Language and Computation

week 6, Thursday, February 20, 2014

Tamás Biró
Yale University
tamas.biro@yale.edu
http://www.birot.hu/courses/2014-LC/

Practical matters

- **Post-reading:** Chapter 5: 5.5. Chapter 6: intro and 6.1-6.5
- **Pre-reading:** Chapter 12 (intro to syntax and comput. syntax)
- **Python:** H 6-10, especially `re` in Chapt. 10.
- **Sections:** Python NLTK
  Bird, Klein, Loper: *Natural Language Processing with Python*, Ch 1, `http://www.nltk.org/book/ch01.html`
- **Homework 3** posted by the weekend, due 03/04.

Today

Hidden Markov models and Ferguson's three problems:

- The Viterbi Algorithm
- The Forward Algorithm
- The Forward-Backward Algorithm

Recap: two examples of Markov Models

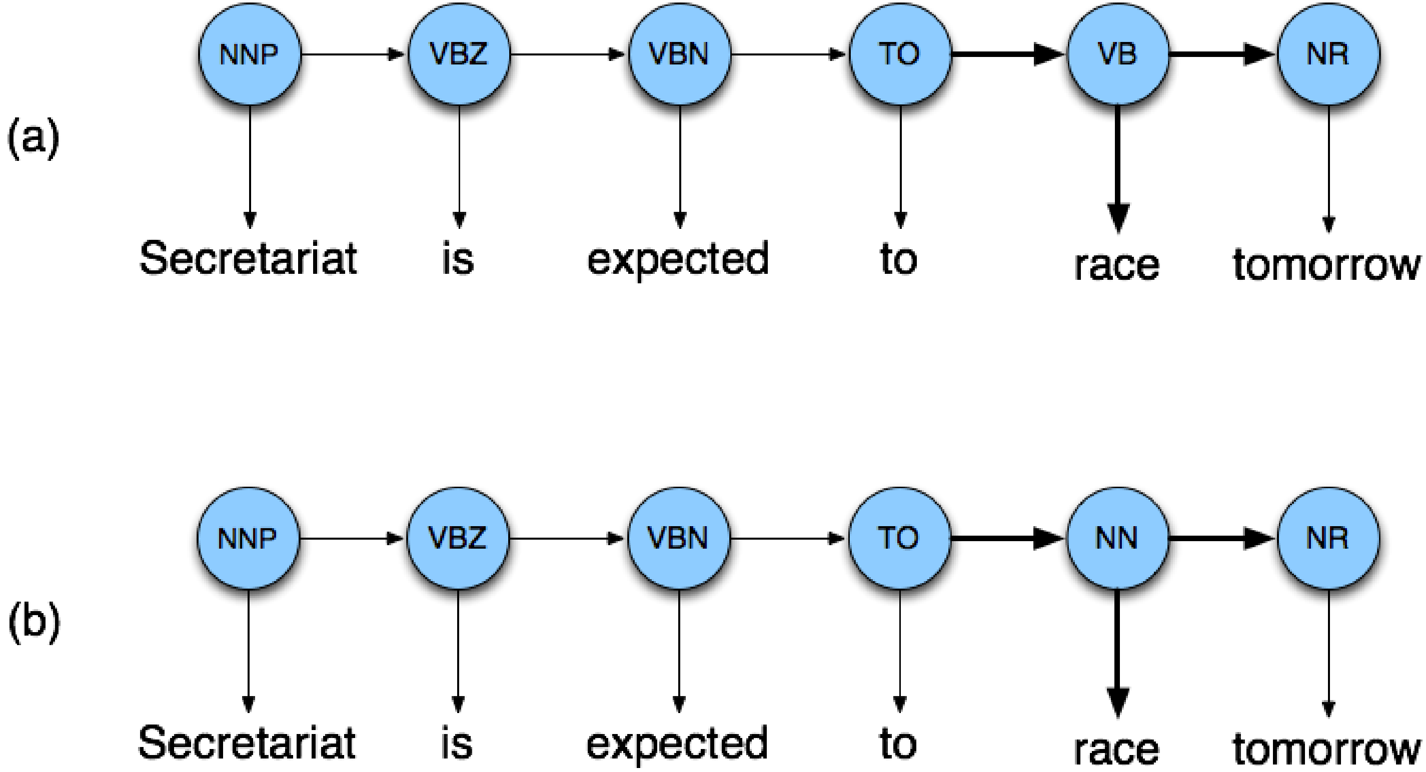
Markov Models

- A: model of underlying series of "causes" (states)

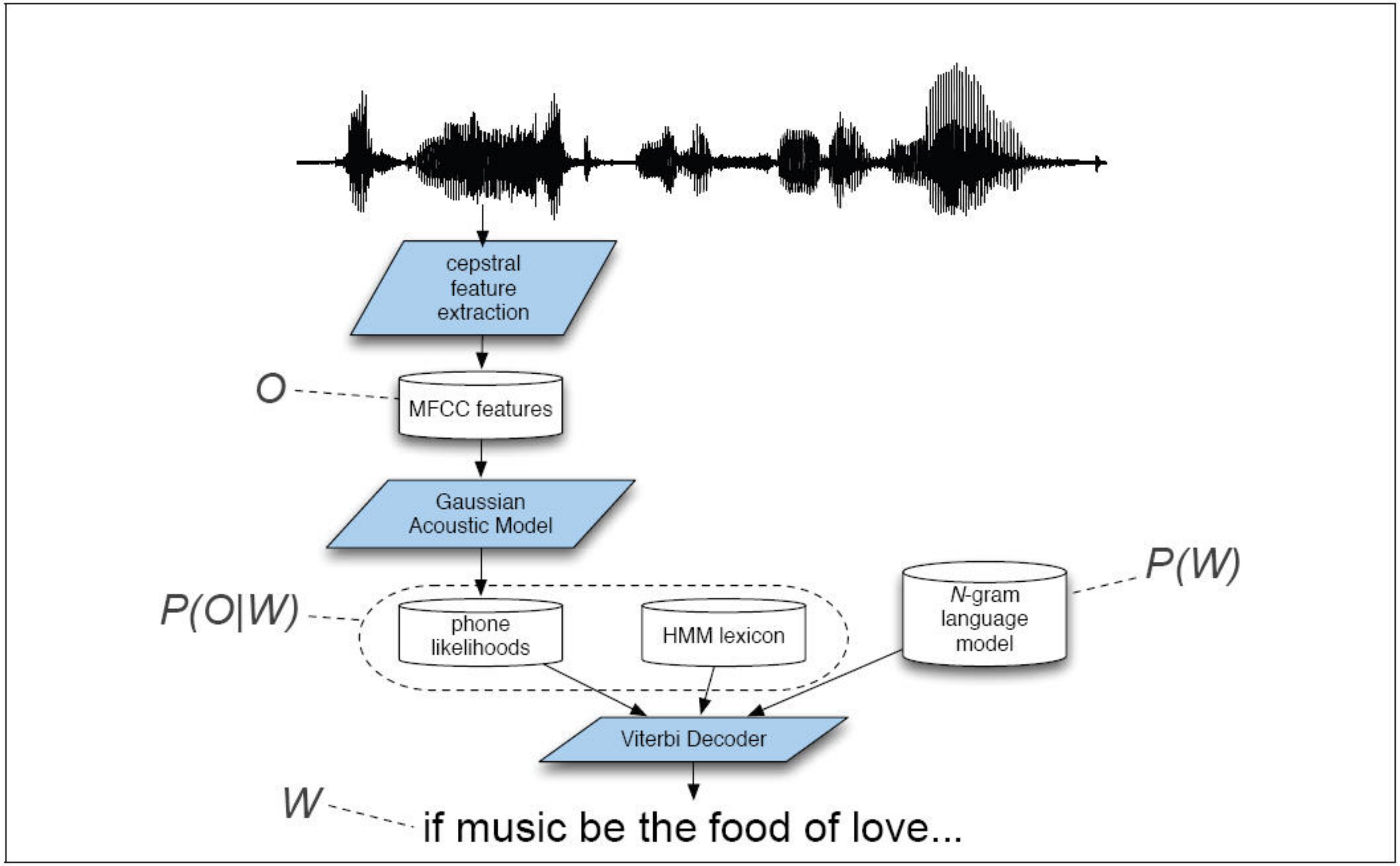
- B: model of observable series of "effects" (emitted signs)

- Given observations, we are interested in their causes.

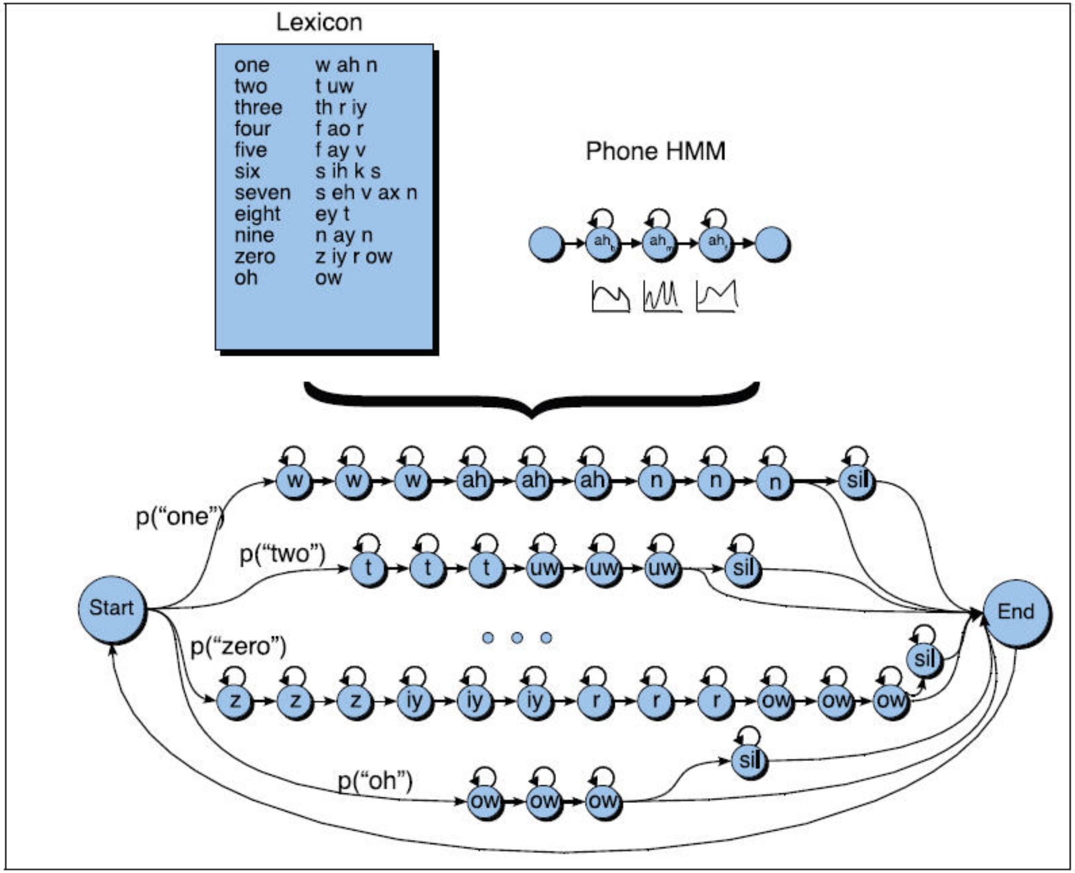
Part-of-Speech Tagging



Speech recognition



Speech recognition



Bayesian inference

- Given observation B, most likely cause A:
  $\arg\max_A P(A|B) =?$

- Bayes' theorem:
$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

  Hence,
$$\arg\max_A P(A|B) = \arg\max_A (P(B|A) \cdot P(A))$$

- $P(A) =$ **prior probabilities**. $P(B|A) =$ **likelihood**.

Markov Chains and Markov Models

Markov Chain: "First-order observable Markov Model"

- No characters read/emitted! That is, characters = states.
- Set of states $Q = \{q_1, \ldots q_n\}$. The state at time t is $q[t]$.
- a_{ij}: probability transitioning $q_i \to q_j$.
  Transition matrix $A = (a_{ij})$. Normed to 1: $\sum_{j=1}^{n} a_{ij} = 1$
- Current state depends **only** on previous state:

$$P(q[t_i] \mid q[t_1] \ldots q[t_{i-1}]) = P(q[t_i] \mid q[t_{i-1}]) = a_{q[t_{i-1}],q[t_i]}$$

Markov Chain: "First-order observable Markov Model"

- Given Markov Chain, generate a string: trivial.
- Given string, learn a Markov Model:
  - $Q =$ observation types.
  - $a_{i,j} = P(q_j|q_i) =?$
  - *Maximum Likelihood Estimate*:

$$a_{i,j} = P(q_j|q_i) = \frac{P(q_i q_j)}{P(q_i)} = \frac{\#\text{ of } q_i q_j \text{ bigrams}}{\#\text{ of } q_i \text{ unigrams}}$$

  - Laplace Smoothing, Good-Turing Discounting, interpolation, backoff.

Markov Models

Probabilistic/Weighted Finite State Automaton

Add probability to transitions:

- A quintuple $(Q, \Sigma, q_0, F, \delta(q, i))$
- $\delta(q, i)$ is
  - $\in Q$ for a **deterministic FSA**
  - $\subseteq Q$ for a **non-deterministic FSA**
  - a probability distribution over Q for a **probabilistic FSA**
- When in state q_j and read character i from input tape: move to state q_k with probability $\delta(q_k, i)[q_k]$, for all $q_k \in Q$.

Markov Models: sextuple $(Q, \Sigma, q_0, Q_F, A, B)$

Slightly different terminology, slightly different idea.

- Q finite set of states $q_1, q_2, \ldots, q_N$.
  Σ set of possible observations (finite? not finite?)
- q_0 start state (or probability distribution π over Q)
  q_F end (final) state (or $F \subseteq Q$?)
- A **transition probability matrix**: $\forall i : \sum_{j=1}^{N} a_{ij} = 1$
- B **emission probabilities**: $\forall i : \sum_{o \in \Sigma} b_i(o) = 1$

(Hidden) Markov Models

$Q = q_1 q_2 \ldots q_N$	a set of **states**
$A = a_{01} a_{02} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_N$	a set of **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$.
$B = b_i(o_t)$	a set of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
q_0, q_{end}	special **start and end states** that are not associated with observations

(Hidden) Markov Models

Markov assumption: $P(q[t_i])$ only depends on $q[t_{i-1}]$, and not on previous states or previous outputs.

Output independence: $P(o[t_i])$ only depends on $q[t_i]$ and not on previous states or previous outputs.

(Hidden) Markov Models

- Given MM $\lambda = (A, B)$, generate series of observation: trivial.
- Given MM $\lambda = (A, B)$, given observation sequence O determine:
  - likelihood $P(O|\lambda)$: **forward algorithm**
  - find most probable sequence of states: **Viterbi algorithm**
- Given an observation sequence O, learn A and B: **forward-backward algorithm** (aka Baum-Welch algorithm, special case of Expectation-Maximization/EM algorithm).

Viterbi algorithm

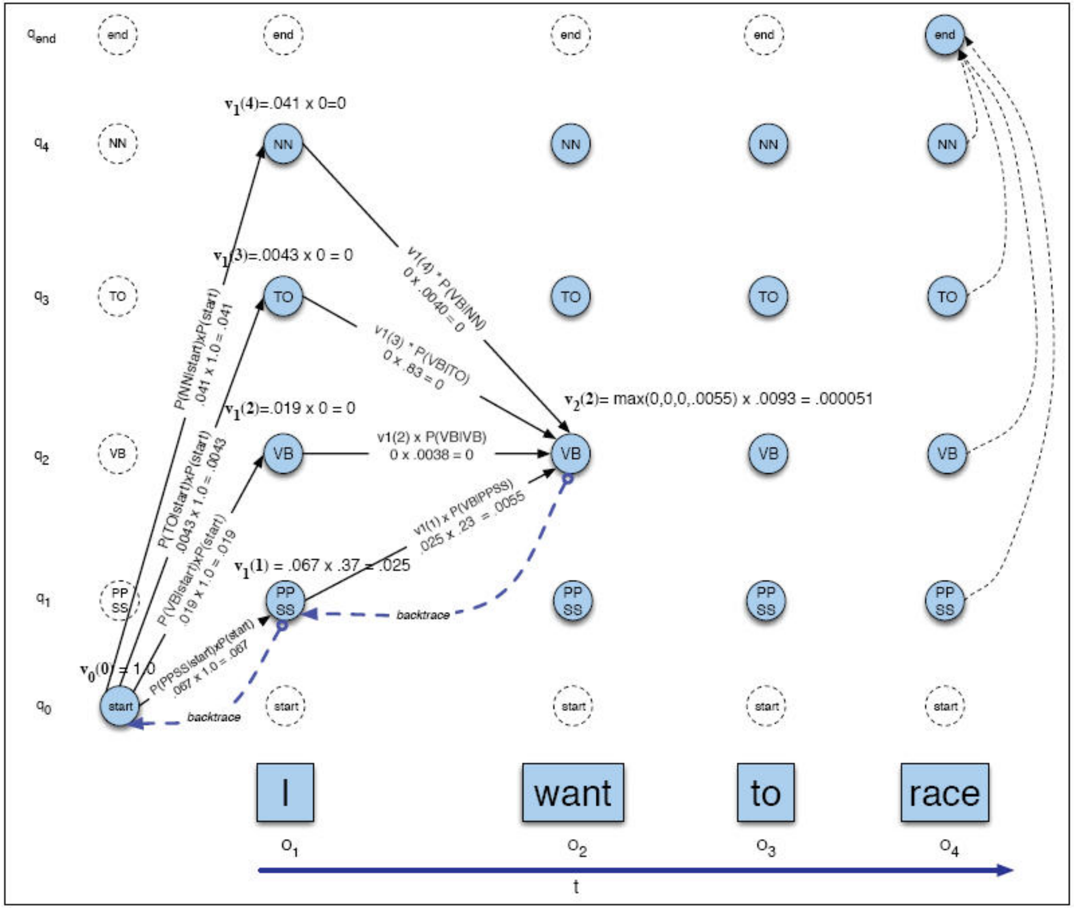
Viterbi algorithm

Problem: **Decoding**

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

Solution:

Viterbi algorithm: dynamic programming, similar to the minimum edit distance algorithm, using a trellis.



v1(4)=.041 x 0=0
v1(3)=.0043 x 0 = 0
v1(2)=.019 x 0 = 0
v1(1) = .067 x .37 = .025
v0(0) = 1.0
v2(2)= max(0,0,0,.0055) x .0093 = .000051
v1(2) x P(VB|VB)
0 x .0038 = 0
backtrace
backtrace
I
want
to
race
t

Viterbi algorithm

$v_{t-1}(i)$	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

$$\forall j : v_t(j) = \max_{i=1}^{n} \left(v_{t-1}(i) \cdot a_{ij} \cdot b_j(o_t) \right)$$

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 $backpointer[q_F,T]$

The Forward Algorithm

Forward algorithm

Problem: **Likelihood**

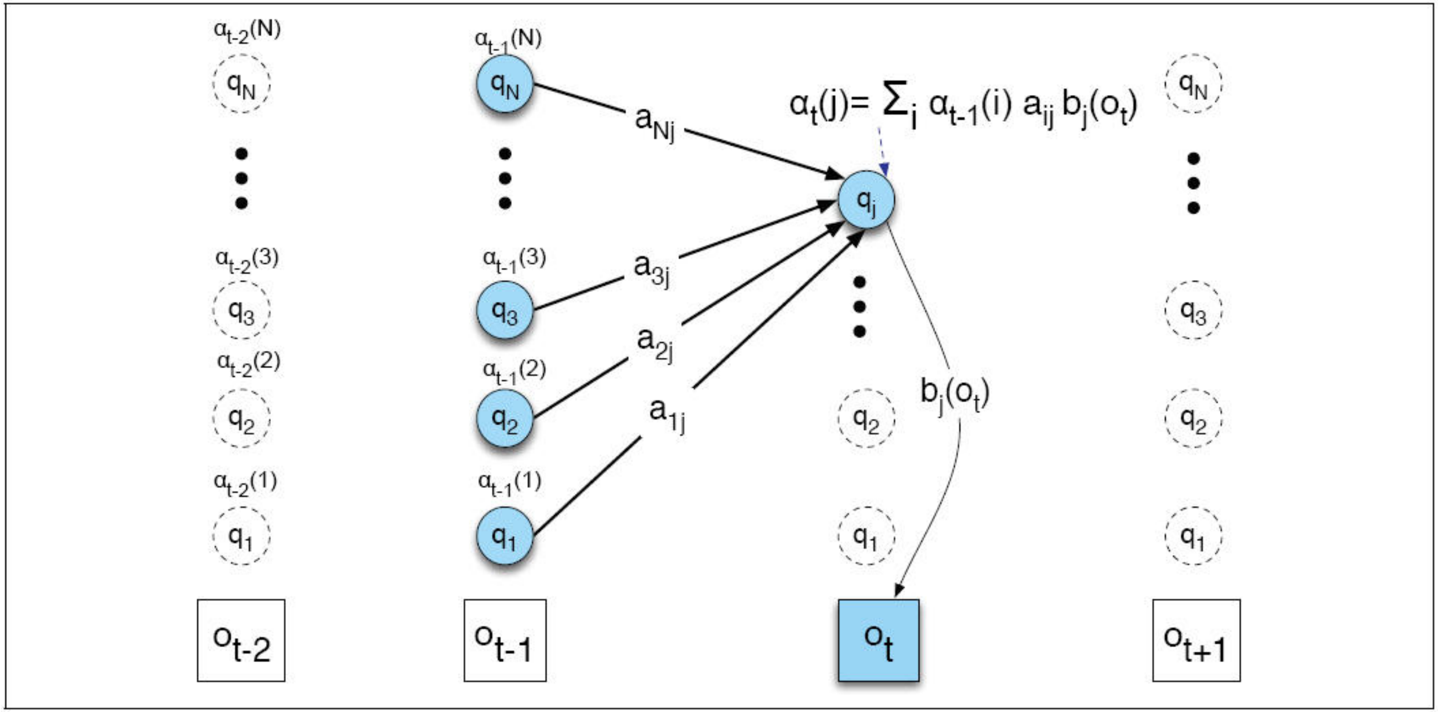
Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, \ldots, o_T$, determine the **likelihood** $P(O|\lambda)$, the probability that HMM λ emits series O.

$$P(O|\lambda) = \sum_{q[t_1],\ldots,q[t_T]} P(o_1, \ldots, o_T \ | \ q[t_1], \ldots, q[t_T], \lambda)$$

Solution:

Forward algorithm: dynamic programming, similar to the minimum edit distance algorithm, using a trellis.

Forward algorithm



Forward algorithm

$\alpha_{t-1}(i)$	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

$$\forall j : \alpha_t(j) = \sum_{i=1}^{n} \alpha_{t-1}(i) \cdot a_{ij} \cdot b_j(o_t)$$

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 $forward[q_F,T]$

The Forward-Backward Algorithm

Forward-Backward algorithm

Problem:

Given as input an observation sequence $O = o_1, o_2, \ldots, o_T$ and the set of possible states in the HMM, **learn** the HMM parameters A and B.

Solution: **Forward-Backward algorithm:**

a.k.a. **Baum-Welch Algorithm**, a special case of the **Expectation-Maximization** (EM) algorithm.

an example of **unsupervised learning**!

Forward-Backward algorithm

- **Forward probability** $\alpha_t(i)$: probability of seeing the observations from time beginning to t, given that we are in state i at time t, and given HMM.

- **Backward probability** $\beta_t(i)$: probability of seeing the observations from time $t+1$ to the end, given that we are in state i at time t, and given HMM.

$$\alpha_t(i) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j)$$

Forward-Backward algorithm

1. Initialize A and B

2. Iterate until convergence

   (a) **E-step:** given current A and B, compute

      i. expected state occupancy count $\gamma_t(j)$: probability of being in state j at time t, given O and HMM

      ii. expected state transition count $\xi_t(i, j)$: probability of being in state i at time t and state j at time $t + 1$, given O and HMM

   (b) **M-step:** recompute A and B probabilities, given current ξ and γ.

3. Return A and B.

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

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \ \forall\, t \text{ and } j$$

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

M-step

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

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

return A, B

See you next week!