Lecture 14: Log Viterbi and Scaled Forward-Backward

Mark Hasegawa-Johnson
All content CC-SA 4.0 unless otherwise specified.

ECE 417: Multimedia Signal Processing, Fall 2020

1. Review: Hidden Markov Models

2. Recognition: The Scaled Forward Algorithm

3. Segmentation: The Viterbi Algorithm

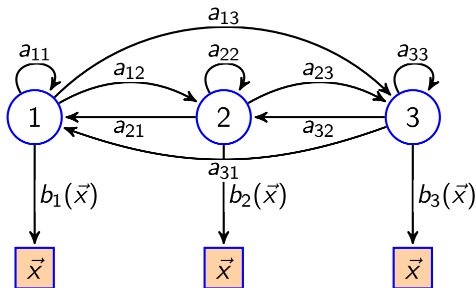
4. Training: The Scaled Backward Algorithm

5. Summary

Outline

1. Review: Hidden Markov Models
2. Recognition: The Scaled Forward Algorithm
3. Segmentation: The Viterbi Algorithm
4. Training: The Scaled Backward Algorithm
5. Summary

The Three Problems for an HMM



1. **Recognition:** Given two different HMMs, Λ_1 and Λ_2, and an observation sequence X. Which HMM was more likely to have produced X? In other words, $p(X|\Lambda_1) > p(X|\Lambda_2)$?
2. **Segmentation:** What is $p(Q|X,\Lambda)$?
3. **Training:** Given an initial HMM Λ, and an observation sequence X, can we find Λ' such that $p(X|\Lambda') > p(X|\Lambda)$?

Outline

1 Review: Hidden Markov Models

2 Recognition: The Scaled Forward Algorithm

3 Segmentation: The Viterbi Algorithm

4 Training: The Scaled Backward Algorithm

5 Summary

The Forward Algorithm

Definition: $\alpha_t(i) \equiv p(\vec{x}_1, \ldots, \vec{x}_t, q_t = i|\Lambda)$. Computation:

1. **Initialize:**

$$\alpha_1(i) = \pi_i b_i(\vec{x}_1), \quad 1 \le i \le N$$

2. **Iterate:**

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

3. **Terminate:**

$$p(X|\Lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Numerical Issues

The forward algorithm is susceptible to massive floating-point underflow problems. Consider this equation:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\vec{x}_t)$$

$$= \sum_{q_1=1}^{N} \cdots \sum_{q_{t-1}=1}^{N} \pi_{q_1} b_{q_1}(\vec{x}_1) \cdots a_{q_{t-1}q_t} b_{q_t}(\vec{x}_t)$$

First, suppose that $b_q(x)$ is discrete, with $k \in \{1, \ldots, K\}$. Suppose $K \approx 1000$ and $T \approx 100$, in that case, each $\alpha_t(j)$ is:

- The sum of N^T different terms, each of which is
- the product of T factors, each of which is
- the product of two probabilities: $a_{ij} \sim \frac{1}{N}$ times $b_j(x) \sim \frac{1}{K}$, so

$$\alpha_T(j) \approx N^T \left(\frac{1}{NK}\right)^T \approx \frac{1}{K^T} \approx 10^{-300}$$

Numerical Issues

Softmax observation probabilities are scaled similarly to discrete pmfs ($b_j(\vec{x}) \sim \frac{1}{1000}$), but Gaussians are much worse. Suppose that $b_j(\vec{x})$ is Gaussian:

$$b_j(\vec{x}) = \frac{1}{\prod_{d=1}^{D} \sqrt{2\pi\sigma_{jd}^2}} e^{-\frac{1}{2}\sum_{d=1}^{D} \frac{(x_d - \mu_{jd})^2}{\sigma_{jd}^2}}$$

Suppose that $D \approx 30$.

- On average, $E\left[\frac{(x_d-\mu_{jd})^2}{\sigma_{jd}^2}\right] = 1$,
- so on average, $b_j(\vec{x}) = \frac{1}{(2\pi)^{15}} e^{-15} = 3 \times 10^{-19}$.

How to Solve Numerical Issues

- Single-precision floating point can represent numbers as small as 2^{-127}.
- One time step of the forward algorithm can be computed with no problem, but 100 time steps is impossible.
- Solution: re-normalize $\alpha_t(j)$ to $\hat{\alpha}_t(j)$ after each time step, so that $\sum_j \hat{\alpha}_t(j) = 1$.

The Scaled Forward Algorithm

1. **Initialize:**

$$\hat{\alpha}_1(i) = \frac{\pi_i b_i(\vec{x}_1)}{\sum_{\ell=1}^{N} \pi_\ell b_\ell(\vec{x}_1)}$$

2. **Iterate:**

$$\hat{\alpha}_t(j) = \frac{\sum_{i=1}^{N} \hat{\alpha}_{t-1}(i) a_{ij} b_j(\vec{x}_t)}{\sum_{\ell=1}^{N} \sum_{i=1}^{N} \hat{\alpha}_{t-1}(i) a_{i\ell} b_\ell(\vec{x}_t)}$$

3. **Terminate:**

$$p(X|\Lambda) = ????$$

What exactly is alpha-hat?

Let's look at this in more detail. $\alpha_t(j)$ is defined to be $p(\vec{x}_1, \ldots, \vec{x}_t, q_t = j|\Lambda)$. Let's define a "scaling term," G_t, equal to the denominator in the scaled forward algorithm. So, for example, at time $t = 1$ we have:

$$G_1 = \sum_{\ell=1}^{N} \alpha_1(\ell) = \sum_{\ell=1}^{N} p(\vec{x}_1, q_1 = \ell|\Lambda) = p(\vec{x}_1|\Lambda)$$

and therefore

$$\hat{\alpha}_1(i) = \frac{\alpha_1(i)}{G_1} = \frac{p(\vec{x}_1, q_1 = i|\Lambda)}{p(\vec{x}_1|\Lambda)} = p(q_1 = i|\vec{x}_1, \Lambda)$$

What exactly is alpha-hat?

At time t, we need a new intermediate variable. Let's call it $\tilde{\alpha}_t(j)$:

$$\begin{aligned}
\tilde{\alpha}_t(j) &= \sum_{i=1}^{N} \hat{\alpha}_{t-1}(i) a_{ij} b_j(\vec{x}_t) \\
&= \sum_{i=1}^{N} p(q_{t-1}=i|\vec{x}_1,\ldots,\vec{x}_{t-1},\Lambda) p(q_t=j|q_{t-1}=i) p(\vec{x}_t|q_t=j) \\
&= p(q_t=j,\vec{x}_t|\vec{x}_1,\ldots,\vec{x}_{t-1},\Lambda) \\
G_t &= \sum_{\ell=1}^{N} \tilde{\alpha}_t(\ell) = p(\vec{x}_t|\vec{x}_1,\ldots,\vec{x}_{t-1},\Lambda)
\end{aligned}$$

$$\hat{\alpha}_t(j) = \frac{\tilde{\alpha}_t(j)}{G_t} = \frac{p(\vec{x}_t,q_t=j|\vec{x}_1,\ldots,\vec{x}_{t-1},\Lambda)}{p(\vec{x}_t|\vec{x}_1,\ldots,\vec{x}_{t-1},\Lambda)} = p(q_t=j|\vec{x}_1,\ldots,\vec{x}_t,\Lambda)$$

Scaled Forward Algorithm: The Variables

So we have not just one, but two new variables:

1. The scaled forward probability:

$$\hat{\alpha}_t(j) = p(q_t = j | \vec{x}_1, \ldots, \vec{x}_t, \Lambda)$$

2. The scaling factor:

$$G_t = p(\vec{x}_t | \vec{x}_1, \ldots, \vec{x}_{t-1}, \Lambda)$$

The Solution

The second of those variables is interesting because we want $p(X|\Lambda)$, which we can now get from the G_ts—we no longer actually need the αs for this!

$$p(X|\Lambda) = p(\vec{x}_1|\Lambda)p(\vec{x}_2|\vec{x}_1,\Lambda)p(\vec{x}_3|\vec{x}_1,\vec{x}_2,\Lambda)\cdots = \prod_{t=1}^{T} G_t$$

But that's still not useful, because if each $G_t \sim 10^{-19}$, then multiplying them all together will result in floating point underflow. So instead, it is better to compute

$$\ln p(X|\Lambda) = \sum_{t=1}^{T} \ln G_t$$

The Scaled Forward Algorithm

1. **Initialize:**

$$\hat{\alpha}_1(i) = \frac{1}{G_1}\pi_i b_i(\vec{x}_1)$$

2. **Iterate:**

$$\hat{\alpha}_t(j) = \frac{1}{G_t}\sum_{i=1}^{N}\hat{\alpha}_{t-1}(i)a_{ij}b_j(\vec{x}_t)$$

3. **Terminate:**

$$\ln p(X|\Lambda) = \sum_{t=1}^{T}\ln G_t$$

Outline

1. Review: Hidden Markov Models
2. Recognition: The Scaled Forward Algorithm
3. Segmentation: The Viterbi Algorithm
4. Training: The Scaled Backward Algorithm
5. Summary

What About State Sequences?

- Remember when we first derived $\gamma_t(i)$, I pointed out a problem: $\gamma_t(i)$ only tells us about one frame at a time! It doesn't tell us anything about the probability of a sequence of states, covering a sequence of frames.
- Today, let's find a complete solution. Let's find the most likely state sequence covering the entire utterance:

$$Q^* = \underset{Q}{\operatorname{argmax}}\, p(Q, X|\Lambda)$$

The Max-Probability State Sequence

The problem of finding the max-probability state sequence is just as hard as the problem of finding $p(X|\Lambda)$, for exactly the same reason:

$$\max_Q p(Q, X|\Lambda) = \max_{q_T=1}^{N} \cdots \max_{q_1=1}^{N} p(Q, X|\Lambda)$$

which has complexity $\mathcal{O}\left\{N^T\right\}$.

The Viterbi Algorithm

Remember that we solved the recognition probability using a divide-and-conquer kind of dynamic programming algorithm, with the intermediate variable

$$\begin{aligned}\alpha_t(j) &\equiv p(\vec{x}_1,\ldots,\vec{x}_t,q_t=j|\Lambda)\\ &= \sum_{q_{t-1}}\cdots\sum_{q_1} p(\vec{x}_1,\ldots,\vec{x}_t,q_1,\ldots,q_{t-1},q_t=j|\Lambda)\end{aligned}$$

The segmentation problem is solved using a similar dynamic programming algorithm called the Viterbi algorithm, with a slightly different intermediate variable:

$$\delta_t(j) \equiv \max_{q_{t-1}}\cdots\max_{q_1} p(\vec{x}_1,\ldots,\vec{x}_t,q_1,\ldots,q_{t-1},q_t=j|\Lambda)$$

The Viterbi Algorithm

Keeping in mind the definition $\delta_t(j) \equiv \max_{q_{t-1}} \cdots \max_{q_1} p(\vec{x}_1, \ldots, \vec{x}_t, q_1, \ldots, q_{t-1}, q_t = j|\Lambda)$, we can devise an efficient algorithm to compute it:

1. **Initialize:**
$$\delta_1(i) = \pi_i b_i(\vec{x}_1)$$
2. **Iterate:**
$$\delta_t(j) = \max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{x}_t)$$
3. **Terminate:** The maximum-probability final state is $q_T^* = \text{argmax}_{j=1}^{N} \delta_T(j)$. But what are the best states at all of the previous time steps?

Backtracing

We can find the optimum states at all times, q_t^*, by keeping a **backpointer** $\psi_t(j)$ from every time step. The backpointer points to the state at time $t-1$ that is most likely to have preceded state j at time t:

$$
\begin{aligned}
\psi_t(j) &= \underset{i}{\operatorname{argmax}} \cdots \max_{q_1} p(\vec{x}_1, \ldots, \vec{x}_t, q_1, \ldots, q_{t-1} = i, q_t = j|\Lambda) \\
&= \underset{i=1}{\overset{N}{\operatorname{argmax}}}\, \delta_{t-1}(i) a_{ij} b_j(\vec{x}_t)
\end{aligned}
$$

Backtracing

If we have the backpointers available, then we can get the entire maximum-probability state sequence by **backtracing** after we terminate:

- **Terminate:** Once we get to time $t = T$, we choose the most probable final state.
  - If we already know which state we want to end in, then we just choose that state as q_T^*.
  - If we don't already know, then we choose $q_T^* = \text{argmax}_j \, \delta_T(j)$
- **Backtrace:** Having found the final state, we work backward, by way of the **backpointers**, $\psi_t(j)$:

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right), \quad T-1 \geq t \geq 1$$

The Viterbi Algorithm

1. **Initialize:**

$$\delta_1(i) = \pi_i b_i(\vec{x}_1)$$

2. **Iterate:**

$$\delta_t(j) = \max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{x}_t)$$

$$\psi_t(j) = \operatorname{argmax}_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{x}_t)$$

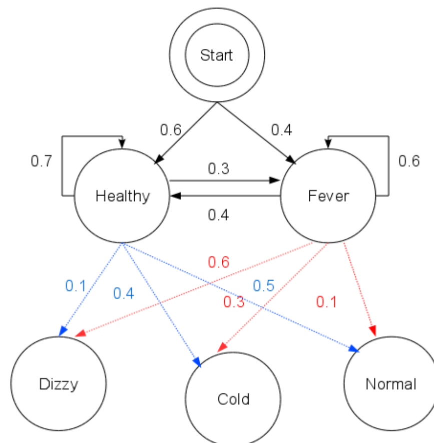
3. **Terminate:**

$$q_T^* = \operatorname{argmax}_{j=1}^{N} \delta_T(j)$$

4. **Backtrace:**

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right)$$

Example



An example of HMM, GFDL by Reelsun, 2012,

`https://commons.wikimedia.org/wiki/File:An_example_of_HMM.png`

Example

Viterbi animated demo, GFDL by Reelsun, 2012,

https://commons.wikimedia.org/wiki/File:Viterbi_animated_demo.gif

Numerical Problems

Viterbi algorithm has the same floating-point underflow problems as the Forward algorithm. But this time, there is an easy solution, because the log of the max is equal to the max of the log:

$$\ln \delta_t(j) = \ln \left(\max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{x}_t) \right)$$

$$= \max_{i=1}^{N} \left(\ln \delta_{t-1}(i) + \ln a_{ij} + \ln b_j(\vec{x}_t) \right)$$

The Log-Viterbi Algorithm

1. **Initialize:**

$$\ln \delta_1(i) = \ln \pi_i + \ln b_i(\vec{x}_1)$$

2. **Iterate:**

$$\ln \delta_t(j) = \max_{i=1}^{N} \left(\ln \delta_{t-1}(i) + \ln a_{ij} + \ln b_j(\vec{x}_t) \right)$$

$$\psi_t(j) = \operatorname*{argmax}_{i=1}^{N} \left(\ln \delta_{t-1}(i) + \ln a_{ij} + \ln b_j(\vec{x}_t) \right)$$

3. **Terminate:** Choose the known final state q_T^*.
4. **Backtrace:**

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right)$$

Outline

1. Review: Hidden Markov Models
2. Recognition: The Scaled Forward Algorithm
3. Segmentation: The Viterbi Algorithm
4. Training: The Scaled Backward Algorithm
5. Summary

Baum-Welch Re-estimation

Unfortunately, the Viterbi algorithm doesn't solve the problem of training. We still need:

$$\begin{aligned}\xi_t(i,j) &\equiv p(q_t = i, q_{t+1} = j | X, \Lambda) \\ &= \frac{\alpha_t(i) a_{ij} b_j(\vec{x}_{t+1}) \beta_{t+1}(j)}{\sum_{k=1}^N \sum_{\ell=1}^N \alpha_t(k) a_{k\ell} b_\ell(\vec{x}_{t+1}) \beta_{t+1}(\ell)}\end{aligned}$$

We have a numerically-safe algorithm for finding $\hat{\alpha}_t(j)$. Can we use that, somehow?

Scaled Baum-Welch Re-estimation

- We already have

$$\hat{\alpha}_t(i) = \frac{\alpha_t(i)}{\prod_{\tau=1}^{t} G_\tau}$$

- Suppose we also define

$$\hat{\beta}_{t+1}(j) = \frac{\beta_{t+1}(j)}{\prod_{\tau=(t+1)}^{T} G_\tau}$$

- Then we get

$$\frac{\hat{\alpha}_t(i) a_{ij} b_j(\vec{x}_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{k=1}^{N} \sum_{\ell=1}^{N} \hat{\alpha}_t(k) a_{k\ell} b_\ell(\vec{x}_{t+1}) \hat{\beta}_{t+1}(\ell)}$$

$$= \frac{\frac{1}{\prod_{\tau=1}^{T} G_\tau} \alpha_t(i) a_{ij} b_j(\vec{x}_{t+1}) \beta_{t+1}(j)}{\frac{1}{\prod_{\tau=1}^{T} G_\tau} \sum_{k=1}^{N} \sum_{\ell=1}^{N} \alpha_t(k) a_{k\ell} b_\ell(\vec{x}_{t+1}) \beta_{t+1}(\ell)}$$

$$= \xi_t(i,j)$$

The Scaled Backward Algorithm

1. **Initialize:**

$$\hat{\beta}_T(i) = 1, \quad 1 \le i \le N$$

2. **Iterate:**

$$\hat{\beta}_t(i) = \frac{1}{G_t} \sum_{j=1}^{N} a_{ij} b_j(\vec{x}_{t+1}) \hat{\beta}_{t+1}(j)$$

The scaling constant, G_t, can be the same for forward algorithm, but doesn't have to be. I get better results using other normalizing constants, for example, $\sum_i \hat{\beta}_t(i) = 1$ for $t < T$.

Outline

1 Review: Hidden Markov Models

2 Recognition: The Scaled Forward Algorithm

3 Segmentation: The Viterbi Algorithm

4 Training: The Scaled Backward Algorithm

5 Summary

The Scaled Forward Algorithm

1. **Initialize:**

$$\hat{\alpha}_1(i) = \frac{1}{G_1}\pi_i b_i(\vec{x}_1)$$

2. **Iterate:**

$$\hat{\alpha}_t(j) = \frac{1}{G_t}\sum_{i=1}^{N}\hat{\alpha}_{t-1}(i)a_{ij}b_j(\vec{x}_t)$$

3. **Terminate:**

$$\ln p(X|\Lambda) = \sum_{t=1}^{T}\ln G_t$$

The Log-Viterbi Algorithm

1. **Initialize:**
$$\ln \delta_1(i) = \ln \pi_i + \ln b_i(\vec{x}_1)$$
2. **Iterate:**
$$\ln \delta_t(j) = \max_{i=1}^{N} \left(\ln \delta_{t-1}(i) + \ln a_{ij} + \ln b_j(\vec{x}_t)\right)$$
$$\psi_t(j) = \operatorname{argmax}_{i=1}^{N} \left(\ln \delta_{t-1}(i) + \ln a_{ij} + \ln b_j(\vec{x}_t)\right)$$
3. **Terminate:** Choose the known final state q_T^*.
4. **Backtrace:**
$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right)$$

Scaled Baum-Welch Re-estimation

$$\xi_t(i,j) \equiv p(q_t = i, q_{t+1} = j | X, \Lambda)$$
$$= \frac{\hat{\alpha}_t(i) a_{ij} b_j(\vec{x}_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{k=1}^{N} \sum_{\ell=1}^{N} \hat{\alpha}_t(k) a_{k\ell} b_\ell(\vec{x}_{t+1}) \hat{\beta}_{t+1}(\ell)}$$