Lecture 15

• Forward & forward/backward algorithms

• HMM parameter estimation
  – Viterbi training
  – Baum-Welch training
Hidden Markov Model

observed symbols

A → G → C → A → T

unobserved states

0 → π₁ → π₂ → π₃ → πᵢ → πₙ → 0

e_{π₁}(A) e_{π₂}(G) e_{π₃}(C) ... e_{πₙ}(T)

a_{0π₁} a_{π₁π₂} a_{π₂π₃} a_{π₃π₄} a_{πᵢπᵢ₊₁}
WDAG for 3-state HMM, length $n$ sequence

weights are emission probabilities $e_k(b_i)$ for $i^{th}$ residue $b_i$

weights are transition probabilities $a_{kl}$

$b_{i-1}$ position $i-1$  

$b_i$ position $i$  

$b_{i+1}$ position $i+1$
Path Weights

\[
e_i(b_{i-1}) \quad e_2(b_i) \quad e_3(b_{i+1})
\]

position \(i-1\) \quad position \(i\) \quad position \(i+1\)
Paths through graph from begin node to end node correspond to sequences of states

Product weight along path

= joint probability of state sequence & observed symbol sequence

Highest-weight path = highest probability state sequence

Sum of (product) path weights, over all paths,

= probability of observed sequence

Sum of (product) path weights over

= posterior probability of that edge or node
• use dynamic programming to find
  – sum of all product path weights
    = “forward algorithm” for probability of observed sequence
  – sum of all product path weights through particular node or particular edge
    = “forward/backward algorithm” to find posterior probabilities

• Now must use product weights and non-log-transformed probabilities
  – because need to \textit{add} probabilities
• In each case, compute successively for each node (by increasing depth: left to right)
  – the sum of the weights of all paths ending at that node
  – N.B. paths are constrained to begin at the begin node, end at end node!
• In forward/backward algorithm,
  – work through all nodes a second time, in opposite direction
    • i.e. in reverse graph – constraining paths to start at end node
For each vertex $v$, let $f(v) = \sum_{\text{paths } p \text{ ending at } v} \text{weight}(p)$, where \text{weight}(p) = \text{product} of edge weights in $p$. Only consider paths starting at ‘begin’ node.

Compute $f(v)$ by dynam. prog: $f(v) = \sum_i w_i f(v_i)$, where $v_i$ ranges over the parents of $v$, and $w_i = \text{weight of the edge from } v_i \text{ to } v$.

Similarly for $b(v) = \sum_p \text{beginning at } v \text{ weight}(p)$

The paths \textit{beginning} at $v$ are the ones \textit{ending} at $v$ in the reverse (or inverted) graph
from lecture 12:

- Can “invert” any WDAG: create graph with
  - same vertices & edge weights
  - direction of each edge reversed
  - is still acyclic!

- inverted WDAG has same paths (& path weights), but in reverse direction
  - “forward” path in inverted WDAG = “backward” path in original WDAG (& vice versa)
\[ f(v)b(v) = \text{sum of the path weights of all paths through } v. \]

\[ f(v')wb(v) = \text{sum of the path weights of all paths through the edge } (v',v) \]
Forward/backward algorithm

• Work through graph in forward direction:
  – compute and store \( f(v) \)

• Then work through graph in backward direction:
  – compute \( b(v) \)
  – compute \( f(v) b(v) \) and \( f(v)wb(v) \) as above, store in appropriate cumulative sums
  – only need to store \( b(v) \) until have computed \( b \)’s at next position

• Posterior probability of being in state \( s \) at position \( i \) is \( f(v) b(v) / \text{total sequence prob} \)
  – where \( v \) is the corresponding graph node
• Numerical issues: multiplying many small values can cause underflow. Remedies:

  – *Scale* weights to be close to 1 (affects all paths by same constant factor – which can be multiplied back later); or
  
  – (where possible) use log weights, so can add instead of multiplying.
  
  – see Rabiner & Tobias Mann links on web page
HMM Parameter Estimation

- **Parameters** = transition & emission probs
  - *parameter values* ↔ *probability model*
- If unknown, estimate from set of training sequences
- **Maximum likelihood** (ML) estimation (= choice of param vals to maximize prob of training data) is preferred
  - optimality properties of ML estimates discussed in Ewens & Grant

↔ finding maximum value on a multi-dimensional surface
  - Hard problem! Can be many local maxima
Parameter estimation when state sequence is *known*

- When underlying state sequence for each training sequence is *known*,
  - e.g.: site model

  then ML estimates are given by:
  - emission probabilities:
    \[ e_k(b) = \frac{\text{# times symbol } b \text{ emitted by state } k}{\text{# times state } k \text{ occurs}} \]
  - transition probabilities:
    \[ a_{kl} = \frac{\text{# times state } k \text{ followed by state } l}{\text{# times state } k \text{ occurs}} \]
    - in denominator above, *omit occurrence at last position of sequence* (for transition probabilities)
      - But include it for emission probs
    - can include pseudocounts, to incorporate prior expectations/avoid small sample overfitting (Bayesian justification)
HMM for *C. elegans* 3’ Splice Sites

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3276</td>
<td>3516</td>
<td>2313</td>
<td>476</td>
<td>67</td>
<td>757</td>
<td>240</td>
<td>8192</td>
<td>0</td>
<td>3359</td>
<td>2401</td>
<td>2514</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>970</td>
<td>648</td>
<td>664</td>
<td>236</td>
<td>129</td>
<td>1109</td>
<td>6830</td>
<td>0</td>
<td>0</td>
<td>1277</td>
<td>1533</td>
<td>1847</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>593</td>
<td>575</td>
<td>516</td>
<td>144</td>
<td>39</td>
<td>595</td>
<td>12</td>
<td>0</td>
<td>8192</td>
<td>1277</td>
<td>1533</td>
<td>1847</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3353</td>
<td>3453</td>
<td>4699</td>
<td>7336</td>
<td>7957</td>
<td>5731</td>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>1017</td>
<td>2957</td>
<td>2264</td>
<td></td>
</tr>
</tbody>
</table>

CONSENSUS: W W W W T T T t C A G r w w

Emission probabilities:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.400</td>
<td>0.429</td>
<td>0.282</td>
<td>0.058</td>
<td>0.008</td>
<td>0.092</td>
<td>0.029</td>
<td>1.000</td>
<td>0.000</td>
<td>0.410</td>
<td>0.293</td>
<td>0.307</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0.118</td>
<td>0.079</td>
<td>0.081</td>
<td>0.029</td>
<td>0.016</td>
<td>0.135</td>
<td>0.834</td>
<td>0.000</td>
<td>0.000</td>
<td>0.156</td>
<td>0.187</td>
<td>0.225</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0.072</td>
<td>0.070</td>
<td>0.063</td>
<td>0.018</td>
<td>0.005</td>
<td>0.073</td>
<td>0.001</td>
<td>0.000</td>
<td>1.000</td>
<td>0.310</td>
<td>0.159</td>
<td>0.191</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0.409</td>
<td>0.422</td>
<td>0.574</td>
<td>0.896</td>
<td>0.971</td>
<td>0.700</td>
<td>0.135</td>
<td>0.000</td>
<td>0.000</td>
<td>0.124</td>
<td>0.361</td>
<td>0.276</td>
<td></td>
</tr>
</tbody>
</table>

3’ ss
Intron
Exon

‘hidden’ states
Parameter estimation when state sequence unknown

• **Viterbi training**
  1. choose starting parameter values
     • must be valid probabilities; avoid 0 unless topology dictates
     • make them *biologically plausible* given state interpretation
  2. find Viterbi highest weight paths for each sequence
  3. estimate new emission and transition probs as above, *assuming* the Viterbi state sequence
  4. iterate steps 2 and 3 until convergence
     – not guaranteed to occur – but nearly always does
  5. repeat steps 1 – 4 with other starting values
     • choose values with highest total path score
• Viterbi training does not necessarily give ML estimates, but often are reasonably good
Baum-Welch training

- Special case of EM (‘expectation-maximization’) algorithm
- like Viterbi training, but
  - uses all paths, each weighted by its probability rather than just highest probability path.
- sometimes give significantly better results than Viterbi
  - e.g. for PFAM
Implementing Baum-Welch

– An edge in the WDAG contributes *fractional* (or *weighted*) *counts* given by its posterior probability:

– \((\ast): \frac{\sum_{\text{all paths } p \text{ through edge } e } \text{weight}(p))}{\sum_{\text{all paths } p} \text{weight}(p)}\)

(Fractional counts are computed using forward-backward algorithm)
\( f(v)b(v) = \) sum of the path weights of all paths through \( v \).

\( f(v')wb(v) = \) sum of the path weights of all paths through the edge \((v',v)\).
Compute new param estimates

- \( e_k(b)^\wedge = \left( \frac{\text{# times symbol } b \text{ emitted by state } k}{\text{# times state } k \text{ occurs}} \right) \)
- \( a_{kl}^\wedge = \left( \frac{\text{# times state } k \text{ followed by state } l}{\text{# times state } k \text{ occurs}} \right) \)
  - (In denom., omit frac counts at last position of sequence)

where “frac. # times” is given by (*) for appropriate edge type (emission or transition)
– New Baum-Welch parameter estimates have higher likelihood
  • general property of EM algorithm
  • not true in general for Viterbi training

– Iterate: get series of estimates converging to a local maximum on likelihood surface
Search of parameter space

- ML estimates correspond by definition to *global* maximum;
- but there may be many *local* maxima, and EM or Viterbi search can get “trapped” in one
- remedies:
  - Consider multiple starts (multiple choices for starting parameters)
  - use “reasonable values” to start search (e.g. unlikely transitions should be given small initial probabilities)
– Allow search to “jump” out of local maxima:
  • Add “noise” to counts at each iteration; gradually reduce the amount of noise
  • Use Viterbi-like training, but
    – sample paths probabilistically
      » (in retracing Viterbi path, choose edge at random according to its prob, rather than taking highest prob parent);
    – use “temperature” T to adjust probabilities;
      » initially with large T making all probs approximately equal;
      » then gradually reduce T
Probabilistic Viterbi Backtracking

reset all weights \( w \) to \( w^{1/T} \). For large \( T \) ( \( >> 1 \)), this makes distinct \( w \)'s relatively close; for small \( T \) ( \( << 1 \)), relatively far apart

choose parent \( v_i \) with probability \( w_i f(v_i) / f(v) \). For large \( T \), all parents almost equally likely to be chosen; for small \( T \), strongly favor largest (max) \( w_i f(v_i) \)

given choice of paths, re-estimate weights; iterate