Bayesian Back-Propagation

Wray L. Buntine
wray@ptolemy.arc.nasa.gov
RIACS & NASA Ames Research Center
Mail Stop 244-17, Moffet Field, CA 94035, USA

Andreas S. Weigend
andreas@psych.stanford.edu
Jordan Hall (Building 420)
Stanford University, CA 94305–2130, USA

July 3, 1991

Abstract

Connectionist feed-forward networks, trained with back-propagation, can be used both for non-linear regression and for (discrete one-of-C) classification, depending on the form of training. This paper works through approximate Bayesian methods to both these problems. Methods are presented for various statistical components of back-propagation: choosing the appropriate cost function and regularizer (interpreted as a prior), eliminating extra weights, estimating the uncertainty of the remaining 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 adjustments for missing values in the training patterns. These techniques refine and extend some popular heuristic techniques suggested in the literature, and in most cases require at most a small additional factor in computation during back-propagation, or computation once back-propagation has finished. The paper begins with a comparative discussion of Bayesian and related frameworks for the training problem.

Submitted.

Contents

1 Introduction

Back-propagation [RHW86] is a popular scheme for training feed-forward connectionist networks. Recent improvements have looked at tasks such as speeding up learning using more sophisticated gradient descent algorithms [BL88, EJM90], eliminating insignificant weights in order to find networks of an optimal size [LDS90, WHR90, WRH91], and various modifications to apply a network to a one-of-C classification task (prediction of a single discrete variable taking one of C mutually exclusive and exhaustive values) rather than a non-linear regression task (prediction of a single real variable) [Bri89, DL91].

There is much to be gained by looking at the intersection between probabilistic techniques and theory, and feed-forward connectionist networks. On the one hand, feed-forward networks offer a broad computational framework for implementing a wide variety of probabilistic functions, and on the other hand, probability theory offer techniques for refining back-propagation schemes. We consider both these issues in this paper.

Since Bayesian probabilistic methods for learning tasks sometimes prove superior to other approaches on smaller samples [Bun90, Ber85, Bun91, OH91], we here frame the probabilistic component of back-propagation in a Bayesian context. Back-propagation algorithms are known to work well in a variety of noisy and uncertain domains using a variety of different network structures and activation functions, so we propose Bayesian modifications and extensions to existing methods. Of course, in adopting a Bayesian justification for the methods presented, we are not claiming any neurological validity for our methods. We view feed-forward networks as a vehicle for massive parallelism in classification, regression and learning. This occurs because feed-forward networks can be constructed from simple component parts but still are able to represent a broad range of functions.

The beauty of the Bayesian approach arises from the fact that the entire method is derived as an approximation to applying the single simple formula

$$\textit{posterior} \;\propto\; \textit{prior} \cdot \textit{likelihood}$$

to the training problem, together with some principles for reasoning about the prior knowledge available.

Section 2 gives some theoretical background and motivation for using Bayesian methods, and discusses some alternative approaches. Section 3 details the notation and form of networks considered here. Section 4 outlines some 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 5 presents a probabilistic analysis of the training of feed-forward neural networks. 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 [WF87, BC91, Ris87] are also explained in Section 6.3. Finally, Section 7 discusses some extensions to back-propagation involving weight–elimination strategies, prediction of variables and generalization error, and handling missing values.

2 On Bayesian methods

Bayesian methods described here are based on estimating posterior probabilities for different network weights. The posterior corresponds to a relative measure of belief in the many possible network weights, similar to the statistical mechanics idea of an ensemble of networks [LTS89]. Statistical mechanics theory has been developed in the context of training a perceptron in a noise-free environment to estimate the generalization error [OH91, SST91]. While this is an impressive demonstration of the theory, we are concerned with developing Bayesian methods for more general feed-forward networks trained in noisy and/or uncertain environments, as commonly found in practice.

Bayesian and statistical mechanics methods both share the view that since we can never determine the "correct" or "best" weights, we should carefully reason about the reasonable possibilities. In Bayesian practice, however, such as [Mac91] more care is required in the selection of the prior and the error model, etc. We present methods for that here. Bayesian methods carefully separate the components of the learning problem: the priors on networks and network weights which represent our expectations before any training and correspond to regularizers; the utility or loss function for the problem which represent the goal of learning and correspond to quantities such as minimum errors or least squares; and 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. Notice that in general there is no "correct" prior, error model or likelihood, since by definition these vary from problem to problem and it is a challenge to try and make a choice that seems appropriate in a given circumstance. Notions such as the "true" subjective probability function [Gol88] or the "correct" likelihood function derived from the additive energy function [LTS89] are therefore only partially consistent with our general approach, though in several cases produce equivalent results. See also [Hau91] for a discussion in the uniform convergence framework.

Bayesian methods should not be confused with recent results that back-propagation methods are Bayes optimal [HP90]. Bayes optimal refers to the property that back-propagation methods should produce a network approaching 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. Also, notice, that this powerful property rests on certain assumptions, which are discussed below. This requires careful use of techniques in practice.

These properties hold because Bayesian methods are "normative" or "rational" [Ber85, HHL86]: any other approach not approximating them should not perform as well on average. As an example, minimum encoding methods such as MML [WF87] and MDL [Ris87] are often viewed as approximate Bayesian methods. Accordingly, a more exact Bayesian approach convincingly outperforms these encoding methods on the task of learning tree classifiers [Bun90] made popular by Quinlan's ID3 algorithm [Qui86].

Uniform convergence methods [BH89, Hau91], the basis for computational learning theory, approximate Bayesian methods when the sample size is large [Bun91]. Uniform convergence methods require a sample size large enough so that the prior term found in Bayesian methods becomes insignificant (and so can be ignored). Bayesian methods for connectionist networks are therefore an important complement to these and can provide keener insight about learning from smaller samples to the algorithm designer.

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, etc. All methods have particular algorithmic, complexity, and statistical properties that make them more appropriate in one engineering context or another. Cross-validation, for instance, can be fairly easy to implement on top of an existing algorithm so the algorithm can be got up and running quickly (even though its subsequent performance may be quite slow).

The theory behind Bayesian methods rests on two critical assumptions:

(I) the choice of models (or hypotheses) being searched must contain the "true" model (in practice this means a fairly accurate approximation to the "true" model),

(II) and the choice of prior over these models should represent a reasonable initial preference for models in the search space.

Although there is considerable literature on how to choose a prior to minimize the assumptions implicit in the prior [Ber85], and in practice we often consider a range of priors for their suitability [Mac91, Bun90]. Bayesian methods then guarantee best average case performance given these two assumptions and a third assumption:

(III) that 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 is inappropriate for such tasks. So Bayesians say "A good Bayesian will usually out-perform a non-Bayesian but a bad Bayesian can do far worse" [Goo83]. In contrast, uniform convergence methods make no assumptions (the model space does not have to contain the "true" model and no prior is needed) but 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 but for smaller samples the worst and average case can be far different [Bun91].

Non-Bayesians often say that "Bayesian methods require the assumption of a prior on networks which may not be known in practice". In fact, every algorithm or method is making implicit assumptions about a prior that can be explicitly calculated by running the algorithm on many different data sets and seeing the networks that result [Bun90]. In other words, some kind of prior assumption is unavoidable in practice, except where sample sizes are sufficiently large so that prior assumptions become submerged in the wealth of information from the sample. So rather than ignoring the unavoidable, Bayesian methods offer principles for developing a prior and evaluating a prior in a practical application. The intent of Bayesian methods then is to make all assumptions underlying a method explicit and then reconstructing the most rational method based on those assumptions. Techniques for making the Bayesian assumptions more robust also exist, and are referred to as robust statistics [Ber85], however we do not consider them here.

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. A directed acyclic graph is one containing no directed cycles, so the function computed by the network is not computed using any fixed point equations. The network deals with real numbers internally, although the inputs may be discrete, represented as, say, reals 0 and 1.

Denote the input variables to the network as a vector of values x and denote the response variable which the network is intended to predict as y. In the case of regression, the network output corresponds to the predicted regression for y given the input variables. This corresponds to the expected value or mean of the real valued variable 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 n nodes and corresponds to a vector of real values summing to 1. The i-th value is the estimated conditional probability that the output variable should have the i-th discrete value.

Connections (given as directed arcs on a graph) correspond to the flow of information between units or nodes. This can be in the forward direction during inference (classification or regression) and in the backward direction when calculating derivatives. So a connection from node n to m represents that node m receives node n's activation during the inference stage, often multiplied with a weight $w_{m,n}$. The nodes in the graph with no incoming connections correspond to input nodes for the neural net. Their value is given to the system from some external source.

Let the non-output nodes in the network be indexed $1, 2, \ldots$. 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.

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 $u_1, \ldots, u_l$,

$$\begin{aligned} u(u_1, \ldots, u_l) &= \frac{1}{1 + e^{-a \sum_{i=1,\ldots,l} w_{n,i} u_i + b_n}} \\ &= \frac{1}{2}\left(1 + \tanh \frac{a}{2}\left(\sum_{i=1,\ldots,l} w_{n,i} u_i + b_n\right)\right) . \end{aligned}$$

The slope of the sigmoid, a, can be absorbed into weights and bias without loss of generality and is set to one. The bias b_n for the node is denoted $w_{n,0}$. Some other activation functions that have been used are independent Gaussians found in radial basis functions, logistic, exponential and various trigonometric functions [Cot90].

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

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

We call this function a normalizing activation function. A related activation function is the Softmax function of Bridle [Bri89] given by

$$o_i(u_1, \ldots, u_C) = \frac{e^{u_i}}{\sum_{i=1,\ldots,C} e^{u_i}} .$$

4 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 has 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 [TSN90]. Probabilistic interpretations of networks is also discussed in [Bri89].

In this section, we present network structures where 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 be 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 [Pea88]. The neural networks are probabilistic in the sense that they implement a well-understood probability function.

4.1 Logistic networks

Suppose all variables are discrete, the problem is classification, and the problem is distinguished by certain marginal probabilities of key features. For instance, for boolean variables $x_0, x_1, \ldots, x_5$, the key features might be $x_0 x_3$, $x_1 x_4$, $x_3 \overline{x_4}$, etc., and the distinguishing characteristics of the problem are the marginal probabilities $Pr(x_0 x_3)$, $Pr(x_1 x_4)$, $Pr(x_3 \overline{x_4})$, etc. All other marginal probabilities are assumed to to follow somehow from these. A standard method of inference is to find the distribution which assumes the least amount of information apart from the importance of the given marginals. This is termed the maximum entropy argument and the class of distributions so derived belong to the general exponential family [MN89], of which the Gaussian and logistic functions are a special case.

Suppose the key features are represented by boolean functions $c_1, \ldots, c_f$ on the input and output variables (x, y) that return 1 if the feature holds and 0 otherwise. Then the exponential distribution takes the form

$$Pr(x,y) = \exp\left(\alpha_0 + \sum_{i=1,\ldots,f} \alpha_i c_i(x,y)\right) \quad ,$$

where α_0 is a normalizing constant and α are some parameters that determine $Pr(c_i(x,y))$. Since we are interested in the conditional probability distribution $Pr(y \mid x)$, we take the conditional version of this to get

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

For y boolean this can be further simplified to

$$Pr(y \mid x) \;=\; \frac{1}{1 + \exp\left(\sum_{i=1,\ldots,f} \alpha_i (c_i(x,0) - c_i(x,1))\right)} \;,$$

which is the general case of multivariate logistic regression (or logit) [Ame85]. In the simplest case with boolean variables, the distinguishing features are yx_i, $y\overline{x_i}$, $\overline{y}x_i$ and $\overline{yx_i}$ for the input variables x_i. That is, the distribution is fully determined by the probabilities of $Pr(x_i \mid y)$. The distribution reduces to the form

$$Pr(y \mid x) \;=\; \frac{1}{1 + \exp\left(\beta_0 + \sum_{i=1,\ldots,a} \beta_i x_i\right)} \;,$$

thus recovering the sigmoid function. More complex cases introduce higher-order terms into the sum such as $x_1 x_5$, and $x_2 \overline{x_3} x_5$, etc. By introducing enough higher-order terms, any conditional probability distribution on boolean variables can be represented. In practice, higher-order terms 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. For a boolean output variable y, one hidden layer computes the higher order terms from the input variables and the output node takes the weighted sum of these through a sigmoid function. In the general case there would have to be a second hidden layer after the first to compute the exponentials in Equation (1) and the final output layer has normalizing activation functions. We refer to both these kinds of networks as logistic networks.

4.2 Cluster networks

Cluster networks perform one-of-C classification, so they output a probability vector for the discrete output variable y. In the case of discrete inputs, cluster networks can be viewed as an extension of logistic networks where the higher-order terms in the network are discovered by the network during training of the first layer of hidden nodes. The first layer of hidden nodes detect hidden mutually exclusive and exhaustive classes in the data of the form usually found using unsupervised learning or clustering techniques [Pan89]. For instance, if input variables are all boolean then each node in the first layer can correspond to the partial occurrence of a particular noisy conjunct. If input variables are continuous, then nodes in the first layer can correspond to radial basis nodes. The detected classes are, however, fed into the second hidden layer which then constructs an unnormalized probability vector for y, to be normalized by the final output nodes.

Cluster networks implement a probability distribution based on the assumption that there is a hidden variable making all other variables independent. We assume there is a hidden variable z with K mutually exclusive and exhaustive values numbered 1 to K. All other variables, the input variables $x_1, \ldots, x_A$ and the output variable y are independent given z, so the model for the joint probability of $z, x_1, \ldots, x_A, y$ is of the form

$$Pr(z, x_1, \ldots, x_A, y) \;=\; Pr(z)Pr(y \mid z) \prod_{i=1,\ldots A} Pr(x_i \mid z) \quad .$$

For K large enough and all variables discrete, this family of distributions can arbitrarily approximate any other. The hidden variable z in practice has an unknown value and we are interested in the conditional probability of y given x, so we manipulate this joint to produce

$$Pr(y \mid x) \;=\; \frac{\sum_{j=1,\ldots K} Pr(y \mid z=j)Pr(z=j)\prod_{i=1,\ldots A} Pr(x_i \mid z=j)}{\sum_{j=1,\ldots K} Pr(z=j)\prod_{i=1,\ldots A} Pr(x_i \mid z=j)} \; .$$

Notice that if the value for x_i is missing, then the corresponding entry in the product $\prod_{i=1,\ldots A}$ can be dropped and the formula still holds. If a variable x_i is continuous, we can let $Pr(x_i \mid z=j)$ be a Gaussian distribution with mean $\mu_{i,j}$ and standard deviation $\sigma_{i,j}$. In practice, we might want to use a totally different distribution for x_i. The current choice is sufficient for presentation.

The following network implements this family of distributions. The network has two hidden layers with full interconnections between layers. The output nodes (numbered layer 3) have normalizing activation functions, and represent the normalized conditional probability vector for y given x. The second hidden layer has C nodes, one for each value of the output variable y, and has linear activation functions. These activations represent an unnormalized conditional probability vector for y given x. These give activation u_i for $i = 1, \ldots, C$ corresponding to

$$u_i \;\equiv\; \sum_{j=1,\ldots K} Pr(y \mid z=j)Pr(z=j) \prod_{i=1,\ldots A} Pr(x_i \mid z=j) \; .$$

The first hidden layer has K nodes, each one corresponding to a hidden class, with exponential activation functions. These activations represent an unnormalized conditional probability vector for z given x. These give activation u_{C+j} for $j = 1, \ldots, K$ corresponding to

$$u_{C+j} \;\equiv\; Pr(z=j) \prod_{i=1,\ldots A} Pr(x_i \mid z=j) \; .$$

¡hand-drawn and not on line¿

Figure 1: A cluster network

The bottom layer is the input layer. The network form is represented in Figure 1.

Activation functions are as follows:

$$
\begin{aligned}
o_i &= \frac{u_i}{\sum_{j=1,\ldots,C} u_j} && \text{for } i = 1, \ldots, C, \\
u_i &= \sum_{j=1,\ldots,K} w_{i,C+j} u_{C+j} && \text{for } i = 1, \ldots, C \ , \\
u_{C+j} &= \exp \left(w_{C+j,0} + \sum_{i \in discrete} \sum_{h=1,\ldots,d_i} w_{j,i,h} 1_{x_i = h} \right. \\
&\quad \left. + \sum_{i \in continuous} \left(\frac{(x_i - \mu_{i,j})^2}{2\sigma_{i,j}^2} - \frac{1}{2} \log(2\pi\sigma_{i,j}^2) \right) \right) && \text{for } j = 1, \ldots, K \ .
\end{aligned}
$$

Notice the standard w notation for weights has been dropped in the case of $\mu_{i,j}$ and $\sigma_{i,j}$. The weights have the following interpretation:

$$
\begin{aligned}
w_{i,C+j} &\equiv Pr(y = i \mid z = j) \ , \\
w_{j,i,h} &\equiv \log Pr(x_i = h \mid z = j) \ , \\
w_{C+j,0} &\equiv \log Pr(z = j) \ .
\end{aligned}
$$

In addition, $\mu_{i,j}$ and $\sigma_{i,j}$ represent the mean and standard deviation of the Gaussian on x_i.

Notice also that the following constraints hold:

$$
\sum_{i=1,\ldots C} w_{i,C+j} = 1 \quad , \sum_{j=1,\ldots K} e^{w_{j+C,0}} = 1 \quad , \sum_{h=1,\ldots d_i} e^{w_{j,i,h}} = 1 \quad ,
$$

and $w_{i,C+j}$ are non-negative and $w_{C+j,0}$ are negative. These constraints can easily be enforced by restricting the changes made to these network weights during back-propagation. This can be done, for instance, using Lagrangian multipliers [LeC88], or simply by insuring that one weight is determined by all the others according to the constraint.

As an example of these network, suppose variables y and $x_1, \ldots, x_5$ are boolean. Then the conjunct $x_1 \wedge \overline{x_3} \wedge x_5$ is represented at the j-th node of the first layer by having $Pr(x_1 \mid z=j)$, $Pr(x_5 \mid z=j)$ and $Pr(\overline{x_3} \mid z=j)$ close to 1 and making $Pr(x_2 \mid z=j)$ and $Pr(\overline{x_4} \mid z=j)$ neither near 0 or 1, say about 0.5. The activation u_j will then be near zero if an input pattern fails to match the conjunct and closer to 0.25 if the conjunct is matched. Several such conjuncts can be represented, and so expressions in disjunctive normal form (a disjunction of conjunctive terms) are approximately represented.

4.3 Regression networks

A statistical model that has a simple network representation is linear regression [HT90], 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, regularization, cross validation, etc., applied to these systems can then be cast in a network framework. MacKay presents a Bayesian model for this [Mac91]. The choice of basis functions depends on the behavior of y anticipated. 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

$$y \;=\; \sum_{i=1,\ldots,K} w_{0,i} f_i(x, w_{i,0}) \;,$$

which again has a straight forward network interpretation.

5 Probabilistic Analysis

This section covers each component of a Bayesian analysis in turn: interpreting networks as a probability function in Section 5.1; thereby calculating the likelihood of the training sample for a given network and weights, in Section 5.2; considering priors for the weights in a network, in Section 5.3; and finally considering posteriors for the weights in a network, in Section 5.4.

The notation $\mathbf{E}_{A|B}\left(f(A,B)\right)$ denotes the expected (mean) value of the function $f(A,B)$ when A is distributed according to the probability function or probability density function

$Pr(A \mid B)$. If A is continuous, this is calculated with an integral

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

and if A is discrete this is calculated as

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

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

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

5.1 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 i-th output $o_i(x,w)$ represents the conditional probability that the discrete output variable y takes its i-th 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 standard deviation σ. It is more realistic that the standard deviation itself should be a function of x too. 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 will have parameters in common with w. Vapnik [Vap82] 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

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

5.2 The sample likelihood

Given a training sample, we need to look at the likelihood of the sample as a function of the network parameters w. This assumes the weights w and any other parameters are known and gives the likelihood that the sample seen would occur. Bayesian learning equations are presented in the next section that show how this leads to learning of the weights.

For a sample consisting of N examples given as a list of input vectors $\boldsymbol{x} \equiv x_1, \ldots, x_N$ and corresponding output values $\boldsymbol{y} \equiv y_1, \ldots, y_N$, the sample likelihood in the classification problem (assuming examples are independently and identically distributed) is given by

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

The sample likelihood in the regression problem with Gaussian error and standard deviation σ is given by

$$L(\boldsymbol{y} \mid \boldsymbol{x}, w, \sigma) = \prod_{i=1,\ldots,N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - o(x_i,w))^2}{2\sigma^2}\right) = \frac{1}{(2\pi)^{\frac{N}{2}} \sigma^N} \exp\left(-\frac{Ns^2}{\sigma^2}\right) ,$$

where

$$s^2 \equiv \frac{1}{N} \sum_{i=1,\ldots,N} (y_i - o(x_i,w))^2$$

is the observed average squared error. The sample likelihood for the Laplacian error model is similar.

Maximum likelihood methods for network training use the negative logarithm of the likelihood as a "cost" function (to be minimumized) [BW87, EJM90].

5.3 Prior probability of the weights

In the classification case, prior probabilities over the network weights w give our *a priori* belief that $o_i(x', w)$ for $i = 1, \ldots, C$ is the "true" distribution for y given x. In the regression case, prior probabilities over the network weights w give our *a priori* belief that $o(x', w)$ is the "true" mean of y given x. The choice of prior represents an assumption about what form w should take. By representing the prior as a probability distribution $Pr(w)$, this prior assumption is not expressed as hard constraints such as $w_{n,m} = 1$ or $0 < w_{n,m} < 0.5$, but rather as preferences among the many different possible values.

There is considerable literature written about priors (see [Ber85] and references therein), and a number of methods exist for obtaining priors that are considered "non-informative" about the class of theories being considered. We suggest a few priors here, however believe more experience and research is needed for designing priors on neural networks. In general there is no "correct" prior, however, it is important to use a prior that has reasonable properties. See [Bun90] where a number of different priors are discussed for trees and [Mac91], where several priors are tried and compared for a single network learning problem.

When dealing with probabilistic neural networks, as given in Section 4, we can often use the probabilistic interpretation of the network to guide us in the choice of prior. Some examples are given below.

Priors on cluster networks

For instance, the cluster networks of Section 4.2 have weights corresponding to Gaussian parameters or proportions. The parameters $w_{i,C+j}$ for $i = 1, \ldots, C$ are parameters for a multinomial distribution, so $\sum_{i=1,\ldots,C} w_{i,C+j} = 1$. A standard non-informative prior for this is [Ber85, Bun90]

$$Pr(w_{1,C+j}, \ldots, w_{C,C+j}) \ \propto \prod_{i=1,\ldots,C} w_{i,C+j}^{1/C-1} \ ,$$

where we have ignored the normalizing constant so the prior is "$\propto$" rather than "$=$". Similarly, suitable priors for the other parameters can be found by using standard non-informative priors for the class of parameters concerned. Notice transformed parameters $e^{w_{C+j,0}}$ are multinomials, so we use the same form of multinomial prior as before for these. However we need a prior on weights $w_{C+1,j}$, not their exponential. This requires a change of variables, and note that $w_{C+K,0}$ is a function of $w_{C+1,0}, \ldots, w_{C+K-1,0}$. Likewise for $e^{w_{C+j,i,h}}$.

$$\begin{aligned}
Pr(w_{C+1,0}, \ldots, w_{C+K-1,0}) &\propto \exp\left(1/K \sum_{j=1,\ldots,K} w_{C+j,0} - w_{C+K,0}\right) \ , \\
Pr(w_{C+j,i,1}, \ldots, w_{C+j,i,d_i-1}) &\propto \exp\left(1/d_i \sum_{h=1,\ldots,d_i} w_{C+j,i,h} - w_{C+j,i,d_i}\right) \ , \\
Pr(\mu_{i,j}, \sigma_{i,j}) &\propto \frac{1}{\sigma_{i,j}} \ .
\end{aligned}$$

Priors on regression networks

For regression networks, priors say how we expect the mean of y given x to change with x. A standard approach is to say we expect y to vary smoothly as the input variables $x_1, \ldots, x_A$ change. For instance, for linear regression networks, one can say the average second derivative of y w.r.t. x should not be too large. This section argues that a suitable prior on the weights takes the form

$$Pr(w) \ \propto \ C(w)^{-K/2}$$

for a quadratic function $C(w)$ on the weights w determined by the basis functions used in the regression. This is equivalent to but simplifies the approach given in [Mac91] for corresponding priors.

With x defined over the multi-dimensional space X, the average magnitude of the second derivative can be expressed as a function $C(w)$ given by

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

where the norm $||\cdot||$ is an average measure of the size of the second derivative at a single point. We assume the space X is finite but relax this condition later on. One way to calculate the norm $||\cdot||$ at the point x is to calculate 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, this is given as:

$$\begin{aligned}
\left|\left|\frac{\mathrm{d}^2 y}{\mathrm{d}x^2}\right|\right| &= \int_{|\Delta x|=1} \left(\frac{1}{2}\sum_{i,j=1}^{A} \Delta x_i \Delta x_j \frac{\partial^2 y}{\partial x_i \partial x_j}\right)^2 \mathrm{d}\Delta x \Big/ \int_{|\Delta x|=1} \mathrm{d}\Delta x \\
&= \left(\frac{\mathrm{d}^2 y}{\mathrm{d}x_1^2}\right)^2 \qquad\qquad \text{for } A{=}1, \text{ and for large } A \\
&\approx \frac{1}{4A^2}\left(\sum_{i,j=1}^{A} \frac{\partial^2 y}{\partial x_i^2}\frac{\partial^2 y}{\partial x_j^2} + 2\sum_{i,j=1}^{A}\left(\frac{\partial^2 y}{\partial x_i \partial x_j}\right)^2\right) \quad .
\end{aligned}$$

We use the approximation below since it is proportional to the exact calculation when $A=1$. Substituting into $C(w)$ and using the regression model $y \;=\; \sum_{i=1,\ldots,K} w_i f_i(x)$, we then get

$$C(w) \;=\; \sum_{m,n=1}^{K} w_m w_n C_{m,n} \quad ,$$

where

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

We can then put a prior on w in the general form

$$Pr(w \mid \alpha) \;\propto\; \exp\left(-\frac{\alpha}{2}C(w)\right) \quad ,$$

where α is a prior "hyper-parameter" to be determined. A large value of α means we have a strong bias towards smooth functions, and a small value means otherwise. We leave α to be an undetermined parameter because we cannot be sure *a prior* how a particular value of α will effect smoothness. MacKay shows how to determine α *a posterior*, however we show below how to marginalize it out.

This gives a prior in the form of a multivariate Gaussian

$$Pr(w \mid \alpha) \;=\; \frac{\alpha^{K/2}\det^{1/2} C}{(2\pi)^{K/2}}\exp\left(-\frac{\alpha}{2}C(w)\right) \quad ,$$

where $K=|w|$ is the dimensionality of the weight set w. Since we are not really sure what $\alpha \in [0,\infty)$ should be anyway, we can marginalize it out. Since α appears as a factor of $C(w)$, it is a scaling quantity so a suitable prior is [Ber85]

$$Pr(\alpha) \;\propto\; 1/\alpha \quad .$$

This gives

$$
\begin{aligned}
Pr(w) &= \int_0^\infty Pr(w \mid \alpha) Pr(\alpha) \mathrm{d}\alpha \\
&\propto \frac{\Gamma(K/2) \det^{1/2} C}{\pi^{K/2} C(w)^{K/2}} \quad \propto \; C(w)^{-K/2} \quad .
\end{aligned}
$$

During gradient descent, the important contribution is

$$
\frac{\partial - log Pr(w)}{\partial w_i} = K/2 \frac{\partial \log C(w)}{\partial w_i} = \frac{K}{2C(w)} \frac{\partial C(w)}{\partial w_i}
$$

If we compare this with the derivative of the original $Pr(w|\alpha)$, then it follows that

$$
\alpha \approx \frac{K}{C(\widehat{w})} ,
$$

where $\widehat{w}$ is the weights actually used. Since $C(w)$ is the measure of smoothness (where smaller is smoother), this says that one should change weights to increase smoothness of the mean function y. Notice from the definition of $C(w)$ that the partial derivatives $\frac{\partial C(w)}{\partial w_i}$ are calculated when calculating $C(w)$. Likewise for second derivatives. We therefore get that all first and second derivatives of $-log Pr(w)$ w.r.t. w are calculated with the same cost it takes to calculate $C(w)$.

Values of $C_{m,n}$ for a given set of basis functions only have to calculated once, but usually lead to a complex matrix. In this case, a diagonal approximation to the matrix may be sufficient. For instance, when using basis functions in the form

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

for different n_i and index sets I_n, then the diagonal term is given by

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

Entropy priors

Zellner's maximal data information prior for a probability function $Pr(z \mid \theta)$ on parameters θ [Zel90] takes the form

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

where

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

is the usual entropy function giving the entropy for z distributed as $Pr(z|\theta)$, which is a function of θ.

When doing classification, we are constructing a network for the conditional distribution $Pr(y \mid x, w)$. Assuming a distribution for x is known, a prior using the above method can then be formed as follows.

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

This assumes the entropy of x, I_x is independent of w. Since the distribution for x is not actually known, the last line estimates the expectation $\mathbf{E}_x(\cdot)$ with the empirical average from the sample of input vectors $x \equiv x_1, \ldots, x_N$. Since we are forming a prior on weights (parameters) for a conditional distribution of y given x, which presumably is independent of the distribution for x, we can use the sample of points for x to estimate a distribution for x. For the classification case,

$$I_{y|x_d,w}(w) \;=\; -\sum_{i=1}^{C} o_i(x_d, w)\log o_i(x_d, w) \quad .$$

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

$$I_{y|x_d,w,r}(w, r) \;=\; \log o'(x_d, r) + constant \quad .$$

Weight elimination and priors

One particular prior we have worked with assume the weights have a prior probability that corresponds to

$$-\log Pr(w) \;=\; \lambda C(w) + \text{constant} \;=\; \lambda \sum_{l,m} \frac{w_{l,m}^2/w_0^2}{1 + w_{l,m}^2/w_0^2} + \text{constant} \quad . \qquad (2)$$

The unspecified constant term is required to ensure $Pr(w)$ normalizes to 1, but its value is unimportant since we primarily deal with derivatives. This term was first proposed by Rumelhart in 1987. w_0 is the scale for the weights: For $|w_{l,m}| \gg w_0$, the cost of a weight approaches unity (times λ), allowing the interpretation of the complexity term as a counter of significantly sized weights. For $|w_{l,m}| \ll w_0$, the cost is close to zero.

Relevant weights are drawn from a uniform distribution (to allow for normalization of the probability, up to a certain maximum size). Weights that are merely the result of "noise" are drawn from a Gaussian-like distribution centered on zero; they are expected to be small. The corresponding bump around zero can be approximated by a Gaussian with variance $\sigma^2 = w_0^2/\lambda$. The larger λ is, the closer to zero a weight must be to have a reasonable probability of being a member of the "noise" distribution.

The learning rule is then to change the weights according to the gradient of the *entire* cost function, continuously doing justice to the trade-off between error and complexity. This differs from methods that consider a set of fixed models, estimate the parameters for each of them, and then compare between the models by considering the number of parameters. Taking a complexity term into account during learning required λ to be adjusted dynamically during learning. The details of this procedure are described in [WRH91]. We believe there may be a Bayesian interpretation for setting λ, similar to the setting of the α parameter described previously, however we have not investigated this.

5.4 Posterior analysis

To do Bayesian analysis, we need to be able to determine relevant posteriors from priors and likelihoods. For the regression case with Gaussian error and unknown standard deviation σ this is as follows. The product rule for probabilities gives the following:

$$L(\boldsymbol{y} \mid \boldsymbol{x}, w, \sigma) Pr(w, \sigma) Pr(\boldsymbol{x} \mid w, \sigma) \;=\; Pr(w, \sigma \mid \boldsymbol{x}, \boldsymbol{y}) Pr(\boldsymbol{y} \mid \boldsymbol{x}) Pr(\boldsymbol{x})$$

where L is the sample likelihood as before, $Pr(w, \sigma \mid \boldsymbol{x}, \boldsymbol{y})$ equals the posterior probability of the network weights w and the parameter to the error model σ. $Pr(\boldsymbol{x} \mid w, \sigma)$ is the probability of seeing the inputs $\boldsymbol{x}$ given w and σ. This can be assumed independent of w and σ since they just parameterize the model for $\boldsymbol{y}$ given $\boldsymbol{x}$, so $Pr(\boldsymbol{x} \mid w, \sigma) = Pr(\boldsymbol{x})$ which is the prior probability of seeing the set of inputs $\boldsymbol{x}$. $Pr(\boldsymbol{x})$ then cancels out both sides. While $Pr(\boldsymbol{y} \mid \boldsymbol{x})$ also has a particular meaning, for our purposes it is a normalizing constant because it is independent of w and σ. Re-expressing this, and assuming that w and σ are *a priori* independent, we get a standard Bayesian learning equation (see, for instance [Bun90])

$$Pr(w, \sigma \mid \boldsymbol{x}, \boldsymbol{y}) \;=\; \frac{L(\boldsymbol{y} \mid \boldsymbol{x}, w, \sigma) Pr(w) Pr(\sigma)}{\int_{w,\sigma} L(\boldsymbol{y} \mid \boldsymbol{x}, w, \sigma) Pr(w) Pr(\sigma) dw\, d\sigma} \;.$$

When using the network, the weights w are the primary concern and σ is of secondary interest, so we would also like to know $Pr(w \mid \boldsymbol{x}, \boldsymbol{y})$, which can be calculated from $\int_\sigma Pr(w, \sigma \mid \boldsymbol{x}, \boldsymbol{y}) d\sigma$. If we assume the standard non-informative prior for the standard error σ, $Pr(\sigma) = 1/\sigma$ [Ber85], the integration for σ given w can be done in closed form using the Γ integral [Ber85] (with a change of variables $\tau = 1/\sigma^2$). This yields

$$Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \;=\; \frac{L(\boldsymbol{y} \mid \boldsymbol{x}, w) Pr(w)}{\int_w L(\boldsymbol{y} \mid \boldsymbol{x}, w) Pr(w) dw} \;,$$

where

$$L(\boldsymbol{y} \mid \boldsymbol{x}, w) \;=\; \frac{\Gamma(\frac{N-1}{2})}{(N\pi)^{\frac{N-1}{2}} \sqrt{N}} \frac{1}{s^{N-1}} \;.$$

Furthermore, the expected value of σ^2 given w and $\boldsymbol{x}, \boldsymbol{y}$ can be calculated using the same integral, giving a standard result in statistics:

$$\mathbf{E}_{\sigma \mid w, \mathrm{x}, \mathrm{y}} \left(\sigma^2\right) \;=\; \frac{N}{N-1} s^2 \;. \tag{3}$$

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

$$\mathbf{E}_{\Delta \mid w, \mathrm{x}, \mathrm{y}} \left(\Delta^2\right) \;=\; \frac{1}{N(N-1)} \left(\sum_{i=1,\ldots,N} |y_i - o(x_i, w)| \right)^2 \;.$$

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

Lemma 5.1 *In the classification and regression frameworks described above for neural networks, assume the input x is a priori independent of the network weights w and other*