Hidden Markov models (HMMs)

1. S - observations
   - $x_1, ..., x_n$ – sequence of observations
2. Q - states
   - $\pi_1, ..., \pi_n$ – hidden sequence of states
   - $f = (f_1, ..., f_n)^T$ - initial probability of states
3. A = $(a_{ij})$ – transition matrix
4. E = $(e_i(x))$ – emission probabilities

Hidden State

- We will distinguish between observed parts of a problem and hidden parts
- In the Markov models we have considered previously, it is clear which state accounts for each part of the observed sequence
- In an HMM there are multiple states that could account for each part of the observed sequence
  - this is the hidden part of the problem
  - states are decoupled from sequence symbols
Prokaryote gene prediction

Gene Prediction using Hidden Markov Models

It has been shown that the gene content and length distribution of prokaryotic genes can be either typical or atypical. Typical genes are in the range of 100 to 500 amino acids with a nucleotide distribution typical of the organism. Atypical genes are shorter or longer with different nucleotide statistics. These genes tend to escape detection using the typical gene model. This means that, to make the algorithm capable of fully describing all genes in a genome, more than one Markov model is needed.

Combining different Markov models that represent typical and atypical nucleotide distributions leads to a HMM prediction algorithm.

Using a Hidden Markov Model

• An HMM allows the calculation of the probability of an input sequence, i.e., you can calculate how likely it is that the sequence is generated by the model.
• For a given input sequence (in the example starting with a start and ending with a stop codon), each sequence symbol maps to a state in the HMM and going to the next state proceeds via transition probabilities (the arrows in the picture). The probability is calculated by taking the product of the emission probabilities (the probability of each sequence symbol in the state to which it maps) and the transition probabilities (when you move to a next sequence symbol).
• There typically are very many different pathways for a sequence through the HMM, one (or a number) of which will lead to an optimal score for that sequence.
  - The calculation of the total probability of a sequence corresponds to summing the probabilities of all possible pathways through the model.
  - Finding the optimal path and associated score is called the decoding problem.

HMM prokaryote gene prediction methods

A number of HMM-based gene finding programs exist for prokaryotic organisms.

• GeneMark (http://opal.biology.gatech.edu/GeneMark/) is a suite of gene prediction programs based on the fifth-order HMMs.
  - The main program—GeneMark.hmm—is trained on a number of complete microbial genomes. If the sequence to be predicted is from a nonlisted organism, the most closely related organism can be chosen as the basis for computation.
  - Another option for predicting genes from a new organism is to use a self-trained program GeneMarkS as long as the user can provide at least 100 kbp of sequence on which to train the model.
  - If the query sequence is shorter than 100 kbp, a GeneMark heuristic program can be used with some loss of accuracy.
  - In addition to predicting prokaryotic genes, GeneMark also has a variant for eukaryotic gene prediction using HMM.

• FGENESB (www.softberry.com/berry.phtml?topic=gfindb) is a web-based program that is also based on fifth-order HMMs for detecting coding regions. The program is specifically trained for bacterial sequences. It uses the Vertibi algorithm (see later) to find an optimal match for the query sequence with the intrinsic model. A linear discriminant analysis (LDA) is then used to further distinguish coding signals from noncoding signals.
HMM for Eukaryotic Gene Finding

To allow for splicing in three different frames three intron models are needed. To get the frame correct "spacer states" are added before and after the intron models.

The Parameters of an HMM

- as in Markov chain models, we have transition probabilities
  \[ a_{ij} = \Pr(\pi_j = l \mid \pi_{j-1} = k) \]
  probability of a transition from state \( k \) to \( l \)
- \( \pi \) represents a path (sequence of states) through the model
- since we’ve decoupled states and characters, we might also have emission probabilities
  \[ e_k(b) = \Pr(x_i = b \mid \pi_i = k) \]
  probability of emitting character \( b \) in state \( k \)

A Simple HMM

The Parameters of an HMM

- as in Markov chain models, we have transition probabilities
  \[ a_{ij} = \Pr(\pi_j = l \mid \pi_{j-1} = k) \]
  probability of a transition from state \( k \) to \( l \)
- \( \pi \) represents a path (sequence of states) through the model
- since we’ve decoupled states and characters, we might also have emission probabilities
  \[ e_k(b) = \Pr(x_i = b \mid \pi_i = k) \]
  probability of emitting character \( b \) in state \( k \)

Three basic problems of HMMs

Once we have an HMM, there are three problems of interest.

1. **The Decoding Problem** – what is the most probable path?
   - Given a model and a sequence of observations, what is the most likely state sequence in the model that produced the observations?

2. **The Evaluation Problem** – how likely is a given sequence?
   - Given an HMM and a sequence of observations, what is the probability that the observations are generated by the model?

3. **The Learning Problem** – how to set the HMM parameters?
   - Given a model and a sequence of observations, how should we adjust the model parameters in order to maximize the generation of the (training) data

Three Important Questions

- What is the most probable “path” for generating a given sequence?
  - Given \( A+E+(x_1, ..., x_n) \) what is \((\pi_1, ..., \pi_n)\)?
    - the Viterbi algorithm

- How likely is a given sequence?
  - Given \( A+E \) what is the probability of \((x_1, ..., x_n)\)?
    - the Forward algorithm

- How can we learn the HMM parameters given a set of sequences?
  - Given \((x_1, ..., x_n)\) what is \( A+E \)?
    - the Forward-Backward (Baum-Welch) algorithm

Three Important Questions

- What is the most probable “path” for generating a given sequence?
  - **Viterbi algorithm**
    - How likely is a given sequence?
    - How can we learn the HMM parameters given a set of sequences?
Finding the Most Probable Path: The Viterbi Algorithm

- Define \( v_k(i) \) to be the probability of the single most probable path accounting for the first \( i \) characters of \( x \) and ending in state \( k \).
- We want to compute \( v_k(L) \), the probability of the most probable path accounting for all of the sequence and ending in the end state.
- Can be defined recursively.
- Can use Dynamic Programming to find \( v_k(L) \) efficiently.

Recap: Dynamic Programming

For each cell you must check three cells that 'transition' into it.

With HMMs, it is possible that many more 'cells' (now called states) should be checked.

\[
H(i,j) = \max \left( \begin{array}{c}
H(i-1,j-1) + s(i,j) \\
H(i-1,j) - g \\
H(i,j-1) - g
\end{array} \right)
\]

This is a recursive formula.

Given a path, what is the score of a sequence generated by taking this path?

- The probability that the path is taken and the sequence is generated:

\[
Pr(x_1,...,x_N,\pi_0,...,\pi_N) = a_{0i} \prod_{j=1}^{j} e_{\pi_j}(x_j)a_{\pi_j\pi_{j+1}}
\]

- (assuming begin/end are the only silent states on path)

Pr (AAC, \( \pi = \pi_0 \pi_1 \pi_3 \pi_5 \)) = a_{01} \times e_1(A) \times a_{13} \times e_3(A) \times a_{35} \times e_3(C) = 0.5 \times 0.4 \times 0.2 \times 0.4 \times 0.8 \times 0.3 = 0.002304

Pr (AAC, \( \pi = \pi_0 \pi_1 \pi_3 \pi_3 \pi_5 \)) = a_{01} \times e_1(A) \times a_{13} \times e_3(A) \times a_{33} \times e_3(C) \times a_{35} = 0.5 \times 0.4 \times 0.8 \times 0.2 \times 0.4 \times 0.3 \times 0.9 = 0.004608

Finding the Most Probable Path: The Viterbi Algorithm

Viterbi – recursive step

What is the probability of the path which ends with \( q_A \rightarrow q_B \) and emission \( E_B \)?

Viterbi – recursive step

What is the most probable path to the state B in step \( i \)?

\[
V(i, q) = e_q(x_i) \cdot \max_s V(i-1, s) a_{s,q}
\]
Finding the Most Probable Path: The Viterbi Algorithm

Initialization:
\[ v_0(0) = 1 \text{ (start), } v_k(0) = 0 \text{ for } k > 0 \]

Recursion for emitting states \((l = 1 \ldots L)\):
\[ v_l(i) = e_i(x_i) \max_k [v_k(i-1)a_{kl}] \]
\[ \text{ptr}(i) = \arg \max_k [v_k(i-1)a_{kl}] \]

Recursion for silent states:
\[ v_l(i) = \max_k [v_k(i)a_{lk}] \]
\[ \text{ptr}(i) = \arg \max_k [v_k(i)a_{lk}] \]

\( \pi_l \) is a silent state, so no emission probability, and sequence stays at same position

Probability that we are in the end state and have observed the entire sequence using the highest scoring path

### Viterbi algorithm example

**Given the sequence** \( x = \text{TAGA} \)

**Initialization**
\[ v_0(0) = 1 \quad v_1(0) = 0 \quad \ldots \quad v_5(0) = 0 \]

**Computing other values**
\[ v_1(1) = 0.3 \times \max \{ 1 \times 0.5, 0 \times 0.2 \} = 0.15 \]
\[ v_2(1) = 0.4 \times \max \{ 1 \times 0.5, 0 \times 0.8 \} = 0.2 \]
\[ v_2(2) = 0.4 \times \max \{ 0.1 \times 0.5, 0.15 \times 0.2 \} = 0.012 \]

\( \Pr(\text{TAGA}) = v_5(4) = \max \{ v_3(4) \times a_{35}, v_4(4) \times a_{45} \} \)

### Finding the Most Probable Path:
The Viterbi Algorithm

- **Termination:**
  \[ \Pr(x, \pi) = \max_k (v_k(L)a_{kk}) \]
  \[ \pi_L = \arg \max_k (v_k(L)a_{kk}) \]

- **Traceback:** follow pointers back starting at \( \pi_L \)

### Calculating probabilities using the Viterbi algorithm

- Multiplying many (small) probabilities will lead to underflow and other computational problems
- Therefore, take the log of all probabilities and use addition instead of multiplication
  - Remember that \( \log(ab) = \log(a) + \log(b) \)
- Many CPUs are faster at adding numbers than multiplying them

### Finding the Most Probable Path: The Viterbi Algorithm

Andrew Viterbi used Manhattan grid model to solve this **decoding** problem.

- Every choice of \( \pi = \pi_1 \ldots \pi_n \) corresponds to a path in the graph.
- Only valid direction in the graph is **eastward**.
- This graph has \( N^2(n-1) \) edges, where \( N \) is number of states
Three Important Questions

• What is the most probable “path” for generating a given sequence?
• How likely is a given sequence?
• How can we learn the HMM parameters given a set of sequences?

How Likely is a Given Sequence?

• The probability that the path is taken and the sequence is generated:
  \[ \Pr(x_1...x_L, \pi_0...\pi_N) = a_{\text{begin}} \prod_{l=1}^{L} c_{x_l}(\pi_l) a_{x_l,\pi_{l+1}} \]
  (assuming begin/end are the only silent states on path)

How Likely is a Given Sequence:

This is the probability of sequence AAC using path \( (0,1,1,3,5) \) – but many alternative paths are possible

How Likely is a Given Sequence:

The probability over all paths is:
  \[ \Pr(x_1...x_L) = \sum_{\pi} \Pr(x_1...x_L, \pi_0...\pi_N) \]
  but the number of paths can be exponential in the length of the sequence...
  • Nonetheless, we need to calculate the probability of a given sequence by summing the probabilities over all paths (producing the query sequence) through the HMM
  • Fortunately, the Forward algorithm enables us to compute this efficiently

How Likely is a Given Sequence: The Forward Algorithm

• Define \( f_k(i) \) to be the probability of being in state \( k \)
• Having observed the first \( i \) characters of \( x \) we want to compute \( f_k(L) \), the probability of being in the end state having observed all of \( x \)
• We can define this recursively

How Likely is a Given Sequence:

• because of the Markov property, don’t have to explicitly enumerate every path – use dynamic programming instead
  e.g. compute \( f_k(i) \) using \( f_k(i-1), f_k(i-1) \)
The forward algorithm

- **Initialisation:**
  - probability that we’re in start state and have observed 0 characters from the sequence
  - $f_0(0) = 1$ (start state), $f_k(0) = 0$ for $k > 0$ (other states $k$)

- **Recursion:**
  - $f(l) = \sum_k f(k) a_{kl}$ (emitting states),
  - $f(l) = \sum_k f(k) a_{kl}$ (if state $l$ is silent)

- **Termination:**
  - $Pr(x) = Pr(s_1, ..., s_L) = f(N)$

**Note:** Durbin et al. (course book) use 0 to denote the end state and $a_{0k}$ to designate the transition probability to the end state.

Forward algorithm example

- given the sequence $x = \text{TAGA}$
- initialisation
  - $f_0(0) = 1$  $f_0(0) = 0$  $f_1(0) = 0$
- computing other values
  - $f_1(1) = 0.3 \times (1 \times 0.5 + 0 \times 0.2) = 0.15$
  - $f_1(1) = 0.4 \times (1 \times 0.5 + 0 \times 0.8)\quad\text{in the T in the sequence}$
  - $f_2(2) = 0.4 \times (0 \times 0.5 + 0.15 \times 0.2)\quad\text{in the A in the sequence}$

Calculating probabilities using the Forward algorithm

- Multiplying many (small) probabilities will lead to underflow and other computational problems
- However, just taking the log of probabilities is not possible because the log of a sum of probabilities must be calculated for the Forward algorithm (so exponents and log function should be calculated which are computationally expensive)
- Durbin et al. (Section 3.6) give two ways to deal with this problem

The Forward and Viterbi Algorithm

- The Forward algorithm is to calculate the total probability of a sequence $x$ being generated by the model (by summing over all paths)
- The Viterbi algorithm is to find the optimal path of the sequence through the model (decoding problem)
- Both algorithms can be implemented using Dynamic Programming
  - With only one small difference: change max to $\Sigma$ (use max in Viterbi and summing in Forward algorithm)

Three Important Questions

- What is the most probable “path” for generating a given sequence?
- How likely is a given sequence?
- How can we learn the HMM parameters given a set of sequences?

The Learning Problem

Generally, the learning problem is how to adjust the HMM parameters, so that the given set of observations (called the training set) is represented by the model in the best way for the intended application.

Thus it would be clear that the “quantity” we wish to optimize during the learning process can be different from application to application. In other words there may be several optimization criteria for learning, out of which a suitable one is selected depending on the application.

There are two main optimization criteria found in the literature; Maximum Likelihood (ML) and Maximum Mutual Information (MMI).
The Learning Task

- Given:
  - a model
  - a set of sequences (the training set)

- Do:
  - find the most likely parameters to explain the training sequences

- The goal is find a model that generalizes well to sequences we haven’t seen before

Learning Parameters

- If we know the state path for each training sequence, learning the model parameters is simple:
  - no hidden state during training
  - count how often each parameter is used
  - normalize/smooth to get probabilities
  - process just like it was for Markov chain models

- If we don’t know the path for each training sequence, how can we determine the counts?
  - key insight: estimate the counts by considering every path weighted by its probability

Learning Parameters if we don’t know the path for each training sequence:

The Baum-Welch Algorithm

- An EM (expectation maximization) approach, iteratively applying the forward-backward algorithm
- Algorithm sketch:
  - initialize parameters of model (e.g. using a prior)
  - iterate until convergence
  - calculate the expected number of times each transition or emission is used (expectation step)
  - adjust the parameters to maximize the likelihood of these expected values (maximization step)
  - Baum-Welch has as important feature that it always converges

The Expectation step

- we want to know the probability of producing sequence $x$ with the $i$th symbol being produced by state $k$ (for all $x$, $i$ and $k$)

- the forward algorithm gives us $f_k(i)$, the probability of being in state $k$ having observed the first $i$ characters of $x$

The Expectation step

- just generate the training sequences and count the number of emissions of a given symbol in state $k$ and make it a probability by normalising using the total number of emissions of state $k$

Note: Durbin et al. use a slightly different notation but perform exactly the same calculation

Learning Parameters

- If we know the state path for each training sequence, learning the model parameters is simple:

  $e_k(c) = \frac{n_{k,c}}{\sum c_i}$

  $a_{ij} = \frac{n_{k \rightarrow l}}{\sum n_{k \rightarrow m}}$

  just generate the training sequences and count the number of emissions of a given symbol in state $k$ and make it a probability by normalising using the total number of emissions of state $k$

  just generate the training sequences and count the number of transitions from state $k$ to $l$ and make it a probability through dividing by the total number of transitions out of state $k$
The Expectation step

- the forward algorithm gives us \( f_i(1) \), the probability of being in state \( k \) having observed the first \( i \) characters of \( x \).

\[
\Pr(\pi_i = k, x) = \Pr(x_i, \ldots, x_1, \pi_i = k) \times \Pr(x_{i+1}, \ldots, x_L | \pi_i = k)
\]

The first term is \( f_i(1) \), computed by the forward algorithm.

The second term is \( b_i(1) \), computed by the backward algorithm.

The Expectation step

- the backward algorithm gives us \( b_i(1) \), the probability of observing the rest of \( x \), given that we're in state \( k \) after \( i \) characters.

\[
\Pr(\pi_i = k / x)
\]

Given this we can compute our expected counts for state transitions and character emissions.

The Expectation step

- putting forward and backward together, we can compute the probability of producing sequence \( x \) with the \( i \)th symbol being produced by state \( q \).

\[
\Pr(\pi_i = q / x) = \frac{f_i(1) \cdot b_i(1)}{\sum_q f_i(1) \cdot b_i(1)}
\]

The Backward Algorithm

\[
b_j(1) = \begin{cases} a_{k} & \text{if } f_j(1) = 0 \\ a_{k} \pi_j(x_j) b_j(1) & \text{if } f_j(1) \neq 0 \end{cases}
\]

The termination step is rarely needed since the total probability of a sequence is normally calculated using the Forward algorithm, but it can be a nice test to check your algorithms.
The Expectation step

- now we can calculate the probability of the $i$th symbol being produced by state $k$, given $x$

$$
\Pr(\pi_i = k \mid x) = \frac{\Pr(\pi_i = k, x)}{\Pr(x)} = \frac{f_k(i)b_k(i)}{\Pr(x)}
$$

The Maximization step

- Let $n_{k,c}$ be the expected number of emissions of $c$ from state $k$ for the training set
- estimate new emission parameters by:

$$
e_k(c) = \frac{n_{k,c}}{\sum_c n_{k,c}}$$

- just like in the simple case
- but typically we’ll do some “smoothing” (e.g. add pseudocounts) to avoid overfitting

The Baum-Welch Algorithm

- Initialize parameters of model (for example using priors)
- Iterate (expectation and maximization) until convergence (using the forward-backward algorithm):
  - calculating the expected number of times each transition or emission is used
  - adjusting the parameters to maximize the likelihood of these expected values
- This algorithm will likely converge to a local maximum (in the likelihood of the data given the model). To avoid this problem it is recommendable to train multiple times with different parameter settings (or priors)
- Convergence is reached usually in a fairly small number of iterations
Using HMMs in sequence analysis

Pairwise sequence alignment

Model for pairwise alignment using affine penalties
- Finite deterministic automaton (FDA) on the left and the corresponding probabilistic version on the right

HMM-based homology searching

This is a HMM for ungapped alignment with Transition and Emission probabilities

Gapped HMMs also have insertion and deletion states (next slide)

Profile HMM: m=match state, I=insert state, d=delete state; go from left to right. I and m states output amino acids; d states are "silent".

HMM-based profile

An alignment of 30 short amino acid sequences chopped out of an alignment of the SH3 domain. The shaded areas are the most conserved and were chosen to be represented by the main (match) states in the HMM. The unshaded area with lower-case letters was chosen to be represented by an insert state.

Note: these choices make setting up an HMM a work of art

HMM-based profile

A profile HMM made from the preceding alignment. Transition lines with no arrow head are transitions from left to right. Transitions with very small probability are shown as dashed lines. Transitions from an insert state to itself are not shown; instead the probability times 100 is shown in the diamond. The numbers in the circular delete states are just position numbers.

HMM-based profile

An example of two sequences that are matched to states of the HMM; the resulting alignment is on the right
At what level of sequence identity can I trust my alignment to have matched two homologous sequences?

HMM-based homology searching

- Most widely used HMM-based profile searching tools currently are SAM-T2K (Karplus et al., 1998) and HMMER2 (Eddy, 1998).
- Formal probabilistic basis and consistent theory behind gap and insertion scores.
- HMMs are good for profile searches, but still not so good for alignment (due to parametrization of the models).
- HMMs are slow, but recent programs are speed optimised and can handle large sequence sets.
- A number of HMM-to-HMM alignment programs have been developed, and these are currently state of the art in sensitive alignment of distantly homologous sequence sets.

HMM-based homology searching

- A number of HMM-to-HMM alignment programs have been developed, and these are currently state of the art in sensitive alignment of distantly homologous sequence sets:
  - PRC (Madeira; Brandt and Heringa, 2009) is a stand-alone program for aligning and scoring two profile hidden Markov models. This can be used to detect remote relationships between profiles more effectively than by doing simple profile-sequence comparisons. PRC takes into account all transition and emission probabilities in both hidden Markov models. The fundamental algorithm is symmetric, so prc HMM1 HMM2 is basically the same as prc HMM2 HMM1, although the reverse null and to a much larger extent the E-value fitting process are currently asymmetric.
  - PRC can read SAM, HMMER, PSI-BLAST and FASTA files. If you have a multiple sequence alignment and want to estimate a profile HMM from it, the SAM package is recommended, in particular the script w0.5.

HMM Software

- HMMER
  - Sean Eddy; Wash. Univ. St. Louis
  - http://hmmer.wustl.edu
  - commands: hmmalign, hmmssearch
  - ls/fs, calibration
- other software: SAM
  - http://www.cse.ucsc.edu/research/compbio/sam.html

Practical HMM

Protein domain prediction

A protein sequence of 300 amino acids or more typically folds into a multi-domain protein structure where structural domains are intervened by so-called linker regions. A protein sequence KL..VPQPSMPWH..IS might fold into DD..DLLLLLDD..DD (where the dots denote a number of residues in D state).

- You will execute the Forward, Viterbi and Baum-Welch algorithms in the practical.
- The best way to learn about an algorithm is by doing...

References

- Seminal review by Lawrence R. Rabiner (1989) on BB
- bioalgorithms.info
- Wikipedia
- Durbin’s book Biological sequence Analysis (course book)
- Book Understanding Bioinformatics