Hidden Markov Models
Part 2: Algorithms

CSE 4309 – Machine Learning

Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Hidden Markov Model

- An HMM consists of:
  - A set of states $s_1, \dots, s_K$.
  - An **initial state probability** function $\pi_k = p(z_1 = s_k)$.
  - A **state transition matrix $\boldsymbol{A}$**, of size $K \times K$, where

$$A_{k,j} = p\left(z_n = s_j \mid z_{n-1} = s_k\right)$$

  - **Observation probability functions**, also called **emission probabilities**, defined as:

$$\varphi_k(x) = p(x_n = x \mid z_n = s_k)$$

The Basic HMM Problems

- We will next see algorithms for the four problems that we usually want to solve when using HMMs.
- Problem 1: learn an HMM using training data. This involves:
  - Learning the initial state probabilities π_k.
  - Learning the state transition matrix $\boldsymbol{A}$.
  - Learning the observation probability functions $P_k(x)$.
- Problems 2, 3, 4 assume we have already learned an HMM.
- Problem 2: given a sequence of observations $X = x_1, \dots, x_N$, compute $p(X \mid \mathrm{HMM})$: the probability of X given the model.
- Problem 3: given an observation sequence $X = x_1, \dots, x_N$, find the hidden state sequence $Z = z_1, \dots, z_N$ maximizing $p(Z \mid X, \mathrm{HMM})$
- Problem 4: given an observation sequence $X = x_1, \dots, x_N$, compute, for any n, k, the probability $p(z_n = s_k \mid X, \mathrm{HMM})$

The Basic HMM Problems

- Problem 1: learn an HMM using training data.
- Problem 2: given a sequence of observations $X = x_1, \ldots, x_N$, compute $p(X \mid \mathrm{HMM})$: the probability of X given the model.
- Problem 3: given an observation sequence $X = x_1, \ldots, x_N$, find the hidden state sequence $Z = z_1, \ldots, z_N$ maximizing $p(Z \mid X, \mathrm{HMM})$
- Problem 4: given an observation sequence $X = x_1, \ldots, x_N$, compute, for any n, k, the probability $p(z_n = s_k \mid X, \mathrm{HMM})$
- We will first look at algorithms for problems 2, 3, 4.
- Last, we will look at the standard algorithm for learning an HMM using training data.

Probability of Observations

- Problem 2: given an HMM model θ, and a sequence of observations $X = x_1, \ldots, x_N$, compute $p(X \mid \theta)$: the probability of X given the model.
- Inputs:
  - $\theta = (\pi, A, \varphi_k)$: a trained HMM, with probability functions π, A, φ_k specified.
  - A sequence of observations $X = x_1, \ldots, x_N$.
- Output: $p(X \mid \theta)$.

Probability of Observations

- Problem 2: given a sequence of observations $X = x_1, \ldots, x_N$, compute $p(X \mid \theta)$.
- Why do we care about this problem?
  - What would be an example application?

Probability of Observations

- Problem 2: given a sequence of observations $X = x_1, \ldots, x_N$, compute $p(X \mid \theta)$.
- Why do we care about this problem?
- We need to compute $p(X \mid \theta)$ to classify X.
  - Suppose we have multiple models $\theta_1, \ldots, \theta_C$.
    - For example, we can have one model for digit 2, one model for digit 3, one model for digit 4…
  - A Bayesian classifier classifies X by finding the θ_c that maximizes $p(\theta_c \mid X)$.
  - Using Bayes rule, we must maximize $\dfrac{p(X \mid \theta_c)\, p(\theta_c)}{p(X)}$.
  - Therefore, we need to compute $p(X \mid \theta_c)$ for each c.

The Sum Rule

- The sum rule, which we saw earlier in the course, states that:

$$p(X) = \sum_{y \in \mathbb{Y}} p(X, Y = y)$$

- We can use the sum rule, to compute $p(X \mid \theta)$ as follows:

$$p(X \mid \theta) = \sum_{Z} p(X, Z \mid \theta)$$

- According to this formula, we can compute $p(X \mid \theta)$ by summing $p(X, Z \mid \theta)$ over **all possible state sequences** Z.

The Sum Rule

$$p(X \mid \theta) = \sum_{Z} p(X, Z \mid \theta)$$

- According to this formula, we can compute $p(X \mid \theta)$ by summing $p(X, Z \mid \theta)$ over **all possible state sequences** Z.
- An HMM with parameters θ specifies a joint distribution function $p(X, Z \mid \theta)$ as:

$$p(X, Z \mid \theta) = p(z_1 \mid \theta) \prod_{n=2}^{N} p(z_n \mid z_{n-1}, \theta) \prod_{n=1}^{N} p(x_n \mid z_n, \theta)$$

- Therefore:

$$p(X \mid \theta) = \sum_{Z} \left[p(z_1 \mid \theta) \prod_{n=2}^{N} p(z_n \mid z_{n-1}, \theta) \prod_{n=1}^{N} p(x_n \mid z_n, \theta) \right]$$

The Sum Rule

$$p(X \mid \theta) = \sum_{Z} \left[p(z_1 \mid \theta) \prod_{n=2}^{N} p(z_n \mid z_{n-1}, \theta) \prod_{n=1}^{N} p(x_n \mid z_n, \theta) \right]$$

- According to this formula, we can compute $p(X \mid \theta)$ by summing $p(X, Z \mid \theta)$ over **all possible state sequences** Z.
- What would be the time complexity of doing this computation directly?
- It would be linear to the number of all possible state sequences Z, which is typically exponential to N (the length of X).
- Luckily, there is a polynomial time algorithm for computing $p(X \mid \theta)$, that uses dynamic programming.

Dynamic Programming for $p(X \mid \theta)$

$$p(X \mid \theta) = \sum_Z \left[p(z_1 \mid \theta) \prod_{n=2}^{N} p(z_n \mid z_{n-1}, \theta) \prod_{n=1}^{N} p(x_n \mid z_n, \theta) \right]$$

- To compute $p(X \mid \theta)$, we use a dynamic programming algorithm that is called the **forward algorithm.**
- We define a 2D array of problems, of size $N \times K$.
  - N: length of observation sequence X.
  - K: number of states in the HMM.
- Problem (n, k):
  - Compute $\alpha[n, k] = p(x_1, \ldots, x_n, z_n = s_k \mid \theta)$

The Forward Algorithm - Initialization

$$p(X \mid \theta) = \sum_{Z} \left[p(z_1 \mid \theta) \prod_{n=2}^{N} p(z_n \mid z_{n-1}, \theta) \prod_{n=1}^{N} p(x_n \mid z_n, \theta) \right]$$

- Problem (n, k):
  - Compute $\alpha[n, k] = p(x_1, \ldots, x_n, z_n = s_k \mid \theta)$
- Problem $(1, k)$:
  - ???

The Forward Algorithm - Initialization

$$p(X \mid \theta) = \sum_Z \left[p(z_1 \mid \theta) \prod_{n=2}^{N} p(z_n \mid z_{n-1}, \theta) \prod_{n=1}^{N} p(x_n \mid z_n, \theta) \right]$$

- Problem (n, k):
  - Compute $\alpha[n, k] = p(x_1, \dots, x_n, z_n = s_k \mid \theta)$
- Problem $(1, k)$:
  - Compute $\alpha[1, k] = p(x_1, z_1 = s_k \mid \theta)$
- Solution:
  - ???

The Forward Algorithm - Initialization

$$p(X \mid \theta) = \sum_{Z} \left[p(z_1 \mid \theta) \prod_{n=2}^{N} p(z_n \mid z_{n-1}, \theta) \prod_{n=1}^{N} p(x_n \mid z_n, \theta) \right]$$

- Problem (n, k):
  - Compute $\alpha[n, k] = p(x_1, \ldots, x_n, z_n = s_k \mid \theta)$
- Problem $(1, k)$:
  - Compute $\alpha[1, k] = p(x_1, z_1 = s_k \mid \theta)$
- Solution:
  - $\alpha[1, k] = p(x_1, z_1 = k \mid \theta) = \pi_k \varphi_k(x_1)$

The Forward Algorithm - Main Loop

- Problem (n, k):
  - Compute $\alpha[n, k] = p(x_1, \ldots, x_n, z_n = s_k | \theta)$
- Solution:

The Forward Algorithm - Main Loop

- Problem (n, k):
  - Compute $\alpha[n, k] = p(x_1, \dots, x_n, z_n = s_k | \theta)$
- Solution:

$$p(x_1, \dots, x_n, z_n = s_k | \theta) = \sum_{j=1}^{K} p\big(x_1, \dots, x_n, z_{n-1} = s_j, z_n = s_k | \theta\big)$$

$$= \varphi_k(x_n) \sum_{j=1}^{K} p\big(x_1, \dots, x_{n-1}, z_{n-1} = s_j, z_n = s_k | \theta\big)$$

$$= \varphi_k(x_n) \sum_{j=1}^{K} \{p\big(x_1, \dots, x_{n-1}, z_{n-1} = s_j | \theta\big) A_{j,k}\}$$

The Forward Algorithm - Main Loop

- Problem (n, k):
  - Compute $\alpha[n, k] = p(x_1, \dots, x_n, z_n = s_k | \theta)$
- Solution:

$$p(x_1, \dots, x_n, z_n = k \mid \theta)$$

$$= \varphi_k(x_n) \sum_{j=1}^{K} \{ p\big(x_1, \dots, x_{n-1}, z_{n-1} = s_j | \theta\big) A_{j,k} \}$$

$$= \varphi_k\ (x_n) \sum_{j=1}^{K} \{ \alpha[n-1, j] A_{j,k} \}$$

The Forward Algorithm - Main Loop

- Problem (n, k):
  - Compute $\alpha[n, k] = p(x_1, \ldots, x_n, z_n = s_k | \theta)$
- Solution:

$$p(x_1, \ldots, x_n, z_n = k \mid \theta) = \varphi_k\ (x_n) \sum_{j=1}^{K} \{\alpha[n-1, j] A_{j,k}\}$$

- Thus, $\alpha[n, k] = \varphi_k\ (x_n) \sum_{j=1}^{K} \{\alpha[n-1, j] A_{j,k}\}$.
- $\alpha[n, k]$ is easy to compute once all $\alpha[n-1, j]$ values have been computed.

The Forward Algorithm

- Our aim was to compute $p(X \mid \theta)$.
- Using the dynamic programming algorithm we just described, we can compute $\alpha[N, k] = p(X, z_N = s_k \mid \theta)$.
- How can we use those $\alpha[N, k]$ values to compute $p(X \mid \theta)$?
- We use the sum rule:

$$p(X \mid \theta) = \sum_{k=1}^{K} \{p(X, z_N = s_k \mid \theta)\} \Rightarrow$$

$$p(X \mid \theta) = \sum_{k=1}^{K} \alpha[n, k]$$

Problem 3: Find the Best Z

- Problem 3:
  - Inputs: a trained HMM θ, and an observation sequence $X = x_1, \ldots, x_N$
  - Output: the state sequence $Z = z_1, \ldots, z_N$ maximizing $p(Z \mid \theta, X)$.
- Why do we care?
- Some times, we want to use HMMs for classification.
  - In those cases, we do not really care about maximizing $p(Z \mid \theta, X)$, we just want to find the HMM parameters θ that maximize $p(\theta | X)$.
- Some times, we want to use HMMs to figure out the most likely values of the hidden states given the observations.
  - In the tree ring example, our goal was to figure out the average temperature for each year.
  - Those average temperatures were the hidden states.
  - Solving problem 3 provides the most likely sequence of average temperatures.

Problem 3: Find the Best Z

- Problem 3:
  - Input: an observation sequence $X = x_1, \ldots, x_N$
  - Output: the state sequence $Z = z_1, \ldots, z_N$ maximizing $p(Z \mid \theta, X)$.
- We want to maximize $p(Z \mid \theta, X)$.
- Using the definition of conditional probabilities:

$$p(Z \mid \theta, X) = \frac{p(X, Z \mid \theta)}{p(X \mid \theta)}$$

- Since X is known, $p(X \mid \theta)$ will be the same over all possible state sequences Z.
- Therefore, to find the Z that maximizes $p(Z \mid \theta, X)$, it suffices to find the Z that maximizes $p(Z, X \mid \theta)$.

Problem 3: Find the Best Z

- Problem 3:
  - Input: an observation sequence $X = x_1, \ldots, x_N$
  - Output: the state sequence $Z = z_1, \ldots, z_N$ maximizing $p(Z \mid \theta, X)$, which is the same as maximizing $p(Z, X \mid \theta)$.
- Solution: (again) dynamic programming.
- Problem (n, k): Compute $\mathrm{G}[n, k]$ and $\mathrm{H}[n, k]$, where:

$$\mathrm{G}[n, k] = \underset{z_1, \ldots, z_{n-1}}{\operatorname{argmax}} p(x_1, \ldots, x_n, z_1, \ldots, z_{n-1}, z_n = s_k \mid \theta)$$

- Note that:
  - We optimize over all possible values $z_1, \ldots, z_{n-1}$.
  - However, **z_n is constrained to be equal to s_k**.
  - Values $x_1, \ldots, x_n$ are known, and thus they are fixed.

Problem 3: Find the Best Z

- Problem 3:
  - Input: an observation sequence $X = x_1, \ldots, x_N$
  - Output: the state sequence $Z = z_1, \ldots, z_N$ maximizing $p(Z \mid \theta, X)$, which is the same as maximizing $p(Z, X \mid \theta)$.
- Solution: (again) dynamic programming.
- Problem (n, k): Compute $\mathrm{G}[n, k]$ and $\mathrm{H}[n, k]$, where:

$$\mathrm{H}[n, k] = p(x_1, \ldots, x_n, \mathrm{G}[n, k] \mid \theta)$$

- $\mathrm{H}[n, k]$ is the joint probability of:
  - the first n observations $x_1, \ldots, x_n$
  - the sequence we stored in $\mathrm{G}[n, k]$

Problem 3: Find the Best Z

- Problem 3:
  - Input: an observation sequence $X = x_1, \dots, x_N$
  - Output: the state sequence $Z = z_1, \dots, z_N$ maximizing $p(Z \mid \theta, X)$, which is the same as maximizing $p(Z, X \mid \theta)$.
- Solution: (again) dynamic programming.
- Problem (n, k): Compute $\mathrm{G}[n, k]$ and $\mathrm{H}[n, k]$, where:

$$\mathrm{H}[n, k] = p(x_1, \dots, x_n, \mathrm{G}[n, k] \mid \theta)$$

- $\mathrm{H}[n, k]$ is the joint probability of:
  - the first n observations $x_1, \dots, x_n$
  - the sequence we stored in $\mathrm{G}[n, k]$
- $\mathrm{G}[n, k]$ stores a **sequence** of state values.
- $\mathrm{H}[n, k]$ stores a **number** (a probability).

The Viterbi Algorithm

- Problem (n, k): Compute $\mathrm{G}[n, k]$ and $\mathrm{H}[n, k]$, where:

$$\mathrm{G}[n, k] = \underset{z_1, \dots, z_{n-1}}{\operatorname{argmax}}\, p(x_1, \dots, x_n, z_1, \dots, z_{n-1}, z_n = s_k \,|\, \theta\,)$$

$$\mathrm{H}[n, k] = p(x_1, \dots, x_n, \mathrm{G}[n, k] \,|\, \theta)$$

- We use a dynamic programming algorithm to compute $\mathrm{G}[n, k]$ and $\mathrm{H}[n, k]$ for all n, k such that $1 \le n \le N, 1 \le k \le K$.
- This dynamic programming algorithm is called the **Viterbi algorithm**.

The Viterbi Algorithm - Initialization

- Problem $(1, k)$: Compute $\mathrm{G}[1, k]$ and $\mathrm{H}[1, k]$, where:

$$\mathrm{G}[1, k] = \underset{z_1, \dots, z_{n-1}}{\operatorname{argmax}} p(x_1, z_1 = s_k | \theta)$$

$$\mathrm{H}[1, k] = p(x_1, z_1 = s_k | \theta)$$

- What is $\mathrm{G}[1, k]$?
  - It is the **empty sequence**.
  - We must maximize over the previous states $z_1, \dots, z_{n-1}$, but $n = 1$, so there are no previous states.
- What is $\mathrm{H}[1, k]$?
  - $\mathrm{H}[1, k] = p(x_1, z_1 = s_k | \theta) = \pi_k \varphi_k(x_1)$.

The Viterbi Algorithm – Main Loop

- Problem (n, k): Compute $\mathrm{G}[n, k]$ and $\mathrm{H}[n, k]$, where:

$$\mathrm{G}[n,k] = \underset{z_1,\dots,z_{n-1}}{\operatorname{argmax}}\, p(x_1, \dots, x_n, z_1, \dots, z_{n-1}, z_n = s_k \mid \theta\,)$$

$$\mathrm{H}[n,k] = p(x_1, \dots, x_n, \mathrm{G}[n,k] \mid \theta)$$

- How do we find $\mathrm{G}[n, k]$?
- $\mathrm{G}[n,k] = G^*[n,k] \otimes s_j$ for some $G^*[n,k]$ and some j.
  - The last element of $\mathrm{G}[n, k]$ is some s_j.
  - Therefore, $\mathrm{G}[n, k]$ is formed by appending some s_j to some sequence $G^*[n, k]$

The Viterbi Algorithm – Main Loop

- Problem (n, k): Compute $\mathrm{G}[n, k]$ and $\mathrm{H}[n, k]$, where:

$$\mathrm{G}[n, k] = \underset{z_1,\ldots,z_{n-1}}{\operatorname{argmax}}\, p(x_1, \ldots, x_n, z_1, \ldots, z_{n-1}, z_n = s_k | \, \theta \,)$$

$$\mathrm{H}[n, k] = p(x_1, \ldots, x_n, \mathrm{G}[n, k] \mid \theta)$$

- $\mathrm{G}[n, k] = G^*[n, k] \otimes s_j$ for some $G^*[n, k]$ and some j.
- We can prove that $G^*[n, k] = G[n-1, j]$, for that j.
  - The proof is very similar to the proof that DTW finds the optimal alignment.
  - If $G[n-1, j]$ is better than $G^*[n, k]$, then $G[n-1, j] \otimes s_j$ is better than $\mathrm{G}[n, k]$, which is a contradiction.

The Viterbi Algorithm – Main Loop

- Problem (n, k): Compute $\mathrm{G}[n,k]$ and $\mathrm{H}[n,k]$, where:

$$\mathrm{G}[n,k] = \underset{z_1,\ldots,z_{n-1}}{\operatorname{argmax}}\, p(x_1, \ldots, x_n, z_1, \ldots, z_{n-1}, z_n = s_k | \, \theta \,)$$

$$\mathrm{H}[n,k] = p(x_1, \ldots, x_n, \mathrm{G}[n,k] \mid \theta)$$

- $\mathrm{G}[n,k] = G[n-1, j] \otimes s_j$ for some j.
- Let's define j^* as: $j^* = \underset{j=1,\ldots,K}{\operatorname{argmax}} \left\{ H[n-1, j] A_{j,k} \varphi_k(x_n) \right\}$
- Then:
  - $\mathrm{G}[n,k] = G[n-1, j^*] \otimes s_{j^*}$.
  - $\mathrm{H}[n,k] = \left\{ H[n-1, j^*] A_{j^*,k} \varphi_k(x_n) \right\}$

The Viterbi Algorithm – Output

- Our goal is to find the Z maximizing $p(Z \mid \theta, X)$, which is the same as finding the Z maximizing $p(Z, X \mid \theta)$.
- The Viterbi algorithm computes $\mathrm{G}[n, k]$ and $\mathrm{H}[n, k]$, where:

$$\mathrm{G}[n, k] = \underset{z_1, \dots, z_{n-1}}{\operatorname{argmax}}\, p(x_1, \dots, x_n, z_1, \dots, z_{n-1}, z_n = s_k \mid \theta)$$

$$\mathrm{H}[n, k] = p(x_1, \dots, x_n, \mathrm{G}[n, k] \mid \theta)$$

- Let's define k^* as: $k^* = \underset{k=1,\dots,K}{\operatorname{argmax}}\{H[N, k]\}$
- Then, **the $\boldsymbol{Z}$ maximizing $\boldsymbol{p(Z, X \mid \theta)}$ is $\boldsymbol{\mathrm{G}[N, k^*] \otimes s_{k^*}}$**

State Probabilities at Specific Times

- Problem 4:
  - Inputs: a trained HMM θ, and an observation sequence $X = x_1, \dots, x_N$
  - Output: an array γ, of size $N \times K$, where $\gamma[n, k] = p(z_n = s_k \mid X, \theta)$.
- In words: given an observation sequence X, we want to compute, for any moment in time n and any state s_k, the probability that the hidden state at that moment is s_k.

State Probabilities at Specific Times

- Problem 4:
  - Inputs: a trained HMM θ, and an observation sequence $X = x_1, \ldots, x_N$
  - Output: an array γ, of size $N \times K$, where $\gamma[n, k] = p(z_n = s_k \mid X, \theta)$.
- We have seen, in our solution for problem 2, that the **forward algorithm** computes an array α, of size $N \times K$, where $\alpha[n, k] = p(x_1, \ldots, x_n, z_n = s_k \mid \theta)$.
- We can also define another array β, of size $N \times K$, where $\beta[n, k] = p(x_{n+1}, \ldots, x_N \mid \theta, z_n = s_k)$.
- In words, $\beta[n, k]$ is the probability of all observations **after** moment n, given that the state at moment n is s_k.

State Probabilities at Specific Times

- Problem 4:
  - Inputs: a trained HMM θ, and an observation sequence $X = x_1, \dots, x_N$
  - Output: an array γ, of size $N \times K$, where $\gamma[n, k] = p(z_n = s_k \mid X, \theta)$.
- We have seen, in our solution for problem 2, that the **forward algorithm** computes an array α, of size $N \times K$, where $\alpha[n, k] = p(x_1, \dots, x_n, z_n = s_k \mid \theta)$.
- We can also define another array β, of size $N \times K$, where $\beta[n, k] = p(x_{n+1}, \dots, x_N \mid \theta, z_n = s_k)$.
- Then, $\alpha[n, k] * \beta[n, k] =$
  $p(x_1, \dots, x_n, z_n = s_k \mid \theta) * p(x_{n+1}, \dots, x_N \mid \theta, z_n = s_k) =$
  $p(X, z_n = s_k \mid \theta)$
- Therefore: $\gamma[n, k] = p(z_n = s_k \mid X, \theta) = \frac{\alpha[n,k] * \beta[n,k]}{P(X \mid \theta)}$

State Probabilities at Specific Times

- The **forward algorithm** computes an array α, of size $N \times K$, where $\alpha[n, k] = p(x_1, \ldots, x_n, z_n = s_k \mid \theta)$.
- We define another array β, of size $N \times K$, where $\beta[n, k] = p(x_{n+1}, \ldots, x_N \mid \theta, z_n = s_k)$.
- Then, $\gamma[n, k] = p(z_n = s_k \mid X, \theta) = \frac{\alpha[n,k] * \beta[n,k]}{P(X \mid \theta)}$
- In the above equation:
  - $\alpha[n, k]$ and $P(X \mid \theta)$ are computed by the forward algorithm.
  - We need to compute $\beta[n, k]$.
- We compute $\beta[n, k]$ in a way very similar to the forward algorithm, but going backwards this time.
  - The resulting combination of the forward and backward algorithms is called (not surprisingly) the **forward-backward algorithm**.

The Backward Algorithm

- We want to compute the values of an array β, of size $N \times K$, where $\beta[n, k] = p(x_{n+1}, \ldots, x_N \mid \theta, z_n = s_k)$.
- Again, we will use dynamic programming.
  - Problem (n, k) is to compute value $\beta[n, k]$.
- However, this time we start from the end of the observations.
- First, compute $\beta[N, k] = p(\{\} \mid \theta, z_N = s_k)$.
- In this case, the observation sequence $x_{n+1}, \ldots, x_N$ is empty, because $n = N$.
- What is the probability of an empty sequence?

Backward Algorithm - Initialization

- We want to compute the values of an array β, of size $N \times K$, where $\beta[n, k] = p(x_{n+1}, \ldots, x_N \mid \theta, z_n = s_k)$.
- Again, we will use dynamic programming.
  - Problem (n, k) is to compute value $\beta[n, k]$.
- However, this time we start from the end of the observations.
- First, compute $\beta[N, k] = p(\{\} | \theta, z_N = s_k)$.
- In this case, the observation sequence $x_{n+1}, \ldots, x_N$ is empty, because $n = N$.
- What is the probability of an empty sequence?
  - $\beta[N, k] = 1$

Backward Algorithm - Initialization

- Next: compute $\beta[N-1,k]$.

$$\beta[N-1,k] = p(x_N \mid \theta, z_{N-1} = s_k) =$$

$$\sum_{j=1}^{K} p\left(x_N, z_N = s_j \mid \theta, z_{N-1} = s_k\right) =$$

$$\sum_{j=1}^{K} \left\{ p\left(x_N \mid z_N = s_j\right) p\left(z_N = s_j \mid \theta, z_{N-1} = s_k\right) \right\}$$

Backward Algorithm - Initialization

- Next: compute $\beta[N-1, k]$.

$$\beta[N-1, k] = p(x_N | \theta, z_{N-1} = s_k)$$

$$= \sum_{j=1}^{K} \{ p\left(x_N \mid z_N = s_j\right) p(z_N = s_j | \theta, z_{N-1} = s_k) \}$$

$$= \sum_{j=1}^{K} \{ \varphi_j(x_N) A_{k,j} \}$$

Backward Algorithm – Main Loop

- Next: compute β$[n, k]$, for $n < N - 1$.

$$\beta[n, k] = p\left(x_{n+1,} \dots, x_N \mid \theta, z_n = s_k\right) =$$

$$\sum_{j=1}^{K} p\left(x_{n+1,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_n = s_k\right) =$$

$$\sum_{j=1}^{K} \{p\left(x_{n+1} \mid z_{n+1} = s_j\right) p\left(x_{n+2,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_{N-1} = s_k\right)\}$$

Backward Algorithm – Main Loop

- Next: compute β$[n, k]$, for $n < N - 1$.

$$\beta[n, k] = p\left(x_{n+1,} \dots, x_N | \, \theta, z_n = s_k\right) =$$

$$\sum_{j=1}^{K} \{p\left(x_{n+1} \,|\, z_{n+1} = s_j\right) p\left(x_{n+2,} \dots, x_N, z_{n+1} = s_j | \, \theta, z_{N-1} = s_k\right)\}$$

$$\sum_{j=1}^{K} \{\varphi_j(x_{n+1}) p\left(x_{n+2,} \dots, x_N | \, \theta, z_{n+1} = s_j\right) A_{k,j}\}$$

We will take a closer look at the last step…

Backward Algorithm – Main Loop

$$\sum_{j=1}^{K}\{p(x_{n+1} \mid z_{n+1} = s_j)p(x_{n+2,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_n = s_k)\} =$$

$$\sum_{j=1}^{K}\{\varphi_j(x_{n+1})p(x_{n+2,} \dots, x_N \mid \theta, z_{n+1} = s_j)A_{k,j}\}$$

- $p(x_{n+1} \mid z_{n+1} = s_j) = \varphi_j(x_{n+1})$, by definition of φ_j.

Backward Algorithm – Main Loop

$$\sum_{j=1}^{K}\{p(x_{n+1} \mid z_{n+1} = s_j)p\left(x_{n+2,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_n = s_k\right)\} =$$

$$\sum_{j=1}^{K}\{\varphi_j(x_{n+1})p\left(x_{n+2,} \dots, x_N \mid \theta, z_{n+1} = s_j\right)A_{k,j}\}$$

$$p\left(x_{n+2,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_n = s_k\right) =$$
$$p\left(x_{n+2,} \dots, x_N \mid \theta, z_n = s_k, z_{n+1} = s_j\right)p\left(z_{n+1} = s_j \mid z_n = s_k\right)$$

by application of chain rule: $P(A, B) = P(A|B)P(B)$,

which can also be written as $P(A, B \mid C) = P(A|B, C)P(B \mid C)$

Backward Algorithm – Main Loop

$$\sum_{j=1}^{K}\{p(x_{n+1} \mid z_{n+1} = s_j)p(x_{n+2,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_n = s_k)\} =$$

$$\sum_{j=1}^{K}\{\varphi_j(x_{n+1})p(x_{n+2,} \dots, x_N \mid \theta, z_{n+1} = s_j)A_{k,j}\}$$

$$p(x_{n+2,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_n = s_k) =$$
$$p(x_{n+2,} \dots, x_N \mid \theta, z_n = s_k, z_{n+1} = s_j)p(z_{n+1} = s_j \mid z_n = s_k) =$$
$$p(x_{n+2,} \dots, x_N \mid \theta, z_n = s_k, z_{n+1} = s_j)A_{k,j}$$

(by definition of $A_{k,j}$)

Backward Algorithm – Main Loop

$$\sum_{j=1}^{K}\{p(x_{n+1} \mid z_{n+1} = s_j)p(x_{n+2,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_n = s_k)\} =$$

$$\sum_{j=1}^{K}\{\varphi_j(x_{n+1})p(x_{n+2,} \dots, x_N \mid \theta, z_{n+1} = s_j)A_{k,j}\}$$

$$p(x_{n+2,} \dots, x_N, z_{n+1} = s_j \mid \theta, z_n = s_k) =$$
$$p(x_{n+2,} \dots, x_N \mid \theta, z_n = s_k, z_{n+1} = s_j)p(z_{n+1} = s_j \mid z_n = s_k) =$$
$$p(x_{n+2,} \dots, x_N \mid \theta, z_n = s_k, z_{n+1} = s_j)A_{k,j} =$$
$$p(x_{n+2,} \dots, x_N \mid \theta, z_{n+1} = s_j)A_{k,j}$$

($x_{n+2,} \dots, x_N$ are conditionally independent of z_n given z_{n+1})

Backward Algorithm – Main Loop

- In the previous slides, we showed that

$$\beta[n,k] = p\left(x_{n+1,} \dots, x_N \mid \theta, z_n = s_k\right) =$$

$$\sum_{j=1}^{K} \left\{ \varphi_j(x_{n+1}) p\left(x_{n+2,} \dots, x_N \mid \theta, z_{n+1} = s_j\right) A_{k,j} \right\}$$

- Note that $p\left(x_{n+2,} \dots, x_N \mid \theta, z_{n+1} = s_j\right)$ is $\beta[n+1,j]$.
- Consequently, we obtain the recurrence:

$$\beta[n,k] = \sum_{j=1}^{K} \left\{ \varphi_j(x_{n+1}) \beta[n+1,j] A_{k,j} \right\}$$

Backward Algorithm – Main Loop

$$\beta[n,k] = \sum_{j=1}^{K} \{\varphi_j(x_{n+1})\beta[n+1,j]A_{k,j}\}$$

- Thus, we can easily compute all values $\beta[n,k]$ using this recurrence.
- The key thing is to proceed in decreasing order of n, since values $\beta[n,*]$ depend on values $\beta[n+1,*]$.

The Forward-Backward Algorithm

- Inputs:
  - A trained HMM θ.
  - An observation sequence $X = x_1, \dots, x_N$
- Output:
  - An array γ, of size $N \times K$, where $\gamma[n, k] = p(z_n = s_k \mid X, \theta)$.
- Algorithm:
  - Use the forward algorithm to compute $\alpha[n, k]$.
  - Use the backward algorithm to compute $\beta[n, k]$.
  - For $n = 1$ to N:
    - For $k = 1$ to K:

$$\gamma[n, k] = p(z_n = s_k \mid X, \theta) = \frac{\alpha[n,k] * \beta[n,k]}{P(X \mid \theta)}$$

Problem 1: Training an HMM

- Goal: Estimate parameters $\theta = (\pi_k, A_{k,j}, \varphi_k)$.
- The training data can consist of multiple observation sequences $X_1, X_2, X_3, \ldots, X_M$.
- We denote the length of training sequence X_i as N_m.
- We denote the elements of each observation sequence X_m as:

$$X_m = (x_{m,1}, x_{m,2}, \ldots, x_{m,N_m})$$

- **Before** we start training, we need to decide on:
  - The number of states.
  - The transitions that we will allow.

Problem 1: Training an HMM

- While we are given $X_1, X_2, X_3, \dots, X_M$, we are **not** given the corresponding hidden state sequences $Z_1, Z_2, Z_3, \dots, Z_M$.
- We denote the elements of each hidden state sequence Z_m as:

$$Z_m = (z_{m,1}, z_{m,2}, \dots, z_{m,N_m})$$

- The training algorithm is called **Baum-Welch algorithm**.
- It is an Expectation-Maximization algorithm, similar to the algorithm we saw for learning Gaussian mixtures.

Expectation-Maximization

- When we wanted to learn a mixture of Gaussians, we had the following problem:
  - If we knew the probability of each object belonging to each Gaussian, we could estimate the parameters of each Gaussian.
  - If we knew the parameters of each Gaussian, we could estimate the probability of each object belonging to each Gaussian.
  - However, we know neither of these pieces of information.
- The EM algorithm resolved this problem using:
  - An initialization of Gaussian parameters to some random or non-random values.
  - A main loop where:
    - The current values of the Gaussian parameters are used to estimate new weights of membership of every training object to every Gaussian.
    - The current estimated membership weights are used to estimate new parameters (mean and covariance matrix) for each Gaussian.

Expectation-Maximization

- When we want to learn an HMM model using observation sequences as training data, we have the following problem:
  - If we knew, for each observation sequence, the probabilities of the hidden state values, we could estimate the parameters θ.
  - If we knew the parameters θ, we could estimate the probabilities of the hidden state values.
  - However, we know neither of these pieces of information.
- The Baum-Welch algorithm resolves this problem using EM:
  - At initialization, parameters θ are given (mostly) random values.
  - In the main loop, these two steps are performed repeatedly:
    - The current values of parameters θ are used to estimate new probabilities for the hidden state values.
    - The current probabilities for the hidden state values are used to estimate new values for parameters θ.

Baum-Welch: Initialization

- As we said before, before we start training, we need to decide on:
  - The number of states.
  - The transitions that we will allow.
- When we start training, we initialize $\theta = (\pi_k, A_{k,j}, \varphi_k)$ to random values, with these exceptions:
  - For any s_k that we do **not** want to allow to ever be an initial state, we set the corresponding π_k to 0.
    - The rest of the training will keep these π_k values always equal to 0.
  - For any s_k, s_j such that we do **not** want to allow transitions from s_k to s_j, we set the corresponding $A_{k,j}$ to 0.
    - The rest of the training will keep these $A_{k,j}$values always equal to 0.

Baum-Welch: Initialization

- When we start training, we initialize $\theta = (\pi_k, A_{k,j}, \varphi_k)$ to random values, with these exceptions:
  - For any s_k that we do **not** want to allow to ever be an initial state, we set the corresponding π_k to 0.
    - The rest of the training will keep these π_k values always equal to 0.
  - For any s_k, s_j such that we do **not** want to allow transitions from s_k to s_j, we set the corresponding $A_{k,j}$ to 0.
    - The rest of the training will keep these $A_{k,j}$values always equal to 0.
- These initial choices constrain the topology of the resulting HMM model.
  - For example, they can force the model to be fully connected, to be a forward model, or to be some other variation.

Baum-Welch: Expectation Step

- Before we start the expectation step, we have some current values for the parameters of $\theta = (\pi_k, A_{k,j}, \varphi_k)$.
- The goal of the expectation step is to use those values to estimate two arrays:
  - $\gamma[m, n, k] = p\big(x_{m,1}, \dots, x_{m,n}, z_{m,n} = s_k \mid \theta, X_m\big)$
    - This is the same $\gamma[n, k]$ that we computed earlier, using the forward-backward algorithm.
    - Here we just need to make the array three-dimensional, because we have multiple observation sequences X_m.
    - We can compute $\gamma[m,*,*]$ by running the forward-backward algorithm with θ and X_m as inputs.
  - $\xi[m, n, j, k] = p\big(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta, X_m\big)$

Baum-Welch: Expectation Step

- To facilitate our computations, we will also extend arrays α and β to three dimensions, in the same way that we extended array γ.
  - $\alpha[m, n, k] = p\left(x_{m,1}, \ldots, x_{m,n}, z_{m,n} = s_k \mid \theta\right)$
  - $\beta[m, n, k] = p\left(x_{m,n+1}, \ldots, x_{m,N_m} \mid \theta, z_{m,n} = s_k\right)$
- Where needed, values $\alpha[\mathrm{m},*,*]$, $\beta[\mathrm{m},*,*]$, $\gamma[\mathrm{m},*,*]$ can be computed by running the forward-backward algorithm with inputs θ and X_m.

Computing the ξ Values

- Using Bayes rule:

$$\xi[m,n,j,k] = p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta, X_m\right)$$

$$= \frac{p\left(X_m \mid \theta, z_{m,n-1} = s_j, z_{m,n} = s_k\right) p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta\right)}{p(X_m \mid \theta)}$$

- $p(X_m \mid \theta)$ can be computed using the forward algorithm.
- We will see how to simplify and compute:
  - $p\left(X_m \mid \theta, z_{m,n-1} = s_j, z_{m,n} = s_k\right)$
  - $p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta\right)$

Computing the ξ Values

$$\xi[m,n,j,k] = p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta, X_m\right)$$

$$= \frac{p\left(X_m \mid \theta, z_{m,n-1} = s_j, z_{m,n} = s_k\right) p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta\right)}{p(X_m \mid \theta)}$$

$$p\left(X_m \mid \theta, z_{m,n-1} = s_j, z_{m,n} = s_k\right) =$$
$$p\left(x_{m,1}, \ldots, x_{m,n-1} \mid \theta, z_{m,n-1} = s_j\right) p\left(x_{m,n}, \ldots, x_{m,N_m} \mid \theta, z_{m,n} = s_k\right)$$

- Why?
  - Earlier observations $x_{m,1}, \ldots, x_{m,n-1}$ are conditionally independent of state $z_{m,n}$ given state $z_{m,n-1}$.
  - Later observations $x_{m,n}, \ldots, x_{m,N_m}$ are conditionally independent of state $z_{m,n-1}$ given state $z_{m,n}$.

Computing the ξ Values

$$\xi[m,n,j,k] = p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta, X_m\right)$$

$$= \frac{p\left(X_m \mid \theta, z_{m,n-1} = s_j, z_{m,n} = s_k\right) p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta\right)}{p(X_m \mid \theta)}$$

$$p\left(X_m \mid \theta, z_{m,n-1} = s_j, z_{m,n} = s_k\right) =$$
$$p\left(x_{m,1}, \ldots, x_{m,n-1} \mid \theta, z_{m,n-1} = s_j\right) p\left(x_{m,n}, \ldots, x_{m,N_m} \mid \theta, z_{m,n} = s_k\right)$$

$$p\left(x_{m,n}, \ldots, x_{m,N_m} \mid \theta, z_{m,n} = s_k\right) =$$
$$p\left(x_{m,n} \mid \theta, z_{m,n} = s_k\right) p\left(x_{m,n+1}, \ldots, x_{m,N_m} \mid \theta, z_{m,n} = s_k\right) =$$
$$\varphi_k(x_{m,n}) \beta[m,n,k]$$

Computing the ξ Values

$$\xi[m, n, j, k] = p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta, X_m\right)$$

$$= \frac{p\left(X_m \mid \theta, z_{m,n-1} = s_j, z_{m,n} = s_k\right) p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta\right)}{p(X_m \mid \theta)}$$

$$p\left(X_m \mid \theta, z_{m,n-1} = s_j, z_{m,n} = s_k\right) =$$
$$p\left(x_{m,1}, \dots, x_{m,n-1} \mid \theta, z_{m,n-1} = s_j\right) \varphi_k\left(x_{m,n}\right) \beta[m, n, k] =$$

$$\frac{p\left(x_{m,1}, \dots, x_{m,n-1}, z_{m,n-1} = s_j \mid \theta\right)}{p\left(z_{m,n-1} = s_j \mid \theta\right)} \varphi_k\left(x_{m,n}\right) \beta[m, n, k] =$$

$$\frac{\alpha[m, n-1, j]}{\gamma[m, n-1, j]} \varphi_k\left(x_{m,n}\right) \beta[m, n, k]$$

Computing the ξ Values

$$\xi[m,n,j,k] =$$

$$\frac{\frac{\alpha[m,n-1,j]}{\gamma[m,n-1,j]}\varphi_k(x_{m,n})\beta[m,n,k]p(z_{m,n-1}=s_j, z_{m,n}=s_k |\, \theta)}{p(X_m \mid \theta)}$$

- We can further process $p(z_{m,n-1}=s_j, z_{m,n}=s_k |\, \theta)$:

$$p(z_{m,n-1}=s_j, z_{m,n}=s_k |\, \theta) =$$
$$p(z_{m,n}=s_k |\, \theta, z_{m,n-1}=s_j)p(z_{m,n-1}=s_j |\, \theta) =$$
$$A_{j,k}\gamma[m,n-1,j]$$

Computing the ξ Values

- So, we get:

$$\xi[m, n, j, k] =$$

$$\frac{\frac{\alpha[m, n-1, j]}{\gamma[m, n-1, j]} \varphi_k(x_{m,n}) \beta[m, n, k] A_{j,k}\, \gamma[m, n-1, j]}{p(X_m \mid \theta)} =$$

$$\frac{\alpha[m, n-1, j] \varphi_k(x_{m,n}) \beta[m, n, k] A_{j,k}}{p(X_m \mid \theta)}$$

Computing the ξ Values

- Based on the previous slides, the final formula for ξ is:

$$\xi[m,n,j,k] = \frac{\alpha[m,n-1,j]\varphi_k\left(x_{m,n}\right)\beta[m,n,k]A_{j,k}}{p(X_m \mid \theta)}$$

- This formula can be computed using the parameters θ and the algorithms we have covered:
  - $\alpha[m,n-1,j]$ is computed with the forward algorithm.
  - $\beta[m,n,k]$ is computed by the backward algorithm.
  - $p(X_m \mid \theta)$ is computed with the forward algorithm.
  - φ_k and $A_{j,k}$ are specified by θ.

Baum-Welch: Summary of E Step

- Inputs:
  - Training observation sequences $X_1, X_2, X_3, \ldots, X_M$.
  - Current values of parameters $\theta = (\pi_k, A_{k,j}, \varphi_k)$.
- Algorithm:
  - For $m = 1$ to M:
    - Run the forward-backward algorithm to compute:
      $\alpha[m,n,k]$ for n and k such that $1 \le n \le N_m$, $1 \le k \le K$.
      $\beta[m,n,k]$ for n and k such that $1 \le n \le N_m$, $1 \le k \le K$.
      $\gamma[m,n,k]$ for n and k such that $1 \le n \le N_m$, $1 \le k \le K$.
      $p(X_m \mid \theta)$
    - For n, j, k such that $1 \le n \le N_m$, $1 \le j,k \le K$:

$$\xi[m,n,j,k] = \frac{\alpha[m,n-1,j]\varphi_k\big(x_{m,n}\big)\beta[m,n,k]A_{j,k}}{p(X_m \mid \theta)}$$

Baum-Welch: Maximization Step

- Before we start the maximization step, we have some current values for:
  - $\gamma[m, n, k] = p\left(z_{m,n} = s_k \mid \theta, X_m\right)$
  - $\xi[m, n, j, k] = p\left(z_{m,n-1} = s_j, z_{m,n} = s_k \mid \theta, X_m\right)$
- The goal of the maximization step is to use those values to estimate new values for $\theta = (\pi_k, A_{k,j}, \varphi_k)$.
- In other words, we want to estimate new values for:
  - Initial state probabilities π_k.
  - State transition probabilities $A_{k,j}$.
  - Observation probabilities φ_k.

Baum-Welch: Maximization Step

- Initial state probabilities π_k are computed as:

$$\pi_k = \frac{\sum_{m=1}^{M} \gamma[m, 1, k]}{\sum_{m=1}^{M} \sum_{j=1}^{K} \gamma[m, 1, j]}$$

- In words, we compute for each k the ratio of these two quantities:
  - The sum of probabilities, over all observation sequences X_m, that state s_k was the initial state for X_m.
  - The sum of probabilities over all observation sequences X_m **<u>and all states</u>** s_j that statess_j was the initial state for X_m.

Baum-Welch: Maximization Step

- State transition probabilities $A_{k,j}$ are computed as:

$$A_{k,j} = \frac{\sum_{m=1}^{M} \sum_{n=2}^{N_m} \xi[m,n,k,j]}{\sum_{m=1}^{M} \sum_{n=2}^{N_m} \sum_{i=1}^{K} \xi[m,n,k,i]}$$

- In words, we compute, for each k, j the ratio of these two quantities:
  - The sum of probabilities, over all state transitions of all observation sequences X_m, that state s_k was followed by s_j.
  - The sum of probabilities, over all state transitions of all observation sequences X_m **and all states** s_i, that state s_k was followed by state s_i.

Baum-Welch: Maximization Step

- Computing the observation probabilities φ_k depends on how we model those probabilities.
- For example, we could choose:
  - Discrete probabilities.
  - Histograms.
  - Gaussians.
  - Mixtures of Gaussians.
  - …
- We will show how to compute distributions φ_k for the cases of:
  - Discrete probabilities.
  - Gaussians.

Baum-Welch: Maximization Step

- Computing the observation probabilities φ_k depends on how we model those probabilities.
- In all cases, the key idea is that we treat observation $x_{m,n}$ as partially assigned to distribution φ_k.
  - $\gamma[m,n,k] = p\left(z_{m,n} = s_k \mid \theta, X_m\right)$is the weight of the assignment of $x_{m,n}$ to φ_k.
- In other words:
  - We don't know what hidden state $x_{m,n}$ corresponds to, but we have computed, for each state s_k, the probability $\gamma[m,n,k]$ that $x_{m,n}$ corresponds to s_k.
  - Thus, $x_{m,n}$ influences our estimate of distribution φ_k with weight proportional to $\gamma[m,n,k]$.

Baum-Welch: Maximization Step

- Suppose that the observations are discrete, and come from a finite set $\boldsymbol{Y} = \{y_1, \ldots, y_R\}$.
  - In that case, each $x_{m,n}$ is an element of $\boldsymbol{Y}$.
- Define an auxiliary function $\mathrm{Eq}(x, y)$ as:

$$\mathrm{Eq}(x, y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ if } x \neq y \end{cases}$$

- Then, $\varphi_k(y_r) = p\big(x_{m,n} = y_r \mid z_{m,n} = s_k\big)$, and it can be computed as:

$$\varphi_k(y_r) = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \{\gamma[m, n, k] \mathrm{Eq}\big(x_{m,n}, y_r\big)\}}{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \gamma[m, n, k]}$$

Baum-Welch: Maximization Step

- If observations come from a finite set $\boldsymbol{Y} = \{y_1, \dots, y_R\}$:

$$\varphi_k(y_r) = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \{\gamma[m,n,k] \mathrm{Eq}\left(x_{m,n}, y_r\right)\}}{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \gamma[m,n,k]}$$

- The above formula can be seen as a weighted average of the times $x_{m,n}$ is equal to y_r, where the weight of each $x_{m,n}$ is the probability that $x_{m,n}$ corresponds to hidden state s_k.

Baum-Welch: Maximization Step

- Suppose that observations are vectors in $\mathbb{R}^D$, and that we model φ_k as a Gaussian distribution.
- In that case, for each φ_k we need to estimate a mean μ_k and a covariance matrix S_k.

$$\mu_k(y_r) = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \{\gamma[m,n,k] x_{m,n}\}}{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \gamma[m,n,k]}$$

- Thus, μ_k is a weighted average of values $x_{m,n}$, where the weight of each $x_{m,n}$ is the probability that $x_{m,n}$ corresponds to hidden state s_k.

Baum-Welch: Maximization Step

- Suppose that observations are vectors in $\mathbb{R}^D$, and that we model φ_k as a Gaussian distribution.
- In that case, for each φ_k we need to estimate a mean μ_k and a covariance matrix S_k.

$$\mu_k = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \{\gamma[m,n,k] x_{m,n}\}}{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \gamma[m,n,k]}$$

$$S_k = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \left\{\gamma[m,n,k] \left(x_{m,n} - \mu_k\right)\left(x_{m,n} - \mu_k\right)^T\right\}}{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \gamma[m,n,k]}$$

Baum-Welch: Summary of M Step

$$\pi_k = \frac{\sum_{m=1}^{M} \gamma[m,1,k]}{\sum_{m=1}^{M} \sum_{j=1}^{K} \gamma[m,1,j]} \qquad A_{k,j} = \frac{\sum_{m=1}^{M} \sum_{n=2}^{N_m} \xi[m,n,k,j]}{\sum_{m=1}^{M} \sum_{n=2}^{N_m} \sum_{i=1}^{K} \xi[m,n,k,i]}$$

- For discrete observations:

$$\varphi_k(y_r) = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \{\gamma[m,n,k] \mathrm{Eq}\left(x_{m,n}, y_r\right)\}}{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \gamma[m,n,k]}$$

Baum-Welch: Summary of M Step

$$\pi_k = \frac{\sum_{m=1}^{M} \gamma[m, 1, k]}{\sum_{m=1}^{M} \sum_{j=1}^{K} \gamma[m, 1, j]} \qquad A_{k,j} = \frac{\sum_{m=1}^{M} \sum_{n=2}^{N_m} \xi[m, n, k, j]}{\sum_{m=1}^{M} \sum_{n=2}^{N_m} \sum_{i=1}^{K} \xi[m, n, k, i]}$$

- For Gaussian observation distributions: φ_k is a Gaussian with mean μ_k and covariance matrix S_k, where:

$$\mu_k = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \{\gamma[m, n, k] x_{m,n}\}}{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \gamma[m, n, k]}$$

$$S_k = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \left\{\gamma[m, n, k] \left(x_{m,n} - \mu_k\right) \left(x_{m,n} - \mu_k\right)^T\right\}}{\sum_{m=1}^{M} \sum_{n=1}^{N_m} \gamma[m, n, k]}$$

Baum-Welch Summary

- Initialization:
  - Initialize $\theta = (\pi_k, A_{k,j}, \varphi_k)$ with random values, but set to 0 values π_k and $A_{k,j}$ to specify the network topology.
- Main loop:
  - E-step:
    - Compute all values $p(X_m \mid \theta), \alpha[m,n,k], \beta[m,n,k], \gamma[m,n,k]$ using the forward-backward algorithm.
    - Compute all values $\xi[m,n,j,k] = \frac{\alpha[m,n-1,j]\varphi_k(x_{m,n})\beta[m,n,k]A_{j,k}}{p(X_m \mid \theta)}$.
  - M-step:
    - Update $\theta = (\pi_k, A_{k,j}, \varphi_k)$, using the values $\gamma[m,n,k]$ and $\xi[m,n,j,k]$ computed at the E-step.

Hidden Markov Models - Recap

- HMMs are widely used to model temporal sequences.
- In HMMs, an observation sequence corresponds to a sequence of hidden states.
- An HMM is defined by parameters $\theta = (\pi_k, A_{k,j}, \varphi_k)$, and defines a joint probability $p(X, Z \mid \theta)$.
- An HMM can be used as a generative model, to produce synthetic samples from distribution $p(X, Z \mid \theta)$.
- HMMs can be used for Bayesian classification:
  - One HMM θ_c per class.
  - Find the class c that maximizes $p(\theta_c \mid X)$.

Hidden Markov Models - Recap

- HMMs can be used for Bayesian classification:
  - One HMM θ_c per class.
  - Find the class c that maximizes $p(\theta_c \mid X)$, using probabilities $p(X \mid \theta_c)$ calculated by the forward algorithm.
- HMMs can be used to find the most likely hidden state sequence Z for observation sequence X.
  - This is done using the Viterbi algorithm.
- HMMs can be used to find the most likely hidden state z_n for a specific value x_n of an observation sequence X.
  - This is done using the forward-backward algorithm.
- HMMs are trained using the Baum-Welch algorithm, which is an Expectation-Maximization algorithm.