SPRING 2023

CS 378: INTRO TO SPEECH AND AUDIO PROCESSING

Hidden Markov Models 2

DAVID HARWATH
Assistant Professor, UTCS

The University of Texas at Austin
Department of Computer Science
College of Natural Sciences

The Backward Algorithm

The Backward Algorithm is very similar to the Forward Algorithm, except we will go backwards in time.

Define the *backward variable*:

$$\beta_t(i) = P(o_{t+1}, \dots, o_T \mid q_t = s_i, \lambda)$$

$\beta_T(i)$ is then the probability of no further observations after time T given $q_t = s_i$, which is always the case so we initialize $\beta_T(i) = 1$

The Backward Algorithm

Now the induction step to derive $\beta_t(i)$ in terms of $\beta_{t+1}(i)$:

$$\beta_t(i) = P(o_{t+1}, \dots, o_T \mid q_t = s_i, \lambda)$$

$$\beta_t(i) = \sum_{j=1}^{N} P\left(o_{t+1}, \dots, o_T, q_{t+1} = s_j \,\middle|\, q_t = s_i, \lambda\right)$$

$$\beta_t(i) = \sum_{j=1}^{N} P\left(o_{t+1}, \dots, o_T \middle| q_{t+1} = s_j, \lambda\right) \underbrace{P(q_{t+1} = s_j | q_t = s_i, \lambda)}_{a_{ij}}$$

$$\beta_t(i) = \sum_{j=1}^{N} \underbrace{P\left(o_{t+1} \middle| q_{t+1} = s_j, \lambda\right)}_{b_j(o_{t+1})} \underbrace{P\left(o_{t+2}, \dots, o_T \middle| q_{t+1} = s_j, \lambda\right)}_{\beta_{t+1}(j)} a_{ij}$$

The Backward Algorithm

Finally, we can compute the score $P(o_1, \dots, o_T \mid \lambda)$:

$$P(o_1, \dots, o_T|\lambda) = \sum_{i=1}^{N} P(o_1, \dots, o_T, q_1 = s_i|\lambda)$$

$$P(o_1, \dots, o_T|\lambda) = \sum_{i=1}^{N} P(o_2, \dots, o_T|q_1 = s_i, \lambda) P(o_1|q_1 = s_i, \lambda) P(q_1 = s_i|\lambda)$$

$$P(o_1, \dots, o_T|\lambda) = \sum_{i=1}^{N} \beta_1(i) b_i(o_1) \pi_i$$

The Backward Algorithm

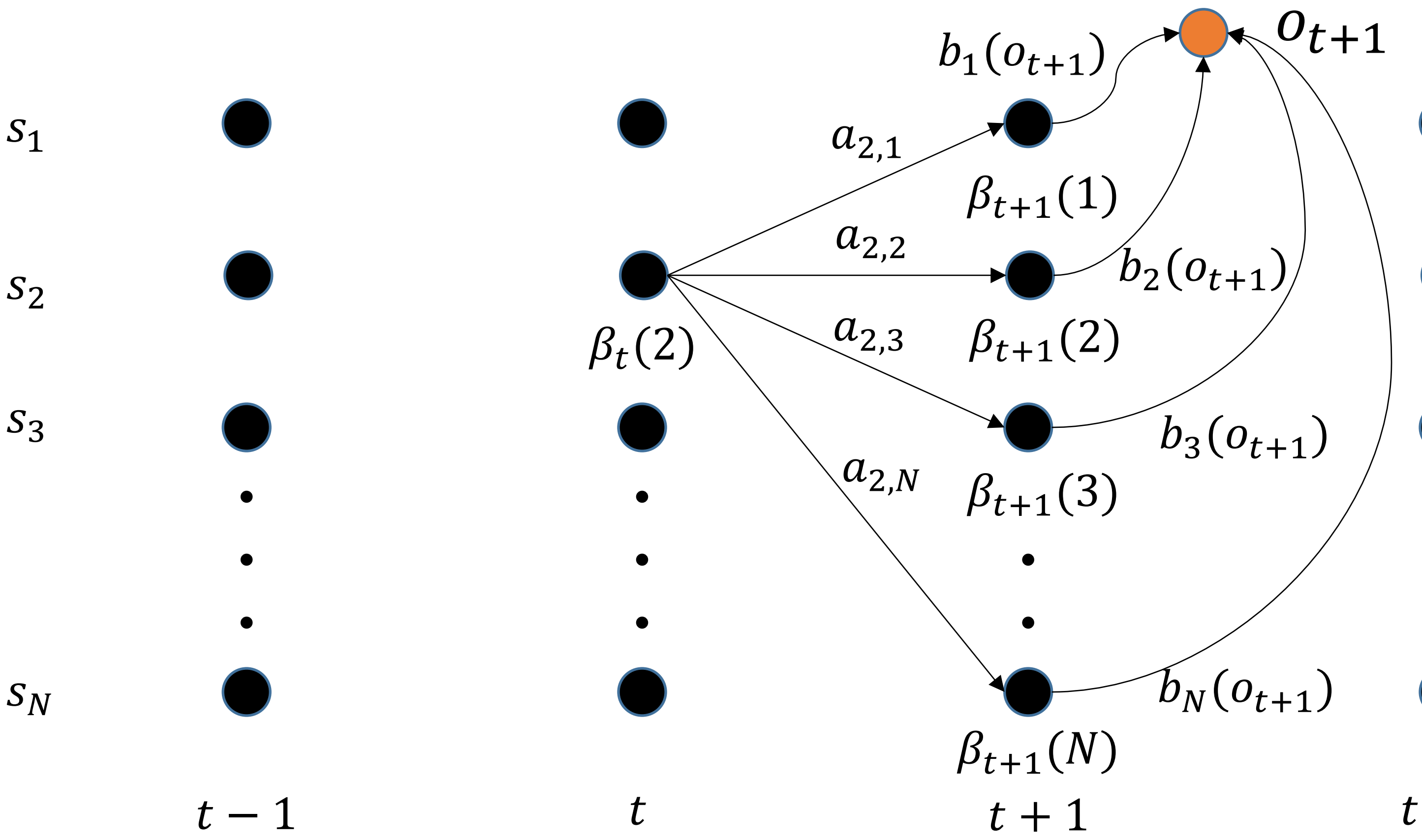
Putting it all together:

Initialization: $\beta_T(i) = 1$

Induction: $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$

Termination: $P(o_1, \ldots, o_T | \lambda) = \sum_{i=1}^{N} \pi_i b_i(o_1) \beta_1(i)$

Backward Illustration



Today’s agenda

- HMM motivation and intuitive introduction
- HMM mathematical formulation
- HMM algorithms
  - Scoring: Forward-Backward Algorithm
  - Decoding: Viterbi and Forward-Backward Algorithms
  - Training: Baum-Welch Algorithm
- HMMs for phone and word modeling in ASR

The HMM Decoding Problem

- We have an HMM model $\lambda = \{A, B, \Pi\}$
- We are given an observation sequence $O = \{o_1, o_2, \dots, o_T\}$
- Goal: find a hidden state sequence $Q = \{q_1, q_2, \dots, q_T\}$ that is optimal in some way
  - **What do we mean by optimal?**

Optimality of State Sequences

We can define optimality for hidden state sequences several different ways.

- If we want to know what state was **individually most likely at any point in time**, we need to marginalize over all possible state sequences.
  - We can use the Forward-Backward Algorithm.
- If we want to find the **single most likely state sequence**, we need to maximize the joint probability of all hidden state variables
  - We can use the Viterbi Algorithm.

The Forward-Backward Algorithm

- Recall that the Forward algorithm produces the forward variable

$$\alpha_t(i) = P(o_1, o_2, \dots o_t, q_t = s_i \mid \lambda)$$

- Also recall that the Backward algorithm produces the backward variable

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \dots o_T \mid q_t = s_i, \lambda)$$

- We can combine these variables to get the marginal probability of being in state s_i at time t

$$\gamma_t(i) = P(q_t = s_i \mid O, \lambda)$$

The Forward-Backward Algorithm

Notice what we get when we multiply the forward variable with the backward variable:

$$\alpha_t(i)\beta_t(i) = P(o_1, o_2, \dots o_t, q_t = s_i \mid \lambda) P(o_{t+1}, o_{t+2}, \dots o_T \mid q_t = s_i, \lambda)$$

$$\alpha_t(i)\beta_t(i) = P(o_1, o_2, \dots o_T, q_t = s_i \mid \lambda) = P(O, q_t = s_i \mid \lambda)$$

The Forward-Backward Algorithm

$$\gamma_t(i) = P(q_t = s_i \mid O, \lambda) = \frac{P(O, q_t = s_i \mid \lambda)}{P(O \mid \lambda)}$$

$$\gamma_t(i) = \frac{P(O, q_t = s_i \mid \lambda)}{\sum_{j=1}^{N} P\big(O, q_t = s_j \mid \lambda\big)}$$

$$\boxed{\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}}$$

The Forward-Backward Algorithm

Given that we now have $\gamma_t(i) = P(q_t = s_i \mid O, \lambda)$, we can compute the most likely state at any time t as

$$q_t^* = \operatorname{argmax}_i \gamma_t(i)$$

Inferring an optimal state sequence in this fashion will result in maximizing the *total number of correct states*.

However, this optimal state sequence may not be nonsensical, because it does not need to obey the state transition constraints

Break here for today, to be continued on Thursday 9/30

The Viterbi Algorithm

An alternative optimality criterion is to find the *single* state sequence with the maximum joint probability:

$$Q^* = \text{argmax}_{q_1,\ldots,q_T} P(q_1, \ldots, q_T \mid O, \lambda)$$

An efficient dynamic programming solution for this problem also exists and is known as the *Viterbi Algorithm.*

The Viterbi Algorithm

Notice that because O is assumed to be constant, we have

$$Q^* = \operatorname{argmax}_Q \, P(Q \mid O, \lambda) = \operatorname{argmax}_Q P(Q, O \mid \lambda)$$

Define:

$$\delta_t(i) = \max_{q_1,\ldots,q_{t-1}} P(q_1, q_2, \ldots, q_t = s_i, o_1, \ldots, o_t \mid \lambda)$$

Think of $\delta_t(i)$ as the best way to get to state s_i at time t given all of the observations that were seen up to time t

The Viterbi Algorithm

$$\delta_t(i) = \max_{q_1,\ldots,q_{t-1}} P(q_1, q_2, \ldots, q_t = s_i, o_1, \ldots, o_t \mid \lambda)$$

If we know the best way to get to s_j at time t for each s_j (N different paths), then the best way to get to reach s_i at time $t+1$ must have followed one of those N paths.

By induction, we can obtain:

$$\delta_{t+1}(i) = \left[\max_j \delta_t(j) a_{ji}\right] b_i(o_{t+1})$$

To recover the Viterbi path, we can use a backtrace matrix:

$$\Psi_t(i) = \operatorname{argmax}_j \delta_t(j) a_{ji}$$

The Viterbi Algorithm (Formally)

1. Initialization: $\delta_1(i) = \pi_i b_i(o_1), \quad \Psi_1(i) = 0 \qquad \text{for } 1 \le i \le N$

2. Induction: $\delta_t(i) = \left[\max_j \delta_{t-1}(j) a_{ji}\right] b_i(o_t)$

   $\Psi_t(i) = \operatorname{argmax}_j \delta_{t-1}(j) a_{ji}$

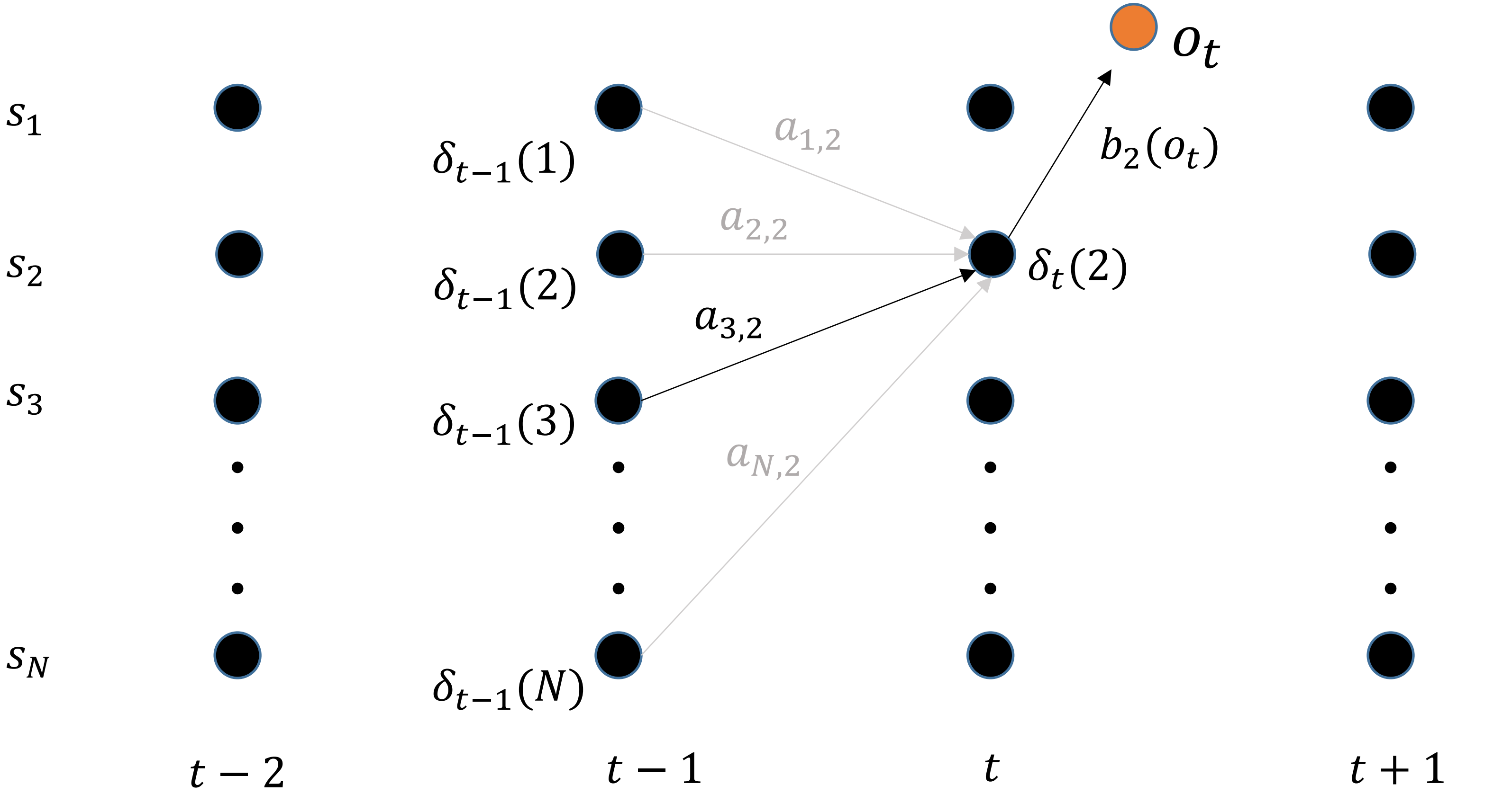
   $\text{for } 1 \le j \le N, \; 2 \le t \le T$

3. Termination: Score of best path: $P^* = \max_i \delta_T(i)$

   Best final state: $q_T^* = \operatorname{argmax}_i \delta_T(i)$

4. Backtrace: $q_t^* = \Psi_{t+1}(q_{t+1}^*) \qquad \text{for } 1 \le t \le T-1$

Viterbi Illustration



Today’s agenda

- HMM motivation and intuitive introduction
- HMM mathematical formulation
- HMM algorithms
  - Scoring: Forward-Backward Algorithm
  - Decoding: Viterbi and Forward-Backward Algorithms
  - Training: Baum-Welch Algorithm
- HMMs for phone and word modeling in ASR

The Baum-Welch Algorithm

- We have an HMM model $\lambda = \{A, B, \Pi\}$, **but now instead of treating the parameters as fixed, we want to adjust them to "fit" our data.**

- Again assume we are given $O = \{o_1, o_2, \dots, o_T\}$

- We will fit the HMM parameters using the *maximum likelihood* (ML) criterion:

$$\text{Maximize}_{\lambda}\ P(O|\lambda)$$

The Baum-Welch Algorithm

There is no known algorithm to $\text{Maximize}_\lambda\ P(O|\lambda)$ in general.

However, if we *knew the underlying hidden state sequence* Q, the ML solution for $\text{Maximize}_\lambda\ P(O|Q,\lambda)$ is extremely easy:

$$\pi_i^* = \frac{\#\text{ of times we were in state } i \text{ at time } t = 1}{\#\text{ of times in we were in any state at time } t = 1}$$

$$a_{ij}^* = \frac{\#\text{ of times we were in state } i \text{ and then transitioned to state } j}{\#\text{ of times we made a transition out of state } i}$$

$$b_i^*(o) = \frac{\#\text{ of times we were in state } i \text{ and output the symbol } o}{\#\text{ of times we made an emission from state } i}$$

The Baum-Welch Algorithm

The insight that "everything would be easy if only we knew Q" gives rise to an iterative algorithm that we can use to *estimate* the optimal value of λ:

1. Guess an initial value of λ.
2. Use the current value of λ to compute $\gamma_t(i) = P(q_t = s_i \mid O, \lambda)$ using the Forward-Backward algorithm. This gives us $\mathrm{E}[Q]$, the *expected value* of Q.
3. Use $\mathrm{E}[Q]$ instead of Q to update the guess of λ using the equations on the previous slide.
4. Go back to Step 2 and repeat until λ converges.

The Baum-Welch Algorithm

- This procedure is known as the Baum-Welch Algorithm, and is a type of Expectation-Maximization (EM) Algorithm
- EM algorithms in general iterate between an E-step (Expectation), where we estimate the value of some hidden variable we wish we had, and an M-step (Maximization) where we use that estimate to maximize the data likelihood as a function of the model parameters.
- EM Algorithms generally cannot find a *globally optimal* solution, but they are guaranteed to converge to a *locally optimal* solution.

Baum-Welch E-Step (Formally)

Compute $\gamma_t(i)$ using the Forward-Backward Algorithm:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

Also compute $\tau_t(i,j) = P(q_t = s_i, q_{t+1} = s_j | O, \lambda)$:

$$\tau_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_T(i)}$$

Baum-Welch M-Step (Formally)

$$\pi_i^* = \frac{\text{E[\# times in state } i \text{ at time } t = 1]}{\text{E[\# times in any state at time } t = 1]} = \gamma_1(i)$$

$$a_{ij}^* = \frac{\text{E[\# times in state } i \text{ and transitioned to state } j]}{\text{E[\# times made a transition from state } i \text{]}} = \frac{\sum_{t=1}^{T-1} \tau_t\,(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$b_i^*(o) = \frac{\text{E[\# times in state } i \text{ and output symbol } o]}{\text{E[\# times in state } i]} = \frac{\sum_{t=1}^{T} \gamma_t(i)\mathbf{1}(o_t = o)}{\sum_{t=1}^{T} \gamma_t(i)}$$