Chapter 7

Hidden Markov Models: The Theory

7.1 Introduction

In the 1990s, only roughly a third of the newly predicted protein sequences show convincing similarity to other known sequences [80, 224, 155], using pairwise comparisons [11, 418]. This situation is even more unfortunate in the case of new, incomplete sequences or fragments. Large databases of fragments are becoming available as a result of various genome, cDNA, and other sequencing projects, especially those producing ESTs (expressed sequences tags) [200]. At the beginning of 1997, approximately half of GenBank consisted of fragment data. Such data cover a substantial fraction, if not all, of the expressed human genome. It is of course of great interest to recognize and classify such fragments, and recover any additional useful information.

A promising approach to improve the sensitivity and speed of current database-searching techniques has been to use consensus models built from multiple alignments of protein families [23, 52, 250, 334, 41, 38]. Unlike conventional pairwise comparisons, a consensus model can in principle exploit additional information, such as the position and identity of residues that are more or less conserved throughout the family, as well as variable insertion and deletion probabilities. All descriptions of sequence consensus, such as profiles [226], flexible patterns [52], and blocks [250], can be seen as special cases of the hidden Markov model (HMM) approach.

HMMs form another useful class of probabilistic graphical models used, over the past few decades, to model a variety of time series, especially in speech recognition [359, 439] but also in a number of other areas, such as

ion channel recordings [48] and optical character recognition [357]. HMMs have also earlier been applied to problems in computational biology, including the modeling of coding/noncoding regions in DNA [130], of protein binding sites in DNA [352], and of protein superfamilies [553] (see also [351]). Only since the mid-1990s [334, 41], though, have HMMs been applied systematically to model, align, and analyze entire protein families and DNA regions, in combination with machine-learning techniques.

HMMs are closely related to, or special cases of, neural networks, stochastic grammars, and Bayesian networks. In this chapter we introduce HMMs directly and show how they can be viewed as a generalization of the multiple dice model of chapter 3. We develop the theory of HMMs—in particular the main propagation and machine-learning algorithms—along the lines of chapter 4. The algorithms are used in the following sections where we outline how to apply HMMs to biological sequences. Specific applications are treated in chapter 8, while relationships to other model classes are left for later chapters.

7.1.1 HMM Definition

A first-order discrete HMM is a stochastic generative model for time series defined by a finite set S of states, a discrete alphabet A of symbols, a probability transition matrix $T = (t_{ji})$, and a probability emission matrix $E = (e_{i\mathsf{X}})$. The system randomly evolves from state to state while emitting symbols from the alphabet. When the system is in a given state i, it has a probability t_{ji} of moving to state j and a probability $e_{i\mathsf{X}}$ of emitting symbol X. Thus an HMM can be visualized by imagining that two different dice are associated with each state: an emission die and a transition die. The essential first-order Markov assumption of course is that the emissions and transitions depend on the current state only, and not on the past. Only the symbols emitted by the system are observable, not the underlying random walk between states; hence the qualification "hidden." The hidden random walks can be viewed as hidden or latent variables underlying the observations.

As in the case of neural networks, the directed graph associated with nonzero t_{ji} connections is also called the architecture of the HMM. Although this is not necessary, we will always consider that there are two special states, the *start* state and the *end* state. At time 0, the system is always in the start state. Alternatively, one can use a distribution over all states at time 0. The transition and emission probabilities are the parameters of the model. An equivalent theory can be developed by associating emissions with transitions, rather than with states. HMMs with continuous alphabets are also possible, but will not be considered here because of our focus on the discrete aspects of biological sequences.

A very simple example of an HMM is given in figure 7.1. In this example, we can imagine that there are two "DNA dice." The first die has an emission probability vector of $(e_{1\mathsf{A}} = 0.25, e_{1\mathsf{C}} = 0.25, e_{1\mathsf{G}} = 0.25, e_{1\mathsf{T}} = 0.25)$. The second die has an emission probability vector of $(e_{2\mathsf{A}} = 0.1, e_{2\mathsf{C}} = 0.1, e_{2\mathsf{G}} = 0.1, e_{2\mathsf{T}} = 0.7)$. The transition probabilities are given in the figure. Suppose that we now observe a sequence such as ATCCTTTTTTTCA. There are at least three questions that one can ask immediately: How likely is this sequence for this particular HMM? (This is the likelihood question.) What is the most probable sequence of transitions and emissions through the HMM underlying the production of this particular sequence? (This is the decoding question.) And finally, assuming that the transition and emission parameters are not known with certainty, how should their values be revised in light of the observed sequence? (This is the learning question.) We recommend that the reader try to answer these questions on the simple example above. Precise algorithmic answers for all three problems in the general case will be given in the following sections. We now consider different types of HMM architectures for biological applications.

7.1.2 HMMs for Biological Sequences

In biological sequence applications, the main HMM alphabets are of course the 20-letter amino acid alphabet for proteins and the four-letter nucleotide alphabet for DNA/RNA problems. Depending on the task, however, a number of other alphabets can be used, such as a 64-letter alphabet of triplets, a three-letter alphabet (α, β, coil) for secondary structure, and Cartesian products of alphabets (see table 6.1). If necessary, a space symbol can be added to any of these alphabets. In this chapter and chapter 8, we use the protein and DNA alphabets only.

In the simple HMM example above, there are only two hidden states, with a fully interconnected architecture between them. In real applications we need to consider more complex HMM architectures, with many more states and typically sparser connectivity. The design or selection of an architecture is highly problem-dependent. In biological sequences, as in speech recognition, the linear aspects of the sequences are often well captured by the so-called left–right architectures. An architecture is left–right if it prevents returning to any state once a transition from that state to any other state has occurred. We first review the most basic and widely used left–right architecture for biological sequences, the standard linear architecture (figure 7.2).

To begin with, consider the problem of modeling a family of related sequences, such as a family of proteins. As in the application of HMMs to speech recognition, a family of proteins can be seen as a set of different utterances of the same word, generated by a common underlying HMM. The standard ar-

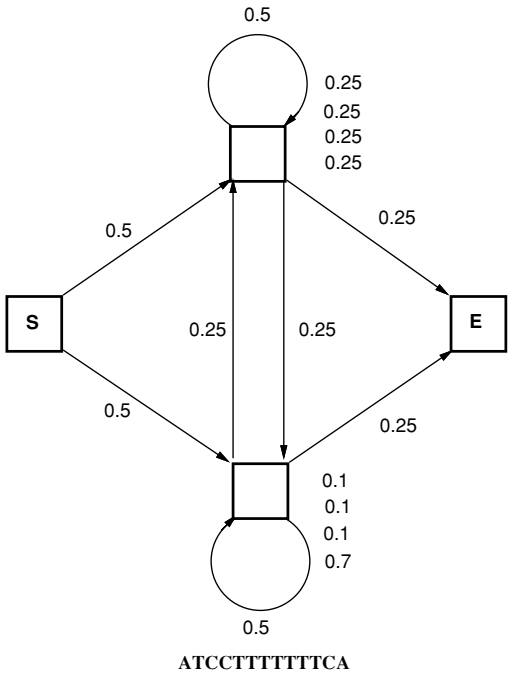

Figure 7.1: A Simple Example of an HMM, with Two States in Addition to the *Start* and *End* States.

chitecture can be seen as a very simple variation of the multiple-die model of chapter 3. The multiple-die model is in fact a trivial HMM with a linear sequence of states, one for each die. Transition probabilities from one state to the next are all set to 1. The emission probability of each die is associated with the composition of the family in the corresponding column. The main problem with such a model, of course, is that there are insertions and deletions: the sequences in the family in general do not have the same length N. Even if a gap symbol is added to the die alphabet, a preexisting multiple alignment is required to determine the emission probabilities of each die. The standard architecture is a simple but fundamental variation of the simple die model, where special states for insertions and deletions are added at all possible positions.

In the standard architecture, in addition to *start* and *end*, there are three other classes of states: the *main* states, the *delete* states, and the *insert* states, with $S = \{\text{start}, m_1, \ldots, m_N, i_1, \ldots, i_{N+1}, d_1, \ldots, d_N, \text{end}\}$. Delete states are

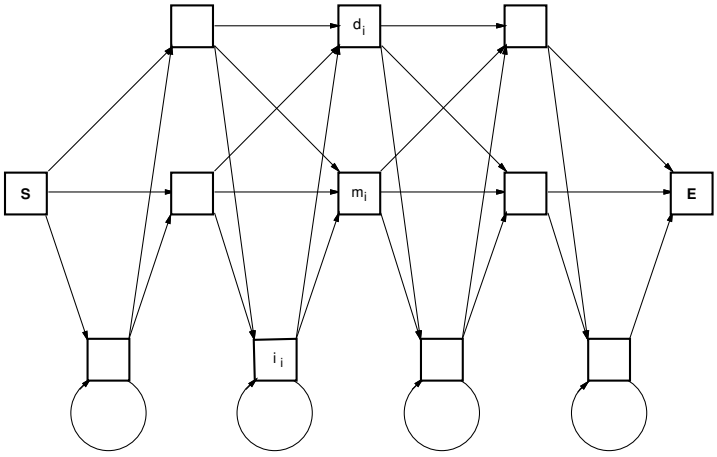

Figure 7.2: The Standard HMM Architecture. S is the start state, E is the end state, and d_i, m_i, and i_i denote delete, main, and insert states, respectively.

also called gap or skip states. N is the length of the model, typically equal to the average length of the sequences in the family. The main and insert states always emit an amino acid symbol, whereas the delete states are mute. This is of course equivalent to adding a space symbol to the alphabet and forcing the emission of the delete states to be concentrated on this symbol. The linear sequence of state transitions, start $\rightarrow m_1 \rightarrow m_2 \ldots \rightarrow m_N \rightarrow$ end, is the backbone of the model. These are the states corresponding to a multiple-die model. For each main state, corresponding insert and delete states are needed to model insertions and deletions. More precisely, there is a 1:1 correspondence between main states and delete states, and a 1:1 correspondence between backbone transitions and insert states. The self-loop on the insert states allows for multiple insertions at a given site. With an alphabet of size $|A|$, the standard architecture has approximately $2N|A|$ emission parameters and $9N$ transition parameters, without taking into account small boundary effects (the exact numbers are $(2N + 1)|A|$ emissions and $9N + 3$ transitions). Thus, for large N, the number of parameters is of the order of $49N$ for protein models and $17N$ for DNA models. Of course, neglecting boundary effects, there are also $2N$ normalization emission constraints and $3N$ normalization transition constraints.

7.2 Prior Information and Initialization

There are a number of ways in which prior information can be incorporated in the design of an HMM and its parameters. In the following sections we will give examples of different architectures. Once the architecture is selected, one can further restrain the freedom of the parameters in some of its portions, if the corresponding information is available in advance. Examples of such situations could include highly conserved motifs and hydrophobic regions. Linking the parameters of different portions is also possible, as in the weight-sharing procedure of NNs. Because of the multinomial models associated with HMM emissions and transitions, the natural probabilistic priors on HMM parameters are Dirichlet distributions (see chapter 2).

7.2.1 Dirichlet Priors on Transitions

In the standard architecture, for the vector of transitions t_{ji} out of a state i, a Dirichlet distribution $\mathcal{D}_{\alpha_i Q_i}(t_{ji})$ works well. One can use the same Dirichlet distribution for all the states of the same type—for instance, for all the main states, except the last one because of boundary effects. Thus three basic priors—$\mathcal{D}_{\alpha_m Q_m}$, $\mathcal{D}_{\alpha_i Q_i}$, and $\mathcal{D}_{\alpha_d Q_d}$—can be used for the transitions out of main, insert, and delete states. The hyperparameters' αs can be further reduced, if desirable, by having $\alpha_m = \alpha_i = \alpha_d$. Notice that the Dirichlet vectors Qs are usually not uniform, and are different for each state type. This is because transitions toward main states are expected to be predominant.

7.2.2 Dirichlet Priors on Emissions

The situation for emissions $\mathcal{D}_{\alpha_i Q_i}(e_{i\mathrm{X}})$ is similar. A simple option is to use the same Dirichlet distribution for all the insert states and all the main states. The vector Q can be chosen as the uniform vector. Another possibility is to have Q equal to the average composition frequency of the training set. In [334] Dirichlet mixtures are also used.

7.2.3 Initialization

The transition parameters are typically initialized uniformly or at random. In the standard architecture, uniform initialization without a prior that favors transitions toward the main states is not, in general, a good idea. Since all transitions have the same costs, emissions from main states and insert states also have roughly the same cost. As a result, insert states may end up being

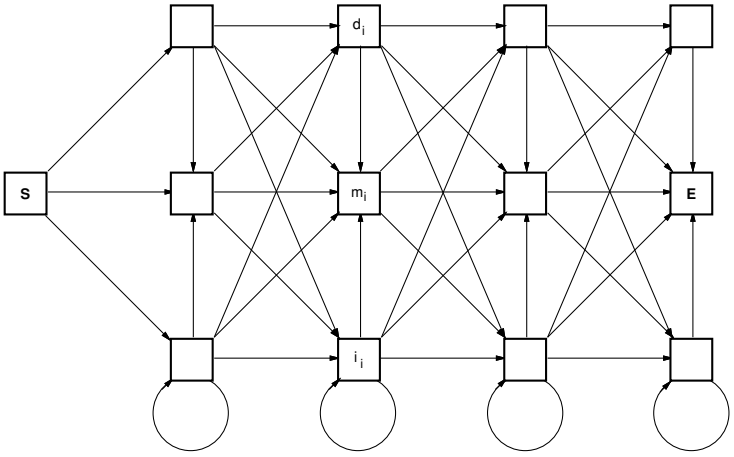

Figure 7.3: Variation on the Standard HMM Architecture. S is the start state, E is the end state, and d_i, m_i, and i_i denote delete, main, and insert states, respectively.

used very frequently, obviously not a very desirable solution. In [41], this problem was circumvented by introducing a slightly different architecture (figure 7.3), where main states have a lower fan-out (3) than insert or delete states (4), and therefore are less costly around the point where transitions out of each state are uniformly distributed. In a similar way, emissions can be initialized uniformly, at random, or sometimes even with the average composition. Any initialization that significantly deviates from uniform can introduce undesirable biases if Viterbi learning (see 7.4.3) is used.

Initialization from Multiple Alignments

Finally, it is important to realize that if a multiple alignment of a training set is available, it can be used to determine the parameters of the corresponding standard architecture, or at least to initialize them prior to learning. In the latter situation, the hope of course is that by starting closer to the optimal solutions, learning might be faster and/or yield a better solution. From a multiple alignment, we can assign a main state to any column of the alignment that contains less than 50% gaps. A column with more than 50% gaps is assigned to a corresponding insert state. Delete states are associated with the gaps in the columns with less than 50% gaps. Emissions of main and insert states can be initialized from the frequency counts of the corresponding columns, although these need to be regularized (with Dirichlet distributions

and/or their mixtures) to avoid emission biases associated with 0 counts. A similar approach can be taken to determine the transition parameters.

7.3 Likelihood and Basic Algorithms

In this section, we study the basic HMM algorithms needed to answer the first two questions raised above. In particular, we study how to compute the likelihood, and the most probable sequence of state transitions and emissions, associated with an observation sequence. These algorithms are recursive and can be viewed as forms of dynamic programming or as propagation algorithms in the directed graph associated with the HMM [439]. All these algorithms are essential building blocks for the learning algorithms of the following section. The presence of delete states here slightly complicates the equations.

First, consider the problem of computing the likelihood $\mathbf{P}(O|w)$ of a sequence $O = \mathsf{X}^1 \dots \mathsf{X}^t \dots \mathsf{X}^T$ according to an HMM $M = M(w)$ with parameter w. We define a *path* π in M to be a sequence of consecutive states of M starting with the *start* state and ending with the *end* state, together with the choice of an emission letter for each of the emitting states along the path. If the sequence of emission letters along the path coincides with O, then

$$\mathbf{P}(O, \pi|w) = \prod_{start}^{end} t_{ji} \prod_{t=1}^{T} e_{i\mathsf{X}^t}, \tag{7.1}$$

where the first product is taken over all transitions along the path π, and the second product over the corresponding emitting states i in π. If the sequence of emission letters along the path does not coincide with O, then obviously $\mathbf{P}(O, \pi|w) = 0$. The likelihood of a sequence can then be expressed by

$$\mathbf{P}(O|w) = \sum_{\pi} \mathbf{P}(O, \pi|w). \tag{7.2}$$

This expression, however, does not lead to an efficient computation of the likelihood or its derivatives, because the number of paths in an architecture is typically exponential. Luckily, there is a more efficient way of organizing the computation of the likelihood, known as the forward algorithm. All the other algorithms in this section are similar and can be seen as ways of organizing calculations using an iterative propagation mechanism through the architecture, in order to avoid looking at all possible hidden paths.

7.3.1 The Forward Algorithm

Let us define

$$\alpha_i(t) = \mathbf{P}(S^t = i, \mathsf{X}^1 \dots \mathsf{X}^t|w), \tag{7.3}$$

the probability of being in state i at time t, having observed the letters $X^1 \dots X^t$ in the model $M(w)$. We can initialize

$$\alpha_{start}(0) = 1. \tag{7.4}$$

Without a *start* state, we would use an initial probability over all states. What we want to compute is $\mathbf{P}(O|w) = \alpha_{end}(T)$. The $\alpha_i(t)$ can be computed recursively by simple propagation:

$$\alpha_i(t+1) = \sum_{j \in S} \alpha_j(t) t_{ij} e_{iX^{t+1}} = \sum_{j \in N^-(i)} \alpha_j(t) t_{ij} e_{iX^{t+1}}. \tag{7.5}$$

The neighborhood notation is used again to stress the advantage of general sparse connectivity. This equation is true for any emitting state. For *delete* states, it must be modified slightly

$$\alpha_i(t+1) = \sum_{j \in N^-(i)} \alpha_j(t+1) t_{ij}. \tag{7.6}$$

At first sight, (7.5) and (7.6) do not define a proper propagation mechanism because in (7.6) the time $t+1$ appears on both sides of the equation. It is easy to see, however, that iterations of (7.5) and (7.6) must converge to a stable set of values $\alpha_i(t+1)$. This is obvious when there are no directed loops in the architecture going through delete states only, as in the case of the standard architecture. In this case (7.6) must be iterated N times at most. But even when there are loops through delete states in the architecture, (7.6) is in general convergent, since the propagation of probabilities through a silent loop gives rise to a geometric series with ratio equal to the product of the transitions along the loop. This ratio is typically less than 1 (see appendix D for more details).

A directed path from j to i in an HMM is said to be *silent* if the only internal nodes it contains correspond to delete (silent) states. The probability of such a path is the product of the probabilities of the transitions it contains. We denote by t^D_{ij} the probability of moving from j to i silently. Thus t^D_{ij} is the sum of the probabilities of all the silent paths joining j to i. In the standard architecture, t^D_{ij} is trivial to compute since there is at most one silent path from j to i. With this notation, the forward propagation can also be expressed by first computing $\alpha_i(t+1)$ for all the emitting states, using (7.5). The forward variables for the delete states can then be computed by

$$\alpha_i(t+1) = \sum_{j \in E} \alpha_j(t+1) t^D_{ij}, \tag{7.7}$$

where E denotes the set of all emitting states. Note that the propagation in (7.5) and (7.6) can be seen as the propagation in a linear neural network with

T layers, one per time step, and M units in each layer, one for each HMM state. All the units are linear. The unit corresponding to emitting states i in layer $t+1$ has a linear transfer function with slope $e_{i\mathsf{X}^{t+1}}$. In this sense, the computation of likelihoods in an HMM is equivalent to forward propagation in a linear network with roughly N layers and $|S|$ units per layer. The presence of delete states adds connections within a layer. In the case of the standard architecture, in spite of these intralayer connections, the NN architecture remains feed-forward: hence the simple convergence of (7.6) during propagation. Because the algorithm consists essentially in updating T layers of M units each, the forward algorithm scales as $O(MT)$ operations. In the standard architecture, both M and T are of the same order as N ($M \approx 3N$), so the forward propagation scales as $O(N^2)$ operations.

Finally, one should also observe that using the forward variables as HMMs can be viewed as a dynamic mixture model. This is because the probability of emitting the letter X^t can be decomposed as $\sum_i \alpha_i(t) e_{i\mathsf{X}^t}$.

7.3.2 The Backward Algorithm

As in the case of neural networks, during learning we will need to propagate probabilities backward. The backward algorithm is the reverse of the forward algorithm. Let us define the backward variables by

$$\beta_i(t) = \mathbf{P}(\mathsf{X}^{t+1} \ldots \mathsf{X}^T | S^t = i, w), \tag{7.8}$$

the probability of being in state i at time t, with a partial observation of the sequence from X^{t+1} until the end. Obviously,

$$\beta_{end}(T) = 1. \tag{7.9}$$

The propagation equation to compute the βs recursively is given by

$$\beta_i(t) = \sum_{j \in N^+(i)} \beta_j(t+1) t_{ji} e_{j\mathsf{X}^{t+1}} \tag{7.10}$$

for the emitting states. For the delete states,

$$\beta_i(t) = \sum_{j \in N^+(i)} \beta_j(t) t_{ji}. \tag{7.11}$$

After updating the emitting states, this can be rewritten as

$$\beta_i(t) = \sum_{j \in E} \beta_j(t) t^D_{ji}. \tag{7.12}$$

The remarks made above about the forward algorithm also apply to the backward algorithm. In particular, in the standard architecture, the complexity of the backward algorithm also scales as $O(N^2)$.

Using the forward and backward variables, we can easily compute the probability of being in state i at time t, given the observation sequence O and the model w, by

$$\gamma_i(t) = \mathbf{P}(S^t = i|O, w) = \frac{\alpha_i(t)\beta_i(t)}{\mathbf{P}(O|w)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j\in S}\alpha_j(t)\beta_j(t)}, \tag{7.13}$$

or the probability $\gamma_{ji}(t)$ of using the $i \to j$ transition at time t by

$$\mathbf{P}(S^{t+1} = j, S^t = i|O, w) = \begin{cases} \alpha_i(t)t_{ji}e_{j\mathsf{X}^{t+1}}\beta_j(t+1)/\mathbf{P}(O|w) & \text{if } j \in E \\ \alpha_i(t)t_{ji}\beta_j(t)/\mathbf{P}(O|w) & \text{if } j \in D \end{cases} \tag{7.14}$$

where D represents the set of delete states. Obviously, we also have

$$\gamma_i(t) = \mathbf{P}(S^t = i|O, w) = \sum_{j\in S}\gamma_{ji}(t). \tag{7.15}$$

By maximizing $\gamma_i(t)$ we can find the most likely state at time t. In the decoding question, however, we are interested in the most likely path. The most likely path will also be useful for learning and for aligning sequences to the model. The most probable path can be computed using the so-called Viterbi algorithm, which is another application of dynamic programming and, in essence, is the same algorithm one uses for pairwise alignments. It is also very similar to the forward algorithm.

7.3.3 The Viterbi Algorithm

For the Viterbi algorithm, we need to define the variables

$$\delta_i(t) = \max_{\pi_i(t)}\mathbf{P}(\pi_i(t)|w), \tag{7.16}$$

where $\pi_i(t)$ represents a "prefix" path, with emissions $\mathsf{X}^1 \ldots \mathsf{X}^t$ ending in state i. Thus, $\delta_i(t)$ is the probability associated with the most probable path that accounts for the first t symbols of O and terminates in state i. These variables can be updated using a propagation mechanism similar to the forward algorithm, where sums are replaced by maximization operations:

$$\delta_i(t+1) = [\max_j \delta_j(t)t_{ij}]e_{i\mathsf{X}^{t+1}} \tag{7.17}$$

for the emitting states, and

$$\delta_i(t+1) = [\max_j \delta_j(t+1)t_{ij}] \tag{7.18}$$

for the delete states. The convergence is even more obvious than in the case of the forward algorithm; a cycle of delete states can never belong to an optimal path, because it decreases the overall probability without producing any letters. In order to recover the optimal path itself one must at each time keep track of the previous optimal state. The resulting Viterbi path will be used below both for learning and multiple alignments.

7.3.4 Computing Expectations

For a given set of parameters w and a given sequence O, $\mathbf{P}(\pi|O,w)$ defines a posterior probability distribution $Q(\pi)$ on the hidden variables, that is, the paths' πs. We have seen in chapters 3 and 4 that Q plays an important role. In particular, during learning, we will need to compute expectations with respect to Q, such as the expected number of times the state i is visited, the expected number of times the letter X is emitted from i, and the expected number of times the $i \to j$ transition is used. Because of the factorial nature of HMMs, Q is easy to compute and the associated expectations can be obtained from the forward–backward variables. Let

- $n(i,\pi,O)$ be the number of times i is visited, given π and O;
- $n(i,\mathsf{X},\pi,O)$ be the number of times the letter X is emitted from i, given π and O;
- $n(j,i,\pi,O)$ be the number of times the $i \to j$ transition is used, given π and O.

Then the respective expectations are given by

$$n_i = \sum_\pi n(i,\pi,O)\mathbf{P}(\pi|O,w) = \sum_{t=0}^{T} \gamma_i(t), \tag{7.19}$$

$$n_{i\mathsf{X}} = \sum_\pi n(i,\mathsf{X},\pi,O)\mathbf{P}(\pi|O,w) = \sum_{t=0,X^t=\mathsf{X}}^{T} \gamma_i(t) \tag{7.20}$$

and, similarly, for the transitions

$$n_{ji} = \sum_\pi n(j,i,\pi,O)\mathbf{P}(\pi|O,w) = \sum_{t=0}^{T} \gamma_{ji}(t). \tag{7.21}$$

We now have all the tools in place to tackle the HMM learning problem.

7.4 Learning Algorithms

Various algorithms are available for HMM training, including the Baum–Welch or EM (expectation maximization) algorithm, as well as different forms of gradient-descent and other GEM (generalized EM) [147, 439, 39] algorithms. Obviously, one could also use simulated annealing, although this remains impractical for large models. As usual, we concentrate on the first level of Bayesian inference: finding the optimal parameters by MAP estimation. We begin with ML estimation, concentrating on emission parameters; the calculations for transition parameters are similar. We also assume first that the data consist of a single sequence O. For each learning algorithm, we thus derive ML online learning equations first. We then briefly indicate how these equations should be modified for batch learning with multiple sequences and when priors are included (MAP). In the case of K training sequences, these can be considered as independent and the overall likelihood is equal to the product of the individual likelihoods. In the case of HMMs, higher levels of Bayesian inference have been used very little so far, even less than with neural networks. These will be discussed only very briefly.

Consider again the likelihood $\mathbf{P}(O|w) = \sum_\pi \mathbf{P}(O,\pi|w)$. In ML, we would like to optimize the Lagrangian

$$\mathcal{L} = -\log \mathbf{P}(O|w) - \sum_{i\in E} \lambda_i (1 - \sum_{\mathsf{X}} e_{i\mathsf{X}}) - \sum_{i\in S} \mu_i (1 - \sum_j t_{ji}), \tag{7.22}$$

where the λs and μs are positive Lagrange multipliers. From (7.1), we have

$$\frac{\partial \mathbf{P}(O,\pi|w)}{\partial e_{i\mathsf{X}}} = \frac{n(i,\mathsf{X},\pi,O)}{e_{i\mathsf{X}}} \mathbf{P}(O,\pi|w). \tag{7.23}$$

By setting the partial derivatives of the Lagrangian to 0, at the optimum we must have

$$\lambda_i e_{i\mathsf{X}} = \sum_\pi n(i,\mathsf{X},\pi,O) Q(\pi) = n_{i\mathsf{X}}, \tag{7.24}$$

and similarly for transition parameters. Recall that Q is the posterior probability $\mathbf{P}(\pi|O,w)$. By summing over all alphabet letters, we find

$$\lambda_i = \sum_\pi \sum_{\mathsf{X}} n(i,\mathsf{X},\pi,O) Q(\pi) = \sum_\pi n(i,\pi,O) Q(\pi) = n_i. \tag{7.25}$$

Thus, at the optimum, we must have

$$e_{i\mathsf{X}} = \frac{\sum_\pi n(i,\mathsf{X},\pi,O) Q(\pi)}{\sum_\pi n(i,\pi,O) Q(\pi)} = \frac{\sum_\pi \mathbf{P}(\pi|O,w) n(i,\mathsf{X},\pi,O)}{\sum_\pi \mathbf{P}(\pi|O,w) n(i,\pi,O)}. \tag{7.26}$$

The ML equations cannot be solved directly because in (7.26) the posterior distribution Q depends on the values of $e_{i\mathsf{X}}$. However, (7.26) suggests a simple iterative algorithm whereby Q is first estimated as $Q(\pi) = \mathbf{P}(\pi|O, w)$, and then the parameters are updated using (7.26). It turns out that this is exactly the EM algorithm for HMMs.

7.4.1 EM Algorithm (Baum–Welch)

Recall that in the EM algorithm we define the energy over hidden configurations, $f(\pi) = -\log \mathbf{P}(O, \pi|w)$. The EM algorithm can be defined as an iterative double minimization process of the function (free energy at temperature 1) $\mathcal{F}(w, Q) = \mathbf{E}_Q(f) - \mathcal{H}(Q)$ with respect first to Q and then to w. The first minimization step yields the posterior $Q(\pi) = \mathbf{P}(\pi|O, w) = \mathbf{P}(\pi, O|w)/\mathbf{P}(O|w)$, which we know how to calculate. For the second minimization step, we must minimize $\mathcal{F}$, with respect to w, under the probability normalization constraints. Since the entropy term is independent of w, we must finally minimize the Lagrangian

$$\mathcal{L} = \mathbf{E}_Q(f) - \sum_{i \in E} \lambda_i (1 - \sum_{\mathsf{X}} e_{i\mathsf{X}}) + \sum_{i \in S} \mu_i (1 - \sum_j t_{ji}), \tag{7.27}$$

with $Q(\pi) = \mathbf{P}(\pi|O, w)$ *fixed*. Using (7.23), we get

$$\lambda_i e_{i\mathsf{X}} = \sum_\pi n(i, \mathsf{X}, \pi, O) Q(\pi) = n_{i\mathsf{X}} \tag{7.28}$$

and, by summing over all alphabet letters,

$$\lambda_i = \sum_\pi \sum_{\mathsf{X}} n(i, \mathsf{X}, \pi, O) Q(\pi) = \sum_\pi n(i, \pi, O) Q(\pi) = n_i. \tag{7.29}$$

These equations are identical to (7.24) and (7.25). It can be checked that they correspond to a minimum, so that the EM reestimation equations are

$$e^+_{i\mathsf{X}} = \frac{\sum_\pi n(i, \mathsf{X}, \pi, O) Q(\pi)}{\sum_\pi n(i, \pi, O) Q(\pi)} = \frac{\sum_{t=0, X^t = \mathsf{X}}^T \gamma_i(t)}{\sum_{t=0}^T \gamma_i(t)} = \frac{n_{i\mathsf{X}}}{n_i}. \tag{7.30}$$

In the case of transition parameters, one similarly obtains

$$t^+_{ji} = \frac{\sum_\pi n(j, i, \pi, O) Q(\pi)}{\sum_\pi n(i, \pi, O) Q(\pi)} = \frac{\sum_{t=0}^T \gamma_{ji}(t)}{\sum_{t=0}^T \gamma_i(t)} = \frac{n_{ji}}{n_i}. \tag{7.31}$$

Thus the EM equations are implemented using the forward and backward procedures. In fact, the EM algorithm for HMMs is sometimes called the forward–backward algorithm. $e^+_{i\mathsf{X}}$ is the expected number of times in state i observing

symbol X, divided by the expected number of times in state i, and t^+_{ji} is the expected number of transitions from i to j, divided by the expected number of transitions from state i. These are exactly the same iteration equations obtained by setting the derivatives of the Lagrangian associated with the likelihood (7.22) to 0. This is a particular property of HMMs and factorial distributions, and *not* a general rule.

In the case of K sequences $O_1, \ldots, O_K$, a similar calculation shows that we have

$$e^+_{i\mathsf{X}} = \frac{\sum_{j=1}^K \sum_\pi n(i, \mathsf{X}, \pi, O_j) \mathbf{P}(\pi | O_j, w)}{\sum_{j=1}^K \sum_\pi n(i, \pi, O_j) \mathbf{P}(\pi | O_j, w)}. \tag{7.32}$$

It should also be clear how to modify the present equations in the case of MAP estimation by EM. Each training sequence requires one forward and one backward propagation. Thus the EM algorithm scales as $O(KN^2)$ operations.

The batch EM algorithm is widely used for HMMs. It must be noted, however, that the online use of the EM algorithm can be problematic. This is because the EM algorithm, unlike gradient descent, does not have a learning rate. The EM algorithm can take large steps in the descent direction generated by each training example in isolation, converging toward poor local minima of $\mathcal{E}$. This "carpet-jumping" effect can be avoided with gradient-descent learning, as long as the learning rate is small.

7.4.2 Gradient Descent

The gradient-descent equations on the negative log-likelihood can be derived by exploiting the relationship between HMMs and NNs, and using the backpropagation equations. Here we derive them directly. Instead of using the Lagrangian with the normalization constraints, as above, we use an equivalent useful reparameterization. We reparameterize the HMM using normalized exponentials, in the form

$$e_{i\mathsf{X}} = \frac{e^{w_{i\mathsf{X}}}}{\sum_Y e^{w_{i\mathsf{Y}}}} \quad \text{and} \quad t_{ji} = \frac{e^{w_{ji}}}{\sum_k e^{w_{ki}}}, \tag{7.33}$$

with $w_{i\mathsf{X}}$ and w_{ij} as the new variables. This reparameterization has two advantages: (1) modification of the ws automatically preserves normalization constraints on emission and transition distributions; (2) transition and emission probabilities can never reach the value 0. A simple calculation gives

$$\frac{\partial e_{i\mathsf{X}}}{\partial w_{i\mathsf{X}}} = e_{i\mathsf{X}}(1 - e_{i\mathsf{X}}) \quad \text{and} \quad \frac{\partial e_{i\mathsf{X}}}{\partial w_{i\mathsf{Y}}} = -e_{i\mathsf{X}} e_{i\mathsf{Y}}, \tag{7.34}$$

and similarly for the transition parameters. By the chain rule,

$$\frac{\partial \log \mathbf{P}(O|w)}{w_{i\mathsf{X}}} = \sum_{\mathsf{Y}} \frac{\partial \log \mathbf{P}(O|w)}{e_{i\mathsf{Y}}} \frac{\partial e_{i\mathsf{Y}}}{w_{i\mathsf{X}}}. \tag{7.35}$$

Therefore, applying (7.2), (7.23), and (7.33) to (7.35), the online gradient-descent equations on the negative log-likelihood are

$$\Delta w_{i\mathsf{X}} = \eta(n_{i\mathsf{X}} - n_i e_{i\mathsf{X}}) \quad \text{and} \quad \Delta w_{ji} = \eta(n_{ji} - n_i t_{ji}), \tag{7.36}$$

where η is the learning rate. $n_{i\mathsf{X}}$ and n_{ji} are again the expected counts derived by the forward–backward procedure for each single sequence if the algorithm is to be used online. Batch gradient-descent equations can easily be derived by summing over all training sequences. For MAP estimation, one needs to add the derivative of the log prior, with respect to the ws, to the online gradient-descent learning equations. For instance, a Gaussian prior on each parameter would add a weight decay term to (7.36).

Just like EM, the gradient-descent equations require one forward and one backward propagation. Therefore $O(KN^2)$ operations must be performed per training cycle. Some care must be taken in the implementation, however, to minimize the overhead introduced by the normalized exponential parameterization. Unlike EM, online gradient descent is a smooth algorithm. A number of other related smooth algorithms are discussed in [39]. A useful aspect of smooth algorithms is that unlearning is easy. If a sequence happens to be in the training set by error (that is, if it does not belong to the family being modeled), it is easy to remove its antagonistic impact from the model by reversing the effect of the gradient-descent equations.

7.4.3 Viterbi Learning

Both the EM and the gradient-descent update equations are based on the calculation of expectations over all possible hidden paths. The general Viterbi learning idea is to replace calculations involving all possible paths with calculations involving only a small number of likely paths, typically only the most likely one, associated with each sequence. Thus an emission count such as $n(i, \mathsf{X}, \pi, O)$ averaged over all paths is replaced by a single number $n(i, \mathsf{X}, \pi^*(O))$, the number of times X is emitted from i along the most probable path $\pi^*(O)$. In the standard architecture, $n(i, \mathsf{X}, \pi^*(O))$ is always 0 or 1, except for the insert states, where it can occasionally be higher as a result of repeated insertions of the same letter. For this reason, a plain online Viterbi EM makes little sense because parameters would mostly be updated to 0 or 1. For online Viterbi gradient descent, at each step along a Viterbi path, and for

any state i on the path, the parameters of the model are updated according to

$$\Delta w_{i\mathsf{X}} = \eta(E_{i\mathsf{X}} - e_{i\mathsf{X}}) \quad \text{and} \quad \Delta w_{ji} = \eta(T_{ji} - t_{ji}). \tag{7.37}$$

$E_{i\mathsf{X}} = 1$ (resp. $T_{ji} = 1$) if the emission of X from i (resp. $i \to j$ transition) is used, and 0 otherwise. The new parameters are therefore updated incrementally, using the discrepancy between the frequencies induced by the training data and the probability parameters of the model.

In the literature, Viterbi learning is sometimes presented as a quick approximation to the corresponding non-Viterbi version. The speed advantage is only relatively minor, and of the order of a factor of 2, since computing $\pi^*(O)$ does not require the backward propagation. As far as approximations are concerned, Viterbi learning is somewhat crude, since sequence likelihoods in general are not sharply peaked around a single optimal path. Thus it is not uncommon to observe significant differences between Viterbi and non-Viterbi learning both during the training phase and at the end. In our experience, we have often observed that Viterbi learning yields good results when modeling protein families, but not when modeling general DNA elements, such as exon or promoter regions, where non-Viterbi learning performs better. This probably is partially due to the fact that optimal paths play a particular role in the case of proteins.

In fact, a complementary view of Viterbi learning is that it constitutes an algorithm per se, trying to optimize a different objective function. We can define a new probability measure P^V, and hence a new model (hidden Viterbi model) on the space of sequences, by

$$P^V(O|w) = \frac{\mathbf{P}(\pi^*(O)|w)}{\sum_O \mathbf{P}(\pi^*(O)|w)}. \tag{7.38}$$

Viterbi learning then is an attempt at minimizing

$$\mathcal{E} = \sum_{k=1}^{K} -\log P^V(O_k|w). \tag{7.39}$$

It is important to note that as the parameters w evolve, the optimal paths π^* can change abruptly, and therefore $\mathcal{E}$ can be discontinuous. Obviously, regularizer terms can be added to (7.39) for a Viterbi version of MAP.

7.4.4 Other Aspects

As usual, many other issues can be raised regarding learning improvements, such as balancing the training set [157, 337], varying the learning rate, or using second-order information by estimating the Hessian of the likelihood. These

issues are discussed in the literature and cannot be covered here in detail for lack of space. We wish, however, briefly to discuss scaling, architecture selection or learning, and ambiguous symbols, since these are of particular practical importance.

Scaling

The probabilities $\mathbf{P}(\pi|O,w)$ are typically very small, since they are equal to the product of many transition and emission probabilities, each less than 1. For most models, this will easily exceed the precision of any machine, even in double precision. Therefore, in the implementation of the learning algorithms, and in particular of the forward and backward procedures, one is faced with precision issues. These can be addressed by using a scaling procedure, where forward and backward variables are scaled during the propagation in order to avoid underflow. The scaling procedure is somewhat technical and is described in appendix D. In Viterbi learning, the precision problem is easily addressed by working with the logarithm of the path probabilities.

Learning the architecture

A natural question to raise is whether the HMM architecture itself can be learned from the data. Algorithms for learning HMM architectures have indeed been developed for general HMMs—for instance, in [504]—and even in the context of biological sequences [193]. The basic idea in [504] is to start with a very complex model, essentially one state per data letter, and then iteratively merge states. The choice of states to merge and the stopping criterion are guided by an evaluation of the posterior probability. In [193], on the other hand, the starting point is a small, fully interconnected HMM. The algorithm in this case proceeds by iteratively deleting transitions with very low probability and duplicating the most connected states, until the likelihood or posterior reaches a sufficient level. In both cases, good results are reported on small test cases associated with HMMs having fewer than 50 states. While these methods may be useful in some instances, they are slow and unlikely to be practical, on current computers, for most of the large HMMs envisioned in chapter 8 without leveraging any available prior knowledge. The number of all possible architectures with $|S|$ states is of course very large. A much more tractable special case of architecture learning is whether the length N of the standard HMM architecture can be learned.

Adaptable model length

The approach described so far for the standard architecture is to fix N to the average length of the sequences being modeled. In practice, this simple approach seems to work quite well. Naturally, after training, if such a value of N does not seem to be optimal, a new value can be selected and the training procedure restarted.

In [334] a "surgery" algorithm is presented for the *dynamic* adjustment of the HMM length during learning. The idea is to add or remove states wherever needed along the architecture, while respecting the overall connectivity pattern. If an insert state is used by more than 50% of the family of sequences being modeled, meaning that the insert state is present in more than 50% of the corresponding Viterbi paths, then a new main state is created at the corresponding position, together with corresponding new delete and insert states. The new state emission and transition probabilities can be initialized uniformly. Likewise, if a delete state is used by more than 50% of the sequences, it can be removed together with the corresponding main and insert states. The rest of the architecture is left untouched, and the training proceeds. Although this approach has not been shown to converge always to a stable length, in practice it seems to do so.

Architectural variations

As already pointed out, a number of other architectures, often related to the standard architecture, have been used in molecular biology applications. These include the multiple HMM architecture (figure 8.5) for classification, and the loop (figure 8.16) and wheel (figure 8.17) architectures for periodic patterns. The standard architecture has also been used to model protein secondary structure [187] and build libraries of secondary structure consensus patterns for proteins with similar fold and function. Several other architectures have been developed for gene finding both in prokaryotes [336] and eukaryotes [107]. Examples of specific applications will be given in chapter 8.

Ambiguous symbols

Because sequencing techniques are not perfect, ambiguous symbols are occasionally present. For instance, `X` represents `A` or `C` or `G` or `T` in DNA sequences, and `B` represents asparagine or aspartic acid in protein sequences. Such symbols can easily be handled in a number of ways in conjunction with HMMs. In database searches, it is prudent practice to use the "benefit of the doubt"

approach, in which an ambiguous symbol is replaced by its most likely alternative in the computation of sequence likelihoods and Viterbi paths. Additional care must be used with sequences having an unusually high proportion of ambiguous symbols, since these are likely to generate false positive responses.

7.5 Applications of HMMs: General Aspects

Regardless of the design and training method, once an HMM has been successfully derived from a family of sequences, it can be used in a number of different tasks, including

1. Multiple alignments
2. Database mining and classification of sequences and fragments
3. Structural analysis and pattern discovery.

All these tasks are based on the computation, for any given sequence, of its probability according to the model as well as its most likely associated path, and on the analysis of the model structure itself. In most cases, HMMs have performed well on all tasks, yielding, for example, multiple alignments that are comparable with those derived by human experts. Specific examples and details on proteins and DNA applications of HMMs will be given in chapter 8. HMM libraries of models can also be combined in a hierarchical and modular fashion to yield increasingly refined probabilistic models of sequence space regions. HMMs could in principle be used in generative mode also to produce *de novo* sequences having a high likelihood with respect to a target family, although this property has not been exploited.

7.5.1 Multiple Alignments

Computing the Viterbi path of a sequence is also called, for obvious reasons, "aligning a sequence to the model." Multiple alignments can be derived, in an efficient way, by aligning the Viterbi paths to each other [334, 41]. While training a model may sometimes be lengthy, it can be done offline. Once the training phase is completed, the multiple alignment of K sequences requires the computation of K Viterbi paths, and therefore scales only as $O(KN^2)$. This is linear in K, and should be contrasted with the $O(N^K)$ scaling of multidimensional dynamic programming alignment, which is exponential in K. The multiple alignments derived by HMMs are in some sense richer than conventional alignments. Indeed, consider a conventional alignment of two sequences and assume that, at a given position, the second sequence has a gap with respect to the first sequence. This gap could be the result of a deletion in the

second sequence or an insertion in the first sequence. These are two distinct sets of Viterbi paths in an HMM that are not distinguished in a conventional alignment.

Another way of looking at this issue is to consider that a conventional multiple alignment could be derived by training an HMM architecture that is similar to the standard architecture, but where the length of the model is fixed to the length of the longest sequence being aligned and all insert states are removed, leaving only main and delete states. Thus, all the Viterbi paths consist only of main-state emissions or gaps with respect to main states. But in any case, it should be clear that the multiple alignments derived by an HMM with both insert and delete states are potentially richer and in fact should be plotted in three dimensions, rather than the two used by conventional multiple alignments (the third dimension being reserved for emissions occurring on HMM insert states). Because this is both graphically difficult and unconventional, HMM alignments are still plotted in two dimensions like conventional ones. Lowercase letters are then often reserved for letters produced by HMM insert states.

The insert and delete states of an HMM represent formal operations on sequences. One important question is whether and how they can be related to evolutionary events. This issue is also related, of course, to the construction of phylogenetic trees, and their relation to HMMs and multiple alignments. The standard architecture by itself does not provide a good probabilistic model of the evolutionary process because it lacks the required tree structure as well as a clear notion of substitution (in addition to insertion and deletion). Probabilistic models of evolution are addressed in chapter 10.

The reader should perhaps be reminded one more time that the treatment of HMM multiple alignments we have just presented is based on a single HMM, and therefore corresponds only to the first step of a full Bayesian treatment. Even for a simple question, such as whether two amino acids in two different sequences should be aligned to each other, a full Bayesian treatment would require integration of the answer across all HMMs with respect to the proper posterior probability measure. To the best of our knowledge, such integrals have not been computed in the case of HMMs for biological sequences (but see [583]). It is difficult to guess whether much could be gained through such a computationally intensive extension of current practice.

Finally, HMMs could also be used in conjunction with substitution matrices [27]. HMM emission distributions could be used to calculate substitution matrices, and substitution matrices could be used to influence HMMs during or after training. In the case of large training sets, one might expect that most substitution information is already present in the data itself, and no major gains would be derived from an external infusion of such knowledge.

7.5.2 Database Mining and Classification

Given a trained model, the likelihood of any given sequence (as well as the likelihood of the associated Viterbi path) can be computed. These scores can be used in discrimination tests and in database searches [334, 38] to separate sequences associated with the training family from the rest. This is applicable to both complete sequences and fragments [42]. One important aspect to be examined in chapter 8 is that such scores must be calibrated as a function of sequence length.

HMMs can also be used in classification problems, for instance, across protein families or across subfamilies of a single protein family. This can be done by training a model for each class, if class-specific training sets are available. We have used this approach to build two HMMs that can reliably discriminate between tyrosine and serine/threonine kinase subfamilies. Otherwise, unsupervised algorithms related to clustering can be used in combination with HMMs to generate classifications. An example here is the discrimination of globin subfamilies (see [334] and chapter 8). It is believed that the total number of protein superfamilies is relatively small, on the order of 1000 [127, 93]. A global protein classification system, with roughly one HMM per family, is becoming a feasible goal, from both an algorithmic and a computational standpoint. Global classification projects of this sort are currently under way, and should become useful auxiliary tools in a number of tasks, such as gene finding, protein classification, and structure/function prediction (see [497]).

7.5.3 Structural Analysis and Pattern Discovery

Information can also be derived, and new patterns discovered, by examining the structure of a trained HMM. The parameters of an HMM can be studied in the same way as the connections of an NN. High emission or transition probabilities are usually associated with conserved regions or consensus patterns that may have structural/functional significance. One convenient way of detecting such patterns is to plot the entropy of the emission distributions along the backbone of the model. Any other function of position, such as hydrophobicity or bendability, can also be averaged and plotted using the HMM probabilities. Patterns that are characteristic of a given family, such as features of secondary structure in proteins (hydrophobicity in alpha-helices) and regions of high bendability in DNA, are often easier to detect in such plots. This is because the variability of individual sequences is smoothed out by the expectations. There are other patterns, such as periodicities, that can be revealed by analyzing the structure of a model. The initial weak detection of such a pattern with the standard architecture can guide the design of more specialized architectures, such as wheel and loop architectures, to enhance the periodic

signal. The ability to detect weak patterns from raw unaligned data is a very useful feature of HMMs. Several examples will be given in chapter 8.

This page intentionally left blank

Chapter 8

Hidden Markov Models: Applications

8.1 Protein Applications

In the case of proteins, HMMs have been successfully applied to many families, such as globins, immunoglobulins, kinases, and G-protein-coupled receptors (see, e.g., [334, 41, 38]). HMMs have also been used to model secondary structure elements, such as alpha-helices, as well as secondary structure consensus patterns of protein superfamilies [187]. In fact, by the end of 1997, HMM data bases of protein families (Pfam) [497] and protein family secondary structures (FORESST) [187] became available. Multiple alignments derived from such HMMs have been reported and discussed in the literature. Large multiple alignments are typically too bulky to be reproduced here. But in most cases, HMM alignments are found to be very good, within the limits of variability found in multiple alignments produced by human experts resulting from diverse degrees of emphasis on structural or phylogenetic information. In the rest of this first half of the chapter, we concentrate on the application of HMMs to a specific protein family, the G-protein-coupled receptors (GCRs or GPCRs), along the lines of [38, 42]. Additional details can be found in these references.

8.1.1 G-Protein-Coupled Receptors

G-protein-coupled receptors are a rich family of transmembrane proteins capable of transducing a variety of extracellular signals carried by hormones, neurotransmitters, odorants, and light (see [436, 325, 508, 227, 552] for recent

reviews). Although the detailed biophysical mechanisms underlying the transduction have not been worked out for all members of the family, in most cases stimulation of the receptor leads to the activation of a guanine nucleotide-binding (G) protein [402]. All the receptors in the family are believed to have similar structure, characterized by seven hydrophobic membrane-spanning alpha-helices. The seven transmembrane regions are connected by three extracellular and three intracellular loops. The amino termini are extracellular and often glycosylated, whereas the carboxyl termini are cytoplasmic and usually phosphorylated. The exact three-dimensional packing of the helices, and more generally the complete tertiary structure, are only partially known [47, 420].

The family is usually divided into subclasses on the basis of transmitter types, such as muscarinic receptors, catecholamine receptors, odorant receptors, and so forth. From a methodological standpoint, the GPCR family is particularly challenging. Its members have very variable lengths and, on average, are fairly long: the length of known GPCRs varies from roughly 200 to 1200 amino acids. The family is highly variable and some of its members have less than 20% residues in common.

8.1.2 Structural Properties

In [38], 142 GPCR sequences extracted from the PROSITE database [23] were used to train an HMM architecture of length $N = 430$, the average length of the training sequences, using on-line Viterbi learning during 12 cycles of iterations through the entire training set.

As an example of a structural property, the entropy of the emission distribution of the main states of the corresponding model is given in figure 8.1. The amplitude profile of the entropy contains seven major oscillations directly related to the seven transmembrane domains. To a first approximation, the hydrophobic domains tend to be less variable, and therefore associated with regions of lower entropy. This structural feature was discovered by the HMM without any prior knowledge of alpha-helices or hydrophobicity.

8.1.3 Raw Score Statistics

To test the discrimination abilities of the model, 1600 random sequences were generated with the same average composition as the GCPRs in the training set, with lengths 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 1000 (100 sequences at each length), and 1500 and 2000 (200 sequences at each length). For any sequence, random or not, its raw score according to the model is calculated. Here, the raw score of a sequence O is the negative log-likelihood of the corresponding Viterbi path. The raw scores of all the random sequences

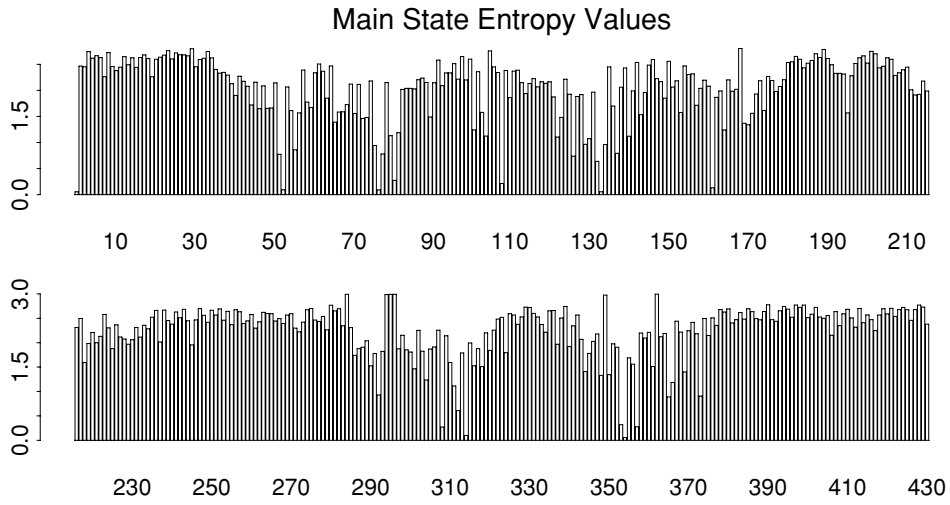

Figure 8.1: Entropy Profile of the Emission Probability Distributions Associated with the Main States of the HMM After 12 Cycles of Training.

are plotted in figure 8.2, together with the scores of the GPCRs in the training set and the scores of all the sequences in the SWISS-PROT database.

The model clearly discriminates random sequences with similar average composition from true GPCRs. Consistent with previous experiments [41, 334], the scores of the random sequences and of the SWISS-PROT sequences cluster along two similar lines. The clustering along a line indicates that the cost of adding one amino acid is roughly constant on average. The linearity is not preserved for very short sequences, since these can have more irregular Viterbi paths. For very long sequences (above model length) the linearity becomes increasingly precise. This is because the Viterbi paths of very long sequences, with a fixed average composition, must rely heavily on insert states and in fact are forced to loop many times in a particular insert state that becomes predominant as the length goes to infinity. The predominant insert state is the most cost-effective one. It is easy to see that the cost-effectiveness of an insert state k depends equally on two factors: its self-transition probability t_{kk} and the cross-entropy between its emission probability vector e_{kX} and the fixed probability distribution associated with the sequences under consideration. More precisely, if we look at the scores of long random sequences generated using a fixed source $P = (p_X)$ as a function of sequence length, the corresponding

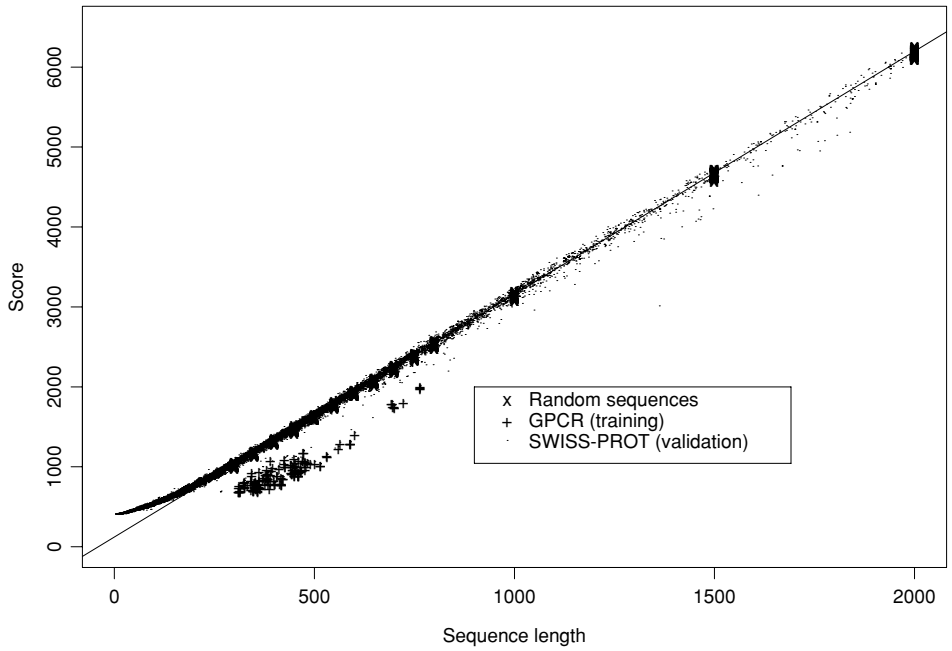

Figure 8.2: Scores (Negative Log-likelihoods of Optimal Viterbi Paths). Represented sequences consist of 142 GPCR training sequences, all sequences from the SWISS-PROT database of length less than or equal to 2000, and 220 randomly generated sequences with same average composition as the GPCRs of length 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800 (20 at each length). The regression line was obtained from the 220 random sequences.

scores cluster along a regression line with slope

$$\min_k[-\log t_{kk} - \sum_{\mathsf{X}} p_{\mathsf{X}} \log e_{k\mathsf{X}}]. \tag{8.1}$$

Furthermore, for a large fixed length l, the scores are approximately normally distributed (Central Limit Theorem) with variance

$$l\,[\mathbf{E}_P(\log^2 e_{k\mathsf{X}}) - \mathbf{E}_P(\log e_{k\mathsf{X}})] = l\,\mathbf{Var}_P[\log e_{k\mathsf{X}}]. \tag{8.2}$$

In particular, the standard deviation of the scores increases as the square root of the length l. Proof of these results and additional details can be found in [38].

Random sequence length l	Number of sequences	Empirical average score (AS)	Predicted AS $3.038l + 122.11$	Empirical SD $0.66\sqrt{l}$	Predicted SD
300	100	1041.4	1033.5	13.24	11.43
350	100	1187.1	1185.4	13.12	12.34
400	100	1337.6	1337.3	12.50	13.20
450	100	1487.6	1489.2	16.85	14.00
500	100	1638.5	1641.1	13.74	14.75
550	100	1790.3	1793.0	15.26	15.47
600	100	1944.4	1944.9	16.70	16.16
650	100	2093.3	2096.8	16.54	16.82
700	100	2250.6	2248.7	18.65	17.46
750	100	2397.9	2400.6	16.96	18.07
800	100	2552.5	2552.5	19.66	18.66
1000	100	3160.2	3160.1	21.62	20.87
1500	200	4678.9	4679.1	25.51	25.56
2000	200	6199.1	6198.1	29.59	29.51

Table 8.1: Statistics of the Scores of Randomly Generated Sequences with Similar Average Composition as the GPCRs (8.2).

The formula derived for the slope is true asymptotically and does not necessarily apply for relatively small lengths, although this is the case for the present model. For the model under consideration, the optimal insert state for average composition identical to GPCRs is insert state 20. The equation of the empirical regression line is $y = 3.038l + 122.11$, whereas the approximation in (8.1) yields a slope prediction of 3.039. From the estimate in (8.2), the standard deviation should increase as $\sigma \approx 0.66\sqrt{l}$. An empirical regression of the standard deviation on the square root of the length gives $\sigma \approx 0.63\sqrt{l} + 1.22$. There is good agreement between the theoretical estimates and the empirical results, as can be seen in table 8.1. Generally, of course, the quality of the fit improves with the length of the sequences, and this is most evident for the standard deviation. In the present case, however, (8.1) and (8.2) are quite accurate even for relatively short sequences, with length comparable to or even less than the length of the model. Similar results are obtained if we use a different random source based on the average composition of the SWISS-PROT sequences.

8.1.4 Score Normalization, Database Searches, and Discrimination Tests

Having done this statistical analysis, we can now address the obvious question of how to conduct discrimination tests, that is, how to decide algorithmically whether a sequence belongs to the GPCR family or not. Clearly, one would like to use the scores produced by the model to discriminate between GPCR and non-GPCR sequences. However, the raw scores cannot be used directly because (a) the scores tend to grow linearly with the length and (b) the dispersion of the scores varies with the length and, at least in the case of long, randomly generated sequences, increases in proportion to the square root of the length. Therefore the raw scores need to be centered and scaled first.

This normalization procedure can be done in several ways. For centering, one can use empirical averages calculated at each length, or averages derived from empirical regression lines, or average estimates derived from (8.1) and (8.2). Depending on the final goal, the base level can be calculated with respect to random sequences of similar composition or with respect to an actual database, such as SWISS-PROT. In the present case, the two are similar but not identical. For scaling, one can use empirical standard deviations or theoretical estimates and these can be calculated again on different sources such as SWISS-PROT or random sequences of similar composition. Each method has its advantages and drawbacks, and in practical situations one may try several of them. In general, empirical estimates may be more accurate but also more costly, especially for long sequences, since the calculation of the corresponding scores grows with the square of the length $O(l^2)$.

When using an actual database for centering or scaling, problems can arise if few sequences are present in the database from a given length interval of interest; it also may not be possible to remove the sequences belonging to the family being modeled from the database if these are not known a priori. This is particularly dangerous in the estimation of standard deviations. Here, it may be necessary to use an iterative algorithm where at each step a new standard deviation is calculated by ignoring the sequences in the database that are detected as members of the family at the corresponding step. The new standard deviation is used to generate a new set of normalized scores, as well as a new set of putative members of the family. Another general problem is that of short sequences, which often behave differently from very long ones. In certain cases, it may be practical to use a different normalization procedure for short sequences. Finally, in the case of an HMM library, a fixed set of randomly generated sequences, with the same average composition as SWISS-PROT, could be used across different models.

In the GPCR example, for any sequence O of length l, we use the normalized score $\mathcal{E}_S(O)$ based on the residual with respect to the empirical regression

line of the random sequences of similar average composition, divided by the approximate standard deviation derived from (8.2):

$$\mathcal{E}_S(O) = \frac{[3.038l + 122.11 - \mathcal{E}(O)]}{0.66\sqrt{l}}, \tag{8.3}$$

where $\mathcal{E}(O)$ is the negative log-likelihood of the Viterbi path. One obvious issue is the setting of the detection threshold. Here, the smallest score on the training set is 16.03 for the sequence labeled UK33_HCMVA. This low score is isolated because there are no other scores smaller than 18. Thus the threshold can be set at 16 or a little higher. By removing very long sequences exceeding the maximal GPCR length as well as sequences containing ambiguous amino acids, the search algorithm presented here yields no false negatives and two false positives (threshold 16) or one false negative and no false positives (threshold 18). At short lengths (below the length of the model), (8.2) is not necessarily a good approximation, so that it may be worthwhile to try a mixed scheme where a normalization factor is calculated empirically at short lengths ($l < N$) and (8.2) is used for larger lengths ($l > N$). Finally, thresholds may be set using the fact that the extreme score of a set of random sequences of fixed length follows an extreme value distribution [550].

8.1.5 Hydropathy Plots

Because of the particular structure of the GPCRs, one may reasonably conclude that it should be possible to detect easily whether a given sequence belongs to this class by drawing its hydropathy plot according to one of the well-known hydropathy scales [166]. If this was the case, it would render the HMM approach much less attractive for detection experiments, at least for this particular family. To check this point, hydropathy plots of a number of sequences were constructed, using a 20-amino-acid window. Examples of plots obtained for three sequences are given in figure 8.3. As can be seen, these plots can be very noisy and ambiguous. Therefore it seems very unlikely that one could achieve good detection rates based on hydropathy plots alone. Consensus patterns, hydropathy plots, and HMMs should rather be viewed as complementary techniques.

One can also compute a hydropathy plot from the HMM probabilities, as explained in chapter 7. Such a plot, shown in figure 8.4, displays the expected hydropathy at each position, rather than the hydropathy observed in any individual sequence. As a result, the signal is amplified and the seven transmembrane regions are clearly identifiable.

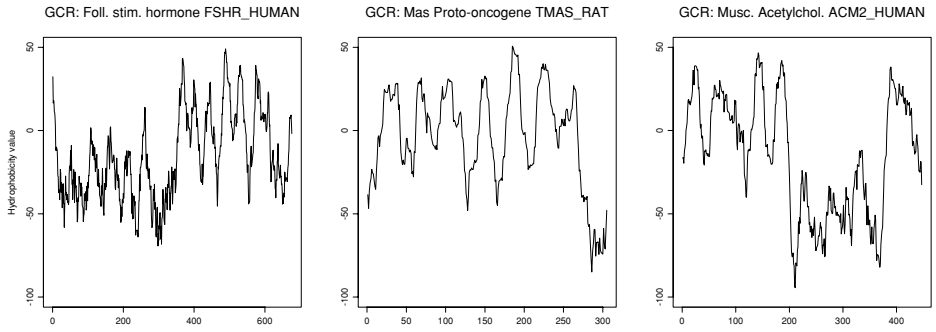

Figure 8.3: Hydropathy Plots for Three GPCRs of Length Less Than 1000, Using a Window of 20 Amino Acids. The vertical axis represents free energy for transferring a hypothetical alpha-helix of length 20, at the corresponding location, from the membrane interior to water. A peak of 20 kcal/mol or more usually signals the possible presence of a transmembrane alpha-helix.

8.1.6 Bacteriorhodopsin

Bacteriorhodopsin (see [317] for a brief review and [248] for a structural model) is a seven-transmembrane-domain protein that functions as a light-driven proton pump in *Halobacterium halobium*. Although it is functionally related to rhodopsin, it is not a GPCR. Structural and evolutionary relationships between bacteriorhodopsin and the GPCRs are not entirely clear at the moment. The raw score given by the HMM to bacteriorhodopsin is 852.27 for the primary sequence given in [411], and 851.62 for the slightly different sequence in [318]. Since the length of bacteriorhodopsin is $l = 248$, these scores are in fact close to the regression line constructed on the random sequences of similar average composition, and slightly below it. The residual of the first sequence, for instance, is 23.26 and its normalized score is 2.23, according to (8.3). This confirms that bacteriorhodopsin is not a GPCR and is consistent with the lack of a significant degree of homology between bacteriorhodopsin and GPCRs.

In [414] it is suggested that a higher degree of homology can be obtained by changing the linear order of the helices, and that the sequences may be evolutionarily related via exon shuffling. We thus constructed a new sequence by moving the seven helices of bacteriorhodopsin into the order (5,6,7,2,3,4,1), as suggested by these authors. Intracellular and extracellular domains were left untouched. The raw HMM score of this artificial sequence is 840.98. Although it is closer to the GPCR scores, the difference does not appear to be particularly significant. The HMM scores therefore do not seem to provide much support

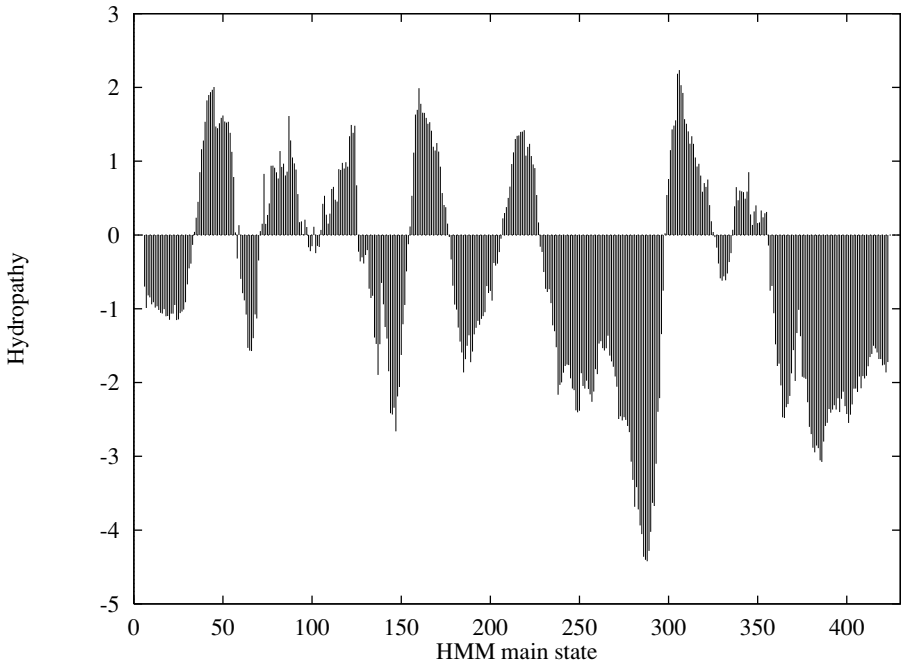

Figure 8.4: Hydropathy Plot for the GPCR HMM.

for the hypothesis presented in [414]. This point, however, requires further work because of the relatively short length of bacteriorhodopsin and the role the nonhelical domain may play in the scores.

8.1.7 Classification

By "classification" we mean the organization of a family of sequences into subclasses. This can be useful, for instance, in phylogenetic reconstruction. Classification using HMMs can be achieved in at least two different ways: (1) by training several models in parallel (figure 8.5) and using some form of competitive learning [334], or (2) by looking at how likelihoods and paths cluster within a single model. The first approach is not suitable here: the total number of sequences we have, especially for some receptor classes, is too small to train—for, say, 15 models in parallel. This experiment would require further algorithmic developments, such as the inclusion of prior information in the models, as well as new versions of the databases with more sequences.

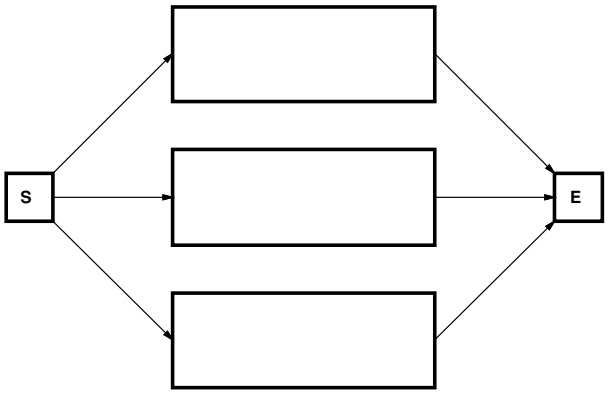

Figure 8.5: Classification HMM. Schematic representation of the type of multiple HMM architecture used in [334] for detecting subfamilies within a protein family. Each "box" between the start and end states corresponds to a single standard HMM.

For the second approach, it is clear from visual inspection of the multiple alignment that there are clusterings and interesting relationships among the Viterbi paths corresponding to different receptor subgroups. For instance, all the thyrotropin receptor precursors (TSHR) have a long initial loop on insert state 20, the same state that is optimal for (8.1). Interestingly, the same is true for the lutropin-gonadotropic hormone receptor precursor (LSHR). Here, we shall not attempt to exploit these relationships systematically to classify GPCRs from scratch, but rather shall analyze the behavior of the HMM scores with respect to the preexisting classification into major receptor classes.

For this purpose, we first extract all receptor classes for which we have at least seven representative sequences in order to avoid major bias effects. The classes and the number of corresponding sequences are olfactory (11), adenosine (9), opsin (31), serotonin (18), angiotensin (7), dopamine (12), acetylcholine (18), and adrenergic (26), for a total of 132 sequences representing 62% of the extended database obtained after searching SWISS-PROT. The histogram of the distances or normalized scores to the random regression line of the sequences in the eight classes selected in this way is plotted in figure 8.6. The normalized scores extend from 20 to 44 and are collected in bins of size 2.

The clustering of all the sequences in a given receptor subclass around a particular distance is striking. Olfactory receptors are the closest to being random. This is perhaps not too surprising, since these receptors must interact with a very large space of possible odorants. Adrenergic receptors are the most distant from the random regression line, and hence appear to be the most con-

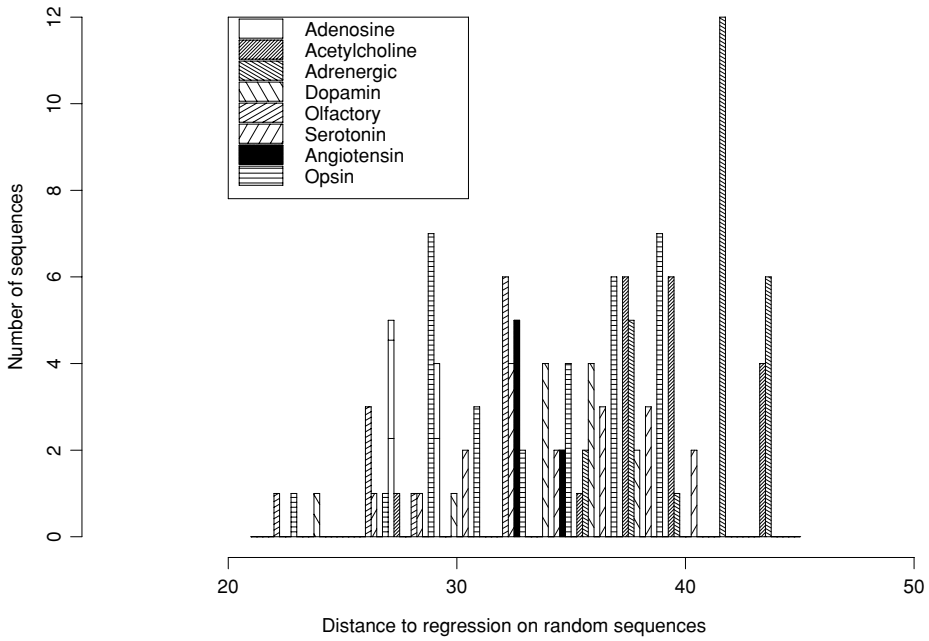

Figure 8.6: Histogram of the Distances (Normalized Scores) to the Randomly Generated Sequences for Different Classes of GPCRs. Olfactory receptors are closest to being random. Adrenergic receptors appear to be the most constrained and the most distant from the line. Different classes of receptors tend to cluster at different distances. Angiotensin receptors have a particularly narrow distribution of distances.

strained. There are also apparent differences on the standard deviation of each class. For instance, the angiotensin receptors occupy a narrow band, and only one angiotensin receptor type is known, whereas the opsin receptors are more spread out. Most classes seem to have a bell-shaped distribution, but there are exceptions. The opsins appear to have a bimodal distribution. This could be the result of the existence of subclasses within the opsins. The second peak corresponds mostly to rhodopsin (OPSD) sequences and a few red-sensitive opsins (OPSR). The presence of two peaks does not seem to result from differences between vertebrate and invertebrate opsins. With future database releases, it may be possible to improve the resolution and reduce sampling effects. But even so, these results suggest a strong relationship between the score assigned to a sequence by the HMM model and the sequence's member-

ship in a given receptor class. On the other hand, it must also be noted that it would be very difficult to recover the underlying class structure from the histogram of scores alone, without any a priori knowledge of receptor types. A detailed classification of the entire GPCR family together with a complete phylogenetic reconstruction is beyond our scope here.

8.1.8 Fragment Detection from ESTs and cDNA

As a result of EST and cDNA sequencing efforts over the past few years, there are several databases of DNA sequences corresponding to protein fragments. It is naturally of interest to be able to recognize and classify such fragments, and to be able to recover any new useful information. HMMs could be tailored to such tasks in several ways. One obvious possibility is, for a given protein family, to train different HMMs to recognize different portions of the protein. Here we conducted a number of preliminary tests using the GPCR family and artificially generated fragments. While the typical length of interest to us was around $l = 150$, we also investigated what happens at smaller lengths, and when sequencing noise is taken into account. Sequencing noise was approximated by converting amino acid sequences to DNA and introducing random independent changes in the DNA with a fixed noise probability p. We concentrated on three length levels: $l = 150$, 100, and 50, and three noise percentage levels: $p = 0$, 5, and 10.

We first constructed five data sets, all containing fragments of fixed length 150. In the first set the fragments were extracted at random locations from the training data set of 142 GPCRs. In the second set, 200 fragments were extracted randomly from a larger database of GPCRs [325]. In the third set, we generated 200 random sequences of fixed length 150 with average composition identical to the GPCRs. In the fourth set, we randomly extracted segments of length 150 from a database of kinase sequences. Finally, in the fifth set we did the same but using the SWISS-PROT database.

As with pairwise sequence alignments, HMMs can be used to produce both local or global alignments. Here we analyze the scores associated with global alignments to the model, that is with the negative log-likelihoods of the complete Viterbi paths. The histograms of the corresponding scores are plotted in figure 8.7. These results show in particular that with a raw score threshold of about 625, such a search can eliminate a lot of the false positives while producing only a reasonable number of false negatives. The same results are plotted in figure 8.8, but with a length $l = 50$ and a noise $p = 10\%$. As can be seen, the overlap between the distributions is now more significant. This of course requires a more careful analysis of how performance deteriorates as a function of fragment length and noise across the entire SWISS-PROT database.

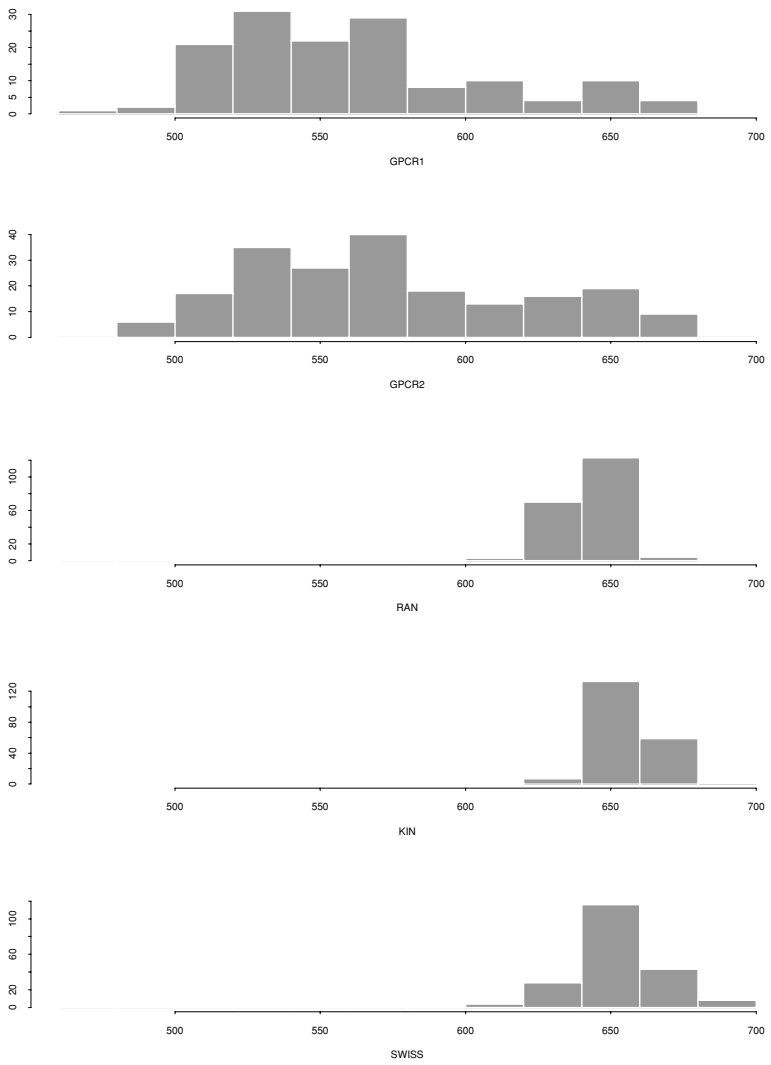

Figure 8.7: Histogram of Scores of Different Fragment Sequences of Length 150. The first histogram is constructed from 142 random fragments taken from the training set. All other histograms are based on 200 fragment sequences taken, in a random fashion, from a larger database of GPCRs, from randomly generated sequences with similar average composition, from a database of kinases, and from SWISS-PROT, respectively.

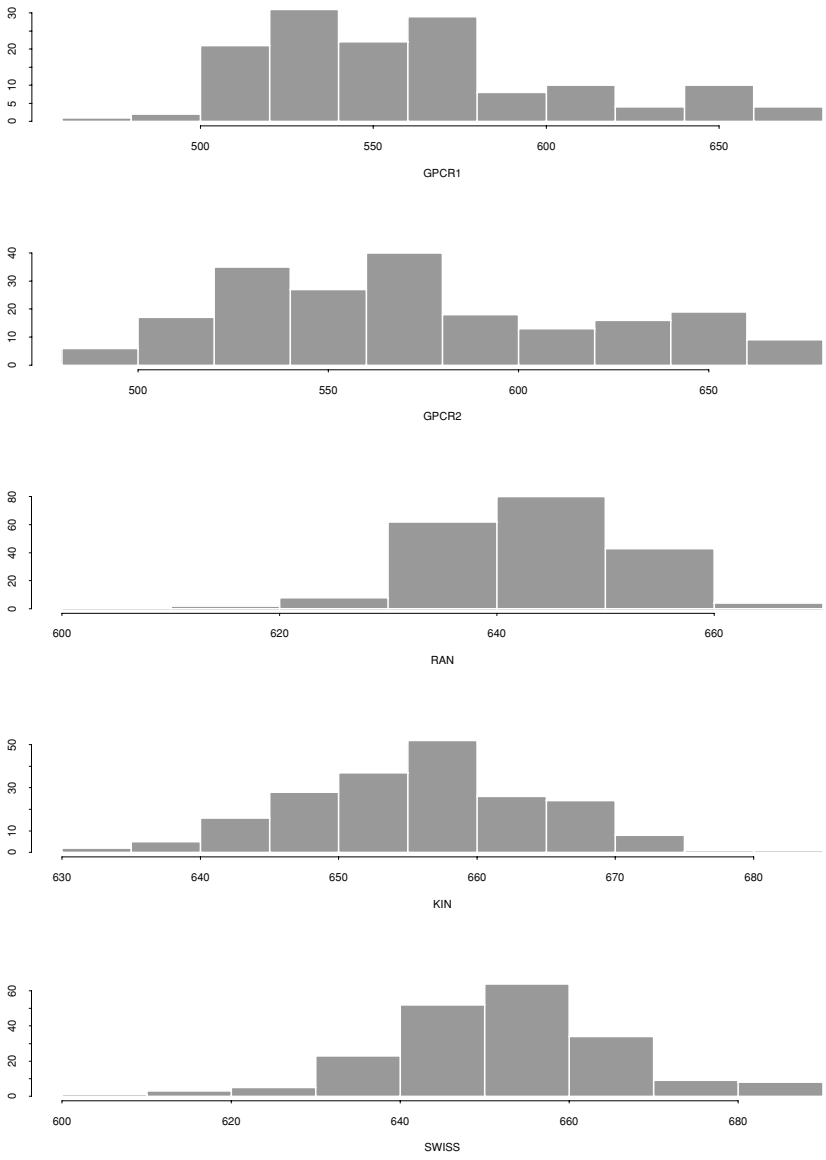

Figure 8.8: As Figure 8.7, with Fragments of Length 50 and $p = 10\%$ Noise Level.

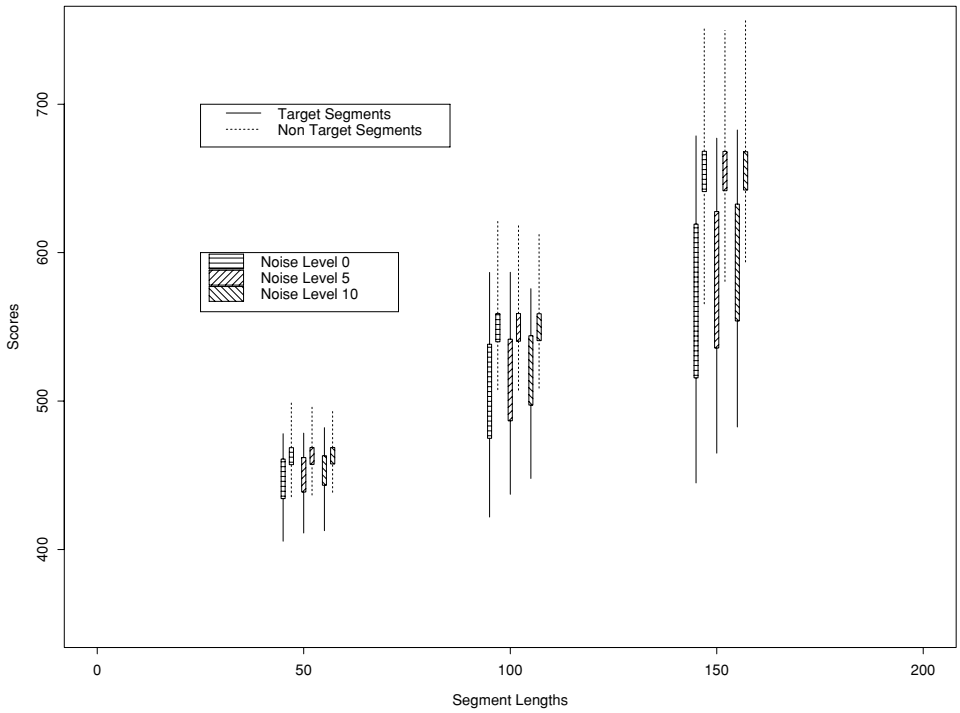

Figure 8.9: Summary of Scores on Entire SWISS-PROT Database. Segment lengths are shown on the horizontal axis, segment scores on the vertical axis. The figure depicts the standard deviations (striped columns), and the ranges (thin lines) of the scores, for both target (GPCR) sequences and non-target sequences, for all three segment lengths (50, 100, and 150) and noise levels (0, 5, and 10).

Summary of Results

The overall results are summarized in figure 8.9. Segment lengths are shown on the horizontal axis. Segment scores are shown on the vertical axis. The figure depicts the standard deviations (striped columns) and the ranges (thin lines) of the scores, for both target (GPCR) sequences and nontarget sequences, for all three fragment lengths (50, 100, and 150) and noise levels (0, 5, and 10). For each fragment length, the lines represent the ranges for all noise levels for target (GPCR) and nontarget sequences. To make all possible ranges for all noise levels visible, the lines representing the score ranges are slightly displaced with respect to the real fragment length.

At a given fragment length (e.g., 50), six lines represent, from left to right, noise level 0 for targets, noise level 0 for nontargets, noise level 5 for targets, noise level 5 for nontargets, noise level 10 for targets and noise level 10 for nontargets. Regression lines can be computed for all scores for all target and all nontarget fragments and each noise level:

- Target sequences

  Noise level 0: $y = 387.4 + 1.199\ l$
  Noise level 5: $y = 384.0 + 1.314\ l$
  Noise level 10: $y = 382.3 + 1.401\ l$

- Non-target sequences

  Noise level 0: $y = 364.7 + 1.909\ l$
  Noise level 5: $y = 364.8 + 1.910\ l$
  Noise level 10: $y = 364.8 + 1.911\ l$

These regression lines are obtained from only three fragment lengths. Therefore they constitute only an approximation to the scores at all intermediary lengths. The lines intercept for a fragment length of about 35. This means that 35 is the approximate length limit for nonzero discrimination based on scores alone.

As expected, for the target sequences the slopes of the regression lines substantially increase with noise. Intercepts do not vary much. The slopes and intercepts for the nontarget sequences are stable; the noise level does not have a strong influence on nontarget sequences. The approximate regression line for all nontarget sequences is $y \approx 364.8 + 1.91l$. Consistent with the results in [38], this slope is inferior to the slope of the similar line that can be derived at greater lengths. The standard deviations of the scores can be studied similarly, as a function of length and noise level.

ROC results

After scoring the entire database, one can compute, for each length and each noise level, the number of true and false positives and the number of true and false negatives, for a given score threshold. These sensitivity/selectivity results can be summarized by drawing the corresponding ROCs (receiver operating characteristics), as in figure 8.10.

ROC curves are obtained by computing, for threshold values scanned within a given range, the sensitivity or hit rate (proportion of true positives) and the selectivity or false alarm rate (proportion of false positives) from the

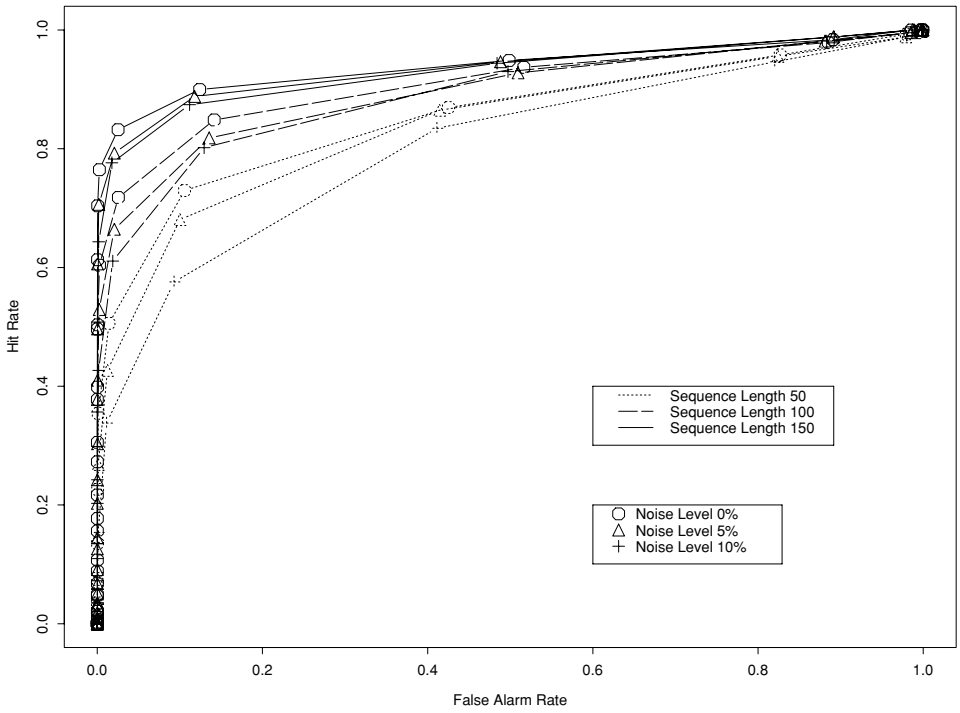

Figure 8.10: ROCs for All Scores of All SWISS-PROT Fragments at Lengths 50, 100, and 150 and Noise Levels 0, 5, and 10. Sequences with ambiguous symbols are filtered out.

number of true/false positives and negatives. Threshold range is a function of fragment length. For each segment length, the minimum threshold is a (rounded) value where no non-GPCR fragment is classified as positive across noise levels; the maximum threshold is a (rounded) value where no known GPCR (from PROSITE) is classified negative across noise levels. These curves provide a convenient means for setting thresholds as a function of desirable goals. As can be seen, there is a nice progressive ordering of the curves as a function of noise and length. The curves tend to "stick" to the vertical axes. This clearly shows that very low false alarm rates are obtained even for high hit rates: there is very good detection of a large number of target sequences. However, the curves do not "stick" to the horizontal axes. This shows that to detect the higher percentage of target sequences, the number of false positives must increase substantially. This is certainly due to the fact

	0	5	10
50	1.16	1.18	1.03
100	1.63	1.49	1.50
150	2.41	2.14	1.96

Table 8.2: Imperfect "Summary" of All Results that Make Possible Estimation of the Performance of Intermediate Length Fragments and Noise Levels.

that GPCRs comprise both relatively conserved and highly variable regions. It is virtually impossible to distinguish a short fragment, extracted from a highly variable region, from the general SWISS-PROT background. Likewise, longer fragments that include more conserved regions are easier to separate from the background. For short fragment lengths and high noise levels, these curves suggest that additional filters should be constructed to improve performance.

Detection analysis with the d' measure

Given the scores of two populations to be discriminated, and assuming that these two distributions are Gaussians with the same standard deviation equal to 1, the d' measure gives the distance between the centers of the two Gaussians for a certain level of false positives and negatives.

A preliminary detection analysis of the SWISS-PROT scores with a d' measure shows that d' varies widely with the classification threshold. This indicates that the score distribution curves are not Gaussian (as can observed from the histograms). Because it would be interesting to give a single measure of performance for each noise level and fragment length, the following method is used. A linear interpolation measure of false alarm rates is computed for a hit rate of 0.9 at each noise level and fragment length. The d' measure is then computed for the resulting pair $(0.9, x)$, where x is the linearly interpolated false alarm value. Table 8.2 gives the results for each noise level and fragment length.

Improving Detection Rates

So far we have examined only the raw scores produced by the HMM, that is, the negative likelihood of the Viterbi paths. HMMs, however, contain considerable additional information that could be used in principle to improve database mining performance. In fact, for each fragment, a number of additional indicators can be built and combined—for instance, in a Bayesian network—to improve performance. Most notably, the structure of the paths themselves can be used. As one might expect, there is a clear difference between the paths of

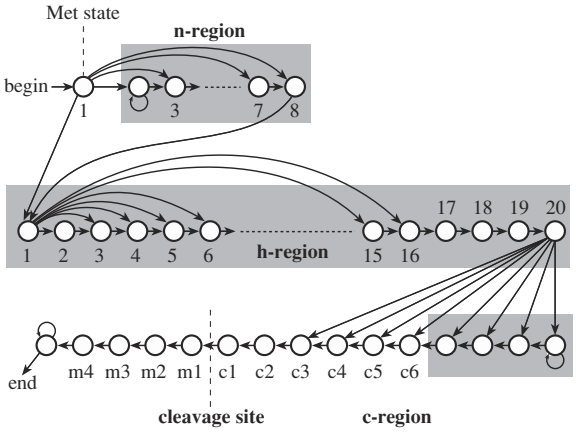

Figure 8.11: The HMM Used for Signal Peptide Discrimination. The model [406] is designed so that it implements an explicit modeling of the length distribution in the various regions.The states in a shaded box are tied to one another.

true and false positives. The path of a false positive is on average more discontinuous and contains a larger number of gaps. Several measures of path discontinuity can be constructed. One can use (1) the number of transitions out of delete states in a path; (2) the length of the longest contiguous block of emitting states in a path; or (3) the logarithm of the probability of the path itself (transitions only/no emissions). In one test, the combination of such measures with raw scores improves the detection of true positives by 15–20%. Other directions for improving detection rates are explored in [42].

8.1.9 Signal Peptide and Signal Anchor Prediction by HMMs

In section 6.4.1 the problem of finding signal peptides in the N-terminal part of prokaryotic and eukaryotic sequences was introduced. The window-based neural network approach [404] can exploit the correlations among the amino acids, in particular around the cleavage site, but without extra input units, it cannot benefit from the pattern in the entire sequence and the different length distributions that characterize signal peptides.

The length properties of signal peptides are in fact known to differ between various types of organisms: bacterial signal peptides are longer than their eukaryotic counterparts, and those of Gram-positive bacteria are longer than those of Gram-negative bacteria. In addition, there are compositional differences that correlate with the position in the signal peptide and also in the first few residues of the mature protein.

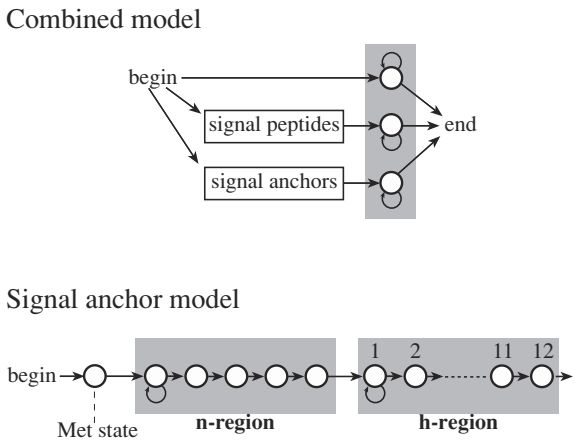

Figure 8.12: The HMM Designed to Discriminate Between Signal Peptides and Signal Anchors. The block diagram (top) shows how the combined model [406] is put together from the signal peptide model and the anchor model. The final states shown in the shaded box are tied to one another, and model all residues not in a signal peptide or an anchor. The model of signal anchors (bottom) has only two types of states (grouped by the shaded boxes) apart from the Met state.

Another important and difficult problem is that some proteins have N-terminal sequences that initiate translocation in the same way as signal peptides do, but are not cleaved by signal peptidase [541, 406]. The uncleaved signal peptide is known as a signal anchor, a special type of membrane protein. Signal anchors typically have hydrophobic regions longer than those of cleaved signal peptides, and other regions differ also in their compositional features.

Nielsen and Krogh [406] constructed a hidden Markov model designed both to discriminate between signal peptides and nonsignal peptides and to locate the cleavage site. The HMM was designed so that it took known signal peptide features into account, in particular the different regions described in section 6.4.1. In their scheme a signal peptide model was combined with a model of signal anchors, in order to obtain a prediction tool that was able to discriminate between signal peptides and anchors.

The signal peptide model is shown in figure 8.11. It implements an explicit modeling of the length distribution in the various regions using *tied* states that have the same amino acid distribution in the emission and transition probabilities associated with them.

To discriminate among signal peptides, signal anchors, and soluble nonsecretory proteins, the model was augmented by a model of anchors as shown

in figure 8.12. The whole model was trained using all types of sequences (known signal peptides and known anchor sequences, as well as cytoplasmic and nuclear sequences). The most likely path through the combined model yields a prediction of which of the three classes the protein belongs to.

In terms of predictive performance in relation to discrimination between signal peptide sequences and nonsignal peptide sequences, the combination of C-score and S-score neural networks (see section 6.4.1) had a discrimination level comparable to that of the HMM. For eukaryotes the networks were slightly better, while for Gram-negative bacteria the HMM was slightly better [406]. For discrimination between cleaved signal peptides and uncleaved signal anchors, the HMM had a correlation coefficient of 0.74, corresponding to a sensitivity of 71% and a specificity of 81%—while the S-score from the neural network could be used to obtain a performance on this task not exceeding 0.4 for the correlation coefficient. The HMM is much better at recognizing signal anchor and therefore at detecting this type of membrane-associated protein.

However, these results should not be taken as a claim that the neural network method is unable to solve the signal anchor problem, since the signal anchors were not included as training data in the neural network model, as was the case for the HMM [406].

A similar approach in the form of a structured HMM has been used to model and predict transmembrane protein toplogy in the TMHMM method [335]. TMHMM can discriminate between soluble and membrane proteins with both specificity and sensitivity better than 99%, although the accuracy drops when signal peptides are present. Due to the high degree of accuracy the method is excellent for scanning entire genomes for detection of integral membrane proteins [335].

8.2 DNA and RNA Applications

Multiple alignments of nucleotide sequences are harder to make than alignments of protein sequences. One reason is that parameters in amino acid substitution matrices can be estimated by means of evolutionary and biochemical analysis, while it is hard to obtain good measures of general mutation and deletion costs of individual nucleotides in nucleic acids. The “twilight zone” of dubious alignment significance is reached faster for sequences from a shorter alphabet, and fewer evolutionary events are therefore needed to get into the twilight zone when aligning DNA.

HMMs do not a priori require an explicit definition of the substitution costs. The HMM approach avoids the computationally hard many-to-many multiple-sequence alignment problem by recasting it as a many-to-one sequence-to-HMM alignment problem [155]. The different positions in a model can in prac-

tice have individual implicit substitution costs associated with them. These features have contributed to the fact that in several cases HMMs applied to nucleic acids have led to the discovery of new patterns not previously revealed by other methods. In protein-related applications, HMMs have more often led to improvements of earlier methods.

8.2.1 Gene Finding in Prokaryotes and Eukaryotes

Gene finding requires the integration of many different signals: promoter regions, translation start and stop context sequences, reading frame periodicities, polyadenylation signals, and, for eukaryotes, intron splicing signals, compositional contrast between exons and introns, potential differences in nucleosome positioning signals, and sequence determinants of topological domains. The last involves the matrix (or scaffold) attachment regions (MARs or SARs), which are associated with higher-order chromosomal organization. The attachment signals may be involved in promoting transcriptional activity in vivo, and have recently been reported to be present between genes. For prokaryotes the DNA sequence also needs to allow strong compaction in a chromatin-like structure. The length of the extended DNA from a single operon corresponds to the diameter of the cell. Since all these signals to a large extent complement each other, in the sense that some may be weak when others are strong, a probabilistic approach for their integration is the natural way to handle the complexity of the problem.

In prokaryotes, gene finding is made simpler by the fact that coding regions are not interrupted by intervening sequences. Still, especially for relatively short open reading frames, it is nontrivial to distinguish between sequences that represent true genes and those that do not. In the highly successful gene finder GeneMark [81, 83, 82], which in its first version was based on frame dependent nonhomogeneous Markov models, a key feature strongly improving the performance is a clever detection of the "shadow" of a true coding region on the non-coding strand (for further detail see chapter 9).

A hidden Markov model has also been developed to find protein-coding genes in *E. coli* DNA [336] (work done before the complete *E. coli* genome became available). This HMM includes states that model the codons and their frequencies in *E. coli* genes, as well as the patterns found in the intergenic region, including repetitive extragenic palindromic sequences and the Shine–Dalgarno motif. To take into account potential sequencing errors and/or frameshifts in a raw genomic DNA sequence, it allows for the (very unlikely) possibility of insertions and deletions of individual nucleotides within a codon. The parameters of the HMM are estimated using approximately 1 million nucleotides of annotated DNA, and the model is tested on a disjoint set of contigs containing

about 325,000 nucleotides. The HMM finds the exact locations of about 80% of the known *E. coli* genes, and approximate locations for about 10%. It also finds several potentially new genes and locates several places where insertion or deletion errors and/or frameshifts may be present in the contigs.

A number of powerful HMMs and other probabilistic models for gene finding in eukaryotes had been developed (see chapter 9 and [343, 107] and references therein). Eukaryotic gene models are typically built by assembling a number of components, such as submodels for splice sites, exons, and introns to take advantage of the corresponding weak consensus signals and compositional differences. The individual submodels must remain relatively small if the goal is to scan entire genomes in reasonable time. Other key elements include the use of three exon submodels in parallel in order to take into account the three possible ways introns may interrupt the reading frame, as well as features to incorporate exon and intron length distributions, promoters, poly-adenylation signals, intergenic sequences, and strand asymmetry. It is often better to train the entire recognition system at once, rather than each of its components separately. In particular, the standard HMM algorithms can be modified in order to optimize the global gene parse produced by the system rather than the sequence likelihoods [333]. The best gene recognition is achieved by some of these models [107], with complete exon recognition rates in the 75 to 80% range (with exact splice sites). Additional work is required to improve the detection rates further. Such improvements may come from the incorporation of new, better submodels of promoters or initial and terminal exons, as well as other physical properties and signals present in the DNA, such as bendability or nucleosome positioning. Such compactification signals, which have been completely neglected so far, are likely to play an important role in the biological gene-finding machinery as well. In the rest of this chapter, we build relatively large models of gene components and describe such possible signals.

8.2.2 HMMs of Human Splice Sites, Exons, and Introns

Strong research efforts have been directed toward the understanding of the molecular mechanism responsible for intron splicing ever since it was discovered that eukaryotic genes contain intervening sequences that are removed from the mRNA molecules before they leave the nucleus to be translated. Since the necessary and sufficient sequence determinants for proper splicing are still largely unknown, probabilistic models in the form of HMMs have been used to characterize the splicing signals found experimentally.

Unlike the case of protein families, it is essential to remark that all exons and their associated splice site junctions are neither directly nor closely related

by evolution. However, they still form a "family" in the sense of sharing certain general characteristics. For example, in a multiple alignment of a set of flanked exons, the consensus sequences of the splice sites should stand out as highly conserved regions in the model, exactly like a protein motif in the case of a protein family. As a result, one should be particularly careful to regard insertions and deletions in the HMM model as formal string operations rather than evolutionary events.

To see whether an HMM would pick up easily known features of human acceptor and donor sites, a model with the standard architecture as shown in figure 7.2 was trained on 1000 randomly selected flanked donor and acceptor sites [32, 33, 35]. By close inspection of the parameters of the HMM trained specifically on the flanked acceptor sites, it was observed that the model learns the acceptor consensus sequence perfectly: ([TC]...[TC][N][CT][A][G][G]). The pyrimidine tract is clearly visible, as are a number of other known weak signals, such as a branching (lariat) signal with a high A in the 3' end of the intron. (See figure 8.13.)

Similarly, the donor sites are clearly visible in a model trained on flanked donor sites but are harder to learn than the acceptor sites. The consensus sequence of the donor site is learned perfectly: ([CA][A][G][G][T][AG][A][G]). The same is true for the G-rich region [164], extending roughly 75 bases downstream from the human donor sites (figure 8.13). The fact that the acceptor site is easier to learn is most likely explained by the more extended nature of acceptor site regions as opposed to donor sites. However, it could also result from the fact that exons in the training sequences are always flanked by *exactly* 100 nucleotides upstream. To test this hypothesis, a similar model using the same sequences, but in *reverse* order, is trained. Surprisingly, the model still learns the acceptor site much better than the donor site (which is now downstream from the acceptor site). The random order of the nucleotides in the polypyrimidine tract region downstream from the acceptor site presumably contributes to this situation. In contrast, the G-rich region in the 5' intron end has some global structure that can be identified by the HMM.

8.2.3 Discovering Periodic Patterns in Exons and Introns by Means of New HMM Architectures

In another set of experiments a standard HMM was trained on human exons flanked by intron sequence. A set of 500 randomly selected flanked internal exons, with the length of the exons restricted to between 100 and 200 nucleotides, was used (internal human exons have an average length of $\approx$ 150 nucleotides).

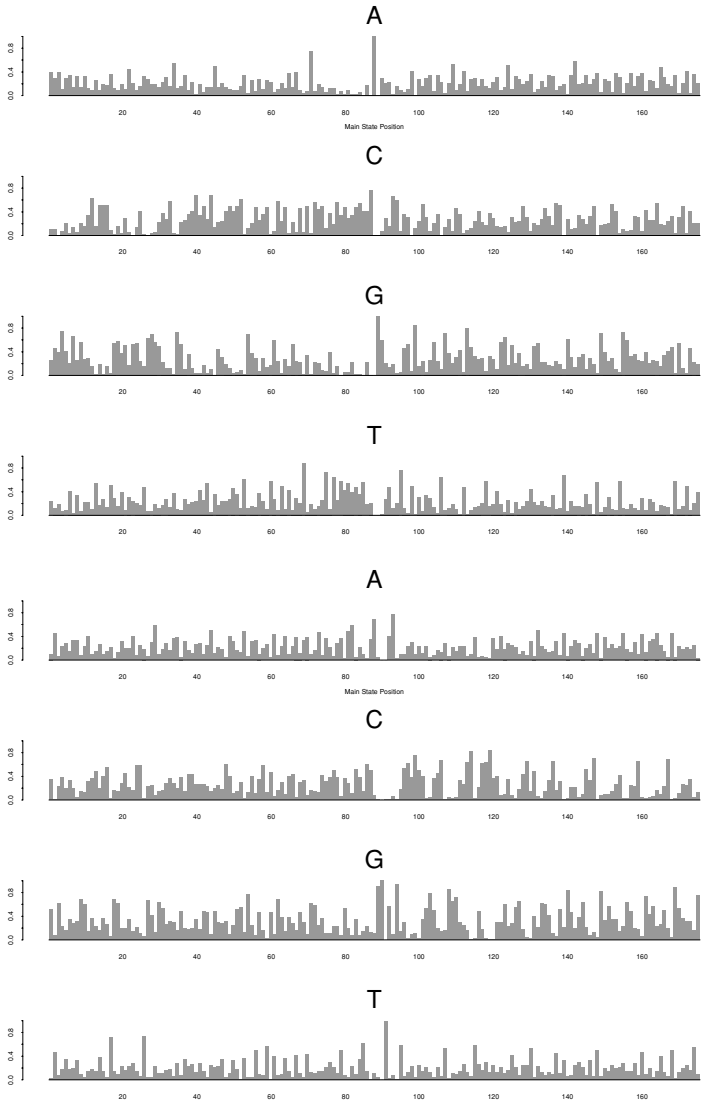

Figure 8.13: Emission Distribution from Main States of an HMM Model Trained on 1000 Acceptor (top) and 1000 Donor Sites (bottom). The flanking sequence is kept constant with 100 nucleotides on each side; the model, however, has length 175. For the acceptor sites, the characteristic consensus sequence is easily recognizable ([TC]...[TC][N][CT][A][G][G]). Note the high A probability associated with the branch point downstream from the acceptor site. The characteristic consensus sequence of the donor site is also easily recognizable ([CA][A][G][G][T][AG][A][G]). Learning is achieved using the standard architecture (figure 7.2) initialized uniformly, and by adding a regularizer term to the objective function that favors the backbone transition path.

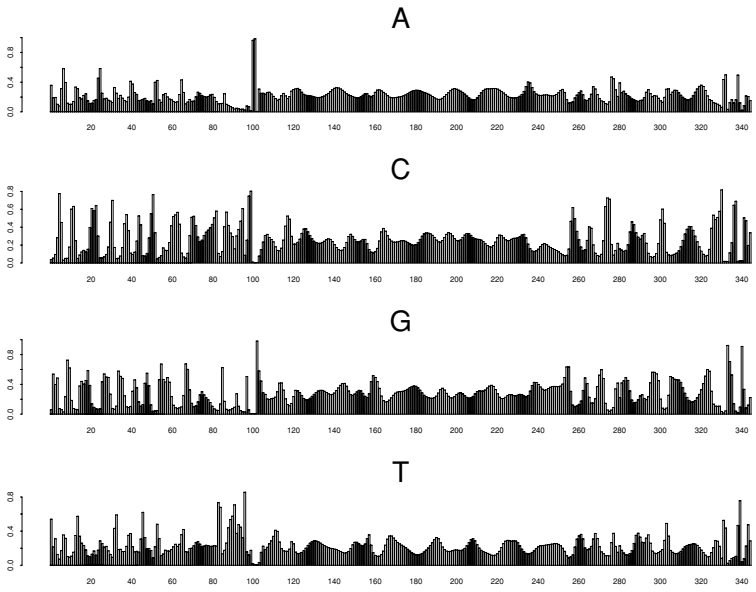

Figure 8.14: Emission Distribution from Main States of an HMM Model Trained on 500 Flanked Internal Exons. The length of the exons was constrained to the interval between 100 and 200 nucleotides, with average of 142, and fixed intron flanking of 100 on each side. The number of main states in the model was 342. Note the oscillatory pattern in the exon region and outside.

The probability of emitting each of the four nucleotides, across the main states of the trained model, is plotted in figure 8.14. We see striking periodic patterns, especially in the exon region, characterized by a minimal period of 10 nucleotides with A and G in phase, and C and T in antiphase. A periodic pattern in the parameters of the models of the form [AT][CG] (or [AT]G), with a periodicity of roughly 10 base pairs, can be seen at positions 10, 19, 28, 37, 46, 55, 72, 81, 90, 99, 105, 114, 123, 132 and 141. The emission profile of the backbone was also compared for two nucleotides jointly. The plots of A+G and C+T are considerably smoother than those of A+T and C+G on both the intron side and the exon side. The 10 periodicity is visible both in the smooth phase/antiphase pattern of A+G and C+T and in the sharp contrast of high A+T followed by high C+G. There is also a rough three-base pair periodicity, especially visible in C+G, where every third emission corresponds to a local minimum. This is consistent with the reading frame features of human genes [525], which are especially strong on the third codon position (≈30% C and ≈26% G; see figure 6.11).

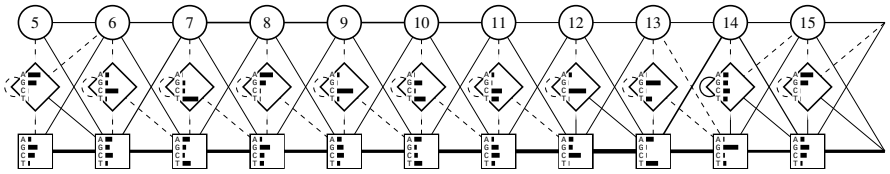

Figure 8.15: The Repeated Segment of the Tied Model. Rectangles represent main states and circles represent delete states. Histograms represent emission distributions from main and insert states. Thickness of connections is proportional to corresponding transition distribution. Position 15 is identical to position 5.

In order further to characterize the periodicity, a wide range of different HMM architectures were trained on nonflanked *internal* exons, in order to separate features from the special gradients in the nucleotide composition known to be present in initial and terminal exons [164]. When training on the bulk of the internal exons in the length interval between 100 and 200 nucleotides, a clear and consistent pattern emerged in the emission probabilities, no matter which architecture was applied. The architectural variation included conventional left–right HMM models, left–right models with identical segments "tied" together, and loop or "wheel" models with better ability to reveal periodic patterns in the presence of noise. Although the conventional type of left–right architecture is not the ideal model of an exon, due to the large length variations, it still identifies the periodic pattern quite well.

To test the periodicity yet further, a "tied" exon model with a hardwired periodicity of 10 was trained [33]. The tied model consists of 14 identical segments of length 10 and five additional positions in the beginning and the end of the model, making a total length of 150. During training the segments are kept identical by *tying* the parameters—that is, the parameters are constrained to be exactly the same throughout learning, as in the weight-sharing procedure for neural networks. The model was trained on 800 internal exon sequences of length between 100 and 200, and it was tested on 262 different sequences. The parameters of the repeated segment after training are shown in figure 8.15. Emission probabilities are represented by horizontal bars of corresponding proportional length. There is a lot of structure in this segment. The most prominent feature is the regular expression `[^T][AT]G` at positions 12–14. The same pattern was often found at positions with very low entropy in the standard models described above. In order to test the significance, the tied model was compared with a standard model of the same length. By comparing the average negative log-likelihood they both assign to the exon sequences and

Loop states	A	C	G	T
I1	0.1957	0.4808	0.1986	0.1249
M1	0.3207	0.0615	0.0619	0.5559
I2	0.0062	0.0381	0.5079	0.4478
M2	0.1246	0.2982	0.5150	0.0622
I3	0.4412	0.1474	0.2377	0.1737
M3	0.2208	0.6519	0.1159	0.0114
I4	0.2743	0.5893	0.0676	0.0689
M4	0.3709	0.0113	0.0603	0.5575
I5	0.1389	0.2946	0.0378	0.5287
M5	0.0219	0.0121	0.9179	0.0481
I6	0.0153	0.9519	0.0052	0.0277
M6	0.0905	0.1492	0.7017	0.0586
I7	0.1862	0.3703	0.3037	0.1399
M7	0.3992	0.2835	0.3119	0.0055
I8	0.2500	0.4381	0.2968	0.0151
M8	0.4665	0.0043	0.1400	0.3891
I9	0.6892	0.0156	0.2912	0.0040
M9	0.0121	0.2000	0.7759	0.0120
I10	0.2028	0.3701	0.0117	0.4155
M10	0.3503	0.3459	0.2701	0.0787
I11	0.1446	0.6859	0.0861	0.0834

Table 8.3: Emission Distributions for the Main and Insert States of a Loop Model (Figure 8.16) After Training on 500 Exon Sequences of Length 100–200.

to random sequences of similar composition, it was clear that the tied model achieves a level of performance comparable with the standard model, but with significantly fewer free parameters. Therefore a period of around 10 in the exons seems to be a strong hypothesis.

As the left–right architectures are not the ideal model of exons, it would be desirable to have a model with a loop structure, possibly such that the segment can be entered as many times as necessary for any given exon. See [336] for a loop structure used for *E. coli* DNA. One example of such a true loop model is shown schematically in figure 8.16. In the actual exon experiment the loop had length 10, with two flanks of length 4. This model was trained using gradient descent and the Dirichlet regularization for the backbone transitions to favor main states. Additional regularization must be used for the anchor state as a result of its particular role and connectivity. The Dirichlet vector used for the anchor state is (0.1689 0.1656 0.1656 0.1689 0.1656 0.1656). The emission distribution of the main and insert states inside the loop is shown in table 8.3. Again the results are remarkably consistent with those obtained with the tied model. The pattern `[^T][AT]G` is clearly visible, starting at main state 3 (M3).

Table 8.4 compares the temporal evolution of the cumulative negative log-likelihood of the training set in an experiment involving three models: a free model, a tied model, and a loop model. Although, as can be expected, the free model achieves the best scores after 12 cycles, this seems to be the result of

Cycle	NLL free model	NLL tied model	NLL loop model
1	1.013e+05	1.001e+05	9.993e+04
2	1.008e+05	9.902e+04	9.886e+04
3	9.965e+04	9.884e+04	9.873e+04
4	9.886e+04	9.875e+04	9.859e+04
5	9.868e+04	9.869e+04	9.855e+04
6	9.854e+04	9.865e+04	9.849e+04
7	9.842e+04	9.862e+04	9.848e+04
8	9.830e+04	9.861e+04	9.852e+04
9	9.821e+04	9.860e+04	9.845e+04
10	9.810e+04	9.859e+04	9.842e+04
11	9.803e+04	9.859e+04	9.844e+04
12	9.799e+04	9.859e+04	9.843e+04

Table 8.4: Evolution of the NLL Scores over 12 Cycles of Gradient Descent, with $\eta = 0.01$, for a Free Model (Figure 7.2), a Tied Model (Figure 8.15), and a Loop Model (Figure 8.16). All models are trained on 500 exons of length between 100 and 200 in all reading frames.

some degree of overfitting. The loop model, and to a lesser extent the tied model, outperform the free model during the first learning cycles. The loop model performs better than the tied models at all cycles. The free model has a better score than the loop model only after cycle 7. This is also an indication that the loop model is a better model for the data.

Finally, a different sort of loop model was trained on both exon and intron sequences. This HMM architecture has the form of a "wheel" with a given number of main states, without flanking states arranged linearly or any distinction between main and insert states, and without delete states. Thus there are no problems associated with potential silent loops. Sequences can enter the wheel at any point. The point of entry can of course be determined by dynamic programming. By using wheels with different numbers of states and comparing the cumulative negative log-likelihood of the training set, the most likely periodicity can be revealed. If wheels of nine states perform better than wheels of 10 states, the periodicity can be assumed to be related to the triplet reading frame rather than to structural aspects of the DNA (see below).

Figure 8.17 displays wheel model architectures (in this case of length 10 nucleotides) where sequences can enter the wheel at any point. The thickness of the arrows from "outside" represents the probability of starting from the corresponding state. After training, the emission parameters in the wheel model showed a periodic pattern [^T][AT]G in a clearly recognizable form in states 8, 9, and 10 of the exon model (top), and in states 7, 8, and 9 in the intron model (bottom). By training wheels of many different lengths, it was found that models of length 10 yielded the best fit. Implicitly, this is also confirmed by the fact that the skip probabilities are not strong in these models. In other words, if the data were nine-periodic, a wheel model with a loop of length 10 should be able to fit the data, by heavy use of the possibility of skipping a state in the

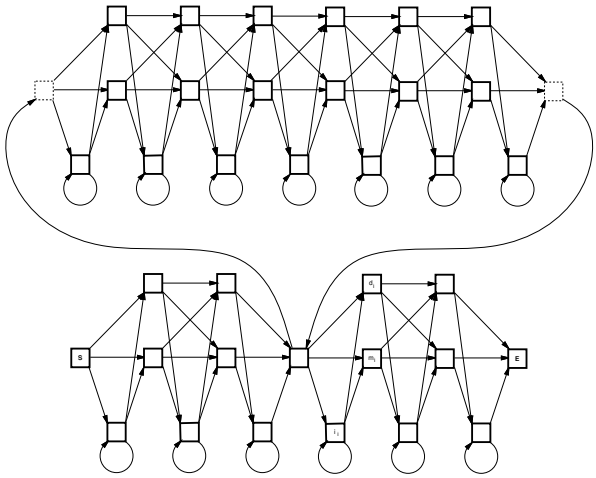

Figure 8.16: A Loop HMM Model Comprising Two Flanks and a Loop Anchored on a Silent State. The flanks and the loop are similar to the standard architecture.

wheel. State repeating in a nine-state wheel is nonequivalent to state skipping in a 10-state wheel. These wheel models do not contain independent *insert states* (as the linear left–right HMM architectures do). A repeat of the same state does not give the same freedom in terms of likelihood as if independent inserts were allowed. Moreover, in analogy to gap penalties in conventional multiple alignments, the HMM training procedure uses a regularization term favoring main states over skip states.

All the experiments were repeated using several subsets of exons starting in the one of the three codon positions in the reading frame, without any significant change in the observed patterns of the emission probabilities. For comparison, figure 8.18 shows the emission probabilities from a nine-state wheel model trained on the coding part of complete mRNA sequences of concatenated exons. This model clearly recognizes the triplet reading frame (compare to figure 6.11). The fact that the pattern is present in intron sequences provides additional evidence against a reading-frame-associated origin for the pattern in the exons.

The experiments indicate that the periodicity is strongest in exons, and possibly also in the immediate flanking intron sequence, but on the average somewhat weaker in arbitrarily selected deep intron segments. In none of the experiments using simple linear left–right HMM architectures was a clear regular oscillation pattern detected in the noncoding sequence. By using the wheel model to estimate the average negative log-likelihood per nucleotide for

various types of sequence—different types of exons, introns, and intragenic regions—it was found that the periodic pattern is strongest in exons. The period in the alignments (average distance between state 9 nucleotides) is on the order of 10.1–10.2 nucleotides.

It is well known that "bent DNA" requires a number of small individual bends that are in phase [488]. Only when bends are phased at $\approx$ 10.5 bp (corresponding to one full turn of the double helix) can stable long-range curvature be obtained. Using the wheel model to perform alignments of introns and exons, it was found that the sequence periodicity has a potential structural implication because the $\approx$ 10-periodic bending potential of the aligned sequences displays the same periodicity. The bendability of the sequences was assessed using parameters for trinucleotide sequence-dependent bendability deduced from DNaseI digestion data [96]. DNaseI interacts with the surface of the minor groove, and bends the DNA molecule away from the enzyme. The experiments [96] therefore quantitatively reveal bendability parameters on a scale where low values indicate no bending potential and high values correspond to large bending or bendability toward the major groove, for the 32 double-stranded triplets: AAA/ATT, AAA/TTT, CCA/TGG, and so on. The profiles of the bending potentials of exons and introns have been related to nucleosome positioning [34]. These differences in the strength of the signals in coding and noncoding regions have possible implications for the recognition of genes by the transcriptional machinery.

8.2.4 HMMs of Human Promoter Regions

We have also trained a number of HMMs using DNA sequences from human promoter regions. In one experiment, promoter data were extracted from the GenBank [62]. Specifically, all human sequences that contained at least 250 nucleotides upstream and downstream from an experimentally determined transcriptional start point were extracted. Sequences containing non-nucleotide symbols were excluded. The redundancy was carefully reduced using the second Hobohm algorithm [259] and a novel method for finding a similarity cutoff, described in [422]. Briefly, this method is based on performing all pairwise alignments for a data set, fitting the resulting Smith-Waterman scores to an extreme value distribution [9, 550], and choosing a value above which there are more observations than expected from the distribution. A standard linear architecture with length $N = 500$ was trained using the remaining 625 sequences, all with length 501 (see [421] for details). The training was facilitated by initializing the main state emissions associated with the TATA-box using consensus probabilities from promoters with experimentally verified TATA-boxes.

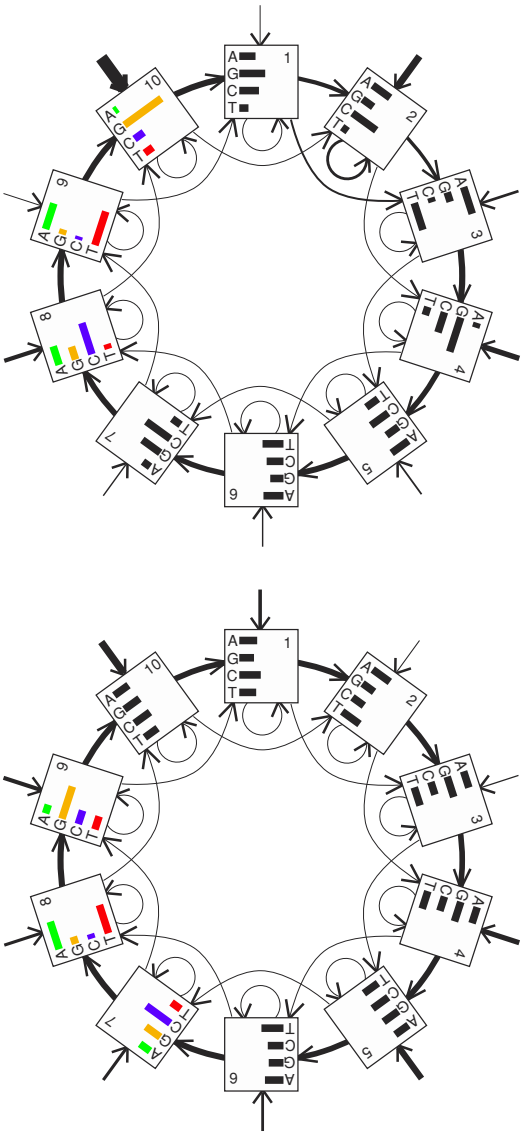

Figure 8.17: Wheel HMMs Used for Identifying Periodic Patterns. A. 10-state wheel trained on 500 internal exons of length between 100 and 200 nucleotides. Nonperfect alignment and interference with the reading frame cause features of the pattern to appear in states 2, 3, and 4 as well as 8, 9, and 10. B. 10-state wheel trained on 2000 human introns. 25 nucleotides were removed at the 5' and 3' ends in order to avoid effects of the conserved sequence patterns at the splice sites.

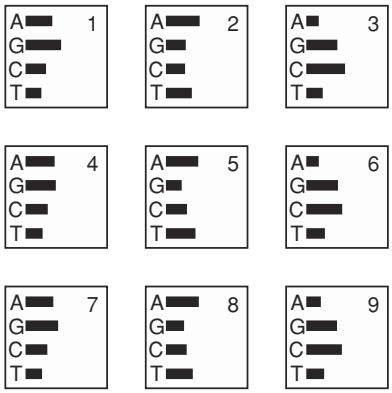

Figure 8.18: The Emission Probabilities from a Nine-State Wheel Model Trained on Complete mRNA Sequences Without the Skip and Loop Arrows. The three-periodic reading frame pattern is clearly visible, with higher frequencies of A and G, A and T, and C and G on the first, second, and third codon positions, respectively.

A bendability profile can be computed directly from the trained HMM (see Appendix D), or from the HMM-derived multiple alignment. A profile derived from a multiple alignment is shown in figure 8.19. The most striking feature is a significant increase in bendability in the region immediately *downstream* of the transcriptional start point. As promoters most often have been characterized by a number of upstream patterns and compositional tendencies, it is interesting that the HMM alignment corresponds to structurally similarity in the downstream region of these otherwise unrelated promoter sequences. They are not biased towards genes related to a specific function, etc. From a careful analysis of the sequence periodicities, we conjecture that the increase in downstream bendability is related to nucleosome positioning and/or facilitation of interaction with other factors involved in transcriptional initiation. We have also computed similar profiles from the HMM backbone probabilities using different physical scales such as stacking energies [410], nucleosome positioning [218], and propeller twist [241]. All profiles consistently show a large signal around the transcriptional start point with differences between the upstream and downstream regions. Additional results, including the periodic patterns, are discussed in [421] (see also [30] for a general treatment on how to apply additive, structural, or other scales to sequence analysis problems).

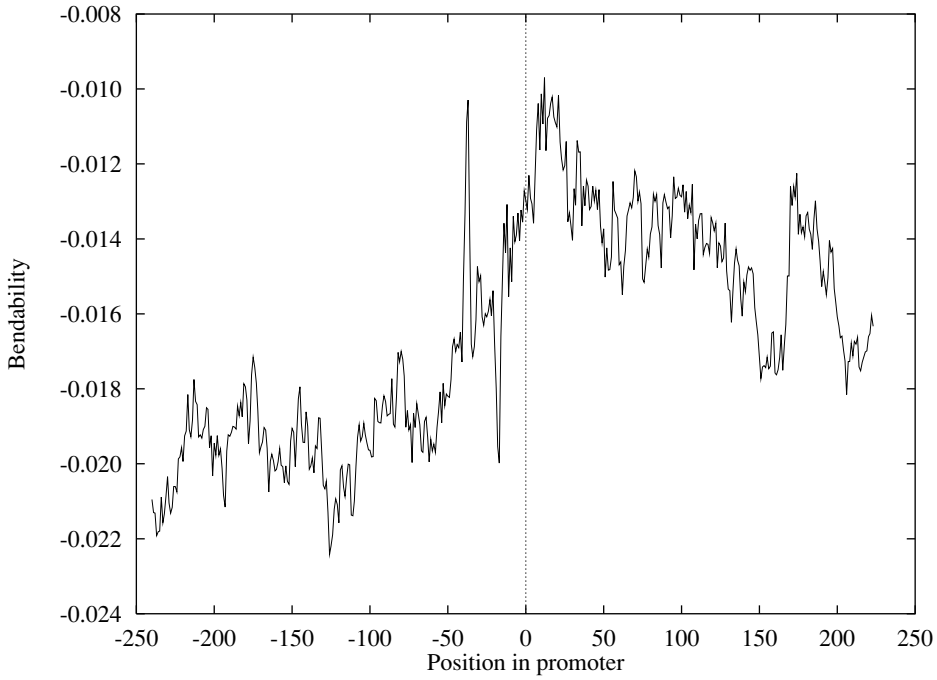

Figure 8.19: The Bendability Profile of Human Promoter Regions. The initiation site is roughly in the middle. The overall bendability is significantly increased downstream from the initiation site. This average profile was made from a multiple sequence alignment. A profile computed from the emission probabilities, instead of the actual triplet frequencies, produced a very similar pattern for the bendability.

8.3 Advantages and Limitations of HMMs

8.3.1 Advantages of HMMs

The numerous advantages of HMMs in computational molecular biology should be obvious by now. HMMs come with a solid statistical foundation and with efficient learning algorithms. They allow a consistent treatment of insertions and deletion penalties, in the form of locally learnable probabilities. Learning can take place directly from raw sequence data. Unlike conventional supervised NNs, HMMs can accommodate inputs of variable length and they do not require a teacher. They are the most flexible generalization of sequence profiles. They can be used efficiently in a number of tasks ranging from multiple alignments, to data mining and classification, to structural analysis

and pattern discovery. HMMs are also easy to combine into libraries and in modular and hierarchical ways.

8.3.2 Limitations of HMMs

In spite of their success, HMMs can suffer in particular from two weaknesses. First, they often have a large number of unstructured parameters. In the case of protein models, the architecture of figure 7.2 has a total of approximately $49N$ parameters ($40N$ emission parameters and $9N$ transition parameters). For a typical protein family, N is on the order of a few hundred, resulting immediately in models with over 10,000 free parameters. This can be a problem when only a few sequences are available in a family, not an uncommon situation in early stages of genome projects. It should be noted, however, that a typical sequence provides on the order of 2N constraints, and 25 sequences or so provide a number of examples in the same range as the number of HMM parameters.

Second, first-order HMMs are limited by their first-order Markov property: they cannot express dependencies between hidden states. Proteins fold into complex 3D shapes determining their function. Subtle long-range correlations in their polypeptide chains may exist that are not accessible to a single HMM. For instance, assume that whenever X is found at position i, it is generally followed by Y at position j, and whenever X′ is found at position i, it tends to be followed by Y′ at j. A single HMM typically has two fixed emission vectors associated with the i and j positions. Therefore, it cannot capture such correlations. Only a small fraction of distributions over the space of possible sequences can be represented by a reasonably constrained HMM.[1] It must be noted, however, that HMMs can easily capture long-range correlations that are expressed in a constant way across a family of sequences, even when such correlations are the result of 3D interactions. This is the case, for example, for two linearly distant regions in a protein family that must share the same hydropathy as a result of 3D closeness. The same hydropathy pattern will be present in all the members of the family and is likely to be reflected in the corresponding HMM emission parameters after training.

Chapters 9 to 11 can be viewed as attempts to go beyond HMMs by combining them with NNs to form hybrid models (chapter 9), by modeling the evolutionary process (chapter 10), and by enlarging the set of HMM production rules (chapter 11).

[1] Any distribution can be represented by a single exponential-size HMM, with a start state connected to different sequences of deterministic states, one for each possible alphabet sequence, with a transition probability equal to the probability of the sequence itself.

This page intentionally left blank