Chapter 9

Probabilistic Graphical Models in Bioinformatics

9.1 The Zoo of Graphical Models in Bioinformatics

High-dimensional probability distributions are one of the fist obstacles one encounters when applying the Bayesian framework to typical real-life problems. This is because the data is high-dimensional, and so are the models we use, often with many thousand parameters and up. High-dimensionality comes also with other so called hidden variables. In general, the resulting global distribution $\mathbf{P}(D, M, H)$ is mathematically intractable and this is where the theory of graphical models comes into play. Using the fact that to a large extent the bulk of the dependencies in the real world are usually local, the high-dimensional distribution is approximated by a product of distributions over smaller clusters of variables defined over smaller spaces and which are tractable [348, 292]. In standard Markovian models, for instance, phenomena at time $t + 1$ may be linked to the past only through what happens in the present at time t. As a result, the global probability distribution $\mathbf{P}(X_1, \ldots, X_N)$ can be factored as a product of local probability distributions of the form $\mathbf{P}(X_{t+1}|X_t)$.

To be more specific, let us concentrate on a particular class of graphical models, namely Bayesian networks [416] (a more formal treatment of graphical models is given in appendix C). A Bayesian network consists of a directed acyclic graph with N nodes. To each node i is associated a random variable X_i. The parameters of the model are the local conditional probabilities, or characteristics, of each random variable given the random variables associated with the parent nodes $\mathbf{P}(X_i|X_j : j \in N^-(i))$, where $N^-(i)$ denotes all the parents of vertex i. The"Markovian" independence assumptions of a Bayesian network

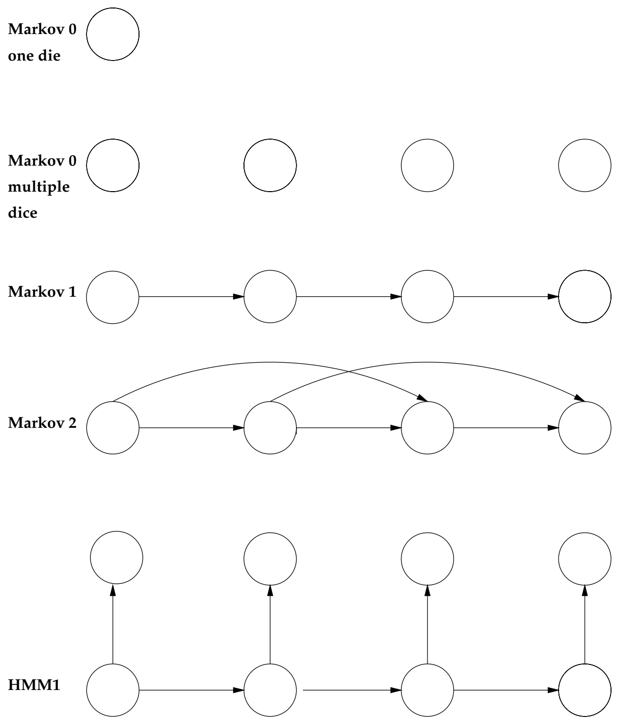

Figure 9.1: Bayesian Network Representation of Markov Models of Increasing Complexity. Markov models of order 0 correspond to a single die or a collection of independent dice. Markov models of order 1 correspond to the standard notion of first order Markov chain. In Markov models of order 2, the present depends on the two previous time steps. All HMMs of order 1 have the same Bayesian network representation given here.

are equivalent to the global factorization property

$$\mathbf{P}(X_1,\ldots,X_N) = \prod_i \mathbf{P}(X_i|X_j : j \in N^-(i)). \tag{9.1}$$

In other words, the global probability distribution is the product of all the local characteristics. In practical applications, the directed nature of the edges of a Bayesian network is used to represent causality or temporal succession. Thus it should come as no surprise that Bayesian networks are being intensively used to model biological sequences, in the same way as they have been used to model speech or other sequential domains, and to construct expert systems.

In fact, the Bayesian framework allows us to build an increasingly complex suite of Bayesian network models for biological (and other) sequences. This hi-

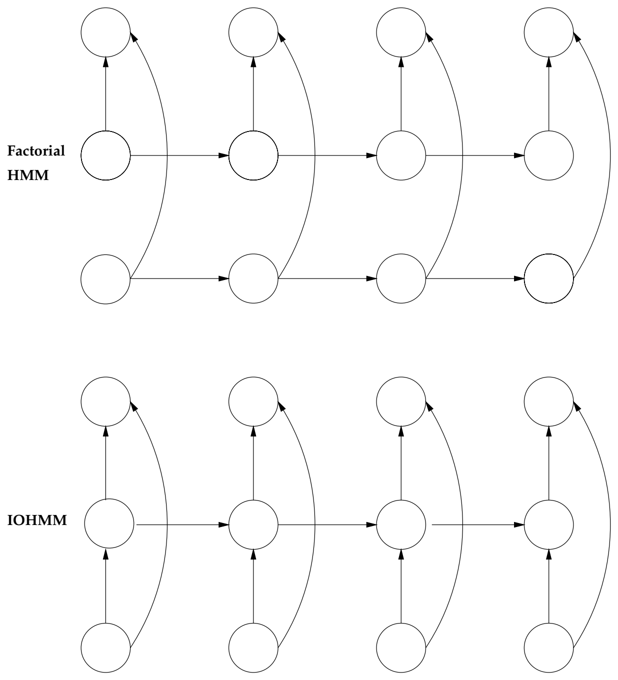

Figure 9.2: Bayesian Network Representation of Factorial HMMs and IOHMMs.

erarchy of models stems from the fact that, at some level, biological sequences have a sequential primary structure. The simplest probabilistic model for biological sequences we can think of is the single-die model of chapter 3, with four (nucleotides for DNA) or 20 (amino acid for proteins) faces, shown in figure 3.1. Such a model is represented by a Bayesian network with a single node or better with multiple identical disconnected nodes, one for each position in a sequence or in a family of sequences. The die model is trivial and remote from actual biological sequences but it serves as a first step and is often used as a background model against which to compare more sophisticated approaches.

At the next level, we can imagine a sequence of distinct dice, one for each position. This is essentially the model used when making profiles, abstracted for instance from pre-existing multiple alignments. If we connect the nodes of this model in a left-right chain, we get a standard first-order Markov model. Second- and higher-order Markov models, where the present may depend on several steps in the immediate past, are also possible. Their Bayesian network representation is obvious as well as their main weakness: a combinatorial ex-

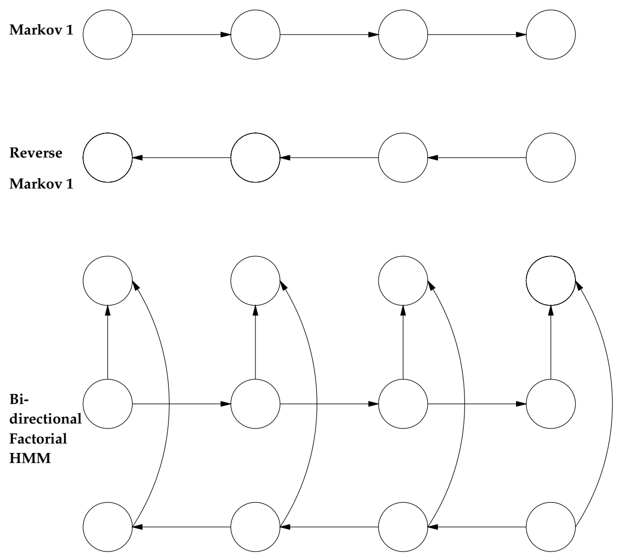

Figure 9.3: Bayesian Networks with Backward Markov Chains. All the backward chains in this figure can be replaced by forward chains via a simple change of variables.

plosion of the parameter space as the degree of the chain increases. For a small alphabet size such as DNA, however, Markov models of order up to six are possible and are commonly used in the literature, for instance in gene finding algorithms (see figure 9.1).

Simple left-right Markov models, however, do not directly capture insertions and deletions. We have seen that such events can be taken into account by using hidden Markov models (HMMs). HMMs can easily be represented as Bayesian networks. As such, their representation is similar to that of other models, such as Kalman filters. The Bayesian network representation of HMMs clarifies their probabilistic structures and the corresponding evidence propagation and learning algorithms, such as the well-known forward-backward algorithms and several other EM/gradient-descent variants [493].

More complex Markovian models have been used in artificial intelligence, for instance, factorial HMMs where the output depends on two or more forward Markov chains. In the speech domain, for instance, one chain can represent audio information and the other video information about lip configuration [203, 205]. Another set of models described in [40, 58] and discussed in a later section are the IOHMMs (input-output HMMs) (see figure 9.2). These models can be used to translate a given input sequence into an output sequence over

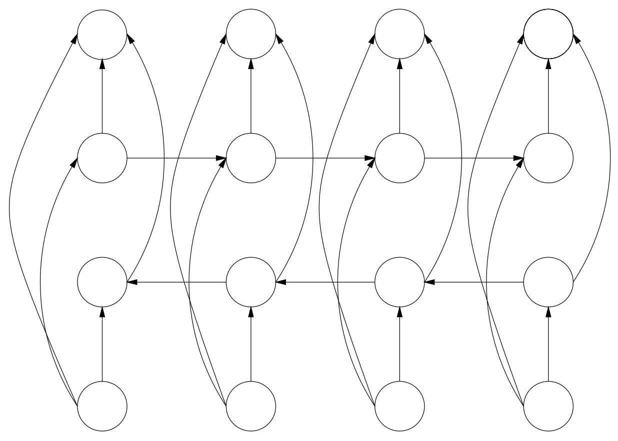

Figure 9.4: Bayesian Network Representation of a BIOHMM. Note the presence of numerous undirected cycles.

a possibly different alphabet.

One important observation about biological sequences is that in reality they have a spatial rather than temporal structure. In particular, information from the "future" could be used to interpret the present without breaking any causality constraint. As a minimum, this suggests introducing backward Markovian chains in the previous models. Yet one must be careful, for it is easy to show that a simple backward Markov chain is entirely equivalent to a forward chain by a change of variables. The parameters of the two corresponding Bayesian network models are related by Bayes's rule. Likewise, if we reverse the direction of one of the chains of a factorial HMM, we obtain another factorial HMM that is entirely identical to the first one, and hence there is little to be gained (see figure 9.3). If we introduce a backward chain in an IOHMM, however, we obtain a new class of models we call BIOHMM (bi-directional IOHMM) [36] (see figure 9.4).

In the last section of this chapter, we will look at the applications of BIOHMMs and related models to the prediction of protein secondary structure. But first, we turn to other applications of probabilistic graphical models to sequence analysis problems, and in particular to DNA symmetries, gene finding, and gene parsing, and to general techniques for combining artificial NNs with graphical models.

9.2 Markov Models and DNA Symmetries

In a piece of double-helical DNA, the number of As is equal to the number of Ts, and the number of Cs is equal to the number of Gs. What appears today as a trivial property in fact was essential in guiding Watson and Crick towards the discovery of the double-helix model in the early 1950s. This property is also known as Chargaff's first parity rule [119]. Chargaff's second parity rule, however, is less known and states that the same thing is approximately true for a piece of *single-stranded* DNA of reasonable size. This rule, first stated in the 1960s [303, 120], has received some recognition in the recent years [430, 185, 231].

The validity of Chargaff's second parity rule can be studied across different organisms, across different kinds of DNA such as coding versus non-coding, and across different length scales. For simplicity, here we look only at genomic DNA in yeast. If we measure the DNA composition of the W and C strands of each chromosome of yeast we find that this composition is remarkably stable and follows Chargaff's second parity rule with approximately 30% for A and T, and 20% for C and G (table 9.1). Notably, the same symmetry is observed in yeast mitochondrial DNA but with a different composition. Likewise, single-stranded genomic DNA in other organisms has a different but still symmetric average composition.

To study the symmetries of double-stranded DNA we count how often each nucleotide occurs on each strand over a given length. These frequencies correspond to a probabilistic Markov model of order 1. It is then natural also to look at Markov models of higher orders (order N) by looking at the statistics of the corresponding N-mers. In particular, we can ask whether Chargaff's second parity rule holds for orders beyond the first, for instance for dinucleotides, equivalent to second-order Markov models.

A DNA Markov model of order N has 4^N parameters associated with the transition probabilities $P(X_N|X_1,\dots,X_{N-1})$, also denoted $P(X_1,\dots,X_{N-1} \rightarrow X_N)$, for all possible $X_1,\dots,X_N$ in the alphabet together with a starting distribution of the form $\pi(X_1,\dots,X_{N-1})$. Because the number of parameters grows exponentially, only models up to a certain order can be determined from a finite data set. A DNA Markov model of order 5, for instance, has 1,024 parameters and a DNA Markov model of order 10 has slightly over one million parameters. Conversely, the higher the order, the larger the data set needed to properly fit the model.

Because of the complementarity between the strands, a Markov model of order N of one strand immediately defines a Markov model of order N on the reverse complement. We say that a Markov model of order N is *symmetric* if it is identical to the Markov model of order N of the reverse complement. Thus a Markov model is symmetric if and only if $P(X_1 \dots X_N) = P(\overline{X_N} \dots \overline{X_1})$.

	A	C	G	T	Total bp
Chr. 1	69,830	44,641	45,763	69,969	230,203
	30.33%	19.39%	19.88%	30.39%	
Chr. 2	249,640	157,415	154,385	251,700	813,140
	30.70%	19.36%	18.99%	30.95%	
Chr. 3	98,210	62,129	59,441	95,559	315,339
	31.14%	19.70%	18.85%	30.30%	
Chr. 4	476,752	289,343	291,354	474,480	1,531,929
	31.12%	18.89%	19.02%	30.97%	
Chr. 5	176,531	109,828	112,314	178,197	576,870
	30.60%	19.04%	19.47%	30.89%	
Chr. 6	82,928	52,201	52,435	82,584	270,148
	30.70%	19.32%	19.41%	30.57%	
Chr. 7	338,319	207,764	207,450	337,403	1,090,936
	31.01%	19.04%	19.02%	30.93%	
Chr. 8	174,022	109,094	107,486	172,036	562,638
	30.93%	19.39%	19.10%	30.58%	
Chr. 9	134,340	85,461	85,661	134,423	439,885
	30.54%	19.43%	19.47%	30.56%	
Chr. 10	231,097	142,211	143,803	228,329	745440
	31.00%	19.08%	19.29%	30.63%	
Chr. 11	206,055	127,713	126,005	206,672	666,445
	30.92%	19.16%	18.91%	31.01%	
Chr. 12	330,586	207,777	207,064	332,745	1,078,172
	30.66%	19.27%	19.21%	30.86%	
Chr. 13	286,296	176,735	176,433	284,966	924,430
	30.97%	19.12%	19.09%	30.83%	
Chr. 14	241,561	151,651	151,388	239,728	784,328
	30.80%	19.34%	19.30%	30.56%	
Chr. 15	339,396	209,022	207,416	335,449	1,091,283
	31.10%	19.15%	19.01%	30.74%	
Chr. 16	293,947	180,364	180,507	293,243	948,061
	31.01%	19.02%	19.04%	30.93%	
Chr. mt	36,169	6,863	7,813	34,934	85,779
	42.17%	8.00%	9.11%	40.73%	
16 nuclear Chr.	3,729,510	2,313,349	2,308,905	3,717,483	12,069,247
	30.90%	19.17%	19.13%	30.80%	
All Chr.	3,765,679	2,320,212	2,316,718	3,752,417	12,155,026
	30.98%	19.09%	19.06%	30.87%	

Table 9.1: First-order Distribution of Yeast Genomic and Mitochondrial DNA per Chromosome.

A → A	0.3643	AA	0.1154
A → T	0.2806	AT	0.0889
A → G	0.1858	AG	0.0589
A → C	0.1684	AC	0.0533
T → A	0.2602	TA	0.0814
T → T	0.3662	TT	0.1146
T → G	0.1858	TG	0.0581
T → C	0.1882	TC	0.0589
G → A	0.3166	GA	0.0581
G → T	0.2784	GT	0.0511
G → G	0.1945	GG	0.0357
G → C	0.2106	GC	0.0387
C → A	0.3304	CA	0.0619
C → T	0.3116	CT	0.0583
C → G	0.1639	CG	0.0307
C → C	0.1941	CC	0.0364

Table 9.2: Second-order Transition Parameters and Dinucleotide Distribution of Yeast 500 bp Upstream Regions.

If we look at genomic DNA in yeast, for instance, we find a very high degree of symmetry in all the higher-order Markov models with orders up to at least 9, even within various subregions of DNA (table 9.2). Some have suggested that this symmetry could easily be explained from the first-order symmetry. Indeed, if $\mathbf{P}(A) = \mathbf{P}(T)$ *and if* $\mathbf{P}(AA) = \mathbf{P}(A)\mathbf{P}(A)$ then automatically $\mathbf{P}(AA) = \mathbf{P}(TT)$. The question then is precisely whether the higher order Markov models are *factorial*, i.e., entirely determined by the products resulting from the lower-order models.

More formally, a Markov model of order N induces a distribution over lower-order M-mers called the restriction or projection of the orginal distribution. This projection is easily obtained for instance by generating a long string with the Markov model of order N and measuring the statistics of the M-mers. In particular, a Markov model of order N induces a first-order equilibrium distribution that must satisfy the balance equation

$$\begin{aligned} P(X_2,\ldots,X_N) \quad &= \quad \sum_Y P(X_N|Y,X_2\ldots,X_{N-1}) \\ &\quad\quad P(Y,X_2,\ldots,X_{N-1}) \end{aligned} \tag{9.2}$$

If a Markov model of order N is symmetric, its restrictions or projections to lower orders are also symmetric. The converse, however is not true. In gen-

	2	3	4	5	6	7	8	9
0	1.0	.99	.99	.99	.99	.99	.97	.95
1	.98	.97	.97	.97	.95	.90	.77	.55
2		.94	.95	.94	.91	.83	.66	.45
3			.97	.94	.89	.77	.57	.36
4				.82	.73	.58	.39	.24
5					.60	.46	.29	.18
6						.34	.21	.14
7							.12	.10
8								.09

Table 9.3: Counts and Symmetry Effects. Row 0 represents the correlation for the counts C of N-mers ($N = 2, \ldots, 9$) between the direct upstream strand and its reverse complement. In rows $M = 1$ to 9, similar correlations are computed but using the ratio $C/E(C)$, where $E(C)$ is the *expected* number of counts produced by a Markov model of order M fitted to the upstream regions. Horizontal = N-mers, vertical = model order.

eral, a symmetric Markov model of order N can have multiple not necessarily symmetric extensions to a Markov model of order M, $M > N$. Thus the fact that the first-order distribution of yeast, for instance, is symmetric does not necessarily imply that the second order distribution is also symmetric. But this is precisely the case. A given Markov model of order N, however, has a unique *factorial extension* to Markov models of order $M > N$. For instance, a first-order Markov model defined by the parameters p_X (p_A, p_C, p_G, p_T) has a second-order factorial extension with parameters $p_{XY} = p_X p_Y$.

For a given Markov model of order N, we can factor out the symmetry effects due to any Markov model of lower order M. For each N-mer and its reverse complement we can get the ratio (or the difference) between the number of expected counts according to the model of order N and to the model of order M used factorially. The residual symmetry can be measured by looking at the correlation of the ratios between N-mers and their respective reverse complements. If we use this approach in yeast, we find a considerable amount of residual symmetry in the higher-order models that cannot be entirely explained, for instance, by the symmetry of the first-order composition (table 9.3).

Thus higher-order Markov models allow us to study Chargaff's second parity rule in great detail. Chargaff's second parity rule is of course not true locally, and it is also violated in some viral genomes. There are also well-known compositional biases around the origin of replication in prokaryotic genomes. But by and large it is remarkably valid and probably results from a complex mixture of influences operating at different scales. It is clear that because of

DNA	W ORFs	C ORFs	Total
Chr. 1	56	51	107
Chr. 2	200	226	426
Chr. 3	75	99	174
Chr. 4	400	419	819
Chr. 5	146	141	287
Chr. 6	67	67	134
Chr. 7	298	273	571
Chr. 8	153	131	284
Chr. 9	106	118	224
Chr. 10	201	186	387
Chr. 11	175	161	336
Chr. 12	261	286	547
Chr. 13	246	244	490
Chr. 14	219	201	420
Chr. 15	295	278	573
Chr. 16	256	244	500
Total	3154	3125	6279

Table 9.4: Number of ORFs of Length Greater than 100 per Strand and per Chromosome in Yeast, Excluding tRNA and rRNA Genes. The total excludes the mitochondrial chromosome.

Chargaff's first parity rule, any force operating on DNA that does not distinguish between the two strands will contribute to Chargaff's second parity rule. Mutations induced for instance by radiation are likely to fall in this class. Likewise, the replication machinery of the cell must be optimized for producing the same number of complementary base pairs, and this also should favor the first-order version of Chargaff's second parity rule. Other effects under study may be more long-ranged, such as the approximately symmetric distribution of genes on each strand (table 9.4). This distribution could also be modeled using probabilistic Markov models.

9.3 Markov Models and Gene Finders

One the most important applications of Markov and graphical models to sequence analysis has been the construction of various gene finders and gene parsers such as GeneMark and GeneMark.hmm [81, 82, 367], GLIMMER [461], GRAIL [529], GenScan [107] and now GenomeScan, and Genie [441]. Our goal here is not to give an exhaustive list of all gene finders, nor to describe each one of them in detail, nor to compare their respective merits and drawbacks,

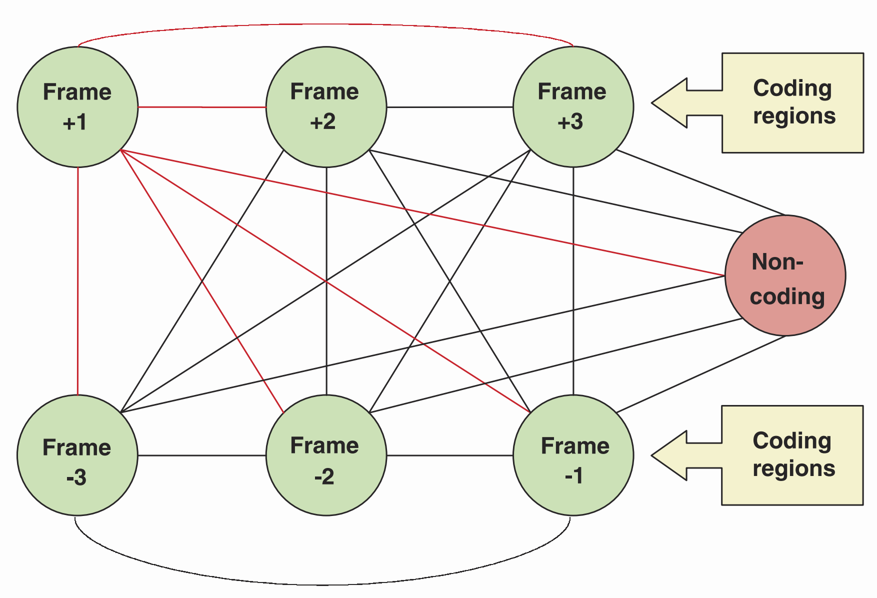

Figure 9.5: Graphical Representation of GeneMark for Prokaryotic Genomes. For prokaryotic genomes, typical high-level modules include modules for coding region and non-coding regions.

but to provide a synthetic overview showing how they can be constructed and understood in terms of probabilistic graphical models.

Integrated gene finders and gene parsers typically have a modular architecture and often share the same basic strategies. They comprise two basic kinds of elementary modules aimed at detecting boundary elements or variable length regions. Examples of boundary modules associated with localized signals include splice sites, start and stop codons, various transcription factor and other protein binding sites (such as the TATA-box), transcription start points, branch points, terminators of transcription, polyadenylation sites, ribosomal binding sites, topoisomerase I cleavage sites, and topoisomerase II binding sites. Region modules instead are usually associated with exons, introns, and intergenic regions. Exon models in turn are often subdivided into initial, internal, and terminal exons due to the well-known statistical dif-

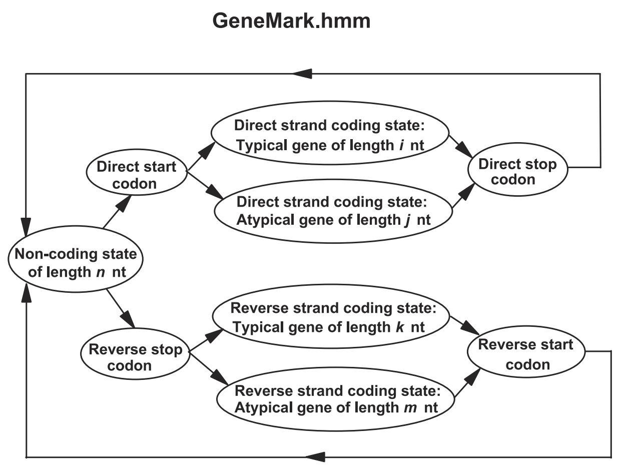
GeneMark.hmm
Direct start codon
Direct strand coding state: Typical gene of length *i* nt
Direct strand coding state: Atypical gene of length *j* nt
Direct stop codon
Non-coding state of length *n* nt
Reverse stop codon
Reverse strand coding state: Typical gene of length *k* nt
Reverse strand coding state: Atypical gene of length *m* nt
Reverse start codon

Figure 9.6: Graphical Representation of GeneMark.hmm for Prokaryotic Genomes.

ferences among these elements. Ultimately, computational models of entire genomes must include also other regions, including various kinds of repetitive regions, such as Alu sequences.

High-level graphical representations of several genefinders are displayed in figures 9.5, 9.6, 9.7, and 9.8. reprinted here with permission from the authors. The high-level representations and the underlying graphical models are of course significantly more complex for eukaryotic gene finders than for prokaryotic ones, due for instance to the presence of exons and introns. It is important to observe that the graphs in these figures do not directly represent Bayesian networks but rather transition state diagrams, in the same way the standard HMM architectures of chapter 7 do not correspond to the Bayesian network representation of HMMs that we saw in the first section of this chapter. In fact, most genefinders can be viewed as HMMs, or variations of HMMs, at least at some level.

The high-level nodes in these graphs represent boundary or region modules. There are some differences between the gene finders in the choice of modules and in how the modules are implemented and trained. In the case of boundary modules, early implementations used simple consensus sequences. These have evolved into profiles or weight matrix schemes, which are special cases of first-order Markov models in which scores are interpreted as log-likelihoods or log ratios, and Markov models. Because the DNA alphabet has only four letters, higher-order Markov models can be used when sufficient training data is available. Neural networks, which algebraically can be viewed

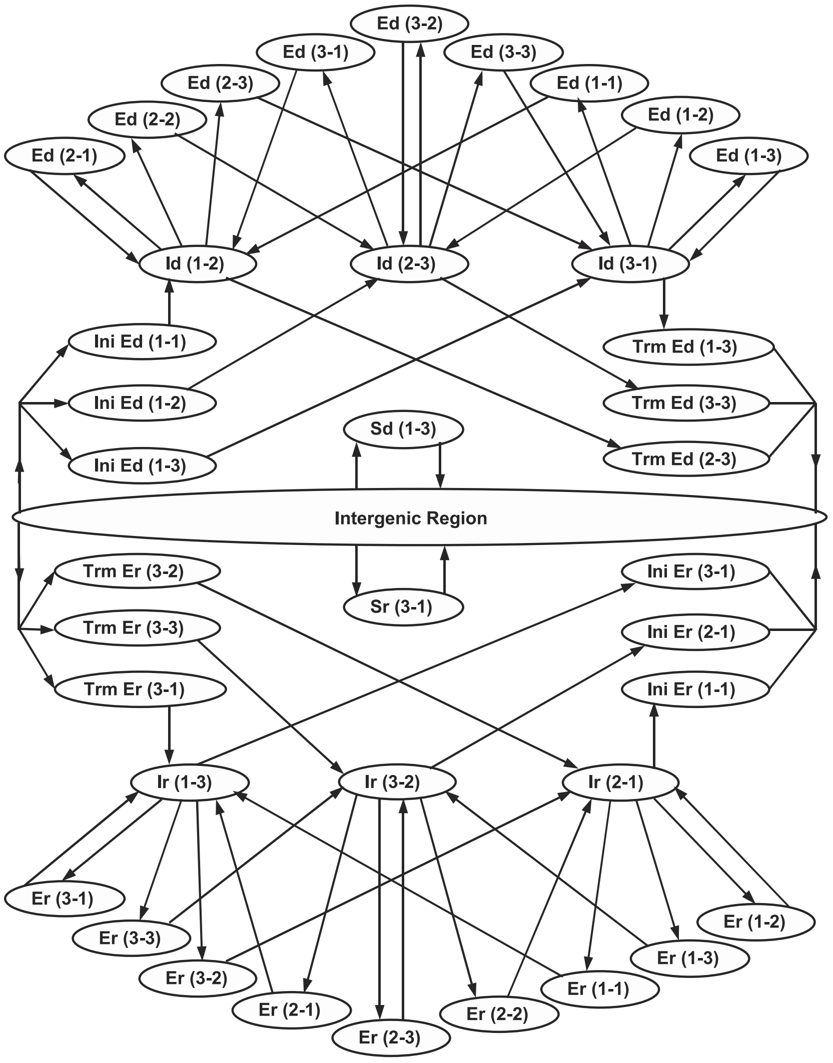

Figure 9.7: Graphical Representation of High-level States of GeneMark.hmm for Eukaryotic Genomes. The model include states corresponding to initial and terminal exons, internal exons, introns, in all reading frames and for the direct and reverse strand.

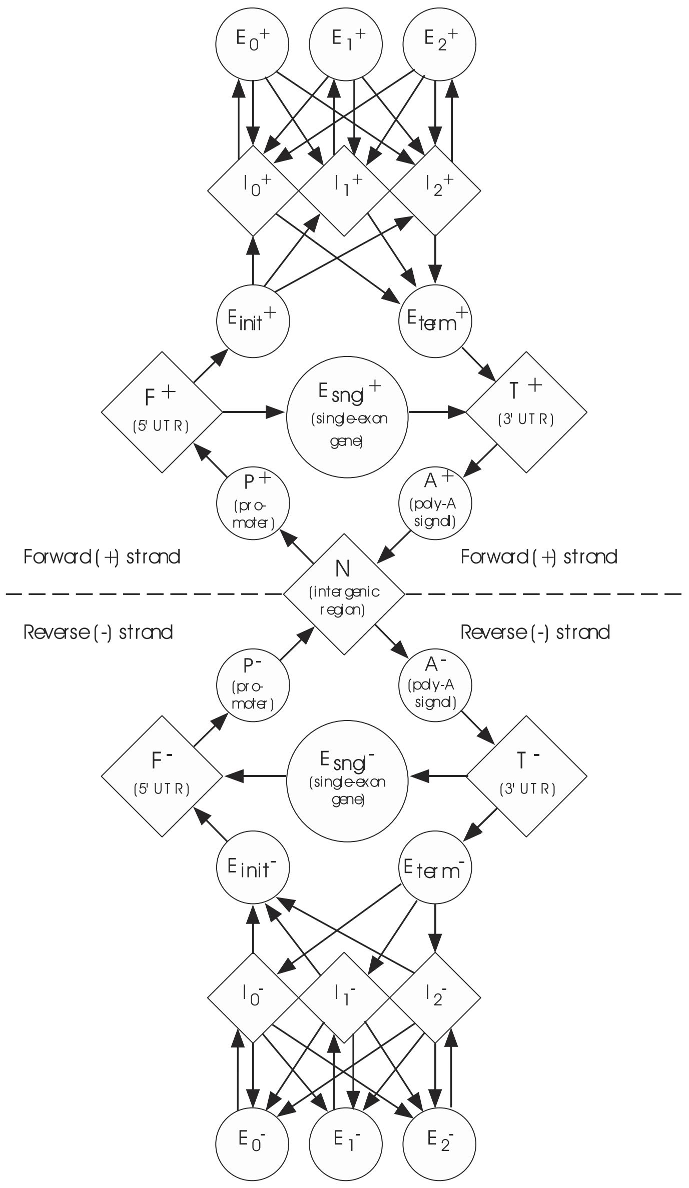

Figure 9.8: Graphical Representation of Hidden States in GenScan. Similar to figure 9.7. Notice the additional states, for instance, for poly-A signals.

as generalization of weight matrices, are also used in some boundary modules.

Variable length regions are usually modeled with Markov models of order up to 6. In particular, coding regions have well known 3- and 6-periodicities that can easily be incorporated into Markov models of order 3 or 6. Exon models must take into account reading frames and knowledge of reading frame must somehow propagate through the intervening sequence to the next exon. The state connectivity can be used to model the different reading frames but also the length distribution of each component, i.e. how long the system ought to remain in each state. Such durations can also be modeled or adjusted using empirical distributions extracted from the available data, or by fitting theoretical distributions to the training data (see also [154]). Because genes can occur on either strand in the 5' to 3' orientation, gene finders must be able to model both situations in mirror fashion. A gene casts a "shadow" on the opposite strand and therefore a single strand can be scanned to find genes located on either strand.

The resulting models can be used to "scan" and parse large genomic regions using dynamic programming and Viterbi paths (maximum likelihood, maximum a posteriori, or even conditional maximum likelihood as in [339]), which, depending on the size of the regions, can be computationally demanding. The "hits" can be further filtered and improved by leveraging the information about coding regions contained in large ESTs and protein databases, including databases of HMM models such as Pfam, using alignments. The parameters of the various boundary or region models can be fitted to different organisms, or even different genomic regions with different compositional biases or different gene classes, resulting in different specialized gene finders and gene parsers.

Although the performance of gene finders is not easy to measure and compare, overall it has significantly improved over the past few years. These programs now play an important role in genome annotation projects. Several significant challenges remain, however, such as creating better models of regulatory regions and of alternative splicing.

9.4 Hybrid Models and Neural Network Parameterization of Graphical Models

9.4.1 The General Framework

In order to overcome the limitations of HMMs, we shall look here at the possibility of combining HMMs and NNs to form hybrid models that contain the expressive power of artificial NNs with the sequential time series aspect of HMMs. In this section we largely follow the derivation in [40]. There are a number of ways in which HMMs and NNs can be combined. Hybrid architectures have been used in both speech and cursive handwriting recognition [84, 126]. In many of these applications, NNs are used as front-end processors to extract features, such as strokes, characters, and phonemes. HMMs are then used in higher processing stages for word and language modeling[1]. The HMM and NN components are often trained separately, although there are some exceptions [57]. In a different type of hybrid architecture, described in [126], the NN component is used to classify the pattern of likelihoods produced by several HMMs. Here, in contrast, we will cover hybrid architectures [40] where the HMM and NN components are inseparable. In these architectures, the NN component is used to reparameterize and modulate the HMM component. Both components are trained using unified algorithms in which the HMM dynamic programming and the NN backpropagation blend together. But before we proceed with the architectural details, it is useful to view the hybrid approach from the general probabilistic standpoint of chapter 2 and of graphical models.

The General Hybrid Framework

From Chapter 2, we know that the fundamental objects we are interested in are probabilistic models $M(\theta)$ of our data, parameterized here by θ. Problems arise, however, whenever there is a mismatch between the complexity of the model and the data. Overly complex models result in overfitting, overly simple models in underfitting.

The general hybrid modeling approach attempts to address both problems. When the model is too complex, it is reparameterized as a function of a simpler parameter vector w, so that $\theta = f(w)$. This is the single-model case. When the data are too complex, the only solution, short of resorting to a different model class, is to model the data with several $M(\theta)$s, with θ varying discretely or continuously as $M(\theta)$ covers different regions of data space. Thus the parameters must be modulated as a function of the input, or context, in the form $\theta = f(I)$. This is the multiple-model case. In the general case, both may be desirable, so that $\theta = f(w, I)$. This approach is hybrid in the sense that the function f can belong to a different model class. Since neural networks have well-known universal approximation properties (see chapter 5), a natural approach is to compute f with an NN, but other representations are possible. This approach is hierarchical because model reparameterizations can easily be nested at several levels. Here, for simplicity, we confine ourselves to a single level of reparameterization.

[1]In molecular biology applications, NNs could conceivably be used to interpret the analog output of various sequencing machines, although this is not our focus here.

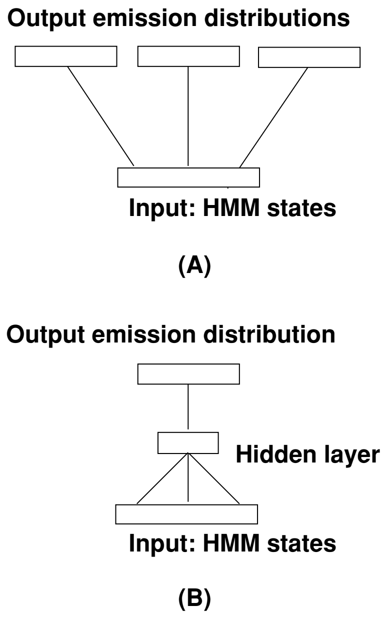

Figure 9.9: From HMM to hybrid HMM/NN. A: Schematic representation of simple HMM/NN hybrid architecture used in [41]. Each HMM state has its own NN. Here, the NNs are extremely simple, with no hidden layer and an output layer of softmax units computing the state emission or transition parameters. For simplicity only output emissions are represented. B: Schematic representation of an HMM/NN architecture where the NNs associated with different states (or different groups of states) are connected via one or several hidden layers.

9.5 The Single-Model Case

Basic Idea

In a general HMM, an emission or transition vector θ is a function of the state i only: $\theta = f(i)$. The first basic idea is to have a NN on top of the HMM for the computation of the HMM parameters, that is, for the computation of the function f. NNs are universal approximators, and therefore can represent any f. More important perhaps, NN representations of the parameters make possible the flexible introduction of many possible constraints. For simplicity, we discuss emission parameters only in a protein context, but the approach extends immediately to transition parameters as well, and to all other alphabets.

In the reparameterization of (7.33), we can consider that each of the HMM emission parameters is calculated by a small NN, with one input set to 1 (bias), no hidden layers, and 20 softmax output units (figure 9.9A). The connections between the input and the outputs are the $w_{i\mathsf{X}}$ parameters. This can be generalized immediately by having arbitrarily complex NNs for the computation of the HMM parameters. The NNs associated with different states can also be linked with one or several common hidden layers, the overall architecture being dictated by the problem at hand (figure 9.9B). In the case of a discrete alphabet, however, such as for proteins, the emission of each state is a multinomial distribution, and therefore the output of the corresponding network should consist of $|A|$ normalized exponential units.

Example

As a simple example, consider the hybrid HMM/NN architecture of figure 9.10, consisting of the following

1. Input layer: one unit for each state i. At each time, all units are set to 0, except one that is set to 1. If unit i is set to 1, the network computes $e_{i\mathsf{X}}$, the emission distribution of state i.
2. Hidden layer: $|H|$ hidden units indexed by h, each with transfer function f_h (logistic by default) and bias b_h ($|H| < |A|$).
3. Output layer: $|A|$ softmax units or normalized exponentials, indexed by X, with bias b_{X}.
4. Connections: $\alpha = (\alpha_{hi}$ connects input position i to hidden unit h). $\beta = (\beta_{\mathsf{X}h}$ connects hidden unit h to output unit X). No confusion with the HMM forward or backward variable should be possible.

For input i, the activity in the hth unit in the hidden layer is given by

$$f_h(\alpha_{hi} + b_h). \tag{9.3}$$

The corresponding activity in the output layer is

$$e_{iX} = \frac{e^{-[\sum_h \beta_{Xh} f_h(\alpha_{hi}+b_h)+b_X]}}{\sum_Y e^{-[\sum_h \beta_{Yh} f_h(\alpha_{hi}+b_h)+b_Y]}}. \tag{9.4}$$

Remarks

For hybrid HMM/NN architectures, a number of points are worth noting:

- The HMM states can be partitioned into groups, with different networks for different groups. In protein applications one can use different NNs for insert states and for main states, or for different groups of states along the protein sequence corresponding to different regions (hydrophobic, hydrophilic, alpha-helical, etc.).

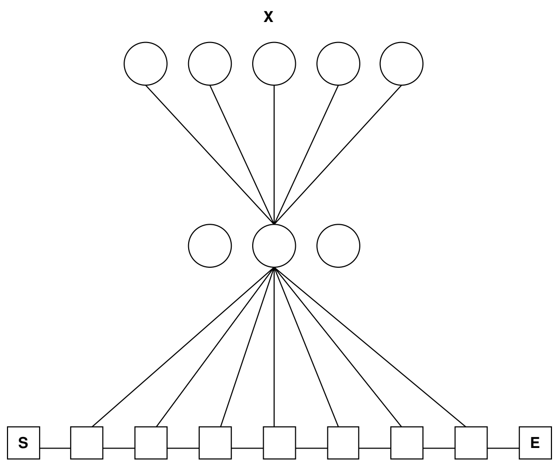

Figure 9.10: Simple Hybrid Architecture. Schematic representation of HMM states. Each state is fully interconnected to the common hidden layer. Each unit in the hidden layer is fully interconnected to each normalized exponential output unit. Each output unit calculates the emission probability e_{iX}.

- HMM parameter reduction can easily be achieved using small hidden layers with $|H|$ hidden units and $|H|$ small compared with N or $|A|$. In figure 9.10, with $|H|$ hidden units and considering only main states, the number of parameters is $|H|(N + |A|)$ in the HMM/NN architecture, versus $N|A|$ in the corresponding simple HMM. For protein models, this yields roughly $|H|N$ parameters for the HMM/NN architecture, versus $20N$ for the simple HMM. $|H| = |A|$ is roughly equivalent to (7.33).

- The number of parameters can be adaptively adjusted to variable training set sizes by changing the number of hidden units. This is useful in environments with large variations in database sizes, as in current molecular biology applications.

- The entire bag of NN techniques—such as radial basis functions, multiple hidden layers, sparse connectivity, weight sharing, Gaussian priors, and hyperparameters—can be brought to bear on these architectures. Several initializations and structures can be implemented in a flexible way. By allocating different numbers of hidden units to different subsets of emissions or transitions, it is easy to favor certain classes of paths in the models when necessary. In the HMM of figure 7.2 one must in general introduce a bias favoring main states over insert states prior to any learning. It is easy also to link different regions of a protein that may have similar properties by weight sharing and other types of long-range correlations. By setting the output bias to the proper values, the model can be initialized to the average composition of the training sequences or any other useful distribution.

- Classical prior information in the form of substitution matrices is easily incorporated. Substitution matrices ([8]; see also chapters 1 and 10) can be computed from databases, and essentially produce a background probability matrix $P = (p_{XY})$, where p_{XY} is the probability that X will be changed into Y over a certain evolutionary time. P can be implemented as a linear transformation in the emission NN.

- Although HMMs with continuous emission distributions are outside the scope of this book, they can also be incorporated in the HMM/NN framework. The output emission distributions can be represented in the form of samples, moments, and/or mixture coefficients. In the classical mixture of Gaussians case, means, covariances, and mixture coefficients can be computed by the NN. Likewise, additional HMM parameters, such as exponential parameters to model the duration of stay in any given state, can be calculated by an NN.

Representation in Simple HMM/NN Architectures

Consider the particular HMM/NN described above (figure 9.10), where a subset of the HMM states is fully connected to $|H|$ hidden units, and the hidden units are fully connected to $|A|$ softmax output units. The hidden unit bias is not really necessary in the sense that for any HMM state i, any vector of biases b_h, and any vector of connections α_{hi}, there is a new vector of connections α'_{hi} that produces the same vector of hidden unit activations with 0 bias. This is not true in the general case—for example, as soon as there are multiple hidden layers, or if the input units are not fully interconnected to the hidden layer. We have retained the biases for the sake of generality, and also because even if they do not enlarge the space of possible representations, they may still facilitate the learning procedure. Similar remarks hold more generally for the transfer functions. With an input layer fully connected to a single hidden layer, the same hidden layer activation can be achieved with different activation functions by modifying the weights.

A natural question to ask is "What is the representation used in the hidden layer, and what is the space of emission distributions achievable in this fashion?" Each HMM state in the network can be represented by a point in the $[-1, 1]^{|H|}$ hypercube. The coordinates of a point are the activities of the $|H|$ hidden units. By changing its connections to the hidden units, an HMM state can occupy any position in the hypercube. Thus, the space of emission distributions that can be achieved is entirely determined by the connections from the hidden layer to the output layer. If these connections are held fixed, then each HMM state can select a corresponding optimal position in the hypercube where its emission distribution, generated by the NN weights, is as close as possible to the truly optimal distribution—for instance, in cross-entropy distance. During on-line learning, all parameters are learned at the same time, so this may introduce additional effects.

Further to understand the space of achievable distributions, consider the transformation from hidden to output units. For notational convenience, we introduce one additional hidden unit numbered 0, always set to 1, to express the output biases in the form $b_{\mathsf{X}} = \beta_{\mathsf{X}0}$. If, in this extended hidden layer, we turn single hidden units to 1, one at a time, we obtain $|H|+1$ different emission distributions in the output layer $P^h = (p^h_{\mathsf{X}})$ $(0 \leq h \leq |H|)$, with

$$p^h_{\mathsf{X}} = \frac{e^{-\beta_{\mathsf{X}h}}}{\sum_{\mathsf{Y}\in A} e^{-\beta_{\mathsf{Y}h}}}. \tag{9.5}$$

Consider now a general pattern of activity in the hidden layer of the form $(1, \mu_1, \ldots, \mu_{|H|})$. By using (9.4) and (9.5), the emission distribution in the output layer is then

$$e_{i\mathsf{X}} = \frac{e^{-\sum_{h=0}{|H|}\beta_{\mathsf{X}h}\mu_h}}{\sum_{\mathsf{Y}\in A} e^{-\sum_{h=0}{|H|}\beta_{\mathsf{Y}h}\mu_h}} = \frac{\prod_{h\in H}[p^h_{\mathsf{X}}]^{\mu_h}[\sum_{\mathsf{Y}\in A} e^{-\beta_{\mathsf{Y}h}}]^{\mu_h}}{\sum_{\mathsf{Y}\in A}\prod_{h\in H}[p^h_{Y}]^{\mu_h}[\sum_{\mathsf{Z}\in A} e^{-\beta_{\mathsf{Z}h}}]^{\mu_h}}. \tag{9.6}$$

After simplification, this yields

$$e_{iX} = \frac{\prod_{h\in H}[p^h_{\mathsf{X}}]^{\mu_h}}{\sum_{\mathsf{Y}\in A}\prod_{h\in H}[p^h_{\mathsf{Y}}]^{\mu_h}}. \tag{9.7}$$

Therefore, all the emission distributions achievable by the NN have the form of (9.7), and can be viewed as "combinations" of $|H|+1$ fundamental distributions P^h associated with each single hidden unit. In general, this combination is different from a convex linear combination of the P^hs. It consists of three operations: (1) raising each component of P^h to the power μ_h, the activity of the hth hidden unit; (2) multiplying all the corresponding vectors componentwise; (3) normalizing. In this form, the hybrid HMM/NN approach is different from a mixture of Dirichlet distributions approach.

Learning

HMM/NN architectures can be optimized according to ML or MAP estimation. Unlike HMMs, for hybrid HMM/NN architectures the M step of the EM algorithm cannot, in general, be carried out analytically. However, one can still use some form of gradient descent using the chain rule, by computing the derivatives of the likelihood function with respect to the HMM parameters, and then the derivatives of the HMM parameters with respect to the NN parameters. The derivatives of the prior term, when present, can easily be incorporated. It is also possible to use a Viterbi learning approach by using only the most likely paths. The derivation of the learning equations is left as an exercise and can also be found in [40]. In the resulting learning equations the HMM dynamic programming and the NN backpropagation components are intimately fused. These algorithms can also be seen as GEM (generalized EM) algorithms [147].

9.5.1 The Multiple-Model Case

The hybrid HMM/NN architectures described above address the first limitation of HMMs: the control of model structure and complexity. No matter how complex the NN component, however, the final model so far remains a single HMM. Therefore the second limitation of HMMs, long-range dependencies, remains. This obstacle cannot be overcome simply by resorting to higher-order HMMs. Most often these are computationally intractable. A possible approach is to try to build Markov models with variable memory length by introducing a new state for each relevant context. This requires a systematic method for determining relevant contexts of variable lengths directly from the data. Furthermore, one must hope the number of relevant contexts remains small. An interesting approach along these lines can be found in [448], where English is modeled as a Markov process with variable memory length of up to ten letters or so.

To address the second limitation without resorting to a different model class, one must consider more general HMM/NN hybrid architectures, where the underlying statistical model is a *set* of HMMs. To see this, consider again the $\mathsf{X}-\mathsf{Y}/\mathsf{X}'-\mathsf{Y}'$ problem mentioned at the end of chapter 8. Capturing such dependencies requires *variable* emission vectors at the corresponding locations, together with a linking mechanism. In this simple case, four different emission vectors are needed: e_i, e_j, e'_i and e'_j. Each one of these vectors must assign a high probability to the letters $\mathsf{X}, \mathsf{Y}, \mathsf{X}'$, and Y', respectively. More important, there must be some kind of memory, that is, a mechanism to link the distributions at i and j, so that e_i and e_j are used for sequence O and e'_i and e'_j are used for sequence O'. The combination of e_i and e'_j (or e'_i and e_j) should be rare or not allowed, unless required by the data. Thus e_i and e_j must belong to a first HMM and e'_i and e'_j to a second HMM, with the possibility of switching from one HMM to the other as a function of input sequence. Alternatively, there must be a single HMM but with variable emission distributions, again modulated by some input.

In both cases, then, we consider that the emission distribution of a given state depends not only on the state itself but also on an additional stream of information I. That is now $\theta = f(i, I)$. Again, in a multiple-HMM/NN hybrid architecture this more complex function f can be computed by an NN. Depending on the problem, the input I can assume different forms, and may be called a "context" or "latent" variable. When feasible, I may even be equal to the currently observed sequence O. Other inputs are possible, however, over different alphabets. An obvious candidate in protein modeling tasks would be the secondary structure of the protein (alpha-helices, beta-sheets and coils). In general, I could also be any other array of numbers representing latent variables for the HMM modulation [374]. We briefly consider two examples.

Mixtures of HMM Experts

A first possible approach is to consider a model M that is a mixture distribution (2.23) of n simpler HMMs' $M_1, \dots, M_n$. For any sequence O, then,

$$\mathbf{P}(O|M) = \sum_{i=1}^{n} \lambda_i \mathbf{P}(O|M_i), \tag{9.8}$$

where the mixture coefficients λ_i satisfy $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$. In generative mode, sequences are produced at random by each individual HMM, and M_i is selected with probability λ_i. Such a system can be viewed as a larger single HMM, with a starting state connected to each one of the HMMs' M_i, with transition probability λ_i (figure 8.5). As we have seen in chapter 8, this type of model is used in [334] for unsupervised classification of globin protein sequences. Note that the parameters of each submodel can be computed by an NN to create an HMM/NN hybrid architecture. Since the HMM experts form a larger single HMM, the corresponding hybrid architecture is identical to what we saw in section 9.2. The only peculiarity is that states have been replicated, or grouped, to form different submodels. One further step is to have variable mixture coefficients that depend on the input sequence or some other relevant information. These mixture coefficients can be computed as softmax outputs of an NN, as in the mixture-of-experts architecture of [277].

Mixtures of Emission Experts

A different approach is to modulate a single HMM by considering that the emission parameters $e_{i\mathsf{X}}$ should also be a function of the additional input I. Thus $e_{i\mathsf{X}} = P(i, \mathsf{X}, I)$. Without any loss of generality, we can assume that P is a

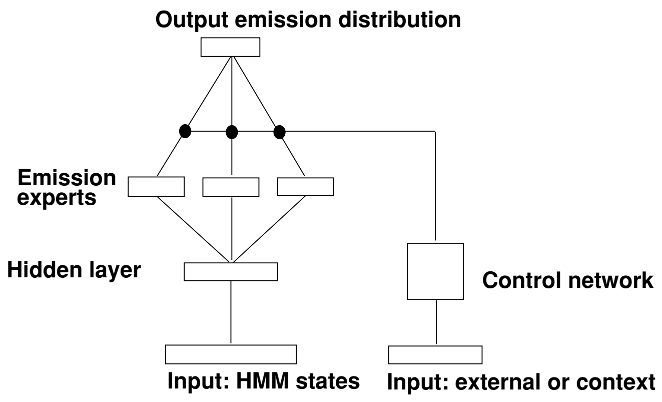

Figure 9.11: Schematic Representation of a General HMM/NN Architecture in Which the HMM Parameters Are Computed by an NN of Arbitrary Complexity That Operates on State Information, but Also on Input or Context. The input or context is used to modulate the HMM parameters, for instance, by switching or mixing different parameter experts. For simplicity, only emission parameters are represented, with three emission experts and a single hidden layer. Connections from the HMM states to the control network, and from the input to the hidden layer, are also possible.

mixture of n emission experts P_j

$$P(i, \mathsf{X}, I) = \sum_{j=1}^{n} \lambda_j(i, \mathsf{X}, I) P_j(i, \mathsf{X}, I). \tag{9.9}$$

In many interesting cases, λ_j is independent of X, resulting in the probability vector equation over the alphabet

$$P(i, I) = \sum_{j=1}^{n} \lambda_j(i, I) P_j(i, I). \tag{9.10}$$

If $n = 1$ and $P(i, I) = P(i)$, we are back to a single HMM. An important special case is derived by further assuming that λ_j does not depend on i, and $P_j(i, \mathsf{X}, I)$ does not depend on I explicitly. Then

$$P(i, I) = \sum_{j=1}^{n} \lambda_j(I) P_j(i). \tag{9.11}$$

This provides a principled way to design the top layers of general hybrid HMM/NN architectures, such as the one depicted in figure 9.11.

The components P_j are computed by an NN, and the mixture coefficients by another gating NN. Naturally, many variations are possible and, in the most general case, the switching network can depend on the state i, and the distributions P_j on the input I. In the case of protein modeling, if the switching depends on position i, the emission experts could correspond to different types of regions, such as hydrophobic and hydrophilic, rather than different subclasses within a protein family.

Learning

For a given setting of all the parameters, a given observation sequence, and a given input vector I, the general HMM/NN hybrid architectures reduce to a single HMM. The likelihood of a sequence, or some other measure of its fitness, with respect to such an HMM can be computed by dynamic programming. As long as it is differentiable in the model parameters, we can then backpropagate the gradient through the NN, including the portion of the network depending on I, such as the control network of figure 9.11. With minor modifications, this leads to learning algorithms similar to those described above. This form of learning encourages cooperation between the emission experts of figure 9.11. As in the usual mixture-of-experts architecture [277], it may be useful to introduce some degree of competition among the experts, so that each of them specializes on a different subclass of sequences.

When the input space has been selected but the value of the relevant input I is not known, it is possible to learn its values together with the model parameters using Bayesian inversion. Consider the case where there is an input I associated with each observation sequence O, and a hybrid model with parameters w, so that we can compute $\mathbf{P}(O|I,w)$. Let $\mathbf{P}(I)$ and $\mathbf{P}(w)$ denote our priors on I and w. Then

$$\mathbf{P}(I|O,w) = \frac{\mathbf{P}(O|I,w)\mathbf{P}(I)}{\mathbf{P}(O|w)}, \tag{9.12}$$

with

$$\mathbf{P}(O|w) = \int_I \mathbf{P}(O|I,w)\mathbf{P}(I)dI. \tag{9.13}$$

The probability of the model parameters, given the data, can then be calculated, again using Bayes's theorem:

$$\mathbf{P}(w|D) = \frac{\mathbf{P}(D|w)\mathbf{P}(w)}{\mathbf{P}(D)} = \frac{[\prod_O \mathbf{P}(O|w)]\mathbf{P}(w)}{\mathbf{P}(D)}, \tag{9.14}$$

assuming the observations are independent. These parameters can be optimized by gradient descent on $-\log \mathbf{P}(w|D)$. The main step is the evaluation

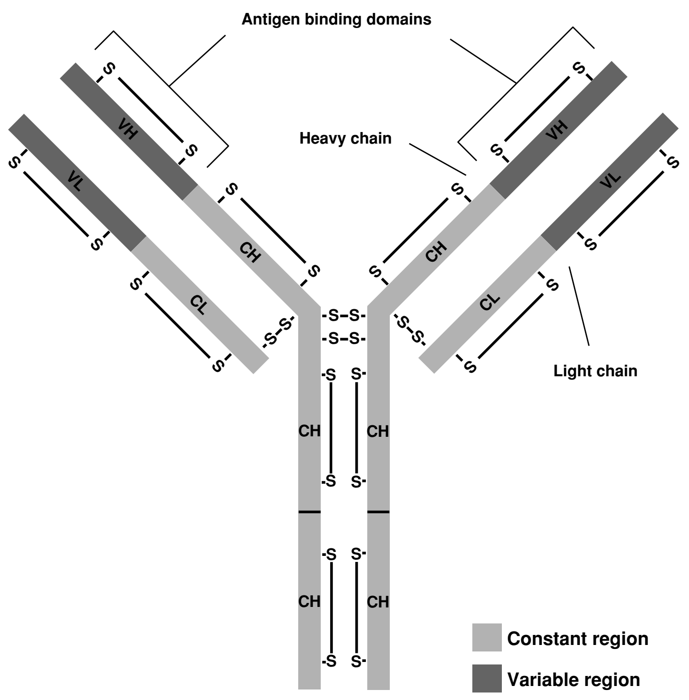

Figure 9.12: A Schematic Model of the Structure of a Typical Human Antibody Molecule Composed of Two Light (L) and Two Heavy (H) Polypeptide Chains. Interchain and intrachain disulfide bonds are indicated (S). Cysteine residues are associated with the bonds. two identical active sites for antigen binding, corresponding to the variable regions, are located in the arms of the molecule.

of the likelihood $\mathbf{P}(O|w)$ and its derivatives with respect to w, which can be done by Monte Carlo sampling. The distribution on the latent variables I is calculated by (9.12). The work in [374] is an example of such a learning approach. The density network used for protein modeling can be viewed essentially as a special case of HMM/NN hybrid architecture where each emission vector acts as a softmax transformation on a low-dimensional, real "hidden" input I, with independent Gaussian priors on I and w. The input I modulates the emission vectors, and therefore the underlying HMM, as a function of sequence.

9.5.2 Simulation Results

We now review a simple application of the principles behind HMM/NN single-model hybrid architectures, demonstrated in [40], on the immunoglobulin protein family. Immunoglobulins, or antibodies, are proteins produced by B cells that bind with specificity to foreign antigens in order to neutralize them or target their destruction by other effector cells. The various classes of immunoglobulins are defined by pairs of light and heavy chains that are held together principally by disulfide bonds. Each light and heavy chain molecule contains one variable (V) region and one (light) or several (heavy) constant (C) regions (figure 9.12). The V regions differ among immunoglobulins, and provide the specificity of the antigen recognition. About one third of the amino acids of the V regions form the hypervariable sites, responsible for the diversity of the vertebrate immune response. The database is the same as that used in [41], and consists of human and mouse heavy chain immunoglobulin V region sequences from the Protein Identification Resources (PIR) database. It corresponds to 224 sequences with minimum length 90, average length $N = 117$, and maximum length 254.

The immunoglobulin V regions were first modeled using a single HMM [41], similar to the one in figure 7.3, containing a total of $52N + 23 = 6107$ adjustable parameters. Here we consider a hybrid HMM/NN architecture with the following characteristics. The basic model is an HMM with the architecture of figure 7.3. All the main-state emissions are calculated by a common NN with two hidden units. Likewise, all the insert-state emissions are calculated by a common NN with one hidden unit. Each state transition distribution is calculated by a different softmax network. Neglecting edge effects, the total number of parameters of this HMM/NN architecture is 1507: $(117 \times 3 \times 3) = 1053$ for the transitions and $(117 \times 3 + 3 + 3 \times 20 + 40) = 454$ for the emissions, including biases. This architecture is used for demonstration purposes and is not optimized. We suspect that the number of parameters could be further reduced.

The hybrid architecture is then trained online, using both gradient descent and the corresponding Viterbi version. The training set consists of a random subset of 150 sequences identical to the training set used in the experiments with a simple HMM. All weights from the input to the hidden layer are initialized with independent Gaussians, with mean 0 and standard deviation 1. All the weights from the hidden layer to the output layer are initialized to 1. This yields a uniform emission probability distribution on all the emitting states.[2] Note that if all the weights are initialized to 1, including those from input layer to hidden layer, then the hidden units cannot differentiate from one another. The transition probabilities out of insert or delete states are initialized uniformly to 1/3. We introduce, however, a small bias along the backbone that

[2]With Viterbi learning, this is probably better than a nonuniform initialization, such as the average composition, since a nonuniform initialization may introduce distortions in the Viterbi paths.

```
                           1                     24
F37262  --------------------------AELM--KPGASVKISCKATG--YKFSS----Y--------WIEWVKQRPGHGLEWIGENL-
B27563  ---------------------LQQPGAELV--KPGASVKLSCKASG--YTFTN----Y--------WIHWVKQRPGRGLEWIGRID-
C30560  ------------------QVHLQQSGAELV--KPGASVKISCKASG--YTFTS----Y--------WMNWVKQRPGQGLEWIGEID-
G1HUDW  ------------------QVTLRESGPALV--RPTQTLTLTCTFSG--FSLSGetmc----------VAWIRQPPGEALEWLAWDI-
S09711  mkhlwffllvraprwclsQVQLQESGPGLV--KPSETLSVTCTVSG------------gsvsssglYWSWIRQPPGKGPEWIGYIY-
B36006  --------------------------------------KISCKGSG--YSFTS----Y--------WIGWVRQMPGKGLEWMGIIY-
F36005  ------------------QVQLVESGGGVV--QPGRSLRLSCAASG--FTFSS----Y--------AMHWVRQAPGKGLEWVAVIS-
A36194  mgwsfiflfllsvtagvhsEVQLQQSGAELV--RAGSSVKMSCKASG--YTFTN----Y--------GINWVKQRPGQGLEWIGYQS-
A31485  ------------------EVKLDETGGGLV--QPGRPMKLSCVASG--FTFSD----Y--------WMNWVRQSPEKGLEWVAQIRN
D33548  ------------------QVQLVQSGAEVK--KPGASVKVSCEASG--YTFTG----H--------YMHWVRQAPGQGLEWMGWIN-
AVMSJ5  ------------------EVKLLESGGGLV--QPGGSLKLSCAASG--FDFSK----Y--------WMSWVRQAPGKGLEWIGEIH-
D30560  ------------------QVQLKQSGPSLV--QPSQSLSITCTVSD--FSLTN----F--------GVHWVRQSPGKGLEWLGVIW-
S11239  melglswifllailkgvqcEVQLVESGGGLV--QPGRSLRLSCAASG--FTFND----Y--------AMHWVRQAPGKGLEWVSGIS-
G1MSAA  ------------------EVQLQQSGAELV--KAGSSVKMSCKATG--YTFSS----Y--------ELYWVRQAPGQGLEDLGYIS-
I27888  ------------------EVQLVESGGGLV--KPGGSLRLSCAASG--FTFSS----Y--------AMSWVRQSPEKRLEWVADIS-
PL0118  --------------------QLQESGSGLV--KPSQTLSLTCAVSGgsISSGG----Y--------SWSWIRQPPGKGLEWIGYIY-
PL0122  ------------------EVQLVESGGGLV--QPGGSLKLSCAASG--FTFSG----S--------AMHWVRQASGKGLEWVGRIRS
A33989  ------------------DVQLDQSESVVI--KPGGSLKLSCTASG--FTFSS----Y--------WMSWVRQAPGKGLQWVSRISS
A30502  ------------------EVQLQQSGPELV--KPGASVKMSCKASG--DTFTS----S--------VMHWVKQKPGQGLEWIGYIN-
PH0097  ------------------DVKLVESGGGLV--KPGGSLKLSCAASG--FTFSS----Y--------IMSWVRQTPEKRLEWVATIS-

              60        70        80        90      100
F37262  -P-G-SDSTKYNEKFKGKATFTADTSSNTAYMQLSSLTSEDSAVYYCARnyygssnlfay---------------------------
B27563  -P-N-SGGTKYNEKFKNKATLTINKPSNTAYMQLSSLTSDDSAVYYCARgydysyya-------------MDYWGQGTsvtvss---
C30560  -P-S-NSYTNNNQKFKNKATLTVDKSSNTAYMQLSSLTSEDSAVYYCARwgtgsswg------------WFAYWGQGTlvtvsa---
G1HUDW  ----lNDDKYYGASLETRLAVSKDTSKNQVVLSMNTVGPGDTATYYCARscgsq---------------YFDYWGQGIlvtvss---
S09711  ---Y-SGSTNYNPSLRSRVTISVDTSKNQFSLKLGSVTAADTAVYYCARvlvsrtsisqysy-------YMDVWGKGTtvtvss---
B36006  -P-G-DSDTRYSPSFQGQVTISADKSISTAYLQWSSLKASDTAMYYCARrrymgygdqa-----------FDIWGQGTmvtvss---
F36005  -Y-D-GSNKYYADSVKGRFTISRDNSKNTLYLQMNSLRAEDTAVYYCAR--------------DRKASDAFDIWGQGTmvtvss---
A36194  -T-G-SFYSTYNEKVKGKTTLTVDKSSSTAYMQLRGLTSEDSAVYFCARsnyyggsys------------FDYWGQGTtltvss---
A31485  KP-Y-NYETYYSDSVKGRFTISRDDSKSSVYLQMNNLRVEDMGIYYCTGsyyg-----------------MDYWGQGTsvtvss---
D33548  -P-N-SGGTNYAEKFQGRVTITRDTSINTAYMELSRLRSDDTAVYYCARasycgydcyy----------FFDYWGQGTlvtvss---
AVMSJ5  -P-D-SGTINYTPSLKDKFIISRDNAKNSLYLQMSKVRSEDTALYYCARlhyygyn---------------AYWGQGTlvtvsae--
D30560  -P-R-GGNTDYNAAFMSRLSITKDNSKSQVFFKMNSLQADDTAIYYCTKegyfgnydya-----------MDYWGQGTsvtvss---
S11239  --wD-SSSIGYADSVKGRFTISRDNAKNSLYLQMNSLRAEDMALYYCVKgrdyydsggyftva-------FDIWGQGTmvtvss---
G1MSAA  -S-S-SAYPNYAQKFQGRVTITADESTNTAYMELSSLRSEDTAVYFCAVrvisryfdg---------------WGQGTlv-------
I27888  -S-G-GSFTYYPDTVTGRFTISRDDAQNTLYLEMNSLRSEDTAIYYCTRdeedpttlvapfa--------MDYWGQGTsvtvs----
PL0118  ---H-SGSTYYNPSLKSRVTISVDRSKNQFSLKLSSVTAADTAVYYCAR--------------------------------------
PL0122  KA-N-SYATAYAASVKGRFTISRDDSKNTAYLQMNSLKTEDTAVYYCTR--------------------------------------
A33989  KA-D-GGSTYYADSVKGRFTISRDNNNNKLYLQMNNLQTEDTAVYYCTRearwggw-------------YFEHWGQGTmvtvts---
A30502  -P-Y-NDGTKYNEKFKGKATLTSDKSSSTAYMELSSLTSEDSAVYYCARgg-------------------FAYWGQGTlvtv-----
PH0097  -S-G-GRYTYYSDSVKGRFTISRDNAKNTLYLQMSSLRSEDTAMYYSTAsgds-----------------FDYWGQGTtltvssak-
```

Figure 9.13: Multiple Alignment of 20 Immunoglobulin Sequences, Randomly Extracted from the Training and Validation Data Sets. Validation sequences: F37262, G1HUDW, A36194, A31485, D33548, S11239, I27888, A33989, A30502. Alignment is obtained with a hybrid HMM/NN architecture trained for 10 cycles, with two hidden units for the main-state emissions and one hidden unit for the insert-state emissions. Lowercase letters correspond to emissions from insert states. Note that the signal peptide on some of the sequences is captured as repeated transitions through the first insert state in the model.

favors main-to-main transitions, in the form of a nonsymmetric Dirichlet prior. This prior is equivalent to introducing a regularization term in the objective function that is equal to the logarithm of the backbone transition path. The regularization constant is set to 0.01 and the learning rate to 0.1. Typically, 10 training cycles are more than sufficient to reach equilibrium.

In figure 9.13, we display the multiple alignment of 20 immunoglobulin sequences, selected randomly from both the training and the validation sets. The validation set consists of the remaining 74 sequences. This alignment is very stable between 5 and 10 epochs. It corresponds to a model trained using Viterbi learning for 10 epochs. This alignment is similar to the multiple alignment previously derived with a simple HMM, having more than four times as many parameters. The algorithm has been able to detect most of the salient features of the family. Most important, the cysteine residues (C) toward the beginning and the end of the region (positions 24 and 100 in this multiple alignment), which are responsible for the disulfide bonds that hold the chains, are perfectly aligned. The only exception is the last sequence (PH0097), which has a serine (S) residue in its terminal portion. This is a rare but recognized exception to the conservation of this position. A fraction of the sequences in the data set came with a signal peptide sequence in the N-terminal (see section 6.4). We did not remove them prior to training. The model is capable of detecting and accommodating these signal peptides by treating them as initial repeated inserts, as can be seen from the alignment of three of the sequences (S09711, A36194, S11239). This multiple alignment also contains a few isolated problems, related in part to the overuse of gaps and insert sates. Interestingly, this is most evident in the hypervariable regions, for instance, at positions 30–35 and 50–55. These problems should be eliminated with a more careful selection of hybrid architecture and/or regularization. Alignments in this case did not seem to improve with use of gradient descent and/or a larger number of hidden units, up to four.

In figure 9.14, we display the activity of the two hidden units associated with each main state. For most states, at least one of the activities is saturated. The activities associated with the cysteine residues responsible for the disulfide bridges (main states 24 and 100) are all saturated and are in the same corner $(-1,+1)$. Points close to the center $(0,0)$ correspond to emission distributions determined by the bias only.

9.5.3 Summary

A large class of hybrid HMM/NN architectures has been described. These architectures improve on single HMMs in two complementary directions. First, the NN reparameterization provides a flexible tool for the control of model complexity, the introduction of priors, and the construction of an input-dependent mechanism for the modulation of the final model. Second, modeling a data set with multiple HMMs allows the coverage of a larger set of distributions and the expression of nonstationarity and correlations inaccessible to single HMMs. Similar ideas have been introduced in [58] using the notion of input/output HMMs (IOHMMs). The HMM/NN approach is meant to complement rather than replace many of the existing techniques for incorporating prior information in sequence models.

Two important issues for the success of a hybrid HMM/NN architecture on a real problem are the design of the NN architecture and the selection of the external input or context. These issues are problem-dependent and cannot

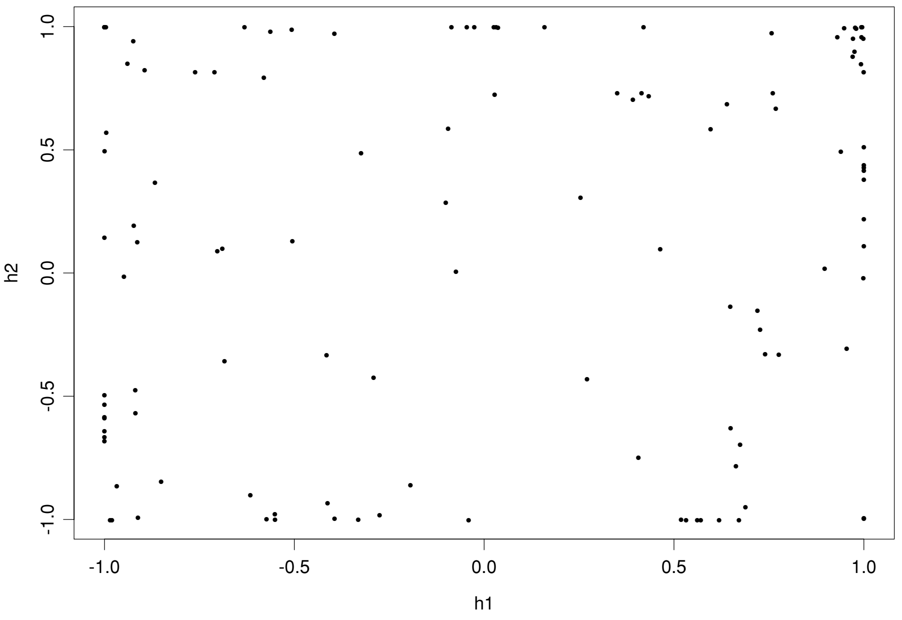

Figure 9.14: Activity of the Two Hidden Units Associated with the Emission of the Main States. The two activities associated with the cysteines (C) are in the upper left corner, almost overlapping, with coordinates $(-1,+1)$.

be handled with generality. We have described some examples of architectures, using mixture ideas for the design of the NN component. Different input choices are possible, such as contextual information, sequences over a different alphabet, or continuous parameterization variables [374].

The methods described in this section are not limited to HMMs, but can be applied to any class of probabilistic models. The basic idea is to calculate and possibly modulate the parameters of the models using NNs (or any other flexible reparameterization). Several implicit examples of hybrid architectures can be found in the literature (for example [395]). In fact, the NN architectures of chapter 5 can be viewed as hybrid architectures. In the standard regression case, a Gaussian model is used for each point in input space. Each Gaussian model is parameterized by its mean. The standard NN architecture simply computes the mean at each point. Although the principle of hybrid modeling is not new, by exploiting it systematically in the case of HMMs, we have generated new classes of models. In other classes the principle has not yet been applied systematically, for example, probabilistic models of evolution (chapter 10) and stochastic grammars (chapter 11). In the next section, we closely follow [37] and apply similar techniques to a larger class of probabilistic models, namely to BIOHMMs and the problem of predicting protein secondary structure.

9.6 Bidirectional Recurrent Neural Networks for Protein Secondary Structure Prediction

Protein secondary structure prediction (see also section 6.3) can be formulated as the problem of learning a synchronous sequential translation from strings in the amino acid alphabet to strings written in the alphabet of structural categories. Because biological sequences are spatial rather than temporal, we have seen that BIOHMMs are an interesting new class of graphical models for this problem. In particular, they offer a sensible alternative to methods based on a fixed-width input window. The expressive power of these models enables them to capture distant information in the form of contextual knowledge stored into hidden state variables. In this way, they can potentially overcome the main disadvantage of feedforward networks, namely the linear growth of the number of parameters with the window size. Intuitively, these models are parsimonious because of the implicit weight sharing resulting from their stationarity; i.e., parameters do not vary over time.

We have used BIOHMMs directly to predict protein secondary structure with some success [36]. As graphical models, however, BIOHMMs contain undirected loops and therefore require a computationally intensive evidence-propagation algorithm (the junction tree algorithm [287]), rather than the simpler Pearl's algorithm for loopless graphs such as HMMs (see also appendix C). Thus to speed up the algorithm, we can use the technique of the previous section and use neural networks, both feedforward and recurrent, to reparameterize the graphical model.

9.6.1 Bidirectional Recurrent Neural Nets

Letting t denote position within a protein sequence, the overall model can be viewed as a probabilistic model that outputs, for each t, a vector $O_t = (o_{1,t}, o_{2,t}, o_{3,t})$ with $o_{i,t} \geq 0$ and $\sum_i o_{i,t} = 1$. The $o_{i,t}$s are the secondary structure class membership probabilities. The output prediction has the functional form

$$O_t = \eta(F_t, B_t, I_t) \tag{9.15}$$

and depends on the forward (upstream) context F_t, the backward (downstream context) B_t, and the input I_t at time t. The vector $I_t \in \mathbb{R}^k$ encodes the external input at time t. In the most simple case, where the input is limited to a single amino acid, $k = 20$, by using the orthogonal binary encoding (see section 6.1). In this case, it is not necessary to include an extra input symbol to represent the terminal portions of the protein. Larger input windows extending over several amino acids are of course also possible. The function η is realized by a neural network $\mathcal{N}_\eta$ (see center and top connections in figure 9.15). Thus to guarantee a consistent probabilistic interpretation, the three output units of network $\mathcal{N}_\eta$ are obtained as normalized exponentials (or *softmax*)

$$o_{i,t} = \frac{\exp(net_{i,t})}{\sum_{l=1}^{3} \exp(net_{l,t})} i = 1, 2, 3; \tag{9.16}$$

where $net_{i,t}$ is the activation of the ith output unit at position t. The performance of the model can be assessed using the usual relative entropy between the estimated and the target distribution.

The novelty of the model is in the contextual information contained in the vectors $F_t \in \mathbb{R}^n$ and especially in $B_t \in \mathbb{R}^m$. These satisfy the recurrent bidirectional equations

$$\begin{aligned} F_t &= \phi(F_{t-1}, I_t) \\ B_t &= \beta(B_{t+1}, I_t) \end{aligned} \tag{9.17}$$

Here $\phi(\cdot)$ and $\beta(\cdot)$ are learnable nonlinear state transition functions. They can be implemented in different forms, but here we assume that they are realized by two NNs, $\mathcal{N}_\phi$ and $\mathcal{N}_\beta$ (left and right subnetworks in figure 9.15), with n and m logistic output units, respectively. Thus, $\mathcal{N}_\phi$ and $\mathcal{N}_\beta$ are fed by $n + k$ and $m + k$ inputs, respectively. Here also larger input windows are possible, especially in combination with the weight-sharing approach described in [445], and different inputs could be used for the computation of F_t, B_t, and O_t. The *forward* chain F_t stores contextual information contained to the left of time t and plays the same role as the internal state in standard RNNs. The novel part of the model is the presence of an additional *backward* chain B_t, in charge of storing contextual information contained to the right of time t, i.e. in the future. The actual form of the bidirectional dynamics is controlled by the connection weights in the subnetworks $\mathcal{N}_\phi$ and $\mathcal{N}_\beta$. As we shall see, these weights can be adjusted using a maximum-likelihood approach. Since (9.17) involves two recurrences, two corresponding boundary conditions must be specified, at the beginning and the end of the sequence. For simplicity, here we use $F_0 = B_{N+1} = 0$, but it is also possible to adapt the boundaries to the data, extending the technique suggested in [184] for standard RNNs.

The discrete time index t ranges from 1 to N, the total length of the protein sequence being examined. Hence the probabilistic output O_t is parameterized by a RNN and depends on the input I_t and on the contextual information, from the *entire* protein, summarized into the pair (F_t, B_t). In contrast, in a conventional NN approach this probability distribution depends only on a relatively short subsequence of amino acids. Intuitively, we can think of F_t and B_t as "wheels" that can be "rolled" along the protein. To predict the class at position t, we roll the wheels in opposite directions from the N and C terminus up

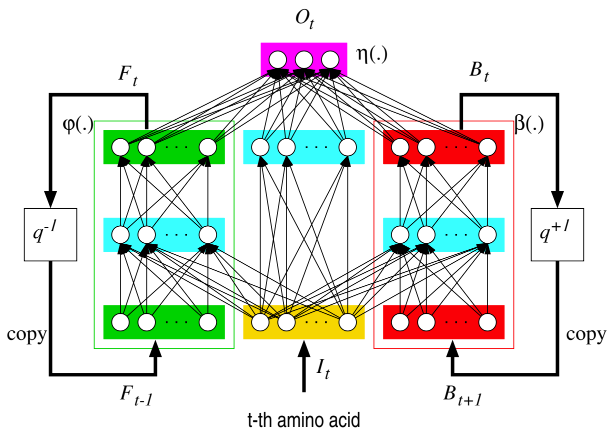

Figure 9.15: Bidirectional Recursive Neural Network Architecture. Inputs correspond to amino acid letters in a given protein sequence. Outputs correspond to secondary structure classification into alpha-helices, beta-sheets, and coils.

to position t and then combine what is read on the wheels with I_t to calculate the proper output using η.

The global mapping from the input amino acid sequence to the sequence of output categories can be described by the graphical model shown in figure 9.16. The network represents the direct dependencies among the variables I_t, F_t, B_t, O_t, unrolled over time for $t = 1, \ldots, N$. Each node is labeled by one of the variables and arcs represent direct functional dependencies. This graph represents the underlying Bayesian network BIOHMMs except that the internal relationships amongst I_t, F_t, B_t, O_t here are deterministic((9.15) and (9.17)), rather than probabilistic. The overall BRNN model, however, is a probabilistic model. As we have seen, inference in the BIOHMMs is tractable but requires time complexity of $O(n^3)$ for each time step (here n is the typical number of states in the chains), limiting their practical applicability to the secondary structure prediction task [36].

An architecture resulting from (9.15) and (9.17) is shown in figure 9.15 where, for simplicity, all the NNs have a single hidden layer. The hidden state F_t is copied back to the input. This is graphically represented in figure 9.15 using the causal *shift operator* q^{-1} that operates on a generic temporal variable X_t and is symbolically defined as $X_{t-1} = q^{-1}X_t$. Similarly, q, the inverse (or noncausal) shift operator is defined $X_{t+1} = qX_t$ and $q^{-1}q = 1$. As shown in Figure 9.15, a noncausal copy is performed on the hidden state B_t. Clearly, removal of $\{B_t\}$ would result in a standard causal RNN.

The number of degrees of freedom of the model depends on two factors: (1) the dimensions n and m of the forward and backward state vectors; (2) the number of hidden units in the three feedforward networks realizing the state transition and the output functions (see figure 9.15). It is important to remark that the BRNN has been defined as a stationary model; that is, the connection weights in the networks realizing $\beta(\cdot)$, $\phi(\cdot)$ and $\eta(\cdot)$ do not change over time, i.e. with respect to position along the protein. This is a form of weight sharing that reduces the number of free parameters and the risk of overfitting, without necessarily sacrificing the capability to capture distant information.

9.6.2 Inference and Learning

Since the graph shown in figure 9.16 is acyclic, nodes can be topologically sorted, defining unambiguously the global processing scheme. Using the network unrolled through time, the BRNN prediction algorithm updates all the states F_t from left to right, starting from $F_0 = 0$. Similarly, states B_t are updated from right to left. After forward and backward propagations have taken place, the predictions O_t can be computed. The forward and backward propagations need to be computed from end to end only once per protein sequence. As a result, the time complexity of the algorithm is $O(NW)$, where W is the number of weights and N the protein length. This is the same complexity as feedforward networks fed by a fixed-size window. In the case of BRNNs, W typically grows as $O(n^2)$ and the actual number of weights can be reduced by limiting the number of hidden units in the subnetworks for $\phi(\cdot)$ and $\beta(\cdot)$. Thus, inference in BRNNs is more efficient than in bidirectional IOHMMs, where the complexity is $O(Nn^3)$ [36].

Learning can be formulated as a maximum likelihood estimation problem, where the log-likelihood is essentially the relative entropy function between the predicted and the true conditional distribution of the secondary structure sequence given the input amino acid sequence

$$\ell = \sum_{\text{sequences}} \sum_{t=1}^{N} z_{i,t} \log o_{i,t}, \tag{9.18}$$

with $z_{i,t} = 1$ if the secondary structure at position t is i and $z_{i,t} = 0$ otherwise. The optimization problem can be solved by gradient ascent. The only difference with respect to standard RNNs is that gradients must be computed by taking into account noncausal temporal dependencies. Because the unrolled network is acyclic, the generalized backpropagation algorithm can be derived as a special case of the backpropagation through structure algorithm [188]. Intuitively, the error signal is first injected into the leaf nodes, corresponding to the output variables O_t. The error is then propagated through time in both

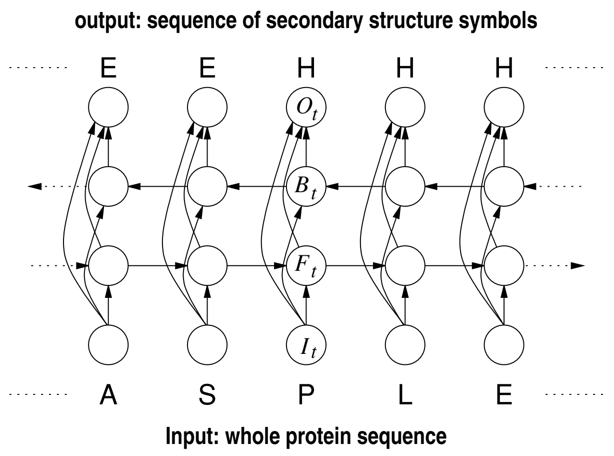
output: sequence of secondary structure symbols
E E H H H
O_t
B_t
F_t
I_t
A S P L E
Input: whole protein sequence

Figure 9.16: Direct Dependencies Among the Variables Involved in a Bidirectional BRNN. The boundary conditions are provided by $F_0 = B_{N+1} = 0$ and by the inputs associated with the current protein sequence.

directions, by following any reverse topological sort of the unrolled network (see figure 9.16). Obviously, this step also involves backpropagation through the hidden layers of the NNs. Since the model is stationary, weights are shared among the different replicas of the NNs at different time steps. Hence, the total gradient is simply obtained by summing all the contributions associated with different time steps.

To speedup convergence, it was found convenient to adopt an online weight-updating strategy. Once gradients relative to a single protein have been computed, weights are immediately updated. This scheme was enriched also with a heuristic adaptive learning rate algorithm that progressively reduces the learning rate if the average error reduction within a fixed number of epochs falls below a given threshold.

9.6.3 Long-range Dependencies

One of the principal difficulties in training standard RNNs is the problem of vanishing gradients [57]. Intuitively, in order to contribute to the output at position or time t, the input signal at time $t-\tau$ must be propagated in the forward chain through τ replicas of the NN that implements the state transition function. However, during gradient computation, error signals must be propagated backward along the same path. Each propagation can be interpreted as the product between the error vector and the Jacobian matrix associated with the transition function. Unfortunately, when the dynamics develop attractors that allow the system to store past information reliably, the norm of the Jacobian is < 1. Hence, when τ is large, gradients of the error at time t with respect to inputs at time $t - \tau$ tend to vanish exponentially. Similarly, in the case of BRNNs, error propagation in both the forward and the backward chains is subject to exponential decay. Thus, although the model has in principle the capability of storing remote information, such information cannot be learnt effectively. Clearly, this is a theoretical argument and its practical impact needs to be evaluated on a per-case basis.

In practice, in the case of proteins, the BRNN can reliably utilize input information located within about ± 15 amino acids (i.e., the total effective window size is about 31). This was empirically evaluated by feeding the model with increasingly long protein fragments. We observed that the average predictions at the central residues did not significantly change if fragments were extended beyond 41 amino acids. This is an improvement over standard NNs with input window sizes ranging from 11 to 17 amino acids [453, 445, 290]. Yet there is presumably relevant information located at longer distances that these models have not been able to discover so far.

To limit this problem, a remedy was proposed motivated by recent studies [364] suggesting that the vanishing-gradients problem can be mitigated by the use of an explicit delay line applied to the output, which provides shorter paths for the effective propagation of error signals. Unfortunately, this idea cannot be applied directly to BRNNs since output feedback, combined with bidirectional propagation, would generate cycles in the unrolled network. A similar mechanism, however, can be implemented using the following modified dynamics:

$$\begin{aligned} F_t &= \phi(F_{t-1}, F_{t-2}, \ldots, F_{t-s}, I_t) \\ B_t &= \beta(B_{t+1}, B_{t+2}, \ldots, B_{t+s}, I_t). \end{aligned} \tag{9.19}$$

The explicit dependence on forward or backward states introduces *shortcut* connections in the graphical model, forming shorter paths along which gradients can be propagated. This is akin to introducing higher-order Markov chains in the probabilistic version. However, unlike Markov chains where the number of parameters would grow exponentially with s, in the present case the number of parameters grows only linearly with s. To reduce the number of parameters, a simplified version of (9.19) limits the dependencies to state vectors located s residues away from t:

$$\begin{aligned} F_t &= \phi(F_{t-1}, F_{t-s}, I_t) \\ B_t &= \beta(B_{t+1}, B_{t+s}, I_t). \end{aligned} \tag{9.20}$$

Another variant of the basic architecture that also lets us increase the effective window size consists in feeding the output networks with a window in the

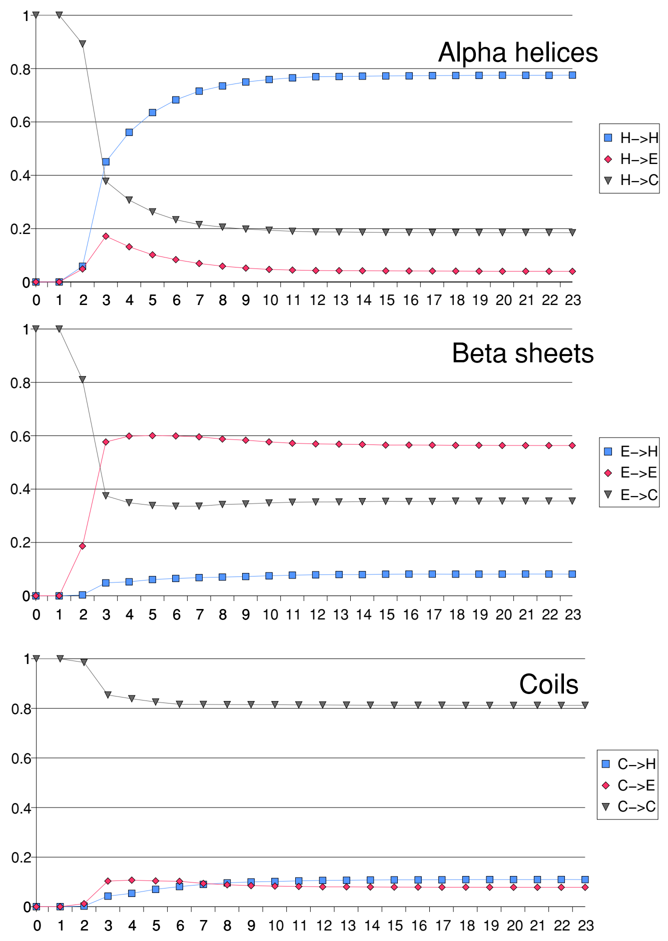

Figure 9.17: Distant Information Exploited by the BRNN. The horizontal axis represents τ, the distance from a given position beyond which all entries are set to null values. Each curve represents a normalized row of the test-set confusion matrix.

forward and backward state chains. In this case, the prediction is computed as

$$O_t = \eta(F_{t-s}, \ldots, F_{t+s}, B_{t-s}, \ldots, B_{t+s}, I_t). \tag{9.21}$$

Notice that the window can extend in the past and the future of t on both vectors F_t and B_t.

9.6.4 Implementation and Results

BRNNs have been used to implement SSpro, a secondary structure prediction server available through the Internet[3]. In addition to BRNNs, SSpro uses other features that over the years have proved to be useful for secondary structure prediction, such as ensembles and profiles (see section 6.3). Profiles, in particular, are most useful when used at the input level. Details of experiments and performance analysis of the first version of SSpro which used BLAST-generated profiles are given in [37]. The most recent version of SSpro uses PSIBLAST profiles and achieves a performance of about 80% correct prediction. SSpro has been ranked among the top predictors both at the 2000 CASP blind prediction competition and through the independent automatic evaluation server EVA of Rost (http://dodo.bioc.columbia.edu/~eva/), based on the new sequences that are deposited each week in the PDB.

Beyond the performance results, to study the capabilities of BRNNs models to capture long-ranged information a number of experiments were performed. For each protein and for each amino acid position t, we fed the BRNN mixture described above with a sequence obtained by replacing all inputs outside the range $[t-\tau, t+\tau]$ with null values. The experiment was repeated for different values of τ from 0 to 23. Figure 9.17 shows the results. Each diagram is a normalized row of the test set confusion table, for the semi-window size τ ranging from 0 to 23. So for example the line labeled $H \to C$ in the first diagram is the percentage of helices classified as coils, as a function of τ. The curves are almost stable for $\tau > 15$. Although the model is not sensitive to *very* distant information, it should be remarked that typical feedforward nets reported in the literature do not exploit information beyond $\tau = 8$.

Given the large number of protein sequences available through genome and other sequencing projects, even small percentage improvements in secondary structure prediction are significant for structural genomics. Machine learning algorithms combined with graphical models and their NN parameterizations are one of the best approaches so far in this area. BRNNs and the related ideas presented here begin to address problems of long-ranged dependencies. As a result, BRNNs have now been developed to predict a number of other structural features including amino acid partners in beta sheets, number of residue contacts, and solvent accessibility [45, 429]. These structural modules are part of a broader strategy towards 3D prediction based on the intermediary prediction of contact maps, with both low (secondary structure) and high (amino acid) resolution, starting from the primary sequence and the predicted structural features. Indeed, prediction of the arrangement of secondary structure elements with respect to each other in three-dimensions would go a long way

[3]SSpro is accessible through http://promoter.ics.uci.edu/BRNN-PRED/.

towards the prediction of protein topology and three-dimensional structure.

There are several directions in which this work could be extended including many architectural variations. In addition to the use of larger input windows for I_t, one may consider non-symmetrical chains for the past and the future, and the use of priors on the parameters and/or the architecture together with a maximum a posteriori learning approach. It may also be advantageous to use an array of "wheels," instead of just two wheels, of various memory capacity, rolling in different directions along the protein and possibly over shorter distances. It is also worth noting that using multi-layered perceptrons for implementing $\beta(.)$ and $\phi(.)$ is just one of the available options. For example, recurrent radial basis functions or a generalization of second-order RNN [208] are easily conceivable alternative parameterization. Finally, the ideas described in this section can be applied to other problems in bioinformatics, as well as other domains, where non-causal dynamical approaches are suitable. Obvious candidates for further tests of the general method include the prediction of protein functional features, such as signal peptides.

Chapter 10

Probabilistic Models of Evolution: Phylogenetic Trees

10.1 Introduction to Probabilistic Models of Evolution

This chapter deals with evolution and the inference of phylogenetic trees from sequence data. It is included of course because sequence evolution is a central topic in computational molecular biology, but also because the ideas and algorithms used are again a perfect illustration of the general probabilistic inference framework of chapter 2.

Evolutionary relationships between organisms—existing or extinct—have been inferred using morphological and/or biochemical characteristics since the time of Darwin. Today, phylogenetic trees are commonly derived from DNA and protein sequences [182]. Due to the extreme stability of the DNA molecule, it can be extracted in large intact pieces even from organisms that have been dead for many years [251]. The extinct elephant-like mammoth has been phylogenetically mapped by its DNA, and for deceased humans precise family relationships can also be established. Among the most recent examples are the proof of the identity of the last Russian tsar, Nicholas II [211, 274], and the disproof of the story of Anna Anderson, who claimed she was the tsar's missing daughter Anastasia [212, 477]. The bones (and the DNA) of the tsar had been lying in the soil since 1918.

The literature contains a number of methods for inferring phylogenetic trees from sequence data. Most of the approaches are variations on two major methods: parsimony methods [181] and likelihood methods [178, 519, 269]. Not surprisingly, likelihood methods are based on a probabilistic model of the evolutionary process (see also [295]). Actually, the term "likelihood methods" is typically used in this field in connection with a particular class of probabilistic models. Although parsimony methods are often described independently of any underlying model of evolution, we will show that they can be viewed as approximations to likelihood methods.

From the general Bayesian framework and the Cox–Jaynes axioms, we know that in order to infer a phylogenetic tree from a set of sequences, we *must* begin with a probabilistic model of evolution. Maximum likelihood (ML) is then the most basic inference step we can perform with respect to such a model. ML encompasses all the other approaches currently found in the literature, including the ML approach in [178] with respect to a particular model class. As we have seen, HMMs are not a complete model of the evolutionary process. Evolution can proceed at the molecular level not only by insertions and deletions but also by substitutions, inversions, and transpositions. Therefore different models must be used. But first we need some elementary background and notation for trees.

10.1.1 Trees

A tree T is a connected acyclic graph. In a tree every two points are joined by a unique path, and the number of vertices always exceeds the number of edges by exactly 1. A tree is *binary* if each vertex has either one or three neighbors. A tree is *rooted* if a node r has been selected and termed the root. In phylogenetic trees, the root is intended to represent an ancestral sequence from which all the others descend. Two important aspects of phylogenetic trees, both rooted and unrooted, are the topology and the branch length. The topology refers to the branching pattern of the tree associated with the times of divergence. The branch length is often used to represent in some way the time distance between events (figure 10.1).

10.1.2 Probabilistic Models

The most basic but still useful probabilistic model of evolution is again a variation of the simple dice model. We can imagine that starting from an ancestral sequence, evolution proceeds randomly, using position-independent substitutions only. If we look at a given fixed position i in the sequences, and if we let $\chi^i(t)$ denote the letter at position i at time t, we can make the usual Markov process assumption that the probability

$$p^i_{\mathsf{YX}}(t) = \mathbf{P}(\chi^i(t+s) = \mathsf{Y} | \chi^i(s) = \mathsf{X}) \tag{10.1}$$

is independent of $s \geq 0$ for $t > 0$. Thus, for each position i, there is a probability $p^i_{\mathsf{YX}}(t)$ that X is substituted into Y over an evolutionary period of time

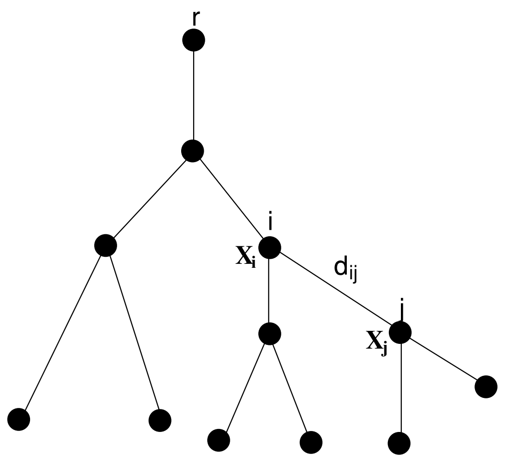

Figure 10.1: A Simple Binary Phylogenetic Tree. r is the root; d_{ji} is the time distance between i and j; X_i is the letter assigned to the hidden vertex i. The observed letters are at the leaves at the bottom. The probability of the substitution from vertex i to vertex j is $p_{X_j X_i}(d_{ji})$.

t. Thus for each t and each position i we have a collection of $|A|$ dice. To simplify the model further, we shall, for now, make the additional approximation that the substitution probabilities are identical at all positions, so that $p^i_{YX}(t) = p_{YX}(t)$. Obviously, we must have $p_{YX}(t) \geq 0$ for any X, Y and any t, and furthermore $\sum_Y p_{YX}(t) = 1$. From (10.1), one must also have the Chapman–Kolmogorov relation,

$$p_{YX}(t+s) = \sum_{Z \in A} p_{YZ}(t) p_{ZX}(s), \tag{10.2}$$

due to the independence of the events at time t and time s.

10.2 Substitution Probabilities and Evolutionary Rates

All that remains to specify the model entirely is to determine the substitution probabilities $p_{YX}(t)$. These are related to the substitution matrices, such as PAM matrices, discussed in chapter 1. It is sensible to make the further assumption that

$$\lim_{t \to 0^+} p_{YX}(t) = \delta(Y, X) = \begin{cases} 1 & \text{if } Y = X \\ 0 & \text{otherwise} \end{cases}. \tag{10.3}$$

If we let $P(t)$ denote the matrix $P(t) = (p_{YX}(t))$, from (10.3) we can define $P(0) = Id$, where Id is the $|A| \times |A|$ identity matrix. One can then show that each component of $P(t)$ is differentiable, so that we can write $P'(t) = (p'_{YX}(t))$. The process is now entirely specified by its right derivative at 0,

$$Q = P'(0) = \lim_{t \to 0^+} \frac{P(t) - Id}{t}. \tag{10.4}$$

This is because (10.4) implies that

$$P'(t) = QP(t) = P(t)Q. \tag{10.5}$$

To see this rapidly, just write $P(t + dt) = P(t)P(dt) = P(t)(P(0) + Qdt) = P(t)(Id + Qdt)$, the first equality resulting from (10.2) and the second from (10.4), so that $P(t + dt) - P(t) = P(t)Qdt$. If we let $Q = (q_{YX})$, from (10.5) the final solution is given by

$$P(t) = e^{Q(t)} = Id + \sum_{n=1}^{\infty} \frac{Q^n t^n}{n!}. \tag{10.6}$$

Note that if Q is symmetric, so is P, and vice versa. Such an assumption may simplify calculations, but it is not biologically realistic and will not be used here. Finally, a distribution column vector $p = (p_X)$ is *stationary* if $P(t)p = p$ for all times t. Thus, once in a stationary distribution, the process remains in it forever. From (10.4), this implies that p is in the kernel of Q: $Qp = 0$. And from (10.6) the two statements are in fact equivalent. If we assume that the observed sequences have been produced with the system in its stationary distribution, then p is easily estimated from the average composition of the observed sequences.

To summarize, we have defined a class of probabilistic models for the evolution of sequences. Such models are characterized by four assumptions:

1. At each site, evolution operates by substitution only, and therefore without indels (insertions and deletions). All observed sequences must have the same length.

2. Substitutions at each position are independent of one another.

3. Substitution probabilities depend only on the current state, and not on the past history (Markov property).

4. The Markov process is the same for all positions.

None of these assumptions are satisfied by real DNA, where the sequence length can change as a result of indels; evolution of different positions is not independent; evolution rates are not uniform both in time and as a function of position; and, last, real DNA is subjected to recombination. But this constitutes a useful first approximation. Many current research efforts concentrate on relaxing some of these assumptions. The first two assumptions are probably the most difficult to relax. For indels, one can easily add a gap symbol to the alphabet A within the present framework, although this is not entirely satisfactory. In any case, to specify a model further within the class described above, one must provide the matrix of rates Q.

10.3 Rates of Evolution

First note that the rate matrix Q is defined up to a multiplicative factor because $P(t) = \exp(Qt) = \exp[(\lambda Q)(t/\lambda)]$ for any $\lambda \neq 0$. In one simple subclass of models, we assume that λdt is the probability that a substitution occurs at a given position over a small time interval dt. Thus λ is the rate of substitution per unit time. Furthermore, when a substitution occurs, a letter is chosen with probability $p = (p_{\mathsf{X}})$. Thus we have

$$p_{\mathsf{YX}}(dt) = (1 - \lambda dt)\delta(\mathsf{Y}, \mathsf{X}) + \lambda dt p_{\mathsf{Y}}. \tag{10.7}$$

This is equivalent to specifying the matrix Q by

$$q_{\mathsf{XX}} = \lambda(p_{\mathsf{X}} - 1) \quad \text{and} \quad q_{\mathsf{YX}} = \lambda p_{\mathsf{Y}} \tag{10.8}$$

for any X and Y. From (10.8) and (10.6), or directly from (10.7) by noting that $e^{-\lambda t}$ is the probability of not having any substitutions at all over a period of length t, we have

$$p_{\mathsf{YX}}(t) = e^{-\lambda t}\delta(\mathsf{Y}, \mathsf{X}) + (1 - e^{-\lambda t})p_{\mathsf{Y}}. \tag{10.9}$$

It is useful to note that the distribution p used in (10.7) can be chosen arbitrarily. However, once chosen, it can be shown that it is the stationary distribution of (10.9); hence the common notation used above. As above, p can be obtained directly from the data if we assume that the data are at equilibrium.

Again, $p_{\mathsf{YX}}(t)$ depends on t via the product λt only. In the absence of any other evidence, we can choose $\lambda = 1$ and thus measure t in units of expected numbers of substitutions. If λ is allowed to vary along each branch of the tree, this is equivalent to measuring time with clocks running at different rates on different branches. Thus the total length from the root to the leaves of the tree need not be constant along all possible paths.

Another useful property of the process defined by (10.9) is that it is *reversible* in the sense that the substitution process looks the same forward and backward in time. This is easily seen from the fact that (10.9) yields the balance equations

$$p_{\mathsf{YX}}(t)p_{\mathsf{X}} = p_{\mathsf{XY}}(t)p_{\mathsf{Y}}. \tag{10.10}$$

Reversibility is also satisfied by other probabilistic models of evolution [302].

10.4 Data Likelihood

Given a set of sequences and an evolutionary probabilistic model, we can try to find the most likely tree topology as well as the most likely lengths for the branches [178, 519]. This explains the use of the expression "ML methods for phylogeny."

We first assume that we have K sequences over the alphabet A, all with the same length N, and a corresponding given phylogenetic tree T with root r and time lengths d_{ji} between adjacent vertices i and j. The first goal is to compute the likelihood $\mathbf{P}(O_1, \ldots, O_K|T)$ according to the evolutionary Markovian models described above. Because of the independence assumption across column positions, we have

$$\mathbf{P}(O_1, \ldots, O_K|T) = \prod_{k=1}^{N} \mathbf{P}(O_1^k, \ldots, O_K^k|T), \tag{10.11}$$

where O_j^k represents the kth letter observed in the jth sequence. Therefore we need to study only the term $\mathbf{P}(O_1^k, \ldots, O_K^k|T)$ associated with the column k and with the letters O_j^k at the K leaves of the tree. In what follows, we will use the generic notation O to denote the set of observed letters at a fixed position. We can consider that at each vertex i of the tree there is a *hidden* random variable χ_i representing the letter associated with vertex i. Thus a phylogenetic tree can be viewed as a simple Bayesian network (appendix C) with a tree structure in which the conditional probability of a node j, given its parent i, is parameterized by the time distance d_{ji} in the form

$$\mathbf{P}(\chi_j = \mathsf{Y}|\chi_i = \mathsf{X}) = p_{\mathsf{YX}}(d_{ji}). \tag{10.12}$$

Thus all the well-known algorithms for Bayesian networks can be applied in this simple case. In particular, the likelihood $\mathbf{P}(O|T) = \mathbf{P}(O_1^k, \ldots, O_K^k|T)$ can be computed in two ways: starting from the root or starting from the leaves.

Starting from the root, let (X_i) denote an assignment of letters to the internal nodes I other than the leaves, and including the root r. The letters assigned to the internal nodes play of course the role of hidden variables, similar to the HMM paths in chapter 7. In this notation, X_i is assigned to vertex i and the notation is extended to include the letters observed at the leaves. The probability of such a global assignment is easily computed:

$$\mathbf{P}(O, (\mathsf{X}_i)|T) = \mathbf{P}((\mathsf{X}_i)|T) = p_r(\mathsf{X}_r) \prod_{i \in I} \prod_{j \in N^+(i)} p_{\mathsf{Y}_j \mathsf{X}_i}(d_{ji}), \tag{10.13}$$

where p_r is the prior probability distribution for the letters at the root node. $N^+(i)$ denotes the set of children of vertex i, the edges being oriented from the root to the leaves. Assuming that the process is at equilibrium, p_r is the stationary distribution $p = p_r$ and thus can be estimated from the average composition. The observation likelihood is computed by summing over all possible assignments:

$$\mathbf{P}(O|T) = \sum_{(\mathsf{X}_i)} p_r(\mathsf{X}_r) \prod_{I-\{r\}} \prod_{j \in N^+(i)} p_{\mathsf{Y}_j\mathsf{X}_i}(d_{ji}). \tag{10.14}$$

The sum above contains $|A|^{|T|-K}$ terms and is computationally not efficient. $|T|$ is the number of trees.

The likelihood is computed more efficiently by recursively propagating the evidence from the observed leaves to the root. Let $O^+(i)$ denote the portion of evidence contained in the subtree rooted at vertex i, that is, the letters observed on the leaves that are descendants of i. Then if i is a leaf of the tree,

$$\mathbf{P}(O^+(i)|\chi_i = \mathsf{X}, T) = \begin{cases} 1 & \text{if X is observed at } i \\ 0 & \text{otherwise} \end{cases}. \tag{10.15}$$

A different distribution can be used when the letter associated with a leaf is known only with some ambiguity. If i is any internal node,

$$\mathbf{P}(O^+(i)|\chi_i = \mathsf{X}, T) = \sum_{\mathsf{Y} \in A} \sum_{j \in N^+(i)} p_{\mathsf{YX}}(d_{ji}) \mathbf{P}((O^+(j)|\chi_j = \mathsf{Y}, T). \tag{10.16}$$

The evidence O can be propagated in this way all the way to the root r. The complete likelihood is then easily shown to be

$$\mathbf{P}(O|T) = \sum_{\mathsf{X} \in A} p_r(\mathsf{X}) \mathbf{P}(O^+(r)|\chi_r = \mathsf{X}, T) = \sum_{\mathsf{X} \in A} p_r(\mathsf{X}) \mathbf{P}(O|\chi_r = \mathsf{X}, T). \tag{10.17}$$

This algorithm, which again is a propagation algorithm for Bayesian networks, is sometimes called the "peeling" or "pruning" algorithm. Note that the average composition for p_r and the $p^k_{\mathsf{YX}}(d_{ji})$ probabilities can be chosen differently for each column position without changing the structure of the previous calculations. Thus the evolutionary models on each site are similar but need not be identical. It is also worth noting that, instead of integrating over all possible assignments of the internal nodes, one could compute an optimal (most probable) assignment of letters for the internal node. This is the equivalent of the Viterbi path computations we have seen for HMMs.

One useful observation is that if the evolutionary model is reversible and if there are no external constraints on the position of the root (e.g., a requirement that all the leaves be contemporaneous), then the likelihood is independent of the position of the root. The process being the same forward or backward, the root can be moved arbitrarily along any edge of the tree and therefore over the

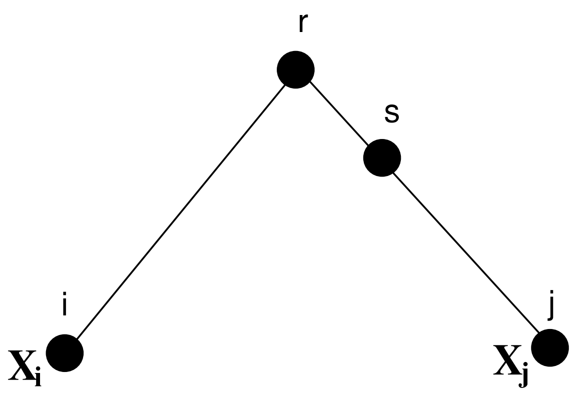

Figure 10.2: Tree Rooted at r with Possible Alternative Root s on the r-to-j Branch.

entire tree. More formally, consider a tree starting with a root r, two children i and j, and an alternative root s on the branch from r to j (figure 10.2). From (10.16) and (10.17), we have

$$\mathbf{P}(O|T) = \sum_{\mathsf{X,Y,Z}\in A} p_r(\mathsf{X}) p_{\mathsf{YX}}(d_{ir}) \mathbf{P}(O^+(i)|\chi_i = \mathsf{Y}, T) p_{\mathsf{ZX}}(d_{sr}) \mathbf{P}(O^+(s)|\chi_s = \mathsf{Z}, T). \tag{10.18}$$

Taking into account the reversibility and assuming the system at equilibrium: $p = p_r = p_s$ and $p_r(\mathsf{X})p_{\mathsf{ZX}}(d_{sr}) = p_s(\mathsf{Z})p_{\mathsf{XZ}}(d_{rs})$. Now

$$\sum_{\mathsf{Y}\in A} p_{\mathsf{YX}}(d_{ir}) \mathbf{P}(O^+(i)|\chi_i = \mathsf{Y}, T) = \mathbf{P}(O^{++}(r)|\chi_r = \mathsf{X}, T), \tag{10.19}$$

where the ++ notation denotes evidence of a tree rooted at s rather than r. Likewise,

$$\mathbf{P}(O^+(s)|\chi_s = \mathsf{Z}, T) = \sum_{\mathsf{W}\in A} \mathbf{P}(O^{++}(j)|\chi_j = \mathsf{W}, T) p_{\mathsf{WZ}}(d_{js}). \tag{10.20}$$

Collecting terms, we finally have

$$\mathbf{P}(O|T) = \sum_{\mathsf{X}\in A} p_s(\mathsf{X}) \mathbf{P}(O^{++}(s)|\chi_s = \mathsf{X}, T). \tag{10.21}$$

Thus we are free to position the root anywhere on the tree without altering the likelihood, or we can speak of the likelihood of the equivalence class associated with the unrooted tree.

10.5 Optimal Trees and Learning

Little work has been done so far to define prior distributions on the space of phylogenetic trees, in terms of both the branching process and the branching lengths. We thus omit the topic of priors and proceed with the first Bayesian inferential step: estimating ML trees. Section 10.4 computed the likelihood for a given tree topology and branching length. For a given topology, the lengths d_{ji} can be viewed as the parameters of the models and therefore can be optimized by ML. As for HMMs, in general the ML estimate cannot be determined analytically but can be approximated using, for example, gradient descent, EM, or perhaps some form of Viterbi learning. We leave it as an exercise for the reader to find EM or gradient-descent equations for the optimization of the branch length [178].

10.5.1 Optimal Topologies

The optimization of the topology is a second problem that requires approximations. The number of possible trees, even unrooted, is exponentially large and the space of topologies cannot be searched exhaustively. Heuristic algorithms for navigating in this space toward good topologies are described in the literature [178] and will not be reviewed here in detail. One widely used heuristic algorithm consists of progressively adding species (i.e., observation sequences) one by one, starting with a two-species tree. At each step, a new species is selected and all its possible positions with respect to the current tree are considered. The most likely is selected before proceeding to the next step. One serious caveat with a search algorithm of this sort is that the final tree topology depends on the order of presentation of the observation sequences.

In any case, it is clear that the ML approach to phylogenetic trees is rather computationally intensive. A complete Bayesian treatment of phylogeny is even more intensive since, in addition to priors, it requires integrating across trees in order, for instance, to estimate the probability that a given substitution has or has not occurred in the past. Parsimony methods can be viewed as fast approximations to ML.

10.6 Parsimony

The basic idea behind parsimony is that the optimal tree is the one requiring the smallest number of substitutions along its branches. In this sense, it is somewhat related to MDL (minimum description length) ideas. More formally, consider again an assignment (X_i) to the internal nodes of the tree, the notation being extended to the leaves. The letters at the leaves are fixed and determined by the observations. Then the parsimony cost (error) of the assignment is defined to be

$$\mathcal{E}_P((\mathsf{X}_i)|T) = \sum_{i \in I} \sum_{j \in N^+(i)} \delta(\mathsf{X}_i, \mathsf{X}_j). \tag{10.22}$$

In other words, a fixed cost is introduced for any nonidentical substitution, and the goal is to find an assignment and then a tree with minimal cost. For a given tree, a minimal assignment is also called a minimum mutation fit.

To see the connection with ML methods, recall that for a given tree, the probability of an assignment (X_i) is given by

$$\mathbf{P}((\mathsf{X}_i)|T) = p_r(\mathsf{X}_r) \prod_{i \in I} \prod_{j \in N^+(i)} p_{\mathsf{X}_j \mathsf{X}_i}(d_{ji}), \tag{10.23}$$

and the negative log-probability by

$$\mathcal{E}((\mathsf{X}_i)|T) = -\log p_r(\mathsf{X}_r) - \sum_{i \in I} \sum_{j \in N^+(i)} \log p_{\mathsf{X}_j \mathsf{X}_i}(d_{ji}). \tag{10.24}$$

If we let

$$p_{\mathsf{X}_j \mathsf{X}_i}(d_{ji}) = \begin{cases} a & \text{if } \mathsf{X}_j = \mathsf{X}_i \\ (1-a)/(|A|-1) & \text{if } \mathsf{X}_j \neq \mathsf{X}_i \end{cases}, \tag{10.25}$$

with $1/|A| < a < 1$, then it is easy to check that there exist two constants $\alpha > 0$ and β such that

$$\mathcal{E} = \alpha \mathcal{E}^P + \beta. \tag{10.26}$$

In fact, $\alpha = \log[a(|A|-1)/(1-a)]$ and $\beta = -|E| \log a + \log |A|$, where $|E|$ is the number of edges in the tree T. In other words, a minimum mutation fit for a given tree is equivalent to a Viterbi (most likely) assignment in an ML phylogeny model defined by (10.25) on the same tree topology. Thus parsimony can be viewed as an approximation to an ML approach to phylogeny. It implicitly assumes that changes are rare, uniform across the alphabet and across time. Thus, if the amount of change is small over the evolutionary times being considered, parsimony methods are statistically justified. Recursive algorithms for parsimony are well known [181]. In weighted parsimony [464], one can relax the assumption of uniform substitutions across the alphabet by introducing different weights $w(\mathsf{Y}, \mathsf{X})$ for each type of substitution. Again, it is easy to see that this can be viewed in a special ML context by letting, for any Y in A,

$$p_{\mathsf{YX}}(d_{ji}) = \frac{e^{-\alpha w(\mathsf{Y},\mathsf{X})}}{\sum_{\mathsf{Z} \in A} e^{-\alpha w(\mathsf{Z},\mathsf{X})}}. \tag{10.27}$$

Parsimony methods are computationally faster than ML methods, and probably this is one of the reasons for their wider use. Parsimony methods, however, can lead to wrong answers when evolution is rapid. Comparison tests between ML and parsimony can be conducted on artificial data generated by a probabilistic evolutionary model. For small samples, obviously both ML and parsimony methods can lead to the wrong phylogeny. In the case of large samples, however, phylogenetic trees are usually correctly reconstructed by ML, but not always by parsimony.

10.7 Extensions

In summary, we have reviewed the basic methodology for constructing phylogenies. The key point is that phylogenetic reconstruction is another application of Bayesian inference methods. The first step required is the construction of tractable probabilistic models of the evolutionary process. Markov substitution models form such a class. We have shown that the main algorithms for phylogenetic reconstruction currently in use, including parsimony methods, are special cases or approximations to ML inference within such a class. Algorithms for phylogenetic reconstruction are computationally intensive, especially when exploration of a large number of possible trees is required.

HMMs and probabilistic tree models of evolution have some complementary strengths and weaknesses. HMMs are needed to produce multiple alignments that are the starting point of evolutionary reconstruction algorithms. Evolutionary models are needed to regularize HMMs, that is to post-process the raw counts of multiple alignment columns to produce emission probabilities that are finely tuned for homology searches in large data bases. It is clear that one direction of research is to try to combine them, that is, to combine trees and alignments, phylogeny, and structure [247, 519], and come up with probabilistic models of the evolutionary process that allow insertions and deletions while *remaining computationally tractable* (see also tree-structured HMMs in appendix C).

A single Markovian substitution process is a bad model of evolution for all the reasons discussed in this chapter, and also because over large evolutionary times it produces a single equilibrium distribution. This is inconsistent with what we observe and, for instance, with the use of a mixture of Dirichlet distributions (Appendix D) as a prior for HMM emission probabilities. Past its relaxation time, a simple Markov model cannot give rise to clusters of distributions and to the different components of a Dirichlet mixture. To account for possible clusters, we must use the simple model over relatively short transient periods, or move to a higher class of evolutionary models.

What would a higher-level model of evolution look like? Imagine that we could observe multiple alignments produced at different times in the course of evolution, say every hundred million years. At each observation epoch, the alignment columns would represent a sample from a complex distribution over possible columns. It is this distribution that evolves over time. Thus in this class of higher-level models, evolution takes place on the distribution over distributions. A simple example of a model in this class can be constructed as follows. We can imagine that the original ($t = 0$) distribution on emission distributions is a Dirichlet mixture $\mathbf{P}(P) = \sum_i \lambda_i \mathcal{D}_{\alpha_i Q_i}(P)$ and that we have a simple Markovian substitution process *operating on the* Qs (and possibly additional processes operating on the αs and λs). At time t, the distribution becomes $\mathbf{P}(P) = \sum_i \lambda_i^t \mathcal{D}_{\alpha_i^t Q_i^t}(P)$. For instance, with a PAM matrix substitution model, if Q_i is chosen equal to the binary unit vector with a single 1 in position i (representing the letter X), then Q_i^t at time t is associated with the ith column of the corresponding PAM matrix (representing $p_{.\mathrm{X}}(t)$). Such a model is consistent with regularizing HMM emissions using Dirichlet mixtures associated with PAM matrix columns [497].