41

Learning as Inference

▶ 41.1 Neural network learning as inference

In Chapter 39 we trained a simple neural network as a classifier by minimizing an objective function

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w}) \tag{41.1}$$

made up of an error function

$$G(\mathbf{w}) = -\sum_n \left[t^{(n)} \ln y(\mathbf{x}^{(n)}; \mathbf{w}) + (1 - t^{(n)}) \ln(1 - y(\mathbf{x}^{(n)}; \mathbf{w})) \right] \tag{41.2}$$

and a regularizer

$$E_W(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2. \tag{41.3}$$

This neural network learning process can be given the following probabilistic interpretation.

We interpret the output $y(\mathbf{x}; \mathbf{w})$ of the neuron literally as defining (when its parameters $\mathbf{w}$ are specified) the probability that an input $\mathbf{x}$ belongs to class $t = 1$, rather than the alternative $t = 0$. Thus $y(\mathbf{x}; \mathbf{w}) \equiv P(t = 1 \,|\, \mathbf{x}, \mathbf{w})$. Then each value of $\mathbf{w}$ defines a different hypothesis about the probability of class 1 relative to class 0 as a function of $\mathbf{x}$.

We define the observed data D to be the *targets* $\{t\}$ – the inputs $\{\mathbf{x}\}$ are assumed to be given, and not to be modelled. To infer $\mathbf{w}$ given the data, we require a likelihood function and a prior probability over $\mathbf{w}$. The likelihood function measures how well the parameters $\mathbf{w}$ predict the observed data; it is the probability assigned to the observed t values by the model with parameters set to $\mathbf{w}$. Now the two equations

$$\begin{array}{rcl} P(t = 1 \,|\, \mathbf{w}, \mathbf{x}) & = & y \\ P(t = 0 \,|\, \mathbf{w}, \mathbf{x}) & = & 1 - y \end{array} \tag{41.4}$$

can be rewritten as the single equation

$$P(t \,|\, \mathbf{w}, \mathbf{x}) = y^t (1 - y)^{1-t} = \exp[t \ln y + (1 - t) \ln(1 - y)]\,. \tag{41.5}$$

So the error function G can be interpreted as minus the log likelihood:

$$P(D \,|\, \mathbf{w}) = \exp[-G(\mathbf{w})]. \tag{41.6}$$

Similarly the regularizer can be interpreted in terms of a log prior probability distribution over the parameters:

$$P(\mathbf{w} \,|\, \alpha) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \tag{41.7}$$

If E_W is quadratic as defined above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = 1/\alpha$, and $1/Z_W(\alpha)$ is equal to $(\alpha/2\pi)^{K/2}$, where K is the number of parameters in the vector $\mathbf{w}$.

The objective function $M(\mathbf{w})$ then corresponds to the *inference* of the parameters $\mathbf{w}$, given the data:

$$P(\mathbf{w} \mid D, \alpha) = \frac{P(D \mid \mathbf{w})P(\mathbf{w} \mid \alpha)}{P(D \mid \alpha)} \tag{41.8}$$

$$= \frac{e^{-G(\mathbf{w})}\, e^{-\alpha E_W(\mathbf{w})}/Z_W(\alpha)}{P(D \mid \alpha)} \tag{41.9}$$

$$= \frac{1}{Z_M} \exp(-M(\mathbf{w})). \tag{41.10}$$

So the $\mathbf{w}$ *found by (locally) minimizing* $M(\mathbf{w})$ *can be interpreted as the (locally) most probable parameter vector,* $\mathbf{w}^*$. From now on we will refer to $\mathbf{w}^*$ as $\mathbf{w}_{\rm MP}$.

Why is it natural to interpret the error functions as *log* probabilities? Error functions are usually additive. For example, G is a *sum* of information contents, and E_W is a *sum* of squared weights. Probabilities, on the other hand, are multiplicative: for independent events X and Y, the joint probability is $P(x, y) = P(x)P(y)$. The logarithmic mapping maintains this correspondence.

The interpretation of $M(\mathbf{w})$ as a log probability has numerous benefits, some of which we will discuss in a moment.

▶ 41.2 Illustration for a neuron with two weights

In the case of a neuron with just two inputs and no bias,

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}, \tag{41.11}$$

we can plot the posterior probability of $\mathbf{w}$, $P(\mathbf{w} \mid D, \alpha) \propto \exp(-M(\mathbf{w}))$. Imagine that we receive some data as shown in the left column of figure 41.1. Each data point consists of a two-dimensional input vector $\mathbf{x}$ and a t value indicated by $\times$ $(t = 1)$ or $\Box$ $(t = 0)$. The likelihood function $\exp(-G(\mathbf{w}))$ is shown as a function of $\mathbf{w}$ in the second column. It is a product of functions of the form (41.11).

The product of traditional learning is a point in $\mathbf{w}$-space, the estimator $\mathbf{w}^*$, which maximizes the posterior probability density. In contrast, in the Bayesian view, the product of learning is an *ensemble* of plausible parameter values (bottom right of figure 41.1). We do not choose one particular hypothesis $\mathbf{w}$; rather we evaluate their posterior probabilities. The posterior distribution is obtained by multiplying the likelihood by a prior distribution over $\mathbf{w}$ space (shown as a broad Gaussian at the upper right of figure 41.1). The posterior ensemble (within a multiplicative constant) is shown in the third column of figure 41.1, and as a contour plot in the fourth column. As the amount of data increases (from top to bottom), the posterior ensemble becomes increasingly concentrated around the most probable value $\mathbf{w}^*$.

▶ 41.3 Beyond optimization: making predictions

Let us consider the task of making predictions with the neuron which we trained as a classifier in section 39.3. This was a neuron with two inputs and a bias.

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}. \tag{41.12}$$

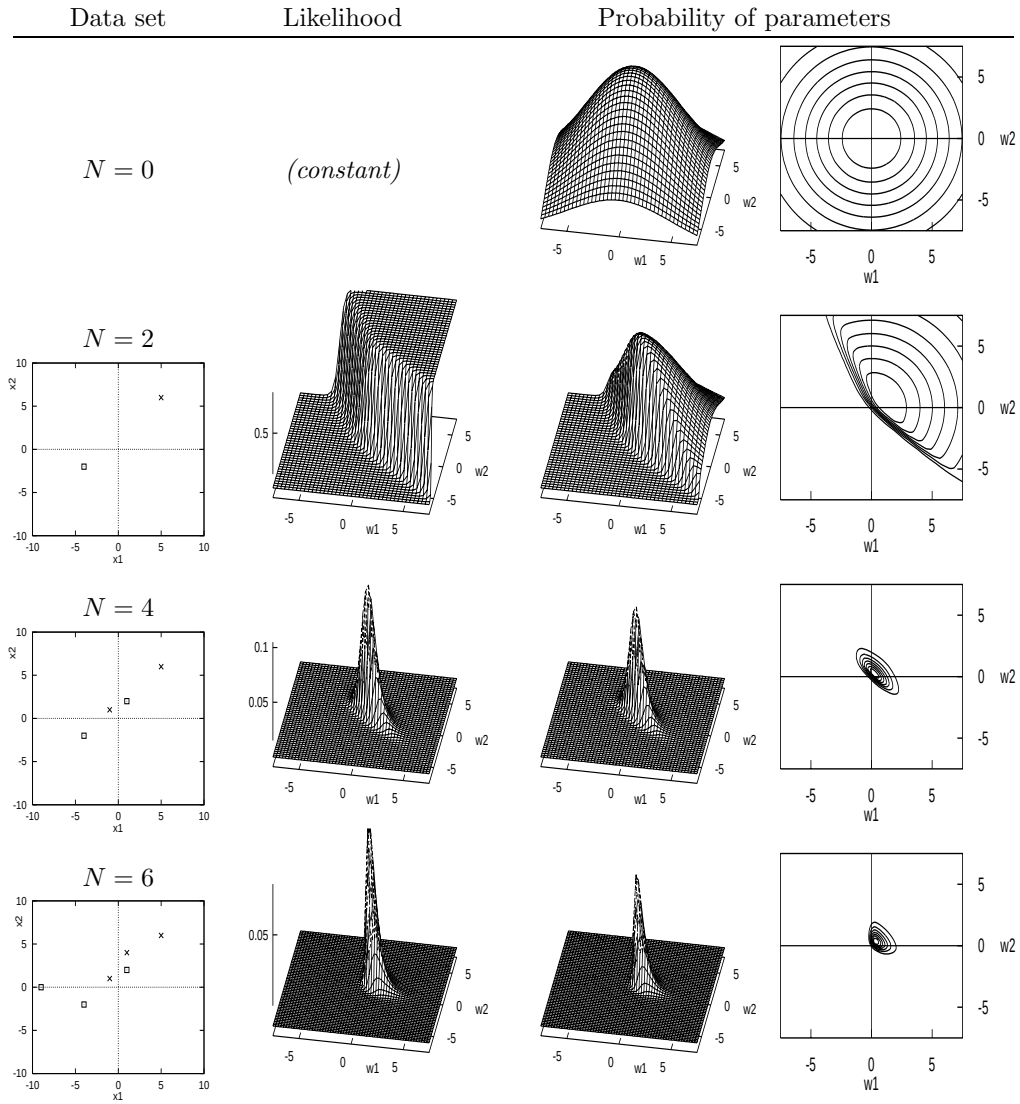

Figure 41.1. The Bayesian interpretation and generalization of traditional neural network learning. Evolution of the probability distribution over parameters as data arrive.

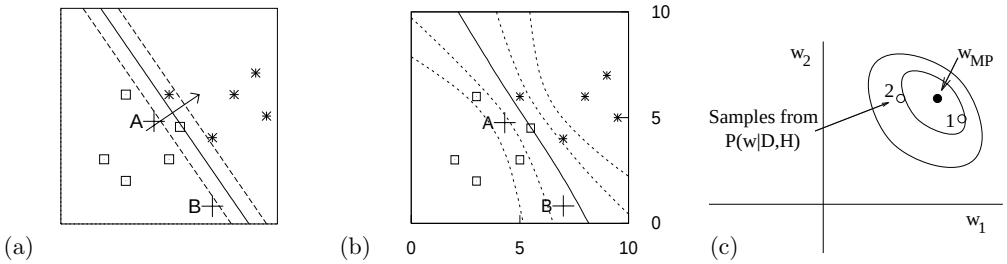

Figure 41.2. Making predictions. (a) The function performed by an optimized neuron $\mathbf{w}_{\rm MP}$ (shown by three of its contours) trained with weight decay, $\alpha = 0.01$ (from figure 39.6). The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. (b) Are these predictions more reasonable? (Contours shown are for $y = 0.5, 0.27$, 0.73, 0.12 and 0.88.) (c) The posterior probability of $\mathbf{w}$ (schematic); the Bayesian predictions shown in (b) were obtained by averaging together the predictions made by each possible value of the weights $\mathbf{w}$, with each value of $\mathbf{w}$ receiving a vote proportional to its probability under the posterior ensemble. The method used to create (b) is described in section 41.4.

When we last played with it, we trained it by minimizing the objective function

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w}). \tag{41.13}$$

The resulting optimized function for the case $\alpha = 0.01$ is reproduced in figure 41.2a.

We now consider the task of predicting the class $t^{(N+1)}$ corresponding to a new input $\mathbf{x}^{(N+1)}$. It is common practice, when making predictions, simply to use a neural network with its weights fixed to their optimized value $\mathbf{w}_{\rm MP}$, but this is not optimal, as can be seen intuitively by considering the predictions shown in figure 41.2a. Are these reasonable predictions? Consider new data arriving at points A and B. The best-fit model assigns both of these examples probability 0.2 of being in class 1, because they have the same value of $\mathbf{w}_{\rm MP} \cdot \mathbf{x}$. If we really knew that $\mathbf{w}$ was equal to $\mathbf{w}_{\rm MP}$, then these predictions would be correct. But we do not know $\mathbf{w}$. The parameters are *uncertain*. Intuitively we might be inclined to assign a less confident probability (closer to 0.5) at B than at A, as shown in figure 41.2b, since point B is far from the training data. *The best-fit parameters* $\mathbf{w}_{\rm MP}$ *often give over-confident predictions.* A non-Bayesian approach to this problem is to downweight all predictions uniformly, by an empirically determined factor (Copas, 1983). This is not ideal, since intuition suggests the strength of the predictions at B should be downweighted more than those at A. A Bayesian viewpoint helps us to understand the cause of the problem, and provides a straightforward solution. In a nutshell, we obtain Bayesian predictions by taking into account the whole posterior ensemble, shown schematically in figure 41.2c.

The Bayesian prediction of a new datum $\mathbf{t}^{(N+1)}$ involves *marginalizing* over the parameters (and over anything else about which we are uncertain). For simplicity, let us assume that the weights $\mathbf{w}$ are the only uncertain quantities – the weight decay rate α and the model $\mathcal{H}$ itself are assumed to be fixed. Then by the sum rule, the predictive probability of a new target $\mathbf{t}^{(N+1)}$ at a location $\mathbf{x}^{(N+1)}$ is:

$$P(\mathbf{t}^{(N+1)} \,|\, \mathbf{x}^{(N+1)}, D, \alpha) = \int \mathrm{d}^K \mathbf{w}\, P(\mathbf{t}^{(N+1)} \,|\, \mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) P(\mathbf{w} \,|\, D, \alpha), \tag{41.14}$$

where K is the dimensionality of $\mathbf{w}$, three in the toy problem. Thus the predictions are obtained by weighting the prediction for each possible $\mathbf{w}$,

$$\begin{array}{rcl} P(\mathbf{t}^{(N+1)} = 1 \,|\, \mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) & = & y(\mathbf{x}^{(N+1)}; \mathbf{w}) \\ P(\mathbf{t}^{(N+1)} = 0 \,|\, \mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) & = & 1 - y(\mathbf{x}^{(N+1)}; \mathbf{w}), \end{array} \tag{41.15}$$

with a weight given by the posterior probability of $\mathbf{w}$, $P(\mathbf{w} \,|\, D, \alpha)$, which we most recently wrote down in equation (41.10). This posterior probability is

$$P(\mathbf{w} \,|\, D, \alpha) = \frac{1}{Z_M} \exp(-M(\mathbf{w})), \tag{41.16}$$

where

$$Z_M = \int \mathrm{d}^K \mathbf{w} \exp(-M(\mathbf{w})). \tag{41.17}$$

In summary, we can get the Bayesian predictions if we can find a way of computing the integral

$$P(\mathbf{t}^{(N+1)} = 1 \,|\, \mathbf{x}^{(N+1)}, D, \alpha) = \int \mathrm{d}^K \mathbf{w}\ \ y(\mathbf{x}^{(N+1)}; \mathbf{w}) \frac{1}{Z_M} \exp(-M(\mathbf{w})), \tag{41.18}$$

which is the average of the output of the neuron at $\mathbf{x}^{(N+1)}$ under the posterior distribution of $\mathbf{w}$.

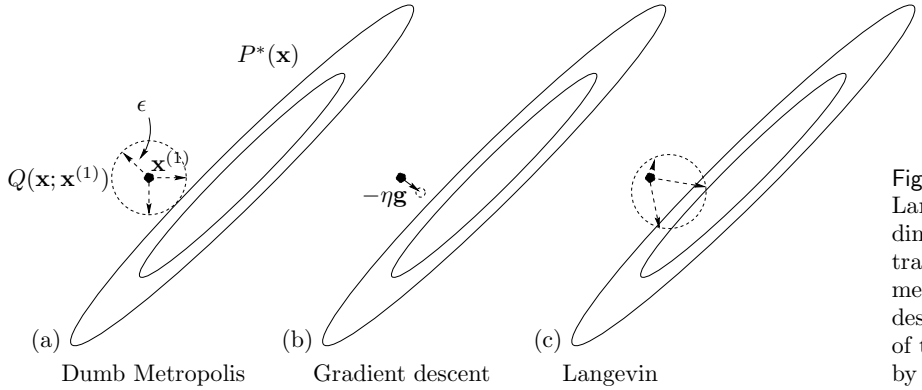

Figure 41.3. One step of the Langevin method in two dimensions (c), contrasted with a traditional 'dumb' Metropolis method (a) and with gradient descent (b). The proposal density of the Langevin method is given by 'gradient descent with noise'.

Implementation

How shall we compute the integral (41.18)? For our toy problem, the weight space is three dimensional; for a realistic neural network the dimensionality K might be in the thousands.

Bayesian inference for general data modelling problems may be implemented by exact methods (Chapter 25), by Monte Carlo sampling (Chapter 29), or by deterministic approximate methods, for example, methods that make Gaussian approximations to $P(\mathbf{w} \mid D, \alpha)$ using Laplace's method (Chapter 27) or variational methods (Chapter 33). For neural networks there are few exact methods. The two main approaches to implementing Bayesian inference for neural networks are the Monte Carlo methods developed by Neal (1996) and the Gaussian approximation methods developed by MacKay (1991).

▶ 41.4 Monte Carlo implementation of a single neuron

First we will use a Monte Carlo approach in which the task of evaluating the integral (41.18) is solved by treating $y(\mathbf{x}^{(N+1)}; \mathbf{w})$ as a function f of $\mathbf{w}$ whose mean we compute using

$$\langle f(\mathbf{w}) \rangle \simeq \frac{1}{R} \sum_r f(\mathbf{w}^{(r)}) \tag{41.19}$$

where $\{\mathbf{w}^{(r)}\}$ are samples from the posterior distribution $\frac{1}{Z_M}\exp(-M(\mathbf{w}))$ (cf. equation (29.6)). We obtain the samples using a Metropolis method (section 29.4). As an aside, a possible disadvantage of this Monte Carlo approach is that it is a poor way of estimating the probability of an improbable event, i.e., a $P(t \mid D, \mathcal{H})$ that is very close to zero, if the improbable event is most likely to occur in conjunction with improbable parameter values.

How to generate the samples $\{\mathbf{w}^{(r)}\}$? Radford Neal introduced the *Hamiltonian Monte Carlo* method to neural networks. We met this sophisticated Metropolis method, which makes use of gradient information, in Chapter 30. The method we now demonstrate is a simple version of Hamiltonian Monte Carlo called the *Langevin Monte Carlo method*.

The Langevin Monte Carlo method

The Langevin method (algorithm 41.4) may be summarized as 'gradient descent with added noise', as shown pictorially in figure 41.3. A noise vector $\mathbf{p}$ is generated from a Gaussian with unit variance. The gradient $\mathbf{g}$ is computed,

```
g = gradM ( w ) ;                          # set gradient using initial w
M = findM ( w ) ;                          # set objective function too

for l = 1:L                                # loop L times
   p = randn ( size(w) ) ;                 # initial momentum is Normal(0,1)
   H = p' * p / 2 + M ;                    # evaluate H(w,p)

*  p = p - epsilon * g / 2 ;               # make half-step in p
*  wnew = w + epsilon * p ;                # make step in w
*  gnew = gradM ( wnew ) ;                 # find new gradient
*  p = p - epsilon * gnew / 2 ;            # make half-step in p

   Mnew = findM ( wnew ) ;                 # find new objective function
   Hnew = p' * p / 2 + Mnew ;              # evaluate new value of H
   dH = Hnew - H ;                         # decide whether to accept
   if ( dH < 0 )                 accept = 1 ;
   elseif ( rand() < exp(-dH) ) accept = 1 ;  # compare with a uniform
   else                          accept = 0 ;  #                variate
   endif
   if ( accept )      g = gnew ;   w = wnew ;    M = Mnew ;    endif
endfor

function gM = gradM ( w )                  # gradient of objective function
   a = x * w  ;                            # compute activations
   y = sigmoid(a) ;                        # compute outputs
   e = t - y   ;                           # compute errors
   g = - x' * e ;                          # compute the gradient of G(w)
   gM = alpha * w + g ;
endfunction

function M = findM ( w )                   # objective function
   G = - (t' * log(y) + (1-t') * log( 1-y )) ;
   EW = w' * w / 2 ;
   M = G + alpha * EW ;
endfunction
```

Algorithm 41.4. `Octave` source code for the Langevin Monte Carlo method. To obtain the Hamiltonian Monte Carlo method, we repeat the four lines marked * multiple times (algorithm 41.8).

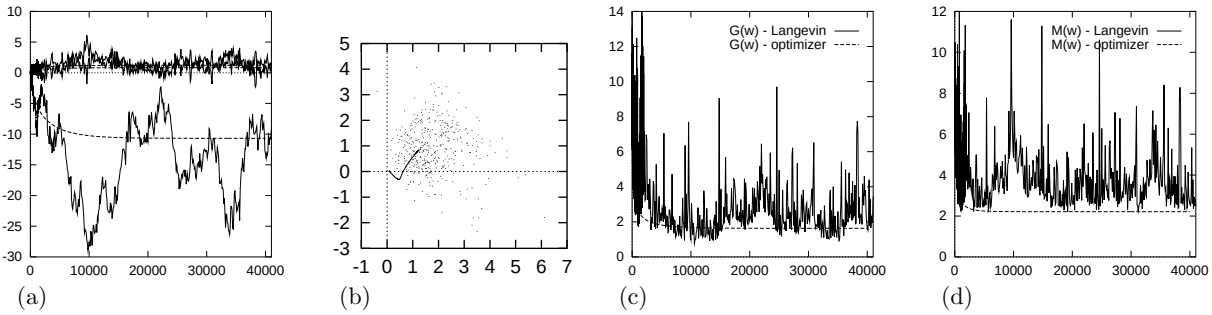

Figure 41.5. A single neuron learning under the Langevin Monte Carlo method. (a) Evolution of weights w_0, w_1 and w_2 as a function of number of iterations. (b) Evolution of weights w_1 and w_2 in weight space. Also shown by a line is the evolution of the weights using the optimizer of figure 39.6. (c) The error function $G(\mathbf{w})$ as a function of number of iterations. Also shown is the error function during the optimization of figure 39.6. (d) The objective function $M(\mathbf{x})$ as a function of number of iterations. See also figures 41.6 and 41.7.

and a step in $\mathbf{w}$ is made, given by

$$\Delta\mathbf{w} = -\tfrac{1}{2}\epsilon^2\mathbf{g} + \epsilon\mathbf{p}. \tag{41.20}$$

Notice that if the $\epsilon\mathbf{p}$ term were omitted this would simply be gradient descent with learning rate $\eta = \frac{1}{2}\epsilon^2$. This step in $\mathbf{w}$ is accepted or rejected depending on the change in the value of the objective function $M(\mathbf{w})$ and on the change in gradient, with a probability of acceptance such that detailed balance holds.

The Langevin method has one free parameter, ϵ, which controls the typical step size. If ϵ is set to too large a value, moves may be rejected. If it is set to a very small value, progress around the state space will be slow.

Demonstration of Langevin method

The Langevin method is demonstrated in figures 41.5, 41.6 and 41.7. Here, the objective function is $M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w})$, with $\alpha = 0.01$. These figures include, for comparison, the results of the previous optimization method using gradient descent on the same objective function (figure 39.6). It can be seen that the mean evolution of $\mathbf{w}$ is similar to the evolution of the parameters under gradient descent. The Monte Carlo method appears to have converged to the posterior distribution after about 10 000 iterations.

The average acceptance rate during this simulation was 93%; only 7% of the proposed moves were rejected. Probably, faster progress around the state space would have been made if a larger step size ϵ had been used, but the value was chosen so that the 'descent rate' $\eta = \frac{1}{2}\epsilon^2$ matched the step size of the earlier simulations.

Making Bayesian predictions

From iteration 10,000 to 40,000, the weights were sampled every 1000 iterations and the corresponding functions of $\mathbf{x}$ are plotted in figure 41.6. There is a considerable variety of plausible functions. We obtain a Monte Carlo approximation to the Bayesian predictions by averaging these thirty functions of $\mathbf{x}$ together. The result is shown in figure 41.7 and contrasted with the predictions given by the optimized parameters. The Bayesian predictions become satisfyingly moderate as we move away from the region of highest data density.

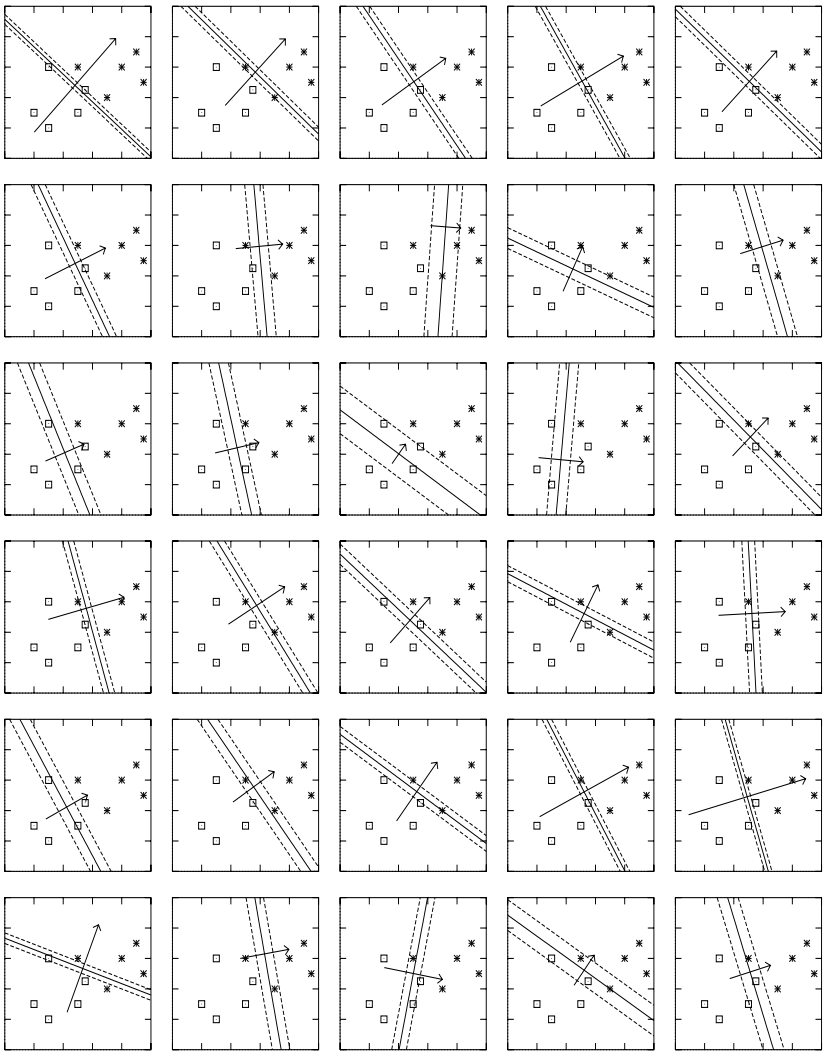

Figure 41.6. Samples obtained by the Langevin Monte Carlo method. The learning rate was set to $\eta = 0.01$ and the weight decay rate to $\alpha = 0.01$. The step size is given by $\epsilon = \sqrt{2\eta}$. The function performed by the neuron is shown by three of its contours every 1000 iterations from iteration 10 000 to 40 000. The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. Also shown is a vector proportional to (w_1, w_2).

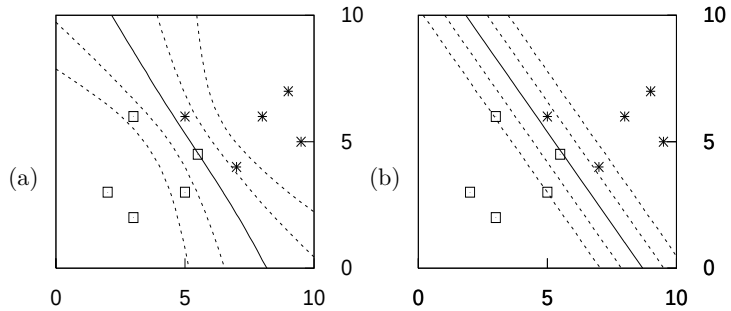

Figure 41.7. Bayesian predictions found by the Langevin Monte Carlo method compared with the predictions using the optimized parameters. (a) The predictive function obtained by averaging the predictions for 30 samples uniformly spaced between iterations 10 000 and 40 000, shown in figure 41.6. The contours shown are those corresponding to $a = 0, \pm 1, \pm 2$, namely $y = 0.5, 0.27$, 0.73, 0.12 and 0.88. (b) For contrast, the predictions given by the 'most probable' setting of the neuron's parameters, as given by optimization of $M(\mathbf{w})$.

```
wnew = w ;
gnew = g ;
for tau = 1:Tau

   p = p - epsilon * gnew / 2 ;     # make half-step in p
   wnew = wnew + epsilon * p ;      # make step in w

   gnew = gradM ( wnew ) ;          # find new gradient
   p = p - epsilon * gnew / 2 ;     # make half-step in p

endfor
```

Algorithm 41.8. `Octave` source code for the Hamiltonian Monte Carlo method. The algorithm is identical to the Langevin method in algorithm 41.4, except for the replacement of the four lines marked `*` in that algorithm by the fragment shown here.

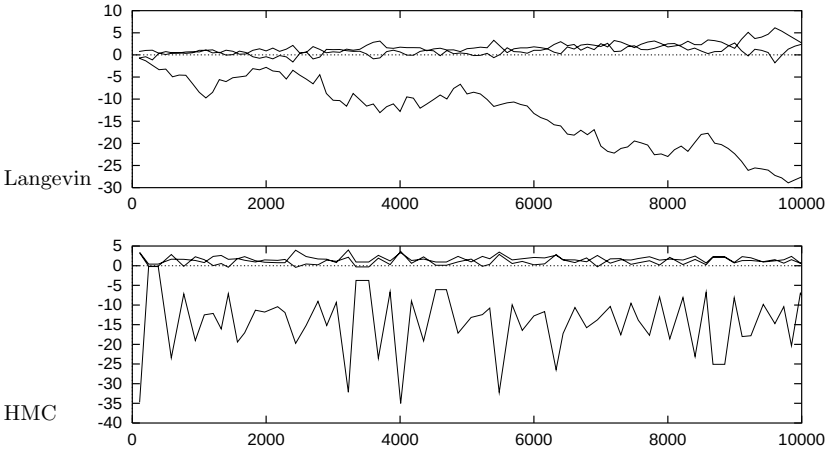

Figure 41.9. Comparison of sampling properties of the Langevin Monte Carlo method and the Hamiltonian Monte Carlo (HMC) method. The horizontal axis is the number of gradient evaluations made. Each figure shows the weights during the first 10,000 iterations. The rejection rate during this Hamiltonian Monte Carlo simulation was 8%.

The Bayesian classifier is better able to identify the points where the classification is uncertain. This pleasing behaviour results simply from a mechanical application of the rules of probability.

Optimization and typicality

A final observation concerns the behaviour of the functions $G(\mathbf{w})$ and $M(\mathbf{w})$ during the Monte Carlo sampling process, compared with the values of G and M at the optimum $\mathbf{w}_{\rm MP}$ (figure 41.5). The function $G(\mathbf{w})$ fluctuates around the value of $G(\mathbf{w}_{\rm MP})$, though not in a symmetrical way. The function $M(\mathbf{w})$ also fluctuates, but it does not fluctuate *around* $M(\mathbf{w}_{\rm MP})$ – obviously it cannot, because M is minimized at $\mathbf{w}_{\rm MP}$, so M could not go any smaller – furthermore, M only rarely drops close to $M(\mathbf{w}_{\rm MP})$. In the language of information theory, *the typical set of* $\mathbf{w}$ *has different properties from the most probable state* $\mathbf{w}_{\rm MP}$.

A general message therefore emerges – applicable to all data models, not just neural networks: one should be cautious about making use of *optimized* parameters, as the properties of optimized parameters may be unrepresentative of the properties of typical, plausible parameters; and the predictions obtained using optimized parameters alone will often be unreasonably overconfident.

Reducing random walk behaviour using Hamiltonian Monte Carlo

As a final study of Monte Carlo methods, we now compare the Langevin Monte Carlo method with its big brother, the Hamiltonian Monte Carlo method. The change to Hamiltonian Monte Carlo is simple to implement, as shown in algorithm 41.8. Each single proposal makes use of multiple gradient evaluations

along a dynamical trajectory in $\mathbf{w}, \mathbf{p}$ space, where $\mathbf{p}$ are the extra 'momentum' variables of the Langevin and Hamiltonian Monte Carlo methods. The number of steps '`Tau`' was set at random to a number between 100 and 200 for each trajectory. The step size ϵ was kept fixed so as to retain comparability with the simulations that have gone before; it is recommended that one randomize the step size in practical applications, however.

Figure 41.9 compares the sampling properties of the Langevin and Hamiltonian Monte Carlo methods. The autocorrelation of the state of the Hamiltonian Monte Carlo simulation falls much more rapidly with simulation time than that of the Langevin method. For this toy problem, Hamiltonian Monte Carlo is at least ten times more efficient in its use of computer time.

▶ 41.5 Implementing inference with Gaussian approximations

Physicists love to take nonlinearities and locally linearize them, and they love to approximate probability distributions by Gaussians. Such approximations offer an alternative strategy for dealing with the integral

$$P(\mathbf{t}^{(N+1)}=1 \,|\, \mathbf{x}^{(N+1)}, D, \alpha) = \int \mathrm{d}^K\mathbf{w}\; y(\mathbf{x}^{(N+1)};\mathbf{w}) \frac{1}{Z_M} \exp(-M(\mathbf{w})), \quad (41.21)$$

which we just evaluated using Monte Carlo methods.

We start by making a Gaussian approximation to the posterior probability. We go to the minimum of $M(\mathbf{w})$ (using a gradient-based optimizer) and Taylor-expand M there:

$$M(\mathbf{w}) \simeq M(\mathbf{w}_{\mathrm{MP}}) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_{\mathrm{MP}})^{\mathsf{T}}\mathbf{A}(\mathbf{w}-\mathbf{w}_{\mathrm{MP}}) + \cdots, \quad (41.22)$$

where $\mathbf{A}$ is the matrix of second derivatives, also known as the *Hessian*, defined by

$$A_{ij} \equiv \frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w}) \Big|_{\mathbf{w}=\mathbf{w}_{\mathrm{MP}}} . \quad (41.23)$$

We thus define our Gaussian approximation:

$$Q(\mathbf{w}; \mathbf{w}_{\mathrm{MP}}, \mathbf{A}) = [\det(\mathbf{A}/2\pi)]^{1/2} \exp\left[-\frac{1}{2}(\mathbf{w}-\mathbf{w}_{\mathrm{MP}})^{\mathsf{T}}\mathbf{A}(\mathbf{w}-\mathbf{w}_{\mathrm{MP}})\right]. \quad (41.24)$$

We can think of the matrix $\mathbf{A}$ as defining *error bars* on $\mathbf{w}$. To be precise, Q is a normal distribution whose variance–covariance matrix is $\mathbf{A}^{-1}$.

Exercise 41.1.[2] Show that the second derivative of $M(\mathbf{w})$ with respect to $\mathbf{w}$ is given by

$$\frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w}) = \sum_{n=1}^{N} f'(a^{(n)}) x_i^{(n)} x_j^{(n)} + \alpha\delta_{ij}, \quad (41.25)$$

where $f'(a)$ is the first derivative of $f(a) \equiv 1/(1+e^{-a})$, which is

$$f'(a) = \frac{\mathrm{d}}{\mathrm{d}a} f(a) = f(a)(1-f(a)), \quad (41.26)$$

and

$$a^{(n)} = \sum_j w_j x_j^{(n)}. \quad (41.27)$$

Having computed the Hessian, our task is then to perform the integral (41.21) using our Gaussian approximation.

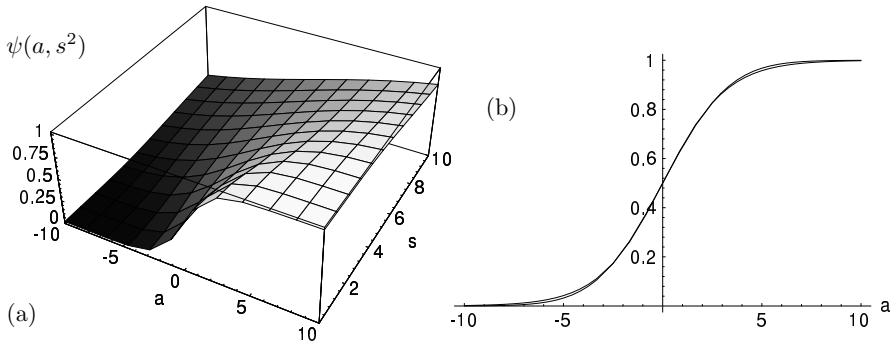

Figure 41.10. The marginalized probability, and an approximation to it. (a) The function $\psi(a, s^2)$, evaluated numerically. In (b) the functions $\psi(a, s^2)$ and $\phi(a, s^2)$ defined in the text are shown as a function of a for $s^2 = 4$. From MacKay (1992b).

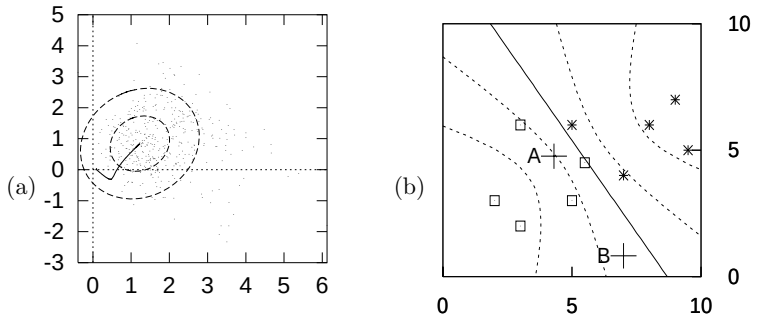

Figure 41.11. The Gaussian approximation in weight space and its approximate predictions in input space. (a) A projection of the Gaussian approximation onto the (w_1, w_2) plane of weight space. The one- and two-standard-deviation contours are shown. Also shown are the trajectory of the optimizer, and the Monte Carlo method's samples. (b) The predictive function obtained from the Gaussian approximation and equation (41.30). (cf. figure 42.2.)

Calculating the marginalized probability

The output $y(\mathbf{x}; \mathbf{w})$ depends on $\mathbf{w}$ only through the scalar $a(\mathbf{x}; \mathbf{w})$, so we can reduce the dimensionality of the integral by finding the probability density of a. We are assuming a locally Gaussian posterior probability distribution over $\mathbf{w} = \mathbf{w}_{\rm MP} + \Delta\mathbf{w}$, $P(\mathbf{w} \,|\, D, \alpha) \simeq (1/Z_Q) \exp(-\frac{1}{2}\Delta\mathbf{w}^{\mathsf{T}}\mathbf{A}\Delta\mathbf{w})$. For our single neuron, the activation $a(\mathbf{x}; \mathbf{w})$ is a linear function of $\mathbf{w}$ with $\partial a/\partial \mathbf{w} = \mathbf{x}$, so for any $\mathbf{x}$, the activation a is Gaussian-distributed.

▷ Exercise 41.2.[2] Assuming $\mathbf{w}$ is Gaussian-distributed with mean $\mathbf{w}_{\rm MP}$ and variance–covariance matrix $\mathbf{A}^{-1}$, show that the probability distribution of $a(\mathbf{x})$ is

$$P(a \,|\, \mathbf{x}, D, \alpha) = \text{Normal}(a_{\rm MP}, s^2) = \frac{1}{\sqrt{2\pi s^2}} \exp\left(-\frac{(a - a_{\rm MP})^2}{2s^2}\right), \tag{41.28}$$

where $a_{\rm MP} = a(\mathbf{x}; \mathbf{w}_{\rm MP})$ and $s^2 = \mathbf{x}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{x}$.

This means that the marginalized output is:

$$P(t=1 \,|\, \mathbf{x}, D, \alpha) = \psi(a_{\rm MP}, s^2) \equiv \int \mathrm{d}a \, f(a) \, \text{Normal}(a_{\rm MP}, s^2). \tag{41.29}$$

This is to be contrasted with $y(\mathbf{x}; \mathbf{w}_{\rm MP}) = f(a_{\rm MP})$, the output of the most probable network. The integral of a sigmoid times a Gaussian can be approximated by:

$$\psi(a_{\rm MP}, s^2) \simeq \phi(a_{\rm MP}, s^2) \equiv f(\kappa(s) a_{\rm MP}) \tag{41.30}$$

with $\kappa = 1/\sqrt{1 + \pi s^2/8}$ (figure 41.10).

Demonstration

Figure 41.11 shows the result of fitting a Gaussian approximation at the optimum $\mathbf{w}_{\rm MP}$, and the results of using that Gaussian approximation and equa-

tion (41.30) to make predictions. Comparing these predictions with those of the Langevin Monte Carlo method (figure 41.7) we observe that, whilst qualitatively the same, the two are clearly numerically different. So at least one of the two methods is not completely accurate.

▷ Exercise 41.3.[2] Is the Gaussian approximation to $P(\mathbf{w} \mid D, \alpha)$ too heavy-tailed or too light-tailed, or both? It may help to consider $P(\mathbf{w} \mid D, \alpha)$ as a function of one parameter w_i and to think of the two distributions on a logarithmic scale. Discuss the conditions under which the Gaussian approximation is most accurate.

Why marginalize?

If the output is immediately used to make a (0/1) decision and the costs associated with error are symmetrical, then the use of marginalized outputs under this Gaussian approximation will make no difference to the performance of the classifier, compared with using the outputs given by the most probable parameters, since both functions pass through 0.5 at $a_{\mathrm{MP}} = 0$. But these Bayesian outputs will make a difference if, for example, there is an option of saying 'I don't know', in addition to saying 'I guess 0' and 'I guess 1'. And even if there are just the two choices '0' and '1', if the costs associated with error are unequal, then the decision boundary will be some contour other than the 0.5 contour, and the boundary will be affected by marginalization.