# CS 224S / LINGUIST 285
# Spoken Language Processing


Dan Jurafsky
Stanford University

Spring 2014

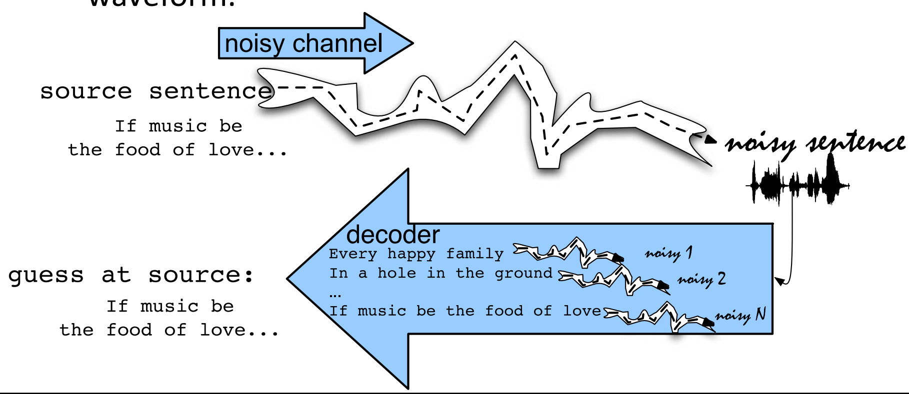
## Lecture 3: ASR: HMMs, Forward, Viterbi

## Outline for Today

- ASR Architecture
- Decoding with HMMs
  - Forward
  - Viterbi Decoding
- How this fits into the ASR component of course
  - On your own: N-grams and Language Modeling
  - Apr 10: Training: Baum-Welch (Forward-Backward)
  - Apr 15: Acoustic Modeling and GMMs
  - Apr 17: Feature Extraction, MFCCs
  - May 27: Andrew on Deep Neural Net Acoustic Models

## 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
  - $O = o_1, o_2, o_3, \ldots, o_t$
- Define a sentence as a sequence of words:
  - $W = w_1, w_2, w_3, \ldots, w_n$

## Noisy Channel Model (III)

- Probabilistic implication: Pick the highest prob S:

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

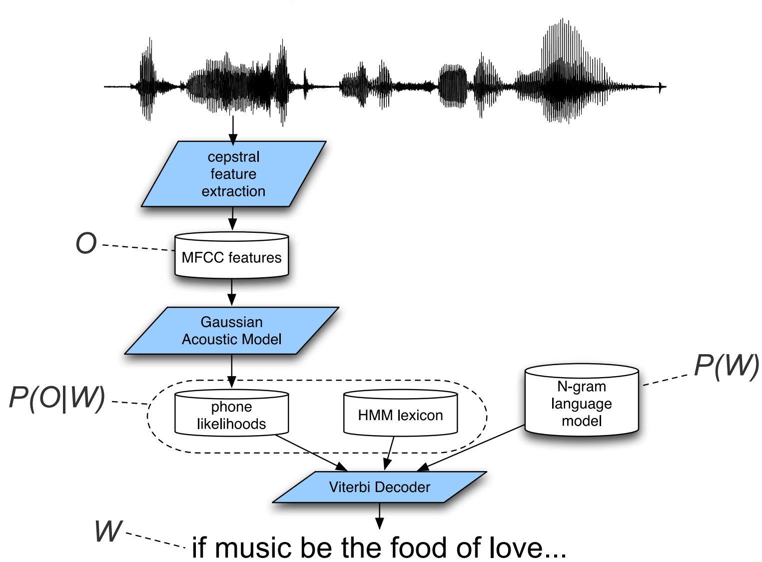
- We can use Bayes rule to rewrite this:

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

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

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

## Speech Recognition Architecture

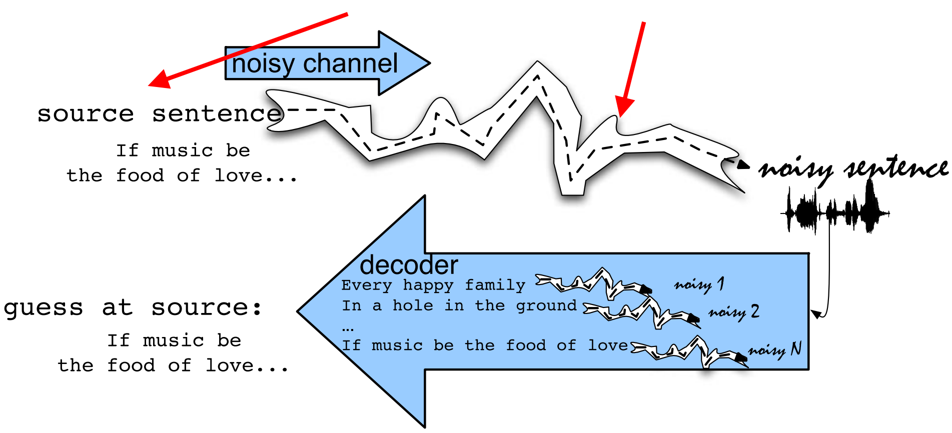


## Noisy channel model

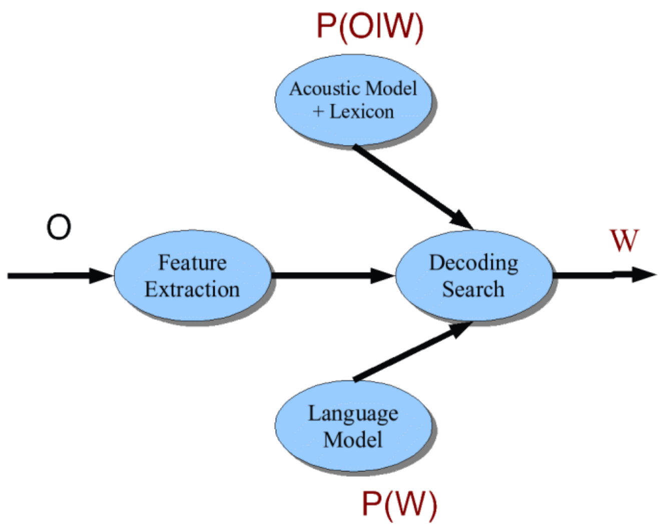
$$\hat{W} = \underset{W \in L}{\arg\max}\, \overset{\text{likelihood}}{P(O|W)}\, \overset{\text{prior}}{P(W)}$$

## The noisy channel model

Ignoring the denominator leaves us with two factors: P(Source) and P(Signal|Source)



## Speech Architecture meets Noisy Channel




## Decoding Architecture: five easy pieces

- Feature Extraction:
  - 39 "MFCC" features
- Acoustic Model:
  - Gaussians for computing $p(o|q)$
- Lexicon/Pronunciation Model
  - HMM: what phones can follow each other
- Language Model
  - N-grams for computing $p(w_i|w_{i-1})$
- Decoder
  - Viterbi algorithm: dynamic programming for combining all these to get word sequence from speech

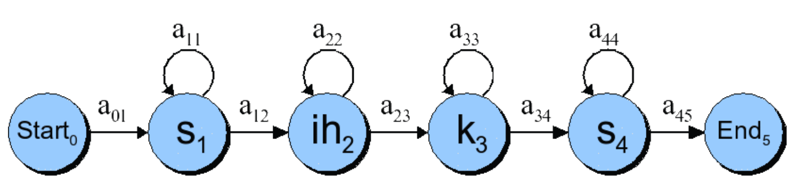
10

## Lexicon

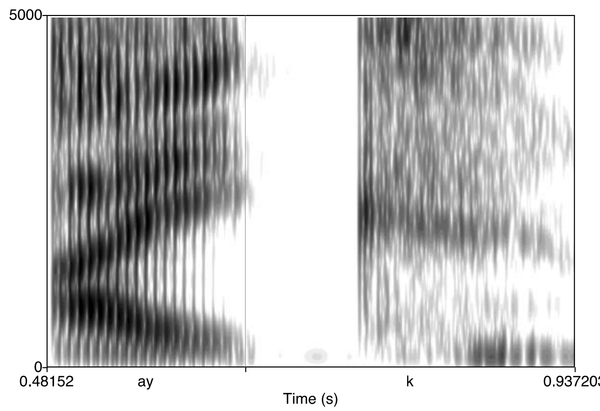
- A list of words
- Each one with a pronunciation in terms of phones
- We get these from on-line pronunciation dictionary
- CMU dictionary: 127K words
  - http://www.speech.cs.cmu.edu/cgi-bin/cmudict
- We'll represent the lexicon as an HMM

11

## HMMs for speech

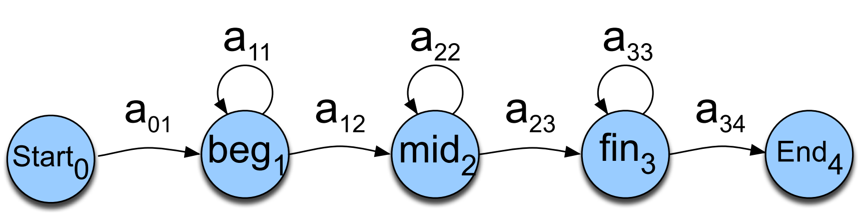


## Phones are not homogeneous!

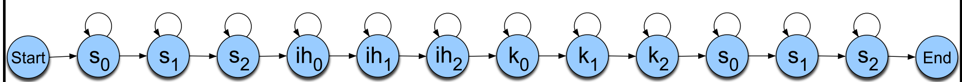



ay k

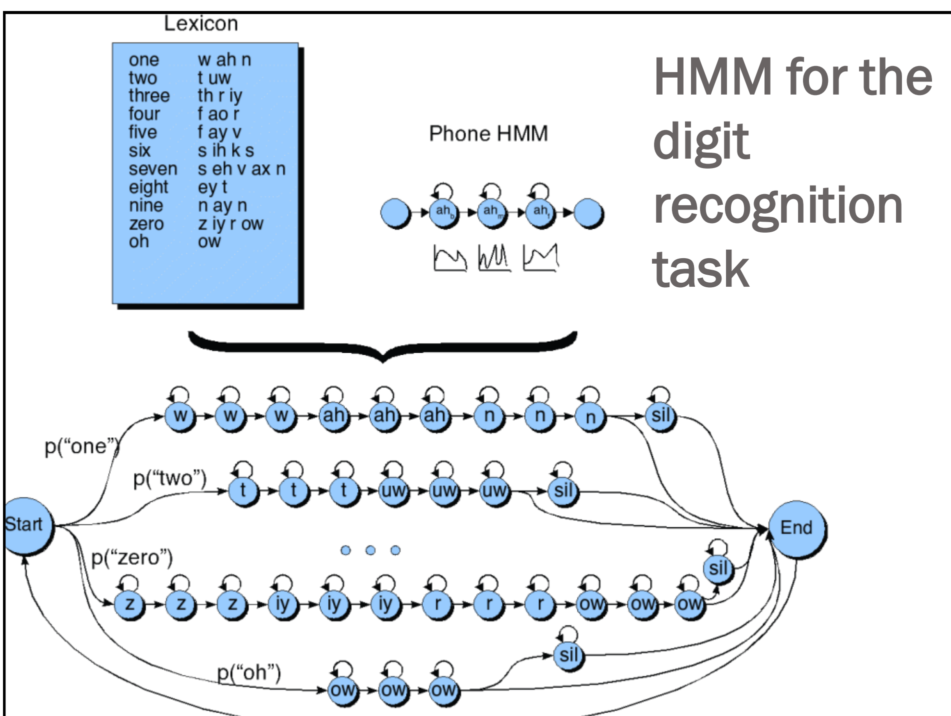
## Each phone has 3 subphones



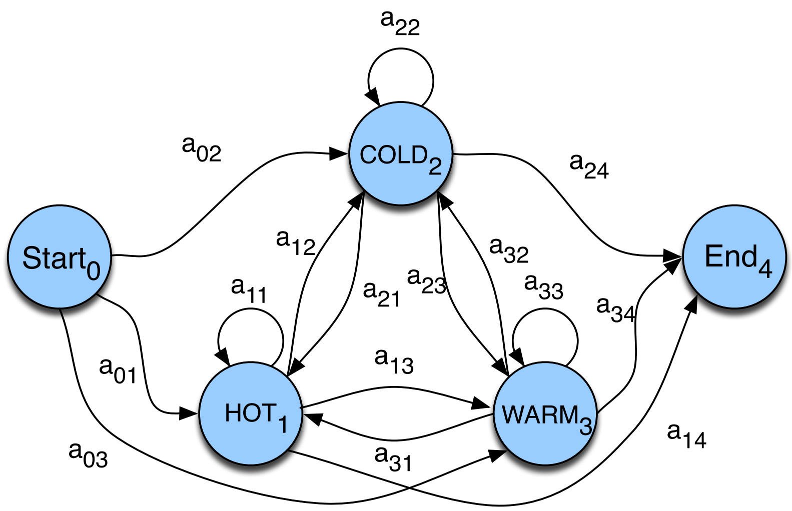
# Resulting HMM word model for "six"



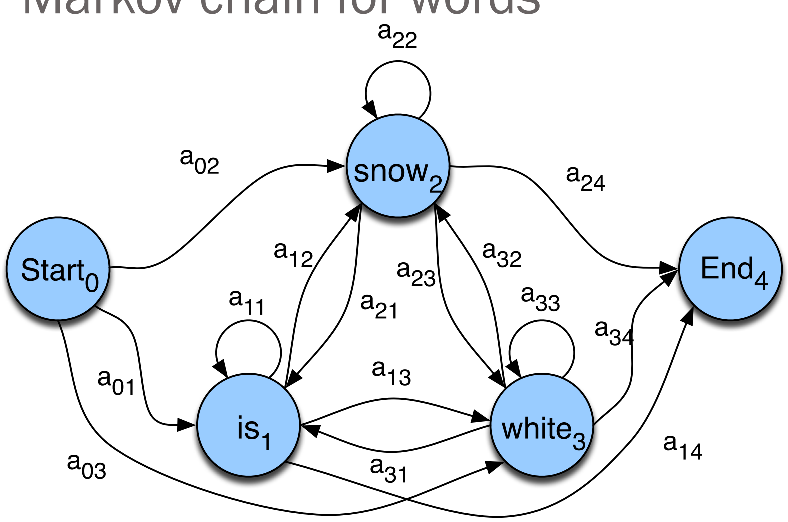

# HMM for the digit recognition task



# Markov chain for weather




# Markov chain for words



## Markov chain = First-order observable Markov Model

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

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

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

- Distinguished start and end states

## Markov chain = First-order observable Markov Model

Current state only depends on previous state

$$\text{Markov Assumption}: \quad P(q_i \mid q_1 \cdots q_{i-1}) = P(q_i \mid q_{i-1})$$

## Another representation for start state

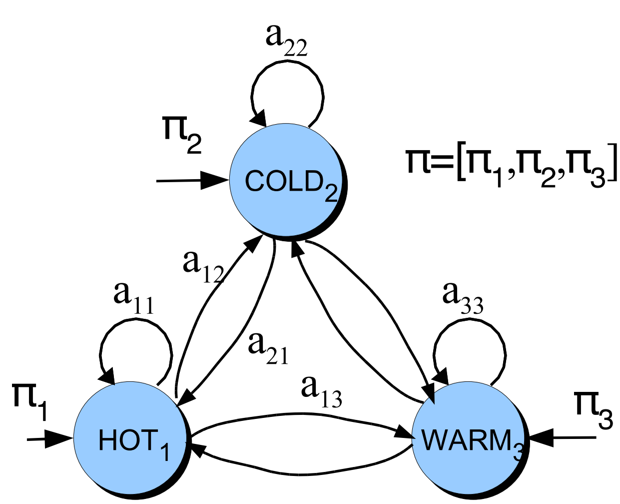
- Instead of start state
- Special initial probability vector $\pi$
  - An initial distribution over probability of start states

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

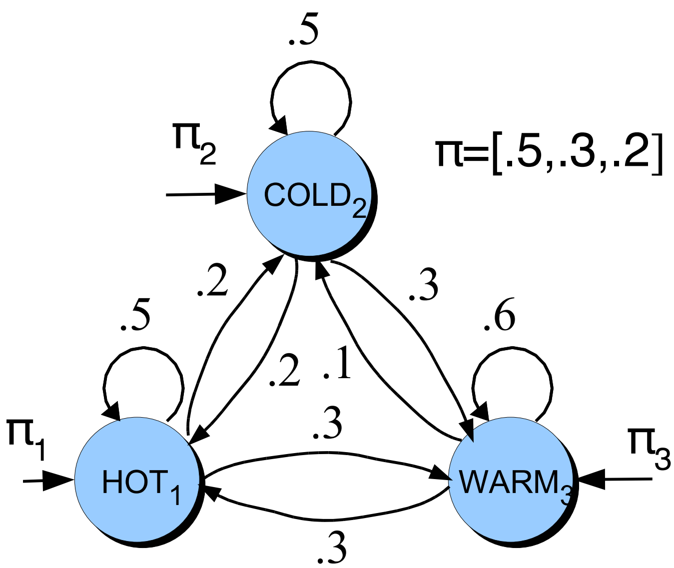
- Constraints:

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

## 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, output symbols = state symbols
  - See **hot** weather: we're in state **hot**
- But not in speech recognition
  - Output symbols: vectors of acoustics (**cepstral features)**
  - Hidden states: **phones**
- 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

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

## Assumptions

- **Markov assumption:**

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

- **Output-independence assumption**

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

## Eisner task

Given

Observed Ice Cream Sequence:

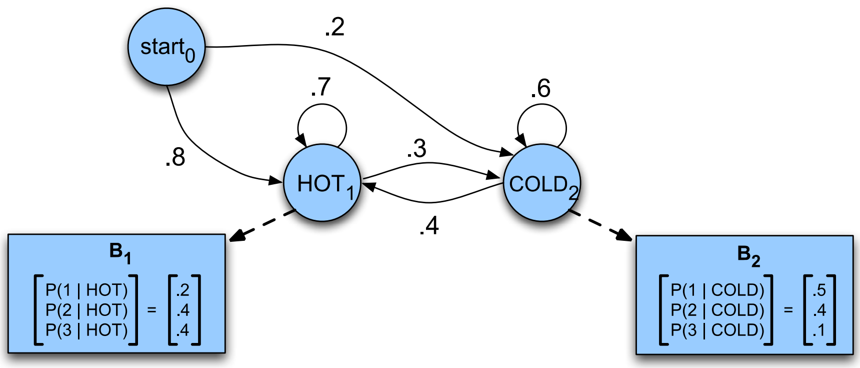
**1,2,3,2,2,2,3…**

Produce:

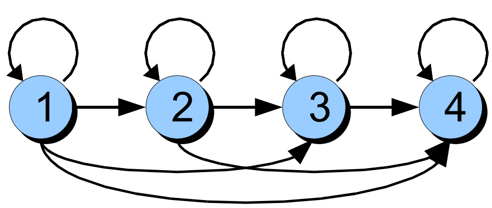
Hidden Weather Sequence:

**H,C,H,H,H,C…**

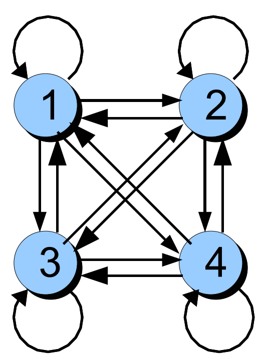
# HMM for ice cream




# 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

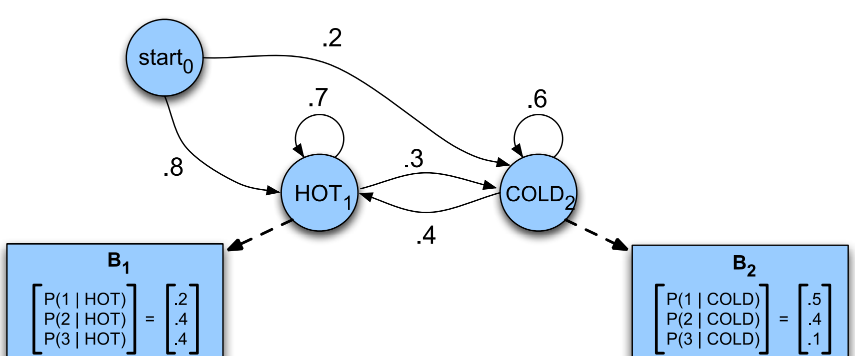
Problem 1 (**Evaluation**): Given the observation sequence $O=(o_1o_2\ldots o_T)$, and an HMM model $\Phi$ = (A,B), how do we efficiently compute $P(O|\ \Phi)$, the probability of the observation sequence, given the model?

Problem 2 (**Decoding**): Given the observation sequence $O=(o_1o_2\ldots o_T)$, and an HMM model $\Phi$ = (A,B), how do we choose a corresponding state sequence $Q=(q_1q_2\ldots q_T)$ 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 $P(O|\lambda)$.




How likely is the sequence 3 1 3?

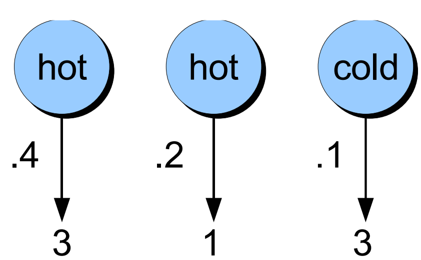
## How to compute likelihood

- 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

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

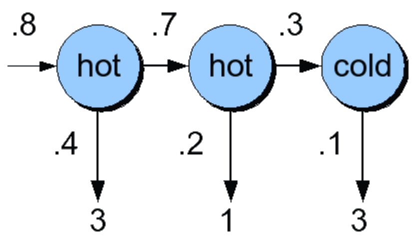
$$P(3\ 1\ 3|\text{hot hot cold}) \;=\; P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold})$$



## 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
  - ….

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

- How many possible hidden state sequences are there for this sequence?

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

- 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 **dynamic programming** algorithm
  - Just like Minimum Edit Distance or CKY Parsing
  - 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

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

- 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

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

## The Forward Recursion

1. Initialization:

$$\alpha_1(j) \;=\; a_{0j}b_j(o_1) \;\; 1 \le j \le N$$

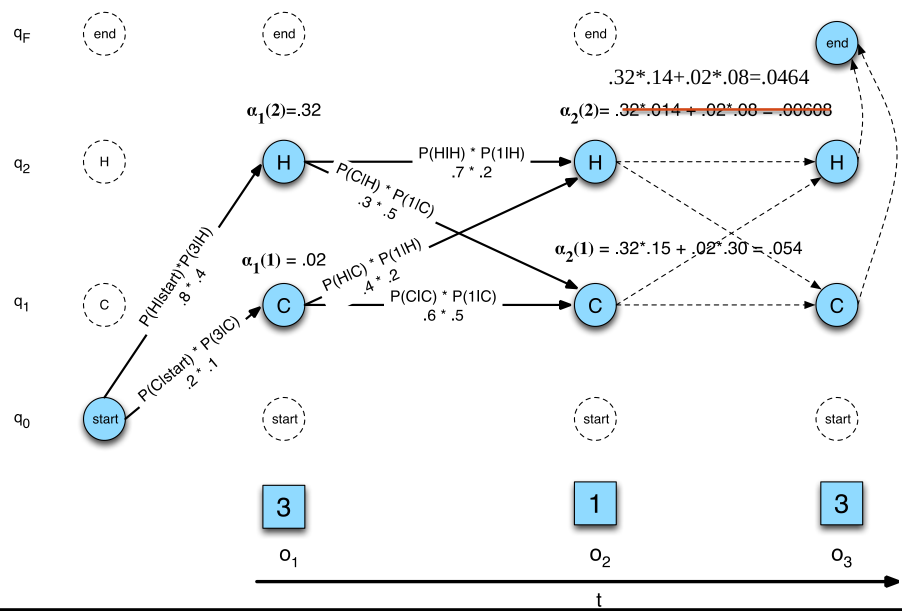
2. Recursion (since states 0 and F are non-emitting):

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

3. Termination:

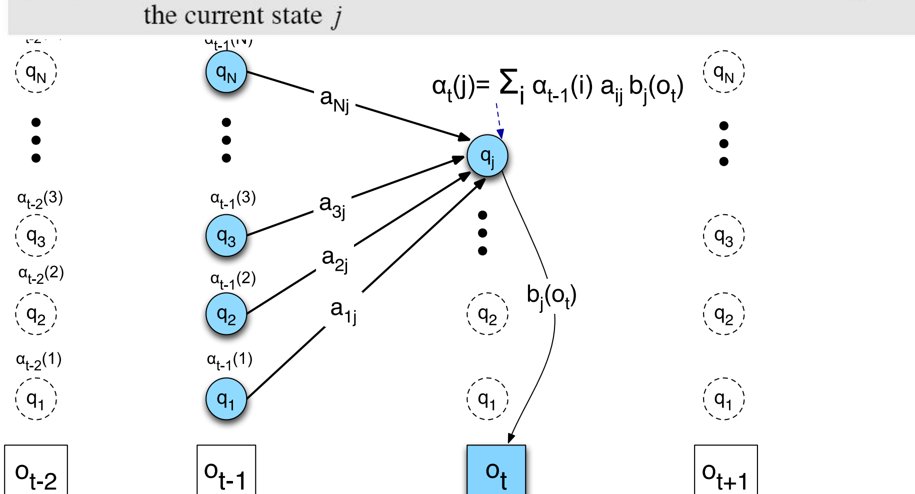
$$P(O|\lambda) = \alpha_T(q_F) = \sum_{i=1}^{N} \alpha_T(i) a_{iF}$$

## 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
$forward[s,1] \leftarrow a_{0,s} * b_s(o_1)$
**for** each time step $t$ **from** 2 **to** $T$ **do** ; recursion step
**for** each state $s$ **from** 1 **to** $N$ **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forward[q_F,T] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$$ ; 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 $O=(o_1o_2\ldots o_T)$, and an HMM model $\Phi$ = (A,B), how do we choose a corresponding state sequence $Q=(q_1q_2\ldots q_T)$ 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

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

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

# Viterbi Recursion

1. **Initialization:**

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

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

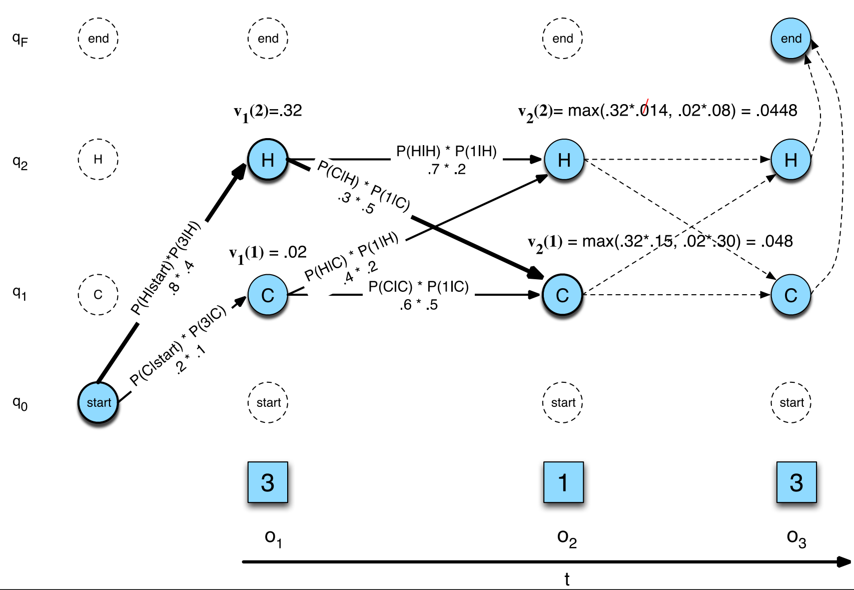
$$\begin{aligned} v_t(j) &= \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \\ bt_t(j) &= \operatorname*{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \end{aligned}$$

3. **Termination:**

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

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

# The Viterbi trellis



## Viterbi intuition

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

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

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

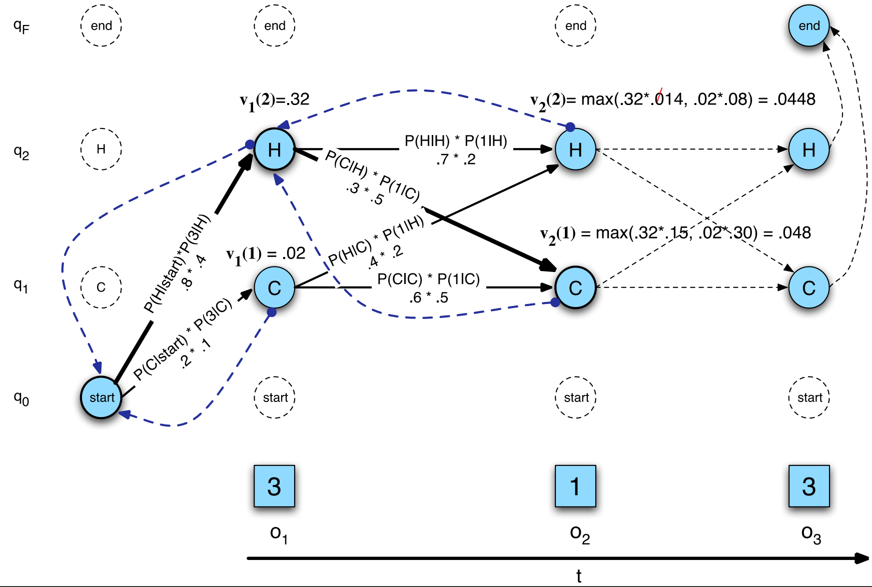
| | |
|---|---|
| $v_{t-1}(i)$ | the **previous Viterbi path probability** from the previous time step |
| $a_{ij}$ | the **transition probability** from previous state $q_i$ to current state $q_j$ |
| $b_j(o_t)$ | the **state observation likelihood** of the observation symbol $o_t$ given the current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

## 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 $backpointer[q_F,T]$

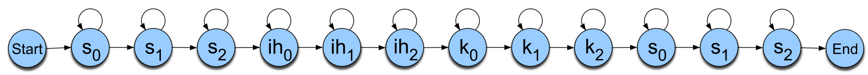
# Viterbi backtrace




# HMMs for Speech

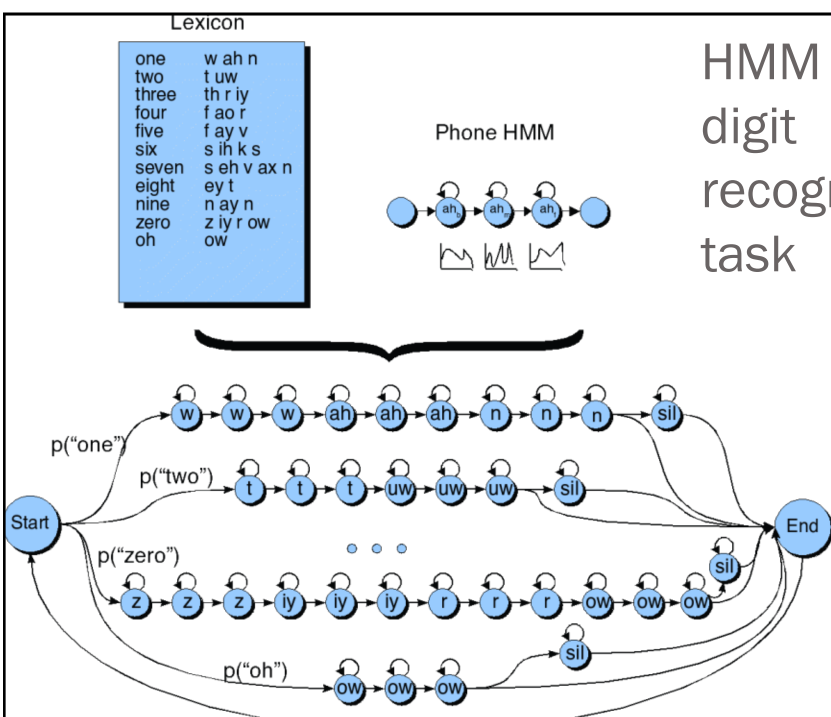
- We haven't yet shown how to **learn** the A and B matrices for HMMs;
  - we'll do that on Thursday
  - The Baum-Welch (Forward-Backward alg)
- But let's return to think about speech

# 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



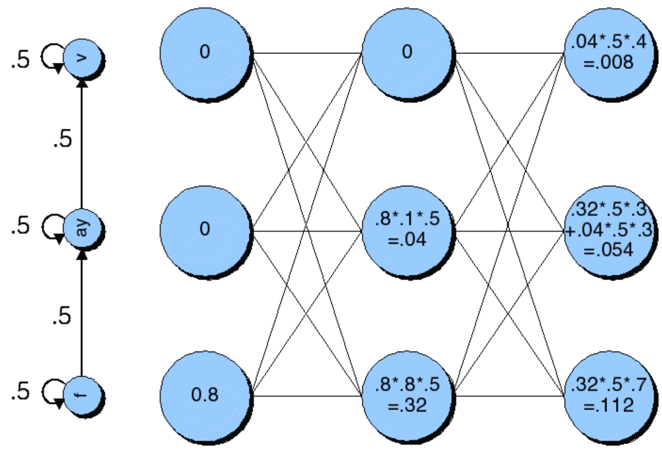
## The Evaluation (forward) problem for speech

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

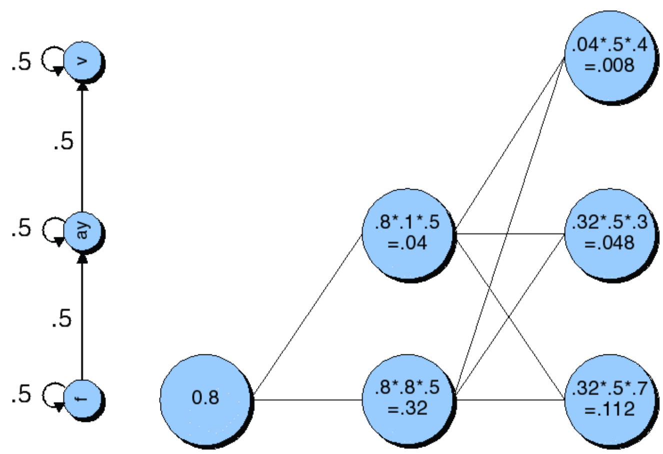
# The forward lattice for “five”




# The forward trellis for “five”

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **V** | 0 | 0 | 0.008 | 0.0093 | 0.0114 | 0.00703 | 0.00345 | 0.00306 | 0.00206 | 0.00117 |
| **AY** | 0 | 0.04 | 0.054 | 0.0664 | 0.0355 | 0.016 | 0.00676 | 0.00208 | 0.000532 | 0.000109 |
| **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* |

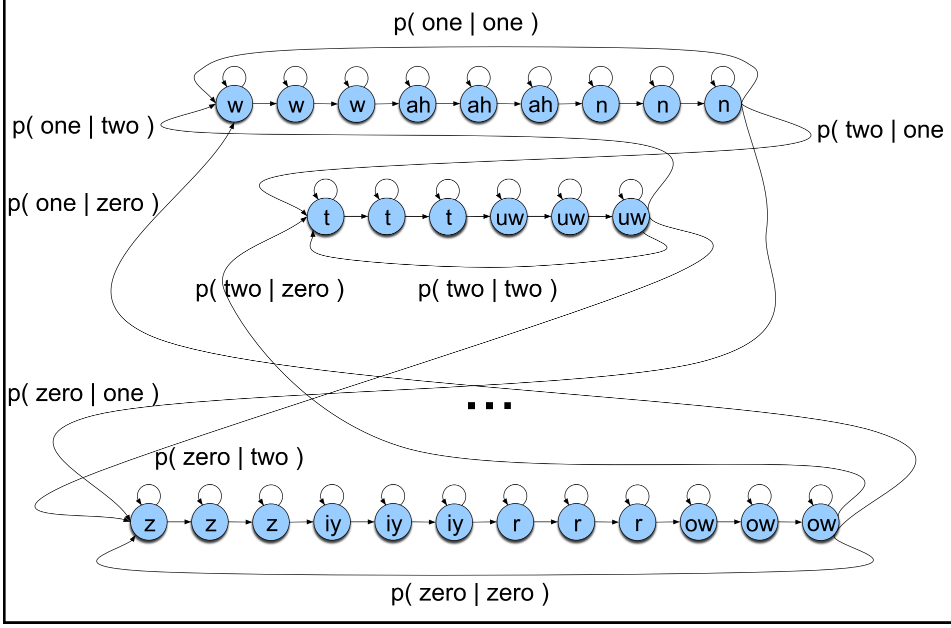
## Viterbi trellis for “five”



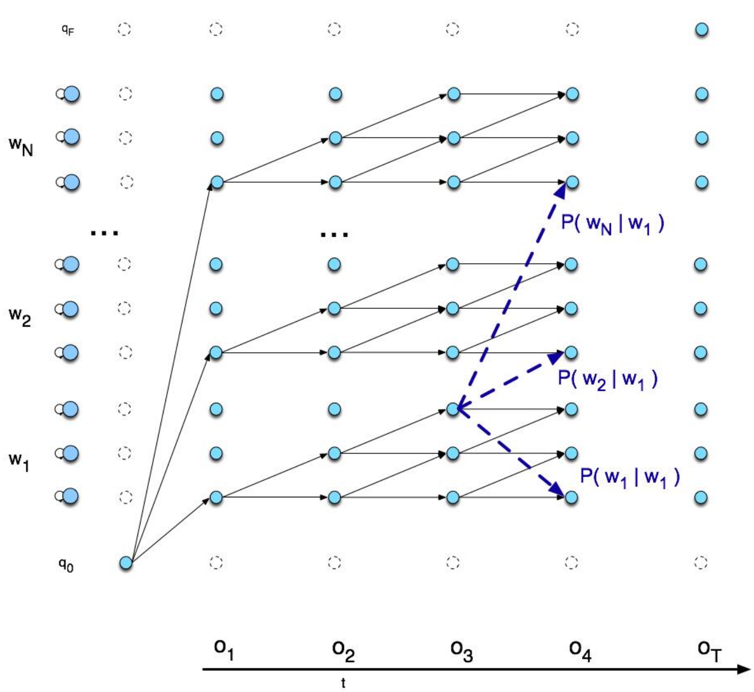

## Viterbi trellis for “five”

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **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

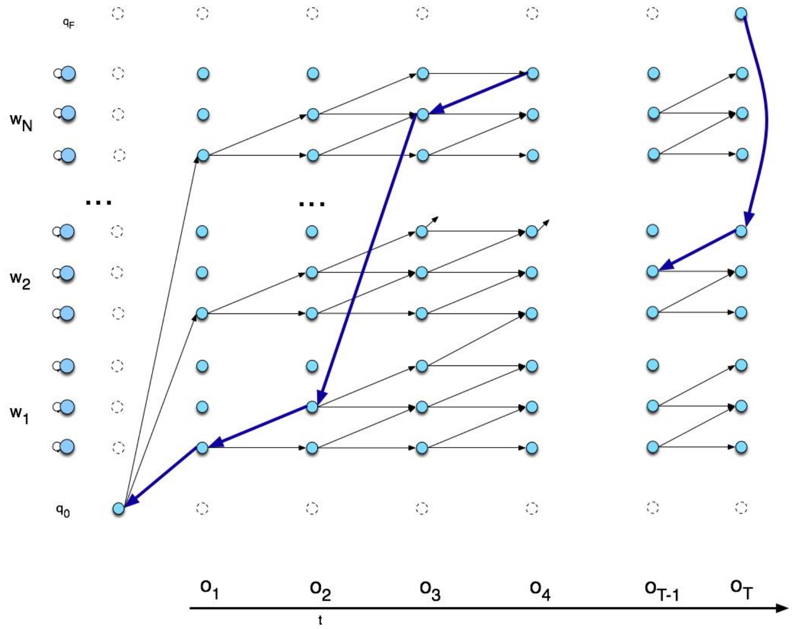



## Viterbi trellis




64

# Viterbi backtrace






# Evaluation

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

## Word Error Rate

- Word Error Rate =

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

- Aligment example:

```
REF:  portable ****  PHONE UPSTAIRS last night so
HYP:  portable FORM  OF    STORES   last night so
Eval           I     S     S
```

- WER = 100 (1+2+0)/6 = 50%

## NIST sctk 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 occurrances (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
```

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

71