7 Hidden Markov models (HMMs)

In the previous chapter we discussed how to analyze a sequence with respect to one or more other sequences, through various alignment techniques. In this chapter we introduce a different model, one that can abstract our biological insight about the available sequences, and can provide findings about a new sequence S. For example, given the simple assumption that in prokaryote genomes the coding regions have more `C` and `G` nucleotides than `A` and `T` nucleotides, we would like to find a *segmentation* of S into coding and non-coding regions. We will develop this toy problem in Example 7.1.

The model introduced by this chapter is called a *hidden Markov model (HMM)*. Even though this is a probabilistic one, which thus copes with the often empirical nature of our biological understanding, we should stress that the focus of our presentation is mainly algorithmic. We will illustrate how simple techniques such as dynamic programming, and in particular the Bellman–Ford method from Section 4.2.2, can provide elegant solutions in this probabilistic context. The exercises given at the end of this chapter derive some other probabilistic models that connect back to the problem of aligning biological sequences from the previous chapter.

Let us start by recalling that a *Markov chain* is a random process generating a sequence $S = s_1 s_2 \cdots s_n$ such that the probability of *generating*, or *emitting*, each s_i is fixed beforehand and depends only on the previously generated symbols $s_1 \cdots s_{i-1}$; we will denote this probability by $\mathbb{P}(s_i \mid s_1 \cdots s_{i-1})$. An important subclass of Markov chains consists in those of *finite history*, namely those in which the probability of generating any symbol s_i depends only on the last k previous symbols, for some constant k; that is, $\mathbb{P}(s_i \mid s_1 \cdots s_{i-1}) = \mathbb{P}(s_i \mid s_{i-k} \cdots s_{i-1})$ (k is called the *order* of the Markov chain).

Consider now for concreteness a Markov chain model of order 1, that is, $\mathbb{P}(s_i \mid s_1 \cdots s_{i-1}) = \mathbb{P}(s_i \mid s_{i-1})$. In such a Markov chain, the probability of emitting s_i depends only on the previous symbol s_{i-1}. In an HMM, however, the model can have different states, in which the probability of emitting s_i depends instead *only* on the current state, which is unknown, or *hidden*, from an observer. Moreover, at each step, the model can also probabilistically choose to transition to another hidden state. In other words, if we observe S, we cannot specify the state of the HMM in which each s_i was generated. Assuming that we know the HMM which generated S, that is, we know the possible transitions and their probabilities between the hidden states, and the emission probabilities in each state, we can instead try to find the *most probable* sequence of states of the HMM which generated S. We will solve this problem in Section 7.2,

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

with the Viterbi algorithm, which is in close kinship with the Bellman–Ford algorithm for finding a shortest path in a graph. Another interesting question is that of how to compute the probability that the HMM generated S, which we solve in Section 7.3 by adapting the Viterbi algorithm into the forward and backward algorithms. Finally, in Section 7.4 we discuss how we get to know the parameters of an HMM in the first place, by showing a technique for learning the transition and emission probabilities from data.

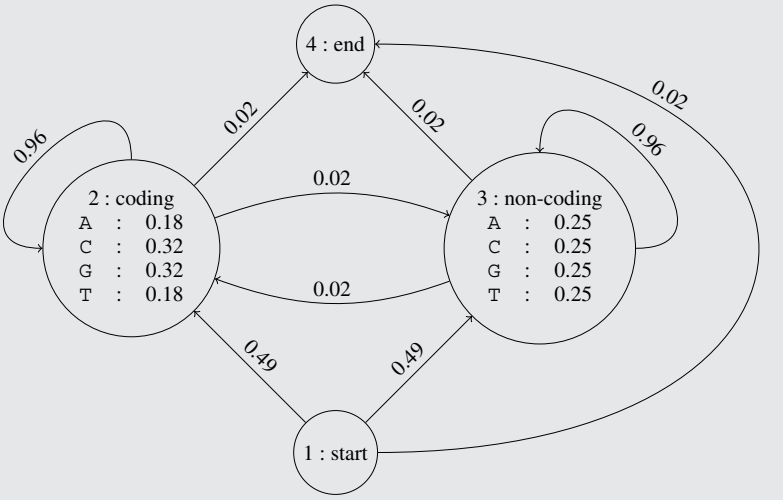
7.1 Definition and basic problems

Before giving the formal definition of an HMM, let us give a concrete example for the segmentation problem mentioned above.

Example 7.1 The segmentation problem

In prokaryote genomes, the coding and non-coding regions have some easily recognizable statistical properties on their nucleotide content. The simplest statistical property to consider is the *GC content*, since coding regions typically have more `G` and `C` nucleotides than `A` and `T`. The segmentation problem is to partition a genome into coding and non-coding regions.

We can model this problem with an HMM that has two hidden states: the coding state, in which it emits `G` and `C` nucleotides with a higher probability than it emits `A` and `T`, and the non-coding state, in which it emits any one of all four nucleotides with the same probability. The system can remain in each of these two states with a large probability, but also transition between them with a smaller one. We assume that HMMs always have a single *start* state, the initial state of the model, in which no symbol is emitted. Analogously, there is a unique *end* state, the final state of the model. We draw such an HMM below:



https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

In this HMM, the probability of transitioning between the coding and non-coding states is 0.02, and the probability of remaining in the same coding or non-coding state is 0.96. In the coding state, the probability of emitting a C or a G is 0.32, and the probability of emitting an A or a T is 0.18. In the non-coding state, all nucleotides are emitted with the same probability.

We should underline that these probabilities have to be derived in the first place from a set of training data; we present in Section 7.4 a technique for obtaining these probabilities.

For a simple example emission sequence, suppose that from the *start* state the HMM chooses, with probability 0.49, to transition to the non-coding state. There, it emits an A, and chooses, with probability 0.96, to remain in the same state. It then emits a T, and chooses with probability 0.02 to transition to the coding state. Once there, it emits a G, and onwards, twice it chooses to remain in the same state, from where it emits C and then G. Finally, it transitions back to the non-coding state, from where it emits A, and then it transitions to the *end* state, with probability 0.02. What we observe in the end is just the sequence of nucleotides $S = \texttt{ATGCGA}$, but we do not know the sequence of hidden states from which each nucleotide was generated. If we manage to find the *most probable* sequence of coding and non-coding states from which S is more likely to have been generated, then we can find a segmentation of S and solve the gene prediction problem.

Observe that an HMM can also be seen as an arc-labeled directed graph, with emission probabilities associated with its vertices, thus we will also refer to a sequence of states as a *path* in the HMM.

Formally, a *hidden Markov model (HMM)* is a tuple $(H, \Sigma, T, E, \mathbb{P})$, where $H = \{1, \ldots, |H|\}$ is the set of hidden states, Σ is the set of symbols, $T \subseteq H \times H$ is the set of transitions, $E \subseteq H \times \Sigma$ is the set of emissions, and $\mathbb{P}$ is the probability function for elements of T and E, satisfying the following conditions.

- There is a single *start state* $h_{\texttt{start}} \in H$ with no transitions $(h, h_{\texttt{start}}) \in T$, and no emissions (for this reason, $h_{\texttt{start}}$ is also called a *silent state*).
- There is a single *end state* $h_{\texttt{end}} \in H$ with no transitions $(h_{\texttt{end}}, h) \in T$, and no emissions (also $h_{\texttt{end}}$ is a *silent state*).
- Let $\mathbb{P}(h \mid h')$ denote the probability for the transition $(h', h) \in T$, and let $\mathbb{P}(c \mid h)$ denote the probability of an emission $(h, c) \in E$, for $h', h \in H$ and $c \in \Sigma$. It must hold that

$$\sum_{h \in H} \mathbb{P}(h \mid h') = 1, \text{for all } h' \in H \setminus \{h_{\texttt{end}}\},$$

and

$$\sum_{c \in \Sigma} \mathbb{P}(c \mid h) = 1, \text{for all } h \in H \setminus \{h_{\texttt{start}}, h_{\texttt{end}}\}.$$

Observe that we denote the probability of a transition *from* h' *to* h by $\mathbb{P}(h \mid h')$, rather than with a notation like $p(h', h)$ (observe the reverse order of the symbols h' and h), contrary to what the reader might have been accustomed to after Chapter 4.

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

A *path through an HMM* is a sequence P of hidden states $P = p_0p_1p_2\cdots p_np_{n+1}$, where $(p_i, p_{i+1}) \in T$, for each $i \in \{0, \ldots, n\}$. The *joint probability* of P and a sequence $S = s_1s_2\cdots s_n$, with each $s_i \in \Sigma$, is

$$\mathbb{P}(P, S) = \prod_{i=0}^{n} \mathbb{P}(p_{i+1} \mid p_i) \prod_{i=1}^{n} \mathbb{P}(s_i \mid p_i). \tag{7.1}$$

We will be mainly interested in the set $\mathcal{P}(n)$ of all paths $p_0p_1\cdots p_{n+1}$ through the HMM, of length $n+2$, such that $p_0 = h_{\texttt{start}}$, and $p_{n+1} = h_{\texttt{end}}$.

Example 7.2 In the HMM in Example 7.1, we have $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$ and $H = \{1, 2, 3, 4\}$, where we have used the names 1, start; 2, coding; 3, non-coding; and 4, end. We took $h_{\texttt{start}}$ = start and $h_{\texttt{end}}$ = end. The sets T and E, and their corresponding probabilities are shown in the tables below.

h'	h	$\mathbb{P}(h \mid h')$
start	coding	0.49
start	non-coding	0.49
start	end	0.02
coding	coding	0.96
coding	non-coding	0.02
coding	end	0.02
non-coding	non-coding	0.96
non-coding	coding	0.02
non-coding	end	0.02

(a) The set T of transitions, and their corresponding probabilities

h	c	$\mathbb{P}(c \mid h)$
coding	A	0.18
coding	C	0.32
coding	G	0.32
coding	T	0.18
non-coding	A	0.25
non-coding	C	0.25
non-coding	G	0.25
non-coding	T	0.25

(b) The set E of emissions, and their corresponding probabilities

Given the path $P \in \mathcal{P}(6)$ and the string S,

$$\begin{array}{lcccccccc} P = & 1 & 3 & 3 & 2 & 2 & 2 & 3 & 4 \\ S = & & \texttt{A} & \texttt{T} & \texttt{G} & \texttt{C} & \texttt{G} & \texttt{A} & \end{array}$$

the joint probability of P and S is

$$\begin{aligned} \mathbb{P}(P, S) = \; & (0.49 \cdot 0.96 \cdot 0.02 \cdot 0.96 \cdot 0.96 \cdot 0.02 \cdot 0.02) \cdot \\ & (0.25 \cdot 0.25 \cdot 0.32 \cdot 0.32 \cdot 0.32 \cdot 0.25). \end{aligned}$$

In the next two sections we solve the following two basic problems on HMMs, whose variations find many applications in biological sequence analysis. In the first one, Problem 7.3 below, we are given a sequence S, and we want to find the most probable sequence of states of the HMM that generated S.

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

Problem 7.3 Most probable path of a sequence in an HMM

Given an HMM M over an alphabet Σ, and a sequence $S = s_1 s_2 \cdots s_n$, with each $s_i \in \Sigma$, find the path P^* in M having the highest probability of generating S, namely

$$P^* = \arg \max_{P \in \mathcal{P}(n)} \mathbb{P}(P, S) = \arg \max_{P \in \mathcal{P}(n)} \prod_{i=0}^{n} \mathbb{P}(p_{i+1} \mid p_i) \prod_{i=1}^{n} \mathbb{P}(s_i \mid p_i). \tag{7.2}$$

In the second one, Problem 7.4 below, we are given an HMM and we need only to compute the probability of it generating S.

Problem 7.4 Generation probability of a sequence by an HMM

Given an HMM M over an alphabet Σ, and a sequence $S = s_1 s_2 \cdots s_n$, with each $s_i \in \Sigma$, compute the probability

$$\mathbb{P}(S) = \sum_{P \in \mathcal{P}(n)} \mathbb{P}(P, S) = \sum_{P \in \mathcal{P}(n)} \prod_{i=0}^{n} \mathbb{P}(p_{i+1} \mid p_i) \prod_{i=1}^{n} \mathbb{P}(s_i \mid p_i). \tag{7.3}$$

Their solutions are based on dynamic programming. The first one can be solved using the *Viterbi* algorithm. The second one is solved using the so-called *forward* algorithm, obtained from the Viterbi algorithm just by replacing the maximum operation with summation. For some variants of these problems (to be covered later), also the reverse of the forward algorithm, called the *backward* algorithm, is useful.

For ease of notation in these algorithms, we also give the following two analogs of $\mathbb{P}(P, S)$, in which we ignore the probabilities of the first and last, respectively, transitions of a path through the HMM. For a path $P = p_0 p_1 p_2 \cdots p_n$ through the HMM, we define

$$\mathbb{P}_{\texttt{prefix}}(P, S) = \prod_{i=0}^{n-1} \mathbb{P}(p_{i+1} \mid p_i) \prod_{i=1}^{n} \mathbb{P}(s_i \mid p_i).$$

Given a path $P = p_1 p_2 \cdots p_n p_{n+1}$ through the HMM, we define

$$\mathbb{P}_{\texttt{suffix}}(P, S) = \prod_{i=1}^{n} \mathbb{P}(p_{i+1} \mid p_i) \prod_{i=1}^{n} \mathbb{P}(s_i \mid p_i).$$

Finally, we must draw the attention of the reader to the numerical errors arising from the repeated multiplications and additions needed in implementing these algorithms. We touch on this issue in Exercises 7.5 and 7.6. However, more advanced techniques for addressing this problem are beyond the scope of this book, and we assume here that all arithmetic operations are exact, and take $O(1)$ time.

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

7.2 The Viterbi algorithm

The Viterbi algorithm solves Problem 7.3 and is very similar to the Bellman–Ford algorithm, which was studied in Section 4.2.2. Recall that in the Bellman–Ford algorithm we dealt with the possible cycles of the graph by adding another parameter to the recurrence, namely the number of arcs in a solution path. The additional recursion parameter exploited by the Viterbi algorithm for "breaking" the possible cycles on the HMM is analogous, namely the length of the prefix of the sequence S generated so far.

Accordingly, for every $i \in \{1, \ldots, n\}$ and every $h \in \{1, \ldots, |H|\}$, define

$$v(i,h) = \max\left\{\mathbb{P}_{\texttt{prefix}}(P, s_1 \cdots s_i) \mid P = h_{\texttt{start}} p_1 \cdots p_{i-1} h\right\}$$

as the largest probability of a path starting in state $h_{\texttt{start}}$ and ending in state h, given that the HMM generated the prefix $s_1 \cdots s_i$ of S (symbol s_i being emitted by state h). Notice the similarity with the dynamic programming table (4.2) in the Bellman–Ford algorithm.

We can easily derive the following recurrence relation for $v(i,h)$:

$$\begin{aligned} v(i,h) &= \max\{\mathbb{P}_{\texttt{prefix}}(h_{\texttt{start}} p_1 \cdots p_{i-1} h', s_1 \cdots s_{i-1}) \mathbb{P}(h \mid h') \mathbb{P}(s_i \mid h) \mid (h',h) \in T\} \\ &= \mathbb{P}(s_i \mid h) \max\{v(i-1,h') \mathbb{P}(h \mid h') \mid (h',h) \in T\}, \end{aligned} \tag{7.4}$$

where we take by convention $v(0, h_{\texttt{start}}) = 1$ and $v(0,h) = 0$ for all $h \neq h_{\texttt{start}}$. Indeed, $v(i,h)$ equals the largest probability of getting to a predecessor h' of h, having generated the prefix sequence $s_1 \cdots s_{i-1}$, multiplied by the probability of the transition (h',h) and by $\mathbb{P}(s_i \mid h)$. Observe that $\mathbb{P}(s_i \mid h)$ depends only on h and s_i, and not on the predecessors of h; this is the reason why we have written $\mathbb{P}(s_i \mid h)$ outside of the max notation in (7.4).

The largest probability of a path for the entire string S (that is, the value maximizing (7.2)) is the largest probability of getting to a predecessor h' of $h_{\texttt{end}}$, having generated the entire sequence $S = s_1 \cdots s_n$ (symbol s_n being emitted by state h'), multiplied by the probability of the final transition $(h', h_{\texttt{end}})$. Expressed in terms of v,

$$\max_{P \in \mathcal{P}(n)} \mathbb{P}(P,S) = \max\left\{v(n,h') \mathbb{P}(h_{\texttt{end}} \mid h') \mid (h', h_{\texttt{end}}) \in T\right\}. \tag{7.5}$$

The values $v(\cdot,\cdot)$ can be computed by filling a table $V[0..n, 1..|H|]$ row-by-row in $O(n|T|)$ time. The most probable path can be traced back in the standard dynamic programming manner, by checking which predecessor h' of $h_{\texttt{end}}$ maximizes (7.5) and then, iteratively, which predecessor h' of the current state h maximizes (7.4). Figure 7.1 illustrates the dynamic programming recurrence.

7.3 The forward and backward algorithms

The *forward* algorithm solves Problem 7.4 and is very similar to the Viterbi algorithm. The only difference is the fact that the maximum operation is replaced by

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

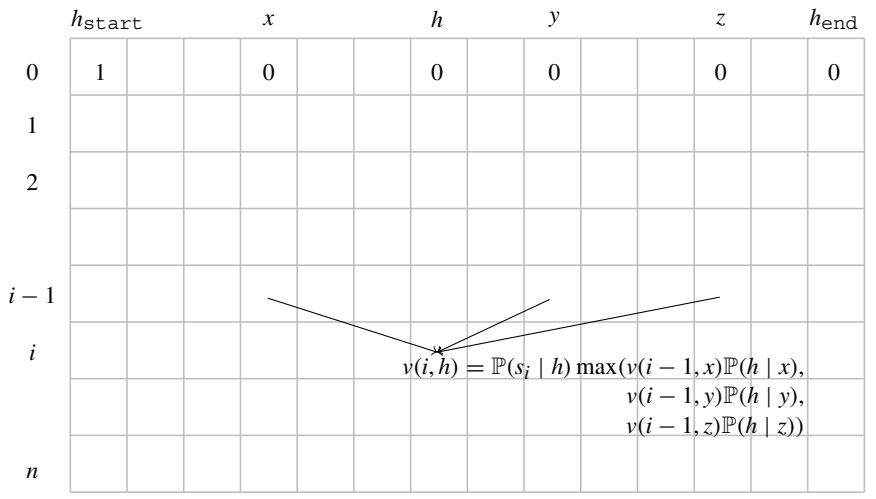


Figure 7.1 The idea behind the Viterbi algorithm, assuming that the predecessors of state h are the states x, y, and z. Notice the similarity between this row-by-row computation of $V_{0..n,1..|H|}$ and the interpretation of the Bellman–Ford algorithm as finding the shortest path in a DAG made up of layered copies of the input graph, illustrated in Figure 4.5.

summation, since we now have to compute $\mathbb{P}(S) = \sum_{P \in \mathcal{P}(n)} \mathbb{P}(P, S)$, instead of $\arg\max_{P \in \mathcal{P}(n)} \mathbb{P}(P, S)$.

More precisely, for every $i \in \{1, \ldots, n\}$ and every $h \in \{1, \ldots, |H|\}$, we define

$$f(i,h) = \sum_{P = h_{\texttt{start}} p_1 \cdots p_{i-1} h} \mathbb{P}_{\texttt{prefix}}(P, s_1 \cdots s_i)$$

as the probability that the HMM, having started in state $h_{\texttt{start}}$, is generating symbol s_i of S while in state h. We compute each $f(i,h)$ as

$$f(i,h) = \mathbb{P}(s_i \mid h) \sum_{(h',h) \in T} f(i-1,h')\mathbb{P}(h \mid h'), \tag{7.6}$$

where we take by convention $f(0, h_{\texttt{start}}) = 1$ and $f(0,h) = 0$ for all $h \neq h_{\texttt{start}}$. As before, the values $f(\cdot,\cdot)$ can be computed by filling in a table $F[0..n, 1..|H|]$ row-by-row in total time $O(n|T|)$. The solution of Problem 7.4 is obtained as

$$\mathbb{P}(S) = \sum_{(h', h_{\texttt{end}}) \in T} f(n,h')\mathbb{P}(h_{\texttt{end}} \mid h'). \tag{7.7}$$

We can ask the converse question, what is the probability that the HMM generates symbol s_i of S in state h, and it finally goes to state $h_{\texttt{end}}$ after generating the symbols $s_{i+1} \cdots s_n$ of S? Let us call this probability $b(i,h)$, where b stands for *backward*:

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

$$b(i,h) = \sum_{P=hp_{i+1}\cdots p_n h_{\texttt{end}}} \mathbb{P}_{\texttt{suffix}}(P, s_i s_{i+1} \cdots s_n).$$

Analogously, for every $i \in \{1,\ldots,n\}$ and every $h \in \{1,\ldots,|H|\}$, we can compute each $b(i,h)$ as

$$b(i,h) = \mathbb{P}(s_i \mid h) \sum_{(h,h')\in T} b(i+1,h')\mathbb{P}(h' \mid h), \tag{7.8}$$

where we take by convention $b(n+1,h_{\texttt{end}}) = 1$ and $b(n+1,h) = 0$ for all $h \neq h_{\texttt{end}}$. The values $b(\cdot,\cdot)$ can also be computed by filling in a table $B[1..n+1, 1..|H|]$ row-by-row in time $O(n|T|)$. This can be done row-by-row, backwards from the nth row up to the first row. Observe that this *backward* algorithm also solves Problem 7.4, since it holds that

$$\mathbb{P}(S) = \sum_{(h_{\texttt{start}},h)\in T} b(1,h)\mathbb{P}(h \mid h_{\texttt{start}}). \tag{7.9}$$

Moreover, observe that the backward algorithm can be seen as identical to the forward algorithm applied on the reverse sequence, and on the HMM in which all transitions have been reversed, and $h_{\texttt{start}}$ and $h_{\texttt{end}}$ have exchanged meaning.

Having computed $f(i,h)$ and $b(i,h)$ for every $i \in \{1,\ldots,n\}$ and $h \in \{1,\ldots,|H|\}$, we can answer more elaborate questions; for example, what is the probability that the HMM emits the ith symbol of the given string S while being in a given state h? Using the tables f and b, this probability can be obtained as

$$\sum_{P=p_0p_1\cdots p_n p_{n+1}\in\mathcal{P}(n)\,:\,p_i=h} \mathbb{P}(P \mid S) = \sum_{(h',h)\in T} f(i-1,h')\mathbb{P}(h \mid h')b(i,h). \tag{7.10}$$

In the next section we will see another example where both tables $F[0..n, 1..|H|]$ and $B[1..n+1, 1..|H|]$ are useful.

7.4 Estimating HMM parameters

Where do we get the transition and emission probabilities from in the first place? Let us come back to our example of segmenting a DNA sequence into coding and non-coding regions based on the GC content.

Assume first that we have *labeled training data*, that is, a collection of DNA sequences in which each position has been labeled as coding or non-coding. In this case, it suffices to count how many times each transition and emission is used in the training data. Let $\texttt{TC}(h',h)$ denote the number of times transition $(h',h) \in T$ takes place (*transition count*), and let $\texttt{EC}(h,c)$ denote the number of times emission $(h,c) \in E$ takes place (*emission count*) in the training data. Then, we can set

$$\mathbb{P}(h \mid h') = \frac{\texttt{TC}(h',h)}{\sum_{h''\in H}\texttt{TC}(h',h'')} \quad \text{and} \quad \mathbb{P}(c \mid h) = \frac{\texttt{EC}(h,c)}{\sum_{c'\in\Sigma}\texttt{EC}(h,c')}. \tag{7.11}$$

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

Insight 7.1 HMM parameters and pseudocounts

Consider the HMM in Example 7.2, but with unknown transition and emission probabilities, and the following labeled sequence as training data:

labels	NNNNNNNNNNCCCCCCCCCCCCCNNNNNNNNNNNNCCCCCCCCCC
S	ATTACATCTAGCGCGAGCGCGCGTATATCTATTATCGATGCGCGCG

where the N label indicates a non-coding symbol of S, and the C label indicates a coding symbol of S. Using relations (7.11), we get for example $\mathbb{P}(\texttt{A} \mid \text{non-coding}) = 8/23$, and $\mathbb{P}(\text{coding} \mid \text{non-coding}) = 2/22$.

Moreover, we have $\mathbb{P}(\texttt{G} \mid \text{non-coding}) = 0/23 = 0$, which prevents *any* emission of G in the non-coding state. This situation is due to there being too few training samples, since we do expect to find symbol G in non-coding regions. To avoid such null probabilities, a common simple solution is to use so-called *pseudocounts*, by adding a constant b to every counter $\texttt{TC}(h', h)$ and $\texttt{EC}(h, c)$. For example, using $b = 1$, we obtain $\mathbb{P}(\text{coding} \mid \text{non-coding}) = (2+1)/(22+3)$, $\mathbb{P}(\texttt{A} \mid \text{non-coding}) = (8+1)/(23+4)$, and $\mathbb{P}(\texttt{G} \mid \text{non-coding}) = 1/(23+4)$.

When the sequences in the training dataset are not labeled (*unlabeled training data*), we cannot distinguish which paths the training sequences correspond to. One way to proceed is to use the following iterative approach, called *Viterbi training*. Assume there are some prior estimates for the transition and emission probabilities. Use the Viterbi algorithm to find the most likely paths for the training sequences, and then use these paths as new labeled training data to improve the prior estimates. Stop when the parameters of the HMM no longer change significantly. See Insight 7.1.

A less greedy approach, called *Baum–Welch training*, is to iteratively estimate each $\texttt{TC}(h', h)$ and $\texttt{EC}(h, c)$ by taking into account all the paths through the HMM. For instance, we want to estimate $\texttt{EC}(h, c)$ as the expected value of emitting a symbol $c \in \Sigma$ while in a state $h \in H$, over all paths in the HMM that emit the training string $S = s_1 \cdots s_n$.

Recall from the previous section that relation (7.10) allows us to compute, using the forward and backward algorithms, the probability

$$\sum_{P = p_0 p_1 \cdots p_n p_{n+1} \in \mathcal{P}(n) \,:\, p_i = h} \mathbb{P}(P \mid S)$$

that the ith symbol of S is generated by the HMM while in state h. Moreover, the sequence S itself has a probability $\mathbb{P}(S)$ of being generated by the HMM, where $\mathbb{P}(S)$ is the solution to Problem 7.4, computable with either the forward or the backward algorithm (recall relations (7.7) and (7.9)).

Therefore, given the sequence S, the expected value of being in a state $h \in H$, over all paths in the HMM that emit the training string $S = s_1 \cdots s_n$, is

$$\frac{\displaystyle\sum_{P = p_0 p_1 \cdots p_n p_{n+1} \in \mathcal{P}(n) \,:\, p_i = h} \mathbb{P}(P \mid S)}{\mathbb{P}(S)}. \tag{7.12}$$

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

Finally, we can obtain $\mathtt{EC}(h,c)$ by summing the above probability (7.12) over positions $1 \le i \le n$ of S such that $s_i = c$:

$$\begin{aligned}\mathtt{EC}(h,c) &= \sum_{i\,:\,s_i=c} \frac{\sum_{P=p_0p_1\cdots p_np_{n+1}\in\mathcal{P}(n)\,:\,p_i=h} \mathbb{P}(P \mid S)}{\mathbb{P}(S)} \\ &= \sum_{i\,:\,s_i=c} \frac{\sum_{(h',h)\in T} f(i-1,h')\mathbb{P}(h \mid h')b(i,h)}{\sum_{(h',h_{\mathtt{end}})\in T} f(n,h')\mathbb{P}(h_{\mathtt{end}} \mid h')}. \end{aligned} \tag{7.13}$$

One can derive an expected case formula for $\mathtt{TC}(h',h)$ analogously.

7.5 Literature

The Viterbi algorithm (Viterbi 1967) and its multiple variations have been vital in the development of genome analysis. Gene discovery was used here as a toy example, but also in practice very similar, yet more tailored, HMM techniques are in use (Mathé *et al.* 2002). Another very popular application of HMMs is the representation of protein families with *profile HMMs* (Krogh *et al.* 1994; Eddy 2011): aligning a new sequence against all such profiles in a database gives a way to categorize the new sequence to the closest family. We will see in the exercises that the Viterbi recurrences are very easy to generalize to such profile HMMs. A more theoretical application of HMMs is to extend them for modeling pair-wise alignments. Such HMMs have match states emitting a symbol from two input sequences simultaneously, and symmetric insertion and deletion states emitting only from one input sequence. These *pair HMMs* (Durbin *et al.* 1998, Chapter 4) offer a fully probabilistic interpretation of alignments, and give a rigorous way to do probabilistic sampling of alignments and to define suboptimal alignments. We will elaborate this connection in one of the assignments below.

We covered only the fundamental algorithmic properties of HMMs. As was already noticeable in Section 7.4 when deriving the Baum–Welch formulas (Baum 1972) for the case of unlabeled training data, the derivations pertaining to HMMs are tightly connected to the probabilistic interpretation, and iterative machine-learning approaches are required for further development of the models. These methods are beyond the scope of this book, and we recommend the excellent textbook by Durbin *et al.* (1998) for an in-depth understanding; our treatment here is evidently highly influenced by this classic book. For a more mathematical introduction to HMMs, we also recommend the book on pattern discovery by Parida (2007).

Finally, we should mention that, if the sequence is compressible, then it is possible to achieve faster implementations of the Viterbi, forward-backward, and Baum–Welch algorithms (Lifshits *et al.* 2009).

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

Exercises

7.1 Argue that $O(|H|)$ working space suffices in Problem 7.3 to compute the final numerical value (not the actual solution path) with the Viterbi algorithm, and in Problem 7.4 with the forward and backward algorithms.

7.2 To trace back the solution of Viterbi algorithm, the naive solution requires $O(n|H|)$ working space. Show that this can be improved to $O(\sqrt{n}|H|)$ by sampling every $\sqrt{n}$th position in S, without asymptotically affecting the running time of the traceback.

7.3 The standard algorithm for the multiplication of two matrices of m rows by m columns takes time $O(m^3)$. More sophisticated approaches achieve time $O(m^\omega)$ with $\omega < 3$ (the current record is $\omega \leq 2.38$ and $\omega = 2$ is the best one can hope for). Suppose that we have to compute many instances of the forward and backward algorithms on different sequences, but on the same HMM. Show how to use fast matrix multiplication to improve the running time of such a computation.

7.4 Given an HMM and a sequence $S = s_n \cdots s_n$, derive an $O(n|H|)$ time algorithm to compute, for all $h \in H$,

$$\operatorname*{argmax}_i \{\mathbb{P}(P, s_1 \cdots s_n) \mid P = p_0 p_1 \cdots p_n p_{n+1} \in \mathcal{P}(n) \text{ and } p_i = h\}. \tag{7.14}$$

7.5 Multiplication is the source of numerical errors in HMM algorithms, when the numbers become too large to fit into a computer word. Show how the Viterbi algorithm can be implemented using a sum of logarithms to avoid these numerical problems.

7.6 For the forward and backward algorithms the sum of logarithms conversion is not enough for numerical stability. Browse the literature to find a solution for this problem.

7.7 The flexibility of choosing the states, transitions, emissions, and their probabilities makes HMMs a powerful modeling device. So far we have used a *zeroth-order Markov model* for emission probabilities (probabilities depended only on the state, not on the sequence context). We could just as well use *first-order Markov chains* or, more generally, *kth-order Markov chains*, in which the probability depends on the state and on the last k symbols preceding the current one: $\mathbb{P}(s_i \mid s_{i-k} \cdots s_{i-1}) = \mathbb{P}(s_i \mid s_1 \cdots s_{i-1})$.

Notice that the states of the HMM are independent, in the sense that each state can choose a Markov chain of a different order from that of the Markov chain for its emission probabilities. In addition to the use of different-order Markov chains, we could adjust how many symbols are emitted in each state. Use these considerations to design a realistic HMM for *eukaryote gene prediction*. Try to take into account intron/exon boundary dinucleotides, codon adaptation, and other known features of eukaryote genes. Consider also how you can train the HMM.

7.8 *Profile HMMs* are an extension of HMMs to the problem of aligning a sequence with an existing multiple alignment (profile). Consider for example a multiple alignment of a protein family:

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

```
AVLSLSKTTNNVSPA
AV-SLSK-TANVSPA
A-LSLSK-TANV-PA
A-LSSSK-TNNV-PA
AS-SSSK-TNNV-PA
AVLSLSKTTANV-PA
```

We considered the problem of aligning a sequence A against a profile in the context of progressive multiple alignment in Section 6.6.4, and the idea was to consider the multiple alignment as a sequence of columns and apply normal pair-wise alignment with proper extensions of substitution and indel scores. Consider $A =$ `AVTLSLSTAANVSPA` aligned to our example profile above, for example, as follows:

```
AVTLSLS- -TAANVSPA
AV-LSLSKTTN-NVSPA
AV- -SLSK-TA-NVSPA
A- -LSLSK-TA-NV-PA
A- -LSSSK-TN-NV-PA
AS- -SSSK-TN-NV-PA
AV-LSLSKTTA-NV-PA
```

Here we have added two gaps to the sequence and two gap columns to the profile following the "once a gap, always a gap" principle.

Profile HMMs are created using *inhomogeneous* Markov chains, such that each of the columns will form separate match, insertion, and deletion states, and transitions go from left to right, as illustrated in Figure 7.2. Match and deletion states emit the columns of the profile, so they do not contain self-loops. Insertion states emit symbols from the input sequence, so they contain self-loops to allow any number of symbols emitted between states that emit also columns of the profile.

Since the resulting HMM is reading only one sequence, the Viterbi, forward, and backward algorithms are almost identical to the ones we have studied so far. The only

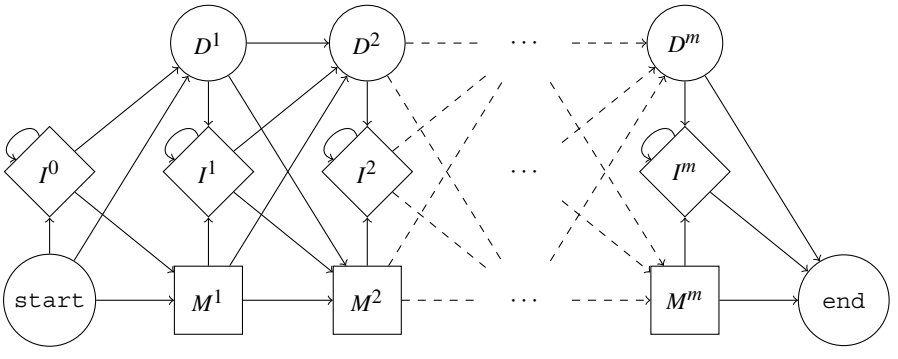


Figure 7.2 Profile HMM illustration without showing the transition and emission probabilities.

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press

difference is that deletion states are *silent* with respect to the input string, since they do not emit any symbol.

(a) Modify the Viterbi recurrences to handle both emitting and silent states.
(b) Derive the Viterbi recurrences specific to profile HMMs.

7.9 Derive a local alignment version of profile HMMs.

7.10 *Pair HMMs* are a variant of HMMs emitting two sequences, such that a path through the HMM can be interpreted as an alignment of the input sequences. Such pair HMMs have a *match* state emitting a symbol from both sequences simultaneously, and symmetric *insertion* and *deletion* states to emit only from one input sequence.

(a) Fix a definition for pair HMMs and derive the corresponding Viterbi, forward, and backward recurrences. *Hint*. The result should look very similar to Gotoh's algorithm for global alignment with affine gap costs from Section 6.4.4.
(b) Apply a derivation for pair HMMs similar to the ones we used in obtaining relation (7.13), in order to define the probability of a_i aligning to b_j over all alignments of $A = a_1 \cdots a_m$ and $B = b_1 \cdots b_n$.
(c) Let p_{ij} denote the probability derived above to align a_i to b_j. We say that the *most robust alignment* of A and B is the alignment maximizing the sum of values p_{ij} over i, j such that the $a_i \rightarrow b_j$ substitution is part of the alignment. Derive a dynamic programming algorithm to compute this most robust alignment.

https://doi.org/10.1017/CBO9781139940023.009 Published online by Cambridge University Press