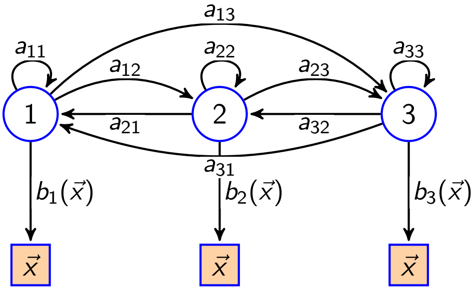
Lecture 16: Numerical Issues in Training HMMs

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing, Fall 2021

Outline

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 $p(X|\Lambda') > p(X|\Lambda)$?

Recognition: The Forward Algorithm

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

1. **Initialize:**

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

2. **Iterate:**

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

3. **Terminate:**

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

Segmentation: The Backward Algorithm

Definition: $\beta_t(i) \equiv p(\vec{x}_{t+1},\ldots,\vec{x}_T|q_t = i,\Lambda)$. Computation:

1. **Initialize:**
$$\beta_T(i) = 1, \quad 1 \le i \le N$$
2. **Iterate:**
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\vec{x}_{t+1})\beta_{t+1}(j), \quad 1 \le i \le N,\ 1 \le t \le T-1$$
3. **Terminate:**
$$p(X|\Lambda) = \sum_{i=1}^{N} \pi_i b_i(\vec{x}_1)\beta_1(i)$$

Segmentation: State and Segment Posteriors

1. **The State Posterior:**

$$\gamma_t(i) = p(q_t = i|X,\Lambda) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{k=1}^N \alpha_t(k)\beta_t(k)}$$

2. **The Segment Posterior:**

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

Training: The Baum-Welch Algorithm

1. **Transition Probabilities:**

$$a'_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{j=1}^{N} \sum_{t=1}^{T-1} \xi_t(i,j)}$$

2. **Gaussian Observation PDFs:**

$$\vec{\mu}'_i = \frac{\sum_{t=1}^{T} \gamma_t(i)\vec{x}_t}{\sum_{t=1}^{T} \gamma_t(i)}$$

$$\Sigma'_i = \frac{\sum_{t=1}^{T} \gamma_t(i)(\vec{x}_t - \vec{\mu}_i)(\vec{x}_t - \vec{\mu}_i)^T}{\sum_{t=1}^{T} \gamma_t(i)}$$

Outline

Numerical Issues in the Training of an HMM

- **Flooring the observation pdf**: $e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T\Sigma^{-1}(\vec{x}-\vec{\mu})}$ can be very small.
- **Scaled forward-backward algorithm:** a_{ij}^T can be very small.
- **Zero denominators:** Sometimes $\sum_i \alpha_t(i)\beta_t(i)$ is zero.
- **Tikhonov regularization:** Re-estimation formulae can result in $|\Sigma_i| = 0$.

Outline

Flooring the observation pdf: Why is it necessary?

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 $D \approx 30$. Then:

Average distance from the mean $\frac{x_d-\mu_{jd}}{\sigma_{jd}}$	Observation pdf $\frac{1}{(2\pi)^{15}} e^{-\frac{1}{2}\prod_{d=1}^{D}\left(\frac{x_d-\mu_{jd}}{\sigma_{jd}}\right)^2}$
1	$\frac{1}{(2\pi)^{15}} e^{-15} \approx 10^{-19}$
3	$\frac{1}{(2\pi)^{15}} e^{-135} \approx 10^{-71}$
5	$\frac{1}{(2\pi)^{15}} e^{-375} \approx 10^{-175}$
7	$\frac{1}{(2\pi)^{15}} e^{-735} \approx 10^{-331}$

Why is that a problem?

- IEEE single-precision floating point: smallest number is 10^{-38}.
- IEEE double-precision floating point (numpy): smallest number is 10^{-324}.

Why is that a problem?

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

- If some (but not all) $a_{ij} = 0$, and some (but not all) $b_j(\vec{x}) = 0$, then it's possible that all $a_{ij} b_j(\vec{x}) = 0$.
- In that case, it's possible to get $\alpha_t(j) = 0$ for all j.
- In that case, recognition crashes.

One possible solution: Floor the observation pdf

There are many possible solutions, including scaling solutions similar to the scaled forward that I'm about to introduce. But for the MP, I recommend a simple solution: floor the observation pdf. Thus:

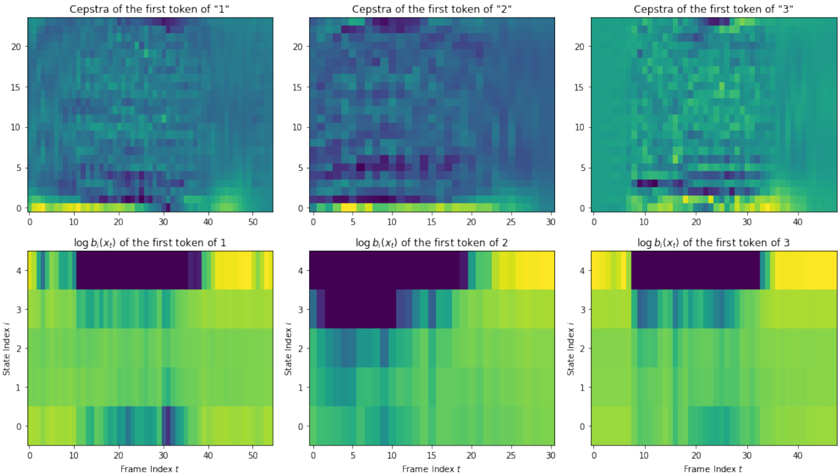
$$b_j(\vec{x}) = \max\left(\text{floor}, \mathcal{N}(\vec{x}|\vec{\mu}_j, \Sigma_j)\right)$$

The floor needs to be much larger than 10^{-324}, but much smaller than "good" values of the Gaussian (values observed for non-outlier spectra). In practice, a good choice seems to be

$$\text{floor} = 10^{-100}$$

Result example

Here is $\ln b_i(\vec{x}_t)$, plotted as a function of i and t, for the words "one," "two," and "three."



Outline

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

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

The Solution: Scaling

The solution is to just re-scale $\alpha_t(j)$ at each time step, so it never gets really small:

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

Now the problem is. . . if $\alpha_t(j)$ has been re-scaled, how do we perform recognition? Remember we used to have $p(X|\Lambda) = \sum_i \alpha_t(i)$. How can we get $p(X|\Lambda)$ now?

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 three new variables:

1. The intermediate forward probability:

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

2. The scaling factor:

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

3. The scaled forward probability:

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

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:**

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

$$g_t = \sum_{j=1}^{N} \tilde{\alpha}_t(j)$$

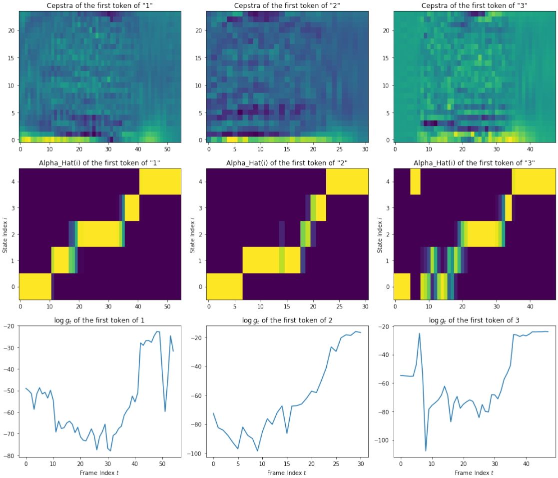
$$\hat{\alpha}_t(j) = \frac{1}{g_t}\tilde{\alpha}_t(j)$$

3. **Terminate:**

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

Result example

Here are $\hat{\alpha}_t(i)$ and $\ln g_t$, plotted as a function of i and t, for the words "one," "two," and "three."



The Scaled Backward Algorithm

This can also be done for the backward algorithm:

1. **Initialize:**

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

2. **Iterate:**

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

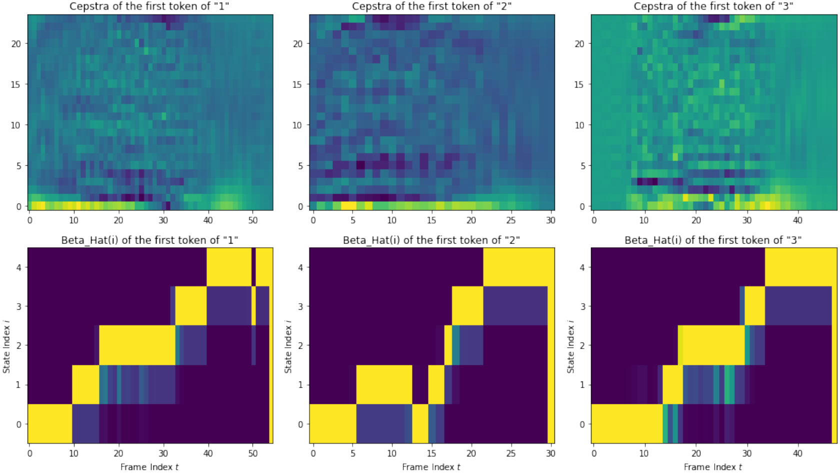
$$\hat{\beta}_t(i) = \frac{1}{c_t} \tilde{\beta}_t(i)$$

Rabiner uses $c_t = g_t$, but I recommend instead that you use

$$c_t = \max_i \tilde{\beta}_t(i)$$

Result example

Here is $\hat{\beta}_t(i)$, plotted as a function of i and t, for the words "one," "two," and "three."



Scaled Baum-Welch Re-estimation

So now we have:

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

$$\hat{\beta}_t(i) = \frac{1}{c_t}\tilde{\beta}_t(i) = \frac{1}{\prod_{\tau=t}^{T} g_\tau}\beta_t(i)$$

During re-estimation, we need to find $\gamma_t(i)$ and $\xi_t(i,j)$. How can we do that?

$$\begin{aligned}
\gamma_t(i) &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{k=1}^{N}\alpha_t(k)\beta_t(k)} \\
&= \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)\prod_{\tau=1}^{t} g_\tau \prod_{\tau=t}^{T} c_\tau}{\sum_{k=1}^{N}\hat{\alpha}_t(k)\hat{\beta}_t(k)\prod_{\tau=1}^{t} g_\tau \prod_{\tau=t}^{T} c_\tau} \\
&= \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{k=1}^{N}\hat{\alpha}_t(k)\hat{\beta}_t(k)}
\end{aligned}$$

State and Segment Posteriors, using the Scaled Forward-Backward Algorithm

So, because both g_t and c_t are independent of the state number i, we can just use $\hat{\alpha}$ and $\hat{\beta}$ in place of α and β:

1. **The State Posterior:**

$$\gamma_t(i) = p(q_t = i|X, \Lambda) = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{k=1}^N \hat{\alpha}_t(k)\hat{\beta}_t(k)}$$

2. **The Segment Posterior:**

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

Outline

Zero-valued denominators

$$\gamma_t(i) = p(q_t = i|X, \Lambda) = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{k=1}^{N} \hat{\alpha}_t(k)\hat{\beta}_t(k)}$$

- The scaled forward-backward algorithm guarantees that $\hat{\alpha}_t(i) > 0$ for at least one i, and $\hat{\beta}_t(i) > 0$ for at least one i.
- But scaled F-B doesn't guarantee that it's the same i! It is possible that $\hat{\alpha}_t(i)\hat{\beta}_t(i) = 0$ for all i.
- Therefore it's still possible to get in a situation with $\sum_{k=1}^{N} \hat{\alpha}_t(k)\hat{\beta}_t(k) = 0$.

The solution: just leave it alone

- Remember what $\gamma_t(i)$ is actually used for:

$$\vec{\mu}_i' = \frac{\sum_{t=1}^{T} \gamma_t(i)\vec{x}_t}{\sum_{t=1}^{T} \gamma_t(i)}$$

- If $\sum_{k=1}^{N} \hat{\alpha}_t(k)\hat{\beta}_t(k) = 0$, that means that the frame $\vec{x}_t$ is **highly unlikely** to have been produced by **any** state (it's an outlier: some sort of weird background noise or audio glitch).
- So the solution: just set $\gamma_t(i) = 0$ for that frame, for all states.

Posteriors, with compensation for zero denominators

1. **The State Posterior:**

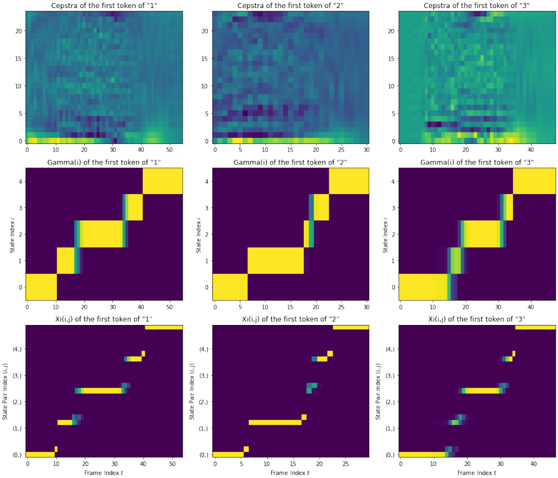
$$\gamma_t(i) = \begin{cases} \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{k=1}^N \hat{\alpha}_t(k)\hat{\beta}_t(k)} & \sum_{k=1}^N \hat{\alpha}_t(k)\hat{\beta}_t(k) > 0 \\ 0 & \text{otherwise} \end{cases}$$

2. **The Segment Posterior:**

$$\xi_t(i,j) = \begin{cases} \frac{\hat{\alpha}_t(i) a_{ij} b_j(\vec{x}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{k=1}^N \sum_{\ell=1}^N \hat{\alpha}_t(k) a_{k\ell} b_\ell(\vec{x}_{t+1})\hat{\beta}_{t+1}(\ell)} & \text{denom} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Result example

Here are $\gamma_t(i)$ and $\xi_t(i,j)$, plotted as a function of i,j and t, for the words "one," "two," and "three."



Outline

Re-estimating the covariance

$$\Sigma_i' = \frac{\sum_{t=1}^{T} \gamma_t(i)(\vec{x}_t - \vec{\mu}_i)(\vec{x}_t - \vec{\mu}_i)^T}{\sum_{t=1}^{T} \gamma_t(i)}$$

Here's a bad thing that can happen:

- $\gamma_t(i)$ is nonzero for fewer than D frames.
- Therefore, the formula above results in a singular-valued Σ_i'. Thus $|\Sigma_i'| = 0$, and $\Sigma_i^{-1} = \infty$.

Writing Baum-Welch as a Matrix Equation

Let's re-write the M-step as a matrix equation. Define two new matrices, X and W:

$$X = \left[\begin{array}{c} (\vec{x}_1 - \vec{\mu}_i)^T \\ (\vec{x}_2 - \vec{\mu}_i)^T \\ \vdots \\ (\vec{x}_T - \vec{\mu}_i)^T \end{array}\right], \quad W = \left[\begin{array}{cccc} \frac{\gamma_1(i)}{\sum_{t=1}^T \gamma_t(i)} & 0 & \cdots & 0 \\ 0 & \frac{\gamma_2(i)}{\sum_{t=1}^T \gamma_t(i)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 00 & \cdots & \frac{\gamma_T(i)}{\sum_{t=1}^T \gamma_t(i)} & \end{array}\right]$$

Writing Baum-Welch as a Matrix Equation

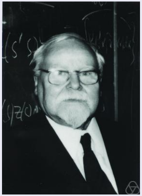
In terms of those two matrices, the Baum-Welch re-estimation formula is:

$$\Sigma_i = X^T W X$$

…and the problem we have is that $X^T W X$ is singular, so that $(X^T W X)^{-1}$ is infinite.

Tikhonov Regularization

Andrey Tikhonov



Andrey Tikhonov studied ill-posed problems (problems in which we try to estimate more parameters than the number of data points, e.g., covariance matrix has more dimensions than the number of training tokens).

Tikhonov regularization

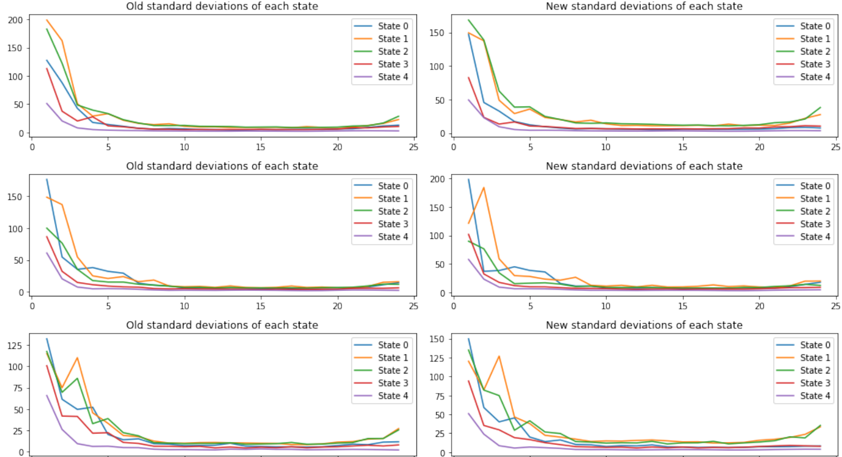
Tikhonov proposed a very simple solution that guarantees Σ_i to be nonsingular:

$$\Sigma_i = X^T W X + \alpha I$$

...where I is the identity matrix, and α is a tunable hyperparameter called the "regularizer."

Result example

Here are the diagonal elements of the covariance matrices for each state, before and after re-estimation. You can't really see it in this plot, but all the variances in the right-hand column have had the Tiknonov regularizer $\alpha = 1$ added to them.



Outline

Numerical Issues: Hyperparameters

We now have solutions to the four main numerical issues. Unfortunately, two of them require "hyperparameters" (a.k.a. "tweak factors").

- The observation pdf floor.
- The Tiknonov regularizer.

These are usually adjusted using the development test data, in order to get best results.

The Scaled Forward Algorithm

1. **Initialize:**

$$\hat{\alpha}_1(i) = \frac{1}{g_1}\pi_i b_i(\vec{x}_1)$$

2. **Iterate:**

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

$$g_t = \sum_{j=1}^{N} \tilde{\alpha}_t(j)$$

$$\hat{\alpha}_t(j) = \frac{1}{g_t}\tilde{\alpha}_t(j)$$

3. **Terminate:**

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

The Scaled Backward Algorithm

1. **Initialize:**

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

2. **Iterate:**

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

$$\hat{\beta}_t(i) = \frac{1}{c_t} \tilde{\beta}_t(i)$$

Rabiner uses $c_t = g_t$, but I recommend instead that you use

$$c_t = \max_i \tilde{\beta}_t(i)$$

Posteriors, with compensation for zero denominators

1. **The State Posterior:**

$$\gamma_t(i) = \begin{cases} \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{k=1}^N \hat{\alpha}_t(k)\hat{\beta}_t(k)} & \sum_{k=1}^N \hat{\alpha}_t(k)\hat{\beta}_t(k) > 0 \\ 0 & \text{otherwise} \end{cases}$$

2. **The Segment Posterior:**

$$\xi_t(i,j) = \begin{cases} \frac{\hat{\alpha}_t(i)a_{ij}b_j(\vec{x}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{k=1}^N \sum_{\ell=1}^N \hat{\alpha}_t(k)a_{k\ell}b_\ell(\vec{x}_{t+1})\hat{\beta}_{t+1}(\ell)} & \text{denom} > 0 \\ 0 & \text{otherwise} \end{cases}$$