Hidden Markov Models

Jim Glass / MIT 6.806-6.864 / Spring 2021

1

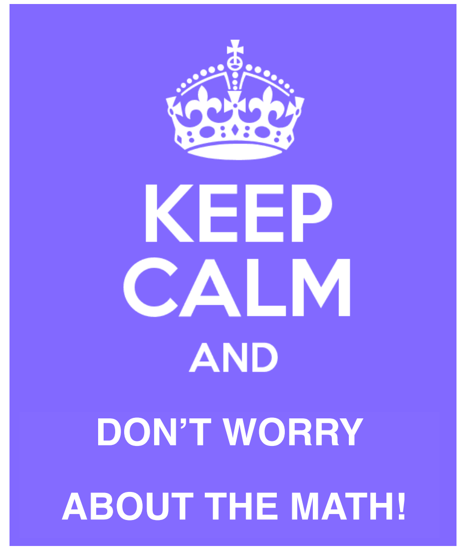
Sequential Labeling

- Human language is fundamentally sequential in nature
- Many NLP tasks involve converting one sequence into another:
  - Part-of-speech tagging
  - Named entity recognition
  - Machine translation
  - Speech recognition
- A range of ML techniques apply to sequence-to-sequence tasks:
  - Hidden Markov models
  - Conditional random fields
  - Recurrent neural networks

2

Today's HMM Storyline

- Part-of-speech tagging
  - Dynamic programming
  - Viterbi search
- Hidden Markov models
  1. Scoring: Forward-backward algorithm
  2. Matching: Viterbi search
  3. Training: Baum-Welch parameter estimation



3

Part-of-Speech Tagging

- POS tagging assigns each word in a sentence a grammatical tag
  - The tag depends on the word and its context (e.g., sentence)
  - POS inventory is language and corpus dependent
  - Typically used for features, or a precursor for other tasks (e.g., parsing)
  - POS tagging also known as word category disambiguation
- An inherent challenge for POS tagging is word category ambiguity

"People can fish"

People: Noun? Verb?
can: Noun? Verb?
fish: Noun? Verb?

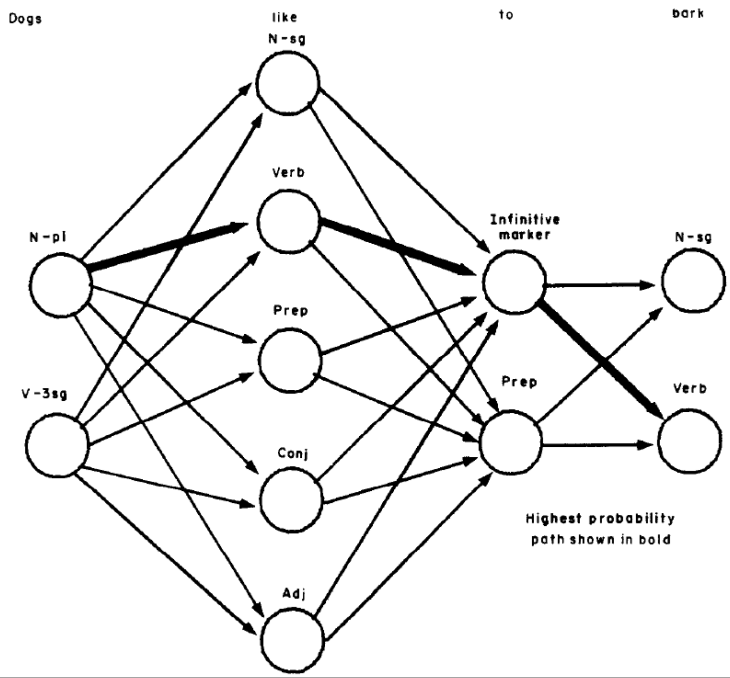
4

Computer Speech and Language (1992) **6**, 225–242

Robust part-of-speech tagging using a hidden Markov model

Julian Kupiec

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo U.S.A.



5

A Probabilistic Formulation for POS Tagging

- Define words $W = \{w_1, \cdots, w_n\}$ and corresponding tags $T = \{t_1, \cdots, t_n\}$
- Given a word sequence, we infer the "hidden" tag sequence T^*

$$T^* = \arg\max_T P(W,T) \quad \text{where } P(W,T) = P\{w_1, \cdots, w_n, t_1, \cdots, t_n\}$$

- Using the chain rule, we can rewrite $P(W,T)$ as

$$P(W,T) = \prod_{i=1}^{n} P(w_i, t_i | w_1, \dots, w_{i-1}, t_1, \dots, t_{i-1})$$

- By making conditional independence assumptions that t_i depends only on t_{i-1}, and w_i depends only on t_i we can rewrite $P(W,T)$ as

$$P(W,T) = \prod_{i=1}^{n} P(w_i\,|t_i) P(t_i|\,t_{i-1})$$

Observation probabilities

Transition probabilities

6

Parameter Estimation and Search

- Observation and transition probabilities can be estimated from annotated data or learned via EM algorithm from unannotated data

Observations $\propto \log P(w_i | t_i)$

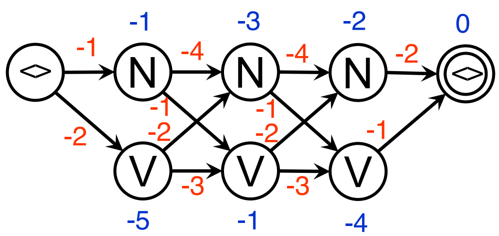
	people	can	fish	<>
<>	$-\infty$	$-\infty$	$-\infty$	0
N	-1	-3	-2	$-\infty$
V	-5	-1	-4	$-\infty$

Transitions $\propto \log P(t_i | t_{i-1})$

	N	V	<>
<>	-1	-2	$-\infty$
N	-4	-1	-2
V	-2	-3	-1

- Search space can be represented as directed acyclic graph (DAG)

<> people can fish <>



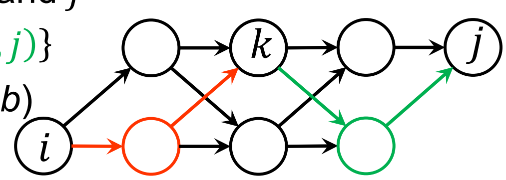
Weighted finite-state transducers are effective representations for DAGs

7

Dynamic Programming (DP)

- DP algorithms such as Viterbi search leverage optimal substructure
  - Let $\phi(i,j)$ be the best path between nodes i and j
  - If k is a node in $\phi(i,j)$: $\phi(i,j) = \{\phi(i,k), \phi(k,j)\}$
  - Let $\varphi(i,j)$ be the cost of $\phi(i,j)$ (e.g., $-$*logprob*)

$$\varphi(i,j) = \min_k(\varphi(i,k) + \varphi(k,j))$$

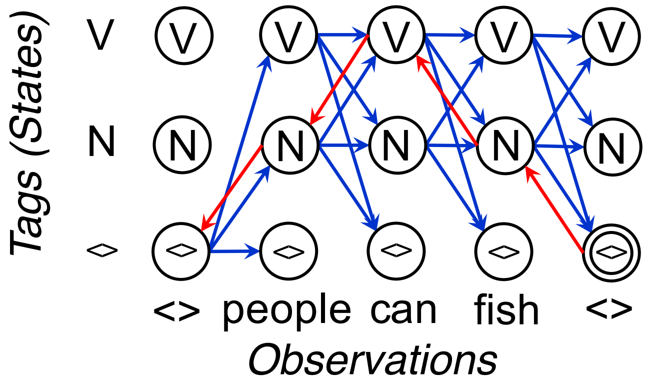


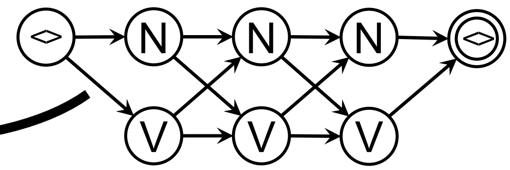
- Solutions to sub-problems need only be computed once
  - Sub-optimal partial paths discarded while staying *admissible*
- Can be implemented efficiently:
  - Node k retains only best path cost of all $\varphi(i,k)$
  - Previous best node index needed to recover best path
- Best-first and A* graph search also leverage optimal substructure

8

Viterbi Search

- Viterbi search arranges search through a fixed-dimension *trellis*
  - Search advances time-synchronously
  - All partial paths ending at a common node converge *at the same moment*
  - Per DP, each node retains best score and back pointer to best partial path
  - Best path is recovered at end via *backtrace*



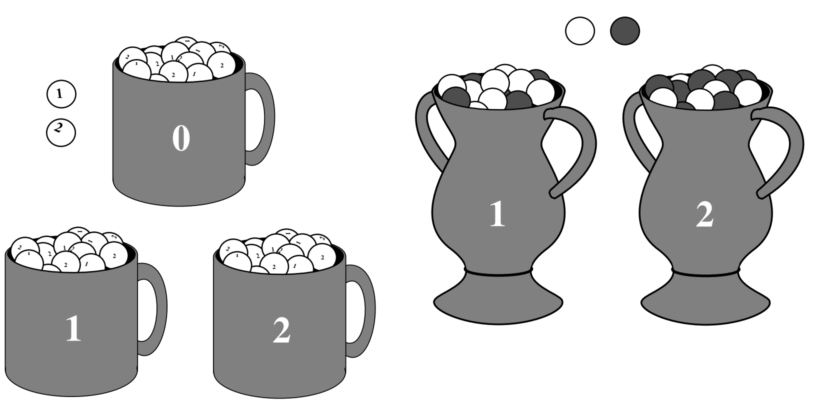


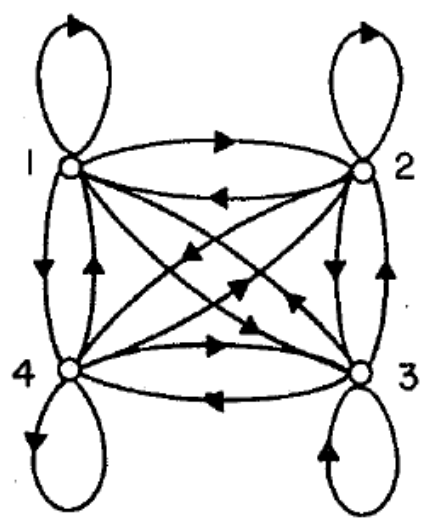
$$T^* = \{<>, N, V, N, <>\}$$

9

An Introduction to Hidden Markov Models

L. R. Rabiner
B. H. Juang

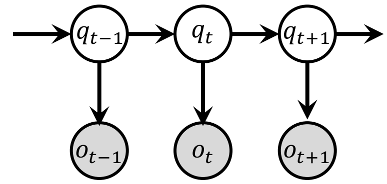




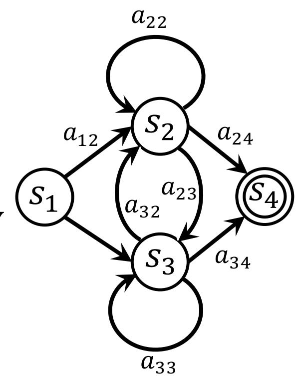
IEEE ASSP MAGAZINE JANUARY 1986

10

Hidden Markov Model Notation



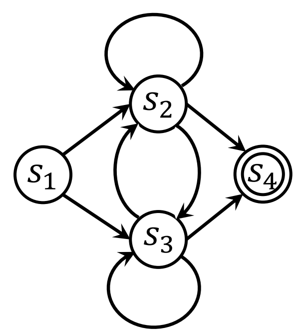
- Underlying HMM states $\boldsymbol{s} = \{s_1, \dots, s_N\}$
  - State at time $t, q_t \in \boldsymbol{s}$
- Set of emitted observations $\boldsymbol{w} = \{w_1, \dots, w_V\}$
  - Observation at time $t, o_t \in \boldsymbol{w}$
- $\boldsymbol{A} = \{a_{ij}\}$: state transition probabilities
  - $a_{ij} = P\big(q_{t+1} = s_j \big| q_t = s_i\big) \quad 1 \le i, j \le N$
- $\boldsymbol{B} = \{b_j(k)\}$: observation probabilities
  - $b_j(k) = P\big(o_t = w_k \big| q_t = s_j\big) \equiv b_j(o_t) \quad 1 \le j \le N, 1 \le k \le V$
- $\pi = \{\pi_i\}$: initial state distribution
  - $\pi_i = P(q_1 = s_i) \quad 1 \le i \le N$
- A HMM is typically written as: $\lambda = \{\boldsymbol{A}, \boldsymbol{B}, \pi\}$



11

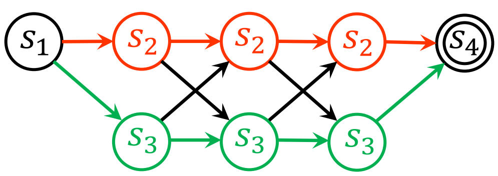
POS Example with HMM Notation

- 4 states (s_1=<>, s_2=*N*, s_3=*V*, s_4=<>)
- *V* word observations ($w_{<>}$, w_{can}, w_{fish}, w_{people}, …)
- 5 input observations (<>, people, can, fish, <>)
  o_1=$w_{<>}$ o_2=w_{people} o_3=w_{can} o_4=w_{fish} o_5=$w_{<>}$
- π_1=1



o_1 o_2 o_3 o_4 o_5

<> people can fish <>



q_1=s_1 q_2=s_2 q_3=s_2 q_4=s_2 q_5=s_4

q_1=s_1 q_2=s_3 q_3=s_3 q_4=s_3 q_5=s_4

12

Three Fundamental HMM Problems

1. Score: Given observation sequence $\boldsymbol{O} = \{o_1, \dots, o_T\}$, and HMM $\lambda = \{\boldsymbol{A}, \boldsymbol{B}, \pi\}$, how do we compute the probability $P(\boldsymbol{O}|\lambda)$?
   – Forward-Backward algorithm
2. Match: Given $\boldsymbol{O} = \{o_1, \dots, o_T\}$, how do we choose the optimum underlying state sequence $\boldsymbol{Q} = \{q_1, \dots, q_T\}$?
   – Viterbi algorithm
3. Train: How to learn ML parameter estimates for $\lambda = \{\boldsymbol{A}, \boldsymbol{B}, \pi\}$?
   – Baum-Welch Estimation

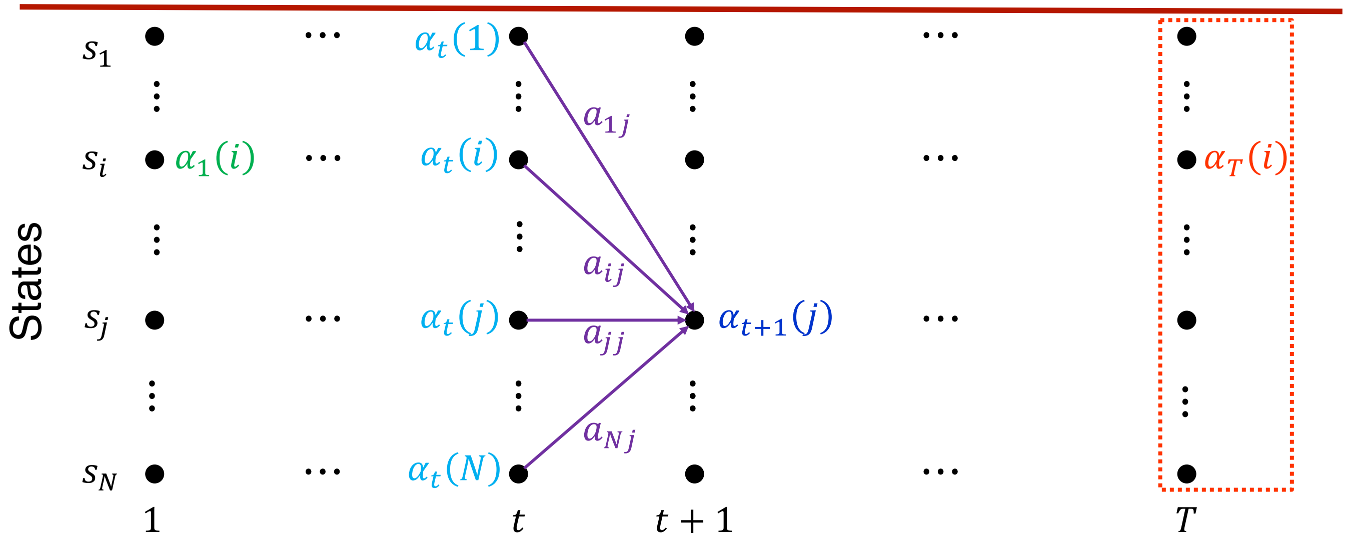
13

The Forward Algorithm

- Goal: compute $P(\boldsymbol{O}|\lambda) = \sum_{\forall \boldsymbol{Q}} P(\boldsymbol{O}, \boldsymbol{Q}|\lambda)$ (brute force: $O(TN^T)$)
- Recursion: define *forward* variable: $\alpha_t(i) = P(o_1, \dots, o_t, q_t = s_i|\lambda)$
  (i.e., probability of seeing observations up to time t, and state s_i at time t)
  1. For $t = 1$, $\alpha_1(i) = \pi_i \, b_i(o_1) \quad 1 \le i \le N$
  2. For $t > 1$, consider all ways of getting to current state at t
  $$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] b_j(o_t) \quad 1 < t \le T \quad 1 \le j \le N$$
  3. Finally: $P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$
- Computation is on the order of $O(TN^2)$

14

Forward Algorithm: $\alpha_t(i) = P(o_1, \dots, o_t, q_t = s_i|\lambda)$



1) $\alpha_1(i) = \pi_i \, b_i(o_1)$ 2) $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1})$ 3) $P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

15

The Backward Algorithm

- Define *backward* variable: $\beta_t(i) = P(o_{t+1}, \dots, o_T|q_t = s_i, \lambda)$
  (i.e., state s_i at time t, probability of seeing remaining observations)
  1. For $t = T$, $\beta_T(i) = 1 \quad 1 \le i \le N$
  2. For $t < T$, consider all ways of getting to current state at t

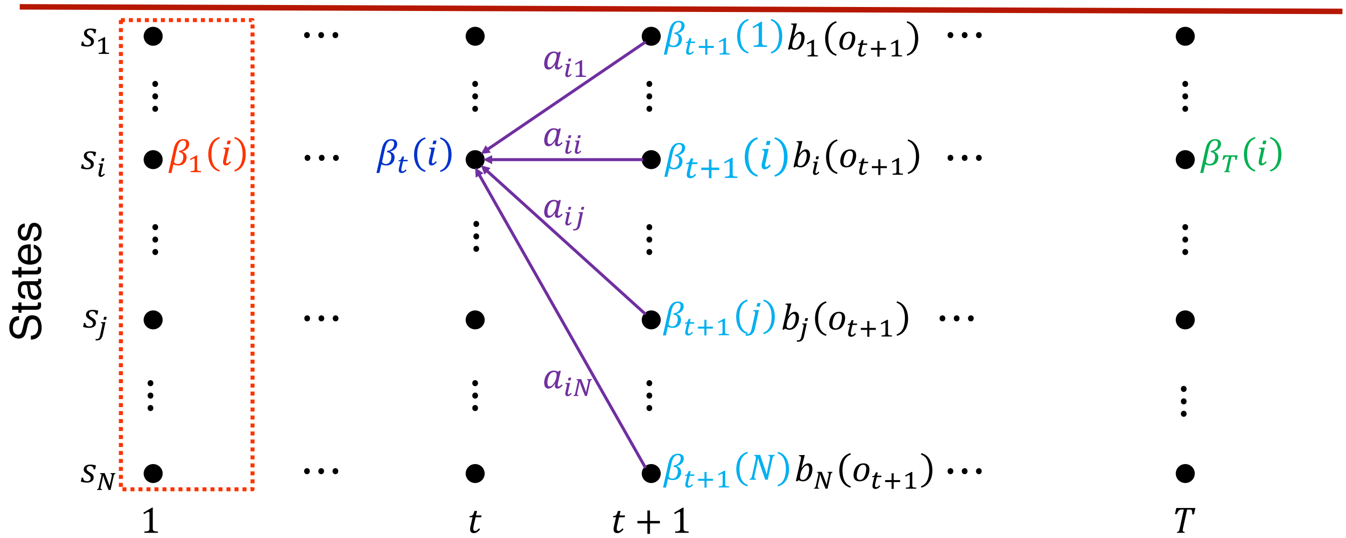
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \quad 1 \le t < T \quad 1 \le i \le N$$

  3. Finally: $P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)$
- Either forward or backward algorithm can be used to compute $P(\boldsymbol{O}|\lambda)$, but both are needed to learn model parameters

$$\alpha_t(i)\beta_t(i) = P(\boldsymbol{O}, q_t = s_i|\lambda)$$

16

Backward Algorithm: $\beta_t(i) = P(o_{t+1}, \ldots, o_T | q_t = s_i, \lambda)$



3) $P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)$ 2) $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$ 1) $\beta_T(i) = 1$

17

HMM Outline

1. Score: Given observation sequence $\boldsymbol{O} = \{o_1, \ldots, o_T\}$, and HMM $\lambda = \{\boldsymbol{A}, \boldsymbol{B}, \pi\}$, how do we compute the probability $P(\boldsymbol{O}|\lambda)$?
   – Forward-Backward algorithm
2. Match: Given $\boldsymbol{O} = \{o_1, \ldots, o_T\}$, how do we choose the optimum underlying state sequence $\boldsymbol{Q} = \{q_1, \ldots, q_T\}$?
   – Viterbi algorithm
3. Train: How to learn ML parameter estimates for $\lambda = \{\boldsymbol{A}, \boldsymbol{B}, \pi\}$?
   – Baum-Welch Estimation

18

Finding Optimal State Sequences: Viterbi Algorithm

- The Viterbi Algorithm chooses the state sequence which maximizes $P(\boldsymbol{Q}|\boldsymbol{O},\lambda)$ (or $P(\boldsymbol{Q},\boldsymbol{O}|\lambda)$)
- Define $\delta_t(i)$ as the highest probability along a single path to state s_i at time t, which accounts for the first t observations

$$\delta_t(i) = \max_{q_1 \dots q_{t-1}} P(q_1 \dots q_{t-1}, q_t = s_i, o_1 \dots o_t|\lambda)$$

- By induction (due to DP optimal substructure):

$$\delta_{t+1}(j) = \left[\max_i \delta_t(i) a_{ij}\right] b_j(o_{t+1})$$

  – Note similarity to the forward algorithm (except max instead of sum)
- To retrieve the best state sequence, we also keep track of the state sequence which gave the best path to state s_i at time t
  – This is done in a separate array $\psi_t(i)$ (i.e., pointer to best prior index)

19

The Viterbi Algorithm

1. Initialization: $\delta_1(i) = \pi_i b_i(o_1) \qquad 1 \le i \le N$

   $\psi_1(i) = 0$

2. Recursion:

$$\delta_{t+1}(j) = \max_i\left[\delta_t(i) a_{ij}\right] b_j(o_{t+1}) \qquad 1 \le t < T \quad 1 \le j \le N$$

$$\psi_{t+1}(j) = \arg\max_i\left[\delta_t(i) a_{ij}\right]$$

3. Termination:

$$P^* = \max_i \delta_T(i)$$

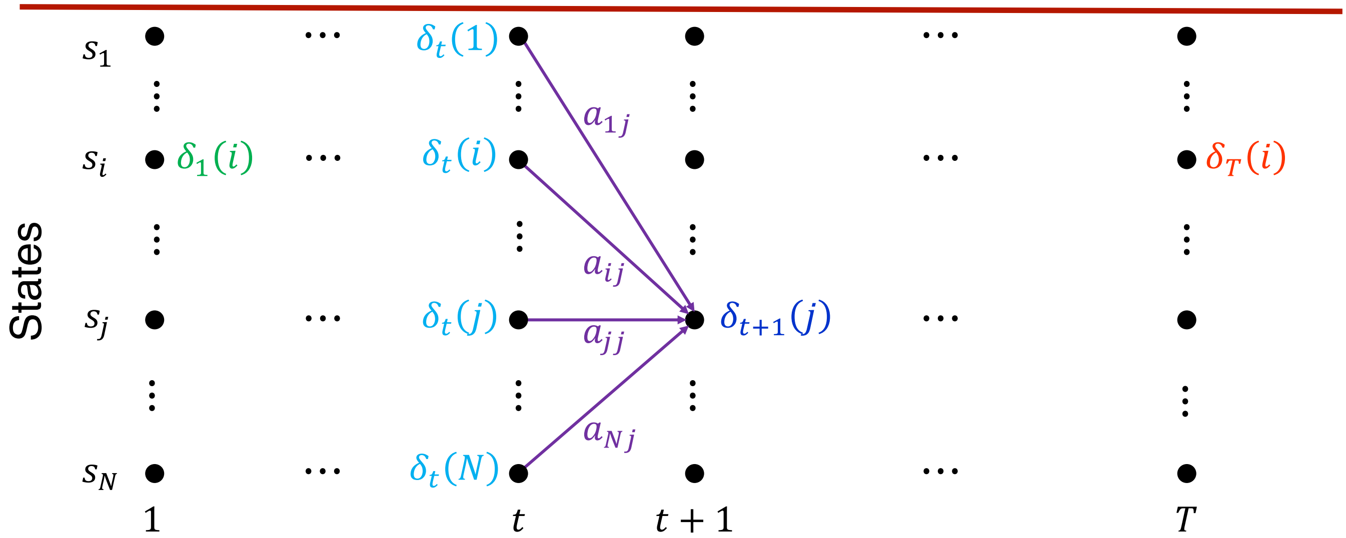
$$q_T^* = \arg\max_i \delta_T(i)$$

4. Path (state-sequence) backtracking

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad 1 \le t < T$$

20

The Viterbi Algorithm

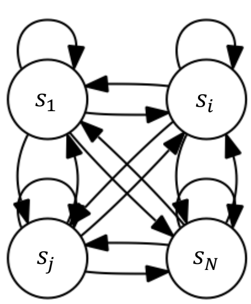


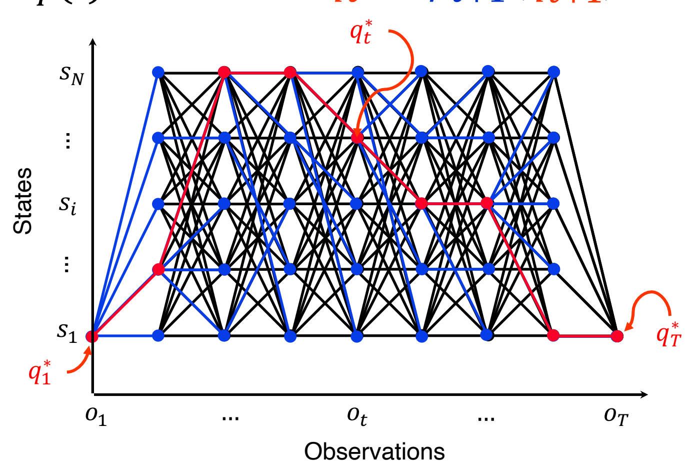
1) $\delta_1(i) = \pi_i \, b_i(o_1)$ 2) $\delta_{t+1}(j) = \max_i \left[\delta_t(i) a_{ij}\right] b_j(o_{t+1})$ 3) $P^* = \max_i \delta_T(i)$

The Viterbi Backtrace

- The Viterbi backtrace begins after the forward recursion completes
  - The backtrace is typically a fraction of the overall computation

$$q_T^* = \arg\max_i \delta_T(i) \qquad q_t^* = \psi_{t+1}(q_{t+1}^*)$$





HMM Outline

1. Score: Given observation sequence $\boldsymbol{O} = \{o_1, \dots, o_T\}$, and HMM $\lambda = \{\boldsymbol{A}, \boldsymbol{B}, \pi\}$, how do we compute the probability $P(\boldsymbol{O}|\lambda)$?
   – Forward-Backward algorithm
2. Match: Given $\boldsymbol{O} = \{o_1, \dots, o_T\}$, how do we choose the optimum underlying state sequence $\boldsymbol{Q} = \{q_1, \dots, q_T\}$?
   – Viterbi algorithm
3. Train: How to learn ML parameter estimates for $\lambda = \{\boldsymbol{A}, \boldsymbol{B}, \pi\}$?
   – Baum-Welch Estimation

23

Baum-Welch Estimation

- Baum-Welch estimation uses EM to determine HMM parameters
- Define $\xi_t(i,j)$ as the probability of being in state s_i at time t and state s_j at time $t+1$, given the model and observation sequence

$$\xi_t(i,j) = P(q_t = s_i, q_{t+1} = s_j |\, \boldsymbol{O}, \lambda) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(\boldsymbol{O}|\lambda)}$$

- Use $\gamma_t(i)$ (probability of being in state i at time t given observations)

$$\gamma_t(i) = P(q_t = s_i | \boldsymbol{O}, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(\boldsymbol{O}|\lambda)} \qquad \gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

- Baum-Welch parameter estimates are based on expected values

$$\hat{E}(s_i \to s_j) = \sum_{t=1}^{T-1} \xi_t(i,j) \qquad \hat{E}(s_j, w_k) = \sum_{\substack{t=1 \\ o_t = w_k}}^{T} \gamma_t(j)$$

24

Baum-Welch Estimation Formulas

Initialization

$$\hat{\pi}_i = \hat{E}(q_1 = s_i) = \gamma_1(i)$$

Expected number of times in state s_i at $t = 1$

Transition

$$\hat{a}_{ij} = \frac{\hat{E}(s_i \to s_j)}{\hat{E}(s_i \to s_*)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\frac{\text{Expected number of transitions from } s_i \text{ to } s_j}{\text{Expected number of transitions from } s_i}$$

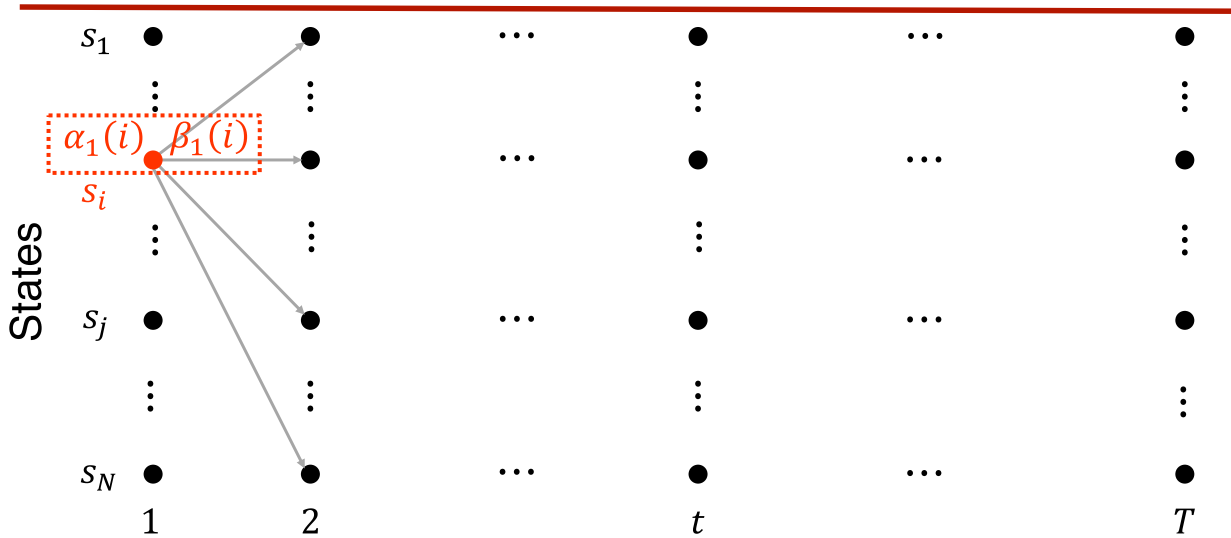
Observation

$$\hat{b}_j(k) = \frac{\hat{E}(s_j, w_k)}{\hat{E}(s_j)} = \frac{\sum_{\substack{t=1 \\ o_t = w_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

$$\frac{\text{Expected number of times in state } s_j \text{ with symbol } w_k}{\text{Expected number of times in state } s_j}$$

25

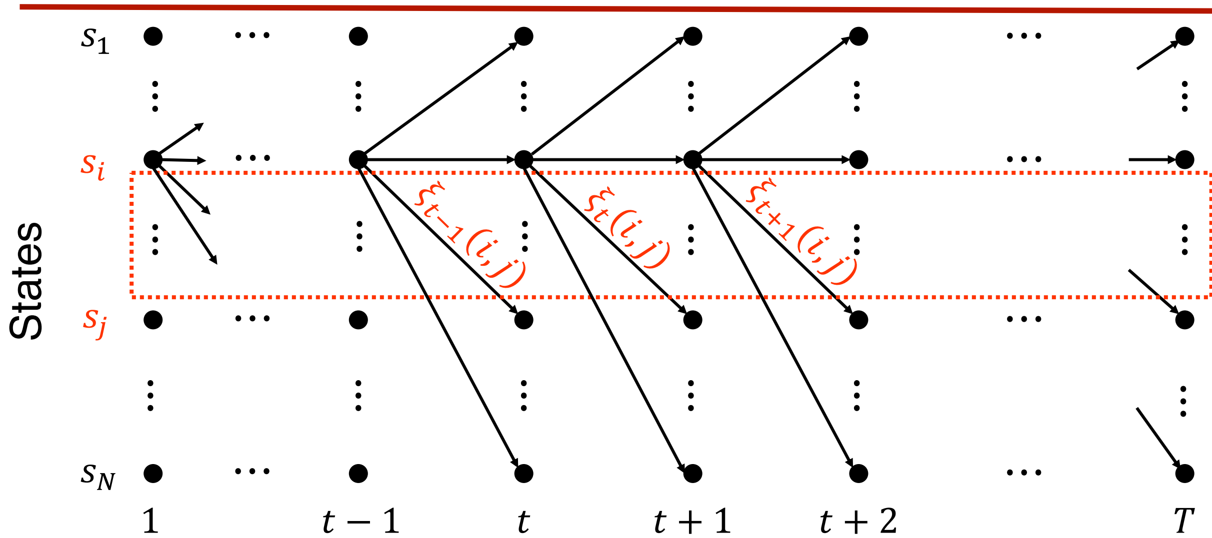
Baum-Welch State Initialization Estimation



$\hat{\pi}_i = \hat{E}(q_1 = s_i) = \gamma_1(i)$ Expected number of times in state s_i at $t = 1$

26

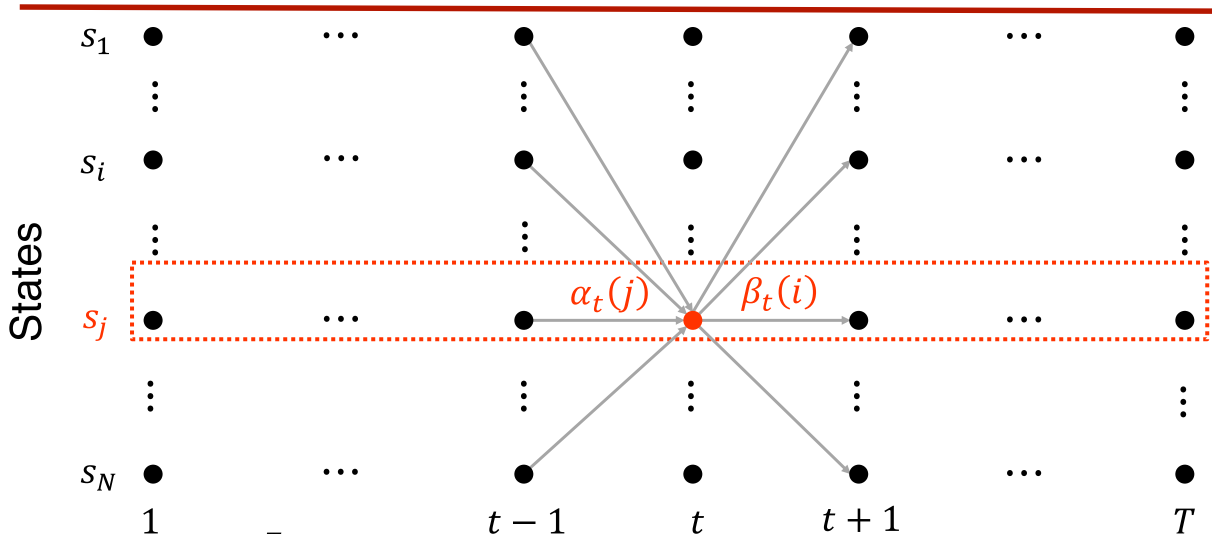
Baum-Welch State Transition Estimation



$$\hat{E}(s_i \to s_j) = \sum_{t=1}^{T-1} \xi_t(i,j) \qquad \hat{E}(s_i \to s_*) = \sum_{j=1}^{N} \hat{E}(s_i \to s_j) \qquad \hat{a}_{ij} = \frac{\hat{E}(s_i \to s_j)}{\hat{E}(s_i \to s_*)}$$

27

Baum-Welch State Observation Estimation



$$\hat{E}(s_j, w_k) = \sum_{\substack{t=1 \\ o_t = w_k}}^{T} \gamma_t(j) \qquad \hat{E}(s_j) = \sum_{t=1}^{T} \gamma_t(j) \qquad \hat{b}_j(k) = \frac{\hat{E}(s_j, w_k)}{\hat{E}(s_j)}$$

28

Baum-Welch Estimation Properties

- If $\lambda = \{\boldsymbol{A}, \boldsymbol{B}, \pi\}$ is the initial model, and $\hat{\lambda} = \{\widehat{\boldsymbol{A}}, \widehat{\boldsymbol{B}}, \hat{\pi}\}$ is the re-estimated model, then it can be proved that either:
  1. The initial model, λ, defines a critical point of the likelihood function, in which case $\lambda = \hat{\lambda}$, or
  2. $P(\boldsymbol{O}|\hat{\lambda}) > P(\boldsymbol{O}|\lambda)$: we have found a new model that is more likely to have generated the observation sequence
- Therefore, we can improve the likelihood $P(\boldsymbol{O}|\lambda)$ if we iterate the re-estimation until some convergence threshold
- The resulting model is called maximum likelihood HMM
  - It is possible to over-fit parameters on a training set: $P(\boldsymbol{O}|\hat{\lambda}) > P(\boldsymbol{O}|\lambda_{true})$

29

Training with a Corpus

- In practice, many observation sequences $\boldsymbol{O} = \{\boldsymbol{O}^1, \dots, \boldsymbol{O}^L\}$ used
  - BW estimation formulas modified to add up counts for each sequence
  - Assume that observation sequences are mutually independent

$$P(\boldsymbol{O}|\lambda) = \prod_{l=1}^{L} P(\boldsymbol{O}^l|\lambda)$$

- Modifications accumulate expected counts across sequences
  - Forward-backward run on each "sentence" individually, then accumulated across the entire training "corpus"

$$\hat{a}_{ij} = \frac{\hat{E}(s_i \to s_j)}{\hat{E}(s_i \to s_*)} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T_l - 1} \xi_t^l(i,j)}{\sum_{l=1}^{L} \sum_{t=1}^{T_l - 1} \gamma_t^l(i)} \qquad \hat{b}_j(k) = \frac{\hat{E}(s_j, w_k)}{\hat{E}(s_j)} = \frac{\sum_{l=1}^{L} \sum_{t=1,\, o_t = w_k}^{T_l} \gamma_t^l(j)}{\sum_{l=1}^{L} \sum_{t=1}^{T_l} \gamma_t^l(j)}$$

30

Practical HMM Issues

- The forward-backward and Viterbi recursions can result in long sequences of probabilities being multiplied that can cause underflow
  - *In practice, computations are performed using logprobs (e.g., your pset!)*
  - In Viterbi, multiplication of probabilities turns into sums of logprobs
  - In forward-backward both multiplications and additions are involved, so probabilities are scaled at each time frame & scale factor retained
- Often the forward-backward algorithm is not used for training, but is replaced by the simpler Viterbi training
  - A Viterbi "best-path" alignment is computed and then used as the basis for estimating transition probabilities, and observation parameters
  - Parameter estimation remains iterative
- For some HMM tasks, relative (and absolute) thresholds are used to eliminate unlikely hypotheses (not admissible, but practical)

31

Final Thoughts

- Hidden Markov models are useful for sequential labeling tasks
  - The Viterbi algorithm is an efficient way to find the best label sequence
  - The Forward-Backward and Baum-Welch estimates enable ML training
- The mathematical formulations for HMMs were developed in 1960s
  - Applied to many disciplines with sequential data e.g., speech recognition (1970s), bioinformatics (1980s), NLP (1990s), handwriting, finance, etc.
- HMMs have been surpassed by discriminative methods such as CRFs and RNNs, but remain popular in low-resource scenarios

32

References

- Readings:
  - Jurafsky & Martin, “Speech and Language Processing,” 2020 (HMMs)

33