Hidden Markov Models

Slides by James Martin, adapted by Diana Inkpen
for CSI 5386 @ uOttawa

Hidden Markov Models

- States Q = $q_1, q_2 \ldots q_{N;}$
- Observations O= $o_1, o_2 \ldots o_{N;}$
  - Each observation is a symbol from a vocabulary V = $\{v_1, v_2, \ldots v_V\}$
- Transition probabilities
  - Transition probability matrix A = $\{a_{ij}\}$

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

- Observation likelihoods
  - Output probability matrix B=$\{b_i(k)\}$

$$b_i(k) = P(X_t = o_k \mid q_t = i)$$

- Special initial probability vector π

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

HMMs for Ice Cream

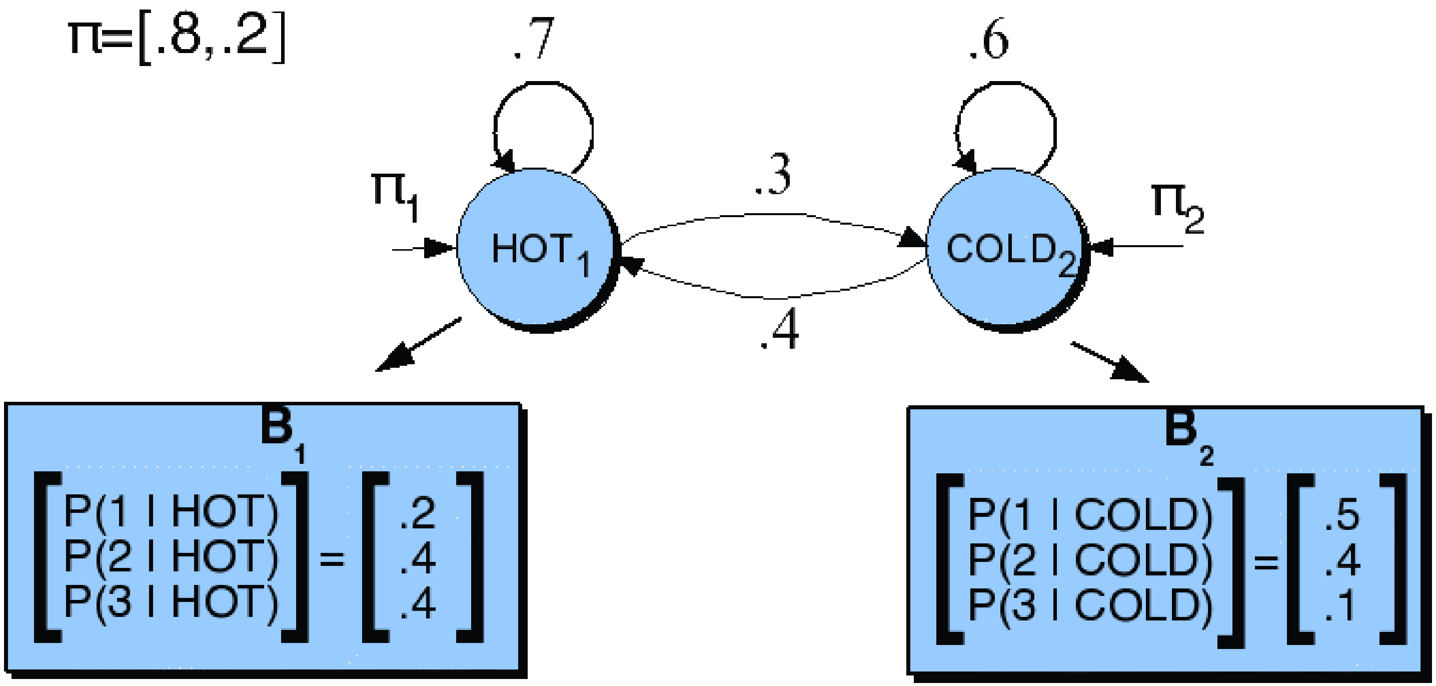
- You are a climatologist in the year 2799 studying global warming
- You can't find any records of the weather in Baltimore for summer of 2007
- But you find Jason Eisner's diary which lists how many ice-creams Jason ate every day that summer
- Your job: figure out how hot it was each day

Eisner Task

- Given
  - Ice Cream Observation Sequence: 1,2,3,2,2,2,3…
- Produce:
  - Hidden Weather Sequence:

    H,C,H,H,H,C, C…

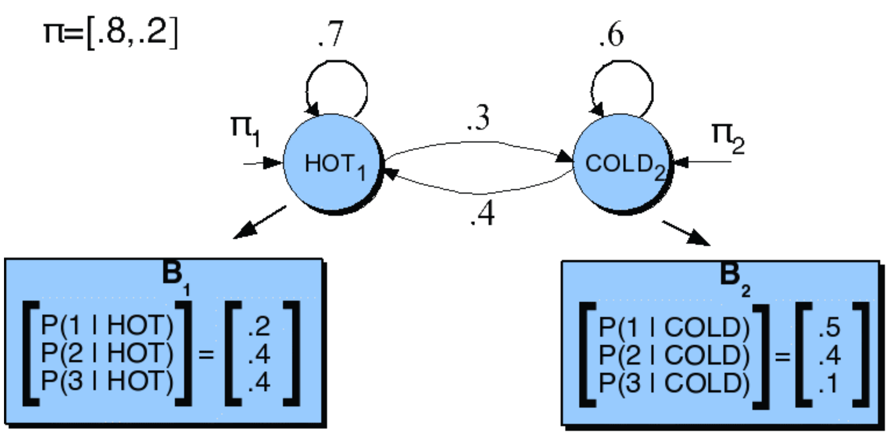
HMM for Ice Cream



Ice Cream HMM

- Let's just do 131 as the sequence
  - How many underlying state (hot/cold) sequences are there?

HHH
HHC
HCH
HCC
CCC
CCH
CHC
CHH



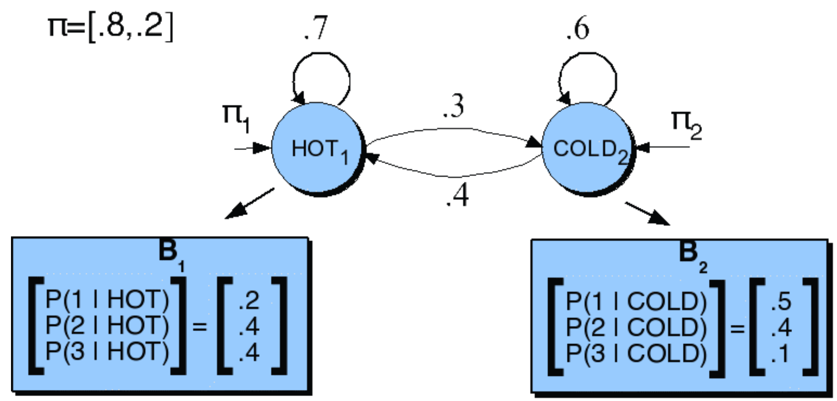
  - How do you pick the right one?

Argmax P(sequence | 1 3 1)

Ice Cream HMM

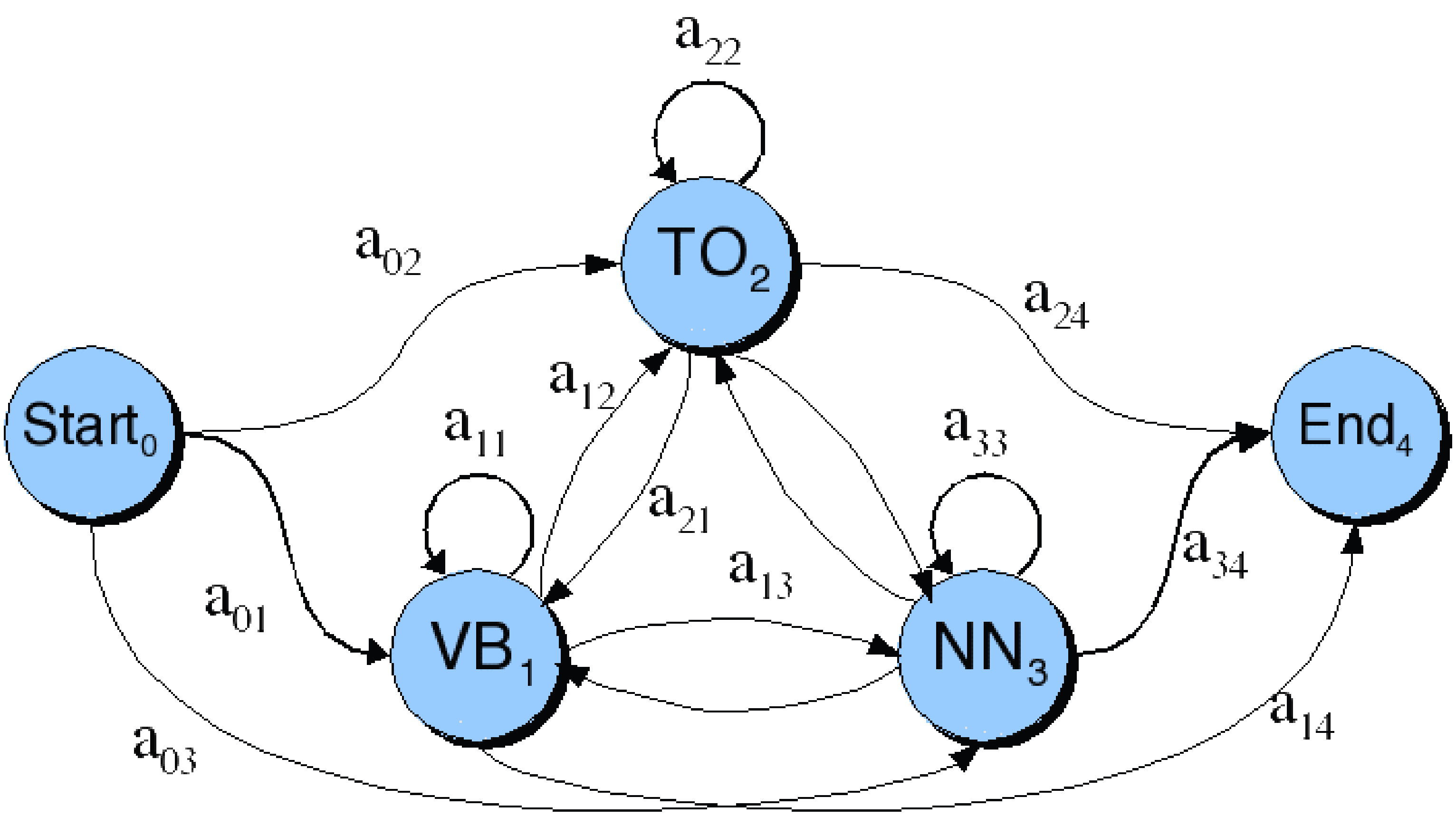
Let's just do 1 sequence: CHC

Cold as the initial state P(Cold|Start)	.2
Observing a 1 on a cold day P(1 | Cold)	.5
Hot as the next state P(Hot | Cold)	.4
Observing a 3 on a hot day P(3 | Hot)	.4
Cold as the next state P(Cold|Hot)	.3
Observing a 1 on a cold day P(1 | Cold)	.5

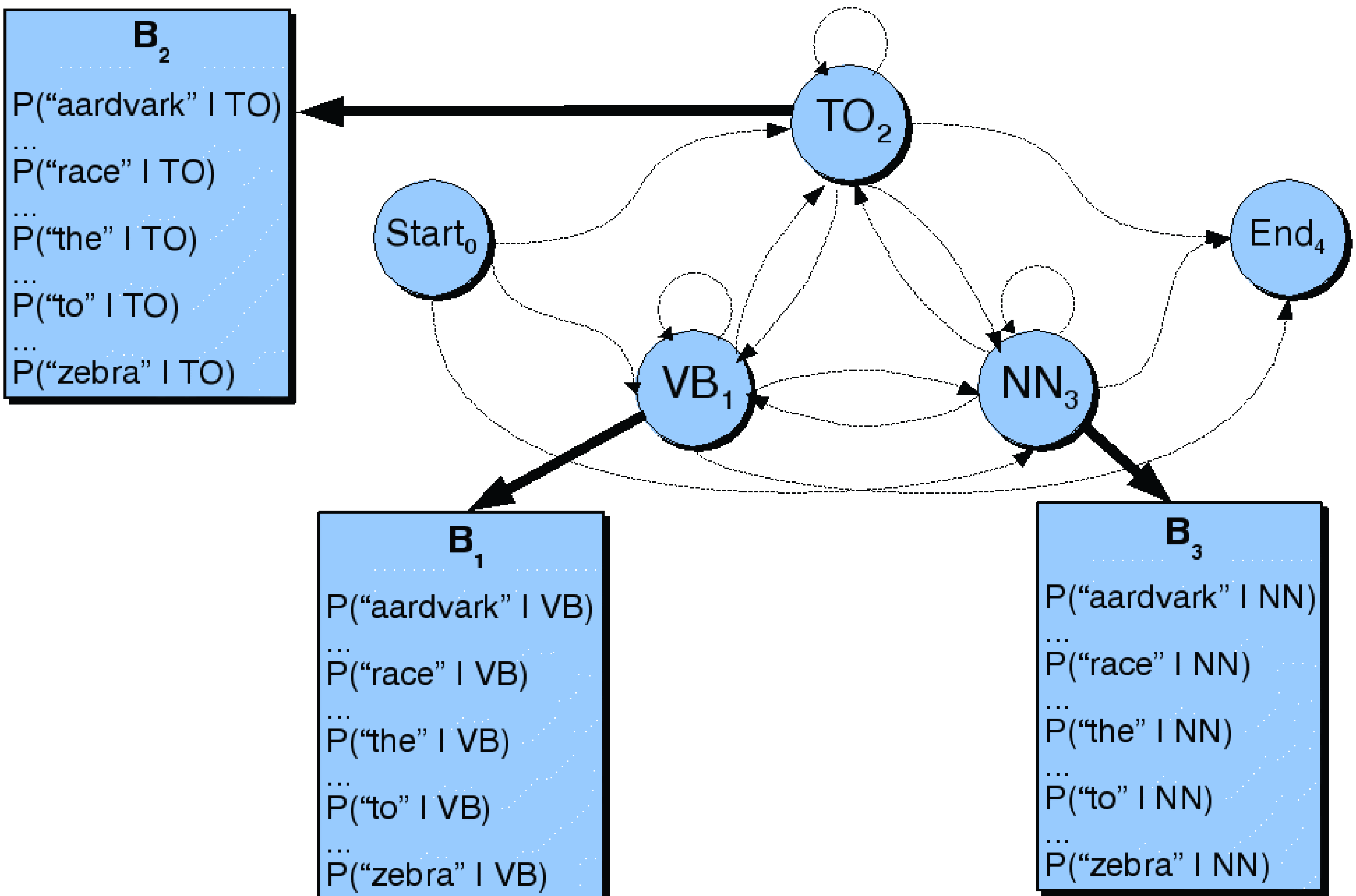


.0024

POS Transition Probabilities



Observation Likelihoods



Question

- If there are 30 or so tags in the Penn set
- And the average sentence is around 20 words...
- How many tag sequences do we have to enumerate to argmax over in the worst case scenario?

$$30^{20}$$

3 Problems

- Given this framework there are 3 problems that we can pose to an HMM
  - Given an observation sequence, what is the probability of that sequence given a model?
  - Given an observation sequence and a model, what is the most likely state sequence?
  - Given an observation sequence, infer the best model parameters for a skeletal model

Problem 1

- The probability of a sequence given a model...

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

- Used in model development... How do I know if some change I made to the model is making it better
- And in classification tasks
  - Word spotting in ASR, language identification, speaker identification, author identification, etc.
    - Train one HMM model per class
    - Given an observation, pass it to each model and compute P(seq|model).

Problem 2

- Most probable state sequence given a model and an observation sequence

Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

  - Typically used in tagging problems, where the tags correspond to hidden states
    - As we'll see almost any problem can be cast as a sequence labeling problem
  - Viterbi solves problem 2

Problem 3

- Infer the best model parameters, given a skeletal model and an observation sequence...
  - That is, fill in the A and B tables with the right numbers...
    - The numbers that make the observation sequence most likely
  - Useful for getting an HMM without having to hire annotators...
    - That is you tell me how many tags there are and give me a boatload of untagged text, and I give you back a part of speech tagger.

Solutions

- Problem 2: Viterbi
- Problem 1: Forward
- Problem 3: Forward-Backward (or Baum-Welch)
  - An instance of EM

Problem 2: Decoding

- Ok, now we have a complete model that can give us what we need. Recall that we need to get

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

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

Intuition

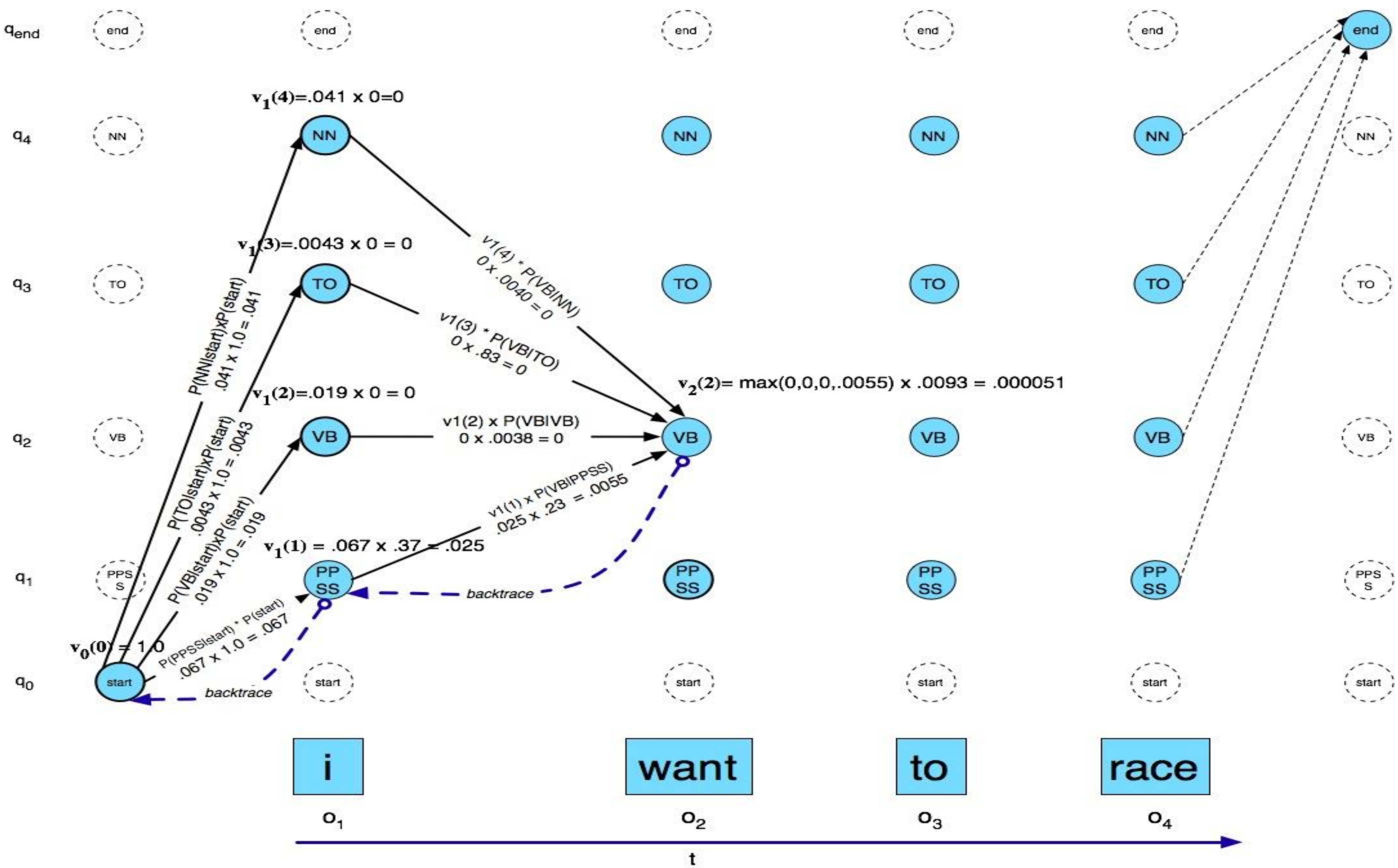
- You're interested in the shortest distance from Boulder to Moab
- Consider a possible location on the way to Moab, say Glenwood Springs.
- What do you need to know about all the different possible ways to get to Glenwood Springs?

The best way (the shortest path)

Intuition

- Consider a state sequence (tag sequence) that ends at state j (i.e., has a particular tag T at the end)
- The probability of that tag sequence can be broken into parts
  - The probability of the BEST tag sequence up through j-1
  - Multiplied by the transition probability from the tag at the end of the j-1 sequence to T.
  - And the observation probability of the observed word given tag T

Viterbi Example



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

$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$

$backpointer[\mathrm{s},1] \leftarrow 0$

for each time step t **from** 2 **to** T **do** ; recursion step

for each state s **from** 1 **to** N **do**

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

$$backpointer[\mathrm{s,t}] \leftarrow \operatorname*{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$$

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

$$backpointer[q_F,\mathrm{T}] \leftarrow \operatorname*{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$$; termination step

return the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$

Viterbi Summary

- Create an array
  - With columns corresponding to inputs
  - Rows corresponding to possible states
- Sweep through the array in one pass filling the columns left to right using our transition probs and observations probs
- Dynamic programming key is that we need only store the MAX prob and path to each cell, (not all paths).

Evaluation

- So once you have you POS tagger running how do you evaluate it?
  - Overall error rate with respect to a gold-standard test set
    - Each token gets a tag, so overall accuracy is a decent measure (number correct/number tagged)
  - But to improve a system we want more detailed information
    - Per word accuracy
    - Confusion matrices

Evaluation

- Results are compared with a manually coded “Gold Standard”
  - Typically accuracy reaches 96-97%
  - This may be compared with result for a baseline tagger (one that uses no context)
- Important: 100% accuracy is impossible even for human annotators
  - Goal is to get system performance near to human performance
    - Beware of claims from systems that claim to exceed the accuracy of human annotators

Detailed Error Analysis

- Look at a confusion matrix

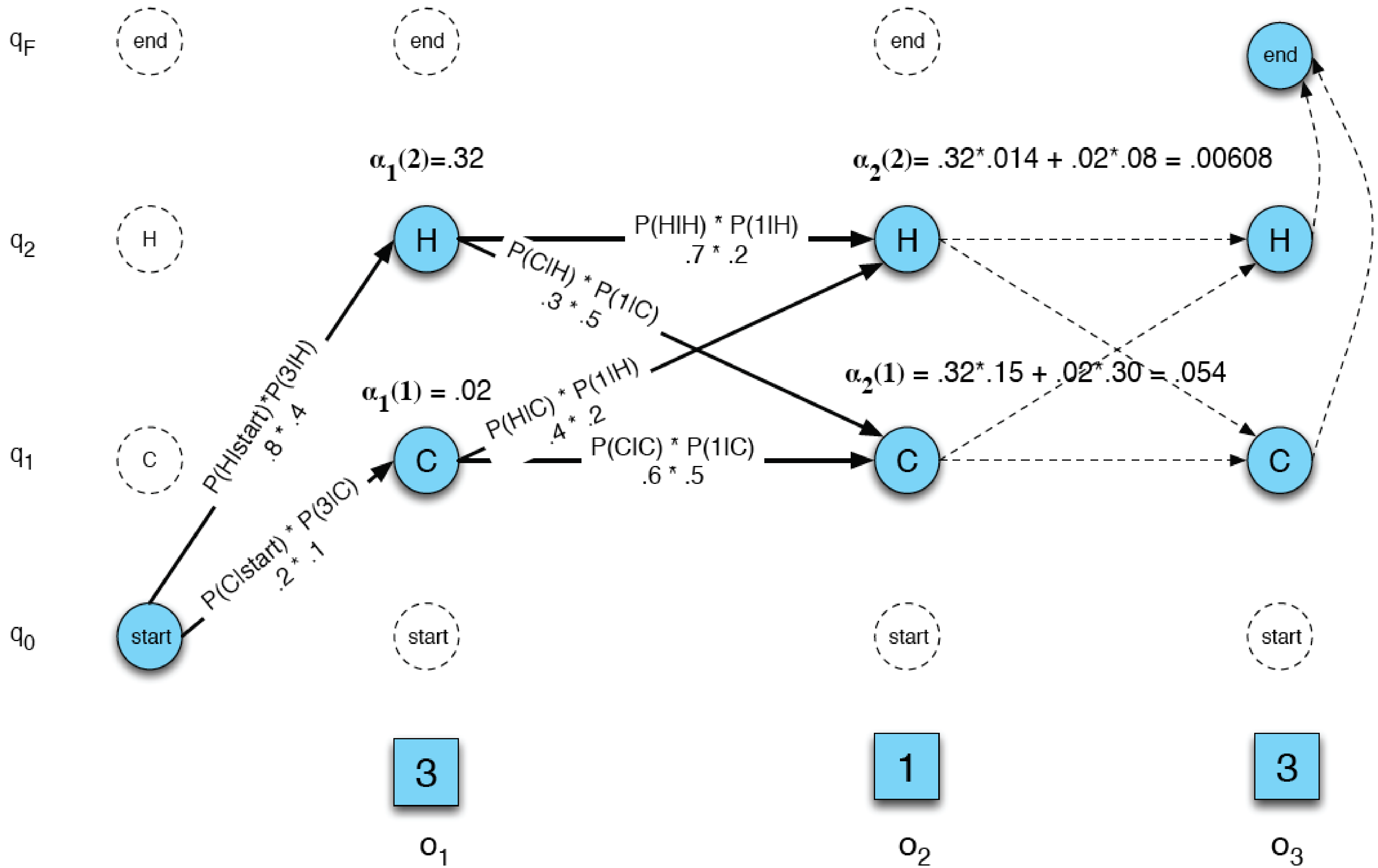
	IN	JJ	NN	NNP	RB	VBD	VBN
IN	—	.2			.7		
JJ	.2	—	**3.3**	2.1	1.7	.2	**2.7**
NN		**8.7**	—				.2
NNP	.2	**3.3**	**4.1**	—	.2		
RB	**2.2**	2.0	.5		—		
VBD		.3	.5			—	**4.4**
VBN		**2.8**				**2.6**	—

- See what errors are causing problems
  - Noun (NN) vs ProperNoun (NNP) vs Adj (JJ)
  - Preterite (VBD) vs Participle (VBN) vs Adjective (JJ)

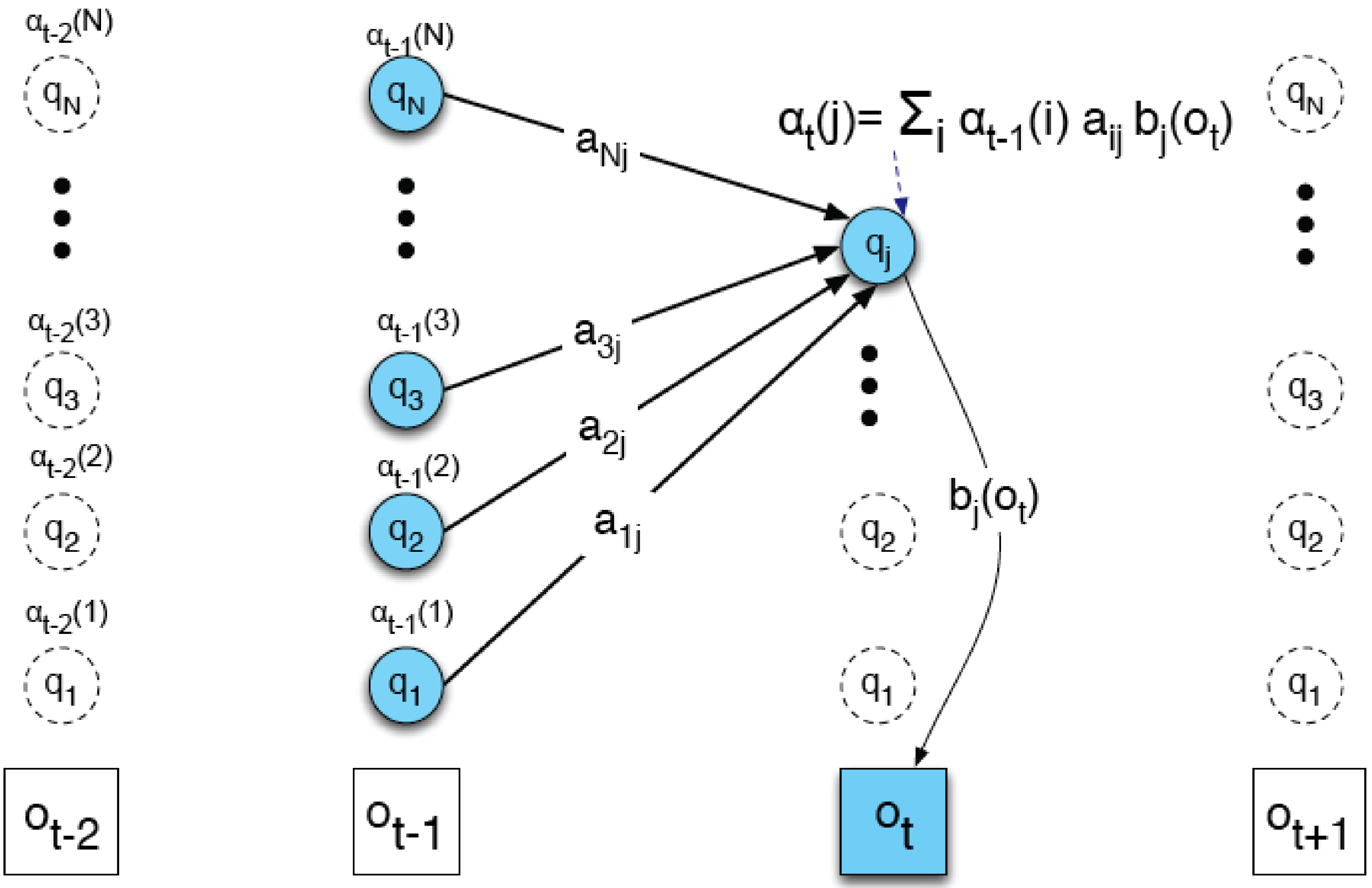
Problem 1: Forward

- Efficiently computes the probability of an observed sequence given a model
  - P(sequence|model)
- Nearly identical to Viterbi; replace the MAX with a SUM

Ice Cream Example



Ice Cream Example



Forward

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 $forward[q_F,T]$

Problem 3: Forward-Backward

Learning: Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B.

- **Baum-Welch** = **Forward-Backward Algorithm** (Baum 1972)
- Is a special case of the EM or Expectation-Maximization algorithm (Dempster, Laird, Rubin)
- The algorithm will let us train the transition probabilities $A = \{a_{ij}\}$ and the emission probabilities $B=\{b_i(o_t)\}$ of the HMM

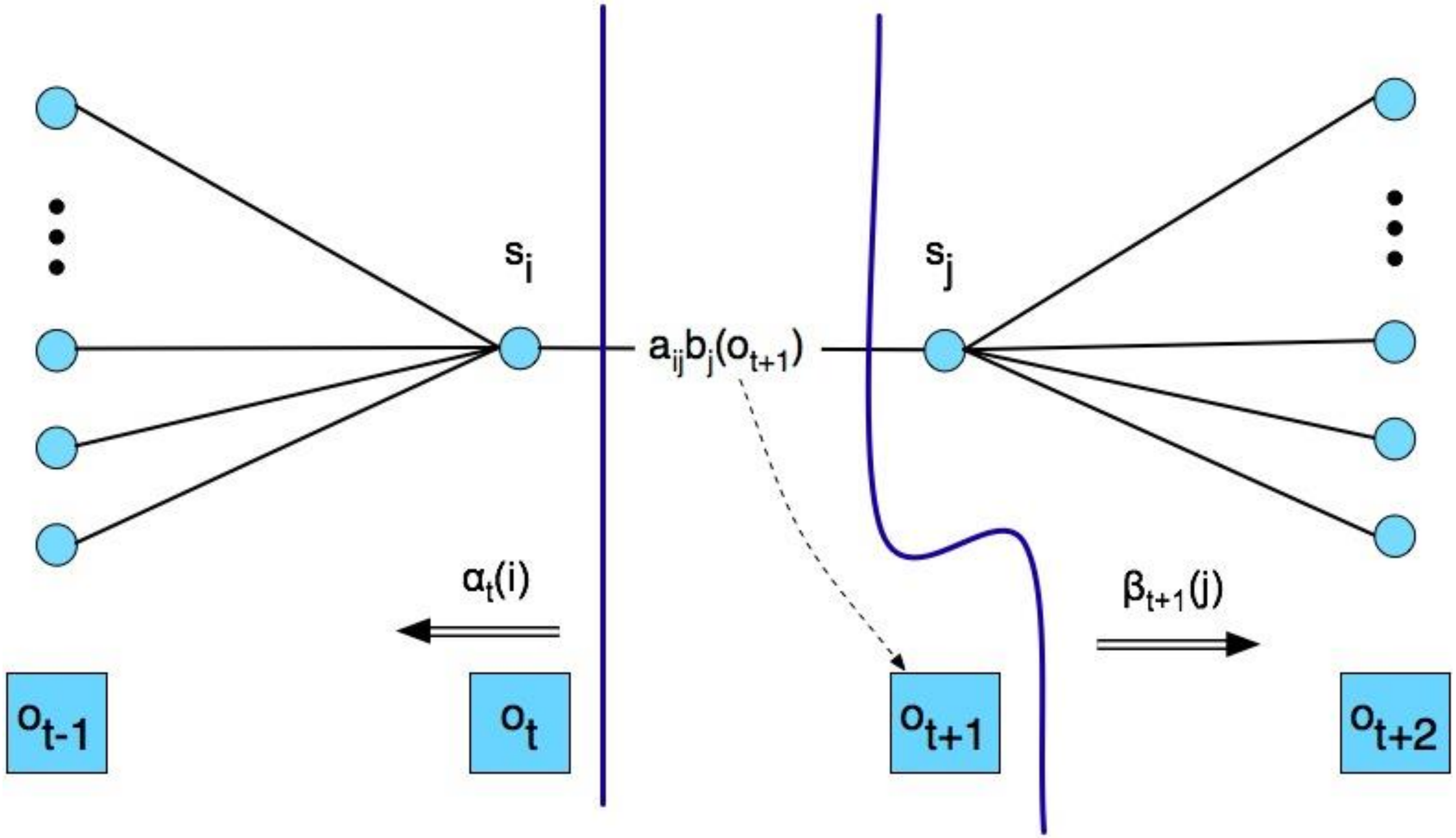
Intuition for re-estimation of a_{ij}

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

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

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

Intuition for re-estimation of a_{ij}



Re-estimating 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 ξ
- 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}

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

The Forward-Backward Alg

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

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

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

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) Intialize Φ=(A,B)
2) Compute α, β, ξ using observations
3) Estimate new Φ'=(A,B)
4) Replace Φ with Φ'
5) If not converged go to 2