HMMs

COSC 6336: Natural Language Processing
Spring 2020

UNIVERSITY of HOUSTON

Some content on these slides was adapted from Ray Mooney

In This Lecture

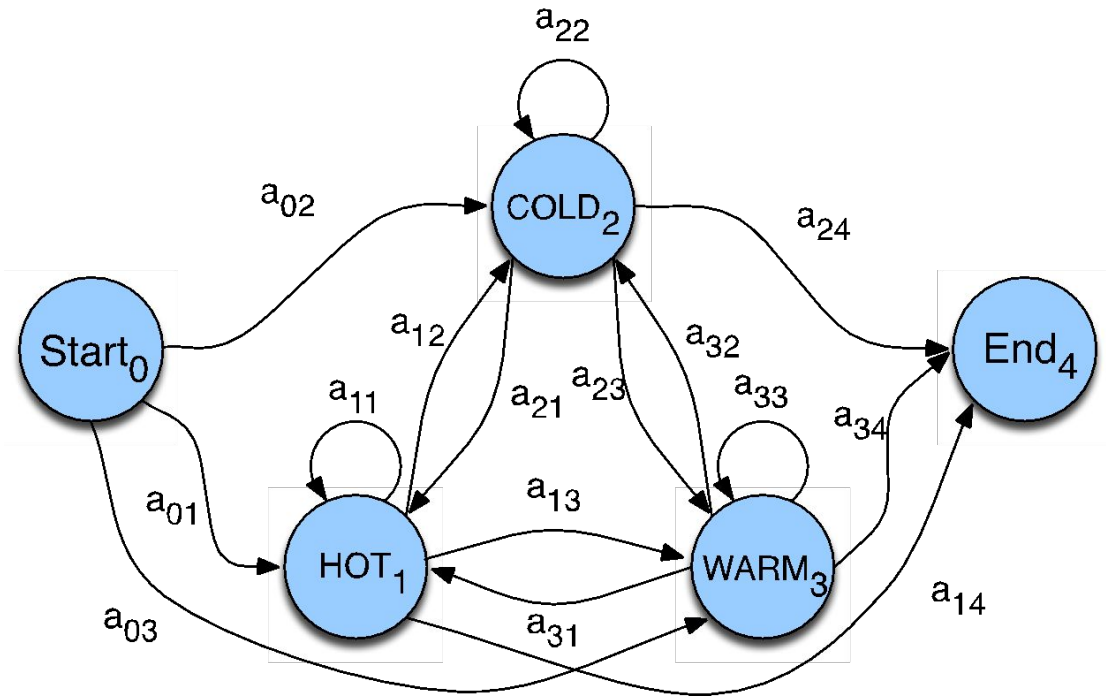
- Introduction to Hidden Markov Models (HMMs)
  - Forward algorithm
  - Viterbi algorithm

More Formally: Toward HMMs

Markov Models

- A Weighted Finite-State Automaton (WFSA)
  - An FSA with probabilities on the arcs
  - The sum of the probabilities leaving any arc must sum to one
- A Markov chain (or observable Markov Model)
  - a special case of a WFSA in which the input sequence uniquely determines which states the automaton will go through
- Markov chains can't represent inherently ambiguous problems
  - Useful for assigning probabilities to unambiguous sequences

Markov Chain for Weather



First-order Observable Markov Model

- A set of states
  - Q = q_1, $q_2 \ldots q_N$; the state at time t is q_t
- Current state only depends on previous state

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

- Transition probability matrix A

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

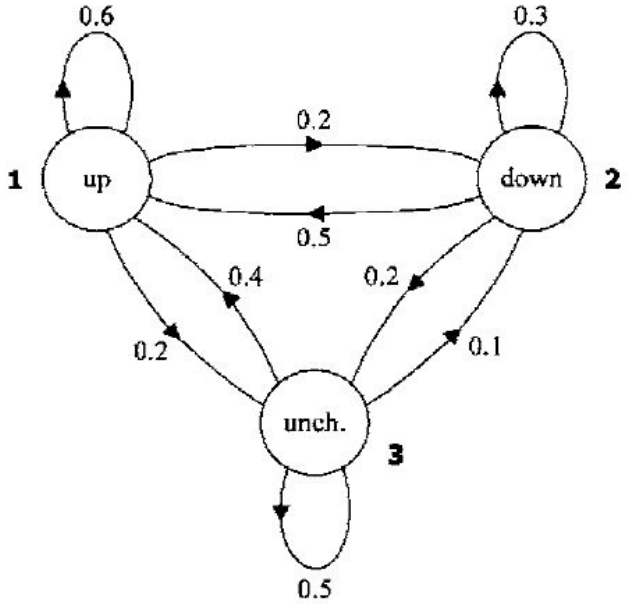
- Special initial probability vector π

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

- Constraints:

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

Markov Model for Dow Jones



Initial state probability matrix

$$\boldsymbol{\pi} = (\pi_i) = \begin{pmatrix} 0.5 \\ 0.2 \\ 0.3 \end{pmatrix}$$

State-transition probability matrix

$$\mathbf{A} = \{a_{ij}\} = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.5 & 0.3 & 0.2 \\ 0.4 & 0.1 & 0.5 \end{bmatrix}$$

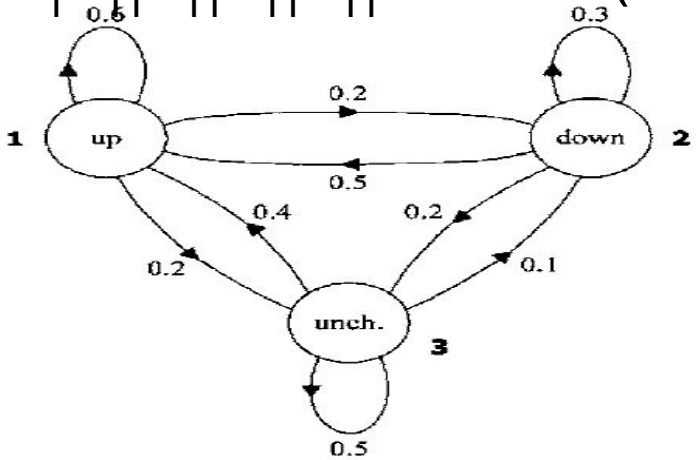
UNIVERSITY of HOUSTON

Markov Model for Dow Jones

- What is the probability of 5 consecutive up days?
- Sequence is up-up-up-up-up
- I.e., state sequence is 1-1-1-1-1
- P(1,1,1,1,1) = ?

Markov Model for Dow Jones

- P(1,1,1,1,1) =
  - $\pi_1 a_{11} a_{11} a_{11} a_{11} = 0.5 \times (0.6)^4 = 0.0648$



Initial state probability matrix

$$\boldsymbol{\pi} = (\pi_i) = \begin{pmatrix} 0.5 \\ 0.2 \\ 0.3 \end{pmatrix}$$

State-transition probability matrix

$$\mathbf{A} = \{a_{ij}\} = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.5 & 0.3 & 0.2 \\ 0.4 & 0.1 & 0.5 \end{bmatrix}$$

Hidden Markov Model

- In many NLP tasks:
  - output symbols are words
  - hidden states are something else
- A Hidden Markov Model is an extension of a Markov chain in which the input symbols are not the same as the states.
- This means we don't know which state we are in.

Hidden Markov Model

- Assume an underlying set of hidden (unobserved, latent) states in which the model can be (e.g. POS).
- Assume probabilistic transitions between states over time (e.g. transition from POS to another POS as sequence is generated).
- Assume a probabilistic generation of tokens from states (e.g. words generated for each POS).

Hidden Markov Models

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_i
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

Assumptions

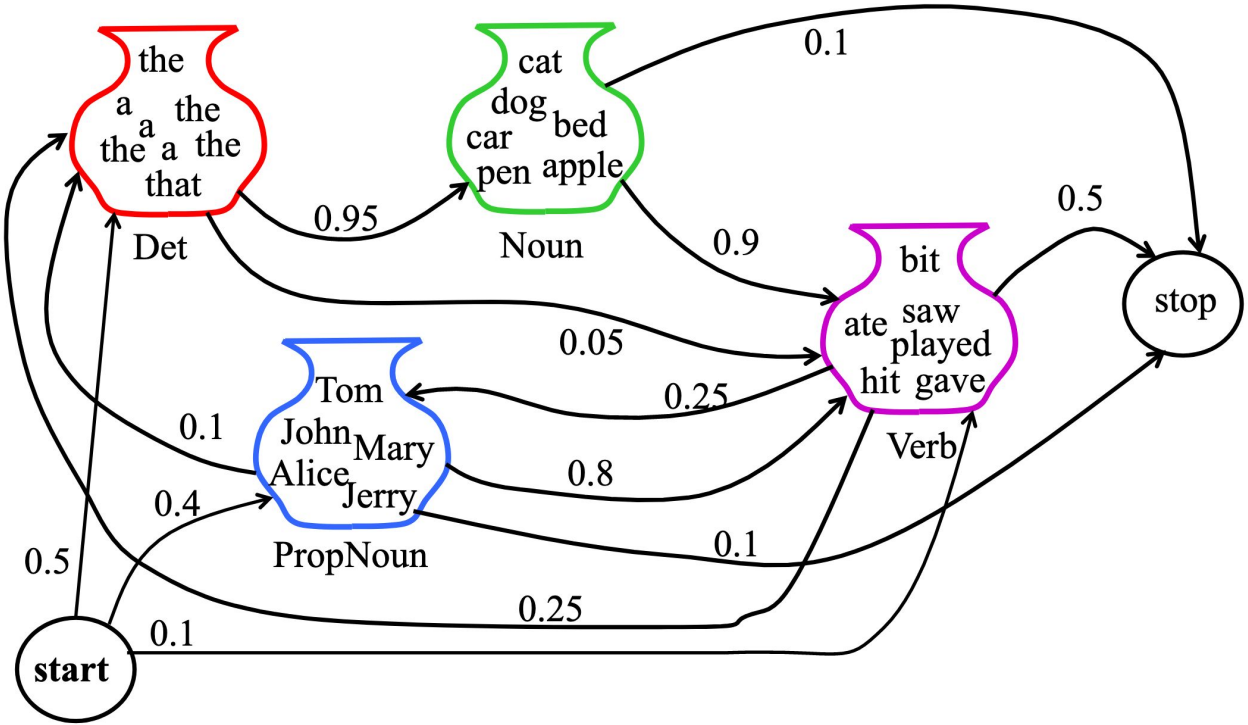
- Markov assumption:

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

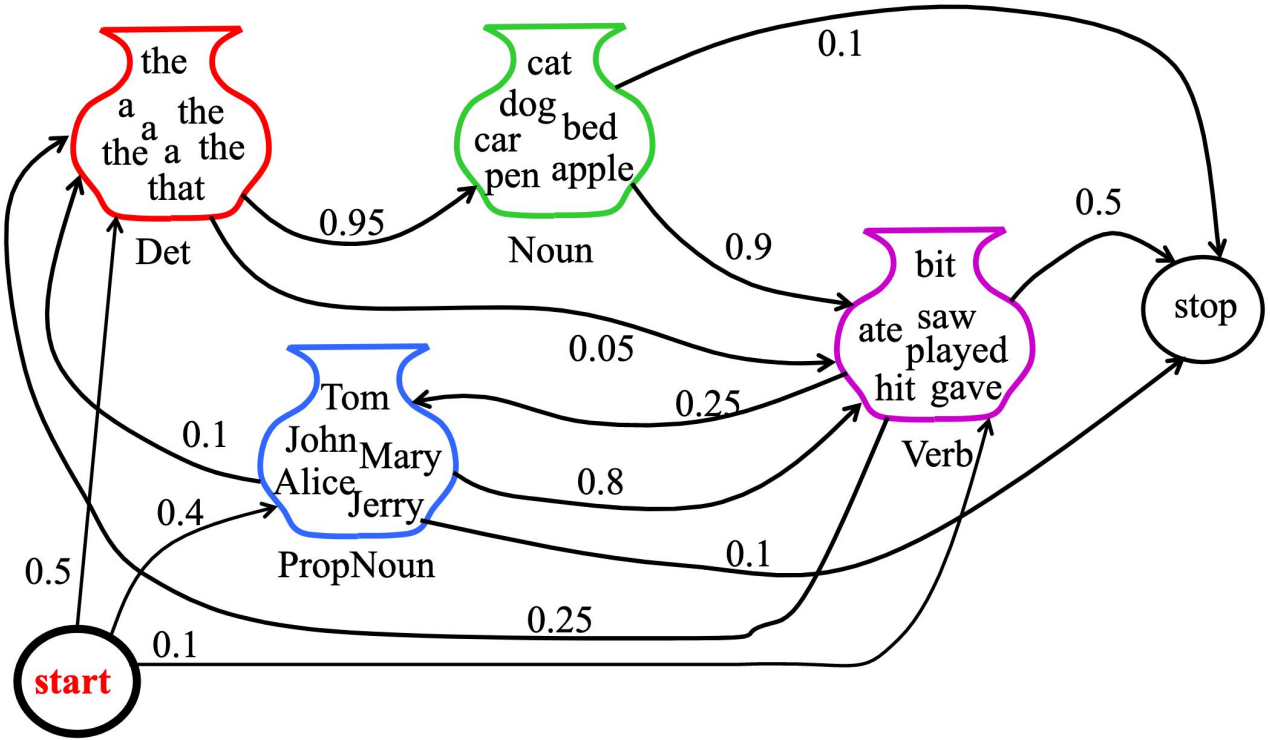
- Output-independence assumption

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

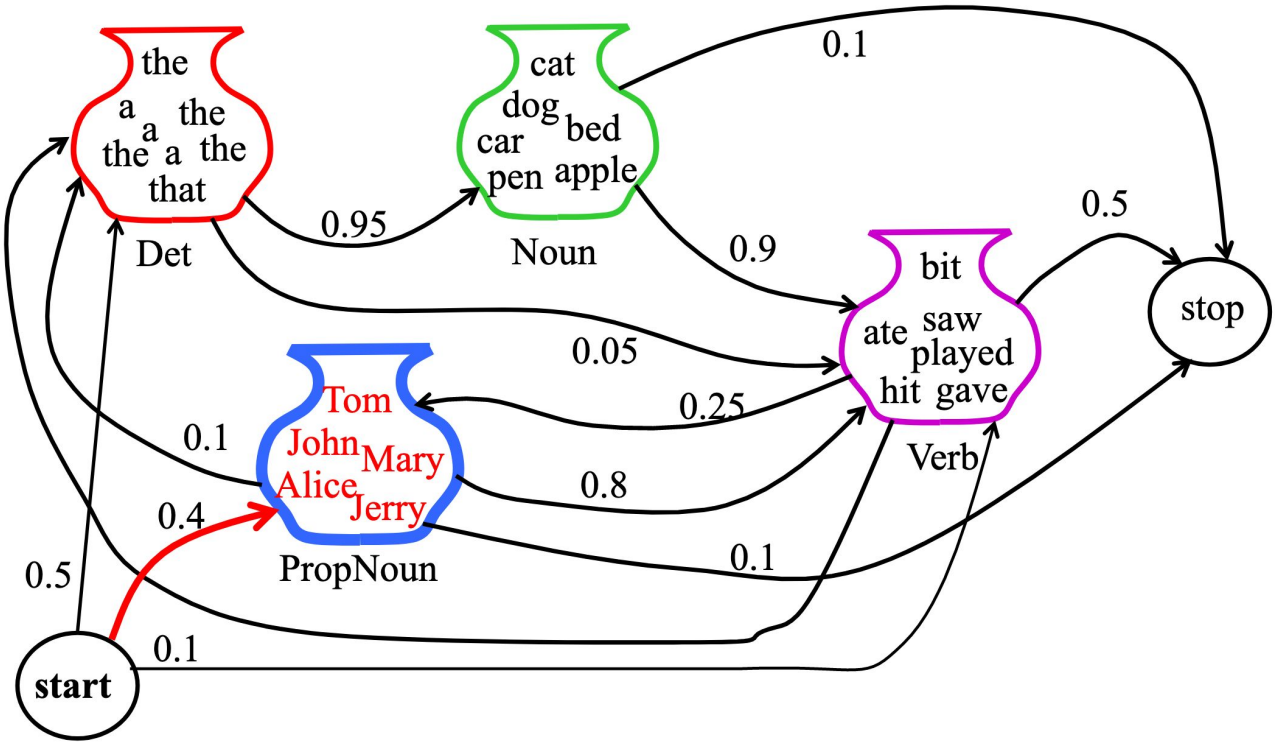
Sample HMM for POS Tagging



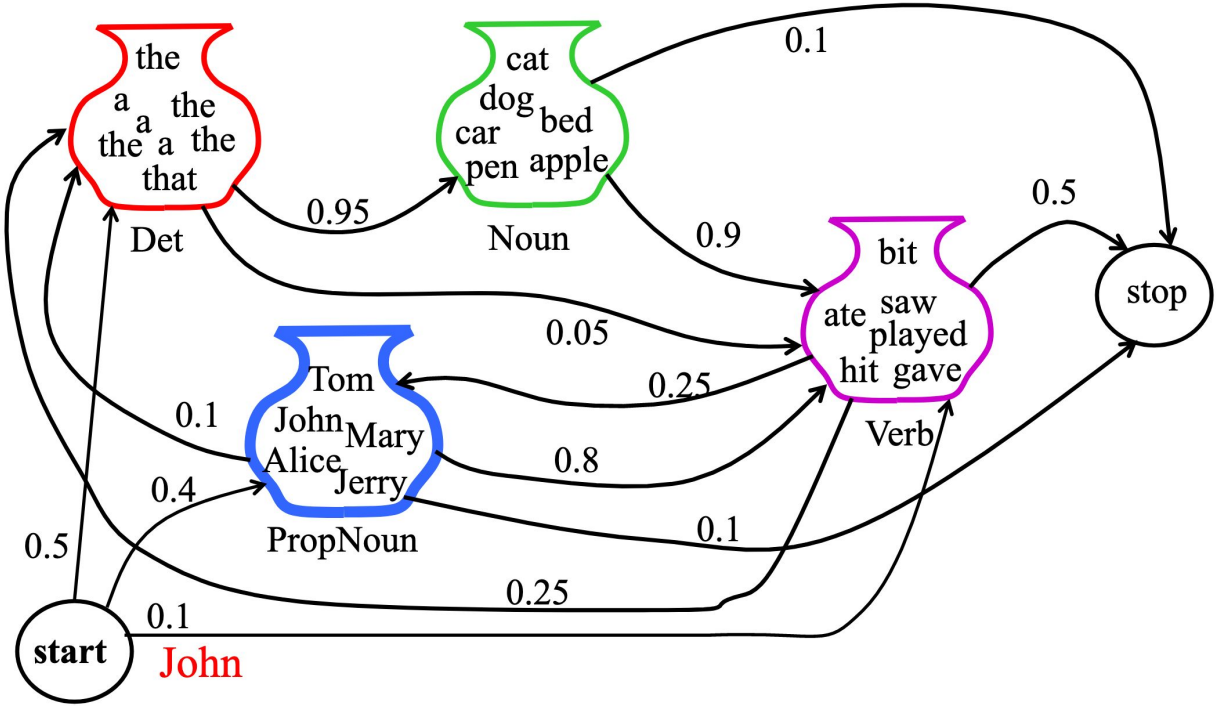
Sample HMM Generation



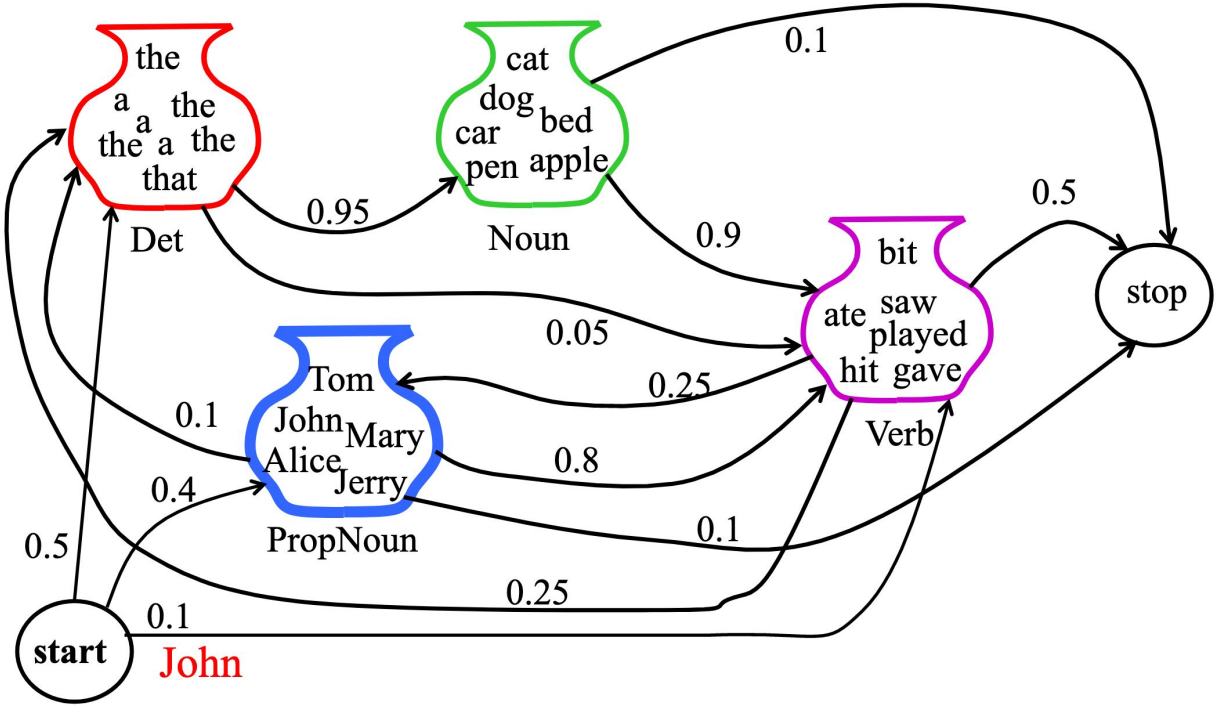
Sample HMM Generation



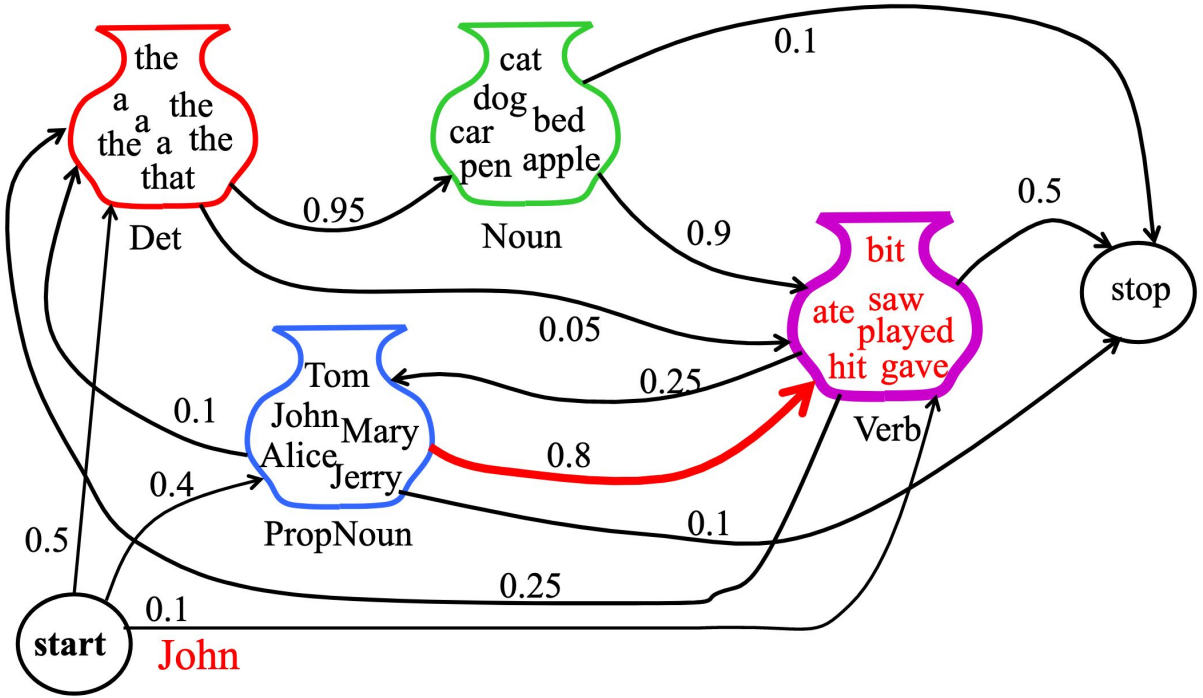
Sample HMM Generation



Sample HMM Generation



Sample HMM Generation



HMM Generation Procedure

To generate a sequence of T observations: $O = o_1 o_2 \ldots o_T$

Set initial state $q_1 = s_0$

For t = 1 to T

Transit to another state $q_{t+1} = s_j$ based on transition distribution a_{ij} for state q_t

Pick an observation $o_t = v_k$ based on being in state q_t using distribution $b_{qt}(k)$

The Three Basic Problems for HMMs

- (From the classic formulation by Larry Rabiner after Jack Ferguson)
- L. R. Rabiner. 1989. A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proc IEEE 77(2), 257-286. Also in Waibel and Lee volume.

The Three Basic Problems for HMMs

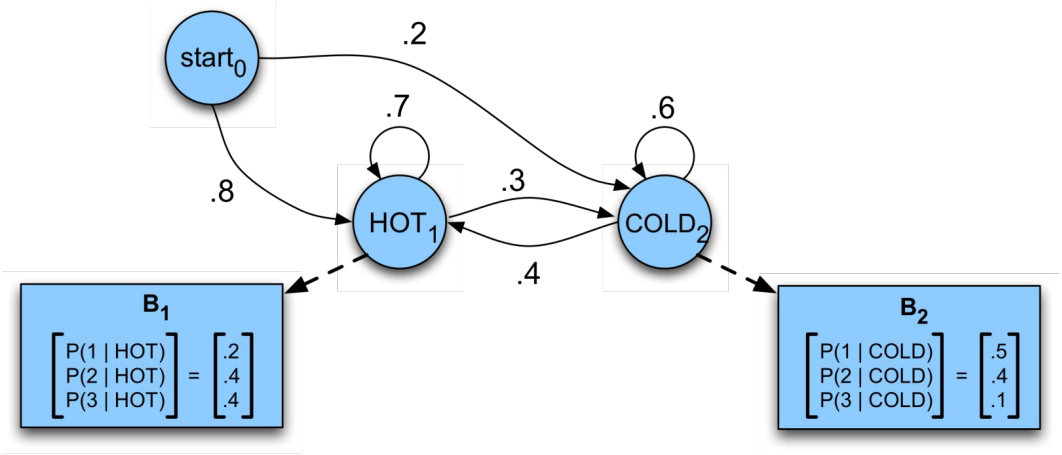
- Problem 1 (**Evaluation/Likelihood**): Given the observation sequence $O=(o_1 o_2 \ldots o_T)$, and an HMM model $\Phi = (A,B)$, how do we efficiently compute $P(O|\Phi)$, the probability of the observation sequence, given Φ
- Problem 2 (**Decoding**): Given the observation sequence $O=(o_1 o_2 \ldots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

Task 1: Observation Likelihood

- Allows HMM to be used as a language model:
  - A formal probabilistic model of a language that assigns a probability to each string saying how likely that string was to have been generated by the language.
- Useful for two tasks:
  - Sequence Classification
  - Most Likely Sequence

Problem 1: Computing the Observation Likelihood

Computing Likelihood: Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$.



How likely is the sequence 3 1 3?

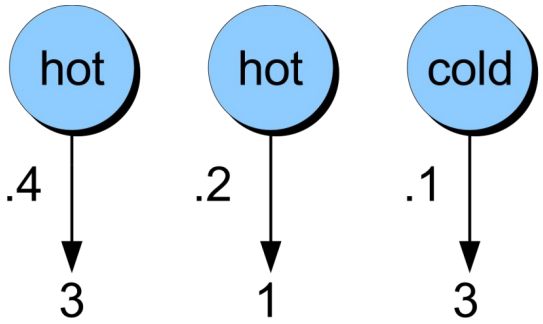
How to Compute Likelihood

- For an HMM, we don't know what the states are!
- So let's start with a simpler situation
- Computing the observation likelihood for a **given** hidden state sequence
  - Suppose we knew the weather and wanted to predict how much ice cream Jason would eat
  - i.e. P(3 1 3 | H H C)

Computing Likelihood of 3 1 3 Given Hidden State Sequence

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

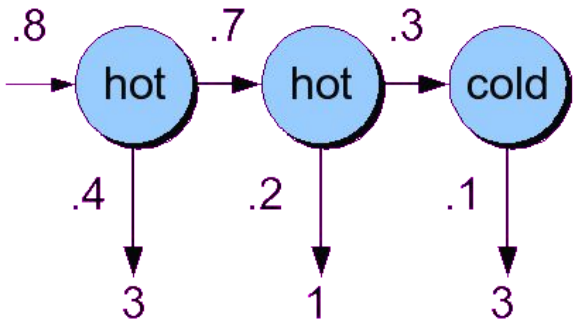
$$P(3\ 1\ 3|\text{hot hot cold}) = P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold})$$



Computing Joint Probability of Observation and a Particular State Sequence

$$P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{n} P(o_i|q_i) \times \prod_{i=1}^{n} P(q_i|q_{i-1})$$

$$\begin{aligned} P(3\ 1\ 3, \text{hot hot cold}) &= P(\text{hot}|\text{start}) \times P(\text{hot}|\text{hot}) \times P(\text{cold}|\text{hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned}$$



Computing Total Likelihood of 3 1 3

- We would need to sum over
  - Hot hot cold
  - Hot hot hot
  - Hot cold hot
  - ….

$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q)$$

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

- How many possible hidden state sequences are there for this sequence?
- How about in general for an HMM with N hidden states and a sequence of T observations?
  - N^T

Computing Observation Likelihood P(O|Φ)

- Why can't we do an explicit sum over all paths?
- What we do instead:
- The Forward Algorithm. $O(N^2T)$
- A kind of **dynamic programming** algorithm
  - Uses a table to store intermediate values
- Idea:
  - Compute the likelihood of the observation sequence by summing over all possible hidden state sequences

The Forward Algorithm

- The goal of the forward algorithm is to compute

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

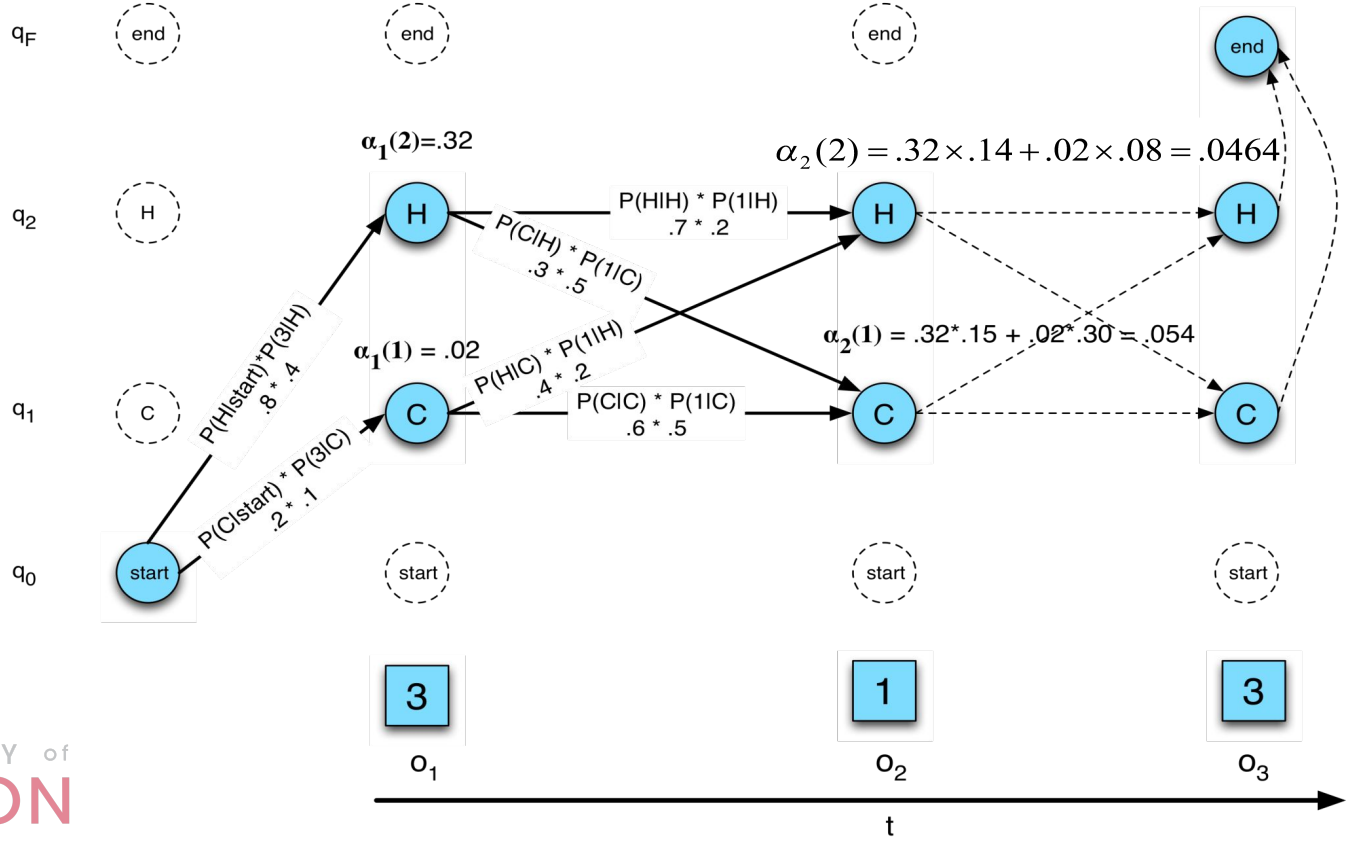
- We'll do this by recursion

The Forward Algorithm

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

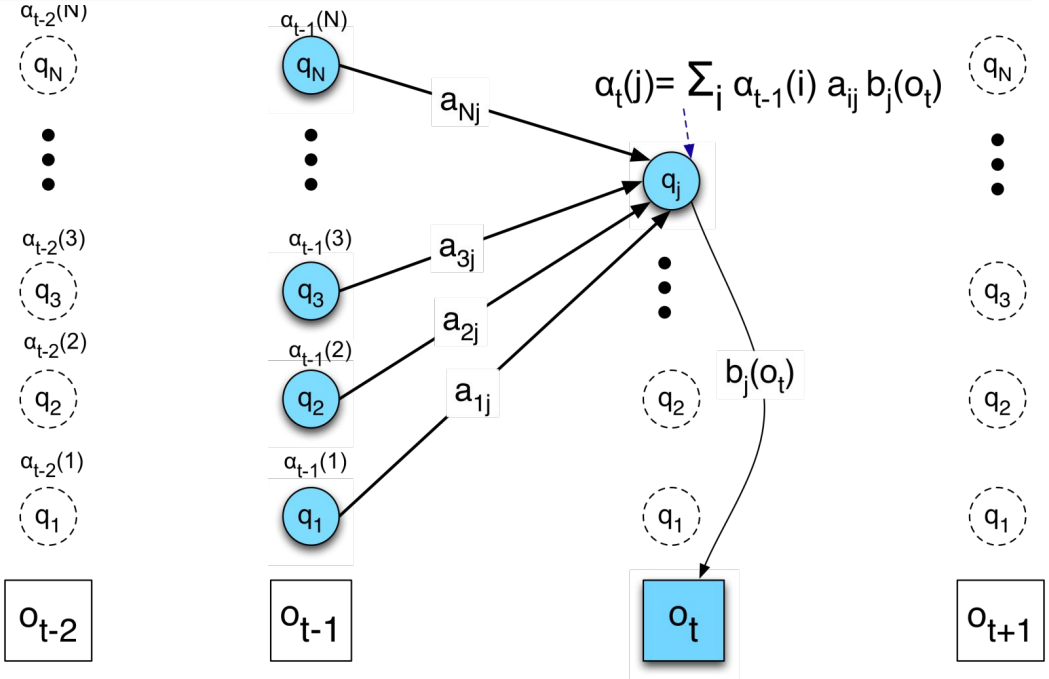
$$\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$$

The Forward Trellis



We update each cell

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



UNIVERSITY of HOUSTON

The Forward Algorithm

- The Idea: Fold these exponential paths into a simple trellis, so that all possible paths will remerge into N states at every time slice.
- We define the *forward probability* as follows: $\alpha_t(i) = P(o_0 o_1 \cdots o_t, q_t = i|\Phi)$
- this is the probability that the HMM Φ is in state i at time t having generated partial observation O_1^t.
- We compute it by induction:
  - Initialization: $\alpha_1(i) = \pi_i P(o_1|q_i), 1 \leq i \leq N$
  - (equivalently: $\alpha_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N$
  - Induction:

$$\alpha_t(j) = [\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}] b_j(o_t), \quad 2 \leq t \leq T, 1 \leq j \leq N \tag{3}$$

  - Termination: $P(O|\Phi) = \sum_{i=1}^{N} \alpha_T(i)$

The Forward Algorithm

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

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

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

$$forward[q_F,\mathrm{T}] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$$; termination step

return $forward[q_F,T]$

The Three Basic Problems for HMMs

- 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 $P(O|\Phi)$?

Decoding

- Given an observation sequence
  - up up down
- And an HMM
- The task of the **decoder**
  - To find the best **hidden** state sequence
- We can calculate P(O|path) for each path and find the best one
- But we can't do this, complexity is $O(N^T)$. Instead:
  - Viterbi Decoding: dynamic programming, slight modification of the forward algorithm

Viterbi intuition

- We want to compute the joint probability of the observation sequence together with the best state sequence

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

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

Viterbi Recursion

1. **Initialization:**

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

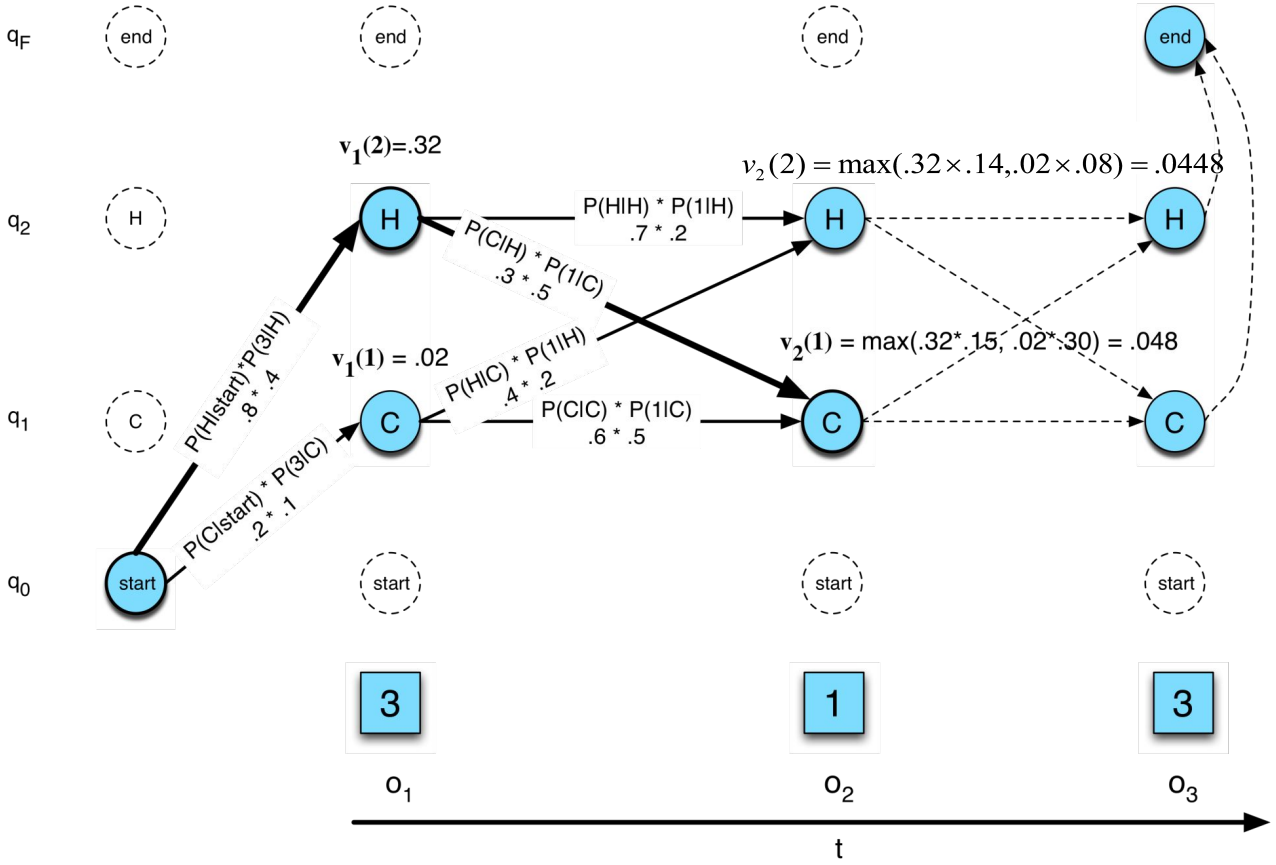
2. **Recursion** (recall that states 0 and q_F are non-emitting):

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

3. **Termination:**

$$\begin{aligned} \text{The best score:} \quad P* = v_t(q_F) &= \max_{i=1}^{N} v_T(i) * a_{i,F} \\ \text{The start of backtrace:} \quad q_T* = bt_T(q_F) &= \operatorname*{argmax}_{i=1}^{N} v_T(i) * a_{i,F} \end{aligned}$$

The Viterbi trellis



UNIVERSITY of HOUSTON

Viterbi Intuition

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

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

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

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

The Viterbi Algorithm

function VITERBI(*observations* of len T, *state-graph* of len N) **returns** *best-path*

create a path probability matrix $viterbi[N+2,T]$
for each state s **from** 1 **to** N **do** ; initialization step
  $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$; termination step
return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

So Far…

- Forward algorithm for evaluation
- Viterbi algorithm for decoding
- Next topic: the learning problem

The Three Basic Problems for HMMs

- 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 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

Starting out with Observable Markov Models

- How to train?
- Run the model on the 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 HMMs, cannot compute these counts directly from observed sequences
- Baum-Welch (forward-backward) 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
  - Two related probabilities: the forward probability and the backward probability

Recall: The Forward Algorithm

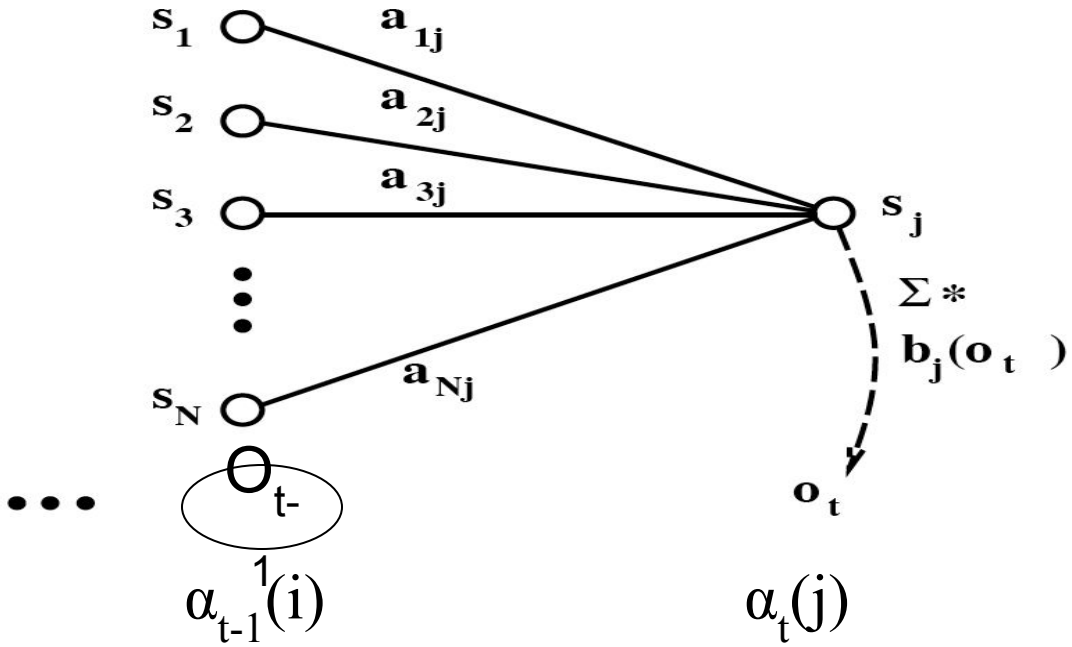
- The Idea: Fold these exponential paths into a simple trellis, so that all possible paths will remerge into N states at every time slice.
- We define the *forward probability* as follows: $\alpha_t(i) = P(o_0 o_1 \cdots o_t, q_t = i|\Phi)$
- this is the probability that the HMM Φ is in state i at time t having generated partial observation O_1^t.
- We compute it by induction:
  - Initialization: $\alpha_1(i) = \pi_i P(o_1|q_i), 1 \leq i \leq N$
  - (equivalently: $\alpha_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N$
  - Induction:

$$\alpha_t(j) = [\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}] b_j(o_t), \quad 2 \leq t \leq T, 1 \leq j \leq N \tag{3}$$

  - Termination: $P(O|\Phi) = \sum_{i=1}^{N} \alpha_T(i)$

The inductive step, from Rabiner and Juang

- Computation of $\alpha_t(j)$ by summing all previous values $\alpha_{t-1}(i)$ for all *i*



- We compute backward prob by induction:

1. **Initialization:**

$$\beta_T(i) = a_{i,F}, \quad 1 \leq i \leq N$$

2. **Recursion** (again since states 0 and q_F are non-emitting):

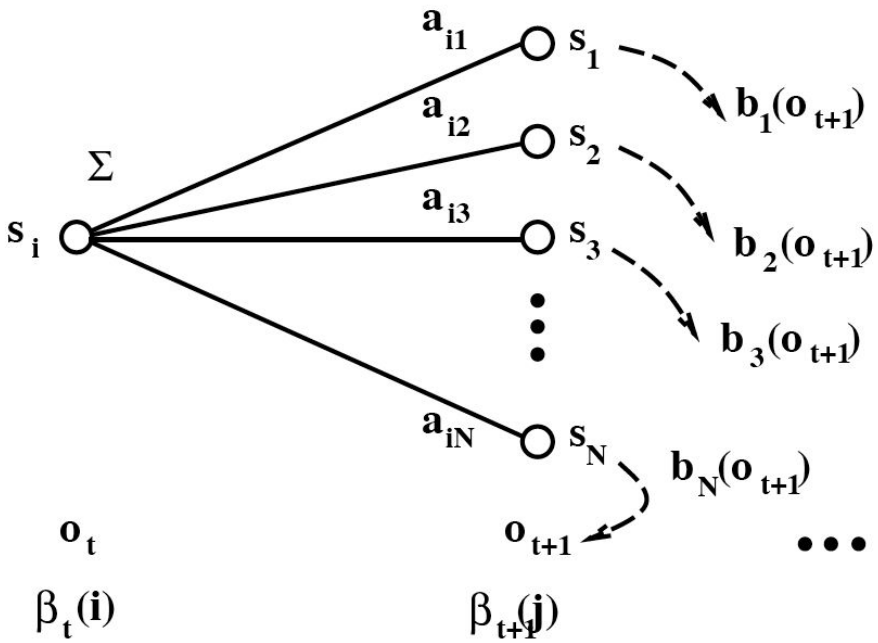
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \leq i \leq N, 1 \leq t < T$$

3. **Termination:**

$$P(O|\lambda) = \alpha_T(q_F) = \beta_1(0) = \sum_{j=1}^{N} a_{0j}\, b_j(o_1)\, \beta_1(j)$$

Inductive Step of the Backward Algorithm (Figure after Rabiner and Juang)

- Computation of $\beta_t(i)$ by weighted sum of all successive values β_{t+1}



Extending Intuition to HMMs

- For HMMs, cannot compute these counts directly from observed sequences
- Baum-Welch (forward-backward) 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
  - Two related probabilities: the forward probability and the backward probability

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 $i \square j$ 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 $i \square j$.

Re-estimation of a_{ij}

- Let γ_t be the probability of being in state *i* at time *t* and state *j* at time *t+1*, given $O_{1..T}$ and model Φ:

$$\gamma_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, \Phi)$$

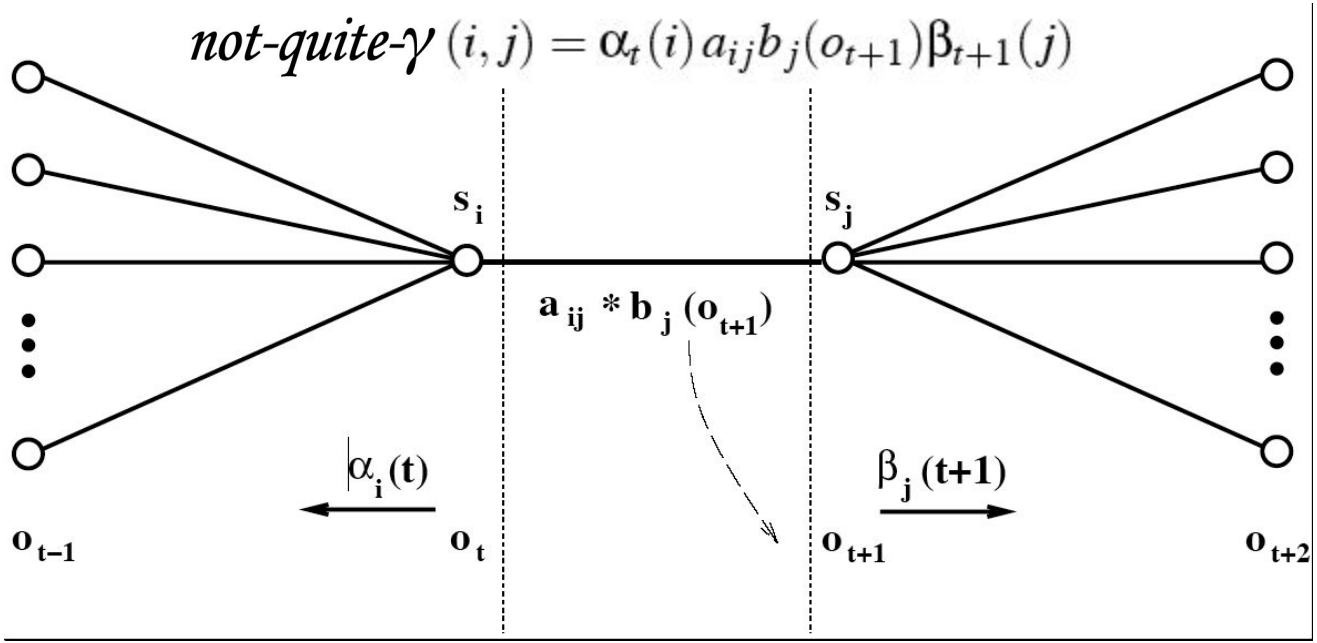
- We can compute γ from *not-quite-*γ, which is:

$$not\_quite\_\gamma_t(i,j) = P(q_t = i, q_{t+1} = j, O \mid \Phi)$$

Computing *not-quite-*γ

The four components of $P(q_t = i, q_{t+1} = j, O \mid \Phi) : \alpha, \beta, a_{ij}$ and $b_j(o_t)$

$$\textit{not-quite-}\gamma\ (i,j) = \alpha_t(i)\, a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$



From *not-quite*-γ to γ

$$\text{not-quite-}\gamma_t(i,j) = \alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)$$

$$\gamma_t(i,j) = P(q_t = i, q_{t+1} = j | O, \Phi)$$

$$\textit{not-quite-}\gamma_t(i,j) = P(q_t = i, q_{t+1} = j, O | \Phi)$$

$$P(X|O,\Phi) = \frac{P(X,O|\Phi)}{P(O|\Phi}$$

$$P(O|\Phi) = \alpha_T(N) = \beta_T(1) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j)$$

$$\gamma_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(N)}$$

From γ to a_{ij}

- $\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$
- The expected number of transitions from state i to state j is the sum over all t of γ.
- The total expected number of transitions out of state i is the sum over all transitions out of state i.
- Final formula for reestimated a_{ij}:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \gamma_t(i,j)} \qquad (14)$$

Re-estimating the Observation Likelihood b

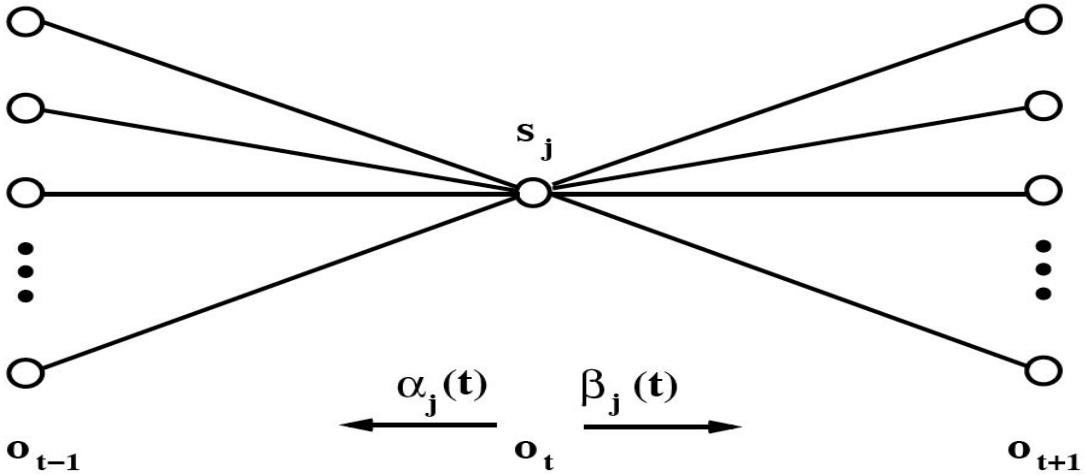
- This is the probability of a given symbol v_k from the observation vocabulary V, given a state j: $\hat{b}_j(v_k)$.

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state j and observing symbol } v_k}{\text{expected number of times in state } j}$$

- For this we will need to know the probability of being in state j at time t, which we will call $\xi_t(j)$ (ξ for **s**tate):
- $\xi_t(j) = P(q_t = j|O, \Phi)$
- We compute this by including the observation sequence in the probability and then normalizing:
- $\xi_t(j) = \frac{P(q_t=j,O|\Phi)}{P(O|\Phi)}$

Computing ξ

Computation of $\xi_j(t)$, the probability of being in state *j* at time *t.*



- $\xi_t(j) = \frac{P(q_t = j, O|\Phi)}{P(O|\Phi)}$
- $\xi_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\Phi)}$

Reestimating the observation Likelihood *b*

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state j and observing symbol } v_k}{\text{expected number of times in state } j}$$

- For numerator, $sum\,\xi_j(t)$ for all *t* in which o_t is symbol v_k

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

Summary of *A* and *B*

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

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

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

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

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** $HMM=(A,B)$

initialize A and B

iterate until convergence

E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N} \xi_t(i,k)} \qquad \hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

Summary: Forward-Backward Algorithm

1) Initialize $\Phi=(\mathcal{A},\mathcal{B},\pi)$
2) Compute α, β, ξ
3) Estimate new $\Phi'=(\mathcal{A},\mathcal{B},\pi)$
4) Replace Φ with Φ'
5) If not converged go to 2

Embedded Training of HMMs

- The entire procedure:
  1. Choose an estimate for *a* and *b*
  2. Re-estimate *a* and *b*
  3. Repeat until convergence
- How do we get initial estimates for *a* and *b*?
- For *a* we assume that from any state all the possible following states are equiprobable
- For *b* we can use a small hand-labelled training corpus

Summary

- ★ Baum-Welch algorithm for learning the A and B matrices of an individual HMM
- ★ It doesn't require training data to be labeled at the state level; all you have to know is that an HMM covers a given sequence of observations, and you can learn the optimal A and B parameters for this data by an iterative process.

The Learning Problem: 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.
- ★ Baum-Welch only guaranteed to find a local max, rather than global optimum