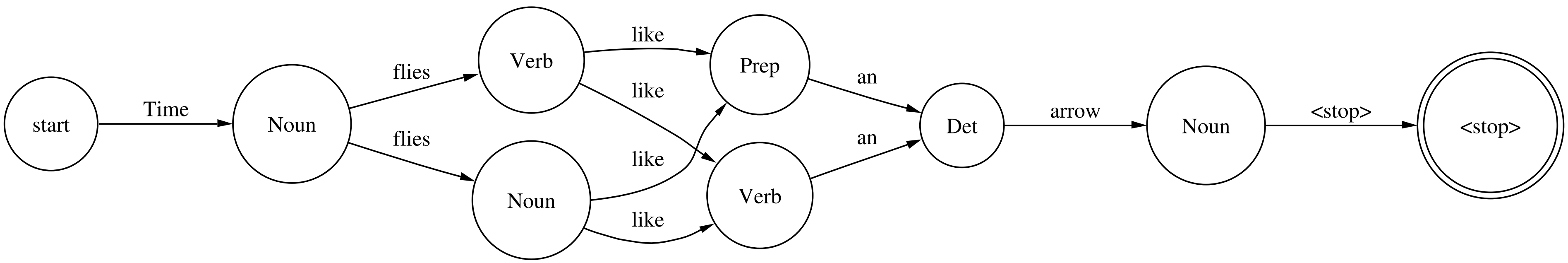
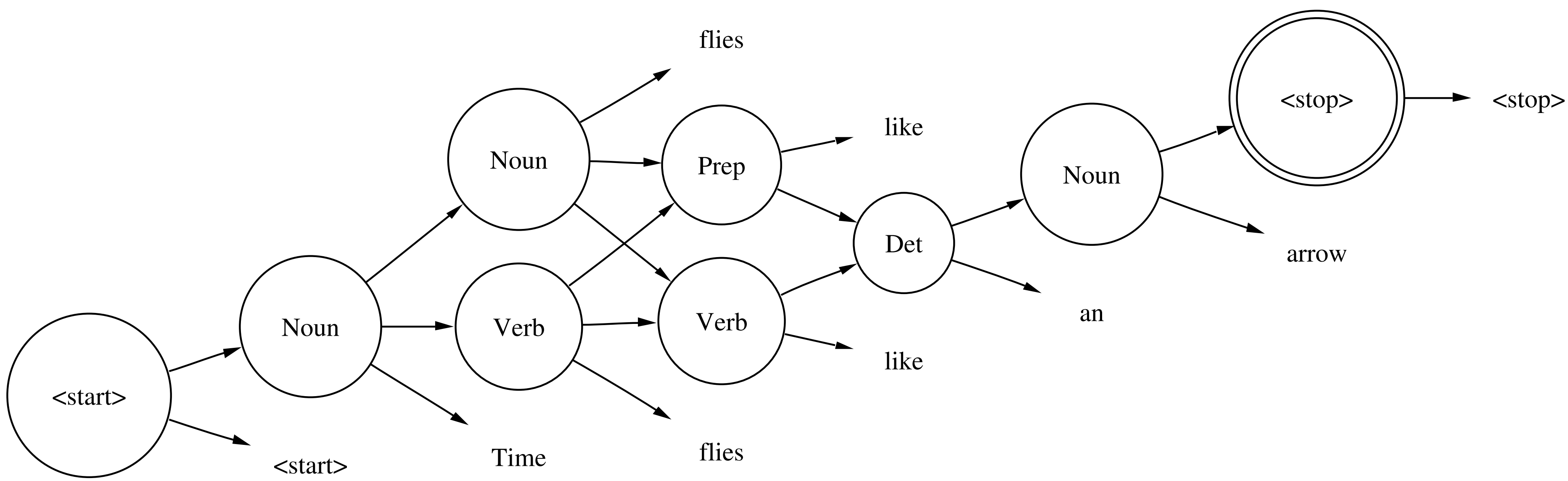
CMPT-413: Computational Linguistics

Anoop Sarkar
anoop@cs.sfu.ca
www.sfu.ca/~anoop/courses/CMPT-413-Spring-2003.html

Part of Speech Tagging: Mealy Machine



Part of Speech Tagging: Moore Machine



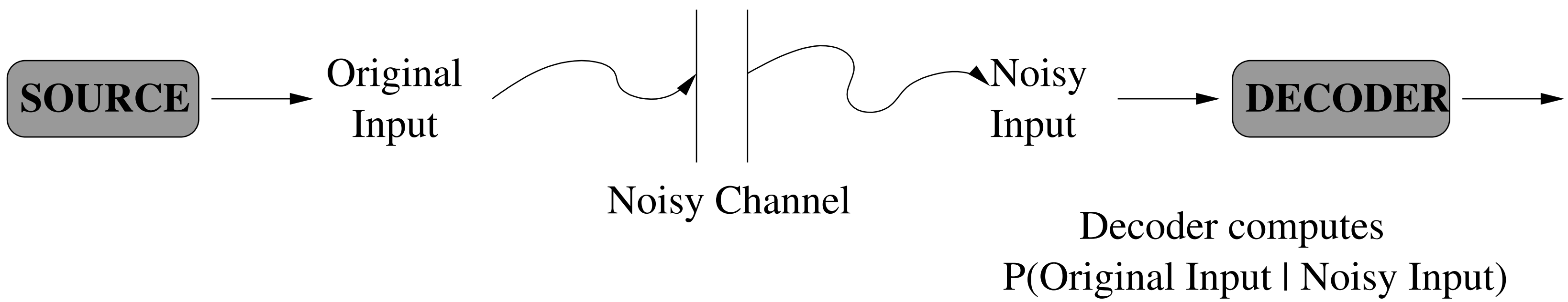
Automatic Speech Recognition (ASR)

- Acoustic observations: signal processing to extract energy levels at each frequency level, extract *features* from the waveform at regular (e.g. 10msec) intervals

- Observation sequence: **o** $\rightarrow$ Transcription: **w**
  Probability $P(\mathbf{o} \mid \mathbf{w})$ of observing **o** when transcription is **w**

$$
\begin{aligned}
\mathbf{w}^* &= \underset{\mathbf{w}}{\arg\max}\ P(\mathbf{w} \mid \mathbf{o}) = \underset{\mathbf{w}}{\arg\max}\ \frac{P(\mathbf{o} \mid \mathbf{w}) P(\mathbf{w})}{P(\mathbf{o})} \\
&= \underset{\mathbf{w}}{\arg\max}\ \underbrace{P(\mathbf{o} \mid \mathbf{w})}_{\substack{generative \\ model}} \ \underbrace{P(\mathbf{w})}_{\substack{language \\ model}}
\end{aligned}
$$

Noisy Channel Model: Bayesian Inference Strikes Again



Generative Models of Speech

- Typical decomposition of $P(\mathbf{o} \mid \mathbf{w})$ into mappings between various levels of linguistic structure (multi-stage cascade):
  - Acoustic model: $P(\mathbf{o} \mid \mathbf{p})$, phone sequences $\longrightarrow$ observation sequences
  - Pronunciation model: $P(\mathbf{p} \mid \mathbf{w})$, word sequences $\longrightarrow$ phone sequences
  - Language model: $P(\mathbf{w})$

Generative Models of Speech

- Acoustic model: $P(\mathbf{o} \mid \mathbf{p})$, phone sequences $\longrightarrow$ observation sequences
  - $P(\mathbf{o} \mid \mathbf{d})$, distribution sequences $\longrightarrow$ observation vectors — *symbolic* $\longrightarrow$ *quantitative*
  - $P(\mathbf{d} \mid \mathbf{m})$, model sequences (context-dependent phone model) $\longrightarrow$ distribution sequences
  - $P(\mathbf{m} \mid \mathbf{p})$, phone sequences $\longrightarrow$ model sequences

Brief History of ASR

- 1920s: Radio Rex
  500 Hz of energy of the vowel in “Rex” caused the toy dog to move
- 1950s: Digit Recognition (Bell Labs)
- 1960s: Advances in Signal Processing and Neural Nets (not much success in ASR)
- 1970s: despite large ARPA funding, not much breakthrough success
- 1980s: Discrete ASR, Language models, corpus collection
  TIMIT corpus (phonetic corpus), ATIS corpus (Air Travel Information System) focus on language understanding
- 1990s: Large Vocabulary Continuous ASR, Dynamic Time Warping (edit distance), better phonetic models using classifiers (decision trees and neural nets), better language models using smoothing, larger corpora: 10^7 to 10^9 words in size
- Current work: multiple languages, multiple speakers (speaker id), noise resistant (telephone speech), software: htk, sphinx.

Hidden Markov Models

- A weighted finite state machine, with probabilities on transitions and on inputs (outputs).
- A set of states, each state is hidden, i.e. not visible in the data. The number of states is arbitrary but fixed and set in advance
- Assume each state is connected to every other state
- Numerous applications in speech, language processing, bioinformatics, cryptography, modeling continuous fns.

Hidden Markov Models

- At each time tick t, we traverse from one state to another and emit an output symbol

- $P(s^i \xrightarrow{w} s^j) = P(S_{t+1} = s^j, W_t = w \mid S_t = s^i)$

- $P(s^i \xrightarrow{w} s^j) = P(w, s^j \mid s^i) = P(w \mid s^i) P(s^j \mid s^i)$
  — the Markov assumption

- transition probability: $P(s_j \mid s_i)$

- output probability: $P(w \mid s_i)$

Hidden Markov Models: Algorithms

- Let's assume we have all the transition probabilities and output probabilities for an HMM

- **Supervised Learning**: estimate these probabilities with counts from human labeled *training data*
  sequence of output symbols $w_1, \ldots, w_n$ along with the correct state for each output symbol $(s_1, \ldots, s_{n+1}; w_1, \ldots, w_n)$

- Now let's assume that we have a sequence of output symbols: $w_1, \ldots, w_n$ and we need to find the best sequence of states $s_1, \ldots, s_{n+1} \longrightarrow$ Viterbi algorithm

<u>Hidden Markov Models</u>: Algorithms

- What if we do not have human labeled training data? We have just the list of sequences of output symbols: $w_1, \ldots, w_n$

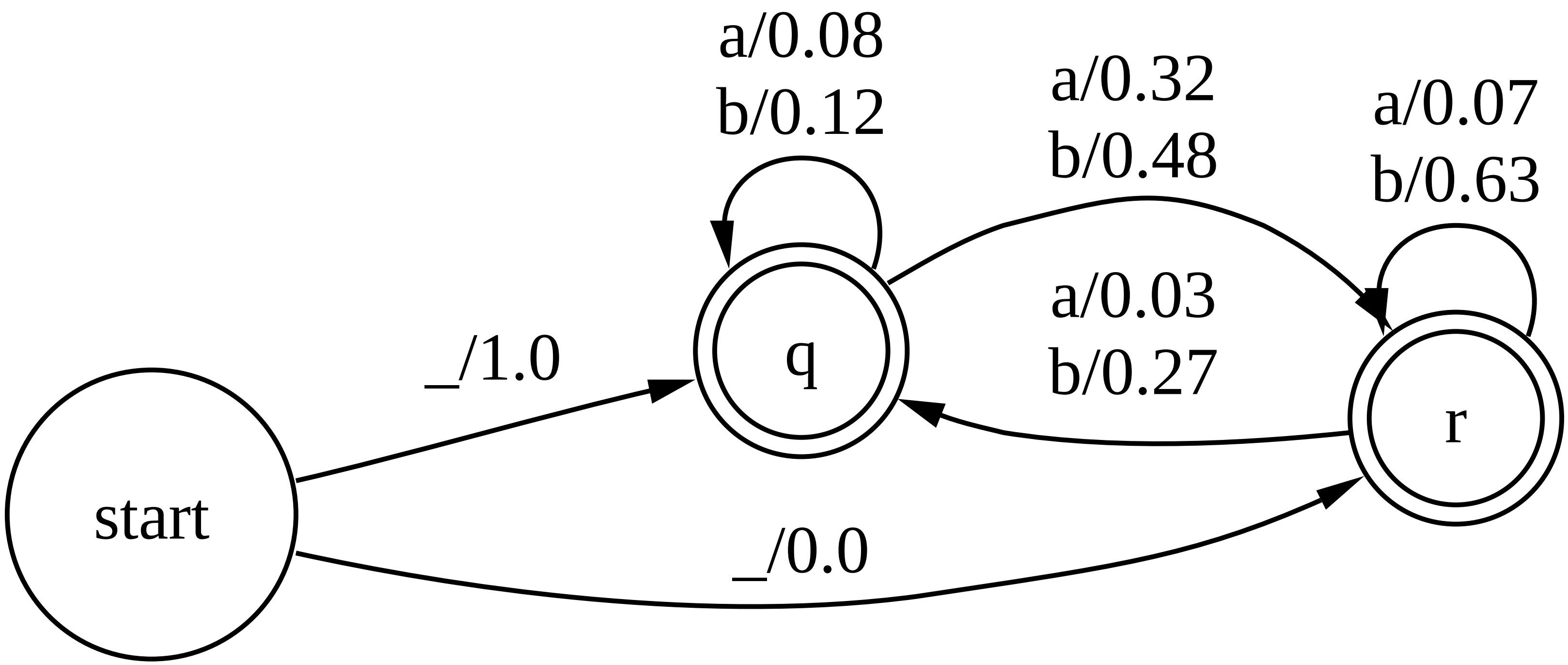
- **Unsupervised Learning**: how can we infer the "hidden" state sequences for each output symbol sequence

- Infer the transition probabilities and the output probabilities $\longrightarrow$ Forward-Backward algorithm

Hidden Markov Models

- $P(w_{(1,n)}) = \sum_{s_{(1,n+1)}} P(w_{(1,n)},\ s_{(1,n+1)})$
- $\sum_{s_{(1,n+1)}} \prod_{i=1}^{n} P(w_i,\ s_{i+1} \mid s_i)$
- $\sum_{s_{(1,n+1)}} \prod_{i=1}^{n} P(s^i \xrightarrow{w_i} s^{i+1})$
- Best path (Viterbi algorithm): $\underset{s_{(1,n+1)}}{\text{arg max}} \prod_{i=1}^{n} P(s^i \xrightarrow{w_i} s^{i+1})$

$$P(w, s_i \mid s_{i-1}) = P(w \mid s_i) \times P(s_i \mid s_{i-1})$$



Viterbi Algorithm: Extension of ND-Recognize

```
function VITERBI (edges, input) returns best-path
 T = length of input;
 num-states = NUM-OF-STATES(edges);
 Create a path matrix: viterbi[num-states+2, T+2]
 for each time step t from 0 to T:
   for each state s from 0 to num-states:
     for each transition s’ from s in edges:
       new-score = viterbi[s,t] *
                   edges[s,s’] *
                   input[s’,obs[t]];
       if ((viterbi[s’, t+1] == 0) or
            (new-score > viterbi[s’, t+1])):
          viterbi[s’, t+1] = new-score;
          back-pointer[s’, t+1] = s;
```

Viterbi Algorithm

- Key Idea 1: storing the best path upto *each* state is enough to find the best path for the entire input sequence.

- Key Idea 2: storing only the single current best path is *not* enough to find the best path for the entire input sequence.

Forward-Backward Algorithm: Baum-Welch

- How can we compute transition and output probabilities, when the state sequences are ”hidden”?

- Intuitively, probability of taking a transition from a state s^i to s^j is $P_e(s^i \stackrel{w}{\rightarrow} s^j) = \frac{C(s^i \stackrel{w}{\rightarrow} s^j)}{\sum_{k,w'} C(s^i \stackrel{w'}{\rightarrow} s^k)}$

- So once we have a method for computing $C(s^i \stackrel{w}{\rightarrow} s^j)$ we can re-estimate each transition probability

- Note that number of times you take a transition also depends on the initial setting of the transition and output probability

Forward-Backward Algorithm

- Hence, the probability of a transition is the number of times it was used in a path (state sequence) times the probability of that path (for all paths)

- Let η be the probability that $s^i \xrightarrow{w} s^j$ appears in the path $s_{(1,n+1)}$ when the output is $w_{(1,n)}$

$$C(s^i \xrightarrow{w} s^j) = \sum_{s_{(1,n+1)}} P(s_{(1,n+1)} \mid w_{(1,n)}) \cdot \eta(s^i \xrightarrow{w} s^j, s_{(1,n+1)}, w_{(1,n)})$$

Forward-Backward Algorithm

$$C(s^i \xrightarrow{w} s^j) = \sum_{s_{(1,n+1)}} P(s_{(1,n+1)} \mid w_{(1,n)}) \cdot \eta(s^i \xrightarrow{w} s^j, s_{(1,n+1)}, w_{(1,n)})$$

$$P(s_{(1,n+1)} \mid w_{(1,n)}) = \frac{P(s_{(1,n+1)}, w_{(1,n)})}{P(w_{(1,n)})}$$

$$C(s^i \xrightarrow{w} s^j) = \frac{1}{P(w_{(1,n)})} \times \sum_{t=1}^{n} \sum_{s_{(1,n+1)}} P(s_{(1,n+1)}, w_{(1,n)}, S_t = s^i, S_{t+1} = s^j, W_t = w)$$

$$P(w_{(1,n)}) = \sum_{s_{(1,n+1)}} P(s_{(1,n+1)}, w_{(1,n)})$$

Forward-Backward Algorithm

$$
\begin{aligned}
C(s^i \xrightarrow{w} s^j) &= \frac{1}{P(w_{(1,n)})} \times \\
&\quad \sum_{t=1}^{n} P(w_{(1,n)}, S_t = s^i, S_{t+1} = s^j, W_t = w) \\
&= \frac{1}{P(w_{(1,n)})} \times \\
&\quad \sum_{t=1}^{n} P(w_{(1,t-1)}, S_t = s^i, S_{t+1} = s^j, W_t = w, w_{(t+1,n)}) \\
&= \frac{1}{P(w_{(1,n)})} \times \sum_{t=1}^{n} P(w_{(1,t-1)}, S_t = s^i) \cdot \\
&\quad P(S_{t+1} = s^j, W_t = w \mid w_{(1,t-1)}, S_t = s^i) \cdot \\
&\quad P(w_{(t+1,n)} \mid w_{(1,t)}, S_t = s^i, S_{t+1} = s^j)
\end{aligned}
$$

Forward-Backward Algorithm

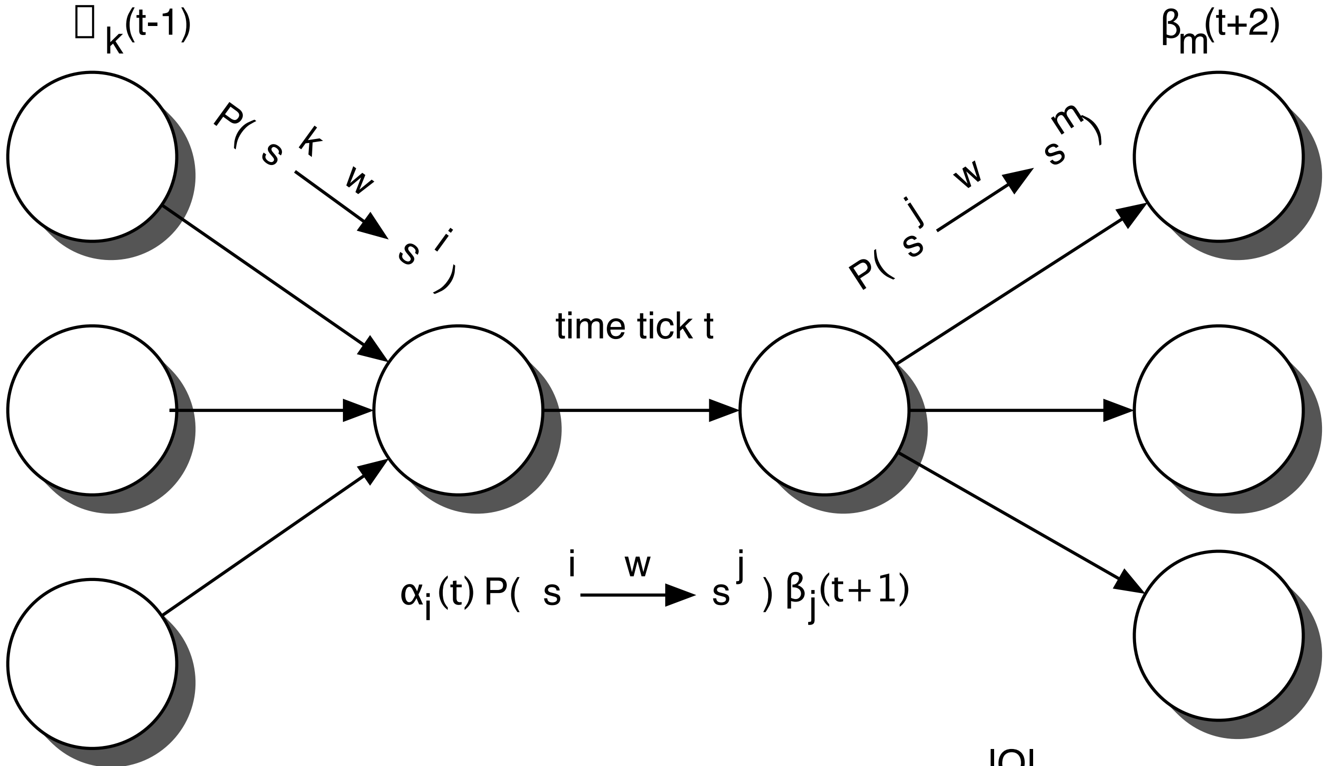
$$
\begin{aligned}
C(s^i \xrightarrow{w} s^j) &= \frac{1}{P(w_{(1,n)})} \times \sum_{t=1}^{n} P(w_{(1,t-1)}, S_t = s^i) \cdot \\
&\quad P(S_{t+1} = s^j, W_t = w \mid \cancel{w_{(1,t-1)}}, S_t = s^i) \cdot \\
&\quad P(w_{(t+1,n)} \mid \cancel{w_{(1,t)}, S_t = s^i}, S_{t+1} = s^j) \\
&= \frac{1}{P(w_{(1,n)})} \times \sum_{t=1}^{n} P(w_{(1,t-1)}, S_t = s^i) \cdot \\
&\quad P(S_{t+1} = s^j, W_t = w \mid S_t = s^i) \cdot \\
&\quad P(w_{(t+1,n)} \mid S_{t+1} = s^j) \\
&= \frac{1}{P(w_{(1,n)})} \times \sum_{t=1}^{n} \alpha_i(t) \cdot P(s^i \xrightarrow{w} s^j) \cdot \beta_j(t+1)
\end{aligned}
$$

Forward-Backward Algorithm

$$
\begin{aligned}
\alpha_i(t) &= P(w_{(1,t-1)}, S_t = s^i) \\
\alpha_{s_1}(1) &= 1.0 \\
\alpha_j(t+1) &= P(w_{(1,t)}, S_{t+1} = s^j) \\
&= \sum_i P(w_{(1,t)}, S_t = s^i, S_{t+1} = s^j) \\
&= \sum_i P(w_{(1,t-1)}, S_t = s^i) \cdot \\
&\quad P(W_t = w, S_{t+1} = s^j \mid \cancel{w_{(1,t-1)}}, S_t = s^i) \\
&= \sum_i \alpha_i(t) \cdot P(s^i \xrightarrow{w} s^j)
\end{aligned}
$$

Forward-Backward Algorithm

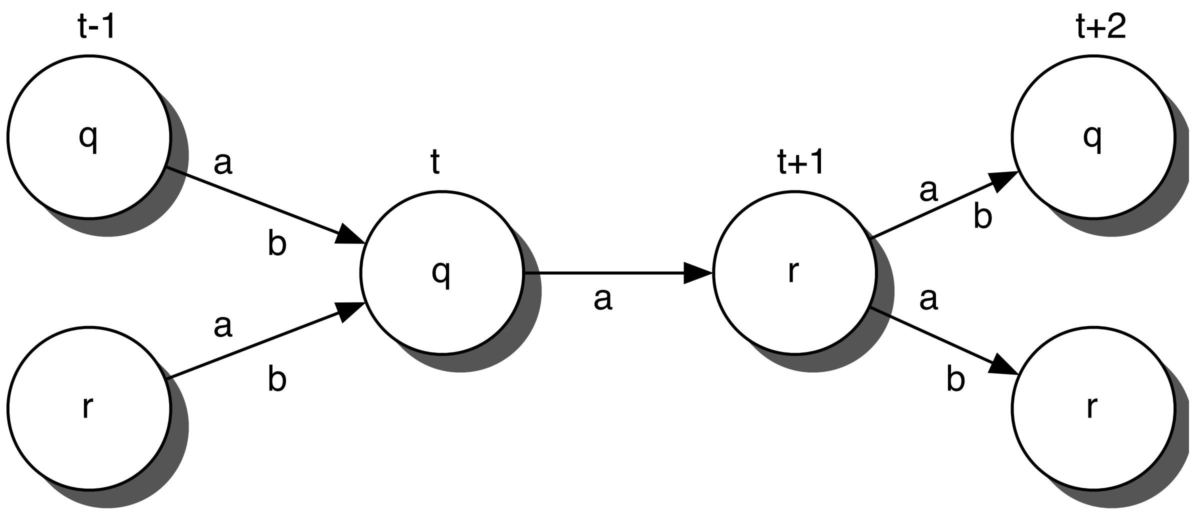
$$
\begin{aligned}
\beta_i(t) &= P(w_{(t,n)} \mid S_t = s^i) \\
\beta_i(n+1) &= P(\epsilon \mid S_{n+1} = s^i) = 1.0 \\
\beta_i(t-1) &= P(w_{(t-1,n)} \mid S_{t-1} = s^i) \\
&= \sum_j P(W_{t-1} = w, S_t = s^j \mid S_{t-1} = s^i) \cdot \\
&\quad\; P(w_{(t,n)} \mid \cancel{W_{t-1} = w}, S_t = s^j, \cancel{S_{t-1} = s^i}) \\
&= \sum_j P(W_{t-1} = w, S_t = s^j \mid S_{t-1} = s^i) \cdot \\
&\quad\; P(w_{(t,n)} \mid S_t = s^j) \\
&= \sum_j P(s^i \xrightarrow{w} s^j) \cdot \beta_j(t)
\end{aligned}
$$



$$\alpha_i(t) = \sum_{k=1}^{|Q|} \alpha_k(t-1)\, P(s^k \xrightarrow{w} s^i)$$

$$\beta_j(t+1) = \sum_{m=1}^{|Q|} P(s^j \xrightarrow{w} s^m)\, \beta_m(t+2)$$

$$C(s^i \xrightarrow{w} s^j) = \frac{1}{P(w_{1,n})} \sum_{t=1}^{n} \alpha_i(t)\, P(s^i \xrightarrow{w} s^j)\, \beta_j(t+1)$$



$$
\begin{aligned}
\alpha_q(t) &= \alpha_q(t-1)P(a,q \mid q) + \alpha_q(t-1)P(b,q \mid q) + \\
&\quad \alpha_r(t-1)P(a,q \mid r) + \alpha_r(t-1)P(b,q \mid r) \\
\beta_r(t+1) &= P(a,q \mid r)\beta_q(t+2) + P(b,q \mid r)\beta_q(t+2) + \\
&\quad P(a,r \mid r)\beta_r(t+2) + P(b,r \mid r)\beta_r(t+2) \\
C(q \xrightarrow{a} r) &= \frac{1}{P(w_{(1,n)})} \sum_{t=1}^{n} \alpha_q(t)P(a,r \mid q)\beta_r(t+1)
\end{aligned}
$$

Forward-Backward Algorithm

- Set initial transition probabilities to appropriate values (usually random)

- Compute $C(s^i \xrightarrow{w} s^j)$ for each state i and then $P_e(s^i \xrightarrow{w} s^j) = \frac{C(s^i \xrightarrow{w} s^j)}{\sum_{k,w'} C(s^i \xrightarrow{w'} s^k)}$

- Compute likelihood $P(w_{(1,n)}) = \beta_{s^1}(1)$; iterate until likelihood is maximized (or entropy is minimized)

- Here we considered the case for one training sentence $w_{(1,n)}$. For a whole corpus, $\prod_k P(w^k_{(1,n)})$ is the likelihood of the entire corpus with k sentences

Forward-Backward Algorithm

- Also known as the Baum-Welch algorithm.
- Likelihood is guaranteed to be non-decreasing (entropy is guaranteed to go down *or* stay the same.
  $\longrightarrow$ This guarantee is due to the theorem by Baum (generalized in DLR77)
  Maximum-likelihood from incomplete data via the EM algorithm. A. P. Dempster, N. M. Laird and D. B. Rubin. *Journal of the Royal Statistics Society*, 1977, 39:1, pp. 1–38
- Forward-Backward Algorithm is an example of purely unsupervised learning