ECE8813
Statistical Natural Language Processing

Lectures 12-13: Hidden Markov Models

Chin-Hui Lee
School of ECE, Georgia Tech
Atlanta, GA 30332, USA
chl@ece.gatech.edu

Markov Assumptions

- Let $X=(X_1, .., X_T)$ be a sequence of random variables taking values in some finite set, $S=\{s_1, \ldots, s_N\}$, the state space. If X possesses the following properties, then X is a Markov Chain

1. **Limited Horizon:**

   $P(X_{t+1}=s_k|X_1, .., X_t)=P(X_{t+1} = s_k|X_t)$ i.e., a word's state only depends on the previous state

2. **Time Invariant (Stationary):**

   $P(X_{t+1}=s_k|X_t)=P(X_2 =s_k|X_1)$ i.e., the dependency does not change over time

Definition of a Markov Chain

A is a stochastic *N x N* matrix
of the probabilities of transitions with:

a_{ij} = Pr{transition from s_i to s_j} = Pr($X_{t+1}=s_j \mid X_t=s_i$)

$\forall i, j, a_{ij} \geq 0$, and $\forall t$:

$$\sum_{j=1}^{n} a_{ij} = 1$$

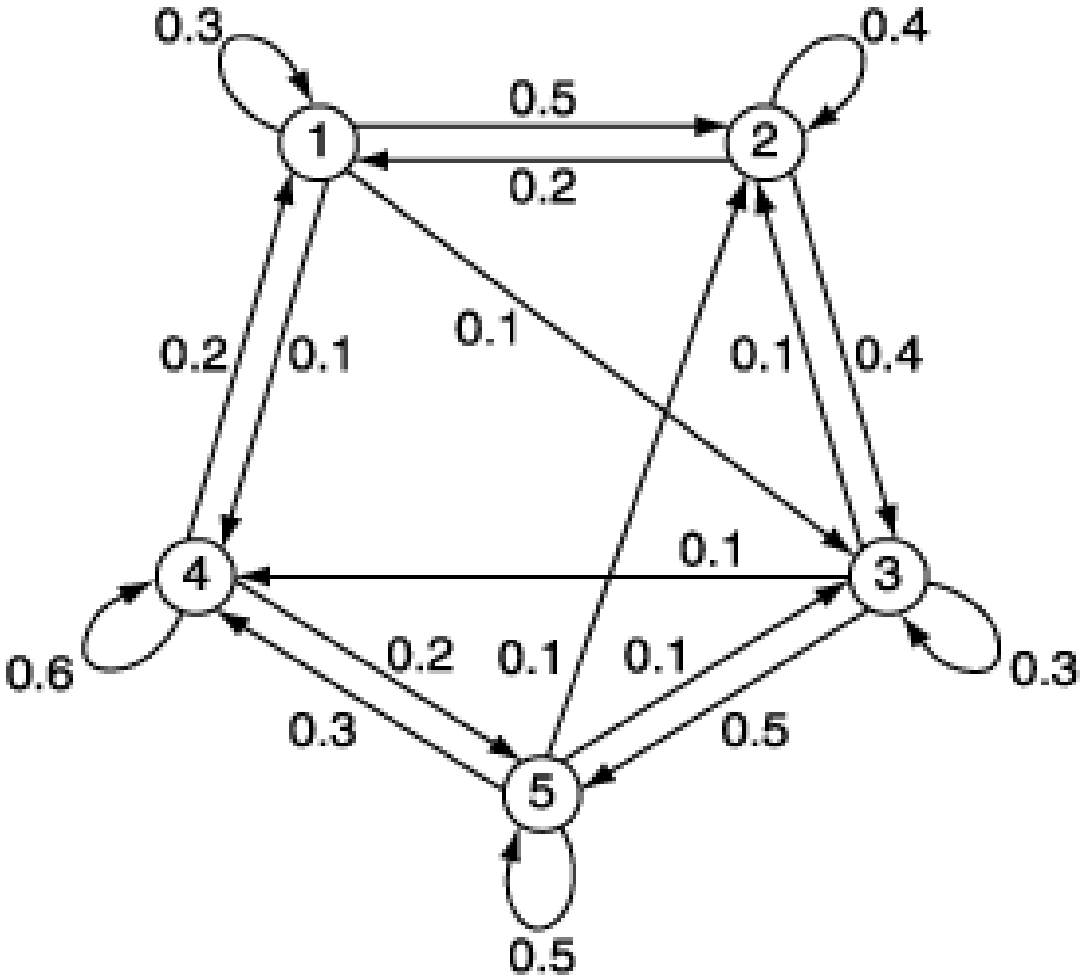
Π is a vector of *N* elements representing the initial state probability distribution with:

π_j = Pr{probability that the initial state is s_i} = Pr($X_1=s_i$) $\forall i$

$$\sum_{i=1}^{n} \pi_i = 1$$

Can avoid this by creating a special start state s_0

Markov Chain: An Example



Markov Process & Language Models

- Bayes formula (chain rule):

  $P(W) = P(w_1,w_2,\ldots,w_T) = P_{i=1..T}\, p(w_i|w_1,w_2,..,w_{i-n+1},..,w_{i-1})$

- *n*-gram language models:

  Markov process (chain) of the order *n*-1:

$P(W) = P(w_1,w_2,\ldots,w_T) = \Pi_{i=1..T}\, p(w_i|w_{i-n+1},w_{i-n+2},..,w_{i-1})$

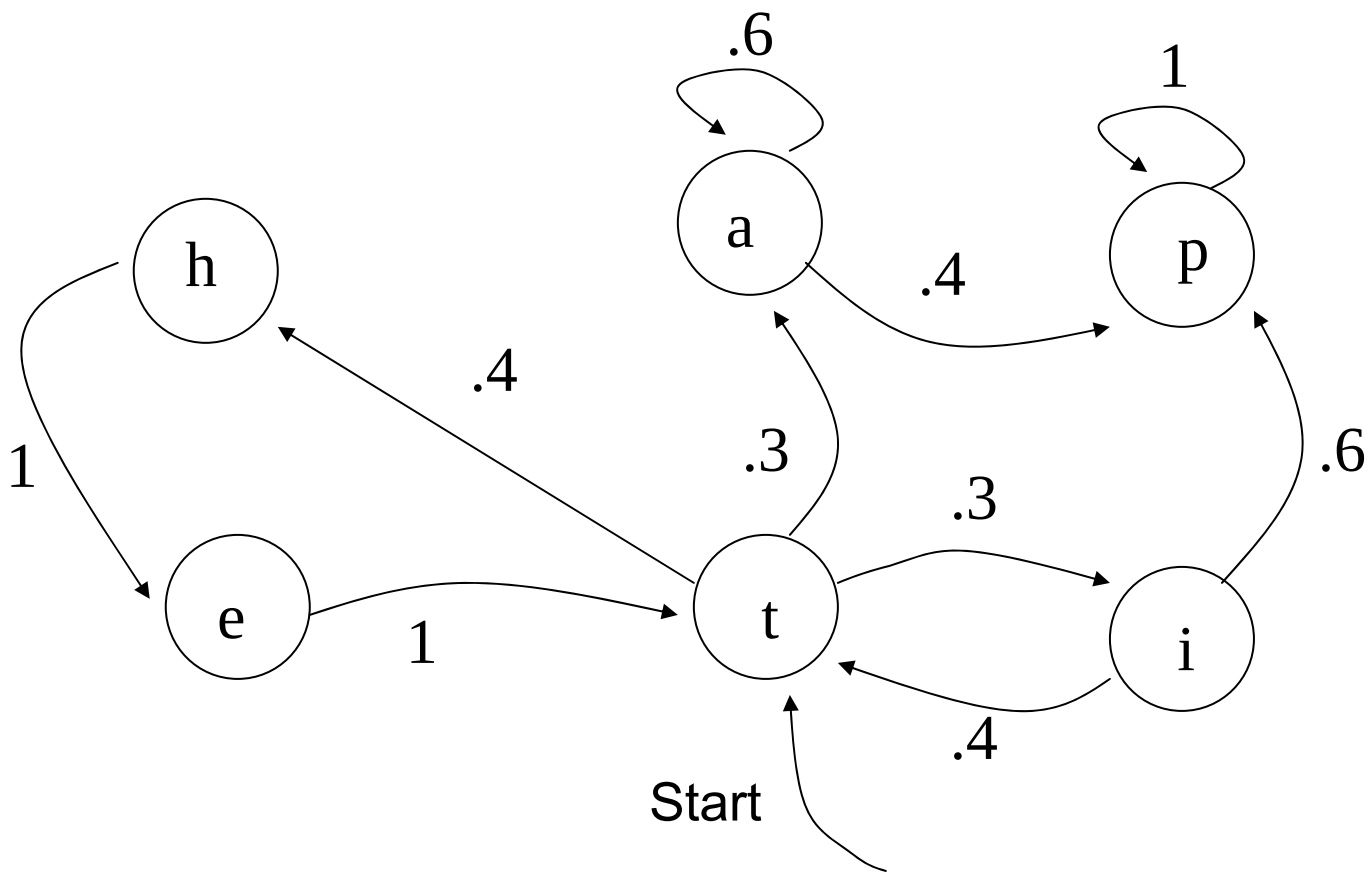
Using just one distribution (Ex.: trigram model: $p(w_i|w_{i-2},w_{i-1})$):

Positions: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Words: My car **broke down ,** and within hours Bob 's car **broke down ,** too .

$p(,|\text{broke down}) = p(w_5|w_3,w_4)) = p(w_{14}|w_{12},w_{13})$

Another Example with a Markov Chain



Probability of a Sequence of States

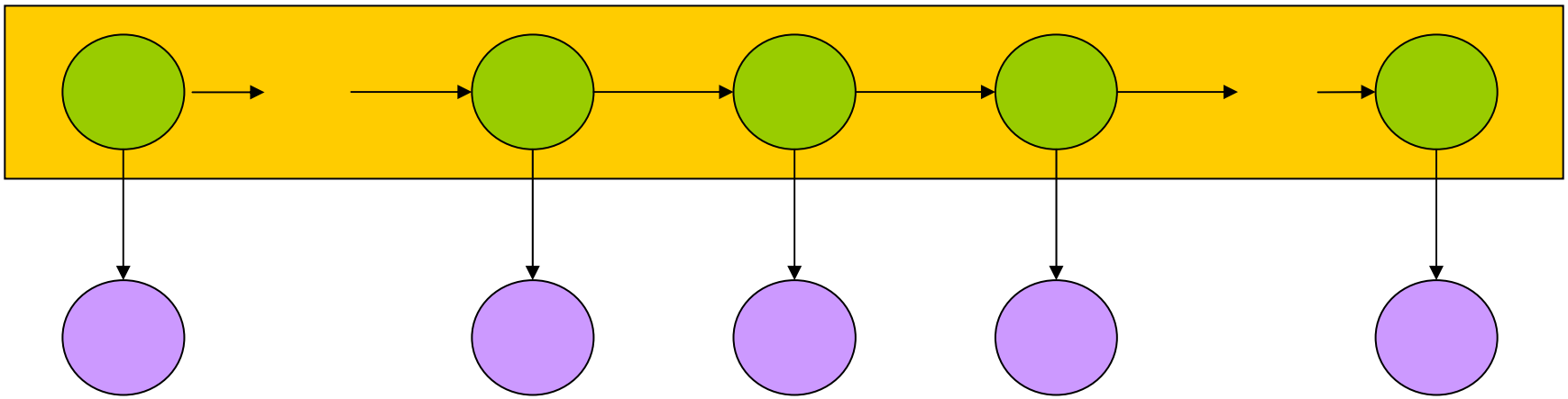
$$P(x_1 x_2 x_3 \ldots x_T) = P(x_1) \times P(x_2 \mid x_1) \times \ldots \times P(x_T \mid x_1 x_2 x_3 \ldots x_{T-1})$$
$$= P(x_1) \times P(x_2 \mid x_1) \times \ldots \times P(x_T \mid x_{T-1})$$
$$= \pi_{x_1} \prod_{t=1}^{T-1} a_{x_t x_{t+1}}$$

$$P(\text{t, a, p, p}) = 1.0 \times 0.3 \times 0.4 \times 1.0 = 0.12$$

Hidden Markov Models (HMMs)

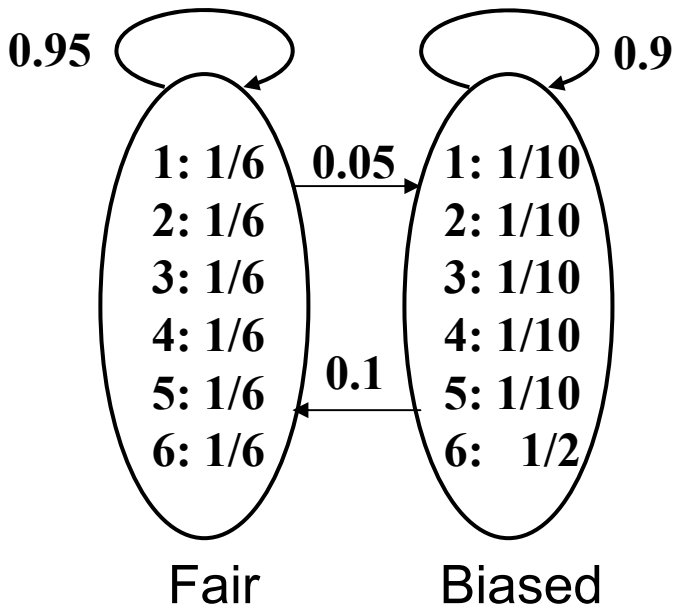
- Sometimes it is not possible to know precisely which states the model passes through; all we can do is to observe some phenomena that occurs when in that state with some probability distribution
- An HMM is an appropriate model for cases when you don't know the state sequence that the model passes through, but only some probabilistic function of it. For example:
  - Word recognition (discrete utterance or within continuous speech)
  - Phoneme recognition
  - Part-of-Speech Tagging
  - Linear Interpolation

What is an HMM?



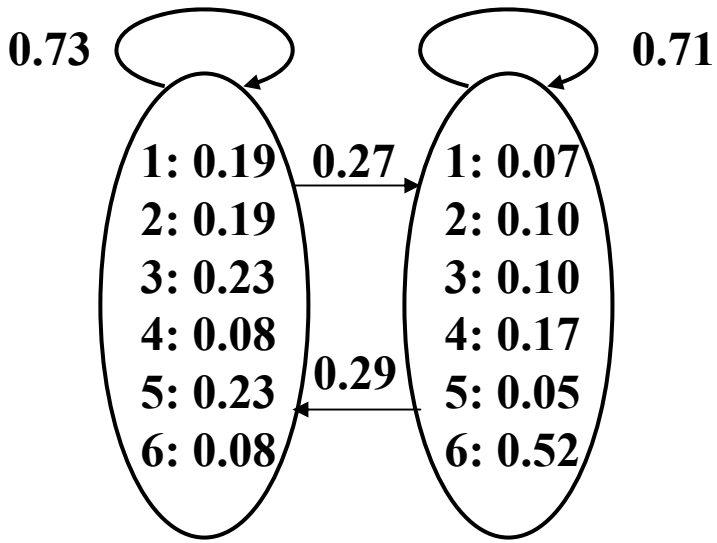
- Green circles are *hidden states with its status d*ependent only on the previous state
- Purple circles are observed events with their observations depend only of their emitting states

HMM: An Occasionally Dishonest Casino

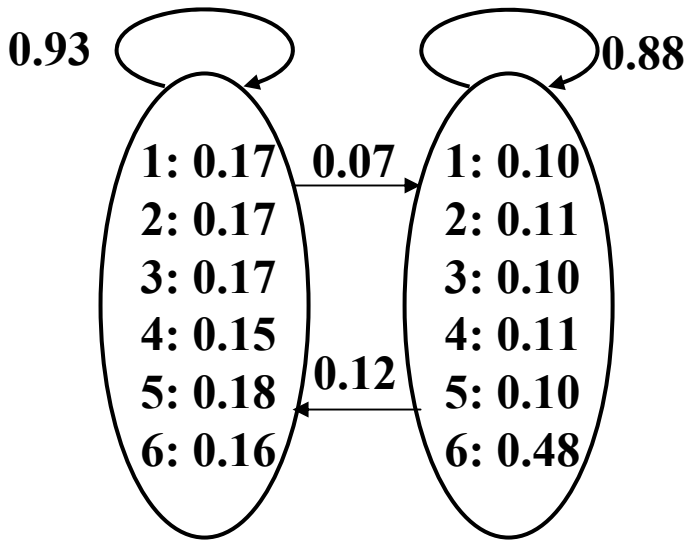


- Assume: A casino switches occasionally to a biased dice to increase winning odds !!
- Can we model it with HMM ?
- How do we prove it cheats ?
- Can we estimate the HMM ?
- How many samples needed ?
- Which dice used at what time?

Estimation: More vs. Less Data

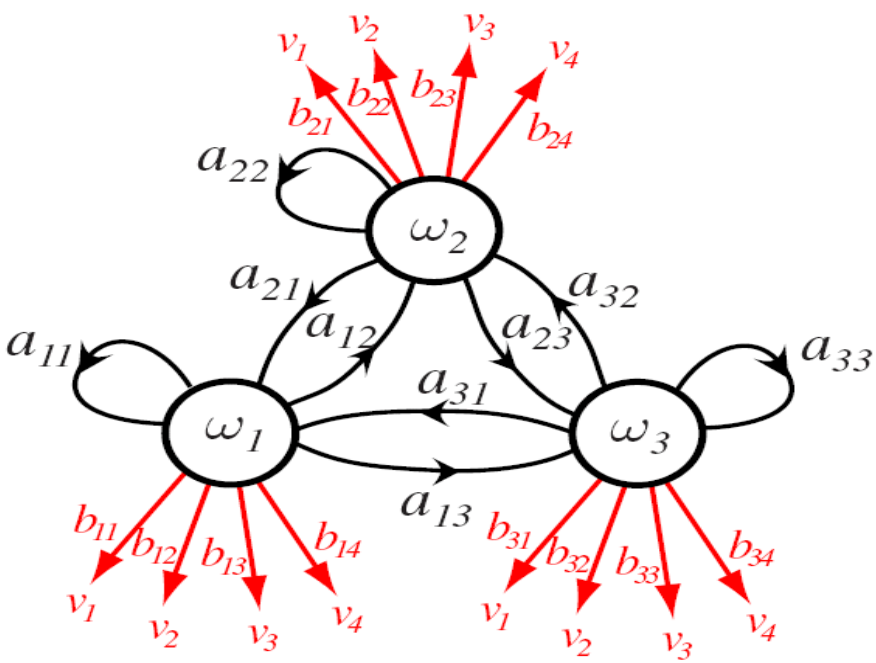


Estimates with 300 rolls



Estimates with 30000 rolls

Observations and Hidden States

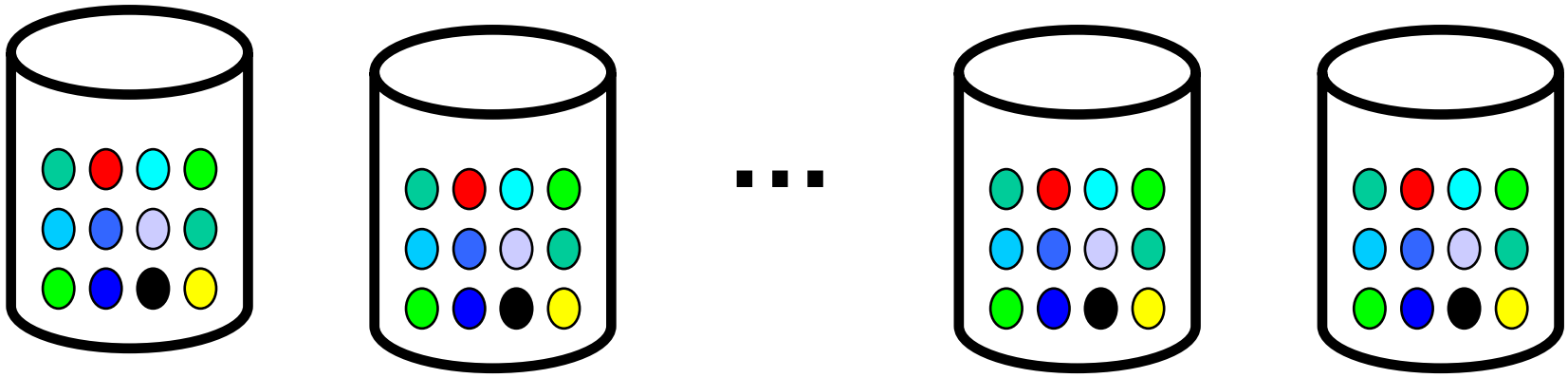


S = ω1, ω4, ω2, ω2, ω1, ω4 (hidden)

O = v4, v1, v1, v4, v2, v3 (observed)

- HMM is also called a probabilistic function of a Markov chain
  - State transition follows a Markov chain
  - In each state, it generates observation symbols based on a probability function. Each state has its own probability function
  - HMM is a doubly embedded stochastic process
- In HMM,
  - State is not directly observable (hidden states)
  - Only observe observation symbols generated from states

An HMM Example: Urn & Ball



Urn 1

Pr(RED) = $b_1(1)$
Pr(BLE) = $b_1(2)$
Pr(GRN) = $b_1(3)$
...

Urn 2

Pr(RED) = $b_2(1)$
Pr(BLE) = $b_2(2)$
Pr(GRN) = $b_2(3)$
…

Urn N-1

Pr(RED) = $b_{N-1}(1)$
Pr(BLE) = $b_{N-1}(2)$
Pr(GRN) = $b_{N-1}(3)$
…

Urn N

Pr(RED) = $b_N(1)$
Pr(BLE) = $b_N(2)$
Pr(GRN) = $b_N(3)$
…

Observation: O = { GRN, GRN, BLE, RED, RED, … BLE}

Elements of an HMM

- HMM (the most general case):

  S five-tuple (S, K, Π, A, B), where:

  S = $\{s_1, s_2, ..., s_N\}$ is the set of states,

  K = $\{k_1, k_2, ..., k_M\}$ is the output alphabet,

  $\Pi = \{\pi_i\}$, $i \in S$,

  $A = \{a_{ij}\}$, $i,j \in S$,

  $B = \{b_{ijk}\}$, $i,j \in S$, $k \in K$ (arc emission)

  $B=\{b_{ik}\}$ $i \in S$, $k \in K$ (state emission)

- State Sequence: $X=(x_1, x_2, .., x_{T+1})$, $x_t : S \rightarrow \{1,2,\ldots, N\}$
- Observation Sequence: $O=(o_1, .., o_T)$, $o_t \in K$

HMM as a Generating Model

- Given an HMM, denoted as $\Lambda = \{A, B, \pi\}$ and an observation sequence $O=\{O_1, O_2, \ldots, O_T\}$
- The HMM can be viewed as a generator to produce O as:
  1. Choose an initial state $q_1=S_i$ according to the initial probability distribution π
  2. Set $t=1$
  3. Choose an observation Ot according to the symbol observation probability distribution in state S_i, i.e., $b_i(k)$
  4. Transit to a new state $q_{t+1} = S_j$ according to the state transition probability distribution, i.e., a_{ij}
  5. Set $t=t+1$, return to step 3 if $t<T$
  6. Terminate the procedure

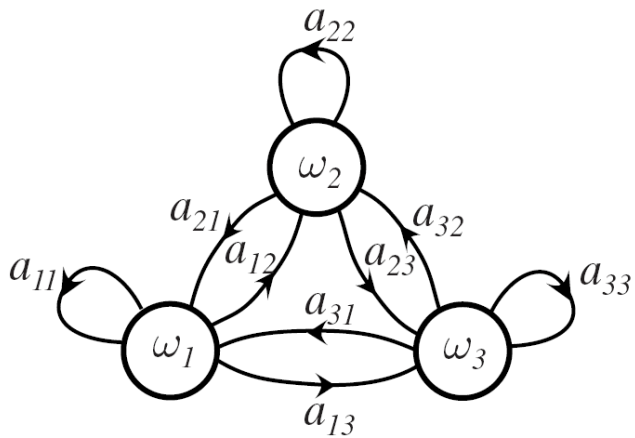
Assumptions in HMM

- Markov Assumption:
  - State transition follows a 1st-order Markov chain
  - This assumption implies the duration in each state is a binomial distribution: $p_j(d) = (a_{jj})^{d-1}(1-a_{jj})$
- Output Independence Assumption: the probability that a particular observation symbol is emitted from HMM at time *t* depends only on the current state *st* and is conditionally independent of the past and future observations
- The two assumptions limit the memory of an HMM and may lead to model deficiency. But they significantly simplify HMM computation, also greatly reduce the number of free parameters to be estimated in practice
  - Some research works to relax these assumptions has been done in the literature to enhance HMM in modeling speech signals

Types of HMMs (I)

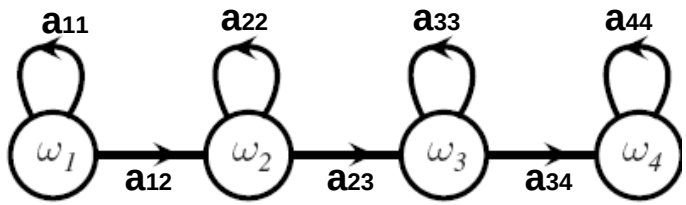
- Different transition matrices:
  - Ergodic HMM Topology

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$



  - Left-to-right HMM: states proceed from left to right

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$



Types of HMMs (II)

- Different observation symbols: discrete vs. continuous
  - Discrete density HMM (DDHMM): observation is discrete, one of a finite set. In discrete density HMM, observation function is a discrete probability density, i.e., a table. In state *j,*

$$O_j(k) = \begin{matrix} v_1 & v_2 & v_3 & v_4 \\ [0.1 & 0.4 & 0.3 & 0.2] \end{matrix}$$

  - Continuous density HMM (CDHMM): observation *x* is continuous in an observation space. In CDHMM, state observation density is a probability density function (p.d.f.). The common function forms:
    - Multivariate Gaussian density

$$N(\mathbf{x};\boldsymbol{\mu},\mathbf{C}) = \frac{1}{\sqrt{(2\pi)^n\,|\mathbf{C}|}}\,\mathrm{e}^{-(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{t}}\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu})/2} \qquad -\infty < \mathbf{x} < \infty$$

    - Gaussian mixture density

$$MG(x) = \sum_{m=1}^{M} \omega_m N(x;\mu_m,\sigma_m^2) \quad \sum_{m=1}^{M} \omega_m = 1 \quad 0 \le \omega_m \le 1 \quad \sigma_i > 0$$

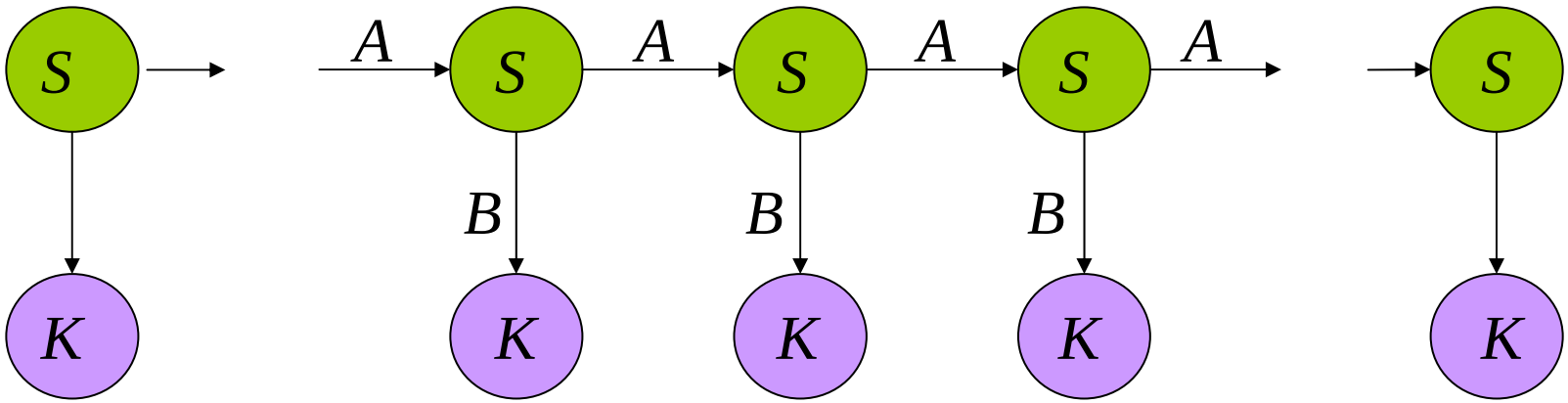
HMM for Data Modeling

- HMM is used as a powerful statistical model for sequential and temporal observation data
- HMM is theoretically (mathematically) sound; relatively simple learning and decoding algorithms exist
- HMM is widely used in pattern recognition, machine learning, etc.
  - Speech recognition: modeling speech signals
  - Statistical language processing: modeling language (word/semantics sequence)
  - OCR (optimal character recognition): modeling 2-d character image
  - Gene finding: modeling DNA sequence

Three Fundamental Problems in HMM

- How to use HMM to model sequential data ?
  - The entire data sequence is viewed as one data sample *O*
  - The HMM is characterized by its parameters $\Lambda = \{A, B, \pi\}$
- Learning Problem: HMM parameters Λ must be estimated from a data sample set $\{O_1, O_2, \ldots, O_T\}$
  - The HMM parameters are set so as to best explain known data
- Evaluation Problem: for an unknown data sample O_x, calculate the probability of the data sample given the model, $p(O_x|\Lambda)$
- Decoding Problem: uncover the hidden information; for an observation sequence $O=\{o_1, o_2, \ldots, o_T\}$, decode a best state sequence $S=\{q_1, q_2, \ldots, q_T\}$ which is optimal in explaining *O*

HMM Formalism



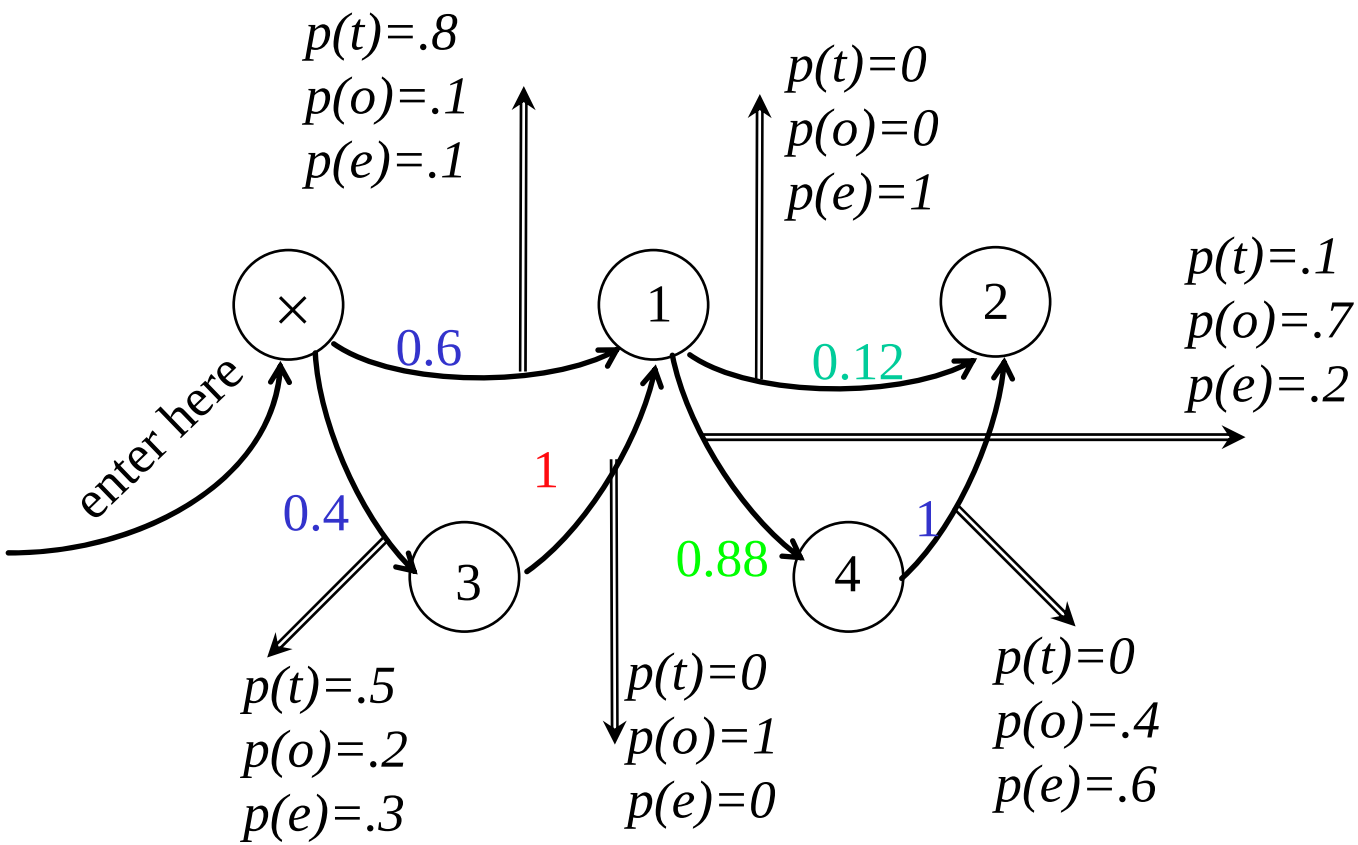
{*S*, *K*, P, *A*, *B*}

$\Pi = \{p_i\}$ are the initial state probabilities

$A = \{a_{ij}\}$ are the state transition probabilities

$B = \{b_{ik}\}$ are the observation probabilities

Example of an Arc Emit HMM



HMM Properties

- N states in the model: a state has some measurable, distinctive behaviors
- At clock time *t*, you make a state transition (to a different state or back to the same state), based on a transition probability distribution that depends on the current state (i.e., the one you're in before making the transition)
- After each transition, you output an *observation output symbol* according to a probability density distribution which depends on the current state or the current arc

➢ Goal: From the observations, determine what model generated the output, e.g., in word recognition with a *different* HMM for each word, the goal is to choose the one that *best* fits the input word (based on state transitions and output observations)

Example HMM

- N states corresponding to urns: q_1, q_2, …,q_N
- M colors for balls found in urns
- B: N x M matrix; for each urn, there is a probability density function for each color.
  - b_{ij}={COLOR=z_j | URN = q_i)
- A: N x N matrix; transition probability distribution (between urns)
- $\prod$: N vector; initial sate probability distribution (where do we start?)

An HMM Evolution Example

- Process:
  - According to π, pick a state q_i. Select a ball from urn i (state is hidden)
  - Observe the color (observable)
  - Replace the ball
  - According to *A*, transition to the next urn from which to pick a ball (state is hidden)
- Design: Choose *N* and *M*. Specify a model $\Lambda = (A, B, \Pi)$ from the training data. Adjust the model parameters to maximize $P(O|\Lambda)$
- Use: Given *O* and $\Lambda = (A, B, \Pi)$, what is $P(O|\Lambda)$? If we compute $P(O|\Lambda)$ for all models Λ, then we can determine which model most likely generated *O*

Simulating a Markov Process

$t := 1$;

Start in state s_i with probability π_i (i.e., $X_1 = i$)

Forever do

Move from state s_i to state s_j with probability a_{ij} (i.e., $X_{t+1} = j$)

Emit observation symbol $o_t = k$ with probability b_{ijk}

$t := t+1$

End

Why Use Hidden Markov Models?

- HMMs are useful when one can think of underlying events probabilistically generating surface events. Example: PoS tagging
- HMMs can be efficiently trained using the EM Algorithm
- Another example where HMMs are useful is in generating parameters for linear interpolation of *n*-gram models
- Assuming that some set of data was generated by a HMM, and then an HMM is useful for calculating the probabilities for possible underlying state sequences

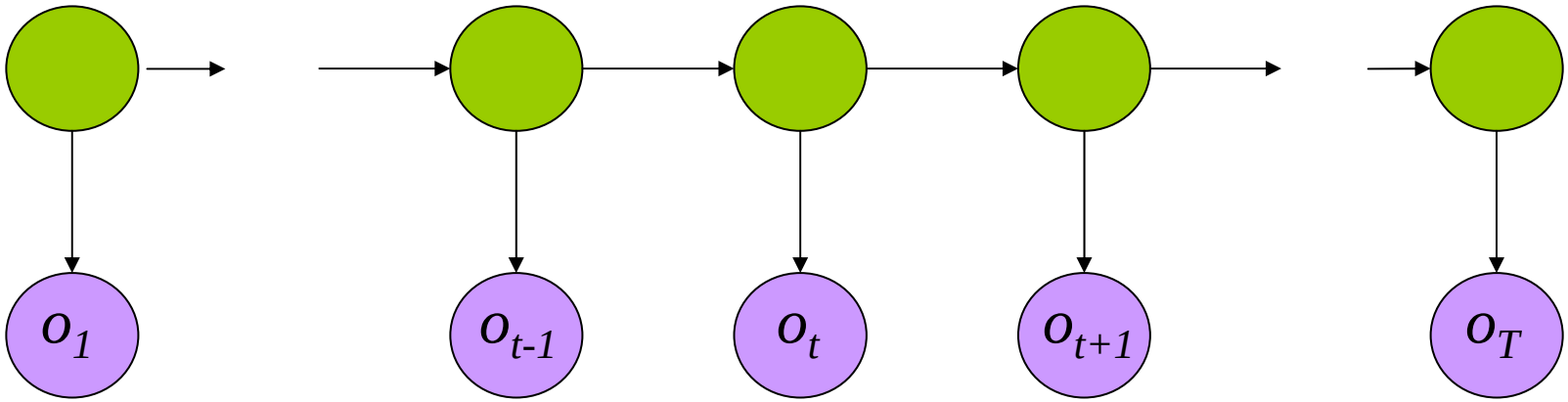
Fundamental Questions for HMMs

- Given a model $\Lambda = (A, B, \Pi)$, how do we efficiently compute how likely a certain observation is, that is, $P(O|\Lambda)$?
- Given an observation sequence O and a model Λ, how do we choose a state sequence $(X_1, \ldots, X_{T+1})$ that best explains the observations?
- Given an observation sequence O, and a space of possible models found by varying the model parameters $\Lambda = (A, B, \pi)$, how do we find the model that best explains the observed data?

Probability of an Observation

- Given the observation sequence $O=(o_1, \ldots, o_T)$ and a model, $\Lambda = (A, B, \Pi)$, we wish to know how to efficiently compute $P(O|\Lambda)$
- For any state sequence, $S=(q_1, \ldots, q_{T+1})$, we find: $P(O|\Lambda)=\Sigma_S P(O|S,\Lambda) P(S|\Lambda)$ *which* is simply the probability of an observation sequence given the model
- Direct evaluation of this expression is extremely inefficient; however, there are dynamic programming methods that compute it quite efficiently

Probability of an Observation

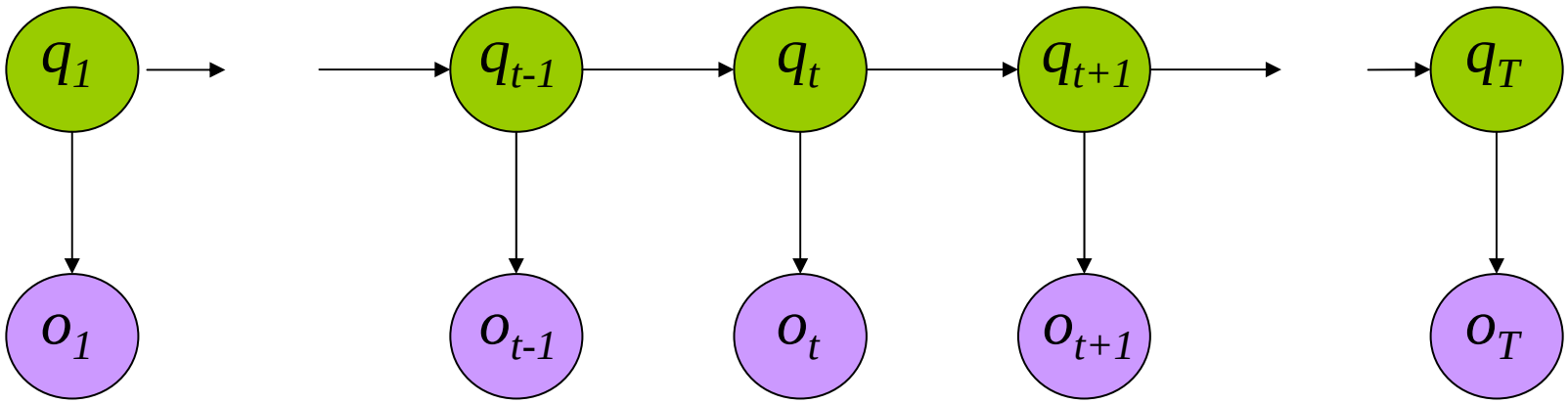


Given an observation sequence and a model, compute the probability of the observation sequence

$$O = (o_1 \ldots o_T), \mu = (A, B, \Pi)$$

Compute $P(O \mid \mu)$

Probability of an Observation (Cont.)



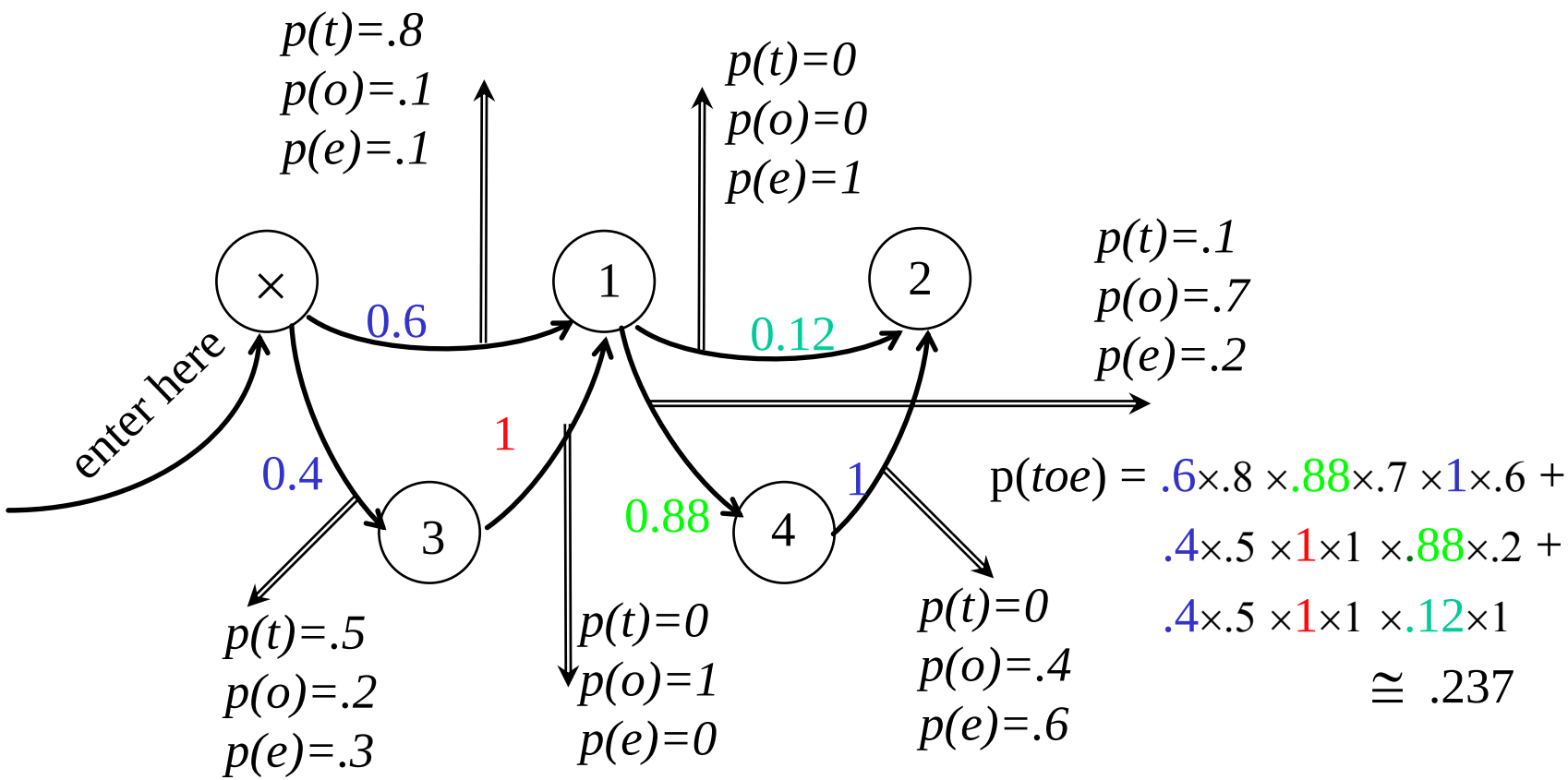
$$P(O \mid S, \mu) = b_{q_1 o_1} b_{q_2 o_2} \ldots b_{q_T o_T}$$

$$P(S \mid \mu) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \ldots a_{q_{T-1} q_T}$$

$$P(O, S \mid \mu) = P(O \mid S, \mu) P(S \mid \mu)$$

$$P(O \mid \mu) = \sum_S P(O \mid S, \mu) P(S \mid \mu)$$

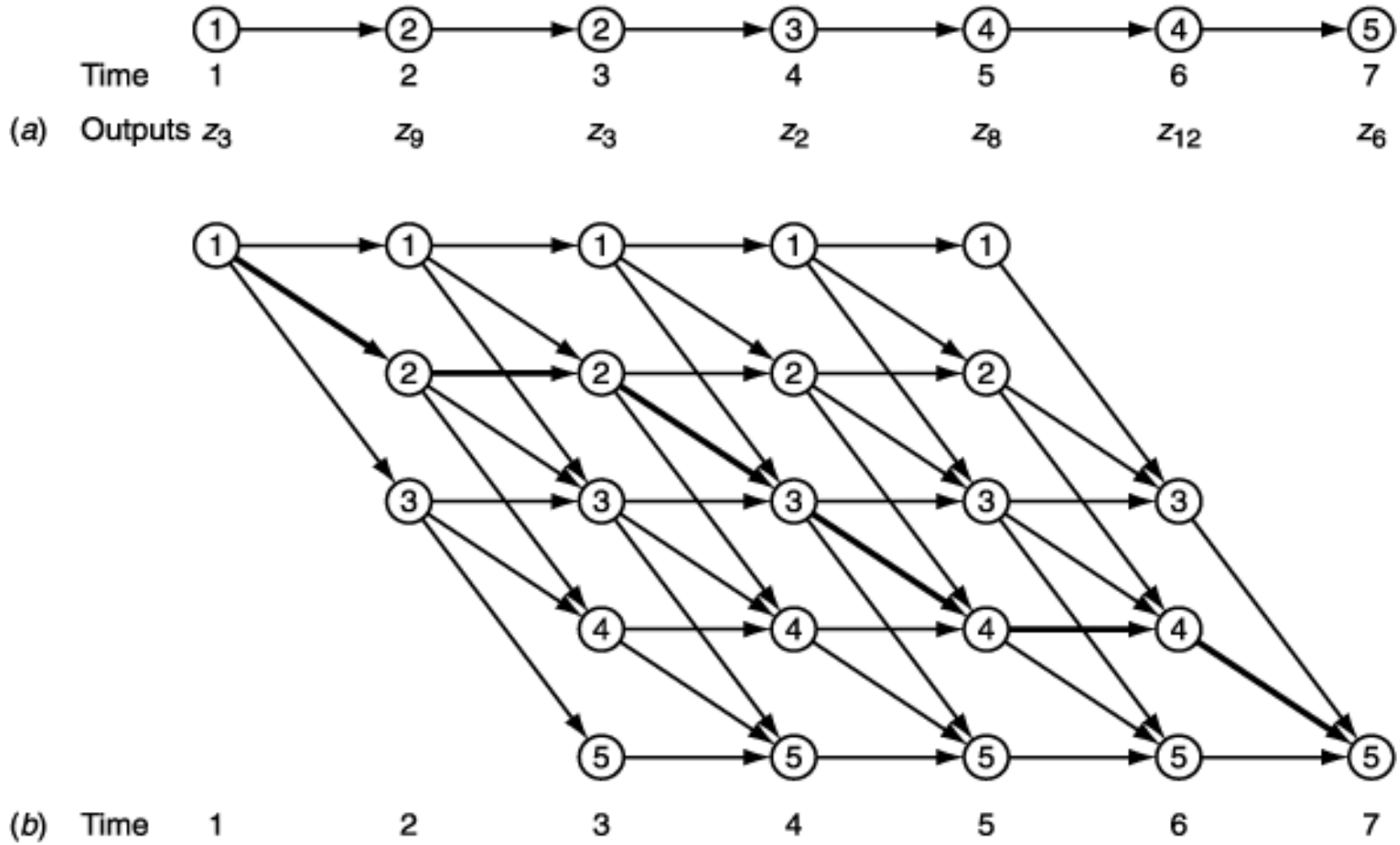
Probability of an Observation with an Arc Emit HMM



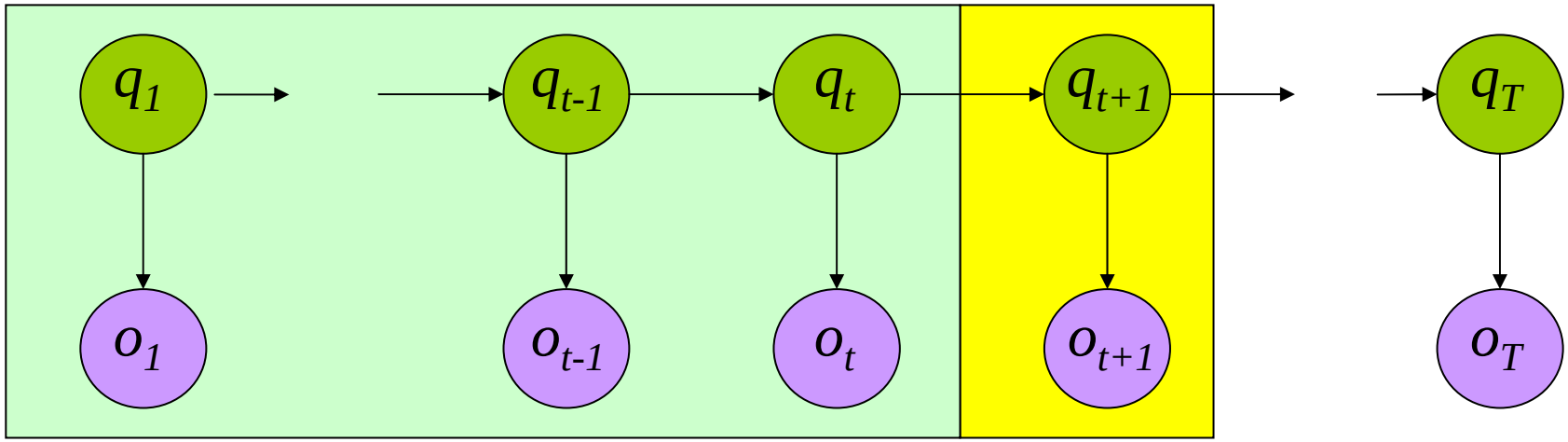
Making Computation Efficient

- To avoid computational complexity, use dynamic programming or memorization techniques due to trellis structure of the problem
- Use an array of states versus time to compute the probability of being at each state at time *t+1* in terms of the probabilities for being in each state at *t*
- A trellis can record the probability of all initial subpaths of the HMM that end in a certain state at a certain time. The probability of longer subpaths can then be worked out in terms of the shorter subpaths
- A forward probability, $\alpha_i(t) = P(o_1 o_2 \ldots o_{t-1}, X_t = i | \mu)$ is stored at (s_i, t) in the trellis and expresses the total probability of ending up in state s_i at time t

The Trellis Structure

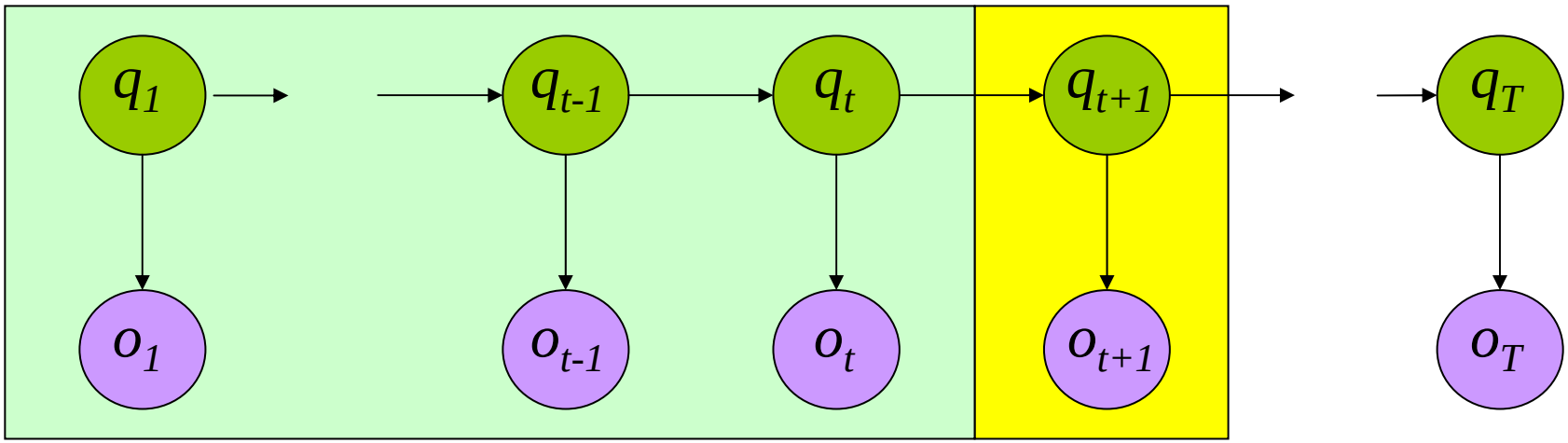


Forward Procedure



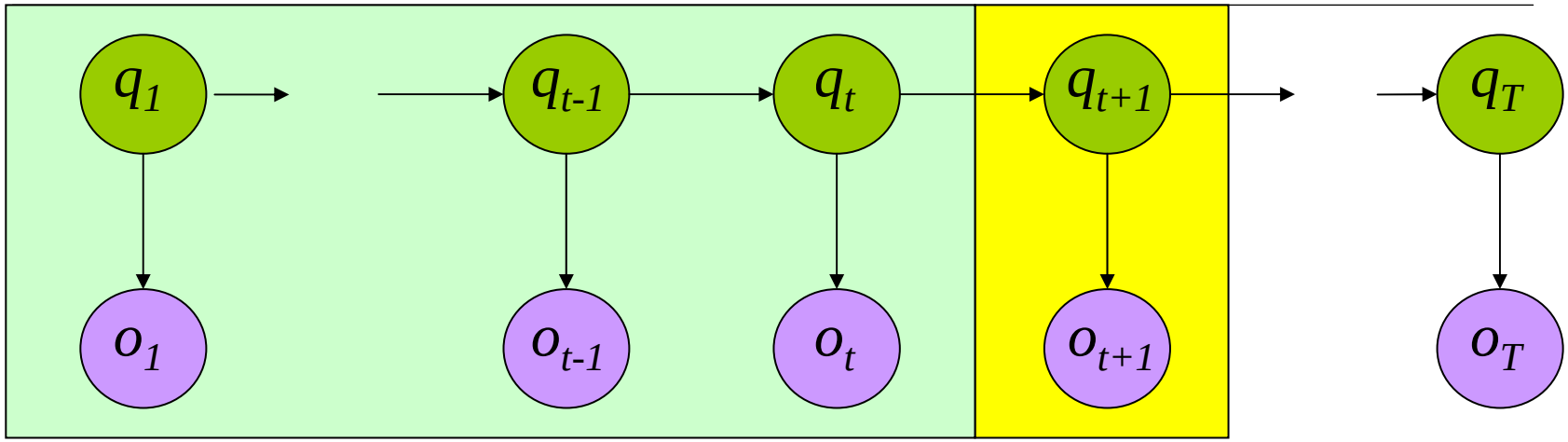
- Special structure gives us an efficient solution using *dynamic programming*
- **Intuition**: Probability of the first *t* observations is the same for all possible *t* +1 length state sequences (so don't recompute it!)
- **Define:** $\alpha_i(t) = P(o_1 \ldots o_t, q_t = i \mid \mu)$

Forward Procedure (Cont.)



$$\alpha_j(t+1) = P(o_1 \ldots o_{t+1}, q_{t+1} = j)$$
$$= P(o_1 \ldots o_{t+1} \mid q_{t+1} = j) P(q_{t+1} = j)$$
$$= P(o_1 \ldots o_t \mid q_{t+1} = j) P(o_{t+1} \mid q_{t+1} = j) P(q_{t+1} = j)$$
$$= P(o_1 \ldots o_t, q_{t+1} = j) P(o_{t+1} \mid q_{t+1} = j)$$

Forward Procedure (Cont.)



$$= \sum_{i=1\ldots N} P(o_1 \ldots o_t, q_t = i, q_{t+1} = j) P(o_{t+1} \mid q_{t+1} = j)$$

$$= \sum_{i=1\ldots N} P(o_1 \ldots o_t, q_{t+1} = j \mid q_t = i) P(q_t = i) P(o_{t+1} \mid q_{t+1} = j)$$

$$= \sum_{i=1\ldots N} P(o_1 \ldots o_t, q_t = i) P(x_{t+1} = j \mid q_t = i) P(o_{t+1} \mid q_{t+1} = j)$$

$$= \sum_{i=1\ldots N} \alpha_i(t) a_{ij} b_{jo_{t+1}}$$

The Forward Algorithm

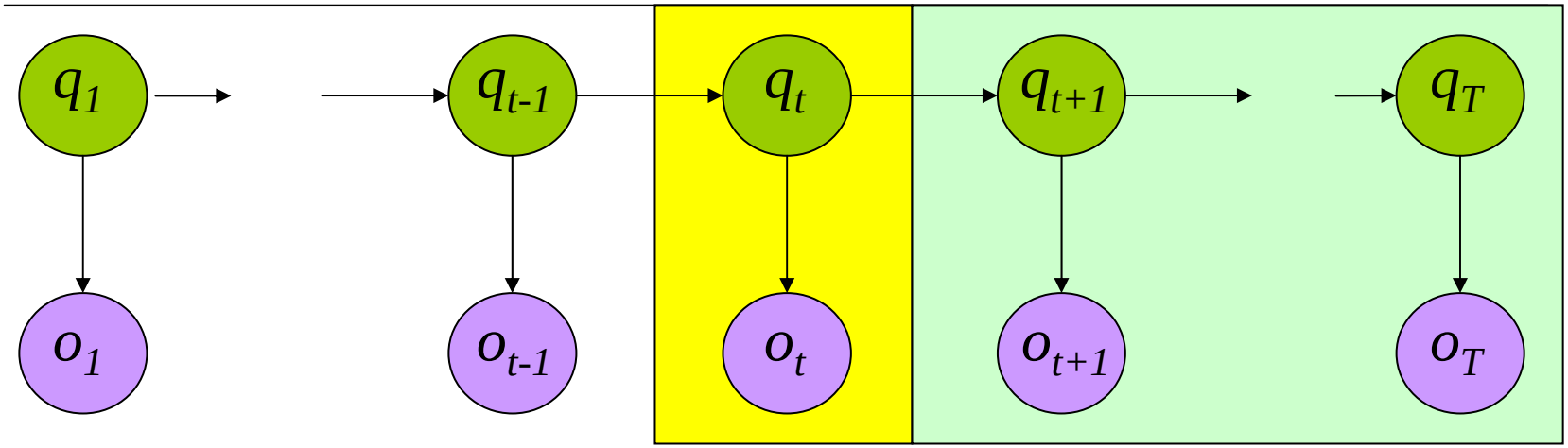
- Forward variables are calculated as follows:
  - Initialization: $\alpha_i(1)= \pi_i$, $1\leq i \leq N$
  - Induction: $\alpha_j(t+1)=\Sigma_{i=1,\ N}\ \alpha_i(t)a_{ij}b_{ijo_t}$, $1\leq t\leq T$, $1\leq j\leq N$
  - Total: $P(O|\mu)= \Sigma_{i=1,N}\ \alpha_i(T+1)$

- This algorithm requires $2N^2T$ multiplications (much less than the direct method which takes $(2T+1)N^{T+1}$

The Backward Procedure

- We could also compute these probabilities by working backward through time
- The backward procedure computes backward variables which are the total probability of seeing the rest of the observation sequence given that we were in state s_i at time t
- $\beta_i(t) = P(o_t \ldots o_T | q_t = i, \Lambda)$ is a backward variable (probability)
- Backward variables are useful for the problem of parameter reestimation

Backward Procedure (Cont.)



$$\beta_i(T+1) = 1$$

$$\beta_i(t) = P(o_t \ldots o_T \mid q_t = i)$$

$$\beta_i(t) = \sum_{j=1\ldots N} a_{ij} b_{io_t} \beta_j(t+1)$$

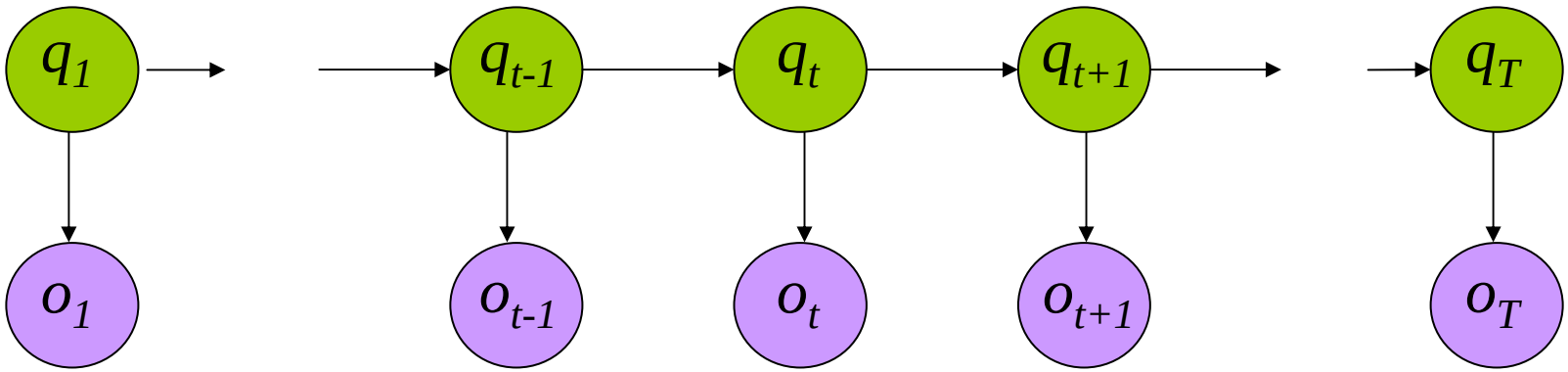
The Backward Algorithm

- Backward variables can be calculated working backward through the treillis as follows:
  - Initialization: $\beta_i(T+1) = 1,\ 1 \le i \le N$
  - Induction: $\beta_i(t) = \Sigma_{j=1,N}\ a_{ij} b_{ijo_t} \beta_j(t+1),\ 1 \le t \le T,\ 1 \le i \le N$
  - Total: $P(O|\mu) = \Sigma_{i=1,\ N}\ \pi_i \beta_i(1)$

- Backward variables can also be combined with forward variables:

  $P(O|\mu) = \Sigma_{i=1,N}\ \alpha_i(t)\beta_i(t),\ 1 \le\ t \le T+1$

Summary: Probability of an Observation



$$P(O \mid \mu) = \sum_{i=1}^{N} \alpha_i(T)$$ Forward Procedure

$$P(O \mid \mu) = \sum_{i=1}^{N} \pi_i \beta_i(1)$$ Backward Procedure

$$P(O \mid \mu) = \sum_{i=1}^{N} \alpha_i(t)\beta_i(t)$$ Combination

Finding the Best State Sequence

- One method consists of optimizing on the states individually
- For each t, $1 \leq t \leq T+1$, we would like to find X_t that maximizes $P(X_t|O, \Lambda)$
- Let $\gamma_i(t) = P(X_t = i \,|O, \Lambda) = P(X_t = i, O|\, \Lambda)/P(O|\, \Lambda) = (\alpha_i(t)\beta_i(t))/\Sigma_{j=1,N}\, \alpha_j(t)\beta_j(t)$
- The individually most likely state is

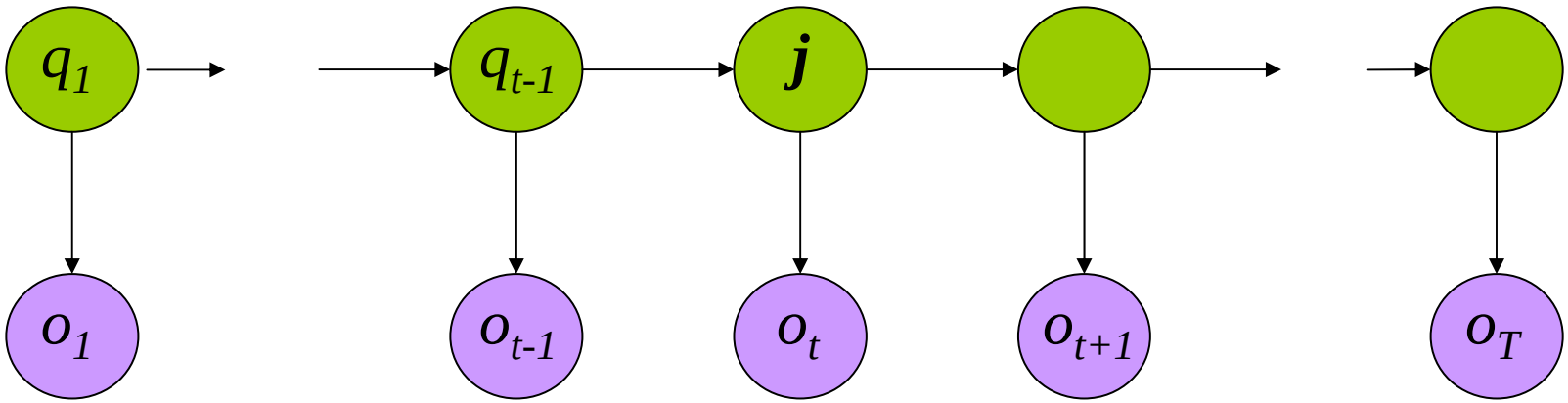
$$X_t = argmax_{1 \leq i \leq N}\, \gamma_i(t),\ 1 \leq t \leq T+1$$

- This quantity maximizes the expected number of states that will be guessed correctly. However, it may yield a quite unlikely state sequence

The Viterbi Algorithm

- The Viterbi algorithm efficiently computes the most likely state sequence.
- To find the most likely complete path compute: $\text{argmax}_S P(S|O, \Lambda)$
- To do this, it is sufficient to maximize for a fixed O: $\text{argmax}_S P(S,O| \Lambda)$
- We define
  $\delta_j(t) = \max_{q_1..q_{t-1}} P(q_1 \ldots q_{t-1}, o_1..o_{t-1}, q_t=j|\mu)$ with $\psi_j(t)$ records the node of the incoming arc that led to this most probable path

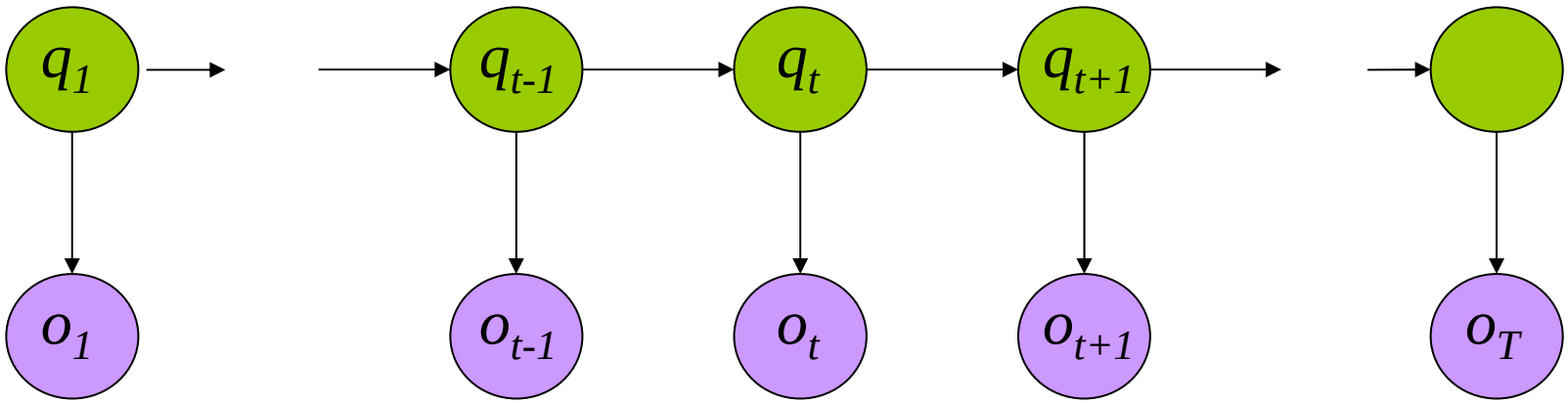
Viterbi Algorithm Properties



$$\delta_j(t) = \max_{q_1 \ldots q_{t-1}} P(q_1 \ldots q_{t-1}, o_1 \ldots o_{t-1}, q_t = j, o_t)$$

The state sub-sequence which maximizes the probability of seeing the observations to time *t -1*, landing in state *j*, and seeing the observation at time *t*

Viterbi Algorithm Properties (Cont.)



$$\delta_j(t) = \max_{q_1 \ldots q_{t-1}} P(q_1 \ldots q_{t-1}, o_1 \ldots o_{t-1}, q_t = j, o_t)$$

$$\delta_j(t+1) = \max_i \delta_i(t) a_{ij} b_{jo_{t+1}}$$

$$\psi_j(t+1) = \arg\max_i \delta_i(t) a_{ij} b_{jo_{t+1}}$$

DP recursive computation

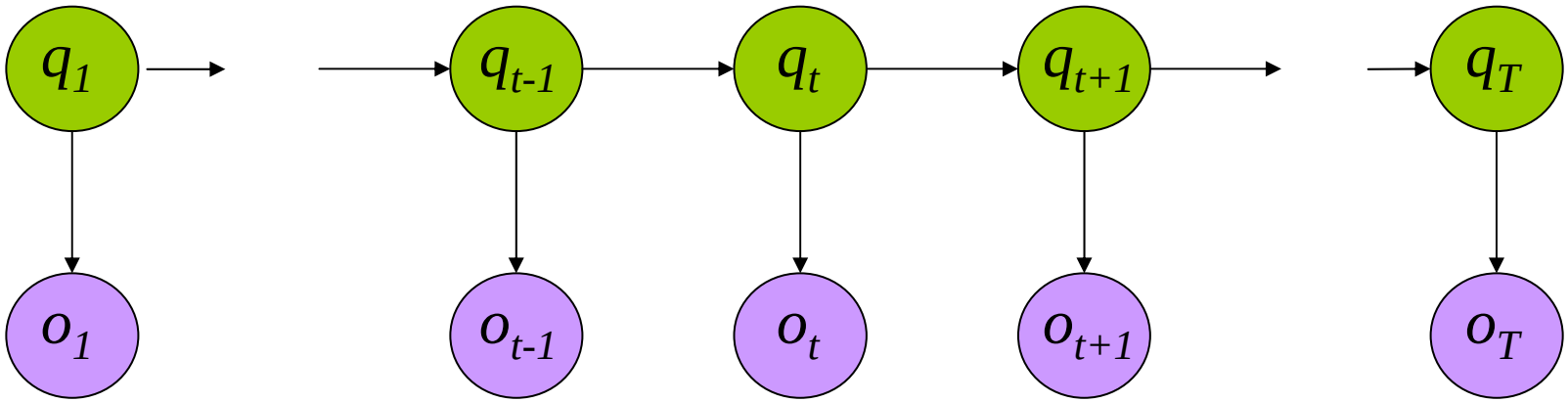
Finally: The Viterbi Algorithm

The Viterbi Algorithm works as follows:

- Initialization: $\delta_j(1) = \pi_j,\ 1 \le j \le N$
- Induction: $\delta_j(t+1) = \max_{1 \le i \le N} \delta_i(t) a_{ij} b_{ijot},\ 1 \le j \le N$
- Store backtrace:

  $\psi_j(t+1) = \text{argmax}_{1 \le i \le N} \delta_j(t) a_{ij}\, b_{ijot},\ 1 \le j \le N$
- Termination and path readout: next page

Viterbi Algorithm: Another Look



$$\hat{q}_T = \arg\max_i \delta_i(T)$$

$$\hat{q}_t = \psi_{\hat{q}_{t+1}}(t+1)$$

$$P(S) = \max_i \delta_i(T)$$

Compute the most likely state sequence by working backwards

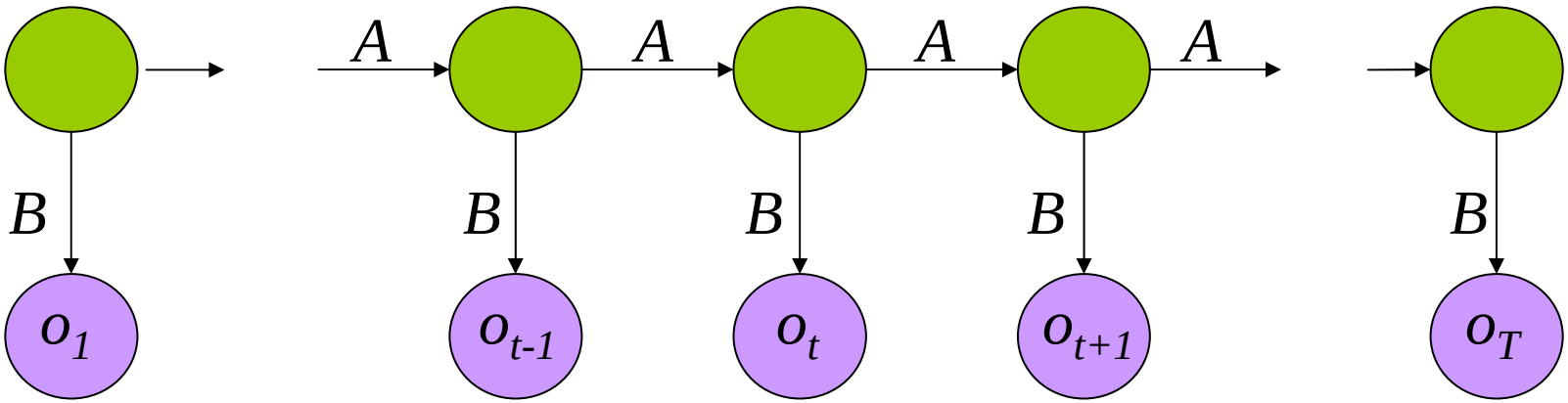
HMM Parameter Estimation

- Given a certain collection of observation sequences, we want to find the values of the model parameters $\Lambda = (A, B, \pi)$ which best explain the observed data
- Using maximum likelihood estimation (MLE) to find values to maximize $P(O|\Lambda)$, i.e. $\text{argmax}_{\mu} P(O_{\text{training}}|\mu)$
- There is no known analytic method to choose μ to maximize $P(O|\Lambda)$. However, we can locally maximize it by an iterative hill-climbing algorithm known as Baum-Welch or forward-backward algorithm (this is a special case of the EM algorithm which is used in many miss data problems in statistical inference)

The Forward-Backward Algorithm

- We don't know what the model is, but we can work out the probability of the observation sequence using some (perhaps randomly chosen) model
- Looking at that calculation, we can see which state transitions and symbol emissions were probably used the most
- By increasing the probability of those, we can choose a revised model which gives a higher probability to the observation sequence

Parameter Estimation



- Given an observation sequence, find the model that is most likely to produce that sequence
- No analytic method
- Given a model and training sequences, update the model parameters to better fit the observations

Some Definitions

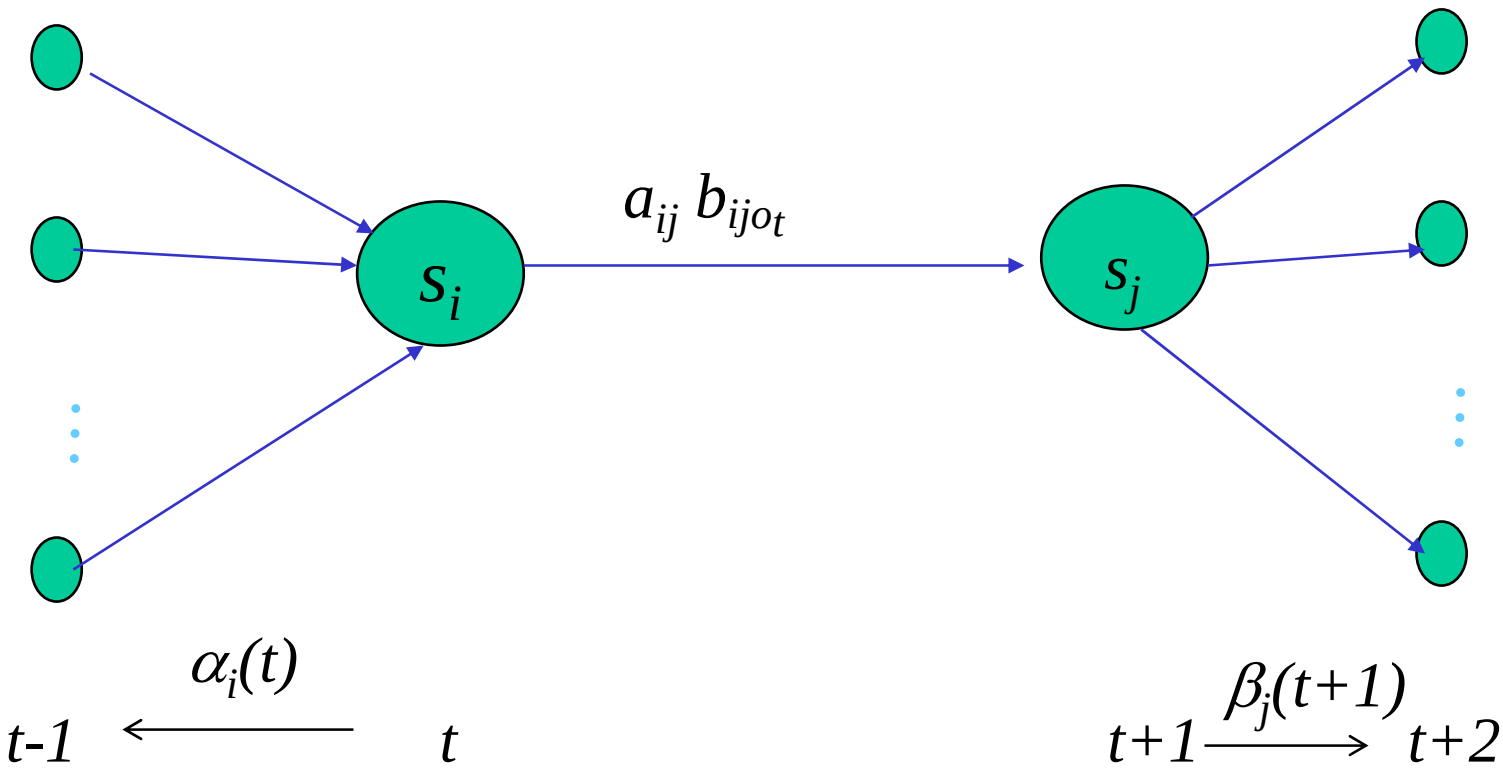
- The probability of traversing a certain arc at time *t* given Observation sequence O:

$$
\begin{aligned}
p_t(i,j) &= p(q_t = i, q_{t+1} = j \mid O, \mu) \\
&= \frac{p(q_t = i, q_{t+1} = j, O \mid \mu)}{p(O \mid \mu)} \\
&= \frac{\alpha_i(t) a_{ij} b_{ijo_{t+1}} \beta_j(t+1)}{\sum_{m=1\ldots N} \alpha_m(t)\beta_m(t)}
\end{aligned}
$$

Let:

$$
\gamma_i(t) = \sum_{j=1}^{N} p_t(i,j)
$$

The Probability of Traversing an Arc



More Definitions

- The expected number of transitions from state *i* in O:

$$\sum_{t=1}^{T} \gamma_i(t)$$

- The expected number of transitions from state *i* to *j* in O

$$\sum_{t=1}^{T} p_t(i, j)$$

Reestimation Procedure

- Begin with model μ perhaps selected at random
- Run *O* through the current model to estimate the expectations of each model parameter
- Change model to maximize the values of the paths that are used a lot while respecting stochastic constraints
- Repeat this process (hoping to converge on optimal values for the model parameters μ)

The Reestimation Formula

From $\mu = (A, B, \Pi)$, derive $\hat{\mu} = (\hat{A}, \hat{B}, \hat{\Pi})$.

Note that $P(O \mid \hat{\mu}) \geq P(O \mid \mu)$.

The expected frequency in state *i* at time t=1: $\hat{\pi}_i = \gamma_i(1)$

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

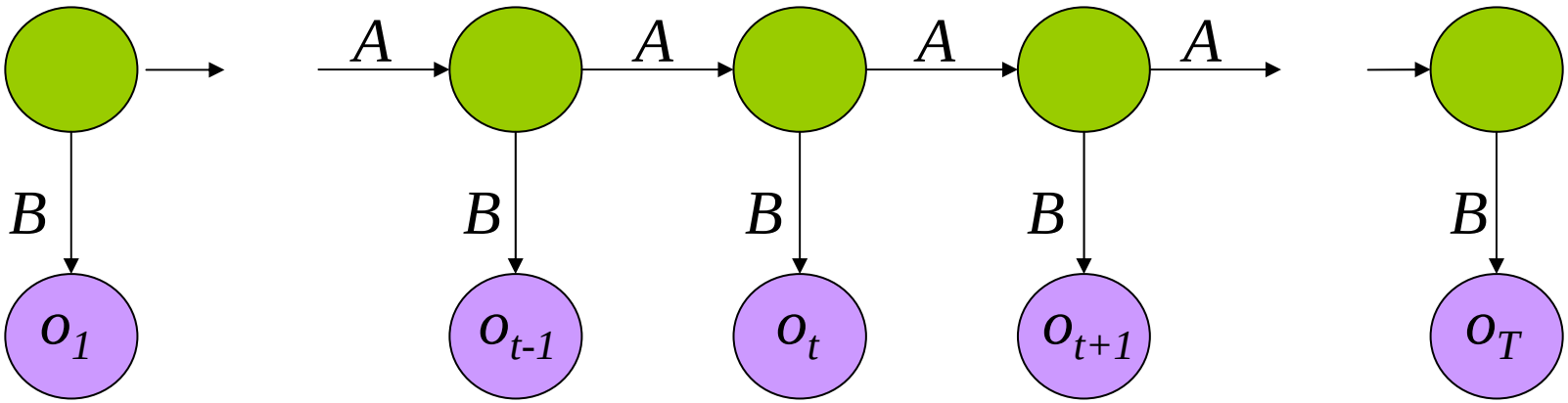
$$= \frac{\sum_{t=1}^{T} p_t(i, j)}{\sum_{t=1}^{T} \gamma_i(t)}$$

Reestimation Formula (Cont.)

$$\hat{b}_{ijk} = \frac{\text{expected \# transitions from state } i \text{ to } j \text{ observing } k}{\text{expected \# transitions from state } i \text{ to } j}$$

$$= \frac{\sum_{(t:o_t=k,1\le t\le T)} p_t(i,j)}{\sum_{t=1}^{T} p_t(i,j)}$$

Parameter Estimation: Summary 1



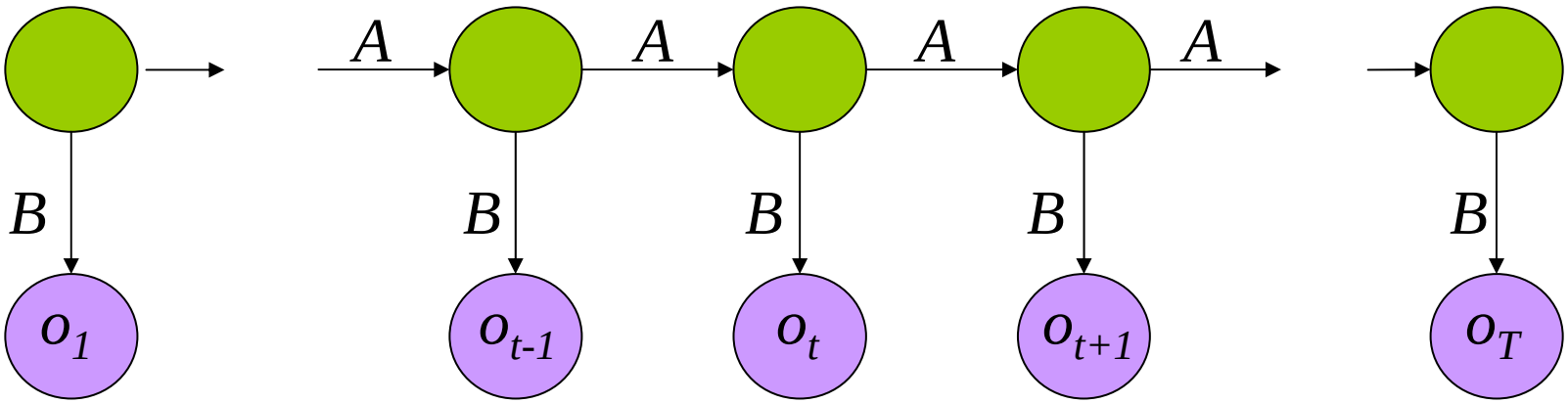
$$p_t(i,j) = \frac{\alpha_i(t) a_{ij} b_{jo_{t+1}} \beta_j(t+1)}{\sum_{m=1\ldots N} \alpha_m(t)\beta_m(t)}$$

Probability of traversing an arc at time *t*

$$\gamma_i(t) = \sum_{j=1\ldots N} p_t(i,j)$$

Probability of being in state *i*

Parameter Estimation: Summary 2



$$\hat{\pi}_i = \gamma_i(1)$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} p_t(i,j)}{\sum_{t=1}^{T} \gamma_i(t)}$$

$$\hat{b}_{ik} = \frac{\sum_{\{t:o_t=k\}} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)}$$

Now we can compute the new estimates of the model parameters

Baum-Welch Algorithm: DDHMM (I)

E-step:

$$Q(\Lambda;\Lambda^{(n)}) = E_{\{S_l\}}\left[\ln p(O_1,\cdots O_L,S_1,\cdots,S_L \mid \Lambda) \mid O_1,\cdots O_L,\Lambda^{(n)}\right]$$

$$= \sum_{S_1\cdots S_L}\left[\sum_{l=1}^{L}\ln p(O_l,S_l \mid \Lambda)\right]\cdot\prod_{l=1}^{L} p(S_l \mid O_l,\Lambda^{(n)}) = \sum_{l=1}^{L}\sum_{S_l}\ln p(O_l,S_l \mid \Lambda)\cdot p(S_l \mid O_l,\Lambda^{(n)})$$

$$= \sum_{l=1}^{L}\sum_{s_{l1}\cdots s_{lT_l}}\left[\ln \pi_{s_{l1}} + \cdot\ln b_{s_{l_1}}(o_{l1}) + \sum_{t=2}^{T_l}\ln a_{s_{t-1}s_t} + \sum_{t=2}^{T_l}\ln b_{s_t}(o_t)\right]\cdot p(S_l \mid O_l,\Lambda^{(n)})$$

$$= \sum_{l=1}^{L}\sum_{i=1}^{N}\ln \pi_i \cdot \Pr(s_{l1}=s_i \mid O_l,\Lambda^{(n)}) + \sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{t=2}^{T_l}\ln a_{ij}\cdot\Pr(s_{lt-1}=s_i,s_{lt}=s_j \mid O_l,\Lambda^{(n)})$$

$$+ \sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{t=1}^{T_l}\ln b_i(v_M)\cdot\Pr(s_{lt}=s_i,o_t=v_m \mid O_l,\Lambda^{(n)})$$

$$= Q(\pi;\pi^{(n)}) + Q(A;A^{(n)}) + Q(B;B^{(n)})$$

Baum-Welch Algorithm: DDHMM (II)

M-step:

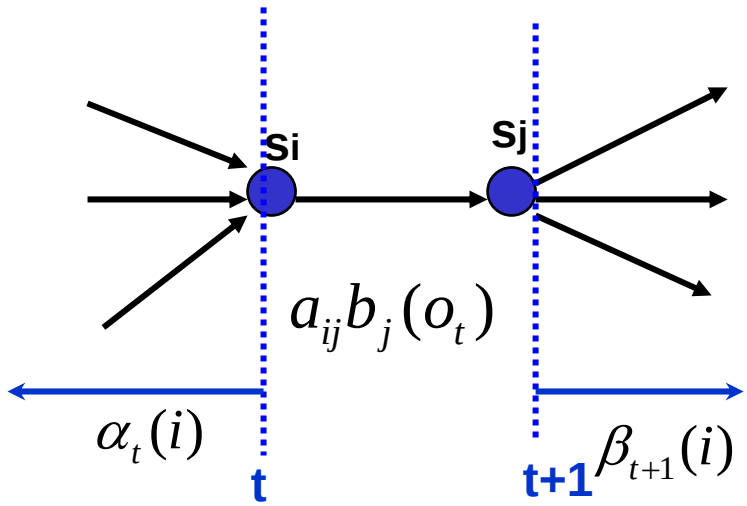
$$\frac{\partial Q(\pi;\pi^{(n)})}{\partial \pi_i}=0 \Rightarrow \pi_i^{(n+1)}=\frac{\sum_{l=1}^{L}\Pr(s_{l1}=s_i\,|\,O_l,\Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{i=1}^{N}\Pr(s_{l1}=s_i\,|\,O_l,\Lambda^{(n)})}$$

$$\frac{\partial Q(A;A^{(n)})}{\partial a_{ij}}=0 \Rightarrow a_{ij}^{(n+1)}=\frac{\sum_{l=1}^{L}\sum_{t=2}^{T_l}\Pr(s_{lt-1}=s_i,s_{lt}=s_j\,|\,O_l,\Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=2}^{T_l}\sum_{j=1}^{N}\Pr(s_{lt-1}=s_i,s_{lt}=s_j\,|\,O_l,\Lambda^{(n)})}=\frac{\sum_{l=1}^{L}\sum_{t=2}^{T_l}\Pr(s_{lt-1}=s_i,s_{lt}=s_j\,|\,O_l,\Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=1}^{T_l-1}\Pr(s_{lt}=s_i\,|\,O_l,\Lambda^{(n)})}$$

$$\frac{\partial Q(B;B^{(n)})}{\partial b_i(v_m)}=0 \Rightarrow b_i(v_m)=\frac{\sum_{l=1}^{L}\sum_{t=1}^{T_l}\Pr(s_t=s_i,o_t=v_m\,|\,O_l,\Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=1}^{T_l}\sum_{m=1}^{M}\Pr(s_t=s_i,o_t=v_m\,|\,O_l,\Lambda^{(n)})}=\frac{\sum_{l=1}^{L}\sum_{t=1}^{T_l}\Pr(s_t=s_i,o_t=v_m\,|\,O_l,\Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=1}^{T_l}\Pr(s_t=s_i\,|\,O_l,\Lambda^{(n)})}$$

Baum-Welch Algorithm: DDHMM (III)

How to calculate the posteriori probabilities of traversing an arc going from state *i* to *j* at time *t*?



$$\Pr(s_{t-1}=s_i, s_t=s_j \mid O,\Lambda) = \frac{\sum \text{prob of all paths } s_i \text{ at } t-1 \text{ and } s_j \text{ at } t}{\sum \text{prob of all paths}}$$

$$= \frac{\alpha_t(i)\cdot a_{ij}b_j(o_t)\cdot\beta_{t+1}(j)}{\Pr(O_l \mid \Lambda)} = \frac{\alpha_t(i)\cdot a_{ij}b_j(o_t)\cdot\beta_{t+1}(j)}{\sum_{i=1}^{N}\alpha_T(i)} = \frac{\alpha_t(i)\cdot a_{ij}b_j(o_t)\cdot\beta_{t+1}(j)}{P(O\mid\Lambda)}$$

$$\equiv \xi_t(i,j)$$

Baum-Welch Algorithm: DDHMM (IV)

Define state occupancy probability:

$$\gamma_i(t) = P(s_t = i \mid O, \Lambda) = \frac{P(s_t = i, O \mid \Lambda)}{P(O \mid \Lambda)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$$

Note $\gamma_i(t) = \sum_{j=1}^{N} \xi_t(i, j)$

$$\sum_{t=1}^{T} \gamma_i(t) = \text{Expected number of transitions from state } i \text{ in } O$$

$$\sum_{t=1}^{T} \xi_t(i, j) = \text{Expected number of transitions from state } i \text{ to } j \text{ in } O$$

Baum-Welch Algorithm: DDHMM (V)

Final results: one iteration, from $\Lambda^{(n)} = \{A^{(n)}, B^{(n)}, \pi^{(n)}\}$

$$\pi_i^{(n+1)} = \frac{\sum_{l=1}^{L} \Pr(s_{l1} = s_i \mid O_l, \Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{i=1}^{N} \Pr(s_{l1} = s_i \mid O_l, \Lambda^{(n)})} = \sum_{l=1}^{L}\sum_{j=1}^{N} \xi_1^{(l)}(i,j)$$

$$a_{ij}^{(n+1)} = \frac{\sum_{l=1}^{L}\sum_{t=2}^{T_l} \Pr(s_{lt-1} = s_i, s_{lt} = s_j \mid O_l, \Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=2}^{T_l}\sum_{j=1}^{N} \Pr(s_{lt-1} = s_i, s_{lt} = s_j \mid O_l, \Lambda^{(n)})} = \frac{\sum_{l=1}^{L}\sum_{t=2}^{T_l} \xi_t^{(l)}(i,j)}{\sum_{l=1}^{L}\sum_{t=2}^{T_l}\sum_{j=1}^{N} \xi_t^{(l)}(i,j)}$$

$$b_i(v_m) = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T_l} \Pr(s_t = s_i, o_t = v_m \mid O_l, \Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=1}^{T_l}\sum_{m=1}^{M} \Pr(s_t = s_i, o_t = v_m \mid O_l, \Lambda^{(n)})} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{T_l} \xi_t^{(l)}(i,j) \cdot \delta(o_{lt} - v_m)}{\sum_{l=1}^{L}\sum_{t=1}^{T_l} \xi_t^{(l)}(i,j)}$$

Baum-Welch: Gaussian Mixture CDHMM (I)

- Treat both state sequence *SI* and mixture component label sequence *II* as missing data.
- Only *B* estimation is different.
- E-step:

$$Q(B;B^{(n)}) = \sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{t=1}^{T_l} \ln b_{ik}(X_{lt}) \cdot \Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)})$$

$$= \sum_{l=1}^{L}\sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{t=1}^{T_l}\left[\ln \omega_{ik} - \frac{n}{2}\ln|\Sigma_{ik}| - \frac{1}{2}\cdot(X_{lt}-\mu_{ik})^t \Sigma_{ik}^{-1}(X_{lt}-\mu_{ik})\right] \cdot \Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)})$$

- M-step:

$$\frac{\partial Q(B;B^{(n)})}{\partial \mu_{ik}} = 0 \Rightarrow \mu_{ik}^{(n+1)} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{L_t} X_{lt} \cdot \Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=1}^{L_t} \Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)})}$$

Baum-Welch: Gaussian Mixture CDHMM (II)

$$\frac{\partial Q(B;B^{(n)})}{\partial \Sigma_{ik}} = 0 \Rightarrow \Sigma_{ik}^{(n+1)} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{L_t}(X_{lt}-\mu_{ik}^{(n)})^t \cdot (X_{lt}-\mu_{ik}^{(n)}) \cdot \Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=1}^{L_t}\Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)})}$$

$$\frac{\partial Q(B;B^{(n)})}{\partial \omega_{ik}} = 0 \Rightarrow \omega_{ik}^{(n+1)} = \frac{\sum_{l=1}^{L}\sum_{t=1}^{L_t}\Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)})}{\sum_{l=1}^{L}\sum_{t=1}^{L_t}\sum_{k=1}^{K}\Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)})}$$

where the posteriori probabilities are calculated as:

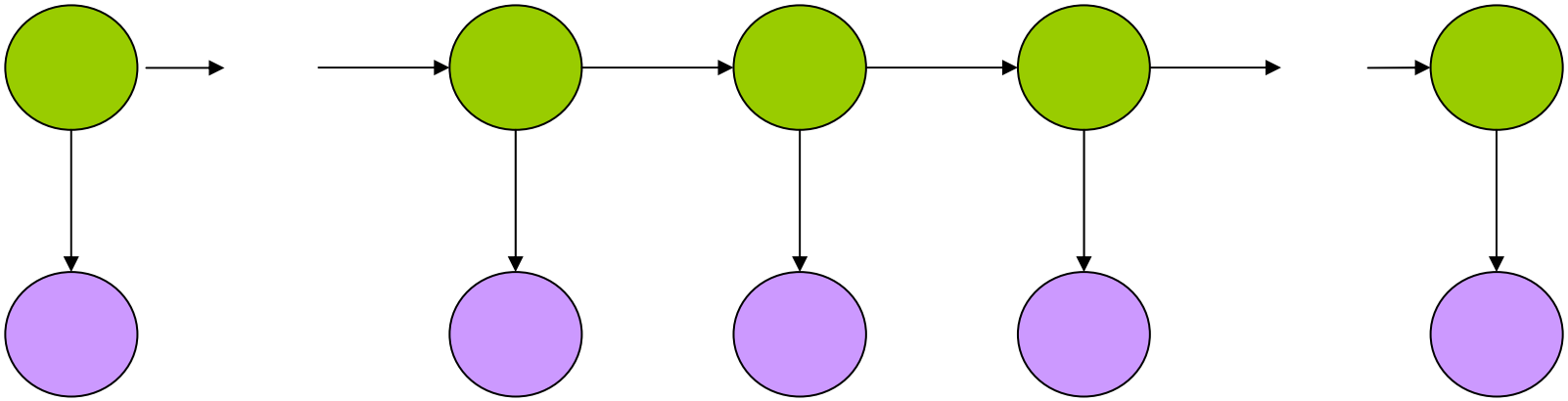
$$\Pr(s_{lt}=s_i, l_{lt}=k \mid O_l, \Lambda^{(n)}) \equiv \zeta_t^{(l)}(i,k) = \frac{\alpha_t^{(l)}(i)\cdot\beta_t^{(l)}(i)\cdot\gamma_{ik}^{(l)}(t)}{P_l \cdot \sum_{k=1}^{K}\gamma_{ik}^{(l)}(t)}$$

$$\text{where } \gamma_{ik}^{(l)}(t) = \frac{\omega_{ik}^{(n)} \cdot N(X_{lt} \mid \mu_{ik}^{(n)}, \Sigma_{ik}^{(n)})}{\sum_{k=1}^{K}\omega_{ik}^{(n)} \cdot N(X_{lt} \mid \mu_{ik}^{(n)}, \Sigma_{ik}^{(n)})}$$

HMM Learning: Summary

- For an HMM model $\Lambda = \{A, B, \pi\}$ and a training data set $D = \{O_1, O_2, \ldots, O_L\}$,
  1. Initialization: $\Lambda^{(0)} = \{A^{(0)}, B^{(0)}, \pi^{(0)}\}$
  2. $n=0$;
  3. For each observation sequence O_l $(l=1,2,\ldots,L)$:
     - Calculate $\alpha_t(i)$ and $\beta_t(i)$ based on $\Lambda^{(n)}$
     - Calculate all other posteriori probabilities
     - Accumulate a numerator and a denominator for each HMM parameter
  4. HMM parameters update: $\Lambda^{(n+1)} =$ the numerators divided by the denominators
  5. $n=n+1$; Go to step 2 until convergence

HMM Applications



- Parameters for interpolated *n*-gram models
- Part-of-speech tagging
- Speech recognition
- Machine translation
- Many others

Summary

- Today's and next classes
  - Hidden Markov Models
- Note
  - Project summary due on 2/24, let's start our discussion
  - Project plan finalize on 3/3 (presentation on 4/16 ???)
  - Lab3 assigned on 2/12 and due on 2/26
  - Midterm on 3/12 (???)
  - Final at 8am on 4/27 (shall we try a take-home ???)
- Reading Assignments
  - Manning and Schutze, Chapter 9