9: Viterbi Algorithm for HMM Decoding

Machine Learning and Real-world Data

Simone Teufel and Ann Copestake

Computer Laboratory
University of Cambridge

Lent 2017

Last session: estimating parameters of an HMM

- The dishonest casino, dice edition
- Two states: L (loaded dice), F (fair dice). States are hidden.
- You estimated transition and emission probabilities.
- Now let's now see how well an HMM can discriminate this highly ambiguous situation.
- We need to write a decoder.

Decoding: finding the most likely path

- Definition of decoding: Finding the most likely state sequence X that explains the observations, given this HMM's parameters.

$$\hat{X} = \operatorname*{argmax}_{X_0 \ldots X_{T+1}} P(X|O,\mu) =$$

$$\operatorname*{argmax}_{X_0 \ldots X_{T+1}} \prod_{t=0}^{T+1} P(O_t|X_t)P(X_t|X_{t-1})$$

- Search space of possible state sequences X is O(N^T); too large for brute force search.

Viterbi is a Dynamic Programming Application

(Reminder from Algorithms course)
We can use Dynamic Programming if two conditions apply:

- Optimal substructure property
  - An optimal state sequence $X_0 \dots X_j \dots X_{T+1}$ contains inside it the sequence $X_0 \dots X_j$, which is also optimal
- Overlapping subsolutions property
  - If both X_t and X_u are on the optimal path, with $u > t$, then the calculation of the probability for being in state X_t is part of each of the many calculations for being in state X_u.

Viterbi is a Dynamic Programming Application

(Reminder from Algorithms course)
We can use Dynamic Programming if two conditions apply:

- Optimal substructure property
  - An optimal state sequence $X_0 \dots X_j \dots X_{T+1}$ contains inside it the sequence $X_0 \dots X_j$, which is also optimal
- Overlapping subsolutions property
  - If both X_t and X_u are on the optimal path, with $u > t$, then the calculation of the probability for being in state X_t is part of each of the many calculations for being in state X_u.

The intuition behind Viterbi

- Here's how we can save ourselves a lot of time.
- Because of the Limited Horizon of the HMM, we don't need to keep a complete record of how we arrived at a certain state.
- For the first-order HMM, we only need to record one previous step.
- Just do the calculation of the probability of reaching each state once for each time step.
- Then memoise this probability in a Dynamic Programming table
- This reduces our effort to $O(N^2 T)$.
- This is for the first order HMM, which only has a memory of one previous state.

Viterbi: main data structure

- Memoisation is done using a *trellis*.
- A trellis is equivalent to a Dynamic Programming table.
- The trellis is $N \times (T + 1)$ in size, with states j as rows and time steps t as columns.
- Each cell j, t records the Viterbi probability $\delta_j(t)$, the probability of the optimal state sequence ending in state s_j at time t:

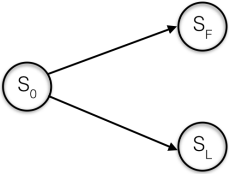
$$\delta_j(t) = \max_{X_0,\ldots,X_{t-1}} P(X_0 \ldots X_{t-1}, o_1 o_2 \ldots o_t, X_t = s_j | \mu)$$

Viterbi algorithm, initialisation

- The initial $\delta_j(1)$ concerns time step 1.
- It stores, for all states, the probability of moving to state s_j from the start state, and having emitted o_1.
- We therefore calculate it for each state s_j by multiplying transmission probability a_{0j} from the start state to s_j, with the emission probability for the first emission o_1.

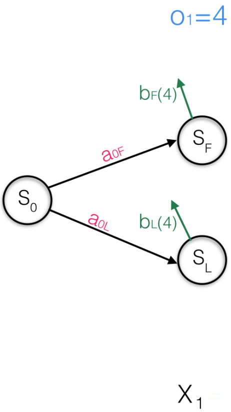
$$\delta_j(1) = a_{0j} b_j(o_1), 1 \leq j \leq N$$

Viterbi algorithm, initialisation

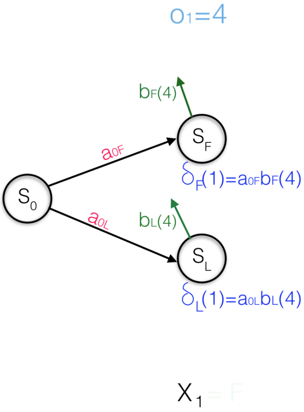


X_1

Viterbi algorithm, initialisation: observation is 4



Viterbi algorithm, initialisation: observation is 4

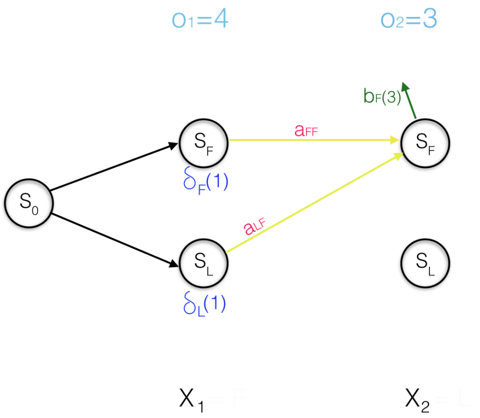


Viterbi algorithm, main step, observation is 3

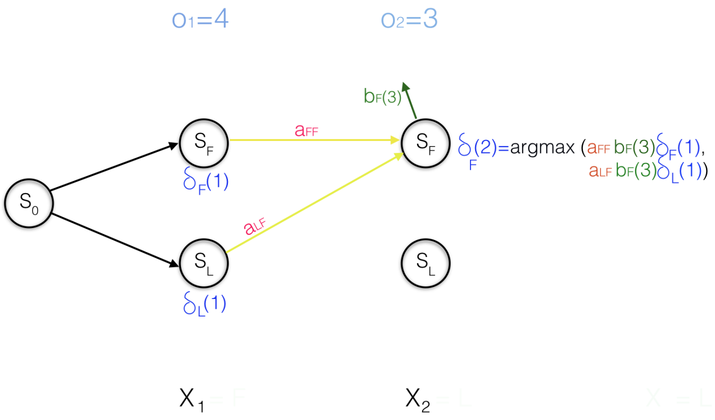
- $\delta_j(t)$ stores the probability of the best path ending in s_j at time step t.
- This probability is calculated by maximising over the best ways of transmitting into s_j for each s_i.
- This step comprises:
  - $\delta_i(t-1)$: the probability of being in state s_i at time $t-1$
  - a_{ij}: the transition probability from s_i to s_j
  - $b_i(o_t)$: the probability of emitting o_t from destination state s_j

$$\delta_j(t) = \max_{1 \leq i \leq N} \delta_i(t-1) \cdot a_{ij} \cdot b_j(o_t)$$

Viterbi algorithm, main step



Viterbi algorithm, main step



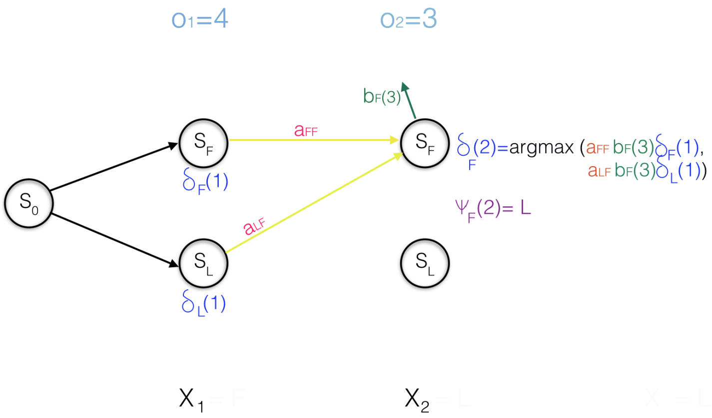
Viterbi algorithm, main step, ψ

- $\psi_j(t)$ is a helper variable that stores the $t-1$ state index i on the highest probability path.

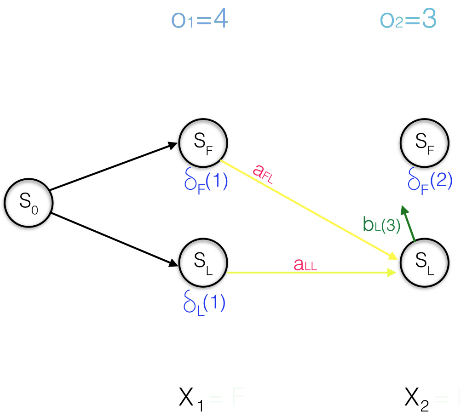
$$\psi_j(t) = \operatorname*{argmax}_{1 \leq i \leq N} \delta_i(t-1) a_{ij} b_j(o_t)$$

- In the backtracing phase, we will use ψ to find the previous cell in the best path.

Viterbi algorithm, main step

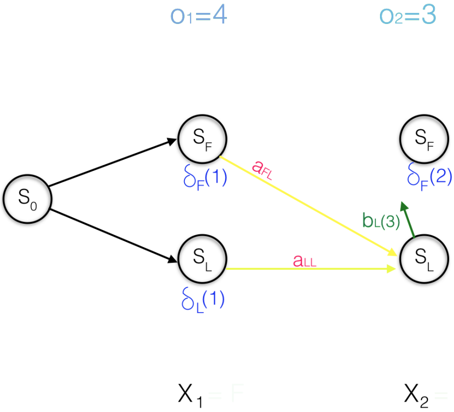


Viterbi algorithm, main step



$$\delta_L(2) = \text{argmax}\,(a_{FL}\, b_L(3)\, \delta_F(1),\; a_{LL}\, b_L(3)\, \delta_L(1))$$

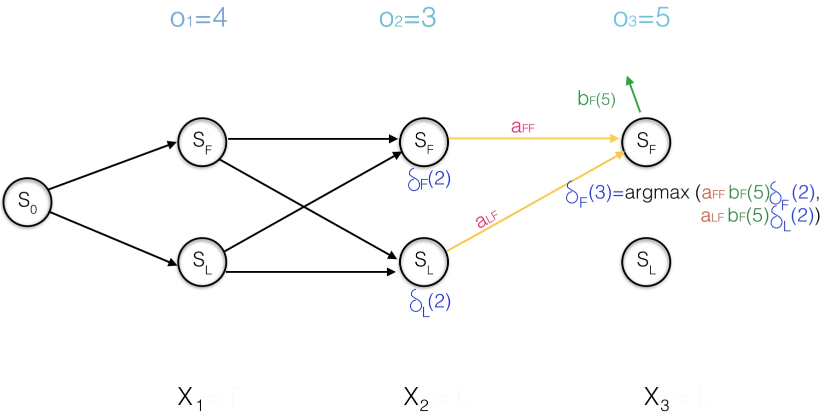
Viterbi algorithm, main step



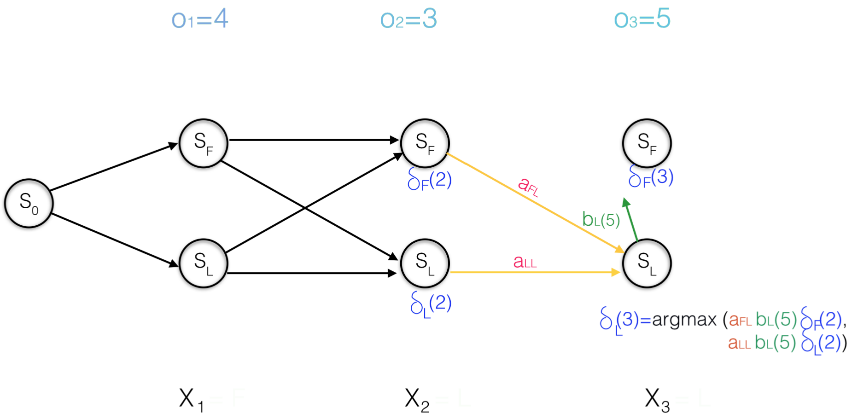
$\delta_L(2) = \text{argmax}\,(a_{FL}\, b_L(3)\, \delta_F(1),$
$a_{LL}\, b_L(3)\, \delta_L(1))$

$\Psi_L(2) = F$

Viterbi algorithm, main step, observation is 5



Viterbi algorithm, main step, observation is 5



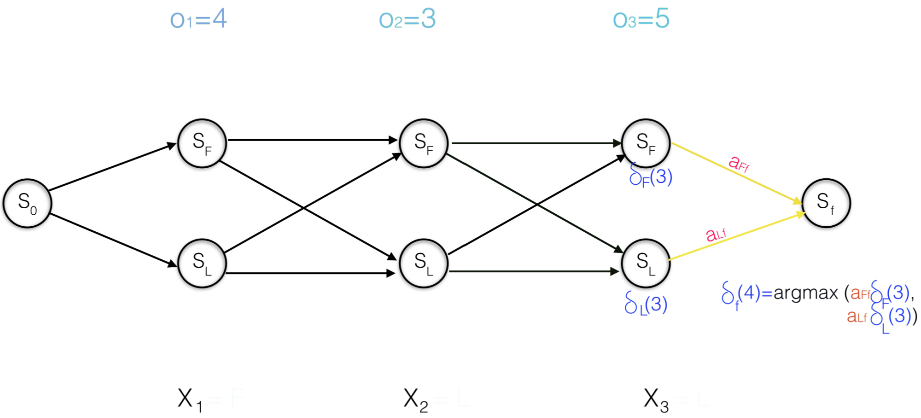
Viterbi algorithm, termination

- $\delta_f(T+1)$ is the probability of the entire state sequence up to point $T+1$ having been produced given the observation and the HMM's parameters.

$$P(X|O,\mu) = \delta_f(T+1) = \max_{1 \le i \le N} \delta_i \cdot (T) a_{if}$$

- It is calculated by maximising over the $\delta_i(T) \cdot a_{if}$, almost as per usual
- Not quite as per usual, because the final state s_f does not emit, so there is no $b_i(o_T)$ to consider.

Viterbi algorithm, termination



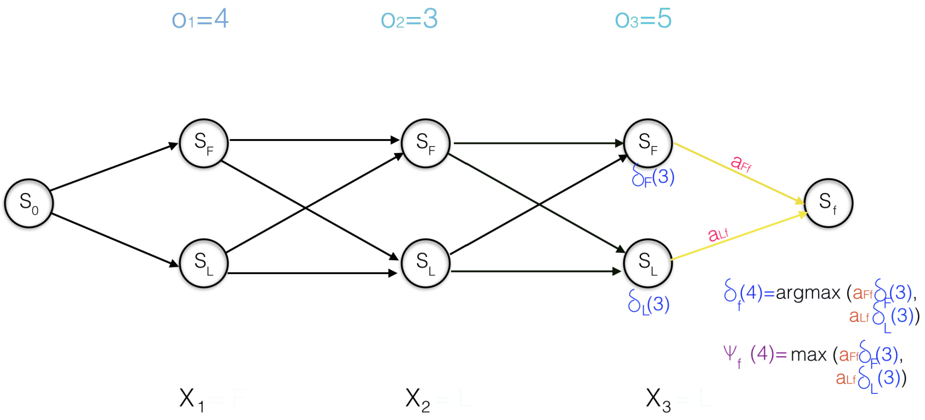
Viterbi algorithm, backtracing

- ψ_f is again calculated analogously to δ_f.

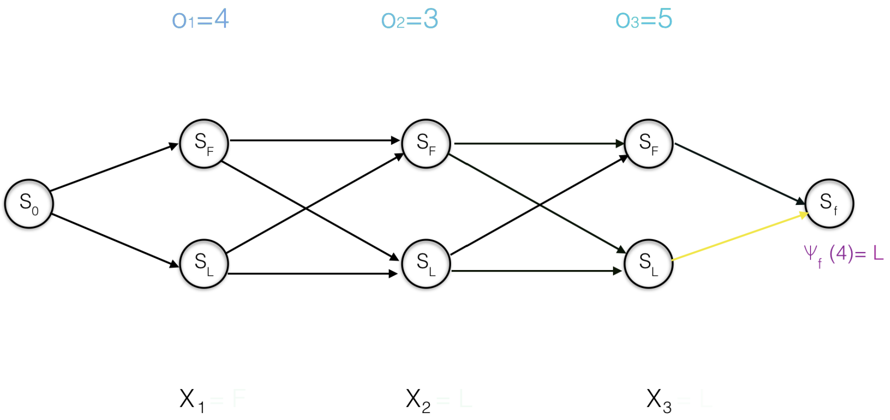
$$\psi_f(T+1) = \underset{1 \leq i \leq N}{\operatorname{argmax}} \, \delta_i(T) \cdot a_{if}$$

- It records X_T, the last state of the optimal state sequence.
- We will next go back to the cell concerned and look up **its** ψ to find the second-but-last state, and so on.

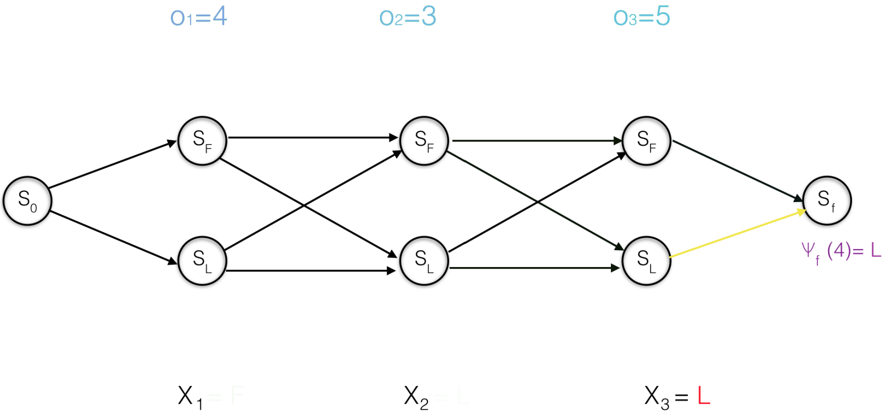
Viterbi algorithm, backtracing



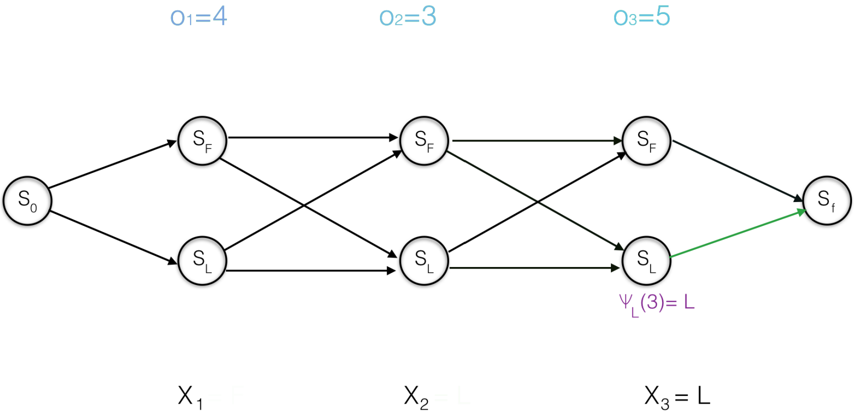
Viterbi algorithm, backtracing



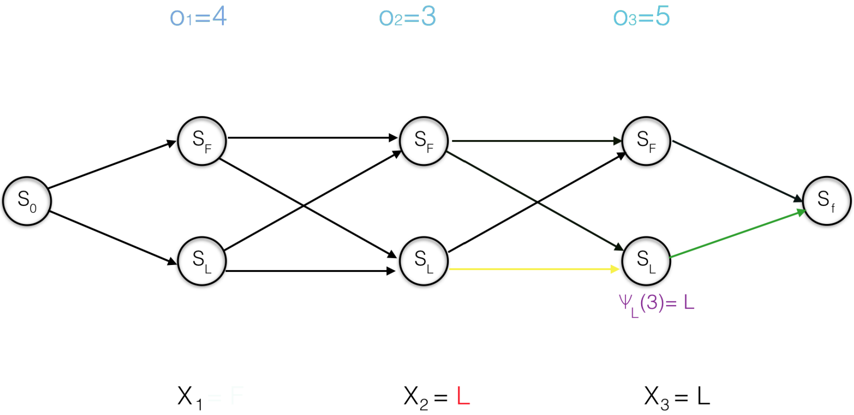
Viterbi algorithm, backtracing



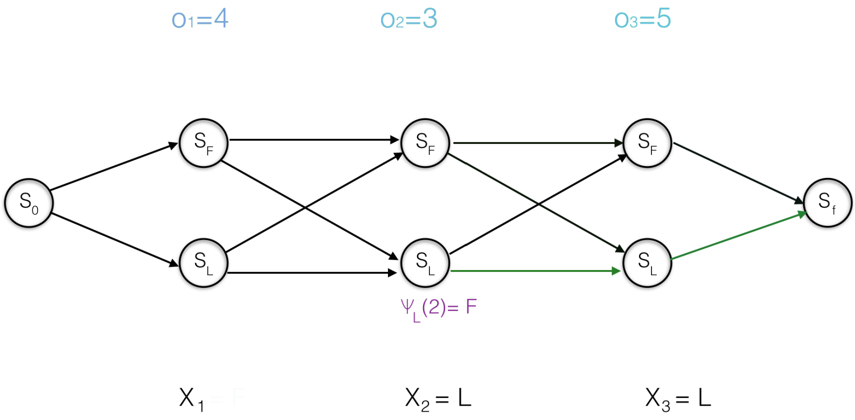
Viterbi algorithm, backtracing



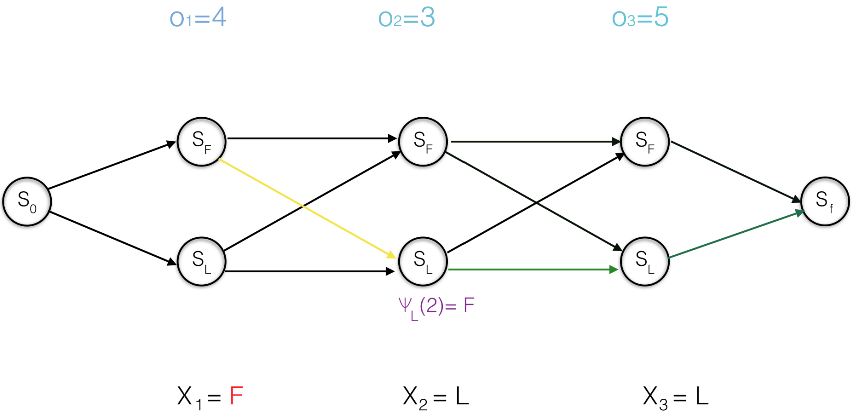
Viterbi algorithm, backtracing



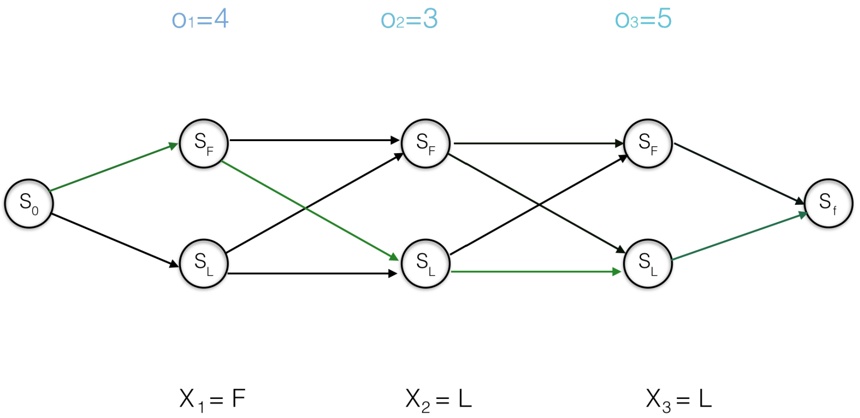
Viterbi algorithm, backtracing



Viterbi algorithm, backtracing



Viterbi algorithm, backtracing



Precision and Recall

- So far we have measured system success in accuracy or agreement in Kappa.
- But sometimes it's only one type of example that we find interesting.
- We don't want a summary measure that averages over interesting and non-interesting examples, as accuracy does.
- In those cases we use precision, recall and F-measure.
- These metrics are imported from the field of information retrieval, where the difference beween interesting and non-interesting examples is particularly high.

Precision and Recall

System says:

Truth is:		F	L	Total
	F	a	b	a+b
	L	c	d	c+d
	Total	a+c	b+d	a+b+c+d

- Precision of L: $P_L = \frac{d}{b+d}$
- Recall of L: $R_L = \frac{d}{c+d}$
- F-measure of L: $F_L = \frac{2 P_L R_L}{P_L + R_L}$
- Accuracy: $A = \frac{a+d}{a+b+c+d}$

Your task today

Task 8:

- Implement the Viterbi algorithm.
- Run it on the dice dataset and measure precision of L (P_L), recall of L (R_L) and F-measure of L (F_L).

Ticking today

- Task 7 – HMM Parameter Estimation

Literature

- Manning and Schutze (2000). Foundations of Statistical Natural Language Processing, MIT Press. Chapter 9.3.2.
  - We use a state-emission HMM, but this textbook uses an arc-emission HMM. There is therefore a slight difference in the algorithm as to in which step the initial and final $b_j(k_t)$ are multiplied in.
- Jurafsky and Martin, 2nd Edition, chapter 6.4