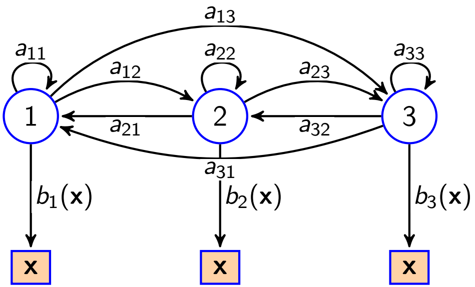
Baum-Welch and Viterbi

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing

Outline

Hidden Markov Model



1. Start in state $q_t = i$ with pmf π_i.
2. Generate an observation, $\mathbf{x}$, with pdf $b_i(\mathbf{x})$.
3. Transition to a new state, $q_{t+1} = j$, according to pmf a_{ij}.
4. Repeat.

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 $p(q_t = i|X, \Lambda)$?
3. **Training:** Given an initial HMM Λ, and an observation sequence X, can we find Λ' such that $p(X|\Lambda') > p(X|\Lambda)$?

The Forward Algorithm

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

1. **Initialize:**
$$\alpha_1(i) = \pi_i b_i(\mathbf{x}_1), \quad 1 \le i \le N$$
2. **Iterate:**
$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{i,j} b_j(\mathbf{x}_t), \;\; 1 \le j \le N, \;\; 2 \le t \le T$$
3. **Terminate:**
$$\Pr\{\mathbf{X}|\Lambda\} = \sum_{i=1}^{N} \alpha_T(i)$$

The Backward Algorithm

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

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

Segmentation

1. **The State Posterior:**

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

2. **The Segment Posterior:**

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

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 $\mathbf{X}$? In other words, $\Pr\{\mathbf{X}|\Lambda_1) > p(\mathbf{X}|\Lambda_2\}$?
2. **Segmentation:** What is $\Pr\{q_t = i|\mathbf{X}, \Lambda\}$?
3. **Training:** Given an initial HMM Λ, and an observation sequence $\mathbf{X}$, can we find Λ' such that $\Pr\{\mathbf{X}|\Lambda'\} > \Pr\{\mathbf{X}|\Lambda\}$?

Outline

Maximum Likelihood Training

Suppose we're given several observation sequences of the form $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$. Suppose, also, that we have some initial guess about the values of the model parameters (our initial guess doesn't have to be very good). Maximum likelihood training means we want to compute a new set of parameters, $\Lambda' = \left\{\pi'_i, a'_{i,j}, b'_j(\mathbf{x})\right\}$ that maximize $\Pr\left\{\mathbf{X}|\Lambda'\right\}$.

1. **Initial State Probabilities:** Find values of π'_i, $1 \le i \le N$, that maximize $\Pr\{X|\Lambda'\}$.
2. **Transition Probabilities:** Find values of $a'_{i,j}$, $1 \le i, j \le N$, that maximize $\Pr\{X|\Lambda'\}$.
3. **Observation Probabilities:** Learn $b'_j(\mathbf{x})$. What does that mean, actually?

Learning the Observation Probabilities

There are three common ways of representing the observation probabilities, $b_j(\mathbf{x})$.

1. Vector quantize $\mathbf{x}$, using some VQ method. Suppose $\mathbf{x}$ is the k^{th} codevector; then we just need to learn $b_j(k)$ such that

$$b_j(k) \geq 0, \quad \sum_{k=0}^{K-1} b_j(k) = 1$$

2. Model $b_j(k)$ as a Gaussian, or some other parametric pdf model, and learn its parameters.
3. Model $b_j(k)$ as a neural net, and learn its parameters.

Maximum Likelihood Training

For now, suppose that we have the following parameters that we need to learn:

1. **Initial State Probabilities:** π'_i such that

$$\pi'_i \geq 0, \quad \sum_{i=1}^{N} \pi'_i = 1$$

2. **Transition Probabilities:** $a'_{i,j}$ such that

$$a'_{i,j} \geq 0, \quad \sum_{j=1}^{N} a'_{i,j} = 1$$

3. **Observation Probabilities:** $b'_j(k)$ such that

$$b'_j(k) \geq 0, \quad \sum_{k=1}^{K} b'_j(k) = 1$$

Maximum Likelihood Training with Known State Sequence

Impossible assumption: Suppose that we actually know the state sequences, $\mathbf{q} = [q_1, \ldots, q_T]^T$, matching with each observation sequence $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$. Then what would be the maximum-likelihood parameters?

Maximum Likelihood Training with Known State Sequence

Our goal is to find $\Lambda = \{\pi_i, a_{i,j}, b_j(k)\}$ in order to maximize

$$\begin{aligned}\mathcal{L}(\Lambda) &= \sum_{\text{sequences}} \ln \Pr\{\mathbf{q}, \mathbf{X}|\Lambda\} \\ &= \ln \pi_{q_1} + \ln b_{q_1}(x_1) + \ln a_{q_1,q_2} + b_{q_2}(x_2) + \ldots \\ &= \sum_{i=1}^{N} \left(s_i \ln \pi_i + \sum_{j=1}^{N} n_{i,j} \ln a_{i,j} + \sum_{k=1}^{K} m_{i,k} \ln b_i(k) \right)\end{aligned}$$

where

- s_i is the number of sequences that started with state i,
- $n_{i,j}$ is the number of frames in which $(q_t = i, q_{t+1} = j)$,
- $m_{i,k}$ is the number of frames in which $(q_t = i, k_t = k)$

Maximum Likelihood Training with Known State Sequence

$$\mathcal{L}(\Lambda) = \sum_{i=1}^{N} \left(s_i \ln \pi_i + \sum_{j=1}^{N} n_{i,j} \ln a_{i,j} + \sum_{k=1}^{K} m_{i,k} \ln b_i(k) \right)$$

When we differentiate that, we find the following derivatives:

$$\frac{\partial \mathcal{L}}{\partial \pi_i} = \frac{s_i}{\pi_i}$$
$$\frac{\partial \mathcal{L}}{\partial a_{i,j}} = \frac{n_{i,j}}{a_{i,j}}$$
$$\frac{\partial \mathcal{L}}{\partial b_j(k)} = \frac{m_{j,k}}{b_j(k)}$$

These derivatives are **never** equal to zero! What went wrong?

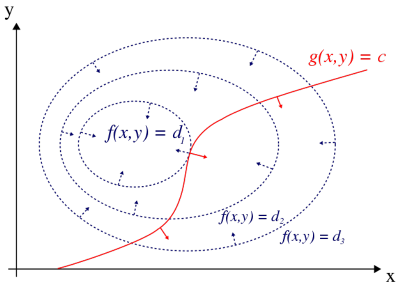
Maximum Likelihood Training with Known State Sequence

Here's the problem: we forgot to include the constraints $\sum_i \pi_i = 1$, $\sum_j a_{i,j} = 1$, and $\sum_k b_j(k) = 1$!
We can include the constraints using the method of Lagrange multipliers.

Lagrange Multipliers

The method of Lagrange multipliers is a general solution to the following problem:

- x and y are parameters
- $f(x, y)$ is a function we're trying to maximize or minimize…
- …subject to the constraint that $g(x, y) = 0$, for some function $g(\cdot)$.



`https://commons.wikimedia.org/wiki/File:LagrangeMultipliers2D.svg`

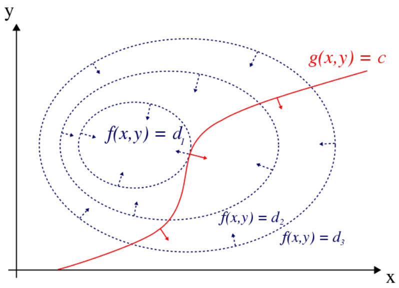
Lagrange Multipliers

The constrained optimum value of x, y can be found by:

1. Invent a scalar variable λ called the "Lagrange multiplier." In terms of λ, find the values $x^*(\lambda), y^*(\lambda)$ that maximize

$$\mathcal{J}(x,y) = f(x,y) + \lambda g(x,y)$$

2. Choose λ so that $g(x^*(\lambda), y^*(\lambda)) = 0$

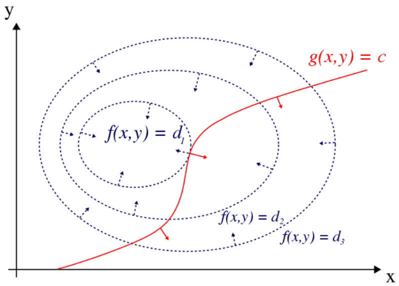


https://commons.wikimedia.org/wiki/File:LagrangeMultipliers2D.svg

Geometric Intuition

Geometric intuition:

1. Suppose, at the peak of $f(x, y)$, the constraint is not satisfied: $g(x, y) < 0$
2. Then we add a penalty term, $f(x, y) + \lambda g(x, y)$, so that the old peak is not as high, and places with higher values of $g(x, y)$ are better



https://commons.wikimedia.org/wiki/File:LagrangeMultipliers2D.svg

Maximum Likelihood Training with Known State Sequence

For the HMM, we want to maximize

$$\mathcal{L}(\Lambda) = \sum_{i=1}^{N} \left(s_i \ln \pi_{q_1} + \sum_{j=1}^{N} n_{i,j} \ln a_{i,j} + \sum_{k=1}^{K} m_{i,k} \ln b_i(k) \right)$$

...subject to the following constraints: $\sum_i \pi_i = 1$, $\sum_j a_{i,j} = 1$, and $\sum_k b_j(k) = 1$.

Maximum Likelihood Training with Known State Sequence

Define the Lagrangian:

$$\mathcal{J}(\Lambda) = \sum_{i=1}^{N} \left(s_i \ln \pi_{q_1} + \sum_{j=1}^{N} n_{i,j} \ln a_{i,j} + \sum_{k=1}^{K} m_{i,k} \ln b_i(k) \right)$$

$$+ \lambda_1 \left(1 - \sum_{i=1}^{N} \pi_i \right) + \sum_{i=1}^{N} \lambda_{2,i} \left(1 - \sum_{j=1}^{N} a_{i,j} \right)$$

$$+ \sum_{j=1}^{N} \lambda_{3,j} \left(1 - \sum_{k=1}^{N} b_j(k) \right)$$

Maximum Likelihood Training with Known State Sequence

The derivatives of the Lagrangian are:

$$\frac{\partial \mathcal{J}}{\partial \pi_i} = \frac{s_i}{\pi_i} - \lambda_1$$

$$\frac{\partial \mathcal{J}}{\partial a_{i,j}} = \frac{n_{i,j}}{a_{i,j}} - \lambda_{2,i}$$

$$\frac{\partial \mathcal{J}}{\partial b_j(k)} = \frac{m_{j,k}}{b_j(k)} - \lambda_{3,i}$$

The optimum values of the parameters are:

$$\pi_i^* = \frac{s_i}{\lambda_1}$$

$$a_{i,j}^* = \frac{n_{i,j}}{\lambda_{2,i}}$$

$$b_j^*(k) = \frac{m_{j,k}}{\lambda_{3,j}}$$

Maximum Likelihood Training with Known State Sequence

The values of λ_1, $\lambda_{2,i}$, and $\lambda_{3,j}$ that cause the constraints to be satisfied are

$$\lambda_1 = \sum_i s_i, \quad \lambda_{2,i} = \sum_j n_{i,j}, \quad \lambda_{3,j} = \sum_k m_{j,k}$$

...which gives the constrained optimum parameters of the HMM to be:

$$\pi_i^* = \frac{s_i}{\sum_i s_i}$$

$$a_{i,j}^* = \frac{n_{i,j}}{\sum_j n_{i,j}}$$

$$b_j^*(k) = \frac{m_{j,k}}{\sum_k m_{j,k}}$$

Maximum Likelihood Training with Known State Sequence

Using the Lagrange multiplier method, the maximum likelihood parameters for the HMM are:

1. **Initial State Probabilities:**

$$\pi'_i = \frac{\#\text{ state sequences that start with } q_1 = i}{\#\text{ state sequences in training data}}$$

2. **Transition Probabilities:**

$$a'_{i,j} = \frac{\#\text{ frames in which } q_{t-1} = i, q_t = j}{\#\text{ frames in which } q_{t-1} = i}$$

3. **Observation Probabilities:**

$$b'_j(k) = \frac{\#\text{ frames in which } q_t = j, k_t = k}{\#\text{ frames in which } q_t = j}$$

Outline

Expectation Maximization

When the true state sequence is unknown, then we can't maximize the likelihood $\Pr\{\mathbf{q}, \mathbf{X}|\Lambda'\}$ directly. Instead, we maximize the *expected* log likelihood, with the expectation taken over all possible state sequences:

$$\mathcal{L} = E_{\mathbf{q}|\mathbf{X}}\left[\sum_{i=1}^{N}\left(s_i \ln \pi_i + \sum_{j=1}^{N} n_{i,j} \ln a_{i,j} + \sum_{k=1}^{K} m_{i,k} \ln b_i(k)\right)\right]$$

The expected log likelihood is always less than or equal to the true log likelihood, because the probability $\Pr\{\mathbf{q}|\mathbf{X}\} \leq 1$.

Expectation Maximization

The only terms in the log likelihood that depend on the state sequence are s_i, $n_{i,j}$, and $m_{i,k}$, so:

$$\begin{aligned}\mathcal{L} &= E_{\mathbf{q}|\mathbf{X}}\left[\sum_{i=1}^{N}\left(s_i \ln \pi_i + \sum_{j=1}^{N} n_{i,j}\ln a_{i,j} + \sum_{k=1}^{K} m_{i,k}\ln b_i(k)\right)\right]\\ &= \sum_{i=1}^{N}\left(E_{\mathbf{q}|\mathbf{X}}\left[s_i\right]\ln \pi_i + \sum_{j=1}^{N} E_{\mathbf{q}|\mathbf{X}}\left[n_{i,j}\right]\ln a_{i,j} + \sum_{k=1}^{K} E_{\mathbf{q}|\mathbf{X}}\left[m_{i,k}\right]\ln b_i(k)\right)\end{aligned}$$

Expectation Maximization: the M-Step (Maximize the expected log likelihood))

Maximizing the expected log likelihood gives us some very reasonable parameter estimates:

1. **Initial State Probabilities:**

$$\pi'_i = \frac{E\left[\#\text{ state sequences that start with } q_1 = i\right]}{\#\text{ state sequences in training data}}$$

2. **Transition Probabilities:**

$$a'_{i,j} = \frac{E\left[\#\text{ frames in which } q_{t-1} = i, q_t = j\right]}{E\left[\#\text{ frames in which } q_{t-1} = i\right]}$$

3. **Observation Probabilities:**

$$b'_j(k) = \frac{E\left[\#\text{ frames in which } q_t = j, k_t = k\right]}{E\left[\#\text{ frames in which } q_t = j\right]}$$

Expectation Maximization: the E-Step (compute the Expected log likelihood)

In order to find quantities like "the expected number of times $q_1 = i$," we need to compute the probabilities of all possible state alignments, $\Pr\{\mathbf{q}\}$. But actually, this simplifies quite a lot. We really only need these three quantities:

$$
\begin{aligned}
E_{\mathbf{q}|\mathbf{X}}[s_i] &= \sum_{\text{sequences}} \Pr\{q_1 = i|\mathbf{X}\} \\
E_{\mathbf{q}|\mathbf{X}}[n_{i,j}] &= \sum_t \Pr\{q_t = i, q_{t+1} = j|\mathbf{X}\} \\
E_{\mathbf{q}|\mathbf{X}}[m_{j,k}] &= \sum_t \Pr\{q_t = j, \mathbf{x}_t = k|\mathbf{X}\} \\
&= \sum_{t:\mathbf{x}_t = k} \Pr\{q_t = j|\mathbf{X}\}
\end{aligned}
$$

Expectation Maximization: the E-Step

$$E_{\mathbf{q}|\mathbf{X}}\left[s_i\right] = \sum_{\text{sequences}} \Pr\{q_1 = i|\mathbf{X}\}$$

$$E_{\mathbf{q}|\mathbf{X}}\left[n_{i,j}\right] = \sum_t \Pr\{q_t = i, q_{t+1} = j|\mathbf{X}\}$$

$$E_{\mathbf{q}|\mathbf{X}}\left[m_{j,k}\right] = \sum_{t:\mathbf{x}_t=k} \Pr\{q_t = j|\mathbf{X}\}$$

But these are things we already know! They are:

$$E_{\mathbf{q}|\mathbf{X}}\left[s_i\right] = \sum_{\text{sequences}} \gamma_1(i)$$

$$E_{\mathbf{q}|\mathbf{X}}\left[n_{i,j}\right] = \sum_t \xi_t(i,j)$$

$$E_{\mathbf{q}|\mathbf{X}}\left[m_{j,k}\right] = \sum_{t:\mathbf{x}_t=k} \gamma_t(j)$$

The Baum-Welch Algorithm

1. **Initial State Probabilities:**

$$\pi_i' = \frac{E\left[\#\text{ state sequences that start with } q_1 = i\right]}{\#\text{ state sequences in training data}}$$
$$= \frac{\sum_{sequences} \gamma_1(i)}{\#\text{ sequences}}$$

The Baum-Welch Algorithm

1.
2. **Transition Probabilities:**

$$a'_{i,j} = \frac{E\left[\#\text{ frames in which } q_{t-1}=i, q_t=j\right]}{E\left[\#\text{ frames in which } q_{t-1}=i\right]}$$
$$= \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{j=1}^{N}\sum_{t=1}^{T-1}\xi_t(i,j)}$$

The Baum-Welch Algorithm

1.
2.
3. **Observation Probabilities:**

$$b'_j(k) = \frac{E\left[\#\text{ frames in which } q_t = j, k_t = k\right]}{E\left[\#\text{ frames in which } q_t = j\right]}$$

$$= \frac{\sum_{t:\mathbf{x}_t=k} \gamma_t(j)}{\sum_t \gamma_t(j)}$$

Summary: The Baum-Welch Algorithm

1. **Initial State Probabilities:**

$$\pi_i' = \frac{\sum_{sequences} \gamma_1(i)}{\#\text{ sequences}}$$

2. **Transition Probabilities:**

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

3. **Observation Probabilities:**

$$b_j'(k) = \frac{\sum_{t:\mathbf{x}_t=k} \gamma_t(j)}{\sum_t \gamma_t(j)}$$

Outline

Other Alphas: the Scaled and Neural Forward-Backward Algorithms

- The standard forward-backward algorithm defines $\alpha_t(i)$ and $\beta_t(i)$ in the way that makes the theory easiest to learn.
- The scaled forward-backward algorithm rescales both to avoid numerical underflow.
- The neural forward-backward algorithm (Graves, 2006) redefines $\beta_t(i)$ in a way that's easier to implement using neural networks.

Numerical Issues

Notice that $a_{i,j} = \mathcal{O}\left\{\frac{1}{N}\right\}$, and with discrete observations, $b_j(\mathbf{x}_t) = \mathcal{O}\left\{\frac{1}{K}\right\}$. A typical 3-second sentence has 300 frames. If $K \approx 1000$, then

$$\begin{aligned}
\alpha_t(i) &= \sum_{j=1}^{N} \alpha_{t-1}(j) a_{j,i} b_i(\mathbf{x}_t) \\
&= \mathcal{O}\left\{\left(\frac{1}{K}\right)^t\right\} = \mathcal{O}\left\{10^{-300}\right\} \\
\beta_t(i) &= \sum_{j=1}^{N} a_{i,j} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j) \\
&= \mathcal{O}\left\{\left(\frac{1}{K}\right)^{T-t}\right\} = \mathcal{O}\left\{10^{-300}\right\}
\end{aligned}$$

That's small enough to cause floating-point underflow in many processors.

The Solution: Scaling

The solution is to redefine $\alpha_t(i)$ and $\beta_t(i)$ so they don't underflow. A useful definition is

$$\hat{\alpha}_t(i) = \frac{\sum_{j=1}^N \hat{\alpha}_{t-1}(j) a_{j,i} b_i(\mathbf{x}_t)}{\sum_{i=1}^N \sum_{j=1}^N \hat{\alpha}_{t-1}(j) a_{j,i} b_i(\mathbf{x}_t)}$$

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

Notice that we compute these by finding the numerator for each i, then normalizing so that $\sum_i \hat{\alpha}_t(i) = \sum_i \hat{\beta}_t(i) = 1$.

Probabilistic Interpretation of Scaled Forward-Backward

Remember that the original forward-backward probabilities had these interpretations:

$$\alpha_t(i) = \Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i | \Lambda)$$
$$\beta_t(i) = \Pr\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T | q_t = i, \Lambda)$$

Rescaling at each time step, so that $\sum_i \hat{\alpha}_t(i) = \sum_i \hat{\beta}_t(i) = 1$, has the following meaning:

$$\hat{\alpha}_t(i) = g_1(t) \Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i | \Lambda)$$
$$\hat{\beta}_t(i) = g_2(t) \Pr\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T | q_t = i, \Lambda),$$

where the constants $g_1(t)$ and $g_2(t)$ depend on the frame index (t), but don't depend on the state index (i).

Baum-Welch with Scaled Forward-Backward

Baum-Welch computes the following probabilities:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i'=1}^N \alpha_t(i')\beta_t(i')} = \frac{g_1(t)g_2(t)\alpha_t(i)\beta_t(i)}{g_1(t)g_2(t)\sum_{i'=1}^N \alpha_t(i')\beta_t(i')}$$

$$= \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{i'=1}^N \hat{\alpha}_t(i')\hat{\beta}_t(i')}$$

Similarly,

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{i,j}b_j(\mathbf{x}_{t+1})\beta_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \alpha_t(i')a_{i',j'}b_{j'}(\mathbf{x}_{t+1})\beta_{t+1}(j')}$$

$$= \frac{\hat{\alpha}_t(i)a_{i,j}b_j(\mathbf{x}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \hat{\alpha}_t(i')a_{i',j'}b_{j'}(\mathbf{x}_{t+1})\hat{\beta}_{t+1}(j')}$$

So scaling has no effect on Baum-Welch re-estimation, as long as $g_1(t)$ and $g_2(t)$ are independent of i.

Neural Baum-Welch

Neural network implementations of Baum-Welch usually make one more modification. Instead of

$$\hat{\alpha}_t(i) = g_1(t)\,\Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i|\Lambda)$$
$$\hat{\beta}_t(i) = g_2(t)\,\Pr\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T|q_t = i, \Lambda),$$

end-to-end neural networks usually rescale $\alpha_t(i)$ and $\beta_t(i)$ as:

$$\check{\alpha}_t(i) = c_1(t)\,\Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i|\Lambda)$$
$$\check{\beta}_t(i) = c_2(t)\,\Pr\{\mathbf{x}_t, \ldots, \mathbf{x}_T|q_t = i, \Lambda),$$

where the constants $c_1(t) = g_1(t)$ but $c_2(t) \neq g_2(t)$.

Neural Baum-Welch

The reason for the neural Baum-Welch is that it makes $\xi_t(i,j)$ a little easier to compute. Instead of

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i)a_{i,j}b_j(\mathbf{x}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \hat{\alpha}_t(i')a_{i',j'}b_{j'}(\mathbf{x}_{t+1})\hat{\beta}_{t+1}(j')},$$

we now have

$$\xi_t(i,j) = \frac{\check{\alpha}_t(i)a_{i,j}\check{\beta}_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \check{\alpha}_t(i')a_{i',j'}\check{\beta}_{t+1}(j')}$$

Summary: Original, Scaled, and Neural Forward-Backward Algorithms

- Original:

$$\alpha_t(i) = \Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i|\Lambda)$$
$$\beta_t(i) = \Pr\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T|q_t = i, \Lambda)$$

- Scaled:

$$\hat{\alpha}_t(i) = g_1(t)\Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i|\Lambda)$$
$$\hat{\beta}_t(i) = g_2(t)\Pr\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T|q_t = i, \Lambda)$$

- Neural:

$$\check{\alpha}_t(i) = c_1(t)\Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i|\Lambda)$$
$$\check{\beta}_t(i) = c_2(t)\Pr\{\mathbf{x}_t, \ldots, \mathbf{x}_T|q_t = i, \Lambda)$$

Summary: Original, Scaled, and Neural Forward-Backward Algorithms

- Original:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{i,j} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \alpha_t(i') a_{i',j'} b_{j'}(\mathbf{x}_{t+1}) \beta_{t+1}(j')}$$

- Scaled:

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i) a_{i,j} b_j(\mathbf{x}_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \hat{\alpha}_t(i') a_{i',j'} b_{j'}(\mathbf{x}_{t+1}) \hat{\beta}_{t+1}(j')}$$

- Neural:

$$\xi_t(i,j) = \frac{\check{\alpha}_t(i) a_{i,j} \check{\beta}_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \check{\alpha}_t(i') a_{i',j'} \check{\beta}_{t+1}(j')}$$

Outline

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:

$$\mathbf{q}^* = \operatorname*{argmax}_{\mathbf{q}} \Pr\{\mathbf{q}, \mathbf{X}|\Lambda\}$$

The Max-Probability State Sequence

The problem of finding the max-probability state sequence is just as hard as the problem of finding $\Pr\{\mathbf{X}|\Lambda\}$, for exactly the same reason:

$$\max_{\mathbf{q}} \Pr\{\mathbf{q},\mathbf{X}|\Lambda\} = \max_{q_T=1}^{N} \cdots \max_{q_1=1}^{N} \Pr\{\mathbf{q},\mathbf{X}|\Lambda\}$$

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

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

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

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

The Viterbi Algorithm

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

1. **Initialize:**
$$\delta_1(i) = \pi_i b_i(\mathbf{x}_1)$$
2. **Iterate:**
$$\delta_t(j) = \max_{i=1}^{N} \delta_{t-1}(i) a_{i,j} b_j(\mathbf{x}_t)$$
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:

$$\begin{aligned}
\psi_t(j) &= \operatorname*{argmax}_i \cdots \max_{q_1} \Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_1, \ldots, q_{t-1} = i, q_t = j | \Lambda\} \\
&= \operatorname*{argmax}_{i=1}^{N} \delta_{t-1}(i) a_{i,j} b_j(\mathbf{x}_t)
\end{aligned}$$

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:**

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

2. **Iterate:**

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

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

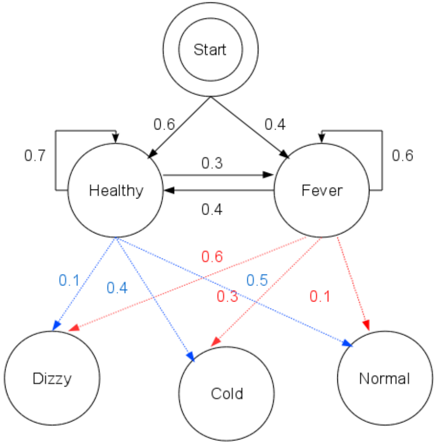
3. **Terminate:**

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

4. **Backtrace:**

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

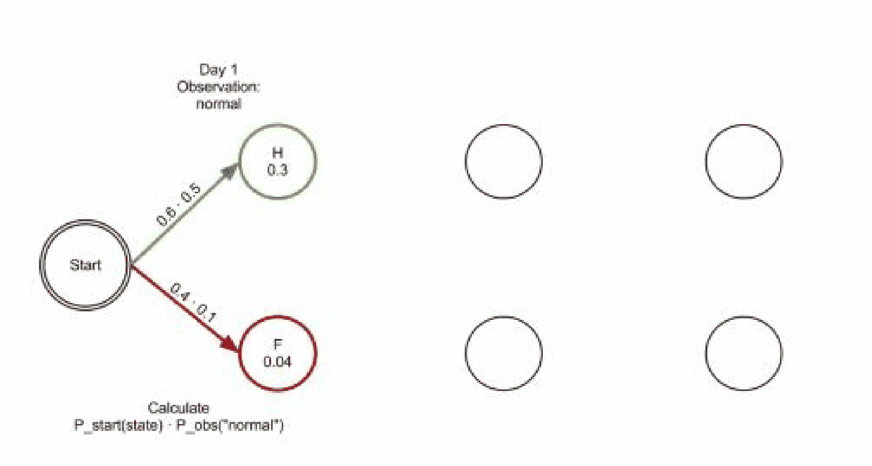
Example



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-backward algorithm. But this time, there is an easy solution, because the log of the max is equal to the max of the log:

$$\begin{aligned}
\ln \delta_t(j) &= \ln \left(\max_{i=1}^{N} \delta_{t-1}(i) a_{i,j} b_j(\mathbf{x}_t) \right) \\
&= \max_{i=1}^{N} \left(\ln \delta_{t-1}(i) + \ln a_{i,j} + \ln b_j(\mathbf{x}_t) \right)
\end{aligned}$$

The Log-Viterbi Algorithm

1. **Initialize:**

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

2. **Iterate:**

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

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

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

Outline

The Baum-Welch Algorithm: Initial and Transition Probabilities

1. **Initial State Probabilities:**

$$\pi'_i = \frac{\sum_{sequences} \gamma_1(i)}{\#\ \text{sequences}}$$

2. **Transition Probabilities:**

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

3. **Observation Probabilities:**

$$b'_j(k) = \frac{\sum_{t:\mathbf{x}_t=k} \gamma_t(j)}{\sum_t \gamma_t(j)}$$

Summary: Original, Scaled, and Neural Forward-Backward Algorithms

- Original:

$$\alpha_t(i) = \Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i|\Lambda)$$
$$\beta_t(i) = \Pr\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T|q_t = i, \Lambda)$$

- Scaled:

$$\hat{\alpha}_t(i) = g_1(t)\Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i|\Lambda)$$
$$\hat{\beta}_t(i) = g_2(t)\Pr\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T|q_t = i, \Lambda)$$

- Neural:

$$\check{\alpha}_t(i) = c_1(t)\Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_t = i|\Lambda)$$
$$\check{\beta}_t(i) = c_2(t)\Pr\{\mathbf{x}_t, \ldots, \mathbf{x}_T|q_t = i, \Lambda)$$

Summary: Original, Scaled, and Neural Forward-Backward Algorithms

- Original:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{i,j} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \alpha_t(i') a_{i',j'} b_{j'}(\mathbf{x}_{t+1}) \beta_{t+1}(j')}$$

- Scaled:

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i) a_{i,j} b_j(\mathbf{x}_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \hat{\alpha}_t(i') a_{i',j'} b_{j'}(\mathbf{x}_{t+1}) \hat{\beta}_{t+1}(j')}$$

- Neural:

$$\xi_t(i,j) = \frac{\check{\alpha}_t(i) a_{i,j} \check{\beta}_{t+1}(j)}{\sum_{i'=1}^N \sum_{j'=1}^N \check{\alpha}_t(i') a_{i',j'} \check{\beta}_{t+1}(j')}$$

The Log-Viterbi Algorithm

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

Outline

Written Example

In a second-order Markov process, q_t depends on both q_{t-2} and q_{t-1}, thus the model parameters are:

$$\pi_{i,j} = \Pr\{q_1 = i, q_2 = j\} \tag{1}$$
$$a_{i,j,k} = \Pr\{q_t = k | q_{t-2} = i, q_{t-1} = i\} \tag{2}$$
$$b_k(\mathbf{x}) = \Pr\{\mathbf{x} | q_t = k\} \tag{3}$$

Suppose you have a sequence of observations for which you have already $\alpha_t(i,j)$ and $\beta_t(i,j)$, defined as

$$\alpha_t(i,j) = \Pr\{\mathbf{x}_1, \ldots, \mathbf{x}_t, q_{t-1} = i, q_t = j | \Lambda\} \tag{4}$$
$$\beta_t(i,j) = \Pr\{\mathbf{x}_{t+1}, \ldots, \mathbf{x}_T | q_{t-1} = i, q_t = j, \Lambda\} \tag{5}$$

In terms of the quantities defined in Eqs. (1) through (5), find a formula that re-estimates a'_{ijk} so that, unless $a_{i,j,k}$ is already optimal,

$$\Pr\{\mathbf{X} | \pi_i, a'_{i,j,k}, b_j(\mathbf{x})\} > \Pr\{\mathbf{X} | \pi_i, a_{i,j,k}, b_j(\mathbf{x})\}$$