Lecture 05

Hidden Markov Models
Part II
Module 1: Aligning and modeling genomes

- **Module 1: Computational foundations**
  - Dynamic programming: exploring exponential spaces in poly-time
  - Introduce Hidden Markov Models (HMMs): Central tool in CS
  - HMM algorithms: Decoding, evaluation, parsing, likelihood, scoring

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

- **Next week: Modeling genomes / exon / CpG island finding**
  - Modeling class of elements, recognizing members of a class
  - Application to gene finding, conservation islands, CpG islands
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 $\pi_i$, sum over 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
Markov chains and Hidden Markov Models (HMMs)

- What you see is what you get: next state only depends on current state (no memory)

All observed

- Markov Chain
  - Q: states
  - p: initial state probabilities
  - A: transition probabilities

- HMM
  - Q: states, p: initial, A: transitions
  - V: observations
  - E: emission probabilities

- Hidden state of the world determines emission probabilities
- State transitions are a Markov chain
HMM nomenclature for this course

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

Transitions: $a_{kl}=P(\pi_i=l|\pi_{i-1}=k)$
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 $k$
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
  \[ P(1) = P(2) = P(3) = P(4) = P(5) = \frac{1}{10} \]
  \[ P(6) = \frac{1}{2} \]

Casino player switches between fair and loaded die on average once every 20 turns

**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
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><strong>Topology / Transitions</strong></td>
<td>2 states, different nucleotide composition</td>
<td>2 states, different conservation levels</td>
<td>2 states, different trinucleotide composition</td>
<td>2 states, different evolutionary signatures</td>
<td>~20 states, different composition/conservation, specific structure</td>
<td>40 states, different chromatin mark combinations</td>
</tr>
<tr>
<td><strong>Hidden States / Annotation</strong></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>First/last/middle coding exon, UTRs, intron1/2/3, intergenic, *(+/- strand)</td>
<td>Enhancer / promoter / transcribed / repressed / repetitive</td>
</tr>
<tr>
<td><strong>Emissions / Observations</strong></td>
<td>Nucleotides</td>
<td>Level of conservation</td>
<td>Triplets of nucleotides</td>
<td>Nucleotide triplets, conservation levels</td>
<td>Codons, nucleotides, splice sites, start/stop codons</td>
<td>Vector of chromatin mark frequencies</td>
</tr>
</tbody>
</table>
The main questions on HMMs

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 backgorund” 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
<table>
<thead>
<tr>
<th>Decoding</th>
<th>Scoring</th>
<th>Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One path</strong></td>
<td><strong>All paths</strong></td>
<td></td>
</tr>
<tr>
<td>1. Scoring $x$, one path</td>
<td>2. Scoring $x$, all paths</td>
<td></td>
</tr>
<tr>
<td>[ P(x, \pi) ]</td>
<td>[ P(x) = \sum_{\pi} P(x, \pi) ]</td>
<td></td>
</tr>
<tr>
<td>Prob of a path, emissions</td>
<td>Prob of emissions, over all paths</td>
<td></td>
</tr>
<tr>
<td>3. Viterbi decoding</td>
<td>4. Posterior decoding</td>
<td></td>
</tr>
<tr>
<td>[ \pi^* = \arg\max_{\pi} P(x, \pi) ]</td>
<td>[ \pi^\Lambda = {\pi_i \mid \pi_i = \arg\max_k \sum_{\pi} P(\pi_i = k</td>
<td>x)} ]</td>
</tr>
<tr>
<td>Most likely path</td>
<td>Path containing the most likely state at any time point.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. Supervised learning, given $\pi$</td>
</tr>
<tr>
<td>[ \Lambda^* = \arg\max_{\Lambda} P(x, \pi</td>
</tr>
<tr>
<td>[ \Lambda^* = \arg\max_{\Lambda} \max_{\pi} P(x, \pi</td>
</tr>
<tr>
<td>Viterbi training, best path</td>
</tr>
</tbody>
</table>

5. Unsupervised learning. |
| \[ \Lambda^* = \arg\max_{\Lambda} \sum_{\pi} P(x, \pi|\Lambda) \] |
| Baum-Welch training, over all paths |
Probability of given path \( \pi \), emissions \( x \)

- \( \pi \) is the (hidden) path
- \( x \) is the (observed) sequence

\[
P(x, \pi) = a_{0\pi_1} \times \prod_i e_{\pi_i}(x_i) \times a_{\pi_i\pi_{i+1}}
\]

Courtesy of Serafim Batzoglou. Used with permission.
Example: One particular P vs. B assignment

\[
P = P(G \mid B)P(B_1 \mid B_0)P(C \mid B)P(B_2 \mid B_1)P(A \mid B)P(P_3 \mid B_2)\ldots P(C \mid B_7)
\]

\[
= (0.85)^3 \times (0.25)^6 \times (0.75)^2 \times (0.42)^2 \times 0.30 \times 0.15
\]

\[
= 6.7 \times 10^{-7}
\]
### One path

1. Scoring $x$, one path
   \[ P(x, \pi) \]
   Prob of a path, emissions

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

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

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

### All paths

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

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

6. Unsupervised learning
   \[ \Lambda^* = \text{argmax}_\Lambda \sum_\pi P(x, \pi | \Lambda) \]
   Baum-Welch training, over all paths
Finding the most likely path

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

1
2
\(\cdots\)
K

\(x_1\)
\(x_2\)
\(x_3\)
\(x_K\)
Calculate maximum $P(x, \pi)$ recursively

**Viterbi algorithm**

Define $V_k(i) = \text{Probability of the most likely path through state } \pi_i = k$

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 $V_k(i) = \max_j \left( \text{this emission} \times \text{Transition from state j} \right)$
The Viterbi Algorithm

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

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

**Iteration:**
\[ V_k(i) = e_K(x_i) \times \max_j a_{jk} V_j(i-1) \]

**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: \( O(KN) \)
<table>
<thead>
<tr>
<th><strong>Decoding</strong></th>
<th><strong>Learning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>One path</td>
<td>All paths</td>
</tr>
<tr>
<td>1. Scoring $x$, one path</td>
<td>2. Scoring $x$, all paths</td>
</tr>
<tr>
<td>$P(x, \pi)$</td>
<td>$P(x) = \sum_{\pi} P(x, \pi)$</td>
</tr>
<tr>
<td>Prob of a path, emissions</td>
<td>Prob of emissions, over all paths</td>
</tr>
<tr>
<td>3. Viterbi decoding</td>
<td>4. Posterior decoding</td>
</tr>
<tr>
<td>$\pi^* = \arg\max_{\pi} P(x, \pi)$</td>
<td>$\pi^\wedge = {\pi_i \mid \pi_i = \arg\max_k \Sigma_{\pi} P(\pi_i = k</td>
</tr>
<tr>
<td>Most likely path</td>
<td>Path containing the most likely state at any time point.</td>
</tr>
<tr>
<td>5. Supervised learning, given $\pi$</td>
<td>6. Unsupervised learning</td>
</tr>
<tr>
<td>$\Lambda^* = \arg\max_{\Lambda} P(x, \pi</td>
<td>\Lambda)$</td>
</tr>
<tr>
<td>Viterbi training, best path</td>
<td>Baum-Welch training, over all paths</td>
</tr>
</tbody>
</table>
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

$$P(x) = \sum_\pi a_{0\pi_1} \prod_i e_{\pi_i}(x_i) \times a_{\pi_i\pi_{i+1}}$$
Calculate total probability $\Sigma_{\pi} P(x, \pi)$ recursively

- Assume we know $f_j$ for the previous time step (i-1)

- Calculate $f_k(i) =$ current sum $e_k(x_i)$ * this emission

  $\sum_j \left( \text{sum ending in state } j \text{ at step } i \times a_{jk} \right)$

  Sum over all previous states $j$
The Forward Algorithm

Input: $x = x_1……x_N$

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

Iteration:
$f_k(i) = e_K(x_i) \times \sum_j a_{jk} f_j(i-1)$

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 $\pi_i$, sum over 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)
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

\( \Rightarrow \) Expanding the number of states
Remember previous nucleotide: expand both states

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

GC-rich: 4 states
Background: 4 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
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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>.180</td>
<td>.274</td>
<td>.426</td>
<td>.120</td>
</tr>
<tr>
<td>C</td>
<td>.171</td>
<td>.368</td>
<td>.274</td>
<td>.188</td>
</tr>
<tr>
<td>G</td>
<td>.161</td>
<td>.339</td>
<td>.375</td>
<td>.125</td>
</tr>
<tr>
<td>T</td>
<td>.079</td>
<td>.355</td>
<td>.384</td>
<td>.182</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>.300</td>
<td>.205</td>
<td>.285</td>
<td>.210</td>
</tr>
<tr>
<td>C</td>
<td>.322</td>
<td>.298</td>
<td>.078</td>
<td>.302</td>
</tr>
<tr>
<td>G</td>
<td>.248</td>
<td>.246</td>
<td>.298</td>
<td>.208</td>
</tr>
<tr>
<td>T</td>
<td>.177</td>
<td>.239</td>
<td>.292</td>
<td>.292</td>
</tr>
</tbody>
</table>
### Examples of HMMs for genome annotation

<table>
<thead>
<tr>
<th>Detection</th>
<th>Detection</th>
<th>Detection</th>
<th>Detection</th>
<th>Detection</th>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>of GC-rich regions</td>
<td>of CpG-rich regions</td>
<td>of conserved regions</td>
<td>of protein-coding exons</td>
<td>of protein-coding conservation</td>
<td>of protein-coding gene structures</td>
</tr>
<tr>
<td>2 states, different nucleotide composition</td>
<td>8 states, different nucleotide composition</td>
<td>2 states, different conservation</td>
<td>2 states, different conservation</td>
<td>2 states, different conservation</td>
<td>~20 states, different chromatin states</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 (intron or intergenic)</td>
<td>First/last/middle coding exon, UTRs, intron1/2/3, intergenic, *(+/- strand)</td>
</tr>
<tr>
<td>Nucleotides</td>
<td>Di-Nucleotides</td>
<td>Level of conservation</td>
<td>Triplets of nucleotides</td>
<td>64x64 matrix of codon substitution frequencies</td>
<td>Codons, nucleotides, splice sites, start/stop codons</td>
</tr>
</tbody>
</table>
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, AATAAA

© Bill Majoros / GeneZilla. All rights reserved. This content is excluded from our Creative Commons license. For more information, see [http://ocw.mit.edu/help/faq-fair-use/](http://ocw.mit.edu/help/faq-fair-use/).
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 $P(x, \pi)$ (algorithm 1)
   - Viterbi algorithm: Find $\pi^* = \operatorname{argmax}_\pi P(x, \pi)$ (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 $\pi_i$, sum over 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
<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>$P(x) = \sum_\pi P(x, \pi)$</td>
</tr>
<tr>
<td>Prob of a path, emissions</td>
<td>Prob of emissions, over all paths</td>
</tr>
<tr>
<td>3. Viterbi decoding</td>
<td>4. Posterior decoding</td>
</tr>
<tr>
<td>$\pi^* = \arg\max_\pi P(x, \pi)$</td>
<td>$\pi^\Lambda = {\pi_i \mid \pi_i = \arg\max_k \sum_\pi P(\pi_i = k \mid x)}$</td>
</tr>
<tr>
<td>Most likely path</td>
<td>Path containing the most likely state at any time point.</td>
</tr>
<tr>
<td>5. Supervised learning, given $\pi$</td>
<td>6. Unsupervised learning</td>
</tr>
<tr>
<td>$\Lambda^* = \arg\max_\Lambda P(x, \pi \mid \Lambda)$</td>
<td>$\Lambda^* = \arg\max_\Lambda \sum_\pi P(x, \pi \mid \Lambda)$</td>
</tr>
<tr>
<td>Viterbi training, best path</td>
<td>Baum-Welch training, over all paths</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

\[ P(\text{Label}_i = B | x) \]
Calculate $P(\pi_7 = \text{CpG+} \mid x_7 = \text{G})$

- **With no knowledge (no characters)**
  - Simply time spent in markov chain states
  - $P(\pi_i = k) = \text{most likely state (prior)}$

- **With very little knowledge (just that character)**
  - Time spent, adjusted for different emission probs.
  - Use Bayes rule to change inference directionality
  - $P(\pi_i = k \mid x_i = \text{G}) = P(\pi_i = \kappa) \times P(x_i = \text{G} \mid \pi_i = k) / P(x_i = \text{G})$

- **With knowledge of entire sequence (all characters)**
  - $P(\pi_i = k \mid x = \text{AGCGCG...GATTATCGTGCATA})$
  - Sum over all paths that emit ‘G’ at position 7

⇒ **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

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

Forward, \( f_k(i) \)  
Backward, \( b_k(i) \)
Define the backward probability:

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

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

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

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

\[ = \sum_l e_l(x_{i+1}) a_{kl} b_l(i+1) \]
Calculate total end probability recursively

- Assume we know $b_i$ for the next time step (i+1)

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

  - current max
  - next emission
  - transition to next state
  - prob sum from state l to end
  - sum over all possible next states
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
  - approximate \( \exp(1+p+q) \)
  - scaling of probabilities

**Running time and space:**
- Time: \( O(K^2N) \)
- Space: \( O(K) \)
Putting it all together: Posterior decoding

- \( P(k) = \frac{f_k(i)b_k(i)}{P(x)} \)
  - Probability that \( i^{th} \) state is \( k \), given all emissions \( x \)

- Posterior decoding
  - Find the most likely state at position \( i \) over all possible hidden paths given the observed sequence \( x \)
  - \( \pi_i^{\hat{\text{a}}} = \arg\max_k P(\pi_i = k | x) \)

- Posterior decoding ‘path’ \( \pi_i^{\hat{\text{a}}} \)
  - 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 $\pi^* = \arg\max_\pi P(x, \pi)$ (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 $\pi_i$, sum over 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
<table>
<thead>
<tr>
<th>Decoding</th>
<th>Scoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>One path</td>
<td>All paths</td>
</tr>
<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_{\pi} P(x, \pi) ]</td>
</tr>
<tr>
<td>Prob of a path, emissions</td>
<td>Prob of emissions, over all paths</td>
</tr>
<tr>
<td>3. Viterbi decoding</td>
<td>4. Posterior decoding</td>
</tr>
<tr>
<td>[ \pi^* = \text{argmax}_{\pi} P(x, \pi) ]</td>
<td>[ \pi^\Lambda = { \pi_i</td>
</tr>
<tr>
<td>Most likely path</td>
<td>Path containing the most likely state at any time point.</td>
</tr>
<tr>
<td>5. Supervised learning, given ( \pi )</td>
<td>6. Unsupervised learning</td>
</tr>
<tr>
<td>[ \Lambda^* = \text{argmax}_\Lambda P(x, \pi</td>
<td>\Lambda) ]</td>
</tr>
<tr>
<td>Viterbi training, best path</td>
<td>Baum-Welch training, over all paths</td>
</tr>
</tbody>
</table>

**Learning**

5. Supervised learning, given \( \pi \)

\[ \Lambda^* = \text{argmax}_\Lambda \max_{\pi} P(x, \pi | \Lambda) \]
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 $x = x_1...x_{1,000,000}$ 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 $P[ x | \theta ]$

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

$\Rightarrow$ Viterbi training:
  guess parameters, find optimal Viterbi path (#2), update parameters (#5), iterate

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

Define:

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

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

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

\[
a_{kl} = \frac{A_{kl}}{\sum_i A_{ki}}
\]

\[
e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)}
\]
If we have a sequence that has islands marked, we can simply count:

\[
P(S|P) = \frac{1}{3} \quad P(S|B) = \frac{2}{3} \quad P(L_{i+1}|L_i) = \begin{array}{cc}
B_{i+1} & P_{i+1} \\
B_i & P_i
\end{array}
\]

Maximum Likelihood Estimation:

\[
P(S|B) = \begin{array}{llll}
A: & 1/5 \\
T: & 0 \\
G: & 2/5 \\
C: & 2/5
\end{array}
\]

ETC..
Case 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 nucleotides, we observe
\[ x = \text{C, A, G, G, T, C, C, A, T, C} \]
\[ \pi = \text{P, P, P, p, p, P, P, P, P, P} \]
Then:
\[ a_{PP} = 1; \quad a_{PB} = 0 \]
\[ e_P(A) = .2; \]
\[ e_P(C) = .4; \]
\[ e_P(G) = .2; \]
\[ e_P(T) = .2 \]
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
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:

<table>
<thead>
<tr>
<th>+</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.180</td>
<td>.274</td>
<td>.426</td>
<td>.120</td>
</tr>
<tr>
<td>C</td>
<td>.171</td>
<td>.368</td>
<td>.274</td>
<td>.188</td>
</tr>
<tr>
<td>G</td>
<td>.161</td>
<td>.339</td>
<td>.375</td>
<td>.125</td>
</tr>
<tr>
<td>T</td>
<td>.079</td>
<td>.355</td>
<td>.384</td>
<td>.182</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.300</td>
<td>.205</td>
<td>.285</td>
<td>.210</td>
</tr>
<tr>
<td>C</td>
<td>.322</td>
<td>.298</td>
<td>.078</td>
<td>.302</td>
</tr>
<tr>
<td>G</td>
<td>.248</td>
<td>.246</td>
<td>.298</td>
<td>.208</td>
</tr>
<tr>
<td>T</td>
<td>.177</td>
<td>.239</td>
<td>.292</td>
<td>.292</td>
</tr>
</tbody>
</table>

\[
\alpha_{st}^+ = \frac{C_{st}^+}{\sum_{t'} C_{st'}^+}
\]
\[
\alpha_{st}^- = \frac{C_{st}^-}{\sum_{t'} C_{st'}^-}
\]

\(C_{st}^+\) is the number of times letter \(t\) followed letter \(s\) inside the CpG islands

\(C_{st}^-\) is the number of times letter \(t\) followed letter \(s\) outside the CpG islands
6: Unsupervised learning

Estimate model parameters based on unlabeled training data
Unlabelled Data

How do we know how to count?

L:

S:

P(L_{i+1}|L_i)

<table>
<thead>
<tr>
<th>B_{i+1}</th>
<th>P_{i+1}</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>B_i</td>
<td>P_i</td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P(S|B)

A:
T:
G:
C:

P(S|P)

A:
T:
G:
C:
An idea:
1. Imagine we start with some parameters
2. We *could* calculate the most likely path, \( P^* \), given those parameters and \( S \)
3. We *could* then use \( P^* \) to update our parameters by maximum likelihood
4. And iterate (to convergence)
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)
<table>
<thead>
<tr>
<th>One path</th>
<th>All paths</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scoring</strong></td>
<td><strong>Scoring</strong></td>
</tr>
<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_{\pi} P(x, \pi) )</td>
</tr>
<tr>
<td>Prob of a path, emissions</td>
<td>Prob of emissions, over all paths</td>
</tr>
<tr>
<td><strong>Decoding</strong></td>
<td><strong>Decoding</strong></td>
</tr>
<tr>
<td>3. Viterbi decoding</td>
<td>4. Posterior decoding</td>
</tr>
<tr>
<td>( \pi^* = \text{argmax}_{\pi} P(x, \pi) )</td>
<td>( \pi^\wedge = {\pi_i \mid \pi_i = \text{argmax}<em>k \sum</em>{\pi} P(\pi_i = k</td>
</tr>
<tr>
<td>Most likely path</td>
<td>Path containing the most likely state at any time point.</td>
</tr>
<tr>
<td><strong>Learning</strong></td>
<td><strong>Learning</strong></td>
</tr>
<tr>
<td>5. Supervised learning, given ( \pi )</td>
<td>7. Unsupervised learning</td>
</tr>
<tr>
<td>( \Lambda^* = \text{argmax}_\Lambda P(x, \pi</td>
<td>\Lambda) )</td>
</tr>
<tr>
<td>6. Unsupervised learning.</td>
<td>Baum-Welch training, over all paths</td>
</tr>
<tr>
<td>( \Lambda^* = \text{argmax}<em>\Lambda \max</em>{\pi} P(x, \pi</td>
<td>\Lambda) )</td>
</tr>
</tbody>
</table>
Simple cases: 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
### One path

1. Scoring $x$, one path
   
   $$P(x, \pi)$$

   Prob of a path, emissions

3. Viterbi decoding
   
   $$\pi^* = \arg\max_{\pi} P(x, \pi)$$

   Most likely path

5. Supervised learning, given $\pi$
   
   $$\Lambda^* = \arg\max_\Lambda P(x, \pi|\Lambda)$$

6. Unsupervised learning
   
   $$\Lambda^* = \arg\max_\Lambda \max_\pi P(x, \pi|\Lambda)$$

   Viterbi training, best path

### All paths

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

   Prob of emissions, over all paths

4. Posterior decoding
   
   $$\pi^\Lambda = \{\pi_i | \pi_i = \arg\max_k \sum_{\pi} P(\pi_i = k|x)\}$$

   Path containing the most likely state at any time point.

6. Unsupervised learning
   
   $$\Lambda^* = \arg\max_\Lambda \sum_{\pi} P(x, \pi|\Lambda)$$

   Baum-Welch training, over all paths
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

EM pervasive in computational biology

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

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

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

   \[ params^t = \arg \max_{params} E_Q \left[ \log P(S, labels | params^{t-1}) \right] \]

4. Iterate

   \[ P(S|Model) \text{ guaranteed to increase each iteration} \]
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 $\theta$

Principle: \textbf{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:

$$P(\pi_i = k, \pi_{i+1} = l \mid x) = \frac{1}{P(x)} \times P(\pi_i = k, \pi_{i+1} = l, x_1 \ldots x_N) = Q/P(x)$$

where $Q = P(x_1 \ldots x_i, \pi_i = k, \pi_{i+1} = l, x_{i+1} \ldots x_N) = P(\pi_{i+1} = l, x_{i+1} \ldots x_N \mid \pi_i = k) P(x_1 \ldots x_i, \pi_i = k)$

So:

$$P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \frac{f_k(i) a_{kl} e_l(x_{i+1}) b_l(i+1)}{P(x \mid \theta)}$$

(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,

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

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 \( N \),

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

Similarly,

\[
E_k(b) = \sum_x (1/P(x)) \sum \{i \mid x^i = b\} f_k(i) b_k(i)
\]
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
<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>$P(x) = \sum_{\pi} P(x, \pi)$</td>
</tr>
<tr>
<td>Prob of a path, emissions</td>
<td>Prob of emissions, over all paths</td>
</tr>
</tbody>
</table>

**Scoring**

3. Viterbi decoding

$$\pi^* = \arg\max_{\pi} P(x, \pi)$$

Most likely path

4. Posterior decoding

$$\pi^\Lambda = \{\pi_i \mid \pi_i = \arg\max_k \sum_{\pi} P(\pi_i=k|x)\}$$

Path containing the most likely state at any time point.

**Decoding**

5. Supervised learning, given $\pi$

$$\Lambda^* = \arg\max_{\Lambda} P(x, \pi | \Lambda)$$

6. Unsupervised learning

$$\Lambda^* = \arg\max_{\Lambda} \sum_{\pi} P(x, \pi | \Lambda)$$

Viterbi training, best path

**Learning**

- Viterbi training, best path
- Baum-Welch training, over all paths
### 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 each +/-, different transition probabilities</td>
<td>2 states, different conservation levels</td>
<td>2 states, different trinucleotide composition</td>
<td>2 states, different evolutionary signatures</td>
<td>~20 states, different composition/conservation, specific structure</td>
<td>40 states, different chromatin mark combinations</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 (intron or intergenic)</td>
<td>First/last/middle 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 conservation</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>
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
    - 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 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)
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 $\pi_i$, sum over 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