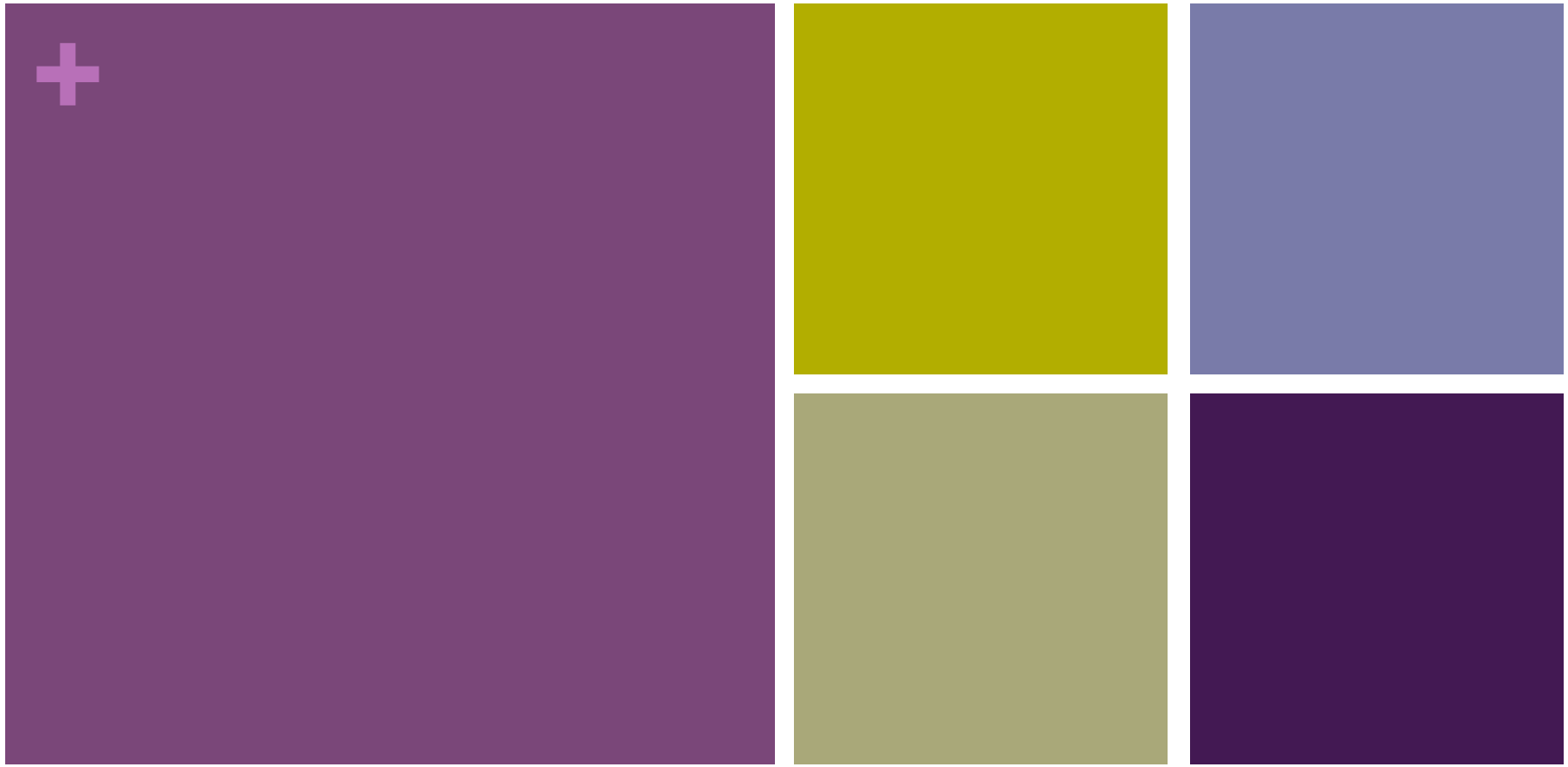
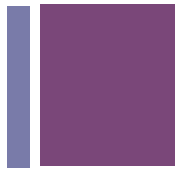


September13, 2016
Professor Meteer

CS 136a Lecture 7
Speech Recognition Architecture: Training models with the Forward backward algorithm

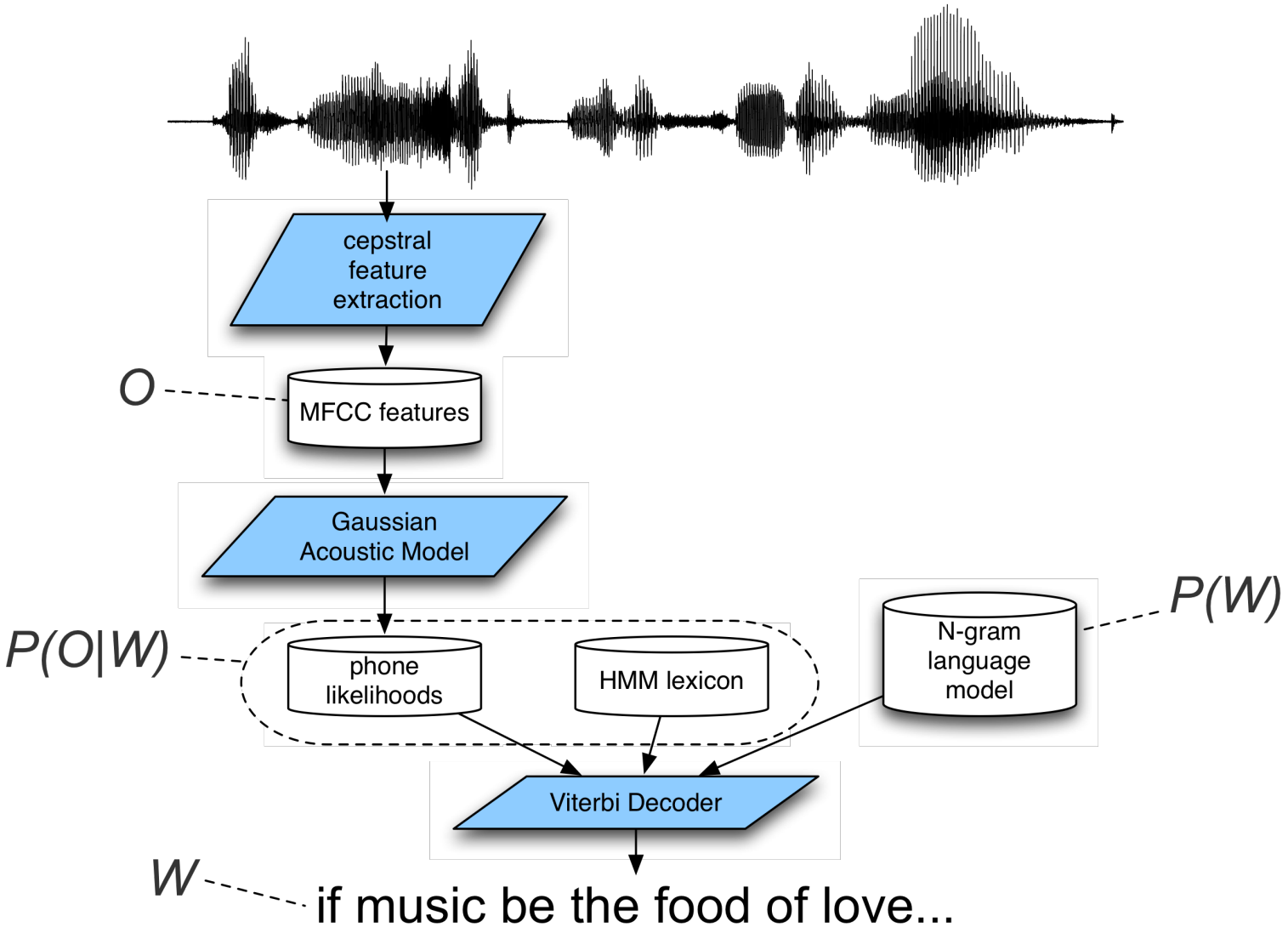
Thanks to Dan Jurafsky for these slides

ASR components



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

Speech Recognition Architecture



Thanks to Dan Jurafsky for these slides

The Three Basic Problems for HMMs

Jack Ferguson at IDA in the 1960s

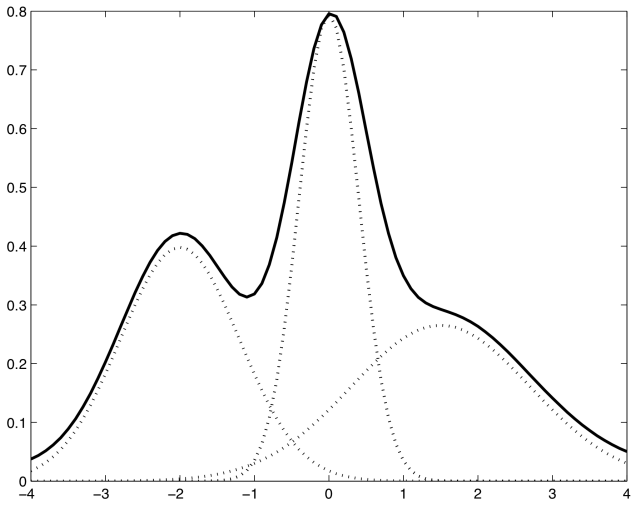
- Problem 1 (**Evaluation**): Given the observation sequence $O=(o_1 o_2 \ldots o_T)$, and an HMM model $\Phi = (A,B)$, how do we efficiently compute $P(O|\Phi)$, the probability of the observation sequence, given the model
- Problem 2 (**Decoding**): Given the observation sequence $O=(o_1 o_2 \ldots o_T)$, 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 $P(O|\Phi)$?

The Learning Problem

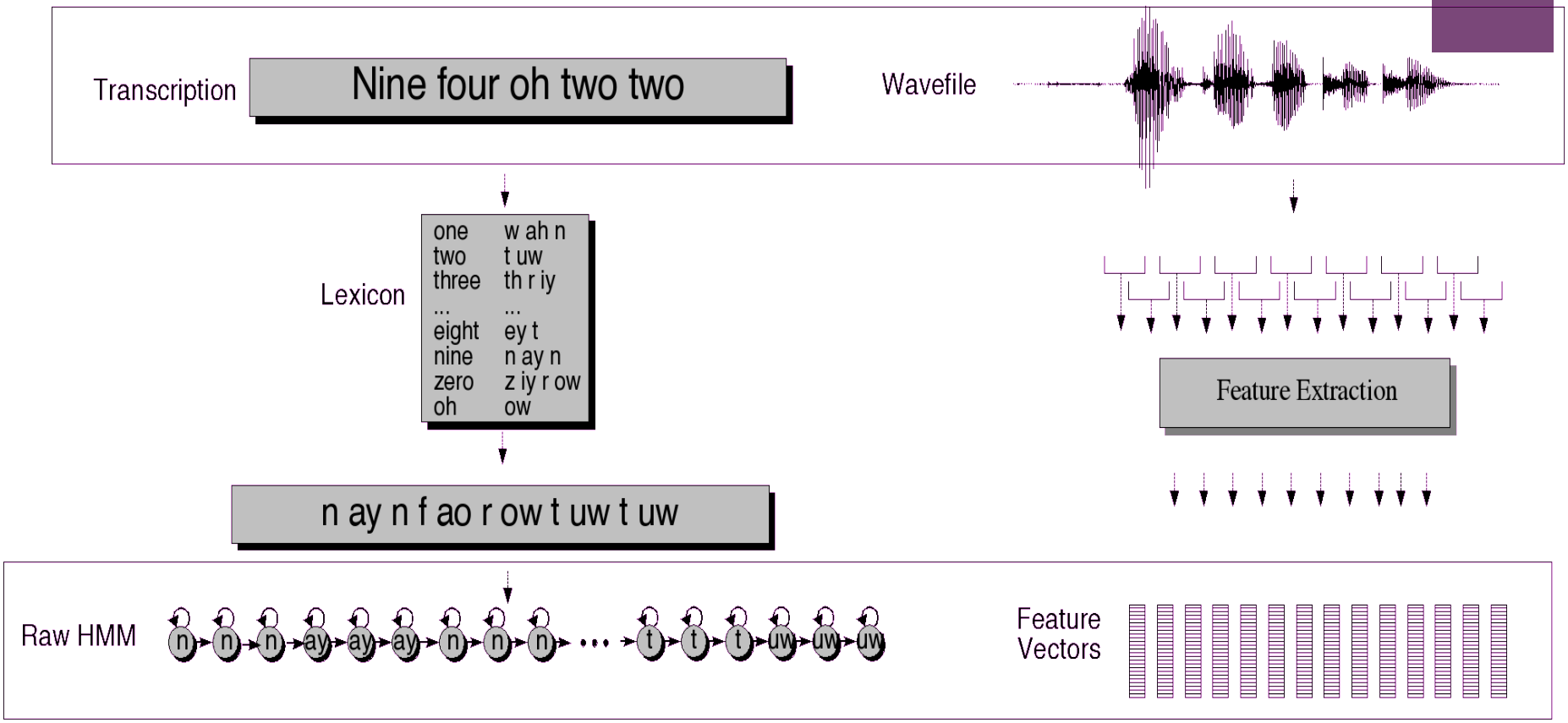
- **Learning: Given an observation sequence O and a the set of possible states in the HMM, learn the HMM parameters A and B**
- **Baum-Welch = Forward-Backward Algorithm** (Baum 1972)
  - Is a special case of the EM or Expectation-Maximization algorithm (Dempster, Laird, Rubin)
- The algorithm will let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

GMMs in Acoustic Modeling

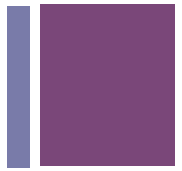
- Multivariant Gaussian Mixture Models
  - Gaussian: Determine the likelihood of a real value to be in a particular state
  - Multivariant: We have a vector of values, not just one
  - Mixture Models: Values may not be best modeled by a single Gaussian
- Learning the “parameters” (means and variances) using the backward forward algorithm



Embedded Training



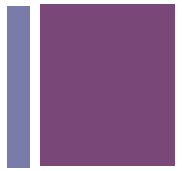
Starting out with Observable Markov Models



- How to train?
- Run the model on observation sequence O.
- Since it's not hidden, we know which states we went through, hence which transitions and observations were used.
- Given that information, training:
  - B = $\{b_k(o_t)\}$: Since every state can only generate one observation symbol, observation likelihoods B are all 1.0
  - A = $\{a_{ij}\}$:

$$a_{ij} = \frac{C(i \rightarrow j)}{\sum_{q \in Q} C(i \rightarrow q)}$$

Extending Intuition to HMMs

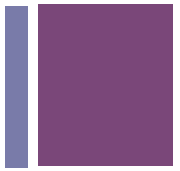


- For HMM, cannot compute these counts directly from observed sequences
- Baum-Welch intuitions:
  - Iteratively estimate the counts.
    - Start with an estimate for a_{ij} and b_k, iteratively improve the estimates
  - Get estimated probabilities by:
    - computing the forward probability for an observation
    - 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

The forward algorithm



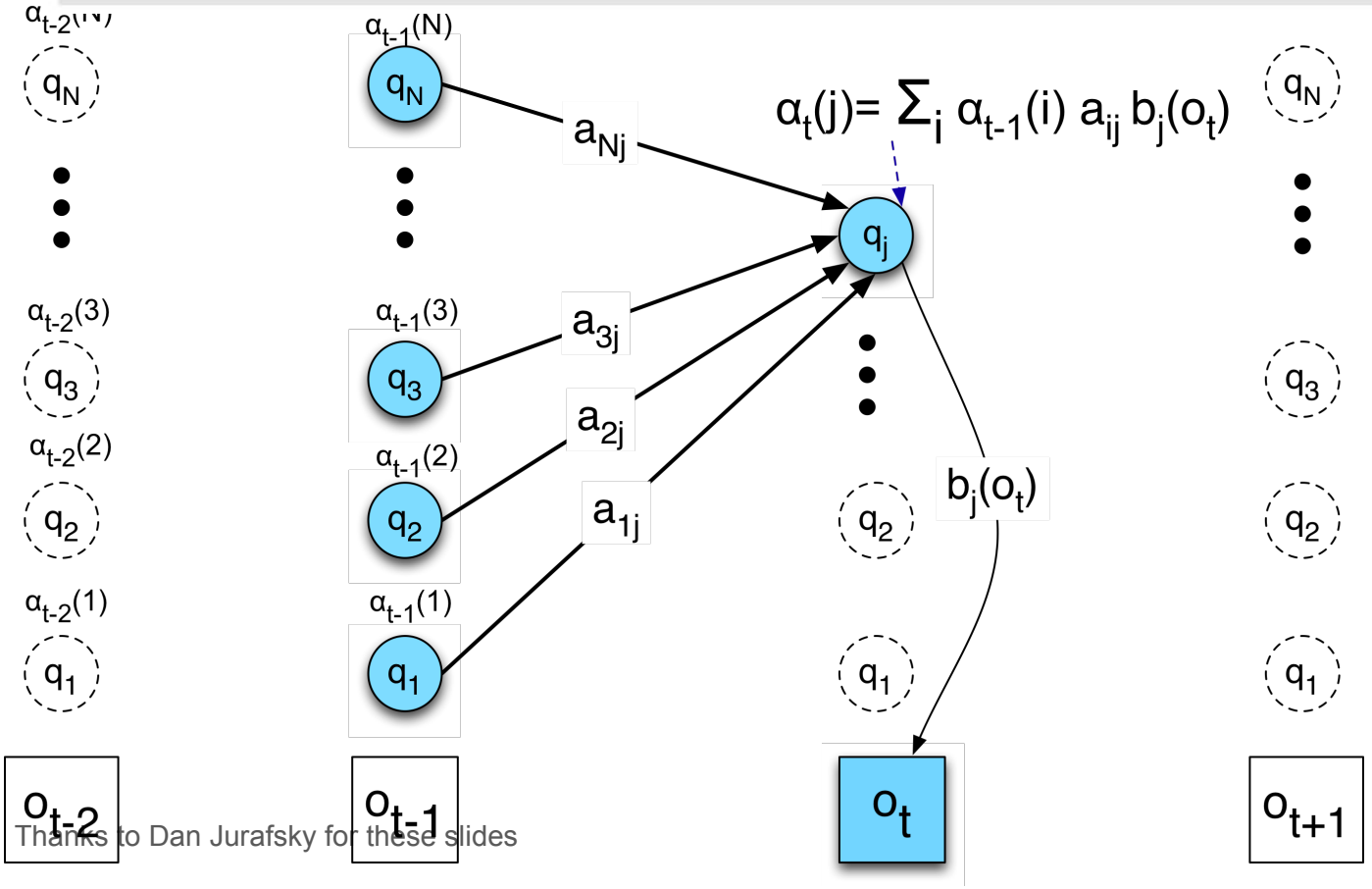
- Each cell of the forward algorithm trellis $alpha_t(j)$
  - Represents the probability of being in state j
  - After seeing the first t observations
  - Given the automaton (model) λ
- Each cell thus expresses the following probability

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

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

The Backward algorithm

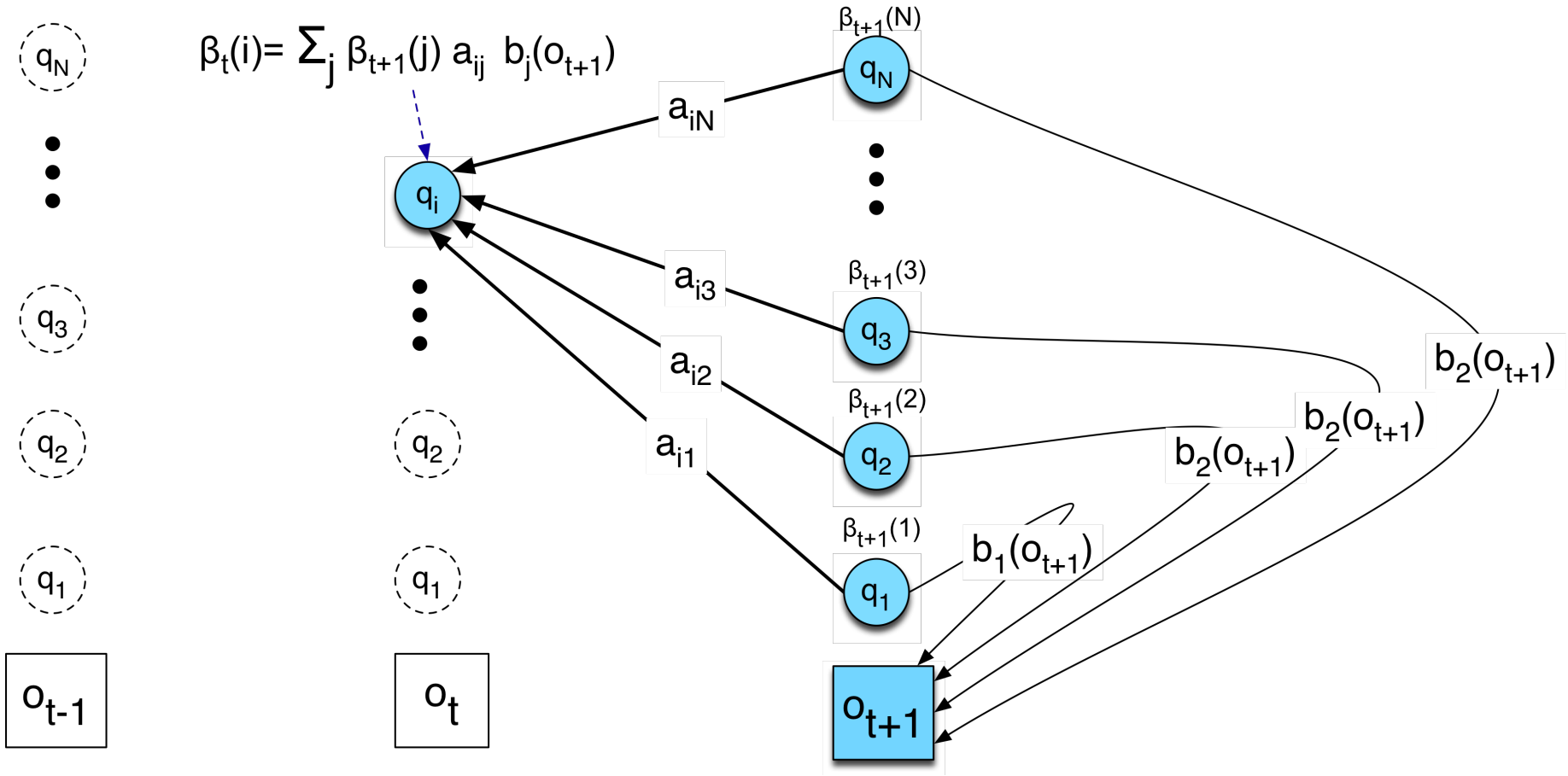
- We define the backward probability as follows:

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

- 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
(figure inspired by Rabiner and Juang)

Computation of $\beta_t(i)$ by weighted sum of all successive values β_{t+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"

Forced Alignment

- "Forced alignment": Computing the "Viterbi path" over the training data
- Because we know which word string to assign to each observation sequence.
- We just don't know the state sequence.
- So we use a_{ij} to constrain the path to go through the correct words
- And otherwise do normal Viterbi
- Result: state sequence!

Summary

- Transition probabilities
  - The ratio between the expected number of transitions from state i to j and the expected number of all transitions from state i

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

- Observation probabilities
  - The ratio between the expected number of times the observation data emitted from state j is v_k, and the expected number of times any observation is emitted from state j

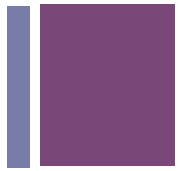
$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Thanks to Dan Jurafsky for these slides

Viterbi Training

- Much faster than Baum-Welch
- But doesn't work quite as well
- But the tradeoff is often worth it.

Input to Baum-Welch

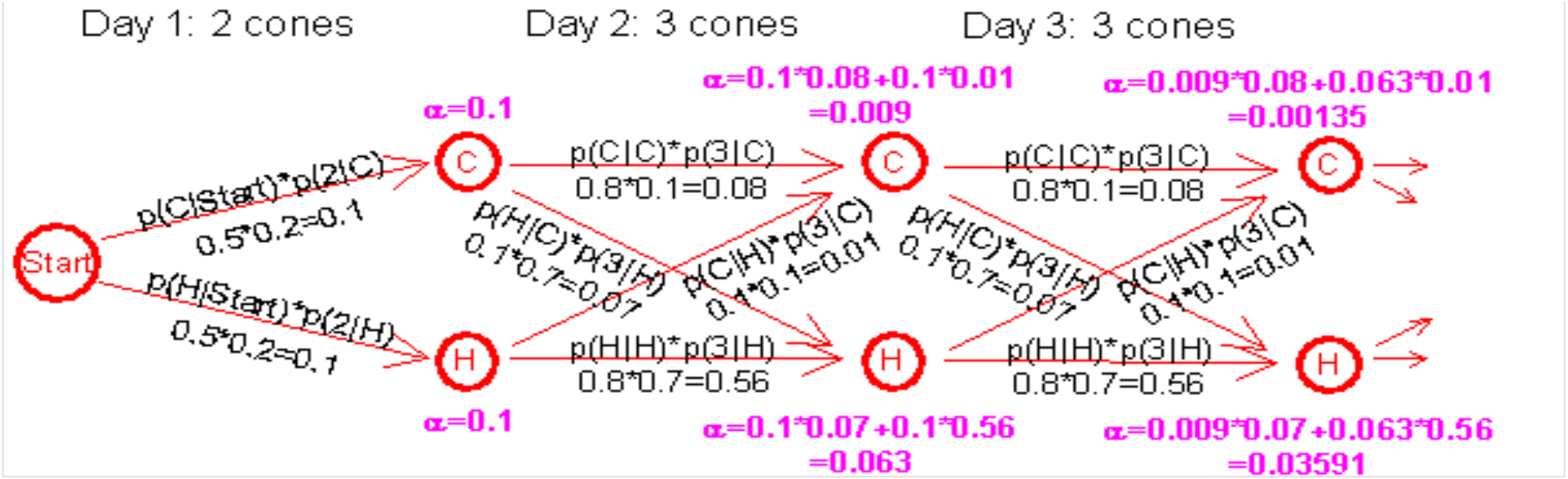


- O unlabeled sequence of observations
- Q vocabulary of hidden states

- 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

	$p_{(\ldots\|C)}$	$p_{(\ldots\|H)}$	p(…|START)
$p_{(1\|\ldots)}$	0.7	0.1	
$p_{(2\|\ldots)}$	0.2	0.2	
$p_{(3\|\ldots)}$	0.1	0.7	
$p_{(C\|\ldots)}$	0.8	0.1	0.5
$p_{(H\|\ldots)}$	0.1	0.8	0.5
$p_{(STOP\|\ldots)}$	0.1	0.1	0

Day	Ice Creams	$\alpha_{(C)}$	$\alpha_{(H)}$	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
			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

Thanks to Dan Jurafsky for these slides

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

+ Eisner Spreadheet

	$p_{(}\ldots\|C_{)}$	$p_{(}\ldots\|H_{)}$	p(…|START)
$p_{(}1\|\ldots_{)}$	0.7	0.1	
$p_{(}2\|\ldots_{)}$	0.2	0.2	
$p_{(}3\|\ldots_{)}$	0.1	0.7	
$p_{(}C\|\ldots_{)}$	0.8	0.1	0.5
$p_{(}H\|\ldots_{)}$	0.1	0.8	0.5
			0

$P_{(}H|Start_{)} * H_{(}2|C)$

Day	Ice Creams	$\alpha_{(}C_{)}$	$\alpha_{(}H_{)}$	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

Thanks to Dan Jurafsky for these slides

Eisner Spreadheet

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

$1A(C)*P(C|C)+1\alpha(H)*P(C|H)*p(3|C$

Day	Ice Creams	$\alpha(C)$	$\alpha(H)$	$\beta(C)$	$\beta(H)$
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

Thanks to Dan Jurafsky for these slides

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

Eisner Spreadheet

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

Day	Ice Creams	$\alpha(C)$	$\alpha(H)$	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

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

$p(STOP|C)$

$p(STOP|H)$

Thanks to Dan Jurafsky for these slides

+ 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

$\alpha_{(C)}*\beta_{(C)}$	$\alpha_{(H)}*\beta_{(H)}$	$\alpha_{(C)}*\beta_{(C)} + \alpha_{(H)}*\beta_{(H)}$	$p_{(\rightarrow C)}$	$p_{(\rightarrow H)}$
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: 14.679	18.321	9.931	3.212	1.537

ICs	$p_{(\rightarrow C,1)}$	$p_{(\rightarrow C,2)}$	$p_{(\rightarrow C,3)}$	$p_{(\rightarrow H,1)}$	$p_{(\rightarrow H,2)}$	$p_{(\rightarrow H,3)}$
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)
0.129	0.871
0.023	0.977
0.011	0.989
. . .	. . .
3.212	**1.537**

	p(…|C)	p(…|H)	p(…|START)
p(1|…)	0.454817043	0.21213604	0
p(2|…)	0.32442849	0.342217822	0
p(3|…)	0.220755108	0.445646139	0

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

	$p(\ldots\|C)$	$p(\ldots\|H)$
$p(1\|\ldots)$	0.7	0.1
$p(2\|\ldots)$	0.2	0.2
$p(3\|\ldots)$	0.1	0.7

- After 1 iteration

	$p(\ldots\|C)$	$p(\ldots\|H)$
$p(1\|\ldots)$	0.6643	0.0592
$p(2\|\ldots)$	0.2169	0.4297
$p(3\|\ldots)$	0.1187	0.5110

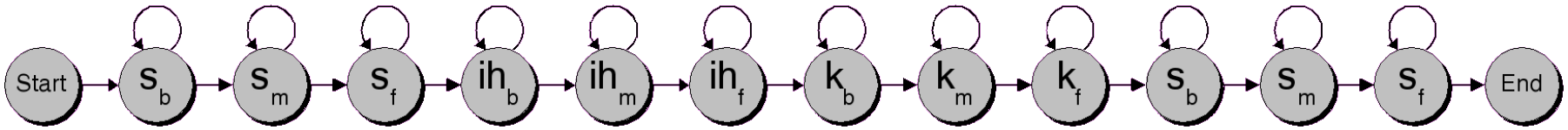
- After 10 iterations

	$p(\ldots\|C)$	$p(\ldots\|H)$
$p(1\|\ldots)$	0.6407	0.0001
$p(2\|\ldots)$	0.1481	0.5342
$p(3\|\ldots)$	0.2112	0.4657

Thanks to Dan Jurafsky for these slides

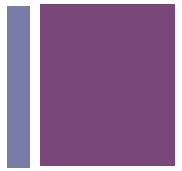
Applying FB to speech: Caveats

- Network structure of HMM is always created by hand
  - no algorithm for double-induction of optimal structure and probabilities has been able to beat simple hand-built structures.
  - Always Bakis network = links go forward in time
  - Subcase of Bakis net: beads-on-string net:



- Baum-Welch only guaranteed to return local max, rather than global optimum

Multivariate Gaussians



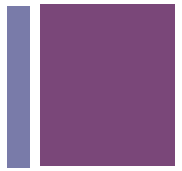
- Instead of a single mean μ and variance σ:

$$f(x \mid \mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- Vector of observations *x* modeled by vector of means μ and covariance matrix Σ

$$f(x \mid \mu,\Sigma) = \frac{1}{(2\pi)^{D/2} \mid \Sigma \mid^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

Training a GMM



- Problem: how do we train a GMM if we don't know what component is accounting for aspects of any particular observation?
- Intuition: we use Baum-Welch to find it for us, just as we did for finding hidden states that accounted for the observation

How to train mixtures?

- Choose M (often 16; or can tune M dependent on amount of training observations)
- Then can do various splitting or clustering algorithms
- One simple method for "splitting":
  1. Compute global mean μ and global variance
  2. Split into two Gaussians, with means $\mu \pm \varepsilon$ (sometimes ε is 0.2σ)
  3. Run Forward-Backward to retrain
  4. Go to 2 until we have 16 mixtures