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

Initial guess

Guess of unknown parameters (probabilities)

Observed structure (words, ice cream)

Guess of unknown hidden structure (tags, parses, weather)

E step

M step
For Hidden Markov Models

Guess of unknown parameters

M step

Observed structure (words, ice cream)

E step

Guess of unknown hidden structure (tags, parses, weather)

initial guess

(probabilities)
For Hidden Markov Models

Guess of unknown parameters (probabilities)

M step

E step

initial guess

Guess of hidden structure (tags, parses, weather)

For Hidden Markov Models

initial guess

Guess of unknown parameters (probabilities)

M step

E step

initial guess

Guess of hidden structure (tags, parses, weather)

For Hidden Markov Models

initial guess

Guess of unknown parameters (probabilities)

M step

E step

initial guess

Guess of hidden structure (tags, parses, weather)

For Hidden Markov Models

initial guess

Guess of unknown parameters (probabilities)

M step

E step

initial guess

Guess of hidden structure (tags, parses, weather)

For Hidden Markov Models

initial guess

Guess of unknown parameters (probabilities)

M step

E step

initial guess

Guess of hidden structure (tags, parses, weather)
For Hidden Markov Models

- **E step**: Observation
  - Observed structure (words, ice cream)
  - Initial guess
  - Guess of unknown parameters (probabilities)

- **M step**: Optimization
  - Guess of hidden structure (tags, parses, weather)

| p(...|C) | p(...|H) | p(...|START) |
|-------|---------|-------------|
| p(1|...) | 0.697   | 0.04        |
| p(2|...) | 0.171   | 0.464       |
| p(3|...) | 0.132   | 0.496       |
| p(C|...) | 0.904   | 0.077 | 0.012 |
| p(H|...) | 0.094   | 0.87  | 0.988 |
| p(STOP|...) | 0.002   | 0.053  | 0   |
Grammar Reestimation

Grammar

cheap, plentiful and appropriate

test sentences

M step

LEARNER

training trees

E step

PARSER

correct test trees

scorer

accuracy

expensive and/or wrong sublanguage

600.465 - Intro to NLP - J. Eisner
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
  - **Expectation**: find all parses of each sentence
  - **Maximization**: retrain on all parses in proportion to their probability (as if we observed fractional count)
  - **Advantage**: $p(\text{training corpus})$ 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*

- **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

\[
p(\text{s}) = p(\text{s} \rightarrow \text{NP VP} | \text{s}) \times p(\text{NP} \rightarrow \text{time} | \text{NP}) \times p(\text{VP} \rightarrow \text{V PP} | \text{VP}) \times p(\text{V} \rightarrow \text{flies} | \text{V}) \times \ldots
\]
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:

- Reestimate:
  - Collect counts: ...; \(c(S \rightarrow NP \ VP) += 12\); \(c(S) += 2 \times 12\); ...
  - Divide: \(p(S \rightarrow NP \ VP \mid S) = c(S \rightarrow NP \ VP) / c(S)\)
  - 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:

- Collect counts fractionally:
  - ...; \(c(S \rightarrow \text{NP VP}) = 10.8\); \(c(S) = 2 \times 10.8\); ...
  - ...; \(c(S \rightarrow \text{NP VP}) = 1.2\); \(c(S) = 1 \times 1.2\); ...

# copies of this sentence in the corpus

Today stocks were up

600.465 - Intro to NLP - J. Eisner
Where are the constituents?

\[ p = 0.5 \]
Where are the constituents?

\[ p = 0.1 \]

coal  energy  expert  witness
Where are the constituents?

\[ p = 0.1 \]

col  energy  expert  witness
Where are the constituents?

$p=0.1$

c coherent energy expert witness
Where are the constituents?

$p=0.2$

coal  energy  expert  witness
Where are the constituents?

\[0.5 + 0.1 + 0.1 + 0.1 + 0.2 = 1\]
Where are NPs, VPs, ... ?

NP locations

VP locations

Time flies like an arrow
Where are NPs, VPs, ...?

NP locations

VP locations

(S (NP Time) (VP flies (PP like (NP an arrow))))

p=0.5
Where are NPs, VPs, ... ?

NP locations

VP locations

(S (NP Time flies) (VP like (NP an arrow)))

p=0.3
Where are NPs, VPs, ...?

NP locations

Time flies like an arrow

VP locations

Time flies like an arrow

(S (VP Time (NP (NP flies) (PP like (NP an arrow))))))

p=0.1
Where are NPs, VPs, ...?

NP locations

VP locations

(S (VP (VP Time (NP flies)) (PP like (NP an arrow))))

p=0.1
Where are NPs, VPs, ... ?

NP locations

VP locations

0.5 + 0.3 + 0.1 + 0.1 = 1
How many NPs, VPs, ... in all?

NP locations

Time  flies  like  an  arrow

VP locations

Time  flies  like  an  arrow

0.5 + 0.3 + 0.1 + 0.1 = 1
How many NPs, VPs, ... in all?

NP locations

VP locations

2.1 NPs
(expected)

1.1 VPs
(expected)
Where did the rules apply?

S → NP VP locations
NP → Det N locations

Time flies like an arrow

Time flies like an arrow
Where did the rules apply?

S $\rightarrow$ NP VP locations
NP $\rightarrow$ Det N locations

(S (NP Time) (VP flies (PP like (NP an arrow))))

p=0.5
Where is $S \rightarrow NP \ VP$ substructure?

$S \rightarrow NP \ VP$ locations

$NP \rightarrow$ Det N locations

(S (NP Time flies) (VP like (NP an arrow)))

$p=0.3$
Where is $S \rightarrow NP \ VP$ substructure?

$S \rightarrow NP \ VP$ locations \hspace{1cm} NP $\rightarrow$ Det N locations

(S (VP Time (NP (NP flies) (PP like (NP an arrow))))

$p=0.1$
Where is $S \rightarrow NP \ VP$ substructure?

$S \rightarrow NP \ VP$ locations  \quad NP \rightarrow Det \ N \ locations$

\begin{align*}
(S \ (VP \ (VP \ Time \ (NP \ flies)) \ (PP \ like \ (NP \ an \ arrow))))
\end{align*}

$p=0.1$
Why do we want this info?

- Grammar reestimation by EM method
  - E step collects those **expected counts**
  - M step sets \( p(S \rightarrow NP \ VP \mid S) \leftarrow \frac{\text{count}(S \rightarrow NP \ VP)}{\text{count}(S)} \)

- Minimum Bayes Risk decoding
  - Find a tree that maximizes expected reward, e.g., expected total # of correct constituents
  - CKY-like dynamic programming algorithm
    - The input specifies the **probability of correctness for each possible constituent** (e.g., VP from 1 to 5)
Why do we want this info?

- Soft features of a sentence for other tasks
  - NER system asks: “Is there an NP from 0 to 2?”
    - True answer is 1 (true) or 0 (false)
    - But we return 0.3, averaging over all parses
    - That’s a perfectly good feature value – can be fed as to a CRF or a neural network as an input feature
  - Writing tutor system asks: “How many times did the student use S → NP[singular] VP[plural]?”
    - True answer is in {0, 1, 2, ...}
    - But we return 1.8, averaging over all parses
Similar, but now we consider all parses of each sentence

Parse our corpus of unparsed sentences:

Collect counts fractionally:

- \( c(S \rightarrow NP \ VP) += 10.8 \); \( c(S) += 2 \times 10.8 \); ...
- \( c(S \rightarrow NP \ VP) += 1.2 \); \( c(S) += 1 \times 1.2 \); ...

But there may be exponentially many parses of a length-n sentence!

How can we stay fast? Similar to taggings...

600.465 - Intro to NLP - J. Eisner
**Analogies to $\alpha$, $\beta$ in PCFG?**

The dynamic programming computation of $\alpha$. ($\beta$ is similar but works back from Stop.)

Day 1: 2 cones

Day 2: 3 cones

Day 3: 3 cones

Call these $\alpha_H(2)$ and $\beta_H(2)$, $\alpha_H(3)$ and $\beta_H(3)$.
“Inside Probabilities”

\[
p(S \mid s) = p(S \rightarrow NP \ VP \mid S) \times p(NP \rightarrow time \mid NP) \\
\times p(VP \rightarrow V PP \mid VP) \\
\times p(V \rightarrow flies \mid V) \times \ldots
\]

- Sum over all VP parses of “flies like an arrow”:
  \[
  \beta_{VP}(1,5) = p(flies \ like \ an \ arrow \mid VP)
  \]
- Sum over all S parses of “time flies like an arrow”:
  \[
  \beta_S(0,5) = p(time \ flies \ like \ an \ arrow \mid S)
  \]
Computing $\beta$ Bottom-Up by CKY

\[
p(S \mid s) = p(S \rightarrow NP \ VP \mid S) \times p(NP \rightarrow \text{time} \mid NP) \\
\times p(VP \rightarrow V PP \mid VP) \\
\times p(V \rightarrow \text{flies} \mid V) \times \ldots
\]

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

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

\[
= \beta_{NP}(0,1) \times \beta_{VP}(1,5) \times p(S \rightarrow NP \ VP \mid s) + \ldots
\]
Compute $\beta$ Bottom-Up by CKY

<table>
<thead>
<tr>
<th>time</th>
<th>1</th>
<th>flies</th>
<th>2</th>
<th>like</th>
<th>3</th>
<th>an</th>
<th>4</th>
<th>arrow</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>NP</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vst</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NP</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>P</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>V</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Det</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NP</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. S → NP VP
2. S → Vst NP
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 $\beta$ Bottom-Up by CKY

<table>
<thead>
<tr>
<th>Time</th>
<th>Phrase</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$NP \ 2^{-3}$</td>
<td>$Vst \ 2^{-3}$</td>
</tr>
<tr>
<td>1</td>
<td>$flies \ 2^{-10}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$like \ 2^{-13}$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$an \ 2^{-8}$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$arrow \ 2^{-24}$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$NP \ 2^{-24}$</td>
<td>$NP \ 2^{-24}$</td>
</tr>
</tbody>
</table>

**Rules:**
- $2^{-1} S \rightarrow NP \ VP$
- $2^{-6} S \rightarrow Vst \ NP$
- $2^{-2} S \rightarrow S \ PP$
- $2^{-1} VP \rightarrow V \ NP$
- $2^{-2} VP \rightarrow VP \ PP$
- $2^{-1} NP \rightarrow Det \ N$
- $2^{-2} NP \rightarrow NP \ PP$
- $2^{-3} NP \rightarrow NP \ NP$
- $2^{-0} PP \rightarrow P \ NP$
### Compute $\beta$ Bottom-Up by CKY

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>flies</th>
<th>like</th>
<th>an</th>
<th>arrow</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NP</td>
<td>2$^{-3}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vst</td>
<td>2$^{-3}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>2$^{-8}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>2$^{-13}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>2$^{-2}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>V</td>
<td>2$^{-5}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Det</td>
<td>2$^{-1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>2$^{-8}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Grammar Rules

- $2^{-1} \text{ S} \rightarrow \text{NP VP}$
- $2^{-6} \text{ S} \rightarrow \text{Vst NP}$
- $2^{-2} \text{ S} \rightarrow \text{S PP}$
- $2^{-1} \text{ VP} \rightarrow \text{V NP}$
- $2^{-2} \text{ VP} \rightarrow \text{VP PP}$
- $2^{-1} \text{ NP} \rightarrow \text{Det N}$
- $2^{-2} \text{ NP} \rightarrow \text{NP PP}$
- $2^{-3} \text{ NP} \rightarrow \text{NP NP}$
- $2^{-0} \text{ PP} \rightarrow \text{P NP}$
### The Efficient Version: Add as we go

**Time** 1 flies like an arrow

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NP</strong> $2^{-3}$</td>
<td><strong>NP</strong> $2^{-10}$</td>
<td></td>
<td><strong>NP</strong> $2^{-24}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Vst</strong> $2^{-3}$</td>
<td><strong>S</strong> $2^{-8}$</td>
<td><strong>S</strong> $2^{-13}$</td>
<td><strong>NP</strong> $2^{-24}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>S</strong> $2^{-22}$</td>
<td><strong>S</strong> $2^{-27}$</td>
<td><strong>S</strong> $2^{-27}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Vp</strong> $2^{-4}$</td>
<td></td>
<td><strong>NP</strong> $2^{-4}$</td>
<td></td>
<td><strong>NP</strong> $2^{-18}$</td>
<td></td>
</tr>
<tr>
<td><strong>Vp</strong> $2^{-4}$</td>
<td></td>
<td><strong>S</strong> $2^{-21}$</td>
<td><strong>S</strong> $2^{-21}$</td>
<td><strong>S</strong> $2^{-21}$</td>
<td></td>
</tr>
<tr>
<td><strong>P</strong> $2^{-2}$</td>
<td></td>
<td><strong>S</strong> $2^{-21}$</td>
<td><strong>Vp</strong> $2^{-18}$</td>
<td><strong>Vp</strong> $2^{-18}$</td>
<td></td>
</tr>
<tr>
<td><strong>V</strong> $2^{-5}$</td>
<td></td>
<td><strong>P</strong> $2^{-2}$</td>
<td><strong>P</strong> $2^{-2}$</td>
<td><strong>P</strong> $2^{-2}$</td>
<td></td>
</tr>
<tr>
<td><strong>Det</strong> $2^{-1}$</td>
<td></td>
<td><strong>Vp</strong> $2^{-16}$</td>
<td><strong>Vp</strong> $2^{-16}$</td>
<td><strong>Vp</strong> $2^{-16}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Det</strong> $2^{-1}$</td>
<td></td>
<td><strong>Det</strong> $2^{-1}$</td>
<td><strong>Det</strong> $2^{-1}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Det</strong> $2^{-1}$</td>
<td><strong>Det</strong> $2^{-1}$</td>
<td><strong>Det</strong> $2^{-1}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Det</strong> $2^{-1}$</td>
<td><strong>Det</strong> $2^{-1}$</td>
<td><strong>Det</strong> $2^{-1}$</td>
<td></td>
</tr>
<tr>
<td><strong>N</strong> $2^{-8}$</td>
<td></td>
<td><strong>N</strong> $2^{-8}$</td>
<td><strong>N</strong> $2^{-8}$</td>
<td><strong>N</strong> $2^{-8}$</td>
<td></td>
</tr>
</tbody>
</table>

#### Production Rules

- $2^{-1} \ S \rightarrow \ NP \ VP$
- $2^{-6} \ S \rightarrow \ Vst \ NP$
- $2^{-2} \ S \rightarrow \ S \ PP$
- $2^{-1} \ VP \rightarrow \ V \ NP$
- $2^{-2} \ VP \rightarrow \ VP \ PP$
- $2^{-1} \ NP \rightarrow \ Det \ N$
- $2^{-2} \ NP \rightarrow \ NP \ PP$
- $2^{-3} \ NP \rightarrow \ NP \ NP$
- $2^{-0} \ PP \rightarrow \ P \ NP$
### The Efficient Version: Add as we go

<table>
<thead>
<tr>
<th>Time</th>
<th>NP</th>
<th>Vst</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(2^{-3})</td>
<td>(2^{-3})</td>
<td>(2^{-8}) + (2^{-13})</td>
</tr>
<tr>
<td>1</td>
<td>(2^{-3})</td>
<td>(2^{-10})</td>
<td>(2^{-8})</td>
</tr>
<tr>
<td>2</td>
<td>(2^{-4})</td>
<td>(2^{-4})</td>
<td>(2^{-2})</td>
</tr>
<tr>
<td>3</td>
<td>(2^{-1})</td>
<td>(2^{-8})</td>
<td>(2^{-1})</td>
</tr>
<tr>
<td>4</td>
<td>(2^{-8})</td>
<td>(2^{-10})</td>
<td>(2^{-12})</td>
</tr>
</tbody>
</table>

- \(\beta_S(0,2)\) * \(\beta_{PP}(2,5)\) * \(p(S \rightarrow S PP \mid S)\)

- \(2^{-1} S \rightarrow NP \ VP\)
- \(2^{-6} S \rightarrow Vst \ NP\)
- \(2^{-2} S \rightarrow S PP\)
- \(2^{-1} VP \rightarrow V NP\)
- \(2^{-2} VP \rightarrow VP PP\)
- \(2^{-1} NP \rightarrow Det \ N\)
- \(2^{-2} NP \rightarrow NP PP\)
- \(2^{-3} NP \rightarrow NP NP\)
- \(2^{-0} PP \rightarrow P NP\)
Compute $\beta$ 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 $X \rightarrow Y Z$

$$\beta_X(i,k) \; \triangleq \; p(X \rightarrow Y Z | X) \star \beta_Y(i,j) \star \beta_Z(j,k)$$

what if you changed $+$ to $\max$?

need some initialization up here for the width-1 case

what if you replaced all rule probabilities by 1?
Inside & Outside Probabilities

\[ \alpha_{VP}(1,5) = p(\text{time VP today} \mid S) \]

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

\[ \alpha_{VP}(1,5) \ast \beta_{VP}(1,5) = p(\text{time [VP flies like an arrow] today} \mid S) \]
Inside & Outside Probabilities

\[ \alpha_{VP}(1,5) = p(\text{time VP today} \mid S) \]

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

\[ \alpha_{VP}(1,5) \ast \beta_{VP}(1,5) / \beta_S(0,6) = \frac{p(\text{time flies like an arrow today} \& VP(1,5) \mid S)}{p(\text{time flies like an arrow today} \mid S)} \]

\[ = p(\text{VP}(1,5) \mid \text{time flies like an arrow today, } S) \]
Inside & Outside Probabilities

\[ \alpha_{VP}(1,5) = p(\text{time VP today} \mid S) \]

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

So \( \alpha_{VP}(1,5) \times \beta_{VP}(1,5) / \beta_s(0,6) \)

is probability that there is a VP here,
given all of the observed data (words)

strictly analogous
to forward-backward
in the finite-state case!
Inside & Outside Probabilities

\[ \alpha_{VP}(1,5) = p(\text{time VP today} \mid S) \]

\[ \beta_V(1,2) = p(\text{flies} \mid V) \]

\[ \beta_{PP}(2,5) = p(\text{like an arrow} \mid PP) \]

So \( \alpha_{VP}(1,5) \times \beta_V(1,2) \times \beta_{PP}(2,5) / \beta_s(0,6) \) is probability that there is VP \( \rightarrow \) V PP here, given all of the observed data (words) ... or is it?
Inside & Outside Probabilities

\[ \alpha_{VP}(1,5) = p(\text{time VP today} \mid S) \]

\[ \beta_V(1,2) = p(\text{flies} \mid V) \]

\[ \beta_{PP}(2,5) = p(\text{like an arrow} \mid PP) \]

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

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

strictly analogous to forward-backward in the finite-state case!
Compute $\beta$ probs bottom-up
(gradually build up larger blue “inside” regions)

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

\[
p(flies \ like \ an \ arrow \mid VP) 
  += p(V \ PP \mid VP) \cdot p(flies \mid V) 
  \cdot p(like \ an \ arrow \mid PP)
\]
Compute $\alpha$ probs top-down

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

(gradually build up larger pink “outside” regions)

$p(\text{time } V \text{ like an arrow today} | S)$

$+= p(\text{time } VP \text{ today} | S)$

$\ast p(V^{PP} | VP)$

$\ast p(\text{like an arrow} | PP)$

600.465 - Intro to NLP - J. Eisner
Compute $\alpha$ probs top-down

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

$p(time \ flies \ PP \ today | S)$

$+= p(time \ VP \ today | S)$

$\times p(V \ PP | VP)$

$\times p(flies | V)$
Details:
Compute $\beta$ 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

$$p(VP \rightarrow VP \ PP) \ast \beta_{VP}(1,2) \ast \beta_{PP}(2,5)$$

Why? $\beta_{VP}(1,5)$ is total probability of all derivations
$p(flies \ like \ an \ arrow \mid VP)$ and we just found another.
(See earlier slide of CKY chart.)
Details:
Compute $\beta$ probs bottom-up (CKY)

for width := 2 to n
    for i := 0 to n-width
        let k := i + width
        for j := i+1 to k-1
            for all grammar rules $X \rightarrow Y Z$
                $\beta_X(i,k) += p(X \rightarrow Y Z) \cdot \beta_Y(i,j) \cdot \beta_Z(j,k)$
Details:
Compute $\alpha$ probs top-down (reverse CKY)

for width := 2 to n
  for i := 0 to n-width
    let k := i + width
    for j := i+1 to k-1
      for all grammar rules $X \rightarrow Y Z$
        $\alpha_Y(i,j) += ???$
        $\alpha_Z(j,k) += ???$
Details:
Compute $\alpha$ probs **top-down** (reverse CKY)

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

When you “unbuild” $\text{VP}(1,5)$ from $\text{VP}(1,2)$ and $\text{VP}(2,5)$, increment $\alpha_{\text{VP}}(1,2)$ by

$$\alpha_{\text{VP}}(1,5) \times p(\text{VP} \rightarrow \text{VP PP}) \times \beta_{\text{PP}}(2,5)$$

and increment $\alpha_{\text{PP}}(2,5)$ by

$$\alpha_{\text{VP}}(1,5) \times p(\text{VP} \rightarrow \text{VP PP}) \times \beta_{\text{VP}}(1,2)$$

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

Compute $\alpha$ probs top-down (reverse CKY)

for width := 2 to n

for i := 0 to n-width

let k := i + width

for j := i+1 to k-1

for all grammar rules $X \rightarrow Y Z$

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

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

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

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

where
\[ Z = \text{total prob of all parses} = \beta_S(0,n) \]
Does Unsupervised Learning Work?

- Merialdo (1994)
  - “The paper that freaked me out ...”
    - Kevin Knight

- EM always improves likelihood
- But it sometimes hurts accuracy
- Why?#@!?
Does Unsupervised Learning Work?

Sometimes, I have some unlabeled data, and I want to put labels on it.

UNSUPERVISED LEARNING, AS SHE IS IMPLEMENTED.

So I write down a generative model, and then tell the data to find parameters that explain the data to me. And if I am not satisfied with the likelihood of this explanation, I tell the data to do it again until I am.

Wow, that sucks for the data.

I know, right? It's not the data's fault that I was too lazy to label it, right?

It seems like there are some deeper issues. Sometimes, most of the variation in the data comes from phenomena that are irrelevant to your desired labeling scheme. For example, the data might use its parameters to explain its semantics, when all you care about is its syntactic properties.

Why do we have to put labels on our data at all? Can't we just appreciate our data for what it is, and recognize that each datum is a unique and precious snowflake? Guys, all I'm saying, is maybe with a little supervision, our data can grow up to be whatever it wants to be!
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 $\mu \cdot v / Z \geq (0.01 \cdot \text{prob of best parse})$
         - 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 \( \nu,\mu \) instead of \( \beta,\alpha \) (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 \( \mu \) (outside pass), we’d first have to **finish parsing** (since \( \mu \) depends on \( \nu \) 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 insideoutside on the sentence using a faster approximate grammar.

\[
\begin{align*}
0.6 & \quad S \rightarrow \text{NP}[\text{sing}] \quad \text{VP}[\text{sing}] \\
0.3 & \quad S \rightarrow \text{NP}[\text{plur}] \quad \text{VP}[\text{plur}] \\
0 & \quad S \rightarrow \text{NP}[\text{sing}] \quad \text{VP}[\text{plur}] \\
0 & \quad S \rightarrow \text{NP}[\text{plur}] \quad \text{VP}[\text{sing}] \\
0.1 & \quad S \rightarrow \text{VP}[\text{stem}] \\
\end{align*}
\]

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

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