Lecture 05
Hidden Markov Models
Part II

Goals for today: HMMs, part II

1. Review: Basics and three algorithms from last time
   - Markov Chains and Hidden Markov Models
   - Calculating likelihoods $P(x, \pi)$ (algorithm 1)
   - Viterbi algorithm: $\pi^* = \text{argmax}_\pi P(x, \pi)$ (alg 3)
   - Forward algorithm: $P(x)$, over all paths (alg 2)

2. Increasing the ‘state’ space / adding memory
   - Finding GC-rich regions vs. finding CpG islands
   - Gene structures GENSCAN, chromatin ChromHMM

3. Posterior decoding: Another way of ‘parsing’
   - Find most likely state $\pi_i$, overall all possible paths

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

Markov chains and Hidden Markov Models (HMMs)

HMM nomenclature for this course

- Vector $x = \text{Sequence of observations}$
- Vector $\pi = \text{Hidden path (sequence of hidden states)}$
- Transition matrix $A = a_{ij} = \text{probability of} \ k \to l \text{ state transition}$
- Emission vector $E = e_k(x) = \text{prob. of observing} \ x \text{ from state} \ k$
- Bayes’s rule:
  $P(\pi_j = k | x) = \frac{P(x | \pi_j = k) \cdot P(\pi_j = k)}{P(x)}$

Module 1: Aligning and modeling genomes

Module 1: Computational foundations
- Dynamic programming: exploring exponential spaces in poly-time
- Linear-time string matching, Hashing, Content-based indexing
- Hidden Markov Models: decoding, evaluation, parsing, learning

Last week: Sequence alignment / comparative genomics
- Local/global alignment: infer nucleotide-level evolutionary events
- Database search: scan for regions that may have common ancestry

This week: Modeling genomes / exon / CpG island finding
- Modeling class of elements, recognizing members of a class
- Application to gene finding, conservation islands, CpG islands
1. Scoring x, one path
   \[ P(x, \pi) \]
   Prob of a path, emissions

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

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

4. Posterior decoding
   \[ \pi^\wedge = \{ \pi_i | \pi_i=\text{argmax}_\pi \sum_{t=1}^T P(t|\pi) \} \]
   Path containing the most likely state at any time point.

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

   \[ \Lambda^* = \text{argmax}_\pi \sum_{t=1}^T P(x, \pi | \Lambda) \]
   Baum-Welch training, over all paths

---

### Example: One particular P vs. B assignment

<table>
<thead>
<tr>
<th>L:</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0.25</td>
<td>0.25</td>
<td>0.42</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

\[
P = P(G | B)P(B | G)P(C | B)P(A | B)P(A | B)P(T | B)P(C | B)P(G | B)P(B | C)P(A | B)P(A | B)P(T | B)P(C | B)
= (0.85)^3 \times (0.25)^6 \times (0.75)^2 \times (0.42)^2 \times 0.30 \times 0.15
= 6.7 \times 10^{-7}
\]

### Finding the most likely path

- Find path \( \pi^* \) that maximizes total joint probability \( P(x, \pi) \)

\[
\text{argmax}_\pi P(x, \pi) = \text{argmax}_\pi \left( \prod_i e_{\pi_i}(x_i) \times a_{\pi_i \pi_{i+1}} \right)
\]

### Calculate maximum \( P(x, \pi) \) recursively

**Viterbi algorithm**

Define \( V_i(j) \) = Probability of the most likely path through state \( \pi_i=k \)

Compute \( V_i(i+1) \) recursively, as a function of \( \text{max}_k \{ V_i(i) \} \)

\[
V_i(j) = \sum_k a_{kj} a_k \text{e}(x_j) \text{e}(x_{i+1}) \text{a}(x_{i+1}) \text{a}(x_{i+2}) \cdots \text{a}(x_i) \text{a}(x_i) \text{e}(x_i) \text{e}(x_{i-1}) \cdots \text{a}(x_1) \text{e}(x_1) \text{a}(x_0)
\]

- Assume we know \( V_{i-1}(j) \) for the previous time step (i-1)
- Calculate \( V_i(j) = \max_k \left( V_{i-1}(j) \times a_k \text{e}(x_j) \text{a}(x_{i+1}) \right) \)

max ending in state \( j \) at step \( i \)
Transition from state \( j \)

---
The Viterbi Algorithm

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

**Initialization:**
\[ V_0(0) = 1, V_k(0) = 0, \text{ for all } k > 0 \]

**Iteration:**
\[ V_k(i) = e_k(x_i) \sum_j a_{jk} V_j(i-1) \]

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

**Traceback:**
Follow max pointers back

**In practice:**
Use log scores for computation

**Running time and space:**

- Time: \( O(K^2N) \)
- Space: \( O(KN) \)

---

P(x) \( \Rightarrow \) Prob that model emits x, sum over all paths

Given a sequence x, What is the probability that x was generated by the model (using any path)?

- \( P(x) = \sum_\pi P(x, \pi) \)
- Challenge: exponential number of paths
  - Sum over all paths, weighing the path probability, and the emission probs
  - Prob of emitting sequence: use individual emission probs from each state
  - Prob of path: use both emission and transition prob, based on previous path

**Calculating total probability \( \Sigma_\pi P(x, \pi) \) recursively**

1. Assume we know \( f_j \) for the previous time step (i-1)
2. Calculate \( f_k(i) = e_k(x_i) \sum_j a_{jk} f_j(i-1) \)

---

The Forward Algorithm

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

**Initialization:**
\[ f_0(0) = 1, f_k(0) = 0, \text{ for all } k > 0 \]

**Iteration:**
\[ f_k(i) = e_k(x_i) \sum_j a_{jk} f_j(i-1) \]

**Termination:**
\[ P(x, \pi^*) = \sum_k f_k(N) \]

**In practice:**
Sum of log scores is difficult
- approximate \( \exp(1+p+q) \)
- scaling of probabilities

**Running time and space:**

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

---

Goals for today: HMMs, part II

1. Review: Basics and three algorithms from last time
   - Markov Chains and Hidden Markov Models
   - Calculating likelihoods \( P(x, \pi) \) (algorithm 1)
   - Viterbi algorithm: Find \( \pi^* = \arg\max_\pi P(x, \pi) \) (alg 3)
   - Forward algorithm: Find \( P(x) \), over all paths (alg 2)
2. Increasing the ‘state’ space / adding memory
   - Finding GC-rich regions vs. finding CpG islands
   - Gene structures GENSCAN, chromatin ChromHMM
3. Posterior decoding: Another way of ‘parsing’
   - Find most likely state \( \pi_i \), overall all possible paths
4. Learning (ML training, Baum-Welch, Viterbi training)
   - Supervised: Find \( e_i(.) \) and \( a_{ij} \) given labeled sequence
   - Unsupervised: given only x \( \Rightarrow \) annotation + params
Increasing the state space (remembering more)

HMM1: Promoters = only Cs and Gs matter

HMM2: Promoters = it’s actually CpGs that matter
(di-nucleotides, remember previous nucleotide)

Remember previous nucleotide: expand both states

“Memory” of previous nucleotide is encoded in the current state.

GC-rich: 4 states
Background: 4 states

Training emission parameters for CpG+/CpG- states

- Count di-nucleotide frequencies:
  - 16 possible di-nucleotides, 16 transition parameters.
  - Alternative: 16 states, each emitting di-nucleotide

- Derive two Markov chain models:
  - ‘+’ model: from the CpG islands
  - ‘-’ model: from the remainder of sequence

- Transition probabilities for each model:
  - Encode differences in di-nucleotide frequencies

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

- A two-state HMM has minimal memory
  - Two states: GC-rich vs. equal probability
  - State, emissions, only depend on current state
  - Current state only encodes 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

HMM for CpG islands

- A single model combines two Markov chains, each of four nucleotides:
  - ‘+’ states: A, C, G, T
    - Emit symbols: A, C, G, T in CpG islands
  - ‘-’ states: A, C, G, T
    - Emit 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

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

Examples of HMMs for genome annotation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2 states, different nucleotide composition</td>
<td>8 states, 4 different transition probabilities</td>
<td>2 states, different conservation levels</td>
<td>2 states, different tri-nucleotide composition</td>
<td>2 states, different evolutionary signatures</td>
<td>~20 states, different composition/conservation, specific structure</td>
<td>40 states, different chromatin mark combination</td>
</tr>
<tr>
<td>GC-rich / AT-rich</td>
<td>CpG-rich / CpG-poor</td>
<td>Conserved / non-conserved</td>
<td>Coding exon / non-coding (intron or intergenic)</td>
<td>Coding exon / non-coding (intronic or intergenic)</td>
<td>First/last/mid coding exon,UTRs, intron1/2/3, intergenic, (+/- strand)</td>
<td>Enhancer / promoter / transcribed / repressed / repetitive</td>
</tr>
<tr>
<td>Nucleotides</td>
<td>Di-Nucleotides</td>
<td>Level of conservatio n</td>
<td>Triplets of nucleotides</td>
<td>64x64 matrix of codon substitution frequencies</td>
<td>Codons, nucleotides, splice sites, start/stop codons</td>
<td>Vector of chromatin mark frequencies</td>
</tr>
</tbody>
</table>

+ A .180 .274 .426 .120
| C .171 .368 .274 .188
| G .161 .339 .375 .125
| T .079 .355 .384 .182

- A .300 .205 .285 .210
| C .322 .298 .278 .302
| G .248 .246 .298 .208
| T .177 .239 .292 .292
HMM architecture matters: Protein-coding genes

- Gene vs. Intergenic
- Start & Stop in/out
- UTR: 5’ and 3’ end
- Exons, Introns
- Remembering frame
  - E₀, E₁, E₂
  - I₀, I₁, I₂
- Sequence patterns to transition between states:
  - ATG, TAG, Acceptor/Donor, TATA, AATAA

Chromatin State: Emission & Transition Matrices

- Emission matrix:
  - Multi-variate HMM
  - Emits vector of values
- Transition matrix:
  - Learn spatial relationships
  - No a-priori ‘gene’ structure

Goals for today: HMMs, part II

1. Review: Basics and three algorithms from last time
   - Markov Chains and Hidden Markov Models
   - Calculating likelihoods P(x,π) (algorithm 1)
   - Viterbi algorithm: Find π* = argmaxₚᵢ P(x,π) (alg 3)
   - Forward algorithm: Find P(x), over all paths (alg 2)
2. Increasing the ‘state’ space / adding memory
   - Finding GC-rich regions vs. finding CpG islands
   - Gene structures GENSCAN, chromatin ChromHMM
3. Posterior decoding: Another way of ‘parsing’
   - Find most likely state πᵢ, overall all possible paths
4. Learning (ML training, Baum-Welch, Viterbi training)
   - Supervised: Find e(ᵢ) and aᵢ given labeled sequence
   - Unsupervised: given only x → annotation + params

One path

1. Scoring x, one path
   P(x,π₁) ✓
   Prob of a path, emissions
2. Scoring x, all paths
   P(x) = ∑ᵢ P(x,πᵢ)
   Prob of emissions, over all paths
3. Viterbi decoding
   π* = argmaxₚᵢ P(x,πᵢ)
   Most likely path ✓

All paths

4. Posterior decoding
   π^ = {πᵢ | πᵢ=argmaxₚᵢ P(x,πᵢ)|} path containing the most likely state at any time point.
5. Supervised learning, given π
   Λ* = argmaxᵢ P(x,πᵢ|Λ)
6. Unsupervised learning
   Λ* = argmaxᵢ P(x,πᵢ|Λ)
   Viterbi training, best path
   Baum-Welch training, over all paths

Calculate most probable label at a single position

4. Decoding, all paths

Find the likelihood an emission xᵢ is generated by a state

- Calculate most probable label, L^*, at each position i
- Do this for all N positions gives us {L₁^*, L₂^*, L₃^*,..., Lₙ^*}
- 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 $P(\pi_7 = \text{CpG}^+ \mid x_7 = \text{G})$

- With no knowledge (no characters)
  - Simply time spent in markov chain states
  - $P(\pi_i = 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
  - $P(\pi_i = k \mid x_i = \text{G}) = P(\pi_i = k) \cdot P(x_i = \text{G} \mid \pi_i = k) / P(x_i = \text{G})$

- With knowledge of entire sequence (all characters)
  - $P(\pi_i = k \mid x = \text{AGCGCG...GATTATCGTCGTA})$
  - 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

$$P(\pi_i = k, x) = P(x_1...x_i, \pi_i = k, x_{i+1}...x_N)$$

$$= P(x_1...x_i, \pi_i = k) \cdot P(x_{i+1}...x_N \mid x_1...x_i, \pi_i = k)$$

$$= P(x_1...x_i, \pi_i = k) \cdot P(x_{i+1}...x_N \mid \pi_i = k)$$

$$= P(x_1...x_i, \pi_i = k) \cdot P(x_{i+1}...x_N \mid \pi_i = k)$$

$$\Rightarrow$$ Forward, $f(i)$  \hspace{1cm} Backward, $b(i)$

---

### The Backward Algorithm – derivation

Define the backward probability:

$$b_k(i) = P(x_{i+1}...x_N \mid \pi_i = k)$$

$$= \sum_{s_{i+1}...s_N} P(x_{i+1}s_{i+2}...x_N, \pi_{i+1}...\pi_N \mid \pi_i = k)$$

$$= \sum_{s_{i+1}...s_N} P(x_{i+1}s_{i+2}...x_N, \pi_{i+1}...\pi_N, s_{i+2}...s_N \mid \pi_i = k)$$

$$= \sum_{s_{i+1}} e_s(x_{i+1}) a_{sk} b_k(i+1)$$

Calculate total end probability recursively

- Assume we know $b_k$ for the next time step ($i+1$)
- Calculate $b_k(i) = \sum_{s_{i+1}} e_s(x_{i+1}) a_{sk} b_{k}(i+1)$

---

### Calculate total end probability recursively

$$b_k(i) = \sum_{s_{i+1}} e_s(x_{i+1}) a_{sk} b_{k}(i+1)$$

In practice:

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

**Iteration:**

$$b_k(i) = \sum_{s_i} e_s(x_{i+1}) a_{sk} b_{k}(i+1)$$

**Running time and space:**

- Time: $O(K^2N)$
- Space: $O(K)$

---

### Putting it all together: Posterior decoding

$P(k) = P(\pi_i = k \mid x) = f(i) b(i) / P(x)$

- Probability that $i^{th}$ state is $k$, given all emissions $x$

- **Posterior decoding**
  - Define most likely state for every of sequence $x$
  - $\pi_i^* = \text{argmax}_k, P(\pi_i = k \mid x)$

- **Posterior decoding ‘path’ $\pi^*$**
  - For classification, more informative than Viterbi path $\pi^*$
  - More refined measure of “which hidden states” generated $x$
  - However, it may give an invalid sequence of states
  - Not all $j \rightarrow k$ transitions may be possible
Goals for today: HMMs, part II

1. Review: Basics and three algorithms from last time
   - Markov Chains and Hidden Markov Models
   - Calculating likelihoods $P(x, \pi)$ (algorithm 1)
   - Viterbi algorithm: Find $\pi^* = \text{argmax}_\pi P(x, \pi)$ (alg 3)
   - Forward algorithm: Find $P(x)$, over all paths (alg 2)

2. Increasing the ‘state’ space / adding memory
   - Finding GC-rich regions vs. finding CpG islands
   - Gene structures GENSCAN, chromatin ChromHMM

3. Posterior decoding: Another way of ‘parsing’
   - Find most likely state $\pi_i$, overall all possible paths

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

Learning: How to train an HMM

Transition probabilities
- e.g. $P(P_{i+1}|B_i)$ – the probability of entering a pathogenicity island from background DNA

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

Two learning scenarios
Case 1. Estimation when the “right answer” is known
- Examples:
  - GIVEN: 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:
  - GIVEN: 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 $P(x|\theta)$

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 $\theta = (E_i, A_{ij})$ that maximize $P(x | \theta)$
    - 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 $\theta = (E_i, A_{ij})$ that maximize $P(x | \theta)$
    - 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

5: Supervised learning
Estimate model parameters based on labeled training data
Case 1. When the right answer is known

Given $x = x_1 ... x_N$ for which the true $S = S_1 ... S_N$ is known,

Define:

- $A_{kl} = \text{# times } k \rightarrow l \text{ transition occurs in } \pi$
- $E_k(b) = \text{# times state } k \text{ in } \pi \text{ emits } b \text{ in } x$

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

$$ a_{kl} = \frac{A_{kl}}{\sum_i A_{ki}}, \quad e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)} $$

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 ($\epsilon < 1$): 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:

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)

One path
1. Scoring $x$, one path
   $P(x, \pi^*)$ 
   Prob of a path, emissions

2. Viterbi decoding
   $\pi^* = \arg\max_{\pi} P(x, \pi)$
   Most likely path

3. Posterior decoding
   $\pi^\Lambda = \{\pi_l \mid \pi_l = \arg\max_k \sum_l P(\pi_l = k|x)\}$
   Path containing the most likely state at any time point.

4. Supervised learning, given $\pi^\Lambda$
   $\Lambda^\Lambda = \arg\max_{\Lambda} \sum_{\pi} P(x, \pi|\Lambda)$
   Viterbi training, best path

5. Unsupervised learning.
   $\Lambda^\Lambda = \arg\max_{\Lambda} \sum_{\pi} P(x, \pi|\Lambda)$
   Baum-Welch training, over all paths

   $\Lambda^\Lambda = \arg\max_{\Lambda} \sum_{\pi} P(x, \pi|\Lambda)$
   Baum-Welch training, over all paths

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 $P(x \mid \theta)$
- 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

Rec 3 (SiPhy), Lec 8 (Kmeans), Lec 9 (motifs)

EM pervasive in computational biology

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
   \[ params' = \arg \max_{params} E_{\theta} \left[ \log P(S, labels | params^{-1}) \right] \]
4. Iterate

P(S|Model) guaranteed to increase each iteration

Case 2. When the right answer is unknown

Starting with our best guess of a model M, parameters θ:

Given \( x = x_1...x_N \)
for which the true \( π = π_1...π_N \) is unknown,

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

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:
\[
P(\pi_i = k, \pi_{i+1} = l | x) = \frac{1}{P(x)} \cdot \frac{P(\pi_i = k, \pi_{i+1} = l, x_1...x_N)}{P(x)}
\]
where \( Q = P(\pi_i = k, \pi_{i+1} = l, x_1...x_N) = P(\pi_{i+1} = l, x_1...x_N, \pi_i = k) \cdot P(\pi_{i+1} = l, x_1...x_N) \cdot P(\pi_i = k, x_1...x_N) \cdot P(\pi_{i+1} = l | \pi_i = k) \cdot P(\pi_i = k) \cdot P(x_1...x_N | \pi_i = k) \cdot P(\pi_{i+1} = l | \pi_i = k) \cdot P(\pi_i = k) \cdot P(x_1...x_N | \pi_i = k)
\]
\[
f_k(i) = \frac{f_k(i)}{P(x | \theta)} P(x | \theta)
\]
(For one such transition, at time step \( i \rightarrow i+1 \))

New parameters given probabilistic parse (M step)

(Sum over all \( k \rightarrow l \) transitions, at any time step \( i \))
So,
\[
A_{kl} = \sum_i P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \frac{\sum_i f_k(i) b_k(x_{i+1}) b_{l+1}}{P(x | \theta)}
\]

Similarly,
\[
E_k(b) = \frac{1}{P(x)} \sum_i \{ l | x_i = b \} f_k(i) b_k(i)
\]

Dealing with multiple training sequences

(Sum over all training seqs, all \( k \rightarrow l \) transitions, all time steps \( i \))
If we have several training sequences, \( x^1, ..., x^M \), each of length \( N \),
\[
A_{kl} = \sum_x \sum_i P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \frac{\sum_x \sum_i f_k(i) b_k(x_{i+1}) b_{l+1}}{P(x | \theta)}
\]

Similarly,
\[
E_k(b) = \frac{1}{P(x)} \sum_x \sum_i \{ l | x_i = b \} f_k(i) b_k(i)
\]
**The Baum-Welch Algorithm**

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

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

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

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

---

**Examples of HMMs for genome annotation**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GC-rich / AT-rich</td>
<td>CpG-rich / CpG-poor</td>
<td>Conserved / non-conserved</td>
<td>Coding exon / non-coding (intron or intergenic)</td>
<td>Coding exon / non-coding (intron or intergenic)</td>
<td>Conserved / non-conserved</td>
<td>First/last/middle coding exon, UTRs, intron1/2/3, intergenic, (*: strand)</td>
<td>Enhancer / promoter / transcribed / repressed / repetitive</td>
<td></td>
</tr>
<tr>
<td>Nucleotides</td>
<td>GC-rich / AT-rich</td>
<td>Conserved / non-conserved</td>
<td>Coding exon / non-coding (intron or intergenic)</td>
<td>Coding exon / non-coding (intron or intergenic)</td>
<td>Conserved / non-conserved</td>
<td>First/last/middle coding exon, UTRs, intron1/2/3, intergenic, (*: strand)</td>
<td>Enhancer / promoter / transcribed / repressed / repetitive</td>
<td></td>
</tr>
<tr>
<td>Highest level of conservation</td>
<td>Triplet distribution</td>
<td>64x64 matrix of codon substitution frequencies</td>
<td>Codons, nucleotides, splice sites, start/stop codons</td>
<td>Vector of chromatin mark frequencies</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Goals for today: HMMs, part II**

1. **Review:** Basics and three algorithms from last time
   - Markov Chains and Hidden Markov Models
   - Calculating likelihoods \( P(x, \pi) \) (algorithm 1)
   - Viterbi algorithm: Find \( \pi^* = \text{argmax}_\pi P(x, \pi) \) (alg 2)
   - Forward algorithm: Find \( P(x) \), over all paths (alg 2)

2. **Increasing the ‘state’ space / adding memory**
   - Finding GC-rich regions vs. finding CpG islands
   - Gene structures GENSCAN, chromatin ChromHMM

3. **Posterior decoding:** Another way of ‘parsing’
   - Find most likely state \( \pi_t \), overall all possible paths

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

---

**What have we learned?**

- **Generative model. Hidden states, observed emissions.**
  - Generate a random sequence
  - Choose random transition, choose random emission (#0)

- **Scoring:** Finding the likelihood of a given sequence
  - Calculate likelihood of annotated path and sequence
  - 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 \( A \) 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)
1. **Scoring x, one path** = Joint probability of a sequence and a path, given the model
   - GIVEN a HMM M, a path \( \pi \), and a sequence \( x \)
   - FIND \( \text{Prob}(x, \pi | M) \)
   - “Running the model”: simply multiply emission and transition probabilities
   - Application: “all promoter” vs. “all background” comparisons

2. **Scoring x, all paths** = Total probability of a sequence, summed across all paths
   - GIVEN a HMM M, a sequence \( x \)
   - FIND the total probability \( P(x | M) \) summed across all paths
   - Forward algorithm, sum score over all paths (same result as backward)

3. **Viterbi decoding** = Parsing a sequence into the optimal series of hidden states
   - GIVEN a HMM M, and a sequence \( x \)
   - FIND the sequence \( \pi^* \) of states that maximizes \( P(x, \pi | M) \)
   - Viterbi algorithm, dynamic programming, max score over all paths, trace pointers find path

4. **Posterior decoding** = Total prob that emission \( x_i \) came from state \( k \), across all paths
   - GIVEN a HMM M, a sequence \( x \)
   - FIND the total probability \( P(\pi_i = k | x, M) \)
   - Posterior decoding: run forward & backward algorithms to & from state \( \pi_i = k \)

5. **Supervised learning** = Optimize parameters of a model given training data
   - 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** = Optimize parameters of a model given training data
   - 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 (#2), update parameters (#5), iterate
   - Baum-Welch training: guess, sum over all emissions/transitions (#4), update (#5), iterate