TTIC 31110
Speech Technologies

April 28, 2020

Announcements

- HW1 grades released
- HW3 posted, due May 8 7pm
- Please let us know ASAP of any issues with compute resources!
- Karen's office hours after lecture today, Ankita's office hours tomorrow 3:30pm

Hidden Markov models (HMMs)

- Can serve as alternative to "templates" and "path weights" in DTW
- Ubiquitous model for speech recognition
- Last week:
  - Solving the scoring and decoding problems
- Today:
  - Solving the training problem
  - Implementation issues, extensions to continuous ASR
  - Modeling sub-word units

Recap: Elements of a (discrete) HMM

- N: Number of states; state at time t: $q_t \in \{1, \ldots, N\}$
- $V = \{v_1, \ldots, v_M\}$: Set of M possible observation labels (or vectors, in general); observation at time t: $o_t \in V$
- $\pi = \{\pi_i\}$: Initial state distribution,
  $\pi_i = P(q_1 = i), \;\; 1 \leq i \leq N$
- $\mathbf{A} = \{a_{ij}\}$: $N \times N$ state transition probability matrix,
  $a_{ij} = P(q_{t+1} = j | q_t = i), \;\; 1 \leq i, j, \leq N$
- $\mathbf{B} = \{b_i(k)\}$: Observation (or *emission*) distribution in state i,
  $b_i(k) = P(o_t = v_k | q_t = i), \;\; 1 \leq i \leq N, 1 \leq k \leq M$

The entire model can be denoted $\lambda = \{\mathbf{A}, \mathbf{B}, \pi\}$

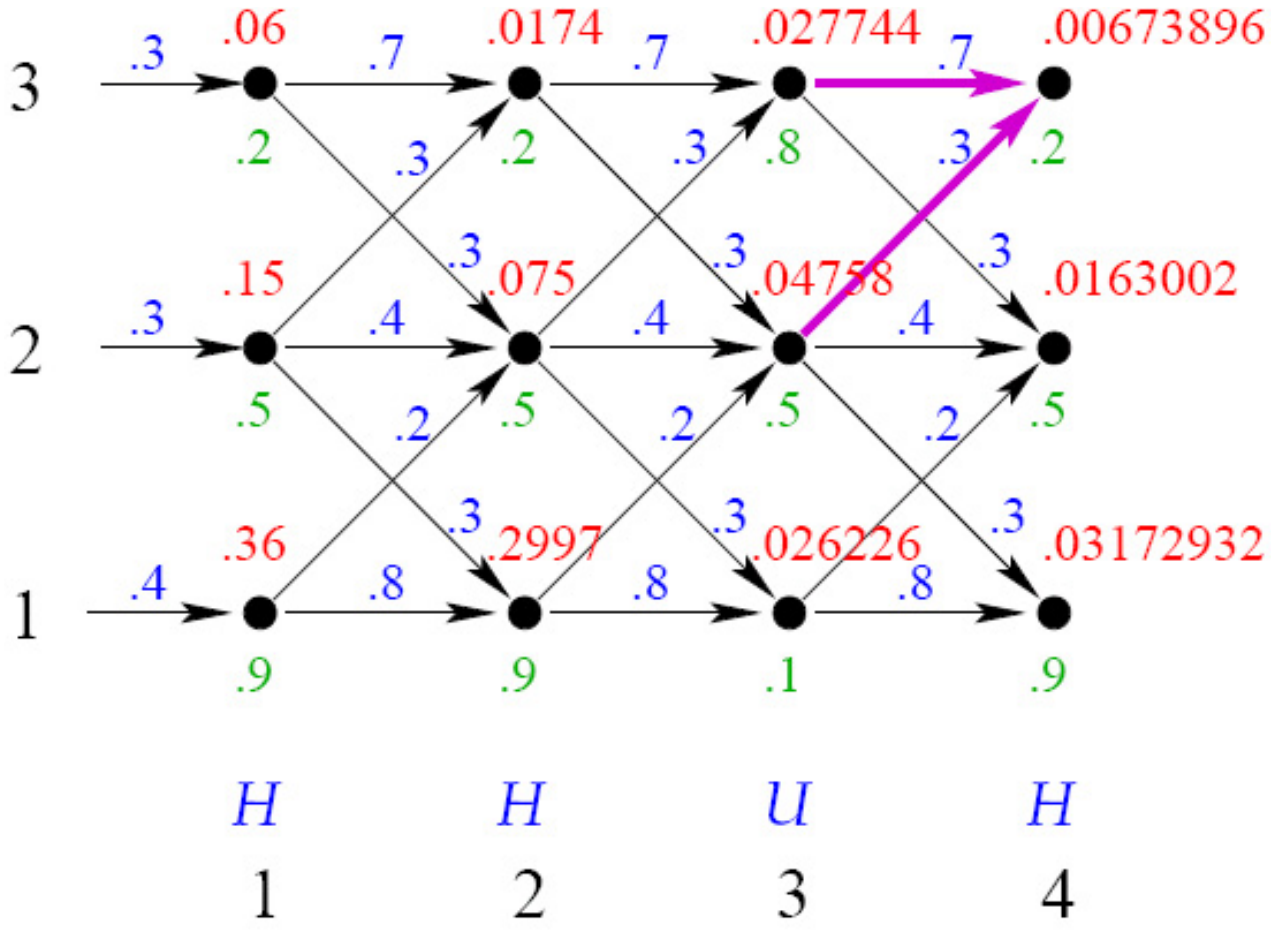
Recap: The three basic HMM problems

1. *Scoring*: Given an observation sequence $\mathbf{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_T\}$ and a model $\lambda = \{\mathbf{A}, \mathbf{B}, \pi\}$, how do we compute $P(\mathbf{O}|\lambda)$, the probability of the observation sequence?
   $\longrightarrow$ The forward & backward algorithms
2. *Decoding*: Given an observation sequence $\mathbf{O} = \{o_1, \ldots, o_T\}$, how do we choose the state sequence $\mathbf{q} = \{q_1, \ldots, q_T\}$ most likely to have generated the observations?
   $\longrightarrow$ The Viterbi algorithm
3. *Training*: Given a training set of observations $\mathbf{O}$, how do we set the model parameters $\lambda = \{\mathbf{A}, \mathbf{B}, \pi\}$ to maximize $P(\mathbf{O}|\lambda)$ (maximum-likelihood estimation)
   $\longrightarrow$ The Baum-Welch algorithm (EM applied to HMMs)

Recap: The forward algorithm for computing $p(O|\lambda)$

- A dynamic programming algorithm
- Define the "forward" variable $\alpha_t(i)$:
  $\alpha_t(i) = P(\mathbf{o}_1\mathbf{o}_2 \ldots \mathbf{o}_t, q_t = i|\lambda), \quad 1 \leq t \leq T, 1 \leq i \leq N$
- For $t = 1$: $\alpha_1(i) = \pi_i b_i(\mathbf{o}_1), 1 \leq i \leq N$
- For $t > 1$: Sum over all ways of getting to current state at time t:
  $$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] b_j(\mathbf{o}_t)$$
- Finally: $p(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$
- Calculation is on the order of N^2T.
  $N = 5, T = 100 \Rightarrow \sim 2500$ computations, instead of 10^{72}

Recap: The forward algorithm



Follow-up from last time: Space complexity is NT, as we need to store the forward variable for each node in the trellis

Recap: The backward algorithm for computing $p(O|\lambda)$

- Define the "backward" variable $\beta_t(i)$:

$$\beta_t(i) = P(\mathbf{o}_{t+1}\mathbf{o}_{t+2}\ldots\mathbf{o}_T|q_t = i, \lambda), \quad \begin{array}{rcl} 1 \leq & t & \leq T-1, \\ 1 \leq & i & \leq N \end{array}$$

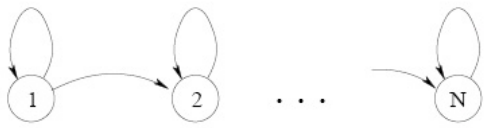
- Initialization: $\beta_T(i) = 1$
- Recursion: $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)$
- Finally: $p(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \pi_i b_i(\mathbf{o}_1) \beta_1(i)$
- As in forward algorithm, computation is N^2T
- Either algorithm alone can be used to compute $p(\mathbf{O}|\lambda)$, but both αs and βs will be needed for the training problem

Recap: Isolated-word recognition with whole-word HMMs

- Model each word w in the vocabulary with an HMM, with parameters λ_w
- Assuming words are equally likely, hypothesized word w^* is the one with the highest $p(\mathbf{O}|\lambda_w)$:

$$\begin{aligned} w^* &= \operatorname*{argmax}_w p(\lambda_w|\mathbf{O}) \\ &= \operatorname*{argmax}_w p(\mathbf{O}|\lambda_w)p(\lambda_w) \end{aligned}$$

- Typical HMMs for ASR are left-to-right:

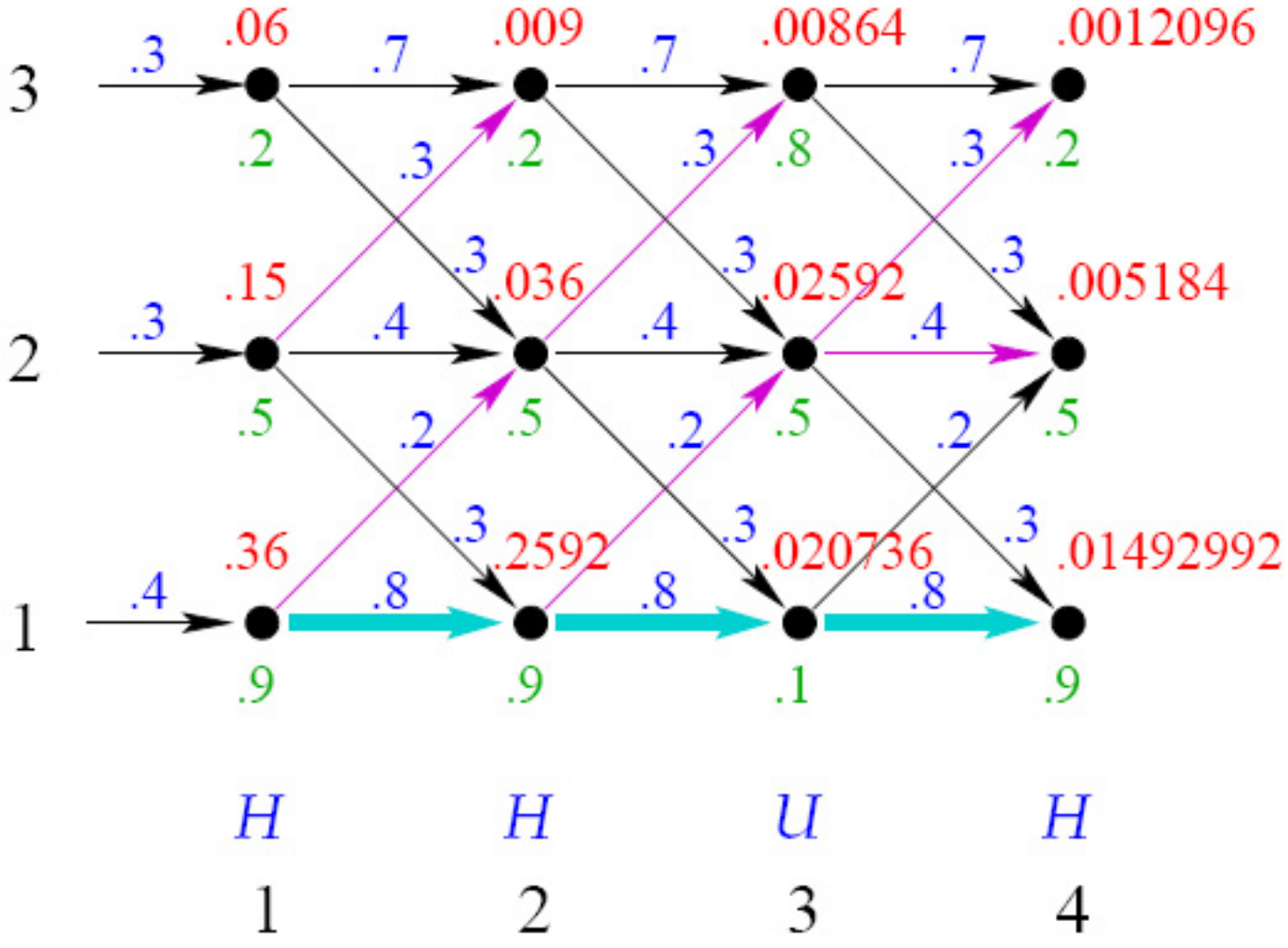


- Each state corresponds roughly to a "stationary" state of the vocal tract

Recap: The Viterbi algorithm for finding the most likely state sequence

- Define $\delta_t(i)$ as the highest probability along a single path to state i at time t that accounts for the first t observations:
  $\delta_t(i) = \max_{q_1,\ldots,q_{t-1}} P(q_1 q_2 \ldots q_t = i, \mathbf{o}_1 \mathbf{o}_2 \ldots \mathbf{o}_t | \lambda)$
- Initialization: $\delta_1(i) = \pi_i b_i(\mathbf{o}_1)$ (Note: this $= \alpha_1(i)$)
- Recursion: $\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] \, b_j(\mathbf{o}_t), \quad 2 \le t \le T$
- Note the similarity to the α recursion:
  $$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \right] b_j(\mathbf{o}_t)$$
  - Same as forward algorithm, but sum replaced by max!
  - To retrieve the best path, also keep a back-pointer from each node to best incoming partial path (as in DTW)
- As in forward algorithm, computation is $\approx N^2 T$

The Viterbi algorithm: Example



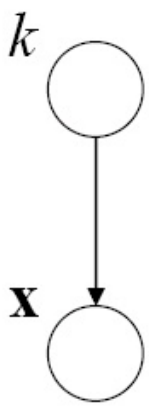
Summary

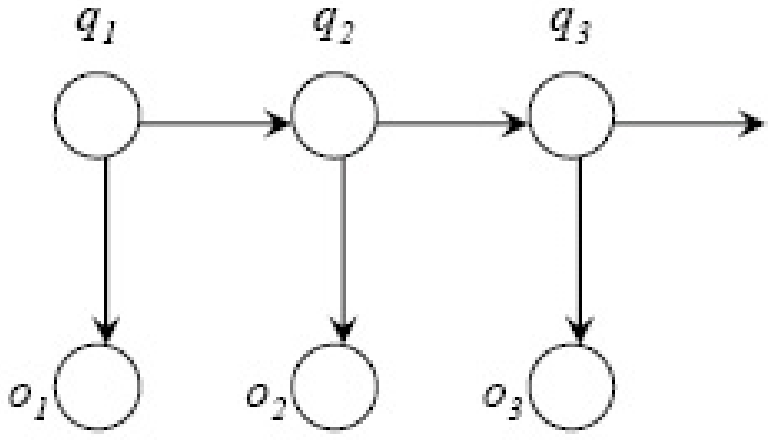
So far:

- Problem 1 (scoring): Find probability of a given observation sequence, given a model $\lambda = \{\pi, \mathbf{A}, \mathbf{B}\}$. Sufficient to do isolated-word (or isolated-phone) recognition
- Problem 2 (decoding): Find most likely state sequence

HMMs as extension of GMMs

- We've argued why HMMs are an extension of DTW
- They are also an extension of GMMs, where observation distributions are Gaussian and "components" are not independent across data points
- Graphical model representation (*not transition diagrams*!) for GMMs and HMMs:



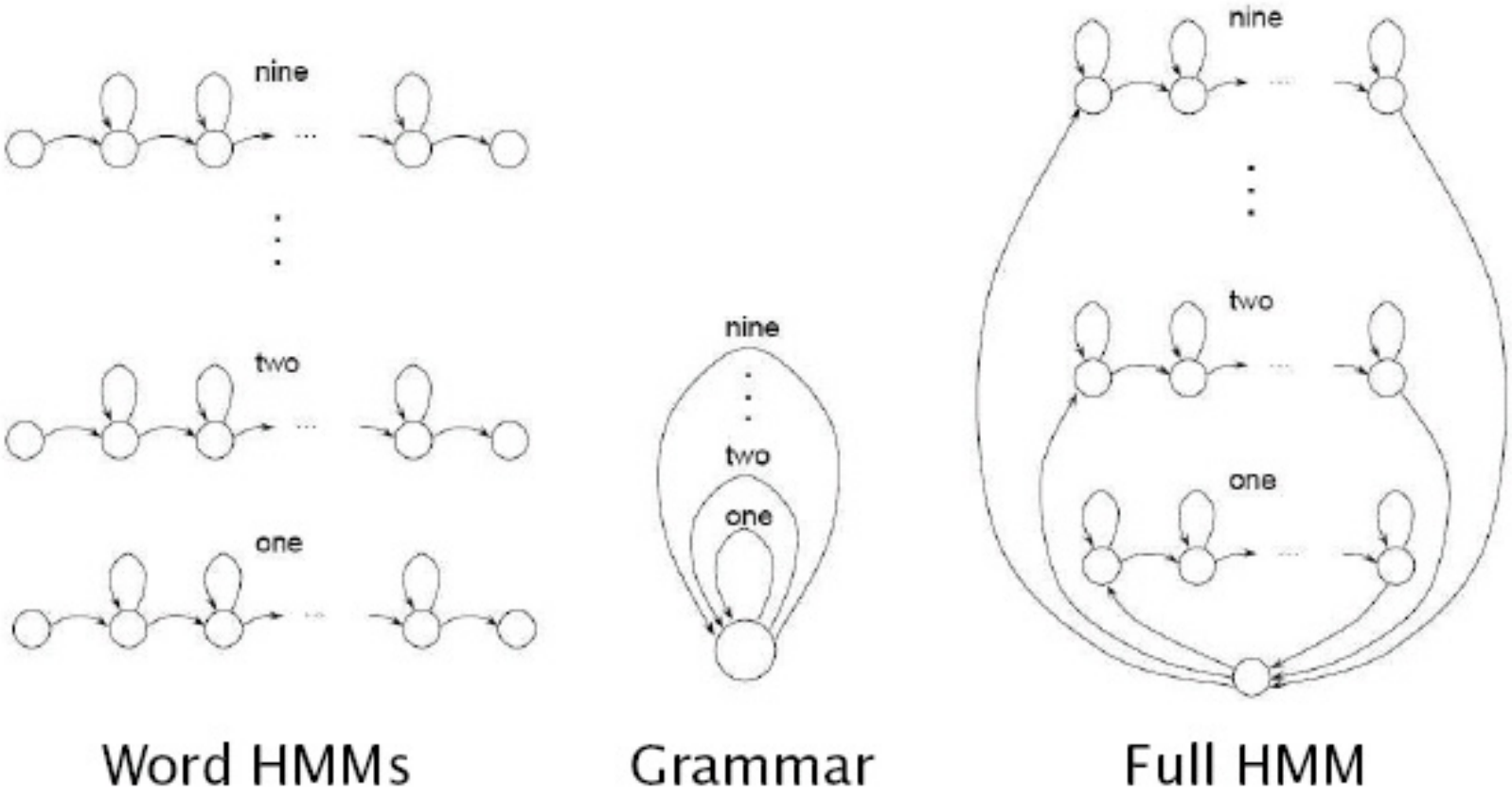


Handling underflow in HMM computations

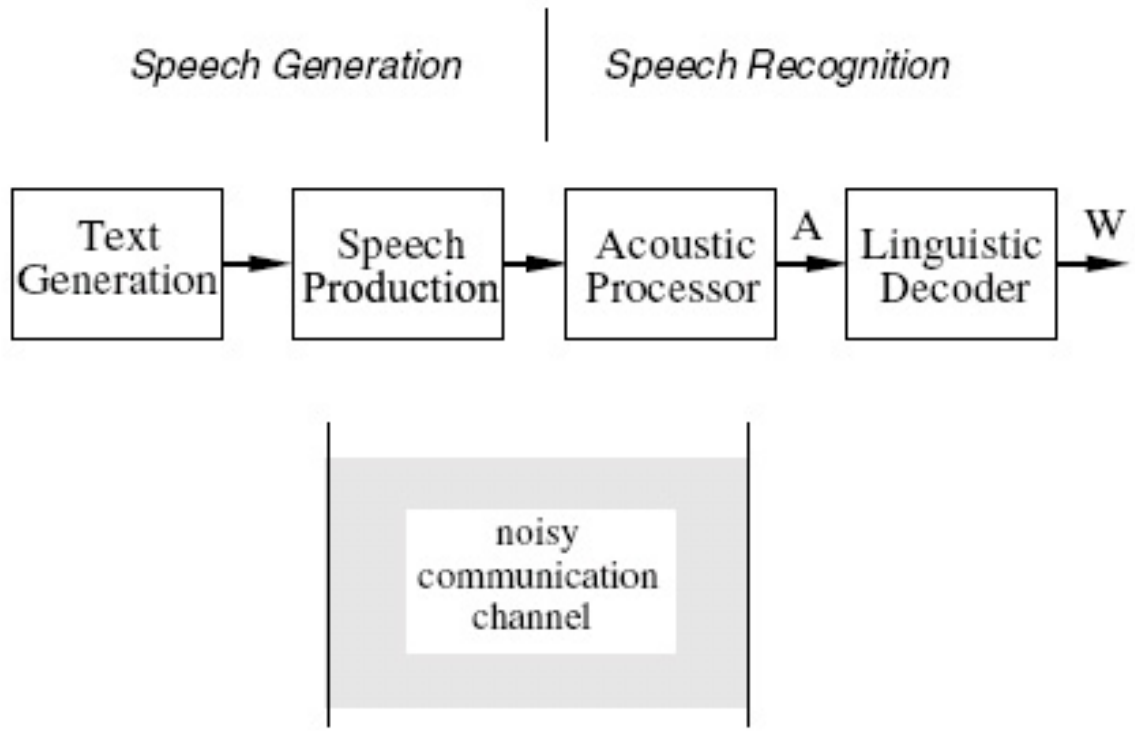
- The algorithms as we've described them don't work out of the box
  - Path probability for a 10-second waveform $\approx .5^{1000} \approx 9 \times 10^{-302}$
  - Path probability for a 20-second waveform $\approx .5^{2000} \approx 0$
- Preventing underflow in HMM computations
  - Viterbi decoding: Only multiplications involved; can sum logs instead of multiplying probabilities
  - Forward-backward: Both multiplications and additions involved; scale the probabilities at each time frame, and keep track of scale factors

Continuous speech recognition with HMMs

Most basic model: String together word HMMs to make sentence HMM, using a grammar (Note: Start and end states of these word HMMs are non-emitting)



Continuous ASR: The noisy channel model



Recognition = finding the most probable word string $\mathbf{w}^*$ given the acoustic observations $\mathbf{O}$: $\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}}\, p(\mathbf{w}|\mathbf{O})$

Continuous ASR: The noisy channel model (2)

From Bayes' rule, $p(\mathbf{w}|\mathbf{O}) = \frac{p(\mathbf{O}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{O})}$. Therefore,

$$\begin{aligned} \mathbf{w}^* &= \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{w}|\mathbf{O}) \\ &= \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{O}|\mathbf{w})p(\mathbf{w}) \end{aligned}$$

- This is the "fundamental equation" of speech recognition
- This is the same thing we did for isolated-word recognition, except now we have word *strings* $\mathbf{w}$
- The "whole-sentence" HMM gives $p(\mathbf{O}|\mathbf{w})$, the acoustic model
- $p(\mathbf{w})$ is the language model
- Both of these are too complex to model directly; much of the research in ASR is about factoring them into manageable chunks

Continuous ASR: The noisy channel model (3)

Summing over all possible state sequences $\mathbf{q}$ for the word string $\mathbf{w}$:

$$\begin{aligned}\mathbf{w}^* &= \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{O}|\mathbf{w})p(\mathbf{w}) \\ &= \operatorname*{argmax}_{\mathbf{w}} \sum_{\mathbf{q}} p(\mathbf{O}|\mathbf{q},\mathbf{w})p(\mathbf{q}|\mathbf{w})p(\mathbf{w})\end{aligned}$$

- $p(\mathbf{O}|\mathbf{q},\mathbf{w})$ is given by the observation (emission) distribution
- $p(\mathbf{q}|\mathbf{w})$ is given by the state transition probabilities
- *Viterbi approximation*: Assume there is a single most probable state sequence $\mathbf{q}^*$ such that all other $\mathbf{q}$ contribute a negligible amount to the sum. Then
  $\mathbf{w}^* = \operatorname*{argmax}_{\mathbf{w}} p(\mathbf{O}|\mathbf{q}^*,\mathbf{w})p(\mathbf{q}^*|\mathbf{w})p(\mathbf{w})$
- So we can maximize jointly over $\mathbf{w}$ and $\mathbf{q}$:
  $\mathbf{w}^*,\mathbf{q}^* = \operatorname*{argmax}_{\mathbf{w},\mathbf{q}} p(\mathbf{O}|\mathbf{q},\mathbf{w})p(\mathbf{q}|\mathbf{w})p(\mathbf{w})$

Summary: Continuous ASR with HMMs under the noisy channel model

- To do continuous speech recognition, we can string together word HMMs according to a *grammar* or *language model*
- For now, assume grammar allows arbitrary word sequences with equal probabilities
- (Will get back to language modeling later)
- We will find the best state sequence using the Viterbi algorithm, and output the corresponding word string

Solving the training problem for HMMs

- Maximum likelihood training: Given an observation sequence (training data), find the HMM parameters $\lambda^* = \{\pi^*, \mathbf{A}^*, \mathbf{B}^*\}$ that maximize the probability of the observations

$$\lambda^* = \underset{\lambda}{\operatorname{argmax}}\, p(\mathbf{O}|\lambda)$$

- If someone *gave us* the state sequence, it would be easy to find the maximum-likelihood estimates for $\mathbf{A}$ and $\mathbf{B}$:

states	1	1	1	1	2	3	3	2	3	2	2	1
observations	H	H	U	H	H	U	U	U	H	H	H	H

$$a_{ij} = \frac{\text{count}(i \rightarrow j)}{\text{count}(\text{transitions from } i)}$$

$$b_i(k) = \frac{\text{count}(v_k \text{ in state } i)}{\text{count}(i)}$$

Solving the training problem for HMMs (2)

- We are *not* given state alignments. Instead we will "generate" them ourselves (as we did with Gaussian component labels in Gaussian mixtures!)
- Baum-Welch Algorithm: EM for HMMs
  - Make initial guess of parameter values, then alternate between:
  - Compute the posterior probabilities of all states at all times, given the observations
  - Compute the maximum-likelihood parameters given the *expected* numbers of occurrences of states and transitions

The Baum-Welch algorithm

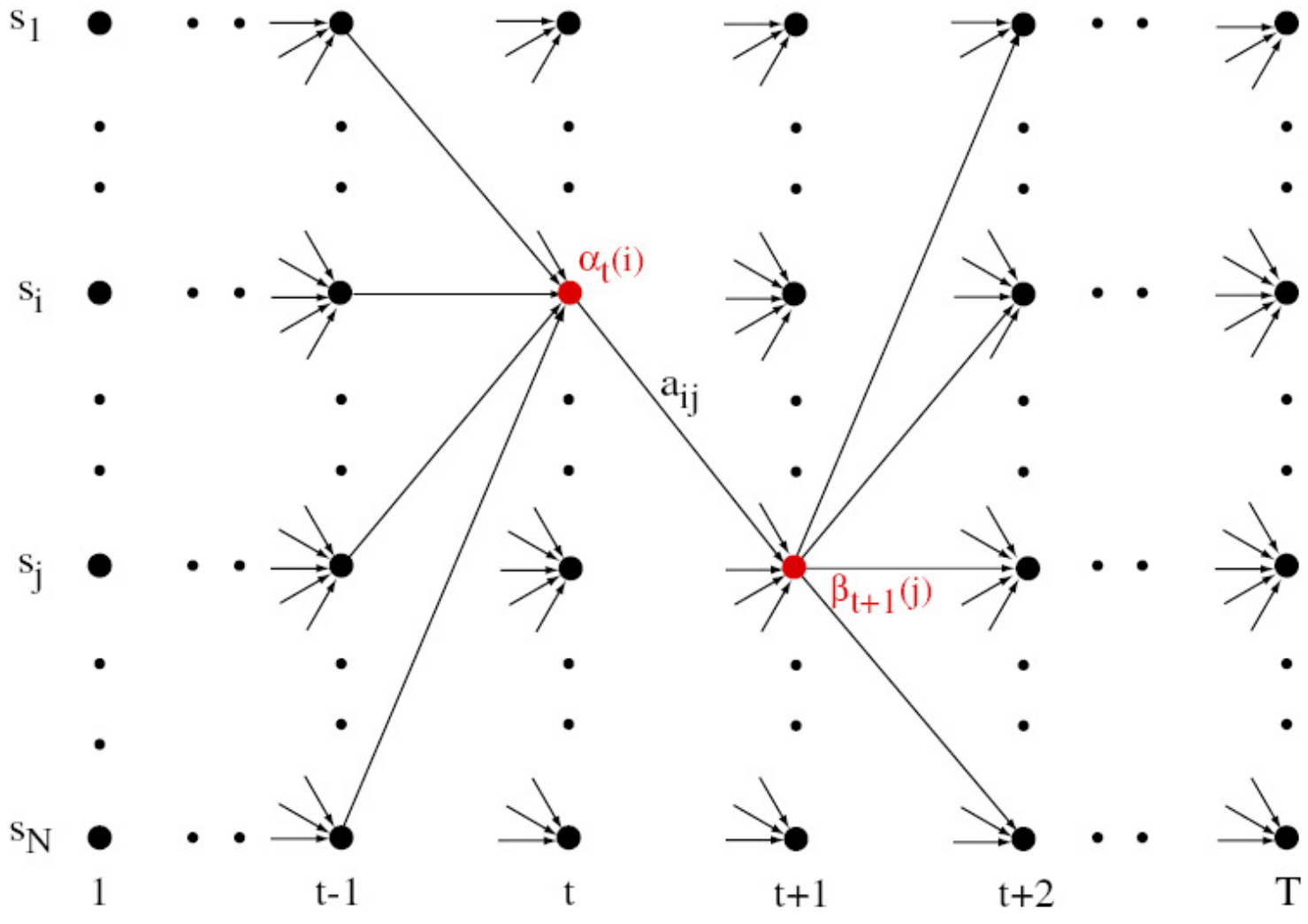
- Define $\xi_t(i,j)$ as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence

$$\begin{aligned}\xi_t(i,j) &= P(q_t = i, q_{t+1} = j|\mathbf{O}, \lambda) \\ &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{p(\mathbf{O}|\lambda)}\end{aligned}$$

- Then note that the state posterior probabilities are

$$\gamma_t(i) = P(q_t = i|\mathbf{O}, \lambda) = \sum_{j=1}^{N} \xi_t(i,j)$$

The Baum-Welch algorithm



The Baum-Welch algorithm

- Now we can compute the *expected counts* we need, e.g.:

$$\sum_{t=1}^{T} \gamma_t(i) \quad = \quad \text{expected count of state } i$$

$$\sum_{t=1}^{T-1} \gamma_t(i) \quad = \quad \text{expected count of transitions from state } i$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) \quad = \quad \text{expected count of transitions from } i \text{ to } j$$

The Baum-Welch re-estimation formulas

$$\hat{a}_{ij} = \frac{\text{expected-count}(i \rightarrow j)}{\text{expected-count}(\text{transitions from } i)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\hat{b}_i(k) = \frac{\text{expected-count}(v_k \text{ in state } i)}{\text{expected-count}(i)} = \frac{\sum_{t=1, o_t=v_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

$$\hat{\pi}_i = \text{expected-count}(q_1 = i) = \gamma_1(i)$$

Properties of Baum-Welch re-estimation

- If λ is the current model, and $\hat{\lambda}$ is the model after one iteration of re-estimation, then it can be shown that $p(\mathbf{O}|\hat{\lambda}) \geq p(\mathbf{O}|\lambda)$
- I.e., the likelihood is increased at each iteration
- Therefore, we can iterate the re-estimation formulas until some convergence threshold is reached (e.g. likelihood increase less than some threshold τ)
- Guaranteed to find a *local* maximum of the likelihood, but not a global one

The Baum-Welch re-estimation formulas

In general, we have *multiple* observation sequences $\mathbf{O}^1, \ldots, \mathbf{O}^L \Rightarrow$

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

$$\hat{b}_i(k) = \frac{\sum_{l=1}^{L} \sum_{t=1, o_t=v_k}^{T} \gamma_t^l(i)}{\sum_{l=1}^{L} \sum_{t=1}^{T} \gamma_t^l(i)}$$

$$\hat{\pi}_i = \frac{1}{L} \sum_{l=1}^{L} \gamma_1^l(i)$$

Parameter estimation with EM

The maximum-likelihood parameter estimation algorithms we've covered for GMMs and HMMs are examples of the more general *expectation-maximization* algorithm:

- Initialize parameters with a guess, then iterate until convergence:
- Given current parameter estimates, compute posteriors of latent variables
- Using posteriors as "counts", find maximum-likelihood estimates of parameters

EM for ASR

In fact, training an ASR system is more than just training a single HMM...

- Typically, we have one HMM per word or phone
- We often don't know the start and end times of each word/phone
- Then we can
  - Estimate the start and end times (do *forced alignment* – more on this later)
  - Consider these to be additional latent variables in the EM algorithm

What can we do now?

- Compare speech sequences with DTW
- Isolated-word recognition with DTW
  - $w^* = \text{argmin}_w \, d(\mathbf{O}, w)$
  - Applicable mainly when we only have one/a few training examples per word
- Isolated-word recognition with HMMs
  - Using forward/backward algorithm:
    $w^* = \text{argmax}_w \, p(\mathbf{O}|\lambda_w)$
  - Useful when we have plenty of training data per word (mainly, small-vocabulary tasks)

What can we do now? (2)

- Continuous-word small-vocabulary recognition with HMMs and "null" grammar or other fixed grammar
  - Using Viterbi algorithm:
    $\mathbf{w}^* = \text{argmax}_{\mathbf{w}} \max_{\mathbf{q}} p(\mathbf{O}, \mathbf{q}|\mathbf{w})p(\mathbf{w})$
    (where now $\mathbf{q}$ are states in the "sentence HMM" of all possible word and state strings)
  - Useful, e.g., for digit strings, command-and-control tasks
- All of the above, with phones instead of words (isolated-phone classification, phonetic recognition with null grammar)
  - For applications like language learning, linguistics research, ...
  - As intermediate task in acoustic modeling research

The need for subword units

Whole-word HMMs have some problems

- Cannot model unseen words, or even words seen too few times in training data
- Number of parameters is proportional to the vocabulary size
- (Note: The more parameters, the more data needed; rule of thumb: $\sim$10 data points per parameter)

$\Rightarrow$ Whole-word models mainly restricted to small-vocabulary tasks

Subword units

- In the same way that sentences can be composed of whole-word HMMs, words can be composed of sub-word HMMs
- Units are then shared among words

Type of units	Approximate # (in English)
words	>100,000
phones	50
diphones	2,000
triphones	10,000
syllables	5,000

- Number of parameters now proportional to the number of sub-word units, not number of words

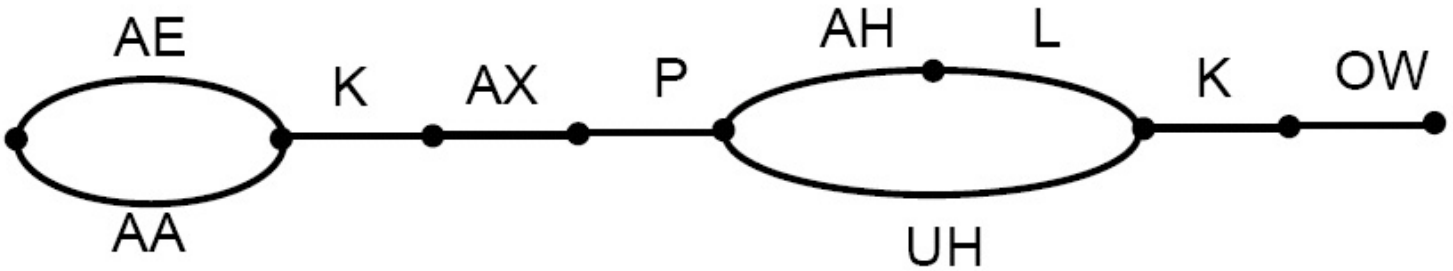
Baseforms

- Each word can be represented as a sequence of phonemes, the word's *baseform*
  `dogs` $\longrightarrow$ d ao g z
- Some words may need more than one baseform
  `the` $\longrightarrow$ dh ah, dh iy
  `either` $\longrightarrow$ iy dh er, ay dh er
- Each word's baseform(s) can be looked up in a dictionary
- All words in training and test data must be in the dictionary, or else we get them wrong
- Typically, baseform dictionaries are written by hand $\longrightarrow$ prone to errors, inconsistencies, disagreements among linguists, ...

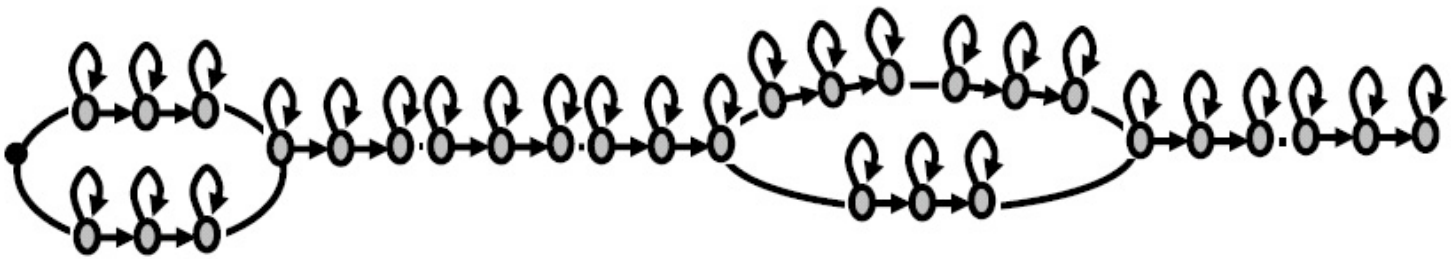
Baseforms

- In general, multiple baseforms can be represented as graph e.g. "acapulco":

  acapulco AE K AX P AH L K OW
  acapulco AA K AX P UH K OW



- Then, to get a word HMM, replace each phone by its HMM:



Reducing parameters by tying

- Multiple states in an HMM, or in different HMMs, can be "tied", i.e. forced to have the same parameters (output distributions or transition probabilities)
- We've seen tying in the context of duration modeling
- Can also tie states to reduce number of parameters $\Rightarrow$ reduce training data requirements

State tying

State tying = constraining states to have identical parameters

- E.g., may tie last states of [ay] and [ey]
- Effective means of reducing number of parameters and training data requirements
- Necessary when number of units is very large
- How to decide which units/states are tied? More on this soon...