BITS Pilani
Pilani Campus

Natural Language Processing DSECL ZG565

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

innovate achieve lead

BITS Pilani
Pilani Campus

Session 6
Date – 30 December 2023

These slides are prepared by the instructor, with grateful acknowledgement of Jurafsky and Martin d many others who made their course materials freely available online.

Session Content

- The Hidden Markov Model
- Likelihood Computation:
    - The Forward Algorithm
- Decoding: The Viterbi Algorithm
- HMM Training:
    - The Forward-Backward Algorithm
- MEMM algorithm

Hidden Markov Models

- It is a **sequence model**.
- Assigns a label or class to each unit in a sequence, thus mapping a **sequence of observations** to a **sequence of labels**.
- Probabilistic sequence model: given a sequence of units (e.g. words, letters, morphemes, sentences), compute a probability distribution over possible sequences of labels and choose the best label sequence.
- This is a kind of *generative* model.

Hidden Markov Model (HMM)

- Oftentimes we want to know what produced the sequence – the **hidden sequence** for the **observed sequence**. For example,
  - Inferring the words (hidden) from acoustic signal (observed) in speech recognition
  - Assigning part-of-speech tags (hidden) to a sentence (sequence of words) – **POS tagging**.
  - Assigning named entity categories (hidden) to a sentence (sequence of words) – Named Entity Recognition.

HMM Applications

- Speech Recognition including siri
- Gene Prediction
- Handwriting recognition
- Transportation forecasting
- Computational finance
- Cryptanalysis (security)

And all applications which requires sequence processing…

Definition of HMM

- 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 $A = \{a_{ij}\}$

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

- 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

$$p_i = P(q_1 = i) \quad 1 \pounds i \pounds N$$

Three Problems

- Given this framework there are 3 problems that we can pose to an HMM
  1. Given an HMM $\lambda = (A,B,\Pi)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$
  2. Given an observation sequence O and a model $\lambda = (A,B,\Pi)$, what is the most likely state sequence?
  3. Given an observation sequence, find the best model parameters for a partially specified model

Problem 1: Observation Likelihood

- Given an HMM $\lambda = (A,B,\Pi)$ 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 things 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: Decoding

- 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

Problem 3: Learning

- Infer the best model parameters, given a partial 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
- This is to learn the probabilities!

Solutions

- Problem 1: Forward (learn observation sequence)
- Problem 2: Viterbi (learn state sequence)
- Problem 3: Forward-Backward (learn probabilities)
  - An instance of EM (Expectation Maximization)

Problem 2: Decoding

- We want, out of all sequences of n tags $t_1 \ldots t_n$ the single tag sequence such that

$$P(t_1 \ldots t_n | w_1 \ldots w_n) \text{ is highest.}$$

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

- Hat ^ means “our estimate of the best one”
- Argmax_x f(x) means “the x such that f(x) is maximized”

Getting to HMMs

- This equation should give us the best tag sequence

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

- But how to make it operational? How to compute this value?
- Intuition of Bayesian inference:
  - Use Bayes rule to transform this equation into a set of probabilities that are easier to compute (and give the right answer)

Using Bayes Rule

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Know this.

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

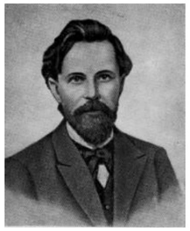
$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

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

Likelihood and Prior

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \overbrace{P(w_1^n|t_1^n)}^{\text{likelihood}} \; \overbrace{P(t_1^n)}^{\text{prior}}$$

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$



$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$

$$\boxed{\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})}$$

Markov Assumption

- Context model (prior)

$$P(t_1, \ldots, t_n) = \prod_{i=1}^{n} P(t_i | t_{i-k}, \ldots, t_{i-1})$$

- Lexical model (likelihood)

$$P(w_1, \ldots, w_n \,|\, t_1, \ldots, t_n) = \prod_{i=1}^{n} P(w_i | t_i)$$

Model Parameters

- Contextual probabilities : $P(t_i|t_{i-k}, \ldots, t_{i-1})$
- Lexical probabilities : $P(w_i|t_i)$
- We can estimate these probabilities from a tagged corpus:

$$\hat{P}_{\text{MLE}}(w_i|t_i) = \frac{c(w_i, t_i)}{c(t_i)} \qquad \hat{P}_{\text{MLE}}(t_i|t_{i-k}, \ldots, t_{i-1}) = \frac{c(t_{i-k}, \ldots, t_{i-1}, t_i)}{c(t_{i-k}, \ldots, t_{i-1})}$$

Computing Probabilities

- The probability of a tagging:

$$P(t_1, \ldots, t_n, w_1, \ldots, w_n) = \prod_{i=1}^{n} P(t_i | t_{i-k}, \ldots, t_{i-1}) P(w_i | t_i)$$

- Finding the most probable tagging:

$$\underset{t_1, \ldots, t_n}{\operatorname{argmax}} \prod_{i=1}^{n} P(t_i | t_{i-k}, \ldots, t_{i-1}) P(w_i | t_i)$$

Example

- Given a sentence of length 3, * * the dog barks STOP and the tag sequence * * DT NN VB * then

- P(w1 w2 w3,y1,y2,y3) = T(DT|*,*) X T(NN|*,DT) X T(VB|DT NN) X T(STOP|NN VB) X E(*|*) X E(*|*) E(the|DT) X E(dog|NN) X E(barks|VB)X E(STOP|*)

- We can also define y_{-1}=* and y_0=* as special symbols.

Hidden Markov Models (formal)

States T = $t_1, t_2 \ldots t_{N;}$

Observations W= $w_1, w_2 \ldots w_{N;}$

- Each observation is a symbol from a vocabulary V = $\{v_1, v_2, \ldots v_V\}$

Transition probabilities

- Transition probability matrix A = $\{a_{ij}\}$

$$a_{ij} = P(t_i = j \mid t_{i-1} = i) \quad 1 \le i, j \le N$$

Observation likelihoods

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

$$b_i(k) = P(w_i = v_k \mid t_i = i)$$

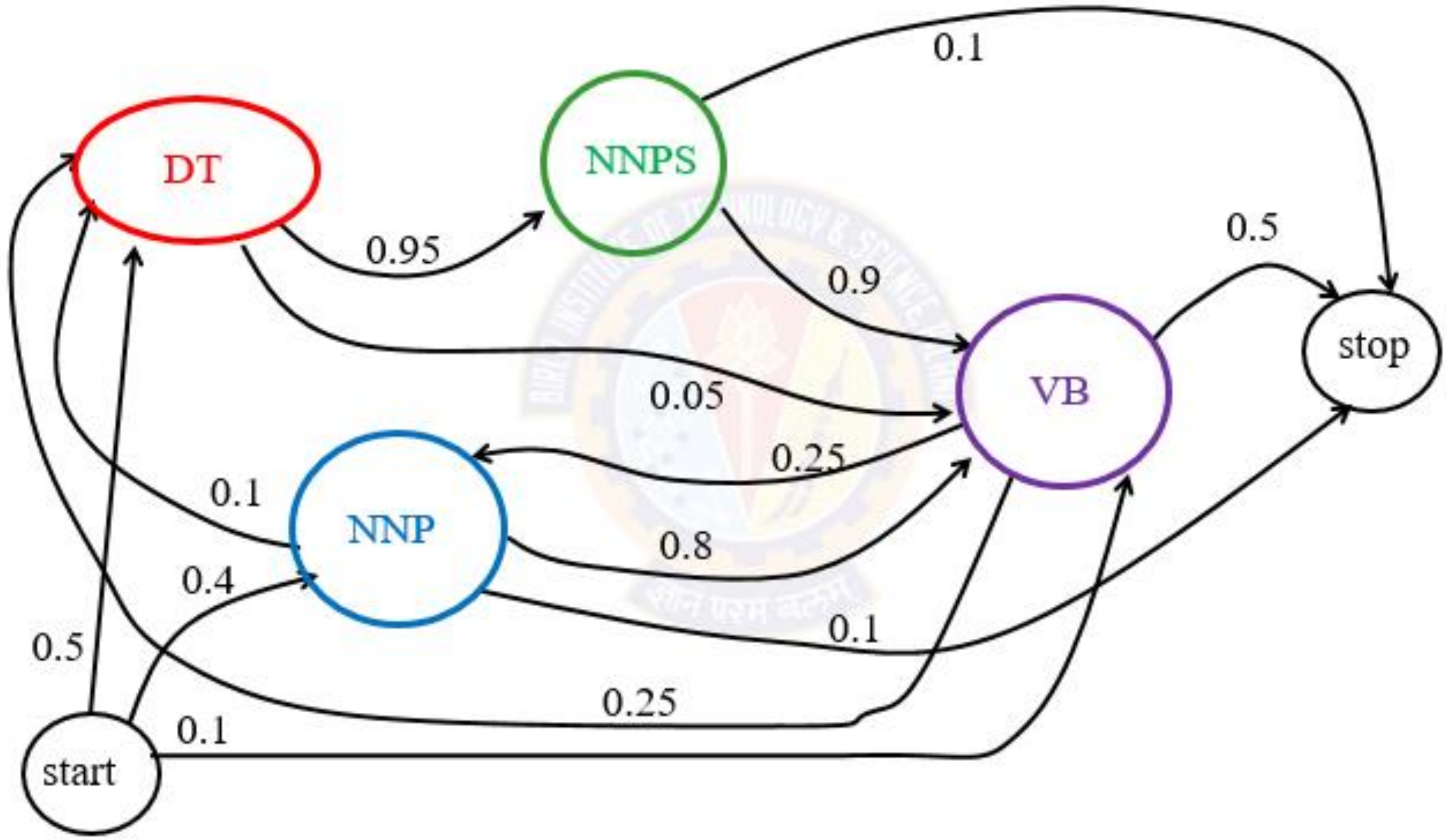
Special initial probability vector π

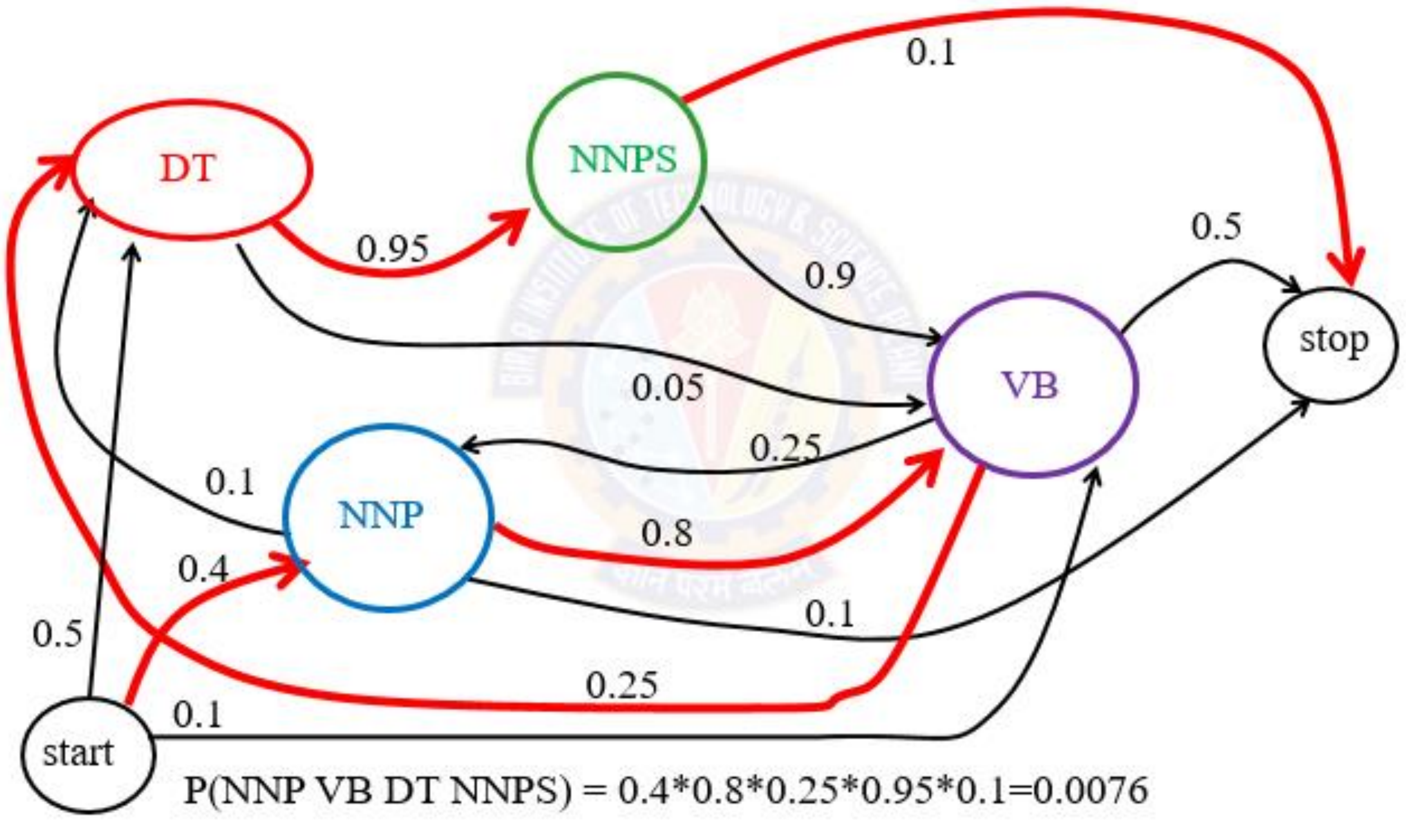
$$\pi_i = P(t_1 = i) \quad 1 \le i \le N$$

Markov Chain

- Formally, a Markov chain is specified by the following components:

$Q = q_1 q_2 \ldots q_N$ a set of N states	A set of N states
$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$	A transition probability matrix A, each a_{ij} representing the probability of moving from state i to state j,
$\pi\pi = \pi_1, \pi_2, \pi_3, \ldots, \pi\pi_n$	An initial probability distribution over states. pi is the probability that the Markov chain will start in state i. Some states j may have pj =0, meaning that they cannot be initial states.


DT
NNPS
VB
stop
NNP
start
0.95
0.1
0.9
0.5
0.05
0.25
0.1
0.8
0.1
0.4
0.5
0.25
0.1



P(NNP VB DT NNPS) = 0.4*0.8*0.25*0.95*0.1=0.0076

Example

Emission Matrix

	*	DT	NNS	VB	NN	IN	STOP
*	1						
the		3/4					
employees			3/4				
pass				2/4			
an		1/4					
exam					1		
wait				1/4			
for						1	
employers			1/4				
fire				1/4			
.							1

Tag Translation Matrix

SECOND TAG

FRIST TAG

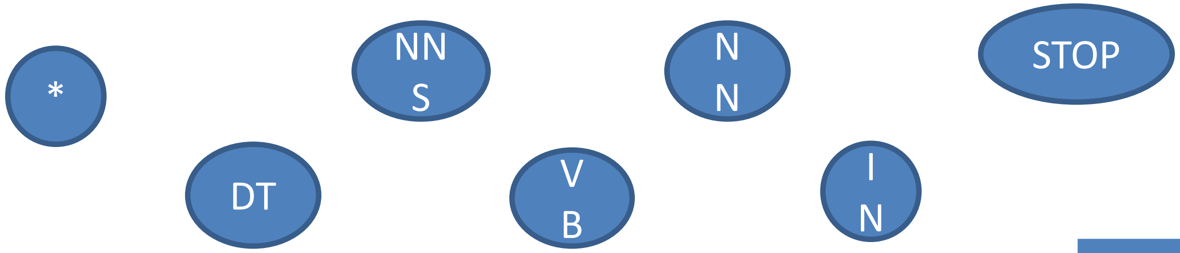
	*	DT	NNS	VB	NN	IN	STOP
*		2/3	1/3				
DT			2/4	1/4	1/4		
NNS				3/4			1/4
VB		1/4	1/4			1/4	1/4
NN							1
IN		1					
STOP							

S1: the Employees pass an exam .
T1: DT NNS VB DT NN STOP

S2: the employees wait for the pass .
T2: DT NNS VB IN DT VB STOP

S3: employers fire employees .
T3: NNS VB NNS STOP

Transition diagram



Tag Translation Matrix

SECOND TAG

FRIST TAG

	*	DT	NNS	VB	NN	IN	STOP
*		2/3	1/3				
DT			2/4	1/4	1/4		
NNS				3/4			1/4
VB		1/4	1/4			1/4	1/4
NN							1
IN		1					
STOP							

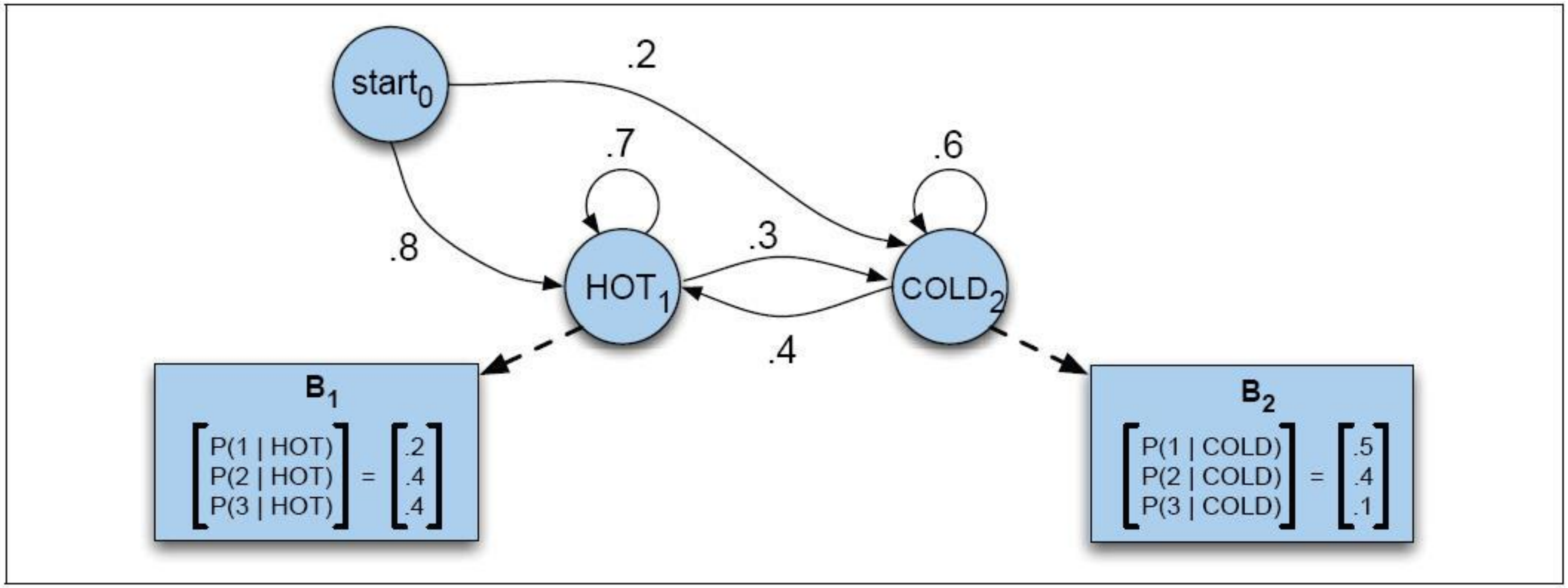
HMMs for Ice Cream

- 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:

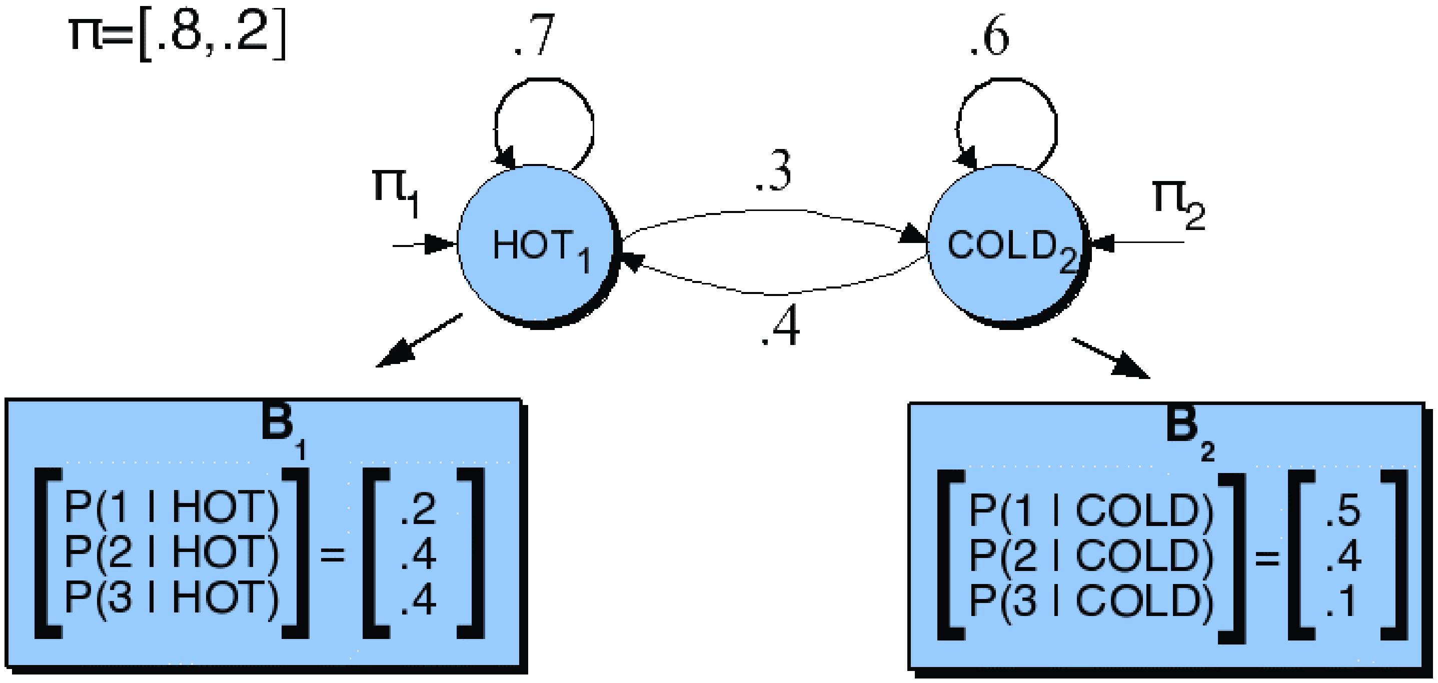
    H,C,H,H,H,C, C…



start0
.2
.7
.6
.8
.3
HOT1
COLD2
.4
B1
P(1 | HOT) P(2 | HOT) P(3 | HOT) = .2 .4 .4
B2
P(1 | COLD) P(2 | COLD) P(3 | COLD) = .5 .4 .1

What’s the state sequence for the observed sequence “1 3 1”?

HMM for Ice Cream

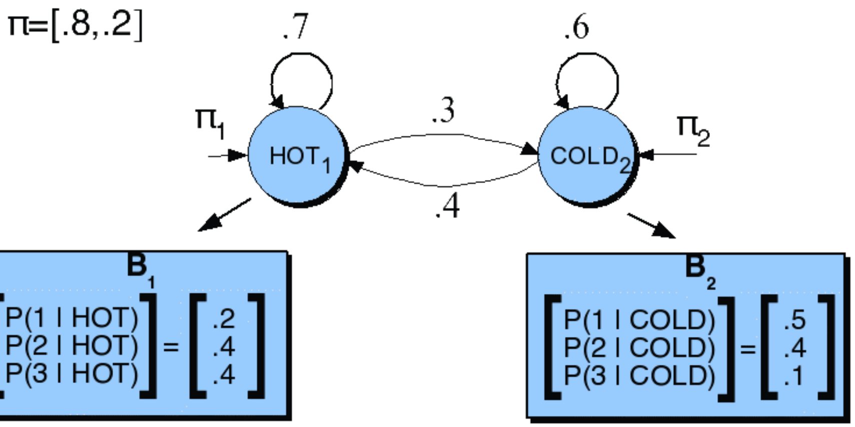


π=[.8,.2]
.7
.6
.3
.4
π_1
π_2
HOT_1
$COLD_2$
B_1
$\begin{bmatrix} P(1 \mid HOT) \\ P(2 \mid HOT) \\ P(3 \mid HOT) \end{bmatrix} = \begin{bmatrix} .2 \\ .4 \\ .4 \end{bmatrix}$
B_2
$\begin{bmatrix} P(1 \mid COLD) \\ P(2 \mid COLD) \\ P(3 \mid COLD) \end{bmatrix} = \begin{bmatrix} .5 \\ .4 \\ .1 \end{bmatrix}$

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



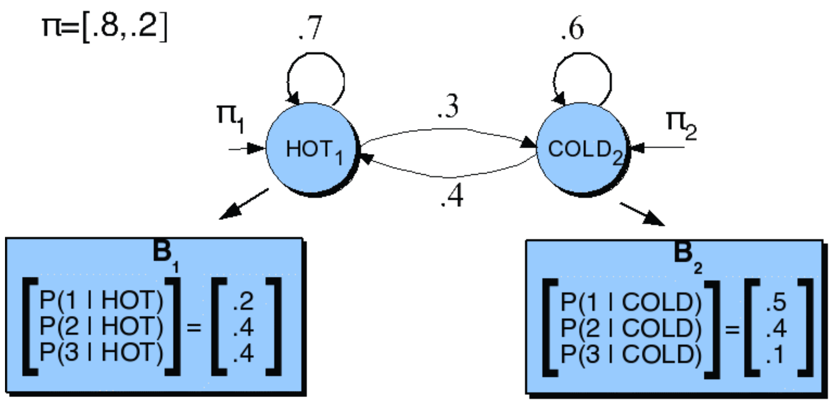
Argmax P(sequence | 1 3 1)

  - How do you pick the right one?

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


π=[.8,.2]
.7
.6
.3
.4
π1
π2
HOT1
COLD2
B1
P(1 I HOT) P(2 I HOT) P(3 I HOT) = .2 .4 .4
B2
P(1 I COLD) P(2 I COLD) P(3 I COLD) = .5 .4 .1

P(C H C)
= 0.2*0.5*0.4*0.4*0.3*0.5
= 0.0024

Viterbi Example 2: Ice Cream

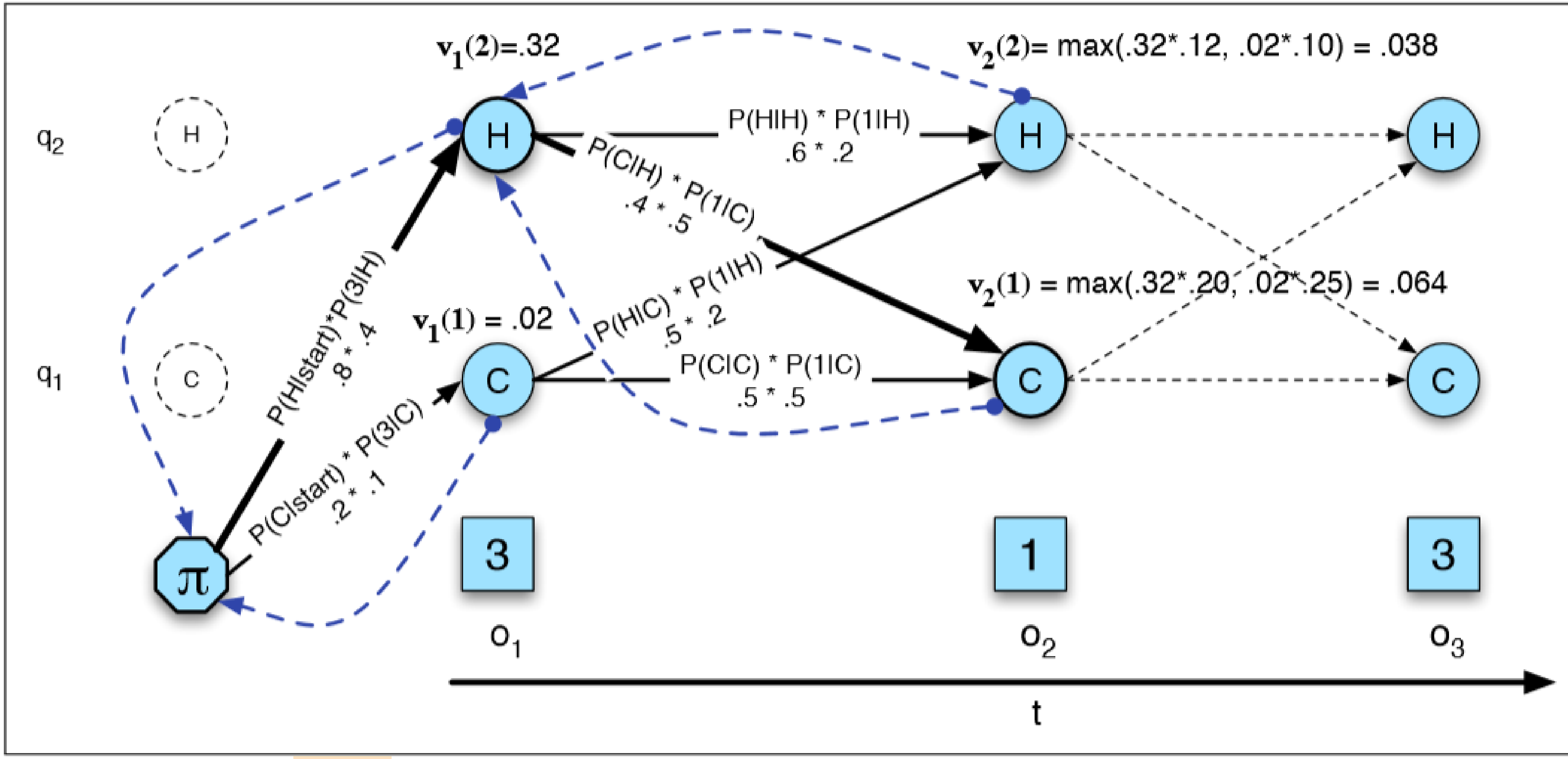
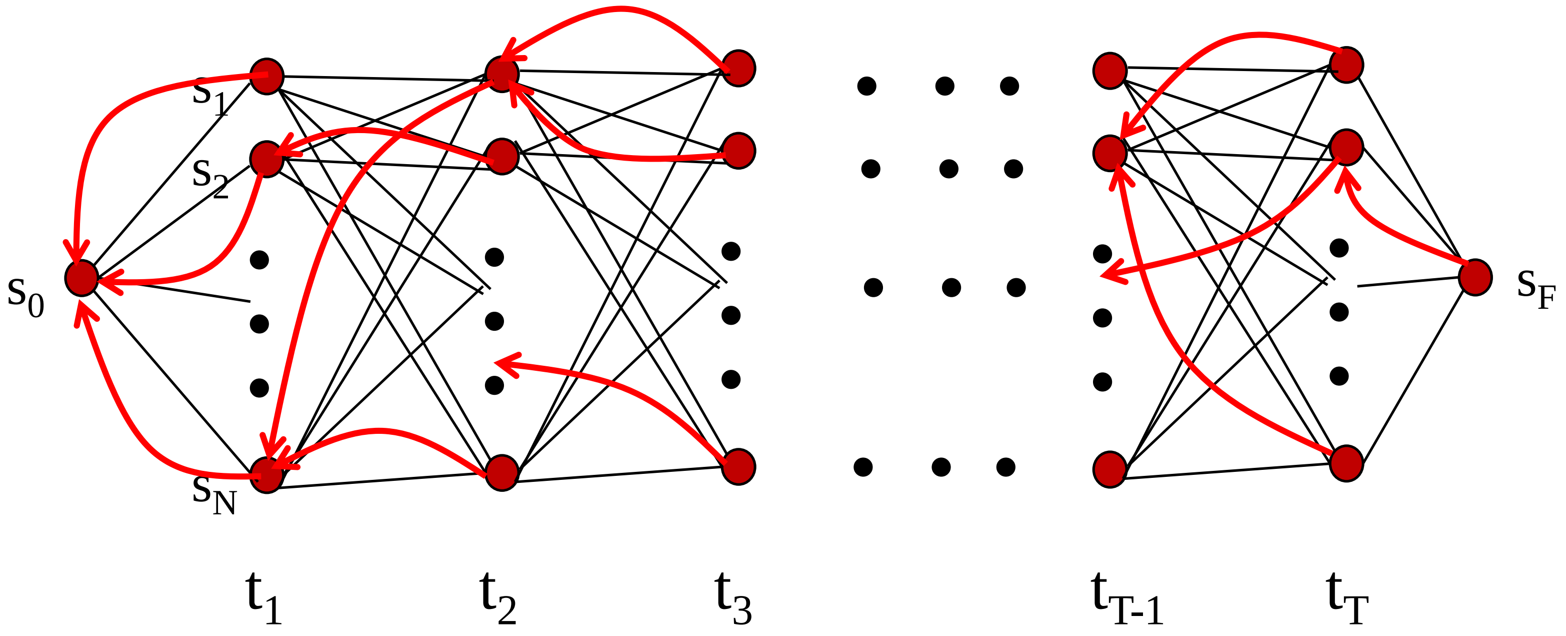
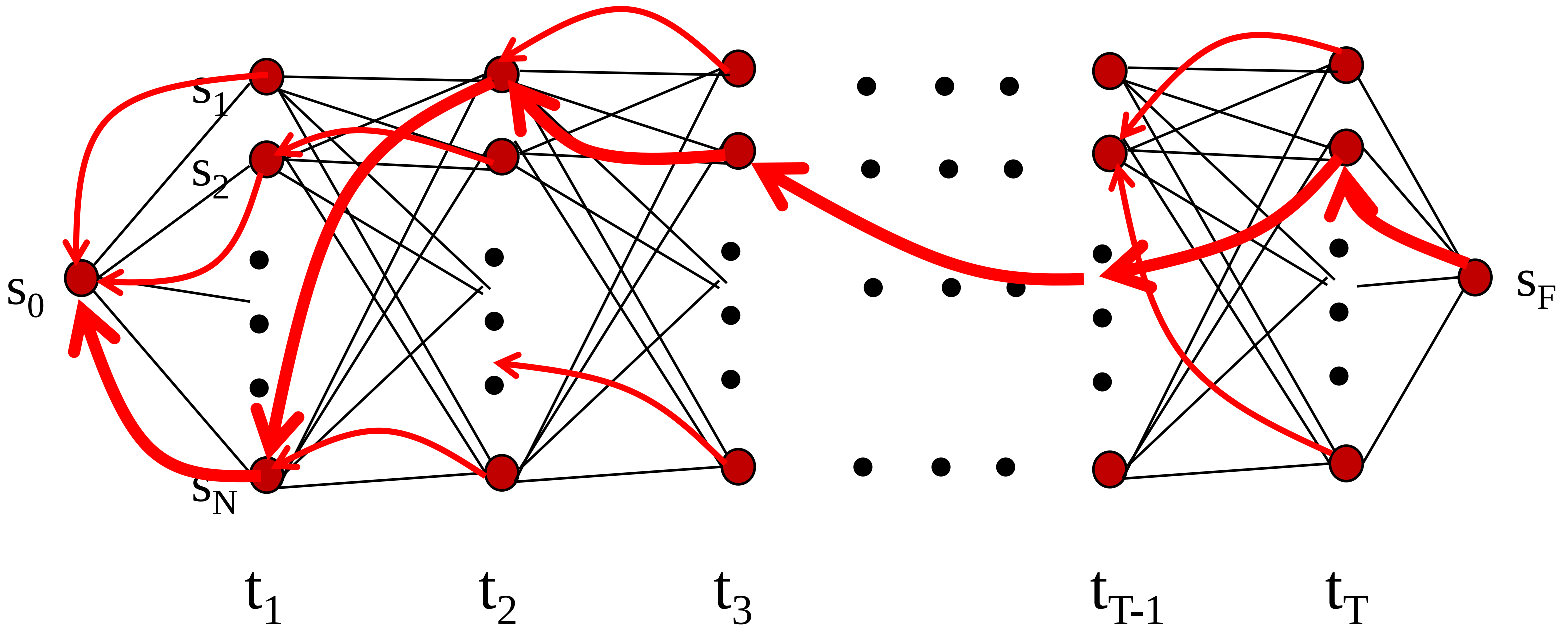


Figure A.10 The Viterbi backtrace. As we extend each path to a new state account for the next observatio
we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Viterbi Backpointers



Viterbi Backtrace



Most likely Sequence: $s_0\ s_N\ s_1\ s_2\ \ldots s_2\ s_F$

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
$backpointer[q_F,T]$
```

Viterbi algorithm

- Create a table V with N+2 rows and T columns:
  - N – the number of states/tags
  - T – the length of the sequence/sentence
- Initialize the first column
- For each tag t in the tagset compute:

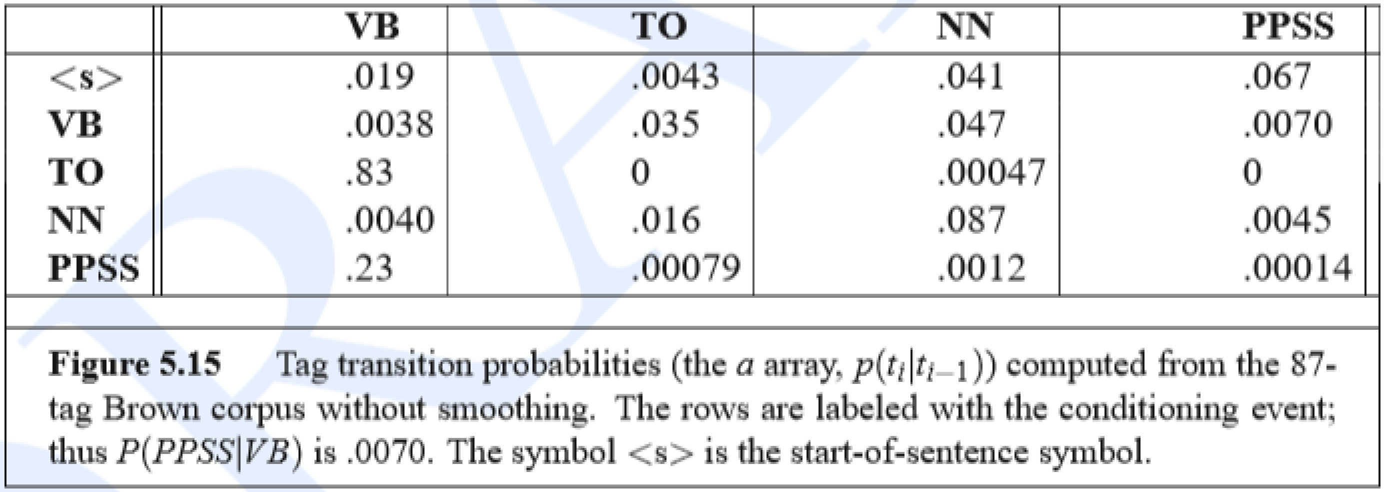
$$V[t,1] = P(t|start)P(w_1|t)$$

- For each column j = 2 to T in the table V:
  - For each tag t in the tagset compute:

$$V[t,j] = \max_{t'} V[t',j-1]P(t|t')P(w_j|t)$$

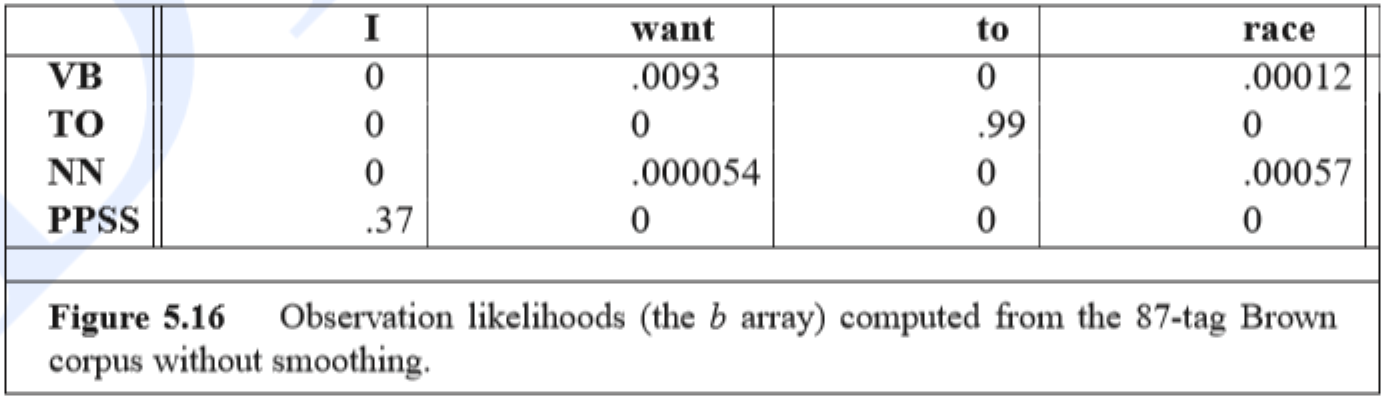
Viterbi Example 1: POS Tagging

Example: I want to race.



	VB	TO	NN	PPSS
<s>	.019	.0043	.041	.067
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

Figure 5.15 Tag transition probabilities (the *a* array, $p(t_i|t_{i-1})$) computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus $P(PPSS|VB)$ is .0070. The symbol <s> is the start-of-sentence symbol.



	I	want	to	race
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

Figure 5.16 Observation likelihoods (the *b* array) computed from the 87-tag Brown corpus without smoothing.

Viterbi Algorithm Example

Algorithm first creates N or four state columns.

- First column corresponds to the observation of the first word “**I**”,
- the second to the second word “**want”**,
- the third to the third word “**to”**, and
- the fourth to the fourth word “**race”**

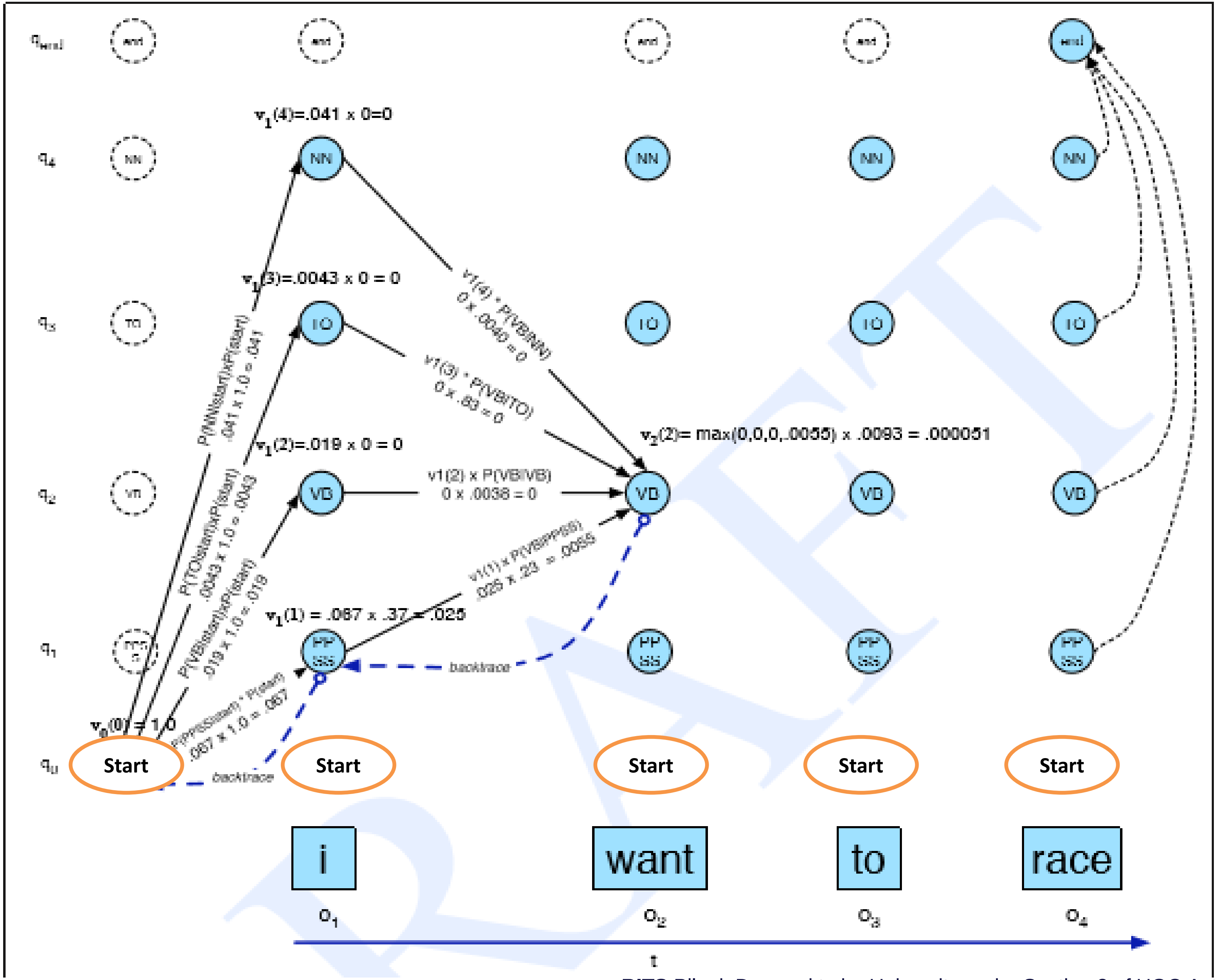
Begin by setting the Viterbi value in each cell to the product of the transition probability (into it from the state state) and the observation probability (of the first word)

Viterbi Algorithm Example

- For each state qj at time t, the value Viterbi [s,t] is computed by taking the maximum over the extensions of all the paths that lead to the current cell, following the following equation:

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

$v_{t-1}(i)$	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



q_{end}
q_4
q_3
q_2
q_1
q_0
end
NN
TO
VB
PPSS
$v_1(4)$=.041 x 0=0
$v_1(3)$=.0043 x 0 = 0
$v_1(2)$=.019 x 0 = 0
$v_1(1)$ = .067 x .37 = .025
$v_0(0)$ = 1.0
P(NN|start)xP(start) .041 x 1.0 = .041
P(TO|start)xP(start) .0043 x 1.0 = .0043
P(VB|start)xP(start) .019 x 1.0 = .019
P(PPSS|start) * P(start) .067 x 1.0 = .067
v1(4) * P(VB|NN) 0 x .0040 = 0
v1(3) * P(VB|TO) 0 x .83 = 0
v1(2) x P(VB|VB) 0 x .0038 = 0
v1(1) x P(VB|PPSS) .025 x .23 = .0055
$v_2(2)$= max(0,0,0,.0055) x .0093 = .000051
backtrace
backtrace
Start
Start
Start
Start
Start
i
want
to
race
o_1
o_2
o_3
o_4
t

Example 3 : POS Tagging

Transition matrix: $P(t_i|t_{i-1})$

	NOUN	Verb	Det	Prep	ADV	STOP
<S>	.3	.1	.3	.2	.1	0
Noun	.2	.4	.01	.3	.04	.05
Verb	.3	.05	.3	.2	.1	.05
Det	.9	.01	.01	.01	.07	0
Prep	.4	.05	.4	.1	.05	0
Adv	.1	.5	.1	.1	.1	.1

Emission matrix: $P(w_i|t_i)$

	a	cat	doctor	in	is	the	very
Noun	0	.5	.4	0	0.1	0	0
Verb	0	0	.1	0	.9	0	0
Det	.3	0	0	0	0	.7	0
Prep	0	0	0	1.0	0	0	0
Adv	0	0	0	.1	0	0	.9

	w1-=the	w2=doctor	w3=is	w4=in	STOP
Noun	0				
Verb	0				
Det	.21				
Prep	0				
Adv	0				

$$V[t,1] = P(t|start)P(w_1|t)$$

V(Noun,the) = P(Noun|<S>)P(the|Noun) = .3 X 0=0
V(Verb,the) = P(Verb|<S>)P(the|Verb) = .1 X 0=0
V(Det,the) = P(Det|<S>)P(the|Det) = .3 X .7=.21
V(Prep,the) = P(Prep|<S>)P(the|Prep) = .2 X .0=0
V(Adv,the) = P(Adv|<S>)P(the|Adv) = .2 X .0=0

Example (Contd..)

V(Noun, doctor) = max$_{t'}$ V(t',the)XP(Noun|t')X P(doctor|Noun)

= max {0, 0, .21 (.9 X.4),0, 0} = .0756

V(Verb, doctor) = max$_{t'}$ V(t',the)XP(Verb|t')X P(doctor|Verb)

= max {0, 0, .21(.01X.1), 0, 0} = .00021

	w1-=the	w2=doctor	w3=is	w4=in	STOP
Noun	0	.0756			
Verb	0	.00021			
Det	.21	0			
Prep	0	0			
Adv	0	0			

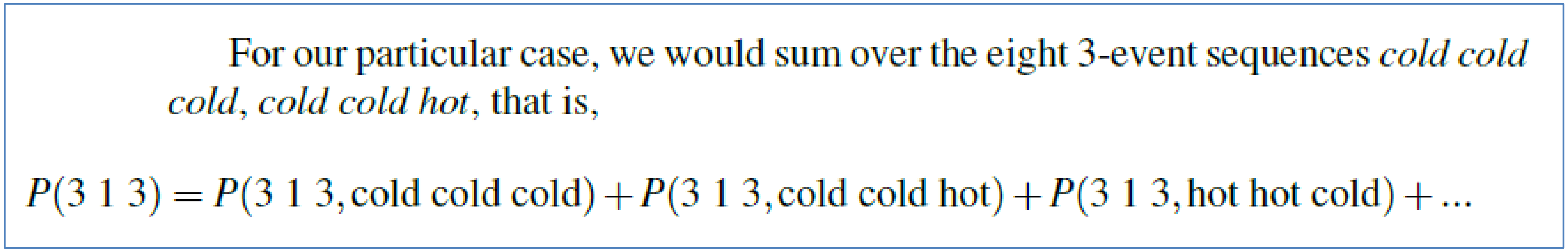
Backtracking the Viterbi Matrix

	w1-=the	w2=doctor	w3=is	w4=in	STOP
Noun	0	.0756	.001512	0	
Verb	0	.00021	.027216	0	
Det	.21	0	0	0	.005443
Prep	0	0	0	.005443	
Adv	0	0	0	.000272	

Det Noun Verb Prep

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**

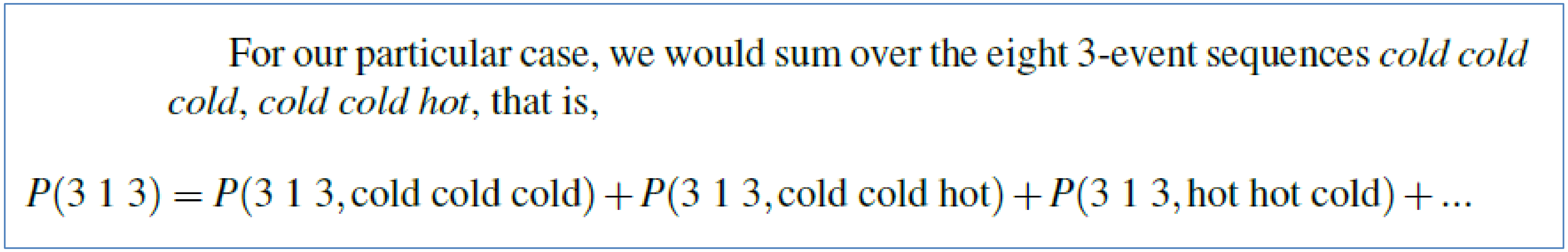


For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

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

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**



For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

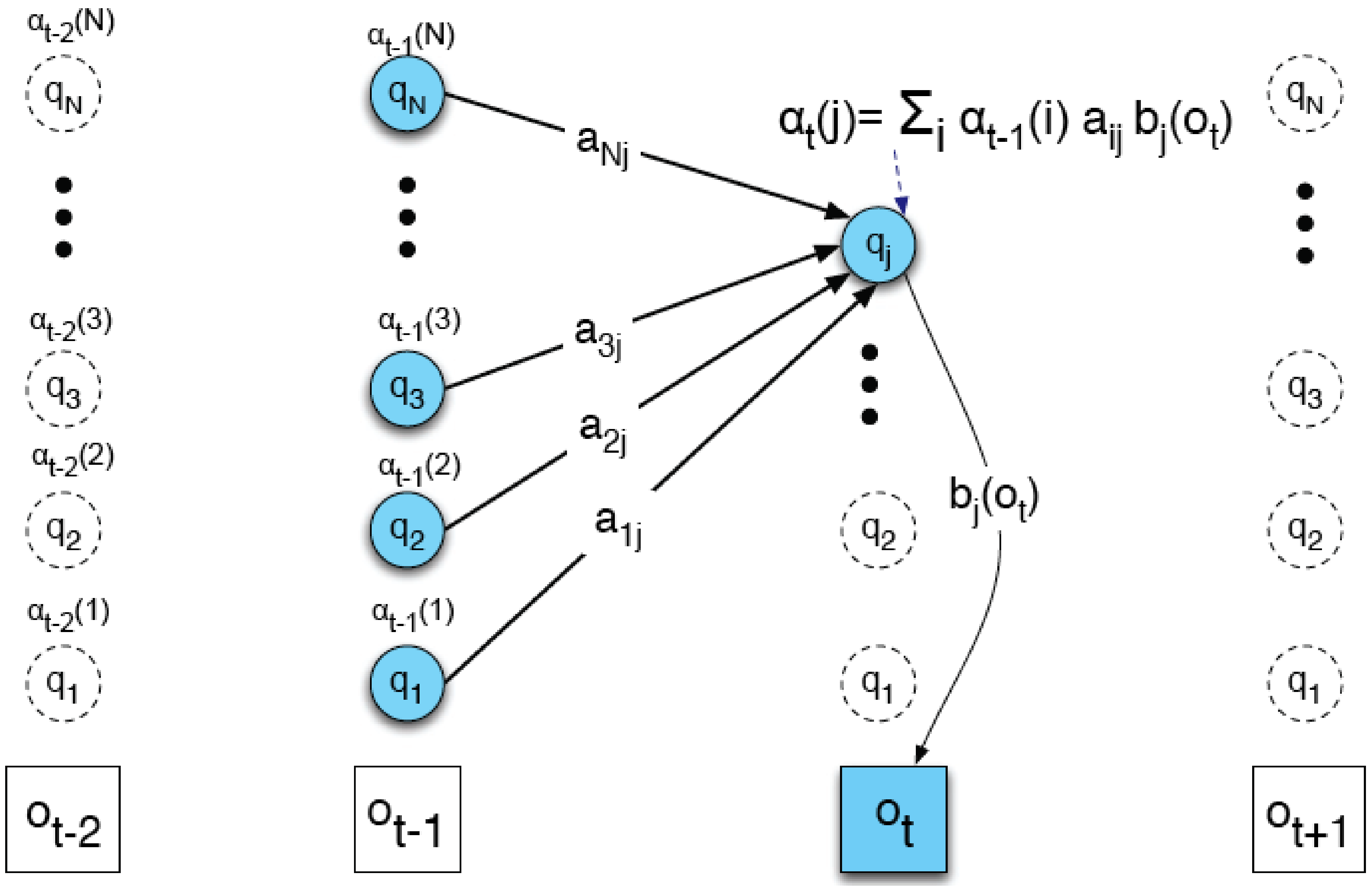
$$P(3\ 1\ 3) = P(3\ 1\ 3, \text{cold cold cold}) + P(3\ 1\ 3, \text{cold cold hot}) + P(3\ 1\ 3, \text{hot hot cold}) + \ldots$$

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
                  $forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$
    $forward[q_F,T] \leftarrow \sum_{s=1}^{N} forward[s,T] * a_{s,q_F}$       ; termination step
  return $forward[q_F,T]$
```

Visualizing Forward



Forward Algorithm: Ice Cream

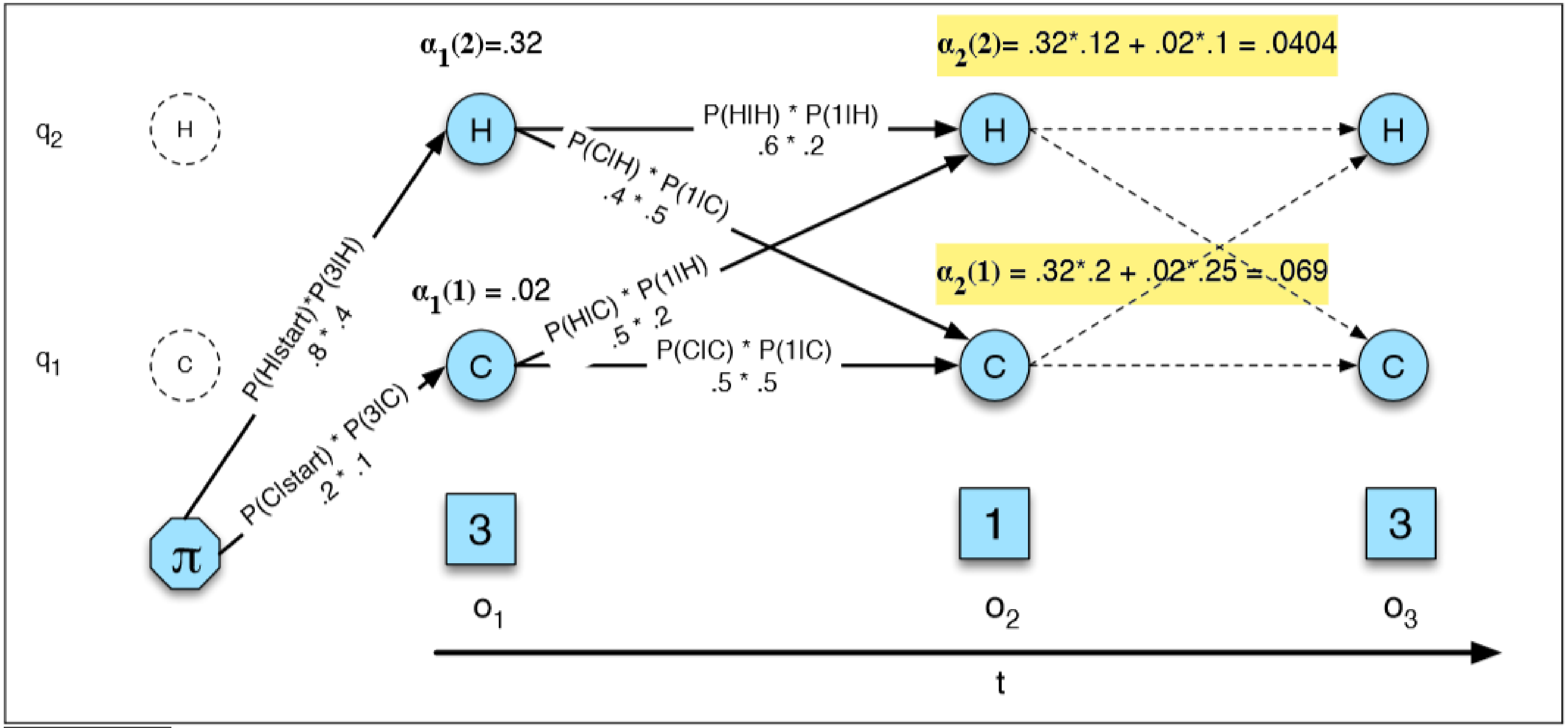


Figure A.5 The forward trellis for computing the total observation likelihood for the ice-cream events *3 1 3*. Hidden states are in circles, observations in squares. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.12: $\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.11: $\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$.

Bidirectionality

- One problem with the HMM models as presented is that they are exclusively run left-to-right.
- Viterbi algorithm still allows present decisions to be influenced indirectly by future decisions

Bidirectionality

- Any sequence model can be turned into a bidirectional model by using multiple passes.
- For example, the first pass would use only part-of-speech features from already-disambiguated words on the left. In the second pass, tags for all words, including those on the right, can be used.
- Alternately, the tagger can be run twice, once left-to-right and once right-to-left.
- In Viterbi decoding, the classifier chooses the higher scoring of the two sequences (left-to-right or right-to-left).
- Modern taggers are generally run bidirectionally.

Issues with HMM

- Unknown Words
  - We do not have the probabilities
    - Use smoothing
- Limited Context
  - Use trigrams/ 4 grams etc.
  - Sparsity

Maximum Entropy Markov Model

- Identify heterogeneous set of features
  - tag given, the previous tag or the word given, the tag you can use, any other features which may contribute to the choice of part of speech tags.
- Turn logistic regression into a discriminative sequence model simply by running it on successive words, using the class assigned to the prior word as a feature in the classification of the next word.
- When we apply logistic regression in this way, it’s called maximum entropy Markov model or MEMM

Maximum Entropy Markov Model

- Let the sequence of words be W = w_1^n and the sequence of tags T = t_1^n
- In an HMM to compute the best tag sequence that maximizes P(T|W) we rely on Bayes' rule and the likelihood P(W|T):

$$\begin{aligned}
\hat{T} &= \underset{T}{\operatorname{argmax}} P(T|W) \\
&= \underset{T}{\operatorname{argmax}} P(W|T)P(T) \\
&= \underset{T}{\operatorname{argmax}} \prod_i P(word_i|tag_i) \prod_i P(tag_i|tag_{i-1})
\end{aligned}$$

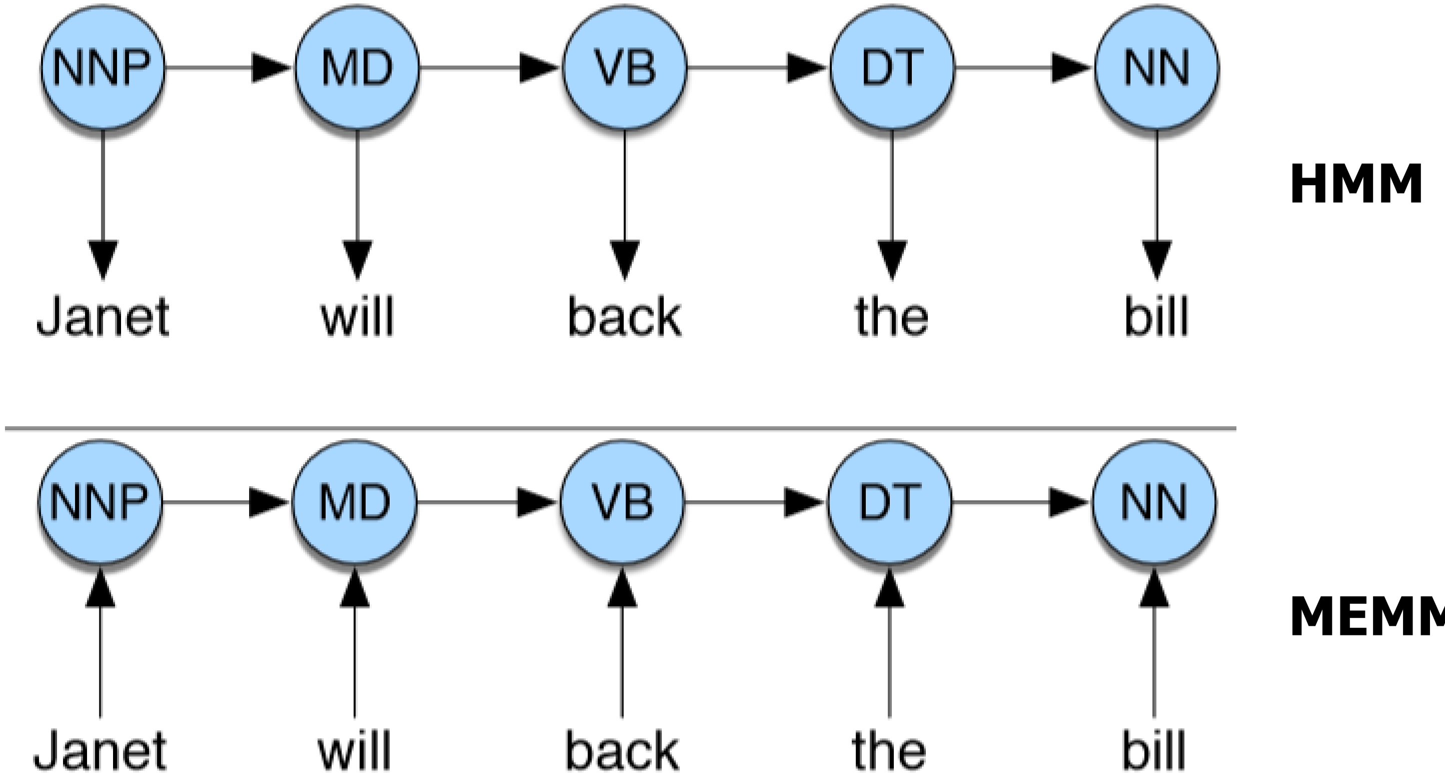
Maximum Entropy Markov Model

- In an MEMM, by contrast, we compute the posterior P(T|W) directly, training it to discriminate among the possible tag sequences:

$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_{T} P(T|W) \\ &= \operatorname*{argmax}_{T} \prod_{i} P(t_i|w_i, t_{i-1})\end{aligned}$$

Maximum Entropy Markov Model

- Consider tagging just one word. A multinomial logistic regression classifier could compute the single probability P(ti|wi,ti−1) in a different way than an HMM
- HMMs compute likelihood (observation word conditioned on tags) but MEMMs compute posterior (tags conditioned on observation words).



Maximum Entropy Markov Model

- Reason to use a discriminative sequence model is that it's easier to incorporate a lot of features

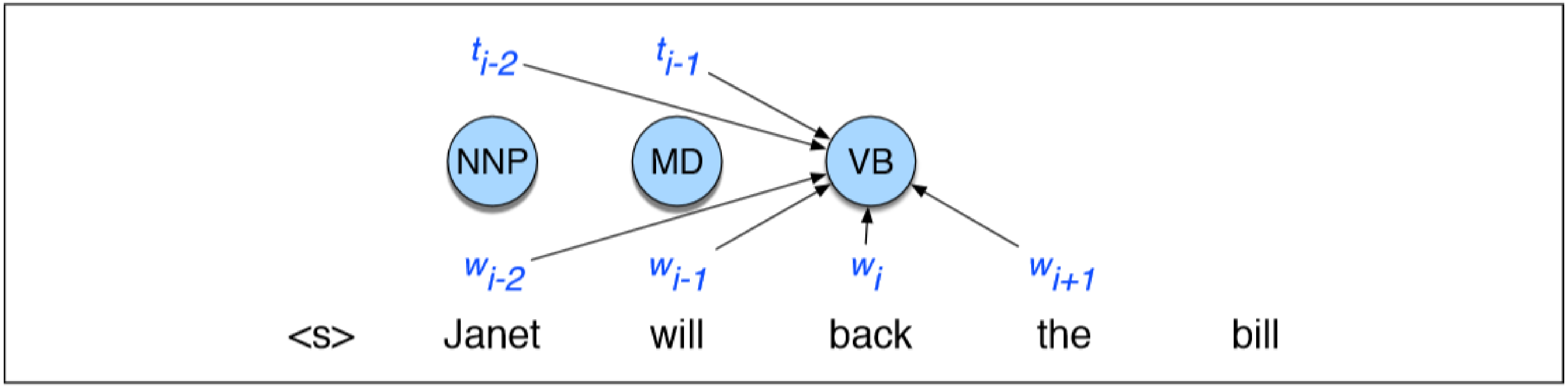


Figure 8.13 An MEMM for part-of-speech tagging showing the ability to condition on more features.

MEMM

- Janet/NNP will/MD back/VB the/DT bill/NN, when wi is the word back, would generate the following features

$$
\begin{aligned}
&t_i = \text{VB and } w_{i-2} = \text{Janet} \\
&t_i = \text{VB and } w_{i-1} = \text{will} \\
&t_i = \text{VB and } w_i = \text{back} \\
&t_i = \text{VB and } w_{i+1} = \text{the} \\
&t_i = \text{VB and } w_{i+2} = \text{bill} \\
&t_i = \text{VB and } t_{i-1} = \text{MD} \\
&t_i = \text{VB and } t_{i-1} = \text{MD and } t_{i-2} = \text{NNP} \\
&t_i = \text{VB and } w_i = \text{back and } w_{i+1} = \text{the}
\end{aligned}
$$

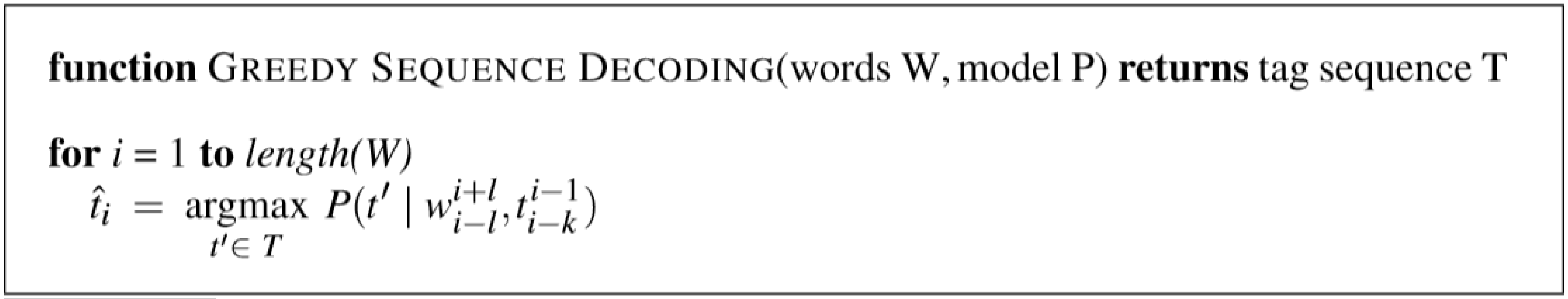
Decoding and Training MEMMs

- The most likely sequence of tags is then computed by combining these features of the input word w_i, its neighbors within l words w^{i+l}_{i-l}, and the previous k tags t^{i-1}_{i-k} as follows (using θ to refer to feature weights instead of w to avoid the confusion with w meaning words):

$$
\begin{aligned}
\hat{T} &= \operatorname*{argmax}_T P(T|W) \\
&= \operatorname*{argmax}_T \prod_i P(t_i | w^{i+l}_{i-l}, t^{i-1}_{i-k}) \\
&= \operatorname*{argmax}_T \prod_i \frac{\exp\left(\sum_j \theta_j f_j(t_i, w^{i+l}_{i-l}, t^{i-1}_{i-k})\right)}{\sum_{t' \in \text{tagset}} \exp\left(\sum_j \theta_j f_j(t', w^{i+l}_{i-l}, t^{i-1}_{i-k})\right)}
\end{aligned}
$$

How to decode to find this optimal tag sequence ˆ T?

- Simplest way to turn logistic regression into a sequence model is to build a local classifier that classifies each word left to right, making a hard classification on the first word in the sentence, then a hard decision on the second word, and so on.
- This is called a greedy decoding algorithm



```
function GREEDY SEQUENCE DECODING(words W, model P) returns tag sequence T

for $i = 1$ to $length(W)$
    $\hat{t}_i = \underset{t' \in T}{\operatorname{argmax}} P(t' \mid w_{i-l}^{i+l}, t_{i-k}^{i-1})$
```

Figure 8.14 In greedy decoding we simply run the classifier on each token, left to right, each time making a hard decision about which is the best tag.

Issue with greedy algorithm

- The problem with the greedy algorithm is that by making a hard decision on each word before moving on to the next word, the classifier can’t use evidence from future decisions.
- Although the greedy algorithm is very fast, and occasionally has sufficient accuracy to be useful, in general the hard decision causes too great a drop in performance, and we don’t use it.
- MEMM with the Viterbi algorithm just as with the HMM, Viterbi finding the sequence of part-of-speech tags that is optimal for the whole sentence

MEMM with Viterbi algorithm

- Finding the sequence of part-of-speech tags that is optimal for the whole sentence. Viterbi value of time t for state j

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

- In HMM

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, P(s_j|s_i)\, P(o_t|s_j) \quad 1 \le j \le N, 1 < t \le T$$

- In MEMM

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, P(s_j|s_i, o_t) \quad 1 \le j \le N, 1 < t \le T$$

Learning MEMM

- Learning in MEMMs relies on the same supervised learning algorithms we presented for logistic regression.
- Given a sequence of observations, feature functions, and corresponding hidden states, we use gradient descent to train the weights to maximize the log-likelihood of the training corpus.

References

- https://www.nltk.org/
- https://likegeeks.com/nlp-tutorial-using-python-nltk/
- https://www.guru99.com/pos-tagging-chunking-nltk.html
- https://medium.com/greyatom/learning-pos-tagging-chunking-in-nlp-85f7f811a8cb
- https://nlp.stanford.edu/software/tagger.shtml
- https://www.forbes.com/sites/mariyayao/2020/01/22/what-are--important-ai--machine-learning-trends-for-2020/#601ce9623239
- https://medium.com/fintechexplained/nlp-text-processing-in-data-science-projects-f083009d78fc
- https://www.nltk.org/book/ch02.html
- https://towardsdatascience.com/pos-tagging-using-crfs-ea430c5fb78b

References

- https://github.com/nadavo/MEMM-POS-Tagger
- https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.352.9257&rep=rep1&type=pdf
- https://www.alibabacloud.com/blog/hmm-memm-and-crf-a-comparative-analysis-of-statistical-modeling-methods_592049
- https://towardsdatascience.com/pos-tagging-using-rnn-7f08a522f849

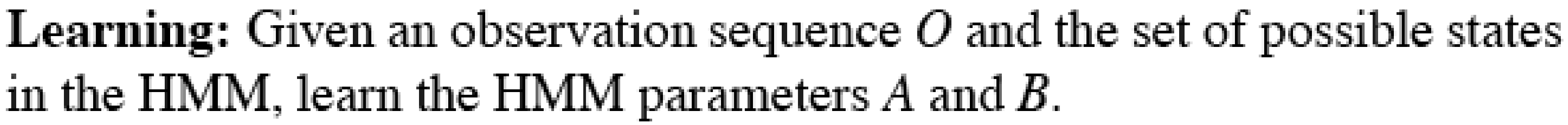
POS tagging in Indian Languages

- https://www.nltk.org/book/ch05.html

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...

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
- The algorithm will let us learn the transition probabilities A= $\{a_{ij}\}$ and the emission probabilities B=$\{b_i(o_t)\}$ of the HMM

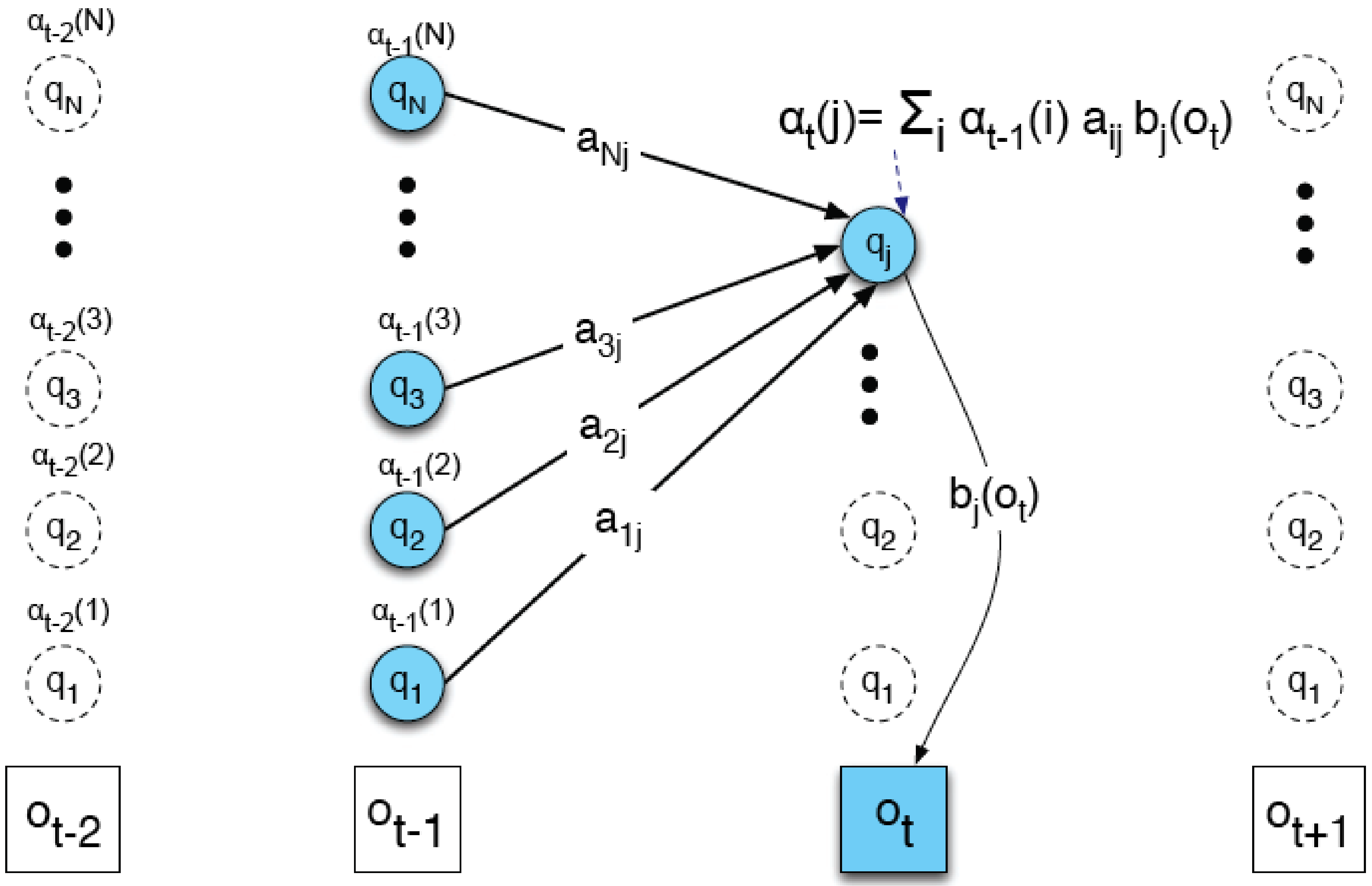
Sketch of Baum-Welch (EM) Algorithm for Training HMMs

Assume an HMM with N states.
Randomly set its parameters $\lambda=(A,B)$
(making sure they represent legal distributions)
Until converge (i.e. λ no longer changes) do:
E Step: Use the forward/backward procedure to
determine the probability of various possible
state sequences for generating the training data
M Step: Use these probability estimates to
re-estimate values for all of the parameters λ

EM Properties

- Each iteration changes the parameters in a way that is guaranteed to increase the likelihood of the data: $P(O|\lambda)$.
- Anytime algorithm: Can stop at any time prior to convergence to get approximate solution.
- Converges to a local maximum.

Visualizing Forward



Backward Probabilities

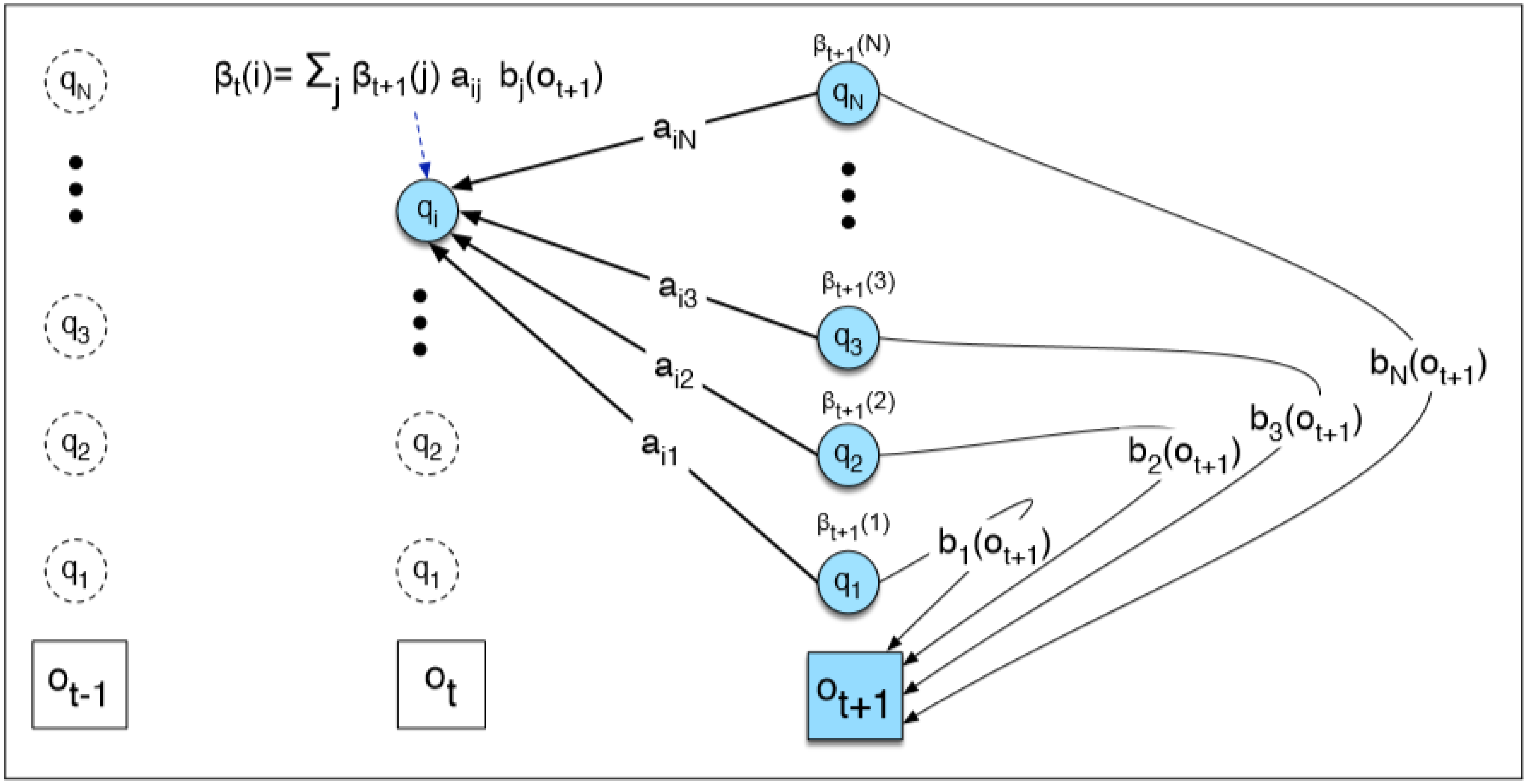


Figure A.11 The computation of $\beta_t(i)$ by summing all the successive values $\beta_{t+1}(j)$ weighted by their transition probabilities a_{ij} and their observation probabilities $b_j(o_{t+1})$. Start and end states not shown.

Intuition for re-estimation of a_{ij}

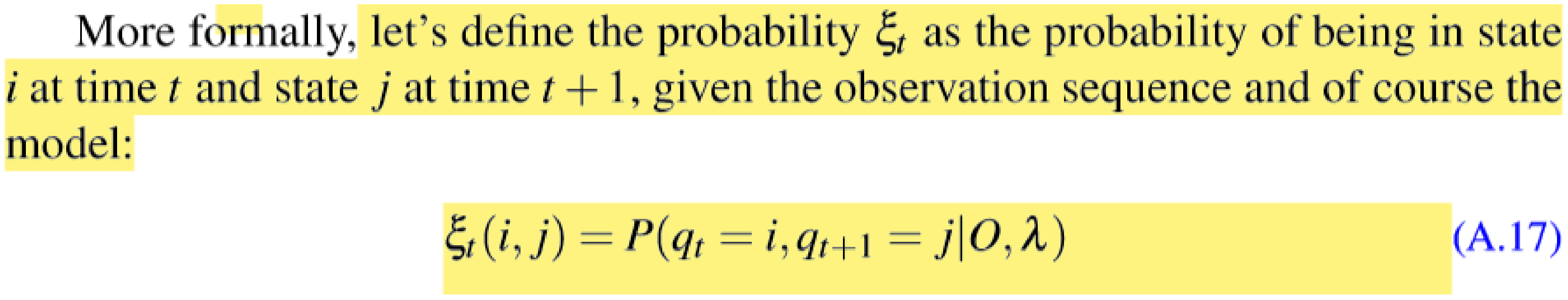
- We will estimate $\hat{a}_{ij}$ via this intuition:

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

- intuition:
  - If we knew this probability for *each* time *t*, we could sum over all *t* to get expected value for i➔j.

Intuition for re-estimation of a_{ij}

- Assume we had some estimate of the probability that a given transition $i \rightarrow j$ was taken at a particular point in time t in the observation sequence.
- If we knew this probability for each particular time t, we could sum over all times t to estimate the total count for the transition $i \rightarrow j$.

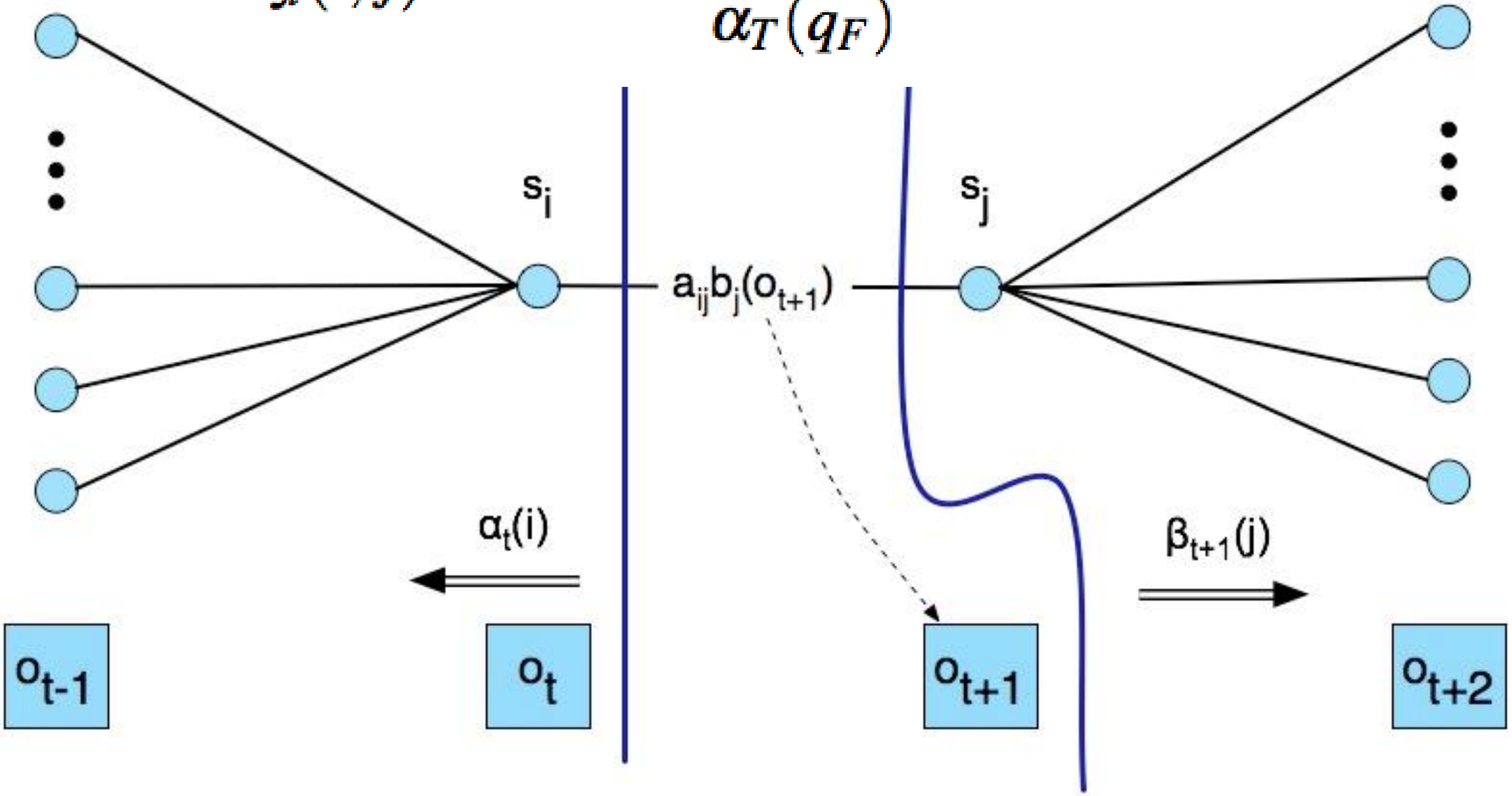


More formally, let's define the probability ξ_t as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence and of course the model:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \quad \text{(A.17)}$$

Intuition for re-estimation of a_{ij}

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



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
          $\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$
          $\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$
        M-step
          $\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N}\xi_t(i,k)}$    $\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$
  return $A$, $B$
```