The Baum-Welch algorithm proceeds as follows:

Initialise: Set starting values for the parameters. Set log-likelihood to $-\infty$.

Iterate:

1. Set A and E to their pseudo count values. For each training sequence x^j:
   (a) Calculate $f_k(i)$ for x^j from forward algorithm
   (b) Calculate $b_k(i)$ for x^j from backward algorithm
   (c) Calculate A^j and E^j and add to A and E using Equations 2 and 3 above.
2. Set new values for a and e, based on A and E
3. Calculate log-likelihood of model
4. If change in log likelihood is small, stop, else, continue.

See lecture slides for a detailed example of applying the Baum-Welch algorithm.

18.7.1 Comments on the Baum-Welch algorithm

The Baum-Welch algorithm is guaranteed to converge to the local maximum — exactly which local maximum it converges to depends on the initial state. Any local maximum is not necessarily the global maximum so the algorithm should be run from multiple different start states to check that a global maximum has been found.

Also remember that convergence is only guaranteed in the limit of an infinite number of iterations so the exact local maximum is never achieved.

The algorithm is a type of Expectation-Maximisation (EM) algorithm that is widely used for maximum likelihood estimation.

18.8 Sampling state paths

The probabilities $f_k(i)$ we calculate in the forward algorithm can be used to sample possible state paths in proportion to their probability. Recall that the Viterbi algorithm provides a method of finding the most probable state path by tracing back though the a matrix, taking the direction that led us to the highest score at each point. We adopt the traceback idea but apply it to the matrix $f_k(i)$ and at each step of the traceback, we choose the state in proportion to the amount it contributed to the current probability.

Assuming an end state is not modelled, the probability of a sequence x is $P(x) = \sum_k f_k(L)$. So the probability that the last state is k is given by

$$P(\pi_L = k|x) = \frac{f_k(L)}{\sum_i f_i(L)}.$$

Now, suppose we are in state l at position $i+1$. We know from the forward algorithm that

$$f_l(i+1) = e_l(x_{i+1}) \sum_k f_k(i) a_{kl}.$$

Thus we move to state k in the ith position with probability

$$\frac{f_k(i)a_{kl}}{\sum_j f_j(i)a_{jl}} = \frac{e_l(x_{i+1})f_k(i)a_{kl}}{f_l(i+1)}.$$

Depending on what you have stored in your algorithm, it may be easier to work with either the left or right hand side of this equation.

Example: Looking again at the CpG island example, sample state paths according to their posterior probabilities for the given sequence $x = TACA$.

Solution: First, get the forward matrix, f. To make it simple, don't use the log transform:

	-	T	A	C	A
0	1				
H	0	0.075	0.014625	0.007914375	0.0009567281
L	0	0.150	0.038250	0.006052500	0.0022766062

Now, $P(x) = \sum_k f_k(L)a_{k0} = 0.003233334$ where $a_{k0} = 1$. So simulate the 4th element of the state path by drawing from $\{H, L\}$ with probabilities $\{\frac{f_H(4)}{P(x)} = 0.2958952, \frac{f_L(4)}{P(x)} = 0.7041048\}$, respectively. Suppose we sampled H. Then the 3rd element of the state path is a draw from $\{H, L\}$ with respective probabilities

$$\left\{\frac{f_H(3)a_{HH}}{f_H(3)a_{HH} + f_L(3)a_{LH}} = 0.6204251, \frac{f_L(3)a_{LH}}{f_H(3)a_{HH} + f_L(3)a_{LH}} = 0.3795749\right\}$$

Suppose we sampled H again. WE now repeat the process, sampling from

$$\left\{\frac{f_H(2)a_{HH}}{f_H(2)a_{HH} + f_L(2)a_{LH}}, \frac{f_L(2)a_{LH}}{f_H(2)a_{HH} + f_L(2)a_{LH}}\right\}$$

and so on. We'll end up with a state path, for example, $LLHH$. □

18.9 HMM model structure

Defining the correct structure, or 'topology', of an HMM is crucial to good estimation but there are no solid rules for doing so. Usually data is limited so we can't over-parametrise otherwise our estimation algorithms with never find decent values. So we can't simply allow all possible transitions and let the computer estimate the correct model. We must decide ourselves, as much as possible, which transition we allow (so that $a_{kl} > 0$) and which we disallow (by setting $a_{kl} = 0$).

18.9.1 Duration modeling

If we want to accurately model the length of a sequence along with the contents, we must model an end state as well as the start state. The basic end state, which is connected to every other state, and has transition probability $q = 1 - p$ produces a sequence of length l with probability

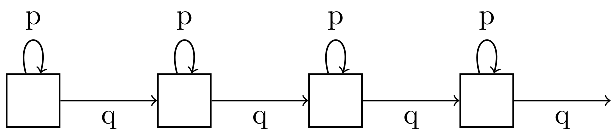
$$P(L = l) = qp^{l-1}.$$

This is a geometric distribution and is the discrete analogue of the exponential distribution. In general, it is not a very good model for lengths and is used largely for the convenience of its mathematical form.

Also note that the length of time an HMM spends in any one state where the probability of leaving that state is q is geometric.

An easy way to get a more flexible and, perhaps, more realistic distribution of lengths is to have, say, n copies of the HMM linked together an it stays in each one for a geometrically distributed number of steps.

An example with 4 states linked together (the states here could be HMMs themselves).



This produces a negative binomial length distribution, so that

$$P(L = l) = \binom{l-1}{n-1} p^{l-n} q^n.$$

19 Applications of HMMs in bioinformatics

19.1 Pairwise alignment with HMMs

We saw that we could tackle the pairwise alignment problem with finite state automata. We now tackle the problem using an HMM, which is sometimes called a stochastic FSA.

We define a *pair HMM* as emitting a pair of sequences (x, y) as opposed to the standard HMMs we have considered so far that emit a single sequence.

The basic HMM which produces a global alignment has three states, X, Y and M. M emits a match, X emits a residue from sequence x and a gap in sequence y (an insertion in x relative to y), while Y emits a residue from y and a gap in x. Emission probabilities for states M, X, Y are $p_{x_i y_j}$, q_{x_i} and q_{y_j}, respectively. We also include begin and end states, B and E. Non-zero transition probabilities are $a_{MX} = a_{MY} = \delta$, $a_{XX} = a_{YY} = \epsilon$, $a_{BX} = a_{BY} = \delta$, $a_{kE} = \tau$ for any k, and $a_{kM} \neq 0$ for $k \neq E$ and can be calculated using the fact that $\sum_k a_{ik} = 1$.

All the algorithms we saw for standard HMMs will work for these pair HMMs but we need a little bit of accounting for the 2 sequences — instead of $v_k(i)$ or $f_k(i)$, we need to work with $v_k(i, j)$ or $f_k(i, j)$ etc where k is one of the 3 states M, X or Y. In each case, we keep 3 score matrices instead of 1 to keep track of which state we are in at every point in the alignment.

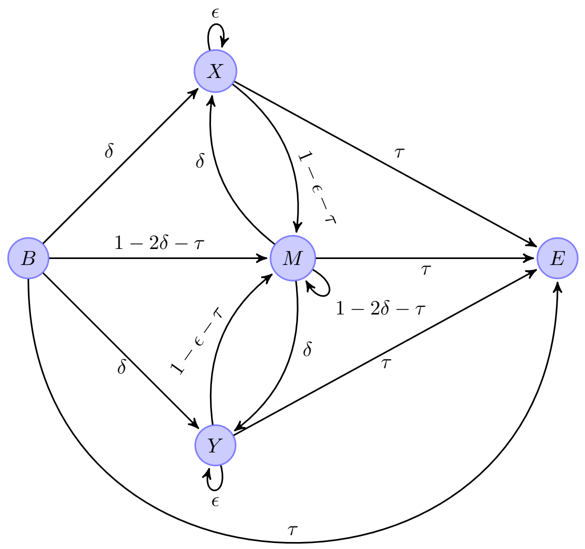


Figure 8: A pair HMM model for global alignment. Emission probabilities for states M, X, Y are $p_{x_i y_j}$, q_{x_i} and q_{y_j}, respectively. Compare it to the simpler FSA in Figure 4.

Durbin et al show how the parameters in these pair HMMs work within the Viterbi algorithm to produce exact analogues of the dynamic programming algorithms we saw earlier.

For example, to the standard quantities we use in the Needleman-Wunsch algorithm from the Viterbi HMM formulation of global alignment, set

$$\begin{aligned} s(a,b) &= \log \frac{p_{ab}}{q_a q_b} + \log 1 - 2\delta - \tau(1-\eta)^2 \\ d &= -\log \frac{\delta(1-\epsilon-\tau)}{(1-\eta)(1-2\delta-\tau)} \\ e &= -\log \frac{\epsilon}{1-\eta}. \end{aligned}$$

These pair HMMs give us more than just another way of viewing the basic alignment algorithms. Since they are couched in the language of probability, we can answer questions about alignments with more depth and nuance than we can with the standard deterministic tools.