

# **CS 224S / LINGUIST 281**

# **Speech Recognition, Synthesis, and**

# **Dialogue**

Dan Jurafsky

---

**Lecture 5: Intro to ASR+HMMs:  
Forward, Viterbi, 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)

# Current error rates

Ballpark numbers; exact numbers depend very much on the specific corpus

Task	Vocabulary	Error Rate%
Digits	11	0.5
WSJ read speech	5K	3
WSJ read speech	20K	3
Broadcast news	64,000+	10
Conversational Telephone	64,000+	20

# HSR versus ASR

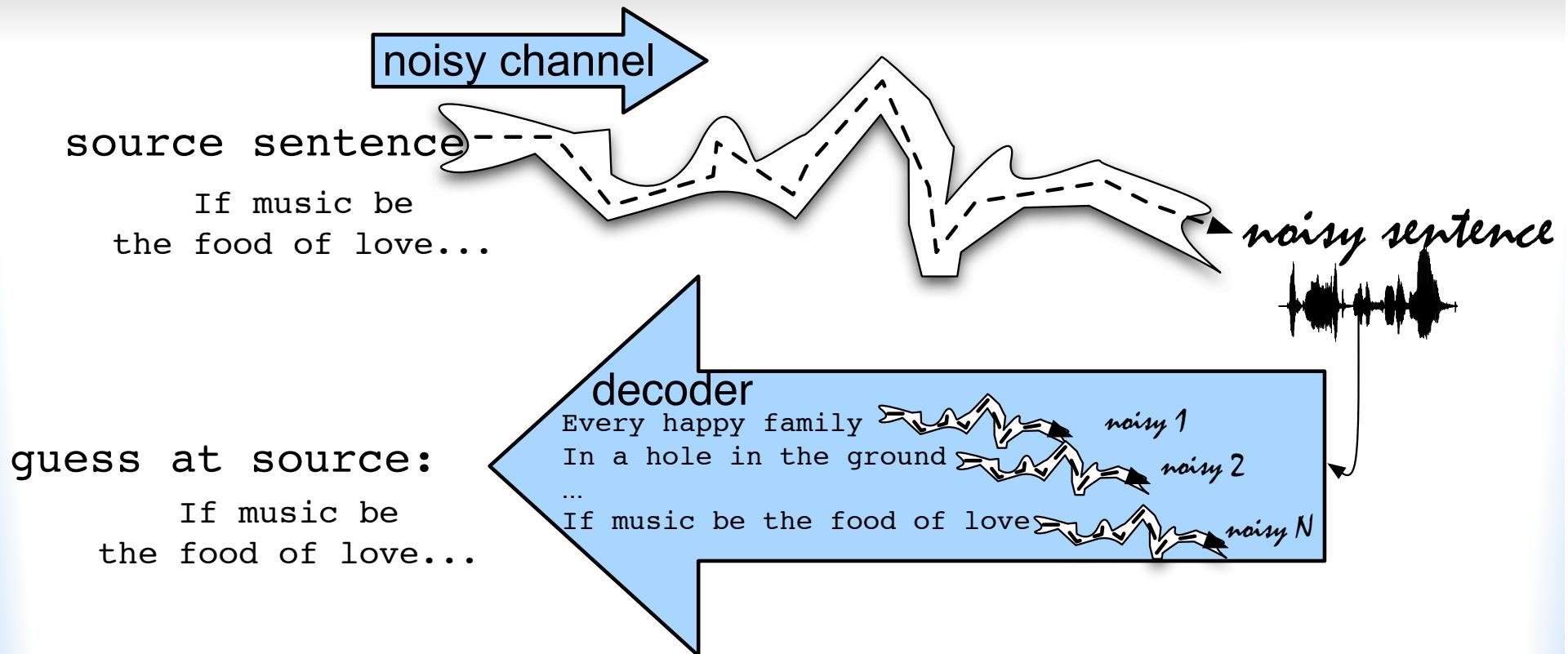
Task	Vocab	ASR	Hum SR
<b>Continuous digits</b>	<b>11</b>	<b>.5</b>	<b>.009</b>
<b>WSJ 1995 clean</b>	<b>5K</b>	<b>3</b>	<b>0.9</b>
<b>WSJ 1995 w/noise</b>	<b>5K</b>	<b>9</b>	<b>1.1</b>
<b>SWBD 2004</b>	<b>65K</b>	<b>20</b>	<b>4</b>

- Conclusions:
  - ◆ Machines about 5 times worse than humans
  - ◆ Gap increases with noisy speech
  - ◆ These numbers are rough, take with grain of salt

# LVCSR Design Intuition

- Build a statistical model of the speech-to-words process
- Collect lots and lots of speech, and transcribe all the words.
- Train the model on the labeled speech
- Paradigm: Supervised Machine Learning + Search

# The Noisy Channel Model



- Search through space of all possible sentences.
- Pick the one that is most probable given the waveform.

# The Noisy Channel Model (II)

- What is the most likely sentence out of all sentences in the language L given some acoustic input O?
- Treat acoustic input O as sequence of individual observations
  - ◆  $O = o_1, o_2, o_3, \dots, o_t$
- Define a sentence as a sequence of words:
  - ◆  $W = w_1, w_2, w_3, \dots, w_n$

# Noisy Channel Model (III)

- Probabilistic implication: Pick the highest prob S:

$$\hat{W} = \arg \max_{W \in L} P(W | O)$$

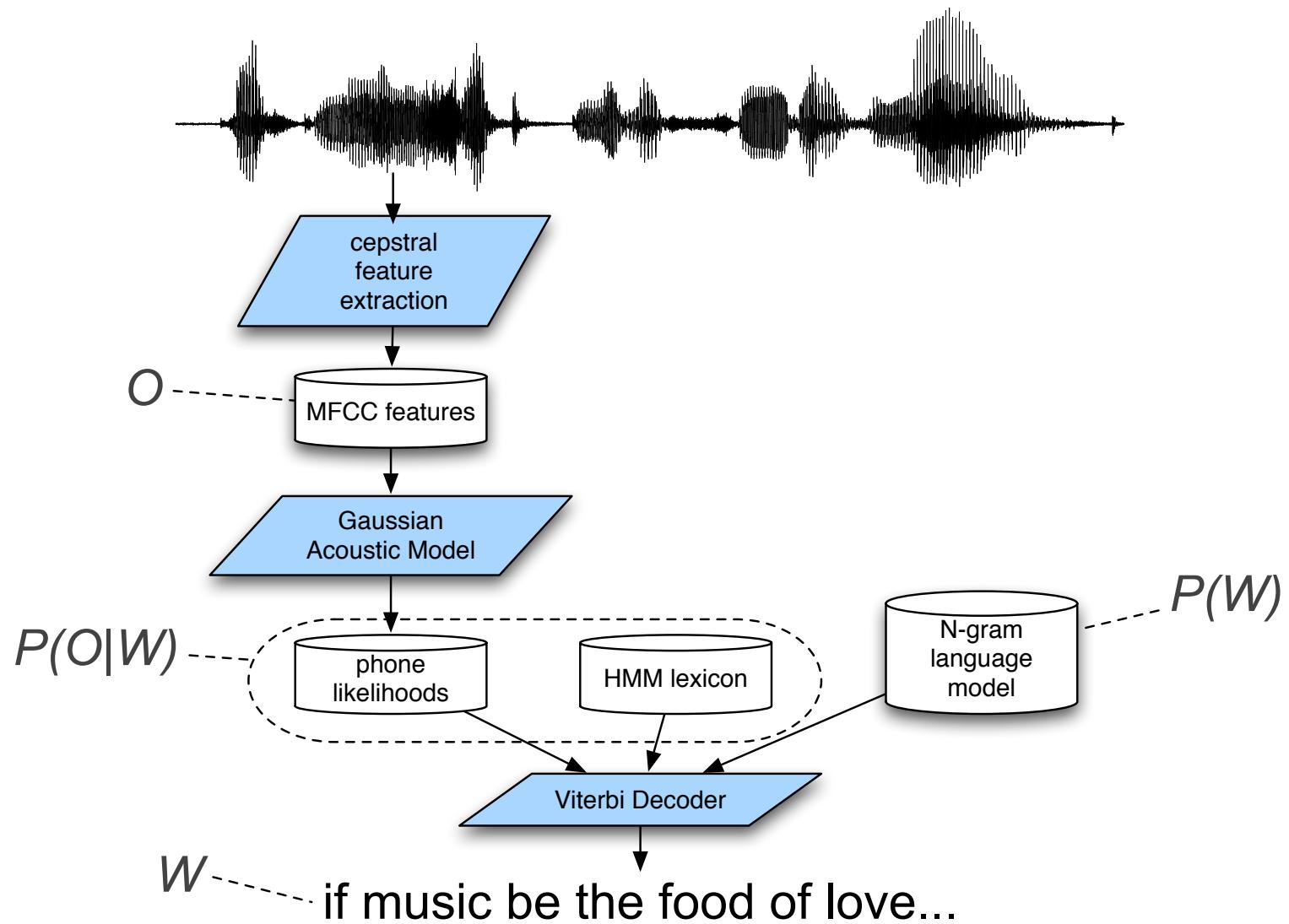
- We can use Bayes rule to rewrite this:

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

- Since denominator is the same for each candidate sentence  $W$ , we can ignore it for the argmax:

$$\hat{W} = \arg \max_{W \in L} P(O|W)P(W)$$

# Speech Recognition Architecture



# Noisy channel model

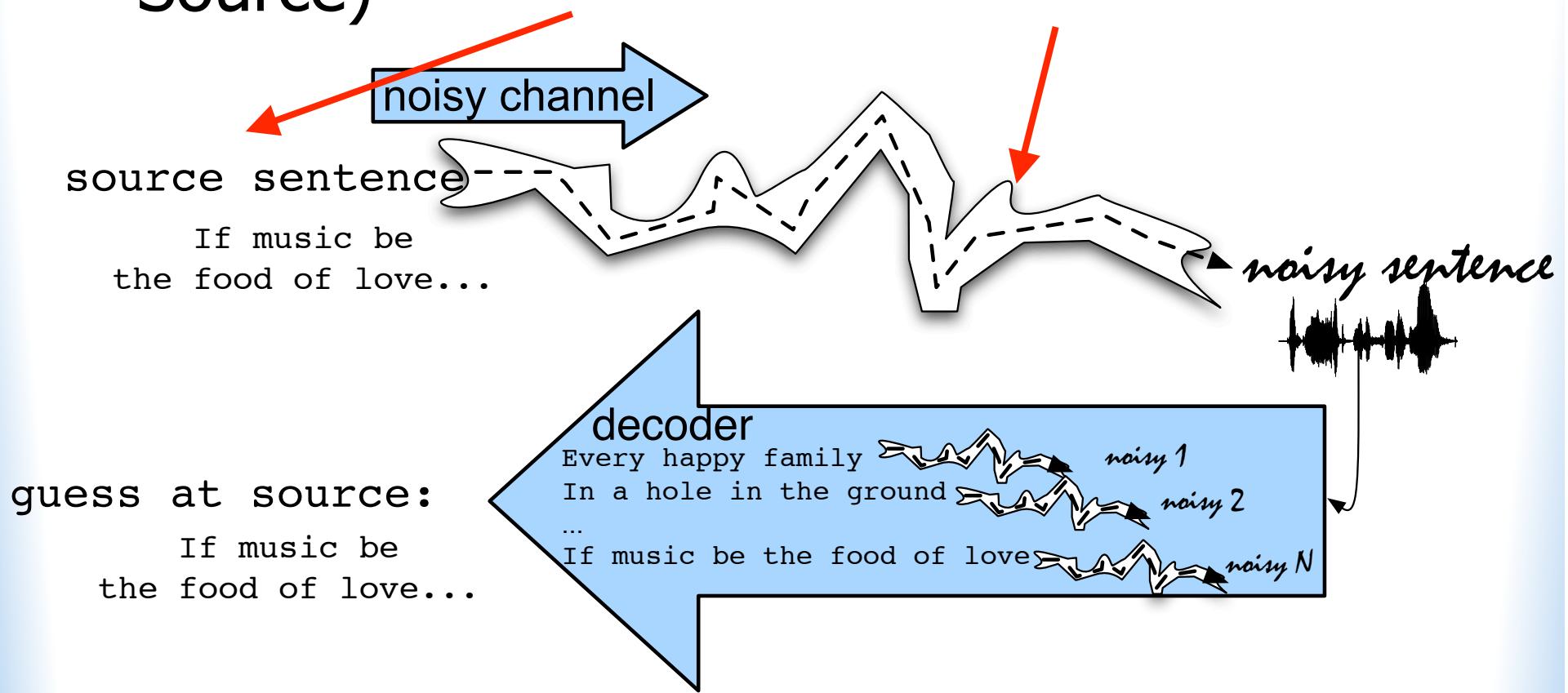
$$\hat{W} = \operatorname{argmax}_{W \in L} P(O | W)P(W)$$

likelihood      prior

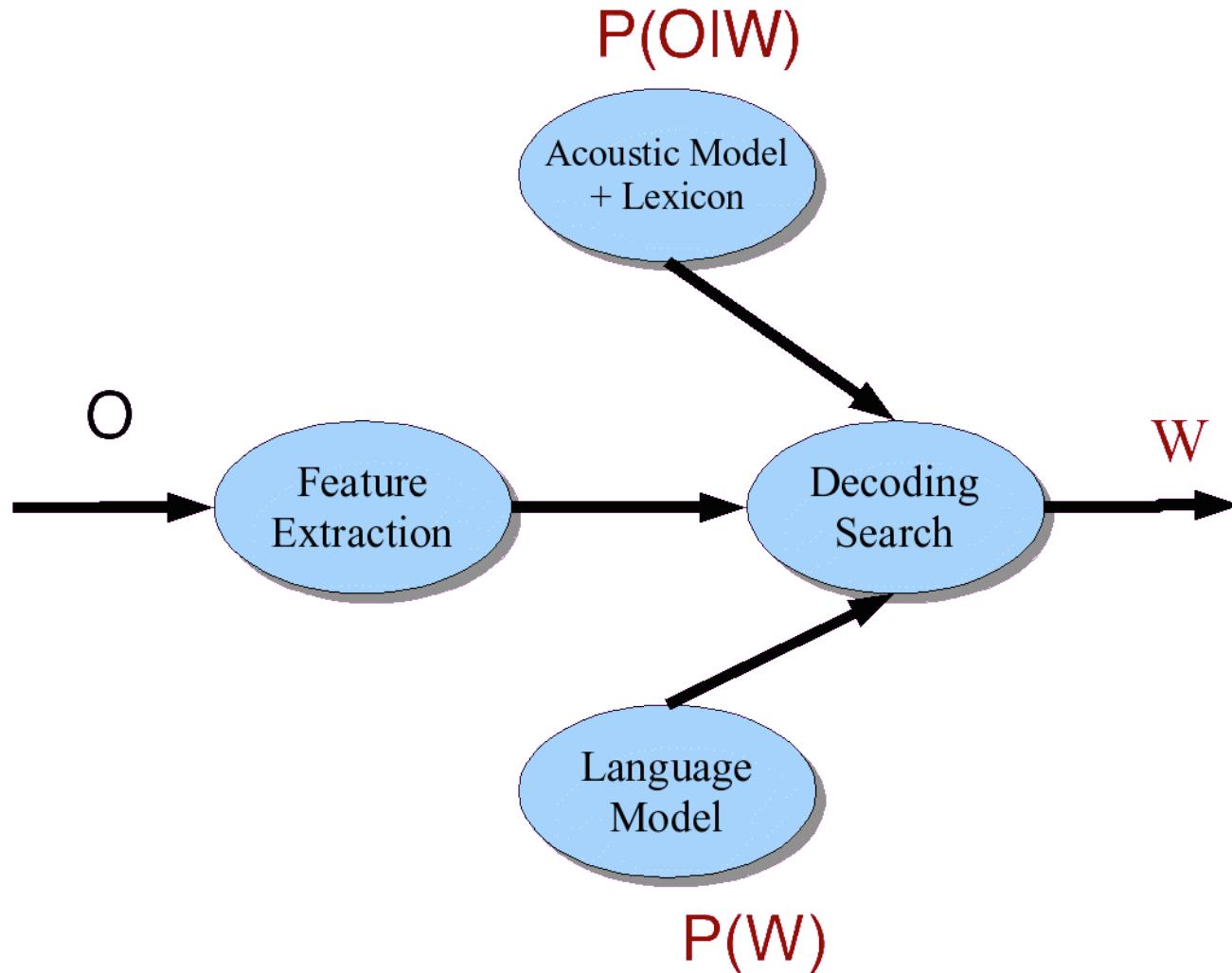


# The noisy channel model

- Ignoring the denominator leaves us with two factors:  $P(\text{Source})$  and  $P(\text{Signal} | \text{Source})$



# Speech Architecture meets Noisy Channel



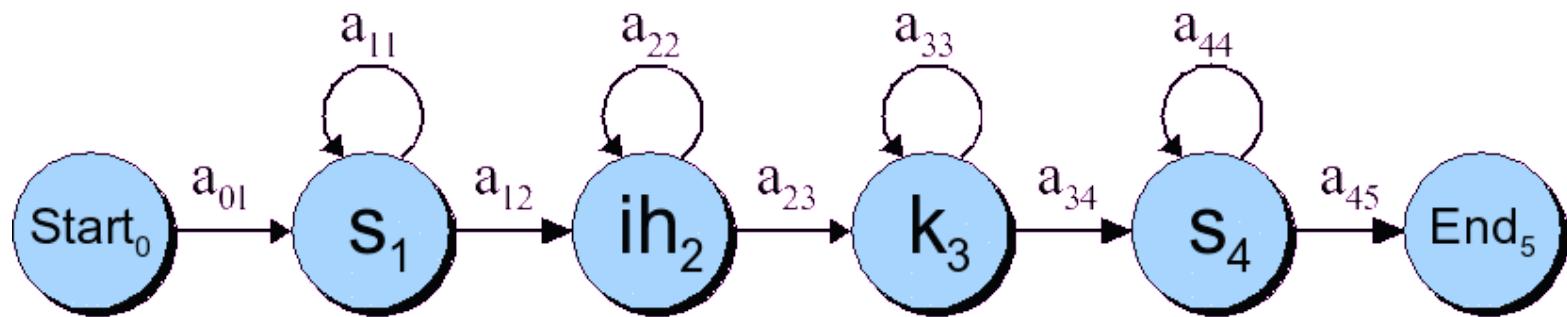
# **Architecture: Five easy pieces (only 2 for today)**

- Feature extraction
- Acoustic Modeling
- HMMs, Lexicons, and Pronunciation
- Decoding
- Language Modeling

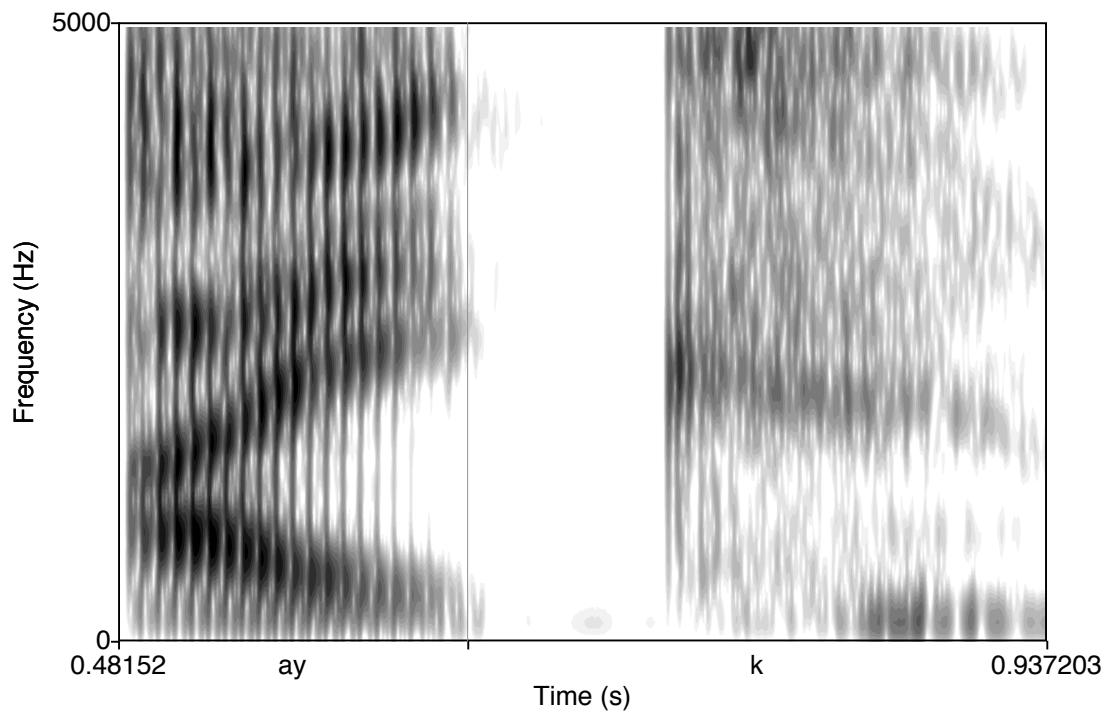
# Lexicon

- A list of words
- Each one with a pronunciation in terms of phones
- We get these from on-line pronunciation dictionary
- CMU dictionary: 127K words
  - ◆ <http://www.speech.cs.cmu.edu/cgi-bin/cmudict>
- We'll represent the lexicon as an HMM

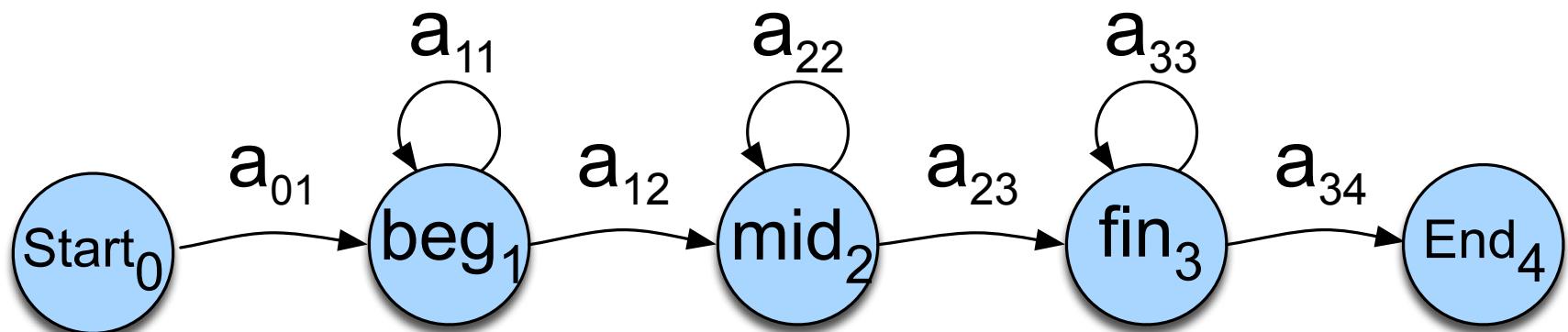
# HMMs for speech



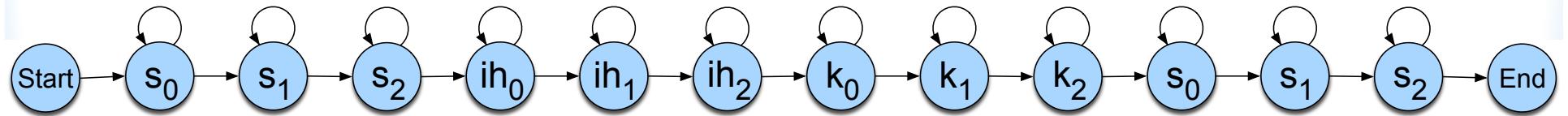
# Phones are not homogeneous!



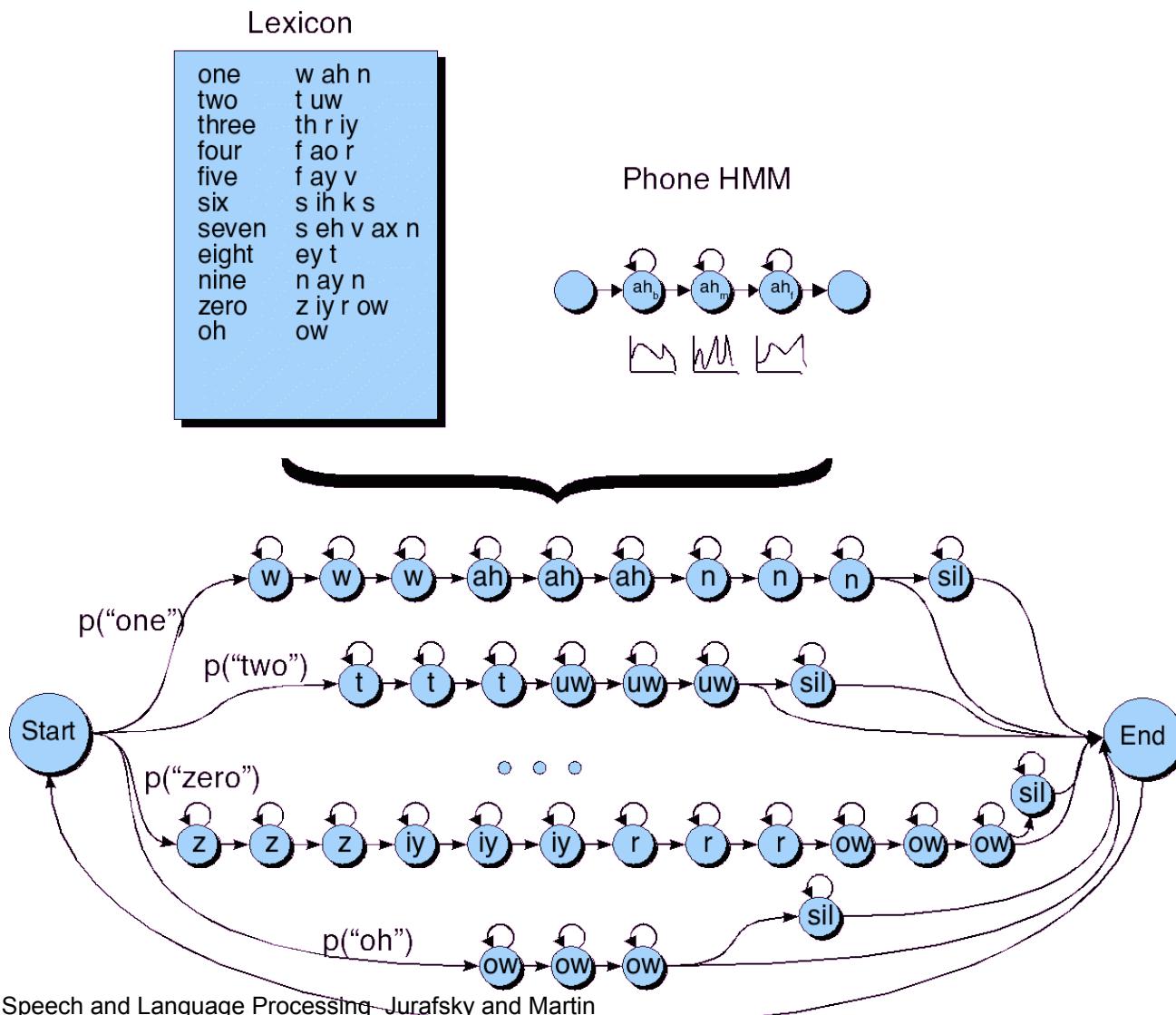
# Each phone has 3 subphones



# Resulting HMM word model for “six”



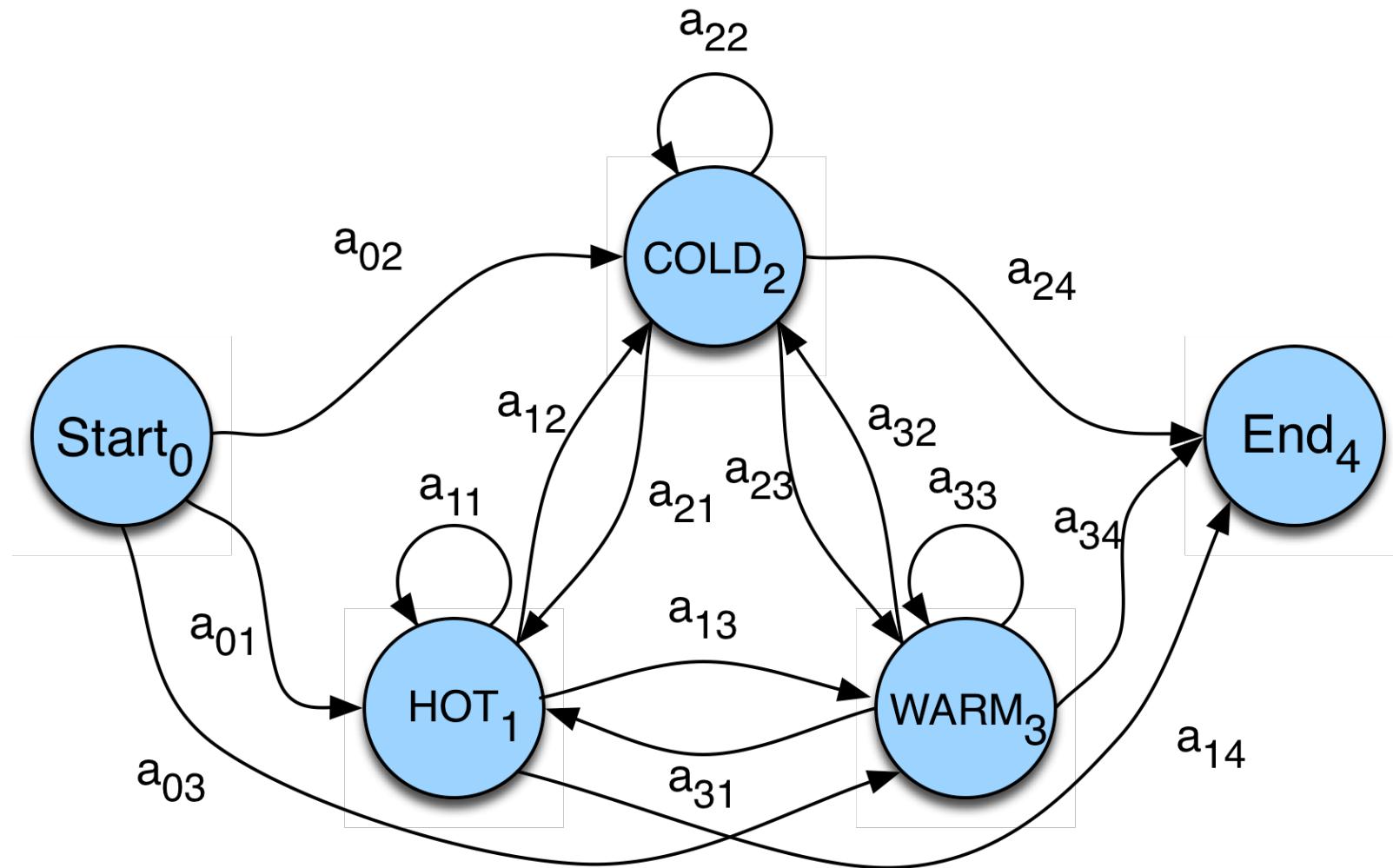
# HMM for the digit recognition task



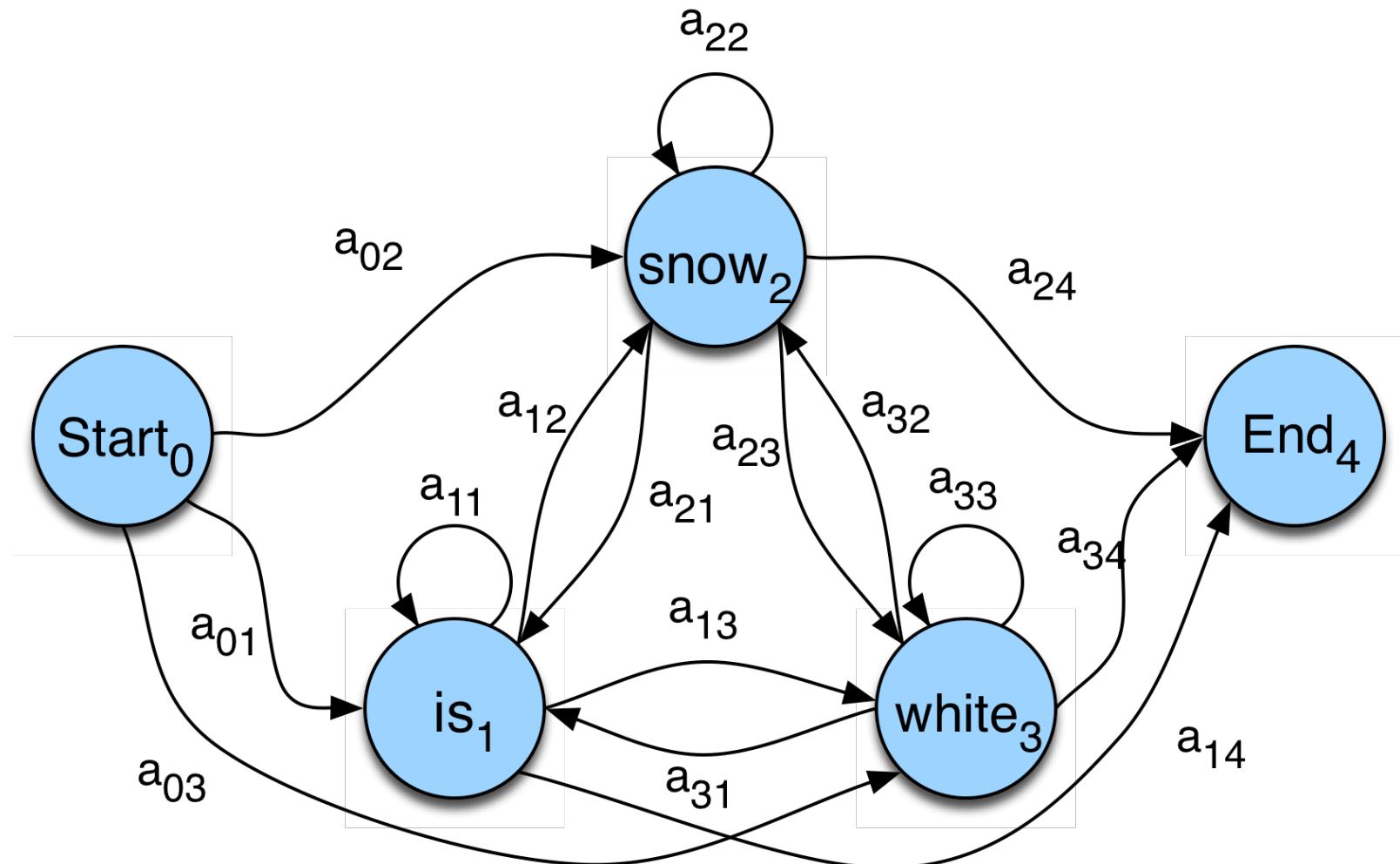
# More formally: Toward HMMs

- A weighted finite-state automaton
  - ◆ An FSA with probabilities on the arcs
  - ◆ The sum of the probabilities leaving any arc must sum to one
- A Markov chain (or observable Markov Model)
  - ◆ a special case of a WFST in which the input sequence uniquely determines which states the automaton will go through
- Markov chains can't represent inherently ambiguous problems
  - ◆ Useful for assigning probabilities to unambiguous sequences

# Markov chain for weather



# Markov chain for words



# Markov chain = “First-order observable Markov Model”

- a set of states
  - ◆  $Q = q_1, q_2 \dots q_N$ ; the state at time  $t$  is  $q_t$
- Transition probabilities:
  - ◆ a set of probabilities  $A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$ .
  - ◆ Each  $a_{ij}$  represents the probability of transitioning from state  $i$  to state  $j$
  - ◆ The set of these is the transition probability matrix  $A$

$$a_{ij} = P(q_t = j \mid q_{t-1} = i) \quad 1 \leq i, j \leq N$$

$$\sum_{j=1}^N a_{ij} = 1; \quad 1 \leq i \leq N$$

- Distinguished start and end states

# **Markov chain = “First-order observable Markov Model”**

- Current state only depends on previous state

Markov Assumption :  $P(q_i | q_1..q_{i-1}) = P(q_i | q_{i-1})$

# Another representation for start state

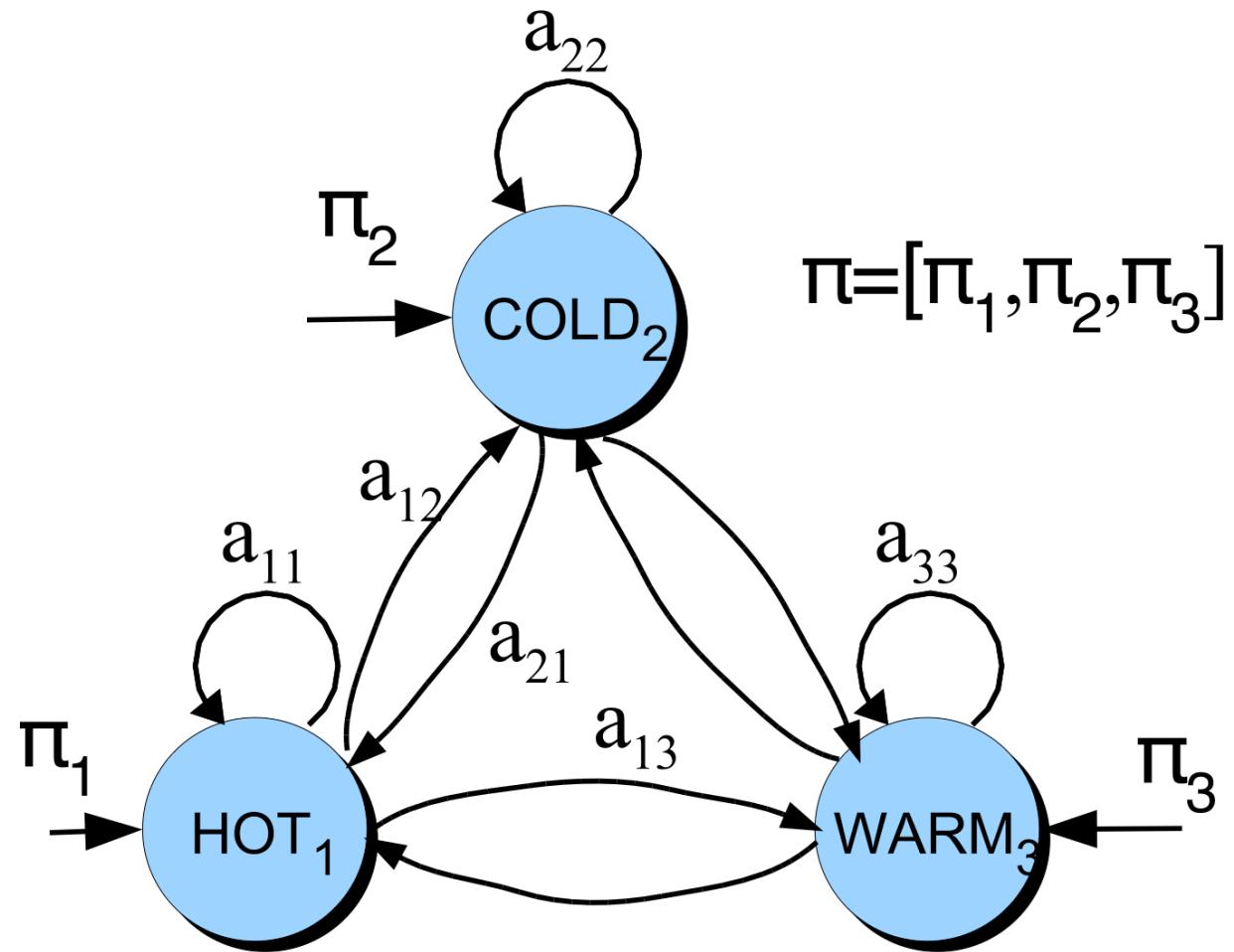
- Instead of start state
- Special initial probability vector  $\pi$ 
  - ◆ An initial distribution over probability of start states

$$\pi_i = P(q_1 = i) \quad 1 \leq i \leq N$$

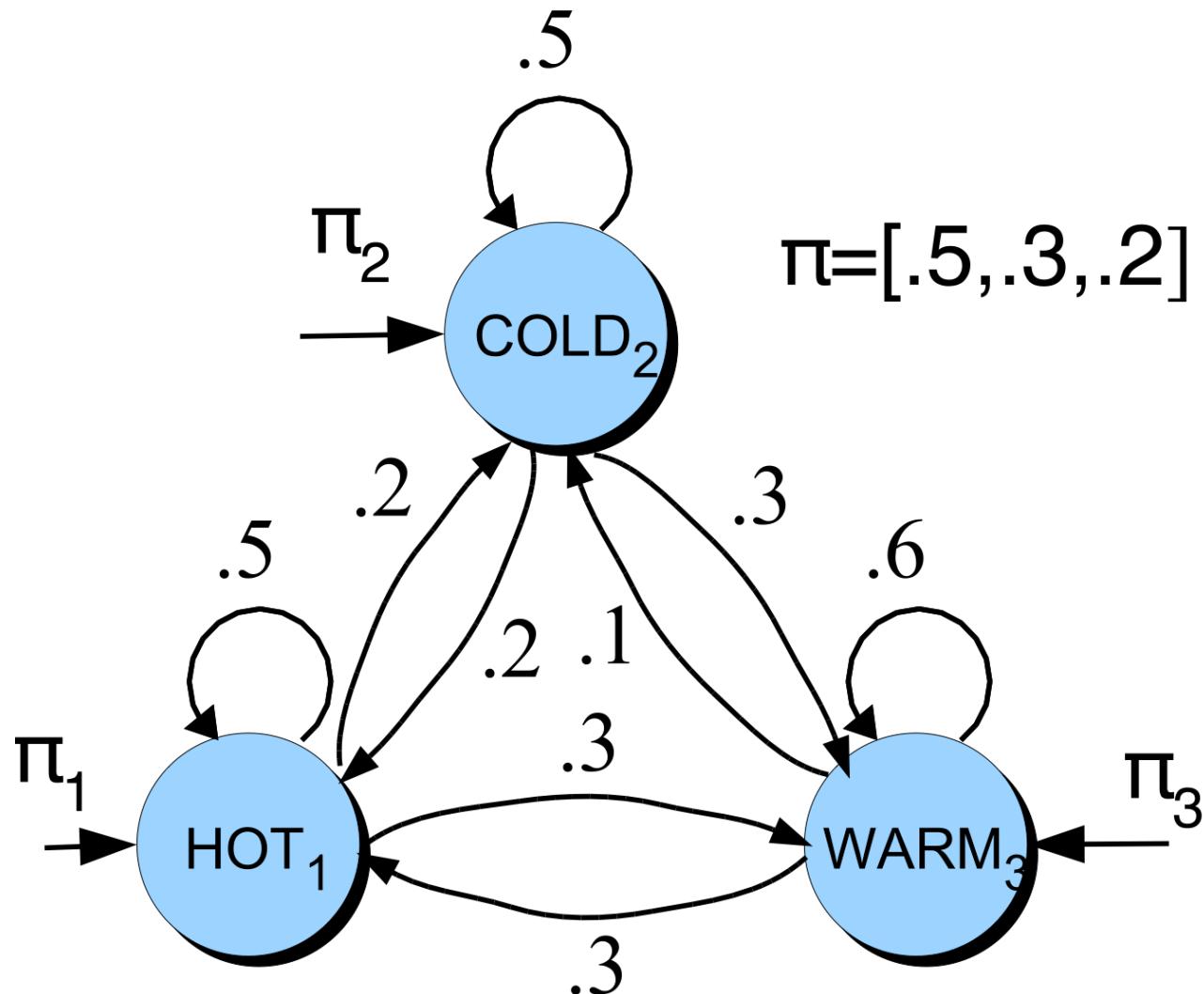
- Constraints:

$$\sum_{j=1}^N \pi_j = 1$$

# The weather figure using pi



# The weather figure: specific example



# Markov chain for weather

- What is the probability of 4 consecutive warm days?
- Sequence is warm-warm-warm-warm
- I.e., state sequence is 3-3-3-3
- $P(3,3,3,3) =$ 
  - ◆  $\pi_3 a_{33} a_{33} a_{33} a_{33} = 0.2 \times (0.6)^3 = 0.0432$

# How about?

- Hot hot hot hot
- Cold hot cold hot
- What does the difference in these probabilities tell you about the real world weather info encoded in the figure?

# HMM for Ice Cream

- You are a climatologist in the year 2799
- Studying global warming
- You can't find any records of the weather in Baltimore, MD for summer of 2008
- But you find Jason Eisner's diary
- Which lists how many ice-creams Jason ate every date that summer
- Our job: figure out how hot it was

# Hidden Markov Model

- For Markov chains, the output symbols are the same as the states.
  - ◆ See **hot** weather: we're in state **hot**
- But in named-entity or part-of-speech tagging (and speech recognition and other things)
  - ◆ The output symbols are **words**
  - ◆ But the hidden states are something else
    - **Part-of-speech tags**
    - **Named entity tags**
- So we need an extension!
- A **Hidden Markov Model** is an extension of a Markov chain in which the input symbols are not the same as the states.
- This means **we don't know which state we are in.**

# Hidden Markov Models

$Q = q_1 q_2 \dots q_N$	a set of $N$ <b>states</b>
$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$	a <b>transition probability matrix</b> $A$ , each $a_{ij}$ representing the probability of moving from state $i$ to state $j$ , s.t. $\sum_{j=1}^n a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of $T$ <b>observations</b> , each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of <b>observation likelihoods</b> , also called <b>emission probabilities</b> , each expressing the probability of an observation $o_t$ being generated from a state $i$
$q_0, q_F$	a special <b>start state</b> and <b>end (final) state</b> that are not associated with observations, together with transition probabilities $a_{01} a_{02} \dots a_{0n}$ out of the start state and $a_{1F} a_{2F} \dots a_{nF}$ into the end state

# Assumptions

- **Markov assumption:**

$$P(q_i \mid q_1..q_{i-1}) = P(q_i \mid q_{i-1})$$

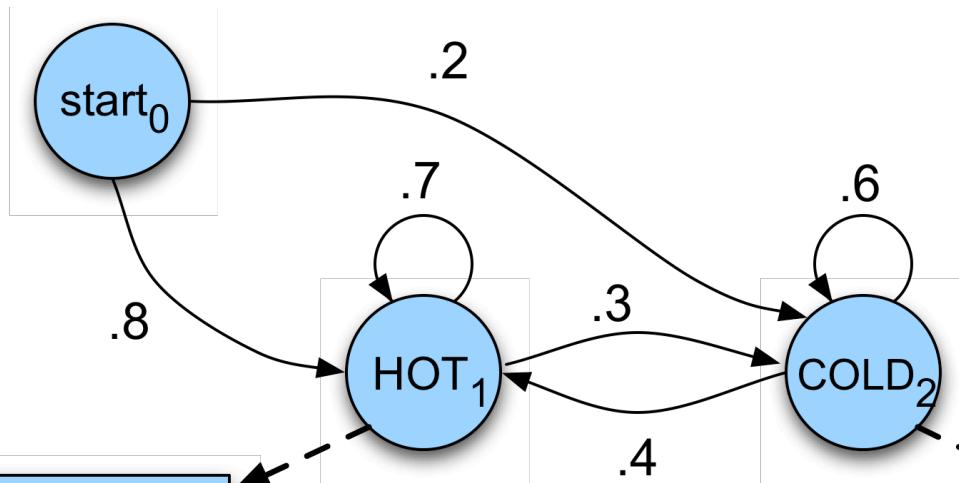
- **Output-independence assumption**

$$P(o_t \mid O_1^{t-1}, q_1^t) = P(o_t \mid q_t)$$

# Eisner task

- Given
  - ◆ Ice Cream Observation Sequence:  
1,2,3,2,2,2,3...
- Produce:
  - ◆ Weather Sequence: H,C,H,H,H,C...

# HMM for ice cream



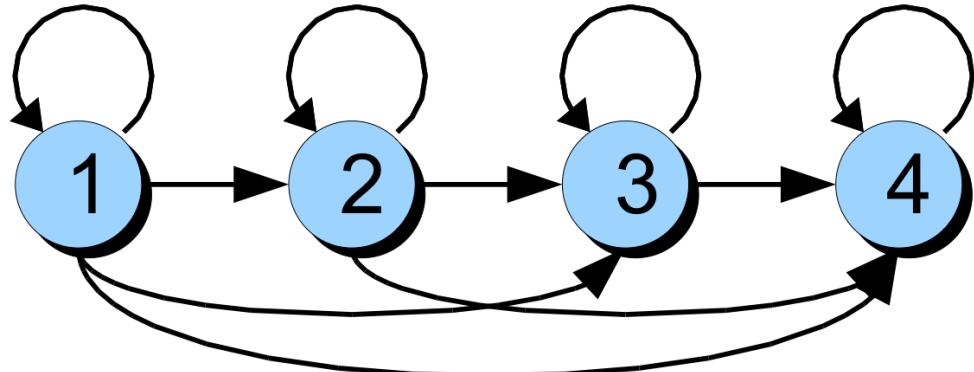
**B<sub>1</sub>**

$$\begin{bmatrix} P(1 \mid \text{HOT}) \\ P(2 \mid \text{HOT}) \\ P(3 \mid \text{HOT}) \end{bmatrix} = \begin{bmatrix} .2 \\ .4 \\ .4 \end{bmatrix}$$

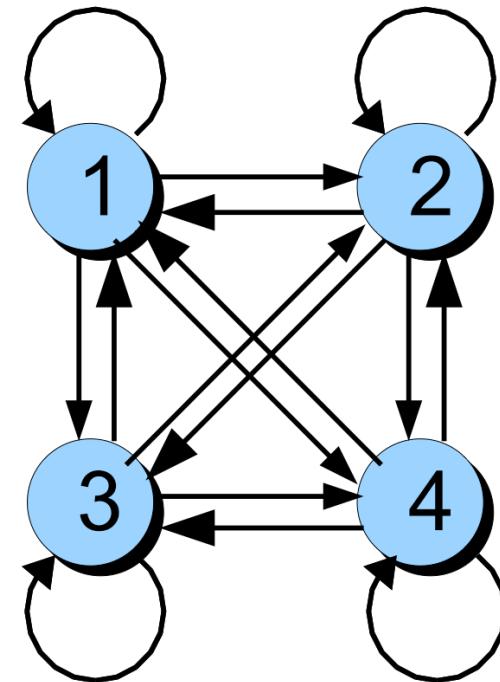
**B<sub>2</sub>**

$$\begin{bmatrix} P(1 \mid \text{COLD}) \\ P(2 \mid \text{COLD}) \\ P(3 \mid \text{COLD}) \end{bmatrix} = \begin{bmatrix} .5 \\ .4 \\ .1 \end{bmatrix}$$

# Different types of HMM structure



Bakis = left-to-right



Ergodic =  
fully-connected

# The Three Basic Problems for HMMs

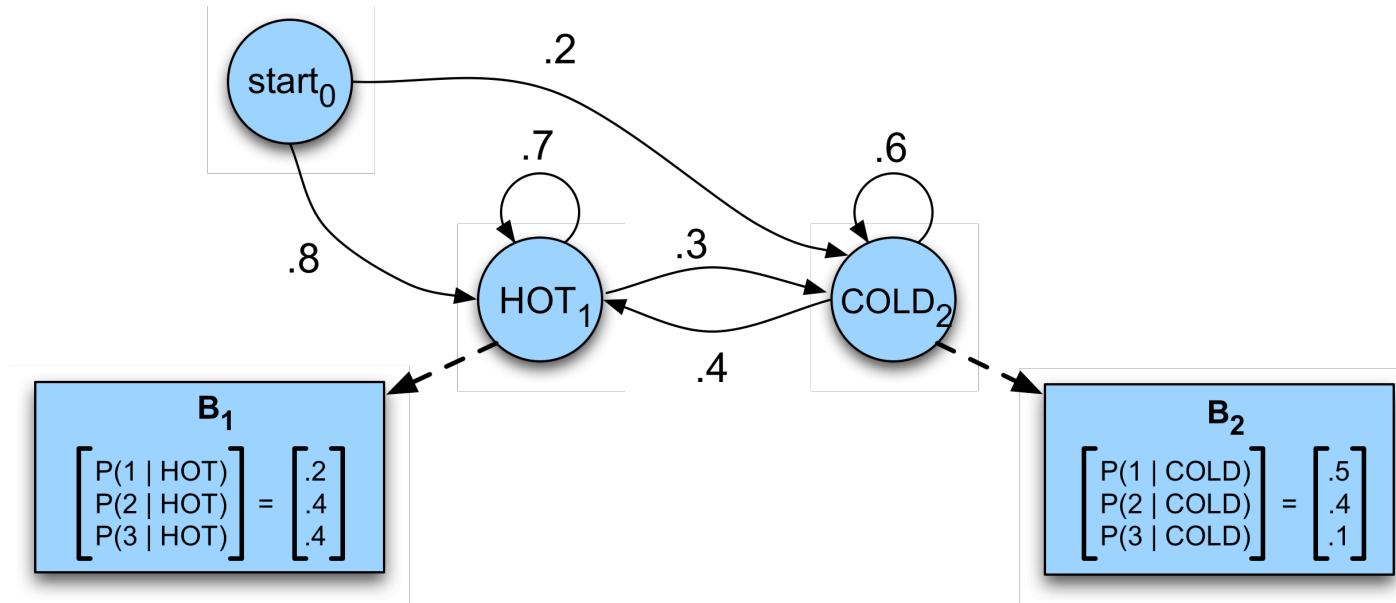
Jack Ferguson at IDA in the 1960s

- Problem 1 (**Evaluation**): Given the observation sequence  $O=(o_1 o_2 \dots o_T)$ , and an HMM model  $\Phi = (A, B)$ , how do we efficiently compute  $P(O | \Phi)$ , the probability of the observation sequence, given the model
- Problem 2 (**Decoding**): Given the observation sequence  $O=(o_1 o_2 \dots o_T)$ , and an HMM model  $\Phi = (A, B)$ , how do we choose a corresponding state sequence  $Q=(q_1 q_2 \dots q_T)$  that is optimal in some sense (i.e., best explains the observations)
- Problem 3 (**Learning**): How do we adjust the model parameters  $\Phi = (A, B)$  to maximize  $P(O | \Phi)$ ?

# Problem 1: computing the observation likelihood

**Computing Likelihood:** Given an HMM  $\lambda = (A, B)$  and an observation sequence  $O$ , determine the likelihood  $P(O|\lambda)$ .

- Given the following HMM:



- How likely is the sequence 3 1 3?

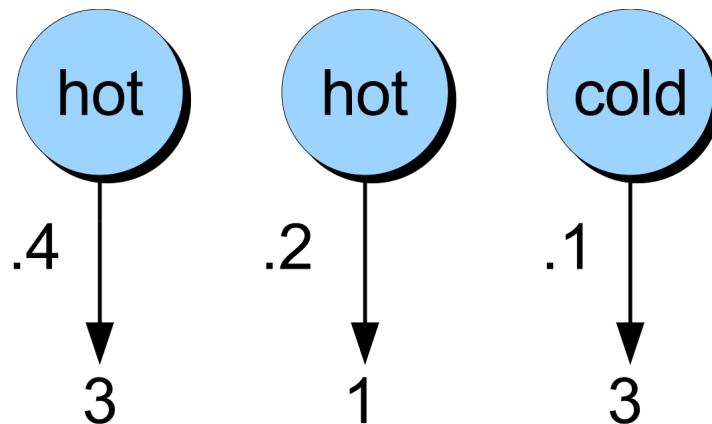
# How to compute likelihood

- For a Markov chain, we just follow the states 3 1 3 and multiply the probabilities
- But for an HMM, we don't know what the states are!
- So let's start with a simpler situation.
- Computing the observation likelihood for a **given** hidden state sequence
  - ◆ Suppose we knew the weather and wanted to predict how much ice cream Jason would eat.
  - ◆ I.e.  $P(3 \ 1 \ 3 | H \ H \ C)$

# Computing likelihood of 3 1 3 given hidden state sequence

$$P(O|Q) = \prod_{i=1}^T P(o_i|q_i)$$

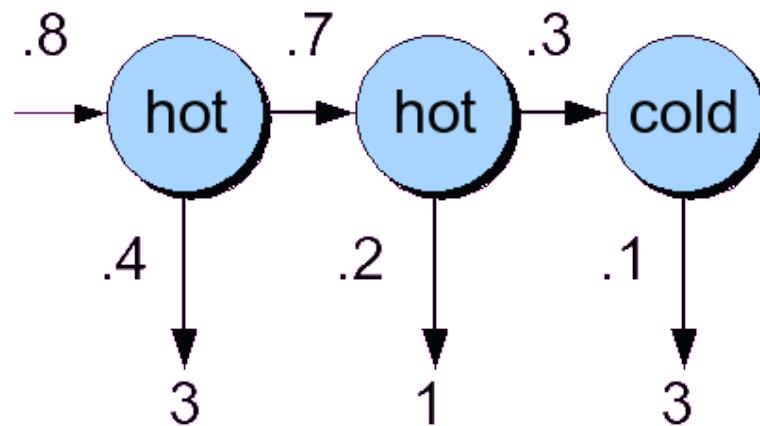
$$P(3\ 1\ 3|\text{hot hot cold}) = P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold})$$



# Computing joint probability of observation and state sequence

$$P(O, Q) = P(O|Q) \times P(Q) = \prod_{i=1}^n P(o_i|q_i) \times \prod_{i=1}^n P(q_i|q_{i-1})$$

$$\begin{aligned} P(3\ 1\ 3, \text{hot hot cold}) &= P(\text{hot}|\text{start}) \times P(\text{hot}|\text{hot}) \times P(\text{cold}|\text{hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned}$$



# Computing total likelihood of 3 1 3

- We would need to sum over
  - ◆ Hot hot cold
  - ◆ Hot hot hot
  - ◆ Hot cold hot
  - ◆ ....
- How many possible hidden state sequences are there for this sequence?

$$P(O) = \sum_Q P(O, Q) = \sum_Q P(O|Q)P(Q)$$

$$P(3\ 1\ 3) = P(3\ 1\ 3, \text{cold cold cold}) + P(3\ 1\ 3, \text{cold cold hot}) + P(3\ 1\ 3, \text{hot hot cold}) + \dots$$

- How about in general for an HMM with N hidden states and a sequence of T observations?
  - ◆  $N^T$
- So we can't just do separate computation for each hidden state sequence.

# Instead: the Forward algorithm

- A kind of **dynamic programming** algorithm
  - ◆ Just like Minimum Edit Distance
  - ◆ Uses a table to store intermediate values
- Idea:
  - ◆ Compute the likelihood of the observation sequence
  - ◆ By summing over all possible hidden state sequences
  - ◆ But doing this efficiently
    - By folding all the sequences into a single **trellis**

# The forward algorithm

- The goal of the forward algorithm is to compute

$$P(o_1, o_2, \dots, o_T, q_T = q_F \mid \lambda)$$

- We'll do this by recursion

# The forward algorithm

- Each cell of the forward algorithm trellis  $\alpha_t(j)$ 
  - ◆ Represents the probability of being in state  $j$
  - ◆ After seeing the first  $t$  observations
  - ◆ Given the automaton
- Each cell thus expresses the following probability

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$$

# The Forward Recursion

1. Initialization:

$$\alpha_1(j) = a_{0j} b_j(o_1) \quad 1 \leq j \leq N$$

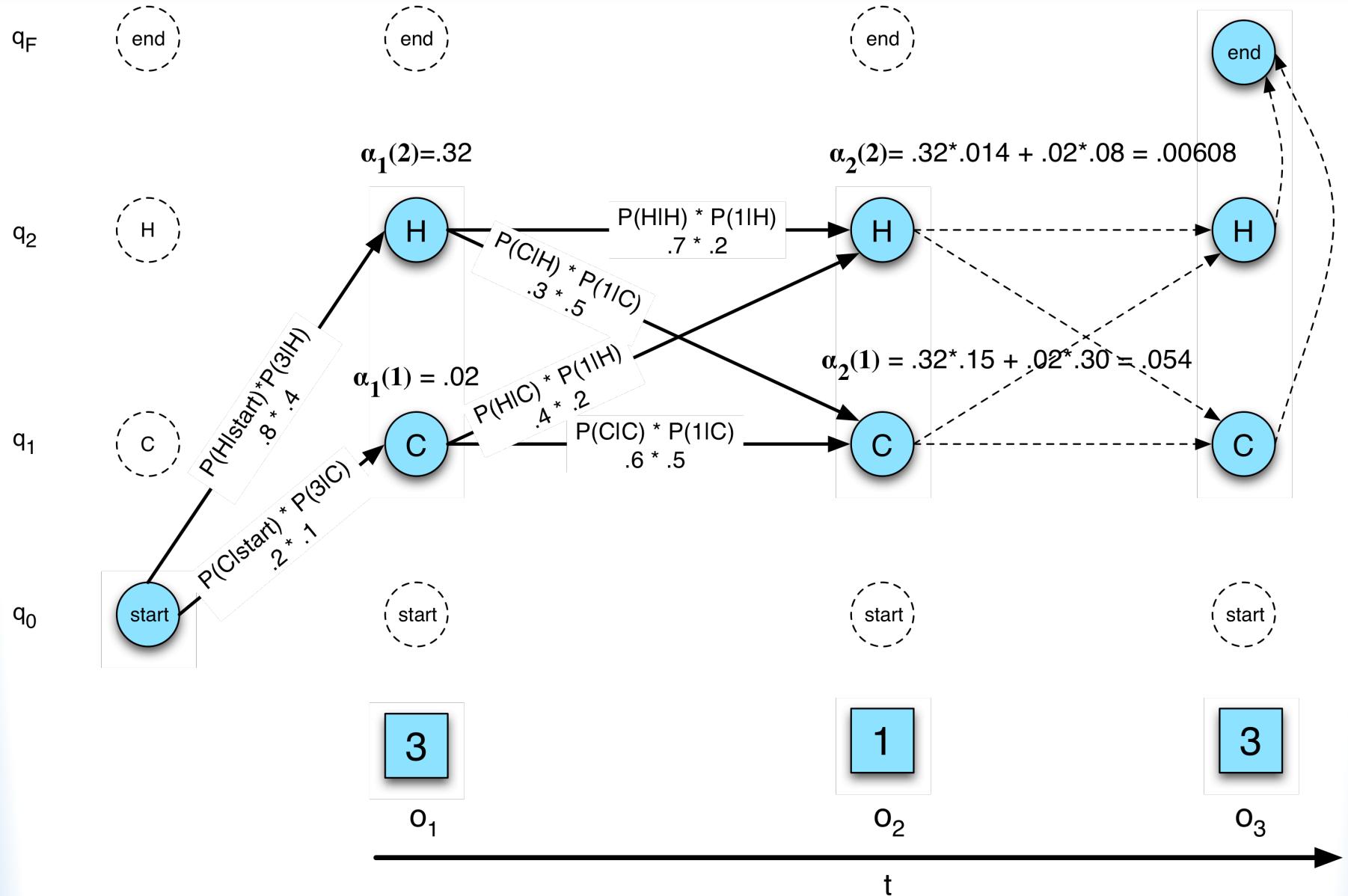
2. Recursion (since states 0 and F are non-emitting):

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

3. Termination:

$$P(O|\lambda) = \alpha_T(q_F) = \sum_{i=1}^N \alpha_T(i) a_{iF}$$

# The Forward Trellis



# We update each cell

$\alpha_{t-1}(i)$

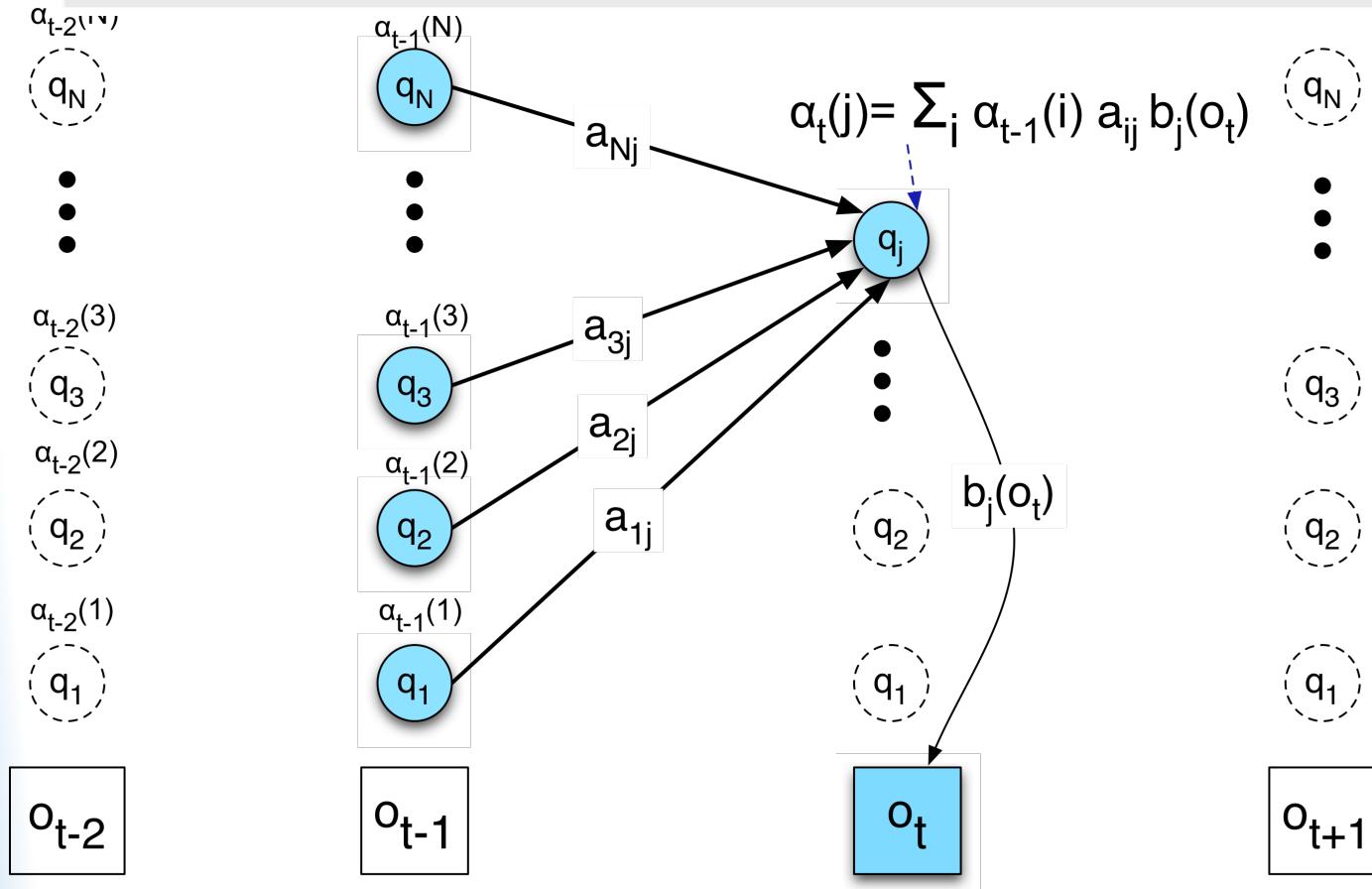
the **previous forward path probability** from the previous time step

$a_{ij}$

the **transition probability** from previous state  $q_i$  to current state  $q_j$

$b_j(o_t)$

the **state observation likelihood** of the observation symbol  $o_t$  given the current state  $j$



# The Forward Algorithm

```
function FORWARD(observations of len  $T$ , state-graph of len  $N$ ) returns forward-prob
    create a probability matrix forward[ $N+2, T$ ]
    for each state  $s$  from 1 to  $N$  do ; initialization step
         $\text{forward}[s, 1] \leftarrow a_{0,s} * b_s(o_1)$ 
    for each time step  $t$  from 2 to  $T$  do ; recursion step
        for each state  $s$  from 1 to  $N$  do
             $\text{forward}[s, t] \leftarrow \sum_{s'=1}^N \text{forward}[s', t-1] * a_{s', s} * b_s(o_t)$ 
     $\text{forward}[q_F, T] \leftarrow \sum_{s=1}^N \text{forward}[s, T] * a_{s, q_F}$  ; termination step
    return forward[ $q_F, T$ ]
```

# Decoding

- Given an observation sequence
  - ◆ 3 1 3
- And an HMM
- The task of the **decoder**
  - ◆ To find the best **hidden** state sequence
- Given the observation sequence  $O = (o_1 o_2 \dots o_T)$ , and an HMM model  $\Phi = (A, B)$ , **how do we choose a corresponding state sequence  $Q = (q_1 q_2 \dots q_T)$**  that is optimal in some sense (i.e., best explains the observations)

# Decoding

- One possibility:
  - ◆ For each hidden state sequence Q
    - HHH, HHC, HCH,
  - ◆ Compute  $P(O|Q)$
  - ◆ Pick the highest one
- Why not?
  - ◆  $N^T$
- Instead:
  - ◆ The Viterbi algorithm
  - ◆ Is again a **dynamic programming** algorithm
  - ◆ Uses a similar trellis to the Forward algorithm

# Viterbi intuition

- We want to compute the joint probability of the observation sequence together with the best state sequence

$$v_t(j) = \max_{q_0, q_1, \dots, q_{t-1}} P(q_0, q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda)$$

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

# Viterbi Recursion

## 1. Initialization:

$$\begin{aligned} v_1(j) &= a_{0j} b_j(o_1) \quad 1 \leq j \leq N \\ bt_1(j) &= 0 \end{aligned}$$

## 2. Recursion (recall that states 0 and $q_F$ are non-emitting):

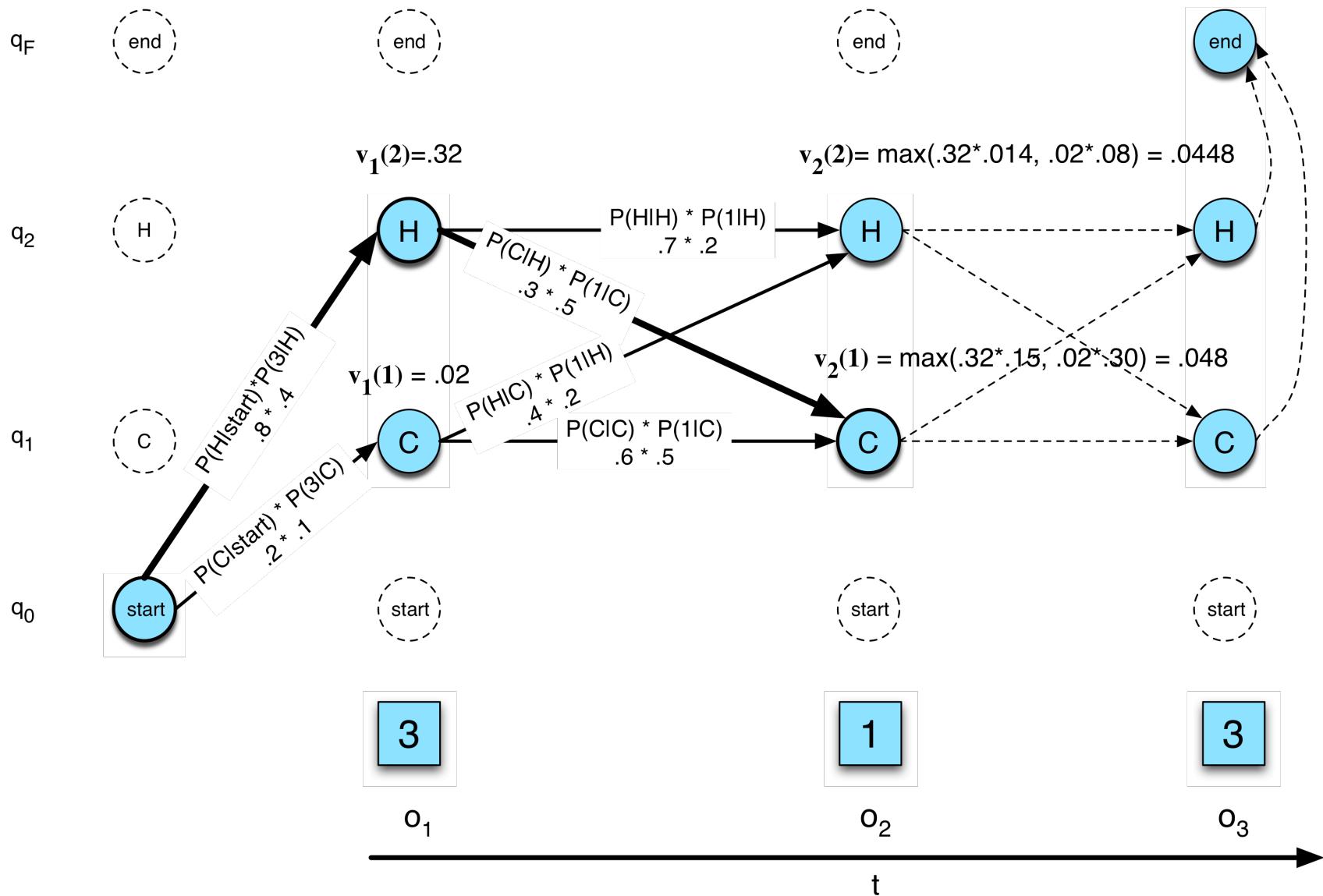
$$\begin{aligned} v_t(j) &= \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \\ bt_t(j) &= \operatorname{argmax}_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \end{aligned}$$

## 3. Termination:

$$\text{The best score: } P* = v_t(q_F) = \max_{i=1}^N v_T(i) * a_{i,F}$$

$$\text{The start of backtrace: } q_T* = bt_T(q_F) = \operatorname{argmax}_{i=1}^N v_T(i) * a_{i,F}$$

# The Viterbi trellis



# Viterbi intuition

- Process observation sequence left to right
- Filling out the trellis
- Each cell:

$$v_t(j) = \max_{q_0, q_1, \dots, q_{t-1}} P(q_0, q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda)$$

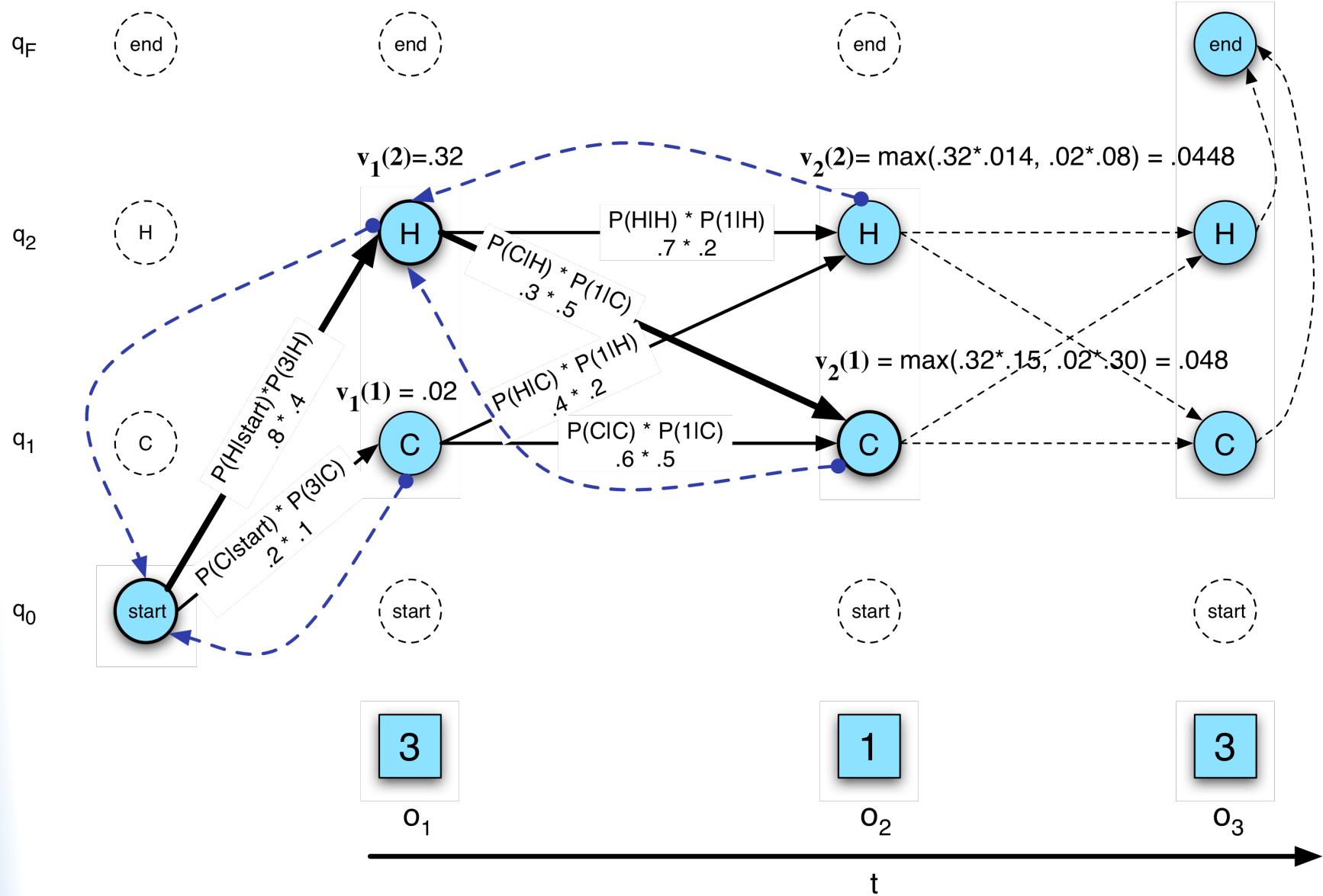
$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

$v_{t-1}(i)$	the <b>previous Viterbi path probability</b> from the previous time step
$a_{ij}$	the <b>transition probability</b> from previous state $q_i$ to current state $q_j$
$b_j(o_t)$	the <b>state observation likelihood</b> of the observation symbol $o_t$ given the current state $j$

# Viterbi Algorithm

```
function VITERBI(observations of len  $T$ , state-graph of len  $N$ ) returns best-path
    create a path probability matrix  $viterbi[N+2,T]$ 
    for each state  $s$  from 1 to  $N$  do ; initialization step
         $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$ 
         $backpointer[s,1] \leftarrow 0$ 
    for each time step  $t$  from 2 to  $T$  do ; recursion step
        for each state  $s$  from 1 to  $N$  do
             $viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$ 
             $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * a_{s',s}$ 
     $viterbi[q_F, T] \leftarrow \max_{s=1}^N viterbi[s, T] * a_{s,q_F}$  ; termination step
     $backpointer[q_F, T] \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T] * a_{s,q_F}$  ; termination step
    return the backtrace path by following backpointers to states back in
          time from  $backpointer[q_F, T]$ 
```

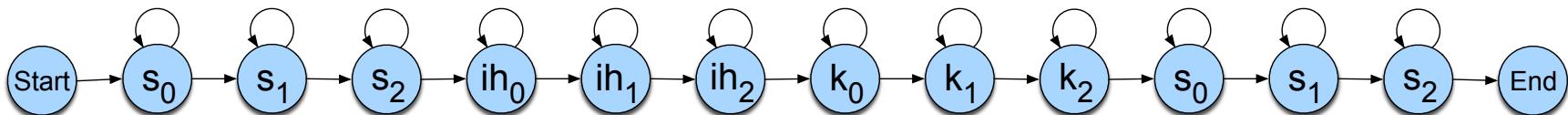
# Viterbi backtrace



# HMMs for Speech

- We haven't yet shown how to learn the A and B matrices for HMMs;
  - ◆ we'll do that on Thursday
  - ◆ The Baum-Welch (Forward-Backward alg)
- But let's return to think about speech

# Reminder: a word looks like this:



$$Q = q_1 q_2 \dots q_N$$

a set of states corresponding to subphones

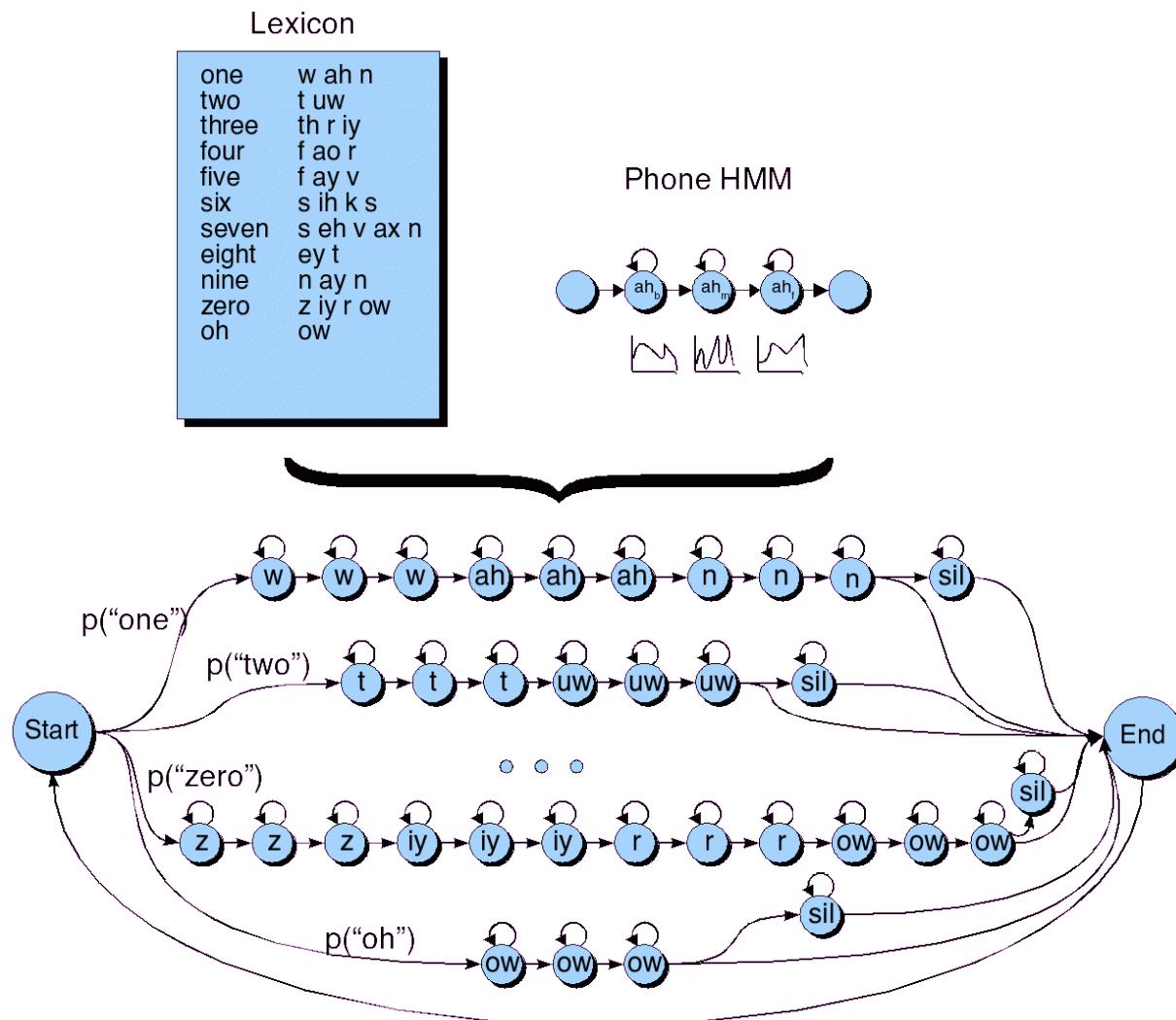
$$A = a_{01} a_{02} \dots a_{n1} \dots a_{nn}$$

a **transition probability matrix**  $A$ , each  $a_{ij}$  representing the probability for each subphone of taking a **self-loop** or going to the next subphone. Together,  $Q$  and  $A$  implement a **pronunciation lexicon**, an HMM state graph structure for each word that the system is capable of recognizing.

$$B = b_i(o_t)$$

A set of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of a cepstral feature vector (observation  $o_t$ ) being generated from subphone state  $i$ .

# HMM for digit recognition task



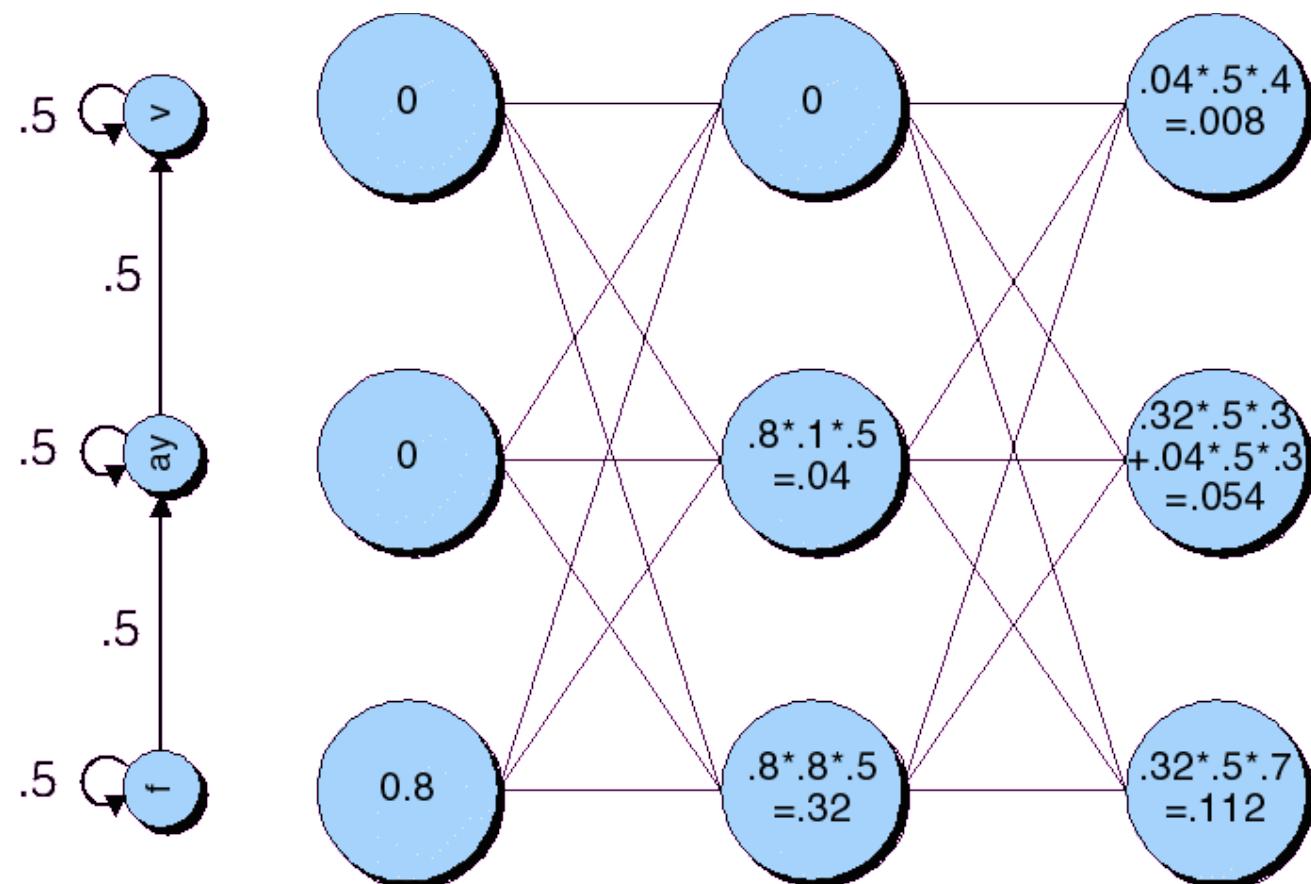
# The Evaluation (forward) problem for speech

- The observation sequence  $O$  is a series of MFCC vectors
- The hidden states  $W$  are the phones and words
- For a given phone/word string  $W$ , our job is to evaluate  $P(O|W)$
- Intuition: how likely is the input to have been generated by just that word string  $W$

# Evaluation for speech: Summing over all different paths!

- f ay ay ay ay v v v v
- f f ay ay ay ay v v v
- f f f f ay ay ay ay v
- f f ay ay ay ay ay ay v
- f f ay ay ay ay ay ay ay v
- f f ay v v v v v v v

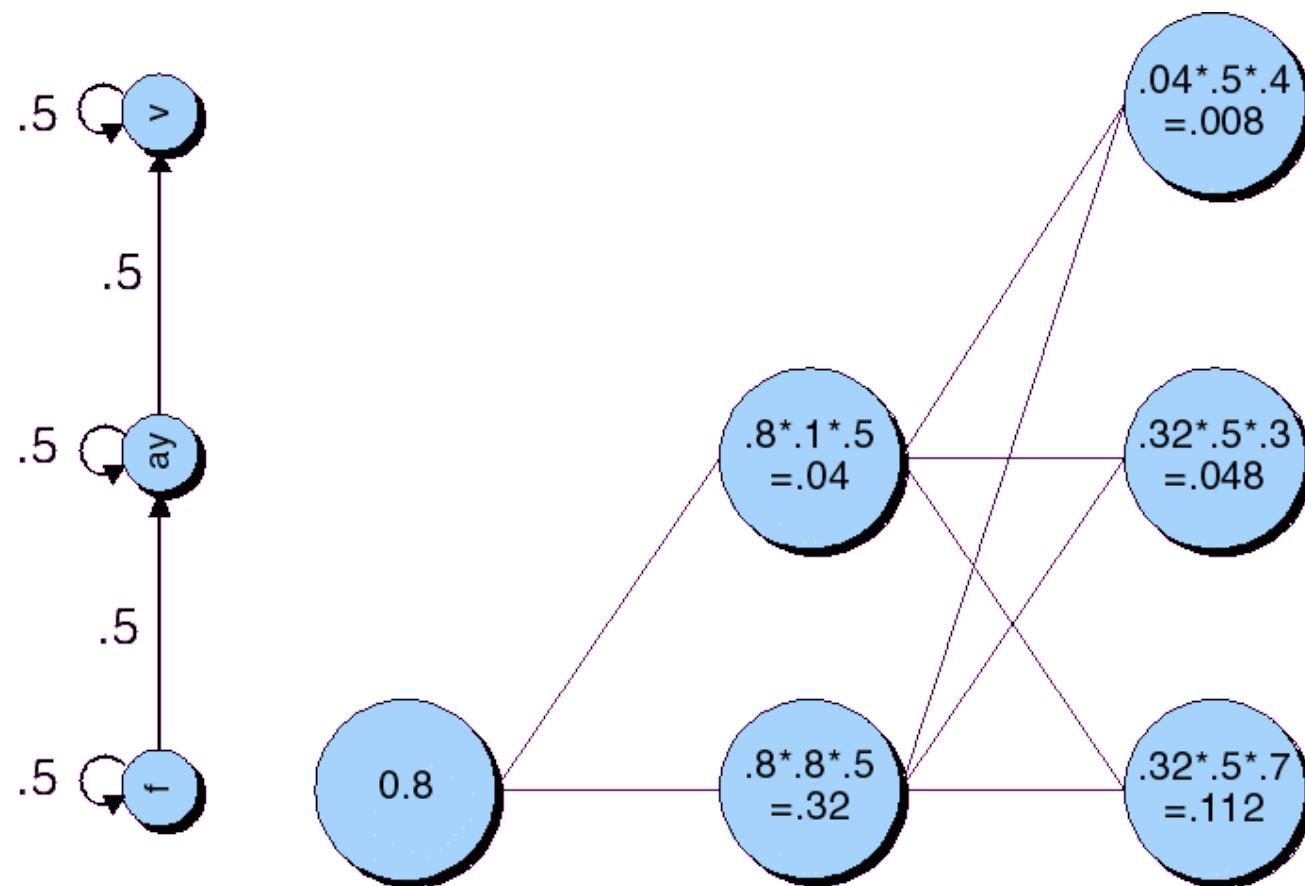
# The forward lattice for “five”



# The forward trellis for “five”

<b>V</b>	0	0	0.008	0.0093	0.0114	0.00703	0.00345	0.00306	0.00206	0.00117
<b>AY</b>	0	0.04	0.054	0.0664	0.0355	0.016	0.00676	0.00208	0.000532	0.000109
<b>F</b>	0.8	0.32	0.112	0.0224	0.00448	0.000896	0.000179	4.48e-05	1.12e-05	2.8e-06
<b>Time</b>	1	2	3	4	5	6	7	8	9	10
<b>B</b>	<i>f</i> 0.8	<i>f</i> 0.8	<i>f</i> 0.7	<i>f</i> 0.4	<i>f</i> 0.4	<i>f</i> 0.4	<i>f</i> 0.4	<i>f</i> 0.5	<i>f</i> 0.5	<i>f</i> 0.5
	<i>ay</i> 0.1	<i>ay</i> 0.1	<i>ay</i> 0.3	<i>ay</i> 0.8	<i>ay</i> 0.8	<i>ay</i> 0.8	<i>ay</i> 0.8	<i>ay</i> 0.6	<i>ay</i> 0.5	<i>ay</i> 0.4
	<i>v</i> 0.6	<i>v</i> 0.6	<i>v</i> 0.4	<i>v</i> 0.3	<i>v</i> 0.3	<i>v</i> 0.3	<i>v</i> 0.3	<i>v</i> 0.6	<i>v</i> 0.8	<i>v</i> 0.9
	<i>p</i> 0.4	<i>p</i> 0.4	<i>p</i> 0.2	<i>p</i> 0.1	<i>p</i> 0.3	<i>p</i> 0.3				
	<i>iy</i> 0.1	<i>iy</i> 0.1	<i>iy</i> 0.3	<i>iy</i> 0.6	<i>iy</i> 0.6	<i>iy</i> 0.6	<i>iy</i> 0.6	<i>iy</i> 0.5	<i>iy</i> 0.5	<i>iy</i> 0.4

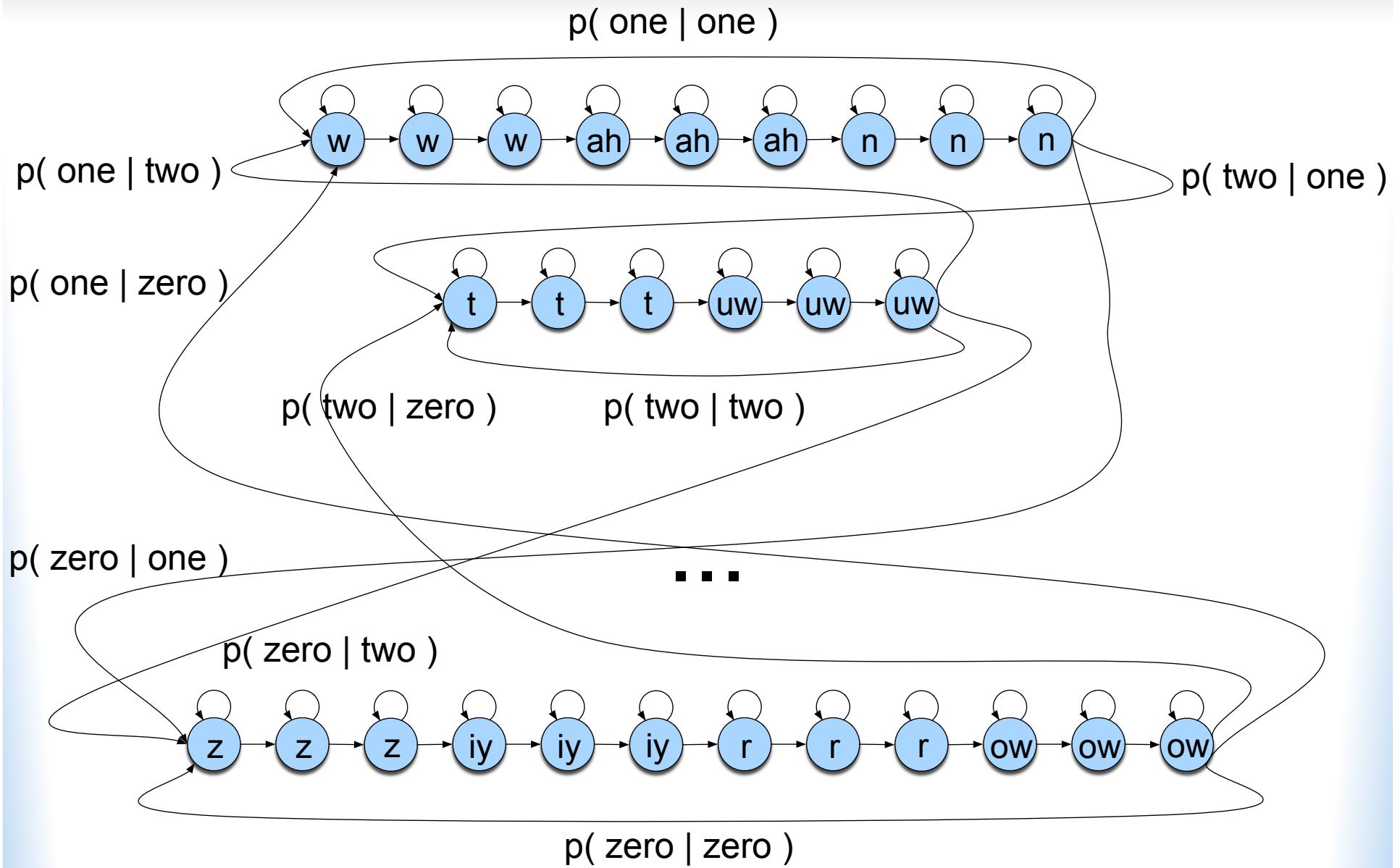
# Viterbi trellis for “five”



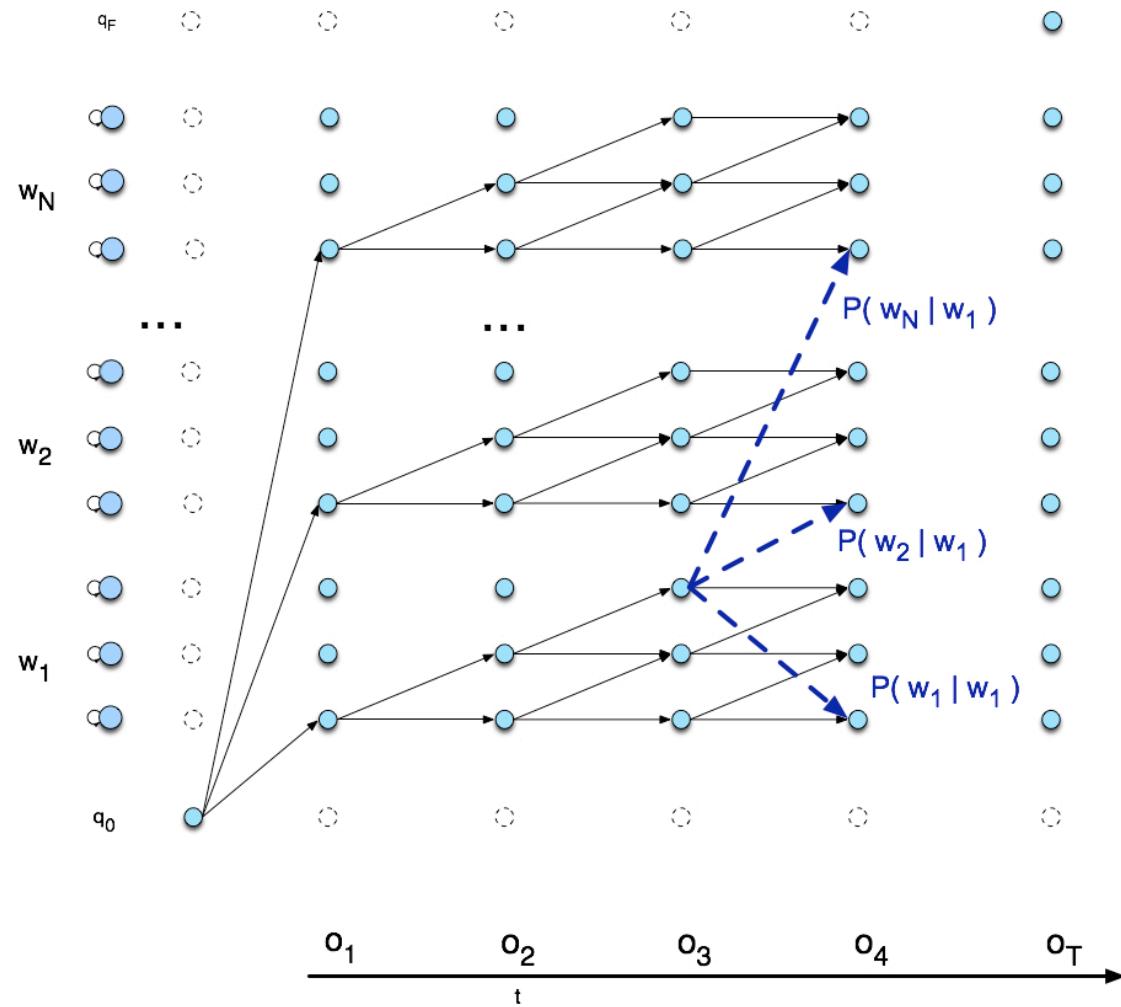
# Viterbi trellis for “five”

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

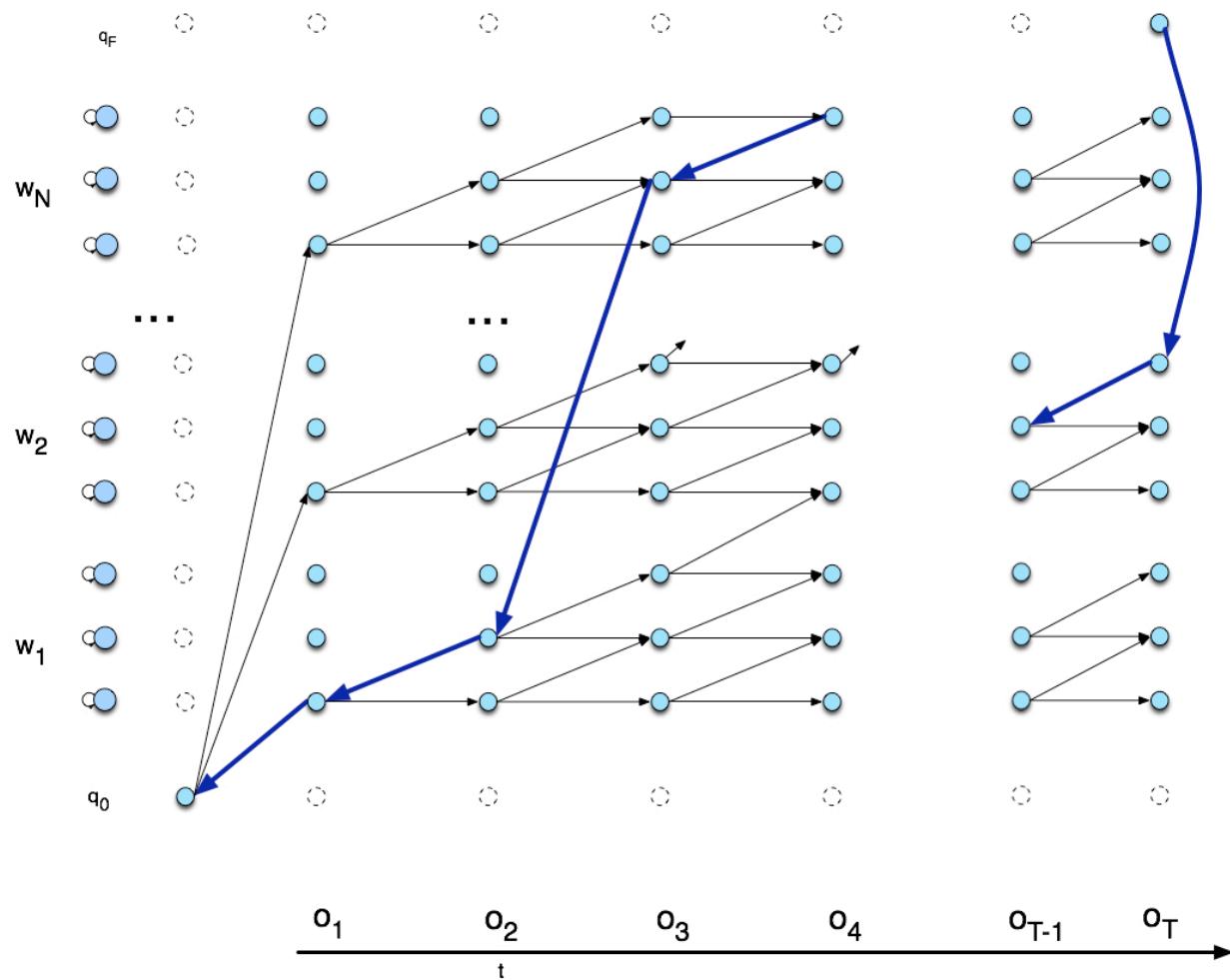
# Search space with bigrams



# Viterbi trellis



# Viterbi backtrace



# Evaluation

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

# Word Error Rate

- Word Error Rate =  
$$\frac{100 \text{ (Insertions+Substitutions + Deletions)}}{\text{Total Word in Correct Transcript}}$$

Aligment example:

REF: portable \*\*\* PHONE UPSTAIRS last night so

HYP: portable FORM OF STORES last night so

Eval I S S

$$\text{WER} = 100 (1+2+0)/6 = 50\%$$

# NIST sctk-1.3 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

CONFUSION PAIRS	Total	(972)
	With >= 1 occurrences	(972)

1:	6	->	(%hesitation) ==> on
2:	6	->	the ==> that
3:	5	->	but ==> that
4:	4	->	a ==> the
5:	4	->	four ==> for
6:	4	->	in ==> and
7:	4	->	there ==> that
8:	3	->	(%hesitation) ==> and
9:	3	->	(%hesitation) ==> the
10:	3	->	(a-) ==> i
11:	3	->	and ==> i
12:	3	->	and ==> in
13:	3	->	are ==> there
14:	3	->	as ==> is
15:	3	->	have ==> that
16:	3	->	is ==> this

# Sclite output for error analysis

```
17:    3 -> it ==> that
18:    3 -> mouse ==> most
19:    3 -> was ==> is
20:    3 -> was ==> this
21:    3 -> you ==> we
22:    2 -> (%hesitation) ==> it
23:    2 -> (%hesitation) ==> that
24:    2 -> (%hesitation) ==> to
25:    2 -> (%hesitation) ==> yeah
26:    2 -> a ==> all
27:    2 -> a ==> know
28:    2 -> a ==> you
29:    2 -> along ==> well
30:    2 -> and ==> it
31:    2 -> and ==> we
32:    2 -> and ==> you
33:    2 -> are ==> i
34:    2 -> are ==> were
```

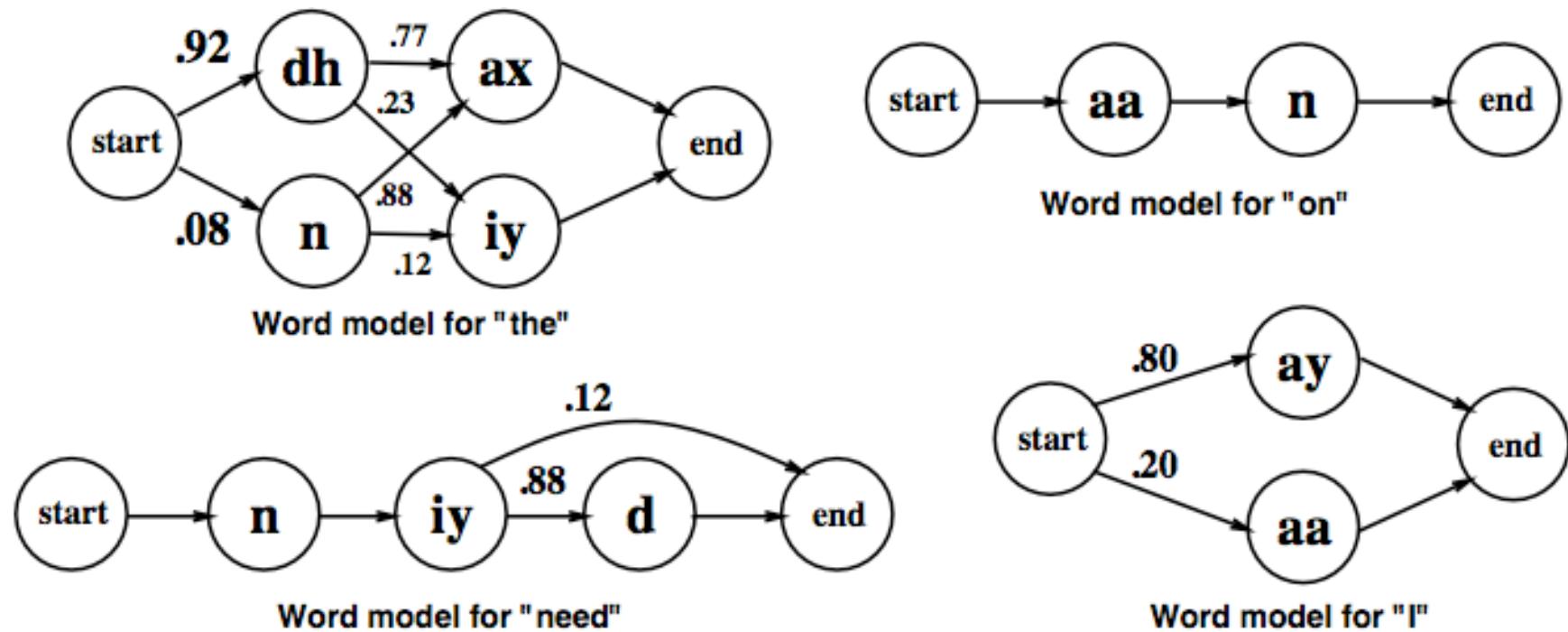
# Better metrics than WER?

- 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

# Summary: ASR Architecture

- Five easy pieces: ASR Noisy Channel architecture
  - 1) Feature Extraction:  
39 “MFCC” features
  - 2) Acoustic Model:  
Gaussians for computing  $p(o|q)$
  - 3) Lexicon/Pronunciation Model
    - HMM: what phones can follow each other
  - 4) Language Model
    - N-grams for computing  $p(w_i|w_{i-1})$
  - 5) Decoder
    - Viterbi algorithm: dynamic programming for combining all these to get word sequence from speech!

# ASR Lexicon: Markov Models for pronunciation



# Summary

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