11
Hidden Markov Models

We divide this brief account of hidden Markov models into three sections: (i) a description of the properties of these models, (ii) the three main algorithms of the models, (iii) applications. For a more complete account of these models, see Rabiner (1989).

11.1 What is a Hidden Markov Model?

A hidden Markov model (HMM) is similar to a Markov chain, but is more general, and hence more flexible, allowing us to model phenomena that we cannot model sufficiently well with a regular Markov chain model. An HMM is a discrete-time Markov model with some extra features. The main addition is that when a state is visited by the Markov chain, the state "emits" a letter from a fixed time-independent alphabet. Letters are emitted via a time independent, but usually state-dependent, probability distribution over the alphabet. When the HMM runs there is, first, a sequence of states visited, which we denote by $q_1, q_2, q_3, \ldots$, and second, a sequence of emitted symbols, denoted by $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \ldots$. Their generation can be visualized as a two-step process as follows:

$$\underset{q_1}{\text{initial}} \rightarrow \underset{\mathcal{O}_1}{\text{emission}} \rightarrow \underset{\text{to } q_2}{\text{transition}} \rightarrow \underset{\mathcal{O}_2}{\text{emission}} \rightarrow \underset{\text{to } q_3}{\text{transition}} \rightarrow \underset{\mathcal{O}_3}{\text{emission}} \rightarrow \cdots$$

We denote the entire sequence of q_i's by Q and the entire sequence of $\mathcal{O}_i$'s by $\mathcal{O}$, and we write "the observed sequence $\mathcal{O} = \mathcal{O}_1, \mathcal{O}_2, \ldots$" and "the state sequence $Q = q_1, q_2, \ldots$."

Often we know the sequence $\mathcal{O}$ but do not know the sequence Q. In such a case the sequence Q is called "hidden." An important feature of HMMs is that we can efficiently answer several questions about $\mathcal{O}$ and Q.

One of these questions concerns the estimation of the hidden state sequence that has the highest probability given the observed sequence. We illustrate this with a simple example. Consider the Markov chain with two states S_1 and S_2, with uniform initial distribution and transition matrix

$$\begin{bmatrix} .9 & .1 \\ .8 & .2 \end{bmatrix}.$$

Let A be an alphabet consisting only of the numbers 1 and 2. State S_1 emits a 1 or 2 with equal probability $\frac{1}{2}$, state S_2 emits a 1 with probability $\frac{1}{4}$ and a 2 with probability $\frac{3}{4}$. Suppose the observed sequence is $\mathcal{O} = 2, 2, 2$. What sequence of states $Q = q_1, q_2, q_3$ has the highest probability given $\mathcal{O}$? In other words, what is

$$\operatorname*{argmax}_Q \text{Prob}(Q \mid \mathcal{O})?$$

There are eight possibilities for Q. Each of these can be written down and its probability calculated, and from this it is found that the answer to the above question is $Q = S_2, S_1, S_1$. The sequence Q contains more S_1's, even though S_2 is more likely to produce a 2 when visited (probability $\frac{3}{4}$) than S_1 (probability $\frac{1}{2}$). The reason is because S_1 is much more likely to be visited than S_2 ($p_{11} = .9$ and $p_{21} = .8$).

We can also calculate

$$\text{Prob}(\mathcal{O}) = \sum_Q \text{Prob}(\mathcal{O} \mid Q) \cdot \text{Prob}(Q). \tag{11.1}$$

This calculation is useful in distinguishing which of several models is most likely to have produced $\mathcal{O}$.

In the above example all of these calculations can be done by hand. However, models arising in practice have many states, sometimes hundreds, and an alphabet with many symbols (often 20, one for each amino acid). In these cases, calculation of the quantities above by exhaustive methods becomes impossible even for the fastest computers. Fortunately, there are dynamic programming approaches that overcome this problem, which we discuss in detail below. Before turning to the algorithms, however, it is necessary to introduce some specific notation. An HMM will consist of the following five components:

(1) A set of N states $S_1, S_2, \ldots, S_N$.

(2) An alphabet of M distinct observation symbols $A = \{a_1, a_2, \ldots, a_M\}$.

(3) The transition probability matrix $P = (p_{ij})$, where

$$p_{ij} = \text{Prob}(q_{t+1} = S_j \mid q_t = S_i)$$

(4) The emission probabilities: For each state S_i and a in A,

$$b_i(a) = \text{Prob}(S_i \text{ emits symbol } a).$$

The probabilities $b_i(a)$ form the elements in an $N \times M$ matrix $B = (b_i(a))$.

(5) An initial distribution vector $\pi = (\pi_i)$, where $\pi_i = \text{Prob}(q_1 = S_i)$.

Components 1 and 2 describe the structure of the model, and 3–5 describe the parameters. It is convenient to let $\lambda = (P, B, \pi)$ represent the full set of parameters. We can now describe the main algorithms.

11.2 Three Algorithms

There are three calculations that are frequently required in HMM theory. Given some observed output sequence $\mathcal{O} = \mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_T$, these are:

(i) Given the parameters λ, efficiently calculate

$$\text{Prob}(\mathcal{O} \mid \lambda).$$

That is, efficiently calculate the probability of some given sequence of observed outputs.

(ii) Efficiently calculate the hidden sequence $Q = q_1, q_2, \ldots, q_T$ of states that is most likely to have occurred, given $\mathcal{O}$. That is, calculate

$$\underset{Q}{\text{argmax}}\ \text{Prob}(Q \mid \mathcal{O}).$$

(iii) Assuming a fixed topology of the model (i.e., a fixed graph structure of the underlying Markov chain, as defined in 4.8), find the parameters $\lambda = (P, B, \pi)$ that maximize $\text{Prob}(\mathcal{O} \mid \lambda)$.

We address these problems in turn.

11.2.1 The Forward and Backward Algorithms

We first consider problem (i). The naive way of calculating $\text{Prob}(\mathcal{O})$ is to use formula (11.1). This calculation involves the sum of N^T multiplications, each being a multiplication of $2T$ terms. The total number of operations is thus on the order of $2T \cdot N^T$.

Unless T is quite small, this calculation is computationally infeasible. For example, if $N = 4$, $T = 100$, the number of calculations is on the order of 10^{60}. It would take the life of the universe to make such a calculation.

Fortunately, there is a much more efficient and computationally feasible procedure, called the *forward algorithm.*

The forward algorithm focuses on the calculation of the quantity

$$\alpha(t,i) = \text{Prob}(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \ldots, \mathcal{O}_t, q_t = S_i), \tag{11.2}$$

which is the joint probability that the sequence of observations seen up to and including time t is $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \ldots, \mathcal{O}_t$, and that the state of the HMM at time t is S_i. The $\alpha(t,i)$ are called the *forwards* variables.

Once we know $\alpha(T,i)$ for all i, then $\text{Prob}(\mathcal{O})$ can be calculated as

$$\text{Prob}(\mathcal{O}) = \sum_{i=1}^{N} \alpha(T,i). \tag{11.3}$$

We calculate the $\alpha(t,i)$'s inductively on t. The first calculation is of the *initialization* step, and uses the obvious result

$$\alpha(1,i) = \pi_i b_i(\mathcal{O}_1). \tag{11.4}$$

Next, the equation

$$\alpha(t+1,i) = \sum_{j=1}^{N} \text{Prob}(\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \ldots, \mathcal{O}_{t+1}, q_{t+1} = S_i \text{ and } q_t = S_j)$$

leads to the *induction* step

$$\alpha(t+1,i) = \sum_{j=1}^{N} \alpha(t,j) p_{ji} b_i(\mathcal{O}_{t+1}). \tag{11.5}$$

This equation gives $\alpha(t+1,i)$ in terms of the $\alpha(t,j)$, so that $\alpha(t+1,i)$ can be calculated quickly once the $\alpha(t,j)$ are known. We use (11.4) to calculate $\alpha(1,i)$ for all i; then we use (11.5) to calculate $\alpha(2,i)$ for all i and again to calculate $\alpha(3,i)$ for all i, and so on, until we have obtained the $\alpha(T,i)$ for all i, needed in (11.3).

This procedure provides an algorithm for the solution to problem (i). The algorithm requires on the order of TN^2 computations, and thus is feasible in practice, even for very large models.

Before going on to problem (ii), we consider briefly the *backward* part of the forward–backward algorithm. This provides another approach to solving problem (i), but we introduce it because we will use the "backwards" variables when we discuss problem (iii).

In the above, we calculated successively $\alpha(1,\cdot), \alpha(2,\cdot), \ldots, \alpha(T,\cdot)$, that is, we calculated forward in time. In the backward algorithm we calculate another quantity backwards in time, as the name suggests. We do not use

these quantities to solve (11.3) instead we will need them for a later calcualtion. The goal of the backwards algorithm is to calculate the probability $\beta(t,i)$, defined by

$$\beta(t,i) = \text{Prob}(\mathcal{O}_{t+1}, \mathcal{O}_{t+2}, \ldots, \mathcal{O}_T \mid q_t = S_i), \tag{11.6}$$

for $1 \leq t \leq T-1$, and for convenience we define $\beta(T,j)$ to be 1 for all j. We then calculate (11.6) working backwards from $t = T-1$. The relevant equations for this procedure are

$$\beta(t-1,i) = \sum_{j=1}^{N} p_{ij}\, b_j(\mathcal{O}_t)\beta(t,j). \tag{11.7}$$

Using these equations we can successively calculate $\beta(T-1,i)$ for all i, $\beta(T-2,i)$ for all $i, \ldots$, and $\beta(1,i)$ for all i.

11.2.2 The Viterbi Algorithm

Given some observed sequence $\mathcal{O} = \mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \ldots, \mathcal{O}_T$ of outputs, we want to compute efficiently a state sequence $Q = q_1, q_2, q_3, \ldots, q_T$ that has the highest conditional probability given $\mathcal{O}$. In other words, we want to find a Q that makes $\text{Prob}(Q \mid \mathcal{O})$ maximal, that is, we want to calculate

$$\underset{Q}{\text{argmax}}\ \text{Prob}(Q \mid \mathcal{O}). \tag{11.8}$$

There may be many Q's that maximize $\text{Prob}(Q \mid \mathcal{O})$. We give an algorithm that finds one of them. It can easily be generalized to find them all. However, for our applications this generalization will not be necessary.

The Viterbi algorithm carries out the efficient computation of (11.8). The algorithm is divided into two parts. It first finds $\max_Q \text{Prob}(Q \mid \mathcal{O})$, and then "backtracks" to find a Q that realizes this maximum. This is another dynamic programming algorithm.

First define, for arbitrary t and i,

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} \text{Prob}(q_1, q_2, \ldots, q_{t-1}, q_t = S_i \text{ and } \mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \ldots, \mathcal{O}_t)$$

($\delta_1(i) = \text{Prob}(q_1 = S_i \text{ and } \mathcal{O}_1)$). In words, $\delta_t(i)$ is the maximum probability of all ways to end in state S_i at time t and have observed sequence $\mathcal{O}_1$, $\mathcal{O}_2, \ldots, \mathcal{O}_t$. Then

$$\max_Q \text{Prob}(Q \text{ and } \mathcal{O}) = \max_i \delta_T(i). \tag{11.9}$$

The probability in this expression is the joint probability of Q and $\mathcal{O}$, not a conditional probability. Our aim is to find a sequence Q for which the

maximum conditional probability (11.8) is achieved. Since

$$\max_Q \text{Prob}(Q \mid \mathcal{O}) = \max_Q \frac{\text{Prob}(Q \text{ and } \mathcal{O})}{\text{Prob}(\mathcal{O})},$$

and since the denominator on the right-hand side does not depend on Q,

$$\operatorname*{argmax}_Q \text{Prob}(Q \mid \mathcal{O}) = \operatorname*{argmax}_Q \frac{\text{Prob}(Q \text{ and } \mathcal{O})}{\text{Prob}(\mathcal{O})} = \operatorname*{argmax}_Q \text{Prob}(Q \text{ and } \mathcal{O}).$$

The first step is to calculate the $\delta_t(i)$'s inductively. Then we will "backtrack" and recover the sequence that gives the largest $\delta_T(i)$. The *initialization* step is

$$\delta_1(i) = \pi_i b_i(\mathcal{O}_1), \quad 1 \le i \le N. \tag{11.10}$$

The induction step is

$$\delta_t(j) = \max_{1 \le i \le N} \delta_{t-1}(i) p_{ij} b_j(\mathcal{O}_t), \quad 2 \le t \le T, \ 1 \le j \le N. \tag{11.11}$$

We recover the q_i's as follows. Define

$$\psi_T = \operatorname*{argmax}_{1 \le i \le N} \delta_T(i),$$

and put $q_T = S_{\psi_T}$. Then q_T is the final state in the state sequence required. The remaining q_t for $t \le T-1$ are found recursively by first defining

$$\psi_t = \operatorname*{argmax}_{1 \le i \le N} \delta_t(i) p_{i\psi_{t+1}},$$

and then putting $q_t = S_{\psi_t}$. If the argmax is not unique, we arbitrarily take one value of i giving the maximum.

11.2.3 The Estimation Algorithms

We now address problem (iii). Suppose we are given a set of observed data from an HMM for which the topology is known (by topology we mean the graph structure of the underlying Markov model). We wish to try to estimate the parameters in that HMM. The parameter space is usually far too large to allow exact calculation of a set of parameter estimates that maximizes the probability of the data. Instead, we employ algorithms that find "locally" best sets of parameters. This partial solution to the problem has proven to be useful in many applications.

The focus on local estimation means that the procedure is heuristic. Therefore, the efficacy of the procedure must evaluated empirically by using benchmarks and test sets for which there are known outcomes. This matter is discussed further below.

Some further comments are in order. It is not necessary to assume that the data come from an HMM. Instead, it is usually more accurate to assume that the data are generated by some random process that we try to "fit" with an HMM. Sometimes it might be possible to achieve a tight fit with an HMM and sometimes it might not.

The discussion above shows that we should use the term "estimation" of parameters cautiously in this section. Our aim is to "set" parameters at values providing a good fit to data rather than to estimate parameters in the sense of Chapter 8.

We now describe the Baum–Welch method of parameter estimation. This is a difficult algorithm, so we do not provide proofs of the claims made but instead indicate the intuition behind the method.

We assume that the alphabet A and number of states N is fixed at the outset, and that the parameters π_i, p_{jk}, and $b_i(a)$ are unknown and are to be "estimated." The data we use to estimate the parameters constitute a set of observed sequences $\{\mathcal{O}^{(d)}\}$. Each observed sequence $\mathcal{O}^{(d)} = \mathcal{O}_1^{(d)}, \mathcal{O}_2^{(d)}, \ldots$ has a corresponding hidden state sequence $Q^{(d)} = q_1^{(d)}, q_2^{(d)}, \ldots$.

The procedure starts by setting the parameters π_i, p_{jk}, and $b_i(a)$ at some initial values. These can be chosen from some uniform distribution or can be chosen to incorporate prior knowledge about them. We then calculate, using these initial parameter values,

$$\overline{\pi}_i = \text{the expected proportion of times in state } S_i \text{ at the first time point, given } \{\mathcal{O}^{(d)}\}, \tag{11.12}$$

$$\overline{p}_{jk} = \frac{E(N_{jk} \mid \{\mathcal{O}^{(d)}\})}{E(N_j \mid \{\mathcal{O}^{(d)}\})}, \tag{11.13}$$

$$\overline{b}_i(a) = \frac{E(N_i(a) \mid \{\mathcal{O}^{(d)}\})}{E(N_i \mid \{\mathcal{O}^{(d)}\})}, \tag{11.14}$$

where N_{jk} is the (random) number of times $q_t^{(d)} = S_j$ and $q_{t+1}^{(d)} = S_k$ for some d and t; N_i is the (random) number of times $q_t^{(d)} = S_i$ for some d and t; and $N_i(a)$ equals the (random) number of times $q_t^{(d)} = S_i$ and it emits symbol a, for some d and t. The expected values in (11.13) and (11.14) are conditional expected values, as defined in (2.56).

We show how to calculate these efficiently below. These are the "reestimation" parameter values that then replace π_i, p_{jk}, and $b_i(a)$. These values follow the form of estimation used, for example, in equation (3.10). The algorithm proceeds by iterating this step.

It can be shown that if $\lambda = (\pi_i, p_{jk}, b_i(a))$ is replaced by $\overline{\lambda} = (\overline{\pi}_i, \overline{p}_{jk}, \overline{b}_i(a))$, then $\text{Prob}(\{\mathcal{O}^{(d)}\} \mid \overline{\lambda}) \geq \text{Prob}(\{\mathcal{O}^{(d)}\} \mid \lambda)$, with equality holding if and only if $\overline{\lambda} = \lambda$. Thus successive iterations continually increase the probability of the data, given the model. Iterations continue until either a local maximum of the probability is reached or until the change in the probability becomes negligible.

In order to discuss the calculations needed for (11.13)–(11.12), define $\xi_t^{(d)}(i,j)$ by

$$\xi_t^{(d)}(i,j) = \text{Prob}(q_t^{(d)} = S_i, q_{t+1}^{(d)} = S_j \,|\, \mathcal{O}^{(d)}), \tag{11.15}$$

where $i,j = 1,\ldots,N$, and $t \geq 1$. The conditional probability formula (1.92) shows that this is equal to

$$\frac{\text{Prob}(q_t^{(d)} = S_i, q_{t+1}^{(d)} = S_j, \mathcal{O}^{(d)})}{\text{Prob}(\mathcal{O}^{(d)})}.$$

The denominator is $\text{Prob}(\mathcal{O}^{(d)})$ and is thus calculated efficiently using the methods of Section 11.2.1. The numerator is calculated efficiently by writing it in terms of the forwards and backwards variables discussed in Section 11.2.1,

$$\text{Prob}(q_t^{(d)} = S_i, q_{t+1}^{(d)} = S_j, \mathcal{O}^{(d)}) = \alpha_t(i) p_{ij} b_j(\mathcal{O}_{t+1}^{(d)}) \beta_{t+1}(j). \tag{11.16}$$

Let $I_t^{(d)}(i)$ be the indicator variables defined by

$$I_t^{(d)}(i) = \begin{cases} 1, & \text{if } q_t^{(d)} = S_i, \\ 0, & \text{otherwise.} \end{cases}$$

The number of times S_i is visited is then $\sum_d \sum_t I_t^{(d)}(i)$. The expected number of times S_i is visited, given $\{\mathcal{O}^{(d)}\}$, is then

$$\sum_d \sum_t E(I_t^{(d)}(i) \,|\, \mathcal{O}^{(d)}). \tag{11.17}$$

Now $E(I_t^{(d)}(i) \,|\, \mathcal{O}^{(d)})$ is $\text{Prob}(q_t^{(d)} = S_i \,|\, \mathcal{O}^{(d)})$, which is

$$\sum_{j=1}^{N} \xi_t^{(d)}(i,j). \tag{11.18}$$

Thus the expected number of times S_i is visited, given $\{\mathcal{O}^{(d)}\}$, is

$$\sum_d \sum_t \sum_{j=1}^{N} \xi_t^{(d)}(i,j).$$

Similarly, the expected number of transitions from S_i to S_j given $\{\mathcal{O}^{(d)}\}$ is

$$\sum_d \sum_t \xi_t^{(d)}(i,j).$$

These expressions give efficient formulae to calculate all the quantities in equations (11.12)–(11.14) except the numerator of (11.14). This is calculated as follows.

Define the indicator random variables $I_t^{(d)}(i,a)$ by

$$I_t^{(d)}(i,a) = \begin{cases} 1, & \text{if } q_t^{(d)} = S_i \text{ and } \mathcal{O}_t^{(d)} = a, \\ 0, & \text{otherwise.} \end{cases}$$

Then $E(I_t^{(d)}(i,a) \mid \mathcal{O}^{(d)})$ is the expected number of times the dth process is in state S_i at time t and emits symbol a, given $\mathcal{O}^{(d)}$. The numerator of (11.14) is equal to $\sum_d \sum_t E(I_t^{(d)}(i,a) \mid \mathcal{O}^{(d)})$, which is

$$\sum_d \sum_t \sum_{\mathcal{O}_t^{(d)}=a} \sum_{j=1}^{N} \xi_t^{(d)}(i,j).$$

11.3 Applications

We sketch here the applications of HMMs in several different areas of computational biology. Only a brief outline of each application is given: further details may be found in the references provided.

11.3.1 Modeling Protein Families

In this section we develop an HMM to model protein families, and we will use the model for two purposes: to construct multiple sequence alignments and to determine the family of a query sequence. These applications were first presented in Krogh et al. (1994). In order to present the main ideas we simplify many of the details.

Figure 11.1 gives an example of the basic type of HMM we will use. This example has "length" five; any length is possible. The underlying Markov model is presented in graphical form (as in Section 4.8). The states are the squares, diamonds, and circles labeled m_0, $m_1, \ldots, m_5$, i_0, $i_1, \ldots, i_4$, and d_1, $d_2, \ldots, d_4$, respectively. The squares are called the *match* states, the diamonds the *insert* states, and the circles the *delete* states. The edges not shown have transition probability zero. State m_0 is the *start* state, so that the process always starts in state m_0. A transition never moves to the left, so that as time progresses the current state gradually moves to the right, eventually ending in match state m_5, the *end* state. When this state is reached the process ends. A match or delete state is never visited more than once.

The alphabet A consists of the twenty amino acids together with one "dummy" symbol representing "delete" (denoted δ). Delete states output δ with probability one. Each insert and match state has its own distribution over the 20 amino acids, and cannot emit a δ. That is, only a delete state can emit a δ, and each delete state emits *only* δ.

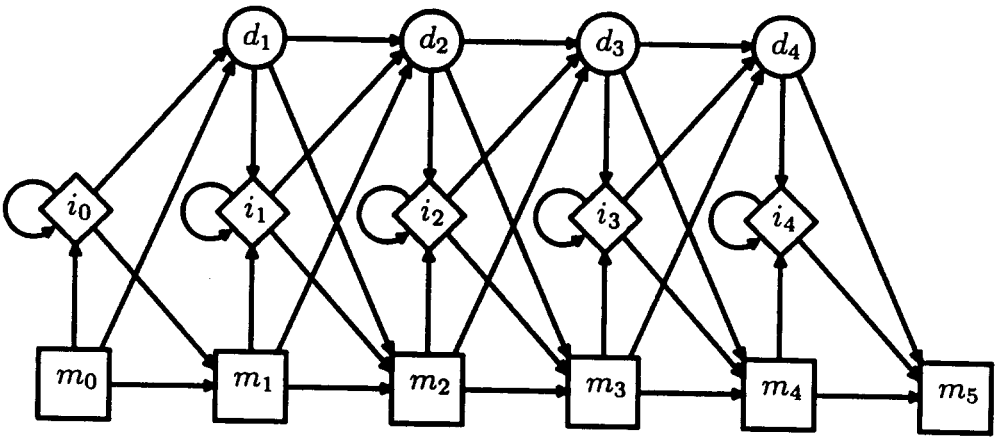


FIGURE 11.1. Hidden Markov model for a protein family.

If the emission probabilities for the match and insert states are uniform over the 20 amino acids, the model will produce random sequences that do not have much in common except possibly their lengths. At the other extreme, if each state emits one specific amino acid with probability one, and if further the transitions from m_i to m_{i+1} have probability one, then the model will always produce the same sequence. Somewhere in between these two extremes the parameters of the model can be set so that it produces sequences that are similar, thus producing what can be thought of as a "family" of sequences. Each choice of parameters produces a different family. This family can be rather "tight," meaning all sequences in it are very similar, or can be "loose," so that there is little similarity between the sequences produced. It is also possible that the similarity is high in some positions of the sequences produced and low in others. This will happen if some match states have distributions concentrated on a few amino acids while the others have distributions in which all amino acids are approximately equally likely. By contrast, the dynamic programming sequence alignment algorithms and BLAST allow one gap open penalty and use one substitution matrix uniformly across the entire length of the sequences compared. Allowing gap penalties and substitution probabilities to vary along the sequences reflects biological reality better. Alignments of related proteins generally have regions of higher conservation and regions of lower conservation. The regions of higher conservation are called functional domains, because their resistance to change indicates that they serve some critical function. Dynamic programming alignment and BLAST are essential for certain applications, such as pairwise alignments, or aligning a small number of sequences. But for modeling large families of sequences,

or constructing alignments of many sequences, HMMs allow for efficiency, and at the same time exploit the larger data sets to increase flexibility.

In the HMM model of a protein family the transition (arrow) from a match state to an insert state corresponds to the gap open penalty, and the arrow from an insert state to itself corresponds to the gap extension penalty. Loosely speaking, the distribution over the amino acids for any state takes the place of a substitution matrix. The probabilities in the model can differ from position to position in the sequence, since each arrow has its own probability and each match and insert state has its own distribution. Thus the HMM model is sufficiently flexible to model the varying features of a protein along its length. While the model can be made even more flexible by adding further parameters, more data are needed to estimate these parameters effectively. The model described has proven in practice to provide a good compromise between flexibility and tractability. Such HMM models of are called *profile* HMMs.

All applications start with *training*, or estimating, the parameters of the model using a set of training sequences chosen from a protein family, such as the set of all globins in GenBank. This estimation procedure uses the Baum–Welch algorithm. The model is chosen to have length equal to the average length of a sequence in the training set, and all parameters are initialized by using uniform distributions (i.e., amino acids are given probability $\frac{1}{20}$, and transitions of the same type are given $\frac{1}{2}$ equal probabilities).[1]

11.3.2 Multiple Sequence Alignments

In this section we describe how to use the theory described above to compute multiple sequence alignments for a family of sequences. The sequences to be aligned are used as the training data, to train the parameters of the model. For each sequence the Viterbi algorithm is then used to determine a path most likely to have produced that sequence. These paths can then be used to construct an alignment. Amino acids are aligned if both are produced by the same match state in their paths. Indels are then inserted appropriately for insertions and deletions.

We illustrate this with an example. Consider the sequences CAEFDDH and CDAEFPDDH. Suppose the model has length 10 and their most likely paths through the model are

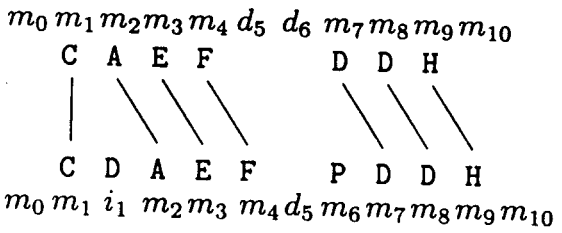
$$m_0 m_1 m_2 m_3 m_4 d_5 d_6 m_7 m_8 m_9 m_{10}$$

[1] What Krogh et al. (1994) do is somewhat more complicated. They allow the length of the model to change along with the parameters after each iteration of the reestimation algorithm. They also must adjust the Baum–Welch algorithm from how we have described it, in order to handle the delete states. The reader interested in implementing these applications should refer to the literature for further details.

and

$$m_0 m_1 i_1 m_2 m_3 m_4 d_5 m_6 m_7 m_8 m_9 m_{10},$$

respectively. Then the alignment induced is found by aligning positions that were generated by the same match state:



This leads to the alignment

```
C-AEF-DDH
CDAEFPDDH
```

More generally, suppose we have the five sequences

```
CAEFTPAVH,
CKETTPADH,
CAETPDDH,
CAEFDDH,
CDAEFPDDH,
```

and the corresponding paths returned by the Viterbi algorithm are

$$m_0 m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_{10},$$
$$m_0 m_1 m_2 m_3 m_4 m_5 m_6 m_7 m_8 m_9 m_{10},$$
$$m_0 m_1 m_2 m_3 d_4 m_5 m_6 m_7 m_8 m_9 m_{10},$$
$$m_0 m_1 m_2 m_3 m_4 d_5 d_6 m_7 m_8 m_9 m_{10},$$
$$m_0 m_1 i_1 m_2 m_3 m_4 d_5 m_6 m_7 m_8 m_9 m_{10}.$$

Then the induced alignment is

```
C-AEF T PAVH
C-KET T PADH
C-AE- T PDDH .
C-AEF--DDH
CDAEF-PDDH
```

This technique can give ambiguous results in some cases. For example, if the model has length two and the sequences ABAC and ABBAC had

$$m_0 m_1 i_1 i_1 m_2 m_3 \text{ and } m_0 m_1 i_1 i_1 i_1 m_2 m_3$$

as paths, then the leading A's and trailing C's will be aligned, but it is not clear how to align the BA from the first sequence to the BBA from the second.

In such cases Krogh et al. (1994) represent the ambiguous symbols with lowercase letters and do not attempt to give alignments of these regions.

This technique can be used to align many sequences with relatively little computing power. By contrast, dynamic programming algorithms cannot in practice align 50 or 100 long sequences. This is the value of a heuristic approach. Another advantage of the method is that it allows the sequences themselves to guide the alignment, rather than having a precomputed substitution matrix and gap penalties. Thus less bias should be introduced.

Krogh et al. (1994) tested this method on a family of 625 globin sequences. They used a published alignment of seven of these sequences constructed using knowledge of the three-dimensional structure of the sequences (Bashford et al. (1987)). (An alignment using a three-dimensional structure is considered reliable and so serves as a benchmark for testing multiple alignment algorithms.) They chose 400 of the 625 globins to train the model and then used the Viterbi algorithm to align all 625. These alignments were then compared to the induced alignment on the seven sequences from the Bashford alignment. The alignments agreed extremely well.

11.3.3 Pfam

Pfam is a web-based resource maintained by the Sanger Center (web URL (`http://www.sanger.ac.uk/Pfam/`). Pfam uses the basic theory described above to determine protein domains in a query sequence. A protein usually has one or more functional domains, namely portions of the protein that have essential function and thus have low tolerance for amino acid substitutions. Proteins in different families often share high homology in one or more domains. Entire protein families can be characterized by HMMs, as in the previous section, or one can characterize just functional domains. Pfam focuses on the latter.

Suppose that a new protein is obtained for which no information is available except the raw sequence. We wish to "annotate" this sequence. Annotation is the process of assigning to a sequence biologically relevant information, such as where the functional domains are, what their homology is to known domains, and what their function is. The typical starting point is a BLAST search. This will return all sequences in the chosen databases that have significant similarity to the query sequence. BLAST can return many such sequences. Though this is an important step in the annotation process, it is also desirable to have a database not of protein sequences themselves, but of protein domains. Pfam is not the first such database; however, previous domain databases do not use methods as flexible as HMMs and consequently tend not to model entire domains, but rather only the most highly conserved "motifs" that can be put in *ungapped* multiple sequence alignments. The use of HMMs allows for more effective characterization of full domains.

The domains in Pfam are determined based on expert knowledge, sequence similarity, and other protein family databases. Currently, Pfam contains 2008 protein domains. For each domain a set of examples of this domain is selected. The sequences representing each domain are put into an alignment, and the alignments themselves are used to set the parameters; that is, Baum–Welch is not used. Recall that an alignment implies for each sequence in the alignment a path through the HMM, as described in the previous section. The proportion of times these paths take a given transition is used to estimate the transition probabilities, and likewise for the emission probabilities. These alignments are called "seed alignments" and are stored in the database. Given the HMMs for all of the domains, a query sequence is then run past each one using the forward algorithm. When a portion of the query sequence has probability of having been produced by an HMM above a certain cutoff, the domain corresponding to that HMM is reported. Furthermore, the sequence can be aligned to the seed alignment using the Viterbi algorithm as described above. For more details, see the Pfam web site.

11.3.4 Gene Finding

Genomic sequences with lengths on the order of many millions of bases are now being produced, and the sequences of entire chromosomes are becoming available. Such sequences consist of a collection of genes separated from each other by long stretches of nonfunctional sequence. It is of central importance to find where the genes are in the sequence. Therefore, computational methods that quickly identify a large proportion of the genes are very useful. The problem involves bringing together a large amount of diverse information, and there have been many approaches to doing this. Currently, a popular and successful gene finder for *human* DNA sequences is GENSCAN (Burge et al. (1997)), which is based on a generalization of hidden Markov models. We sketch below an algorithm similar in spirit to that in GENSCAN in order to illustrate the basic concept of an HMM human gene finder. To increase the accuracy of the procedure it is necessary to introduce many details that we do not describe here. The interested reader is encouraged to read Burge et al. (1997) and Burge (1997).

Semihidden Markov Models

In an HMM, suppose that p is the probability of the transition from any state to itself. The probability that the process stays in this state for n steps is $p^{n-1}(1-p)$, so that the length of time the process stays in that state follows a geometric distribution. For the gene model we construct, it is necessary to allow other distributions for this length. In a *semihidden HMM* (semiHMM) all transition probabilities from a state to itself are zero,