The Expectation Maximization (EM) Algorithm

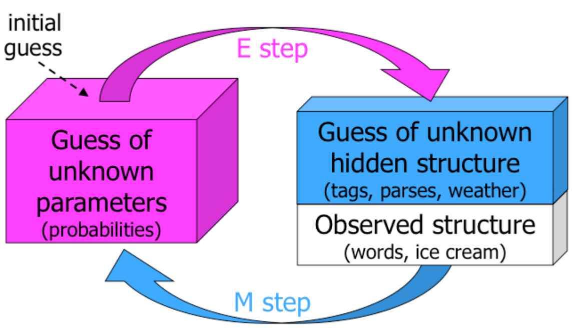
... continued!

General Idea

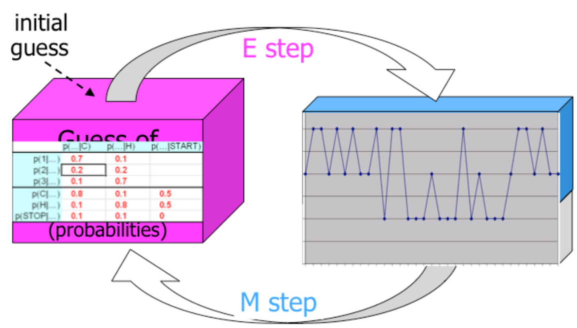
- Start by devising a noisy channel
  - Any model that predicts the corpus observations via some hidden structure (tags, parses, …)
- Initially **guess** the parameters of the model!
  - Educated guess is best, but random can work
- **Expectation step:** Use current parameters (and observations) to reconstruct hidden structure
- **Maximization step:** Use that hidden structure (and observations) to reestimate parameters

Repeat until convergence!

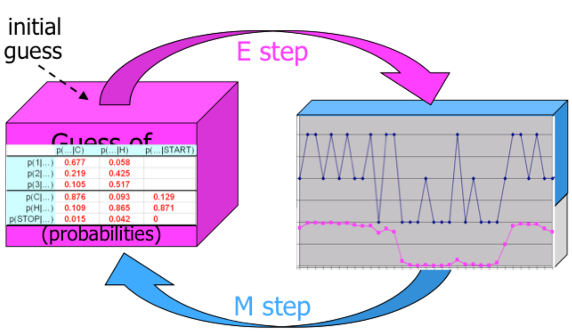
General Idea



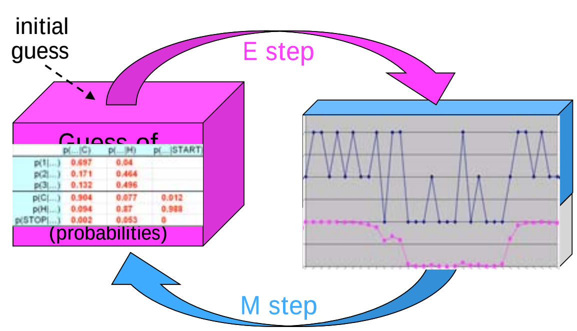
For Hidden Markov Models



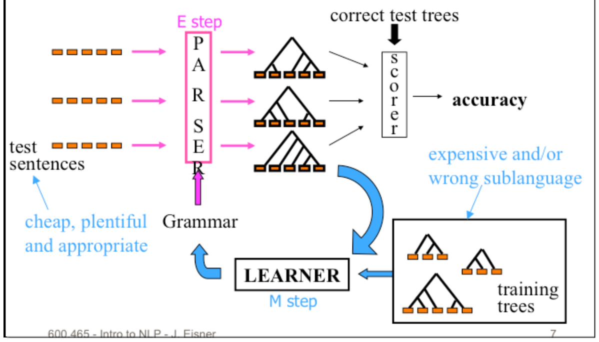
For Hidden Markov Models



For Hidden Markov Models



Grammar Reestimation



EM by Dynamic Programming: Two Versions

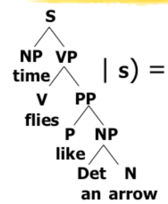
- The Viterbi approximation
  - Expectation: pick the *best* parse of each sentence
  - Maximization: retrain on this best-parsed corpus
  - *Advantage:* Speed!
- Real EM why slower?
  - Expectation: find *all* parses of each sentence
  - Maximization: retrain on *all* parses in proportion to their probability (as if we observed fractional count)
  - *Advantage:* p(training corpus) guaranteed to increase
  - Exponentially many parses, so don't extract them from chart – need some kind of clever counting

Examples of EM

- **Finite-State** case: Hidden Markov Models
  - "forward-backward" or "Baum-Welch" algorithm
  - Applications:
    - explain ice cream in terms of underlying weather sequence
    - explain words in terms of underlying tag sequence
    - explain phoneme sequence in terms of underlying word
    - explain sound sequence in terms of underlying phoneme

    compose these?
- **Context-Free** case: Probabilistic CFGs
  - "inside-outside" algorithm: unsupervised grammar learning!
  - Explain raw text in terms of underlying cx-free parse
    - In practice, local maximum problem gets in the way
    - But *can* **improve** a good starting grammar via raw text
- **Clustering** case: explain points via clusters

Our old friend PCFG

p( | s) = p(s → NP VP | s) * p(NP → time | NP)

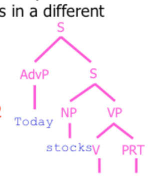
* p(VP → V PP | VP)

* p(V → flies | V) * …

Viterbi reestimation for parsing

- Start with a "pretty good" grammar
  - E.g., it was trained on supervised data (a treebank) that is small, imperfectly annotated, or has sentences in a different style from what you want to parse.
- Parse a corpus of unparsed sentences:

copies of this sentence in the corpus 12 … Today stocks were up … → 12

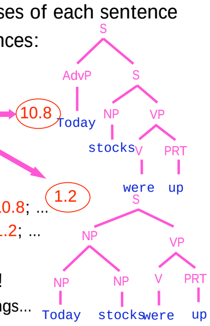


- Reestimate:
  - Collect counts: …; c(S → NP VP) += 12; c(S) += 2*12; …
  - Divide: p(S → NP VP | S) = c(S → NP VP) / c(S)
  - May be wise to smooth

True EM for parsing

- Similar, but now we consider all parses of each sentence
- Parse our corpus of unparsed sentences:

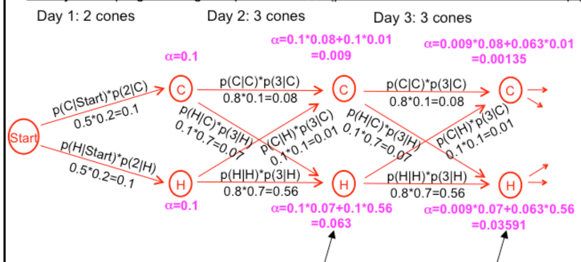
copies of this sentence in the corpus 12 … Today stocks were up … → 10.8, 1.2



- Collect counts **fractionally**:
  - …; c(S → NP VP) += 10.8; c(S) += 2*10.8; …
  - …; c(S → NP VP) += 1.2; c(S) += 1*1.2; …
- But there may be **exponentially** many parses of a length-n sentence!
  - How can we stay fast? Similar to taggings…

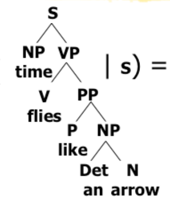
Analogies to α, β in PCFG?

The dynamic programming computation of α. (β is similar but works back from Stop.)



Call these $\alpha_H(2)$ and $\beta_H(2)$ $\alpha_H(3)$ and $\beta_H(3)$

"Inside Probabilities"



$p(\ \cdot \mid s) = p(s \rightarrow \text{NP VP} \mid s) * p(\text{NP} \rightarrow \text{time} \mid \text{NP})$

$* \ p(\text{VP} \rightarrow \text{V PP} \mid \text{VP})$

$* \ p(\text{V} \rightarrow \text{flies} \mid \text{V}) * \ldots$

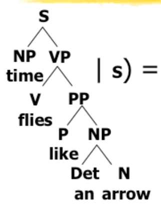
- Sum over all VP parses of "flies like an arrow":

  $\beta_{VP}(1,5) = p(\text{flies like an arrow} \mid \text{VP})$

- Sum over all S parses of "time flies like an arrow":

  $\beta_S(0,5) = p(\text{time flies like an arrow} \mid \text{S})$

Compute β Bottom-Up by CKY



$p(\ \cdot \mid s) = p(s \rightarrow \text{NP VP} \mid s) * p(\text{NP} \rightarrow \text{time} \mid \text{NP})$

$* \ p(\text{VP} \rightarrow \text{V PP} \mid \text{VP})$

$* \ p(\text{V} \rightarrow \text{flies} \mid \text{V}) * \ldots$

$\beta_{VP}(1,5) = p(\text{flies like an arrow} \mid \text{VP}) = \ldots$

$\beta_S(0,5) = p(\text{time flies like an arrow} \mid \text{S})$

$= \beta_{NP}(0,1) * \beta_{VP}(1,5) * p(\text{S} \rightarrow \text{NP VP} \mid \text{S}) + \ldots$

Compute β Bottom-Up by CKY

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 3 Vst 3	NP 10 S 8 S 13			NP 24 NP 24 S 22 S 27 S 27
1		NP 4 VP 4			NP 18 S 21 VP 18
2			P 2 V 5		PP 12 VP 16
3				Det 1	NP 10

1 S → NP VP
6 S → Vst NP
2 S → S PP
1 VP → V NP
2 VP → VP PP
1 NP → Det N
2 NP → NP PP
3 NP → NP NP
0 PP → P NP

Compute β Bottom-Up by CKY

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 2^{-3} Vst 2^{-3}	NP 2^{-10} S 2^{-8} S 2^{-13}			NP 2^{-24} NP 2^{-24} S 2^{-22} S 2^{-27} S 2^{-27} S 2^{-22}
1		NP 2^{-4} VP 2^{-4}			NP 2^{-18} S 2^{-21} VP 2^{-18}
2			P 2^{-2} V 2^{-5}		PP 2^{-12} VP 2^{-16}
3				Det 2^{-1}	NP 2^{-10}

2^{-1} S → NP VP
2^{-6} S → Vst NP
2^{-2} S → S PP
2^{-1} VP → V NP
2^{-2} VP → VP PP
2^{-1} NP → Det N
2^{-2} NP → NP PP
2^{-3} NP → NP NP
2^{-0} PP → P NP

Compute β Bottom-Up by CKY

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 2^{-3} Vst 2^{-3}	NP 2^{-10} S 2^{-8} S 2^{-13}			NP 2^{-24} NP 2^{-24} S 2^{-22} S 2^{-27} S 2^{-27} S 2^{-22} S 2^{-27}
1		NP 2^{-4} VP 2^{-4}			NP 2^{-18} S 2^{-21} VP 2^{-18}
2			P 2^{-2} V 2^{-5}		PP 2^{-12} VP 2^{-16}
3				Det 2^{-1}	NP 2^{-10}

2^{-1} S → NP VP
2^{-6} S → Vst NP
2^{-2} S → S PP
2^{-1} VP → V NP
2^{-2} VP → VP PP
2^{-1} NP → Det N
2^{-2} NP → NP PP
2^{-3} NP → NP NP
2^{-0} PP → P NP

The Efficient Version: Add as we go

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 2^{-3} Vst 2^{-3}	NP 2^{-10} S 2^{-8} S 2^{-13}			NP 2^{-24} NP 2^{-24} S 2^{-22} S 2^{-27} S 2^{-27}
		NP 2^{-4} VP 2^{-4}			NP 2^{-18} S 2^{-21} VP 2^{-18}
2			P 2^{-2} V 2^{-5}		PP 2^{-12} VP 2^{-16}
3				Det 2^{-1}	NP 2^{-10}

2^{-1} S → NP VP
2^{-6} S → Vst NP
2^{-2} S → S PP
2^{-1} VP → V NP
2^{-2} VP → VP PP
2^{-1} NP → Det N
2^{-2} NP → NP PP
2^{-3} NP → NP NP
2^{-0} PP → P NP

The Efficient Version: Add as we go

	time 1	flies 2	like 3	an 4	arrow 5
0	NP 2^{-3} Vst 2^{-3}	NP 2^{-10} S 2^{-8} $+2^{-13}$			NP 2^{-24} $+2^{-24}$ S 2^{-22} $+2^{-27}$ $+2^{-27}$ $+2^{-22}$ $+2^{-27}$
		NP 2^{-4} VP 2^{-4}			NP 2^{-18} S 2^{-21} VP 2^{-18}
2			P 2^{-2} V 2^{-5}		PP 2^{-12} VP 2^{-16}
3				Det 2^{-1}	NP 2^{-10}

$\beta_S(0,2)$

$\beta_S(0,5)$

$\beta_S(0,2)$* $\beta_{PP}(2,5)$*p(S → S PP | S)

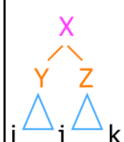
$\beta_{PP}(2,5)$

2^{-1} S → NP VP
2^{-6} S → Vst NP
2^{-2} S → S PP
2^{-1} VP → V NP
2^{-2} VP → VP PP
2^{-1} NP → Det N
2^{-2} NP → NP PP
2^{-3} NP → NP NP
2^{-0} PP → P NP

Compute β probs bottom-up (CKY)

need some initialization up here for the width-1 case

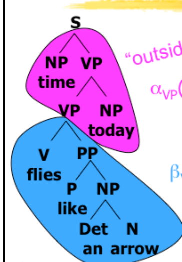
```
for width := 2 to n                  (* build smallest first *)
  for i := 0 to n-width              (* start *)
    let k := i + width               (* end *)
    for j := i+1 to k-1              (* middle *)
      for all grammar rules X → Y Z
        βX(i,k) += p(X → Y Z | X) * βY(i,j) * βZ(j,k)
```



what if you changed + to max?

what if you replaced all rule probabilities by 1?

Inside & Outside Probabilities

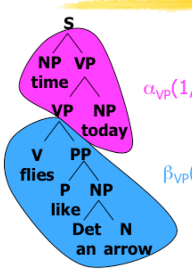


$\alpha_{VP}(1,5)$ = p(time VP today | S)

$\beta_{VP}(1,5)$ = p(flies like an arrow | VP)

$\alpha_{VP}(1,5)$ * $\beta_{VP}(1,5)$
= p(time [VP flies like an arrow] today | S)

Inside & Outside Probabilities



$\alpha_{VP}(1,5)$ = p(time VP today | S)

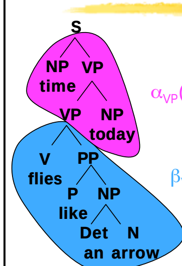
$\beta_{VP}(1,5)$ = p(flies like an arrow | VP)

$\alpha_{VP}(1,5)$ * $\beta_{VP}(1,5)$ / $\beta_S(0,6)$

= $\dfrac{\text{p(time flies like an arrow today \& VP(1,5) | S)}}{\text{p(time flies like an arrow today | S)}}$

= p(VP(1,5) | time flies like an arrow today, S)

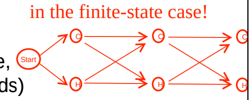
Inside & Outside Probabilities



$\alpha_{VP}(1,5)$ = p(time VP today | S)

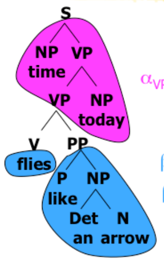
$\beta_{VP}(1,5)$ = p(flies like an arrow | VP)

strictly analogous
to forward-backward
in the finite-state case!



So $\alpha_{VP}(1,5)$ * $\beta_{VP}(1,5)$ / $\beta_S(0,6)$
is probability that there is a VP here,
given all of the observed data (words)

Inside & Outside Probabilities



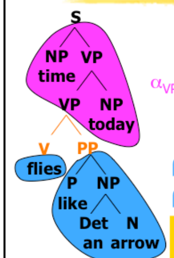
$\alpha_{VP}(1,5)$ = p(time VP today | S)

$\beta_V(1,2)$ = p(flies | V)

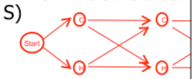
$\beta_{PP}(2,5)$ = p(like an arrow | PP)

So $\alpha_{VP}(1,5) * \beta_V(1,2) * \beta_{PP}(2,5) / \beta_s(0,6)$
is probability that there is VP → V PP here,
given all of the observed data (words) … *or is it?*

Inside & Outside Probabilities



strictly analogous
to forward-backward
in the finite-state case!



$\alpha_{VP}(1,5)$ = p(time VP today | S)

$\beta_V(1,2)$ = p(flies | V)

$\beta_{PP}(2,5)$ = p(like an arrow | PP)

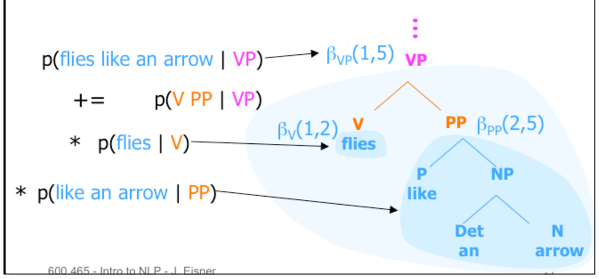
sum prob over all position triples like (1,2,5) to get expected c(VP → V PP); reestimate PCFG!

So $\alpha_{VP}(1,5)$ * p(VP → V PP) * $\beta_V(1,2) * \beta_{PP}(2,5) / \beta_s(0,6)$
is probability that there is VP → V PP here (at 1-2-5),
given all of the observed data (words)

Compute β probs bottom-up
(gradually build up larger blue "inside" regions)

Summary: $\beta_{VP}(1,5)$ += p(V PP | VP) * $\beta_V(1,2)$ * $\beta_{PP}(2,5)$

inside 1,5 inside 1,2 inside 2,5

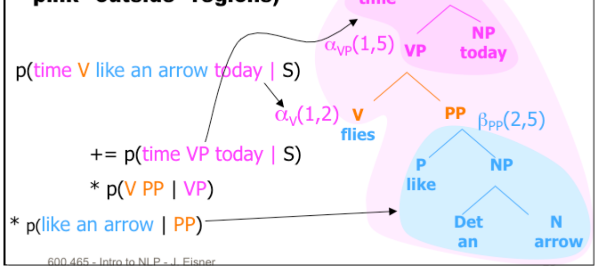


Compute α probs top-down

Summary: $\alpha_V(1,2)$ += p(V PP | VP) * $\alpha_{VP}(1,5)$ * $\beta_{PP}(2,5)$

outside 1,2 outside 1,5 inside 2,5

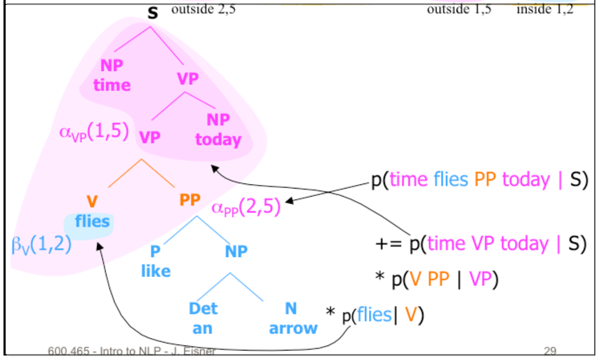
(gradually build up larger pink "outside" regions)



Compute α probs top-down

Summary: $\alpha_{PP}(2,5)$ += p(V PP | VP) * $\alpha_{VP}(1,5)$ * $\beta_V(1,2)$

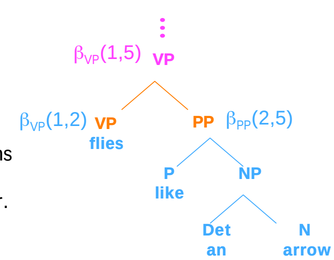
outside 2,5 outside 1,5 inside 1,2



Details:
Compute β probs bottom-up

When you build VP(1,5),
from VP(1,2) and VP(2,5)
during CKY,
increment $\beta_{VP}(1,5)$ by
p(VP → VP PP) *
$\beta_{VP}(1,2) * \beta_{PP}(2,5)$

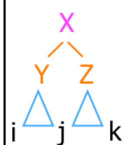
Why? $\beta_{VP}(1,5)$ is total
probability of all derivations
p(flies like an arrow | VP)
and we just found another.
(See earlier slide of CKY chart.)



Details:
Compute β probs bottom-up (CKY)

```
for width := 2 to n                           (* build smallest first *)
  for i := 0 to n-width                       (* start *)
    let k := i + width                        (* end *)
    for j := i+1 to k-1                       (* middle *)
      for all grammar rules X → Y Z
        βX(i,k) += p(X → Y Z) * βY(i,j) * βZ(j,k)
```

$\beta_X(i,k)$ += p(X → Y Z) * $\beta_Y(i,j)$ * $\beta_Z(j,k)$



Details:
Compute α probs top-down (reverse CKY)

for width := ~~2 to n~~ n downto 2 — "unbuild" biggest first (* ~~build smallest first~~ *)

for i := 0 to n-width (* start *)

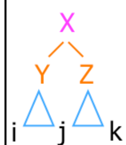
let k := i + width (* end *)

for j := i+1 to k-1 (* middle *)

for all grammar rules X → Y Z

$\alpha_Y(i,j)$ += ???

$\alpha_Z(j,k)$ += ???

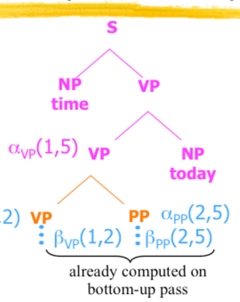


Details:
Compute α probs top-down (reverse CKY)

After computing β during CKY, revisit constits in reverse order (i.e., bigger constituents first).

When you "unbuild" VP(1,5) from VP(1,2) and VP(2,5), increment $\alpha_{VP}(1,2)$ by $\alpha_{VP}(1,5)$ * p(VP → VP PP) * $\beta_{PP}(2,5)$

and increment $\alpha_{PP}(2,5)$ by $\alpha_{VP}(1,5)$ * p(VP → VP PP) * $\beta_{VP}(1,2)$



$\boxed{\alpha_{VP}(1,2) \text{ is total prob of all ways to gen VP(1,2) and all outside words.}}$

Details:
Compute α probs top-down (reverse CKY)

for width := ~~2 to n~~ n downto 2 — "unbuild" biggest first (* ~~build smallest first~~ *)

for i := 0 to n-width (* start *)

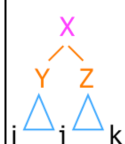
let k := i + width (* end *)

for j := i+1 to k-1 (* middle *)

for all grammar rules X → Y Z

$\alpha_Y(i,j)$ += $\alpha_X(i,k)$ * p(X → Y Z) * $\beta_Z(j,k)$

$\alpha_Z(j,k)$ += $\alpha_X(i,k)$ * p(X → Y Z) * $\beta_Y(i,j)$



What Inside-Outside is Good For

1. As the E step in the EM training algorithm
2. Predicting which nonterminals are probably where
3. Viterbi version as an A* or pruning heuristic
4. As a subroutine within non-context-free models

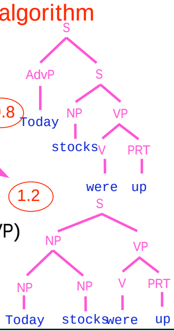
What Inside-Outside is Good For

1. As the E step in the EM training algorithm
   - That's why we just did it

…
12 Today stocks were up
…

10.8

1.2



c(S) += $\sum_{i,j} \alpha_S(i,j)\cdot\beta_S(i,j)/Z$

c(S → NP VP) += $\sum_{i,j,k} \alpha_S(i,k)\cdot p(S \to NP\ VP) \cdot\beta_{NP}(i,j) \cdot\beta_{VP}(j,k)/Z$

where

Z = total prob of all parses = $\beta_S(0,n)$

What Inside-Outside is Good For

1. As the E step in the EM training algorithm
2. Predicting which nonterminals are probably where
   - Posterior decoding of a single sentence
     - Like using $\alpha \cdot \beta$ to pick the most probable tag for each word
     - But can't just pick most probable nonterminal for each span …
       - Wouldn't get a tree! (Not all spans are constituents.)
     - So, find the tree that maximizes expected # correct nonterms.
       - Alternatively, expected # of correct rules.
     - For *each* nonterminal (or rule), at each position:
       - $\alpha \cdot \beta$ tells you the probability that it's correct.
       - For a given tree, sum these probabilities over all positions to get that tree's expected # of correct nonterminals (or rules).
     - **How can we find the tree that maximizes this *sum*?**
       - Dynamic programming – just weighted CKY all over again.
       - But now the weights come from $\alpha \cdot \beta$ (run inside-outside first).

What Inside-Outside is Good For

1. As the E step in the EM training algorithm
2. Predicting which nonterminals are probably where
   - Posterior decoding of a single sentence
   - As soft features in a predictive classifier
     - You want to predict whether the substring from i to j is a name
     - Feature 17 asks whether your parser thinks it's an NP
     - If you're sure it's an NP, the feature fires
       - add $1 \cdot \theta_{17}$ to the log-probability
     - If you're sure it's not an NP, the feature doesn't fire
       - add $0 \cdot \theta_{17}$ to the log-probability
     - But you're not sure!
       - The chance there's an NP there is $p = \alpha_{NP}(i,j) \cdot \beta_{NP}(i,j)/Z$
       - So add $p \cdot \theta_{17}$ to the log-probability

What Inside-Outside is Good For

1. As the E step in the EM training algorithm
2. Predicting which nonterminals are probably where
   - Posterior decoding of a single sentence
   - As soft features in a predictive classifier
   - Pruning the parse forest of a sentence
     - To build a packed forest of all parse trees, keep all backpointer pairs
     - Can be useful for subsequent processing
       - Provides a set of possible parse trees to consider for machine translation, semantic interpretation, or finer-grained parsing
       - But a packed forest has size $O(n^3)$; single parse has size $O(n)$
     - To speed up subsequent processing, prune forest to manageable size
       - Keep only constits with prob $\alpha \cdot \beta / Z \geq 0.01$ of being in true parse
       - Or keep only constits for which $\mu \cdot \nu / Z \geq (0.01 \cdot$ prob of best parse)
         - I.e., do **Viterbi** inside-outside, and keep only constits from parses that are competitive with the best parse (1% as probable)

What Inside-Outside is Good For

1. As the E step in the EM training algorithm
2. Predicting which nonterminals are probably where
3. Viterbi version as an A* or pruning heuristic
   - Viterbi inside-outside uses a semiring with max in place of +
     - Call the resulting quantities ν, μ instead of β, α (as for HMM)
   - Prob of best parse that contains a constituent x is $\nu(x) \cdot \mu(x)$
     - Suppose the best overall parse has prob p. Then all its constituents have $\nu(x) \cdot \mu(x) = p$, and all other constituents have $\nu(x) \cdot \mu(x) < p$.
     - So if we only knew $\nu(x) \cdot \mu(x) < p$, we could skip working on x.
   - In the parsing tricks lecture, we wanted to prioritize or prune x according to "$p(x) \cdot q(x)$." We now see better what q(x) was:
     - p(x) was just the Viterbi inside probability: $p(x) = \nu(x)$
     - q(x) was just an *estimate* of the Viterbi outside prob: $q(x) \approx \mu(x)$.

What Inside-Outside is Good For

1. As the E step in the EM training algorithm
2. Predicting which nonterminals are probably where
3. Viterbi version as an A* or pruning heuristic
   - [continued]
   - q(x) was just an *estimate* of the Viterbi outside prob: $q(x) \approx \mu(x)$.
     - If we could define $q(x) = \mu(x)$ exactly, prioritization would first process the constituents with maximum $\nu \cdot \mu$, which are just the correct ones! So we would do no unnecessary work.
       - But to compute μ (outside pass), we'd first have to **finish parsing** (since μ depends on ν from the inside pass). So this isn't really a "speedup": it tries everything to find out what's necessary.
     - But if we can guarantee $q(x) \geq \mu(x)$, get a safe A* algorithm.
       - We can find such q(x) values by first running Viterbi inside-outside on the sentence using a simpler, faster, approximate grammar …

What Inside-Outside is Good For

1. As the E step in the EM training algorithm
2. Predicting which nonterminals are probably where
3. Viterbi version as an A* or pruning heuristic
   - [continued]
   - If we can guarantee $q(x) \geq \mu(x)$, get a safe A* algorithm.
     - We can find such q(x) values by first running Viterbi inside-outside on the sentence using a faster approximate grammar.

```
0.6 S → NP[sing] VP[sing]
0.3 S → NP[plur] VP[plur]
0   S → NP[sing] VP[plur]
0   S → NP[plur] VP[sing]
0.1 S → VP[stem]
      …
```

max

⇒ `0.6 S → NP[?] VP[?]`

This "coarse" grammar ignores features and makes optimistic assumptions about how they will turn out. Few nonterminals, so fast.

Now define $q_{NP[sing]}(i,j) = q_{NP[plur]}(i,j) = \mu_{NP[?]}(i,j)$

What Inside-Outside is Good For

1. As the E step in the EM training algorithm
2. Predicting which nonterminals are probably where
3. Viterbi version as an A* or pruning heuristic
4. As a subroutine within non-context-free models
   - We've always defined the weight of a parse tree as the sum of its rules' weights.
   - Advanced topic: Can do better by considering additional features of the tree ("non-local features"), e.g., within a log-linear model.
   - CKY no longer works for finding the best parse. ☹
     - Approximate "reranking" algorithm: Using a simplified model that uses only local features, use CKY to find a parse forest. Extract the best 1000 parses. Then re-score these 1000 parses using the full model.
     - Better approximate and exact algorithms: Beyond scope of this course. But they usually call inside-outside or Viterbi inside-outside as a subroutine, often several times (on multiple variants of the grammar, where again each variant can only use local features).