10. Hidden Markov Models
Overview

Goal:

find a sequence of classes corresponding to a sequence of observations/features

Examples:

part-of-speech tagger, speech recognition, …

Topics:

Evaluation
Decoding
Learning
HMMs in Speech Recognition

Feature vectors: \( x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6 \rightarrow x_7 \rightarrow x_8 \rightarrow x_9 \)

Sounds (phonems): \( s \rightarrow s \rightarrow a \rightarrow a \rightarrow a \rightarrow g \rightarrow e \rightarrow n \rightarrow n \)

Classification has to be done in the context of the neighboring feature vectors.
Classes (e.g. sounds) may “repeat”
Part-Of-Speech Tags

- **45 Tags**

**Examples:**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordinating Conjunction</td>
<td>and, but, or</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
<td>one, two, three</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
<td>a. the</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
<td>yellow</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, sing. or mass</td>
<td>province</td>
</tr>
<tr>
<td>NNP</td>
<td>Proper noun, singular</td>
<td>IBM</td>
</tr>
<tr>
<td>RB</td>
<td>Adverb</td>
<td>quickly, never</td>
</tr>
<tr>
<td>VB</td>
<td>Verb, base form</td>
<td>eat</td>
</tr>
<tr>
<td>VBD</td>
<td>Verb, past tense</td>
<td>ate</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Part-Of-Speech Tagging

Sentence: Next you flour the pan .

Tags: JJ ─ PRP ─ VBP ─ DT ─ NN ─ .
Image Annotation

tiger, grass, sky

Difference:
No linear ordering
Formal Approach to HMMs

Bayes classifier for hidden sequence:

\[
(\hat{\pi}_1, \hat{\pi}_2...\hat{\pi}_N) = \arg \max_{\pi_1, \pi_2...\pi_N} \left[ P(x_1, x_2...x_N \mid \pi_1, \pi_2...\pi_N) P(\pi_1, \pi_2...\pi_N) \right]
\]

\[
= \arg \max_{\pi_1, \pi_2...\pi_N} \left[ P(x_1, x_2...x_N, \pi_1, \pi_2...\pi_N) \right]
\]
Formal Approach to HMMs

\[ P(x_1, x_2 \ldots x_N, \pi_1, \pi_2 \ldots \pi_N) \]

\[ = P(x_1, x_2 \ldots x_N, \pi_2 \ldots \pi_N \mid \pi_1)P(\pi_1) \]

\[ = P(x_2 \ldots x_N, \pi_2 \ldots \pi_N \mid x_1, \pi_1)P(x_1 \mid \pi_1)P(\pi_1) \]

\[ = P(x_2 \ldots x_N, \pi_3 \ldots \pi_N \mid x_1, \pi_2, \pi_1)P(\pi_2 \mid x_1, \pi_1)P(x_1 \mid \pi_1)P(\pi_1) \]

\[ = \ldots \]

\[ = \prod_{i=1}^{N} P(x_i \mid x_1, \ldots, x_{i-1}, \pi_1, \ldots, \pi_i)P(\pi_i \mid x_1, \ldots, x_{i-1}, \pi_1 \ldots \pi_{i-1}) \]
Formal Approach to HMMs

\[ P(x_1, \ldots, x_N, \pi_1, \ldots, \pi_N) \]

\[ = \prod_{i=1}^{N} P(x_i \mid x_1, \ldots, x_{i-1}, \pi_1, \ldots, \pi_i) P(\pi_i \mid x_1, \ldots, x_{i-1}, \pi_1 \ldots \pi_{i-1}) \]

Possible approximation for emission probabilities:
\[ P(x_i \mid x_1, \ldots, x_{i-1}, \pi_1, \ldots, \pi_i) \approx P(x_i \mid \pi_i) \]

Possible approximation for transition probabilities:
\[ P(\pi_i \mid x_1, \ldots, x_{i-1}, \pi_1 \ldots \pi_{i-1}) \approx P(\pi_i \mid \pi_{i-1}) \]

or
\[ P(\pi_i \mid x_1, \ldots, x_{i-1}, \pi_1 \ldots \pi_{i-1}) \approx P(\pi_i \mid \pi_{i-2}, \pi_{i-1}) \]

or \ldots
Example: The Dishonest Casino

A casino has two dice:

- **Fair die**
  \[ P(1) = P(2) = P(3) = P(5) = P(6) = \frac{1}{6} \]

- **Loaded die**
  \[ P(1) = P(2) = P(3) = P(5) = \frac{1}{10} \]
  \[ P(6) = \frac{1}{2} \]

Casino player switches back-&-forth between fair and loaded die once every 20 turns on average

**Game:**

1. You bet $1
2. You roll (always with a fair die)
3. Casino player rolls (maybe with fair die, maybe with loaded die)
4. Highest number wins $2
The dishonest casino model

FAIR

loaded

LOADED

P(1|F) = 1/6
P(2|F) = 1/6
P(3|F) = 1/6
P(4|F) = 1/6
P(5|F) = 1/6
P(6|F) = 1/6

P(1|L) = 1/10
P(2|L) = 1/10
P(3|L) = 1/10
P(4|L) = 1/10
P(5|L) = 1/10
P(6|L) = 1/2
Question # 1 – Evaluation

**GIVEN**

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

**QUESTION**

How likely is this sequence, given our model of how the casino works?

This is the **EVALUATION** problem in HMMs
Question # 2 – Decoding

GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

QUESTION

What portion of the sequence was generated with the fair die, and what portion with the loaded die?

This is the DECODING question in HMMs
Question # 3 – Learning

GIVEN

A sequence of rolls by the casino player

1245526462146146136136661664661636616366163616515615115146123562344

QUESTION

How “loaded” is the loaded die? How “fair” is the fair die? How often does the casino player change from fair to loaded, and back?

This is the LEARNING question in HMMs
Definition of a hidden Markov model

**Definition:** A hidden Markov model (HMM)

- **Alphabet of possible observations** \( \Sigma = \{ b_1, b_2, \ldots, b_M \} \)
- **Set of states** \( Q = \{ 1, \ldots, K \} \)
- **Transition probabilities** between any two states

\[ a_{ij} = \text{transition prob from state } i \text{ to state } j \]

\[ a_{i1} + \ldots + a_{iK} = 1, \quad \text{for all states } i = 1 \ldots K \]

- **Start probabilities** \( a_{i0} \)

\[ a_{i1} + \ldots + a_{iK} = 1 \]

- **Emission probabilities** within each state

\[ e_i(b) = P( x_i = b | \pi_i = k ) \]

\[ e_i(b_1) + \ldots + e_i(b_M) = 1, \quad \text{for all states } i = 1 \ldots K \]
A Hidden Markov Model is memory-less

At each time step $t$,
the only thing that affects future states
is the current state $\pi_t$

\[
P(\pi_{t+1} = k \mid \text{“whatever happened so far”}) = P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t) = P(\pi_{t+1} = k \mid \pi_t)\]
A path of a sequence

Given a sequence $x = x_1 \ldots x_N$, a path of $x$ is a sequence of states $\pi = \pi_1, \ldots, \pi_N$. 

Diagram showing transitions between states with labels $1, 2, K$ and corresponding observations $x_1, x_2, x_3, x_K$. 

Diagram with states connected by arrows representing transitions, with states $1, 1, 1$ and $2, 2, K, K$.
Likelihood of a path
Example: the dishonest casino

Let the sequence of rolls be:

\[ x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4 \]

Then, what is the likelihood of

\[ \pi = \text{Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair, Fair} \]

(say initial probs \( a_0^{\text{Fair}} = \frac{1}{2} \), \( a_0^{\text{Loaded}} = \frac{1}{2} \))

\[ \frac{1}{2} \times P(1 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) P(2 \mid \text{Fair}) P(\text{Fair} \mid \text{Fair}) \ldots P(4 \mid \text{Fair}) = \]

\[ \frac{1}{2} \times (\frac{1}{6})^{10} \times (0.95)^9 = 0.00000000521158647211 = 5.21 \times 10^{-9} \]
Example: the dishonest casino

So, the likelihood the die is fair in all this run is just $5.21 \times 10^{-9}$

OK, but what is the likelihood of

$$\pi = \text{Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded, Loaded?}$$

$$\frac{1}{2} \times P(1 \mid \text{Loaded}) P(\text{Loaded, Loaded}) \ldots P(4 \mid \text{Loaded}) =$$

$$\frac{1}{2} \times (1/10)^8 \times (1/2)^2 (0.95)^9 = .00000000078781176215 = 7.9 \times 10^{-10}$$

Therefore, it is after all 6.61 times more likely that the die is fair all the way, than that it is loaded all the way.
Example: the dishonest casino

Let the sequence of rolls be:

\[ x = 1, 6, 6, 5, 6, 2, 6, 6, 3, 6 \]

Now, what is the likelihood \( \pi = F, F, \ldots, F \)?

\[ \frac{1}{2} \times (1/6)^{10} \times (0.95)^9 = 0.5 \times 10^{-9}, \text{ same as before} \]

What is the likelihood \( \pi = L, L, \ldots, L \)?

\[ \frac{1}{2} \times (1/10)^4 \times (1/2)^6 (0.95)^9 = 0.00000049238235134735 = 0.5 \times 10^{-7} \]

So, it is 100 times more likely the die is loaded
Problem 1: Decoding

- Find the best path of a sequence
Decoding

Given \( x = x_1x_2……x_N \)

We want to find \( \pi = \pi_1, \ldots, \pi_N \), such that \( P[ x, \pi ] \) is maximized

\( \pi^* = \arg\max_\pi P[ x, \pi ] \)

Recursive scheme: dynamic programming

Let \( V_k(i) = \max_{\pi_1, \ldots, \pi_{i-1}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k] \)

\[ = \text{Probability of most likely sequence of states ending at state } \pi_i = k \]
Decoding – main idea

Given that for all states $k$, and for a fixed position $i$,

$$V_k(i) = \max_{\{\pi_1, \ldots, \pi_{i-1}\}} P[x_1 \ldots x_i, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$$

What is $V_k(i+1)$?

From definition,

$$V_{i+1} = \max_{\{\pi_1, \ldots, i\}} P[ x_i \ldots x_i, \pi_i, \ldots, \pi_{i-1}, x_{i+1}, \pi_{i+1} = l ]$$

$$= \max_{\{\pi_1, \ldots, i\}} P(x_{i+1}, \pi_{i+1} = l \mid x_i \ldots x_i, \pi_i, \ldots, \pi_i) P[x_i \ldots x_i, \pi_i, \ldots, \pi_i]$$

$$= \max_{\{\pi_1, \ldots, i\}} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i) P[x_i \ldots x_i, \pi_i, \ldots, \pi_i]$$

$$= \max_k P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k) \max_{\{\pi_1, \ldots, i\}} P[x_i \ldots x_i, \pi_i, \ldots, \pi_i, x_i, \pi_i = k]$$

$$= e_l(x_{i+1}) \max_k a_{kl} V_k(i)$$
The Viterbi Algorithm

Input: $x = x_1, \ldots, x_N$

Initialization:
\[ V_0(0) = 1 \quad (0 \text{ is the imaginary first position}) \]
\[ V_k(0) = 0, \text{ for all } k > 0 \]

Iteration:
\[ V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1) \]
\[ \text{Ptr}_j(i) = \arg\max_k a_{kj} V_k(i-1) \]

Termination:
\[ P(x, \pi^*) = \max_k V_k(N) \]
\[ \pi_N^* = \arg\max_k V_k(N) \]

Traceback:
\[ \pi_{i-1}^* = \text{Ptr}_{\pi_i}(i) \]
The Viterbi Algorithm

Similar to “aligning” a set of states to a sequence

**Time:**

$O(K^2N)$

**Space:**

$O(KN)$
Underflows are a significant problem

$$P[ x_1, \ldots, x_i, \pi_1, \ldots, \pi_i ] = a_0 \pi_1 a_1 \pi_2 \ldots a_i e^{\pi_1(x_1)} \ldots e^{\pi_i(x_i)}$$

These numbers become extremely small – underflow

**Solution:** Take the logs of all values

$$V(i) = \log e^{k(x_i)} + \max_k [ V(k-1) + \log a_k ]$$
Problem 2: Evaluation

• Find the likelihood a sequence is generated by the model
Generating a sequence by the model

Given a HMM, we can generate a sequence of length n as follows:

1. Start at state $\pi_1$ according to prob $a_{0\pi_1}$
2. Emit letter $x_1$ according to prob $e_{\pi_1}(x_1)$
3. Go to state $\pi_2$ according to prob $a_{x_1\pi_2}$
4. ... until emitting $x_n$
Evaluation

We will develop algorithms that allow us to compute:

\[
P(x) \quad \text{Probability of } x \text{ given the model}
\]

\[
P(x_i \ldots x_j) \quad \text{Probability of a substring of } x \text{ given the model}
\]

\[
P(\pi_i = k \mid x) \quad \text{Probability that the } i^{th} \text{ state is } k, \text{ given } x
\]
The Forward Algorithm

We want to calculate

\[ P(x) = \text{probability of } x, \text{ given the HMM} \]

Sum over all possible ways of generating \( x \):

\[ P(x) = \sum_{\pi} P(x, \pi) = \sum_{\pi} P(x | \pi) P(\pi) \]

To avoid summing over an exponential number of paths \( \pi \), define

\[ f_k(i) = P(x_1 \ldots x_i, \pi_i = k) \text{ (the forward probability)} \]
The Forward Algorithm – derivation

Define the forward probability:

\[ f_i(i) = P(x_1 \ldots x_i, \pi_i = l) \]

\[ = \sum_{\pi_1 \ldots \pi_{i-1}} P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, \pi_i = l) e_i(x_i) \]

\[ = \sum_k \sum_{\pi_1 \ldots \pi_{i-2}} P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-2}, \pi_{i-1} = k) a_{k l} e_i(x_i) \]

\[ = e_i(x_i) \sum_k f_k(i-1) a_{k l} \]
The Forward Algorithm

We can compute $f_k(i)$ for all $k, i$, using dynamic programming!

**Initialization:**
- $f_0(0) = 1$
- $f_k(0) = 0$, for all $k > 0$

**Iteration:**
- $f_l(i) = e_l(x_i) \sum_k f_k(i-1) a_{kl}$

**Termination:**
- $P(x) = \sum_k f_k(N) a_{k0}$

Where, $a_{k0}$ is the probability that the terminating state is $k$ (usually $= a_{0k}$)
Relation between Forward and Viterbi

**VITERBI**

**Initialization:**
- $V_0(0) = 1$
- $V_k(0) = 0$, for all $k > 0$

**Iteration:**
- $V_i(i) = e_j(x_i) \max k V_k(i-1) a_{ki}$

**Termination:**
- $P(x, \pi^*) = \max V_i(N)$

**FORWARD**

**Initialization:**
- $f_0(0) = 1$
- $f_k(0) = 0$, for all $k > 0$

**Iteration:**
- $f(i) = e_i(x_i) \sum k f_k(i-1) a_{ki}$

**Termination:**
- $P(x) = \sum f_k(N) a_{k0}$
Motivation for the Backward Algorithm

We want to compute

\[ P(\pi_i = k \mid x), \]

the probability distribution on the \( i^{th} \) position, given \( x \)

We start by computing

\[
P(\pi_i = k, x) = P(x_1 \ldots x_i, \pi_i = k, x_{i+1} \ldots x_N) = P(x_1 \ldots x_i, \pi_i = k) \cdot P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, \pi_i = k)
\]

\[
= P(x_1 \ldots x_i, \pi_i = k) \cdot P(x_{i+1} \ldots x_N \mid \pi_i = k)
\]

*Forward, \( f(i) \)*
*Backward, \( b(i) \)*
The Backward Algorithm – derivation

Define the backward probability:

\[ b_k(i) = P(x_{i+1} \ldots x_N \mid \pi_i = k) \]

\[ = \sum_{\pi_{i+1} \ldots \pi_N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k) \]

\[ = \sum_{l} \sum_{\pi_{i+1} \ldots \pi_N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1} = l, \pi_{i+2}, \ldots, \pi_N \mid \pi_i = k) \]

\[ = \sum_{l} e_l(x_{i+1}) a_{k,l} \sum_{\pi_{i+1} \ldots \pi_N} P(x_{i+2}, \ldots, x_N, \pi_{i+2}, \ldots, \pi_N \mid \pi_{i+1} = l) \]

\[ = \sum_{l} e_l(x_{i+1}) a_{k,l} b_{l(i+1)} \]
The Backward Algorithm

We can compute $b_k(i)$ for all $k, i$, using dynamic programming

Initialization:

$$b_k(N) = a_{k0}, \text{ for all } k$$

Iteration:

$$b_k(i) = \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1)$$

Termination:

$$P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)$$
Computational Complexity

What is the running time, and space required, for Forward, and Backward?

Time: $O(K^2 N)$
Space: $O(KN)$
We can now calculate

\[
P(\pi_i = k | x) = \frac{f_k(i) b_k(i)}{P(x)}
\]

Then, we can ask

What is the most likely state at position \(i\) of sequence \(x\):

Define \(\pi^*\) by Posterior Decoding:

\[
\pi_i^* = \arg\max_k P(\pi_i = k | x)
\]
Problem 3: Learning

• Re-estimate the parameters of the model based on training data
Two learning scenarios

1. Estimation when the “right answer” is known

**Example:**

**GIVEN:** the casino player allows us to observe him one evening, as he changes dice and produces 10,000 rolls

1. Estimation when the “right answer” is unknown

**Example:**

**GIVEN:** 10,000 rolls of the casino player, but we don’t see when he changes dice

**QUESTION:**

Update the parameters $\theta$ of the model to maximize $P(x|\theta)$
1. When the right answer is known

Given \( x = x_1 \ldots x_N \)
for which the true \( \pi = \pi_1 \ldots \pi_N \) is known,

**Define:**

\[
A_{ki} = \text{# times } k \rightarrow l \text{ transition occurs in } \pi \\
E_k(b) = \text{# times state } k \text{ in } \pi \text{ emits } b \text{ in } x
\]

We can show that the maximum likelihood parameters \( \theta \) are:

\[
a_{ki} = \frac{A_{ki}}{\sum_i A_{ki}} \\
e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)}
\]
1. When the right answer is known

**Intuition:** When we know the underlying states,
Best estimate is the average frequency of transitions & emissions that occur in the training data

**Drawback:**
Given little data, there may be **overfitting**:
P(x|θ) is maximized, but θ is unreasonable
0 probabilities – VERY BAD

**Example:**
Given 10 casino rolls, we observe
x = 2, 1, 5, 6, 1, 2, 3, 6, 2, 3
Then:
\[ a_{FF} = 1; \quad a_{FL} = 0 \]
\[ e_F(1) = e_F(3) = .2; \]
\[ e_F(2) = .3; \quad e_F(4) = 0; \quad e_F(5) = e_F(6) = .1 \]
Pseudocounts

Solution for small training sets:

Add pseudocounts

\[ A_{kl} = \# \text{ times } k \rightarrow l \text{ transition occurs in } \pi + r_{kl} \]
\[ E_k(b) = \# \text{ times state } k \text{ in } \pi \text{ emits } b \text{ in } x + r_k(b) \]

\( r_{kl}, \, r_k(b) \) are pseudocounts representing our prior belief

Larger pseudocounts \( \Rightarrow \) Strong prior belief

Small pseudocounts (\( \varepsilon < 1 \)): just to avoid 0 probabilities
Pseudo counts

**Example:** dishonest casino

We will observe player for one day, 500 rolls

Reasonable pseudocounts:

\[
\begin{align*}
    r_{0F} &= r_{0L} = r_{F0} = r_{L0} = 1; \\
    r_{FL} &= r_{LF} = r_{FF} = r_{LL} = 1; \\
    r_{F(1)} &= r_{F(2)} = \ldots = r_{F(6)} = 20 \quad \text{(strong belief fair is fair)} \\
    r_{F(1)} &= r_{F(2)} = \ldots = r_{F(6)} = 5 \quad \text{(wait and see for loaded)}
\end{align*}
\]

Above #s pretty arbitrary – assigning priors is an art
Alternative Smoothing Methods

- Absolute discounting
- Linear smoothing
- Good-Turing
- Dirichlet Prior

How would you learn if the right sequence is unknown
2. When the right answer is unknown

Principle: EXPECTATION MAXIMIZATION

Starting with our best guess of a model M, parameters \( \theta \):

Given \( x = x_1 \ldots x_N \)

for which the true \( \pi = \pi_1 \ldots \pi_N \) is unknown,

We can get to a provably more likely parameter set \( \theta \)

1. Estimate \( A_{kl}, E_k(b) \) in the training data
2. Update \( \theta \) according to \( A_{kl}, E_k(b) \)
3. Repeat 1 & 2, until convergence
Estimating new parameters

To estimate $A_{ki}$:

At each position $i$ of sequence $x$,

Find probability transition $k \rightarrow l$ is used:

$$P(\pi_i = k, \pi_{i+1} = l \mid x) = \frac{1}{P(x)} \times P(\pi_i = k, \pi_{i+1} = l, x_{i} \ldots x_{n}) = \frac{Q}{P(x)}$$

where $Q = P(x_1 \ldots x_l, \pi_i = k, \pi_{i+1} = l, x_{i+1} \ldots x_n) = P(\pi_{i+1} = l, x_{i+1} \ldots x_n \mid \pi_i = k) P(x_1 \ldots x_l, \pi_i = k) = b_{l}(i+1) e_{l}(x_{i+1}) a_{kl} f_k(i)$

So:

$$P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \frac{f_k(i) a_{kl} e_{l}(x_{i+1}) b_{l}(i+1)}{P(x \mid \theta)}$$
Estimating new parameters

So,

\[ A_{kl} = \sum_i P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \sum_i \frac{P(x \mid \theta)}{P(x \mid \theta)} \]

Similarly,

\[ E_k(b) = \left[ \frac{1}{P(x)} \right] \sum \{ i \mid x_i = b \} f_k(i) b_k(i) \]
The Baum-Welch Algorithm

**Initialization:**
Pick the best-guess for model parameters
(or arbitrary)

**Iteration:**
Forward
Backward
Calculate $A_{k\mid l}$, $E_k(b)$
Calculate new model parameters $a_{k\mid l}$, $e_k(b)$
Calculate new log-likelihood $P(x \mid \theta)$

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

Until $P(x \mid \theta)$ does not change much
The Baum-Welch Algorithm – comments

Time Complexity:

\# \text{iterations} \times \mathcal{O}(K^2N)

- Guaranteed to increase the log likelihood of the model

\[ P(\theta \mid x) = P(x, \theta) / P(x) = P(x \mid \theta) / (P(x) P(\theta)) \]

- Not guaranteed to find globally best parameters

Converges to local optimum, depending on initial conditions

- Too many parameters / too large model: Overtraining
Alternative: Viterbi Training

**Initialization:** linear alignment

**Iteration:**
- Perform Viterbi, to find $\pi'$
- Calculate $A_{k|l}$, $E_k(b)$ according to $\pi'$ + pseudocounts
- Calculate the new parameters $a_{k|l}$, $e_k(b)$

Until convergence

**Notes:**
- Does not maximize $P(x | \theta)$
- In general, worse performance than Baum-Welch
- Convenient – when using Viterbi anyway for decoding, no need to implement additional procedures (Forward, Backward)!!
Summary

• Evaluation
• Decoding
• Learning
• Modeling of the emission probabilities
• Applications:
  • Speech recognition, part-of-speech tagging, image recognition