Computational Linguistics 1

CMSC/LING 723, LBSC 744

Kristy Hollingshead Seitz
Institute for Advanced Computer Studies
University of Maryland

Lecture 9: 29 September 2011

Agenda

- HW2 – due today
  - Collect printouts
  - Questions about the homework to be posted to the class list, or to compling723.fall2011@gmail.com
  - Language Modeling with <s>
  - Logarithmic Math
- HW3 – online today, due in two weeks
- Forward Algorithm
- Viterbi Algorithm

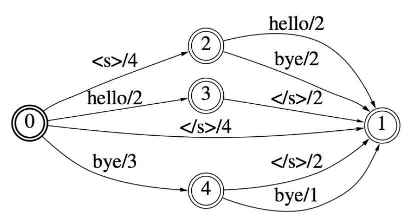
Language Modeling with <s>

- Example corpus

corpus.txt	wl	
<s> hello </s>	ϵ	0
<s> bye </s>	hello	1
<s> hello </s>	bye	2
<s> bye bye </s>	<s>	3
	</s>	4

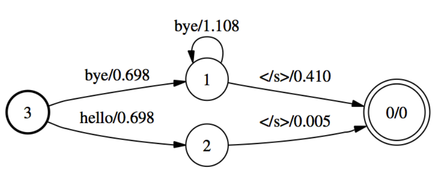
Language Modeling with <s>

- Compile into a (weighted) FSM LM



Language Modeling with <s>

- Determinize and minimize FSM LM



Log-Domain Mathematics

When multiplying many numbers together, we run the risk of underflow errors… one solution is to transform everything into the log domain:

linear domain	***log domain***
x^y	**$e^y \cdot x$**
$x \cdot y$	**$x+y$**
$x+y$	**logAdd(x,y)**

logAdd(*a*,*b*) computes the log-domain sum of *a* and *b* when both *a* and *b* are already in log domain. In the linear domain:

$$\log(x+y) = \log(x + \left[\frac{y}{x} \cdot x\right])$$
$$\log(x\left[1 + \frac{y}{x}\right])$$
$$\log(x) + \log(1 + \frac{y}{x})$$
$$\log(x) + \log(1 + e^{\log(y) - \log(x)})$$

Major point: logAdd(*x*,*y*) is NOT same as log(*x*×*y*) = log(*x*)+log(*y*)

A little trick with logs...

Recall: $e^{x+y} = e^x e^y$

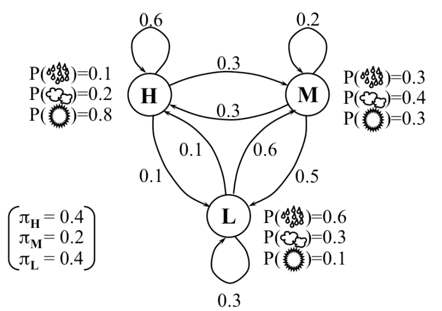
$$\begin{aligned}\log(e^A + e^B) &= \log(e^{B+A-B} + e^B)\\ &= \log(e^B e^{A-B} + e^B)\\ &= \log(e^B(e^{A-B} + 1))\\ &= \log e^B + \log(e^{A-B} + 1)\\ &= B + \log(e^{A-B} + 1)\\ &= A + \log(e^{B-A} + 1)\end{aligned}$$

Don't want e^{A-B} **to be large. Hence, if** $A > B$**, calculate** $A + \log(e^{B-A} + 1)$

Agenda

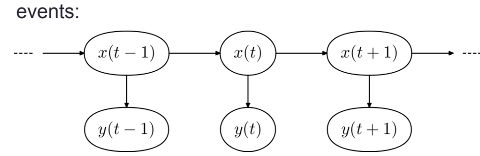
- HW2 – due today
  - Collect printouts
  - Questions about the homework to be posted to the class list, or to compling723.fall2011@gmail.com
  - Language Modeling with <s>
  - Logarithmic Math
- HW3 – online today, due in two weeks
- Forward Algorithm
- Viterbi Algorithm

HMM Example 2: Weather and Atmospheric Pressure



HMM Independence

- The probability of an output symbol depends only on the state generating it
  $$P(o_t | q_1, q_2, \ldots, q_N, o_1, o_2, \ldots, o_T) = P(o_t | q_i)$$
- Where *x* are the (hidden) states and *y* are our (observed) events:



HMM Example 2: Weather and Atmospheric Pressure

What is probability of O={sun, sun, cloud, rain, cloud, sun} and the sequence {H, M, M, L, L, M}, given the model?

= $\pi_H \cdot b_H(s) \cdot a_{HM} \cdot b_M(s) \cdot a_{MM} \cdot b_M(c) \cdot a_{ML} \cdot b_L(r) \cdot a_{LL} \cdot b_L(c) \cdot a_{LM} \cdot b_M(s)$

= 0.4 · 0.8 · 0.3 · 0.3 · 0.2 · 0.4 · 0.5 · 0.6 · 0.3 · 0.3 · 0.6 · 0.3

= 1.12×10^{-5}

What is probability of O={sun, sun, cloud, rain, cloud, sun} and the sequence {H, H, M, L, M, H}, given the model?

= $\pi_H \cdot b_H(s) \cdot a_{HH} \cdot b_H(s) \cdot a_{HM} \cdot b_M(c) \cdot a_{ML} \cdot b_L(r) \cdot a_{LM} \cdot b_M(c) \cdot a_{MH} \cdot b_H(s)$

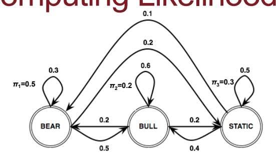
= 0.4 · 0.8 · 0.6 · 0.8 · 0.3 · 0.4 · 0.5 · 0.6 · 0.6 · 0.4 · 0.3 · 0.6

= 2.39×10^{-4}

HMMs: Three Problems

- **Likelihood:** Given an HMM *λ* = (*A*, *B*, ∏), and a sequence of observed events *O*, find *P*(*O*|*λ*)
- **Decoding:** Given an HMM *λ* = (*A*, *B*, ∏), and an observation sequence *O*, find the most likely (hidden) state sequence
- **Learning:** Given a set of observation sequences and the set of states *Q* in *λ*, compute the parameters *A* and *B*

Computing Likelihood



t: 1 2 3 4 5 6

O: ↑ ↓ ↔ ↑ ↓ ↔

$$\begin{bmatrix} P(\uparrow|Bear)=0.1 \\ P(\downarrow|Bear)=0.6 \\ P(\leftrightarrow|Bear)=0.3 \end{bmatrix} \begin{bmatrix} P(\uparrow|Bull)=0.7 \\ P(\downarrow|Bull)=0.1 \\ P(\leftrightarrow|Bull)=0.2 \end{bmatrix} \begin{bmatrix} P(\uparrow|Static)=0.3 \\ P(\downarrow|Static)=0.3 \\ P(\leftrightarrow|Static)=0.4 \end{bmatrix}$$

λ_{stock}

Assuming λ_{stock} models the stock market, how likely are we to observe the sequence of outputs?

Computing Likelihood

- Easy, right?
  - Sum over all possible ways in which we could generate O from λ

$$P(O|\lambda) = \sum_Q P(O,Q|\lambda) = \sum_Q P(O|Q,\lambda)P(Q|\lambda)$$

$$= \sum_{q_1,q_2 \ldots q_T} \pi_{q_1} b_{q_1}(o_1) a_{q_1 q_2} \ldots a_{q_{T-1} q_T} b_{q_T}(o_T)$$

  - What's the problem? Takes O(N^T) time to compute!
- Right idea, wrong algorithm!

Computing Likelihood

- What are we doing wrong?
  - State sequences may have a lot of overlap…
  - We're recomputing the shared subsequences every time
  - Let's store intermediate results and reuse them!
  - Can we do this?
- Sounds like a job for dynamic programming!

Forward Algorithm

- Use an $N \times T$ trellis or chart [α_{tj}]
- Forward probabilities: α_{tj} or $\alpha_t(j)$
  - = P(being in state j after seeing t observations)
  - = $P(o_1, o_2, \ldots o_t, q_t=j)$
- Each cell = ∑ extensions of all paths from other cells
  $\alpha_t(j) = \sum_i \alpha_{t-1}(i)\, a_{ij}\, b_j(o_t)$
  - $\alpha_{t-1}(i)$: forward path probability until (t-1)
  - a_{ij}: transition probability of going from state i to j
  - $b_j(o_t)$: probability of emitting symbol o_t in state j
- $P(O|\lambda) = \sum_i \alpha_T(i)$
- What's the running time of this algorithm?

Forward Algorithm: Formal Definition

- Initialization

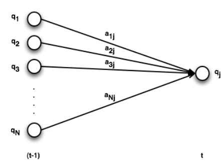
$$\alpha_1(j) = \pi_j b_j(o_1), 1 \le j \le N$$

- Recursion

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); 1 \le j \le N, 2 \le t \le T$$

- Termination

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

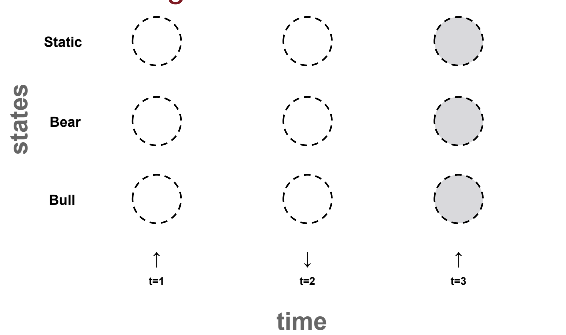


Forward Algorithm

O = ↑ ↓ ↑

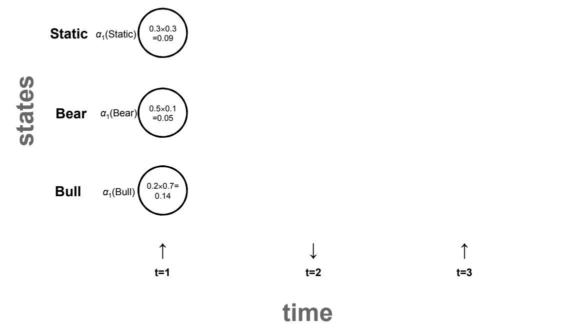
find $P(O|\lambda_{stock})$

Forward Algorithm



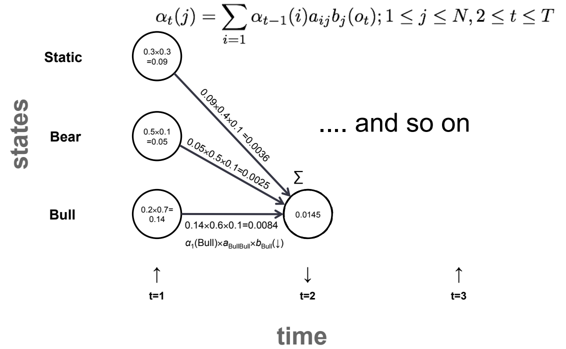
Forward Algorithm: Initialization

$$\alpha_1(j) = \pi_j b_j(o_1), 1 \le j \le N$$



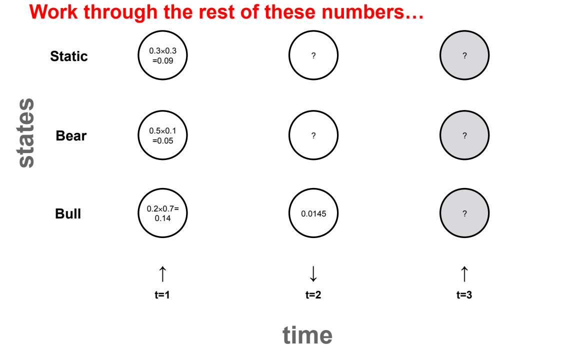
Forward Algorithm: Recursion

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); 1 \le j \le N, 2 \le t \le T$$



Forward Algorithm: Recursion

Work through the rest of these numbers…

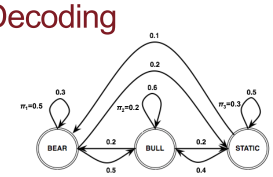


What's the asymptotic complexity of this algorithm?

HMMs: Three Problems

- **Likelihood:** Given an HMM λ = (A, B, ∏), and a sequence of observed events O, find $P(O|\lambda)$
- **Decoding:** Given an HMM λ = (A, B, ∏), and an observation sequence O, find the most likely (hidden) state sequence
- **Learning:** Given a set of observation sequences and the set of states Q in λ, compute the parameters A and B

Decoding



t: 1 2 3 4 5 6

O: ↑ ↓ ↔ ↑ ↓ ↔

$$\begin{bmatrix} P(\uparrow|Bear) = 0.1 \\ P(\downarrow|Bear) = 0.6 \\ P(\leftrightarrow|Bear) = 0.3 \end{bmatrix} \quad \begin{bmatrix} P(\uparrow|Bull) = 0.7 \\ P(\downarrow|Bull) = 0.1 \\ P(\leftrightarrow|Bull) = 0.2 \end{bmatrix} \quad \begin{bmatrix} P(\uparrow|Static) = 0.3 \\ P(\downarrow|Static) = 0.3 \\ P(\leftrightarrow|Static) = 0.4 \end{bmatrix}$$

λ_{stock}

Given λ_{stock} as our model and O as our observations, what are the most likely states the market went through to produce O?

Decoding

- "Decoding" because states are hidden
- First try:
  - Compute $P(O)$ for all possible state sequences, then choose sequence with highest probability
  - What's the problem here?
- Second try:
  - For each possible hidden state sequence, compute $P(O)$ using the forward algorithm
  - What's the problem here?

Viterbi Algorithm

- "Decoding" = computing most likely state sequence
  - Another dynamic programming algorithm
  - Efficient: polynomial vs. exponential (brute force)
- Same idea as the forward algorithm
  - Store intermediate computation results in a trellis
  - Build new cells from existing cells

Viterbi Algorithm

- Use an $N \times T$ trellis $[v_{tj}]$
  - Just like in forward algorithm
- v_{tj} or $v_t(j)$
  - = P(in state j after seeing t observations and passing through the most likely state sequence so far)
  - = $P(q_1, q_2, \ldots q_{t-1}, q_{t=j}, o_1, o_2, \ldots o_t)$
- Each cell = extension of most likely path from other cells
  $v_t(j) = \max_i v_{t-1}(i)\, a_{ij}\, b_j(o_t)$
  - $v_{t-1}(i)$: Viterbi probability until ($t-1$)
  - a_{ij}: transition probability of going from state i to j
  - $b_j(o_t)$: probability of emitting symbol o_t in state j
- $P = \max_i v_T(i)$

Viterbi vs. Forward

- Maximization instead of summation over previous paths
- This algorithm is still missing something!
  - In forward algorithm, we only care about the probabilities
  - What's different here?
- We need to store the most likely path (transition):
  - Use "backpointers" to keep track of most likely transition
  - At the end, follow the chain of backpointers to recover the most likely state sequence

Viterbi Algorithm: Formal Definition

- Initialization

$$v_1(j) = \pi_i b_i(o_1); 1 \le i \le N$$
$$BT_1(i) = 0$$

- Recursion

$$v_t(j) = \max_{i=1}^{N} [v_{t-1}(i) a_{ij}]\, b_j(o_t); 1 \le i \le N, 2 \le t \le T$$

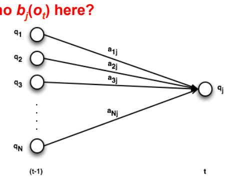
But here?

$$BT_1(i) = \arg\max_{i=1}^{N} [v_{t-1}(i) a_{ij}]$$

Why no $b_j(o_t)$ here?

- Termination

$$P^* = \max_{1=1}^{N} v_T(j)$$
$$q_T^* = \arg\max_{1=i}^{N} v_T(j)$$

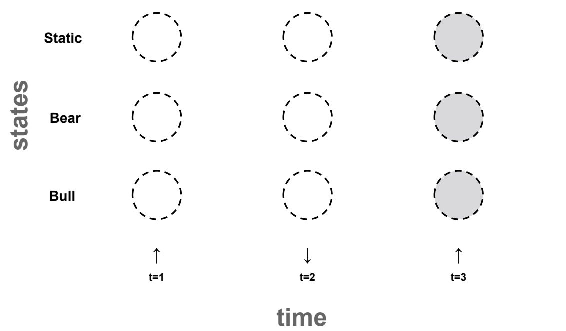


Viterbi Algorithm

O = ↑ ↓ ↑

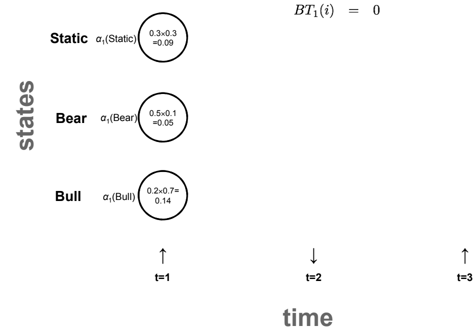
find most likely state sequence given λ_{stock}

Viterbi Algorithm



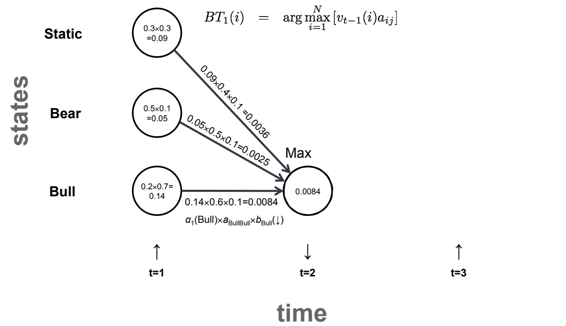
Viterbi Algorithm: Initialization

$$v_1(j) = \pi_i b_i(o_1); 1 \le i \le N$$
$$BT_1(i) = 0$$



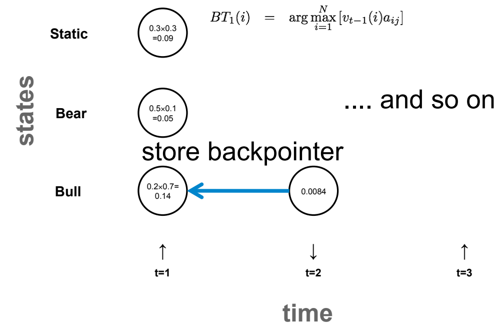
Viterbi Algorithm: Recursion

$$v_t(j) = \max_{i=1}^{N} [v_{t-1}(i) a_{ij}]\, b_j(o_t); 1 \le i \le N, 2 \le t \le T$$
$$BT_1(i) = \arg\max_{i=1}^{N} [v_{t-1}(i) a_{ij}]$$



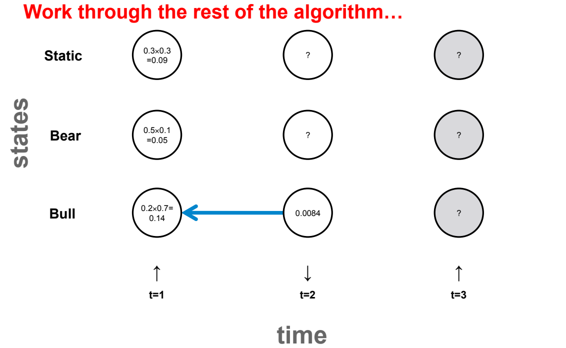
Viterbi Algorithm: Recursion

$$v_t(j) = \max_{i=1}^{N} [v_{t-1}(i) a_{ij}]\, b_j(o_t); 1 \le i \le N, 2 \le t \le T$$
$$BT_1(i) = \arg\max_{i=1}^{N} [v_{t-1}(i) a_{ij}]$$



Viterbi Algorithm: Recursion

Work through the rest of the algorithm…



POS Tagging with HMMs

Modeling the problem

- What's the problem?
  - The/DT grand/JJ jury/NN commmented/VBD on/IN a/DT number/ NN of/IN other/JJ topics/NNS ./.
- What should the HMM look like ?
  - States: part-of-speech tags (t_1, t_2, ..., t_N)
  - Output symbols: words (w_1, w_2, ..., $w_{|V|}$)
- Given HMM *λ* (A, B, ∏), POS tagging = reconstructing the best state sequence given input
  - Use Viterbi decoding (best = most likely)
- But wait…

HMM Training

- What are appropriate values for *A*, *B*, ∏?
- Before HMMs can decode, they must be trained…
  - *A*: transition probabilities
  - *B*: emission probabilities
  - ∏: prior
- Two training methods:
  - Supervised training: start with tagged corpus, count stuff to estimate parameters
  - Unsupervised training: start with untagged corpus, bootstrap parameter estimates and improve estimates iteratively

HMMs: Three Problems

- **Likelihood:** Given an HMM *λ* = (*A*, *B*, ∏), and a sequence of observed events *O*, find *P*(*O*|*λ*)
- **Decoding:** Given an HMM *λ* = (*A*, *B*, ∏), and an observation sequence *O*, find the most likely (hidden) state sequence
- **Learning:** Given a set of observation sequences and the set of states *Q* in *λ*, compute the parameters *A* and *B*

Supervised Training

- A tagged corpus tells us the hidden states!
- We can compute Maximum Likelihood Estimates (MLEs) for the various parameters
  - MLE = fancy way of saying "count and divide"
- These parameter estimates maximize the likelihood of the data being generated by the model

Supervised Training

- Transition Probabilities
  - Any $P(t_i \mid t_{i-1}) = C(t_{i-1}, t_i) / C(t_{i-1})$, from the tagged data
  - Example: for P(NN|VB), count how many times a noun follows a verb and divide by the total number of times you see a verb
- Emission Probabilities
  - Any $P(w_i \mid t_i) = C(w_i, t_i) / C(t_i)$, from the tagged data
  - For *P*(bank|NN), count how many times bank is tagged as a noun and divide by how many times anything is tagged as a noun
- Priors
  - Any $P(q_1 = t_i) = \pi_i = C(t_i)/N$, from the tagged data
  - For π_{NN} , count the number of times NN occurs and divide by the total number of tags (states)
  - A better way?

Agenda

- HW2 – due today
  - Collect printouts
  - Questions about the homework to be posted to the class list
- HW3 – online today, due in two weeks
- Language Modeling with <s>
- Logarithmic Math
- Forward Algorithm, Viterbi Algorithm
- Next time:
  - Unsupervised training 'teaser'
  - Other HMM/tagging tasks