Outline for Today

- Word error rate (WER) computation
- Training
  - Baum-Welch = EM = Forward Backward
    - Detailed example in slides appendix
  - How we train LVCSR systems in practice
- Advanced decoding
Administrative items

- Homework 1 due by 11:59pm tonight on Gradescope
- Homework 2 released tonight (due in 2 weeks)
- Project handout released tonight
  - We will compile and post to piazza project ideas over the next 1-2 weeks
  - Proposals due May 1
- Background survey released today. Complete by Friday (part of class participation grade)
Outline for Today

• **Word error rate (WER) computation**

• **Training**
  • Baum-Welch = EM = Forward Backward
    • Detailed example in slides appendix
  • How we train LVCSR systems in practice

• **Advanced decoding**
Evaluation

- How to evaluate the word string output by a speech recognizer?
Word Error Rate

- Word Error Rate =
  \[
  100 \left( \text{Insertions} + \text{Substitutions} + \text{Deletions} \right) \]
  \[
  \text{Total Word in Correct Transcript}
  
- Alignment example:
  
REF: portable **** PHONE UPSTAIRS last night so
HYP: portable FORM OF STORES last night so
Eval: I S S

- WER = 100 (1+2+0)/6 = 50%
NIST sctk scoring software: Computing WER with sclite

http://www.nist.gov/speech/tools/

- Sclite aligns a hypothesized text (HYP) (from the recognizer) with a correct or reference text (REF) (human transcribed)

id: (2347-b-013)
Scores: (#C #S #D #I) 9 3 1 2

REF: was an engineer SO I i was always with **** **** MEN UM and they
HYP: was an engineer ** AND i was always with THEM THEY ALL THAT and they

Eval: D S I I S S
### Sclite output for error analysis

<table>
<thead>
<tr>
<th>CONFUSION PAIRS</th>
<th>Total</th>
<th>With &gt;= 1 occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(972)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(972)</td>
</tr>
<tr>
<td>1: 6 - &gt; (%hesitation) ==&gt; on</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2: 6 - &gt; the ==&gt; that</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3: 5 - &gt; but ==&gt; that</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4: 4 - &gt; a ==&gt; the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5: 4 - &gt; four ==&gt; for</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6: 4 - &gt; in ==&gt; and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7: 4 - &gt; there ==&gt; that</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8: 3 - &gt; (%hesitation) ==&gt; and</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9: 3 - &gt; (%hesitation) ==&gt; the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10: 3 - &gt; (a-) ==&gt; i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11: 3 - &gt; and ==&gt; i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12: 3 - &gt; and ==&gt; in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13: 3 - &gt; are ==&gt; there</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14: 3 - &gt; as ==&gt; is</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15: 3 - &gt; have ==&gt; that</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16: 3 - &gt; is ==&gt; this</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Better metrics than WER?

• WER has been useful
• But should we be more concerned with meaning ("semantic error rate")?
  • Good idea, but hard to agree on
  • Has been applied in dialogue systems, where desired semantic output is more clear
Outline for Today

- Word error rate (WER) computation
- **Training**
  - Baum-Welch = EM = Forward Backward
    - Detailed example in slides appendix
  - How we train LVCSR systems in practice
- Advanced decoding
HMM for ice cream

A: Transition parameters
B: Observation/Emission parameters

- Transition parameters:
  - Start state to Hot: 0.2
  - Hot to Cold: 0.6
  - Hot to Hot: 0.7
  - Cold to Hot: 0.3
  - Cold to Cold: 0.4

- Observation/Emission parameters:
  - B₁:
    - P(1|HOT) = 0.2
    - P(2|HOT) = 0.4
    - P(3|HOT) = 0.4
  - B₂:
    - P(1|COLD) = 0.5
    - P(2|COLD) = 0.4
    - P(3|COLD) = 0.1
The Learning Problem

Learning: Given an observation sequence \( O \) and the set of possible states in the HMM, learn the HMM parameters \( A \) and \( B \).

- Baum-Welch = Forward-Backward Algorithm (Baum 1972)
- Is a special case of the EM or Expectation-Maximization algorithm (Dempster, Laird, Rubin)
- The algorithm will let us train the transition probabilities \( A = \{a_{ij}\} \) and the emission probabilities \( B = \{b_i(o_t)\} \) of the HMM
The Learning Problem

- Baum-Welch / EM enables maximum likelihood training of (A,B)
- In practice we do not train A
  - Why?
HMM for the digit recognition task

Lexicon

<table>
<thead>
<tr>
<th>Digit</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>w ah n</td>
</tr>
<tr>
<td>two</td>
<td>t uw</td>
</tr>
<tr>
<td>three</td>
<td>th r iy</td>
</tr>
<tr>
<td>four</td>
<td>f ao r</td>
</tr>
<tr>
<td>five</td>
<td>f ay v</td>
</tr>
<tr>
<td>six</td>
<td>s ih k s</td>
</tr>
<tr>
<td>seven</td>
<td>s eh v ax n</td>
</tr>
<tr>
<td>eight</td>
<td>e y t</td>
</tr>
<tr>
<td>nine</td>
<td>n ay n</td>
</tr>
<tr>
<td>zero</td>
<td>z iy r ow</td>
</tr>
<tr>
<td>oh</td>
<td>ow</td>
</tr>
</tbody>
</table>
The Learning Problem

- Baum-Welch / EM implicitly does “soft assignment” of hidden states when updating observation/emission model parameters
- In practice we use “hard assignment” / Viterbi training
Estimating hidden states in training

- Updating parameters in EM/Soft Assignment
  - $B_{ahm} \sim 0*o_1 + 0.15*o_2 + 0.5*o_3 + 0.05*o_4$
- Updating parameters with Viterbi/Hard Assignment
  - $B_{ahm} \sim o_3$
Typical training procedure in LVCSR

• Generate a forced alignment with existing model
  • Viterbi decoding with a very constrained prior (the transcript)
  • Assigns observations to HMM states
• Create new observation models from update alignments
• Iteratively repeat the above steps, occasionally introducing a more complex observation model or adding more difficult training examples
Outline for Today

- Word error rate (WER) computation
- Training
  - Baum-Welch = EM = Forward Backward
    - Detailed example in slides appendix
  - How we train LVCSR systems in practice
- Advanced decoding
Advanced Search (= Decoding)

- How to weight the AM and LM
- Speeding things up: Viterbi beam decoding
- Multipass decoding
  - N-best lists
  - Lattices
  - Word graphs
  - Meshes/confusion networks
- Finite State Methods
What we are searching for

- Given Acoustic Model (AM) and Language Model (LM):

\[
\hat{W} = \arg\max_{W \in L} P(O | W) P(W)
\]
Combining Acoustic and Language Models

• We don’t actually use equation (1)

\[
\hat{W} = \arg\max_{W \in L} P(O \mid W) P(W)
\]

• AM underestimates acoustic probability
  • Why? Bad independence assumptions
  • Intuition: we compute (independent) AM probability estimates; but if we could look at context, we would assign a much higher probability. So we are underestimating
  • We do this every 10 ms, but LM only every word.
  • Besides: AM isn’t a true probability

• AM and LM have vastly different dynamic ranges
Language Model Scaling Factor

- Solution: add a language model weight (also called language weight LW or language model scaling factor LMSF)

\[ \hat{W} = \arg \max_{W \in L} P(O|W)P(W)^{LMSF} \]

\[ = \arg \max \log P(O|W) + LMSF \log P(W) \]

- Value determined empirically, is positive (why?)
- Often in the range 10 +/- 5.
- Kaldi uses an acoustic model scaling factor instead, but it achieves the same effect
Language Model Scaling Factor

- As LMSF is increased:
  - More deletion errors (since increase penalty for transitioning between words)
  - Fewer insertion errors
  - Need wider search beam (since path scores larger)
  - Less influence of acoustic model observation probabilities
Word Insertion Penalty

- But LM prob $P(W)$ also functions as penalty for inserting words
  - Intuition: when a uniform language model (every word has an equal probability) is used, LM prob is a $1/V$ penalty multiplier taken for each word
  - Each sentence of $N$ words has penalty $N/V$
  - If penalty is large (smaller LM prob), decoder will prefer fewer longer words
  - If penalty is small (larger LM prob), decoder will prefer more shorter words
- When tuning LM for balancing AM, side effect of modifying penalty
- So we add a separate word insertion penalty to offset

$$
\hat{W} = \arg\max_{W} P(O | W) P(W)^{LMSF} WIP^{N(W)}
$$
Word Insertion Penalty

- Controls trade-off between insertion and deletion errors
  - As penalty becomes larger (more negative)
  - More deletion errors
  - Fewer insertion errors
- Acts as a model of effect of length on probability
  - But probably not a good model (geometric assumption probably bad for short sentences)
Log domain

- We do everything in log domain
- So final equation:

\[
\hat{W} = \arg\max_{W \in L} \log P(O \mid W) + L_{MSF} \log P(W) + N \log WIP
\]
Speeding things up

• Viterbi is $O(N^2T)$, where $N$ is total number of HMM states, and $T$ is length
• This is too large for real-time search
• A ton of work in ASR search is just to make search faster:
  • Beam search (pruning)
  • Fast match
  • Tree-based lexicons
Beam search

- Instead of retaining all candidates (cells) at every time frame

- Use a threshold $T$ to keep subset:
  - At each time $t$
  - Identify state with lowest cost $D_{\text{min}}$
  - Each state with cost $> D_{\text{min}} + T$ is discarded ("pruned") before moving on to time $t+1$
  - Unpruned states are called the active states
Viterbi Beam Search

Slide from John-Paul Hosom
Viterbi Beam search

- Most common search algorithm for LVCSR
- Time-synchronous
  - Comparing paths of equal length
- Two different word sequences W1 and W2:
  - We are comparing $P(W1|O_{0t})$ and $P(W2|O_{0t})$
  - Based on same partial observation sequence $O_{0t}$
  - So denominator is same, can be ignored
- Time-asynchronous search ($A^*$) is harder
Viterbi Beam Search

- Empirically, beam size of 5-10% of search space
- Thus 90-95% of HMM states don’t have to be considered at each time \( t \)
- Vast savings in time.
On-line processing

- Problem with Viterbi search
  - Doesn’t return best sequence til final frame

- This delay is unreasonable for many applications.

- On-line processing
  - usually smaller delay in determining answer
  - at cost of always increased processing time.
On-line processing

- At every time interval I (e.g. 1000 msec or 100 frames):
  - At current time $t_{curr}$, for each active state $q_{tcurr}$, find best path $P(q_{tcurr})$ that goes from from $t0$ to $t_{curr}$ (using backtrace ($\psi$))
  - Compare set of best paths $P$ and find last time $t_{match}$ at which all paths $P$ have the same state value at that time

- If $t_{match}$ exists {
  Output result from $t0$ to $t_{match}$
  Reset/Remove $\psi$ values until $t_{match}$
  Set $t0$ to $t_{match}+1$
}

- Efficiency depends on interval I, beam threshold, and how well the observations match the HMM.

Slide from John-Paul Hosom
• Example (Interval = 4 frames):

At time 4, all best paths for all states A, B, and C have state B in common at time 2. So, $t_{\text{match}} = 2$.

Now output states BB for times 1 and 2, because no matter what happens in the future, this will not change. Set $t_0$ to 3.
On-line processing

- Now $t_{match} = 7$, so output from $t=3$ to $t=7$: BBABB, then set $t_0$ to 8.

- If $T=8$, then output state with best $\delta_8$, for example C. Final result (obtained piece-by-piece) is then BBBBABBBC

Slide from John-Paul Hosom
Problems with Viterbi

- It’s hard to integrate sophisticated knowledge sources
  - Trigram grammars
  - Parser-based or Neural Network LM
    - long-distance dependencies that violate dynamic programming assumptions
  - Knowledge that isn’t left-to-right
    - Following words can help predict preceding words

Solutions

- Return multiple hypotheses and use smart knowledge to rescore them
- Use a different search algorithm, A* Decoding (=Stack decoding)
Multipass Search

Speech Input → Simple Knowledge Source → N-Best Decoder → N-Best List

3. Alice was beginning to get...
3. Every happy family
3. In a hole in the ground...
3. If music be the food of love...
3. If music be the foot of dove...

Smarter Knowledge Source → Rescoring

1-Best Utterance

If music be the food of love
Ways to represent multiple hypotheses

- **N-best list**
  - Instead of single best sentence (word string), return ordered list of N sentence hypotheses

- **Word lattice**
  - Compact representation of word hypotheses and their times and scores

- **Word graph**
  - FSA representation of lattice in which times are represented by topology
Another Problem with Viterbi

- The forward probability of observation given word string

\[ P(O|W) = \sum_{S \in S^T_1} P(O, S|W) \]

- The Viterbi algorithm makes the “Viterbi Approximation”

\[ P(O|W) \approx \max_{S \in S^T_1} P(O, S|W) \]

- Approximates P(O|W)
  - with P(O|best state sequence)
Solving the best-path-not-best-words problem

- Viterbi returns best path (state sequence) not best word sequence
  - Best path can be very different than best word string if words have many possible pronunciations
- Two solutions
  - Modify Viterbi to sum over different paths that share the same word string.
    - Do this as part of N-best computation
      - Compute N-best word strings, not N-best phone paths
  - Use a different decoding algorithm (A*) that computes true Forward probability.
## Sample N-best list

<table>
<thead>
<tr>
<th>Rank</th>
<th>Path</th>
<th>AM logprob</th>
<th>LM logprob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>it’s an area that’s naturally sort of mysterious</td>
<td>-7193.53</td>
<td>-20.25</td>
</tr>
<tr>
<td>2.</td>
<td>that’s an area that’s naturally sort of mysterious</td>
<td>-7192.28</td>
<td>-21.11</td>
</tr>
<tr>
<td>3.</td>
<td>it’s an area that’s not really sort of mysterious</td>
<td>-7221.68</td>
<td>-18.91</td>
</tr>
<tr>
<td>4.</td>
<td>that scenario that’s naturally sort of mysterious</td>
<td>-7189.19</td>
<td>-22.08</td>
</tr>
<tr>
<td>5.</td>
<td>there’s an area that’s naturally sort of mysterious</td>
<td>-7198.35</td>
<td>-21.34</td>
</tr>
<tr>
<td>6.</td>
<td>that’s an area that’s not really sort of mysterious</td>
<td>-7220.44</td>
<td>-19.77</td>
</tr>
<tr>
<td>7.</td>
<td>the scenario that’s naturally sort of mysterious</td>
<td>-7205.42</td>
<td>-21.50</td>
</tr>
<tr>
<td>8.</td>
<td>so it’s an area that’s naturally sort of mysterious</td>
<td>-7195.92</td>
<td>-21.71</td>
</tr>
<tr>
<td>9.</td>
<td>that scenario that’s not really sort of mysterious</td>
<td>-7217.34</td>
<td>-20.70</td>
</tr>
<tr>
<td>10.</td>
<td>there’s an area that’s not really sort of mysterious</td>
<td>-7226.51</td>
<td>-20.01</td>
</tr>
</tbody>
</table>
N-best lists

- Again, we don’t want the N-best paths
- That would be trivial
  - Store N values in each state cell in Viterbi trellis instead of 1 value
- But:
  - Most of the N-best paths will have the same word string
    - Useless!!!
  - It turns out that a factor of N is too much to pay
Computing N-best lists

- In the worst case, an admissible algorithm for finding the N most likely hypotheses is exponential in the length of the utterance.
- For example, if AM and LM score were nearly identical for all word sequences, we must consider all permutations of word sequences for whole sentence (all with the same scores).
- But of course if this is true, can’t do ASR at all!
Computing N-best lists

- Instead, various non-admissible algorithms:
  - (Viterbi) Exact N-best
  - (Viterbi) Word Dependent N-best
- And one admissible
  - A* N-best
Word-dependent (‘bigram’) N-best

• Intuition:
  • Instead of each state merging all paths from start of sentence
  • We merge all paths that share the same previous word

• Details:
  • This will require us to do a more complex traceback at the end of sentence to generate the N-best list
Word-dependent (‘bigram’) N-best

- At each state preserve total probability for each of k $<< N$ previous words
  - K is 3 to 6; N is 100 to 1000
- At end of each word, record score for each previous word hypothesis and name of previous word
  - So each word ending we store “alternatives”
- But, like normal Viterbi, pass on just the best hypothesis
- At end of sentence, do a traceback
  - Follow backpointers to get 1-best
  - But as we follow pointers, put on a queue the alternate words ending at same point
  - On next iteration, pop next best
Word Lattice

SO IT'S
IT'S
THERE'S
THAT'S

AN
AREA
THAT'S
SCENARIO
THE
THAT

NATURALLY
NOT
REALLY
SORT
OF

MYSTERIOUS
Word Graph

- Timing information removed

Result is a WFST

Natural extension to N-gram language model
Converting word lattice to word graph

- Word lattice can have range of possible end frames for word
- Create an edge from \((w_i, t_i)\) to \((w_j, t_j)\) if \(t_{j-1}\) is one of the end-times of \(w_i\)
Lattices

• Some researchers are careful to distinguish between word graphs and word lattices
• But we’ll follow convention in using “lattice” to mean both word graphs and word lattices.

• Two facts about lattices:
  • Density: the number of word hypotheses or word arcs per uttered word
  • Lattice error rate (also called “lower bound error rate”): the lowest word error rate for any word sequence in lattice
    • Lattice error rate is the “oracle” error rate, the best possible error rate you could get from rescoring the lattice.
    • We can use this as an upper bound
Posterior lattices

- We don’t actually compute posteriors:

\[ \hat{W} = \arg\max_{W \in \mathcal{L}} \frac{P(O|W)P(W)}{P(O)} = \arg\max_{W \in \mathcal{L}} P(O|W)P(W) \]

- Why do we want posteriors?
  - Without a posterior, we can choose best hypothesis, but we can’t know how good it is!
  - In order to compute posterior, need to
    - Normalize over all different word hypothesis at a time
  - Align all the hypotheses, sum over all paths passing through word
Mesh = Sausage = pinched lattice
Summary: one-pass vs. multipass

- Potential problems with multipass
  - Can’t use for real-time (need end of sentence)
    - (But can keep successive passes really fast)
  - Each pass can introduce inadmissible pruning
    - (But one-pass does the same w/beam pruning and fastmatch)

- Why multipass
  - Very expensive KSs. (NL parsing, higher-order n-gram, etc.)
  - Spoken language understanding: N-best perfect interface
  - Research: N-best list very powerful offline tools for algorithm development
  - N-best lists needed for discriminant training (MMIE, MCE) to get rival hypotheses
Weighted Finite State Transducers for ASR

- The modern paradigm for ASR decoding
- Used by Kaldi
- Weighted finite state automaton that transduces an input sequence to an output sequence
Simple State Machine

On

Flip switch down

Flip switch up

Off
Weighted Finite State Acceptors
Weighted Finite State Transducers
WFST Algorithms

**Composition**: combine transducers at different levels. If $G$ is a finite state grammar and $P$ is a pronunciation dictionary, $P \circ G$ transduces a phone string to word strings allowed by the grammar.

**Determinization**: Ensures each state has no more than one output transition for a given input label.

**Minimization**: transforms a transducer to an equivalent transducer with the fewest possible states and transitions.
WFST-based decoding in Kaldi: HCLG

- Represent the following components as WFSTs:
  - H: HMM structure
  - C: Phonetic context dependency
  - L: Lexicon (Pronunciation dictionary)
  - G: Grammar (Language model)
- The decoding network is defined by their composition:
  $$H \circ C \circ L \circ G$$

- Successively determinize and combine the component transducers, then minimize the final network
G (Language model)

0
__jim/1.386__
jill/0.693
__bill/1.386__

1
read/0.400
wrote/1.832
fled/1.771

2/0
L (Pronunciation dictionary)
min(det(L o G))
Advanced Search (= Decoding)

- How to weight the AM and LM
- Speeding things up: Viterbi beam decoding
- Multipass decoding
  - N-best lists
  - Lattices
  - Word graphs
  - Meshes/confusion networks
- Finite State Methods
  - For a more thorough introduction to WFST decoding in Kaldi:
Appendix: Baum-Welch Training
Input to Baum-Welch

- $O$ unlabeled sequence of observations
- $Q$ vocabulary of hidden states

For ice-cream task
- $O = \{1, 3, 2, \ldots, \}$
- $Q = \{H, C\}$
Starting out with Observable Markov Models

• How to train?
• Run the model on observation sequence O.
• Since it’s not hidden, we know which states we went through, hence which transitions and observations were used.

• Given that information, training:
  • \( B = \{ b_k(o_t) \} \): Since every state can only generate one observation symbol, observation likelihoods B are all 1.0
  • \( A = \{ a_{ij} \} \):

\[
a_{ij} = \frac{C(i \rightarrow j)}{\sum_{q \in Q} C(i \rightarrow q)}
\]
Extending Intuition to HMMs

- For HMM, cannot compute these counts directly from observed sequences
- Baum-Welch intuitions:
  - Iteratively estimate the counts.
    - Start with an estimate for $a_{ij}$ and $b_k$, iteratively improve the estimates
  - Get estimated probabilities by:
    - computing the forward probability for an observation
    - dividing that probability mass among all the different paths that contributed to this forward probability
The Backward algorithm

- We define the backward probability as follows:

\[ t(i) = P(o_{t+1}, o_{t+2}, \ldots o_T, | q_t = i, F) \]

- This is the probability of generating partial observations \( O_{t+1}^T \) from time \( t+1 \) to the end, given that the HMM is in state \( i \) at time \( t \) and of course given \( \Phi \).
The Backward algorithm

1. **Initialization:**

   \[ \beta_T(i) = a_{i,F}, \quad 1 \leq i \leq N \]

2. **Recursion** (again since states 0 and \( q_F \) are non-emitting):

   \[ \beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N, 1 \leq t < T \]

3. **Termination:**

   \[ P(O|\lambda) = \alpha_T(q_F) = \beta_1(\rho) = \sum_{j=1}^{N} a_{0j} b_j(o_1) \beta_1(j) \]
Inductive step of the backward algorithm

- Computation of $\beta_t(i)$ by weighted sum of all successive values $\beta_{t+1}$

$$\beta_t(i) = \sum_j \beta_{t+1}(j) \ a_{ij} \ b_j(o_{t+1})$$
Intuition for re-estimation of $a_{ij}$

- We will estimate $\hat{a}_{ij}$ via this intuition:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

- Numerator intuition:
  - Assume we had some estimate of probability that a given transition $i \rightarrow j$ was taken at time $t$ in observation sequence.
  - If we knew this probability for each time $t$, we could sum over all $t$ to get expected value (count) for $i \rightarrow j$. 


Re-estimation of $a_{ij}$

- Let $\xi_t$ be the probability of being in state $i$ at time $t$ and state $j$ at time $t+1$, given $O_{1..T}$ and model $\Phi$:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O)$$

- We can compute $\xi$ from not-quite-$\xi$, which is:

$$not\_quite\_\_t(i,j) = P(q_t = i, q_{t+1} = j, O)$$
Computing not-quite-$\xi$

The four components of $P(q_t = i, q_{t+1} = j, O | t)$: $a, a_{ij}$ and $b_j(o_t)$

$$\text{not-quite-}$\xi_t(i, j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$
From not-quite-\(\xi\) to \(\xi\)

- We want:
  \[ t(i, j) = P(q_t = i, q_{t+1} = j \mid O, ) \]

- We’ve got:
  \[ \text{not\_quite\_} t(i, j) = P(q_t = i, q_{t+1} = j, O \mid ) \]

- Which we compute as follows:
  \[ \text{not\_quite\_} \xi_t (i, j) = \alpha_t (i) a_{ij} b_j (o_{t+1}) \beta_{t+1} (j) \]
From not-quite-$\xi$ to $\xi$

- We want:
  \[
  t(i, j) = P(q_t = i, q_{t+1} = j \mid O, )
  \]

- We’ve got:
  \[
  not\_quite\_t(i, j) = P(q_t = i, q_{t+1} = j, O \mid )
  \]

- Since:
  \[
  P(X\mid Y, Z) = \frac{P(X, Y\mid Z)}{P(Y\mid Z)}
  \]

- We need:
  \[
  t(i, j) = \frac{not\_quite\_t(i, j)}{P(O \mid )}
  \]
From not-quite-$\xi$ to $\xi$

$$ t(i,j) = \frac{\text{not}_{\text{-quite}} - t(i,j)}{P(O|\lambda)} $$

$$ \text{not-quite-}\xi_t(i,j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) $$

$$ P(O|\lambda) = \alpha_T(q_F) = \beta_T(q_0) = \sum_{j=1}^{N} \alpha_t(j) \beta_t(j) $$

$$ \xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\alpha_T(q_F)} $$
From \( \xi \) to \( a_{ij} \)

\[
\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}
\]

- The expected number of transitions from state \( i \) to state \( j \) is the sum over all \( t \) of \( \xi_t \)
- The total expected number of transitions out of state \( i \) is the sum over all transitions out of state \( i \)
- Final formula for reestimated \( a_{ij} \):

\[
\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i, k)}
\]
Re-estimating the observation likelihood \( b \)

- This is the probability of a given symbol \( v_k \) from the observation vocabulary \( V \), given a state \( j \): \( \hat{b}_j(v_k) \).

\[
\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}
\]

We’ll need to know \( \gamma_t(j) \): the probability of being in state \( j \) at time \( t \):

\[
\gamma_t(j) = P(q_t = j | O, \lambda)
\]

\[
\gamma_t(j) = \frac{P(q_t = j, O | \lambda)}{P(O | \lambda)}
\]

\[
\gamma_t(j) = \frac{\alpha_t(j) \beta_t(j)}{P(O | \lambda)}
\]
Computing $\gamma$

$$\gamma_t(j) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)}$$

$$\gamma_t(j) = \frac{\alpha_t(j) \beta_t(j)}{P(O|\lambda)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^{T \text{s.t.} O_t = v_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$
Summary

\[ \hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i, k)} \]

The ratio between the expected number of transitions from state i to j and the expected number of all transitions from state i.

\[ \hat{b}_j(v_k) = \frac{\sum_{t=1}^{T} \text{s.t. } O_t = v_k \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \]

The ratio between the expected number of times the observation data emitted from state j is \( v_k \), and the expected number of times any observation is emitted from state j.
The Forward-Backward Algorithm

function FORWARD-BACKWARD(observations of len $T$, output vocabulary $V$, hidden state set $Q$) returns $HMM=(A,B)$

initialize $A$ and $B$
iterate until convergence

E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall \ t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall \ t, i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \sum_{t=1}^{T-1} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 \text{s.t. } O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return $A, B$
Summary: Forward-Backward Algorithm

- Initialize $\Phi=(A,B)$
- Compute $\alpha, \beta, \xi$
- Estimate new $\Phi'=(A,B)$
- Replace $\Phi$ with $\Phi'$
- If not converged go to 2
Applying FB to speech: Caveats

- Network structure of HMM is always created by hand
  - no algorithm for double-induction of optimal structure and probabilities has been able to beat simple hand-built structures.
- Always Bakis network = links go forward in time
- Subcase of Bakis net: beads-on-string net:

- Baum-Welch only guaranteed to return local max, rather than global optimum
- At the end, we throw away A and only keep B