

Lecture 11: ASR: Training & Systems

- 1 Training HMMs
- 2 Language modeling
- 3 Discrimination & adaptation

Dan Ellis <dpwe@ee.columbia.edu>
<http://www.ee.columbia.edu/~dpwe/e6820/>

Columbia University Dept. of Electrical Engineering
Spring 2003



1

HMM review

- HMM M_j is specified by:

- states q^i



- transition probabilities a_{ij}



$$p(q_n^j | q_{n-1}^i) \equiv a_{ij}$$

	k	a	t	•
•	1.0	0.0	0.0	0.0
k	0.9	0.1	0.0	0.0
a	0.0	0.9	0.1	0.0
t	0.0	0.0	0.9	0.1

- emission distributions $b_i(x)$



$$p(x | q^i) \equiv b_i(x)$$



+ (initial state probabilities $p(q_1^i) \equiv \pi_i$)

- See e6820/papers/Rabiner89-hmm.pdf



HMM summary (1)

- **HMMs are a generative model:**
recognition is **inference** of $p(M_j|X)$
- **During generation, behavior of model depends only on current state q_n :**
 - **transition** probabilities $p(q_{n+1} | q_n) = a_{ij}$
 - **observation** distributions $p(x_n | q_n) = b_i(x)$
- **Given states** $Q = \{q_1, q_2, \dots, q_N\}$

+ **observations** $X = X_1^N = \{x_1, x_1, \dots, x_N\}$

Markov assumption makes

$$p(X, Q|M) = \prod_n p(x_n|q_n)p(q_n|q_{n-1})$$

- **Given **observed emissions** X , can calculate:**
$$p(X|M_j) = \sum_{\text{all } Q} p(X|Q, M)p(Q|M)$$



HMM summary (2)

- Calculate $p(X|M)$ via **forward recursion**:

$$p(X_1^n, q_n^j) = \alpha_n(j) = \left[\sum_{i=1}^S \alpha_{n-1}(i) a_{ij} \right] \cdot b_j(x_n)$$

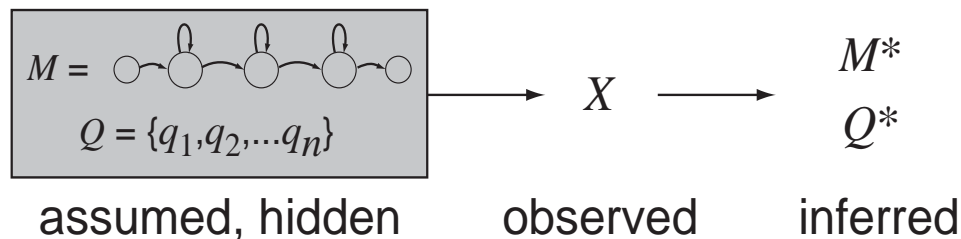
- **Viterbi (best path) approximation**

$$\alpha_n^*(j) = \left[\max_i \{ \alpha_{n-1}^*(i) a_{ij} \} \right] \cdot b_j(x_n)$$

- then backtrace...

$$Q^* = \operatorname{argmax}_Q p(X, Q|M)$$

- **Pictorially:**



Outline

- 1 Hidden Markov Model review
- 2 **Training HMMs**
 - Viterbi training
 - EM for HMM parameters
 - Forward-backward (Baum-Welch)
- 3 Language modeling
- 4 Discrimination & adaptation



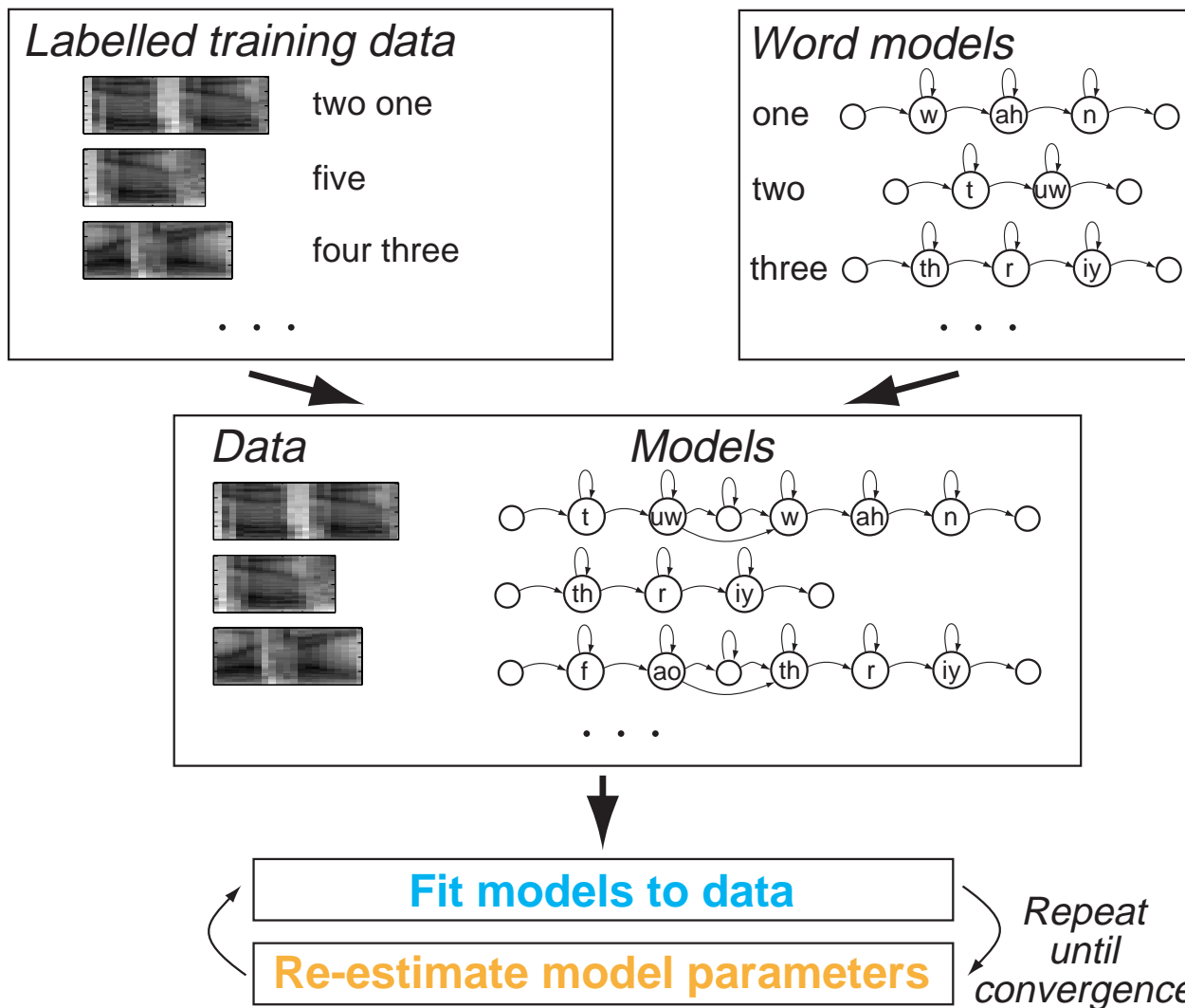
2

Training HMMs

- **Probabilistic foundation allows us to **train** HMMs to ‘fit’ training data**
 - i.e. estimate $a_{ij}, b_i(x)$ given data
 - better than DTW...
 - **Algorithms to improve $p(M | X)$ are key to success of HMMs**
 - **maximum-likelihood** of models...
 - **State alignments Q of training examples are generally unknown**
 - else estimating parameters would be easy
- **Viterbi training**
- choose ‘best’ labels (heuristic)
- **EM training**
- ‘fuzzy labels’ (guaranteed local convergence)

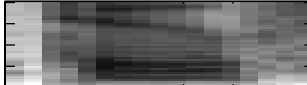


Overall training procedure



Viterbi training

- “Fit models to data”
= Viterbi best-path alignment

Data 

Viterbi labels Q^* 

- “Re-estimate model parameters”:

pdf e.g. 1D Gauss: $\mu_i = \frac{\sum_{n \in q^i} x_n}{\#(q_n^i)}$

count transitions: $a_{ij} = \frac{\#(q_{n-1}^i \rightarrow q_n^j)}{\#(q_n^i)}$

- And repeat...
- But: converges only if good initialization



EM for HMMs

- **Expectation-Maximization (EM):**
optimizes models with unknown parameters
 - finds locally-optimal parameters Θ
to maximize data likelihood $p(x_{train} | \Theta)$
 - makes sense for decision rules like
 $p(x | M_j) \cdot p(M_j)$

- **Principle:**
Adjust Θ to maximize expected log likelihood of known x & unknown u :

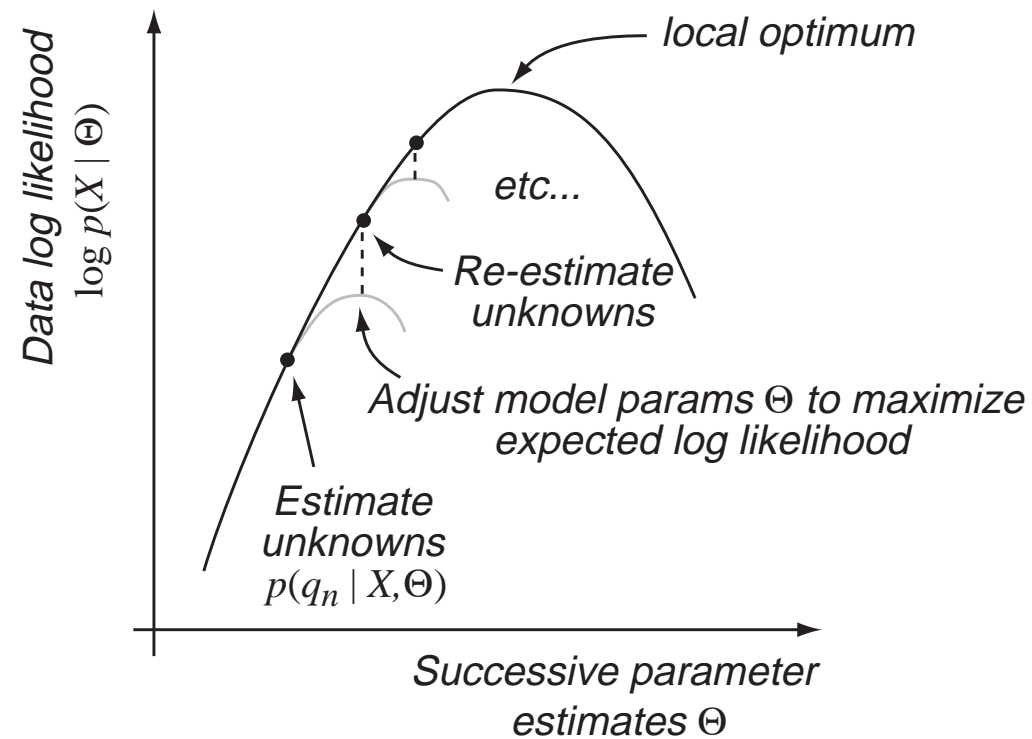
$$E[\log p(x, u | \Theta)] = \sum_u p(u | x, \Theta_{old}) \log [p(x | u, \Theta) p(u | \Theta)]$$

- for GMMs, unknowns = mix assignments k
 - for HMMs, unknowns = hidden state q_n
(take Θ to include M_j)
- **Interpretation: “fuzzy” values for unknowns**



What EM does

- Maximize data likelihood by repeatedly estimating unknowns and re-maximizing expected log likelihood:



EM for HMMs (2)

- **Expected log likelihood for HMM:**

$$\begin{aligned} & \sum_{\text{all } Q_k} p(Q_k | X, \Theta_{\text{old}}) \log [p(X | Q_k, \Theta) p(Q_k | \Theta)] \\ &= \sum_{\text{all } Q_k} p(Q_k | X, \Theta_{\text{old}}) \log \left[\prod_n p(x_n | q_n) \cdot p(q_n | q_{n-1}) \right] \\ &= \sum_{n=1}^N \sum_{i=1}^S p(q_n^i | X, \Theta_{\text{old}}) \log p(x_n | q_n^i, \Theta) \\ & \quad + \sum_{i=1}^S p(q_1^i | X, \Theta_{\text{old}}) \log p(q_1^i | \Theta) \\ & \quad + \sum_{n=2}^N \sum_{i=1}^S \sum_{j=1}^S p(q_{n-1}^i, q_n^j | X, \Theta_{\text{old}}) \log p(q_n^j | q_{n-1}^i, \Theta) \end{aligned}$$

- closed-form **maximization** by differentiation etc.



EM update equations

- **For acoustic model (e.g. 1-D Gauss):**

$$\mu_i = \frac{\sum_n p(q_n^i | X, \Theta_{\text{old}}) \cdot x_n}{\sum_n p(q_n^i | X, \Theta_{\text{old}})}$$

- **For transition probabilities:**

$$p(q_n^j | q_{n-1}^i) = a_{ij}^{\text{new}} = \frac{\sum_n p(q_{n-1}^i, q_n^j | X, \Theta_{\text{old}})}{\sum_n p(q_{n-1}^i | X, \Theta_{\text{old}})}$$

- **Fuzzy versions of Viterbi training**

- reduce to Viterbi if $p(q|X) = 1/0$

- **Require ‘state occupancy probabilities’,**

$$p(q_n^i | X_1^N, \Theta_{\text{old}})$$



The forward-backward algorithm

- We need $p(q_n^i | X_1^N)$ for EM updates (Θ implied)
- Forward algorithm gives $\alpha_n(i) = p(q_n^i, X_1^n)$ ←
 - excludes influence of remaining data X_{n+1}^N
- Hence, define $\beta_n(i) = p(X_{n+1}^N | q_n^i, X_1^n)$
so that $\alpha_n(i) \cdot \beta_n(i) = p(q_n^i, X_1^N)$ ←
then
$$p(q_n^i | X_1^N) = \frac{\alpha_n(i) \cdot \beta_n(i)}{\sum_j \alpha_n(j) \cdot \beta_n(j)}$$
- **Recursive** definition for β :
$$\beta_n(i) = \sum_j \beta_{n+1}(j) a_{ij} b_j(x_{n+1})$$
 - recurses **backwards** from final state N



Estimating a_{ij} from α & β

- From EM equations:

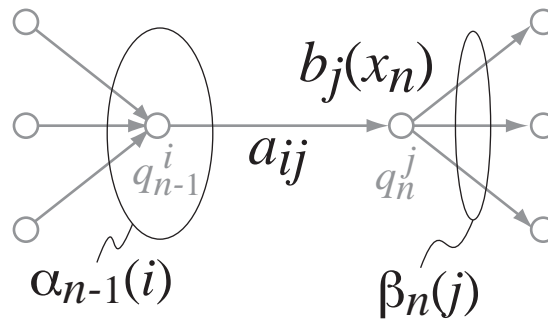
$$p(q_n^j | q_{n-1}^i) = a_{ij}^{\text{new}} = \frac{\sum_n p(q_{n-1}^i, q_n^j | X, \Theta_{\text{old}})}{\sum_n p(q_{n-1}^i | X, \Theta_{\text{old}})}$$

- prob. of transition normalized by prob. in first

- Obtain from $p(q_{n-1}^i, q_n^j, X | \Theta_{\text{old}})$

$$= p(X_{n+1}^N | q_n^j) p(x_n | q_n^j) p(q_n^j | q_{n-1}^i) p(q_{n-1}^i, X_1^{n-1})$$

$$= \beta_n(j) \cdot b_j(x_n) \cdot a_{ij} \cdot \alpha_{n-1}(i)$$



GMM-HMMs in practice

- **GMMs as acoustic models:**
train by including **mixture indices** as unknowns
 - just more complicated equations...

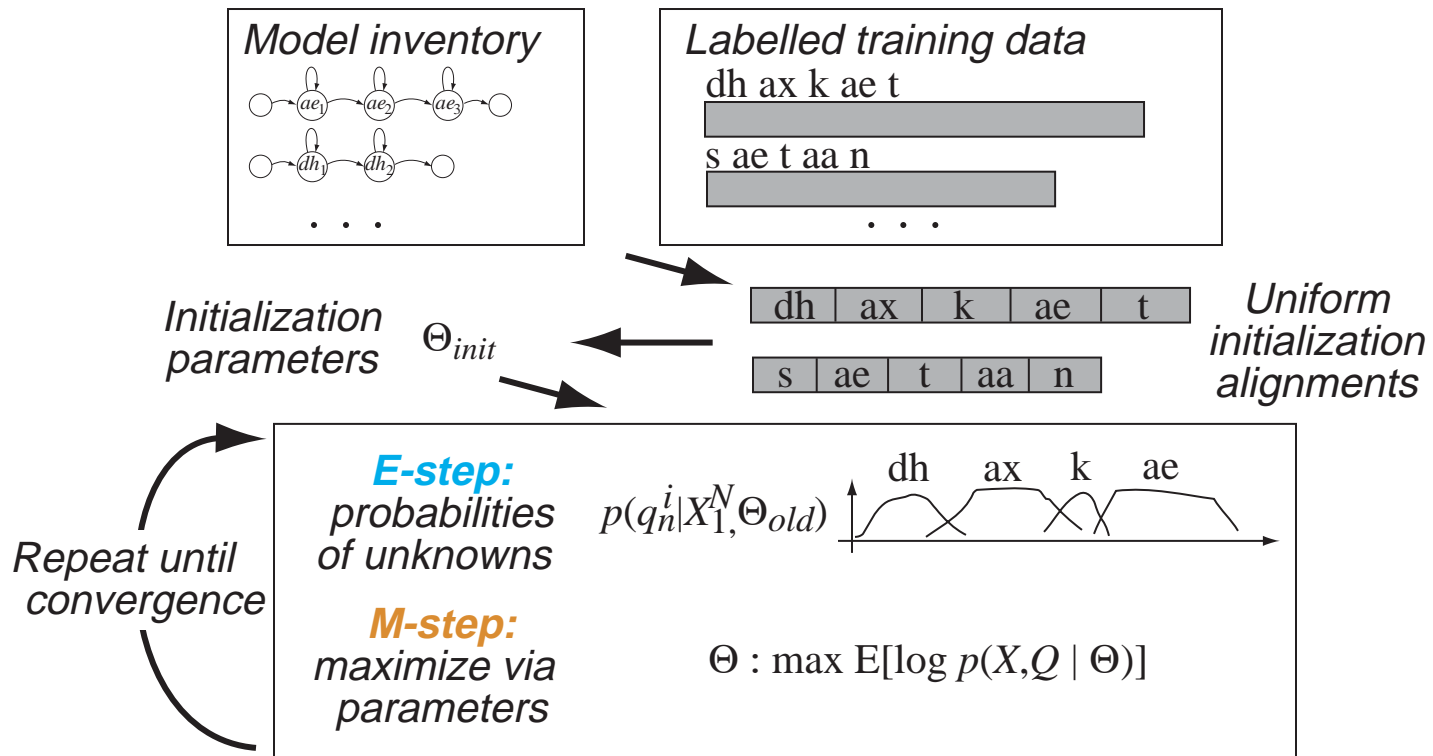
$$\mu_{ik} = \frac{\sum_n p(m_k | q^i, x_n, \Theta_{\text{old}}) p(q_n^i | X, \Theta_{\text{old}}) \cdot x_n}{\sum_n p(m_k | q^i, x_n, \Theta_{\text{old}}) p(q_n^i | X, \Theta_{\text{old}})}$$

- **Practical GMMs:**
 - 9 to 39 feature dimensions
 - 2 to 64 Gaussians per mixture
depending on number of training examples
- **Lots of data → can model more classes**
 - e.g context-independent (CI): $q^i = \mathbf{ae aa ax} \dots$
→ **context-dependent** (CD): $q^i = \mathbf{b-ae-b b-ae-k} \dots$



HMM training in practice

- EM only finds **local** optimum
 - **critically dependent on initialization**
 - approximate parameters / rough alignment

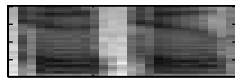


- **Applicable for more than just words...**

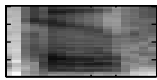


Training summary

- **Training data + basic model topologies**
→ **derive fully-trained models**

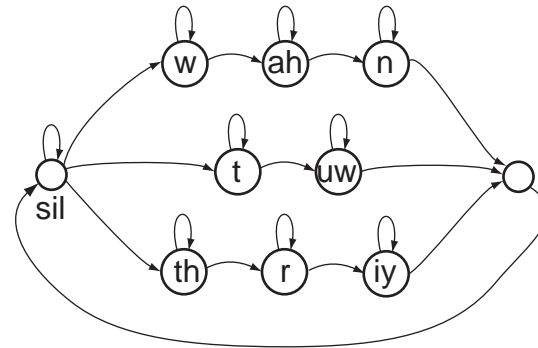


TWO ONE

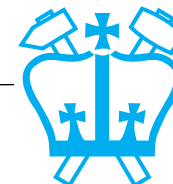


FIVE

ONE = w ah n
TWO = t uw



- alignment all handled implicitly
- **What do the states end up meaning?**
 - not necessarily what you intended;
whatever locally maximizes data likelihood
- **What if the models or transcriptions are bad?**
 - slow convergence, poor discrimination in models
- **Other kinds of data, transcriptions**
 - less constrained initial models...



Outline

- 1 Hidden Markov Models review
- 2 Training HMMs
- 3 Language modeling**
 - Pronunciation models
 - Grammars
 - Decoding
- 4 Discrimination & adaptation



3

Language models

- **Recall, MAP recognition criterion:**

$$\begin{aligned} M^* &= \operatorname{argmax}_{M_j} p(M_j | X, \Theta) \\ &= \operatorname{argmax}_{M_j} p(X | M_j, \Theta_A) p(M_j | \Theta_L) \end{aligned}$$

- **So far, looked at** $p(X | M_j, \Theta_A)$
- **What about** $p(M_j | \Theta_L)$?
 - M_j is a particular **word sequence**
 - Θ_L are parameters related to the **language**
- **Two components:**
 - link state sequences to words $p(Q | w_i)$
 - priors on word sequences $p(w_i | M_j)$



HMM Hierarchy

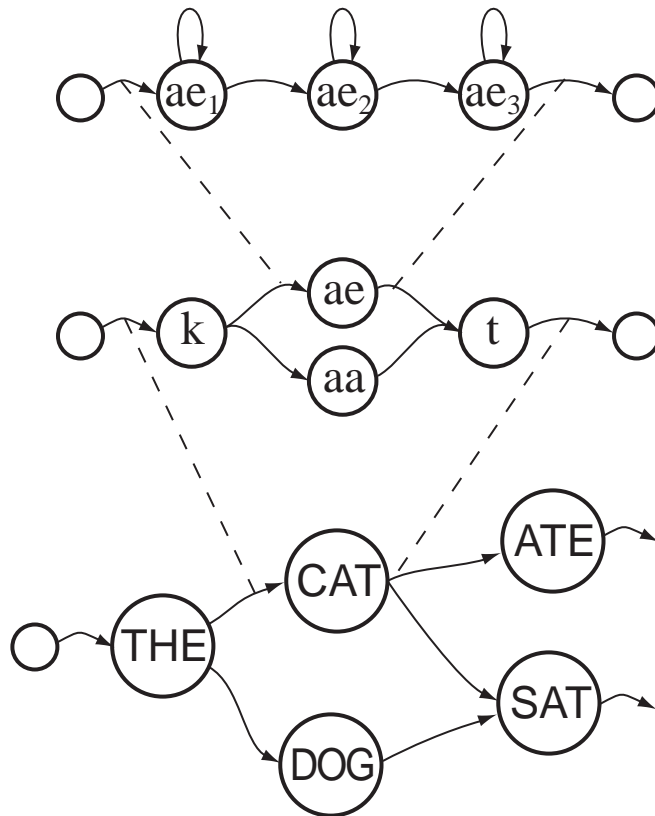
- HMMs support **composition**
 - can handle time dilation, pronunciation, grammar all within the same framework

$$p(q|M) = p(q, \Phi, w|M)$$

$$= p(q|\phi) \cdot$$

$$p(\phi|w) \cdot$$

$$p(w_n | w_1^{n-1}, M)$$



Pronunciation models

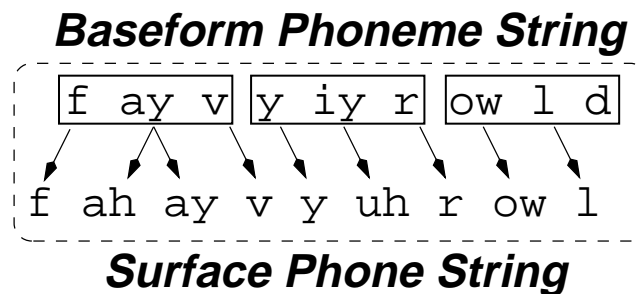
- Define states within each word $p(Q|w_i)$
- Can have **unique states** for each word ('whole-word' modeling), or ...
- Sharing (tying) **subword units** between words to reflect underlying phonology
 - more training examples for each unit
 - generalizes to unseen words
 - (or can do it automatically...)
- Start e.g. from pronouncing **dictionary**:

```
ZERO(0.5)    z i y r ow
ZERO(0.5)    z i h r ow
ONE(1.0)     w a h n
TWO(1.0)     t c l t uw
... 
```

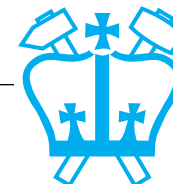


Learning pronunciations

- ‘Phone recognizer’ transcribes training data as phones
 - align to ‘canonical’ pronunciations



- infer modification rules
 - predict other pronunciation variants
- e.g. ‘**d deletion**’:
 $d \rightarrow \emptyset / l _ [\text{stop}] \quad p = 0.9$
- **Generate pronunciation variants;**
use forced alignment to find weights



Grammar

- **Account for different likelihoods of different words and word sequences** $p(w_i | M_j)$
- **'True' probabilities are very complex for LVCSR**
 - need parses, but speech often agrammatic

→ **Use n-grams:**

$$p(w_n | w_1^L) = p(w_n | w_{n-K}, \dots, w_{n-1})$$

- e.g. n-gram models of Shakespeare:

- n=1 To him swallowed confess hear both. Which. Of save on ...
- n=2 What means, sir. I confess she? then all sorts, he is trim, ...
- n=3 Sweet prince, Falstaff shall die. Harry of Monmouth's grave...
- n=4 King Henry. What! I will go seek the traitor Gloucester. ...

- **Big win in recognizer WER**
 - raw recognition results often highly ambiguous
 - grammar guides to 'reasonable' solutions



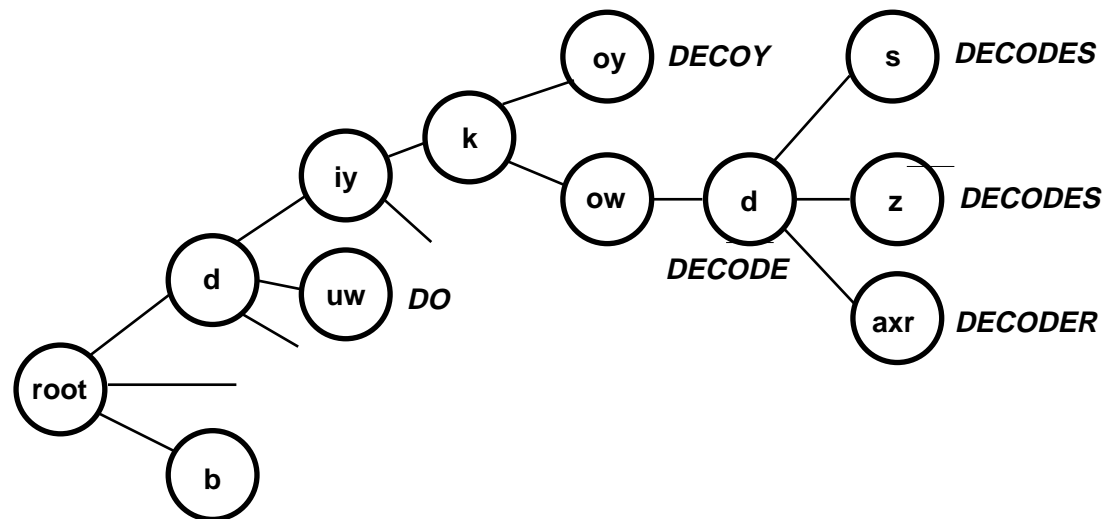
Smoothing LVCSR grammars

- **n-grams (n=3 or 4) are estimated from large text corpora**
 - 100M+ words
 - but: not like spoken language
- **100,000 word vocabulary → 10^{15} trigrams!**
 - never see enough examples
 - unobserved trigrams should NOT have $Pr=0$!
- **Backoff to bigrams, unigrams**
 - $p(w_n)$ as an approx to $p(w_n | w_{n-1})$ etc.
 - interpolate 1-gram, 2-gram, 3-gram with learned weights?
- **Lots of ideas e.g. category grammars**
 - e.g. $p(\text{PLACE} | \text{“went”, “to”}) \cdot p(w_n | \text{PLACE})$
 - how to define categories?
 - how to tag words in training corpus?



Decoding

- **How to find the MAP word sequence?**
 - **States, prons, words define one big HMM**
 - with 100,000+ individual states for LVCSR!
- **Exploit hierarchic structure**
- phone states independent of word
 - next word (semi) independent of word history



Decoder pruning

- **Searching ‘all possible word sequences’?**
 - need to restrict search to most promising ones:
beam search
 - sort by estimates of total probability
= Pr(so far) + lower bound estimate of remains
 - trade **search errors** for speed

- **Start-synchronous algorithm:**

- extract top hypothesis from queue:

$$[P_n, \{w_1, \dots, w_k\}, n]$$

pr. so far words next time frame

- find plausible words $\{w^i\}$ starting at time n
→ new hypotheses:

$$[P_n \cdot p(X_n^{n+N-1} | w^i) \cdot p(w^i | w_k \dots), \{w_1, \dots, w_k, w^i\}, n + N]$$

- discard if too unlikely, or queue is too long
- else re-insert into queue and repeat



Outline

- 1 Hidden Markov Models review
- 2 Training HMMs
- 3 Language modeling
- 4 **Discrimination & adaptation**
 - Discriminant models
 - Neural net acoustic models
 - Model adaptation



4

Discriminant models

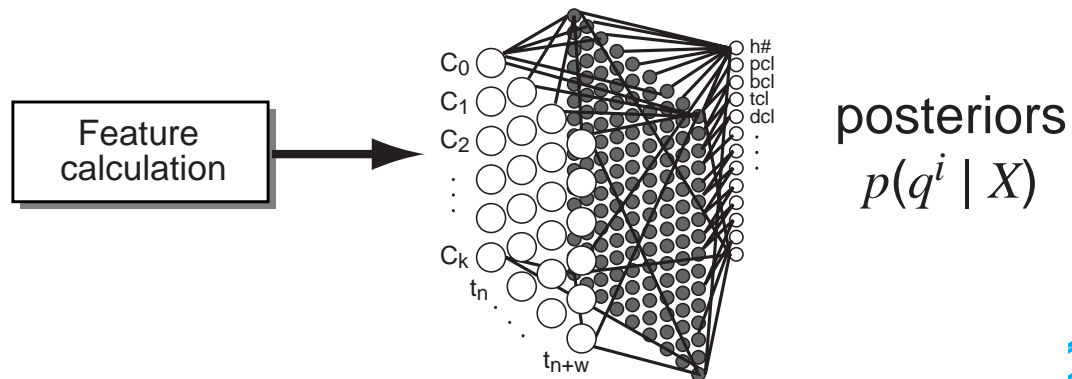
- **EM training of HMMs is maximum likelihood**
 - i.e. choose single Θ to $\max p(X_{trn} | \Theta)$
 - **Bayesian** approach: actually $p(\Theta | X_{trn})$
- **Decision rule is** $\max p(X | M) \cdot p(M)$
 - training will increase $p(X | M_{correct})$
 - may also increase $p(X | M_{wrong})$...as much?
- **Discriminant training tries directly to increase discrimination between right & wrong models**
 - e.g. Maximum Mutual Information (MMI)

$$\begin{aligned}
 I(M_j, X | \Theta) &= \log \frac{p(M_j, X | \Theta)}{p(M_j | \Theta) p(X | \Theta)} \\
 &= \log \frac{p(X | M_j, \Theta)}{\sum_k p(X | M_k, \Theta) p(M_k | \Theta)}
 \end{aligned}$$



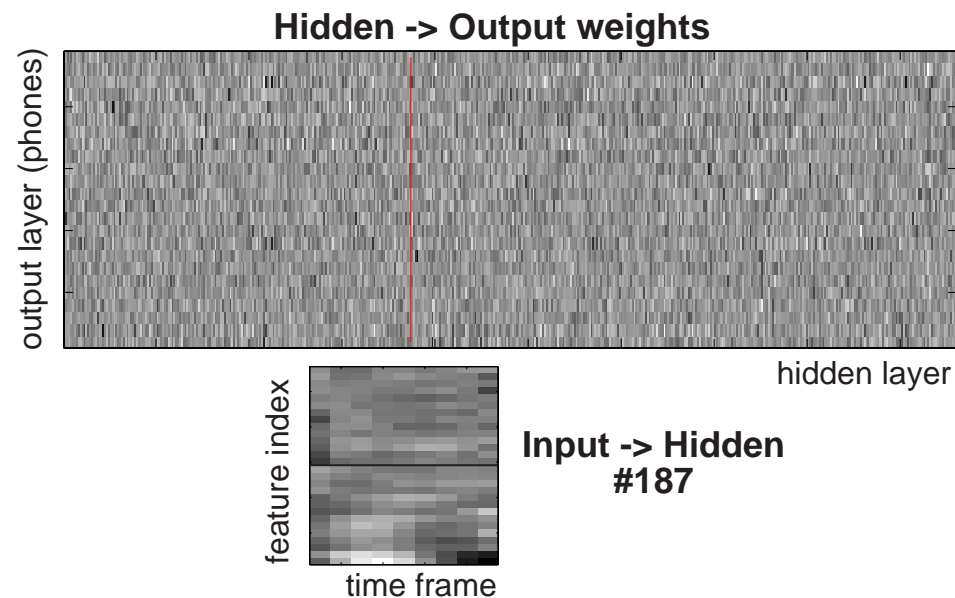
Neural Network Acoustic Models

- **Single model generates posteriors directly for all classes at once = frame-discriminant**
- **Use regular HMM decoder for recognition**
 - set $b_i(x_n) = p(x_n | q^i) \propto p(q^i | x_n) / p(q^i)$
- **Nets are less sensitive to input representation**
 - skewed feature distributions
 - correlated features
- **Can use temporal context window to let net 'see' feature dynamics:**



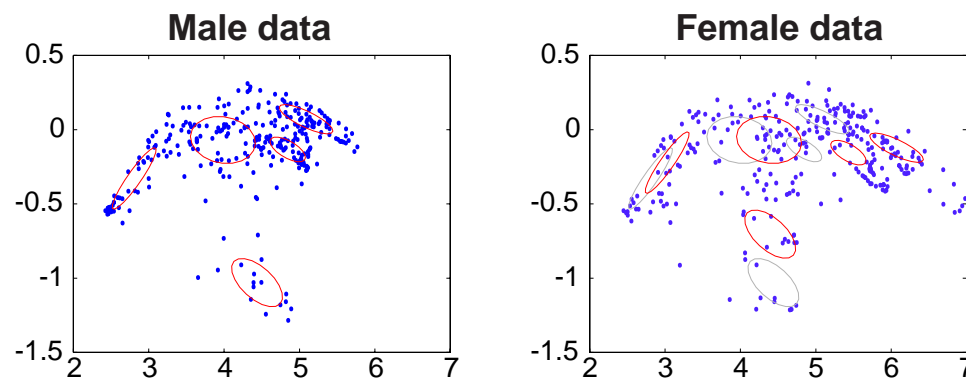
Neural nets: Practicalities

- **Typical net sizes:**
 - input layer: 9 frames x 9-40 features ~ 300 units
 - hidden layer: 100-8000 units, dep. train set size
 - output layer: 30-60 context-independent phones
- **Hard to make context dependent**
 - problems training many classes that are similar?
- **Representation is partially opaque:**

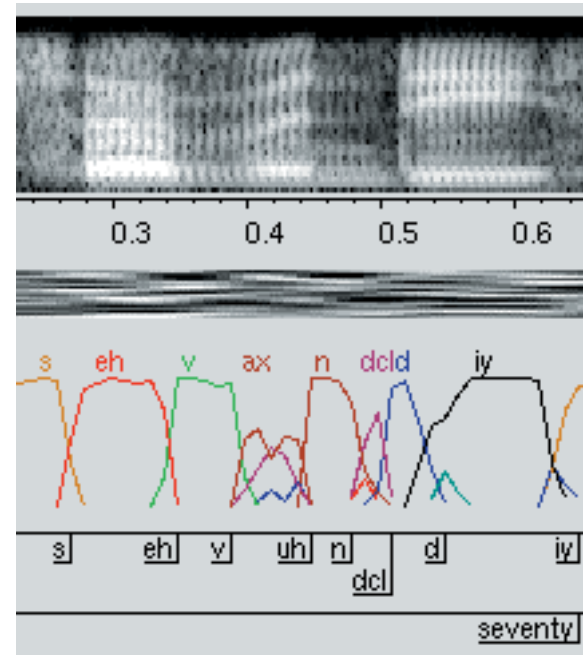
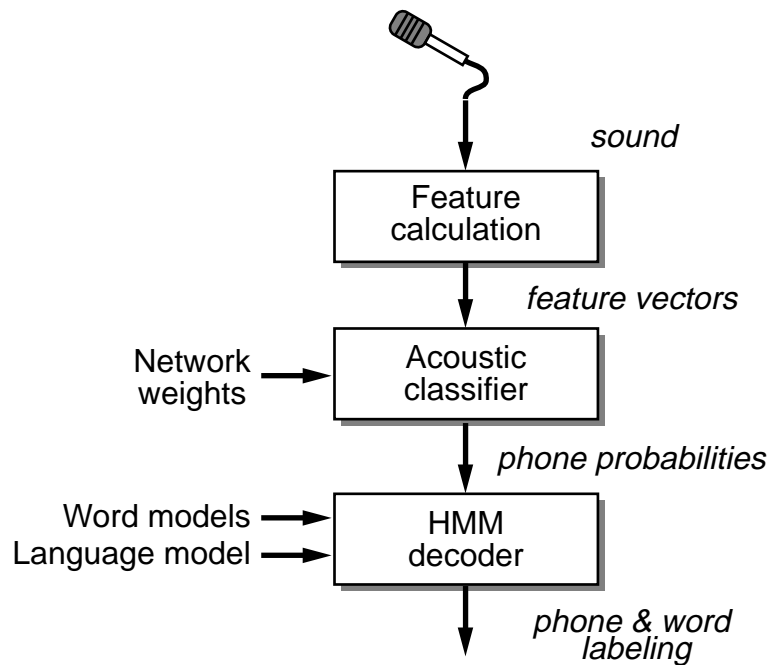


Model adaptation

- **Practical systems often suffer from mismatch**
 - test conditions are not like training data:
accent, microphone, background noise ...
- **Desirable to continue tuning during recognition = adaptation**
 - but: no 'ground truth' labels or transcription
- **Assume that recognizer output is correct; Estimate a few parameters from those labels**
 - e.g. Maximum Likelihood Linear Regression (MLLR)



Recap: Recognizer Structure



- **Now we have it all!**



Summary

- **Hidden Markov Models**
 - state transitions and emission likelihoods in one
 - best path (Viterbi) performs recognition
- **HMMs can be trained**
 - Viterbi training makes intuitive sense
 - EM training is guaranteed to converge
 - acoustic models (e.g. GMMs) train at same time
- **Language modeling captures higher structure**
 - pronunciation, word sequences
 - fits directly into HMM state structure
 - need to 'prune' search space in decoding
- **Further improvements...**
 - discriminant training moves models 'apart'
 - adaptation adjusts models in new situations

