CS 224S / LINGUIST 281
Speech Recognition, Synthesis, and Dialogue

Dan Jurafsky

Lecture 6: Forward-Backward (Baum-Welch) and Word Error Rate

IP Notice:

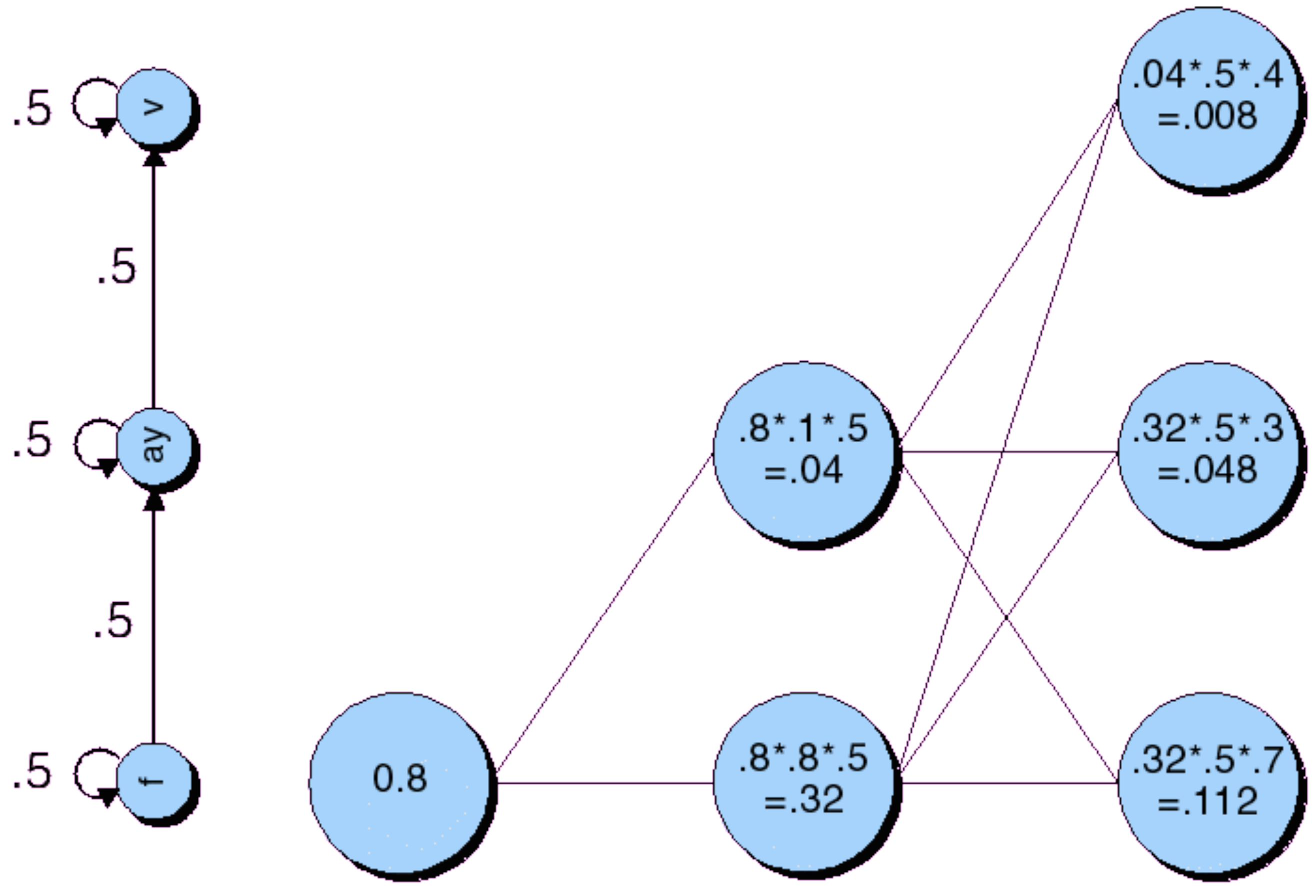
Outline for Today

- Speech Recognition Architectural Overview
- Hidden Markov Models in general and for speech
  - Forward
  - Viterbi Decoding
- How this fits into the ASR component of course
  - July 27 (today): HMMs, Forward, Viterbi,
  - Jan 29 Baum-Welch (Forward-Backward)
  - Feb 3: Feature Extraction, MFCCs
  - Feb 5: Acoustic Modeling and GMMs
  - Feb 10: N-grams and Language Modeling
  - Feb 24: Search and Advanced Decoding
  - Feb 26: Dealing with Variation
  - Mar 3: Dealing with Disfluencies

LVCSR

- Large Vocabulary Continuous Speech Recognition
- ~20,000-64,000 words
- Speaker independent (vs. speaker-dependent)
- Continuous speech (vs isolated-word)

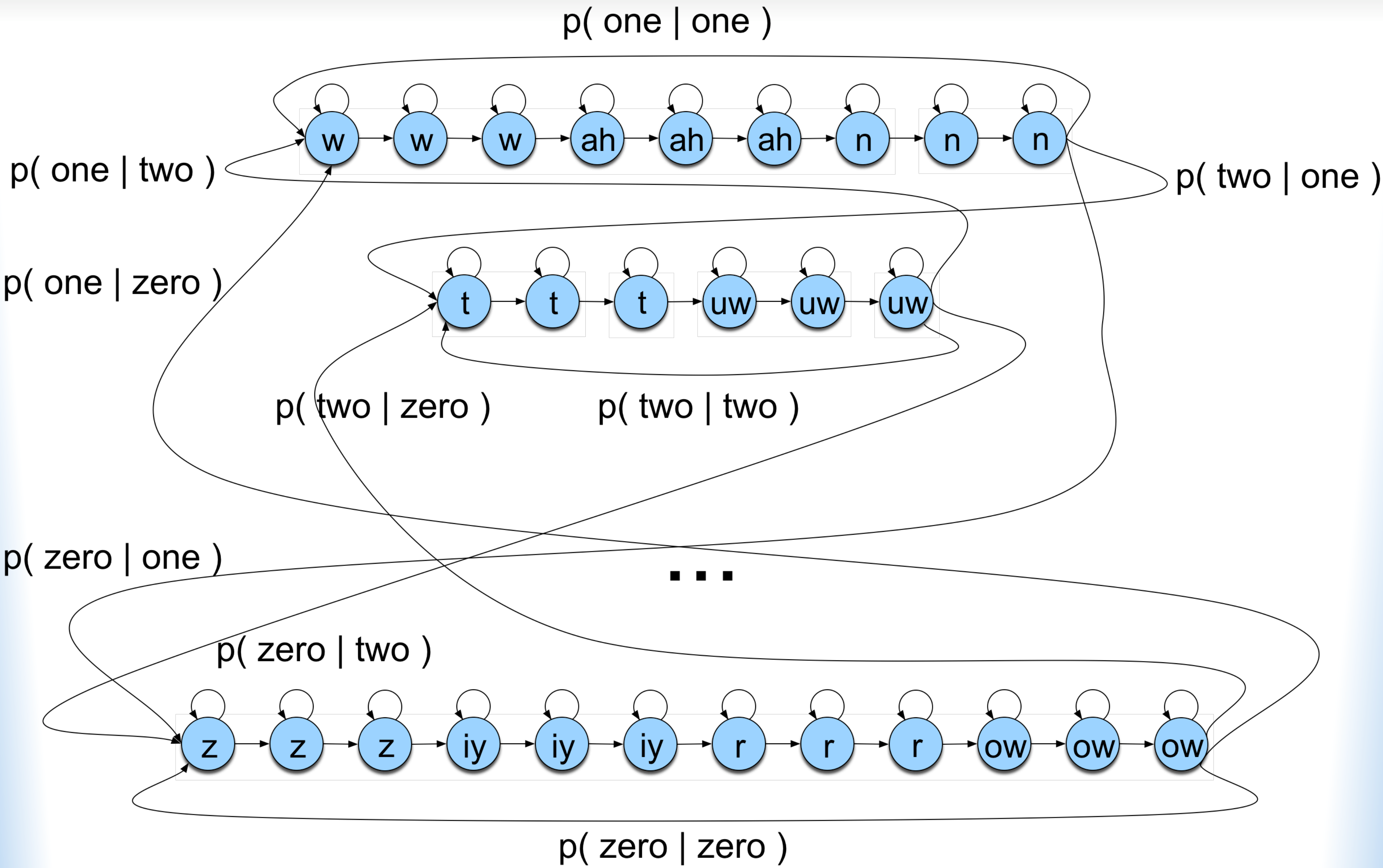
Viterbi trellis for "five"



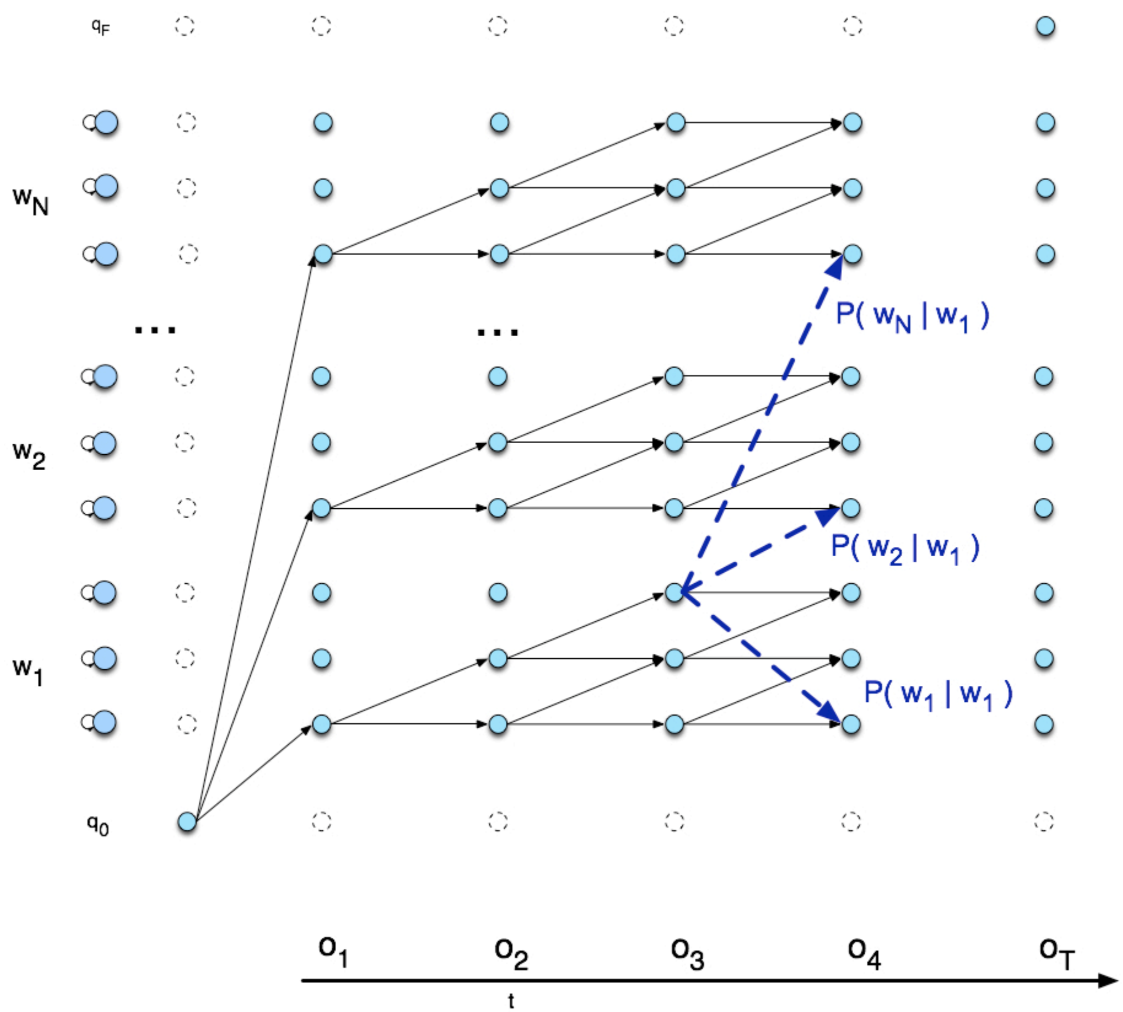
Viterbi trellis for "five"

V	0	0	0.008	0.0072	0.00672	0.00403	0.00188	0.00161	0.000667	0.000493
AY	0	0.04	0.048	0.0448	0.0269	0.0125	0.00538	0.00167	0.000428	8.78e-05
F	0.8	0.32	0.112	0.0224	0.00448	0.000896	0.000179	4.48e-05	1.12e-05	2.8e-06
Time	1	2	3	4	5	6	7	8	9	10
	f 0.8	*f* 0.8	*f* 0.7	*f* 0.4	*f* 0.4	*f* 0.4	*f* 0.4	*f* 0.5	*f* 0.5	*f* 0.5
	ay 0.1	*ay* 0.1	*ay* 0.3	*ay* 0.8	*ay* 0.8	*ay* 0.8	*ay* 0.8	*ay* 0.6	*ay* 0.5	*ay* 0.4
B	*v* 0.6	*v* 0.6	*v* 0.4	*v* 0.3	*v* 0.3	*v* 0.3	*v* 0.3	*v* 0.6	*v* 0.8	*v* 0.9
	p 0.4	*p* 0.4	*p* 0.2	*p* 0.1	*p* 0.1	*p* 0.1	*p* 0.1	*p* 0.1	*p* 0.3	*p* 0.3
	iy 0.1	*iy* 0.1	*iy* 0.3	*iy* 0.6	*iy* 0.6	*iy* 0.6	*iy* 0.6	*iy* 0.5	*iy* 0.5	*iy* 0.4

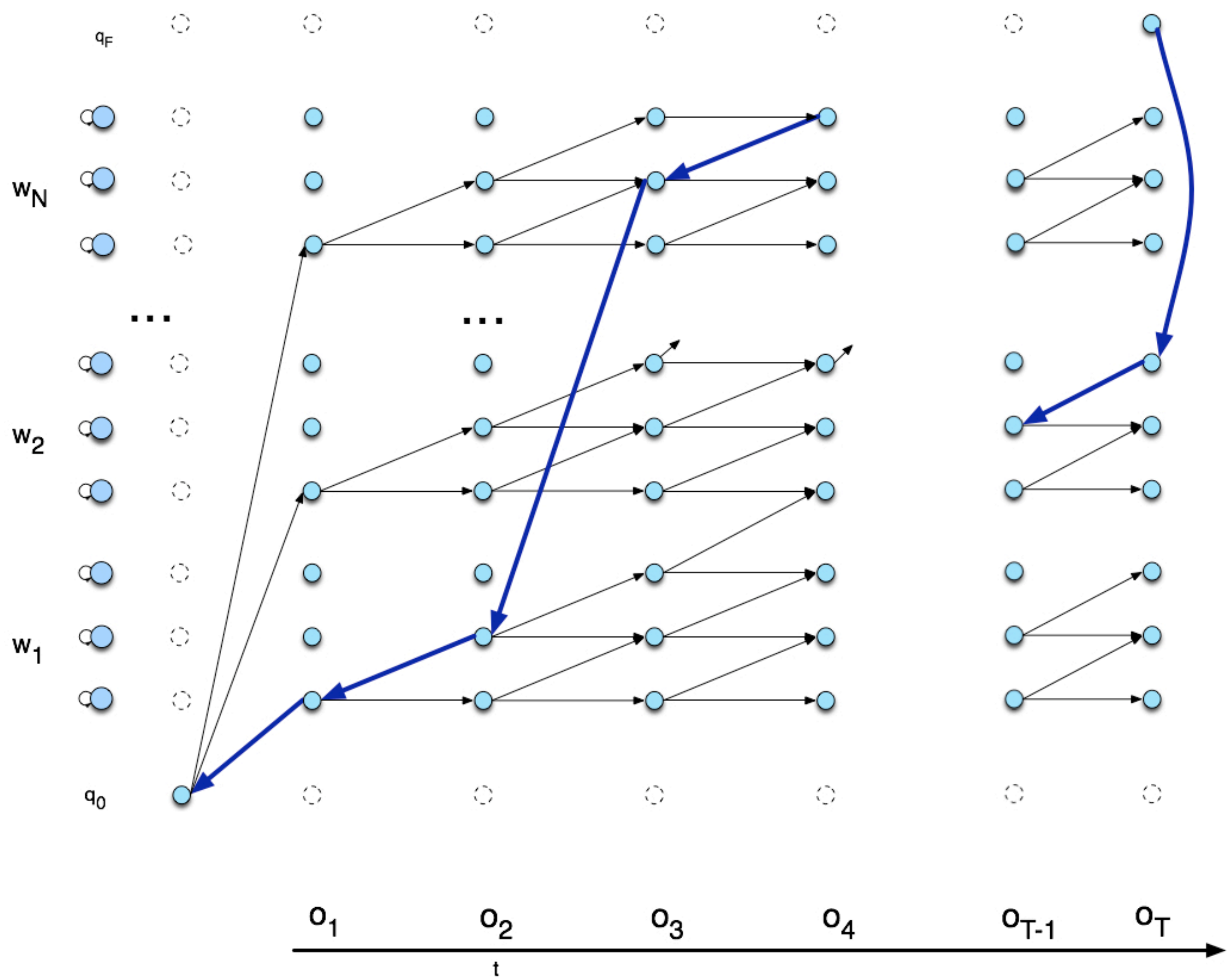
Search space with bigrams



Viterbi trellis



Viterbi backtrace



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

Input to Baum-Welch

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

- For ice-cream task
  - O = {1,3,2.,,,.}
  - 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
  - $A = \{a_{ij}\}$:

$$a_{ij} = \frac{C(i \to j)}{\sum_{q \in Q} C(i \to q)}$$

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:

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots o_T, | q_t = i, \Phi)$$

- 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 Φ.

The Backward algorithm

1. **Initialization:**

$$\beta_T(i) \;=\; a_{i,F}, \quad 1 \le i \le N$$

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

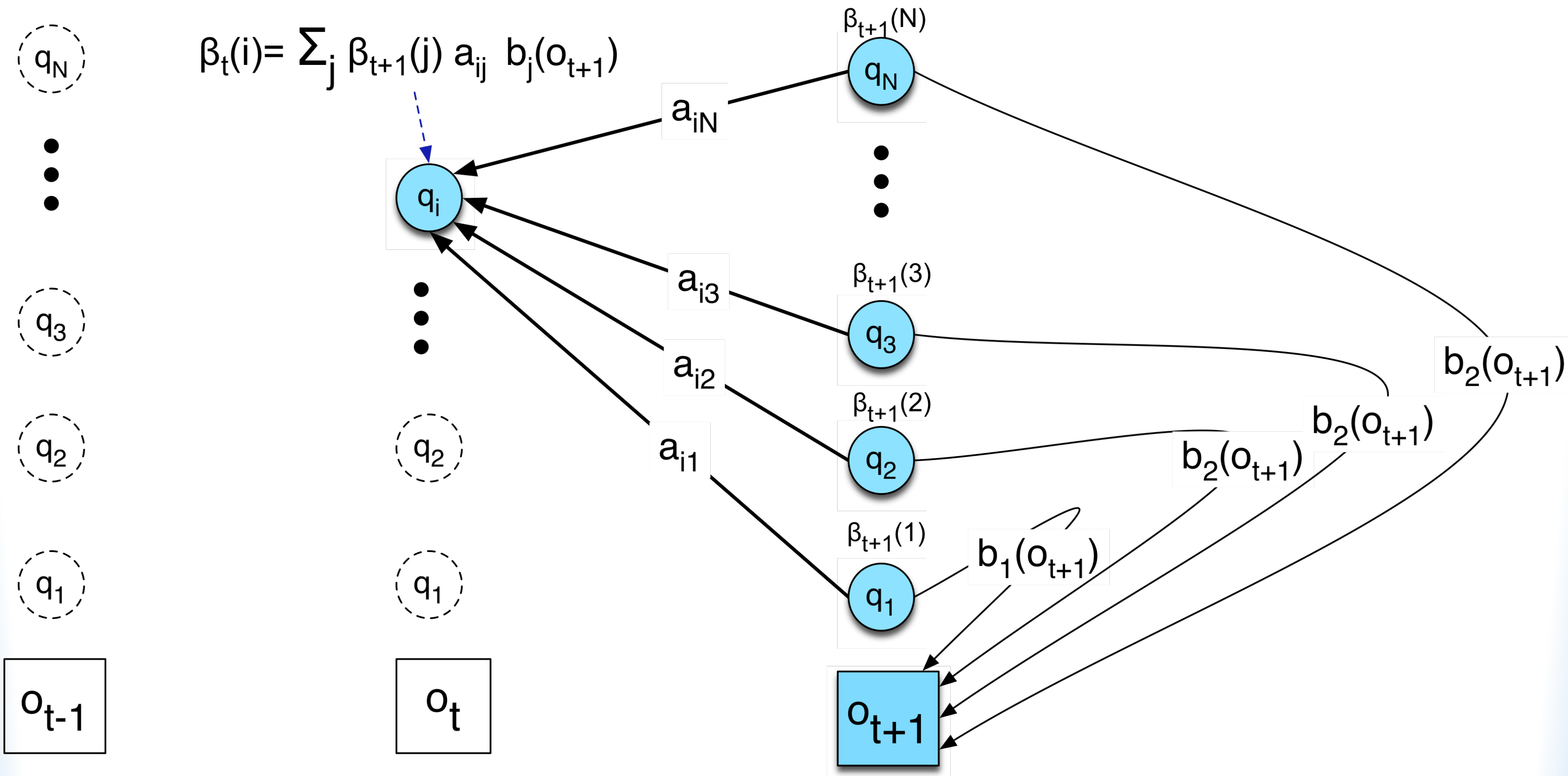
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \le i \le N, 1 \le t < T$$

3. **Termination:**

$$P(O|\lambda) = \alpha_T(q_F) = \beta_1(0) = \sum_{j=1}^{N} a_{0j}\, b_j(o_1)\, \beta_1(j)$$

Inductive step of the backward algorithm
(figure inspired by Rabiner and Juang)

Computation of $\beta_t(i)$ by weighted sum of all successive values β_{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➔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➔j.

Re-estimation of a_{ij}

- Let ξ_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 Φ:

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

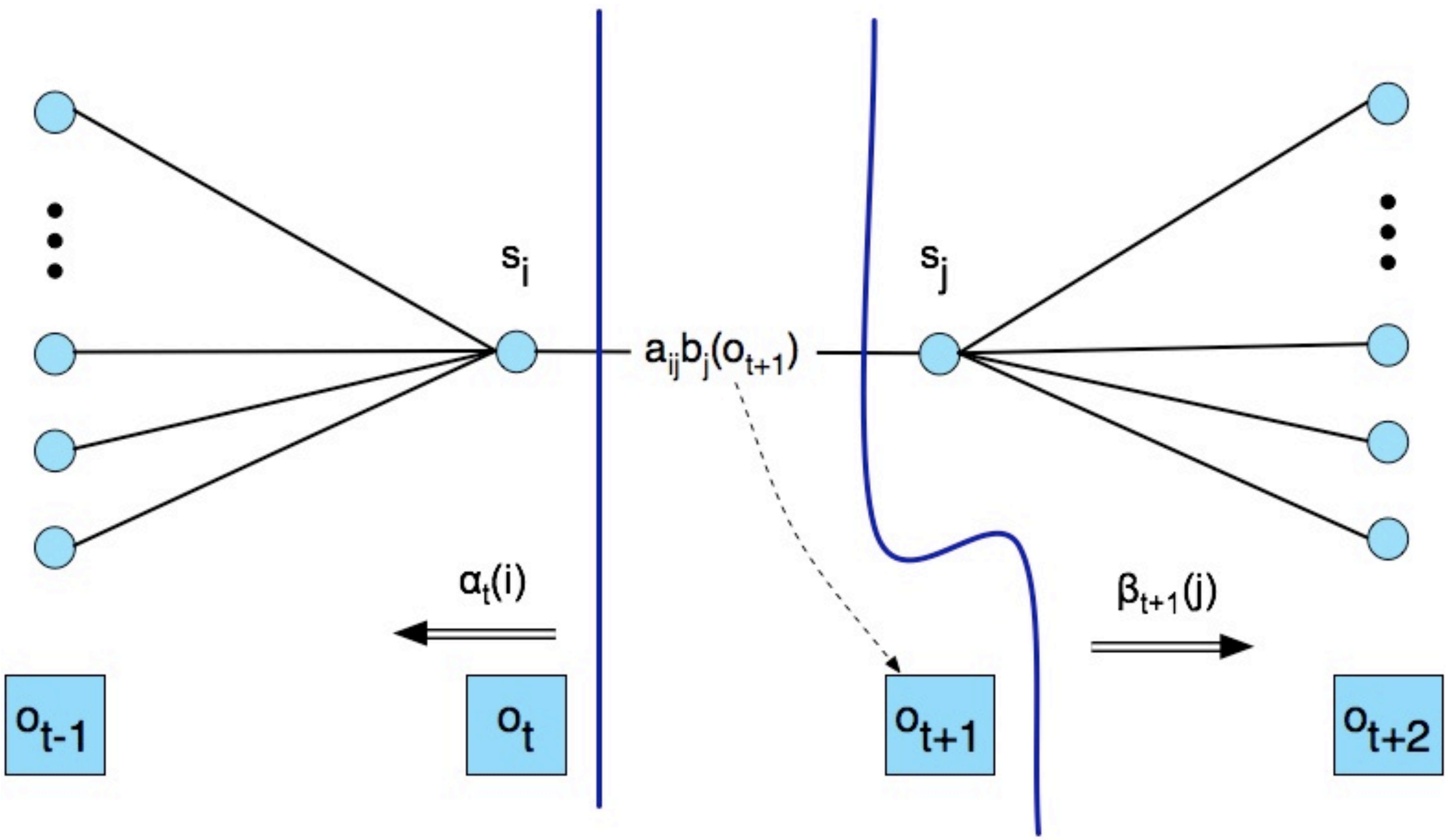
- We can compute ξ from not-quite-ξ, which is:

$$not\_quite\_\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O \mid \lambda)$$

Computing not-quite-ξ

The four components of $P(q_t = i, q_{t+1} = j, O \mid \lambda)$: α, β, 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-ξ to ξ

- We want:

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

- We've got:

$$not\_quite\_\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O \mid \lambda)$$

- 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-ξ to ξ

- We want:

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

- We've got:

$$not\_quite\_\xi_t(i,j) = P(q_t = i, q_{t+1} = j, O \mid \lambda)$$

- Since:

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)}$$

- We need:

$$\xi_t(i,j) = \frac{not\_quite\_\xi_t(i,j)}{P(O \mid \lambda)}$$

From not-quite-ξ to ξ

$$\xi_t(i,j) = \frac{not\_quite\_\xi_t(i,j)}{P(O \mid \lambda)}$$

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

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

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

From ξ 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 ξ
- 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: $\hat{b}_j(v_k)$.

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

We'll need to know the probability of being in state j at time t:

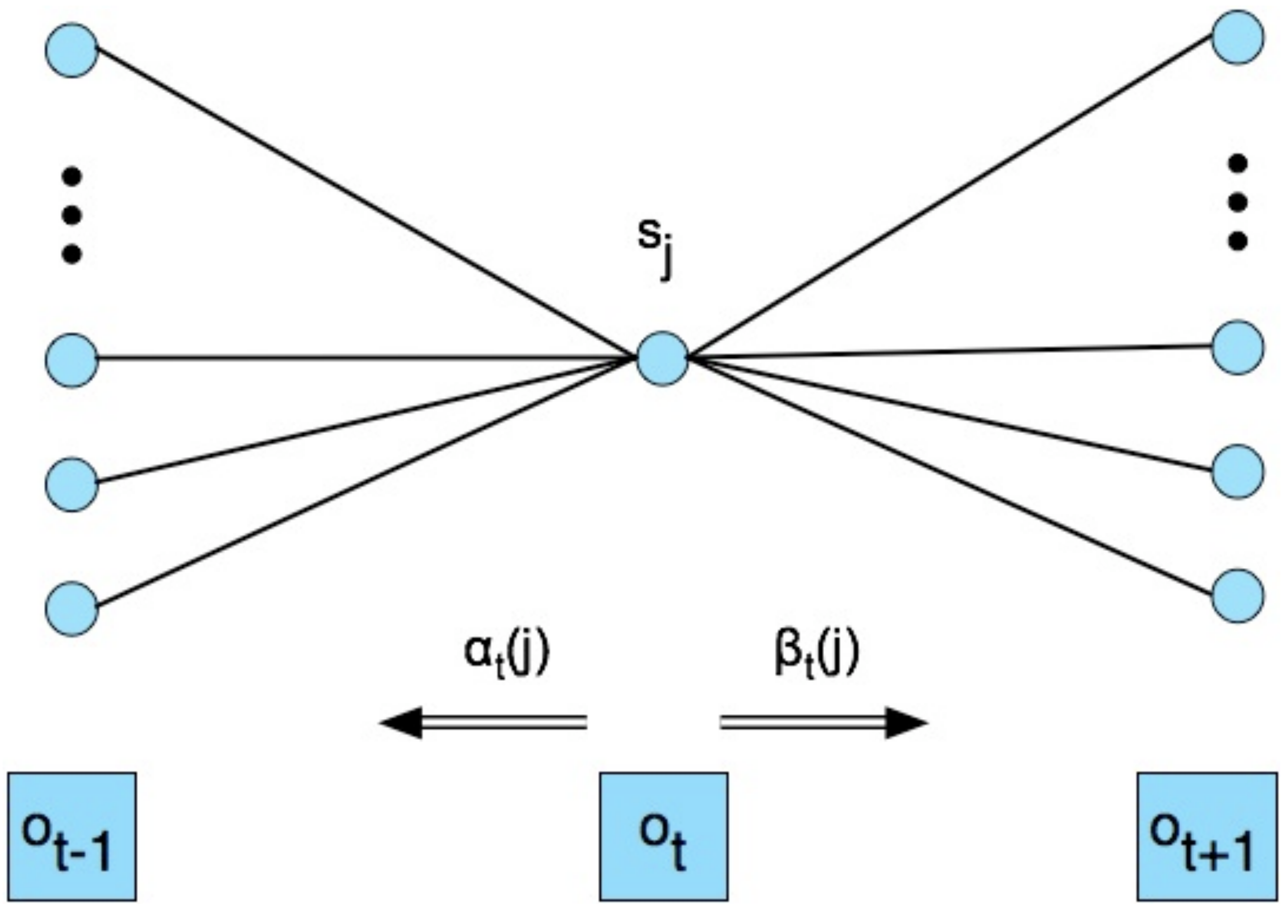
$$\gamma_t(j) = P(q_t = j|O,\lambda)$$

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

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

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

Computing γ (gamma)



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

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

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 Alg

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

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

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

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)} \qquad \hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

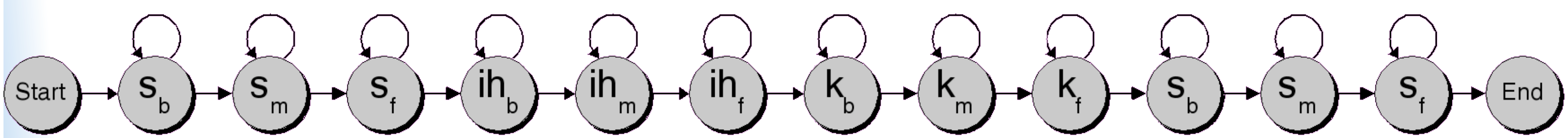
return A, B

Summary: Forward-Backward Algorithm

1) Intialize Φ=(A,B)
2) Compute α, β, ξ
3) Estimate new Φ'=(A,B)
4) Replace Φ with Φ'
5) 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:

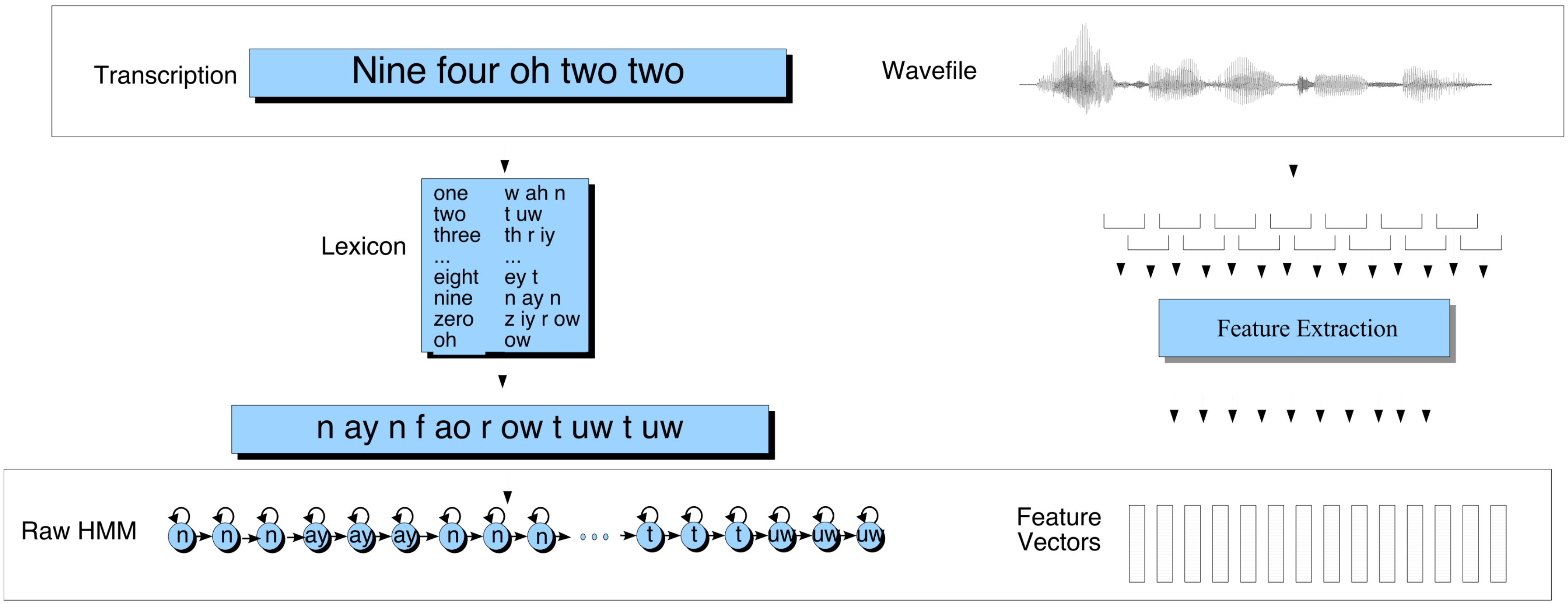


- Baum-Welch only guaranteed to return local max, rather than global optimum

Complete Embedded Training

- Setting all the parameters in an ASR system
- Given:
  - training set: wavefiles & word transcripts for each sentence
  - Hand-built HMM lexicon
- Uses:
  - Baum-Welch algorithm
- We'll return to this after we've introduced GMMs

Embedded Training



What we are searching for

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

AM (likelihood)   LM (prior)

$$(1) \quad \hat{W} = \underset{W \in L}{\arg\max}\, P(O \mid W)\, P(W)$$

Combining Acoustic and Language Models

- We don't actually use equation (1)

$$(1)\quad \hat{W} = \underset{W \in L}{\arg\max}\, P(O|W)P(W)$$

  - 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 (as we've seen) 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

$$(2)\quad \hat{W} = \underset{W \in L}{\arg\max}\, P(O \mid W) P(W)^{LMSF}$$

- Value determined empirically, is positive (why?)
- Often in the range 10 +- 5.

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

$$(3)\quad \hat{W} = \operatorname*{argmax}_{W \in L} P(O \mid W) P(W)^{LMSF} WIP^{N(W)}$$

Log domain

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

$$(4)\quad \hat{W} = \underset{W \in L}{\operatorname{argmax}} \log P(O \mid W) + LMSF \log P(W) + N \log WIP$$

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

Text from Bryan Pellom’s slides

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)

Text augmented from Bryan Pellom’s slides

Summary

- Speech Recognition Architectural Overview
- Hidden Markov Models in general
  - Forward
  - Viterbi Decoding
- Hidden Markov models for Speech
- Evaluation