Lecture 07
Hidden Markov Models
Part II

Module II: Modeling genes and gene expression

• Computational Foundations
  – Hidden Markov Models (HMMs): Central tool in CS
  – Decoding, evaluation, parsing, likelihood, scoring
  – Unsupervised Learning: Expectation Maximization
  – Supervised learning: generative/discriminative models

• Biological frontiers:
  – PS2: Modeling conservation, GC content, CpG islands
  – L6/L7: Genome annotation and parsing
  – L8: Gene expression analysis: cluster genes/conditions
  – L9: Regulatory motif discovery: EM, gibbs sampling, info

Goals for today: HMMs, part II

• Review: Three algorithms from last time
  – Markov Chains and Hidden Markov Models
  – Increasing the ‘state’ space / adding memory
  – Calculating likelihoods $P(x, \pi)$
  – Viterbi algorithm: Find $\pi^* = \arg\max_\pi P(x, \pi)$

• Counting over all paths
  – Forward algorithm: Find $P(x)$, over all paths
  – Model comparison: ex: “CpGs” vs. “Gs and Cs”

• Posterior decoding: Another way of ‘parsing’
  – Find most likely state $k_i$, overall all possible paths

• Learning (ML training, Baum-Welch, Viterbi training)
  – Supervised: Find $e_i(.)$ and $a_{ij}$ given labeled sequence
  – Unsupervised: given only $x \rightarrow$ annotation + params

Markov Chains & Hidden Markov Models

• Markov Chain
  – $Q$: states
  – $p$: initial state probabilities
  – $A$: transition probabilities

• HMM
  – $Q$: states
  – $V$: observations
  – $p$: initial state probabilities
  – $A$: transition probabilities
  – $E$: emission probabilities

HMM nomenclature for this course

Transitions: $a_{jk}=P(\pi_i=l|\pi_{i-1}=k)$
Transition probability from state $k$ to state $l$

Emissions: $e_k(x_i)=P(x_i|\pi_i=k)$
Emission probability of symbol $x_i$ from state $k$

• Vector $x =$ Sequence of observations
• Vector $\pi =$ Hidden path (sequence of hidden states)
• Transition matrix $A=a_{jk}$=probability of $k \rightarrow l$ state transition
• Emission vector $E=e_k(x_i)$ = prob. of observing $x_i$ from state $k$
• Bayes’ rule: Use $P(x_i|\pi_i=k)$ to estimate $P(\pi_i=k|x)$
1. Scoring probability of a path + sequence

Multiply emissions, transitions

2. Viterbi decoding

$\pi^* = \text{argmax}_\pi P(x, \pi)$

Most likely path

Path containing the most likely state at any time point.

3. Posterior decoding

$\pi^* = \{ \pi_i | \pi_i = \text{argmax}_k \sum P(\pi_i = k | x) \}$

Path containing the most likely state at any time point.

4. Supervised learning, given $\Lambda$

$\Lambda^* = \text{argmax}_\Lambda P(x, \pi | \Lambda)$

5. Unsupervised learning.

$\Lambda^* = \text{argmax}_\Lambda \sum P(x, \pi | \Lambda)$

6. Viterbi training, best path

Baum-Welch training, over all paths

Example: One particular P vs. B assignment

$P = P(G | B)P(B | R_0)P(C | B)P(R_0 | R)P(A | B)P(P | R) \ldots P(C | R)$

$= (0.85)^3 \times (0.25)^2 \times (0.75)^2 \times (0.42)^2 \times (0.30) \times 0.15$

$= 6.7 \times 10^{-5}$
Finding the most likely path

- Find path $\pi^*$ that maximizes total joint probability $P(x, \pi)$
- $P(x, \pi) = a_{0\pi_1} \prod_{i=1}^n e_{\pi_i}(x_i) \times a_{\pi_i\pi_{i+1}}$

Viterbi algorithm

1. Define $V_k(i) = \text{Probability of the most likely path through state } \pi_i = k$
2. Compute $V_k(i+1)$ recursively, as a function of $\max_{k'} V_{k'}(i)$

- Assume we know $V_j$ for the previous time step $(i-1)$
- Calculate $V_k(i) = e_k(x_i) \times \max_j (V_j(i-1) \times a_{jk})$ for all possible previous states $j$

The Viterbi Algorithm

<table>
<thead>
<tr>
<th>State 1</th>
<th>2</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
</tr>
</tbody>
</table>

Input: $x = x_1 \ldots x_N$

- **Initialization:** $V_{k}(0) = 1$, $V_j(0) = 0$, for all $k > 0$
- **Iteration:** $V_k(i) = a_k(x_i) \times \max_{j} a_{jk} V_{j}(i-1)$
- **Termination:** $P(x, \pi^*) = \max_k V_k(N)$

**Scoring**

- 1. Scoring $x$, one path
- 2. Scoring $x$, all paths

- **Decoding**
- 3. Viterbi decoding
- 4. Posterior decoding

- **Learning**
- 5. Supervised learning, given $\Lambda^*$
- 6. Unsupervised learning

2. Model evaluation:

Total $P(x|M)$, summed over all paths

- **Forward algorithm**

Simple: Given the model, generate some sequence $x$

- 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 $a_{\pi_1\pi_2}$
  4. ... until emitting $x_n$

We have some sequence $x$ that can be emitted by $p$. Can calculate its likelihood. However, in general, many different paths may emit the same sequence $x$. How do we find the total probability of generating a given $x$ over any path?
Complex: Given x, was it generated by the model?

Given a sequence x,

1. What is the probability that x was generated by the model (using any path)?
   
   \[ P(x) = \sum P(x, \pi) \]
   
   - Challenge: exponential number of paths
   - (cheap) alternative: Calculate probability over maximum (Viterbi) path \( \pi^* \)
   - (real) solution: Calculate sum iteratively using principles of dynamic programming

The Forward Algorithm – derivation

Define the forward probability:

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

\[ = \sum_{x_{i+1}} P(x_1, \ldots, x_i, x_{i+1}, \pi_{i+1} = l) \theta_l(x) \]

\[ = \sum_{x_{i+1} \pi_{i+1}} P(x_1, \ldots, x_i, \pi_{i+1} = l) a_{kl} \theta_l(x) \]

\[ = \theta_l(x) \sum_{x_{i+1}} f(i) a_{kl} \]

Calculate total probability \( \sum_{x} P(x, \pi) \) recursively

\[ f(j) = \theta_j(x) \sum_{x_{j+1}} a_{jk} f(j+1) \]

- Assume we know \( f \) for the previous time step \( (i-1) \)

- Calculate \( f(i) = \theta_i(x) \sum_{x_{i+1}} a_{ik} f(i+1) \)

The Forward Algorithm

Input: \( x = x_1 \ldots x_N \)

Initialization: \( f(0) = 1, f(i) = 0, \text{ for all } k > 0 \)

Iteration: \( f(i) = \theta_i(x) \sum_{x_{i+1}} a_{ik} f(i) \)

Termination: \( P(x, \pi^*) = \sum_k f(N) \)

In practice:

- Sum of log scores is difficult
- \( \exp(1+p+q) \) scaling of probabilities

Time: \( O(K^2N) \)

Space: \( O(K) \)

Application: Distinguishing between two models

- HMM1: Promoters = only Cs and Gs matter
- HMM2: Promoters = it’s actually CpGs that matter (*C*-phosphate-*G*, i.e. on the same strand!)

(increasing the state space)

In the human genome, CpG islands matter!

- Regions of regulatory importance in promoters of many genes
  - CpGs more important than simply the abundance of Cs and Gs
  - Provide evidence of methylation state!
- Methylation process in the human genome (form of silencing):
  - Methylation signature: high chance of methyl-C mutating to T in CpG
  - CpG dinucleotides are rare, throughout the genome
  - BUT methylation is suppressed for active promoters
  - CpG dinucleotides are much more frequent than elsewhere
    - Such regions are called CpG islands
    - A few hundred to a few thousand bases long
- Problems:
  - Given a short sequence, does it come from a CpG island or not?
  - How to find the CpG islands in a long sequence
  - How do we encode this in an hidden Markov model?
Increasing the state of the system (looking back)

- Markov Models are memory-less
  - In other words, all memory is encoded in the states
  - To remember additional information, augment state
- Our first HMM had minimal memory
  - State, emissions, only depend on current state
  - Current state only encoded one previous nucleotide
- How do you count di-nucleotide frequencies?
  - CpG islands: di-nucleotides
  - Codon triplets: tri-nucleotides
  - Di-codon frequencies: six nucleotides

Expanding the number of states

Training emission parameters for CpG+/CpG- states

Example 2: CpG islands: incorporating memory

Modeling CpG islands: incorporating memory

HMM for CpG islands

Why we need so many states...
In our simple GC-content example, we only had 2 states (+|-)
Why do we need 8 states here:  4 CpG+ / 4 CpG- ?

Encode ‘memory’ of previous state: nucleotide transitions

HMM for CpG islands

- Build a single model that combines two such Markov chains:
  - ‘+’ states: A, C, G, T
  - Emits symbols: A, C, G, T in CpG islands
  - ‘-’ states: A, C, G, T
  - Emits symbols: A, C, G, T in non-islands
- Emission probabilities distinct for the ‘+’ and the ‘-’ states
  - Infer most likely set of states, giving rise to observed emissions
  - ‘Paint’ the sequence with + and - states

One path

All paths

1. Scoring x, one path
   \[ P(x, \pi) \]
   \[ P(x) = \sum \pi \]

2. Scoring x, all paths
   \[ P(x, \pi) \]
   \[ P(x) = \sum \pi \]

3. Viterbi decoding
   \[ \pi^* = \text{argmax}_{x} P(x, \pi) \]

4. Posterior decoding
   \[ \pi^* = \left\{ \pi \mid \pi \text{argmax}_{\pi} \sum_{P(x, \pi)} \right\} \]

5. Supervised learning, given \( \pi \)
   \[ \Lambda^* = \text{argmax}_{P(x, \pi)} \]

6. Unsupervised learning
   \[ \Lambda^* = \text{argmax}_{\pi} \sum_{P(x, \pi)} \]

Viterbi training, best path
Baum-Welch training, over all paths

Learning

Scoring

Decoding

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>274</td>
<td>426</td>
<td>120</td>
</tr>
<tr>
<td>171</td>
<td>368</td>
<td>274</td>
<td>188</td>
</tr>
<tr>
<td>161</td>
<td>336</td>
<td>375</td>
<td>125</td>
</tr>
<tr>
<td>179</td>
<td>353</td>
<td>384</td>
<td>182</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>.300</td>
<td>.205</td>
<td>.285</td>
<td>.210</td>
</tr>
<tr>
<td>.322</td>
<td>.298</td>
<td>.278</td>
<td>.302</td>
</tr>
<tr>
<td>.344</td>
<td>.246</td>
<td>.290</td>
<td>.208</td>
</tr>
<tr>
<td>.177</td>
<td>.239</td>
<td>.252</td>
<td>.252</td>
</tr>
</tbody>
</table>
4. Decoding, all paths

Find the likelihood an emission \( x_i \) is generated by a state

Calculate most probable label at a single position

- Calculate most probable label, \( L^*_i \), at each position \( i \)
- Do this for all \( N \) positions gives us \( \{L^*_1, L^*_2, L^*_3, \ldots, L^*_N\} \)
- How much information have we observed? Three settings:
  - Observed nothing: Use prior information
  - Observed only character at position \( i \): Prior + emission probability
  - Observed entire sequence: Posterior decoding

Calculate \( P(\pi_i = C \mid x_i = G) \)

- With no knowledge (no characters)
  - Simply time spent in markov chain states
  - \( P(\pi = k) = \) most likely state (prior)
- With very little knowledge (just that character)
  - Time spent, adjusted for different emission probs.
  - Use Bayes rule to change inference directionality
  - \( P(\pi = k \mid x = G) = P(\pi = k) \cdot P(x = G \mid \pi) / P(x = G) \)
- With knowledge of entire sequence (all characters)
  - \( P(\pi = k \mid x = \text{AGCGCG...GATTACGTCGTA}) \)
  - Sum over all paths that emit ‘G’ at position 7
    \( \Rightarrow \) Posterior decoding

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

\[ P(\pi_i = k \mid x) = P(\pi_i = k, x) = P(x_1, x_2, \ldots, x_i, \pi_i = k, x_{i+1}, \ldots, x_N) \]

\[ = P(x_1, x_2, \ldots, x_i, \pi_i = k) \cdot P(x_{i+1}, \ldots, x_N \mid x_1, x_2, \ldots, x_i, \pi_i = k) \]

\[ = \sum_{\pi_{i+1} \ldots \pi_N} P(x_1, x_2, \ldots, x_i, \pi_i = k, x_{i+1}, \ldots, x_N \mid \pi_{i+1} \ldots \pi_N) \cdot P(\pi_{i+1} \ldots \pi_N) \]

Define the backward probability:

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

\[ = \sum_{\pi_{i+1} \ldots \pi_N} P(x_1, x_2, \ldots, x_i, \pi_i = k, x_{i+1}, \ldots, x_N \mid \pi_{i+1} \ldots \pi_N) \]

\[ = P(\pi_i = k, x) = \sum_{\pi_{i+1} \ldots \pi_N} P(x_1, x_2, \ldots, x_i, \pi_i = k, x_{i+1}, \ldots, x_N) \]

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

\[ = \sum_{\pi_{i+1} \ldots \pi_N} \begin{cases} a_{\pi_{i+1}} & \text{if } \pi_i = k \\ b_k(i) & \text{if } \pi_i = k \end{cases} \]

\[ = \sum_{\pi_{i+1} \ldots \pi_N} \begin{cases} a_{\pi_{i+1}} & \text{if } \pi_i = k \\ b_k(i) & \text{if } \pi_i = k \end{cases} \]

\[ = \sum_{\pi_{i+1} \ldots \pi_N} b_k(i+1) \]

The Backward Algorithm – derivation

Calculate total end probability recursively

- Assume we know \( b_k \) for the next time step \((i+1)\)

\[ b_k(i) = \sum_{\pi_{i+1} \ldots \pi_N} \begin{cases} a_{\pi_{i+1}} & \text{if } \pi_i = k \\ b_k(i+1) & \text{if } \pi_i = k \end{cases} \]

\[ = \sum_{\pi_{i+1} \ldots \pi_N} \begin{cases} a_{\pi_{i+1}} & \text{if } \pi_i = k \\ b_k(i+1) & \text{if } \pi_i = k \end{cases} \]
The Backward Algorithm

Input: \( x = x_1 \ldots x_N \)

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:
\[
 P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)
\]

In practice:
- Sum of log scores is difficult
- \( \approx \text{exp}(1+p+q) \)
- Scaling of probabilities

Running time and space:
- Time: \( O(K^2N) \)
- Space: \( O(K) \)

Putting it all together: Posterior decoding

\[
 P(k) = P(\pi_i = k \mid x) = f_k(i) b_k(i) / P(x)
\]

- Probability that \( i \text{ th state is } k, \text{ given all emissions } x \)

- Posterior decoding
  - Define most likely state for every of sequence \( x \)
  - \( \pi^* = \text{argmax}_k P(\pi_i = k \mid x) \)
  - For classification, more informative than Viterbi path \( \pi^* \)
  - However, it may give an invalid sequence of states
- Not all \( j \rightarrow k \) transitions may be possible

Summary this far
- Generative model. Hidden states, observed emissions.
  - Generate a random sequence
    - Choose random transition, choose random emission (\#0)
  - Scoring the likelihood of a sequence
    - Calculate likelihood of annotated path and sequence
      - Multiply emission and transition probabilities (\#1)
    - Without specifying a path, total probability of generating \( x \)
      - \( P(x) = \sum_{\pi} P(x, \pi) \)
  - Decoding: Finding the most likely path, given a sequence
    - What is the most likely path generating entire sequence?
      - Viterbi algorithm (\#2)
    - What is the most probable state at each time step?
      - \#3 + backward algorithms
  - Next: Learning (\#5 and \#6)

Learning: How to train an HMM

Transition probabilities
- \( P(P_{i+1} \mid B_i) \) – the probability of entering a pathogenicity island from background DNA

Emission probabilities
- i.e. the nucleotide frequencies for background DNA and pathogenicity islands

One path
1. Scoring, one path
\[
 P(x, \pi) = \sum_{\pi} P(x, \pi)
\]

2. Scoring, all paths
\[
 P(x) = \sum_{\pi} P(x, \pi)
\]

3. Viterbi decoding
\[
 \pi^* = \text{argmax}_\pi P(x, \pi)
\]

4. Posterior decoding
\[
 \pi^* = \sum_{\pi} P(x, \pi) \; \text{over all paths}
\]

All paths
5. Supervised learning, given \( \pi^* \)
\[
 \Lambda^* = \text{argmax}_{\Lambda} P(x, \pi^* \mid \Lambda)
\]

6. Unsupervised learning
\[
 \Lambda^* = \text{argmax}_{\Lambda} \sum_{\pi} P(x, \pi \mid \Lambda)
\]

Two learning scenarios

Case 1. Estimation when the “right answer” is known

Examples:
- A genomic region \( x = x_1 \ldots x_{1,000,000} \) where we have good (experimental) annotations of the CpG islands

Case 2. Estimation when the “right answer” is unknown

Examples:
- The porcupine genome; we don’t know how frequent are the CpG islands there, neither do we know their composition

QUESTION: Update the parameters \( \theta \) of the model to maximize \( P(x \mid \theta) \)
Two types of learning: Supervised / Unsupervised

5. Supervised learning
infer model parameters given labeled training data
- GIVEN:
  • a HMM M, with unspecified transition/emission probs.
  • labeled sequence x,
- FIND:
  • parameters \( \theta = (E, A) \) that maximize \( P(x | \theta) \)
  \( \Rightarrow \)
  Simply count frequency of each emission and transition, as observed in the training data

6. Unsupervised learning
infer model parameters given unlabelled training data
- GIVEN:
  • a HMM M, with unspecified transition/emission probs.
  • unlabeled sequence x,
- FIND:
  • parameters \( \theta = (E, A) \) that maximize \( P(x | \theta) \)
  \( \Rightarrow \)
  Viterbi training:
  guess parameters, find optimal Viterbi path (#2), update parameters (#5), iterate
  Baum-Welch training:
  guess parameters, sum over all paths (#4), update parameters (#5), iterate

---

**Case 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:**

\[
A_{kl} = \text{# times } k \rightarrow l \text{ transition occurs in } x
\]

\[
E_k(b) = \text{# times state } k \text{ in } \pi \text{ emits } b \text{ in } x
\]

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

\[
a_{kl} = \frac{A_{kl}}{\sum_l A_{kl}} \quad e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)}
\]

---

**Learning From Labelled Data**

**Maximum Likelihood Estimation**

If we have a sequence that has islands marked, we can simply count

![Diagram](image)

**Pseudocounts**

Solution for small training sets:

Add pseudocounts

\[
A_{kl} = \# \text{ times } k \rightarrow l \text{ transition occurs in } \pi \text{ } + f_{kl}
\]

\[
E_k(b) = \# \text{ times state } k \text{ in } \pi \text{ emits } b \text{ in } x \text{ } + f_k(b)
\]

\( f_{kl}, f_k(b) \) are pseudocounts representing our prior belief

Larger pseudocounts \( \Rightarrow \) Strong prior belief

Small pseudocounts \((c < 1)\): just to avoid 0 probabilities
Example: Training Markov Chains for CpG islands

- Training Set:
  - set of DNA sequences w/ known CpG islands
- Derive two Markov chain models:
  - ‘+’ model: from the CpG islands
  - ‘-’ model: from the remainder of sequence
- Transition probabilities for each model:

\[
\begin{align*}
\sum_{t' st'} a_{stc}^+ &= \text{the number of times letter } t \text{ followed letter } s \text{ inside the CpG islands} \\
\sum_{t' st'} a_{stc}^- &= \text{the number of times letter } t \text{ followed letter } s \text{ outside the CpG islands}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{cccc}
A & C & G & T \\
C & 0.180 & 0.274 & 0.426 & 0.120 \\
G & 0.079 & 0.355 & 0.384 & 0.182 \\
T & 0.171 & 0.368 & 0.274 & 0.188 \\
\end{array}
\end{align*}
\]

6: Unsupervised learning

Estimate model parameters based on unlabeled training data

Learning case 2. When the right answer is unknown

We don’t know the true \( A_{kl}, E_k(b) \)

Idea:
- We estimate our “best guess” on what \( A_{kl}, E_k(b) \) are (M step, maximum-likelihood estimation)
- We update the probabilistic parse of our sequence, based on these parameters (E step, expected probability of being in each state given parameters)
- We repeat

Two settings:
- Simple: Viterbi training (best guest = best path)
- Correct: Expectation maximization (all paths, weighted)
# Simple case: Viterbi Training

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

**Iteration:**
1. Perform Viterbi, to find $\pi^*$
2. Calculate $A_{kl}, E_k(b)$ according to $\pi^* +$ pseudocounts
3. Calculate the new parameters $a_{kl}, e_k(b)$

Until convergence

**Notes:**
- Convergence to local maximum guaranteed. Why?
- Does not maximize $P(x | \theta)$
- In general, worse performance than Baum-Welch

---

## Expectation Maximization (EM)

**The basic idea is the same:**
1. Use model to estimate missing data (E step)
2. Use estimate to update model (M step)
3. Repeat until convergence

EM is a general approach for learning models (ML estimation) when there is “missing data” Widely used in computational biology

---

### Case 2. When the right answer is unknown

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 $\hat{\theta}$

**Principle:** Expectation Maximization

1. Estimate probabilistic parse based on parameters (E step)
2. Update parameters $A_{kl}, E_k$ based on probabilistic parse (M step)
3. Repeat 1 & 2, until convergence

---

### Estimating probabilistic parse given params (E step)

To estimate $A_{kl}$:

- At each position $i$:
  
  Find probability transition $k \rightarrow l$ is used:

  \[
  P(\pi_i = k, \pi_{i+1} = l | x) = \frac{1}{Q} \sum_{x_i} P(x, \pi_i = k, \pi_{i+1} = l | x) = P(x, \pi | \theta)
  \]

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

  where $Q = \sum_{x_i} P(x, \pi | \theta)$

- So:
  
  \[
  P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \frac{P(x, \pi_i = k, \pi_{i+1} = l | x, \theta)}{Q(x)}
  \]

  (For one such transition, at time step $i \rightarrow i+1$)

---

## Expectation Maximization (EM)

1. Initialize parameters randomly
2. E Step Estimate expected probability of hidden labels $Q$, given current (latest) parameters and observed (unchanging) sequence

\[
Q = P(Labels | S, params^{-1})
\]

3. M Step Choose new maximum likelihood parameters over probability distribution $Q$, given current probabilistic label assignments

\[
\hat{\theta} = \arg \max_{\theta} P(S | \theta)
\]

4. Iterate

### P(\theta | S) guaranteed to increase each iteration

---

## Decoding

**Scoring**

<table>
<thead>
<tr>
<th>One path</th>
<th>All paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Scoring x, one path</td>
<td>2. Scoring x, all paths</td>
</tr>
<tr>
<td>$P(x, \pi)$</td>
<td>$P(x) = \sum P(x, \pi)$</td>
</tr>
<tr>
<td>Prob of a path, emissions</td>
<td>Prob of emissions, over all paths</td>
</tr>
</tbody>
</table>

- 3. Viterbi decoding
  - $\pi^* = \arg \max P(x, \pi)$
  - Most likely path

- 4. Posterior decoding
  - $\pi^* = \{ \pi_i | \pi_i = \arg \max_k \sum P(x | \pi_i = k) \}$
  - Path containing the most likely state at any time point.

### Learning

5. Supervised learning, given $\pi^*$

6. Unsupervised learning

- $\Lambda^* = \arg \max_{\Lambda} P(x, \pi | \Lambda)$
- Viterbi training, best path
- Baum-Welch training, over all paths
New parameters given probabilistic parse (M step)

(Sum over all k→l transitions, at any time step i)

So,

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

Similarly,

\[ E_k(b) = \frac{1}{P(x)} \sum_{\{i \mid x_i = b\}} f_k(i) b_k(i) \]

Dealing with multiple training sequences

(Sum over all training seqs, all k→l transitions, all time steps i)

If we have several training sequences, \(x_1, \ldots, x_M\), each of length \(N\),

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

Similarly,

\[ E_k(b) = \sum_x \frac{1}{P(x)} \sum_{\{i \mid x_i = b\}} f_k(i) b_k(i) \]

The Baum-Welch Algorithm

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

**Iteration:**
1. Forward
2. Backward
3. \(\Rightarrow\) Calculate new log-likelihood \(P(x \mid \theta)\) (E step)
4. Calculate \(A_{kl}, E_k(b)\)
5. \(\Rightarrow\) Calculate new model parameters \(a_{kl}, e_k(b)\) (M step)

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

Until \(P(x \mid \theta)\) does not change much

The Baum-Welch Algorithm – comments

**Time Complexity:**

\# iterations \times O(K^2N)

- Guaranteed to increase the log likelihood of the model
  \[ P(\theta \mid x) = \frac{P(x, \theta)}{P(x)} = \frac{P(x \mid \theta)}{(P(x) P(\theta))} \]
- Not guaranteed to find globally best parameters
  Converges to local optimum, depending on initial conditions
- Too many parameters / too large model: Overtraining

One path

<table>
<thead>
<tr>
<th>Scoring</th>
<th>All paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Scoring x, one path (P(x, \pi))</td>
<td>2. Scoring x, all paths (P(x) = \sum \pi \cdot P(x, \pi))</td>
</tr>
<tr>
<td>Prob of a path, emissions</td>
<td>Prob of emissions, over all paths</td>
</tr>
</tbody>
</table>

3. Viterbi decoding
\[ \pi^* = \text{argmax}_\pi P(x, \pi) \]
Most likely path

4. Posterior decoding
\[ \pi^* = \{ \pi \mid \pi = \text{argmax}_\pi \sum \pi \cdot P(\pi \mid x) \} \]
Path containing the most likely state at any time point.

5. Supervised learning, given \(\pi\)
\[ \Lambda^* = \text{argmax}_\Lambda P(x, \pi \mid \Lambda) \]
6. Unsupervised learning
\[ \Lambda^* = \text{argmax}_\Lambda \sum P(x, \pi \mid \Lambda) \]

Viterbi training, best path

Baum-Welch training, over all paths

What have we learned?

- Generative model. Hidden states, observed emissions.
  - Generate a random sequence
    - Choose random transition, choose random emission (#5)
- Scoring: Finding the likelihood of a given sequence
  - Calculate likelihood of annotated path and sequence
    - Multiply emission and transition probabilities (#1)
  - Without specifying a path, total probability of generating \(x\)
    - Sum probabilities over all paths
      - Forward algorithm (#3)
- Decoding: Finding the most likely path, given a sequence
  - What is the most likely path generating entire sequence?
    - Viterbi algorithm (#2)
  - What is the most probable state at each time step?
    - Forward + backward algorithms, posterior decoding (#4)
- Learning: Estimating HMM parameters from training data
  - When state sequence is known
    - Simply compute maximum likelihood \(\Lambda\) and \(E\) (#5a)
  - When state sequence is not known
    - Viterbi training: iterative estimation of best path / frequencies (#5b)
    - Baum-Welch: iterative estimation over all paths / frequencies (#6)
## The main questions on HMMs

<table>
<thead>
<tr>
<th><strong>SCORING</strong></th>
<th><strong>PARSING</strong></th>
<th><strong>LEARNING</strong></th>
</tr>
</thead>
</table>
| 1. **Scoring x, one path**
  - GIVEN a HMM $M$, a path $\pi$, and a sequence $x$,
  - FIND: $\text{prob}(\pi, x | M)$
  - *Running the model*: simply multiply emission and transition probabilities
  - Application: "all promoter" vs. "all background" comparisons
| 2. **Scoring x, all paths**
  - GIVEN a HMM $M$, a sequence $x$,
  - FIND: $\text{total probability } \sum_{\pi} \text{prob}(\pi, x | M)$
  - *Forward algorithm*: sum score over all paths (same result as backward)
| 3. **Viterbi decoding**
  - GIVEN a HMM $M$, and a sequence $x$,
  - FIND: $\text{sequence } \pi^* \text{ of states that maximize } \text{prob}(x, \pi | M)$
  - *Viterbi algorithm*: dynamic programming, max score over all paths, trace pointers find path
| 4. **Posterior decoding**
  - GIVEN a HMM $M$, a sequence $x$,
  - FIND: $\text{total probability } \sum_{\pi} \text{prob}(\pi, x | M)$
  - *Posterior decoding*: run forward & backward algorithms to & from state $\pi_k$
| 5. **Supervised learning**
  - GIVEN a HMM $M$, with unspecified transition/emission probs., labeled sequence $x$,
  - FIND: parameters $\theta = (e_i, a_{ij})$ that maximize $P(x | \theta)$
  - Simply count frequency of each emission and transition observed in the training data
| 6. **Unsupervised learning**
  - GIVEN a HMM $M$, with unspecified transition/emission probs., unlabeled sequence $x$,
  - FIND: parameters $\theta = (e_i, a_{ij})$ that maximize $P(x | \theta)$
  - *Viterbi training*: guess parameters, find optimal Viterbi path ($\pi^*$), update parameters ($\theta$), iterate
  - *Baum-Welch training*: guess parameters, sum over all emissions/transitions ($\theta$), update ($\theta$), iterate