Chapter 3

Probabilistic Modeling and Inference: Examples

What are the implications of a Bayesian approach for modeling? For any type of model class, it is clear that the first step must be to make the likelihood $\mathbf{P}(D|M)$ and the prior $\mathbf{P}(M)$ explicit. In this chapter, we look at a few simple applications of the general probabilistic framework. The first is a very simple sequence model based on die tosses. All other examples in the chapter, including the basic derivation of statistical mechanics, are variations obtained either by increasing the number of dice or by varying the observed data.

3.1 The Simplest Sequence Models

The simplest, but not entirely trivial, modeling situation is that of a single coin flip. This model has a single parameter p and the data consist of a string, containing a single letter, over the alphabet $A = \{\mathsf{H}, \mathsf{T}\}$, H for head and T for tail. Since we are interested in DNA sequences, we shall move directly to a slightly more complex version with four letters, rather than two, and the possibility of observing longer strings.

3.1.1 The Single-Die Model with Sequence Data

The data D then consist of DNA strings over the four-letter alphabet $A = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. The simple model we want to use is to assume that the strings have been obtained by independent tosses of the same four-sided die (figure 3.1).

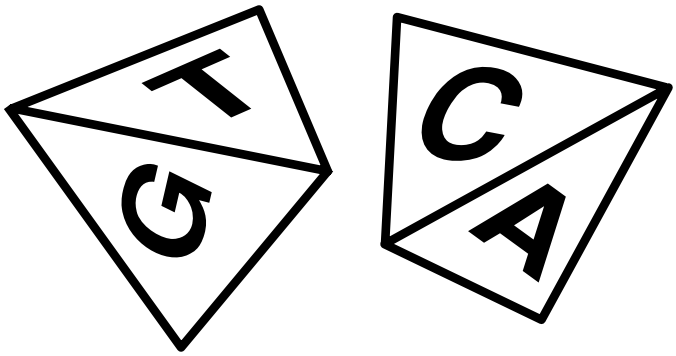

Figure 3.1: Two Views of the Four-Sided DNA Die Used to Generate of DNA Strings.

Because the tosses are independent and there is a unique underlying die, for likelihood considerations it does not really matter whether we have many strings or a single long string. So we assume that the data consist of a single observation sequence of length N: $D = \{O\}$, with $O = \mathsf{X}^1 \ldots \mathsf{X}^N$ and $\mathsf{X}^i \in A$. Our model M has four parameters $p_\mathsf{A}, p_\mathsf{C}, p_\mathsf{G}, p_\mathsf{T}$ satisfying $p_\mathsf{A} + p_\mathsf{C} + p_\mathsf{G} + p_\mathsf{T} = 1$. The likelihood is then given by

$$\mathbf{P}(D|M) = \prod_{\mathsf{X} \in A} p_\mathsf{X}^{n_\mathsf{X}} = p_\mathsf{A}^{n_\mathsf{A}} p_\mathsf{C}^{n_\mathsf{C}} p_\mathsf{G}^{n_\mathsf{G}} p_\mathsf{T}^{n_\mathsf{T}}, \tag{3.1}$$

where n_X is the number of times the letter X appears in the sequence O. The negative log-posterior is then

$$-\log \mathbf{P}(M|D) = -\sum_{\mathsf{X} \in A} n_\mathsf{X} \log p_\mathsf{X} - \log \mathbf{P}(M) + \log \mathbf{P}(D). \tag{3.2}$$

If we assume a uniform prior distribution over the parameters, then the MAP parameter estimation problem is identical to the ML parameter estimation problem and can be solved by optimizing the Lagrangian

$$\mathcal{L} = -\sum_{\mathsf{X} \in A} n_\mathsf{X} \log p_\mathsf{X} - \lambda (1 - \sum_{\mathsf{X} \in A} p_\mathsf{X}) \tag{3.3}$$

associated with the negative log-likelihood and augmented by the normalizing constraint. Here, and in the rest of the book, positivity constraints are checked directly in the results. Setting the partial derivatives $\partial \mathcal{L} / \partial p_\mathsf{X}$ to zero immediately yields $p_\mathsf{X} = n_\mathsf{X}/\lambda$. Using the normalization constraint gives $\lambda = N$ so that finally, as expected, we get the estimates

$$p_X^* = \frac{n_X}{N} \quad \text{for all} \quad \mathsf{X} \in A. \tag{3.4}$$

Note that the value of the negative log-likelihood per letter, for the optimal parameter set P^*, approaches the entropy (see appendix B) $\mathcal{H}(P^*)$ of P^* as $N \to \infty$:

$$\lim_{N\to\infty} -\frac{1}{N}\sum_{\mathsf{X}\in A} n_{\mathsf{X}} \log \frac{n_{\mathsf{X}}}{N} = -\sum_{\mathsf{X}\in A} p^*_{\mathsf{X}} \log p^*_{\mathsf{X}} = \mathcal{H}(P^*). \tag{3.5}$$

Another way of looking at these results is to say that except for a constant entropy term, the negative log-likelihood is essentially the relative entropy between the fixed die probabilities p_{X} and the observed frequencies n_{X}/N. In the section on statistical mechanics below, we will see how this is also related to the concept of free energy.

The observed frequency estimate $p_{\mathsf{X}} = n_{\mathsf{X}}/N$ is of course natural when N is large. The strong law of large numbers tells us that for large enough values of N, the observed frequency will almost surely be very close to the true value of p_{X}. But what happens if N is small, say $N = 4$? Suppose that in a sequence of length 4 we do not observe the letter A at all? Do we want to set the probability p_{A} to zero? Probably not, especially if we do not have any reason to suspect that the die is highly biased. In other words, our prior beliefs do not favor model parameters with value 0. As described in chapter 2, the corresponding natural prior in this case is not a uniform prior but rather a Dirichlet prior on the parameter vector P. Indeed, with a Dirichlet prior $\mathcal{D}_{\alpha Q}(P)$ the negative log-posterior becomes

$$-\log \mathbf{P}(M|D) = -\sum_{\mathsf{X}\in A} [n_{\mathsf{X}} + \alpha q_{\mathsf{X}} - 1] \log p_{\mathsf{X}} + \log Z + \log \mathbf{P}(D). \tag{3.6}$$

Z is the normalization constant of the Dirichlet distribution that does not depend on the probabilities p_{X}. Thus the MAP optimization problem is very similar to the one previously solved, except that the counts n_{X} are replaced by $n_{\mathsf{X}} + \alpha q_{\mathsf{X}} - 1$. We immediately get the estimates

$$p^*_{\mathsf{X}} = \frac{n_{\mathsf{X}} + \alpha q_{\mathsf{X}} - 1}{N + \alpha - |A|} \quad \text{for all} \quad \mathsf{X} \in A \tag{3.7}$$

provided this estimate is positive. In particular, the effect of the Dirichlet prior is equivalent to adding pseudocounts to the observed counts. With the proper choice of average distribution Q (for instance, Q uniform) and α, the estimates p^*_X can never be negative or 0. When Q is uniform, we say that the Dirichlet prior is *symmetric*. Notice that the uniform distribution over P is a special case of symmetric Dirichlet prior, with $q_{\mathsf{X}} = 1/\alpha = 1/|A|$. It is also clear from (3.6) that the posterior distribution $\mathbf{P}(M|D)$ is a Dirichlet distribution $\mathcal{D}_{\beta R}$ with $\beta = N + \alpha$ and $r_{\mathsf{X}} = (n_{\mathsf{X}} + \alpha q_{\mathsf{X}})/(N + \alpha)$.

The expectation of the posterior is the vector r_{X} which is slightly different from the MAP estimate (3.1). This suggests using an alternative estimate for

p_X, the predictive distribution or MP (mean posterior) estimate

$$p_X^* = \frac{n_X + \alpha q_X}{N + \alpha}. \tag{3.8}$$

This is in general a better choice. Here in particular the MP estimate minimizes the expected relative entropy distance $f(P^*) = \mathbf{E}(\mathcal{H}(P, P^*))$, where the expectation is taken with respect to the posterior $\mathbf{P}(P|D)$.

The die model with a single Dirichlet prior is simple enough such that one can proceed analytically with higher levels of Bayesian inference. For instance, we can compute the evidence $\mathbf{P}(D)$:

$$\mathbf{P}(D) = \int \mathbf{P}(D|w)\mathbf{P}(w)dw = \int_{\sum p_X = 1} \prod_{X \in A} p_X^{n_X + \alpha q_X - 1} \frac{\Gamma(\alpha)}{\Gamma(\alpha q_X))} dp_X. \tag{3.9}$$

This integral is very similar to the integral of a Dirichlet distribution and therefore can be easily calculated, yielding

$$\mathbf{P}(D) = \frac{\Gamma(\alpha)}{\prod_{X \in A} \Gamma(\alpha q_X)} \frac{\prod_{X \in A} \Gamma(\beta r_X)}{\Gamma(\beta)}, \tag{3.10}$$

This evidence is the ratio of the normalizing constants of the prior and posterior distributions.

We leave it as an exercise for the reader to continue playing with the Bayesian machinery. Useful exercises would be to find the values of α and q_X that maximize the evidence, to define a prior on α and q_X using hyperparameters, and to study MAP and MP estimates when the prior distribution is a mixture of Dirichlet distributions

$$\mathbf{P}(P) = \sum_i \lambda_i \mathcal{D}_{\alpha_i Q_i}(P) \tag{3.11}$$

(see also appendix D and [489]). In the latter case, the posterior is also a mixture of Dirichlet distributions. This is a general result: whenever the prior distribution is a mixture of conjugate distributions, the posterior is also a mixture of conjugate distributions.

3.1.2 The Single-Die Model with Counts Data

With the same die model, we now assume that available data consists of the counts themselves, $D = \{n_X\}$, rather than the actual sequence. A simple combinatorial calculation shows that the likelihood now has the form

$$\mathbf{P}(D|M) = \mathbf{P}(n_X|p_X) = \frac{N!}{\prod_{X \in A} n_X!} \prod_{X \in A} p_X^{n_X} \tag{3.12}$$

with $\sum_x n_x = N$. This is similar to (3.1), except for the factorial term that counts the number of ways the set of numbers (n_x) can be realized in a sequence of length N. This distribution on the counts n_x generated by a simple die model is also called a *multinomial distribution*, generalizing the notion of binomial distribution associated with coin (that is, two-sided die) flips. With a little abuse, the die model itself will sometimes be called a multinomial model.

With a Dirichlet prior $\mathcal{D}_{\alpha Q}(P)$ on the parameter vector P, a calculation similar to the one above shows that the posterior distribution on P is also a Dirichlet distribution $\mathcal{D}_{\beta R}(P)$ with $\beta = N + \alpha$ and $r_x = (n_x + \alpha q_x)/\beta$. Not surprisingly, the MAP and MP estimates P^* are identical to (3.7) and (3.8).

We now consider the distribution that a fixed vector P induces on the counts n_x. Taking the logarithm of (3.12) and using Stirling's approximation formula for factorials

$$n! \approx (\frac{n}{e})^n \sqrt{2\pi n}, \tag{3.13}$$

we get

$$\log(\mathbf{P}(D|P) \approx C - \mathcal{H}(n_x/N, p_x), \tag{3.14}$$

where C is a constant independent of n_x and $\mathcal{H}$ is the relative entropy between the empirical distribution and P. When P is uniform except for constant terms, the relative entropy above reduces to the entropy of the empirical distribution. Therefore in this case

$$\mathbf{P}(D|P) \approx \frac{e^{\mathcal{H}(n_x/N}}{Z}. \tag{3.15}$$

This is called the *entropic* distribution. In other words, a uniform P induces an entropic distribution over the counts n_x, that is, over the space of all possible histograms. As we will see in section 3.2, this is one of the standard justifications for the MaxEnt principle that amounts to using an entropic prior. Notice the similarities but also the differences between a Dirichlet distribution and an entropic distribution

$$\frac{\exp(-\sum_x p_x \log p_x)}{Z} \tag{3.16}$$

over P. We leave it as an exercise to show that if P has an entropic prior, the posterior after observing n_x is not entropic, nor Dirichlet. The entropic distribution is not the conjugate of a multinomial. With an entropic prior, the MAP estimate is still of the form $p_x^* = n_x/N$.

While the simple die model is of course very crude, it is important to note that this is exactly the model we adopt when we compute first-order statistics, that is, the proportion of each letter in a given family of sequences, such as exons, introns, or a protein family. This can be viewed as a first step in an iterative modeling process, and therefore the performance of subsequent models must be evaluated with respect to the first-order model. The multiple-die model of the next section and, in chapter 7, hidden Markov models (HMMs)

are just slightly more complex generalizations of the simple die model. The simple die model can trivially be extended by having strings of letters on each face. This is equivalent to extending the alphabet. For instance, one can use a die with 64 faces to model DNA triplets.

3.1.3 The Multiple-Die Model with Sequence Data

Another simple sequence model is the multiple-die model. Here the data consist of K sequences, each of length N. For instance, the reader could think of a multiple alignment of K sequences in which case the gap symbol "–" could be considered one of the symbols in the alphabet. In the multiple-die model we assume that there are N independent dice, one for each position, and that each sequence is the result of flipping the N dice in a fixed order. Let p_{X}^i denote the probability of producing the letter X with die number i, and n_{X}^i the number of times the letter X appears in position i. Because the dice and the sequences are assumed to be independent, the likelihood function is

$$P(D|M) = \prod_{i=1}^{N} \prod_{\mathsf{X} \in A} p_{\mathsf{X}}^{n_{\mathsf{X}}^i}. \tag{3.17}$$

With uniform prior across all dice, a calculation identical to the single-die case yields

$$p_{\mathsf{X}}^{i*} = \frac{n_{\mathsf{X}}^i}{K} \quad \text{for all} \quad \mathsf{X} \in A. \tag{3.18}$$

Again we leave it as an exercise for the reader to study the effect of Dirichlet priors on this model, and to consider possible generalizations (see also [376]). A well-known class of models used in language modeling is the n-gram models. In an n-gram model, there are $|A|^{n-1}$ dice. Each die is associated with a different prefix of length $n-1$. Each die is a simple die with $|A|$ faces, one letter per face. Sequences are generated by scanning a window of length n, selecting the die associated with the current prefix, and flipping it at random. Thus the choice of the die to be flipped is not independent of the previous flips. These n-gram models can be viewed as Markov models of order equal to the length of the prefix, also called the "memory" of the model. The single-die model has memory of length 0. There exist also variants with variable memory length (see [448] for an example with application to biological sequences), as well as mixtures of higher-order Markov models, also called interpolated Markov models. Higher-order models are computationally more expensive, with the number of possible prefixes growing very rapidly with the size of the alphabet and the memory length. With the small DNA alphabet, however, Markov models of order 5 or so remain feasible.

3.2 Statistical Mechanics

There are at least five good reasons to understand the rudiments of statistical mechanics in connection with machine learning and computational biology. First, statistical mechanics can be viewed as one of the oldest and best examples of Bayesian reasoning [280, 281], although the presentation often given is slightly flawed in our opinion because of the confusion between MaxEnt and Bayes. Second, statistical mechanics has traditionally been concerned with deriving the statistical macroscopic properties of large ensembles of simple microscopically interacting units—the equilibrium behavior, the phase transitions, and so on. The results and techniques of statistical mechanics are useful in understanding the properties and learning evolution of a number of graphical models used in machine learning [252, 482, 50]. Statistical mechanical models have also been applied directly to biological macromolecules—for instance, in the protein-folding problem (see [151] for a review). Finally, statistical mechanics is useful for understanding several algorithms that are fundamental for machine learning, such as simulated annealing and the EM algorithms described in chapter 4.

Here we give a Bayesian derivation of statistical mechanics from first principles, and develop the basic concepts, especially those of the Boltzmann-Gibbs distribution and free energy, that will be needed in the next chapters. In the basic statistical mechanics framework, one considers a stochastic system that can be in a number of "microscopic" states: $\mathbf{S} = \{s_1, \ldots, s_{|S|}\}$, with p_s denoting the probability of being in state s for a distribution $P = (p_s)$. This can be viewed as a die model $M(w)$, with parameters $w = p_s$, although for the time being it is not necessary to assume that the tosses are independent. The key difference from the examples above is in the data. The faces of the die, the microscopic states, are not observable but act as hidden variables. Instead, we assume that there is a function $f(s)$ of the states and that the only "macroscopic" observable quantity, the data, is the expectation or average of f. So, with a slight abuse of notation, in this section we write $D = \mathbf{E}(f) = \sum_s p_s f(s)$.

Very often in statistical mechanics the states have a microscopic structure so that $s = (x_1, \ldots, x_n)$, where the x_i are local variables. For instance, the x_i can be binary spin variables, in which case $|S| = 2^n$. Likewise, f is typically the energy of the system and can be written as a quadratic function in the local variables: $f(s) = f(x_1, \ldots, x_n) = \sum_{ij} w_{ij} x_i x_j + \sum_i w_i x_i$. The interaction parameters w_{ij} can be local, as in the case of spins on a lattice, or long-range, and are related to the underlying graphical model. While such assumptions are important in modeling particular systems and developing a detailed theory, they will not be needed in the following sections. The first question we can ask is: Given the observation of the average of f, what can we say about the state distribution P?

3.2.1 The Boltzmann-Gibbs Distribution

Standard Derivation

Most standard treatments at this point are based on the maximum entropy principle. Without any additional information, one ought to choose the distribution P that satisfies the constraint $\sum_s f(s)p_s = D$ and has the highest entropy, because this is the solution that is the most "spread out" and makes the fewest additional assumptions. This problem can easily be solved by writing down the Lagrangian $\mathcal{L}$, which consists of a linear combination of the function being optimized with the relevant constraints:

$$\mathcal{L} = -\sum_s p_s \log p_s - \lambda(\sum_s p_s f(s) - D) - \mu(\sum_s p_s - 1). \tag{3.19}$$

By equating the partial derivatives of $\mathcal{L}$ with respect to p_s to 0, we immediately find that the only solution distributions are of the form

$$p_s^*(\lambda) = \frac{e^{-\lambda f(s)}}{Z(\lambda)}, \tag{3.20}$$

where the normalizing factor $Z(\lambda) = \sum_s e^{-\lambda f(s)}$ is called the partition function. In statistical mechanics, the Lagrange multiplier is related to the temperature T of the system by the definition $\lambda = 1/kT$, where k is the Boltzmann constant. For all our purposes here we will not need to consider the temperature and will work directly with the parameter λ. Note, however, that λ, and therefore the temperature, are entirely determined by the observation D, since we must have

$$\sum_s \frac{e^{-\lambda f(s)}}{Z(\lambda)} f(s) = D. \tag{3.21}$$

Often, it will even be sufficient to assume that $\lambda = 1$. The optimal distribution P^* is called the Boltzmann-Gibbs distribution of the system. It is important to realize that *any* distribution P can be represented as a Boltzmann-Gibbs distribution, at least at a fixed temperature, by using an energy function proportional to $-\log P$. It is also easy to see that a similar formula is derived when there are multiple linear constraints on the parameters p_s.

While the Boltzmann-Gibbs distribution is very useful, from a Bayesian standpoint the standard derivation is not entirely satisfactory for three reasons: (1) The prior distribution is not explicit. As a result, how would one incorporate additional prior information on the p_s, such as knowing that the first state occurs more frequently than the others? (2) The probabilistic model is not explicit. In particular, how one would calculate the likelihood $\mathbf{P}(D|p_s)$; (3) The justification for the use of MaxEnt is weak. In particular, is there any

connection between MaxEnt and ML or MAP estimation? In all fairness, the use of MaxEnt is partially justified by the combinatorial argument given above, which shows that maximizing the entropy is essentially equivalent to maximizing the number of possible realizations $N!/\prod_s n_s!$ when the tosses are independent [282]. In this sense, the MaxEnt solution is the one that can be realized in the largest number of ways. Such an argument, however, is based only on the number of realizations and does not take into account their relative probabilities. We now address these three criticisms.

Bayesian Derivation

The main problem with the standard derivation is that the probabilistic model is not really explicit. In particular, the likelihood function $\mathbf{P}(D|p_s)$ is not clearly defined and little progress can be made in this direction without considering actual runs of the system. Thus we must enlarge the initial setup by assuming that there is a fixed number N that is very large and that the system is observed over such a period. Variable observation times could also be considered but would further complicate the analysis. Accordingly, we decide to parameterize the model using the counts n_s. Note also that what is really observed is $D = (\sum_s n_s f(s))/N \neq \sum_s p_s f(s)$.

Several priors on the counts n_s are possible. As we have seen, a natural prior would be to use a Dirichlet prior on n_s/N. A nonsymmetric Dirichlet prior could easily incorporate any additional information regarding the frequency of occurrence of any particular state. We leave it as an exercise for the reader to calculate the posterior obtained with a Dirichlet prior, but this is obviously not the Boltzmann-Gibbs solution. For instance, if the prior is uniform and $f(s_1) = D$, then the vector $(N, 0, \ldots, 0)$, with the lowest possible entropy, maximizes the probability of the data by rendering it certain! Here we rather decide to use the entropic prior, which is the distribution on n_s obtained when P is uniform. Again, such a prior is best justified when the runs are independent, that is, the underlying probabilistic model is a simple die with $|S|$ faces. Although in what follows we confine ourselves to this zeroth-order Markov model, one could consider higher-order Markov models. A Markov model of order 1, for instance, would include a different set of parameters associated with the transition probabilities from state to state, equivalent to $|S|$ dice. Certain aspects of Markov models of order 1 and Boltzmann-Gibbs distributions are treated in chapter 4.

The likelihood function is then trivial and has value 1 or 0, depending on whether or not $D = \sum_s f(s) n_s/N$. We can finally proceed with the first step of Bayesian inference, and estimate the parameters n_s by MAP estimation. Using

the formalism introduced earlier this leads immediately to the Lagrangian

$$\mathcal{L} = -\sum_s \frac{n_s}{N} \log \frac{n_s}{N} - \lambda(\sum_s (f(s)\frac{n_s}{N} - D) - \mu(\sum_s n_s - N), \tag{3.22}$$

where the entropy act as a regularizer. This is of course virtually identical to (3.19) and yields a MAP Boltzmann-Gibbs distribution for n_s/N. A similar result can be derived using the parameters p_s instead of n_s, but in a more cumbersome way in terms of both justifying the entropic prior and calculating the likelihood function, since different values of n_s can be consistent with D.

In conclusion, the Boltzmann-Gibbs distribution corresponds to a first step of Bayesian inference by MAP, with an entropic prior. Therefore MaxEnt is best viewed not as an universal principle but simply as a shortcut for the first level of Bayesian inference in a multinomial setting associated with an entropic prior. Such prior can be challenged and examples can be constructed where MaxEnt leads to the "wrong" solution. We leave it as an exercise for the reader to construct such examples and envision how to proceed again with higher steps of Bayesian inference (hyperparameters, integration over priors).

3.2.2 Thermodynamic Limit and Phase Transitions

The temperature is a good example of an *intensive* quantity, that is, a quantity that by definition is independent of system size. On the other hand, *extensive* quantities, such as the energy, grow with the size of the system. For large systems with local interactions, this growth is typically linear with the size of the system. Thus the value of an extensive quantity per unit of volume tends to a limiting value as the size of the system goes to infinity, the so-called *thermodynamic limit*.

One of the main goals of statistical mechanics is to estimate the thermodynamic limit of macroscopic quantities, that is, to approximate expectations with respect to the Boltzmann-Gibbs distribution. In particular, one of the main goals is to approximate the partition function $Z(\lambda)$, since this function contains most of the relevant information about the system. In particular, it is easy to show that all the moments of the function f can be computed from $Z(\lambda)$, and more precisely from its logarithm. For instance, for the first two moments, the mean and the variance, an elementary calculation gives

$$\mathbf{E}(f) = -\frac{\partial}{\partial \lambda} \log Z(\lambda) \tag{3.23}$$

$$\mathbf{Var}(f) = -\frac{\partial^2}{\partial \lambda^2} \log Z(\lambda). \tag{3.24}$$

Likewise, the entropy of the Boltzmann-Gibbs distribution P^* can be expressed as

$$\mathcal{H}(P^*) = -\sum_s P^*(s) \log P^*(s) = \log Z(\lambda) + \lambda \mathbf{E}(f). \tag{3.25}$$

Another central topic of statistical mechanics is the study of phase transitions, that is abrupt changes in the behavior of the system as some of the parameters, especially the temperature T or equivalently λ, are varied. A first-order phase transition is said to occur at a critical value λ_C if $\mathbf{E}(f)$ is discontinuous at λ_C. A second-order phase transition occurs at λ_C if $\mathbf{E}(f)$ is continuous but $\mathbf{Var}(f)$ is discontinuous. The study of phase transitions is also important in learning theory [252, 482], but this is beyond the scope of this book.

3.2.3 The Free Energy

The logarithm of the partition function is called the *free energy* because of its important role (see (3.23), (3.24), and (3.25)). More precisely, the free energy $\mathcal{F} = \mathcal{F}(f, \lambda) = \mathcal{F}(\lambda)$ is defined to be

$$\mathcal{F}(\lambda) = -\frac{1}{\lambda} \log Z(\lambda). \tag{3.26}$$

The above formula can obviously be rewritten in terms of the free energy. For instance,

$$\mathcal{H}(P^*) = -\lambda \mathcal{F}(\lambda) + \lambda \mathbf{E}(f). \tag{3.27}$$

This is equivalent to

$$\mathcal{F}(\lambda) = \mathbf{E}(f) - \frac{1}{\lambda} \mathcal{H}(P^*), \tag{3.28}$$

which is sometimes used as an alternative definition of the free energy. In this definition, the free energy depends on the function f, the parameter λ, and the distribution P^* over states. The definition therefore can be extended to any other distribution $Q(s)$:

$$\mathcal{F}(f, Q, \lambda) = \mathcal{F}(Q, \lambda) = \mathbf{E}_Q(f) - \frac{1}{\lambda} \mathcal{H}(Q), \tag{3.29}$$

where $\mathbf{E}_Q$ denotes expectations with respect to the distribution Q. Here we drop the dependency on f, but the choice of f as a negative log-probability is important in statistical applications, such as the derivation of the EM algorithm, as described below and in chapter 4. By comparing this free energy with the Lagrangian above, it is also clear that the Boltzmann-Gibbs distribution is equivalently characterized as the distribution that minimizes the free energy.

Consider now any two distributions $Q(s)$ and $R(s)$. We want to be able to compare their free energies. A simple calculation gives

$$\mathcal{F}(Q,\lambda) - \mathcal{F}(R,\lambda) = \sum_s [Q(s) - R(s)][f(s) + \frac{1}{\lambda}\log R(s)] + \frac{1}{\lambda}\mathcal{H}(Q,R), \quad (3.30)$$

where $\mathcal{H}(Q,R) = \sum_s Q(s)\log(Q(s)/R(s))$ is the relative entropy between Q and R.

It is useful to remark that if we take the energy of s to be the negative likelihood $f(s) = -\log R(s)$, where R is some distribution over the states, then the Boltzmann-Gibbs distribution is proportional to $R^\lambda(s)$. In particular, at $\lambda = 1$ the Boltzmann-Gibbs distribution of the system is R itself: $P^*(s,1) = R$, and the free energy reduces to 0. Furthermore, for any other distribution Q, the difference in free energies is then equal to the relative entropy

$$\mathcal{F}(Q,1) - \mathcal{F}(R,1) = \mathcal{H}(Q,R). \quad (3.31)$$

Since the relative entropy is always nonnegative, then $\mathcal{F}(Q,1) \geq \mathcal{F}(R,1)$, with equality if and only if $Q = R$. Again the Boltzmann-Gibbs distribution minimizes the free energy. It is also important to note that there is nothing special about the $\lambda = 1$ temperature. We could, for instance, define $f(s) = -\log R(s)/\lambda$, and then obtain $\mathcal{F}(Q,\lambda) - \mathcal{F}(R,\lambda) = \mathcal{H}(Q,R)/\lambda$.

3.2.4 The Hidden Variables Case

In many modeling situations there are hidden/unobserved/latent variables or causes denoted by H. If D denotes the data, we assume that there is available a joint distribution on the hidden and observed variables $\mathbf{P}(D,H|w)$, parameterized by w. In the case of interest to us, w as usual denotes the parameters of a model. From a statistical mechanics perspective, we can consider that the states of the system are the values assumed by the hidden variables. If we define f by

$$f(H) = -\log \mathbf{P}(D,H|w), \quad (3.32)$$

then at $\lambda = 1$ the Boltzmann-Gibbs distribution is given by the posterior

$$P^* = P^*(H,1) = \mathbf{P}(H|D,w) \quad (3.33)$$

and the free energy by

$$\mathcal{F}(P^*,1) = -\log \mathbf{P}(D|w), \quad (3.34)$$

which is the negative log-likelihood of the data. Furthermore, for any other distribution Q, the difference in free energies is given by

$$\mathcal{F}(Q,1) - \mathcal{F}(P^*,1) = \mathcal{H}(Q,P^*) \quad (3.35)$$

or

$$\log \mathbf{P}(D|w) = -\mathcal{F}(Q,1) + \mathcal{H}(Q,P^*). \tag{3.36}$$

In order to maximize the data likelihood, when the posterior $\mathbf{P}(H|D,w)$ and the corresponding expectations are difficult to calculate, one can sometimes use a suboptimal strategy based on a different family of distributions Q for which calculations are more tractable, without departing too much from the true posterior. This idea of minimizing the free energy term $\mathcal{F}(Q,\lambda)$ is developed in [146, 255] and in the section on variational methods in appendix A.

This page intentionally left blank

Chapter 4

Machine Learning Algorithms

4.1 Introduction

In this chapter we cover the main algorithms for machine-learning applications that will be used thoughout the rest of the book. We briefly describe each of the algorithms and provide pointers to the vast literature on this topic.

Once a parameterized model $M(w)$ for the data has been constructed, we have seen that the next steps are the following:

1. The estimation of the complete distribution $\mathbf{P}(w, D)$ and the posterior $\mathbf{P}(w|D)$
2. The estimation of the optimal set of parameters w by maximizing $\mathbf{P}(w|D)$, the first level of Bayesian inference
3. The estimation of marginals and expectations with respect to the posterior, that is, for instance, of integrals of the form $\mathbf{E}(f) = \int f(w)\mathbf{P}(w|D)dw$, the higher levels of Bayesian inference

Thus the algorithms can be subdivided into three categories, depending on whether the goal is to estimate a probability density, one of its modes, or the corresponding expectations. For practical reasons we shall use this distinction, although it is somewhat arbitrary. Indeed, any problem can be reformulated as an optimization problem, and the probability of an event is the expectation of the corresponding indicator function: $\mathbf{P}(A) = \mathbf{E}(1_A)$. Likewise, dynamic programming, which is often used to estimate sequence data likelihoods, can be viewed as an optimization technique.

In section 4.2, we briefly review dynamic programming, one of the key algorithms in sequence analysis, and its application in the estimations of sequence likelihoods. In the following two sections we look at algorithms for

the optimization of $\mathbf{P}(w|D)$, including gradient descent and EM (expectation maximization)/GEM (generalized expectation maximization). The treatment of simulated annealing is postponed to section 4.6, after the treatment in section 4.5 of Monte Carlo Markov chain methods (MCMC) for the stochastic sampling of high-dimensional distributions and the computation of the corresponding expectations. This is because simulated annealing relies heavily on stochastic sampling. In section 4.7 we take a brief look at evolutionary algorithms, and conclude in section 4.8 with several complements and practical aspects.

4.2 Dynamic Programming

Dynamic programming [66] is to a very general optimization technique that can be applied any time a problem can be recursively subdivided into two similar subproblems of smaller size, such that the solution to the larger problem can be obtained by piecing together the solutions to the two subproblems. The prototypical problem to which dynamic programming can be applied is that of finding the shortest path between two nodes in a graph. Clearly the shortest path from node A to node B, going through node C, is the concatenation of the shortest path from A to C with the shortest path from C to B. This is also called the "Bellman principle." A general solution to the original problem is then constructed by recursively piecing together shorter optimal paths.

Dynamic programming and its many variations are ubiquitous in sequence analysis. The Needleman–Wunch and Smith–Waterman algorithms [401, 481, 492], as well as all other sequence-alignment algorithms such as the Viterbi decoding algorithm of electrical engineers, are examples of dynamic programming. Alignment algorithms can be visualized in terms of finding the shortest path in the appropriate graph with the appropriate metric. Aligning two sequences of length of N requires finding a shortest path in a graph with N^2 vertices. Since dynamic programming essentially requires visiting all such vertices once, it is easy to see that its time complexity scales as $O(N^2)$.

In chapters 7 and 8, dynamic programming and the Viterbi algorithm are heavily used to compute likelihoods and align sequences to HMMs during the training and exploitation phases. Accordingly, we give there a detailed derivation of the corresponding algorithms. Other variations on dynamic programming used in other chapters are sketched or left as an exercise. Because dynamic programming is very well known and is at the root of many conventional algorithms for sequence analysis, we refer the reader to the abundant literature on the topic (in particular [550] and references therein). Reinforcement-learning algorithms are also another important class of learning algorithms that can be viewed as generalizations of dynamic programming ideas [298].

4.3 Gradient Descent

Often we are interested in parameter estimation, that is, in finding the best possible model $M(w)$ that minimizes the posterior $f(w) = -\log \mathbf{P}(w|D)$, or possibly the likelihood $-\log \mathbf{P}(D|w)$. Whenever a function $f(w)$ is differentiable, one can try to find its minima by using one of the oldest optimization algorithms, gradient descent. As its name indicates, gradient descent is an iterative procedure that can be expressed vectorially as

$$w^{t+1} = w^t - \eta \frac{\partial f}{\partial w^t}, \tag{4.1}$$

where η is the step size, or learning rate, which can be fixed or adjusted during the learning process.

While the general gradient-descent principle is simple, in complex parameterized models it can give rise to different implementations, depending on how the gradient is actually computed [26]. In graphical models, this often requires the propagation of information "backwards." As we will see in the next chapters, this is the case for gradient-descent learning applied to neural networks (the backpropagation algorithm) and to hidden Markov models (the forward–backward procedure). Obviously the outcome of a gradient-descent procedure depends on the initial estimate. Furthermore, if the function being optimized has a complex landscape, gradient descent in general will terminate in a local minimum rather than a global one. Whenever feasible, therefore, it is wise to run the procedure several times, with different starting points and learning rates.

It is well known that there are situations where plain gradient descent can be slow and inefficient. To overcome such problems, a number of variations on gradient descent are possible, such as conjugate gradient descent, that use second-order information or more complex directions of descent constructed from the current gradient and the history of previous directions. Additional details and references can be found in [434]. In spite of its relative crudeness, gradient descent remains useful, easy to implement, and widely used.

4.3.1 Random-Direction Descent

There are a number of other descent procedures that do not necessarily follow the line of steepest descent. These can be useful when the gradient is difficult to compute, when the physics of the hardware directly supports such approaches, or when escaping from local minima is important. For instance, one could generate a random perturbation of the current estimate and accept it only if it lies below the current level. If it does not, the opposite perturbation is accepted, or alternatively a new perturbation is tried. In *line search*

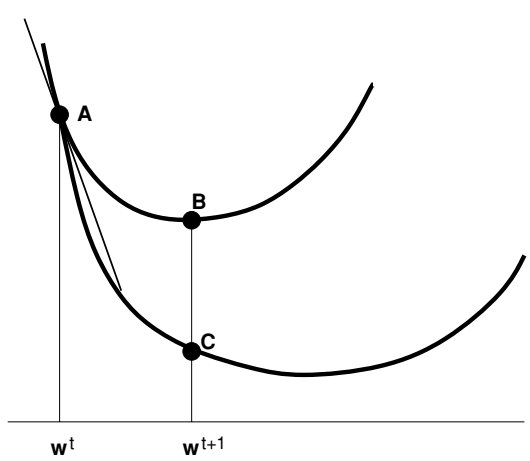

Figure 4.1: Three Consecutive Points of the EM Algorithm. Starting from w^t, in order to minimize the likelihood surface $F(w) = -\log \mathbf{P}(D|w)$, the EM algorithm minimizes a surface $G(w)$, with $G(w^t) = F(w^t) = A$. The surface G dominates the surface F, and the two surfaces have the same gradient at $w = w^t$. w^{t+1} corresponds to point B, the minimum of G. Point C is determined by calculating the new posterior on the hidden variables $\mathbf{P}(H|D, w^{t+1})$.

algorithms, once a direction of descent has been determined, the lowest point along that direction is searched before generating a new direction. Ideas related to line search and random descent are also found in the EM algorithm in the next section, and in evolutionary algorithms toward the end of the chapter.

4.4 EM/GEM Algorithms

Another important class of optimization algorithms is the expectation maximization (EM) and generalized expectation maximization (GEM) algorithms [147, 387]. Such algorithms have been used in many different applications and also in sequence analysis [352, 113]. In the case of HMMs, the EM algorithm is also called the Baum–Welch algorithm [54]. Since the usefulness of these algorithms goes beyond HMMs, we give here a general treatment of EM/GEM algorithms, using the concept of free energy of chapter 3, along the lines suggested in [400].

The EM algorithm is useful in models and situations with hidden variables. Typical examples of hidden variables are missing or unobservable data, mixture parameters in a mixture model, and hidden node states in graphical models (hidden units in NNs, hidden states in HMMs). If D denotes the data, we

assume that there is available a parameterized joint distribution on the hidden and observed variables $\mathbf{P}(D,H|w)$, parameterized by w. In the case of main interest to us, w denotes, as usual, the parameters of a model. Let us assume that the objective is to maximize the likelihood $\log \mathbf{P}(D|w)$. The same ideas can easily be extended to the case of MAP estimation. Since in general it is difficult to optimize $\log \mathbf{P}(D|w)$ directly, the basic idea is to try to optimize the expectation $\mathbf{E}(\log \mathbf{P}(D|w))$:

$$\mathbf{E}(\log \mathbf{P}(D|w)) = \mathbf{E}(\log \mathbf{P}(D,H|w) - \log \mathbf{P}(H|D,w)). \tag{4.2}$$

The EM algorithm is an iterative algorithm that proceeds in two alternating steps, the E (expectation) step and the M (maximization) step. During the E step, the distribution of the hidden variables is computed, given the observed data and the current estimate of w. During the M step, the parameters are updated to their best possible value, given the presumed distribution on the hidden variables. Starting with an estimate w^0 at time 0, the EM algorithm can be written more precisely at time t as follows:

1. E step: Compute the distribution $Q^*(H)$ over H such that $Q^*(H) = \mathbf{P}(H|D,w^{t-1})$.

2. M step: Set $w^t = \arg_w \max \mathbf{E}_{Q^*}[\log \mathbf{P}(D,H|w)]$.

As seen in chapter 3, if we define the energy of a hidden configuration H to be $f(H) = -\log \mathbf{P}(D,H|w)$, then the Boltzmann–Gibbs distribution at $\lambda = 1$ is given by the posterior $\mathbf{P}(H|D,w)$. In other words, the first step of the EM algorithm is the minimization, with respect to Q, of the free energy

$$\mathcal{F}(f,Q,1) = \mathcal{F}(w,Q,1) = \mathcal{F}(w,Q) = \mathbf{E}_Q(f) - \mathcal{H}(Q). \tag{4.3}$$

The second step is a minimization with respect to f, that is, with respect to w. Thus, omitting the constant parameter $\lambda = 1$, the EM algorithm can be rephrased in the following form:

1. E step: Compute the Boltzmann–Gibbs distribution $Q^*(H)$ that minimizes $\mathcal{F}(w^{t-1},Q)$.

2. M step: Set w^t to minimize $\mathcal{F}(w^{t-1},Q^*)$.

It is important to note that although Q^* depends on w, Q^* is held *fixed* during the M step. Also from chapter 3, the value of the free energy for the Boltzmann–Gibbs distribution is equal to the negative log-likelihood of the data, $\mathcal{F}(w,Q^*,1) = -\log \mathbf{P}(D|w)$.

In summary, the EM algorithm is an optimization procedure on the free energy $\mathcal{F}$ that proceeds by alternate optimization in the Q and w directions. Hence it produces a sequence of estimates of the form

$$(w^t, Q^t) \rightarrow (w^t, Q^{t+1}) \rightarrow (w^{t+1}, Q^{t+1}) \rightarrow (w^{t+1}, Q^{t+2}) \ldots, \tag{4.4}$$

satisfying, for every t

1. $\mathcal{F}(w^t, Q^t) \geq \mathcal{F}(w^t, Q^{t+1}) \geq \mathcal{F}(w^{t+1}, Q^{t+1}) \geq \mathcal{F}(w^{t+1}, Q^{t+2}) \geq \ldots$
2. $\mathcal{F}(w^t, Q^{t+1}) = -\log \mathbf{P}(D|w^t)$
3. $Q^{t+1} = \mathbf{P}(H|D, w^t)$ and $\mathcal{F}(w^t, Q^t) - \mathcal{F}(w^t, Q^{t+1}) = \mathcal{H}(Q^t, Q^{t+1})$

It is then clear that, except for rare saddle points, the EM algorithm converges to a local minimum of $\mathcal{F}(w, Q)$ which is also a local minimum of $-\log \mathbf{P}(D|M)$, as desired.

It is instructive to look at the EM algorithm from the point of view of w only. Suppose we have an estimate w^t at time t, with the corresponding likelihood $-\log \mathbf{P}(D|w^t)$. Then

$$w^{t+1} = \arg_w \min[-\mathbf{E}_{Q^{t+1}} \log \mathbf{P}(H, D|w)] \tag{4.5}$$

with $Q^{t+1} = \mathbf{P}(H|D, w^t)$. By writing $\mathbf{P}(H, D|w) = \mathbf{P}(H|D, w)\mathbf{P}(D|w)$ and collecting terms, this is equivalent to

$$w^{t+1} = \arg_w \min[-\log \mathbf{P}(D|w) + \mathcal{H}(Q^{t+1}, \mathbf{P}(H|D, w))]. \tag{4.6}$$

Thus, starting from w^t, the EM algorithm finds the minimum of the surface $G(w) = -\log \mathbf{P}(D|w) + \mathcal{H}(Q^{t+1}, \mathbf{P}(H|D, w))$ that dominates the surface $F(w) = -\log \mathbf{P}(D|w)$ that one really wants to optimize. Thus the optimization procedure tends to maximize the likelihood, without going too far from the current value of $\mathbf{P}(H|D, w^t)$, to keep the cross-entropy term small. Taking derivatives vectorially yields

$$\frac{\partial G}{\partial w} = -\frac{\partial \log \mathbf{P}(D|w)}{\partial w} - \sum_H Q^{t+1}(H) \frac{\partial \mathbf{P}(H|D, w)/\partial w}{\mathbf{P}(H|D, w)}. \tag{4.7}$$

The second term in the right-hand side cancels when $w = w^t$. Therefore,

$$\frac{\partial G}{\partial w}|_{w=w^t} = -\frac{\partial \log \mathbf{P}(D|w)}{w}|_{w=w^t}. \tag{4.8}$$

The tangent to the new surface G is identical to the tangent to the original surface $F(w) = -\log \mathbf{P}(D|w)$. Thus gradient descents on the negative log-likelihood and the EM algorithm are descending in the same directions (figure

4.1). The EM algorithm is further simplified when the distribution $\mathbf{P}(D, H|w)$ belongs to the exponential family. In particular, in this case, the function G is always convex. The particularization of the EM algorithm to exponential distributions is left as an exercise.

Finally, any algorithm that descends the function G (without necessarily finding its minimum), and hence improves the likelihood, is called a GEM (generalized EM) algorithm [147]. The geometric picture above shows that gradient descent on the likelihood can be viewed as a GEM algorithm (see also [400] for a discussion of how the E and M steps can be executed partially, for instance, online).

4.5 Markov-Chain Monte-Carlo Methods

Markov-chain Monte-Carlo (MCMC) methods belong to an important class of stochastic methods that are related to statistical physics and are increasingly used in Bayesian inference and machine learning [578, 202, 396, 520, 69]. Recall that one of the basic goals derived from the general Bayesian framework is to compute expectations with respect to a high-dimensional probability distribution $P(x_1, \ldots, x_n)$, where the x_i can be the values of model parameters or hidden variables, as well as observed data. The two basic ideas behind MCMC are very simple. The first idea (Monte Carlo) is to approximate such expectations by

$$\mathbf{E}(f) = \sum_{x_1,\ldots,x_n} f(x_1, \ldots, x_n) P(x_1, \ldots, x_n) \approx \frac{1}{T} \sum_{t=0}^{T} f(x_1^t, \ldots, x_n^t) \tag{4.9}$$

for large T, provided $(x_1^t, \ldots, x_n^t)$ are sampled according to their distribution $P(x_1, \ldots, x_n)$. In order to sample from P, the second basic idea is to construct a Markov chain having P as its equilibrium distribution, then simulate the chain and try to sample from its equilibrium distribution.

Before we proceed with the rudiments of Markov chains, it is worth noting a few points. The mean of the estimator on the right-hand side of (4.9) is $\mathbf{E}(f)$. If the samples are independent, its variance is $\mathbf{Var}(f)/T$. In this case, the precision of the estimate does not depend on the dimension of the space being sampled. Importance sampling and rejection sampling are two well-known Monte-Carlo algorithms for generating independent samples that will not be reviewed here. Both algorithms tend to be inefficient in high-dimensional state spaces. The samples created using Markov-chain methods are not independent. But at equilibrium they are still representative of P. The dependence of one sample on the previous one is the key to the better efficiency of MCMC methods with higher-dimensional spaces. After all, if P is differentiable or

even just continuous, the probability $P(x_1, \ldots, x_n)$ of a sample provides information about its neighborhood. This remains true even in cases where P can be computed efficiently only up to a constant normalizing factor. Finally, MCMC methods, like any other method based on a single estimator, are at best an approximation to the ideal Bayesian inference process that would rely on the calculation of $\mathbf{P}(\mathbf{E}(f)|D)$ given any sample D.

4.5.1 Markov Chains

The theory of Markov chains is well established [176]. Here we review only the most basic concepts and refer the reader to the textbook literature for more information. As in statistical mechanics, consider a system $S = \{s_1, s_2, \ldots, s_{|S|}\}$ with $|S|$ states. Let $S^0, S^1, \ldots, S^t, \ldots$ be the sequence of variables representing the state of the system at each time. Thus each integer from 1 to $|S|$ is associated with one state of the chain, and at any time the chain is in one particular state. The variables S^t form a *Markov chain* if and only if for any t

$$\mathbf{P}(S^{t+1}|S^0, .., S^t) = \mathbf{P}(S^{t+1}|S^t). \tag{4.10}$$

Intuitively, this can be rephrased by saying that the future depends on the past only through the present. S^t is called the state of the chain at time t. A Markov chain is entirely defined by the initial distribution $P(S^0)$ and the *transition probabilities* $P^t = P(S^{t+1}|S^t)$. Here we will be concerned only with *stationary* Markov chains, where the transition probabilities are constant, that is, independent of time. The transition matrix of the chain is then the matrix $T = (t_{ij})$, where t_{ij} is the probability of moving from state s_j to state s_i. Note that, in relation to (4.9), the state space of the chain is defined by the coordinates $x_1, \ldots, x_n$; that is, each S^t is an n-dimensional variable.

A distribution over the state space of the chain is said to be *stable* if, once reached, it persists forever. Thus a stable distribution Q must satisfy the balance equation

$$Q(s_i) = \sum_{k=1}^{|S|} t_{ik} Q(s_k) = (1 - \sum_{j \neq i} t_{ji}) Q(s_i) + \sum_{j \neq i} t_{ij} Q(s_j) \tag{4.11}$$

or equivalently

$$-\sum_{j \neq i} t_{ji} Q(s_i) + \sum_{j \neq i} t_{ij} Q(s_j) = 0. \tag{4.12}$$

Thus, a *sufficient* condition for stability is the *pairwise* balance equation

$$t_{ji} Q(s_i) = t_{ij} Q(s_j) \tag{4.13}$$

for every i and j. This expresses the fact that the average number of transitions from s_i to s_j is equal to the average number of transitions from s_j to s_i, and therefore the overall distribution over states is preserved.

A Markov chain can in general have several stable distributions. Markov chains with finite state space always have at least one stable distribution. Obviously, in MCMC sampling procedures, we will be interested in stable distributions, in fact in the even stronger conditions of *ergodic* distributions. Here, a distribution is defined to be ergodic if and only if the chain always converges to it, regardless of the choice of the initial distribution at time 0. In the case of an ergodic Markov chain, there is only one stable distribution, called the *equilibrium* distribution. Conditions for the ergodicity of a Markov chain, and bounds on the rate of convergence to the equilibrium distribution, are well known [150, 180].

In order to achieve our goal of sampling from $P(x_1, \ldots, x_n)$, we now turn to the two main MCMC algorithms: Gibbs sampling and the Metropolis algorithm.

4.5.2 Gibbs Sampling

Gibbs sampling, also known as the heatbath method, is the simplest MCMC algorithm [199]. It can be applied to a wide range of situations, especially when the conditional distributions $P(x_i|x_j : j \neq i)$ can be computed easily, or when the variables X_i take on values from a small set. In Gibbs sampling, one iteratively samples each single variable, conditioned on the most recent value of all the other variables. Starting from $(x_1^t, \ldots, x_n^t)$,

1. Select x_1^{t+1} according to $P(X_1|x_2^t, x_3^t, \ldots, x_n^t)$.
2. Select x_2^{t+1} according to $P(X_2|x_1^{t+1}, x_2^t, \ldots, x_n^t)$.
3. ....
n. Select x_n^{t+1} according to $P(X_n|x_1^{t+1}, x_2^{t+1}, \ldots, x_{n-1}^{t+1})$.

In this version, we cycle through the variables sequentially. It is also possible to cycle through the variables in any order, or to uniformly select the variables at each step. One can even use any other fixed distribution, as long as each variable has a nonzero probability of being visited. It is also possible to sample variables by groups rather than one by one. By applying the definition, it is trivial to check that the Gibbs sampling algorithm leads to a stable distribution. Proofs of ergodicity and further information can be found in the general references on MCMC methods given above and in [209, 191, 490]. An example of specific Gibbs sampling equations for Bayesian networks is given in appendix C. We now turn to another MCMC method, the Metropolis algorithm, of which Gibbs sampling is a special case.

4.5.3 Metropolis Algorithm

Again let us suppose that the goal is to sample from a given distribution $P(s) = P(x_1, \ldots, x_n)$. The Metropolis algorithm [388] randomly generates perturbations of the current state, and accepts or rejects them depending on how the probability of the state is affected.

More precisely, the Metropolis algorithm is defined using two auxiliary families of distributions Q and R. $Q = (q_{ij})$ is the selection distribution; q_{ij} is the probability of selecting state s_i while being in state s_j. $R = (r_{ij})$ is the acceptance distribution; r_{ij} is the probability of accepting state s_i while being in state s_j and having selected s_i as a possible next state. Obviously, we must have $q_{ij} \geq 0$ and $r_{ij} \geq 0$, and $\sum_i q_{ij} = 1$. For the time being, and in most practical cases, one can assume that Q is symmetric, $q_{ij} = q_{ji}$, but this hypothesis can also be relaxed. Starting from a state s_j at time t ($S^t = s_j$), the algorithm proceeds as follows:

1. Randomly select a state s_i according to the distribution q_{ij}.
2. Accept s_i with probability r_{ij}. That is, $S^{t+1} = s_i$ with probability r_{ij} and $S^{t+1} = s_j$ with probability $1 - r_{ij}$.

In the most common version of the Metropolis algorithm, the acceptance distribution is defined by

$$r_{ij} = \min\left(1, \frac{P(s_i)}{P(s_j)}\right). \tag{4.14}$$

We leave it as an exercise to show that Gibbs sampling can be rewritten as a Metropolis algorithm. When P is expressed in terms of an energy function $P(s) = e^{-\mathcal{E}(s)/kT}/Z$, this can be rewritten as

$$r_{ij} = \min(1, e^{-[\mathcal{E}(s_i)-\mathcal{E}(s_j)]/kT}) = \min(1, e^{-\Delta_{ij}\mathcal{E}/kT}). \tag{4.15}$$

Note that only the ratio of the probabilities is needed, not the partition function. As a result, the algorithm can be expressed in its most familiar form:

1. Randomly select a state s_i according to the distribution q_{ij}.
2. If $\mathcal{E}(s_i) \leq \mathcal{E}(s_j)$ accept s_i. If $\mathcal{E}(s_i) > \mathcal{E}(s_j)$, accept s_i only with probability $e^{-\Delta_{ij}\mathcal{E}/kT}$. If s_i is rejected, stay in s_j.

It is easy to see that the distribution P is stable under the Metropolis algorithm. We have $t_{ij} = q_{ij}P(s_i)/P(s_j)$ and $t_{ji} = q_{ji}$. Since Q is symmetric, this immediately gives

$$P(s_j)t_{ij} = P(s_i)t_{ji}. \tag{4.16}$$

In other words, since the pairwise balance equations are satisfied, P is stable.

To ensure ergodicity, it is necessary and sufficient to ensure that there are no absorbing states in the chain, or equivalently that there is always a path of transitions with *nonzero probability* from any s_i to any s_j. This of course depends on the structure of q_{ij}. Several general remarks can be made. We can construct a graph G by connecting two points i and j with an edge if and only if $q_{ij} > 0$. If the resulting graph is complete (or even just very dense), the chain is clearly ergodic. This type of Metropolis algorithm can be termed "global" because there is a nonzero probability of moving from any state i to any state j in one step, or at most very few steps, if the graph is dense but not complete. When the graph is more sparse, one obtains more "local" versions of the Metropolis algorithm. Ergodicity is still preserved, provided any two points are connected by at least one path. An example of this situation is when the algorithm is applied componentwise, perturbing one component at a time. In most practical applications, the selection probability q_{ji} is chosen uniformly over the neighbors j of vertex i. Usually, q_{ii} is also chosen to be 0, although this does not really impact any of the results just described.

Finally, there are several variations and generalizations of the Metropolis algorithm using, for instance, the derivatives of the energy function, other acceptance functions [242, 396], and cluster Monte Carlo algorithms [510, 547]. In particular, it is even possible to remove the condition that Q be symmetric, as long as the balance is preserved by modifying the acceptance function R accordingly:

$$r_{ij} = \min\left(1, \frac{P(s_i)q_{ij}}{P(s_j)q_{ji}}\right). \tag{4.17}$$

4.6 Simulated Annealing

Simulated annealing [321] (see also [67] for a review) is a general-purpose optimization algorithm inspired by statistical mechanics. It combines MCMC ideas such as the Metropolis algorithm with a schedule for lowering the temperature. The name has its origin in metallurgy, where metals that have been annealed (cooled slowly) exhibit strength properties superior to metals that have been quenched (cooled rapidly). The greater macroscopic strength is associated with internal molecular states of lower energy.

Consider the problem of minimizing a function $f(x_1, \ldots, x_n)$. Without any loss of generality, we can assume that $f \geq 0$ everywhere. As usual, we can regard f as representing the energy of a statistical mechanical system with states $s = (x_1, \ldots, x_n)$. We have seen that the probability of being in state s at temperature T is given by the Boltzmann–Gibbs distribution $P(s) = P(x_1, \ldots, x_n) = e^{-f(s)/kT}/Z$. The first key observation in order to understand simulated annealing is that at low temperatures, the Boltzmann–

Gibbs distribution is dominated by the states of lowest energy, which become the most probable. In fact, if there are m states where the minimum of the function f is achieved, we have

$$\lim_{T\to 0} P(s) = \begin{cases} 1/m & \text{if } s \text{ is a ground state} \\ 0 & \text{otherwise.} \end{cases} \tag{4.18}$$

If we could simulate the system at temperatures near 0, we would immediately have the ground states, that is, the minima of f. The catch is that any MCMC method fails in general to reach the Boltzmann–Gibbs equilibrium distribution in a reasonable time, because movement in state space is inhibited by regions of very low probability, that is, by high energy barriers. Simulated annealing attempts to overcome this problem by starting with a high temperature, where the Boltzmann–Gibbs distribution is close to uniform, and progressively lowering it according to some annealing schedule. While simulated annealing is usually used in combination with the Metropolis algorithm, it is in fact applicable to any MCMC method, and in particular to Gibbs sampling.

The annealing schedule of course plays a crucial role. There are a number of theoretical results [199] showing that for a logarithmic annealing schedule of the form

$$T^t = \frac{K}{\log t} \tag{4.19}$$

($t \geq 1$), the algorithm converges almost surely to one of the ground states, for some value of the constant K (see [230] for a lower bound on K). (From the context, no confusion should arise between T the temperature and T the time horizon.) Intuitively, this is easy to see [396]. If we let $s_{\max}$ and $s_{\min}$ denote two states with maximal and minimal energy, then from the Boltzmann–Gibbs distribution we have,

$$\frac{P^t(s_{\max})}{P^t(s_{\min})} = \left(\frac{1}{t}\right)^{\Delta\mathcal{E}/kK}, \tag{4.20}$$

where $\Delta\mathcal{E} = \mathcal{E}(s_{\max}) - \mathcal{E}(s_{\min})$. If we take $K = \Delta\mathcal{E}/k$, we then have $P^t(s_{\max}) = P^t(s_{\min})/t$. Therefore, for any state s,

$$P^t(s) \geq P^t(s_{\max}) = \frac{1}{t}P^t(s_{\min}) \geq \frac{1}{t}P^1(s_{\min}). \tag{4.21}$$

In particular, the number of times any state s is visited during the annealing is lower-bounded by $P^1(s_{\min}) \sum_t 1/t$, which is divergent. Thus, with K scaled with respect to the highest energy barrier, it is impossible for the algorithm to remain trapped in a bad local minimum.

It must be noted, however, that a logarithmic annealing schedule is very slow and generally impractical. A logarithmic schedule suggests that a significant fraction of all possible states is visited, and therefore is essentially equivalent to an exhaustive search. Thus it is not surprising that it is guaranteed

to find the global optimum. On the other hand, if an exhaustive search had been an alternative, it would have been used in the first place. Most problems of interest are typically NP complete, with an exponential number of possible states ruling out any possibility of conducting exhaustive searches. In practice, simulated annealing must be used with faster schedules, such as geometric annealing schedules of the form

$$T^t = \mu T^{t-1} \tag{4.22}$$

for some $0 < \mu < 1$. Naturally, the best one can then hope for is to converge in general to approximate solutions corresponding to points of low energy, but not to the global minima.

Other interesting algorithms related to simulated annealing [547, 381] and MCMC basic ideas, such as dynamical and hybrid Monte Carlo methods [152, 396], are discussed in the references.

4.7 Evolutionary and Genetic Algorithms

In the present context, evolutionary algorithms [261, 476] perhaps have a special flavor since their source of inspiration, evolution, is at the heart of our domain. Evolutionary algorithms are a broad class of optimization algorithms that attempt to simulate in some way the inner workings of evolution, as we (think we) understand it. One component common to all these algorithms is the generation of random perturbations, or mutations, and the presence of a fitness function that is used to assess the quality of a given point and filter out mutations that are not useful. In this sense, random descent methods and even simulated annealing can be viewed as special cases of evolutionary algorithms. One of the broadest subclasses of evolutionary algorithms is the genetic algorithms.

Genetic algorithms [328, 330] and the related field of artificial life push the evolutionary analogy one step further by simulating the evolution of populations of points in fitness space. Furthermore, in addition to mutations, new points are generated by a number of other operations mimicking genetic operators and sexual reproduction, such as crossover. While genetic algorithms are particularly flexible and make possible the evolution of complex objects, such as computer programs, they remain quite slow even on current computers, although this is of course subject to yearly improvements. Applications of genetic algorithms to problems in molecular biology can be found in [329, 233, 415]. Other evolutionary algorithms are described in [53] and references therein. Evolutionary algorithms will not be considered any further in this book.

4.8 Learning Algorithms: Miscellaneous Aspects

In connection with learning algorithms, there is a wide range of implementation details, heuristics, and tricks that have significant practical importance. Abundant material on such tricks can be found, for instance, in the annual proceedings of NIPS (Neural Information Processing Conference). Here we cover only a small subset of them from a general standpoint. A few model-specific tricks are presented in the relevant chapters.

4.8.1 Control of Model Complexity

In one form or another, modelers are constantly confronted with the problem of striking a balance between underfitting and overfitting the data, between models that have too few and too many degrees of freedom. One approach to this problem is to regularize the objective likelihood function with a term that takes model complexity into account. The most principled versions of this approach are based on equalities or bounds relating the training error $\mathcal{E}_T$ to the generalization error $\mathcal{E}_G$. These bounds typically state that with high probability $\mathcal{E}_G \leq \mathcal{E}_T + C$, where C is a term reflecting the complexity of the model. Examples of such a formula can be found in [533], using the concept of VC dimension, and in [5, 16], using statistical asymptotic theory. The generalization error is then minimized by minimizing the regularized training error $\mathcal{E}_T + C$. The term $\mathcal{E}_T$ measures the data fit and the term C can often be viewed as a prior favoring simpler models. Such practices can yield good results and have heuristic value. But, as pointed out in chapter 2, from a Bayesian point of view they also have some weaknesses. With complex data, a prior expecting the data to be generated by a simple model does not make much sense. In general, we would recommend instead using powerful flexible models, with many degrees of freedom and strong priors on their parameters and structure, rather than their overall complexity, to control overfitting.

4.8.2 Online/Batch Learning

Training is said to be *online* when some degree of model fitting or parameter adjustment occurs as the data come in, or after the presentation of each example. In *batch* or *offline* learning, on the other hand, parameter values are adjusted only after the presentation of a large number of examples, if not the entire training set. Obviously there is a spectrum of possibilities in between. Online learning can have some advantages in that it does not require holding many training examples in memory, and it is more flexible and easier to implement. It is also closer to the Bayesian spirit of updating one's belief as

data become available, and to the way biological systems seem to learn. More important, perhaps, learning after the presentation of each example may introduce a degree of stochasticity that may be useful to explore the space of solutions and avoid certain local minima. It can also be shown, of course, that with sufficiently small learning rates, online learning approximates batch learning (see also [49]). Accordingly, in this book we usually provide online learning equations.

4.8.3 Training/Test/Validation

One of the most widely used practices consists in using only a subset of the data for model fitting and the remaining data, or portions of it, for the validation of the model. It is important to note that such a practice is not entirely Bayesian, since in the general framework of chapter 2 all the data are used for model fitting, without any reference or need for validation. In practice, cross-validation techniques remain very useful because they are generally easy to implement and yield good results, especially when data are abundant. A second remark, of course, is that there are many ways of splitting the data into different subsets and allocating such subsets to training or validation experiments. For instance, different data sets can be used to train different experts that are subsequently combined, or validation sets can be used to determine the values of hyperparameters. Such matters become even more important when data are relatively scarce. Whenever feasible, it is good to have at least three distinct data sets: one for training, one for validation and training adjustments, and one for testing overall performance.

Special additional care is often required in bioinformatics because sequences have a high probability of being related through a common ancestor. In chapter 1 the problem of constructing low-similarity test sets, which may be essential to assess reliably the predictive performance of a method obtained by machine learning, was addressed in detail.

4.8.4 Early Stopping

When a model is too flexible with respect to the available data—because it contains too many parameters—overfitting is observed during training. This means that while the error on the training set decreases monotonically as a function of training epochs, the error on a validation set also decreases at first, then reaches a minimum and begins to increase again. Overfitting is then associated with the model's memorizing the training data or fitting noise in the data to a point that is deleterious for generalization. The correct approach in such a situation of course would be to modify the model. Another widely

used but less sound alternative is early stopping, whereby training is stopped as soon as the error rate on the training set reaches a certain threshold, or after a fixed number of training cycles. The threshold itself, or the number of cycles, is not easy to determine. One possibility is to stop training as soon as the error rate begins to increase on a validation set different from the training set. The drawback of such an approach is that data must be sacrificed from the training set for validation. Furthermore, this type of early stopping can still lead to a partial overfitting of the validation data with respect to the test data. In other words, the performance of the model on the validation set used to decide when to stop is typically somewhat better than the overall generalization performance on new data. Early stopping, like other validation methods, is, however, easy to implement and useful in practice, especially with abundant data.

4.8.5 Ensembles

When a complex model is fitted to the data by ML or MAP optimization, different model parameters are derived by varying a number of factors during the learning procedure, such as the initial parameter values, the learning rate, the order of presentation of the examples, the training set, and so on. Furthermore, different classes of models may be tried. It is natural to suspect that better prediction or classification may be achieved by averaging the opinion of different models or experts in some way (appendix A and [223, 237, 277, 568, 426, 340, 339]). A pool of models for a given task is also called an ensemble, in analogy to statistical mechanics (see also the notion of the committee machine in the literature). Mathematically, this intuition is based on the fact that for convex error functions, the error of the ensemble is less than the average error of its members (Jensen's inequality in appendix B). Thus the ensemble performs better than a typical single expert. There are different ways of combining the predictions produced by several models. Uniform averages are widely used, but other schemes are possible, with variable weights, including the possibility of learning the weights during training. Note that in the case of a well-defined class of models within the Bayesian framework of chapter 2, the optimal prediction is obtained by integrating over all possible models (see (2.18)). Thus averaging models can be construed as an approximation to such an integral.

4.8.6 Balancing and Weighting Schemes

An important issue to consider is whether or not training sets are balanced. In binomial classification problems, the number of available positive exam-

ples can differ significantly from the number of negative examples. Likewise, in multinomial classification problems, significant variations can exist in the proportions in which each class is represented in the data. This situation can be particularly severe with biological databases where, for instance, certain organisms or certain types of sequences are overrepresented due to a large number of different factors, as described in chapter 1.

Ideally, for the purpose of correct classification, all relevant classes should be equally represented in the training set. In chapter 6 such balanced training strategies will be described. In some cases, underrepresentation of a certain class in the training data has led to a low test prediction performance on that particular class. Such behavior has often been interpreted as evidence for missing information, for example that beta-sheet prediction requires more long-range sequence information than does helix prediction. While any protein structure prediction method will gain from the proper addition of long-range information, beta-sheet performance has been substantially improved just by applying a balanced training scheme [452].

Another possibility is to use weighting schemes to artificially balance training sets, equivalent to effectively duplicating rare exemplars several times over. A number of weighting schemes have been developed for DNA and protein sequences, especially in the context of multiple alignments [10, 536, 487, 201, 249, 337]. The weighting scheme in [337] is particularly interesting, and optimal in a maximum entropy sense.

There is a number of other techniques that we do not cover for lack of space. Again these can easily be found in the literature (NIPS Proceedings) and other standard references on neural network techniques. They include:

- Active sampling.
- Pruning methods. These are methods that perform simplification of models during or after learning. Typically, they consist of finding ways to determine which parameters in a model have little impact on its performance, and then removing them. Redundant parameters will often be equivalent not just to those with small numerical values, but also large weights that inhibit each other may contribute little to the quality of a model.
- Second-order methods. These methods take advantage of second-order information by computing or approximating the Hessian of the likelihood—for instance, to adjust learning rates or compute error bars. The efficient approximation of the Hessian is an interesting problem that must be considered in the context of each model.

This page intentionally left blank