Complex Systems **5** (1991) 603–643

Bayesian Back-Propagation

Wray L. Buntine*
RIACS & NASA Ames Research Center
Mail Stop 269-2, Moffet Field, CA 94035, USA

Andreas S. Weigend†
Xerox Palo Alto Research Center,
3333 Coyote Hill Rd., Palo Alto, CA, 94304, USA

Abstract. Connectionist feed-forward networks, trained with back-propagation, can be used both for nonlinear regression and for (discrete one-of-C) classification. This paper presents approximate Bayesian methods to statistical components of back-propagation: choosing a cost function and penalty term (interpreted as a form of prior probability), pruning insignificant weights, estimating the uncertainty of weights, predicting for new patterns ("out-of-sample"), estimating the uncertainty in the choice of this prediction ("error bars"), estimating the generalization error, comparing different network structures, and handling missing values in the training patterns. These methods extend some heuristic techniques suggested in the literature, and in most cases require a small additional factor in computation during back-propagation, or computation once back-propagation has finished.

1. Introduction

Back-propagation [32] is a popular scheme for training feed-forward connectionist networks. It can be applied to both the tasks of classification (prediction of discrete variables taking one of C mutually exclusive and exhaustive values) and regression (prediction of real variables). Design issues in this scheme are primarily computational—for instance, what variations of gradient descent should be used—or probabilistic—for instance, what cost function should be used and how generalization error can be predicted. Here we frame the probabilistic component of back-propagation in a Bayesian context [6, 3, 7, 27]. In adopting a Bayesian justification for the methods presented, we are not claiming any neurological validity for our methods. We view

*Electronic mail address: wray@kronos.arc.nasa.gov
†Electronic mail address: andreas@psych.stanford.edu

feed-forward networks as a vehicle for massive parallelism in classification, regression, and learning.

Some of the material presented here is review in that it is an application of existing Bayesian theory and approximation methods. This applies, for instance, to the results on generalization error in section 7.1, and to the basic posterior probabilities presented in Lemma 4.1 (see, for instance, [37]). We only address the case in which there is a single real or discrete variable being predicted, although the discrete variable may have multiple values; extension to the multiple-variable case is quite straightforward. We also only address problems in which noise or uncertainty exists in classification or prediction because noise-free problems are rare in practice and the mathematical behavior of the noise-free case is quite different.

Section 2 gives some theoretical background and motivation for using Bayesian methods, and discusses some alternative approaches. Section 3 outlines the notation and describes some simple network forms that correspond to variations of standard probability functions. A good feed-forward network system should therefore subsume the tasks of several special-purpose statistical systems, albeit at some computational cost. Section 4 presents a probabilistic analysis of the training of feed-forward neural networks. Section 5 expands on the important issue of designing a prior distribution over network weights. Section 6 uses these results to present Bayesian embellishments for the standard back-propagation algorithm: cost functions and weight-evaluation measures. Several minimum encoding approaches [40, 2, 33] are also explained in section 6.3. Finally, section 7 discusses some extensions to back-propagation that involve weight pruning strategies, prediction of variables and generalization error, and handling of missing values.

2. On Bayesian methods

Bayesian methods are usually considered attractive because, in the context of average-case analysis, they offer a rigorous framework for making all assumptions in a learning problem explicit, and they come with a guarantee of average case optimality conditioned on the assumptions made. The optimality properties of Bayesian methods hold because the methods are "normative" or "rational" [3, 15]; any other approach not approximating them should not perform as well on average. For the learning problem, the entire method is derived as an approximation to applying the single simple principle

$$\text{posterior} \propto \text{prior} \cdot \text{sample-likelihood}$$

to the training problem, together with some principles for reasoning about the prior knowledge available about networks. The term "probabilities" here corresponds to a relative measure of belief in the many possible network weights. Since we can never determine the "correct" or "best" weights, we should carefully reason about the many reasonable possibilities for network weights.

The Bayesian framework carefully separates the components of a learning problem:

Utilities: the utility or loss functions for the problem, which represent the goal of learning and correspond to quantities such as minimum errors or least squares.

Network likelihood model: the likelihood function for the network, such as a "Gaussian error model" as used in regression, which gives a statistical model of the network and how the data is expected to have been generated initially.

Priors: the priors[1] on networks and network weights, which represent our expectations about networks before receiving any data, and correspond to penalty terms or regularizers.

See also [14] for a discussion in the uniform convergence framework.

Notice that in general there can be no "correct" prior, error model, or likelihood since by definition these vary from problem to problem. It is challenging to make a choice that seems appropriate in a given circumstance, and Bayesians place considerable emphasis on these modeling tasks. The optimality of Bayesian methods rests on modeling assumptions that correspond to the above components:

(I) The choice of models or hypotheses (in our case the network likelihood model) being searched must contain the "true" model (in practice this means a fairly accurate approximation to the "true" model).

(II) The choice of a prior over these models should represent a reasonable initial preference for models in the search space. There is considerable literature on how to choose a prior to minimize the assumptions implicit in the prior [3].

Bayesian methods then guarantee best average-case performance given these two assumptions and a third assumption:

(III) The approximations made in implementation are sufficiently accurate.

Poor modeling will lead to strong but inappropriate assumptions. For example, a single-layer perceptron network cannot capture certain higher-order functions, so it is inappropriate for tasks known to have higher-order components.

There is a subtle distinction between some statistical mechanics formulations and a Bayesian approach. Statistical mechanics theory has been developed in the context of training a perceptron in a noise-free environment to estimate the generalization error [35]. The statistical mechanics idea of

[1]We use the term "prior" to mean a prior distribution, prior probability, or prior probability density function. Likewise for posterior.

an ensemble of networks [22] corresponds to the Bayesian posterior. Notions such as the "true" subjective probability function [13], the "correct" likelihood function derived from the additive energy function [22], and the use of probabilistic modeling in the context of "unrealizable" models [35] (when assumption (I) fails so the search space does not contain the "true" model) are only partially consistent with a Bayesian approach. Apart from some philosophical distinctions such as these, the approaches are equivalent. Seung, Sompolinsky, and Tishby suggest Gibbs sampling[2] to approximate the posterior distribution of network weights, whereas in the approach described here we use more efficient but sometimes less accurate Gaussian approximations around local maxima of the posterior distribution of network weights.

Bayesian methods should not be confused with results that back-propagation methods are Bayes optimal [17]. Bayes optimal refers to the property that back-propagation methods should produce a network that approaches the greatest lower bound on error (or, more generally, risk) as the training sample size approaches infinity. Bayesian methods described here share this property but also have a more powerful property: they are approximating a method that produces a classifier/regression that will on average have equal or lower error (risk) than a classifier/regression produced by *any* other method applied to the same training sample. Notice, this holds for the current training sample, not just the infinite one considered by Bayes optimality. It does, however, rely on the modeling assumptions listed above.

Uniform convergence methods [4, 14], the basis for earlier computational learning theory, approximate Bayesian methods when the sample size is large [7, section 4.2]. Uniform convergence methods require a sample size that is large enough that the prior term found in Bayesian methods becomes insignificant (and so can be ignored). Uniform convergence methods make no assumptions,[3] but instead have to ignore useful information in the training set [7, section 4.2]. As a consequence, they can only guarantee good performance in the worst case, so they sacrifice the guarantee of average-case optimality. Notice that for larger samples the worst case and the average case converge, so uniform convergence methods are simpler and adequate. For smaller samples the worst and average cases can be far different [7], so the extra effort involved in the Bayesian approach is worth it.

It is important to bear in mind, however, that Bayesian methods are not a replacement for uniform convergence methods, as indeed they are not a replacement for other techniques such as cross-validation or minimum encoding methods. All methods have particular algorithmic, complexity, and statistical properties that make them more appropriate in one engineering context or another. Minimum encoding methods such as MML [40] and MDL [33] are often viewed as approximate Bayesian methods and are used

[2]See [12] for a motivated tutorial introduction to this widely used technique.

[3]The model space does not have to contain the "true" model and no prior is needed.

as a useful metaphor, and their implementation is somewhat simpler than a full Bayesian approach with averaging [6, 36].

3. Multi-layer networks

The networks we consider consist of a directed acyclic graph, giving the network structure together with the activation functions at each unit or node, which relate inputs to activation output. The network deals with real numbers internally, although the inputs may be discrete but represented as real numbers 0 and 1. We first explain the notation we use in section 3.1, and then describe some simple networks in section 3.2 that form the basis of later examples.

3.1 Notation

Denote the input variables to the network as a vector of values x and denote the response variable that the network is intended to predict as y. In the case of regression, the network output corresponds to the predicted regression or expected value for y conditioned on the values of the input variables x. In the case of one-of-C classification, the network outputs a conditional probability distribution over C possible (mutually exclusive and exhaustive) values for the discrete variable y, conditioned on the values of the input variables. The output comes from C nodes and corresponds to a vector of real values summing to 1. The ith value is the estimated conditional probability that the output variable should have the ith discrete value.

Let the non-output nodes in the network be indexed $n = 1, 2, \ldots$. In regression the output node has activation o, and in one-of-C classification the output nodes have activation $o_1, \ldots, o_C$. The activation of any non-output node n is denoted u_n, which is a real number. The activation function for the node m is a function of the activations u_n for nodes n inputting to node m, and the function is parameterized by a vector of parameters w_m $(= w_{m,1}, w_{m,2}, \ldots)$.

The exact details of the activation functions relating inputs to a nodes activation is not important for the analysis here, except that they are parameterized by network weights. A typical activation function for a node n with vector of weights w_n would be the sigmoid acting on the weighted sum of its inputs $u_1, \ldots, u_l$,

$$u(u_1, \ldots, u_l) = \frac{1}{1 + \exp\left(\sum_{i=1,\ldots,l} w_{n,i} u_i + w_{n,0}\right)}.$$

Some other activation functions that have been used are independent Gaussians found in radial basis, exponential, and various trigonometric functions [10].

In the case of one-of-C classification, the output activations need to be non-negative and sum to one. To do this, the output activation functions

may normalize a set of non-negative activations from nodes in the previous layer $u_1, \ldots, u_C$ using the function

$$o_i(u_1, \ldots, u_C) = \frac{u_i}{\sum_{j=1,\ldots,C} u_j}.$$

We call this function a normalizing activation function. A similar activation function, the Softmax function [5], first uses the exponential function to ensure positivity.

3.2 Probabilistic neural networks

Two key issues in network design are whether the network can sufficiently approximate the "true" function that needs to be computed, and whether the network structure and weights have an interpretation or intrinsic form that can be explained as the "knowledge" in the network rather than treating the network as a black box. This second issue is more important than one might think. If network weights have some clear interpretation then we can configure the network in coherent ways, an expert can view the knowledge represented by the network, and we can more readily assign meaningful priors or some other initialization for the network weights (for instance, a non-probabilistic scheme for initialization of weights is described in [38]). Probabilistic interpretations of networks are also discussed in [5, 26].

In this section we present network structures in which these issues are well understood. One of the advantages of the general network approach, however, is that the one-network package can be used to implement all these network types and more, just by replacing the network activation functions and their derivatives. While we call these probabilistic neural networks, they should not be confused with the probabilistic networks that are used in artificial intelligence [28]. The neural networks are probabilistic in the sense that they implement a well-known probability function. Logistic networks and regression networks, both discussed below, are well known in the connectionist community. We include them here for completeness, and as a basis for section 5.

3.2.1 Logistic networks

Logistic networks generalize the sigmoid activation function to produce a family of simple networks that reproduce logistic statistical models for discrimination and regression [24]. They are used for classification where the output variable y is discrete with values $1, \ldots, C$.

Suppose we have key features of the input variables that may take the form of boolean functions on binary variables $x_1 x_4$, $x_3 \overline{x_4}$, or $x_2 \overline{x_3} x_5$, or quadratic and possibly multivariate functions on real variables x_1^2 or $x_1 x_2$. In general the key features are represented by real-valued functions $c_1, \ldots, c_f$ of the input variables. Then an exponential distribution [24] on (x, y) can be

defined in terms of these features,

$$\Pr(x, y) = \exp\left(\alpha_{0,y} + \sum_{i=1,\ldots,f} \alpha_{i,y} c_i(x)\right),$$

where $\alpha_{i,y}$ are some parameters. We are interested in the conditional probability distribution of y given x, so we take the conditional version of this,

$$\Pr(y \mid x, \alpha) = \frac{\exp\left(\alpha_{0,y} + \sum_{i=1,\ldots,f} \alpha_{i,y} c_i(x)\right)}{\sum_y \exp\left(\alpha_{0,y} + \sum_{i=1,\ldots,f} \alpha_{i,y} c_i(x)\right)}. \tag{1}$$

The representation in terms of α is redundant, however. To see this, divide both sides by $\exp(\alpha_{0,1} + \sum_{i=1,\ldots,f} \alpha_{i,1} c_i(x))$ and change variables $(\alpha_{i,y} - \alpha_{i,1})$ to $w_{i,y}$. This gives an equivalent form without redundant variables,

$$\Pr(y \mid x, w) = \begin{cases} \dfrac{1}{1 + \sum_{y'>1} \exp(w_{0,y'} + \sum_{i=1,\ldots,f} w_{i,y'} c_i(x))} & \text{for } y = 1, \\ \dfrac{\exp(w_{0,y} + \sum_{i=1,\ldots,f} w_{i,y} c_i(x))}{1 + \sum_{y'>1} \exp\left(w_{0,y'} + \sum_{i=1,\ldots,f} w_{i,y'} c_i(x)\right)} & \text{otherwise.} \end{cases}$$

Notice that, for y binary and the features $c_i(x)$ taking the simple form x_i, we recover the sigmoid function. By introducing enough higher-order features, any conditional probability distribution on binary variables can be represented. In practice, higher-order features would have to be carefully chosen to represent those kinds of features expected in the data.

These functional forms can be implemented as networks in the following manner. The first layer computes the higher-order features, if any. The second layer computes from the first the $C - 1$ exponential functions (for $y' > 1$), where the weights are $w_{i,y'}$. The final layer is a standard normalizing layer with one input node always set to 1.

3.2.2 Regression networks

Another statistical model that has a simple network representation is linear regression [18], where the mean of the response variable is usually given by

$$y = \sum_{i=1,\ldots,K} w_i f_i(x)$$

for "basis functions" f_i and "parameters" or weights w_i. A corresponding network has K hidden nodes without weights, which compute the functions $f_i(x)$, leading into linear output nodes that compute the linear regression. The regression is called linear because it is a linear function of the weights w_i, whereas the functions f_i can be arbitrary.

The various techniques such as smoothing, penalized least-squares, and cross validation applied to these systems can then be cast in a network framework; see MacKay [23] for a Bayesian interpretation. The choice of basis functions depends on the anticipated behavior of y. Some choices are products

of $\sin n_i x_i$ and $\cos n_i x_i$ for integer n_i when $x_i \in [-\pi, \pi]$; or combinations of Legendre polynomials or their integrals when $x_i \in [-1, 1]$ and y is expected to be in the form of a polynomial; or combinations of Hermite functions when $x_i \in [-\infty, \infty]$ and y is expected to be in the form of a product of $\exp(-\sum_{i=1}^{A} x_i^2/2)$ and some polynomial in x. Notice each of these choices may require rescaling the x_i's initially.

Of course, in the more general case, we can use a semi-linear model such as

$$y = \sum_{i=1,\ldots,K} w_{0,i} f_i \left(\sum_j w_{i,j} x_j \right),$$

which again has a straightforward network interpretation.

4. Probabilistic analysis

This section covers each component of a Bayesian analysis in turn: interpreting networks as a probability function in section 4.1; thereby calculating the likelihood of the training sample for a given network and weights, in section 4.2; and finally considering posteriors for the weights in a network, in section 4.4. A more thorough treatment of priors for the weights in a network is left until section 5.

The expected value notation described below is used throughout the subsequent analysis. $\mathbf{E}_{A|B}(f(A,B))$ denotes the expected (mean) value of the function $f(A,B)$ when A is distributed according to the probability function or probability density function $\Pr(A \mid B)$. That is, B is given, and we wish to find the mean of $f(A,B)$. If A is continuous, this is calculated with an integral

$$\mathbf{E}_{A|B}(f(A,B)) \equiv \int_A f(A,B) \Pr(A \mid B)\, dA,$$

and if A is discrete this is calculated with a sum

$$\mathbf{E}_{A|B}(f(A,B)) \equiv \sum_A f(A,B) \Pr(A \mid B).$$

For mixed continuous and discrete variables, combinations of these formulas apply. Similarly, the notation $\mathbf{v}_{A|B}(f(A,B))$ denotes the variance of $f(A,B)$, usually calculated as

$$\mathbf{v}_{A|B}(f(A,B)) \equiv \mathbf{E}_{A|B}\left(\left(f(A,B) - \mathbf{E}_{A|B}(f(A,B)) \right)^2 \right).$$

4.1 The network likelihood function

To do Bayesian or likelihood analysis on feed-forward networks, we have to use the network output $o(x,w)$, a function of the inputs x and the weights w, to construct a likelihood function $l(y \mid x, w)$ for the observed pattern (x,y). This likelihood function gives the conditional probability distribution for the

output variable y conditioned on the input variables x, given a particular network and weights. This likelihood function is the basis of all subsequent analysis.

In the case of classification, the likelihood function is a probability distribution representing the network's estimate of the "true" conditional distribution for y given x. The C network outputs therefore give a conditional probability vector. The ith output $o_i(x, w)$ represents the conditional probability that the discrete output variable y takes its ith value. So

$$l(y \mid x, w) = o_y(x, w).$$

In the case of regression, the likelihood function is a probability density function (it integrates over the domain of y to 1) of the form $l(y \mid x, w, F)$ for the output variable y conditioned on x, the network weights, and some other information F such as a standard deviation of error. The network output is usually defined to be the mean of the likelihood function given by

$$o(x, w) \equiv \int_y y \; l(y \mid x, w, F) \, dy.$$

The likelihood is usually defined in terms of some *error model* that is a function of y, the mean $o(x, w)$, and some other error parameters. An error model commonly used is the Gaussian distribution with mean $o(x, w)$ and standard deviation σ that is unknown (so has to be estimated along with the weights w). This means the true y is expected to vary about the mean $o(x, w)$ with constant standard deviation σ. It is more realistic that the standard deviation itself should also be a function of x. In this case, a second output node can be connected to the network to estimate the standard deviation; we denote its output by $o'(x, r)$, and note that the weights r may have parameters in common with w. Vapnik [39] has also suggested using the Laplace distribution as an error model when the experimental conditions may vary with maximal uncertainty, but the standard deviation is still independent of x. This leads to the following choice of error models: with known/unknown standard deviation σ of

$$l_G(y \mid x, w, \sigma) \equiv \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - o(x, w))^2}{2\sigma^2}\right);$$

with x-dependent standard deviation $o'(x, r)$ of

$$l_{GD}(y \mid x, w, r) \equiv \frac{1}{\sqrt{2\pi}o'(x, r)} \exp\left(-\frac{(y - o(x, w))^2}{2o'(x, r)^2}\right);$$

and with experiment-dependent standard deviation of

$$l_L(y \mid x, w, \Delta) \equiv \frac{1}{2\Delta} \exp\left(-\frac{|y - o(x, w)|}{\Delta}\right).$$

While other error models may be appropriate for a given problem, these three are sufficient to demonstrate our methods.

A similar calculation can be done for the Laplacian error model with unknown standard deviation Δ, and this time

$$\mathbf{E}_{\Delta|w,\boldsymbol{x},\boldsymbol{y}}(\Delta) = \frac{1}{N-1} \sum_{i=1,\ldots,N} |y_i - o(x_i, w)|.$$

To summarize, posterior probabilities for a set of network weights w are as follows.

Lemma 4.1. *Consider the classification and regression frameworks described above for neural networks. Posterior probabilities of network weights are as follows. For regression with Gaussian error and unknown σ with prior $\Pr(\sigma \mid \boldsymbol{x}, w) = 1/\sigma$,*

$$\begin{aligned}\Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) &\propto \Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) \\ &= \Pr(w \mid \boldsymbol{x}) \frac{\Gamma(N/2)}{2\pi^{N/2} \left(\sum_{i=1,\ldots,N} (y_i - o(x_i, w))^2\right)^{N/2}},\end{aligned}$$

for regression with Laplacian error and unknown Δ with prior $\Pr(\Delta \mid \boldsymbol{x}, w) = 1/\Delta$,

$$\begin{aligned}\Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) &\propto \Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) \\ &= \Pr(w \mid \boldsymbol{x}) \frac{\Gamma(N)}{2^N \left(\sum_{i=1,\ldots,N} |y_i - o(x_i, w)|\right)^N};\end{aligned}$$

and for classification,

$$\Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \propto \Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) = \Pr(w \mid \boldsymbol{x}) \prod_{i=1,\ldots,N} o_{y_i}(x_i, w).$$

These posterior probabilities, together with suitable priors, define the Bayesian solution to the problem of training networks. High posterior weights w shift the weights away from the maximum likelihood (or minimum errors) solution toward a solution with higher prior $\Pr(w \mid \boldsymbol{x})$. With a Gaussian error model, the trade-off is with mean-square error; with a Laplacian error model, the average absolute deviation. In all cases, as the sample size N gets large, the shift due to the prior term will become negligible.

In the noise-free case, either the training sample agrees with the network output, or not. In this simple case the likelihood function becomes a delta function, so the posterior becomes the projection of the prior onto the subspace of networks whose output is everywhere consistent with the training sample.

$$\Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \propto \Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) = \begin{cases} \Pr(w \mid \boldsymbol{x}) & \text{if } w \text{ consistent with } (\boldsymbol{y}, \boldsymbol{x}), \\ 0 & \text{otherwise.} \end{cases}$$

This is very different to analyze mathematically than the noisy cases above, and we do not consider this here (see, for instance, [16, 27]).

5. Prior probability of the weights

A prior for a network is related to the notions of "penalty term" or "regularizer" used in the statistics literature [18], weight decay [29] and weight elimination [41], and the "complexity term" used in minimum encoding methods [2]. Rather than using the weights that give minimum error, as for instance plain back-propagation tries to do, priors allow a shift in weight space toward a set of weights yielding for instance a "smoother" network. The use of priors gives a probabilistic regime for modulating this shift and, as mentioned in section 2, can make a difference in performance with smaller training samples.

We suggest a few priors here; however, we believe more experience and research is needed in designing priors for feed-forward networks. In general there is no "correct" prior, because each learning problem is unique. However, it is important to use a prior that has reasonable properties, and to choose a particular prior that matches the properties of the problem. See [6] where a number of different priors are discussed for trees, and [23] where several priors are tried and compared for a single network learning problem.

Priors for networks fall into three broad styles.

First, there are priors that say how accurate the predictions should be on individual examples. In regression, this corresponds to saying you expect the variance σ^2 to be "just so," small or large, whatever that might be. In classification this corresponds to saying you expect the class probability vector given by the network outputs, $(o_1(x,w),\ldots,o_C(x,w))$, to be extreme (one probability is near 1.0 and the others near 0.0), or roughly around the population base rate, and so forth. For instance, in the two-class case, suppose 90% of examples in the population as a whole are known to be of class 1, so 10% are of class 2. Then you might say you expect network outputs for a particular example to vary little from this, or to vary quite widely. These kinds of priors, when well chosen, can improve the out-of-sample accuracy of tree algorithms by several percent [6]. Entropic priors, described in section 5.1, are concerned with this accuracy of prediction.

Second, there are priors that say how "smooth" you expect the predictions to be, that is, how predictions vary as you change the network inputs. "Penalty terms" and "smoothers" are used in statistics to achieve this. The idea is that you expect *a priori* a fairly smooth function of the input, or you expect output to be of a certain magnitude. When there is little data to fit, you therefore choose the smoother fit over the rougher fit. While this should not be true in general, it is a property that one aims for by judicious choice of "key" input variables, or by careful structuring of the network to capture key features such as "spatial invariance." We describe smoothness measures for both logistic and regression networks in sections 5.2 and 5.3. Following that, we discuss in section 5.4 how these measures can be used to construct a prior. We believe similar strategies can be developed for the more general networks used in practice, but we do not present any here.

Finally, there are priors that embody much more specific information than general notions of "smoothness" or "accuracy." Schemes for initialization of weights as described in [38] are a step in this direction.

5.1 Entropic priors

Entropic priors [34] are becoming accepted as a standard of the so-called non-informative priors. They take the form (we ignore the "measure term" here)

$$\Pr(\theta) \propto \exp\left(-\alpha I_{z|\theta}(\theta)\right),$$

where α is a negative or positive prior parameter, and $I_{z|\theta}(\theta)$ denotes the entropy of z when θ is known. The function

$$I_{z|\theta}(\theta) = -\int \Pr(z|\theta) \log \Pr(z|\theta)\, dz$$

is the usual entropy function giving the entropy for z distributed as $\Pr(z|\theta)$, which is a function of θ. When $\alpha = 1$, these include Zellner's maximal data information prior [42], which can be used to yield most of the textbook "non-informative" priors. For α positive, these priors prefer a more informative prediction about a network output. So in regression they prefer a low variance σ^2, and for classification they prefer class probabilities $(o_y(x, w))$ to be output that make one of the classes highly likely.

When doing classification, we are constructing a network for the conditional distribution $\Pr(y \mid x, w)$. For these we need a prior distribution on the weights w conditioned on knowing the input vectors $\boldsymbol{x} \equiv x_1, \ldots, x_N$ from the training sample, $\Pr(w \mid \boldsymbol{x})$. An entropic prior can be formed as follows:

$$\begin{aligned}
\Pr(w \mid \boldsymbol{x}) &\propto \exp\left(-\left(I_{x|\boldsymbol{x}} + \mathbf{E}_{x|\boldsymbol{x}}\left(I_{y|x,w}(w)\right)\right)\right) \\
&\propto \exp\left(-\mathbf{E}_{x|\boldsymbol{x}}\left(I_{y|x,w}(w)\right)\right) \\
&\approx \exp\left(-\frac{1}{N}\sum_{d=1}^{N} I_{y|x_d,w}(w)\right).
\end{aligned}$$

The entropy of x, namely $I_{x|\boldsymbol{x}}$, is ignored since it is independent of w. The last line estimates the expectation $\mathbf{E}_{x|\boldsymbol{x}}(\cdot)$ with the empirical average from the sample of input vectors $\boldsymbol{x}$.

For the classification case,

$$\begin{aligned}
I_{y|x_d,w}(w) &= -\sum_{i=1}^{C} o_i(x_d, w) \log o_i(x_d, w), \\
\Pr(w \mid \boldsymbol{x}) &\propto \prod_{d=1}^{N}\prod_{i=1}^{C} o_i(x_d, w)^{o_i(x_d,w)/N}.
\end{aligned}$$

For the regression case with x-dependent standard deviation of $o'(x, r)$,

$$I_{y|x_d,w,r}(w, r) = \log o'(x_d, r) + \text{constant},$$
$$\Pr(w, r \mid \boldsymbol{x}) \propto \prod_{d=1}^{N} \frac{1}{o'(x_d, r)^{1/N}}.$$

Notice that this prior is independent of the weights w. This is because the entropic prior is only concerned about information content of network outputs, which in turn is a function of variance or error and not of the predicted value for y. Applying the same entropic construction to the case of regression, with unknown σ or Δ, we reconstruct the priors used in Lemma 4.1 for σ and Δ.

5.2 Smoothness of logistic networks

A measure of smoothness on logistic networks that can be readily calculated is the mean square change in log-odds. While we do not claim this is in any sense the "best" measure of smoothness, it is at least a quadratic function. We develop this here for the simple case where all input variables and the output variable y are binary. This corresponds to the case where there is a single sigmoid in the network receiving input from higher-order binary features of the input variables. The network output is

$$o_1(x, w) = \frac{1}{1 + \exp\left(w_0 + \sum_{i=1,\ldots,f} w_i c_i(x)\right)}.$$

Here, the functions $c_i(x)$ take the value 0 (off) or 1 (on), and we will restrict them to be conjunctions of the inputs, such as $c_1(x) = x_1 \overline{x_3} x_6$.

Suppose the input x is an A-dimensional binary vector taking values such as (0,0,1,0,1,1) when $A = 6$. A particular input x has a set of neighbor values $\text{nbr}(x)$ that differ from x in just one place. For $x = (0, 0, 1, 0, 1, 1)$ these are given by

$$\text{nbr}(\,(0,0,1,0,1,1)\,) = \left\{ \begin{array}{lll} (1,0,1,0,1,1), & (0,1,1,0,1,1), & (0,0,0,0,1,1), \\ (0,0,1,1,1,1), & (0,0,1,0,0,1), & (0,0,1,0,1,0) \end{array} \right\}.$$

A local measure of the smoothness at this input x is the mean square change in log-odds between x and its neighbors, given by

$$\sum_{x' \in \text{nbr}(x)} \log^2 \frac{o_2(x, w) o_1(x', w)}{o_1(x, w) o_2(x', w)}.$$

An average measure of smoothness for a network is then

$$S(w) = \frac{1}{2} \sum_x \sum_{x' \in \text{nbr}(x)} \log^2 \frac{o_2(x, w) o_1(x', w)}{o_1(x, w) o_2(x', w)} \Big/ \sum_x 1$$
$$= \sum_{i,j} w_i w_j S_{i,j},$$

where

$$S_{i,j} = \frac{1}{2}\sum_x \sum_{x' \in \mathrm{nbr}(x)} (c_i(x) - c_i(x'))(c_j(x) - c_j(x')) \Big/ \sum_x 1.$$

Notice the bias w_0 does not appear in this expression. The diagonals of the quadratic form, $S_{i,i}$, give the proportion of cases the binary feature c_i changes when the input is changed by one variable. The off-diagonals of the quadratic form, $S_{i,j}$, measure how often the features c_i and c_j change together. $(c_i(x) - c_i(x'))(c_j(x) - c_j(x'))$ is 1 if both features change to on or to off, -1 if both features change but in different ways, and 0 if any one feature is constant.

To evaluate the quadratic form S we need additional terms, which we will explain using $c_1(x) = x_1\overline{x_3}x_6$, $c_2(x) = x_1x_3x_5$, $c_3(x) = x_2x_3$, and $c_4(x) = x_1$.

$\mathrm{confl}_{i,j}$: number of conflicting literals between the conjuncts for c_i and c_j.

$\mathrm{same}_{i,j}$: number of equivalent literals between the conjuncts for c_i and c_j.

$\mathrm{sep}_{i,j}$: number of literals only in one or the other of c_i and c_j.

Then $\mathrm{confl}_{1,2} = 1$ (due to x_3), $\mathrm{confl}_{2,3} = 0$, $\mathrm{same}_{1,2} = 1$ (due to x_1), $\mathrm{same}_{2,3} = 1$, $\mathrm{sep}_{1,2} = 2$ (due to x_5 and x_6), and $\mathrm{sep}_{2,3} = 3$.

Two features c_i and c_j can only be both on if they have 0 conflicts, $\mathrm{confl}_{i,j} = 0$. Changing a single variable such as x_3 can only cause both features to go on together if the variable x_3 is shared by the features. Likewise, changing a single variable can only cause one feature to go off and one to go on if the variable is conflicting. Therefore,

$$S_{i,j} = \begin{cases} \dfrac{\mathrm{same}_{i,j}}{2^{\mathrm{same}_{i,j}+\mathrm{sep}_{i,j}}} & \mathrm{confl}_{i,j} = 0, \\ \dfrac{-1}{2^{\mathrm{same}_{i,j}+\mathrm{sep}_{i,j}+1}} & \mathrm{confl}_{i,j} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Notice that $\mathrm{confl}_{i,i} = 0$ and $\mathrm{same}_{i,i}$ is the number of literals in the conjunct for c_i, so the diagonals $S_{i,i}$ are handled in the first case above. So, in the simple example above, $S_{1,1} = 3/8$, $S_{1,3} = -1/16$, $S_{3,3} = 1/2$, and $S_{2,4} = 1/8$.

If the features are simple linear terms $c_i(x) = x_i$—so the network computes a sigmoid of $w_0 + \sum_{i=1,\ldots,A} w_i x_i$—then the smoothness measure becomes $S(w) = \sum_i w_i^2$. If the features are quadratic terms $c_{i,j}(x) = x_i x_j$ for $j < i$ and $c_{i,i}(x) = x_i = x_i x_i$—so the network computes a sigmoid of $w_0 + \sum_{j \le i=1,\ldots,A} w_{i,j} x_i x_j$—then the smoothness measure becomes

$$S(w) = \sum_i w_{i,i}^2 + \frac{1}{4}\sum_{j<i=1,\ldots,A} \left((w_{i,i} + w_{j,j})w_{i,j} + 2w_{i,j}^2\right) + \frac{1}{8}\sum_{j,k<i} w_{i,j}w_{i,k} + \frac{1}{8}\sum_{j,k>i} w_{j,i}w_{k,i} + \frac{1}{4}\sum_{k>i>j} w_{i,j}w_{k,i}.$$

5.3 Smoothness of regression networks

For regression networks, priors say how we expect the mean of y given x to vary with the input x. Some things we might know about *a priori* are the typical smoothness or curvature of y, the order of magnitude of y, or the typical slope of y. This section develops a measure of smoothness for linear regression networks. Incorporation of information such as typical magnitude or slope could be done to develop a more sophisticated measure.

Smoothness is related to curvature, which is measured by the second derivative of y. With x defined over the multi-dimensional space X, the average magnitude of the second derivative of y is given by

$$S(w) = \int_{x \in X} \left\| \frac{d^2 y}{dx^2} \right\| w(x) dx,$$

Here $w(x)$ is some weighting in the space X with $\int_{x \in X} w(x) dx = 1$, for instance, given by $w(x) = 1/\int_{x \in X} dx$. The norm $\| \cdot \|$ is an average measure of the size of the second derivative at a single point. This is given by the average change in y about a ball in x-space of fixed radius from x (i.e., $x + \Delta x$ where $|\Delta x| = 1$) due to the second derivative. Using a second order expansion,

$$\begin{aligned}
\left\| \frac{d^2 y}{dx^2} \right\| &= \int_{|\Delta x|=1} \left(\frac{1}{2} \sum_{i,j=1}^{A} \Delta x_i \Delta x_j \frac{\partial^2 y}{\partial x_i \partial x_j} \right)^2 d\Delta x \Big/ \int_{|\Delta x|=1} d\Delta x \\
&= \left(\frac{d^2 y}{dx_1^2} \right)^2 \qquad \text{for } A = 1, \\
&\approx \frac{1}{4A^2} \left(\sum_{i,j=1}^{A} \frac{\partial^2 y}{\partial x_i^2} \frac{\partial^2 y}{\partial x_j^2} + 2 \sum_{i,j=1}^{A} \left(\frac{\partial^2 y}{\partial x_i \partial x_j} \right)^2 \right) \qquad \text{for large } A.
\end{aligned}$$

The derivation of this approximation is detailed but fairly straightforward, so we do not reproduce it here. We use the approximation since it is proportional to the exact calculation when $A = 1$. To interpret this measure notice that if y represents the top of a sphere of radius a in any dimension, then $\|d^2 y/dx^2\| = 1/a^2$.

This measure of smoothness evaluates on regression models as follows. Substituting into $S(w)$, and using the regression model $y = \sum_{i=1,\ldots,K} w_i f_i(x)$, we get

$$S(w) = \sum_{m,n=1}^{K} w_m w_n S_{m,n},$$

where

$$S_{m,n} = \frac{1}{4A^2} \sum_{i,j=1}^{A} \int_{x \in X} \left(\frac{\partial^2 f_m(x)}{\partial x_i^2} \frac{\partial^2 f_n(x)}{\partial x_j^2} + 2 \frac{\partial^2 f_m(x)}{\partial x_i \partial x_j} \frac{\partial^2 f_n(x)}{\partial x_i \partial x_j} \right) w(x) dx.$$

Values of $S_{m,n}$ for a given set of basis functions $f_n(\cdot)$ only have to be calculated once, but usually lead to a complex matrix. If the basis functions and their derivatives are somewhat orthogonal, then a diagonal approximation to the matrix is quite good, as shown below. For basis functions such as $\{1, x, x^2, x^3, \ldots\}$, changing the x^3 term has dramatic consequences to the x^2 term, and so forth, so this orthogonality property does not hold. We therefore aim to choose basis functions in which orthogonality holds quite well (in the sense that off-diagonal terms of $S_{m,n}$ will be small).

As an example, for the simple case of one-dimensional inputs with basis functions,

$$\{1, \cos x, \cos 2x, \cos 3x, \ldots, \sin x, \sin 2x, \sin 3x, \ldots\}$$

the matrix becomes

1 :	0	0	0	0	0	...	0	0	0	0
$\cos x$:	0	1.178	0	0	0	...	0	2	−2.531	3.2
$\cos 2x$:	0	0	18.85	0	0	...	−1	0	16.2	−16
$\cos 3x$:	0	0	0	95.43	0	...	0.8437	−10.8	0	61.71
$\cos 4x$:	0	0	0	0	301.6	...	−0.8	8	−46.29	0
... :	...	...	...	...	...	...	...	...	...	...
$\sin x$:	0	0	−1	0.8437	−0.8	...	1.1781	0	0	0
$\sin 2x$:	0	2	0	−10.8	8	...	0	18.85	0	0
$\sin 3x$:	0	−2.531	16.2	0	−46.29	...	0	0	95.43	0
$\sin 4x$:	0	3.2	−16	61.71	0	...	0	0	0	301.6
... :	...	...	...	...	...	...	...	...	...	...

Here the diagonal terms are in the form $3\pi n^4/8$ and the non-zero terms in the block off-diagonals are in the form $(3n^3m^2)/(n^2 - m^2)$. Notice the matrix is largely diagonally dominant, so we can use a diagonal approximation. In the multivariate case, consider basis functions of the form

$$f_n(x_1, \ldots, x_A) = \prod_{i \in I_n} \sin n_i x_i \prod_{i \notin I_n} \cos n_i x_i$$

where $n = (n_1, \ldots, n_A, I_n)$ and index set $I_n \subseteq \{1, \ldots, A\}$. Then the diagonal terms are given by

$$S_{n,n} \propto \left(\sum_{i=1}^{A} n_i^2 \right)^2 .$$

5.4 Smoothness priors

Previous sections showed how to develop quadratic smoothness functions on different kinds of networks. This section shows how to take a penalty term such as quadratic smoothness and construct a prior that requires no setting of additional hyperparameters or decay factors [29, 41, 23]. Unfortunately the approach does not work for the penalty term of weight elimination [41], which has constant tails, because the important marginalization step of equation (4) has no solution. We can interpret this by saying that the decay factor (α below, but usually λ) in weight elimination cannot be set using Bayesian methods, but should be chosen from prior information alone.

Historically, penalty terms on weights w such as smoothness $S(w)$ have been interpreted as log-posteriors, where some constant has been added. This has been done because of the similarity of the Bayesian log-posterior with penalized least squares, ridge regression, and regularization [18, 23]. That is, we put a prior on w in the general form

$$\Pr(w \mid \alpha) \propto \alpha^{|w|/2} \cdot \exp\left(-\alpha S(w)\right), \tag{3}$$

for some constant α. Since $S(w)$ is quadratic this defines a multivariate Gaussian over weight space. The term $\alpha^{|w|/2}$ is included for normalization, and $|w|$ is the dimensionality of the weight space.

Given this prior, changing to orthogonal coordinates and then polar coordinates and simplifying yields the prior distribution on smoothness itself. Notice this is all we need since, by equation (3), points in weight space of equal smoothness are also equiprobable *a priori*:

$$\Pr(S(w) \mid \alpha) \propto S(w)^{|w|/2-1} \cdot \exp\left(-\alpha S(w)\right).$$

This prior distribution on $S(w)$ is from the Gamma family [3], so *a priori* we expect $S(w)$ to have mean $|w|/2\alpha$ and standard deviation $\sqrt{|w|}/(\sqrt{2}\alpha)$. For a large number of weights, as is usually the case in networks, $\sqrt{|w|} \ll |w|$. This says we have a very strong idea *a priori* that $S(w)$ should be near $|w|/2\alpha$. Also, given a particular value of α, the prior will ensure that the *a posteriori* smoothness found after training will most probably be quite near $|w|/2\alpha$. Of course, this is wrong. We expect smoothness $S(w)$ to be small, but *a priori* we cannot even specify the value roughly. This prior is therefore very sensitive to the "correct" setting for α and is not appropriate in general.

MacKay overcomes this poor choice of prior by making α a "hyperparameter" and setting it using a Bayesian maximum *a posteriori* approach. Better still, we can marginalize out[5] α and thus sidestep the need to determine α. Since α appears as a factor of $S(w)$ it is a scaling quantity, so a suitable prior is [3] $\Pr(\alpha) \propto 1/\alpha$. This gives the Gamma integral, so

$$\begin{aligned} \Pr(w) &= \int_0^\infty \Pr(w \mid \alpha) \Pr(\alpha)\, d\alpha \\ &\propto S(w)^{-|w|/2}. \end{aligned} \tag{4}$$

Using the same trick as before we find that this corresponds to a prior on the smoothness of

$$\Pr(S(w)) \propto S(w)^{-1}.$$

This says we *a priori* expect smoothness $S(w)$ to be small, with no restrictions on how small. Notice that this is the prior we should have used for smoothness originally since smoothness is a magnitude (a quantity where scale is important) [3]. This prior is improper (it cannot be normalized); however, this is adequate for subsequent analyses in many cases.

[5] That is, calculate $\Pr(w)$ from $\Pr(w|\alpha)$ and so remove α from the problem.

We can illustrate this prior by graphing the different kinds of regression curves that result when using one prior or another. We use one input variable x and the sine and cosine basis functions of the previous section up to $\sin 30x$ and $\cos 30x$. We ignore the constant basis function, 1, so that all regression curves will center about the origin 0. Weights are drawn according to the prior, and then the corresponding curve is drawn. The top graph in figure 1 shows the case for the smoothing prior just derived, and the bottom graph shows the case for the prior corresponding to weight decay [29],

$$\Pr(w) \propto \exp\left(-\sum_i w_i^2\right),$$

where all weights are distributed identically and independently. Notice how the curves for the smoothing prior give a range of different scales and curvatures, but more curves tend to be smoother. In the second graph, allowing weights for $\sin 30x$ to be of similar size to weights for $\sin x$ means high-order frequencies dominate the curves. Use of this second prior means that these high-frequency curves will tend to be chosen to fit small samples rather than the smoother curves in the top graph.

During gradient descent, the important contribution from the prior is

$$\frac{\partial - \log \Pr(w)}{\partial w_i} = \frac{|w|}{2} \frac{\partial \log S(w)}{\partial w_i} = \frac{|w|}{2S(w)} \frac{\partial S(w)}{\partial w_i}.$$

If we compare this with the derivative of the original prior $\Pr(w|\alpha)$ in equation (3), then it follows that α could also be set dynamically with

$$\alpha \approx \frac{|w|}{2S(w)},$$

where w is the weight value at the current point in the search.

6. Analyzing weights

The section describes how we can use the posterior analysis given in section 4.4 to assist basic tasks done during network training. In section 6.1 we describe how we derive "cost functions" for a given set of weights from the posterior formulas, and section 6.2 describes how we can evaluate one set of weights with another. The cost function allows us to find a locally optimum set of weights, and the evaluation allows us to compare the quality of one local optimum with another. We do not consider here the important computational problem of how the minimization of the cost function should be done.

6.1 Cost functions

The posterior probabilities of section 4.4 represent functions of the weights that should be maximized. Calculation is usually done in logarithms to

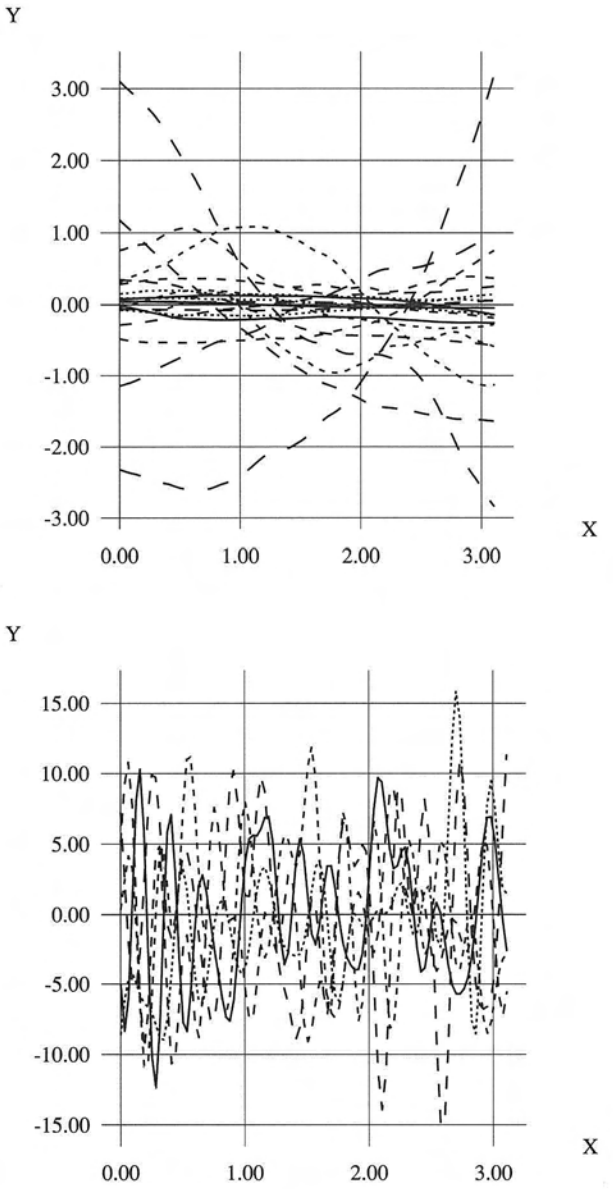

Figure 1: Typical graphs for two priors.

prevent arithmetic underflow, to turn products into sums, and because log-likelihoods are similar to the "error functions" that traditionally have been minimized in neural networks. Hence when searching for a high posterior set of weights we would like to minimize the "cost function" $-\log \Pr(w \mid \boldsymbol{x}, \boldsymbol{y})$. Notice that

$$-\log \Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) = -\log \Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) + \log \Pr(\boldsymbol{y} \mid \boldsymbol{x}),$$

and since the term $\log \Pr(\boldsymbol{y} \mid \boldsymbol{x})$ is constant in w and difficult to calculate in general, we can instead minimize the cost function

$$\text{Cost}(w) = -\log \Pr(w, \boldsymbol{y} \mid \boldsymbol{x}),$$

which can be computed directly using Lemma 4.1. Below we ignore constant terms, such as the normalizing constant for the prior on the weights $\Pr(w \mid \boldsymbol{x})$ (that is, the prior is not ignored, just its normalizing constant). This is correct when different weights w are being compared using the same prior on weights. If entirely different network structures are being compared, then the constant terms should be included.

These cost functions differ from maximum likelihood methods for network training [8, 11] in that they introduce the important prior term. The cost functions also differ because they sometimes have "nuisance" parameters eliminated. For instance, with regression σ is termed a "nuisance" parameter when trying to determine a good set of weights to do prediction. Estimates of σ can be recovered after a good set of weights are found using techniques developed later, such as equation (13). Involving "nuisance" parameters like σ in the search process is an extra complication (compare with [23]).

With maximum *a posteriori* analysis we would minimize in the regression case with Gaussian error and unknown σ:

$$\begin{aligned} &-\log \Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) \\ &\quad = \frac{N}{2} \log \sum_{i=1,\ldots,N} (y_i - o(x_i, w))^2 - \log \Pr(w \mid \boldsymbol{x}) + \text{constant}, \end{aligned} \tag{5}$$

where the logarithm of the prior is determined as before. If the value of σ is known, the cost to be minimized is

$$\frac{1}{2\sigma^2} \sum_{i=1,\ldots,N} (y_i - o(x_i, w))^2 - \log \Pr(w \mid \boldsymbol{x}) + N \log \sigma + \text{constant}. \tag{6}$$

Notice the difference between these above two cost functions. In the second case, the cost function is proportional to the mean squared error plus a penalty term; but in the first case, when σ is unknown, the logarithm of the mean squared error is taken.

A common training approach [29, 41] is to minimize a penalized error term such as

$$\frac{1}{2} \sum_{i=1,\ldots,N} (y_i - o(x_i, w))^2 + \lambda S(w).$$

This compares with the cost function of equation (6) if we multiply it by σ^2. λ can therefore be interpreted as a combination of σ^2 with prior parameters in the log-prior term $-\log \Pr(w \mid \boldsymbol{x})$. Perhaps this explains why λ is inherently hard to "set," and is often set dynamically during learning.

If in regression the value of the standard deviation (σ) is determined from the inputs as well, as $o'(x_i, r)$, the cost to be minimized is now the log-posterior of both weights w and r together, $-\log \Pr(w, r, \boldsymbol{y} \mid \boldsymbol{x})$, given by

$$\sum_{i=1,\ldots,N} \frac{1}{2o'(x_i,r)^2}(y_i - o(x_i,w))^2 + \sum_{i=1,\ldots,N} \log o'(x_i,r)$$
$$-\log \Pr(r, w \mid \boldsymbol{x}) + \text{constant}.$$

Minimization has to be done for r and w concurrently. A prior for r was given in section 5.1.

In the classification case, the overall cost is given by

$$-\log \Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) =$$
$$-\sum_{i=1,\ldots,N} \log o_{y_i}(x_i, w) - \log \Pr(w \mid \boldsymbol{x}) + \text{constant}. \tag{7}$$

If the output variable is binary and represented as 1 for true and 0 otherwise, then the first sum expands to

$$-\sum_{i=1,\ldots,N} (y_i \log o_1(x_i, w) + (1 - y_i) \log(1 - o_1(x_i, w))) .$$

Notice this is the familiar cross entropy error for the sample.

Notice that the formulation given here seems to imply that learning should be done in batch (or epoch) mode because the cost functions are expressed in terms of the full training sample. This is not the case. The sums above can readily be approximated on a sub-sample as is common in back-propagation implementations, although, to ensure accuracy, batch learning should be done on the full sample when near a local minimum of the cost function.

6.2 Weight evaluation

To obtain a measure of the quality of each local maximum *a posteriori* estimate $\widehat{w}$ found during search, an estimate of the local area under the posterior around $\widehat{w}$ is usually done. The actual posterior value itself is not the best measure of quality because some peaks may be thinner than others so they contain much less of the posterior probability in their vicinity. For instance, consider an idealized learning problem where the scalar parameter a is being learned. Suppose a has posterior given in figure 2. Notice the left peak is higher but much thinner. The expected value of a, $\mathbf{E}_a(a)$, and other functions would therefore come more from the fatter peak on the right. This becomes more pronounced in higher dimensions, or when comparing networks

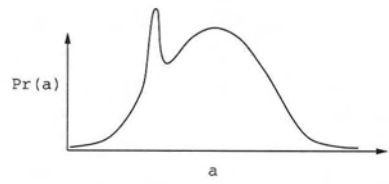

Figure 2: Posterior for a.

of different dimension. This local area estimate of quality lets different local maximum *a posteriori* estimates be compared on an equal footing, for instance networks with different numbers of layers or connections. This local area estimate and successively coarser approximations to it correspond to the various encoding measures such as MML [40] and MDL [33], discussed in the next section.

To make this estimate, we approximate the log-posterior at the local maximum $\widehat{w}$ by a quadratic function of the weights, giving

$$\begin{aligned} &\Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \\ &\quad \approx \Pr(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \exp\left(-\frac{1}{2}(w-\widehat{w})^T I(\widehat{w})(w-\widehat{w}) + O((w-\widehat{w})^3)\right), \end{aligned} \tag{8}$$

where $I(w)$ is a second derivative of the cost function, such as the log-posterior equations (5) or (7),

$$\begin{aligned} I(w) &= -\frac{d^2 \log \Pr(w \mid \boldsymbol{x}, \boldsymbol{y})}{dw\, dw} = -\frac{d^2 \log \Pr(w, \boldsymbol{y} \mid \boldsymbol{x})}{dw\, dw} \\ &= \frac{d^2 \mathrm{Cost}(w)}{dw\, dw}. \end{aligned}$$

Notice the differentials are dw rather than ∂w because the derivative represents the full matrix of second derivatives with respect to the different weights $w_{n,m}$. Because the constants in equations (5) and (7) are independent of w, they can be ignored when evaluating the derivative.

Roughly, this approximation is valid for large sample size N because the posterior is found by normalizing for w a formula of the form $g(w)^N$ (see, for instance, [3, page 224]). Generally, the sample size N should be at least some factor of the number of weights in the network. In some simple non-network models—such as distributions from the exponential family [3], class probability trees, and Bayesian networks on discrete variables [6]—the posterior can be dealt with exactly so the approximation using $I(\widehat{w})$ is not necessary.

We shall refer to the second derivative evaluated at the local maximum, $I(\widehat{w})$, as the *information matrix*. When using a uniform prior, $I(\widehat{w})$ is referred to as the observed Fisher information matrix, and its determinant is referred to as the Fisher information. We later develop an approximation, given in

equation (10), called the expected Fisher information matrix. These play a central part in statistical theory [1, 3].

The local posterior area for w near $\widehat{w}$ is found by integrating out w using the multivariate Gaussian approximation, and is given by

$$\Pr(\text{near } \widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \approx \Pr(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \frac{(2\pi)^{|w|/2}}{\det(I(\widehat{w}))^{1/2}},$$

where $|w|$ is the dimensionality of the variable set w and $\det(\cdot)$ is the matrix determinant.

The desired quality measure is now given by the negative logarithm of the local area (and adding in the constant $-\log \Pr(\boldsymbol{y} \mid \boldsymbol{x})$)

$$\begin{aligned} \text{Eval}(\widehat{w}) &= -\log \Pr(\text{near } \widehat{w}, \boldsymbol{y} \mid \boldsymbol{x}) \\ &= -\log \Pr(\boldsymbol{y} \mid \widehat{w}, \boldsymbol{x}) - \log \Pr(\widehat{w} \mid \boldsymbol{x}) \\ &\quad - \frac{|w|}{2} \log(2\pi) + \frac{1}{2} \log \det(I(\widehat{w})), \end{aligned} \tag{9}$$

which wants to be minimized. Notice the first and second terms together give $\text{Cost}(\widehat{w})$. We separate them out here because they correspond to the likelihood and prior components, respectively.

The determinant $\det(I(\widehat{w}))$ could be approximated by looking only at the diagonals or block diagonals (block corresponds to one node that is square in the number of weights for the node). Individual second derivatives can be determined using a number of methods [9]. Fortunately, this only has to be done once for each local minimum $\widehat{w}$ found, instead of repeatedly during search, so a fast approximation is not essential.

Several subsequent uses of the information matrix $I(\widehat{w})$ require computing the inverse. This will not exist if the matrix has zero eigenvectors. Zero eigenvectors themselves will usually exist only if the weight space is redundant, in the sense that there are ways of altering weight values without altering the input-output behavior of the network. For instance, the parameterization of the sigmoid activation function given by

$$\frac{1}{1 + \exp\left(a\left(\sum_{i=1,\ldots,l} w_i u_i\right) + b\right)}$$

is redundant because the a parameter can be absorbed into the weights w_i. The activation function as given is specified by $l + 2$ parameters, but only requires $l + 1$ parameters. The logistic networks of section 3.2 were designed to remove redundancy between weights at different nodes.

6.3 Minimum encoding methods

Quality measures similar to $\text{Eval}(\widehat{w})$ can be derived using various quantization, encoding, and approximation methods. We discuss these briefly here to draw the strong connections between $\text{Eval}(\widehat{w})$ and the often discussed encoding measures given in [40, 2, 33]. We view these methods as potentially useful approximations, particularly the MML measure given second, but do not consider their application here.

Rissanen's MDL

Rissanen develops his minimum description language (MDL) measure [33] as an upper bound on $-\log \Pr(w, \boldsymbol{y} \mid \boldsymbol{x})$. For exposition purposes, we develop a related bound here. $I(\widehat{w})$ is a positive indefinite symmetric matrix (since $\widehat{w}$ is a local minimum), so[6]

$$\det I(\widehat{w}) \leq \left(\frac{\operatorname{trace} I(\widehat{w})}{|w|} \right)^{|w|},$$

where trace A is the sum of the diagonal entries in the matrix A. Now, for large N, trace $I(\widehat{w})/N$ is a mean and thus is approximately Gaussian, with mean $O(|w|)$ and standard deviation $O(|w|/\sqrt{N})$. Therefore, with probability approaching 1 as N gets larger,

$$\frac{1}{2} \log \det I(\widehat{w}) = \frac{|w|}{2} \log N + O(|w| \log |w|).$$

We get that

$$\begin{aligned} \text{Eval}(\widehat{w}) \;=\; & -\log \Pr(\boldsymbol{y} \mid \widehat{w}, \boldsymbol{x}) - \log \Pr(\widehat{w} \mid \boldsymbol{x}) + \frac{|w|}{2} \log N \\ & + O(|w| \log |w|). \end{aligned}$$

This is an approximation developed by Schwarz and others [2]. The corresponding MDL form ignores the term $\log \Pr(\widehat{w} \mid \boldsymbol{x})$, which is probably $O(|w|)$ anyway, and has an additional term $(|w|/2) \log |w|$. In our case, $O(|w| \log |w|)$ is quite large because we are dealing with networks with many weights. It is not unusual in practice for the number of weights to be a similar order of magnitude to the sample size N, so ignoring terms of order $O(|w| \log |w|)$, as this approximation does, is probably unwise. We conclude that this approximation is informative but perhaps too crude.

Wallace and Freeman's and Barron and Cover's measures

Wallace and Freeman [40] and Barron and Cover [2] interpret a form like $\text{Eval}(\widehat{w})$ as the cost of encoding $\boldsymbol{y}$ and $\widehat{w}$ given $\boldsymbol{x}$. Wallace et al. refer to this as the minimum message length (MML). Of course, $\boldsymbol{y}$ and $\widehat{w}$ can only be encoded to finite precision (in classification, $\boldsymbol{y}$ is finite already). The precision of $\boldsymbol{y}$ is implicit in the supplied data, so if the volume of the precision for the data vector $\boldsymbol{y}$ is δ then the cost of encoding $\boldsymbol{y}$ given $\widehat{w}$ and $\boldsymbol{x}$ is

$$-\log \Pr(\boldsymbol{y} \mid \widehat{w}, \boldsymbol{x}) - \log \delta.$$

Because $\boldsymbol{y}$ is encoded given $\widehat{w}$, we must encode $\widehat{w}$ without knowledge of $\boldsymbol{y}$. The second, third and fourth terms of equation (9) are not an appropriate code length, however, because $I(\widehat{w})$ is dependent on $\boldsymbol{y}$. The third and

[6] The determinant is a product of eigenvalues whereas the trace is a sum of eigenvalues, which in this case are all non-negative. The inequality follows by maximizing $\prod_i \lambda_i$ given a fixed value for $\sum_i \lambda_i$.

fourth terms of equation (9) essentially correspond to the "precision" of $\widehat{w}$. To see this, suppose the continuous space for w is quantified. Then the cell in the neighborhood of $\widehat{w}$ has prior probability given approximately by $\Pr(\widehat{w})\text{Area}(\widehat{w})$, where $\text{Area}(\widehat{w})$ is the area of the cell. Therefore $-\log \text{Area}(\widehat{w})$ is the precision to which $\widehat{w}$ is specified, and this matches with the third and fourth terms of equation (9). To interpret this precision component, notice that if the weights w happened to be *a posteriori* independent (an unlikely event), then there would be no off-diagonal entries in the information matrix and

$$-\frac{|w|}{2}\log(2\pi) + \frac{1}{2}\log\det(I(\widehat{w})) \approx -\sum_{n,m}\log\left(2.51\cdot\left(\frac{\partial^2 \text{Cost}(w)}{\partial w_{n,m}^2}\right)^{-1/2}\right).$$

We show later in section 7.2 that the inverse square root of the second derivative with respect to $w_{n,m}$ is approximately the posterior standard deviation for the weight $w_{n,m}$. The recommended precision for encoding each weight is therefore approximately 2.5 times the posterior standard deviation for the weight.

Wallace et al. and Barron et al. make the fourth term in equation (9) independent of y using an involved quantification argument. We can interpret this by noting

$$\frac{1}{2}\log\det(I(\widehat{w})) = \frac{|w|}{2}\log N + \frac{1}{2}\log\det\left(\frac{1}{N}I(\widehat{w})\right).$$

The term $(1/N)I(\widehat{w})$ is now in the form of an average since, in general, $I(\widehat{w})$ is a sum across the N points in the sample plus a fixed prior term. We now replace this "average" information matrix $(1/N)I(\widehat{w})$ by what is called the expected Fisher information matrix, $\overline{I}(\widehat{w})$, to get

$$\begin{aligned}\text{Coding}(\boldsymbol{y},\widehat{w}) \;=\; & -\log\Pr(\boldsymbol{y}\mid\widehat{w},\boldsymbol{x}) - \log\delta - \log\Pr(\widehat{w}\mid\boldsymbol{x}) \\ & -\frac{|w|}{2}\log(2\pi) - \frac{|w|}{2} + \frac{|w|}{2}\log N + \frac{1}{2}\log\det(\overline{I}(\widehat{w})).\end{aligned}$$

The expected Fisher information matrix is the expected value of the information matrix for a single pattern, given by[7]

$$\overline{I}(w) = \mathbf{E}_{x,y|w}\left(-\frac{d^2\log\Pr(x,y\mid w)}{dw\,dw}\right),$$

where (x,y) in this formula denotes a single pattern rather than the training sample $(\boldsymbol{x},\boldsymbol{y})$. This simplifies using properties of the expected Fisher

[7]Wallace et al. and Barron et al. develop their framework to deal with likelihoods of the form $p(x|w)$ for data x and parameters w, whereas we are considering conditional likelihoods such as $p(y|w,x)$. Here we give our interpretation of their methods in this different context.

information matrix [37, page 35] to

$$\begin{aligned}\overline{I}(w) &= \mathbf{E}_{x,y|w}\left(\frac{d\log\Pr(x,y\mid w)}{dw}\frac{d\log\Pr(x,y\mid w)}{dw}^T\right)\\ &= \mathbf{E}_x\left(\mathbf{E}_{y|w,x}\left(\frac{d\log\Pr(x,y\mid w)}{dw}\frac{d\log\Pr(x,y\mid w)}{dw}^T\right)\right)\\ &\approx \frac{1}{N}\sum_{i=1,\ldots,N}\mathbf{E}_{y|w,x_i}\left(\frac{d\log\Pr(x,y\mid w)}{dw}\frac{d\log\Pr(x,y\mid w)}{dw}^T\right). \quad (10)\end{aligned}$$

The term $d\log\Pr(x,y\mid w)/dw$ denotes a column vector of first derivatives. Notice that the last formula is an estimate of $(1/N)I(\widehat{w})$ involving knowledge of the sample inputs $\boldsymbol{x}$ but not the sample outputs $\boldsymbol{y}$. It is therefore a valid inclusion in the code length of w.

For instance, in this case of classification, the approximate expected Fisher information becomes

$$\overline{I}(w) \approx \frac{1}{N}\sum_{i=1,\ldots,N}\sum_{j=1,\ldots,C}\frac{1}{o_j(w,x_i)}\frac{do_j(w,x_i)}{dw}\frac{do_j(w,x_i)}{dw}^T.$$

Here j runs over output classes and i runs over patterns. In the case of regression with Gaussian error, the approximate expected Fisher information can be simplified to

$$\overline{I}(w) \approx \frac{1}{N\sigma^2}\sum_{i=1,\ldots,N}\frac{do(w,x_i)}{dw}\frac{do(w,x_i)}{dw}^T.$$

The advantage of this formula over the original information matrix $I(w)$ is that it can be computed from first derivative information, from the results of back-propagation, without the need for second derivatives.

7. Applications to network training

This section describes some applications of the tools developed above. This illustrates how the tools can be applied to address more pragmatic problems in the use of feed-forward networks.

7.1 Prediction and generalization error

Once search has located some local minimum $\widehat{w}$ of the cost function, it can be used in inference on new patterns. A first approximation is to say that w must now be $\widehat{w}$ and to make inference about y and σ^2, and so forth, based entirely on this local minimum $\widehat{w}$ using $y = o(x', \widehat{w})$, and so on. This is naive because we cannot be sure that the "true" or "optimum" w is $\widehat{w}$. For instance, with a sample twice the size we might change our estimate of w quite dramatically. A full Bayesian approach takes into account the many other values of w near $\widehat{w}$ or even some other local minimum, because other

values just might happen to be the "true" weights. Only when we have a really large sample (for instance, when uniform convergence bounds tell us the sample is large enough [14]) can we be sure that $\widehat{w}$ is a sufficiently good estimate so that no other minima need be considered.

In principle we want to calculate for a new pattern x' the posterior expected value of the network output and of the variance, $\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y}}\left(o(x',w)\right)$ and $\mathbf{E}_{w,\sigma|\boldsymbol{x},\boldsymbol{y}}\left(\sigma^2\right)$, and so forth. These values give us the posterior average of the quantities $o(x',w)$ and σ^2 that give better predictions for these quantities on average than those calculated at the local minimum $w = \widehat{w}$. Unfortunately, we cannot calculate the full posterior of w, $\Pr(w \mid \boldsymbol{x},\boldsymbol{y})$, in reasonable time, and only have the Gaussian approximation to the posterior described above.

This section describes how these predictions can be approximated using the posterior approximations available. These approximations all make use of the second derivatives of the cost function and the approximate weight variance discussed above. While it is hard to predict how accurate these approximations will be in practice, their form at least gives some idea of the interaction between weight variances and the behavior of the network output for w in the vicinity of $\widehat{w}$.

Predictions are given for three quantities.

$\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y}}\left(o(x',w)\right)$: the posterior expected output of the network given input x'. In classification this can be used to get an estimate of the posterior probability for different predictions y given x', and can be used to predict generalization error for classification. In regression this gives an estimate of the posterior mean of y.

$\mathbf{v}_{w|\boldsymbol{x},\boldsymbol{y}}\left(o(x',w)\right)$: the posterior variance of the network output. This is a measure of how uncertain we are about our prediction above for the network output. Notice that as the sample size gets large this uncertainty will shrink to zero. We give two versions of this uncertainty depending on which approximation to $\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y}}\left(o(x',w)\right)$ is used.

$\mathbf{E}_{w,\sigma|\boldsymbol{x},\boldsymbol{y}}\left(\sigma^2\right)$: the posterior expected regression variance. This predicts generalization error for regression with Gaussian error model and unknown standard deviation.

The forms given for generalization error, equations (11) and (13), are Bayesian versions of those in [25].

Predictions from the single "best" local minimum

The first set of predictions we give handle the case in which we have found a single "best" local minimum and make an approximation based entirely on this without considering the effects of other local minima. To indicate that an assumption is being made, we condition probabilities with the form "*near* $\widehat{w}$," which indicates we are assuming the "true" w is near the local minimum $\widehat{w}$.

In what follows, we use the notation $\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}(U(w))$ to denote the expectation of the quantity $U(w)$ when averaged according to the approximate Gaussian posterior for w in the vicinity of $\widehat{w}$. That is,

$$\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}(U(w)) = \int_w U(w)\,\frac{\det(I(\widehat{w}))^{1/2}}{(2\pi)^{|w|/2}}\exp\left(-\frac{1}{2}(w-\widehat{w})^T I(\widehat{w})(w-\widehat{w})\right)dw.$$

We evaluate these integrals by using standard moments of the multivariate Gaussian [3], and by approximating $U(w)$ in the vicinity of $\widehat{w}$ using the second-order Taylor expansion, which in vector notation is as follows:

$$U(w) \approx U(\widehat{w}) + (w-\widehat{w})^T \cdot \left.\frac{dU(w)}{dw}\right|_{w=\widehat{w}} + \frac{1}{2}\,(w-\widehat{w})^T \cdot \left.\frac{d^2U(w)}{dw\,dw}\right|_{w=\widehat{w}} \cdot (w-\widehat{w}).$$

Again, $do(x',w)/dw$ denotes a vector of first derivatives, and the second derivative denotes a matrix. This gives

$$\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}(U(w)) \approx U(\widehat{w}) + \frac{1}{2}\text{trace}\left(I(\widehat{w})^{-1}\cdot\left.\frac{d^2U(w)}{dw\,dw}\right|_{w=\widehat{w}}\right),$$

where, again, trace A denotes the sum of the diagonal entries of the matrix A. If any of the approximate weight variances are large, then good approximations will require accurate estimates of $U(w)$ for w far from $\widehat{w}$. In this case, the Taylor expansion will be a poor approximation and our approximate predictions will be poor. Notice that we could also use the diagonal approximation for the matrices $I(\widehat{w})$ and the second derivatives of $U(w)$; however, the estimates may then become very poor and be useful only to indicate where potential problems lie. These and other approximations are discussed in [30].

$\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}(o(x',w))$: the posterior expected output of the network given input x', averaged over weights in the vicinity of $\widehat{w}$. This is approximated by

$$o(x',\widehat{w}) + \frac{1}{2}\text{trace}\left(I(\widehat{w})^{-1}\left.\frac{d^2o(x',w)}{dw\,dw}\right|_{w=\widehat{w}}\right).$$

This modifies the output $o(x',\widehat{w})$ to account for how the output $o(x',w)$ varies in the neighborhood of $\widehat{w}$, tempered by the posterior variance-covariance of w. In the case of classification, this gives an estimate of out-of-sample accuracy for the network given by

$$\mathbf{E}_{w,x|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}\left(\max_y o_y(x,w)\right) \approx \frac{1}{N}\sum_{i=1}^{N}\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}\left(\max_y o_y(x_i,w)\right)$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} \max_y o_y(x_i, \widehat{w}) + \frac{1}{2N}\text{trace}\left(I(\widehat{w})^{-1} \left. \frac{d^2 \sum_{i=1}^{N} \max_y o_y(x_i, w)}{dw\, dw} \right|_{w=\widehat{w}} \right). \quad (11)$$

$\mathbf{E}_{w|x,y,\text{near}\,\widehat{w}}((o(x', w) - o(x', \widehat{w}))^2)$: when using $o(x', \widehat{w})$ to estimate $o(x', w)$, the expected variance of the output in the neighborhood of $\widehat{w}$. This gives a measure of variance for our posterior uncertainty when using $o(x', \widehat{w})$ to estimate $o(x', w)$. For instance, in the regression case $o(x', \widehat{w})$ corresponds to an estimate of the mean value of y conditioned on x'. The variance of our estimate for the network output is approximated by

$$\left. \frac{do(x', w)}{dw} \right|^{T}_{w=\widehat{w}} \cdot I(\widehat{w})^{-1} \cdot \left. \frac{do(x', w)}{dw} \right|_{w=\widehat{w}} \quad (12)$$

$\mathbf{v}_{w|x,y,\text{near}\,\widehat{w}}(o(x', w))$: the variance of the estimate $\mathbf{E}_{w|x,y,\text{near}\,\widehat{w}}(o(x', w))$ in the neighborhood of $\widehat{w}$. This is approximated by

$$\left. \frac{do(x', w)}{dw} \right|^{T}_{w=\widehat{w}} \cdot I(\widehat{w})^{-1} \cdot \left. \frac{do(x', w)}{dw} \right|_{w=\widehat{w}} - \frac{1}{4}\text{trace}\left(I(\widehat{w})^{-1} \cdot \left. \frac{d^2 o(x', w)}{dw\, dw} \right|_{w=\widehat{w}} \right).$$

Notice that this is lower than the previous variance estimate of equation (12) due to the second term, which corrects for the fact that we are now using a better estimate for $o(x', w)$.

$\mathbf{E}_{w,\sigma|x,y,\text{near}\,\widehat{w}}(\sigma^2)$: the posterior expected value of σ^2 in the regression case with Gaussian error and unknown σ. This is sometimes termed the generalization error. This is approximated using equation (2) and simplified to

$$\frac{N}{N-2}\, s^2\Big|_{w=\widehat{w}} + \frac{N}{2(N-2)}\text{trace}\left(I(\widehat{w})^{-1} \cdot \left. \frac{d^2 s^2}{dw\, dw} \right|_{w=\widehat{w}} \right). \quad (13)$$

Notice that the first term is the variance estimate assuming the local minimum $\widehat{w}$ is correct, and the second term increases this because the posterior uncertainty in the weights w induces additional variance.

Predictions from multiple local minima

If multiple local minima $\widehat{w}$ of similar quality have been found during restarts with different initializations of w, then the expected values above can be averaged over the different local minima $\widehat{w}$ to produce a pooled estimate. This same "multiple models" approach can produce significant improvement in

out-of-sample prediction when learning classification trees [6]. It corresponds to a Monte Carlo estimation of the full posterior for w, rather than just an approximation about $\widehat{w}$. Suppose a small set $\widehat{W}$ of local minima are found, and for each we have a proportion $p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y})$ to be used in the Monte Carlo estimation below, where $\sum_{\widehat{w}\in\widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) = 1$. For instance, these proportions could be constructed proportional to $\Pr(\text{near}\, \widehat{w}, \boldsymbol{y} \mid \boldsymbol{x})$, although equal proportions are sometimes used in Monte Carlo estimation. The estimate of the posterior for the weights w is now a weighted sum of the Gaussians in the neighborhood of each $\widehat{w}$:

$$\Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \approx \sum_{\widehat{w}\in\widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \Pr(w \mid \boldsymbol{x}, \boldsymbol{y}, \text{near}\, \widehat{w}),$$

where the Gaussian approximation described at the beginning of this section is used for each $\Pr(w \mid \boldsymbol{x}, \boldsymbol{y}, \text{near}\, \widehat{w})$.

This leads to the following corrections for the previous predictions. To find the expected output of the network for different weights, we pool the expected outputs for the individual local minima in $\widehat{W}$:

$$\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y}}\left(o(x', w)\right) \approx \sum_{\widehat{w}\in\widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near}\,\widehat{w}}\left(o(x', w)\right).$$

To find the variance of this value, we pool the variances for individual local minima together with a measure of how much the output varies from minimum to minimum:

$$\begin{aligned}\mathbf{v}_{w|\boldsymbol{x},\boldsymbol{y}}\left(o(x', w)\right) \approx & \sum_{\widehat{w}\in\widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \mathbf{v}_{w|\boldsymbol{x},\boldsymbol{y},\text{near}\,\widehat{w}}\left(o(x', w)\right) \\ & + \mathbf{v}_{\widehat{w}|\boldsymbol{x},\boldsymbol{y}}\left(\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near}\,\widehat{w}}\left(o(x', w)\right)\right).\end{aligned}$$

7.2 Weight variance and network pruning

The Gaussian approximation to the likelihood near a local maximum *a posteriori* also gives a statistically sound method for the pruning of networks [20]. The idea here is that we might significantly reduce the size of a network but still retain a network that is *a posteriori* nearly as good as the original network. The pruning is not done because we expect it to improve generalization error, because weights have already been reduced (most likely) due to the effect of the prior in the cost function. We prune merely to reduce the size of the network.

The matrix inverse of the information matrix $I(\widehat{w})$, namely $I(\widehat{w})^{-1}$, is an approximate posterior variance-covariance matrix for w [3, page 224]. It tells us how much we currently believe the "true" weights might vary about our estimate $\widehat{w}$. That is, in the notation of the previous section,

$$\mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near}\,\widehat{w}}\left((w - \widehat{w})(w - \widehat{w})^T\right) = I(\widehat{w})^{-1}.$$

This follows from the functional form of equation (8). This means, for instance, that the posterior variance of the weight $w_{n,m}$ in the neighborhood of $\widehat{w}$,

$$\mathbf{v}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}(w_{n,m}) = \mathbf{E}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}((w_{n,m} - \widehat{w}_{n,m})(w_{n,m} - \widehat{w}_{n,m}))\,,$$

is given by the diagonal entry for $w_{n,m}$ in the matrix $I(\widehat{w})^{-1}$. If this variance is large, then the weight $w_{n,m}$ is poorly determined by the data.

We can use the posterior variance-covariance matrix for w to test if a weight is significantly different from zero. Those that are not can be set to zero. Notice we are not intending this be used as a method of handling "overfitting," or as some substitute to cross-validation or weight elimination [41]. We are therefore not interested in just decreasing some of the weights, but in removing them entirely from the network (zeroing them) to produce a smaller network. There is a very useful side effect of this technique: it will tend to remove weights with high posterior variance (i.e., we are unable to locate even an approximate value for the "best" value of certain weights because the data is insufficient in quantity) since we cannot be sure that these are sufficiently different from zero. These high variance weights contribute to ill-conditioning[8] of the matrix $I(\widehat{w})$, thus making it difficult to compute the posterior variance-covariance matrix for w, namely $I(\widehat{w})^{-1}$.

A simple test is as follows. We use the diagonal approximation to the information matrix $I(\widehat{w})$ to estimate the posterior variance of the weight $w_{n,m}$ in the neighborhood of $\widehat{w}$,

$$\mathbf{v}_{w|\boldsymbol{x},\boldsymbol{y}}(w_{n,m}) \approx \mathbf{v}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}(w_{n,m}) \approx -1 \Big/ \left.\frac{\partial \log \Pr(w \mid \boldsymbol{x},\boldsymbol{y})}{\partial w_{n,m}\partial w_{n,m}}\right|_{w=\widehat{w}}.$$

Because $\widehat{w}$ is a local maximum for the posterior, the second derivative must be non-positive so the variance estimate is non-negative. A zero variance estimate means some other estimate will have to be made. A large variance means the sample gave us little idea as to what value the weight should be. So if the magnitude of a weight is within a standard deviation, the weight can safely be set to zero [19].

A more thorough test follows. The quadratic term $(w - \widehat{w})^T I(\widehat{w})(w - \widehat{w})$ appears in the multivariate Gaussian approximation for the posterior for the weights w, equation (8). For large N, this quadratic term is approximately chi-squared with $|w|$ degrees of freedom. The $X\%$ confidence region (for instance, $X = 99.0$) for the weights w is the set of weights within the upper $X\%$ point of the $\chi^2_{|w|}$ distribution, $\chi^2_{|w|,X}$, giving

$$w : (w - \widehat{w})^T I(\widehat{w})(w - \widehat{w}) \leq \chi^2_{|w|,X}.$$

For instance, the upper 99% point for 30 weights is given from standard chi-squared tables as 50.89. We therefore proceed as follows. We repeatedly

[8] An ill-conditioned matrix has some small eigenvalues (so its inverse has some large eigenvalues), and it is difficult to compute the matrix inverse accurately.

set weights to zero while the resultant weights remain in the $X\%$ confidence region. Weights could be set to zero in an order leading to the least increase in the quadratic term. We might also wish to zero groups of weights to remove entire nodes from the network. Since this procedure follows from a quadratic expansion around the local minimum $\widehat{w}$ of the cost function, and since our objective is to test if the network with zeroed weights is significantly different from the network at the local minimum, it is not necessary—and indeed it is incorrect—to recompute $I(\widehat{w})$ as we zero weights. Notice also that this simple test can be done directly from $I(\widehat{w})$ without having the expensive calculation of a determinant or inverse, so it could also be performed at stages during the back-propagation cycle.

7.3 Adjustments for missing values

In many practical problems, training patterns are supplied with some values of variables missing, unknown, or undetermined. Of course, if a pattern has its output value missing, the pattern can just be ignored. If an input variable has a value missing, however, we need to deal with it. Approximate methods for dealing with this kind of problem exist when learning tree classifiers [31]:

- One can ignore the training pattern with missing values. Stochastic learning can proceed initially using only the patterns without missing values.
- One can replace the missing value by some simple estimate such as the modal or mean value. For instance, if the binary input variable x_i is either 0 (false) or 1 (true) and if it is unknown, set it to be about 0.5.
- One can complete the pattern in various ways (to fill in missing values) and treat each of these completions as a partial pattern (so the sum of fractional patterns adds to 1). We explain this idea in the context of feed-forward networks below. Tree learning algorithms do this completion in an efficient demand-driven manner (i.e., they complete a missing value only when the value is asked for by the algorithm) that unfortunately is not available with feed-forward networks.

The second approach works quite well and is the simplest to implement; however, the third approach gives the best performance in terms of out-of-sample prediction.

In this section we develop a modified algorithm for handling missing values related to the third approach above. It also approximates the Bayesian normative approach for handling missing values because we derive the approach here using standard laws of probability. The method is another example of the use of mixture models [26] used for adaptive mixtures of local experts, and the equations derived have a related form. We do the analysis for the regression problem with Gaussian error and unknown standard deviation. The other cases are similar. We assume we have a model of some kind, denoted F, that gives a rough prediction of the missing values from

the known values. The model F might correspond to simple linear models or some other form readily calculated from the data.

The training sample consists of input vectors $x_1, \ldots, x_N$ and corresponding output values $y_1, \ldots, y_N$. But in this case some of the input vectors x_i have missing values. Let unknown(x_i) denote the set of variables in the ith input vector x_i that have missing values. For instance, given the ith pattern $x_i = (1, 0, ?, 0, ?, ?)$, where ? denotes a missing value, then the unknown(x_i) are the third, fifth, and sixth variables. Given an assignment to these, $x' \in$ unknown(x_i), let $x_i x'$ denote one possible completion for the input vector x_i. Then the likelihood of the pattern is now given by

$$l(y_i \mid x_i, w, \sigma, F) = \mathbf{E}_{x' \in \text{unknown}(x_i) \mid x_i, F} \left(l(y \mid x_i x', w, \sigma) \right),$$

where the expectation is done using the model F for predicting the unknown values. We can approximate this stochastically by selecting a few possible completions compl(x_i) and weighting them to give

$$l(y_i \mid x_i, w, \sigma, F) \approx \sum_{x' \in \text{compl}(x_i)} p(x' \mid x_i, F) l(y \mid x_i x', w, \sigma), \tag{14}$$

where the weights $p(x' \mid x_i, F)$ indicate the likelihood of the different completions based on the model F. This formula can now be evaluated because each likelihood $l(y \mid x_i x', w, \sigma)$ can be determined as in section 4.1. For instance, in the example above, we may select three completions according to the model for predicting missing values as

$$\text{compl}(x_i) = \{ (1,0,0,0,0,1), (1,0,0,0,1,1), (1,0,1,0,1,0) \}$$

and give them equal weighting of 1/3 each. Completions and weightings can be determined once before the network training begins. Notice that the crudest estimate is to use only the single modal or mean value for the missing values with a weight of 1, as is often done.

Network training now consists of using these modified likelihoods. In the regression case we are considering, the standard deviation can no longer be marginalized out (that is, no longer integrated out using the technique for Lemma 4.1) so the cost function is now $-\log \Pr(w, \sigma, \boldsymbol{y} | F, \boldsymbol{x})$. Suppose the patterns in *missing* have missing values and those in *complete* have all values given, then the cost is:

$$\sum_{i \in \text{complete}} \frac{1}{2\sigma^2} (y_i - o(x_i, w))^2 + (N+1) \log \sigma - \log \Pr(w) + \text{constant}$$
$$- \sum_{i \in \text{missing}} \log \left(\sum_{x' \in \text{compl}(x_i)} p(x' | x_i, F) \exp\left(\frac{-1}{2\sigma^2} (y_i - o(x_i x', w))^2 \right) \right).$$

While this looks more involved than the original cost function of equation (5), its derivatives are not that different:

$$\frac{\partial \mathrm{Cost}(w)}{\partial w_{n,m}} = -\frac{\partial \log \Pr(w)}{\partial w_{n,m}} - \frac{1}{2\sigma^2} \sum_{i \in \text{complete}} (y_i - o(x_i, w)) \frac{\partial o(x_i, w)}{\partial w_{n,m}} \tag{15}$$

$$- \frac{1}{2\sigma^2} \sum_{i \in \text{missing}} \sum_{x' \in \text{compl}(x_i)} w(x' \mid x_i, F, \sigma) \cdot (y_i - o(x_i x', w)) \frac{\partial o(x_i x', w)}{\partial w_{n,m}},$$

where the proportions $w(x'|x_i, F, \sigma)$ are calculated as

$$w(x'|x_i, F, \sigma) \equiv \frac{p(x' \mid x_i, F) \exp\left((-1/2\sigma^2)(y_i - o(x_i x', w))^2\right)}{\sum_{x' \in \text{compl}(x_i)} p(x' \mid x_i, F) \exp\left((-1/2\sigma^2)(y_i - o(x_i x', w))^2\right)}.$$

Compare the two sums in equation (16). In the second, the term for the completion $x_i x'$ appears weighted by the proportion $w(x'|x_i, F, \sigma)$, but in the term for the first sum each (already complete) pattern x_i is effectively weighted by 1. For this reason we say each completion participates as a partial pattern. Notice also that the proportions $w(x'|x_i, F, \sigma)$ essentially pick out the completion x' that gives the lowest squared error. Since the derivative of $o(x_i x', w)$ with respect to the completion x' is available after the back-propagation step, we could dynamically alter the completions x' during each learning step so the mean-square error for each is lowered.

The derivative of Cost with respect to σ is in a similar form, but this time a fixed-point equation (σ also occurs on the right-hand side) for σ can be derived:

$$\sigma = \frac{1}{N+1} \left(\sum_{i \in \text{complete}} (y_i - o(x_i, w))^2 + \sum_{i \in \text{missing}} \sum_{x' \in \text{compl}(x_i)} w(x' \mid x_i, F, \sigma) \cdot (y_i - o(x_i x', w))^2 \right).$$

The standard deviation σ can be updated iteratively along with the weights w in each cycle. Again we see the "partial patterns" alter the usual form for the variance. This is a common result in mixture models.

A disadvantage of this method is that each pattern with missing values now produces a number of completed patterns, each of which must be run through the network. For instance, if every pattern has missing values and c different completions are used for each, then computation is increased by a factor of c. One way around this would be to have the network first learn for patterns with no missing values, and then to fine tune by including the remaining patterns. Of course, the extra computation should give improved performance as c is increased due to the normative justification of the approach. Also, the completed patterns can be altered dynamically during training with little extra overhead to produce completions with lower mean-square error.

8. Conclusion

This paper has covered Bayesian theory relevant to the problem of training feed-forward connectionist networks. We now sketch out how this might be put together in practice, and conclude with a brief discussion of research issues.

For network training, the principle steps are as follows.

(1) Choose an appropriate network structure and size based on prior knowledge about the application (see, for instance, the discussion in [21] regarding choice of network), and select a prior on the weights. Notice that a small number of different structures could be selected, and the method will then select the best.

(2) As discussed in section 7.3, construct completions and their proportions $p(x'|x_i, F)$ for each training pattern with missing values. If stochastic training is used instead of epoch training, each set of completions should always be run through the network together so that the appropriate term in equation (16) can be calculated.

(3) Train to a local minimum $\widehat{w}$ as per the usual, but incorporate the adjustments for missing values described in section 7.3. Appropriate cost functions are given in section 6.1. In principle, the weight variances, weight evaluations, and predictions apply only if the $\widehat{w}$ found is a true local minimum of the cost function, so epoch training might have to be used in the last few cycles to ensure this.

(4) If possible, use the weight pruning strategy of section 7.2 to force some weights to zero, and continue training the network with forced weights remaining at zero.

(5) Once a local minimum is found, estimate the quality of the local minimum by finding second derivatives for every training pattern and combining them in the evaluation measure (Eval($\widehat{w}$)) of section 6.2. Calculation of second derivatives is described in [9].

(6) Perform random restarts of the network to repeat the last three steps and find other local minima. The estimated weight variances $\mathbf{v}_{w|\boldsymbol{x},\boldsymbol{y},\text{near }\widehat{w}}(w_{n,m})$ or $I(\widehat{w})$ and the evaluation measure Eval($\widehat{w}$) should be retained for each saved low cost local minima $\widehat{w}$. If several different network structures are being tried, repeat the last three steps for these as well.

(7) Choose a few networks with the best evaluation (Eval($\widehat{w}$)). Notice that the networks or local minima should not be chosen on the basis of their generalization error (as given in section 7.1) because the generalization error for a particular local minima $\widehat{w}$ is estimated based on the assumption that the network structure is "correct" and the "true" weights are in fact quite near $\widehat{w}$. That is, use of generalization error assumes we know already what we are trying to determine.

(8) Estimate the generalization error (for out-of-sample prediction) using the equation (11) or (13), possibly combined using the pooled versions at the end of section 7.1. Now that a high posterior structure and a set of weights have been chosen, the assumptions behind the formulas are reasonable.

Once some local minima have been found, inference can be done on new patterns. Relevant approximate predictions are described in section 7.1 that apply if the weight variances are small. Standard inference does forward propagation of the inputs for the new pattern to obtain the output. The adjustments described involve approximation of posterior variance of the output, better approximation of expected output and variance, and averaging over multiple local minima.

These adjustments to the standard back-propagation procedure increase computation during back-propagation in most cases by at most a small factor. Subsequent inference such as calculating variances requires standard matrix calculations. While these methods come with the normative backing of Bayesian statistics, implementation often reveals lessons on how the various approximations and optimizations could be made better.

There are several important areas in which additional research is required. First, what is the quality of the Gaussian approximation to the posterior of the weights of section 6.2? Since the whole approach rests on this, more evaluation, experience, and better approximations are required, for instance, in taking expected values and estimating generalization error. Second, priors or "penalty terms" have been discussed here only for simple network types. What forms are useful for the wider class of networks used in practice? Third, a smooth transition also needs to be developed between Bayesian and uniform convergence methods to handle those cases in which training samples become larger, and to improve the robustness of Bayesian methods when, for instance, the choice of error model is poor. Finally, how might probabilistic methods be applied to ease the computational problems inherent in back-propagation?

Acknowledgments

Wray would like to thank Robin Hanson and Andreas would like to thank David Rumelhart for many fruitful discussions. We also thank Mark Plutowski, Peter Williams, and John Moody for their comments.

References

[1] T. Amemiya, *Advanced Econometrics* (Cambridge, MA, Harvard University Press, 1985).

[2] A. R. Barron and T. M. Cover, "Minimum Complexity Density Estimation," *IEEE Transactions on Information Theory*, **37**(4), 1991.

[3] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis* (New York, Springer-Verlag, 1985).

[4] E.B. Baum and D. Haussler, "What Size Net Gives Valid Generalization?" *Neural Computation*, **1** (1989) 151–160.

[5] J. S. Bridle, "Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition," in F. Fougelman-Soulie and J. Hérault, editors, *Neuro-computing: Algorithms, Architectures and Applications* (New York, Springer-Verlag, 1989).

[6] W. L. Buntine, "Learning Classification Trees," Technical Report FIA-91-30, RIACS and NASA Ames Research Center, Moffett Field, CA, 1991; submitted to *Proceedings of the Third International Workshop on Artificial Intelligence and Statistics.*

[7] W. L. Buntine, "A Theory of Learning Classification Rules" (Doctoral dissertation, University of Technology, Sydney, Australia, 1991).

[8] E. B. Baum and F. Wilczek, "Supervised Learning of Probability Distributions by Neural Networks," pages 52–61 in *Neural Information Processing Systems (NIPS)*, edited by D. Z. Anderson (1987).

[9] W. L. Buntine and A. S. Weigend, "Calculating Second Derivatives on Feedforward Networks," submitted (1991).

[10] N. E. Cotter, "The Stone-Weierstrass Theorem and Its Application to Neural Networks," *IEEE Transactions on Neural Networks*, **1**(4) (1990) 290–295.

[11] A. El-Jaroudi and J. Makhoul, "A New Error Criterion for Posterior Probability Estimation with Neural Nets, pages 185–192 in *International Joint Conference on Neural Networks*, volume III, San Diego, CA (1990).

[12] A. E. Gelfand, S. E. Hills, A. Racine-Poon, and A. F. M. Smith, "Illustration of Bayesian Inference in Normal Data Models Using Gibbs Sampling," *Journal of the American Statistical Association*, **85**(412) (1990) 972–985.

[13] R. M. Golden, "A Unified Framework for Connectionist Systems," *Biological Cybernetics*, **59** (1988) 109–120.

[14] D. Haussler, "A Decision Theoretic Generalization of the PAC Learning Model and Its Application to Some Feed-forward Neural Networks," *Information and Control*, to appear (1991).

[15] E. J. Horvitz, D. E. Heckerman, and C. P. Langlotz, "A Framework for Comparing Alternative Formalisms for Plausible Reasoning," pages 210–214 in *Fifth National Conference on Artificial Intelligence*, Philadelphia, PA (1986).

[16] D. Haussler, M. Kearns, and R. E. Schapire, "Unifying Bounds on the Sample Complexity of Bayesian Learning Using Information Theory and the VC Dimension," in *COLT'91: Workshop on Computational Learning Theory* (San Mateo, CA, Morgan Kaufmann, 1991).

[17] J. B. Hampshire II and B. A. Pearlmutter, "Equivalence Proofs for Multi-layer Perceptron Classifiers and the Bayesian Discrimination Function," in *Proceedings of the 1990 Connectionist Models Summer School*, edited by David S. Touretzky, Jeffrey L. Elman, Terrence J. Sejnowski, and Geoffrey E. Hinton (San Mateo, CA, Morgan Kaufmann, 1990).

[18] T. J. Hastie and R. J. Tibshirani, *Generalised Additive Models* (London, Chapman and Hall, 1990).

[19] M. Ishikawa, "A Structural Learning Algorithm with Forgetting of Link Weights," Technical Report TR-90-7, Electrotechnical Laboratory, Life Electronics Research Center, Tokyo (1990); modified version of a paper presented at *International Joint Conference on Neural Networks*, Washington, D.C. (1989).

[20] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," pae 589 in *Advances in Neural Information Processing Systems 2 (NIPS*89)*, edited by David S. Touretzky (San Mateo, CA, Morgan Kaufmann, 1990).

[21] Y. Le Cun, "Generalization and Network Design Strategies," Technical Report CRG-TR-89-4, Deptartment of Computer Science, University of Toronto, Toronto, M5S 1A4, Canada (1989).

[22] E. Levi, N. Tishby, and S. A. Solla, "A Statistical Approach to Learning and Generalization in Layered Neural Networks," pages 245–260 in *COLT'89: Second Workshop on Computational Learning Theory*, edited by R. Rivest, D. Haussler, and M. K. Warmuth, University of California, Santa Cruz (San Mateo, CA, Morgan Kaufmann, 1989).

[23] D. J. C. MacKay, "A Practical Bayesian Framework for Backprop Networks," submitted to *Neural Computation* (1991).

[24] P. McCullagh and J. A. Nelder, *Generalised Linear Models*, second edition (London, Chapman and Hall, 1989).

[25] J. E. Moody, "Note on Generalization, Regularization, and Architecture Selection in Non-linear Learning Systems," in *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing* (Los Alamitos, CA, IEEE Computer Society, 1991).

[26] S. J. Nowlan, *Soft Competitive Adaption*, (Doctoral dissertation, Carnegie Mellon University, 1991); available as Technical Report CMU-CS-91-126 from the School of Computer Science.

[27] M. Opper and D. Haussler, "Generalised Performance of Bayes Optimal Classification Algorithm for Learning a Perceptron," in *COLT'91: 1991 Workshop on Computational Learning Theory* (San Mateo, CA, Morgan Kaufmann, 1991).

[28] J. Pearl, *Probabilistic Reasoning in Intelligent Systems* (Los Altos, CA, Morgan Kauffman, 1988).

[29] D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on Learning by Back-propagation," Technical Report CMU-CS-86-126, Carnegie Mellon University, Pittsburgh, PA 15213 (1986).

[30] S. J. Press, *Bayesian Statistics* (New York, Wiley, 1989).

[31] J. R. Quinlan, "Unknown Attribute Values in Induction," in *Proceedings of the Sixth International Machine Learning Workshop*, Cornell, New York (San Mateo, CA, Morgan Kaufmann, 1989).

[32] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, "Learning Internal Representations by Error Propagation," page 318 in *Parallel Distributed Processing*, edited by David E. Rumelhart, James L. McClelland, and the PDP Research Group (Cambridge, MA, MIT Press, 1986).

[33] J. Rissanen, "Stochastic Complexity," *Journal of the Royal Statistical Society B*, **49**(3) (1987) 223–239.

[34] C. C. Rodriguez, "Objective Bayesianism and Geometry," in *Maximum Entropy and Bayesian Methods*, edited by P. F. Fougère (Norwell, MA, Kluwer, 1990).

[35] H. S. Seung, H. Sompolinsky, and N. Tishby, "Learning Curves in Large Neural Networks," in *COLT'91: Workshop on Computational Learning Theory* (San Mateo, Morgan Kaufmann, 1991).

[36] L. Stewart, "Hierarchical Bayesian Analysis Using Monte Carlo Integration: Computing Posterior Distributions When There Are Many Possible Models," *The Statistician*, **36** (1987) 211–219.

[37] G. A. F. Seber and C. J. Wild, *Nonlinear Regression* (New York, Wiley, 1989).

[38] G. G. Towell, J. W. Shavlik, and M. O. Noordewier, "Refinement of Approximate Domain Theories by Knowledge-based Neural Networks," pages 861–866 in *Eighth National Conference on Artificial Intelligence*, Boston, MA (1990).

[39] V. Vapnik, *Estimation of Dependencies Based on Empirical Data* (New York, Springer-Verlag, 1982).

[40] C. S. Wallace and P. R. Freeman, "Estimation and Inference by Compact Encoding," *Journal of the Royal Statistical Society B*, **49**(3) (1987) 240–265.

[41] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart, "Predicting the Future: A Connectionist Approach," *International Journal of Neural Systems*, **1** (1990) 193–209.

[42] A. Zellner, "Bayesian Methods and Entropy in Economics and Econometrics," in *Maximum Entropy and Bayesian Methods*, edited by W. T. Grandy, Jr. and L. Schlick, (Norwell, MA, Kluwer, 1990).