# Chapter 1. Probabilistic Models for Biological Sequences

Ever since the discovery of the genetic code, linguistic metaphors have been used when describing biological sequences. Who has not heard the human genome being referred to as the 'book of life'? This chapter shows that these are not mere metaphors used for their evocative power but that the relation really goes deeper. Techniques from the fields of linguistics and formal language theory can be applied and adapted for the description and analysis of biological sequences.[17]
As a first example, consider the problem of describing a common pattern in an ensemble of biological sequences. Here, you can think of signals for splice sites, common regulatory elements, or conserved domains in protein families. A simple approach to such a problem would be to build a string-based description of a pattern. Formal language theory has come up with very efficient algorithms for searching for such patterns. However, when applied to biological sequences these methods often fail. This is because biological 'spelling' is far sloppier than English spelling; *e.g.,* proteins with the same function from two related organisms can have less than 30% identical amino acids.[13] Sloppiness is caused by the random processes inherent in biological evolution. To deal with this uncertainty typical for biological sequences, *probabilistic* methods are needed. In this chapter, we gradually go from a simple deterministic description using a consensus sequence to a complex probabilistic model in terms of so-called hidden Markov models. Along the way, you will also learn something about the algorithms behind these models and a wide range of applications will be shown.
As a second example, we will see that hidden Markov models also capture certain grammatical features that are essential for successful (eukaryotic) gene finding algorithms. Most of the basic concepts of probability theory, information theory, and statistical inference used in the current chapter are summarized in the Appendix (section 1.9).

**Note:** Many of the examples deal with nucleotide sequences and, therefore, an alphabet of four symbols. All models can just as well be applied to amino acid sequences or sequences over other alphabets.

## 1.1. Consensus and Regular Expressions

A *site* is a short sequence containing some signal, often being recognized by an enzyme. Examples of nucleotide sequence sites are intron splice sites, transcription start sites, and transcription factor binding sites. Instances of a single site are characterized by slight variations on most sequence positions. How could we give a compact description of such a site that retains most information?

A simple description of a site in an ensemble of sequences is given by a *consensus* sequence. Consider the following five related nucleotide sequences:

```
A C A A T G
T C A A T C
A C A A G C
A G A A T C
A C C A T C
```

| | |
|---|---|
| M | A/C |
| R | A/G |
| W | A/T |
| S | C/G |
| Y | C/T |
| K | G/T |
| B | C/G/T |
| D | A/G/T |
| H | A/C/T |
| V | A/C/G |
| N | A/C/G/T |

**Table 1. Consensus symbols for nucleotides (IUPAC).**

The consensus sequence is, in general, found by choosing at each sequence position the most prominent nucleotide (or combination of nucleotides, Table 1). In our example, the consensus sequence obtained by a majority vote for each sequence position is

```
A  C  A  A  T  C.
```

Using the IUPAC code, the consensus sequence could also have been described as

```
W  S  M  A  K  S,
```

although there is no consensus on how to do this: on each sequence position, there clearly is one dominant nucleotide and the other nucleotide might be an exception that one does not want to include in the consensus sequence. The main disadvantage of a consensus sequence is that it only gives the description of an average site and is rather limited in capturing its true specificities.

Related to consensus sequences are descriptions using regular expressions. A regular expression for this set of short sequences is `[AT][CG][AC]A[TG][GC]`. This succinctly describes that in all sequences the first position is either `A` or `T`, the second `C` or `G`, etc.

A problem with a description of a set of sequences in these terms is that it does not distinguish between a highly unlikely sequence such as

```
T  G  C  A  G  G
```

and the consensus sequence obtained by majority vote.

## 1.2. Weight Matrices

A more complete description of a site in a set of sequences can be given by counting the number of times a particular nucleotide occurs at a certain position and summarizing this in a *weight matrix*. For our example, this leads to

$$\begin{array}{c} \\ A \\ C \\ G \\ T \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 4 & 0 & 4 & 5 & 0 & 0 \\ 0 & 4 & 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 4 & 0 \end{pmatrix} \end{array},$$

and the more common formulation in terms of (estimated) probabilities:

$$W = \begin{array}{c} \\ A \\ C \\ G \\ T \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0.8 & 0.0 & 0.8 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.8 & 0.2 & 0.0 & 0.0 & 0.8 \\ 0.0 & 0.2 & 0.0 & 0.0 & 0.2 & 0.2 \\ 0.2 & 0.0 & 0.0 & 0.0 & 0.8 & 0.0 \end{pmatrix} \end{array},$$

where each entry $w_{ij}$ gives the (estimated) probability of finding nucleotide *i* at position *j*, also written as $P_j(i)$. For a set of sequences of length *N* consisting of symbols from an alphabet of size *M*, the weight matrix is a $M \times N$ matrix, the columns of which sum to one. A weight matrix is the correct model if we assume that sequence positions are independent (1.17). The probability of sequence $x = x_1 x_2 ... x_N$ given weight matrix *W* is then:

$$P(x_1x_2...x_N \mid W) = \prod_{i=1}^{N} w_{x_i,i} = \prod_{i=1}^{N} P_i(x_i \mid W)\,. \qquad \textbf{(1.1)}$$

In our example, the probability of the consensus sequence is

$$P(\texttt{ACAATC} \mid W) = P_1(\texttt{A})P_2(\texttt{C})P_3(\texttt{A})P_4(\texttt{A})P_5(\texttt{T})P_6(\texttt{C}) = 0.8\times0.8\times0.8\times1\times0.8\times0.8 = 0.33\,.$$

It is the largest probability of any sequence of length six, because each position contains the most probable residue. The probability of the second sequence described above is

$$P(\texttt{TGCAGG} \mid W) = P_1(\texttt{T})P_2(\texttt{G})P_3(\texttt{C})P_4(\texttt{A})P_5(\texttt{G})P_6(\texttt{G}) = 0.2\times0.2\times0.2\times1\times0.2\times0.2 = 0.00032$$

which is indeed highly unlikely. Let us consider another sequence that closely resembles the consensus sequence:

$$P(\texttt{CCAATC} \mid W) = P_1(\texttt{C})P_2(\texttt{C})P_3(\texttt{A})P_4(\texttt{A})P_5(\texttt{T})P_6(\texttt{C}) = 0\times0.8\times0.8\times1\times0.8\times0.8 = 0\,.$$

This probability equals zero since the cytosine (C) observed on the first position does not occur in our tiny set of sequences. The outcome is rather a side effect though of the fact that we do not have enough data. A common solution is to add some fake counts (*pseudocounts*) to the observed counts. The simplest alternative is to add one to each observed number of counts. The weight matrix then becomes:

$$\begin{array}{c} A \\ C \\ G \\ T \end{array}\begin{pmatrix} 5 & 1 & 5 & 6 & 1 & 1 \\ 1 & 5 & 2 & 1 & 1 & 5 \\ 1 & 2 & 1 & 1 & 2 & 2 \\ 2 & 1 & 1 & 1 & 5 & 1 \end{pmatrix} \qquad W' = \begin{array}{c} A \\ C \\ G \\ T \end{array}\begin{pmatrix} 0.56 & 0.11 & 0.56 & 0.67 & 0.11 & 0.11 \\ 0.11 & 0.56 & 0.22 & 0.11 & 0.11 & 0.56 \\ 0.11 & 0.22 & 0.11 & 0.11 & 0.22 & 0.22 \\ 0.22 & 0.11 & 0.11 & 0.11 & 0.56 & 0.11 \end{pmatrix}.$$

The estimated probabilities of the three sequences described above can consequently be updated:

$$\begin{aligned} P(\texttt{ACAATC} \mid W') &= P_1(\texttt{A})P_2(\texttt{C})P_3(\texttt{A})P_4(\texttt{A})P_5(\texttt{T})P_6(\texttt{C}) = 0.56^5\times0.67 = 0.037 \\ P(\texttt{TGCAGG} \mid W') &= P_1(\texttt{T})P_2(\texttt{G})P_3(\texttt{C})P_4(\texttt{A})P_5(\texttt{G})P_6(\texttt{G}) = 0.22^5\times0.67 = 0.00034 \\ P(\texttt{CCAATC} \mid W') &= P_1(\texttt{C})P_2(\texttt{C})P_3(\texttt{A})P_4(\texttt{A})P_5(\texttt{T})P_6(\texttt{C}) = 0.11\times0.56^4\times0.67 = 0.0072\,. \end{aligned}$$

These outcomes correspond well to what we would expect given our small ensemble of sequences.

Suppose we want to test whether a new sequence $x$ is more likely to be a site corresponding to weight matrix $W$ or a nonsite from the *background* model (also called *null* model) $R$. We assume the background model to be independent of specific positions, that is, $P_j(i \mid R)$ does not depend on $j$ and is, therefore, denoted as $P(i \mid R)$. The *odds ratio*, well-known to those placing bets, compares the two models for sequence $x$:

$$\frac{P(x_1x_2...x_N \mid W)}{P(x_1x_2...x_N \mid R)} = \frac{\prod_{i=1}^{N} P_i(x_i \mid W)}{\prod_{i=1}^{N} P(x_i \mid R)}\,.$$

An odds ratio of at least one[a] means that the sequence is more likely to be a site as described by model *W*, whereas an odds ratio of less than one pleads in favour of the background model.

For numerical reasons and to enforce symmetry between denominator and numerator, it is more convenient to calculate the *log-odds ratio*:

$$\log_2\left(\frac{P(x_1x_2...x_N\,|\,W)}{P(x_1x_2...x_N\,|\,R)}\right)=\log_2\left(\frac{\prod_{i=1}^{N}P_i(x_i\,|\,W)}{\prod_{i=1}^{N}P(x_i\,|\,R)}\right)=\sum_{i=1}^{N}\log_2\left(\frac{P_i(x_i\,|\,W)}{P(x_i\,|\,R)}\right), \qquad \textbf{(1.2)}$$

which is the sum of the individual log-likelihood ratios. When a sequence fits the model *W* well, the log-odds ratio is positive. When it fits the background model better, the score is negative. The higher the score for a candidate site, the more likely it is to be real site.

To test for sites it is convenient to transform a weight matrix into a *position-specific scoring matrix* (PSSM), the entries of which are the log-likelihood ratios. This is done by dividing each element of the weight matrix by the probability according to the background model and taking the logarithm.
If we assume that under the background model all nucleotides are equally likely ($P(\texttt{A}|R) = P(\texttt{C}|R) = P(\texttt{G}|R) = P(\texttt{T}|R) = 0.25$), the scoring matrix corresponding to *W’* is

$$\begin{array}{c} A \\ C \\ G \\ T \end{array}\begin{pmatrix} 1.16 & -1.17 & 1.16 & 1.42 & -1.17 & -1.17 \\ -1.17 & 1.16 & -0.17 & -1.17 & -1.17 & 1.16 \\ -1.17 & -0.17 & -1.17 & -1.17 & -0.17 & -0.17 \\ -0.17 & -1.17 & -1.17 & -1.17 & 1.16 & -1.17 \end{pmatrix},$$

where, *e.g.*, the first entry equals $\log_2(0.56/0.25)=1.16$. The log-odds ratio (1.2) can now be used to score the three sequences used above by adding the corresponding entries of the scoring matrix

$$\begin{aligned}\text{log-odds}(\texttt{ACAATC}) &= 1.16+1.16+1.16+1.42+1.16+1.16 = 7.22 \\ \text{log-odds}(\texttt{TGCAGG}) &= -0.17-0.17-0.17+1.42-0.17-0.17 = 0.57 \quad . \\ \text{log-odds}(\texttt{CCAATC}) &= -1.17+1.16+1.16+1.42+1.16+1.16 = 4.89\end{aligned}$$

The log-odds score accentuates the difference between the unlikely second sequence and the other two sequences, although also the second sequence has a positive score. This illustrates the importance of choosing a suitable threshold for deciding whether a sequence is a site or not. An example of an even more unlikely sequence is

$$\text{log-odds}(\texttt{CTTGAT}) = 6\times -1.17 = -7.02$$

which indeed has a negative score, as one would expect.

### 1.2.1. Sequence logos

A visual representation of a weight matrix can be useful for the interpretation of a large set of sequences. A *sequence logo* displays the degree of consensus at each position as the total height of a

[a] A different cutoff can also be chosen, see the discussion of Bayes’ decision rule in section 1.3.3.

stack of nucleotides, along with the frequencies of nucleotides at that position, as the relative heights of letters in the stack (Figure 1). The total height of the stack at position *j* is calculated as follows:[b]

$$H(j) = 2 - \sum_{i \in \{A,C,G,T\}} P_j(i) \log_2 \frac{1}{P_j(i)}. \tag{1.3}$$

This function has a maximum of two if only one symbol occurs at position *j* (full consensus) and a minimum of zero if all four nucleotides are equally likely (no consensus). Formally, this is the relative entropy (1.23) of *P* and the uniform background distribution at position *j*. The height of each nucleotide symbol in a stack is proportional to the relative entropy and the frequency of the nucleotide given by $H(j) \times P_j(i)$. Symbols are stacked in order of increasing frequency starting from the bottom.

As an example, we compute the relative entropy at the first position in weight matrix *W*:[c]

$$H(1) = 2 - 0.8 \log_2\left(\frac{1}{0.8}\right) - 2 \times 0 \log_2\left(\frac{1}{0}\right) - 0.2 \log_2\left(\frac{1}{0.2}\right) = 1.28.$$

The first position is indeed relatively well conserved with an `A` in four sequences and a `T` in the remaining sequence. Figure 1 depicts the sequence logo corresponding to weight matrix *W* and an alternative representation called a Hinton diagram.[5]

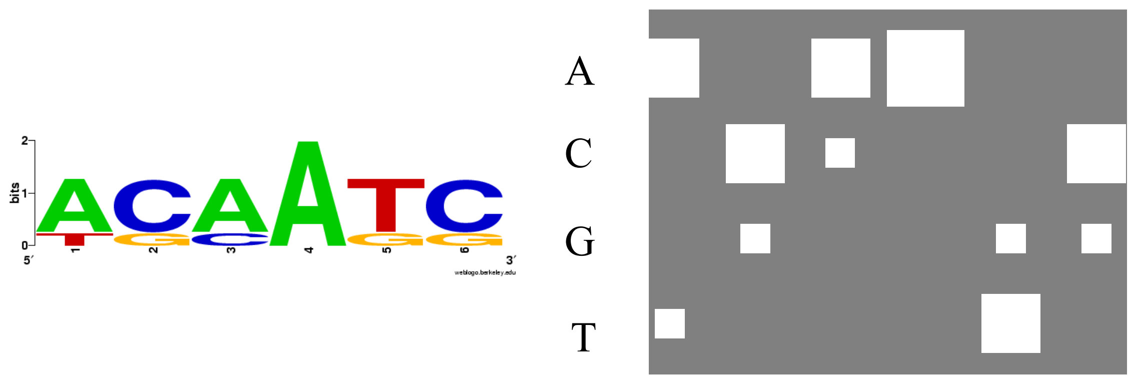


**Figure 1.** Left: Sequence logo[d] of weight matrix *W*. One can easily read of the consensus sequence `ACAATC`. Right: Hinton diagram of weight matrix *W*. The area of each square is proportional to the value in the weight matrix.

## 1.3. Dependencies and Markov Chains

When using weight matrices one assumes that a symbol in any position is independent of symbols in any other position. In the English language, for example, this is not the case. An "a" followed by an

[b] You might want to look up the definition of (relative) entropy in Section 1.9.3 before reading on.

[c] $0 \times \log_2(1/0) = 0$.

[d] Made with `http://weblogo.berkeley.edu/`

"o" is far less common than would be expected from the frequent occurrence of each of them separately. Figure 2 gives a more general picture of this phenomenon.

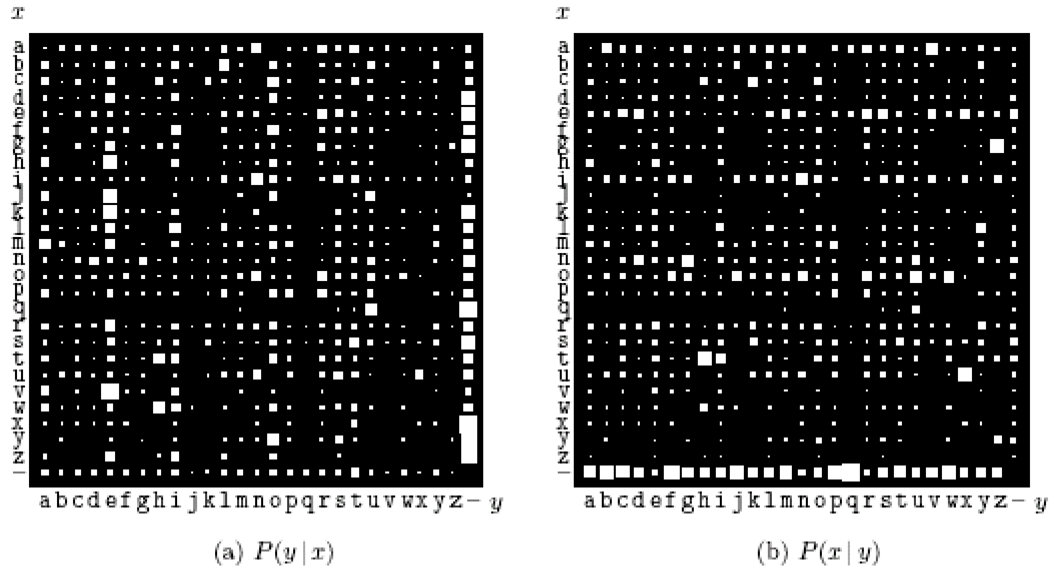


**Figure 2.** Conditional probability distributions: Hinton diagrams (a) *P(y|x)*: Each *row* shows the conditional distribution of the second letter, *y*, given the first letter, *x*. (b) *P(x|y)*: Each *column* shows the conditional distribution of the first letter, *x*, given the second letter, *y*. Text for estimating the estimating the probabilities was *The Frequently Asked Questions Manual for Linux* [From Ref. 14].
If variables *x* and *y* were independent, all conditional distributions *P(y|x)* would be identical functions of *y*, regardless of *x*.

Genome sequences also contain many such dependencies. An example is the distribution of `CG` dinucleotides along the human genome: `CG`-pairs are rarer in most parts of the genome than would be expected from the probabilities of `C` and `G` independently. Dependencies are not limited to neighbouring nucleotides but can also be longer-ranging.

### 1.3.1. Dinucleotide frequencies

How much dependence is there between dinucleotides (that is, *adjacent* nucleotides) in a DNA sequence? Since there are four nucleotides, there are 16 possible pairs of nucleotides. Let $P_{ij}$ be the frequency of the nucleotide *i* immediately followed by the nucleotide *j*. To calculate the frequencies of each such pair (*i,j*) in a sequence, one counts the total number of observed occurrences of *i* followed immediately by *j* and then divide by the total number of pairs, which is the length of the sequence minus one. In addition let $P_i$ be the frequency of nucleotide *i* in the single nucleotide distribution. The value

$$s_{ij} = \frac{P_{ij}}{P_i P_j} \qquad \textbf{(1.4)}$$

gives a score representing the validity of the positional independence assumption. If $s_{ij} = 1$, then the independence assumption is valid for nucleotide *i* followed by nucleotide *j* (1.18). As the deviation from one increases, the independence assumption becomes less valid.

**Example:** This example calculates $s_{ij}$ for *M. jannaschii*, which is a prokaryote that lives in extremely high temperature environments such as thermal springs. The ratios of the observed dinucleotide frequency to the expected dinucleotide frequency (assuming independence) in *M. jannaschii*:[21]

$$\begin{array}{c|cccc} & A & C & G & T \\ \hline A & 1.13 & 0.73 & 1.10 & 0.94 \\ C & 1.03 & 1.37 & 0.32 & 1.11 \\ G & 1.05 & 1.12 & 1.39 & 0.71 \\ T & 0.83 & 1.05 & 1.13 & 1.14 \end{array}$$

with the nucleotide $i$ indexed by row and the nucleotide $j$ indexed by column, where nucleotide $i$ immediately precedes $j$ in the sequence. Upon examination of the matrix, one can see that there are sizable deviations from one. For example the pairs (`C,C`) and (`G,G`) occur much more often than expected if they were independent, and (`A,C`), (`C,G`), and (`G,T`) occur much less often. Also, the diagonal entries show that two consecutive occurrences of the same nucleotide occur more often than expected.

### 1.3.2. Markov Chains

A simple way to incorporate such dependencies is to assume that each symbol depends *only* on the previous few symbols in the sequence; such models are called *Markov chains.* These were already presented in the introductory course.[12] Here, we just give a brief reminder of the most important ingredients.

Imagine that we want to make a probabilistic model of a set of sequences from a certain alphabet. The probability of a sequence $q_1 q_2 ... q_N$ can be written as

$$P(q_N, q_{N-1}, ..., q_1) = P(q_N \mid q_{N-1}, ..., q_1) P(q_{N-1} \mid q_{N-2}, ..., q_1) ... P(q_1) = \prod_{t=2}^{N} P(q_t \mid q_{t-1}, ..., q_1) P(q_1)$$

by successive application of the product rule (1.16). In general, one does not want to model all these conditional probabilities. Markov chains correspond to a specific way of restricting this decomposition. In a Markov chain, the probability of a symbol only depends on the previous symbol:[e]

$$P(q_N, q_{N-1}, ..., q_1) = \prod_{t=2}^{N} P(q_t \mid q_{t-1}) P(q_1) .$$

In a Markov chain, the values that a random variable can take are called *states*. The probability to go from one state to another ($P(q_t = j \mid q_{t-1} = i)$) is called a *transition probability*,[f] also written as $a_{ij}$. By adding an initial state $q_0$=0 to the model, we can write the whole model in terms of transition probabilities:

$$P(q_N, q_{N-1}, ..., q_1) = \prod_{t=1}^{N} P(q_t \mid q_{t-1}) . \qquad \textbf{(1.5)}$$

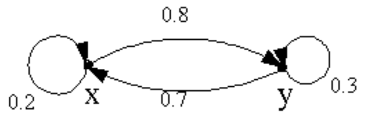
Similarly, we can also add an end state $q_{N+1}$ to ensure that the end of a sequence is modelled, although it is often assumed that a sequence can end in any state.

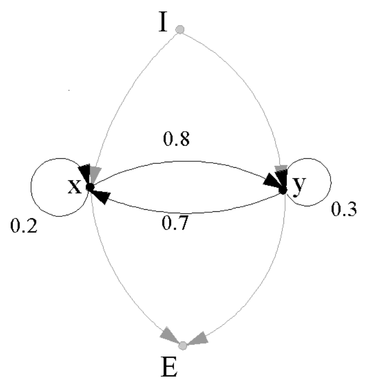
Two standard ways of representing Markov chains are matrices and graphs. Since a Markov chain is completely defined by its transition probabilities, it can be summarized by the *transition matrix*

[e] Actually, this is called a first-order Markov chain. In an *n*th-order Markov chain each symbol depends on the previous *n* symbols (Section 1.6).
[f] This is equivalent to the dinucleotide frequency defined above.

$A=(a_{ij})$. This can also be depicted as a directed graph where the states are the nodes of the graph with directed edges of value $a_{ij}$ pointing from state *i* to state *j*. Here is a simple example of a two-state Markov chain:

$$A=\begin{array}{c|cc} & x & y \\ \hline x & 0.2 & 0.8 \\ y & 0.7 & 0.3 \end{array}$$





The right-hand Markov chain explicitly includes an initial (I) and an end (E) state.

Given a set of DNA sequences *D*, a simple Markov chain has five states (nucleotides `A`, `C`, `G`, and `T`, and the initial state). Transition probabilities are estimated by counting the number of times $c_{ij}$ that nucleotide *i* is followed by nucleotide *j* in *D*:

$$a_{ij}=\frac{c_{ij}}{\sum_{j'} c_{ij'}}. \tag{1.6}$$

The transition probabilities from the initial state are estimated by counting the number of occurrences $d_i$ of nucleotide *i* in *D*:

$$a_{0i}=\frac{d_i}{\sum_{j} d_j}. \tag{1.7}$$

These are the maximum likelihood (Section 1.9.2) estimators for the transition probabilities.

### 1.3.3. A different view on log-odds: Bayes' decision rule

Imagine that we estimated two Markov chains, one from a set of sequences corresponding to class *A* (of sites, for example) and a second one from a set of sequences corresponding to class *B* (of non-sites). A collaborator gives us a new sequence $x=x_1x_2...x_N$ and wants to know to which of the two classes his sequence is likely to belong. The best way to take such a decision is by selecting the class for which the probability of the class given the sequence is largest.[1] This strategy which minimizes the probability of misclassification is known as Bayes' decision rule:

$$x \text{ is assigned to class } A \quad \Leftrightarrow \quad P(\text{class } A|\, x) > P(\text{class } B|\, x).$$

Using Bayes' theorem (1.17) this can be rewritten as

$$x \text{ is assigned to class } A \;\Leftrightarrow\; \frac{P(x \mid \text{class } A)P(A)}{P(x)} > \frac{P(x \mid \text{class } B)P(B)}{P(x)}$$
$$\Leftrightarrow\; \frac{P(x \mid \text{class } A)}{P(x \mid \text{class } B)} > \frac{P(B)}{P(A)} \quad .$$

*Priors P(A)* and *P(B)* represent the probability that a sequence belongs to class *A* or *B,* respectively, when nothing is known about the sequence itself. Priors correspond to the fractions of sequences in each class, in the limit of an infinite number of observations. Assuming the prior probabilities to be equal:[g]

$$x \text{ is assigned to class } A \;\Leftrightarrow\; \frac{P(x \mid \text{class } A)}{P(x \mid \text{class } B)} > 1 \;\Leftrightarrow\; \log\left(\frac{P(x \mid \text{class } A)}{P(x \mid \text{class } B)}\right) > 0 .$$

The term on the left-hand side of the last inequality is the familiar log-odds ratio (1.2). If the log-odds ratio is larger than zero, we can tell our collaborator that his sequence is likely to belong to class *A* otherwise class *B* is more likely.
Unequal prior probabilities would justify a cut-off different from zero for the log-odds ratio. If we expect to see far more sequences of class *B*, say *P*(*B*)=0.7, than of class *A*, *P*(*A*)=0.3, a cut-off of log(0.7/0.3)=1.22 should be chosen. This captures the idea that it is more important to make no mistakes on sequences from class *B* since they are more frequent.

Since in our case class *A* and *B* were modelled with Markov chains, we can use (1.5) to rewrite the log-odds ratio as

$$\log\left(\frac{P(x \mid \text{class } A)}{P(x \mid \text{class } B)}\right) = \log\left(\frac{\prod_{t=1}^{N} P_A(x_t \mid x_{t-1})}{\prod_{t=1}^{N} P_B(x_t \mid x_{t-1})}\right) = \sum_{t=1}^{N} \log\left(\frac{P_A(x_t \mid x_{t-1})}{P_B(x_t \mid x_{t-1})}\right) , \qquad \textbf{(1.8)}$$
$$= \sum_{t=1}^{N} \log\left(\frac{a^A_{x_{t-1},x_t}}{a^B_{x_{t-1},x_t}}\right)$$

that is, the log-odds ratio can be written as the sum of the log-likelihood ratios of corresponding transition probabilities.

[g] From now on, we will just write log instead of $\log_2$.

## 1.4. Hidden Markov Models

Let us now extend the sequence ensemble introduced in section 1.1 with some insertions and gaps: [13]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| A | C | A | - | - | - | A | T | G |
| T | C | A | A | C | T | A | T | C |
| A | C | A | C | - | - | A | G | C |
| A | G | A | - | - | - | A | T | C |
| A | C | C | G | - | - | A | T | C |

You can think of this as being the result of a multiple sequence alignment, for example. Since not all sequences have the same length such a sequence ensemble cannot be summarized by a simple weight matrix. *Profiles*[11] were introduced to deal with this problem. All of the profile methods are more or less statistical descriptions of the consensus of a multiple sequence alignment. They use position-specific scores for nucleotides (or amino acids) and position-specific penalties for opening and extending a gap. A disadvantage of most early profile methods is that the choice of scoring parameters is heuristic and requires careful craftsmanship. In this section, we will see how an extension of Markov chains can be used to lay a sound probabilistic basis for profile methods.

For a start, we could have come up with a regular expression describing the set of sequences `[AT][CG][AC][ACGT]*A[GT][CG]`. The term `[ACGT]*` means that any of the four symbols can occur any number of times; this term corresponds to the insertions and gaps. The regular expression can be interpreted as a Markov chain with seven states, one state for each term in the regular expression. The big difference is that while a state in a Markov chain corresponds to one symbol, here a state corresponds to several symbols. This is best captured by combining weight matrices and Markov chains in one model by making the states generate symbols according to a certain probability distribution.

Such a model for the five sequences given above is shown in Figure 3. The first state corresponds to the term [AT], which in its turn corresponds to the first position of the multiple sequence alignment. The probability distribution in the first state is obtained by counting the number of times a symbol occurs at the first position: 4/5 for an `A` and 1/5 for a `T`. Similarly for the second position, the probability of `C` is 4/5 and for `G` is 1/5. The transition probability between state one and state two is of course equal to one .The more difficult case is the term `[ACGT]*` which is represented in Figure 3 with the state above the other states. This *insert* state corresponds to the positions 4-6 in the sequence ensemble:

Three out of five sequences contain insertions, thus the transition probability from state three `[AC]` to state four is 3/5 and from state three to state five is 2/5. The probability of each letter on the insert state is found by counting all occurrences of the four nucleotides in this region. The total counts are one `A`, two `C`s, one `G`, and one `T`, giving probabilities 1/5, 2/5, 1/5, and 1/5 respectively. Finally, for the transition probabilities going out of the insert state, we note that in total there are five transitions from the insert state: two more inserts in sequence two and three transitions to state five. The probability of staying in the insert state is 2/5 and the probability of going from the insert state to state five is 3/5.

Like with weight matrices, we could estimate the probability of the consensus sequence (given the state sequence 1-2-3-4-5-6-7)

$$P(\mathtt{ACACATC}) = 0.8\times1\times0.8\times1\times0.8\times0.6\times0.4\times0.6\times1\times1\times0.8\times1\times0.8 = 0.047 \qquad \mathbf{(1.9)}$$

and also define a log-odds score.

You can think of the model depicted in Figure 3 as a stochastic generator of DNA sequences. Starting from state 1, the generation of a sequence consists of drawing a nucleotide according to the probability distribution associated with state 1 (*p(*`A`*)*=0.8 & *p(*`T`*)*=0.2). Then a new state is chosen according to the transition probabilities of state 1 to other states, which in the case of Figure 3 leaves only state 2, and so forth.

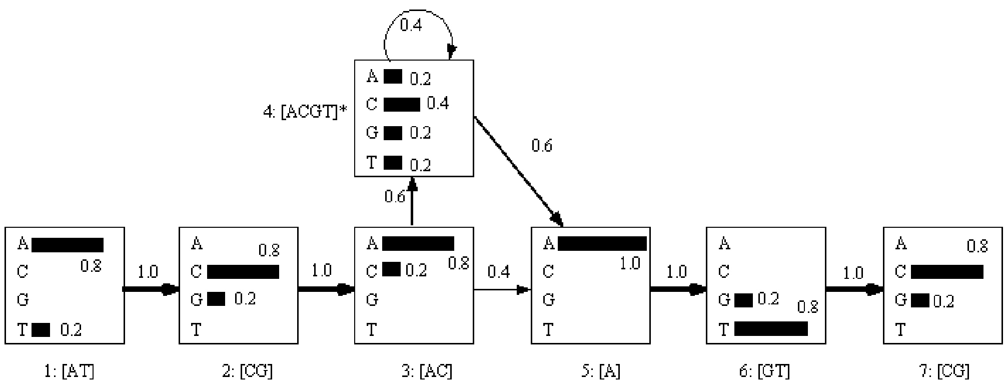


**Figure 3.** Hidden Markov model derived from the alignment discussed in the text. A box is called a state. The state number and the corresponding terms of the regular expression are given below the states. Transitions are shown with arrows the thickness of which indicates their probability. In each state the histogram shows the probabilities of the four nucleotides [Adapted from Ref. 13].

In a standard Markov chain, each state uniquely corresponds to an observation (*e.g.*, a nucleotide). In Figure 3 each state has an associated probability distribution of observations. This implies that if there is a C in a sequence generated by the model, we do not know whether it comes from state 2, 3, 4 or 7. The sequence of states is *hidden* from us and can only be observed indirectly through the probability distributions defined for each state. Such models are a well-known extension of Markov chains and are known as *hidden Markov models* (HMM).

Formally a HMM consists of

1. A set of states $Q=\{0,1,2,\ldots,S\}$, state 0 is the initial state. In general, the sequences we study have a finite length $X = x_1 x_2 \ldots x_N$ and the model is often extended with an *end* state $q_{N+1}=S+1$. Since the role of initial and end states is to model the start and end of a sequence, they do not have an associated emission probability distribution.

2. An alphabet $K$ of symbols (observations), the size of the alphabet is denoted as $M$.

3. Transition probabilities from state $i$ to state $j$

$$a_{ij} = P(q_t = j \mid q_{t-1} = i), \quad 0 \le i, j \le S+1 .$$

   These can be summarized in a S+2 by $S$+2 matrix $A = (a_{ij})$. No transitions are possible to the begin state ($a_{i0} = 0$) and from the end state ($a_{S+1,i} = 0$).

4. The *emission* probability distributions of state $i$

$$b_i(x) = P(x \mid i) \quad x \in K,\ 1 \le i \le S .$$

Let us now have a look at an extreme case. If for each state there is only one possible transition, with probability equal to one in that case, the state sequence is fully defined and the model is a weight matrix defined by the emission probabilities. Indeed, hidden Markov models are a, hopefully useful, extension of the models we have seen in the previous sections. In our example, the set of states

$Q$={1,2,..7}, there are no initial and end states, the alphabet is $K$={`A`, `C`, `G`, `T`} of size $M$=4, and the transition probability matrix is the rather sparse matrix (empty entries are equal to zero)

$$\begin{array}{c|ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & & 1.0 & & & & & \\ 2 & & & 1.0 & & & & \\ 3 & & & & 0.6 & 0.4 & & \\ A=4 & & & & 0.4 & 0.6 & & \\ 5 & & & & & & 1.0 & \\ 6 & & & & & & & 1.0 \\ 7 & & & & & & & \end{array} .$$

The emission probability distributions are given by the histograms inside each state (Figure 3).

The three main problems that one would like to solve for an HMM are

1. *Evaluation*: Compute the probability of an observed sequence, given the model. Solving the evaluation problem enables us, for example, to compare different models by calculating their odds ratio.
2. *Decoding*: Given an observed sequence find a corresponding state sequence that is optimal in a meaningful sense. This can be used to align new sequences to a HMM that models the multiple alignment of a set of sequences. Another application is the prediction of genes in a sequence of unannotated DNA (section 1.7).
3. *Estimation*: How to adjust the parameters (transition and emission probabilities) of the model to a given set of sequences?

### 1.4.1. Evaluation

Above, we could easily calculate the probability of the consensus sequence (1.9) since we knew its state sequence. This holds for any state sequence $Q = q_0 q_1 q_2 ... q_N$ and observed sequence of symbols $x = x_1 x_2 ... x_N$.[h] First, using the product rule (1.16)

$$P(x,Q) = P(Q)P(x \mid Q) \qquad \textbf{(1.10)}$$

The probability of a state sequence $P(Q)$ follows a Markov chain and using (1.5)

$$P(Q) = \prod_{t=1}^{N} P(q_t \mid q_{t-1}) .$$

The probability of an observed sequence given the state sequence is

$$P(x \mid Q) = P(x_N \mid x_{N-1},...,x_1,Q)P(x_{N-1} \mid x_{N-2},...,x_1,Q)...P(x_1 \mid Q) = \prod_{t=1}^{N} P(x_t \mid q_t) ,$$

[h] Note that, for ease of notation we do not include an end state here. It is easy to extend what follows with an explicit end state.

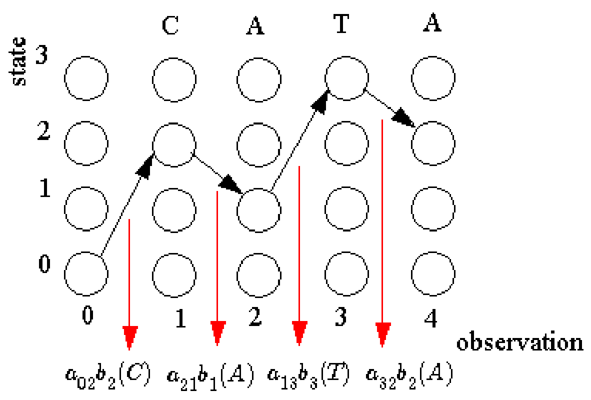


**Figure 4.** Trellis for the calculation of the probability of a given observation sequence (`CATA`) and state sequence (0-2-1-3-2), 0 is the non-emitting initial state.

since in an HMM, we assume that the observed symbol only depends on the current state. Eq. (1.10) can now be written as

$$P(x,Q) = \prod_{t=1}^{N} P(q_t \mid q_{t-1}) \prod_{t=1}^{N} P(x_t \mid q_t) = \prod_{t=1}^{N} a_{q_{t-1},q_t} \prod_{t=1}^{N} b_{q_t}(x_t) . \qquad \textbf{(1.11)}$$

Indeed, the probability of the consensus sequence (1.9) is an example of the application of this equation. We say that the joint probability of the observed sequence and the state sequence *factorizes.*

This computation can be visualized on a grid or *trellis* (Figure 4). Each node corresponds to a state $q_t$ at step $t$; hence, the size of the trellis is $(N+1)\times(S+1)$. The state sequence specifies a path through the trellis indicated by the arrows (or directed edges). The edges are weighted by the product of the corresponding transition and emission probability. Eq. (1.11) now corresponds to following the path indicated by the arrows through the trellis and multiplying the associated weights.

However, the state sequence is usually not known and the probability of an observed sequence has to be calculated over all possible state sequences (sum rule (1.15)):

$$P(x) = \sum_{Q} P(x,Q) \qquad \text{, where } Q \text{ has } N\text{+1 states.} \qquad \textbf{(1.12)}$$

This means that we would have to consider all possible paths through the trellis. Doing this explicitly is computationally infeasible.[i] With a clever rewriting we can limit the effort considerably:

$$P(x) = \sum_{i=0}^{S} P(x, q_N = i) = \sum_{i=0}^{S} \alpha(N,i) ,$$

where the *forward variable*

$$\alpha(t,i) = P(x_1 x_2 \dots x_t, q_t = i) ,$$

[i] Approximately $2NS^N$ operations: there are $S^N$ state sequences and for each of them calculating *P(x,Q)* using Eq.(1.11) requires about 2*N* multiplications.

that is, the probability of having observed $x_1x_2...x_t$ and being in state *i* at step *t*. Forward variables can be calculated recursively. Using the sum rule and then the product rule ($1 \le t \le N, 0 \le i, j \le S$):

$$\begin{aligned}\alpha(t,i) = P(x_{1..t}, q_t = i) &= \sum_j P(x_{1..t}, q_{t-1} = j, q_t = i) \\ &= \sum_j P(x_{1..t-1}, q_{t-1} = j) P(x_t, q_t = i \mid x_{1..t-1}, q_{t-1} = j)\end{aligned}.$$

where $x_{1..t} = x_1x_2...x_t$.

Since the observed symbol and the state at step *t* only depend on the previous state:

$$\begin{aligned}&= \sum_j P(x_{1..t-1}, q_{t-1} = j) P(x_t, q_t = i \mid q_{t-1} = j) \\ &= \sum_j \alpha(t-1, j) P(q_t = i \mid q_{t-1} = j) P(x_t \mid q_t = i). \\ &= \sum_j \alpha(t-1, j) a_{ji} b_i(x_t)\end{aligned}$$

This is where the recursion comes in: calculating the forward variables at step *t* only requires the forward variables at step *t*-1.
All sequences have to start in the initial state $q_0$=0, so $\alpha(0,0) = 1$. The resulting algorithm, called the *forward algorithm* is

**Forward Algorithm**

**initialization:** $\alpha(0,0) = 1$, $\alpha(0, j) = 0$ for $1 \le j$
**recursion** : $\alpha(t,i) = \sum_j \alpha(t-1, j) a_{ji} b_i(x_t)$ ($1 \le t \le N, 0 \le i, j \le S$)
**end** : $P(x) = \sum_{i=0}^{S} \alpha(N,i)$

The forward algorithm is far more efficient than a naïve calculation. Computational complexity of the forward algorithm is on the order of $NS^2$ operations.[j] The forward algorithm can also be visualized as a calculation on a trellis (Figure 5).

### 1.4.2. Decoding

The goal of decoding is to find a state sequence which best explains the observed sequence. The most widely used criterion is to find the most probable state sequence for sequence *x*:

$$V(x) = \max_Q P(Q \mid x) = \max_Q \frac{P(x,Q)}{P(x)} = \max_Q P(x,Q),$$

[j] There are about *NS* different alphas to calculate and each of them requires approximately *S* operations (see Ref. 15 for more details).

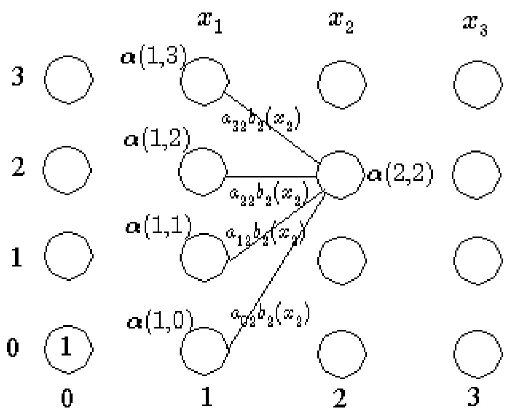


**Figure 5.** Forward algorithm on a trellis. The calculation of a forward variable only depends on the forward variables in the preceding layer: $\alpha(2,2) = \sum_{j=0}^{3} \alpha(1,j)a_{j2}b_2(x_2)$.

using the product rule (1.16). As with the forward algorithm, we want to find a way to do this recursively in order to find an efficient algorithm.

$$V(x) = \max_Q P(x,Q) = \max_i \left[\max_{Q_{0..N-1}} P(x,Q_{0..N-1},q_N = i)\right] = \max_i \left[v(N,i)\right],$$

where, the *Viterbi* variable

$$v(t,i) = \max_{Q_{0..t-1}} \left[P(x_{1..t},Q_{0..t-1},q_t = i)\right], \tag{1.13}$$

that is, the probability of having observed $x_1x_2...x_t$ along the most probable path ending in state *i* at step *t*.
The Viterbi variable can be calculated recursively using the product rule ($1 \le t \le N, 0 \le i,j \le S$):

$$\begin{aligned} v(t,i) &= \max_{Q_{0..t-1}} \left[P(x_{1..t},Q_{0..t-1},q_t = i)\right] = \max_j \left\{\max_{Q_{1..t-2}} \left[P(x_{1..t},Q_{0..t-2},q_{t-1} = j,q_t = i)\right]\right\} \\ &= \max_j \left\{\max_{Q_{0..t-2}} \left[P(x_{1..t-1},Q_{0..t-2},q_{t-1} = j)P(x_t,q_t = i \mid x_{1..t-1},Q_{0..t-2},q_{t-1} = j)\right]\right\} \end{aligned}.$$

Using the definition of the Viterbi variable and the fact that the observed symbol and the state at step *t* only depend on the previous state:

$$= \max_j \left[v(t-1,j)P(x_t,q_t = i \mid q_{t-1} = j)\right].$$

Applying once more the product rule:

$$\begin{aligned} &= \max_j \left[v(t-1,j)P(q_t = i \mid q_{t-1} = j)P(x_t \mid q_t = i)\right] \\ &= \max_j \left[v(t-1,j)a_{ji}b_i(x_t)\right] \end{aligned}.$$

All sequences have to start in the initial state $q_0$=0, so $v(0,0)=1$. This gives a recursive algorithm very similar to the forward algorithm, the main difference being that sums are replaced by a max operator. The Viterbi algorithm can again be visualized as a calculation on a trellis. As we fill in the trellis we also keep track of the node *p(t,i)* in the preceding layer from which the maximum *v(t,i)* originated. This enables us to find the optimal state sequence by *backtracking*. This is very similar to what you might already know from algorithms for pairwise sequence alignment. Both Viterbi and alignment algorithms are indeed examples of *dynamic programming* algorithms.

The resulting algorithm, called the *Viterbi algorithm* for finding the optimal state sequence $Q^* = q_0^* q_1^* q_2^* ... q_N^*$ is

**Viterbi Algorithm**

**initialization :** $v(0,0)=1$ , $v(0,j)=0 \quad \text{for } 1 \le j$

**recursion :** $v(t,i) = \max_j \; [v(t-1,j)a_{ji}]b_i(x_t) \qquad (1 \le t \le N, 0 \le i,j \le S)$

$p(t,i) = \operatorname{argmax}_j \; [v(t-1,j)a_{ji}]$

**end :** $V(x) = \max_i \; [v(N,i)]$

$q_N^* = \arg\max_i [v(N,i)]$

**backtracking:** $q_t^* = p(t+1, q_{t+1}^*) \qquad (0 \le t \le N-1)$

The general principle behind the Viterbi algorithm is that the optimal path to node *B* at step *t* always consists of the optimal path from the begin state to a node at an intermediary step and the optimal path from there to node *B*.

Let us have a look at a simple example (adapted from Ref. 9). In a casino a fair die is used most of the time but now and then they switch to a loaded die with a bias for sixes. This can be seen as an HMM with states *F* (fair), *L* (loaded) and 0 (begin state) with transition probabilities:

$$A = \begin{array}{c} 0 \\ F \\ L \end{array} \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0 & 0.95 & 0.05 \\ 0 & 0.1 & 0.9 \end{pmatrix} \quad \text{and emission probabilities:} \quad \begin{array}{ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ F: & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ L: & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{2} \end{array}.$$

We are now going to search the optimal state sequence for the observation sequence 3-1-4-6-6-6.
The trellis corresponding to this sequence is shown in Figure 6.

The example of Figure 6 also illustrates a typical problem of both the forward and the Viterbi algorithm as we have formulated them. The calculation of the forward and Viterbi variables consists of products of a large number of values less than one. For sufficiently long sequences, forward and Viterbi variables will become too small for floating point arithmetic. The forward algorithm can be made more robust by scaling intermediary values; the reader is referred to Ref. 15 for a detailed

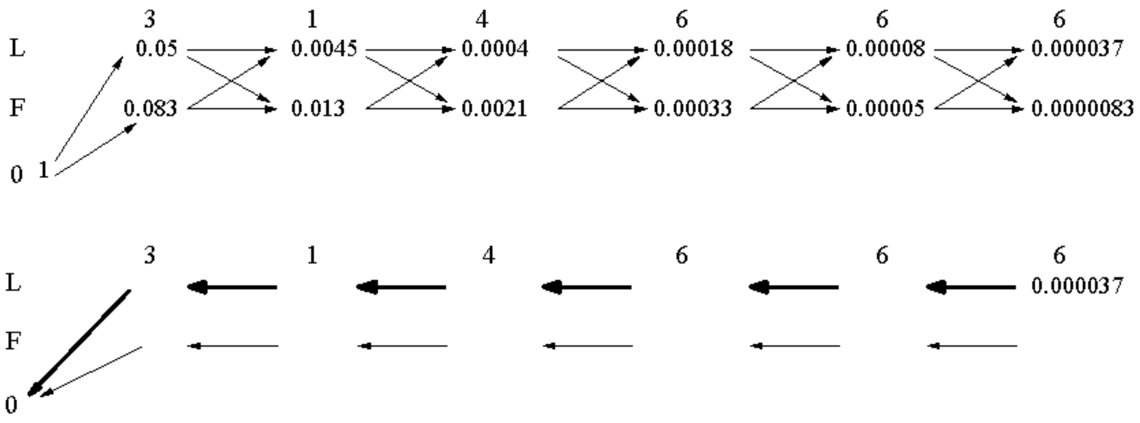


**Figure 6.** Trellis of the Viterbi algorithm for a sequence from an occasionally dishonest casino. Upper: calculation of the Viterbi variables *v(t,i)* on the trellis. For example, $v(2,\mathrm{L}) = \max[v(1,\mathrm{F})a_{\mathrm{FL}}b_{\mathrm{L}}(1)\ ,\ v(1,\mathrm{L})a_{\mathrm{LL}}b_{\mathrm{L}}(1)] = \max[(0.083\times0.05\times0.1\ ,\ 0.05\times0.9\times0.1] = 0.0045$. Lower: arrows indicate the pointers for backtracking. The optimal state sequence is found by following the (bold) arrows back from the maximal value of 0.000037 till the begin state. This gives 0-L-L-L-L-L-L as optimal state sequence for the observation sequence 3-1-4-6-6-6.

description. An elegant solution for the Viterbi algorithm is to do all calculations using logarithms.

### 1.4.3. Estimation

How to adjust the parameters (transition and emission probabilities) of a HMM to a given set of sequences? There are well-developed methods that find the maximum likelihood (Section 1.9.2) estimates of the parameters using the *Expectation-Maximization* (EM) algorithm (Section 1.9.5), in the context of HMMs also known as the *Baum-Welch* algorithm.[15] To avoid overloading this chapter with more equations, I will not present the Baum-Welch algorithm in full but only give an outline of the basic ingredients.

The problem we have to solve is to estimate values for the parameters $\theta$ of the HMM given a set of sequences $D = \{x^1, \ldots, x^n\}$. In Section 1.9.2 it is explained that a general method for doing this is to maximize the joint probability density, also called the likelihood. Assuming that the sequences are independent the likelihood of the parameters with respect to the data is:

$$\begin{aligned} P(x^1,\ldots,x^n \mid \theta) &= \prod_{i=1}^{n} P(x^i \mid \theta) \\ &= \prod_{i=1}^{n} \sum_{Q} P(x^i, Q \mid \theta) \end{aligned},$$

with $Q$ a state sequence. Maximizing this likelihood is a difficult problem because of the sum over all state sequences. The Baum-Welch algorithm offers a way out and finds maximum likelihood estimates for the parameters of a HMM in reasonable time. Full details of the Baum-Welch algorithm can be found in Refs. 9 and 15. A disadvantage of the Baum-Welch algorithm is that it only guarantees convergence to a *local* maximum of the likelihood. Especially, for large HMMs with many states the likelihood surface is riddled with local maxima and which one you end up with depends strongly on the starting values of the parameters.

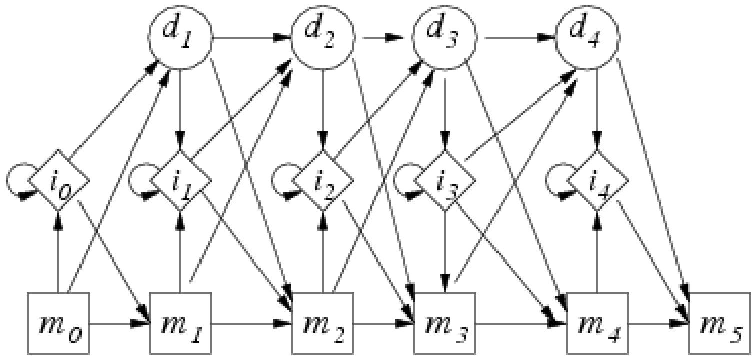


**Figure 7.** Structure of a profile hidden Markov model.

The EM algorithm does give estimates for the parameters of the model but does not help us with another major problem: the choice of the structure of an HMM. In a certain sense, the two examples you saw above represent two extremes. The HMM for the occasionally dishonest casino is fully connected, that is, transitions are possible from each state to any other state (in our example, there are only two states). The HMM for a multiple alignment with gaps on the other hand is very sparse. However, which structure to choose for a given problem is more of an art than a science. An important guideline is that the number of parameters of the model should not be too large to be reliably estimated from the data at hand. Moreover, the structure of the model should have an interpretation in terms of our knowledge of the problem. This was certainly the case for the multiple alignment HMM as we will see in more detail in the next section.

### 1.4.4. Profile HMMs

At the beginning of this section, we saw an example of how a multiple alignment with gaps can be represented by a HMM. What we actually constructed is a simplified version of a so-called *profile* HMM (Figure 7).[9,13] This model contains three different types of states indicated in the figure by squares, diamonds, and circles. The square shaped states are called *match* states, because they model the columns of the alignment. In these states the probability distribution is just the frequency of the nucleotides at the corresponding position in the multiple alignment, as in our example. States $m_0$ and $m_5$ are the begin and end state of the profile HMM. The diamond shaped states are *insert* states, which are used to model highly variable regions in an alignment. These are like state four in our example (Figure 3). The circular states are called *delete* states. Like the begin and end state, these are non-emitting states. Delete states allow skipping one or more columns in an alignment, that is, to model the situation when just a few of the sequences have a '-' in the multiple alignment at a position.

A profile HMM can of course be used to model a "family" of sequences from a given reliable multiple alignment and search a set of sequences for other likely members of the family. The other principal use of a profile HMM is to align a new sequence to it. This can be done by finding the most probable state sequence in the profile HMM using the Viterbi algorithm. PFAM (Protein families database of alignments and HMMs) does exactly this to determine protein domains in a query sequence.[k] A more detailed description of these and other issues related to profile HMMs such as model estimation and non-global alignment can be found in Ref. 9.

## 1.5. Intermezzo on gene structure

The goal of gene finding algorithms is to predict the location and structure of all genes in a genome as reliably as possible. Weight matrices, Markov chains, and HMMs form essential ingredients of almost

[k] `https://www.ebi.ac.uk/interpro/`

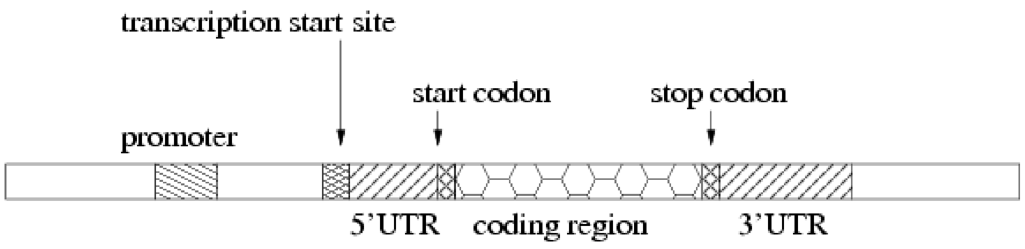


**Figure 8.** Structure of a prokaryotic gene. The coding region of a prokaryotic gene consists of a single stretch of uninterrupted nucleotide sequence that encodes the amino acid sequence of a protein.

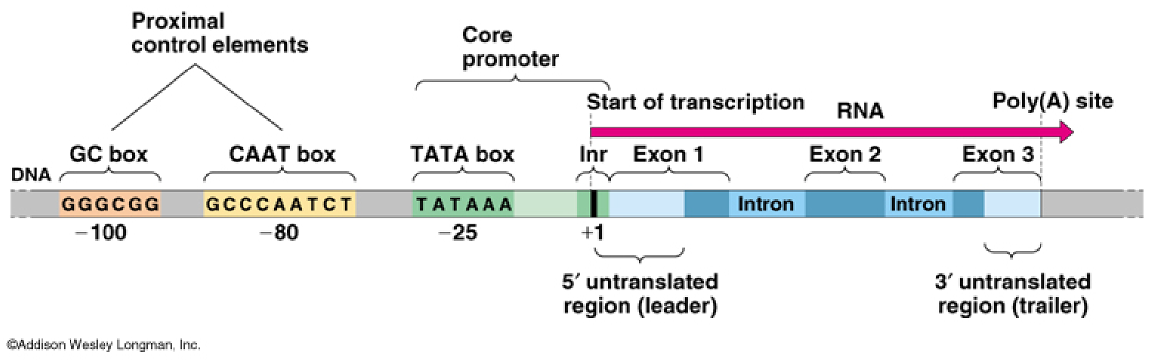


**Figure 9.** Detailed structure of a eukaryotic gene [From Ref. 1]. Exons of most eukaryotic genes are interrupted by introns (noncoding sequences). Promoters for transcription are indicated in green.

all state-of-the art gene finding algorithms. Before going into more detail, this section gives a short reminder of the structure of protein coding genes in prokaryotes and eukaryotes.[12]

**Prokaryotes** In prokaryotic cells, that is cells without a nucleus, most of the DNA sequence is coding for protein. A gene starts with the promoter region which is followed by a transcribed but non-coding region called 5' untranslated region (5' UTR). Then follows the coding region delimited by a start codon (ATG) and a stop codon (TAA, TAG, TGA). It is followed by another non-coding region called the 3' UTR.

**Eukaryotes** Gene structure and the gene expression mechanism in eukaryotes are far more complex than in prokaryotes. The protein coding region of the DNA is composed of alternating stretches of exons and introns. During transcription, both exons and introns are transcribed onto mRNA. Then *splicing* takes place: introns are excised from the mRNA sequence. Remaining mRNA segments, those corresponding to exons, are ligated to form a mature RNA strand. A typical multi-exon gene has the structure shown in Figure 9. The gene starts with the promoter region, which is followed by the non-coding 5' UTR. Then follows the initial exon which contains the start codon.[1] Following the initial exon, there is an alternating series of introns and internal exons, followed by the terminating exon, which contains the stop codon. It is followed by the non-coding 3' UTR. At the end of the gene there is a polyA signal (often ATTAAA or AATAAA), which is followed by a polyadenylation site. This site ends the transcription after which a polyA tail is added to the mRNA.
The problem of gene identification is complicated in the case of eukaryotes by the vast variation found in gene structure: number and length of exons, length of the untranslated regions, and length of introns.

[1] Sometimes, the start codon only occurs in the second or third exon.

## 1.6. Gene finding in prokaryotes: extensions of Markov chains

Since the structure of prokaryotic genes is simple, one can hope for rather basic models being able to predict their location reliably. Like before, we could build a Markov chain for coding regions and a null model for non-coding regions and use the log-odds ratio to decide whether a stretch of sequence is coding or non-coding.
A first problem of this approach is that in order to estimate the parameters of the coding region Markov chain, one needs a reliable set of coding regions in the first place. For prokaryotes there is an easy way out of this chicken and egg problem: just extract long open reading frames (ORF)[m]. Since three of the 64 codons are stop codons, one would expect the mean length of an ORF to be around 20. Therefore, a sufficiently long (300-500 base pairs) ORF has a high probability of being a gene. A very reliable training set for the coding region Markov chain can be constructed by extracting long ORFs that do not overlap long ORFs in other reading frames. However, it turns out that a first-order Markov model is not very successful in locating genes.[9]

We might draw inspiration from Markov chains as applied to human language. In his seminal paper "A Mathematical Theory of Communication"[18] which laid the basis for the field of information theory, Claude E. Shannon gives a nice example of how a series of ever more complex Markov chains approaches the English language. Some typical sequences in the approximations to English are given below. In all cases a 27-symbol alphabet was assumed, the 26 letters and a space.

1. Zero-order approximation (symbols independent but with frequencies of English text).
```
OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL.
```

2. First-order Markov (transition probabilities as in English).
```
ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT
TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.
```

3. Second-order Markov (transition probabilities as in English).
```
IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF
THE REPTAGIN IS REGOACTIONA OF CRE.
```

In a second-order Markov chain, the probability of a symbol depends only on the two symbols preceding it. This model does seem to give a better approximation of English than a first-order model. However, when applying a second-order model to prokaryotic gene finding results are hardly better than with a first-order Markov chain.[9] Continuing with Markov chains of even higher order does not help much and, moreover, has the disadvantage of requiring a large training set for estimating the transition probabilities: a $k$th-order Markov chain on the nucleotide alphabet has $4^{k+1}$ transition probabilities.

Shannon continued by switching from letters to words as the states of a Markov chain.

4. Zero-order word approximation. Words are chosen independently but with their appropriate frequencies.
```
REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE
HE THE A IN CAME THE TOOF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE
HAD BE THESE.
```

5. First-order Markov (on words). Word transition probabilities are as in English.
```
THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF
THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO
EVER TOLD THE PROBLEM FOR AN UNEXPECTED.
```

[m] A sequence of codons without stop codon.

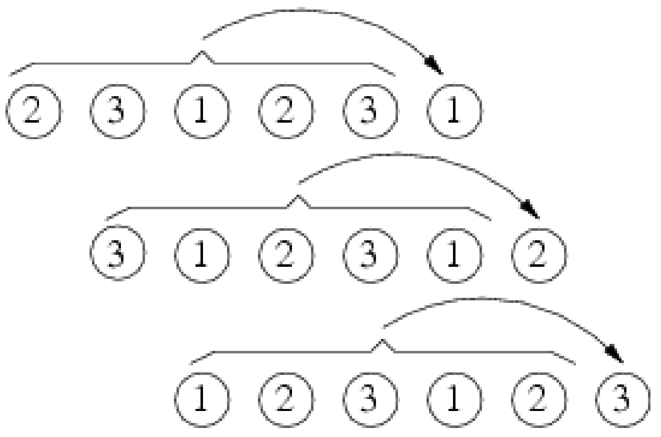


**Figure 10.** An inhomogeneous Markov chain consisting of three fifth-order Markov chains. Circles represent consecutive nucleotides, numbers indicate the codon position, and the arrows indicate that the probability of a nucleotide depends on the five preceding nucleotides [Adapted from Ref. 8].

In the last example, sequences of four or more words can easily be placed in grammatically correct sentences. There is even an understandable sequence of ten words: "attack on an English writer that the character of this". The equivalent in the DNA alphabet of Shannon's first-order Markov model on words is a Markov chain on codons, and hence with 64 states. Such a model performs reasonably well in locating prokaryotic genes.[9] One of the reasons for the good performance is that the codon-based model better captures an organism's preference for a specific codon over other codons that encode for the same amino acid.

A related approach also tries to model codon statistics, especially the fact that the three codon positions in most organisms have quite different statistics. Therefore, it is natural to construct three different Markov chains, one for each of the three possible codon positions (Figure 10). Such a model is called an *inhomogeneous* Markov chain and is used in several state-of-the-art prokaryotic gene-finding methods such as GeneMark[6] and Glimmer.[16] Both methods use a fifth-order inhomogeneous Markov chain to find coding regions. This choice allows any nucleotide to depend on all the nucleotides in its codon and the immediately preceding codon.
Estimation of the parameters of an inhomogeneous *k*-th order Markov chain is done separately for each of the three component models. For any of the three codon positions, this is done by counting the occurrences of all substrings of length *k*+1 ending in that codon position. As already indicated above, a *k*th-order Markov chain requires estimating $4^{k+1}$ transition probabilities, that is, 4096 parameters for *k*=5. An inhomogeneous Markov chain consists of three separate such chains and, therefore, has more than 12.000 parameters. Each of these 12.000 substrings of length six must occur often enough in the training set to support a statistically reliable sample. Some substrings of length five are already too infrequent in prokaryotic sequences, yet some substrings of length eight are frequent enough to be statistically reliable. Glimmer[16] uses a so-called *interpolated* Markov chain, in which a high-order model is used in contexts where enough data is available and a lower-order model for infrequent substrings.

## 1.7. HMMs in Eukaryotic Gene Finding

The problem of predicting the location and structure of all genes in a eukaryotic genome is far more difficult, for two main reasons. Firstly, coding density in eukaryotic genomes is much lower than in prokaryotic genomes. In the prokaryote *H. influenzae*, 85% of the genome is in coding regions, whereas in fly and worm only 25% of the genome is in coding regions, and the number falls to just a few per cent in humans.[20] An additional complication is the presence of splicing (Figure 9). In the human genome, a typical exon is 150 base pairs long and a typical intron several kilobases, and there is no clear delineation between the intergenic regions that separate adjacent genes and the intragenic regions that separate exons. Defining the precise start and stop position of a gene and the splicing pattern of its exons among all the non-coding sequence is very difficult.

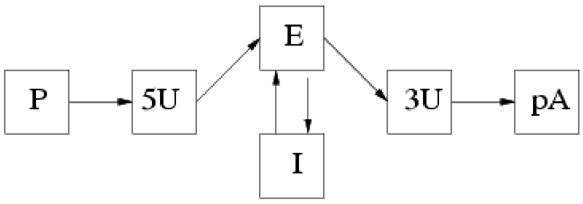


**Figure 11.** HMM representation of the regular expression `P 5U E (I E)* 3U pA`

When trying to predict eukaryotic gene structure, there are many different signals that can be taken into account: promoters, acceptor and donor splice sites, codon bias, and polyA site all give some clue about gene structure. However, any isolated signal of a gene is hard to predict. Current methods for promoter prediction, for example, either predict a large number of false positives or a small proportion of true promoters.[3] It turns out to be a far better approach to incorporate all these different signals into one consistent model. Splice site prediction, for instance, can be greatly improved when not only modelling the splice site itself but also the coding exon region next to the splice site. This approach can rule out candidate splice sites that are not adjacent to something looking like a coding region.

Therefore, state-of-the art gene finding methods all predict whole gene structures from a combination of different modules. The modules are often made up of models we have seen in the previous sections. Weight matrices, first- and higher-order Markov chains, inhomogeneous and interpolated Markov chains are all used as building blocks of current gene finders. How to combine these modules into one consistent model? We start by observing that a combination of modules has to follow certain rules since all eukaryotic genes have a particular structure (Figure 9). This structure is defined by what we could call the *grammar* of a gene. If you consider exons and introns as the 'words' in language, the sentences are of the form exon-intron-exon-intron ... intron-exon. The sentences can never start or end with an intron, for example. We could even come up with a regular expression describing the structure of a gene in more detail:

```
promoter 5'UTR exon (intron exon)* 3'UTR polyA
```
**(1.14)**

As we have seen in Section 1.4, such a regular expression can be represented by a hidden Markov model. A HMM corresponding to the above regular expression is given in Figure 11. Thus, all theory developed for HMMs can also be applied to the problem of gene finding. For example, one can predict genes in a sequence of anonymous DNA by finding the optimal path through the HMM with the Viterbi algorithm. When this path goes through an "exon" state, an exon is predicted, when it goes through a 5'UTR state, a 5' untranslated region is predicted, and so on.

State-of-the-art gene finders do not have as simple a structure as the one shown in Figure 11. They often include a module that ensures frame consistency throughout a gene. This is necessary because a single codon may be split between two exons: the reading frame in one exon has to fit the one in the next. This can be handled by creating three different intron states, one for each reading frame.[13] A good example is the HMM used in GenScan,[7] one of the best ab initio gene finding algorithms, shown in Figure 12. Though considerably more complex than our simple diagram, basic principles are more or less the same.

## 1.8. References

1. B. Alberts et al., Essential Cell Biology: An Introduction to the Molecular Biology of the Cell, Garland Publishing (New York, 1998).
2. B. Alberts et al., Molecular Biology of the Cell, Garland Publishing (New York, 2002).
3. V.B. Bajic, Promoter prediction analysis on the whole genome. *Nat. Biotech.* 22: 1467-1473 (2004).
4. W.M. Becker et al., The World of the Cell, Benjamin Cummings (4th edition, 2000).

5. C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press (Oxford, 1995).
6. M. Borodovsky et al, GeneMark: Parallel gene recognition for both DNA strands. *Comp.Chem.*, 17(2):123–132 (1993).
7. C. Burge et al., Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.,* 268:78-94 (1997).
8. C.B. Burge et al., Finding the genes in genomic DNA. *Current Opinion in Structural Biology*, 8:345-354 (1998).
9. R. Durbin et al., Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press (Cambridge, 1998).
10. W. Ewens et al, Statistical Methods in Bioinformatics: An Introduction, Springer-Verlag (New York, 2001).
11. M. Gribskov et al., Profile Analysis: Detection of Distantly Related Proteins. *Proc. Natl. Acad. Sci. USA*, 84: 4355–4358 (1987).
12. A.H.C. van Kampen et al., Introduction to Genefinding. Chapter 5 in *Introduction to Bioinformatics*, lecture notes available on request (Amsterdam, 2004).
13. A. Krogh, An Introduction to Hidden Markov Models for Biological Sequences. In S. Salzberg, D. Searls, and S. Kasif, eds., *Computational Methods in Molecular Biology,* pp.45–63, Elsevier (1998).
14. D.J.C. MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge University Press (Cambridge, 2003).
15. L. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE* 77: 257-286 (1989).
16. S.L. Salzberg et al., Microbial gene identification using interpolated Markov models. *Nucleic Acids Research*, 26(2):544–548 (1998).
17. D.B. Searls, The Language of Genes. *Nature*, 420: 211–217 (2002).
18. C.E. Shannon, A Mathematical Theory of Communication. *The Bell System Technical Journal* 27: 379-423, 623-656 (1948).
19. E. Sonnhammer et al., PFAM: A Comprehensive Database of Protein Families Based on Seed Alignments. *Proteins*, 28: 405–420 (1997).
20. L. Stein, Genome Annotation from Sequence to Biology, *Nature Reviews Genetics* 2: 493-503 (2001).
21. M. Tompa. Computational Biology (2000). Lecture notes available from: `https://courses.cs.washington.edu/courses/cse527/00wi/`

For those interested in the application of hidden Markov models in automatic speech recognition, the following two references are recommended:

1. L. Rabiner and B. Juang, Fundamentals of Speech Recognition, Prentice Hall (1993).
2. L. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, Proceedings of the IEEE 77: 257-286 (1989).

David MacKay's book (Ref. 14) is instructive and fun to read. He even made it downloadable at `http://www.inference.org.uk/mackay/itila/` (although I think it is worth its price).

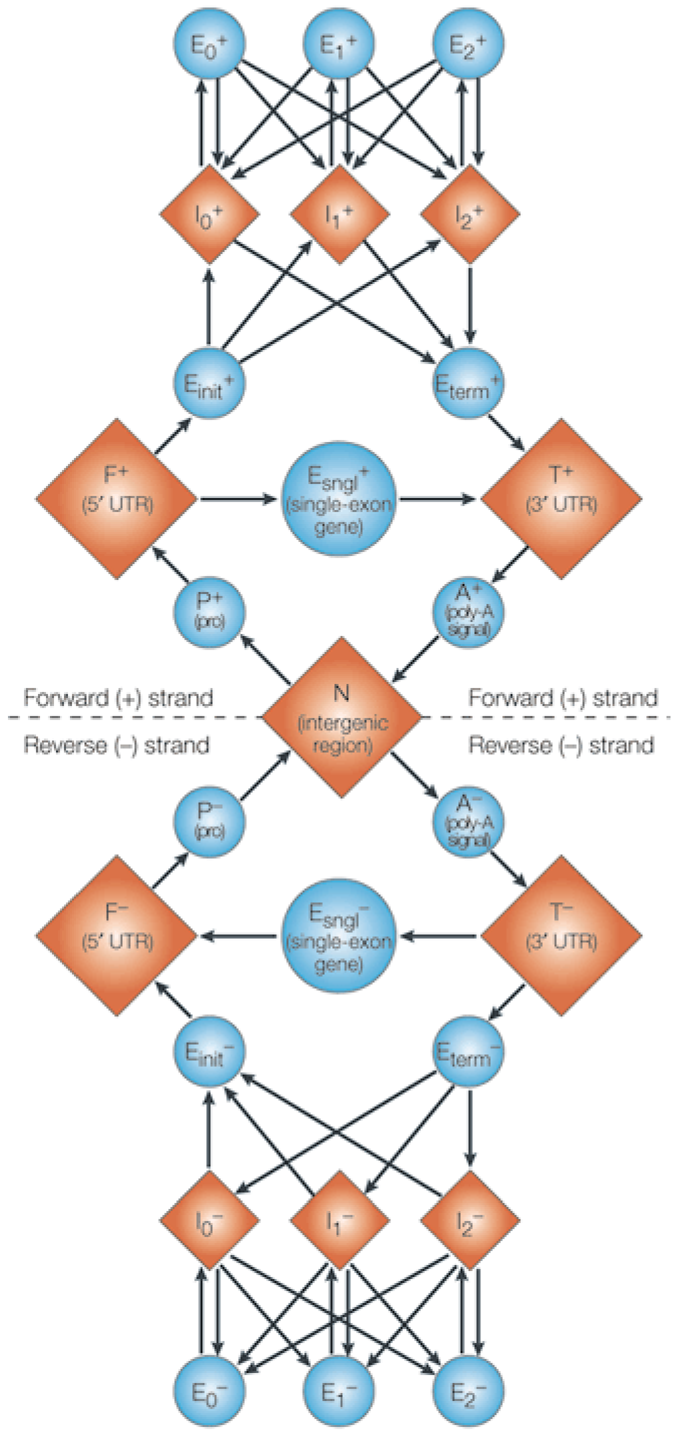


**Figure 12.** A hidden Markov model explicitly models the probabilities for the transition from one part of a genomic region to another. In this model, used by the GENSCAN algorithm, each circle or diamond represents a functional unit of a eukaryotic gene. E=exon; I=intron; UTR=untranslated region; pro=promoter. For example, $E_{init}$ is the initial exon (from start codon to donor splice site) and $E_{term}$ is the last exon (from acceptor splice site to stop codon). The upper half of the figure corresponds to states of a gene on the forward strand (indicated with superscript +). States that occur below the dashed line correspond to a gene on the opposite strand (indicated with superscript -). Arrows represent the probability of a transition from one state to another. [From Ref. 20].

## 1.9. Appendix

### 1.9.1. Probabilities

First some notation:

$P(x = a_i)$ denotes the probability that random variable *x* takes on value $a_i$ from a possible set of values $\{a_1, a_2, ..., a_N\}$. Often the random variable is clear from the context and we will just write $P(a_i)$ or $p_i$ with $p_i \geq 0$ and $\sum_{i=1}^{N} p_i = 1$.

The *joint* probability of two random variables *x* and *y* is denoted as $P(x = a_i, y = b_j)$. The *conditional* probability that *y* equals $a_i$, given that *x* equals $b_j$ is denoted as $P(x = a_i \mid y = b_j)$. Often the specific value of the random variable will not be specified explicitly and we will just write *P(x)*, *P(x,y)*, and *P(x|y)*.

Probabilities obey the following basic (and intuitive) rules:

**Sum rule**

$$P(x) = \sum_y P(x, y) \qquad \textbf{(1.15)}$$

This is called the marginal probability.

**Product rule**

$$P(x, y) = P(x \mid y)P(y) = P(y \mid x)P(x) = P(y, x) \qquad \textbf{(1.16)}$$

**Bayes' theorem** A straightforward consequence of the sum and product rule is Bayes' rule:

$$P(x \mid y) = \frac{P(y \mid x)P(x)}{P(y)} = \frac{P(y \mid x)P(x)}{\sum_{x'} P(y \mid x')P(x)}. \qquad \textbf{(1.17)}$$

**Independence** Two random variables *x* and *y* are independent if and only if

$$P(x, y) = P(x)P(y). \qquad \textbf{(1.18)}$$

**Expectation** The *expected* value of a function *f*(*x*) with *x* a random variable is

$$E[f(x)] = \sum_x P(x) f(x) \qquad \textbf{(1.19)}$$

Above, we assumed the random variables to be discrete; what happens of they are continuous? Then, the probability that random variable *x* takes on a specific value is in general zero. It makes more sense to look at the probability that *x* takes on a value in a certain interval:

$$P(a \le x \le b) = \int_a^b p(x)dx$$

where *p(x)* is the so-called probability density function with properties $p(x) \ge 0$ and $\int_{-\infty}^{\infty} p(x)dx = 1$.

The basic probability rules given above are easily adapted to the case of continuous random variables by replacing sums by integrals.

### 1.9.2. Maximum likelihood

An example of a discrete probability distribution is the *Bernoulli* distribution of an experiment having two possible outcomes labelled by $x = 0$ and $x = 1$ in which $x = 1$ ("success") occurs with probability *p* and $x = 0$ ("failure") occurs with probability $1-p$. The probability density function of the Bernoulli distribution is:

$$P(x) = p^x (1-p)^{1-x} \tag{1.20}$$

An example of a continuous probability distribution is the *Gaussian* or *normal* distribution, the probability density function of which is defined as:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{1.21}$$

Both distributions have parameters, the probability of success *p* for the Bernoulli distribution and the *mean μ* and *standard deviation σ* for the normal distribution. In general, the values of these parameters are not known and have to be estimated from data $x_1, ..., x_n$. *Maximum likelihood* gives a general method for doing exactly this by maximizing the joint probability density, also called the likelihood. For the Bernoulli distribution (1.20) the *likelihood* of *p* with respect to $x_1, ..., x_n$ is (assuming the data to be independent (1.18)):

$$\begin{aligned} P(X_1 = x_1, ..., X_n = x_n \mid p) &= \prod_{i=1}^{n} P(x_i \mid p) = p^{x_1}(1-p)^{1-x_1} ... p^{x_n}(1-p)^{1-x_n} \\ &= p^{n_1}(1-p)^{n-n_1} \end{aligned},$$

where $n_1$ is the number of successes. The likelihood is a function of parameter *p* which can be maximized by taking the derivative with respect to *p* and solving the equation:

$$\frac{d(p^{n_1}(1-p)^{n-n_1})}{dp} = 0.$$

Some algebra leads to the estimate $p = (n_1 / n)$, that is, the probability of success is the number of successes among the outcomes divided by the total number of outcomes, which makes sense. The maximum likelihood estimates for the parameters of the normal distribution are the sample mean and the sample variance. In general, the maximum likelihood solution cannot be found by analytical means but require iterative procedures such as those discussed in the chapter on local optimization.

### 1.9.3. Entropy

Information theory is the discipline analyzing the related concepts of information content, compression, communication, and coding. Information theory is based on the following basic definitions.[14]

**Information content** is a measure of the number of bits needed to encode the outcome of $x = a_i$ from possible values $\{a_1, a_2, ..., a_N\}$:

$$h(x = a_i) = \log_2 \frac{1}{p_i} .$$

Intuitively this makes sense, which can be seen by looking at an extreme case. If one outcome $a_i$ occurs with a probability of one and all others with probability zero, the most compact encoding would be to code outcome $a_i$ with zero bits and all other outcomes with infinitely many bits. This can also be interpreted as outcome $a_i$ giving no extra information at all, since we already knew the outcome in advance.

**Entropy** measures the average information content, that is, the *expected* number of bits needed for encoding a set of possible values $\{a_1, a_2, ..., a_N\}$:

$$H_N(x) = \sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i}$$

In the case of just two outcomes, this boils down to the binary entropy function

$$H_2(x) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)}$$

shown in Figure 13.[n] Indeed, as you would expect, the maximum of the entropy function occurs if both outcomes are equally probable and the minimum if one of the outcomes has a probability of one.

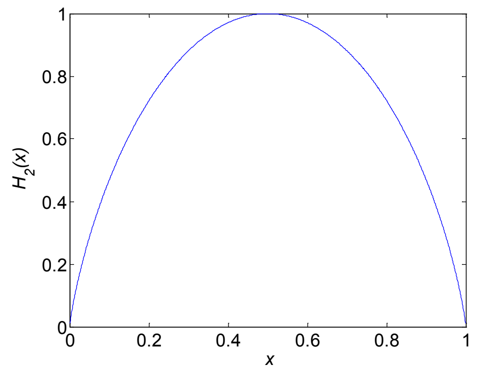


**Figure 13.** Binary entropy function.

**Jensen's inequality** A function is convex over an interval $(a,b)$ if for all $x_1, x_2 \in (a,b)$ and $0 \le \lambda \le 1$,

$$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda) f(x_2) .$$

This is the formal definition of convexity but you mainly need to remember that a convex function is $\cup$-shaped. Examples of such functions are $x^2, e^x$, and $\log(1/x)$.[o]

[n] $0 \times \log_2(1/0) = 0$ and $0 \times \log_2(0) = 0$.

For a convex function $f$, Jensen's inequality is often useful:

$$\sum_{i=1}^{N} p_i f(x_i) \geq f\left(\sum_{i=1}^{N} p_i x_i\right),$$

with equality only if *P* is uniform, that is, $p_i = 1/N$.

**Relative entropy** Until now we only defined the entropy of one probability distribution, but often we would like to compare two distributions over the same alphabet. This concept is captured by the relative entropy or *Kullback-Leibler* divergence:

$$D_{KL}(P \| Q) = \sum_{i=1}^{N} p_i \log \frac{p_i}{q_i}. \tag{1.23}$$

The relative entropy satisfies *Gibbs' inequality*

$$D_{KL}(P \| Q) \geq 0 \tag{1.24}$$

with equality only if *P=Q*. The relative entropy measures how different the distributions *P* and *Q* are.

**Proof** Gibbs' inequality can be proven using Jensen's inequality. Choose $f(y) = \log(1/y)$ as convex function and $y = p_i / q_i$. Then

$$\begin{aligned} D_{KL}(P \| Q) &= \sum_{i=1}^{N} p_i \log \frac{p_i}{q_i} = \sum_{i=1}^{N} p_i f\left(\frac{q_i}{p_i}\right) \\ &\geq f\left(\sum_{i=1}^{N} p_i \frac{q_i}{p_i}\right) = \log\left(\frac{1}{\sum_i q_i}\right) = \log(1) = 0 \end{aligned}$$

with equality only if *P/Q* is uniform, that is, if *P=Q*. □

### 1.9.4. Lagrange multipliers

Consider the problem of finding extremes of a function $f(x_1, x_2)$ subject to a particular constraint of the form $g(x_1, x_2) = 0$. A general technique for dealing with such problems is the introduction of Lagrange multipliers.

From a geometric perspective, the constraint $g(x_1, x_2) = 0$ is a curve (Figure 14). From the previous chapter on math review, we know that $\nabla g$ is perpendicular to the constraint curve. It so happens that at an extremum of *f,* $\nabla f$ points in the same direction as $\nabla g$:

$$\nabla f = \lambda \nabla g, \tag{1.25}$$

where $\lambda$ is called a Lagrange multiplier.

[o] From now on, we will just write log instead of $\log_2$.

A simple example illustrates that this is all we need to find extremes of a function subject to a constraint. Define $f(x_1,x_2)=x_1x_2$ and the constraint $g(x_1,x_2)=x_1+x_2-1=0$, then Eq. (1.25) gives the following linear system:

$$\frac{\partial f}{\partial x_1}=\frac{\partial \lambda g}{\partial x_1}\equiv x_2=\lambda$$

$$\frac{\partial f}{\partial x_2}=\frac{\partial \lambda g}{\partial x_2}\equiv x_1=\lambda\,.$$

Together with constraint $x_1+x_2-1=0$, we have three equations and three unknowns and solving these three equations gives the maximum $(x_1,x_1)=\left(\tfrac{1}{2},\tfrac{1}{2}\right)$.

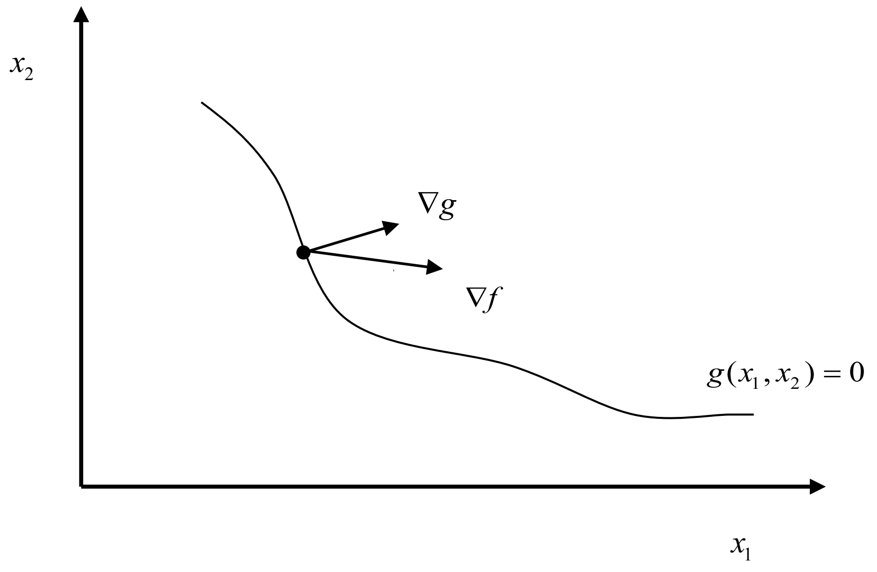


**Figure 14.** Geometrical picture of the main ingredients of Lagrange multipliers [From Ref. 1].

## 1.9.5. Expectation-Maximization Algorithm

The Expectation-Maximization (EM) algorithm is a general approach for dealing with maximum likelihood estimation in the presence of hidden variables. We start out with the log-likelihood with respect to the data $D=\{x_1, \ldots, x_n\}$ and introduce an arbitrary distribution $q$ over the hidden variables $z$ through the sum rule (Eq. (1.15)):

$$\log p(D)=\sum_x \log p(x)=\sum_x\sum_z q(z)\,\log p(x)\,.$$

Now using the product rule (Eq. (1.16)):

$$\begin{aligned}
&= \sum_z q(z) \log \frac{p(x,z)}{p(z \mid x)} = \sum_z q(z) \log \left( \frac{p(x,z)}{p(z \mid x)} \times \frac{q(z)}{q(z)} \right) \\
&= \sum_z q(z) \log \left( \frac{p(x,z)}{q(z)} \right) + \sum_z q(z) \log \left( \frac{q(z)}{p(z \mid x)} \right) \qquad \textbf{(1.26)} \\
&= F(p_{\text{joint}}, q) + D_{KL}(q \,\|\, p_{\text{post}})
\end{aligned}$$

where in the last step we used the definition of relative entropy (1.23) and defined $p_{\text{joint}}$ and $p_{\text{post}}$. As all of you who studied statistical physics already guessed, the first term is known as the *free energy*, which is why it has been denoted as $F$.
What is all this formula juggling good for? It provides us with a simple two-step iterative procedure to maximize the likelihood ($\theta$ denotes the parameters of the model).

**EM Algorithm**:[p]

```
loop
{$\log(p(x \mid \theta)) = F(p_{\text{joint}}, q) + D_{KL}(q \,\|\, p_{\text{post}})$}
E-step:   $q^{\text{new}}(z \mid x) = p_{\text{post}} = p(z \mid x)$
{$\log(p(x \mid \theta)) = F(p_{\text{joint}}, q^{\text{new}})$ since $D_{KL}(q^{\text{new}} \,\|\, p_{\text{post}}) = 0$}
M-step:  maximize $F(p_{\text{joint}}, q^{\text{new}})$ with respect to the parameters $\theta$ of the model
{$\log(p(x \mid \theta^{\text{new}})) \geq F(p_{\text{joint}}, q^{\text{new}}) \geq \log(p(x \mid \theta))$}
end
```

Each iteration is guaranteed to increase the likelihood unless it is already at a local maximum. The algorithm is illustrated in Figure 15.

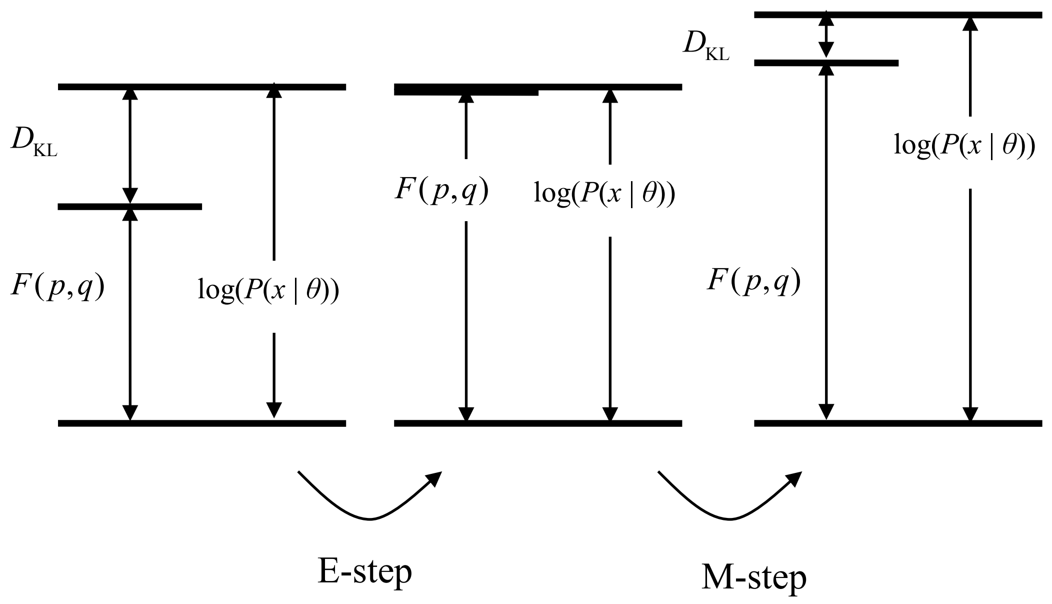


**Figure 15.** Expectation-Maximization algorithm for maximum likelihood estimation.

[p] Text between curly brackets {} in the pseudo-code denotes comments/annotation.

The M-step is often reformulated; maximizing $F(p_{\text{joint}},q^{\text{new}})$ with respect to the parameters $\theta$ of the model can be written as

$$\begin{aligned}\theta^{\text{new}} &:= \arg\max_{\theta} F(p_{\text{joint}}, q^{\text{new}} \mid \theta) \\ &= \arg\max_{\theta} \sum_{x,z} p(z \mid x) \log \left( \frac{p(x,z \mid \theta)}{p(z \mid x)} \right). \\ &= \arg\max_{\theta} \sum_{x,z} p(z \mid x) \log p(x,z \mid \theta)\end{aligned}$$

Using this rewrite the EM algorithm consists of iterating:

E-step: $p_{\text{post}} = p(z \mid x,\theta)$, that is, calculate the distribution of the hidden variables given the data and the model parameters.

M-step: $\arg\max_{\theta} \sum_{x,z} p(z \mid x) \log p(x,z \mid \theta)$, that is, maximize the expected (with respect to hidden variables) log-likelihood of the complete data.

**Example** A very simple model with one hidden variable is a 2-component *mixture* model which is the weighted sum of two probability density functions:

$$p(x) = m_1 p_1(x \mid \theta) + m_2 p_2(x \mid \theta).$$

The mixing coefficients $m_1$ and $m_2$ sum to one and describe the probability that data is generated from mixture component $p_1$ and $p_2$, respectively. Estimating values for parameters $\theta$, $m_1$, and $m_2$ given data $D$ would be easy if for each data point, we also had a label indicating to which of the two mixture components it belongs. Maximum likelihood estimation can then be done for each component separately for "its" part of the data. However, we do not have such a label, this is the hidden variable $z \in \{1,2\}$ of the mixture model. These labels being hidden make it a suitable problem for the EM algorithm. The E-step for component $j \in \{1,2\}$ and data point $x$ can then be written using Bayes' rule as:

$$p(z = j \mid x,\theta) = \frac{p(z = j \mid \theta) p(x \mid z = j,\theta)}{p(x \mid \theta)} = \frac{m_j p_j(x \mid \theta)}{p(x)}.$$

This can be interpreted as a sort of "responsibility" that each component takes for a data point. Using these responsibilities the M-step tries to find new estimates for the parameters $\theta$ by maximizing:

$$\sum_{x,z \in \{1,2\}} p(z \mid x) \log p(x,z \mid \theta).$$

For many probability density functions this is relatively straightforward since in this expression the label $z$ is not hidden anymore. See, for example, Ref. 5 for a detailed derivation of the parameter estimates for a mixture model of Gaussian component densities.