Speech Recognition Architecture:

HMMs: Decoding and Learning

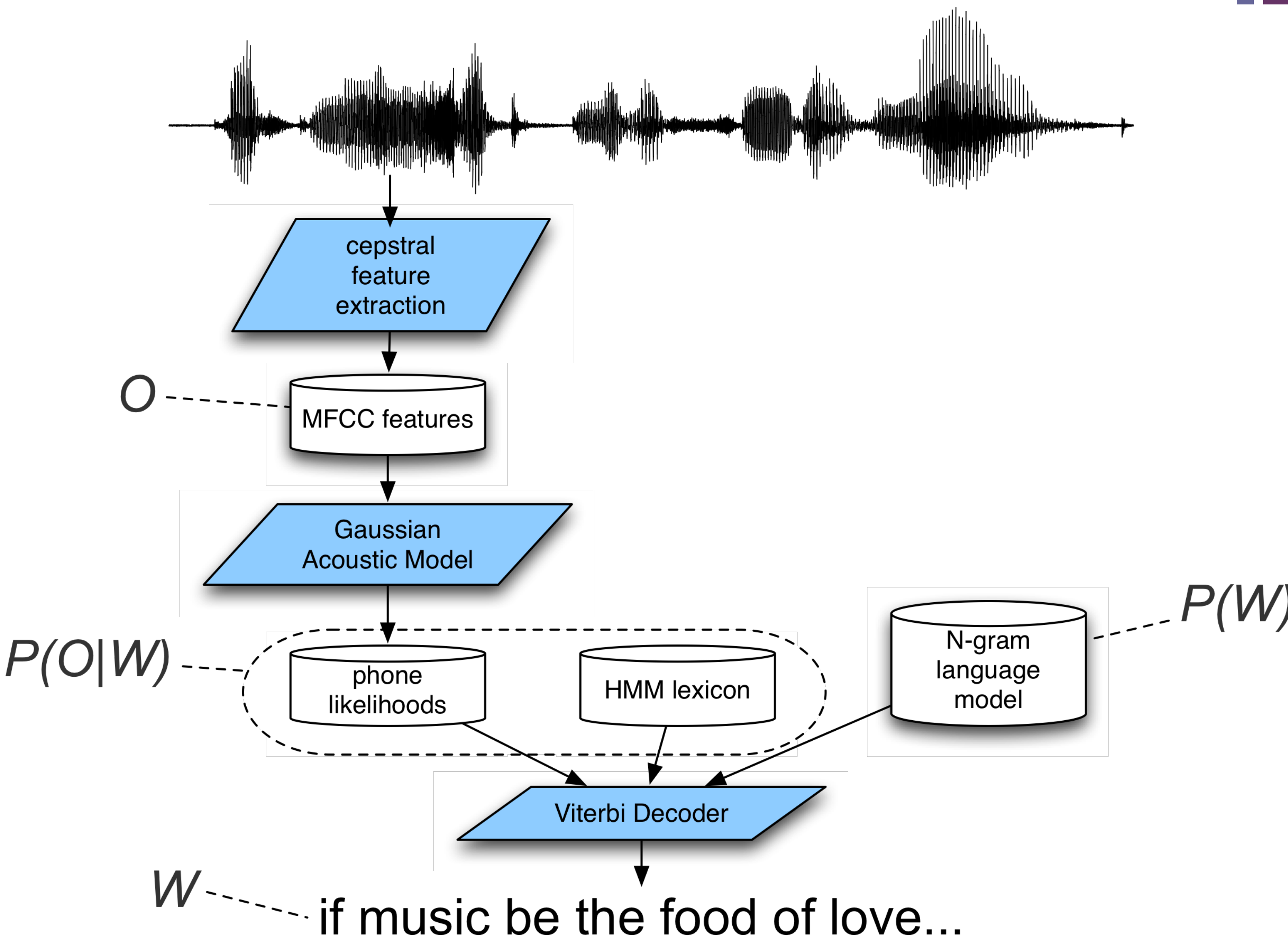
CS 136a Speech Recognition

February 7, 2020

Professor Meteer

Thanks to Dan Jurafsky for these slides

Speech Recognition Architecture



Thanks to Dan Jurafsky for these slides

+ ASR components

- Feature Extraction, MFCCs, start of AM
- HMMs, Forward, Viterbi,
- Baum-Welch (Forward-Backward)
- Acoustic Modeling and GMMs
- N-grams and Language Modeling
- Search and Advanced Decoding
- Dealing with Variation

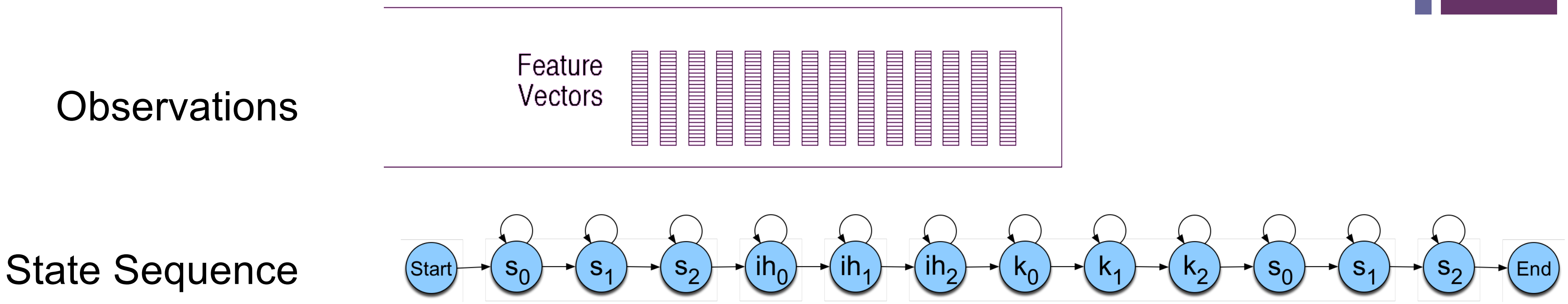
+ The Three Basic Problems for HMMs

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 $P(O|\Phi)$?

Thanks to Dan Jurafsky for these slides

Two kinds of probabilities



- **A:** State transition probabilities
  - What's the likelihood of being state i, given the previous state is state i-1
- **B:** Observation likelihood probabilities $p(o_i|s_i)$
  - Given this observation, what's the likelihood to be in the state

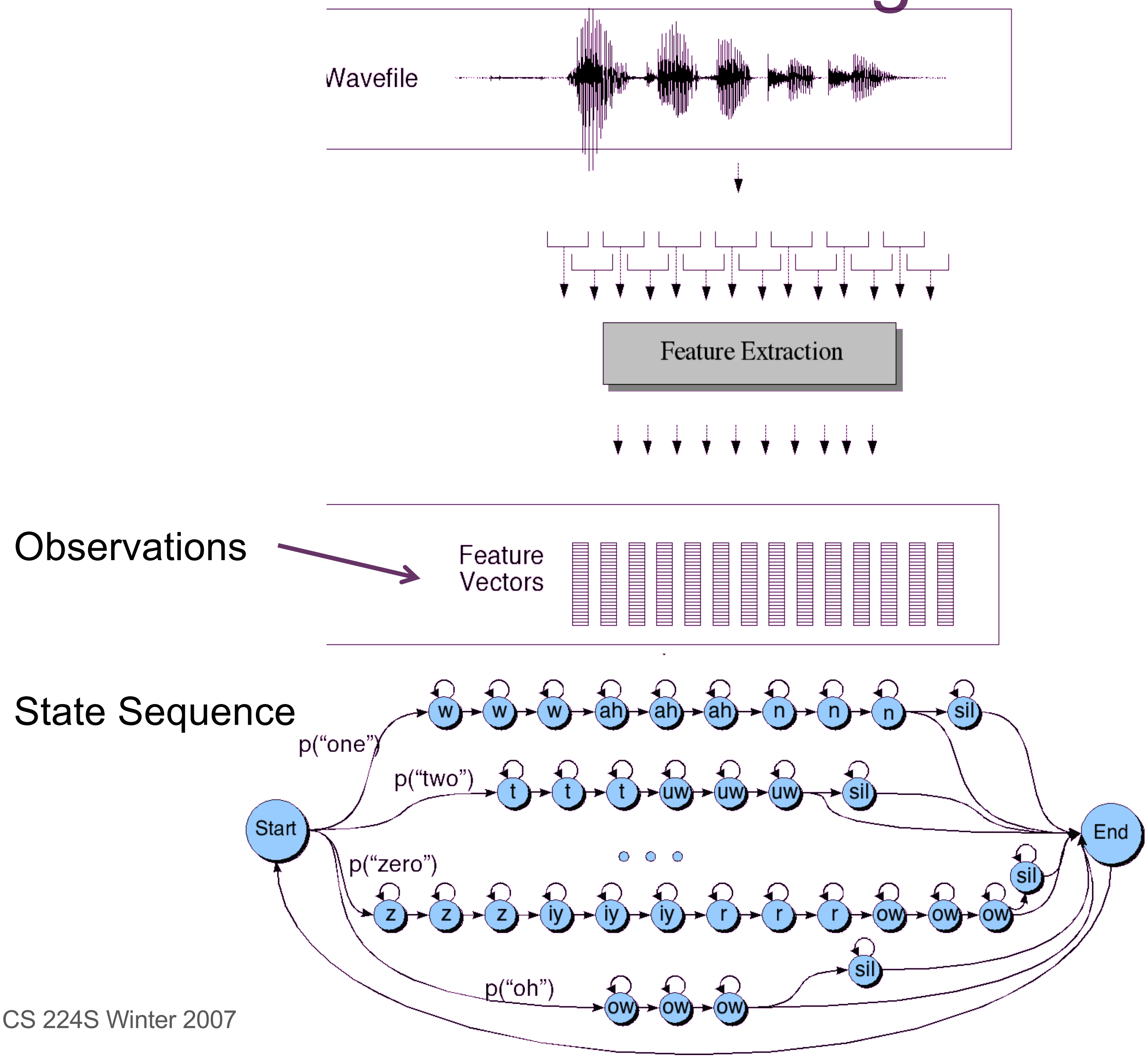
Thanks to Dan Jurafsky for these slides

+ Where we are

- Assume we have learned our observations and transition probabilities and can just look them up
  - We'll get to where those come from
- We have an audio signal (e.g. sequence of observations) and now
  - We want to know the sequence of states
    - Which will tell us the sequence of phonemes
    - Which will tell us the words
- Right now, we are only looking at states within a word
  - word sequence probabilities come from the language model

Thanks to Dan Jurafsky for these slides

+ Front End and Decoding



Which is the best path?

+ Decoding

- Imagine we have a complete model that can give us both transition and observation probabilities.

- This equation is guaranteed to give us the best state sequence

$$\hat{s}_1^n = \operatorname{argmax}_{s_1^n} P(s_1^n \mid o_1^n)$$

- We could just enumerate all paths given the input and use the model to assign probabilities to each.
  - Not a good idea (NT)
  - Luckily dynamic programming helps us here

- But how to make it operational? How to compute this value?

- Intuition of Bayesian classification:
  - Use Bayes rule to transform this equation into a set of other probabilities that are easier to compute

Using Bayes Rule

$$\boxed{P(x|y) = \frac{P(y|x)P(x)}{P(y)}}$$

$$P(s \mid o) = \frac{P(o \mid s)P(s)}{P(o)}$$

Words and tags
(example in the book)

Observations and states

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

$$\hat{s}_1^n = \operatorname{argmax}_{s_1^n} \frac{P(o_1^n \mid s_1^n)P(s_1^n)}{P(o_1^n)}$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n|t_1^n)P(t_1^n)$$

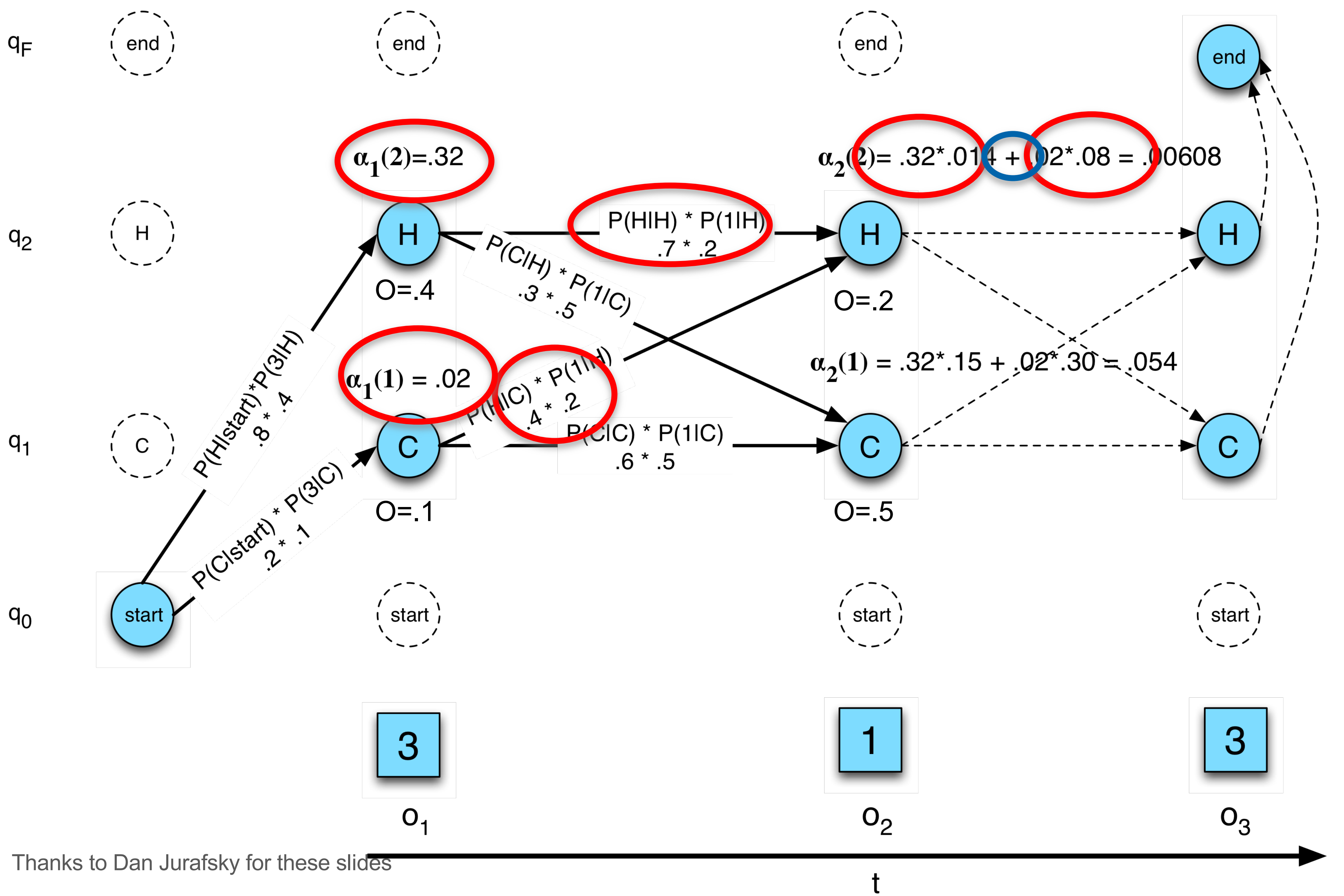
$$\hat{s}_1^n = \operatorname{argmax}_{s_1^n} P(o_1^n \mid s_1^n)P(s_1^n)$$

+ Using 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**

Thanks to Dan Jurafsky for these slides

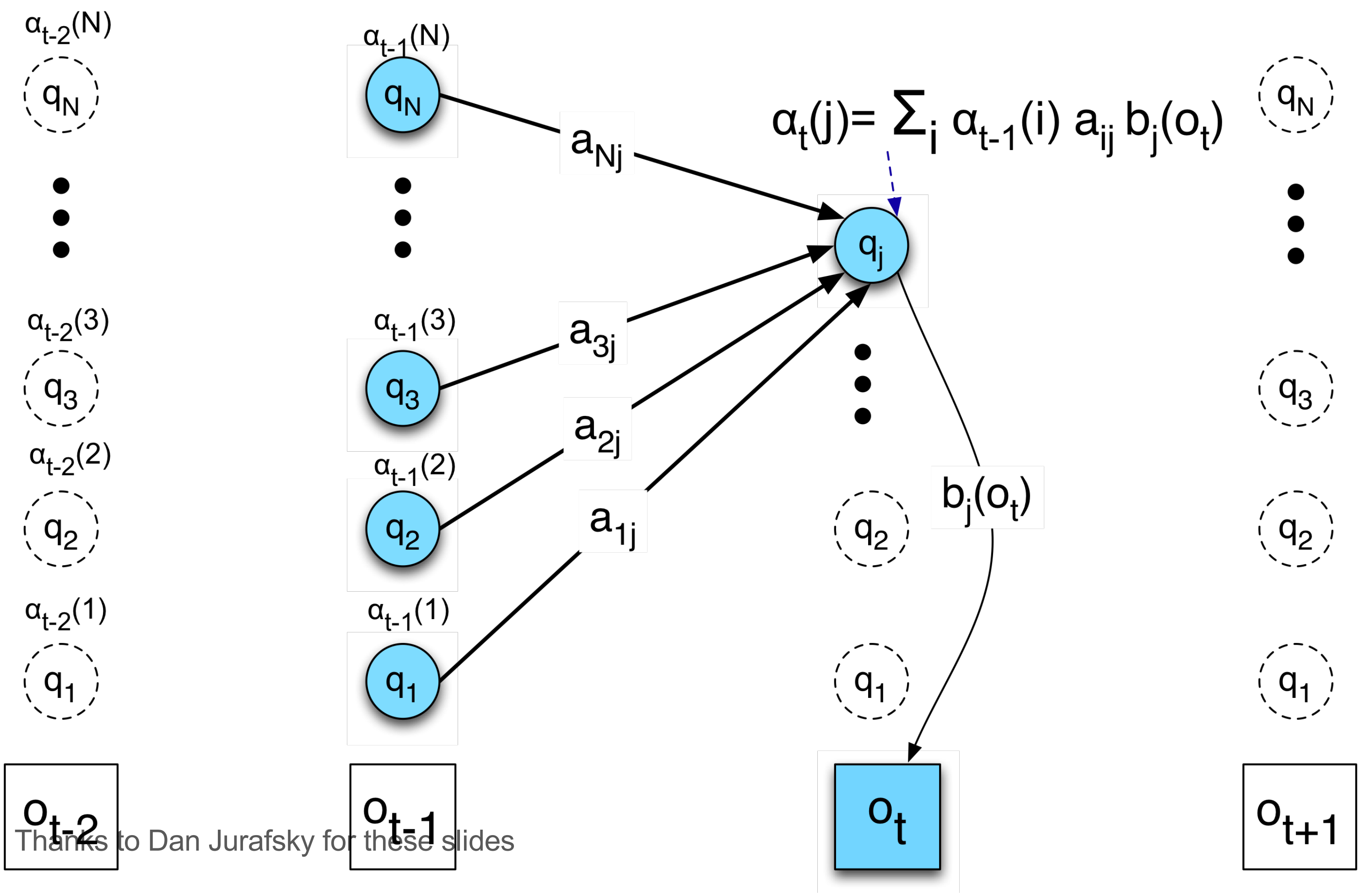
+ The Forward Trellis (quick look)



Thanks to Dan Jurafsky for these slides

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



Thanks to Dan Jurafsky for these slides

+ Decoding

- Given an observation sequence and an HMM, the task of the **decoder:** find the best **hidden** state sequence

- Given
  - the observation sequence $O=(o_1 o_2 \ldots o_T)$,
  - and an HMM model $\Phi = (A,B)$,
  - how do we choose a corresponding state sequence $Q=(q_1 q_2 \ldots q_T)$ that is optimal in some sense (i.e., best explains the observations)

- One possibility:
  - For each hidden state sequence Q compute P(O|Q)
  - Pick the highest one

- Why not? N^T

Thanks to Dan Jurafsky for these slides

+ Instead: The Viterbi algorithm

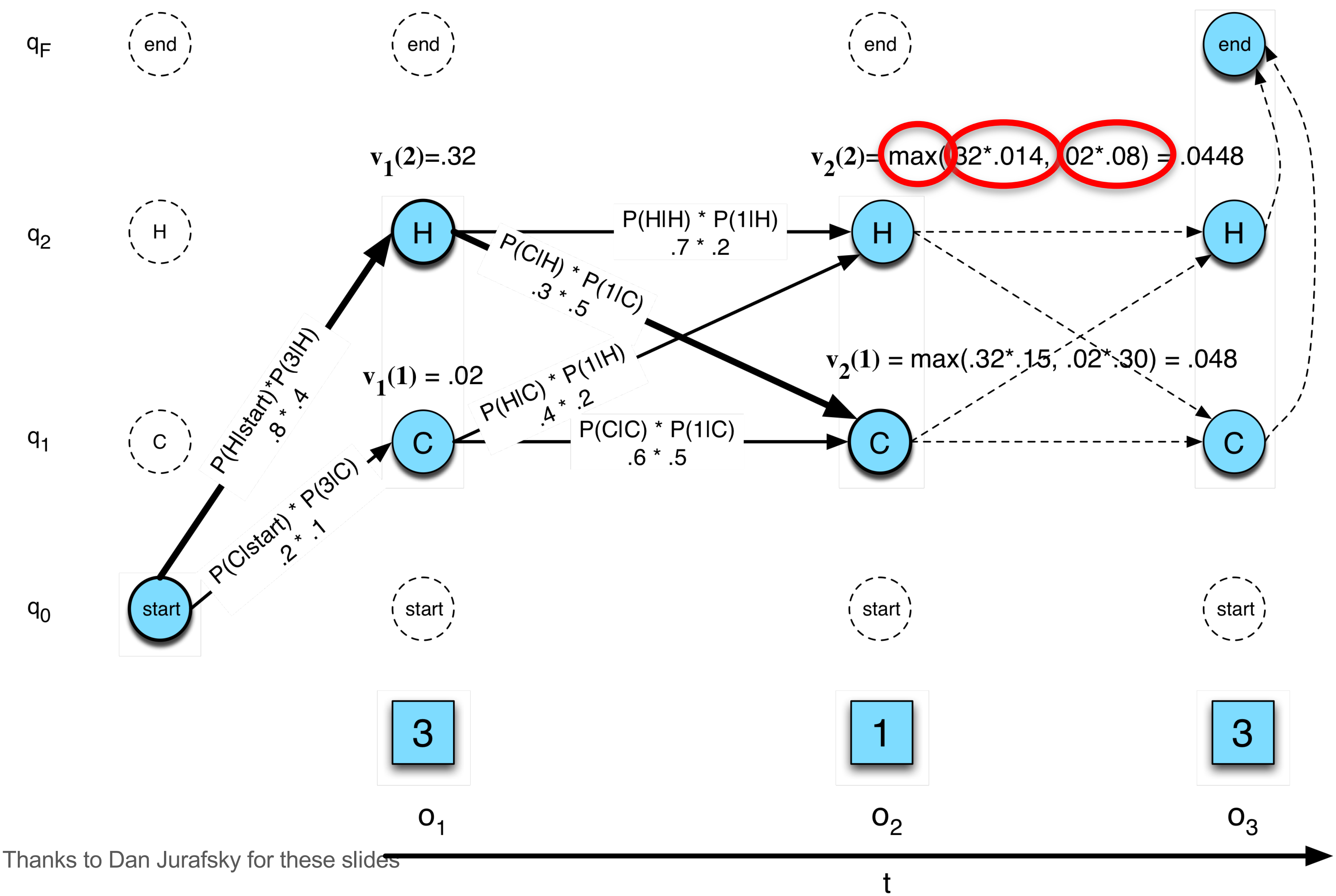
- Dynamic programming algorithm similar to the Forward algorithm
- Viterbi intuition
  - We want to compute the joint probability of the observation sequence together with the best state sequence

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

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

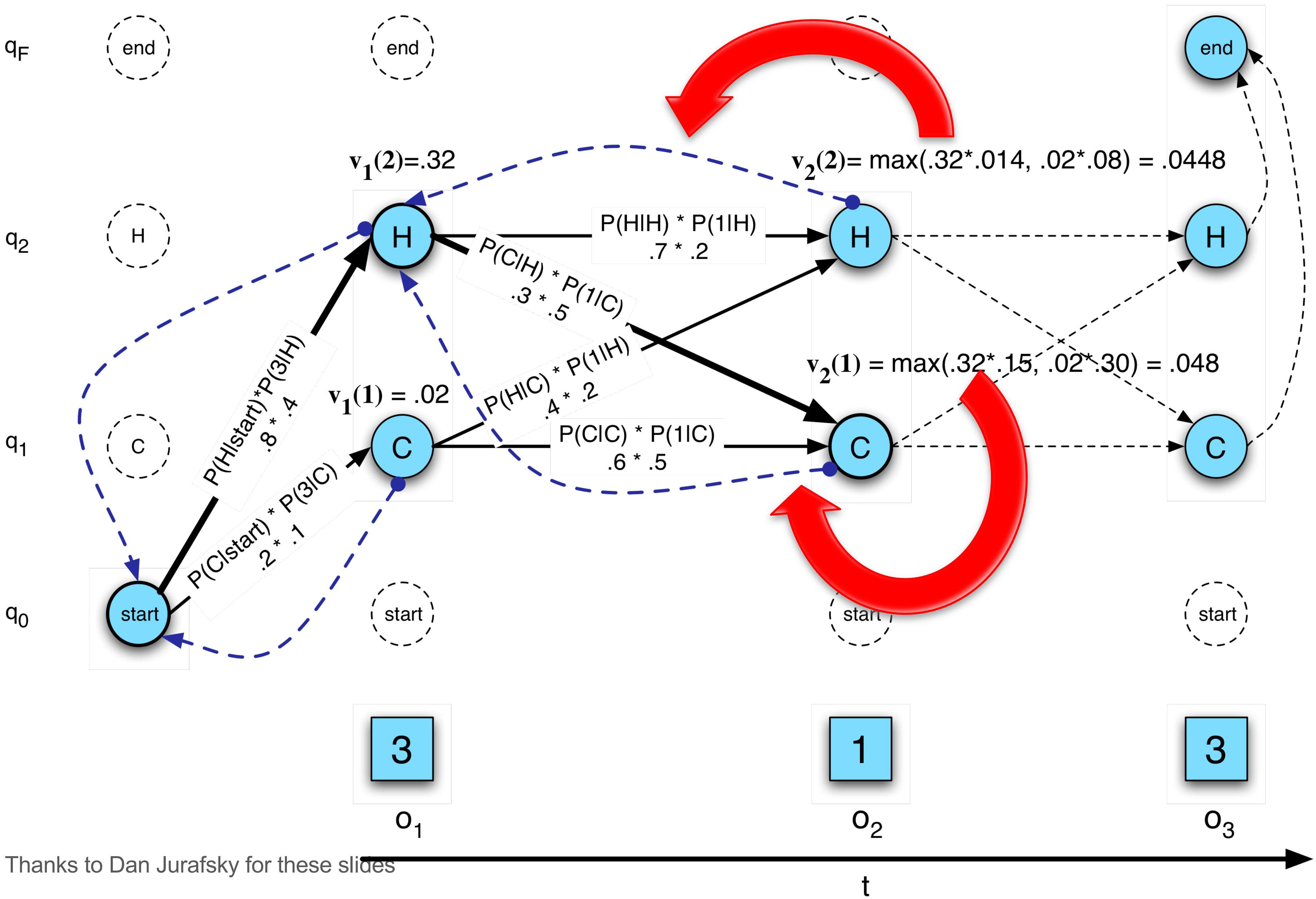
Thanks to Dan Jurafsky for these slides

+ The Viterbi trellis



Thanks to Dan Jurafsky for these slides

+ Viterbi backtrace



Thanks to Dan Jurafsky for these slides

+ HMMs for Speech

- But let's return to think about speech

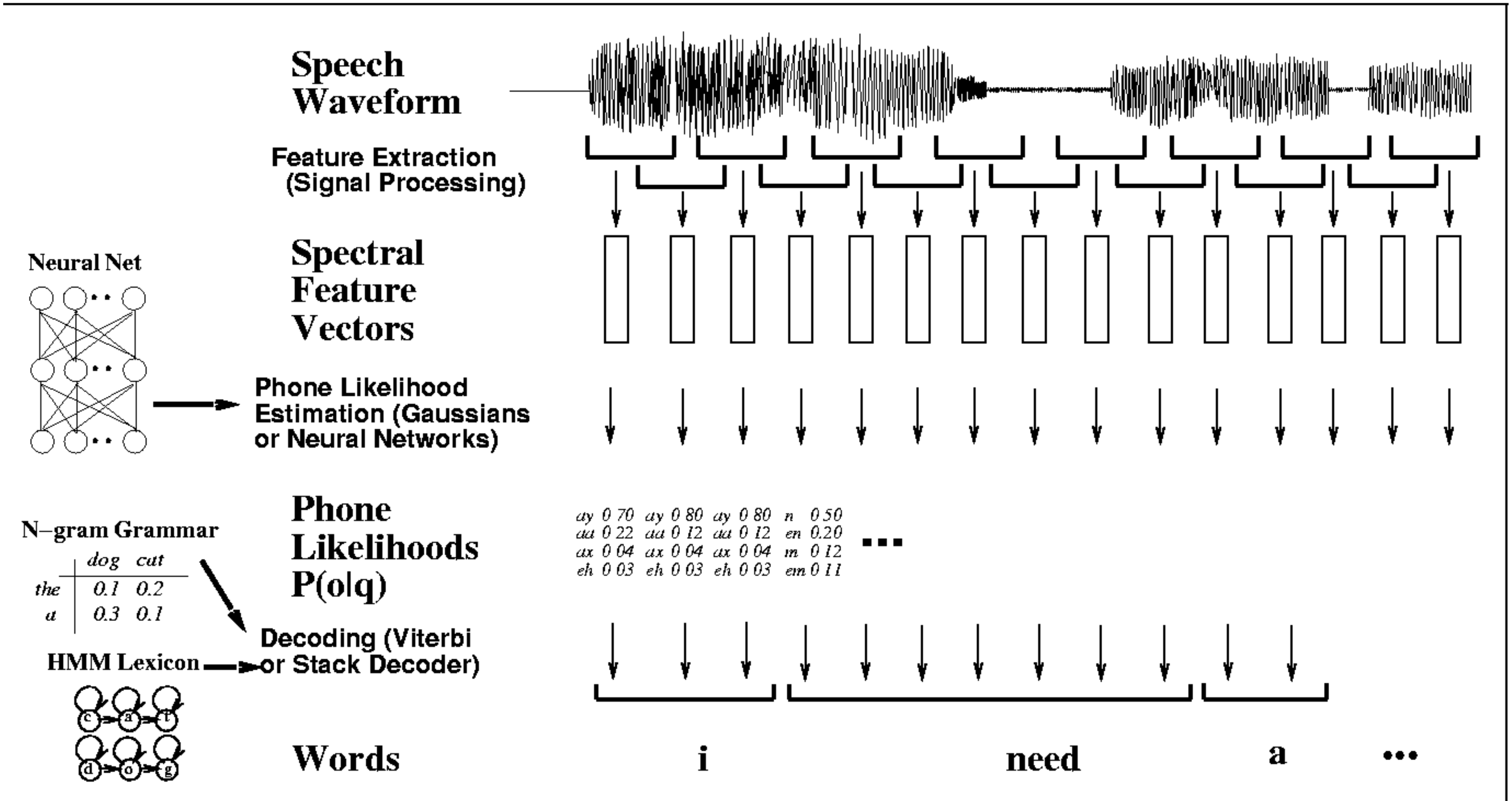
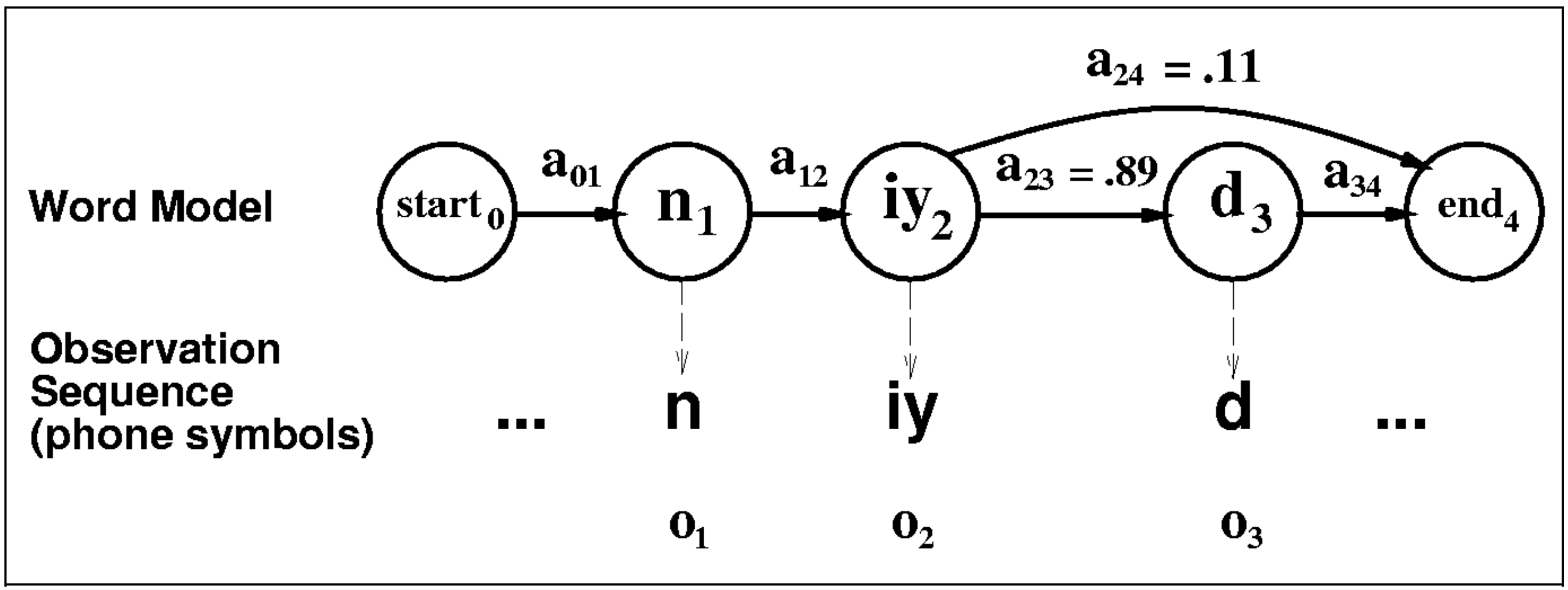


Figure 7.2 Schematic architecture for a (simplified) speech recognizer

Thanks to Dan Jurafsky for these slides

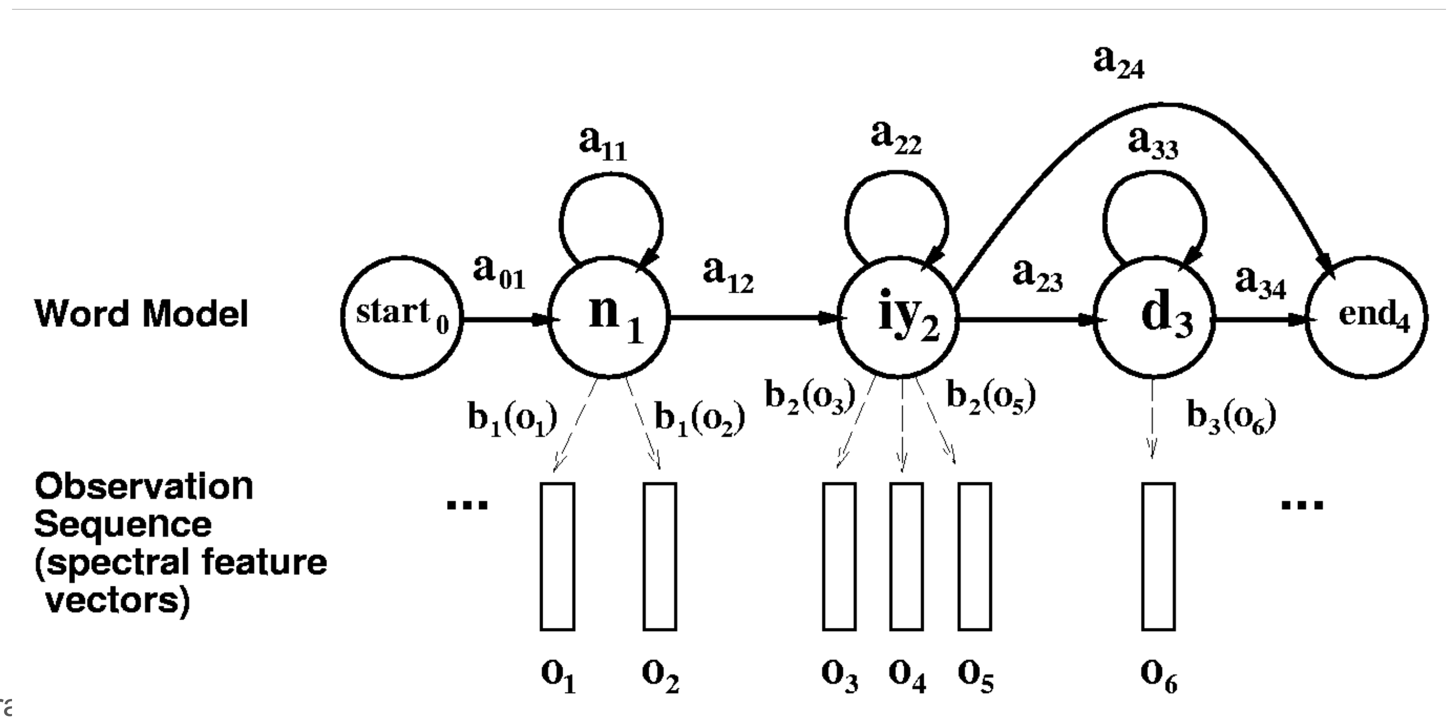
+ Word model

- A simple weighted automaton or Markov chain pronunciation network for the word *need,*
  - showing the transition probabilities, and a sample observation sequence.
  - The transition probabilities *a x y* between two states *x* and *y* are 1.0 unless otherwise specified.



BUT Observations are vectors not phonemes

- An HMM pronunciation network for the word *need,*
  - showing the transition probabilities, and a sample observation sequence.
- Note the addition of the output probabilities *B.*
  - Self-loops on the states to model variable phone durations.

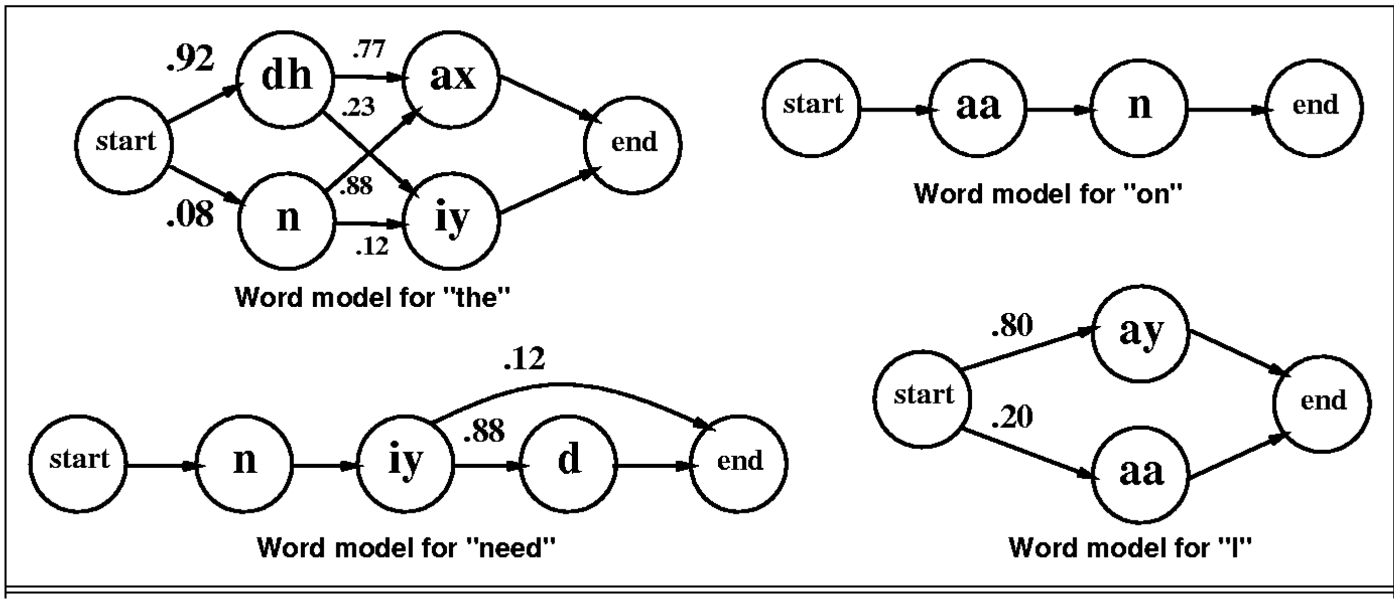


Thanks to Dan Jura

+ Example

Input	Output
[aa n iy dh ax]	*I need the*

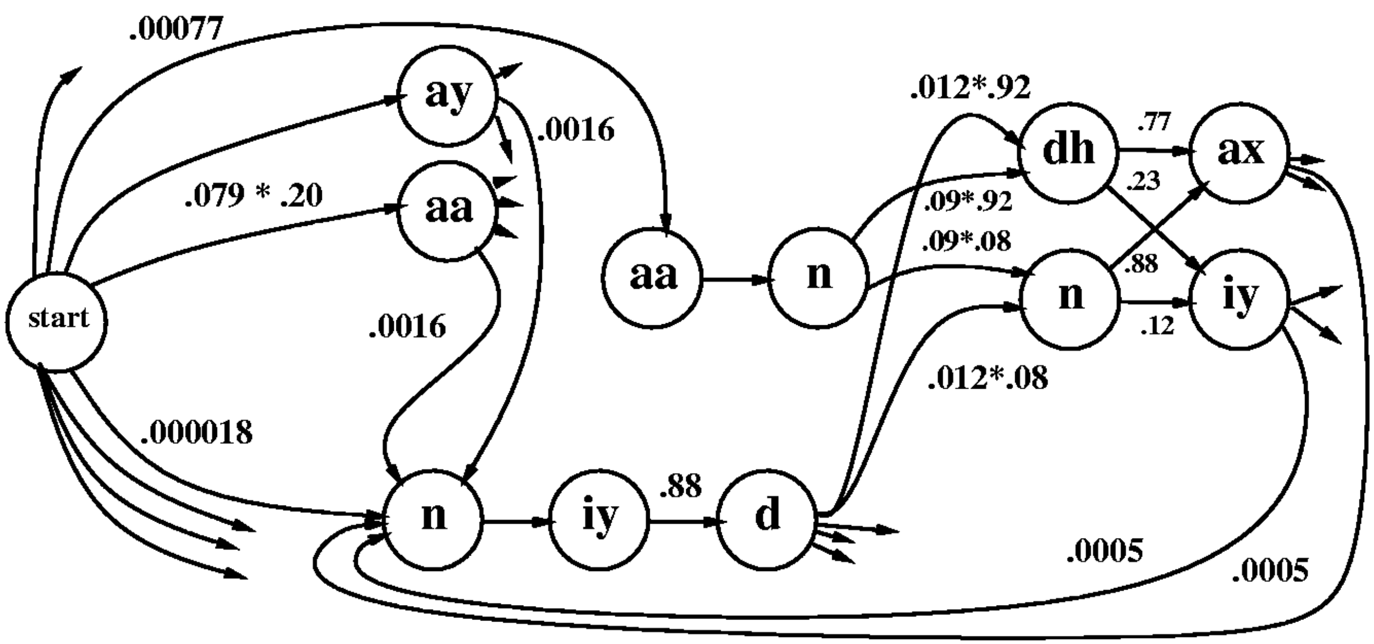
- Pronunciation networks for the words *I, on, need,* and *the.* All networks (especially *the)* are significantly simplified.



+ The transition probabilities

- Single automaton made from the words *I, need, on,* and *the.*
  - The arcs between words have probabilities computed from the bigrams in the table below

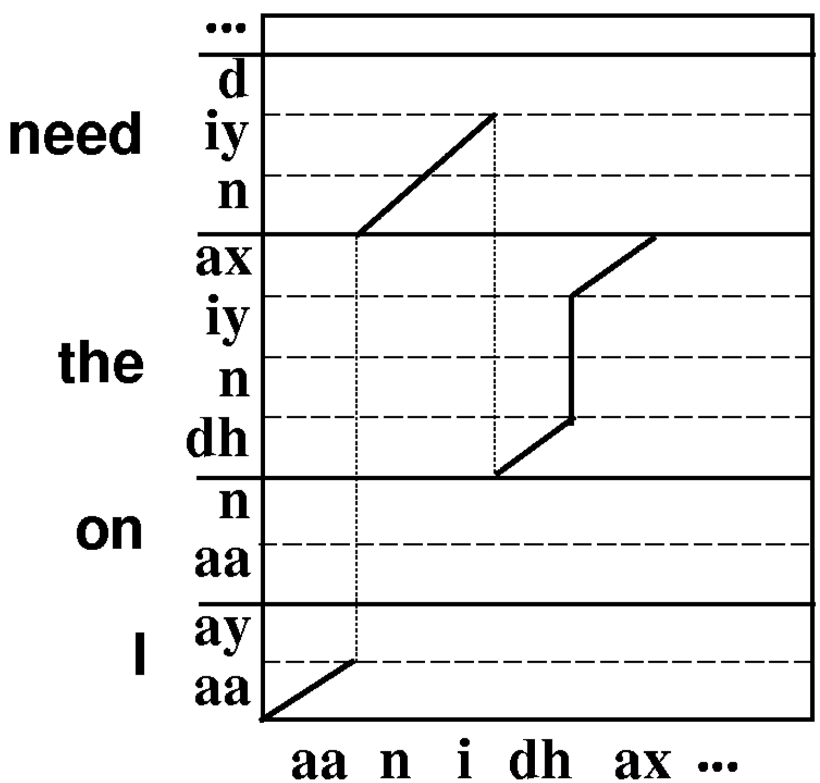
I need	0.0016	need need	0.000047	# Need	0.000018
I the	0.00018	need the	0.012	# The	0.016
I on	0.000047	need on	0.000047	# On	0.00077
I I	0.039	need I	0.000016	# I	0.079



Thanks to Dan

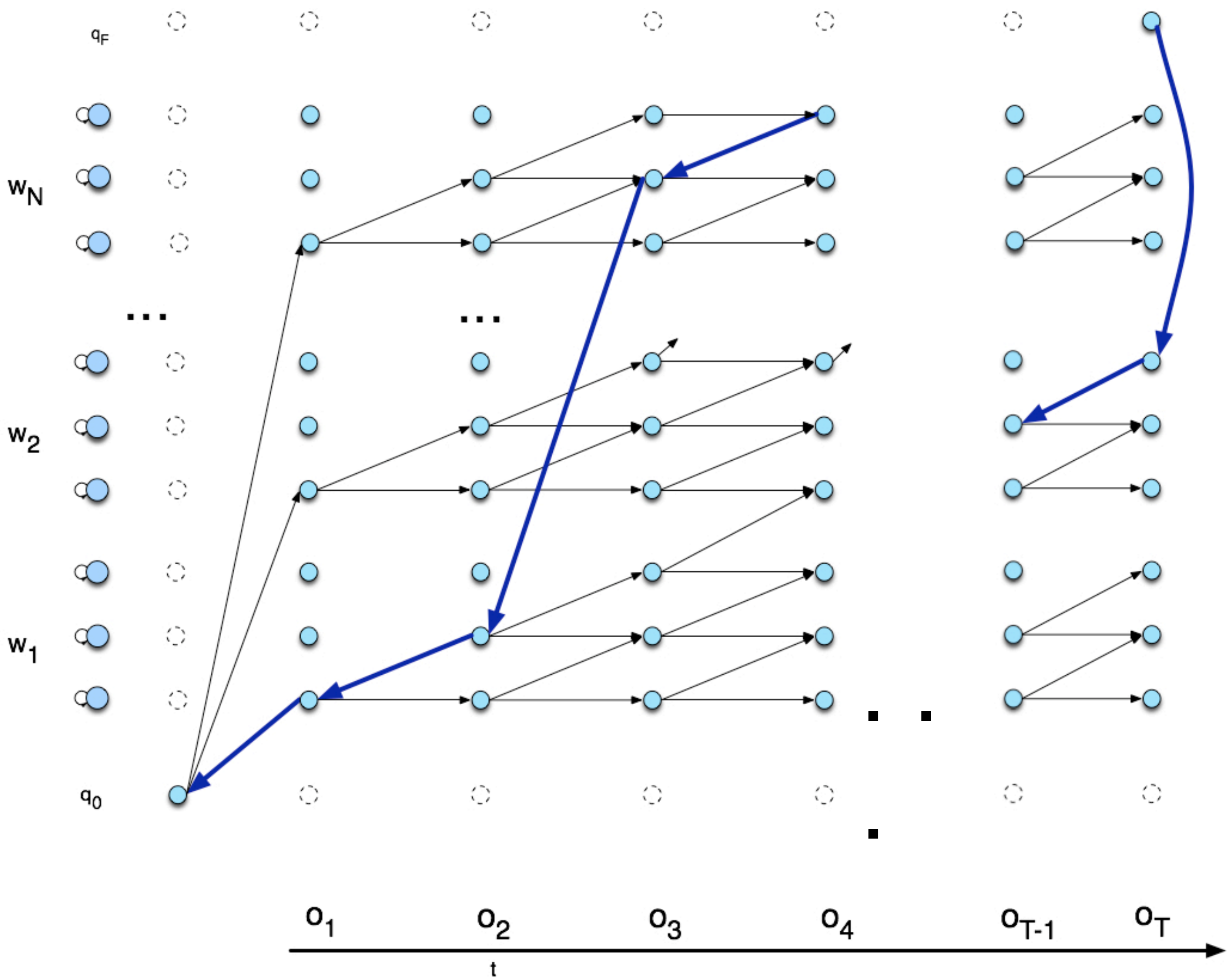
+ Viterbi on speech

- An illustration of the results of the Viterbi algorithm used to find the most-likely phone sequence (and hence estimate the most-likely word sequence).



Thanks to Dan Jurafsky for these slides

+ Viterbi backtrace



Thanks to Dan Jurafsky for these slides

+ Summary

- The Forward Algorithm
  - Dynamic programming
    - Keeps partial results
      - The likelihood of being in state t given the observations (t-1) and the model
    - Sums over all possible paths to that state
  - Finds the probability of the observation sequence

- The Viterbi Algorithm
  - Dynamic programming
    - Keeps partial results
      - The **max** of the probabilities of the paths to that state
    - Keeping a backpointer to which state was the max lets you trace back
  - Finds the “best” path
    - Note that hill climbing means this is not a guaranteed result

+ The Three Basic Problems for HMMs

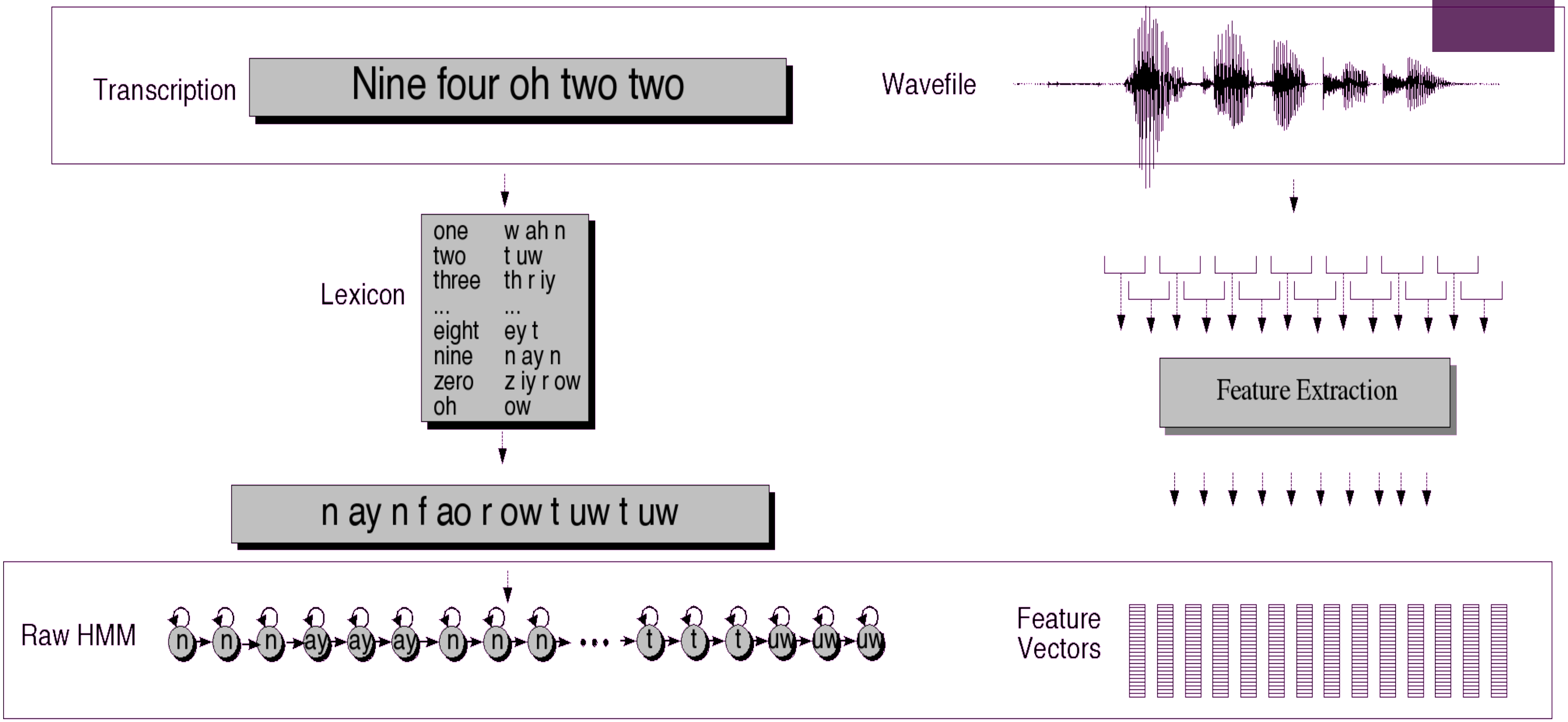
Jack Ferguson at IDA in the 1960s

- Problem 1 (**Evaluation**): Given the observation sequence $O=(o_1 o_2 \ldots o_T)$, 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 $O=(o_1 o_2 \ldots o_T)$, and an HMM model $\Phi = (A,B)$, how do we choose a corresponding state sequence $Q=(q_1 q_2 \ldots q_T)$ 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 $P(O|\Phi)$?

+ Embedded Training



+ The Learning Problem

- **Learning:** Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B
- **Baum-Welch = Forward-Backward Algorithm** (Baum 1972)
  - Is a special case of the EM or Expectation-Maximization algorithm (Dempster, Laird, Rubin)
- The algorithm will let us train
  - the transition probabilities $A = \{a_{ij}\}$ and
  - the emission probabilities $B=\{b_i(o_t)\}$ of the HMM

+ Intuition of HMMs

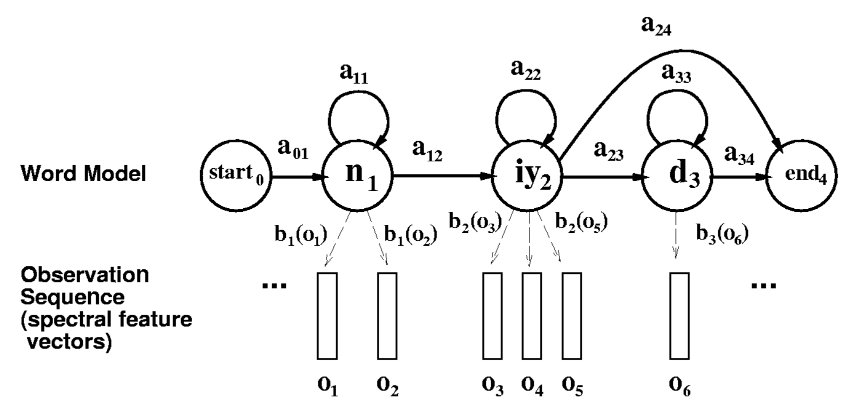
- For Markov chain, the observations are given, so there are no observation probability

- For HMM, cannot compute these counts directly from observed sequences

- Baum-Welch intuitions:
  - Iteratively estimate the counts.
    - Start with an estimate for a_{ij} and b_k, iteratively improve the estimates
  - Get estimated probabilities by:
    - computing the forward probability for an observation
    - dividing that probability mass among all the different paths that contributed to this forward probability

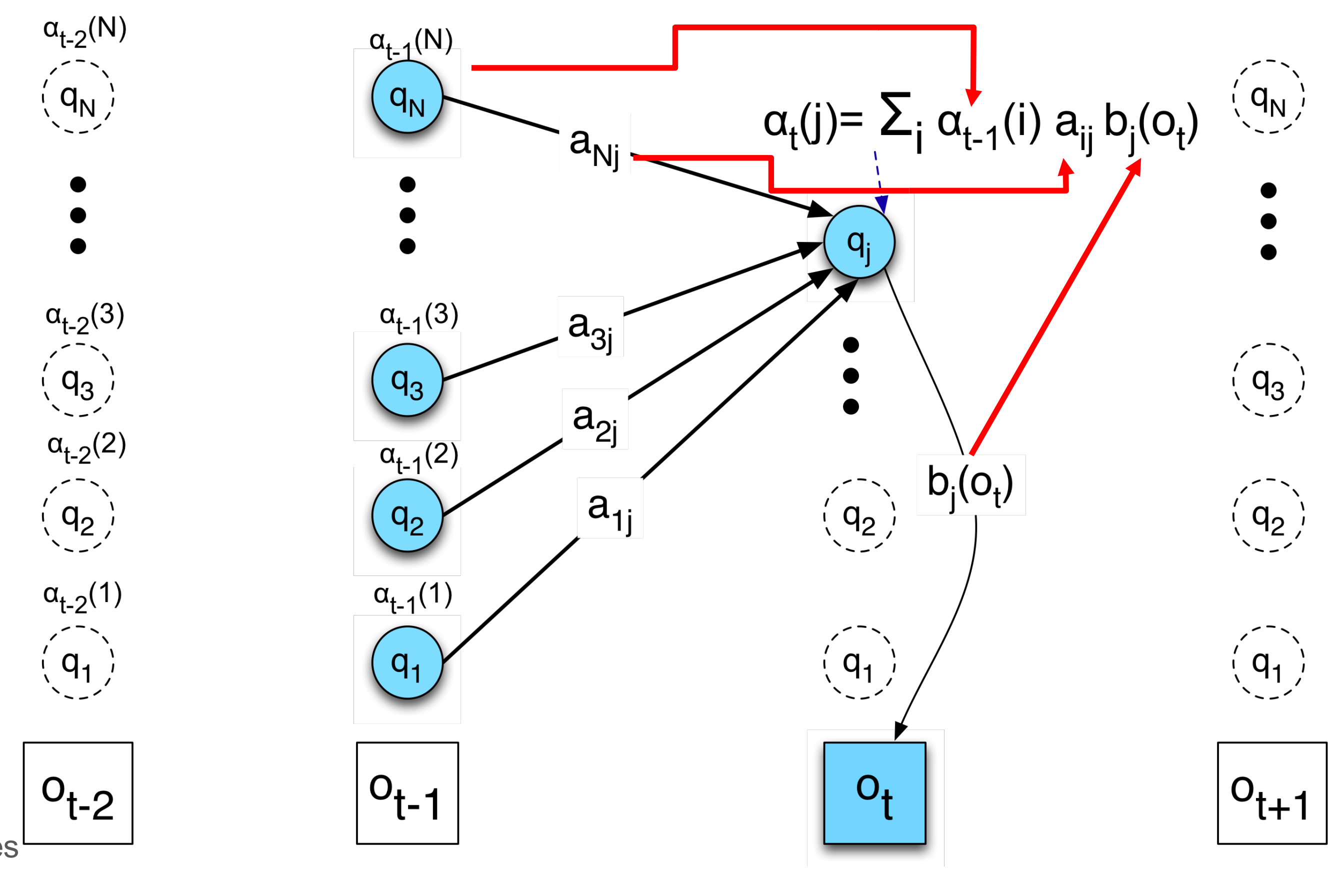
+ Embedded Training

- Given: phoneset, pronunciation lexicon, transcribed wavefiles
  - Build a whole sentence HMM for each sentence
  - Initialize A (transition) probs to 0.5, or to zero
  - Initialize B (observation) probs to global mean and variance
  - Run multiple iterations of Baum Welch
    - During each iteration, we compute forward and backward probabilities
  - Use them to re-estimate A and B
  - Run Baum-Welch til converge



+ We update each cell

- 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 (model) λ



Thanks to Dan Jurafsky for these slides

+ The Backward algorithm

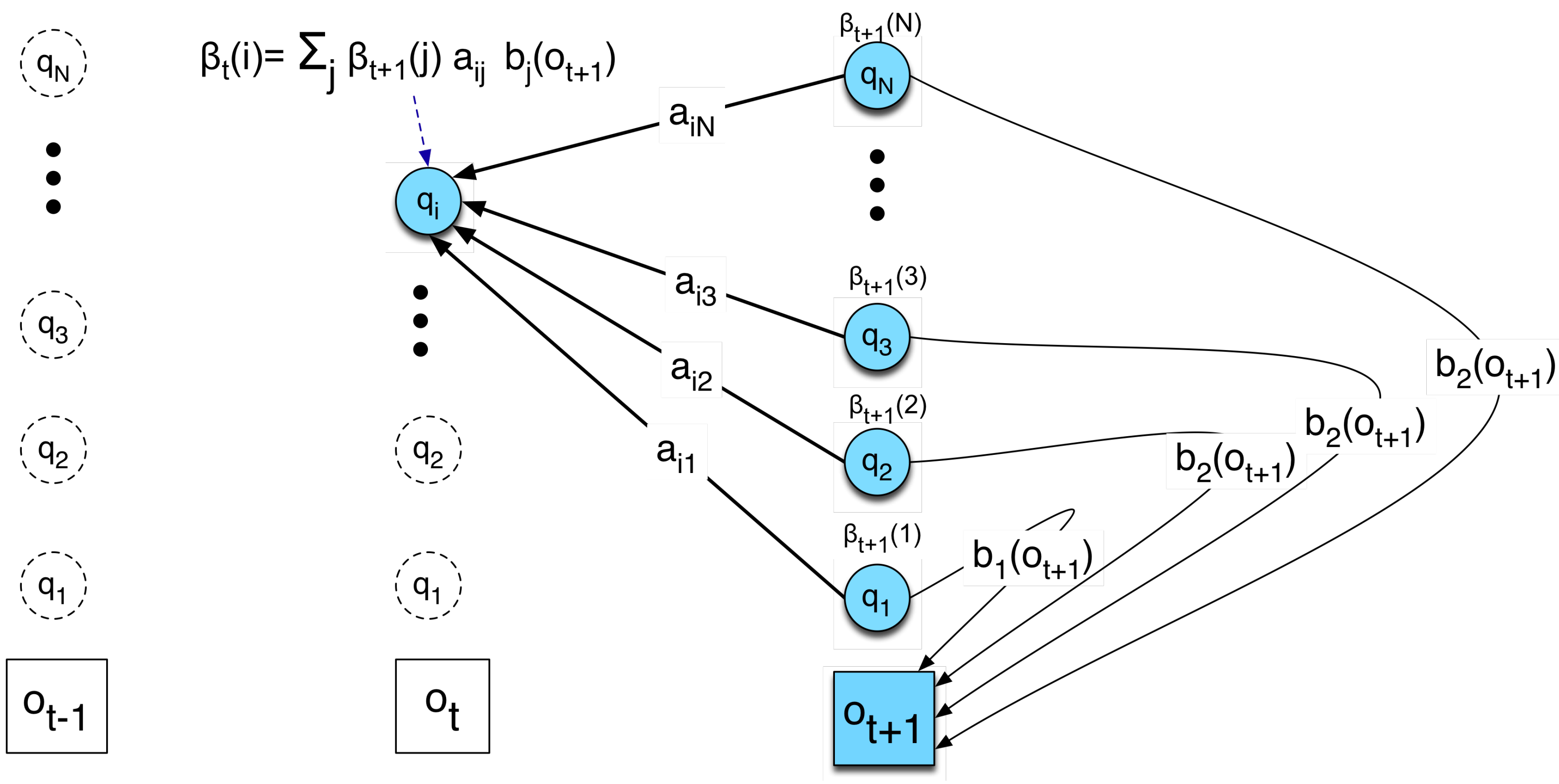
- We define the backward probability as follows:

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

- This is the probability of generating partial observations O_{t+1}^T from time *t+1* to the end, given that the HMM is in state *i* at time t and (of course) given Φ.

+ Inductive step of the backward algorithm

- This is the probability of generating partial observations Ot+1T from time t+1 to the end, given that the HMM is in state i at time t and (of course) given Φ.
- Computation of βt(i) by weighted sum of all successive values βt+1



+ Intuition for re-estimation of a_{ij}

- We will estimate $\hat{a}_{ij}$ via this intuition:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

- Numerator intuition:
  - Assume we had some estimate of probability that a given transition **i ➔j** was taken at time t in observation sequence.
  - If we knew this probability for each time *t,* we could sum over all *t* to get expected value (count) for i➔j.

+ Viterbi training

- Baum-Welch training says:
  - We need to know what state we were in, to accumulate counts of a given output symbol o_t
  - We'll compute $\xi i(t)$, the probability of being in state i at time t, by using forward-backward to sum over all possible paths that might have been in state i and output o_t.
- Viterbi training says:
  - Instead of summing over all possible paths, just take the single most likely path
  - Use the Viterbi algorithm to compute this "Viterbi" path
  - Via "forced alignment"
- Result
  - Much faster than Baum-Welch
  - But doesn't work quite as well
  - But the tradeoff is often worth it.

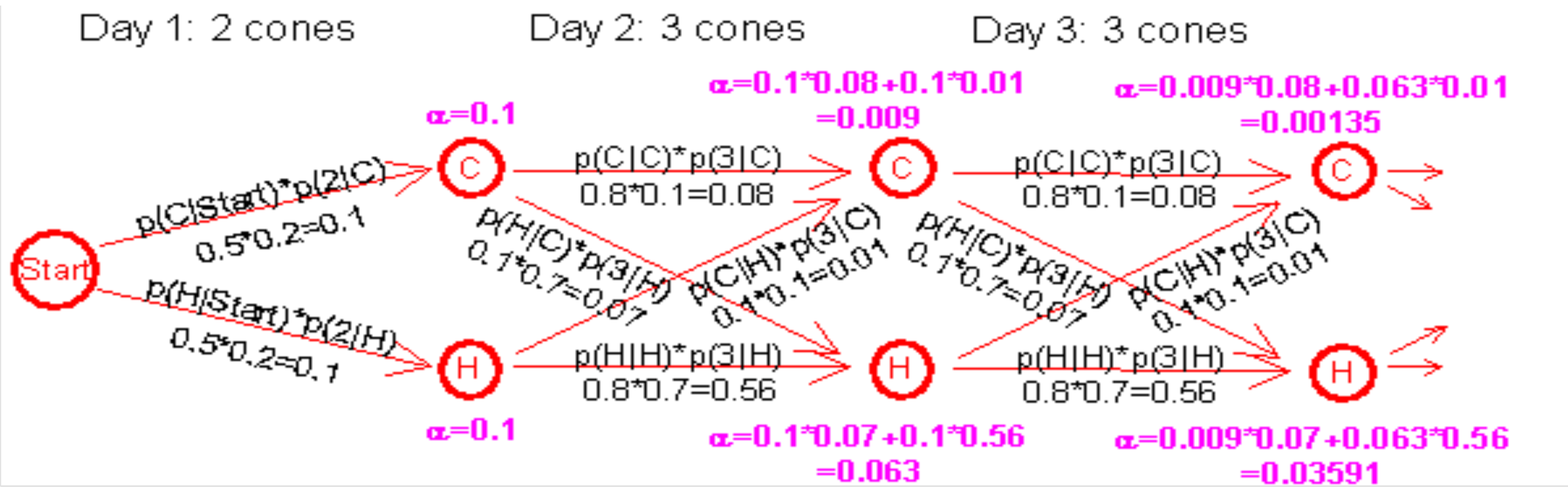
+ Input to Baum-Welch

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

- For ice-cream task
  - O = {1,3,2.,,,,.}
  - Q = {H,C}

Forward Algorithm (α)

- The dynamic programming computation of α
  - Backward (β) is similar but works back from Stop.



Thanks to Dan Jurafsky for these slides

+ Eisner Spreadheet

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.7	0.1	
p(2|…)	0.2	0.2	
p(3|…)	0.1	0.7	
p(C|…)	0.8	0.1	0.5
p(H|…)	0.1	0.8	0.5
p(STOP|…)	0.1	0.1	0

Day	Ice Creams	α(C)	α(H)	β(C)	β(H)
1	2	0.1	0.1	1.17798E-18	7.94958E-18
2	3	0.009	0.063	2.34015E-18	1.41539E-17
3	3	0.00135	0.03591	7.24963E-18	2.51453E-17
. . .	. . .	. . .	. . .	. . .	. . .
32	2	7.39756E-18	4.33111E-17	0.018	0.018
33	2	2.04983E-18	7.07773E-18	0.1	0.1

Tha

=C$14*E59*INDEX(C$11:C$13,$B59,1)+C$15*F59*INDEX(D$11:D$13,$B59,1)

+ Eisner Spreadheet

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.7	0.1	
p(2|…)	0.2	0.2	
p(3|…)	0.1	0.7	
p(C|…)	0.8	0.1	0.5
p(H|…)	0.1	0.8	0.5
			0

E$14*INDEX(C$11:C$13,$B27,1)
p(C|Start) * p(2|C)

Day	Ice Creams	α(C)	α(H)	β(C)	β(H)
1	2	0.1	0.1	1.17798E-18	7.94958E-18
2	3	0.009	0.063	2.34015E-18	1.41539E-17
3	3	0.00135	0.03591	7.24963E-18	2.51453E-17
32	2	7.39756E-18	4.33111E-17	0.018	0.018
33	2	2.04983E-18	7.07773E-18	0.1	0.1

=C$14*E59*INDEX(C$11:C$13,$B59,1)+C$15*F59*INDEX(D$11:D$13,$B59,1)

+ Eisner Spreadheet

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.7	0.1	
p(2|…)	0.2	0.2	
p(3|…)	0.1	0.7	
p(C|…)	0.8	0.1	0.5
p(H|…)	0.1	0.8	0.5
p(STOP|…)	0.1	0.1	0

P(H|Start) * H(2|C)

Day	Ice Creams	α(C)	α(H)	β(C)	β(H)
1	2	0.1	0.1	1.17798E-18	7.94958E-18
2	3	0.009	0.063	2.34015E-18	1.41539E-17
3	3	0.00135	0.03591	7.24963E-18	2.51453E-17
…	…	…	…	…	…
32	2	7.39756E-18	4.33111E-17	0.018	0.018
33	2	2.04983E-18	7.07773E-18	0.1	0.1

Tha

+ Eisner Spreadheet

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.7	0.1	
p(2|…)	0.2	0.2	
p(3|…)	0.1	0.7	
p(C|…)	0.8	0.1	0.5
p(H|…)	0.1	0.8	0.5
p(STOP|…)	[illegible]	[illegible]	0

1A(C)*P(C|C)+ 1α (H)*P(C|H) * p(3|C

Day	Ice Creams	α(C)	α(H)	β(C)	β(H)
1	2	0.1	0.1	1.17798E-18	7.94958E-18
2	3	0.009	0.063	2.34015E-18	1.41539E-17
3	3	0.00135	0.03591	7.24963E-18	2.51453E-17
…	…	…	…	…	…
32	2	7.39756E-18	4.33111E-17	0.018	0.018
33	2	2.04983E-18	7.07773E-18	0.1	0.1

Tha

=C$14*E59*INDEX(C$11:C$13,$B59,1)+C$15*F59*INDEX(D$11:D$13,$B59,1)

+ Eisner Spreadheet

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.7	0.1	
p(2|…)	0.2\	0.2	
p(3|…)	0.1	0.7	
p(C|…)	0.8	0.1	0.5
p(H|…)	0.1	0.8	0.5
p(STOP|…)	0.1	0.1	0

Day	Ice Creams	α(C)	α(H)	β(C)	β(H)
1	2	0.1	0.1	1.17798E-18	7.94958E-18
2	3	0.009	0.063	2.34015E-18	1.41539E-17
3	3	0.00135	0.03591	7.24963E-18	2.51453E-17
…	…	…	…	…	…
32	2	7.39756E-18	4.33111E-17	0.018	0.018
33	2	2.0498	E-18	0.1	0.1

p(C|C)*33B(C)*p(2|C) + p(H|C)* 33B(H) * p(2|H)

p(STOP|C)

p(STOP|H)

Tha

+ Working through Eisner

All the paths through C up to time t

All the paths through H up to time t

Total prob of the data

Prob of getting to C/H at time t given all the data

α(C)*β (C	α(H)*β (H)	α(C)*β (C) +α(H)*β(H)	p(□C)	p(□H)
1.17798E-19	7.94958E-19	9.12756E-19	0.129	0.871
2.10613E-20	8.91695E-19	9.12756E-19	0.023	0.977
9.78701E-21	9.02969E-19	9.12756E-19	0.011	0.989
. . .	. . .	. . .	. . .	. . .

SUM:

ICs	p(□C,1)	p(□C,2)	p(□C,3)	p(□H,1)	p(□H,2)	p(□H,3)
2	0	0.129	0	0	0.871	0
3	0	0	0.023	0	0	0.977
3	0	0	0.011	0	0	0.989
	. . .	. . .	. . .	. . .	. . .	. . .
SUM	**9.931**	**3.212**	**1.537**	**1.069**	**7.788**	**9.463**

Tha

+ Working through Eisner

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.454817043	0.21213604	0
p(2|…)	0.324427849	0.342217822	0
p(3|…)	0.220755108	0.445646139	0

p(□C)	p(□H)
0.129	0.871
0.023	0.977
0.011	0.989
. . .	. . .
3.212	**1.537**

SUM of Prob of going thru C when IC = 1

SUM of Prob of going thru C given all the data

SUM:

ICs	p(□C,1)	p(□C,2)	p(□C,3)	p(□H,1)	p(□H,2)	p(□H,3)
2	0	0.129	0	0	0.871	0
3	0	0	0.023	0	0	0.977
3	0	0	0.011	0	0	0.989
	. . .	. . .	. . .	. . .	. . .	. . .
SUM	**9.931**	**3.212**	**1.537**	**1.069**	**7.788**	**9.463**

Tha

+ Eisner results

- Start

	p(…|C)	p(…|H)
p(1|…)	0.7	0.1
p(2|…)	0.2	0.2
p(3|…)	0.1	0.7

- After 1 iteration

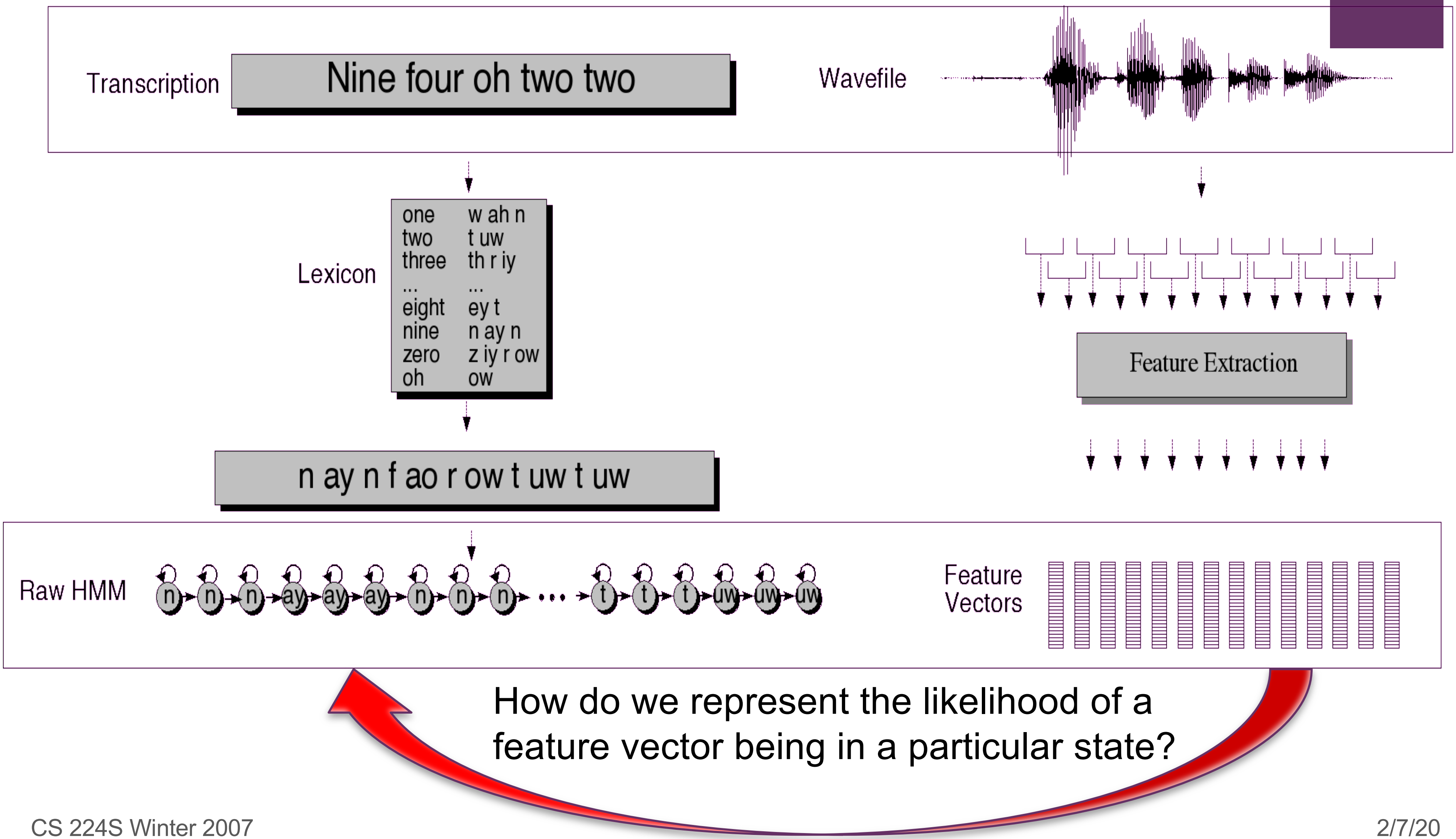
	p(…|C)	p(…|H)
p(1|…)	0.6643	0.0592
p(2|…)	0.2169	0.4297
p(3|…)	0.1187	0.5110

- After 10 iterations

	p(…|C)	p(…|H)
p(1|…)	0.6407	0.0001
p(2|…)	0.1481	0.5342
p(3|…)	0.2112	0.4657

Thanks to Dan Jurafsky for these slides

+ Back to Embedded Training



How do we represent the likelihood of a feature vector being in a particular state?