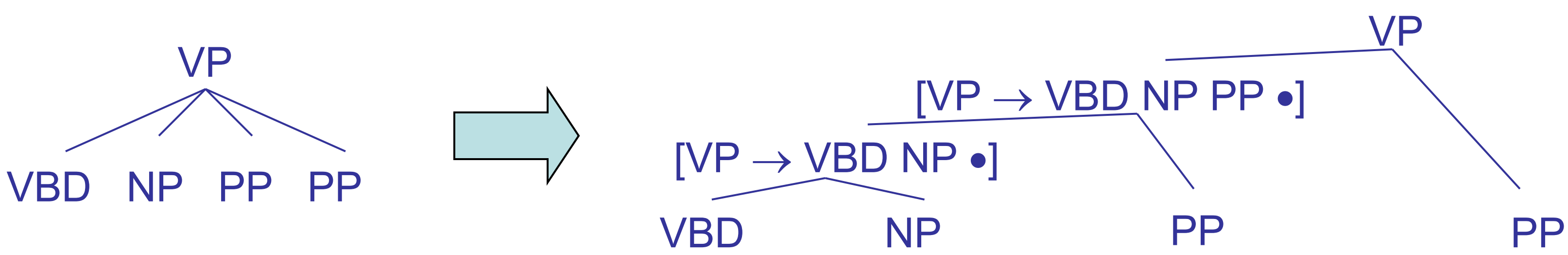
Treebank Grammar Scale

- Treebank grammars can be enormous
  - As FSAs, the raw grammar has ~10K states, excluding the lexicon
  - Better parsers usually make the grammars larger, not smaller



Chomsky Normal Form

- Chomsky normal form:
  - All rules of the form X → Y Z or X → w
  - In principle, this is no limitation on the space of (P)CFGs
    - N-ary rules introduce new non-terminals



    - Unaries / empties are "promoted"
  - In practice it's kind of a pain:
    - Reconstructing n-aries is easy
    - Reconstructing unaries is trickier
    - The straightforward transformations don't preserve tree scores
  - Makes parsing algorithms simpler!

CKY Parsing

A Recursive Parser

```
bestScore(X,i,j,s)
   if (j = i+1)
        return tagScore(X,s[i])
   else
        return max score(X->YZ) *
                   bestScore(Y,i,k) *
                   bestScore(Z,k,j)
```

- Will this parser work?
- Why or why not?
- Memory requirements?

A Memoized Parser

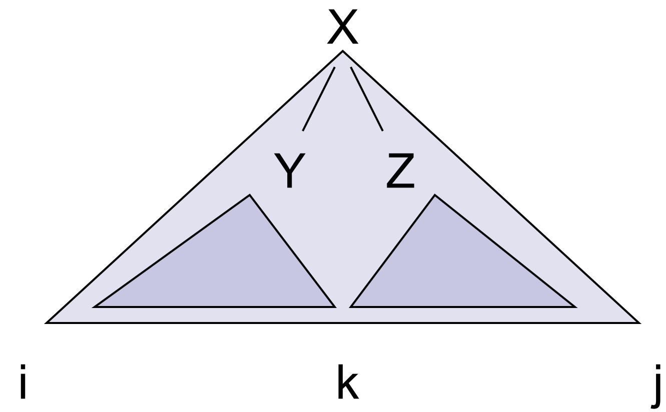
- One small change:

```
bestScore(X,i,j,s)
  if (scores[X][i][j] == null)
      if (j = i+1)
          score = tagScore(X,s[i])
      else
          score = max score(X->YZ) *
                      bestScore(Y,i,k) *
                      bestScore(Z,k,j)
      scores[X][i][j] = score
  return scores[X][i][j]
```

A Bottom-Up Parser (CKY)

- Can also organize things bottom-up

```
bestScore(s)
   for (i : [0,n-1])
     for (X : tags[s[i]])
       score[X][i][i+1] =
           tagScore(X,s[i])
   for (diff : [2,n])
     for (i : [0,n-diff])
       j = i + diff
       for (X->YZ : rule)
         for (k : [i+1, j-1])
           score[X][i][j] = max score[X][i][j],
                                score(X->YZ) *
                                score[Y][i][k] *
                                score[Z][k][j]
```



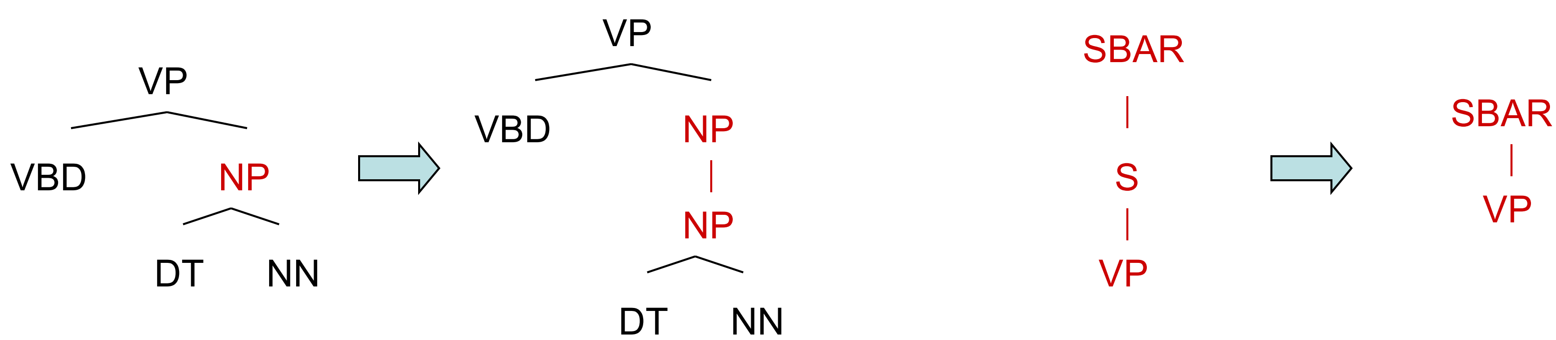
Unary Rules

- Unary rules?

```
bestScore(X,i,j,s)
   if (j = i+1)
       return tagScore(X,s[i])
   else
       return max max score(X->YZ) *
                      bestScore(Y,i,k) *
                      bestScore(Z,k,j)
                 max score(X->Y) *
                      bestScore(Y,i,j)
```

CNF + Unary Closure

- We need unaries to be non-cyclic
  - Can address by pre-calculating the *unary closure*
  - Rather than having zero or more unaries, always have exactly one



  - Alternate unary and binary layers
  - Reconstruct unary chains afterwards

Alternating Layers

```
bestScoreB(X,i,j,s)
      return max max score(X->YZ) *
                      bestScoreU(Y,i,k) *
                      bestScoreU(Z,k,j)


bestScoreU(X,i,j,s)
   if (j = i+1)
       return tagScore(X,s[i])
   else
       return max max score(X->Y) *
                      bestScoreB(Y,i,j)
```

Analysis

Memory

- How much memory does this require?
  - Have to store the score cache
  - Cache size: |symbols|*n^2 doubles
  - For the plain treebank grammar:
    - X ~ 20K, n = 40, double ~ 8 bytes = ~ 256MB
    - Big, but workable.

- Pruning: Beams
  - score[X][i][j] can get too large (when?)
  - Can keep beams (truncated maps score[i][j]) which only store the best few scores for the span [i,j]

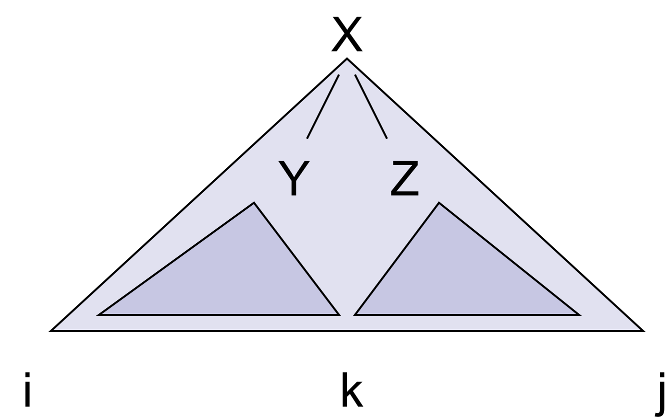
- Pruning: Coarse-to-Fine
  - Use a smaller grammar to rule out most X[i,j]
  - Much more on this later…

Time: Theory

- How much time will it take to parse?

  - For each diff (<= n)
    - For each i (<= n)
      - For each rule $X \rightarrow Y\ Z$
        - For each split point k

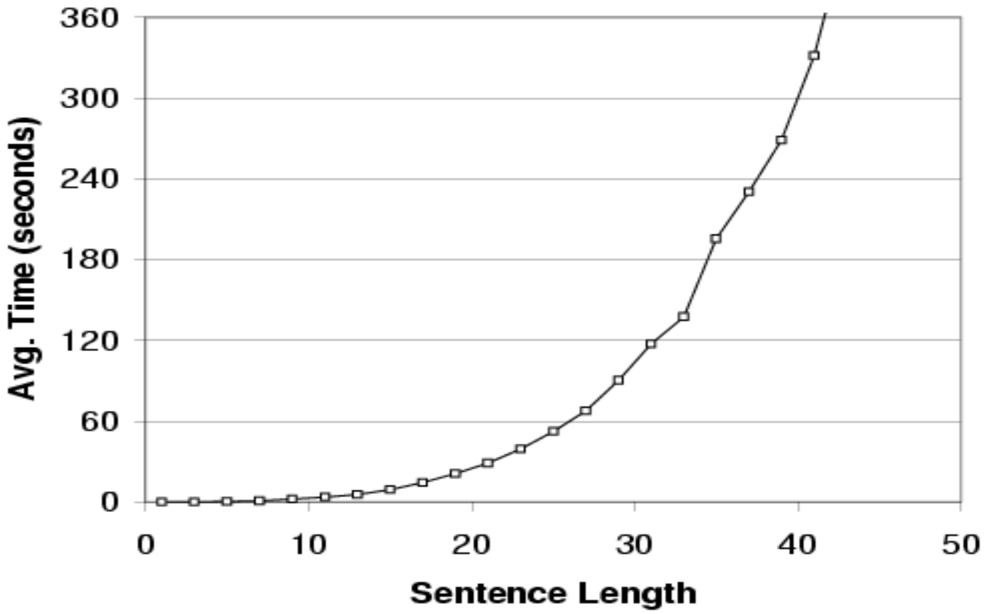
          Do constant work



  - Total time: $|\text{rules}| * n^3$
  - Something like 5 sec for an unoptimized parse of a 20-word sentence

Time: Practice

- Parsing with the vanilla treebank grammar:



~ 20K Rules

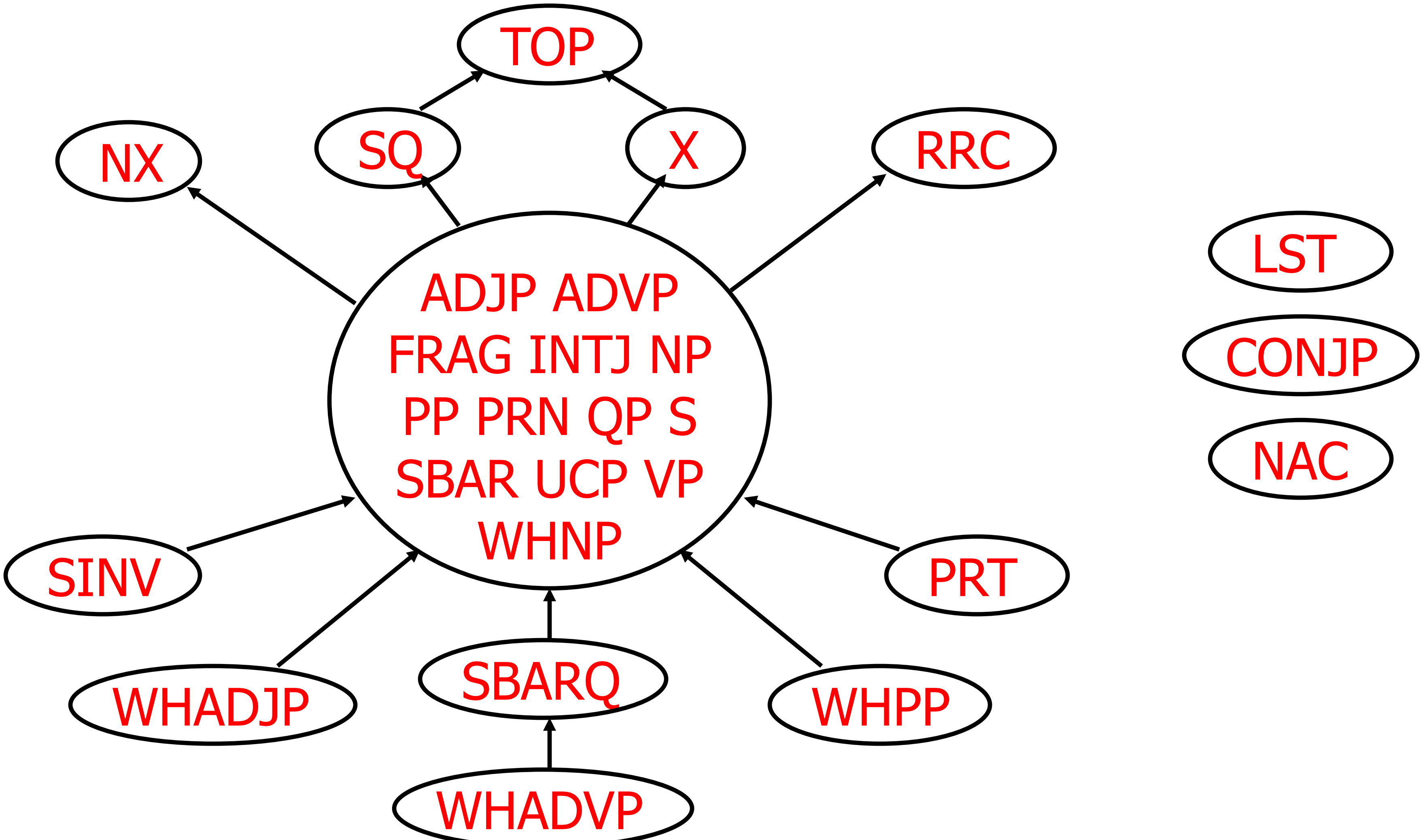
(not an optimized parser!)

Observed exponent:

3.6

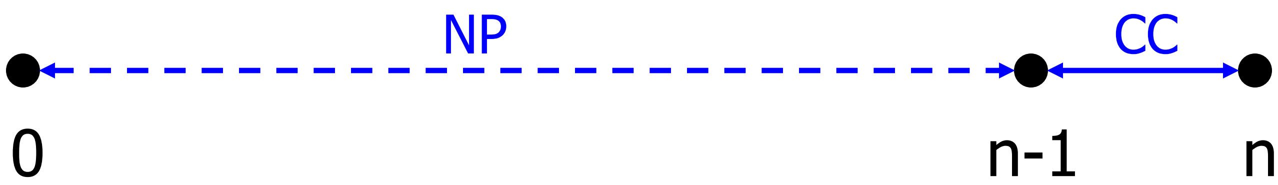
- Why's it worse in practice?
  - Longer sentences "unlock" more of the grammar
  - All kinds of systems issues don't scale

Same-Span Reachability



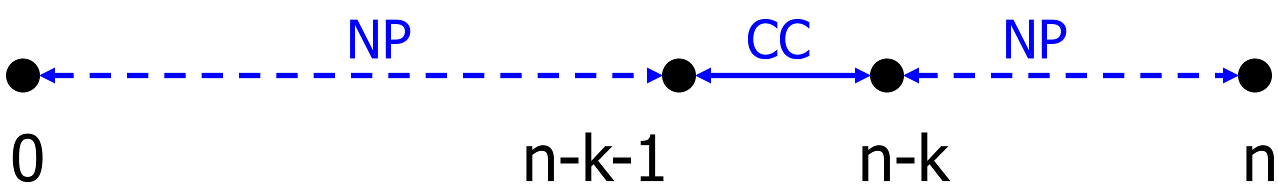
Rule State Reachability

Example: NP CC •



1 Alignment

Example: NP CC NP •



n Alignments

- Many states are more likely to match larger spans!

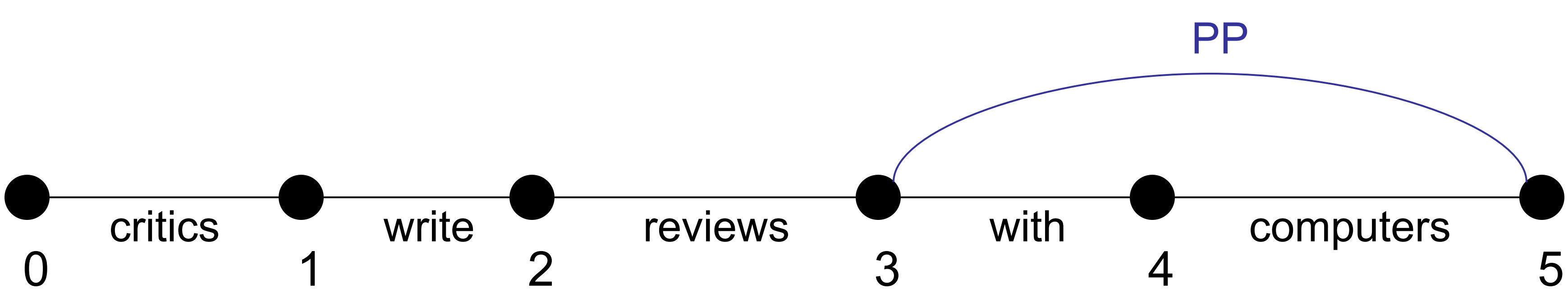
Efficient CKY

- Lots of tricks to make CKY efficient
  - Some of them are little engineering details:
    - E.g., first choose k, then enumerate through the Y:[i,k] which are non-zero, then loop through rules by left child.
    - Optimal layout of the dynamic program depends on grammar, input, even system details.
  - Another kind is more important (and interesting):
    - Many X[i,j] can be suppressed on the basis of the input string
    - We'll see this next class as figures-of-merit, A* heuristics, coarse-to-fine, etc

Agenda-Based Parsing

Agenda-Based Parsing

- Agenda-based parsing is like graph search (but over a hypergraph)
- Concepts:
  - Numbering: we number fenceposts between words
  - "Edges" or items: spans with labels, e.g. PP[3,5], represent the sets of trees over those words rooted at that label (cf. search states)
  - A chart: records edges we've expanded (cf. closed set)
  - An agenda: a queue which holds edges (cf. a fringe or open set)



Word Items

- Building an item for the first time is called discovery. Items go into the agenda on discovery.
- To initialize, we discover all word items (with score 1.0).

AGENDA

critics[0,1], write[1,2], reviews[2,3], with[3,4], computers[4,5]

CHART [EMPTY]

0 1 2 3 4 5

critics write reviews with computers

Unary Projection

- When we pop a word item, the lexicon tells us the tag item successors (and scores) which go on the agenda

critics[0,1]	write[1,2]	reviews[2,3]	with[3,4]	computers[4,5]
NNS[0,1]	VBP[1,2]	NNS[2,3]	IN[3,4]	NNS[4,5]

critics write reviews with computers

Item Successors

- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)

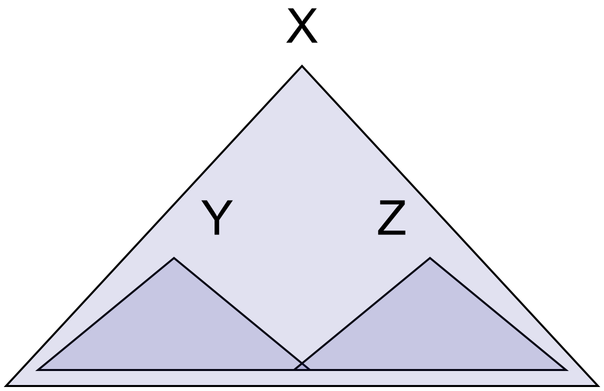
$$Y[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j]$$

  - Hypergraph successors: combine with items already in our chart

$$Y[i,j] \text{ and } Z[j,k] \text{ with } X \rightarrow Y\ Z \text{ form } X[i,k]$$

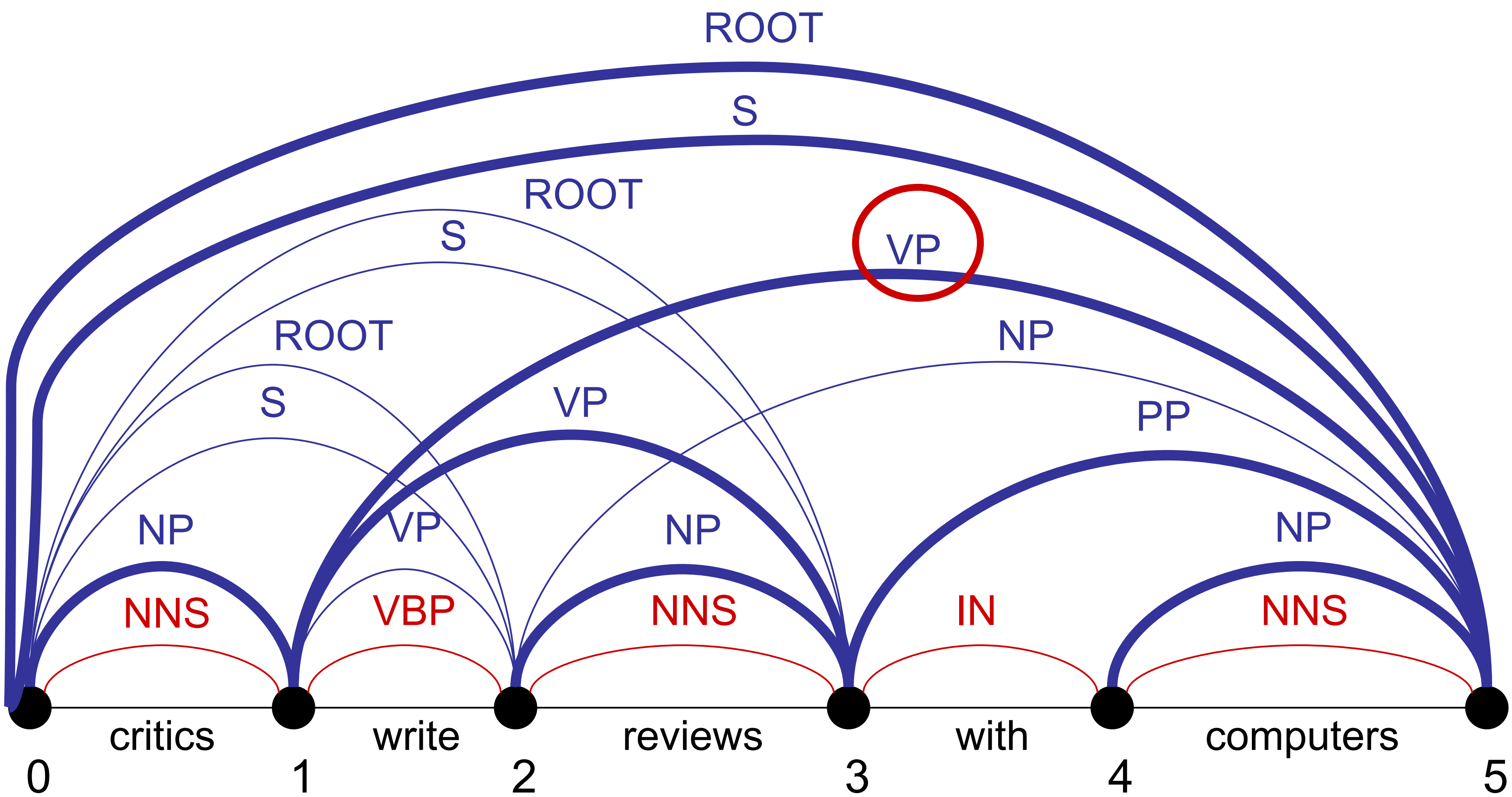
  - Enqueue / promote resulting items (if not in chart already)
  - Record backtraces as appropriate
  - Stick the popped edge in the chart (closed set)

- Queries a chart must support:
  - Is edge X[i,j] in the chart? (What score?)
  - What edges with label Y end at position j?
  - What edges with label Z start at position i?



An Example

NNS[0,1] VBP[1,2] NNS[2,3] IN[3,4] NNS[3,4] NP[0,1] VP[1,2] NP[2,3] NP[4,5] S[0,2] VP[1,3] PP[3,5] ROOT[0,2] S[0,3] VP[1,5] NP[2,5] ROOT[0,3] S[0,5] ROOT[0,5]



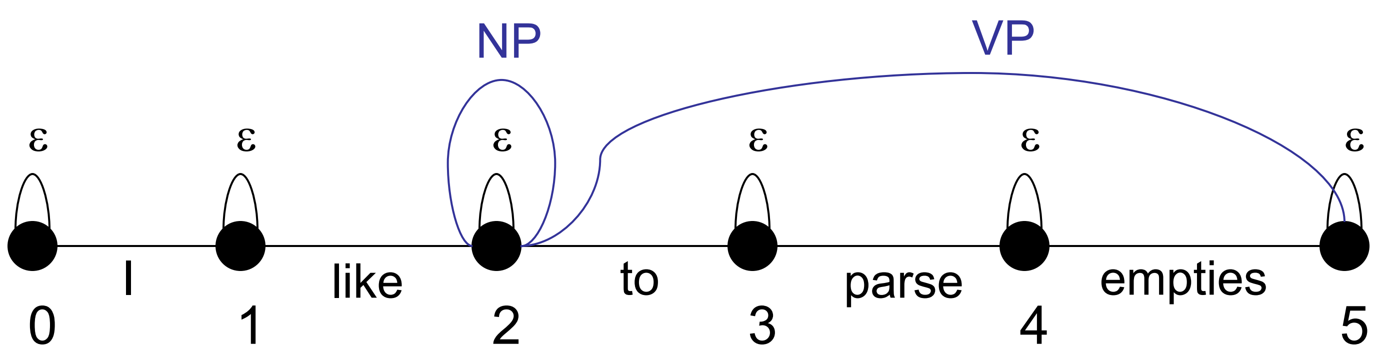
Empty Elements

- Sometimes we want to posit nodes in a parse tree that don't contain any pronounced words:

I want you to parse this sentence

I want [ ] to parse this sentence

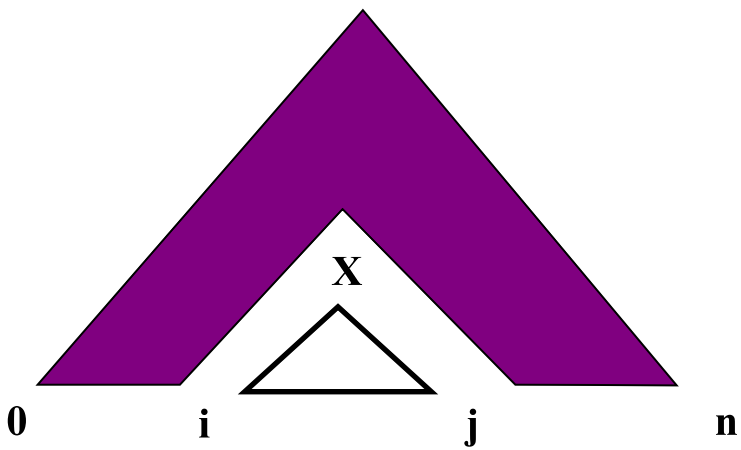
- These are easy to add to a agenda-based parser!
  - For each position i, add the "word" edge $\varepsilon[i,i]$
  - Add rules like NP $\rightarrow \varepsilon$ to the grammar
  - That's it!



UCS / A*

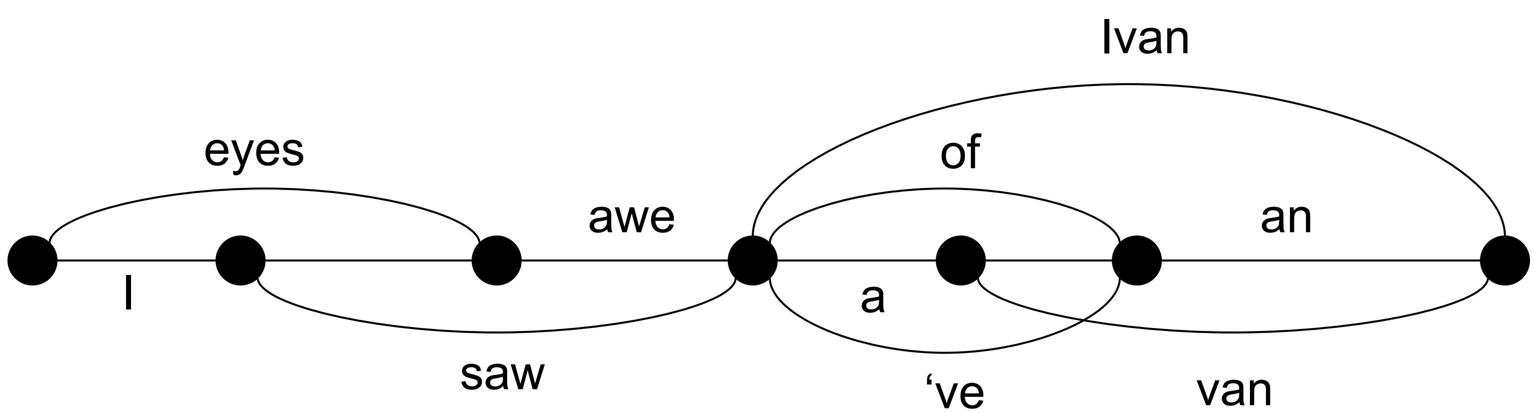
- With weighted edges, order matters
  - Must expand optimal parse from bottom up (subparses first)
  - CKY does this by processing smaller spans before larger ones
  - UCS pops items off the agenda in order of decreasing Viterbi score
  - A* search also well defined

- You can also speed up the search without sacrificing optimality
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimiality [Klein and Manning 03]



(Speech) Lattices

- There was nothing magical about words spanning exactly one position.
- When working with speech, we generally don't know how many words there are, or where they break.
- We can represent the possibilities as a lattice and parse these just as easily.



Unsupervised Tagging

Unsupervised Tagging?

- AKA part-of-speech induction
- Task:
  - Raw sentences in
  - Tagged sentences out
- Obvious thing to do:
  - Start with a (mostly) uniform HMM
  - Run EM
  - Inspect results

EM for HMMs: Process

- Alternate between recomputing distributions over hidden variables (the tags) and reestimating parameters
- Crucial step: we want to tally up how many (fractional) counts of each kind of transition and emission we have under current params:

$$\text{count}(w, s) = \sum_{i: w_i = w} P(t_i = s | \mathbf{w})$$

$$\text{count}(s \rightarrow s') = \sum_{i} P(t_{i-1} = s, t_i = s' | \mathbf{w})$$

- Same quantities we needed to train a CRF!

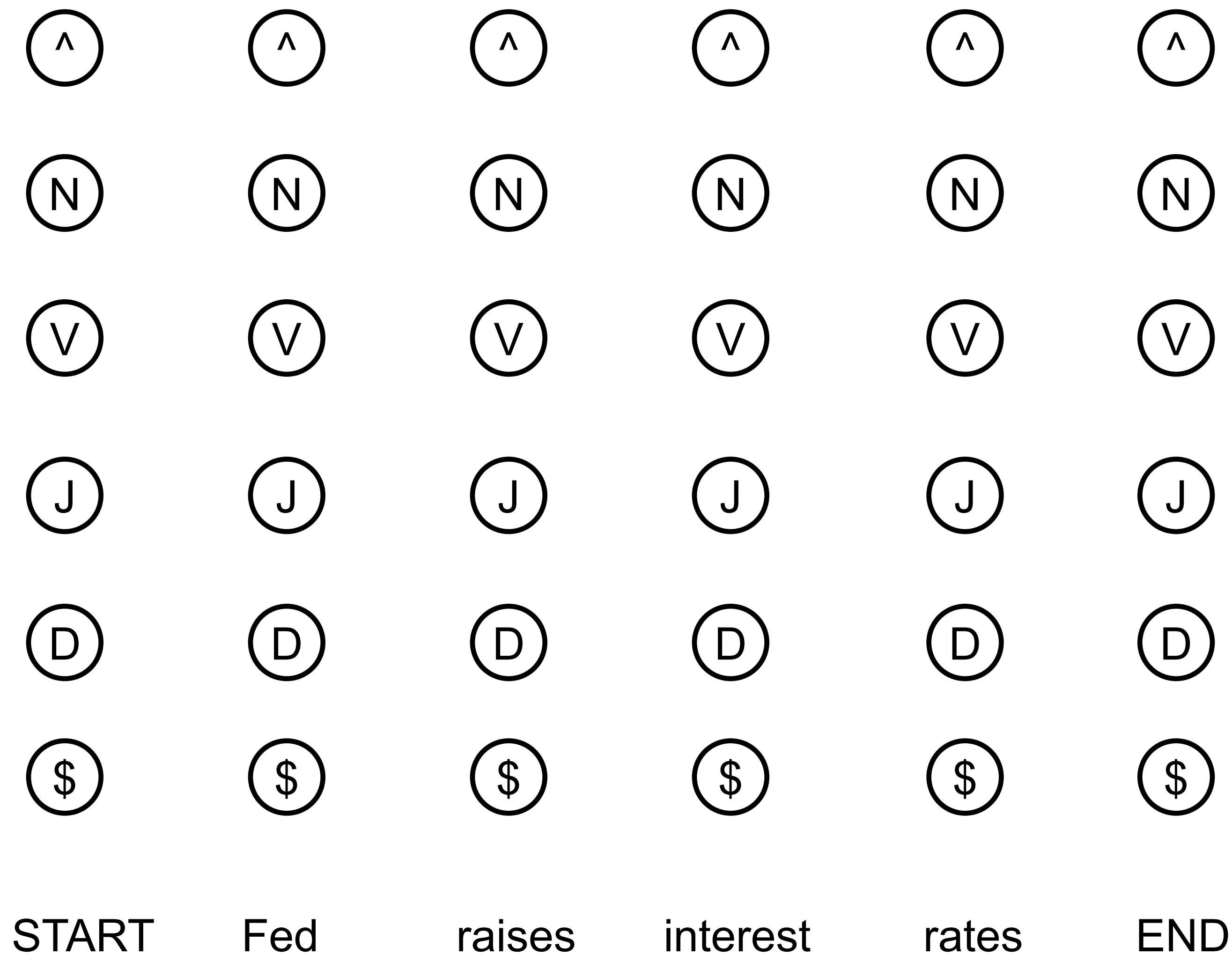
EM for HMMs: Quantities

- Total path values (correspond to probabilities here):

$$\begin{aligned}
\alpha_i(s) &= P(w_0 \ldots w_i, s_i) \\
&= \sum_{s_{i-1}} P(s_i|s_{i-1}) P(w_i|s_i) \alpha_{i-1}(s_{i-1})
\end{aligned}$$

$$\begin{aligned}
\beta_i(s) &= P(w_i + 1 \ldots w_n|s_i) \\
&= \sum_{s_{i+1}} P(s_{i+1}|s_i) P(w_{i+1}|s_{i+1}) \beta_{i+1}(s_{i+1})
\end{aligned}$$

The State Lattice / Trellis



EM for HMMs: Process

- From these quantities, can compute expected transitions:

$$\text{count}(s \to s') = \frac{\sum_i \alpha_i(s) P(s'|s) P(w_i|s) \beta_{i+1}(s')}{P(\mathbf{w})}$$

- And emissions:

$$\text{count}(w, s) = \frac{\sum_{i:w_i=w} \alpha_i(s) \beta_{i+1}(s)}{P(\mathbf{w})}$$

Merialdo: Setup

- Some (discouraging) experiments [Merialdo 94]

- Setup:
  - You know the set of allowable tags for each word
  - Fix k training examples to their true labels
    - Learn $P(w|t)$ on these examples
    - Learn $P(t|t_{-1},t_{-2})$ on these examples
  - On n examples, re-estimate with EM

- Note: we know allowed tags but not frequencies

Merialdo: Results

	Number of tagged sentences used for the initial model						
	0	100	2000	5000	10000	20000	all
Iter	Correct tags (% words) after ML on 1M words						
0	77.0	90.0	95.4	96.2	96.6	96.9	97.0
1	80.5	92.6	95.8	96.3	96.6	96.7	96.8
2	81.8	93.0	95.7	96.1	96.3	96.4	96.4
3	83.0	93.1	95.4	95.8	96.1	96.2	96.2
4	84.0	93.0	95.2	95.5	95.8	96.0	96.0
5	84.8	92.9	95.1	95.4	95.6	95.8	95.8
6	85.3	92.8	94.9	95.2	95.5	95.6	95.7
7	85.8	92.8	94.7	95.1	95.3	95.5	95.5
8	86.1	92.7	94.6	95.0	95.2	95.4	95.4
9	86.3	92.6	94.5	94.9	95.1	95.3	95.3
10	86.6	92.6	94.4	94.8	95.0	95.2	95.2

Distributional Clustering

♦ *the president said that the downturn was over* ♦

president	*the __ of*
president	*the __ said*
governor	*the __ of*
governor	*the __ appointed*
said	*sources __ ♦*
said	*president __ that*
reported	*sources __ ♦*

president
governor

the
a

said
reported

[Finch and Chater 92, Shuetze 93, many others]

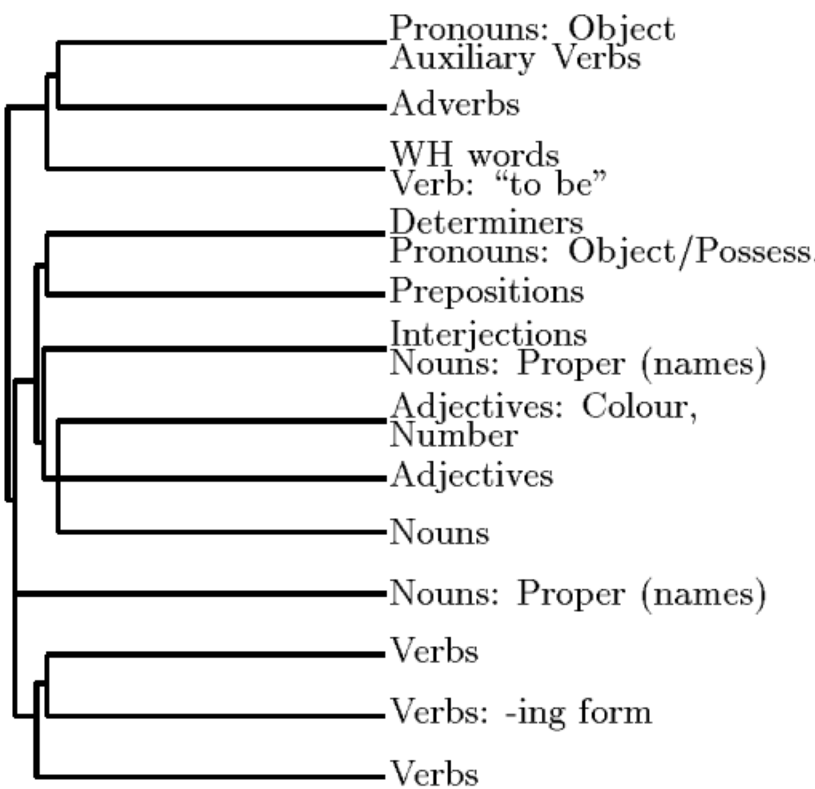
Distributional Clustering

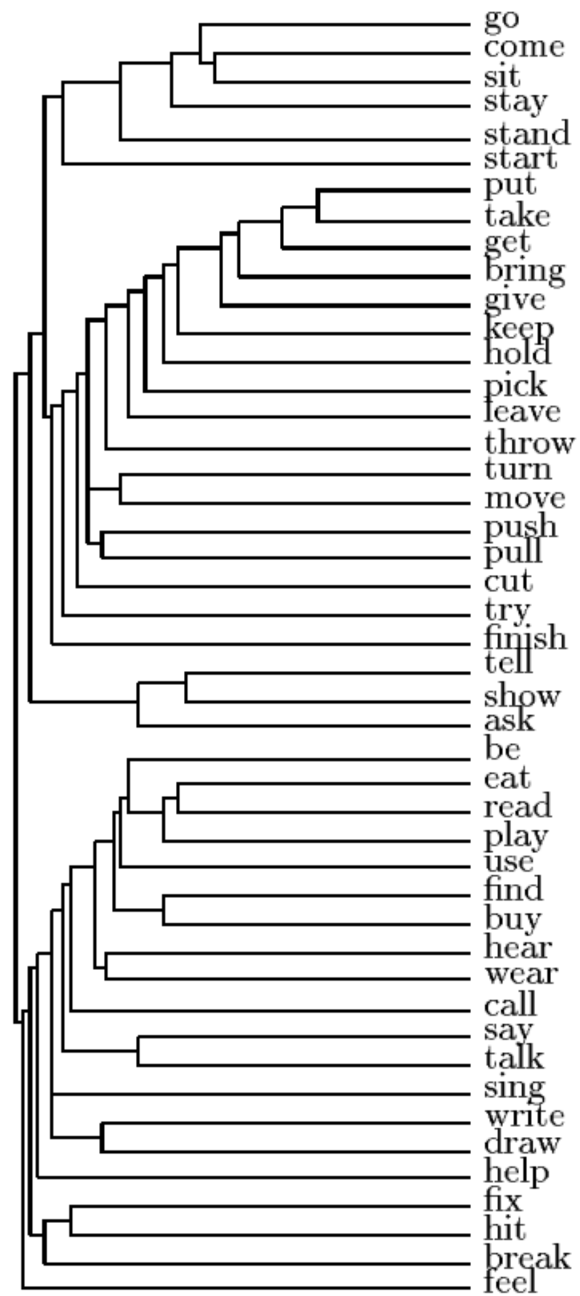
- Three main variants on the same idea:
  - Pairwise similarities and heuristic clustering
    - E.g. [Finch and Chater 92]
    - Produces dendrograms
  - Vector space methods
    - E.g. [Shuetze 93]
    - Models of ambiguity
  - Probabilistic methods
    - Various formulations, e.g. [Lee and Pereira 99]

Nearest Neighbors

word	nearest neighbors
accompanied	submitted banned financed developed authorized headed canceled awarded barred
almost	virtually merely formally fully quite officially just nearly only less
causing	reflecting forcing providing creating producing becoming carrying particularly
classes	elections courses payments losses computers performances violations levels pictures
directors	professionals investigations materials competitors agreements papers transactions
goal	mood roof eye image tool song pool scene gap voice
japanese	chinese iraqi american western arab foreign european federal soviet indian
represent	reveal attend deliver reflect choose contain impose manage establish retain
think	believe wish know realize wonder assume feel say mean bet
york	angeles francisco sox rouge kong diego zone vegas inning layer
on	through in at over into with from for by across
must	might would could cannot will should can may does helps
they	we you i he she nobody who it everybody there

Dendrograms





Dendrograms