EECS E6870: Lecture 4: Hidden Markov Models

Stanley F. Chen, Michael A. Picheny and Bhuvana Ramabhadran
IBM T. J. Watson Research Center
Yorktown Heights, NY 10549

stanchen@us.ibm.com, picheny@us.ibm.com,
bhuvana@us.ibm.com

September 29, 2009

Administrivia

- Main feedback from Lecture 3:

  -Pace good

  -Some of you are familiar with the material

  -Muddiest topics: C(i,j) (x1), BIC (x1), E-M (x1), GMM (x1)

  (We will be repeating some of this again in this lecture)

- Lab 1 is due Sept. 30 (that is tomorrow!)

Recap

- Can model distributions of feature vectors using mixtures of Gaussians.
- Maximum Likelihood estimation can be used to estimate the parameters of Gaussian distributions.
- An iterative algorithm was developed to estimate parameters of mixtures of Gaussian distributions.
- Generalized the idea of Dynamic Time Warping to a probabilistic framework – the Hidden Markov Model.
- Introduced the three main operations that need to be addressed with Hidden Markov Models
  - Computing the probability of a sequence of feature vectors
  - Finding the best sequence of states that produced the sequence of feature vectors
  - Estimating the parameters of a Hidden Markov Model from data

Goals for Today

- Introduce a general probabilistic framework for speech recognition
- Explain how Hidden Markov Models fit in this overall framework.
- Review some of the concepts of ML estimation in the context of an HMM framework.
- Describe how the three basic HMM operations are computed.

"The" Probabilistic Model for Speech Recognition

$$W^* = \arg\max_W P(W \mid X, \Theta)$$

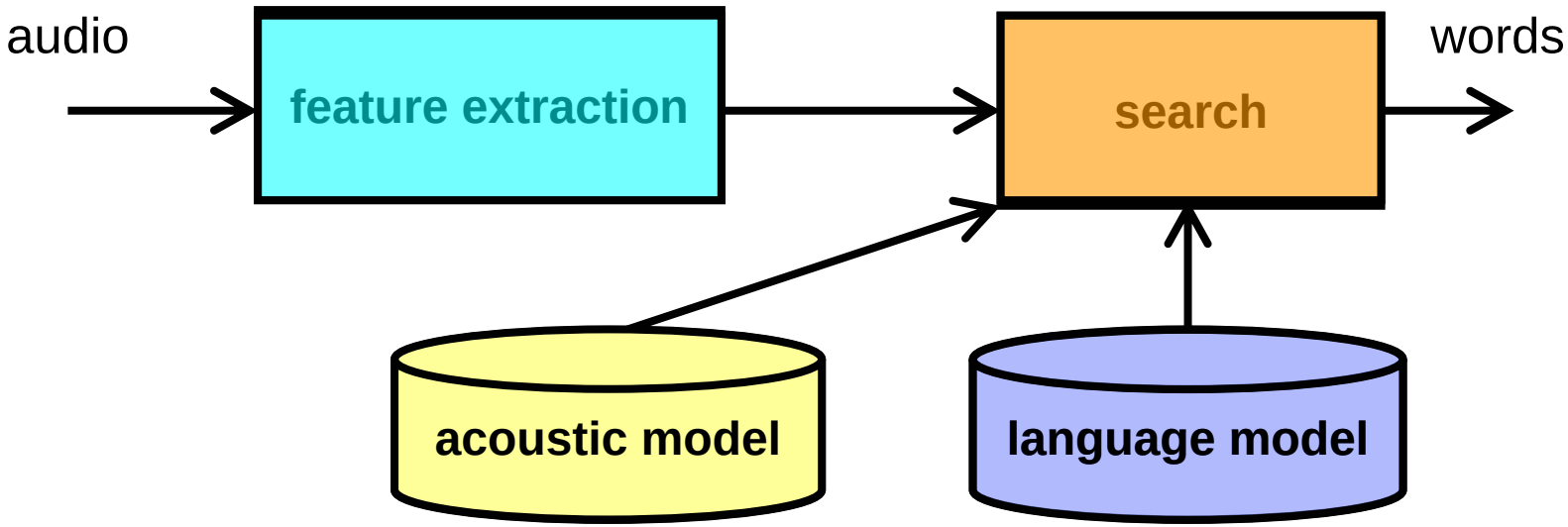
$$= \arg\max_W \frac{P(X \mid W, \Theta) P(W \mid \Theta)}{P(X)} \qquad \text{Bayes' rule}$$

$$= \arg\max_W P(X \mid W, \Theta) P(W \mid \Theta) \qquad \text{P(X) doesn't depend on W}$$

- W is a sequence of words, W* is the best sequence.
- X is a sequence of acoustic features
- Θ is a set of model parameters

A.K.A. The Fundamental Equation of Speech Recognition

Automatic speech recognition – Architecture



$$W^* = \arg\max_W P(X \mid W, \Theta) \, P(W \mid \Theta)$$

Today's Subject

Acoustic Modeling from 30000 feet

- Goal is to model sequences (i.e., compute probabilities) of feature vectors
- Assume a word is made up from a sequence of speech sounds

  Cat: K AE T

  Dog: D AO G

  Fish: F IH SH
- When a speech sound is uttered, a sequence of feature vectors is produced according to a GMM associated with each sound
- But the distributions overlap: you can't uniquely identify which speech sound produced the feature vector – so how can you compute the likelihood if you don't know which speech sound generated the feature vector?

  If you did, you could just use the techniques we discussed last week.
- Solution is the HMM
- For simplicity, build up HMM concepts by using the outcome of a coin flip as our observation (rather than a feature vector)

Models without Memory: Probability Sequence Computation

- A coin has probability of “heads” = p , probability of “tails” = 1-p
- Flip the coin 10 times. Assume I.I.D. random sequence. There are 2^{10} possible sequences.
- Sequence: 1 0 1 0 0 0 1 0 0 1
  Probability: $p(1-p)p(1-p)(1-p)(1-p)\ p(1-p)(1-p)p = p^4(1-p)^6$

  Models without memory: Observations are Independent. Probability is the same for all sequences with 4 heads & 6 tails. Order of heads & tails does not matter in assigning a probability to the sequence, only the number of heads & number of tails

- Probability of

  0 heads $(1-p)^{10}$

  1 head $p(1-p)^9$

  …

  10 heads p^{10}

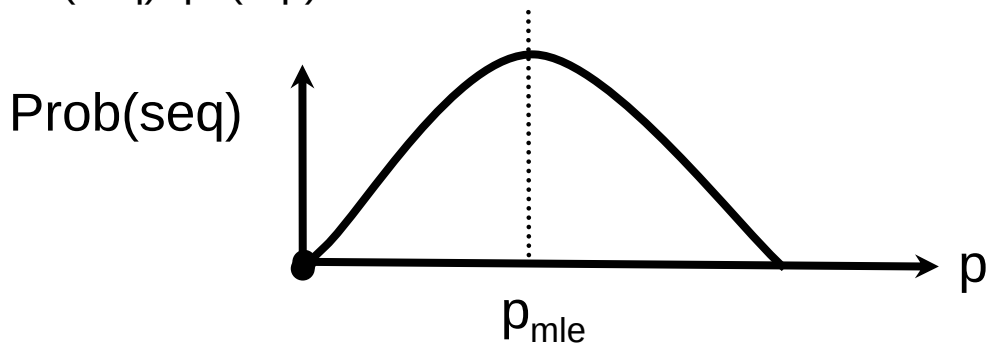
Models without Memory: Learning Model Parameters

If p is known, then it is easy to compute the probability of the sequence. Now suppose p is unknown.

We toss the coin N times, obtaining H heads and T tails, where H+T=N
We want to estimate p

A "reasonable" estimate is p=H/N. Is this actually the "best" choice for p?

What is "best"? Consider the probability of the observed sequence.
Prob(seq)=$p^H(1-p)^T$



The value of p for which Prob(seq) is maximized is the Maximum Likelihood Estimate (MLE) of p. (Denote p_{mle})

Models without Memory: Example, cont'd

Assertion: $p_{mle} = H/N$

Proof: $\text{Prob(seq)} = p^H(1-p)^T$

Maximizing Prob is equivalent to maximizing log(Prob)

$L = \log(\text{Prob(seq)}) = H \log p + T \log (1-p)$

$$\frac{\partial L}{\partial p} = H/p + T/(1-p)$$

L maximized when $\frac{\partial L}{\partial p} = 0$

$H/p_{mle} - T/(1-p_{mle}) = 0$

$H - H\, p_{mle} = T\, p_{mle}$

$H = T\, p_{mle} + H\, p_{mle} = p_{mle}\,(T + H) = p_{mle}\, N$

$p_{mle} = H/N$

Models without Memory Example, cont'd

- We showed that in this case
  MLE = Relative Frequency = H/N
- We will use this idea many times.
- Often, parameter estimation reduces to
  counting and normalizing.

Models with Memory: Markov Models

- Flipping a coin was memory-less. The outcome of each flip did not depend on the outcome of the other flips.
- Adding memory to a memory-less model gives us a Markov Model. Useful for modeling sequences of events.

Markov Model Example

- Consider 2 coins.
  Coin 1: $p_H = 0.9$, $p_T = 0.1$
  Coin 2: $p_H = 0.2$, $p_T = 0.8$

- Experiment:
  Flip Coin 1.
  for J = 2 ; J<=4; J++
  if (previous flip == “H”) flip Coin 1;
  else flip Coin 2;

- Consider the following 2 sequences:
  H H T T prob = 0.9 x 0.9 x 0.1 x 0.8 = .0648
  H T H T prob = 0.9 x 0.1 x 0.2 x 0.1 = .0018

- Sequences with consecutive heads or tails are more likely.
- The sequence has memory - order matters.
- Order matters for speech too.

  The sequence of feature vectors for “rat” are different from the sequence of vectors for “tar.”

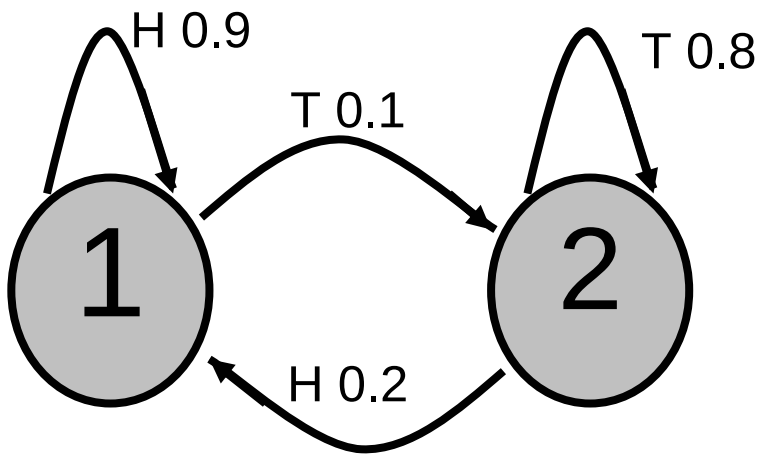
Markov Models – State Space Representation

- Consider 2 coins.

  Coin 1: $p_H = 0.9$, $p_T = 0.1$

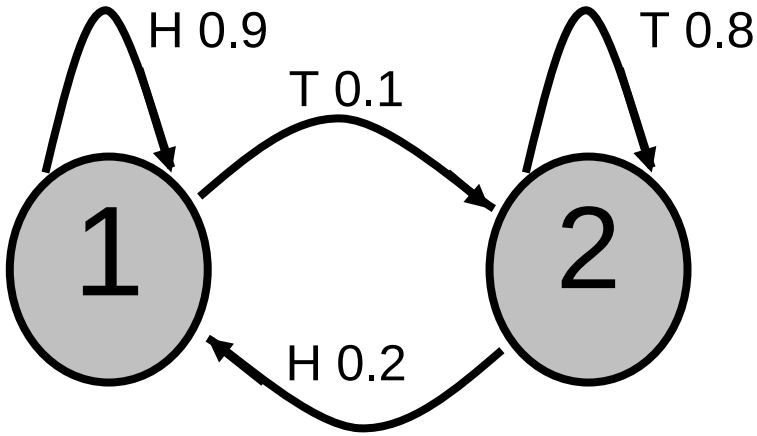
  Coin 2: $p_H = 0.2$, $p_T = 0.8$

State-space representation of previous example



Markov Models – State Space Representation (Con't)

- State sequence can be uniquely determined from the outcome sequence, given the initial state.
- Output probability is easy to compute. It is the product of the transition probs for state sequence.



- Example:

O:	H		T	T	T	
S:	1(given)		1	2	2	
Prob:	0.9	x	0.1	x 0.8	x 0.8	

Back to Memory-Less Models: Hidden Information

Let's return to the memory-less coin flip model

Consider 3 coins. Coin 0: $p_H = 0.7$
Coin 1: $p_H = 0.9$
Coin 2 $p_H = 0.2$

Experiment:
For J=1..4
Flip coin 0. If outcome == "H"
Flip coin 1 and record.
else
Flip coin 2 and record.

Hiding Information (cont.)

Coin 0: $p_H = 0.7$ Coin 1: $p_H = 0.9$ Coin 2: $p_H = 0.2$

We cannot uniquely determine the output of the Coin 0 flips. This is hidden.

Consider the sequence H T T T.
What is the probability of the sequence?

Order doesn't matter (memory-less)

p(head)=p(head|coin0=H)p(coin0=H)+
p(head|coin0=T)p(coin0=T)= 0.9x0.7 + 0.2x0.3 = 0.69

p(tail) = 0.1 x 0.7 + 0.8 x 0.3 = 0.31

$P(HTTT) = .69 \times .31^3$

Hidden Markov Model

- The state sequence is hidden.
- Unlike Markov Models, the state sequence cannot be uniquely deduced from the output sequence.

- Experiment:
  Flip the same two coins. This time, flip each coin twice. The first flip gets recorded as the output sequence. The second flip determines which coin gets flipped next.

- Now, consider output sequence H T T T.
- No way to know the results of the even numbered flips, so no way to know which coin is flipped each time.

- Unlike previous example, order now matters (start with coin 1, $p_H = 0.9$)

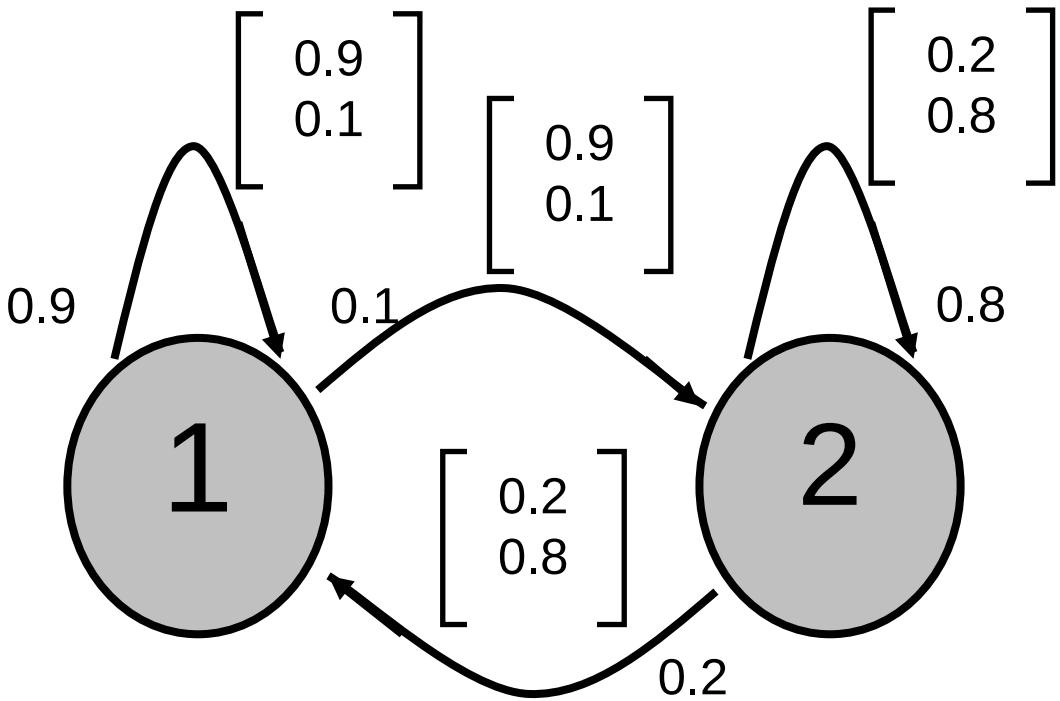
  H H T T T H T = .9 x .9 x .1 x .1 x .8 x .2 x .1 = .0001296

  T T T H T T H = .1 x .1 x .8 x .2 x .1 x .1 x .2 = .0000032
- Even worse, same output sequence corresponds to multiple probabilities!

  H T T H T T T = .9 x .1 x .8 x .2 x .1 x .1 x .8 = .0001152

Hidden Markov Model

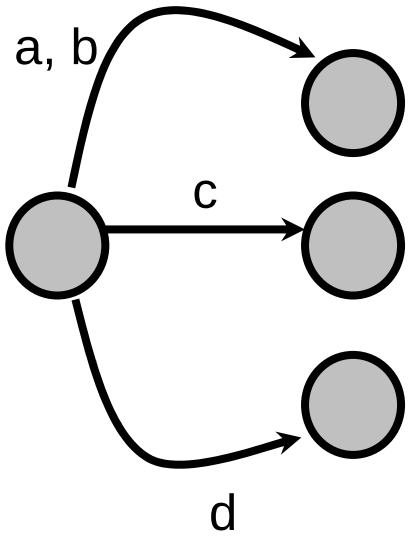
- The state sequence is hidden. Unlike Markov Models, the state sequence cannot be uniquely deduced from the output sequence.



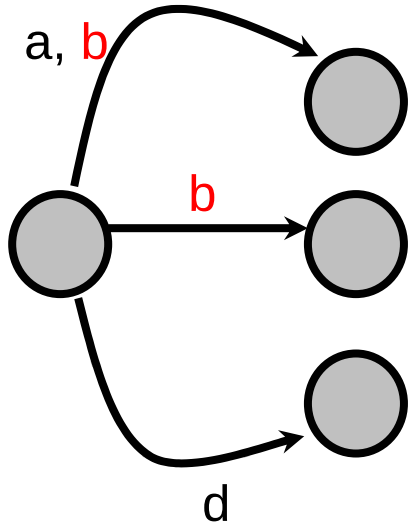
- In speech, the underlying states can be, say the positions of the articulators. These are hidden – they are not uniquely deduced from the output features. We already mentioned that speech has memory. A process which has memory and hidden states suggests using an HMM.

Is a Markov Model Hidden or Not?

A necessary and sufficient condition for being state-observable is that all transitions from each state produce different outputs



State-observable



Hidden

Three problems of general interest for an HMM

3 problems need to be solved before we can use HMM's:

- 1. Given an observed output sequence $X=x_1x_2..x_T$, compute $P_\theta(X)$ for a given model θ (scoring)
- 2. Given X, find the most likely state sequence (Viterbi algorithm)

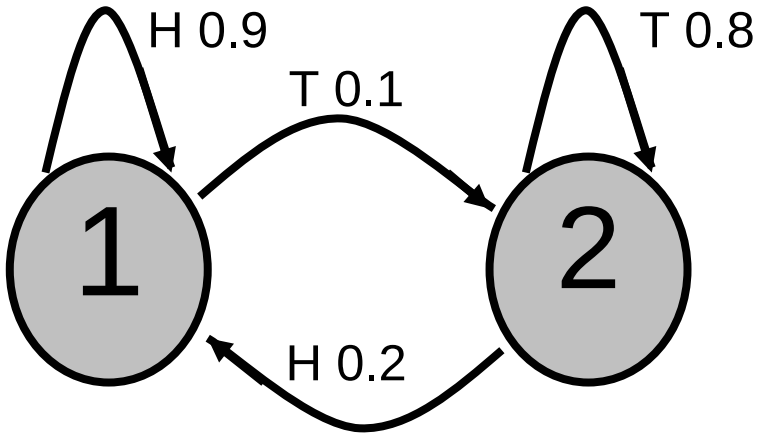
3. Estimate the parameters of the model (training)

These problems are easy to solve for a state-observable Markov model. More complicated for an HMM because we need to consider all possible state sequences. Must develop a generalization….

Problem 1

1. Given an observed output sequence $X=x_1x_2..x_T$, compute $P_\theta(X)$ for a given model θ

- Recall the state-observable case



- Example:

O:	H		T		T		T
S:	1(given)		1		2		2
Prob:	0.9	x	0.1	x	0.8	x	0.8

Problem 1

1. Given an observed output sequence $X=x_1x_2..x_T$, compute $P_\theta(X)$ for a given model θ

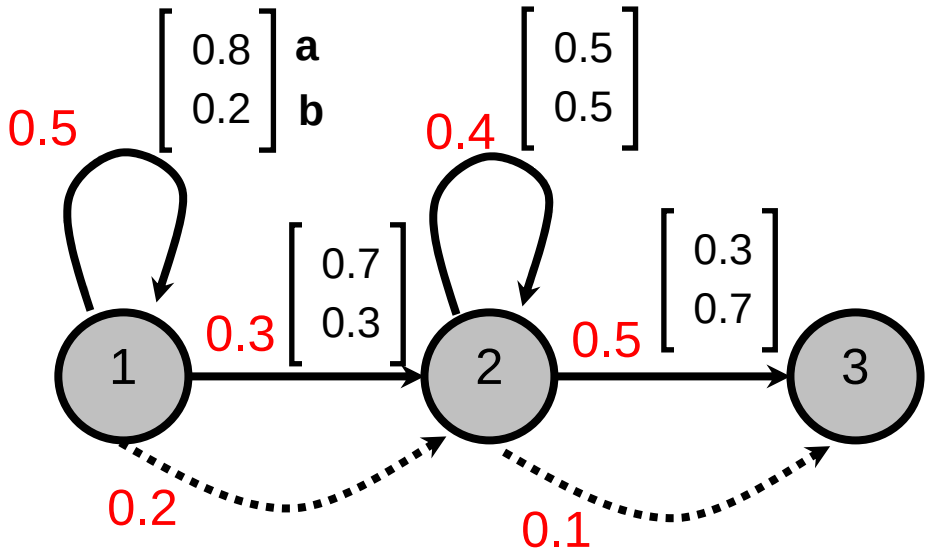
Sum over all possible state sequences:

$$P_\theta(X)=\sum_S P_\theta(X,S)$$

The obvious way of calculating $P_\theta(X)$ is to enumerate all state sequences that produce X

Unfortunately, this calculation is exponential in the length of the sequence

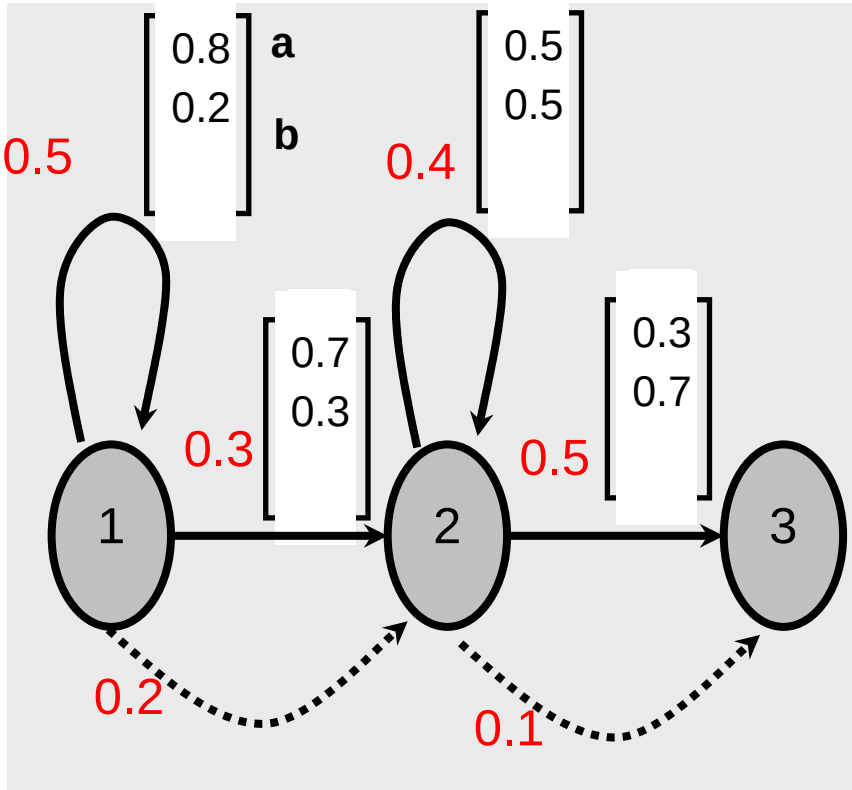
Example for Problem 1

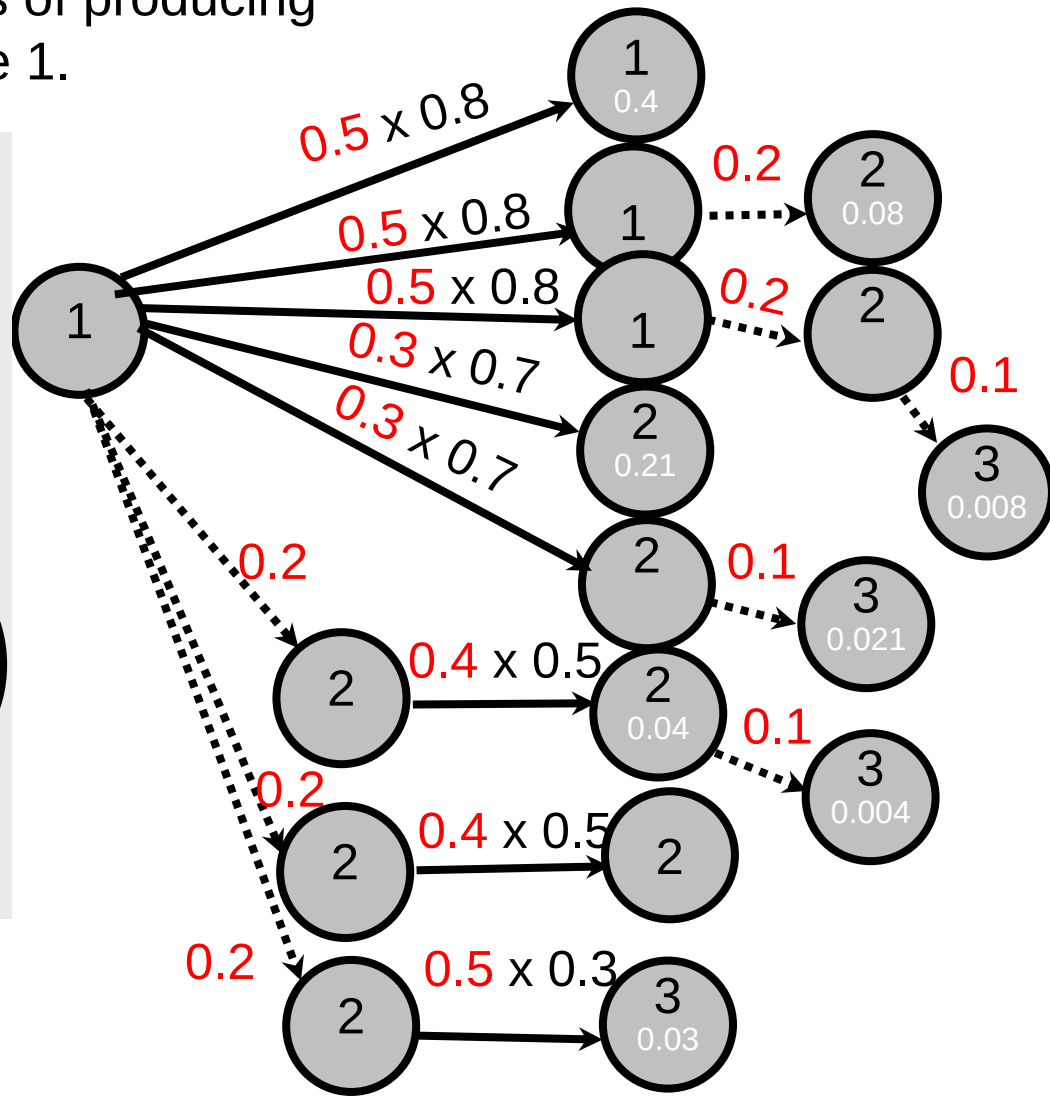


Compute $P_\theta(X)$ for X=aabb, assuming we start in state 1

Example for Problem 1,cont'd

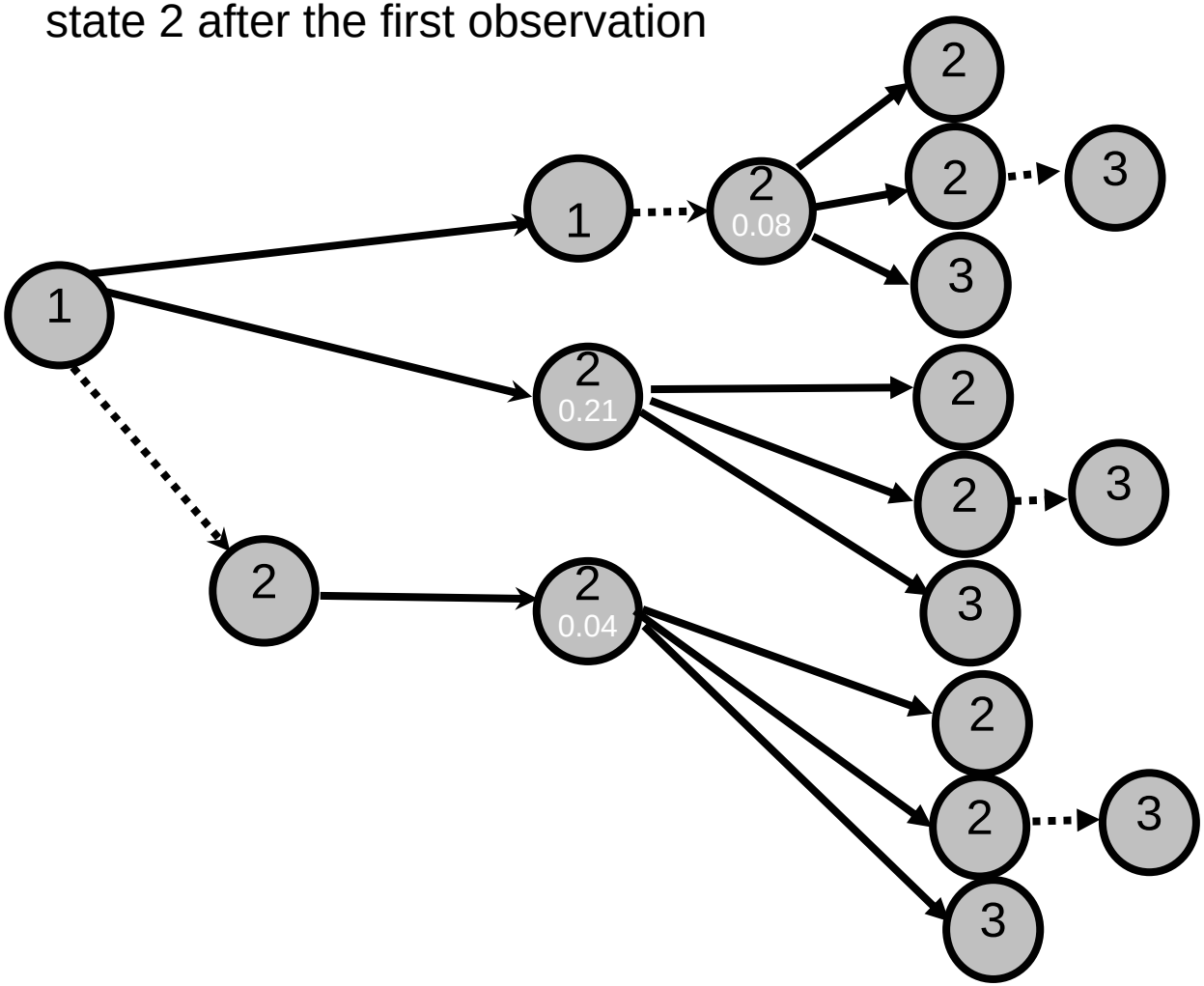
Let's enumerate all possible ways of producing x_1=a, assuming we start in state 1.





Example for Problem 1, cont'd

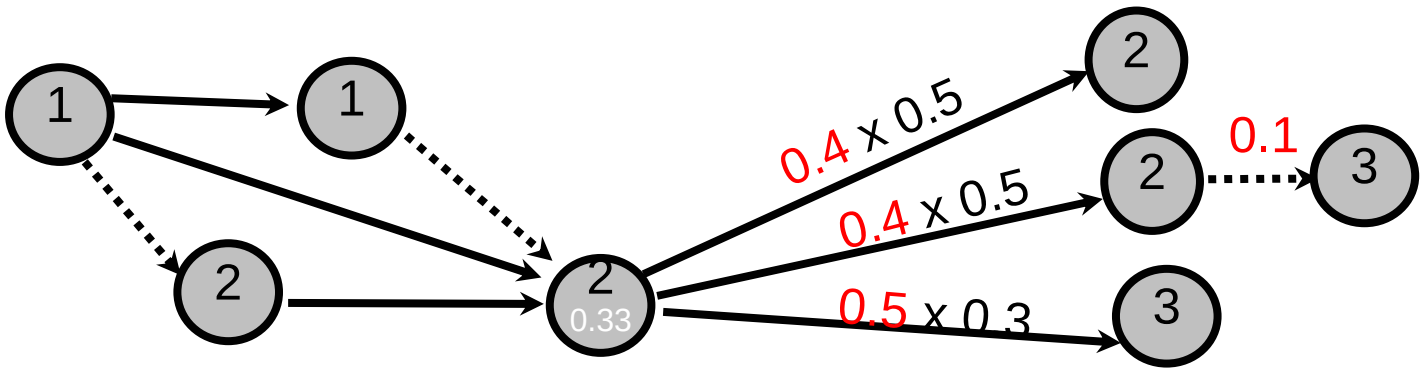
- Now let's think about ways of generating x1x2=aa, for all paths from state 2 after the first observation



Example for Problem 1,cont'd

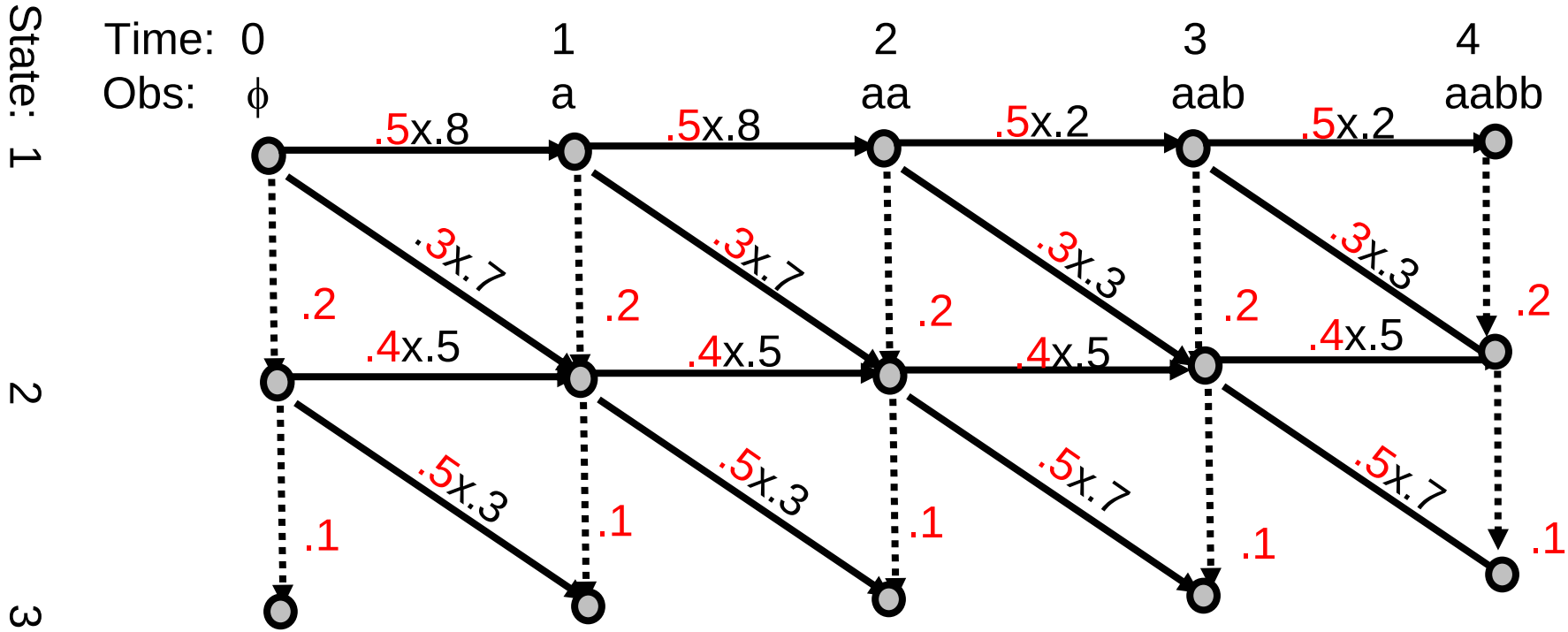
We can save computations by combining paths.

This is a result of the Markov property, that the future doesn't depend on the past if we know the current state



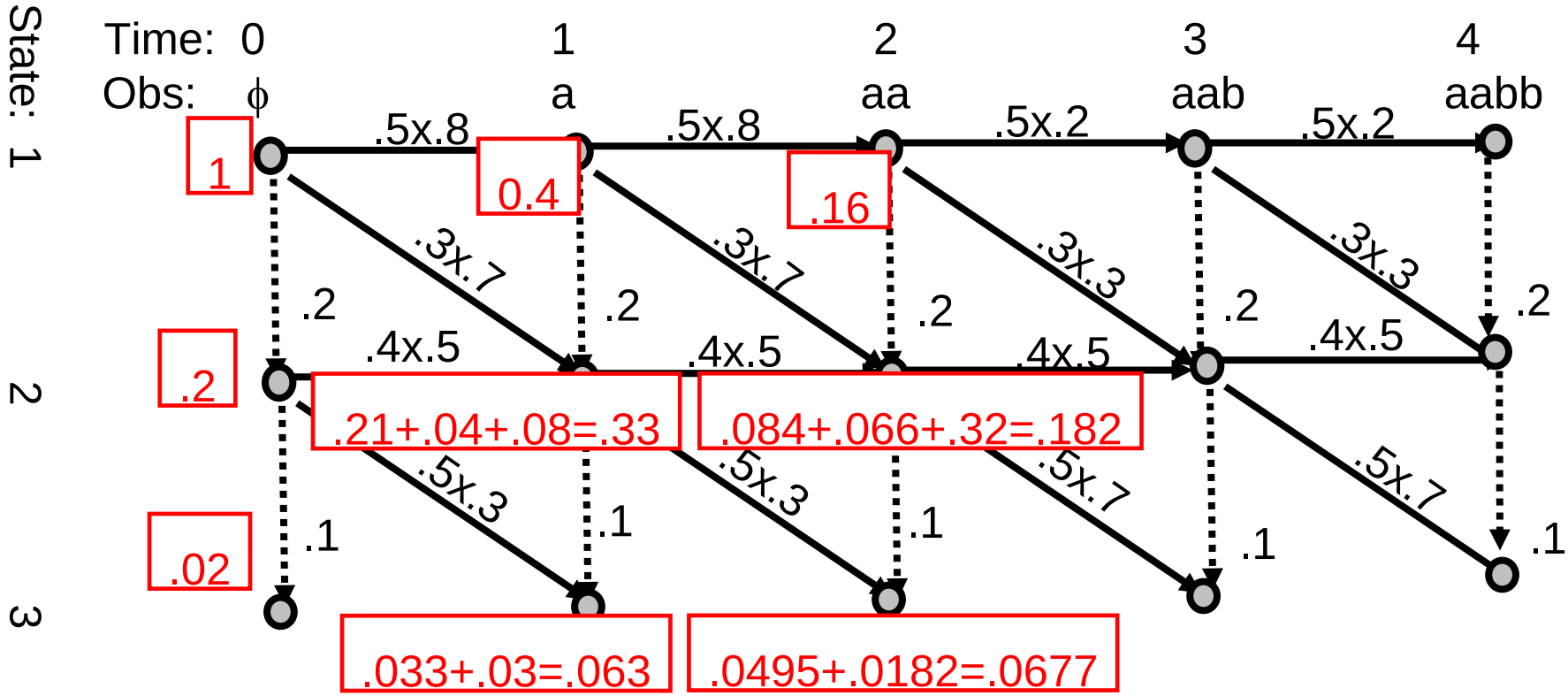
Problem 1: Trellis Diagram

- Expand the state-transition diagram in time.
- Create a 2-D lattice indexed by state and time.
- Each state transition sequence is represented exactly once.



Problem 1: Trellis Diagram, cont'd

- Now let's accumulate the scores. Note that the inputs to a node are from the left and top, so if we work to the right and down all necessary input scores will be available.



Problem 1: Trellis Diagram, cont'd

Boundary condition:
Score of (state 1, ϕ) = 1.

Basic recursion:
Score of node i = 0

For the set of predecessor nodes j:
Score of node i += score of predecessor node j x
the transition probability from j to i x
observation probability along
that transition if the transition is not null.

Problem 1: Forward Pass Algorithm

Let $\alpha_t(s)$ for t ε {1..T} be the probability of being in state s at time t and having produced output $x_1^t = x_1..x_t$

$$\alpha_t(s) = \sum_{s'} \alpha_{t-1}(s') P_\theta(s|s') P_\theta(x_t|s'->s) + \sum_{s'} \alpha_t(s') P_\theta(s|s')$$

1st term: sum over all output producing arcs 2nd term: all null arcs

This is called the Forward Pass algorithm.

Important: The computational complexity of the forward algorithm is linear in time (or in the length of the observation sequence)

This calculation allows us to solve Problem 1:

$$\mathbf{P_\theta(X) = \sum_s \alpha_T(s)}$$

Note that in many cases you may wish to compute $\mathbf{P_\theta(X,s_f)}$

Problem 2

Given the observations X, find the most likely state sequence

This is solved using the Viterbi algorithm

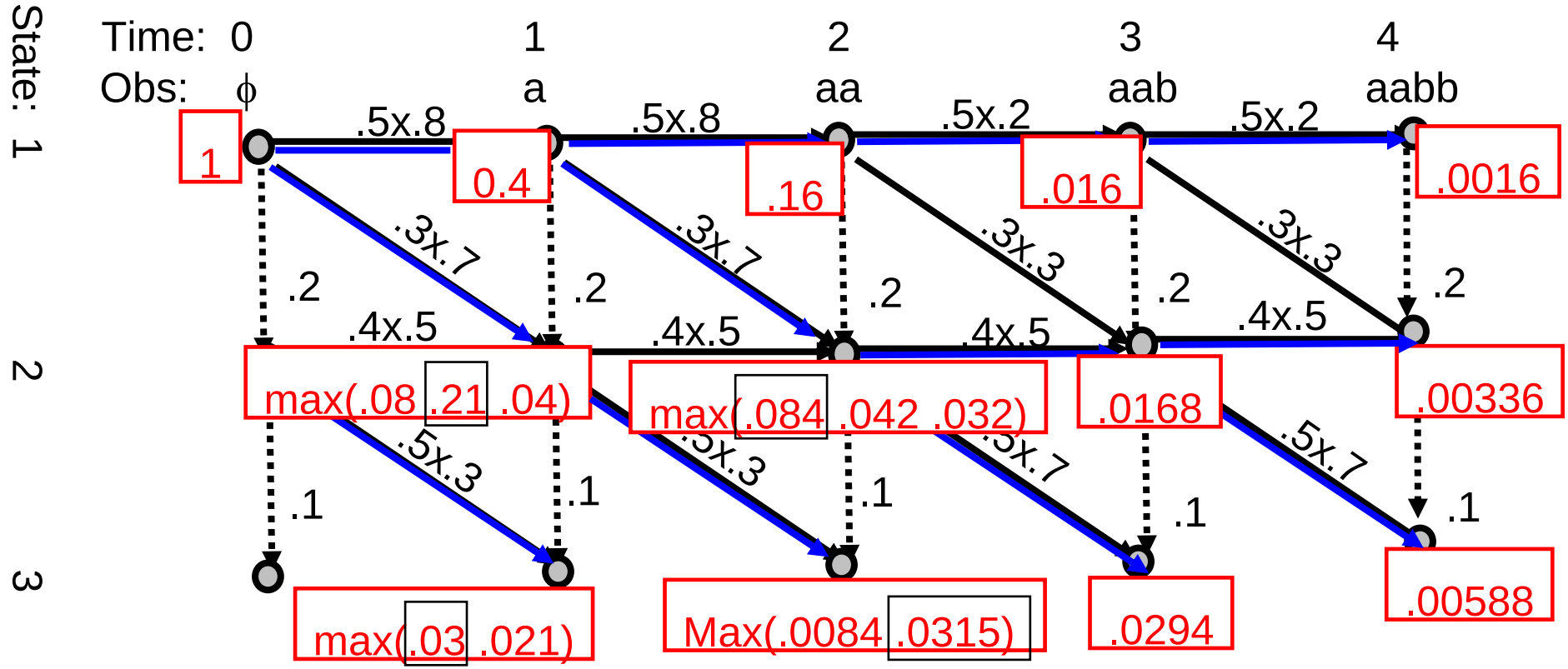
Preview:

The computation is similar to the forward algorithm, except we use max( ) instead of +

Also, we need to remember which partial path led to the max

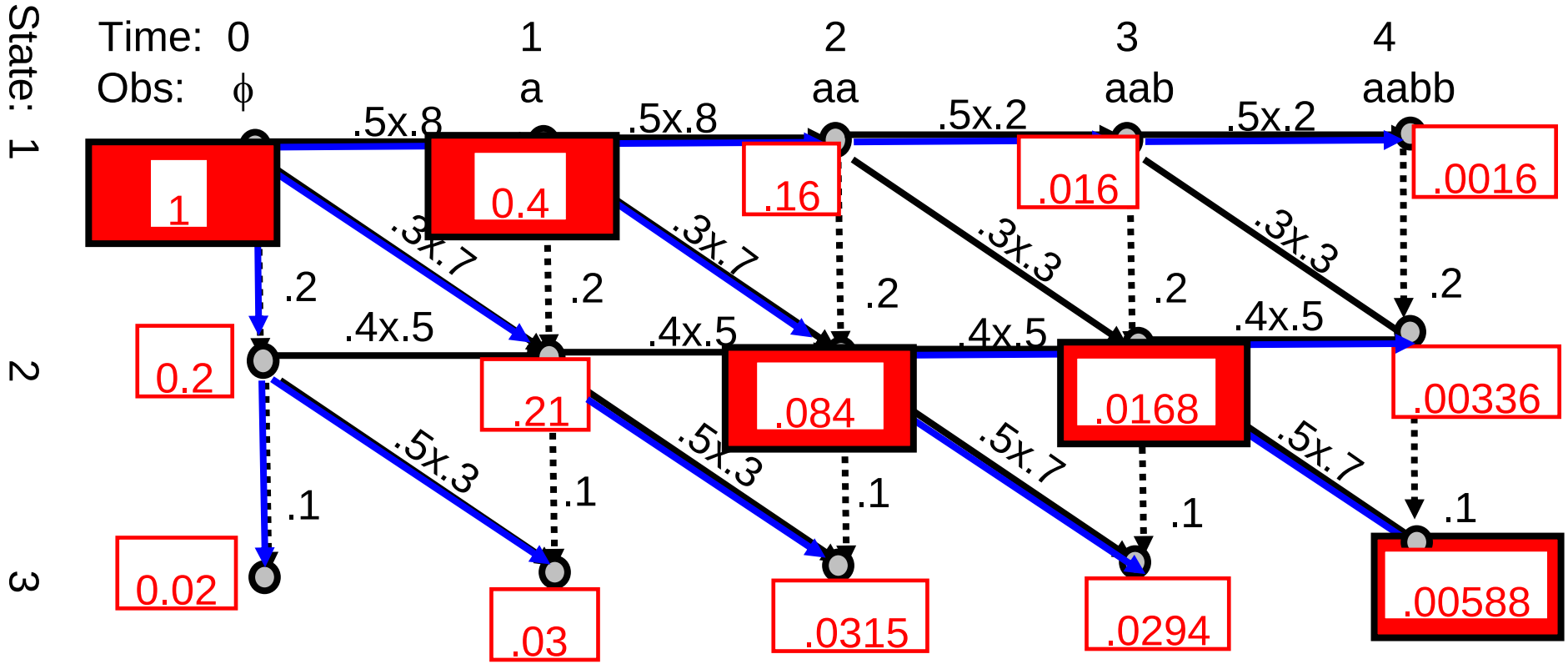
Problem 2: Viterbi algorithm

Returning to our example, let's find the most likely path for producing aabb. At each node, remember the max of predecessor score x transition probability. Also store the best predecessor for each node.



Problem 2: Viterbi algorithm, cont'd

Starting at the end, find the node with the highest score.
Trace back the path to the beginning, following best arc leading into each node along the best path.



Problem 3

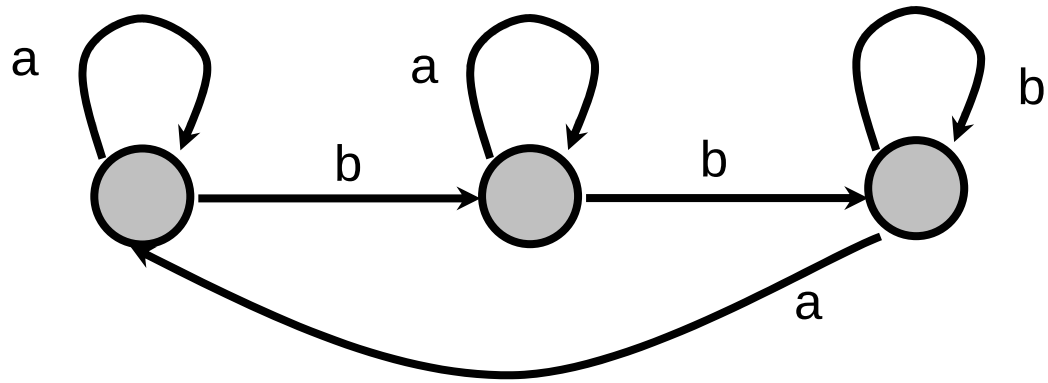
Estimate the parameters of the model. (training)

- Given a model topology and an output sequence, find the transition and output probabilities such that the probability of the output sequence is maximized.

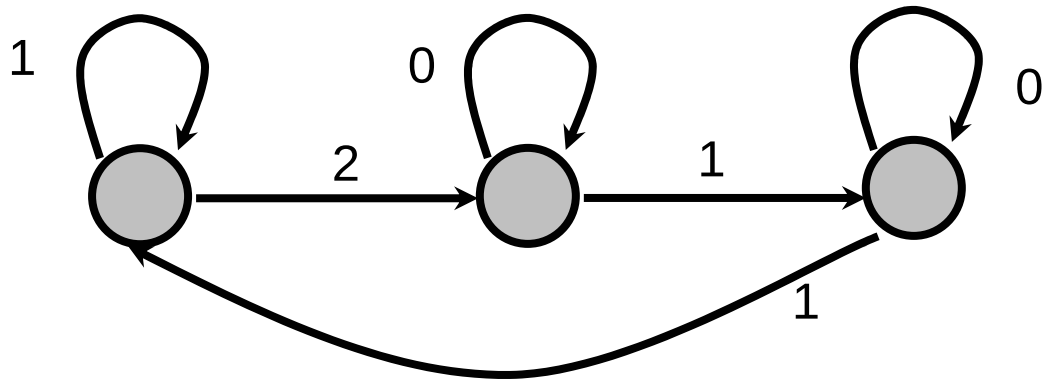
- Recall in the state-observable case, we simply followed the unique path, giving a count to each transition.

Problem 3 – State Observable Example

- Assume the output sequence X=abbab, and we start in state 1.

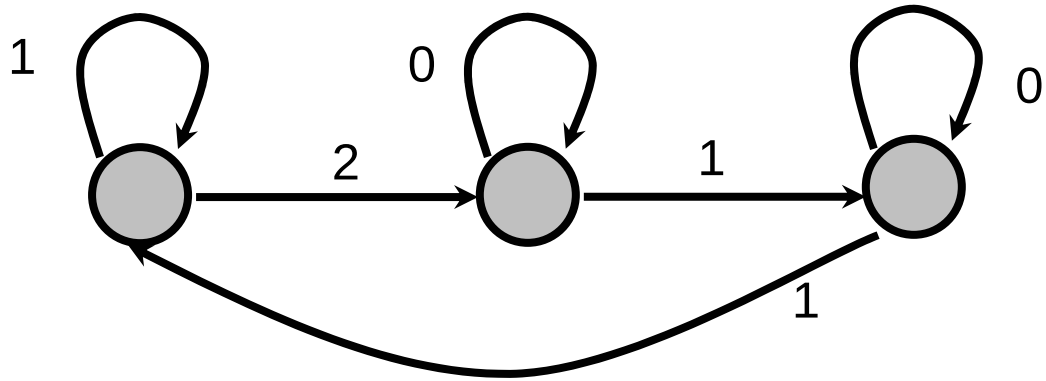


- Observed counts along transitions:

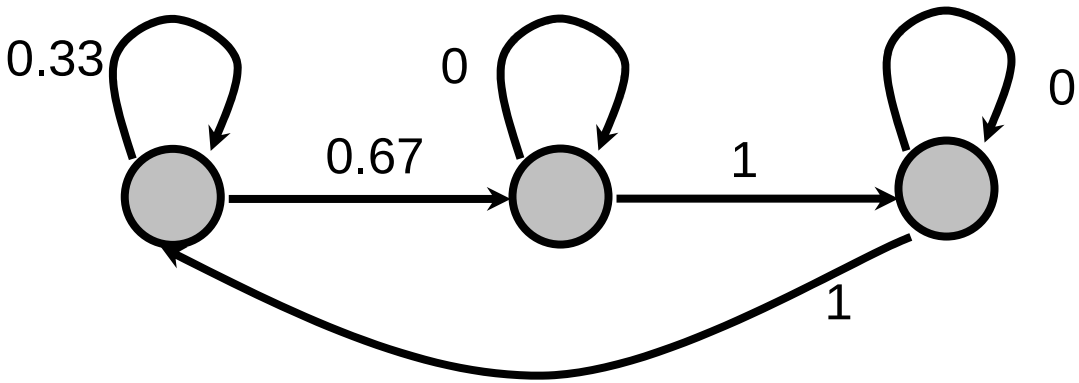


Problem 3 – State Observable Example

Observed counts along transitions:



Estimated transition probabilities. (this is of course too little data to estimate these well.)



Generalization to Hidden MM case

State-observable

- Unique path
- Give a count of 1 to each transition along the path

Hidden states

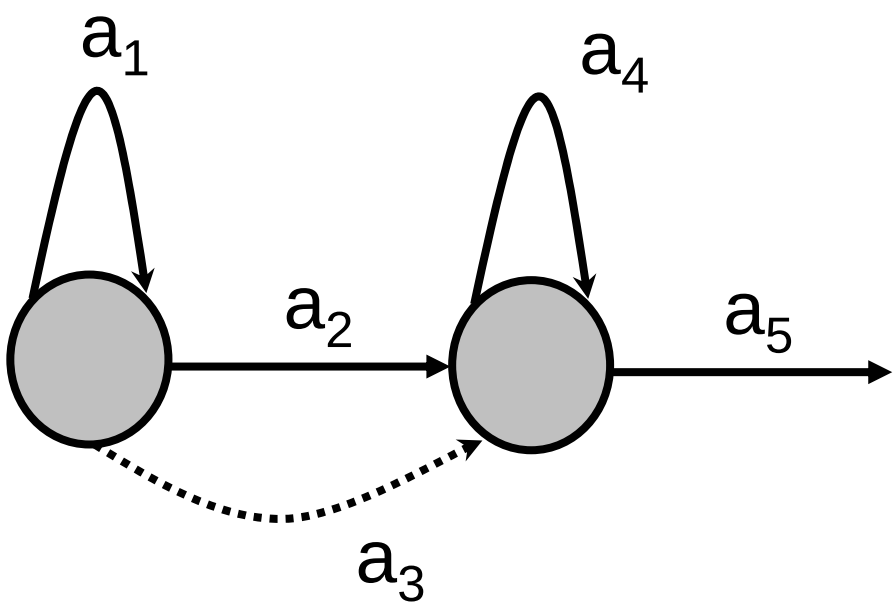
- Many paths
- Assign a fractional count to each path
- For each transition on a given path, give the fractional count for that path
- Sum of the fractional counts =1
- How to assign the fractional counts??

How to assign the fractional counts to the paths

- Guess some values for the parameters
- Compute the probability for each path using these parameter values
- Assign path counts in proportion to these probabilities
- Re-estimate parameter values
- Iterate until parameters converge

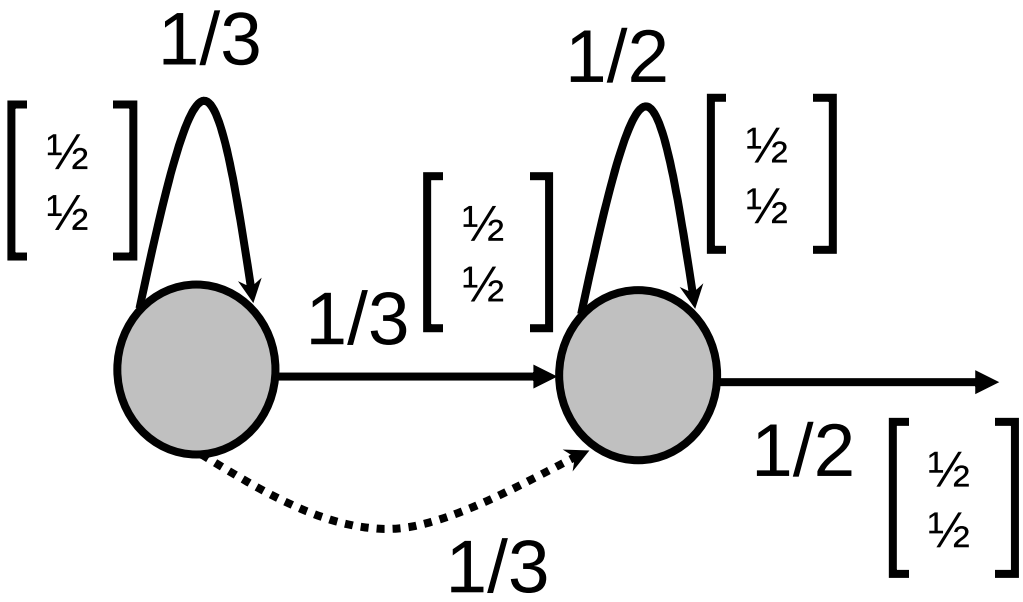
Problem 3: Enumerative Example – Assigning fractional counts

- For the following model, estimate the transition probabilities and the output probabilities for the sequence X=abaa

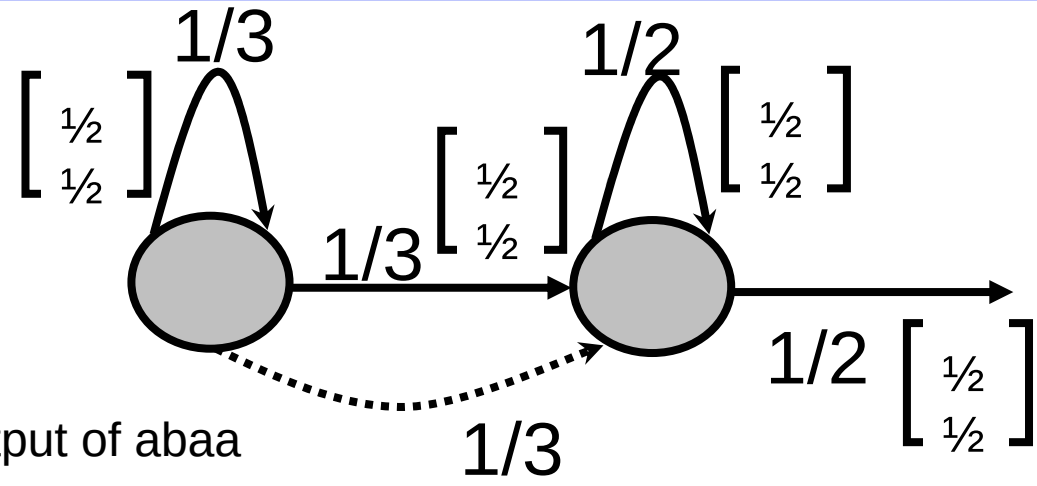


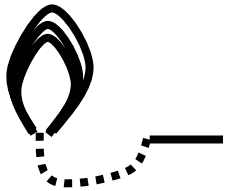
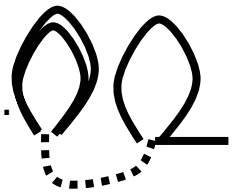
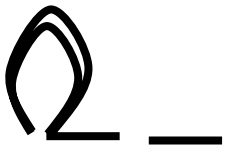
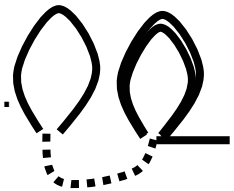
Problem 3: Enumerative Example - Assigning fractional counts

- Initial guess: equiprobable

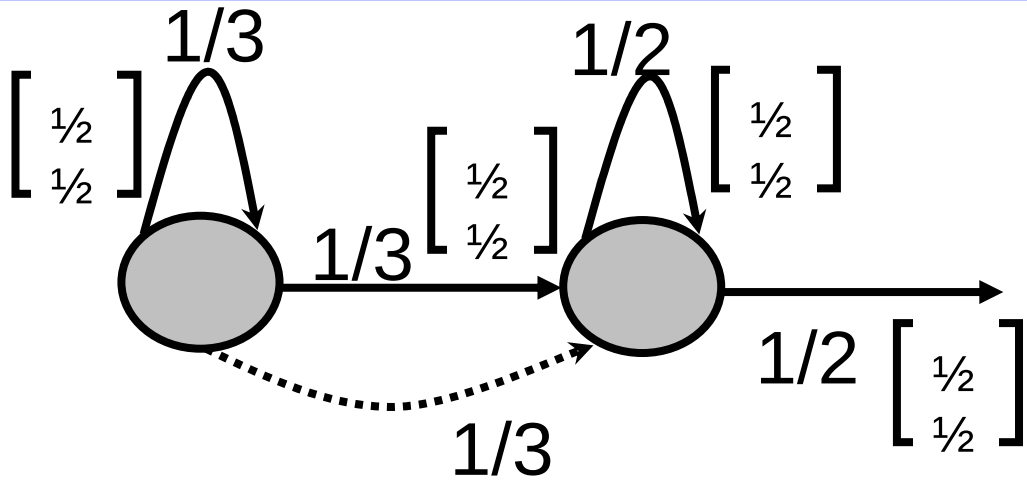


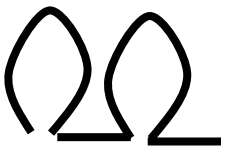
Problem 3: Enumerative Example cont'd



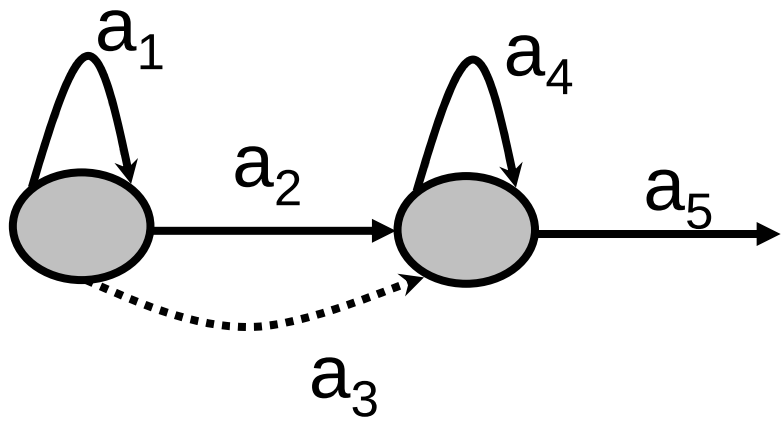
- 7 paths corresponding to an output of abaa
- 1.  pr(X,path1)=1/3x1/2x1/3x1/2x1/3x1/2x1/3x1/2x1/2=.000385
- 2.  pr(X,path2)=1/3x1/2x1/3x1/2x1/3x1/2x1/2x1/2x1/2=.000578
- 3.  pr(X,path3)=1/3x1/2x1/3x1/2x1/3x1/2x1/2x1/2=.001157
- 4.  pr(X,path4)=1/3x1/2x1/3x1/2x1/2x1/2x1/2x1/2x1/2=.000868

Problem 3: Enumerative Example cont'd



- 7 paths:
- 5.  pr(X,path5)=1/3x1/2x1/3x1/2x1/2x1/2x1/2x1/2=.001736
- 6.  pr(X,path6)=1/3x1/2x1/2x1/2x1/2x1/2x1/2x1/2x1/2=.001302
- 7.  pr(X,path7)=1/3x1/2x1/2x1/2x1/2x1/2x1/2x1/2=.002604
- $Pr(X) = \Sigma_i\ pr(X,path_i) = .008632$

Problem 3: Enumerative Example cont'd



- Let C_i be the a posteriori probability of path i
- $C_i = pr(X,path_i)/pr(X)$
-
- $C_1 = .045$ $C_2 = .067$ $C_3 = .134$ $C_4=.100$ $C_5 =.201$ $C_6=.150$ $C_7=.301$

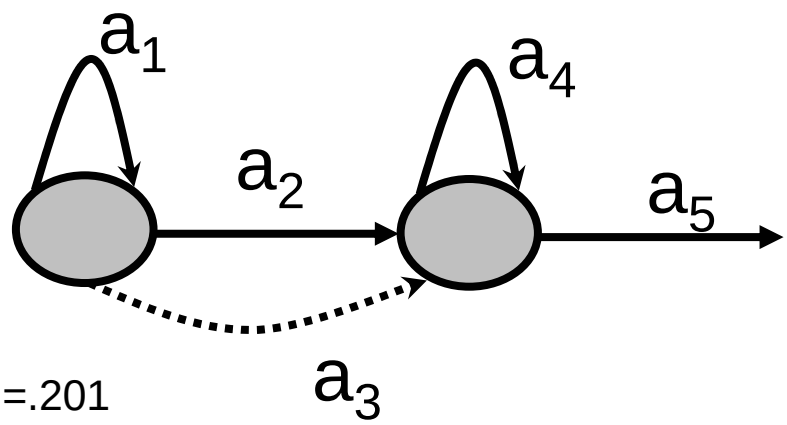
- $Count(a_1)= 3C_1+2C_2+2C_3+C_4+C_5 = .838$
- $Count(a_2)=C_3+C_5+C_7 = .637$
- $Count(a_3)=C_1+C_2+C_4+C_6 = .363$

- New estimates:
- $a_1 =.46$ $a_2 = .34$ $a_3=.20$

- $Count(a_1,'a') = 2C_1+C_2+C_3+C_4+C_5 = .592$ $Count(a_1,'b')=C_1+C_2+C_3=.246$

- New estimates:
- $p(a_1,'a')= .71$ $p(a_1,'b')= .29$

Problem 3: Enumerative Example cont'd



- Count(a_2,'a') = C_3+C_7 = .436 Count(a_2,'b')=C_5 =.201

- New estimates:
- p(a_2,'a')= .68 p(a_2,'b')= .32

- Count(a_4)=C_2+2C_4+C_5+3C_6+2C_7 = 1.52
- Count(a_5)=C_1+C_2+C_3+C_4+C_5+C_6+C_7 = 1.00

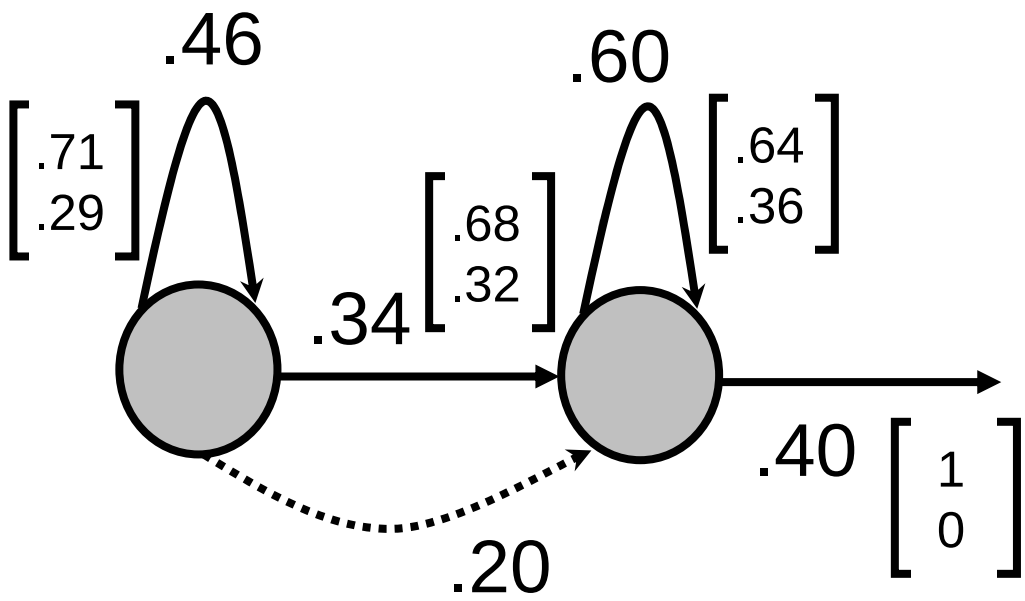
- New estimates: a_4=.60 a_5=.40
- Count(a_4,'a') = C_2+C_4+C_5+2C_6+C_7 = .972 Count(a_4,'b')=C_4+C_6+C_7=.553

- New estimates:
- p(a_4,'a')= .64 p(a_4,'b')= .36

- Count(a_5,'a') = C_1+C_2+C_3+C_4+C_5+2C_6+C_7 = 1.0 Count(a_5,'b')=0
- New estimates:
- p(a_5,'a')= 1.0 p(a_5,'b')= 0

Problem 3: Enumerative Example cont'd

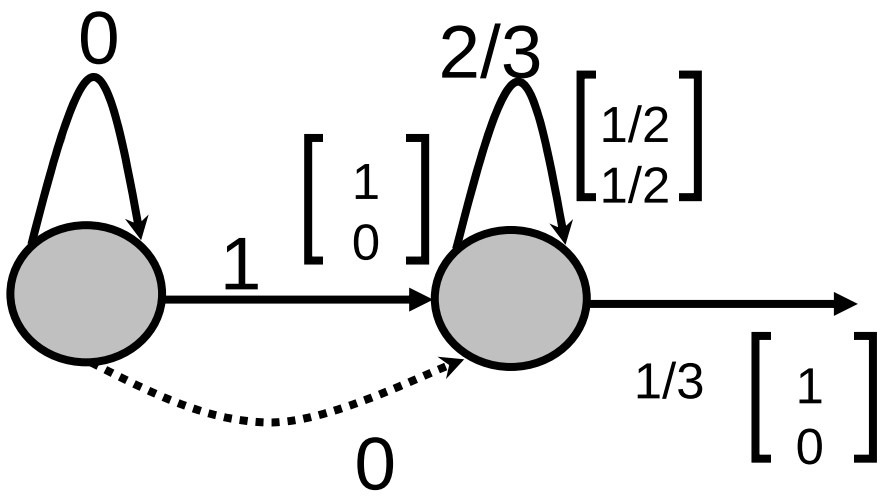
- New parameters



- Recompute Pr(X) = .02438 > .008632
- Keep on repeating…..

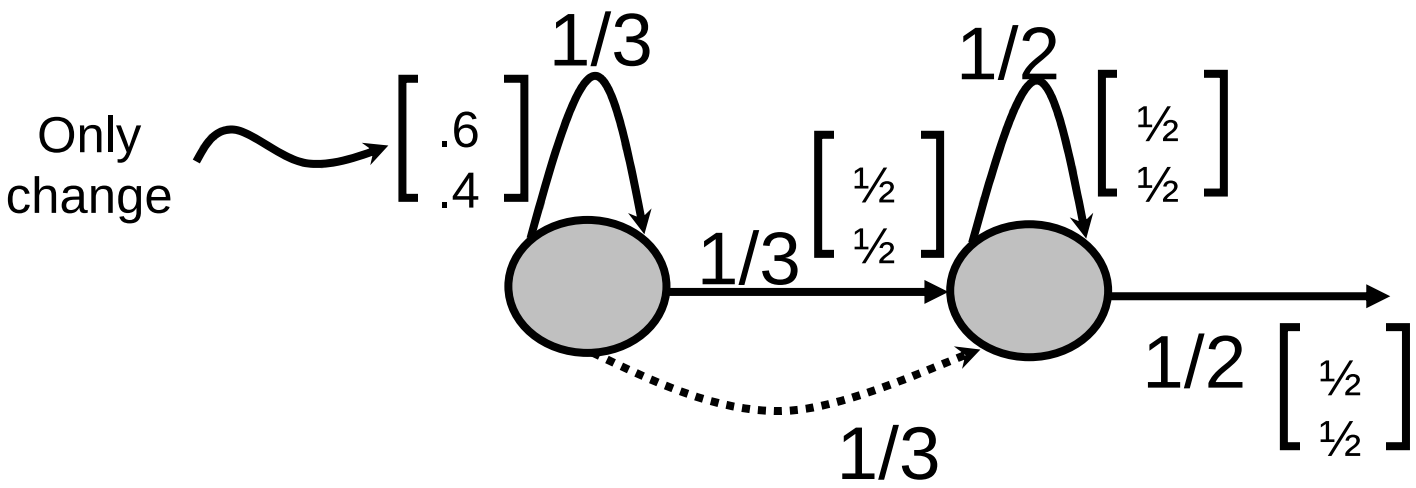
Problem 3: Enumerative Example cont'd

Step	Pr(X)	
1	0.008632	
2	0.02438	
3	0.02508	
100	0.03125004	
600	0.037037037	converged



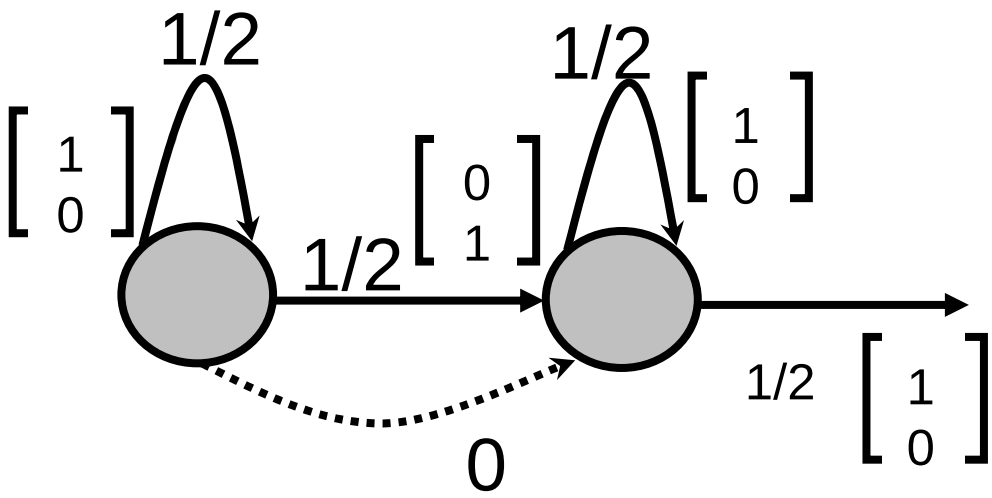
Problem 3: Enumerative Example cont'd

- Let's try a different initial parameter set



Problem 3: Enumerative Example cont'd

Step	Pr(X)
1	0.00914
2	0.02437
3	0.02507
10	0.04341
16	0.0625 converged



Problem 3: Parameter Estimation Performance

- The above re-estimation algorithm converges to a local maximum.
- The final solution depends on the starting point.
- The speed of convergence depends on the starting point.

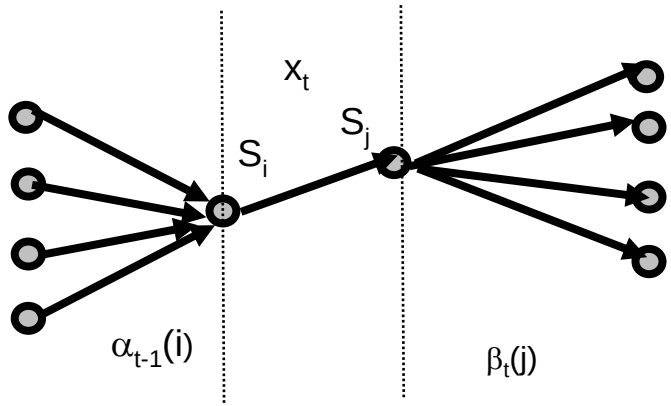
Problem 3: Forward-Backward Algorithm

- The forward-backward algorithm improves on the enumerative algorithm by using the trellis
- Instead of computing counts for each path, we compute counts for each transition at each time in the trellis.
- This results in the reduction from exponential computation to linear computation.

Problem 3: Forward-Backward Algorithm

Consider transition from state i to j, tr_{ij}

Let $p_t(tr_{ij},X)$ be the probability that tr_{ij} is taken at time t, and the complete output is X.



$$p_t(tr_{ij},X) = \alpha_{t-1}(i)\ a_{ij}\ b_{ij}(x_t)\ \ \beta_t(j)$$

Problem 3: F-B algorithm cont'd

$p_t(tr_{ij},X) = \alpha_{t-1}(i)\, a_{ij}\, b_{ij}(x_t)\ \ \beta_t(j)$

where:

$\alpha_{t-1}(i) = Pr(state=i, x_1 \ldots x_{t-1})$ = probability of being in state i and having produced $x_1 \ldots x_{t-1}$

a_{ij} = transition probability from state i to j

$b_{ij}(x_t)$ = probability of output symbol x_t along transition ij

$\beta_t(j) = Pr(x_{t+1} \ldots x_T | state= j)$ = probability of producing $x_{t+1} \ldots x_T$ given you are in state j

Problem 3: F-B algorithm cont'd

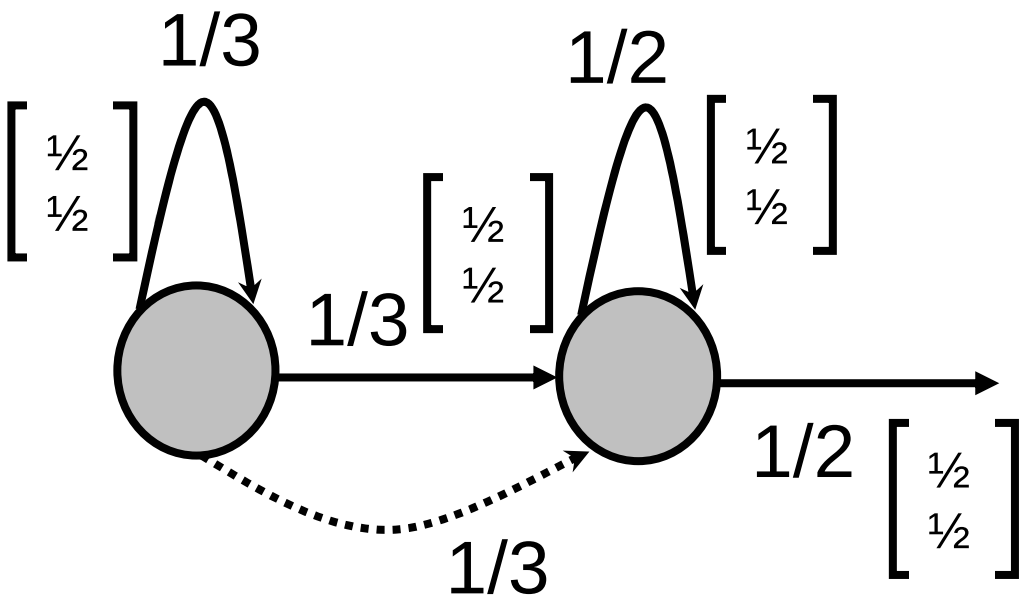
- Transition count $c_t(tr_{ij}|X) = p_t(tr_{ij},X) / Pr(X)$
- The β's are computed recursively in a backward pass (analogous to the forward pass for the α's)

$$\beta_t(j) = \Sigma_k \beta_{t+1}(k) \; a_{jk} \; b_{jk}(x_{t+1}) \quad \text{(for all output producing arcs)}$$

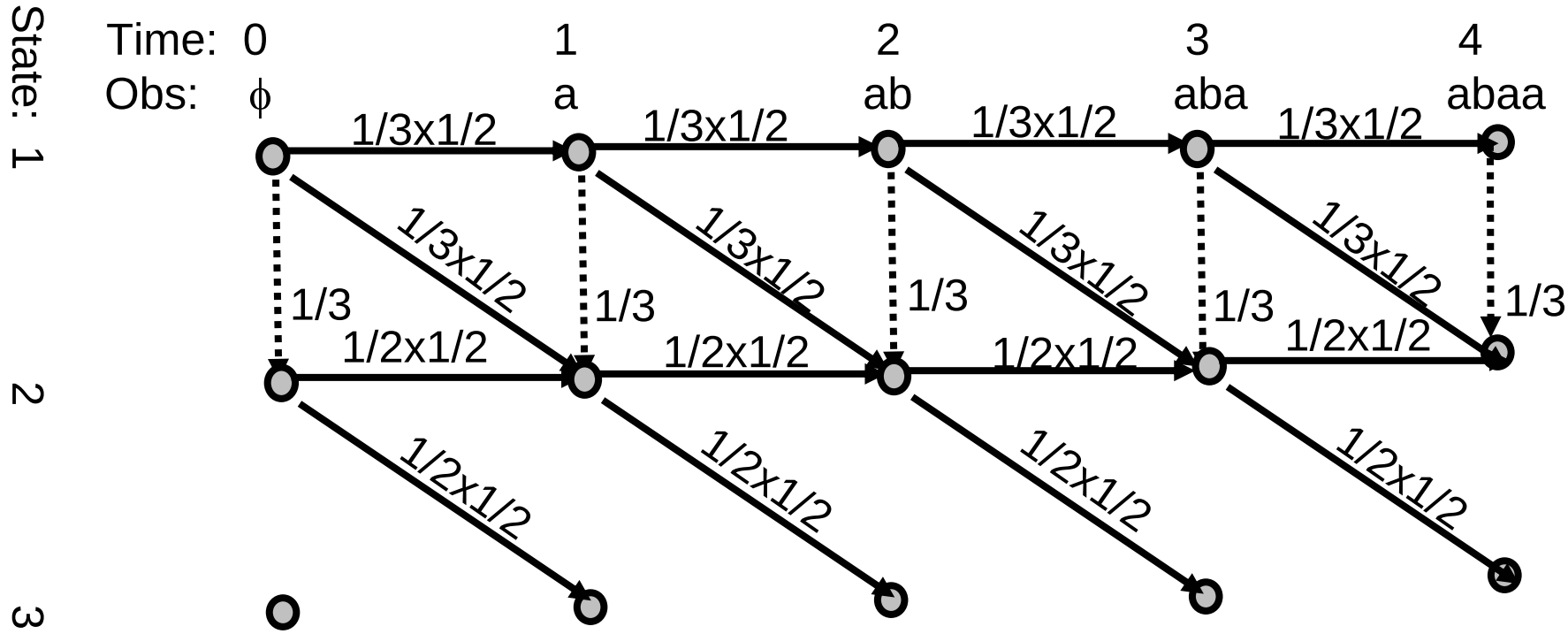
$$+ \Sigma_k \beta_t(k) \; a_{jk} \quad \text{(for all null arcs)}$$

Problem 3: F-B algorithm cont'd

- Let's return to our previous example, and work out the trellis calculations

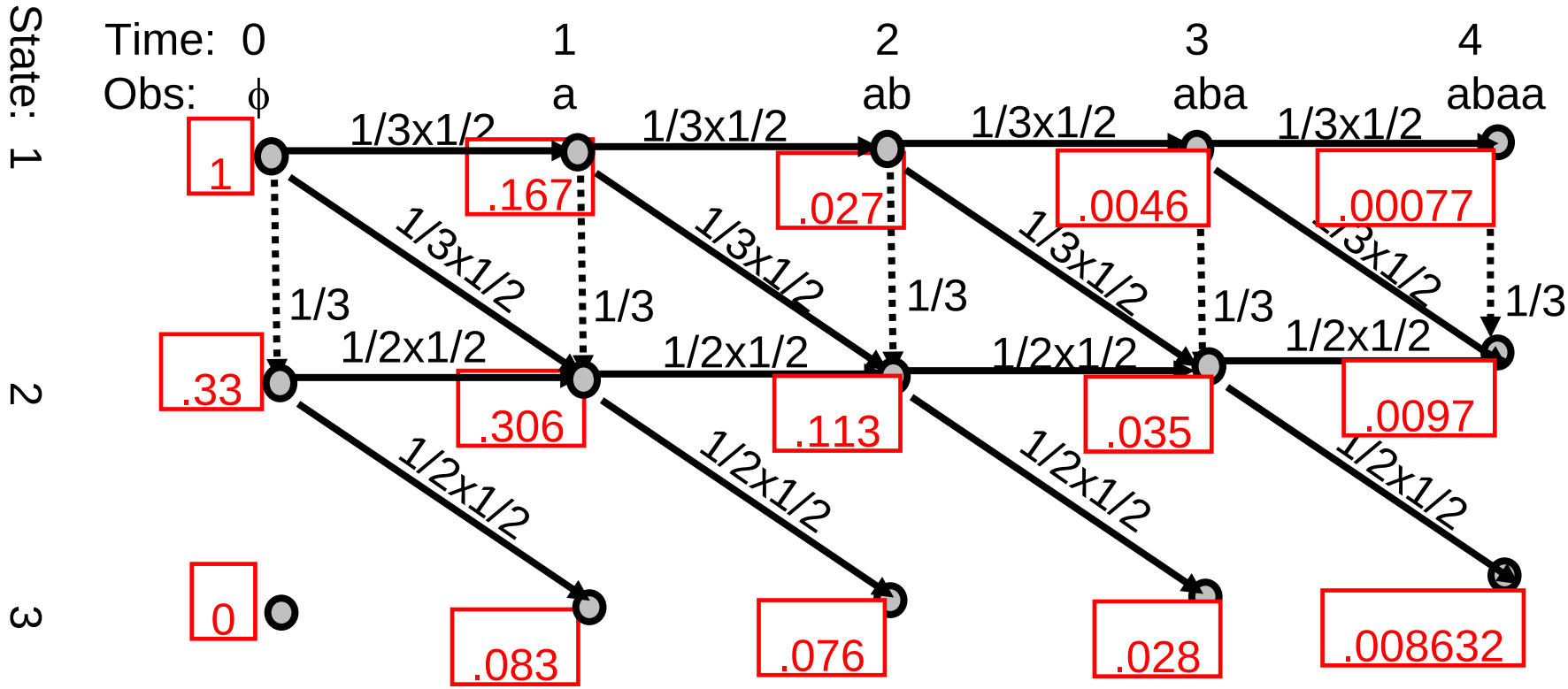


Problem 3: F-B algorithm, cont'd



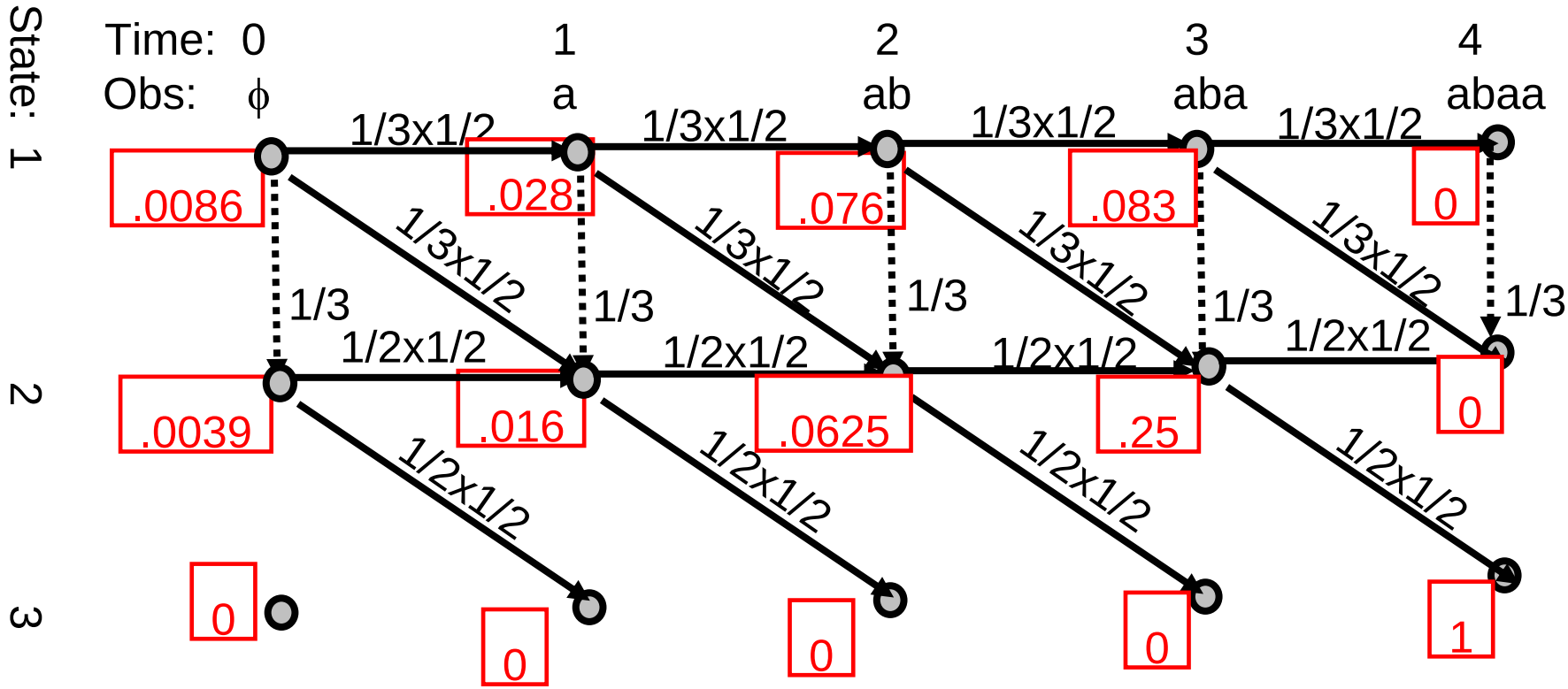
Problem 3: F-B algorithm, cont'd

Compute α's. since forced to end at state 3, α_T=.008632=Pr(X)



Problem 3: F-B algorithm, cont'd

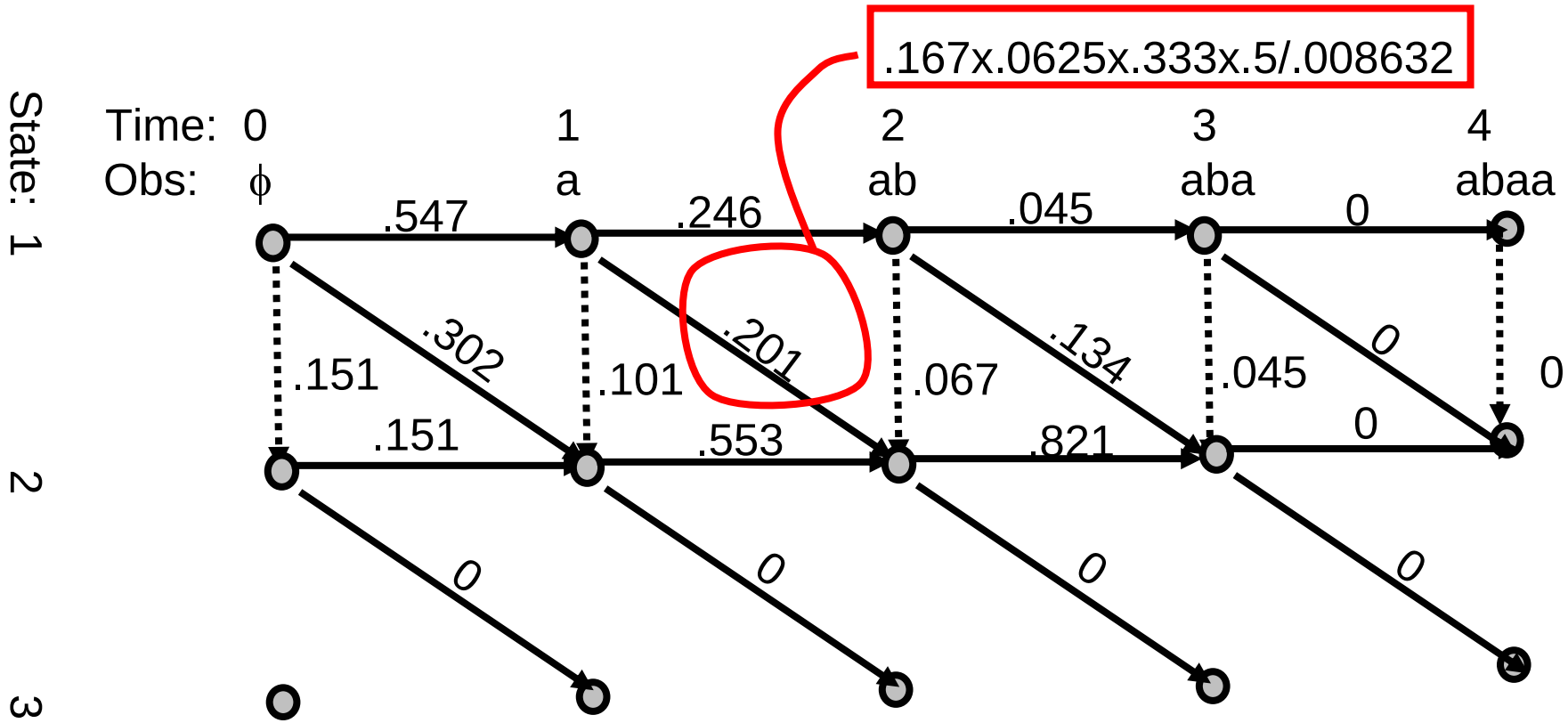
Compute β's.



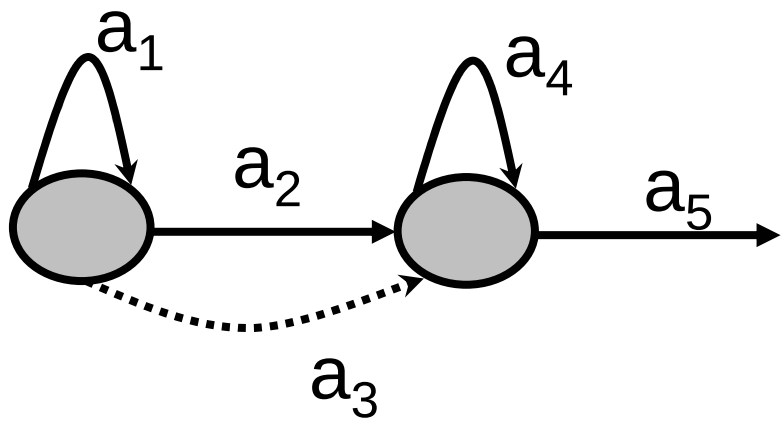
Problem 3: F-B algorithm, cont'd

Compute counts. (a posteriori probability of each transition)

$$c_t(tr_{ij}|X) = \alpha_{t-1}(i)\, a_{ij}\, b_{ij}(x_t)\ \ \beta_t(j)/\Pr(X)$$



Problem 3: F-B algorithm cont'd

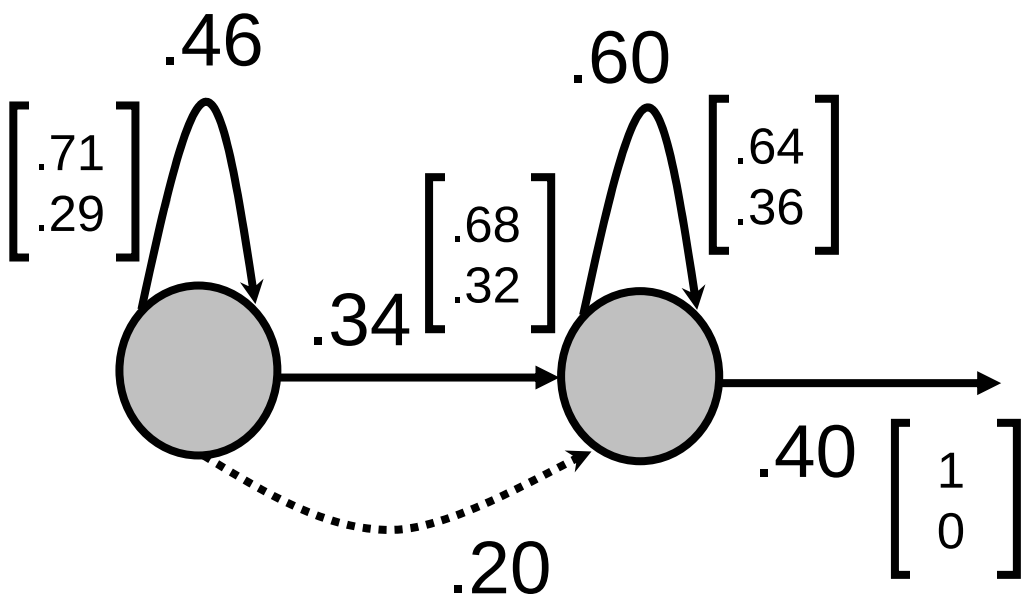


- $C(a_1)$=.547+.246+.045
- $C(a_2)$=.302+.201+.134
- $C(a_3)$=.151+.101+.067+.045
- $C(a_4)$=.151+.553+.821
- $C(a_5)$=1

- C(a1,'a')=.547+.045, C(a1,'b')=.246
- C(a2,'a')=.302+.134, C(a2,'b')=.201
- C(a4,'a')=.151+.821, C(a4,'b')=.553
- C(a5,'a')=1, C(a5,'b')=0

Problem 3: F-B algorithm cont'd

Normalize counts to get new parameter values.



Result is the same as from the enumerative algorithm!!

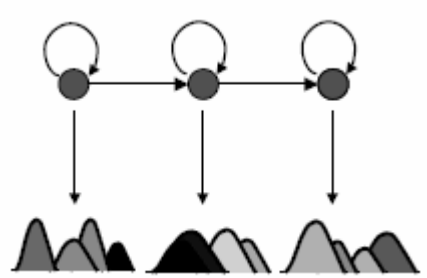
Continuous Hidden Markov models – Parameterization

Continuous Hidden Markov models (HMMs) replace the discrete observation probabilities by a continuous *pdf* and have 3 sets of parameters.

1. A prior distribution over the states $\pi = P(s_0 = j);\ j = 1 \ldots N$

2. Transition probabilities between the states,
   $a_{ij} = P(s = j \mid s' = i);\ i, j = 1 \ldots N$

3. A set of transition-conditioned observation probabilities,
   $b_{ij}(x_t) = P(x_t \mid s = j, s' = i)$



Let's use the mixture of Gaussians we discussed in the previous lecture:

$$b_{ij}(x_t) = \sum_{m=1}^{M_{ij}} b_{ijm}(x_t) = \sum_{m=1}^{M_{ij}} \frac{p_{ijm}}{(2\pi)^{n/2} \mid \Sigma_{ijm} \mid^{1/2}} e^{-\frac{1}{2}(x_t - \mu_{ijm})^T \Sigma_{ijm}^{-1} (x_t - \mu_{ijm})}$$

Continuous Hidden Markov Models – The Three Problems

- The forward pass and Viterbi algorithm can be treated as before:

$$\alpha_t(s) = \Sigma_{s'} \alpha_{t-1}(s') \; P_\theta(s|s') \; P_\theta(x_t|s'->s) \quad + \quad \Sigma_{s'} \alpha_t(s') \; P_\theta(s|s')$$

1st term: sum over all output producing arcs 2nd term: all null arcs

- The forward-backward algorithm is still computed as before, where the posteriori probability of each transition:

$$c_t(tr_{ij}|X) = \alpha_{t-1}(i) \; a_{ij} \; b_{ij}(x_t) \; \beta_t(j) \; / \; Pr(X)$$

and $b_{ij}(x_t)$ is the Gaussian mixture likelihood.

- The count for the m^{th} Gaussian on transition tr_{ij} can be computed as

$$c_t(ijm|X) = \alpha_{t-1}(i) \; a_{ij} \; b_{ijm}(x_t) \; \beta_t(j) \; / \; Pr(X)$$

- So the mean of the m^{th} Gaussian on transition t_{ij} would be

$$\mu_{ijm} = \frac{\sum_t c_t(ijm \mid X) \, x_t}{\sum_t c_t(ijm \mid X)}$$

and similarly for the variance/covariance matrix

Parameter Tying

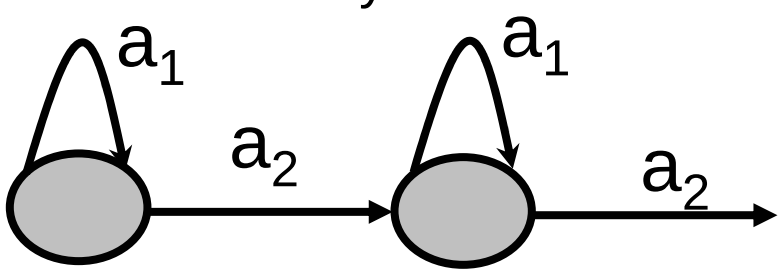
- Represent word as sequence of speech sounds:

  cat: K AE T

  bat: B AE T

  nap: N AE P

- Forward pass/Viterbi is easy:



- Forward-backward algorithm:
  - Map transition ij to index k, where k is a unique identifier (e.g., number of the speech sound)
  - Establish set of accumulators on k rather than transition ij

Practical Matters – Dynamic Range Issues

- Typical Gaussian likelihood for 13-dimensional Gaussian
- Forward pass calculation:

$$\alpha_t(s) = \Sigma_{s'} \alpha_{t-1}(s') \; P_\theta(s|s') \; P_\theta(x_t|s'->s) \quad + \quad \Sigma_{s'} \; \alpha_t(s') \; P_\theta(s|s')$$

so likelihood of 100 frame utterance on the order of 10^{-600}

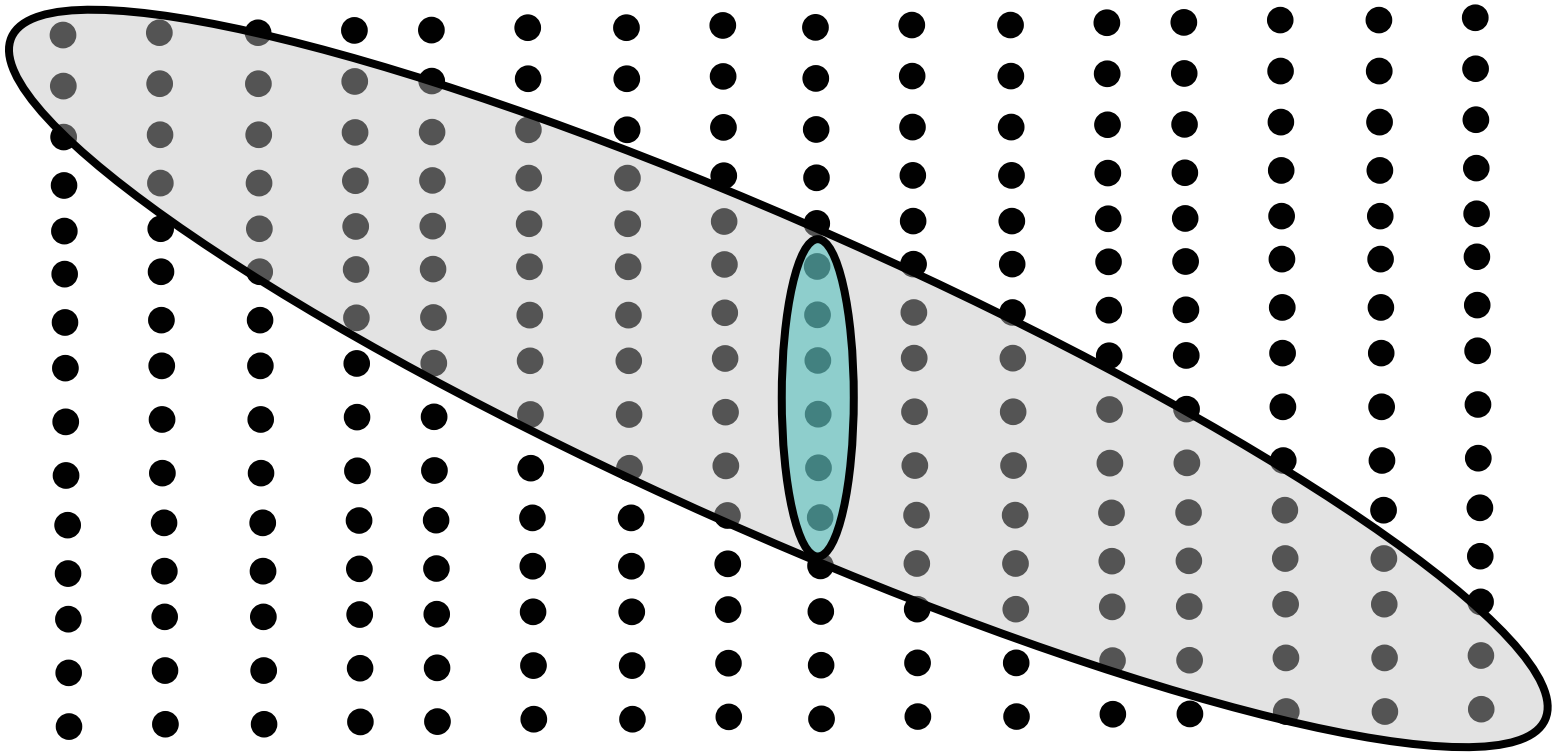
- Clear issue if calculations done in linear likelihood/probability domain

Practical Matters – Dynamic Range Issues

- Solution: Do calculation in log domain
- Let L_1 , L_2 and L_3 be likelihoods and let $x = \ln_b L_1$, $y = \log_b L_2$, $z = \log_b L_3$
- For the case when we need to compute $L_3 = L_1 L_2$ $z = x + y$. Wonderful!
- But what about $L_3 = L_1 + L_2$?
- Assume $L_1 > L_2$
- $L_1 + L_2 = b^x + b^y = b^x \ (1+b^{y-x})$
- $z = x + \log_b (1 + b^{y-x})$
- If $L_1 >> L_2$ (say 5 orders of magnitude) then set $z = x$
- If we choose b = 1.001, then (y-x) ~ 1000 log (L_2 / L_1), so a difference of five orders of magnitude can be stored in a table of size roughly 10000 with direct lookup of entries of round (x-y)
- So for addition:

```
If x > y
    if x > y + 11000 return x
    else return x + table [x-y]
Else
    if y > x + 11000 return y
    else return y + table[y-x]
```

Practical Matters: Thresholding



- The large fraction of the counts are in an ellipse along the trellis diagonal
- Can therefore threshold the αs in the forward pass.
- Let T =sum of the live αs in each timeslice. Do not extend α to next timeslice if $\alpha < k\ T$.

Other Practical Matters

- Since you only need to save the αs in a narrow band, you don't need to keep an N (number of states) x M (time) trellis in memory. You just need to store two timeslices of N states and keep swapping them, and save the αs (in the F-B algorithm). Much less storage.

- Keep track of the total logprob of the data. It MUST increase after each iteration or you did not program the algorithm correctly – a nice property of the F-B algorithm.

Sketch of Proof of Convergence of F-B Algorithm

- Goal is to find set of parameters θ that maximize $P_\theta(X)$.
- If we knew the actual transition sequence t_{ij} this would be easy, as discussed earlier.
- What we do instead is to assume a θ , estimate $P_\theta(t_{ij}|X)$ and use this to compute a new θ
- Why does this converge?

Sketch of Proof – Jensen's Inequality

- Say we have two probability distributions, p(x) and q(x)
- Definition: Expectation of a convex transformation of a random variable (q(x)/p(x)) is less than equal to the convex transformation of the expectation of the random variable
- Σ_x p(x) log (q(x) / p(x)) <= log Σ_x p(x) q(x) / p(x) <= 0
- Σ_x p(x) log q(x) - Σ_x p(x) log p(x)) <= 0
- The inequality $-\Sigma_x$ p(x) log q(x) >= $-\Sigma_x$ p(x) log p(x) [= H(p(x))] is known as **Jensen's Inequality**

Sketch of Proof

- Start with set of parameters θ, want to find a θ‘ that increases the likelihood of the data
- $\log P_{\theta'}(X) = \log P_{\theta'}(X,T) - \log P_{\theta'}(T|X)$ (Bayes’ Law)

  T a set of transitions (i.e., a path) through the lattice
- Multiply by $P_{\theta}(T|X)$ and sum over all paths T:

  $\Sigma_T P_{\theta}(T|X) \log P_{\theta'}(X) = \Sigma_T P_{\theta}(T|X) \log P_{\theta'}(X,T) - \Sigma_T P_{\theta}(T|X) \log P_{\theta'}(T|X)$

  $\log P_{\theta'}(X) = Q(\theta,\theta') - H(\theta,\theta')$
- If we can find a θ‘ such that $Q(\theta,\theta') > Q(\theta,\theta)$ then by Jensen’s inequality, $H(\theta,\theta')$ will also decrease.
- But what is $Q(\theta,\theta')$? It is just the weighted sum of the path likelihoods with respect to the new parameters θ. We can therefore collect terms and maximize as in ordinary ML estimation. Note the likelihood of the data is guaranteed to increase after each iteration.
- The two steps – the “Estimation” of the Q function, and its’ Maximization – is called the “E-M” algorithm and the proof can be generalized to a wide variety of cases in which there is a set of hidden variables whose values, if known, would simplify ML estimation.

The EM algorithm

An expectation-maximization (EM) algorithm is used in statistics for finding maximum likelihood estimates of parameters in probabilistic models, where the model depends on unobserved hidden variables.

EM alternates between performing an expectation (E) step, which computes an expectation of the likelihood by including the latent variables as if they were observed, and a maximization (M) step, which computes the maximum likelihood estimates of the parameters by maximizing the expected likelihood found on the E step. The parameters found on the M step are then used to begin another E step, and the process is repeated.

The EM algorithm was explained and given its name in a classic 1977 paper by A. Dempster and D. Rubin in the Journal of the Royal Statistical Society.

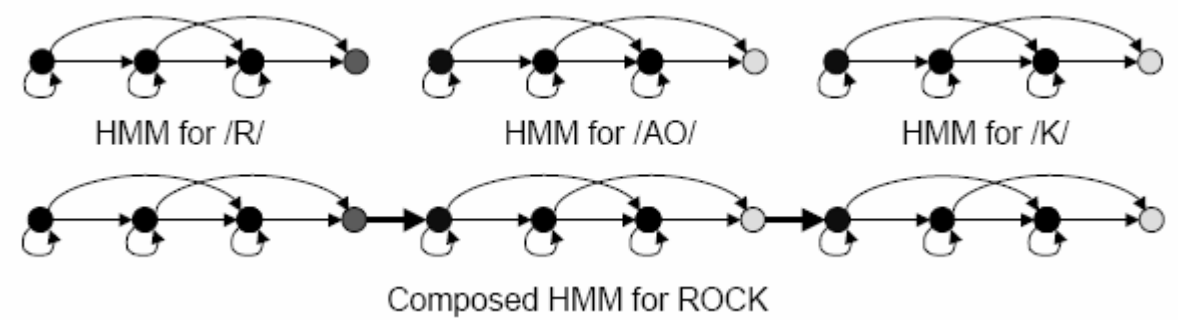
The Baum-Welch algorithm

The Baum-Welch algorithm is a generalized expectation-maximization algorithm for computing maximum likelihood estimates for the parameters of a Hidden Markov Model when given only observations as training data.

It is a special case of the EM algorithm for HMMs.

HMMs in Speech Recognition

- Each phoneme represented by its own context-independent HMM



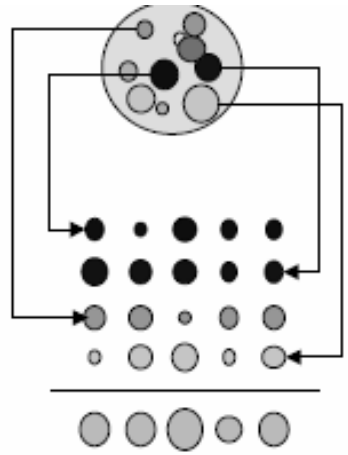
- Training requires grouping data from each subword unit followed by parameter estimation
- For a 5-state HMM:

  Segment data from each instance of sub-word unit into 5 parts

  Aggregate all data from corresponding parts

  Estimate parameters for each of the aggregates

  Repeat



Summary of Markov Modeling Basics

- Key idea 1: States for modeling sequences

  Markov introduced the idea of state to capture the dependence on the past (time evolution). A state embodies all the relevant information about the past. Each state represents an equivalence class of pasts that influence the future in the same manner.

- Key idea 2: Marginal probabilities

  To compute Pr(X), sum up over all of the state sequences than can produce X

  $$\Pr(X) = \sum_s \Pr(X,S)$$

  For a given S, it is easy to compute Pr(X,S)

- Key idea 3: Trellis

  The trellis representation is a clever way to enumerate all sequences. It uses the Markov property to reduce exponential-time enumeration algorithms to linear-time trellis algorithms.

Summary (…contd.)

- We introduced the Forward-Backward algorithm for the maximum likelihood estimation of model parameters
- Discussed it from an intuitive point (counting) of view and information-theoretic (Q-function) point of view