In press: A. Tucker, (Ed.) *CRC Handbook of Computer Science*, CRC Press, Boca Raton, FL.

Neural Networks

Michael I. Jordan
Massachusetts Institute of Technology

Christopher M. Bishop
Aston University

January 18, 1996

1 Introduction

Within the broad scope of the study of artificial intelligence, research in neural networks is characterized by a particular focus on pattern recognition and pattern generation. Many neural network methods can be viewed as generalizations of classical pattern-oriented techniques in statistics and the engineering areas of signal processing, system identification and control theory. As in these parent disciplines, the notion of "pattern" in neural network research is essentially probabilistic and numerical. Neural network methods have had their greatest impact in problems where statistical issues dominate and where data are easily obtained.

A neural network is first and foremost a graph, with patterns represented in terms of numerical values attached to the nodes of the graph, and transformations between patterns achieved via simple message-passing algorithms. Many neural network architectures, however, are also statistical processors, characterized by making particular probabilistic assumptions about data. As we will see, this conjunction of graphical algorithms and probability theory is not unique to neural networks, but characterizes a wider family of probabilistic systems in the form of chains, trees, and networks that are currently being studied throughout AI [Spiegelhalter, et al., 1993].

Neural networks have found a wide range of applications, the majority of which are associated with problems in pattern recognition and control theory. In this context, neural networks can best be viewed as a class of algorithms for statistical modeling and prediction. Based on a source of *training data*, the aim is to produce a statistical model of the process from which the data are generated, so as to allow the best predictions to be made for new data. We shall find it convenient to distinguish three broad types of statistical modeling problem, which we shall call *density estimation*, *classification* and *regression*.

For density estimation problems (also referred to as *unsupervised learning* problems), the goal is to model the unconditional distribution of data described by some vector $\mathbf{x}$. A practical example of the application of density estimation involves the interpretation of X-ray images (mammograms) used for breast cancer screening [Tarassenko, 1995]. In this case the training vectors $\mathbf{x}$ form a sample taken from normal (non-cancerous) images, and a network model is used to build a representation of the density $p(\mathbf{x})$. When a new input vector $\mathbf{x}'$ is presented to the system, a high value for $p(\mathbf{x}')$ indicates a normal image while a low value indicates a novel input which might be characteristic of an abnormality. This is used to label regions of images which are unusual, for further examination by an experienced clinician.

For classification and regression problems (often referred to as *supervised learning* problems), we need to distinguish between *input* variables, which we again denote by $\mathbf{x}$, and *target* variables which we denote by the vector $\mathbf{t}$. Classification problems require that each input vector $\mathbf{x}$ be assigned to one of C classes $\mathcal{C}_1, \ldots, \mathcal{C}_C$, in which case the target variables represent class labels. As an example, consider the problem of recognizing handwritten digits [LeCun, et al., 1989]. In this case the input vector would be some (pre-processed) image of the digit, and the network would have ten outputs, one for each digit, which can be used to assign input vectors to the appropriate class (as discussed in Section 2).

Regression problems involve estimating the values of continuous variables. For example, neural networks have been used as part of the control system for adaptive optics telescopes [Sandler, et al., 1991]. The network input $\mathbf{x}$ consists of one in-focus and one de-focused image of a star and the output $\mathbf{t}$ consists of a set of coefficients that describe the phase distortion due to atmospheric turbulence. These output values are then used to make real-time adjustments of the multiple mirror segments to cancel the atmospheric distortion.

Classification and regression problems can also be viewed as special cases of density estimation. The most general and complete description of the data is given by the probability distribution function $p(\mathbf{x}, \mathbf{t})$ in the joint input-target space. However, the usual goal is to be able to make good predictions for the target variables when presented with new values of the inputs. In this case it is convenient to decompose the joint distribution in the form:

$$p(\mathbf{x}, \mathbf{t}) = p(\mathbf{t}|\mathbf{x})p(\mathbf{x}) \tag{1}$$

and to consider only the conditional distribution $p(\mathbf{t}|\mathbf{x})$, in other words the distribution of $\mathbf{t}$ *given* the value of $\mathbf{x}$. Thus classification and regression involve the estimation of *conditional* densities, a problem which has its own idiosyncracies.

The organization of the chapter is as follows. In Section 2 we present examples of network representations of unconditional and conditional densities. In Section 3 we discuss the problem of adjusting the parameters of these networks to fit them to data. This problem has a number of practical aspects, including the choice of optimization procedure and the method used to control network complexity. We then discuss a broader perspective on probabilistic network models in Section 4. The final section presents further information and pointers to the literature.

2 Representation

In this section we describe a selection of neural network architectures that have been proposed as representations for unconditional and conditional densities. After a brief discussion of density estimation, we discuss classification and regression, beginning with simple models that illustrate the fundamental ideas and then progressing to more complex architectures. We focus here on representational issues, postponing the problem of learning from data until the following section.

2.1 Density estimation

We begin with a brief discussion of density estimation, utilizing the Gaussian mixture model as an illustrative model. We return to more complex density estimation techniques later in the chapter.

Although density estimation can be the main goal of a learning system, as in the diagnosis example mentioned in the introduction, density estimation models arise more often as components of the solution to a more general classification or regression problem. To return to Eq. 1, note that the joint density is composed of $p(\mathbf{t}|\mathbf{x})$, to be handled by classification or regression models, and $p(\mathbf{x})$, the (unconditional) input density. There are several reasons for wanting to form an explicit model of the input density. First, real-life data sets often have missing components in the input vector. Having a model of the density allows the missing components to be "filled in" in an intelligent way. This can be useful both for training and for prediction [cf. Bishop, 1995]. Second, as we see in Eq. 1, a model of $p(\mathbf{x})$ makes possible an estimate of the joint probability $p(\mathbf{x}, \mathbf{t})$. Thus

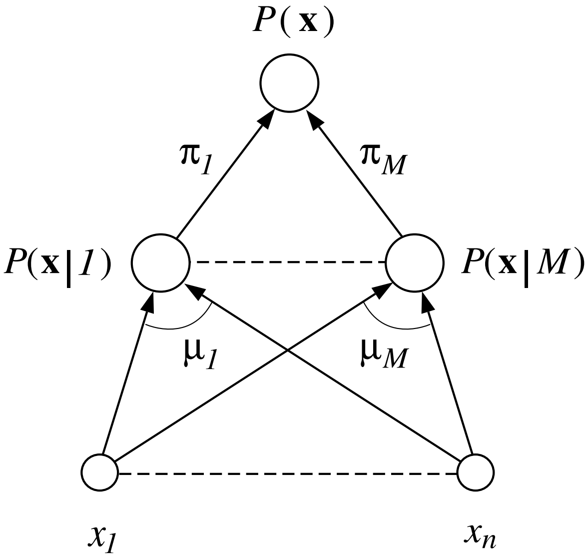

Figure 1: A network representation of a Gaussian mixture distribution. The input pattern $\mathbf{x}$ is represented by numerical values associated with the input nodes in the lower level. Each link has a weight μ_{ij}, which is the j^{th} component of the mean vector for the i^{th} Gaussian. The i^{th} intermediate node contains the covariance matrix $\boldsymbol{\Sigma}_i$ and calculates the Gaussian conditional probability $p(\mathbf{x}|i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. These probabilities are weighted by the mixing proportions π_i and the output node calculates the weighted sum $p(\mathbf{x}) = \sum_i \pi_i p(\mathbf{x}|i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$.

in turn provides us with the necessary information to estimate the "inverse" conditional density $p(\mathbf{x}|\mathbf{t})$. The calculation of such inverses is important for applications in control and optimization.

A general and flexible approach to density estimation is to treat the density as being composed of a set of M simpler densities. This approach involves modeling the observed data as a sample from a *mixture density*:

$$p(\mathbf{x}|\mathbf{w}) = \sum_{i=1}^{M} \pi_i p(\mathbf{x}|i, \mathbf{w}_i), \tag{2}$$

where the π_i are constants known as *mixing proportions*, and the $p(\mathbf{x}|i, \mathbf{w}_i)$ are the *component densities*, generally taken to be from a simple parametric family. A common choice of component density is the multivariate Gaussian, in which case the parameters $\mathbf{w}_i$ are the means and covariance matrices of each of the components. By varying the means and covariances to place and orient the Gaussians appropriately, a wide variety of high-dimensional, multi-modal data can be modeled. This approach to density estimation is essentially a probabilistic form of clustering.

Gaussian mixtures have a representation as a network diagram as shown in Figure 1. The utility of such network representations will become clearer as we proceed; for now, it suffices to note that not only mixture models, but also a wide variety of other classical statistical models for density estimation are representable as simple networks with one or more layers of adaptive weights. These methods include *principal component analysis*, *canonical correlation analysis*, *kernel density* *estimation* and *factor analysis* [Anderson, 1984].

2.2 Linear regression and linear discriminants

Regression models and classification models both focus on the conditional density $p(\mathbf{t}|\mathbf{x})$. They differ in that in regression the target vector $\mathbf{t}$ is a real-valued vector, whereas in classification $\mathbf{t}$ takes its values from a discrete set representing the class labels.

The simplest probabilistic model for regression is one in which $\mathbf{t}$ is viewed as the sum of an underlying deterministic function $f(\mathbf{x})$ and a Gaussian random variable ϵ:

$$\mathbf{t} = f(\mathbf{x}) + \epsilon. \tag{3}$$

If ϵ has zero mean, as is commonly assumed, $f(\mathbf{x})$ then becomes the *conditional mean* $E(\mathbf{t}|\mathbf{x})$. It is this function that is the focus of most regression modeling. Of course, the conditional mean describes only the first moment of the conditional distribution, and, as we discuss in a later section, a good regression model will also generally report information about the second moment.

In a linear regression model the conditional mean is a linear function of $\mathbf{x}$: $E(\mathbf{t}|\mathbf{x}) = W\mathbf{x}$, for a fixed matrix W. Linear regression has a straightforward representation as a network diagram in which the j^{th} input unit represents the j^{th} component of the input vector x_j, each output unit i takes the weighted sum of the input values, and the weight w_{ij} is placed on the link between the j^{th} input unit and the i^{th} output unit.

The conditional mean is also an important function in classification problems, but most of the focus in classification is on a different function known as a *discriminant function.* To see how this function arises and to relate it to the conditional mean, we consider a simple two-class problem in which the target is a simple binary scalar that we now denote by t. The conditional mean $E(t|\mathbf{x})$ is equal to the probability that t equals one, and this latter probability can be expanded via Bayes rule:

$$p(t=1|\mathbf{x}) = \frac{p(\mathbf{x}|t=1)p(t=1)}{p(\mathbf{x})} \tag{4}$$

The density $p(t|\mathbf{x})$ in this equation is referred to as the *posterior probability* of the class given the input, and the density $p(\mathbf{x}|t)$ is referred to as the the *class-conditional density.* Continuing the derivation, we expand the denominator and (with some foresight) introduce an exponential:

$$\begin{aligned} p(t=1|\mathbf{x}) &= \frac{p(\mathbf{x}|t=1)p(t=1)}{p(\mathbf{x}|t=1)p(t=1) + p(\mathbf{x}|t=0)p(t=0)} \\ &= \frac{1}{1 + \exp\left\{-\ln\left[\frac{p(\mathbf{x}|t=1)}{p(\mathbf{x}|t=0)}\right] - \ln\left[\frac{p(t=1)}{p(t=0)}\right]\right\}} \end{aligned} \tag{5}$$

We see that the posterior probability can be written in the form of the *logistic function*:

$$y = \frac{1}{1+e^{-z}}, \tag{6}$$

where z is a function of the likelihood ratio $p(\mathbf{x}|t=1)/p(\mathbf{x}|t=0)$, and the prior ratio $p(t=1)/p(t=0)$. This is a useful representation of the posterior probability if z turns out to be simple.

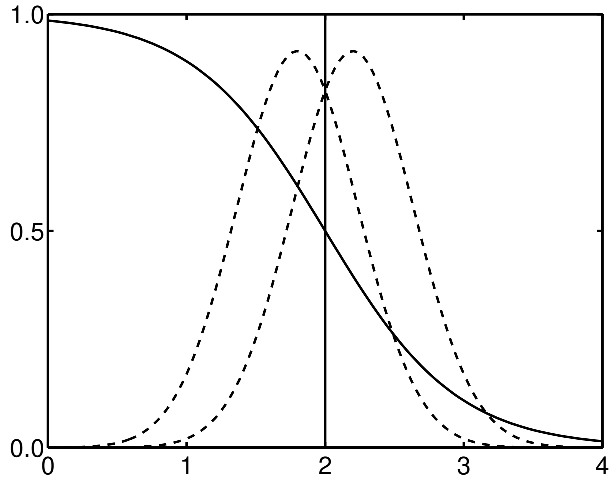

Figure 2: This shows the Gaussian class-conditional densities $p(x|\mathcal{C}_1)$ (dashed curves) for a two-class problem in one dimension, together with the corresponding posterior probability $p(\mathcal{C}_1|x)$ (solid curve) which takes the form of a logistic sigmoid. The vertical line shows the decision boundary for $y = 0.5$ which coincides with the point at which the two density curves cross.

It is easily verified that if the class conditional densities are multivariate Gaussians with identical covariance matrices, then z is a linear function of $\mathbf{x}$: $z = \mathbf{w}^T\mathbf{x} + w_0$. Moreover this representation is appropriate for any distribution in a broad class of densities known as the exponential family (which includes the Gaussian, the Poisson, the gamma, the binomial, and many other densities). All of the densities in this family can be put in the following form:

$$g(\mathbf{x}; \theta, \phi) = \exp\{(\theta^T\mathbf{x} - b(\theta))/a(\phi) + c(\mathbf{x}, \phi)\}, \tag{7}$$

where θ is the *location parameter*, and ϕ is the *scale parameter*. Substituting this general form in Eq. 5, where θ is allowed to vary between the classes and ϕ is assumed to be constant between classes, we see that z is in all cases a linear function. Thus the choice of a linear-logistic model is rather robust.

The geometry of the two-class problem is shown in Figure 2, which shows Gaussian class-conditional densities, and suggests the logistic form of the posterior probability.

The function z in our analysis is an example of a discriminant function. In general a discriminant function is any function that can be used to decide on class membership (Duda and Hart, 1972); our analysis has produced a particular form of discriminant function that is an intermediate step in the calculation of a posterior probability. Note that if we set $z = 0$, from the form of the logistic function we obtain a probability of 0.5, which shows that $z = 0$ is a *decision boundary* between the two classes.

The discriminant function that we found for exponential family densities is linear under the given conditions on ϕ. In more general situations, in which the class-conditional densities are more complex than a single exponential family density, the posterior probability will not be well characterized by the linear-logistic form. Nonetheless it is still useful to retain the logistic function and focus on *nonlinear* representations for the function z. This is the approach taken within the neural network field.

To summarize, we have identified two functions that are important for regression and classification, respectively: the conditional mean and the discriminant function. These are the two functions that are of concern for simple linear models and, as we now discuss, for more complex nonlinear models as well.

2.3 Nonlinear regression and nonlinear classification

The linear regression and linear discriminant functions introduced in the previous section have the merit of simplicity, but are severely restricted in their representational capabilities. A convenient way to see this is to consider the geometrical interpretation of these models. When viewed in the d-dimensional $\mathbf{x}$-space, the linear regression function $\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$ is constant on hyper-planes which are orthogonal to the vector $\mathbf{w}$. For many practical applications we need to consider much more general classes of function. We therefore seek representations for nonlinear mappings which can approximate any given mapping to arbitrary accuracy. One way to achieve this is to transform the original $\mathbf{x}$ using a set of M nonlinear functions $\phi_j(\mathbf{x})$ where $j = 1, \ldots, M$, and then to form a linear combination of these functions, so that:

$$y_k(\mathbf{x}) = \sum_j w_{kj}\phi_j(\mathbf{x}). \tag{8}$$

For a sufficiently large value of M, and for a suitable choice of the $\phi_j(\mathbf{x})$, such a model has the desired 'universal approximation' properties. A familiar example, for the case of 1-dimensional input spaces, is the simple polynomial, for which the $\phi_j(\mathbf{x})$ are simply successive powers of x and the w's are the polynomial coefficients. Models of the form in Eq. 8 have the property that they can be expressed as network diagrams in which there is a *single* layer of adaptive weights.

There are a variety of families of functions in one dimension that can approximate any continuous function to arbitrary accuracy. There is, however, an important issue which must be addressed, called the *curse of dimensionality.* If, for example, we consider an M^{th}-order polynomial then the number of independent coefficients grows as d^M [Bishop, 1995]. For a typical medium-scale application with, say, 30 inputs a fourth-order polynomial (which is still quite restricted in its representational capability) would have over 46,000 adjustable parameters. As we shall see in Section 3.3 in order to achieve good generalization it is important to have more data points than adaptive parameters in the model, and this is a serious problem for methods that have a power law or exponential growth in the number of parameters.

A solution to the problem lies in the fact that, for most real-world data sets, there are strong (often nonlinear) correlations between the input variables such that the data does not uniformly fill the input space but is effectively confined to a sub-space whose dimensionality is called the *intrinsic dimensionality* of the data. We can take advantage of this phenomenon by considering again a model of the form in Eq. 8 but in which the basis functions $\phi_j(\mathbf{x})$ are *adaptive* so that they themselves contain weight parameters whose values can be adjusted in the light of the observed data set. Different models result from different choices for the basis functions, and here we consider

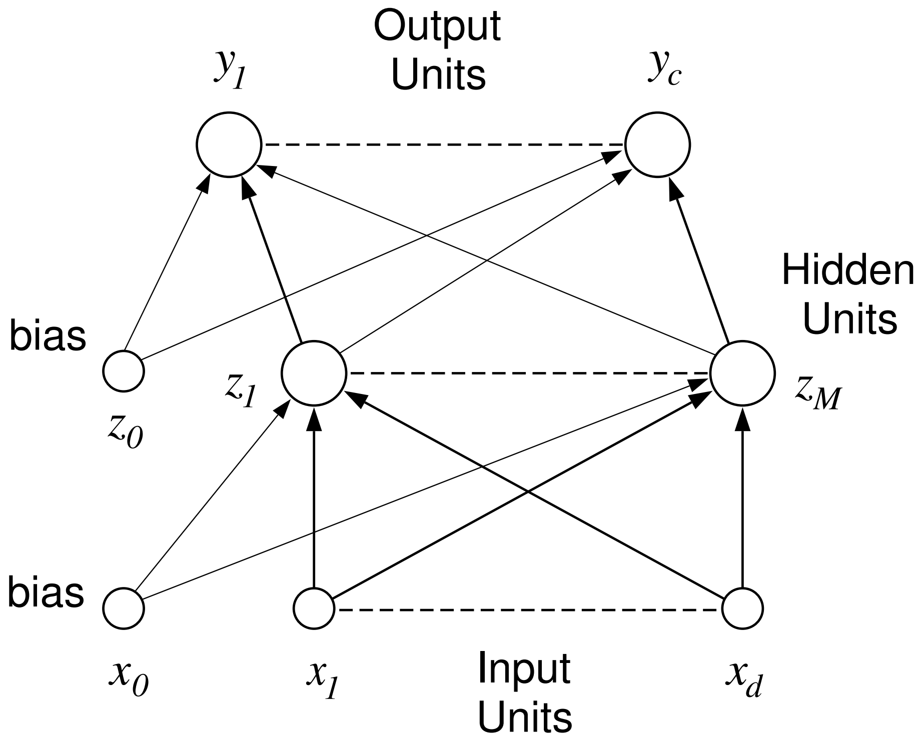

Figure 3: An example of a feed-forward network having two layers of adaptive weights. The bias parameters in the first layer are shown as weights from an extra input having a fixed value of $x_0 = 1$. Similarly, the bias parameters in the second layer are shown as weights from an extra hidden unit, with activation again fixed at $z_0 = 1$.

the two most common examples. The first of these is called the *multilayer perceptron* (MLP) and is obtained by choosing the basis functions to be given by linear-logistic functions (Eq. 6). This leads to a multivariate nonlinear function that can be expressed in the form:

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj} g\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + w_{k0}. \tag{9}$$

Here w_{j0} and w_{k0} are *bias* parameters, and the basis functions are called *hidden units.* The function $g(\cdot)$ is the logistic sigmoid function of Eq. 6. This can also be represented as a network diagram as in Figure 3. Such a model is able to take account of the intrinsic dimensionality of the data because the first-layer weights w_{ji} can adapt and hence orient the surfaces along which the basis function response is constant. It has been demonstrated that models of this form can approximate to arbitrary accuracy any continuous function, defined on a compact domain, provided the number M of hidden units is sufficiently large. The MLP model can be extended by considering several successive layers of weights. Note that the use of nonlinear activation functions is crucial, since if $g(\cdot)$ in Eq. 9 were replaced by the identity, the network would reduce to several successive linear transformations which would itself be linear.

The second common network model is obtained by choosing the basis functions $\phi_j(\mathbf{x})$ in Eq. 8 to be functions of the radial variable $\mathbf{x} - \boldsymbol{\mu}_j$ where $\boldsymbol{\mu}_j$ is the *center* of the jth basis function, which gives rise to the *radial basis function* (RBF) network model. The most common example uses Gaussians of the form:

$$\phi_j(\mathbf{x}) = \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)\right\}. \tag{10}$$

Here both the mean vector $\boldsymbol{\mu}_j$ and the covariance matrix $\boldsymbol{\Sigma}_j$ are considered to be adaptive parameters. The curse of dimensionality is alleviated because the basis functions can be positioned and oriented in input space such as to overlay the regions of high data density and hence to capture the nonlinear correlations between input variables. Indeed, a common approach to training an RBF network is to use a two-stage procedure [Bishop, 1995]. In the first stage the basis function parameters are determined using the input data alone, which corresponds to a density estimation problem using a mixture model in which the component densities are given by the basis functions $\phi_j(\mathbf{x})$. In the second stage the basis function parameters are frozen and the second-layer weights w_{kj} are found by standard least-squares optimization procedures.

2.4 Decision trees

MLP and RBF networks are often contrasted in terms of the support of the basis functions that compose them. MLP networks are often referred to as "global," given that linear-logistic basis functions are bounded away from zero over a significant fraction of the input space. Accordingly, in an MLP, each input vector generally gives rise to a distributed pattern over the hidden units. RBF networks, on the other hand, are referred to as "local," due to the fact that their Gaussian basis functions typically have support over a local region of the input space. It is important to note, however, that local support does not necessarily mean non-overlapping support; indeed, there is nothing in the RBF model that prefers basis functions that have non-overlapping support. A third class of model that does focus on basis functions with non-overlapping support is the *decision tree* model [Breiman, et al., 1984]. A decision tree is a regression or classification model that can be viewed as asking a sequence of questions about the input vector. Each question is implemented as a linear discriminant, and a sequence of questions can be viewed as a recursive partitioning of the input space. All inputs that arrive at a particular leaf of the tree define a polyhedral region in the input space. The collection of such regions can be viewed as a set of basis functions. Associated with each basis function is an output value which (ideally) is close to the average value of the conditional mean (for regression) or discriminant function (for classification; a majority vote is also used). Thus the decision tree output can be written as a weighted sum of basis functions in the same manner as a layered network.

As this discussion suggests, decision trees and MLP/RBF neural networks are best viewed as being different points along the continuum of models having overlapping or non-overlapping basis functions. Indeed, as we show in the following section, decision trees can be treated probabilistically as mixture models, and in the mixture approach the sharp discriminant function boundaries of classical decision trees become smoothed, yielding partially-overlapping basis functions.

There are tradeoffs associated with the continuum of degree-of-overlap—in particular, non-overlapping basis functions are generally viewed as being easier to interpret, and better able to reject noisy input variables that carry little information about the output. Overlapping basis functions are often viewed as yielding lower variance predictions and as being more robust.

2.5 General mixture models

The use of mixture models is not restricted to density estimation; rather, the mixture approach can be used quite generally to build complex models out of simple parts. To illustrate, let us consider using mixture models to model a conditional density in the context of a regression or classification problem. A mixture model in this setting is referred to as a "mixtures of experts" model [Jacobs, et al., 1991].

Suppose that we have at our disposal an elemental conditional model $p(\mathbf{t}|\mathbf{x}, \mathbf{w})$. Consider a situation in which the conditional mean or discriminant exhibits variation on a local scale that is a good match to our elemental model, but the variation differs in different regions of the input space. We could use a more complex network to try to capture this global variation; alternatively we might wish to combine local variants of our elemental models in some manner. This can be achieved by defining the following probabilistic mixture:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{M} p(i|\mathbf{x}, \mathbf{v}) p(\mathbf{t}|\mathbf{x}, i, \mathbf{w}_i). \tag{11}$$

Comparing this mixture to the unconditional mixture defined earlier (Eq. 2), we see that both the mixing proportions and the component densities are now conditional densities dependent on the input vector $\mathbf{x}$. The former dependence is particularly important—we now view the mixing proportion $p(i|\mathbf{x}, \mathbf{v})$ as providing a probabilistic device for choosing different elemental models ("experts") in different regions of the input space. A learning algorithm that chooses values for the parameters $\mathbf{v}$ as well as the values for the parameters $\mathbf{w}_i$ can be viewed as attempting to find both a good partition of the input space and a good fit to the local models within that partition.

This approach can be extended recursively by considering mixtures of models where each model may itself be a mixture model [Jordan and Jacobs, 1994]. Such a recursion can be viewed as providing a probabilistic interpretation for the decision trees discussed in the previous section. We view the decisions in the decision tree as forming a recursive set of probabilistic selections among a set of models. The total probability of a target $\mathbf{t}$ given an input $\mathbf{x}$ is the sum across all paths down the tree:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{M} p(i|\mathbf{x}, \mathbf{u}) \sum_{j=1}^{M} p(j|\mathbf{x}, i, \mathbf{v}_i) \cdots p(\mathbf{t}|\mathbf{x}, i, j, \ldots, \mathbf{w}_{ij\ldots}), \tag{12}$$

where i and j are the decisions made at the first level and second level of the tree, respectively, and $p(\mathbf{t}|\mathbf{x}, i, j, \ldots, \mathbf{w}_{ij\ldots})$ is the elemental model at the leaf of the tree defined by the sequence of decisions. This probabilistic model is a conditional hierarchical mixture. Finding parameter values $\mathbf{u}$, $\mathbf{v}_i$, etc. to fit this model to data can be viewed as finding a nested set of partitions of the input space and fitting a set of local models within the partition.

The mixture model approach can be viewed as a special case of a general methodology known as *learning by committee*. Bishop [1995] provides a discussion of committees; we will also meet them in the section on Bayesian methods later in the chapter.

3 Learning from Data

The previous section has provided a selection of models to choose from; we now face the problem of matching these models to data. In principle the problem is straightforward: given a family of models of interest we attempt to find out how probable each of these models is in the light of the data. We can then select the most probable model (a selection rule known as *maximum a posteriori* or *MAP* estimation), or we can select some highly probable subset of models, weighted by their probability (an approach that we discuss below in the section on Bayesian methods). In practice there are a number of problems to solve, beginning with the specification of the family of models of interest. In the simplest case, in which the family can be described as a fixed structure with varying parameters (e.g., the class of feedforward MLP's with a fixed number of hidden units), the learning problem is essentially one of *parameter estimation.* If on the other hand the family is not easily viewed as a fixed parametric family (e.g., feedforward MLP's with variable number of hidden units), then we must solve the *model selection* problem.

In this section we discuss the parameter estimation problem. The goal will be to find MAP estimates of the parameters by maximizing the probability of the parameters given the data $\mathcal{D}$. We compute this probability using Bayes rule:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}, \tag{13}$$

where we see that to calculate MAP estimates we must maximize the expression in the numerator (the denominator does not depend on $\mathbf{w}$). Equivalently we can minimize the negative logarithm of the numerator. We thus define the following *cost function* $J(\mathbf{w})$:

$$J(\mathbf{w}) = -\ln p(\mathcal{D}|\mathbf{w}) - \ln p(\mathbf{w}), \tag{14}$$

which we wish to minimize with respect to the parameters $\mathbf{w}$. The first term in this cost function is a (negative) log likelihood. If we assume that the elements in the training set $\mathcal{D}$ are conditionally independent of each other given the parameters, then the likelihood factorizes into a product form. For density estimation we have:

$$p(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{w}) \tag{15}$$

and for classification and regression we have:

$$p(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{N} p(\mathbf{t}_n|\mathbf{x}_n, \mathbf{w}). \tag{16}$$

In both cases this yields a log likelihood which is the sum of the log probabilities for each individual data point. For the remainder of this section we will assume this additive form; moreover, we will assume that the log prior probability of the parameters is uniform across the parameters and drop the second term. Thus we focus on *maximum likelihood* (ML) estimation, where we choose parameter values $\mathbf{w}_{ML}$ that maximize $\ln p(\mathcal{D}|\mathbf{w})$.

3.1 Likelihood-based cost functions

Regression, classification and density estimation make different probabilistic assumptions about the form of the data and therefore require different cost functions.

Eq. 3 defines a probabilistic model for regression. The model is a conditional density for the targets $\mathbf{t}$ in which the targets are distributed as Gaussian random variables (assuming Gaussian errors ϵ) with mean values $f(\mathbf{x})$. We now write the conditional mean as $f(\mathbf{x}, \mathbf{w})$ to make explicit the dependence on the parameters $\mathbf{w}$. Given the training set $\mathcal{D} = \{\mathbf{x}_n, \mathbf{t}_n\}_{n=1}^N$, and given our assumption that the targets $\mathbf{t}_n$ are sampled independently (given the inputs $\mathbf{x}_n$ and the parameters $\mathbf{w}$), we obtain:

$$J(\mathbf{w}) = \frac{1}{2} \sum_n \|\mathbf{t}_n - f(\mathbf{x}_n, \mathbf{w})\|^2, \tag{17}$$

where we have assumed an identity covariance matrix and dropped those terms that do not depend on the parameters. This cost function is the standard least squares cost function which is traditionally used in neural network training for real-valued targets. Minimization of this cost function is typically achieved via some form of gradient optimization, as we discuss in the following section.

Classification problems differ from regression problems in the use of discrete-valued targets, and the likelihood accordingly takes a different form. For binary classification the Bernoulli probability model $p(t|\mathbf{x}, \mathbf{w}) = y^t(1-y)^{1-t}$ is natural, where we use y to denote the probability $p(t = 1|\mathbf{x}, \mathbf{w})$. This model yields the following log likelihood:

$$J(\mathbf{w}) = -\sum_n \left[t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\right], \tag{18}$$

which is known as the *cross entropy* function. It can be minimized using the same generic optimization procedures as are used for least squares.

For multi-way classification problems in which there are C categories, where $C > 2$, the multinomial distribution is natural. Define $\mathbf{t}_n$ such that its elements $t_{n,i}$ are one or zero according to whether the n^{th} data point belongs to the i^{th} category, and define $y_{n,i}$ to be the network's estimate of the posterior probability of category i for data point n; i.e., $y_{n,i} \equiv p(t_{n,i} = 1|\mathbf{x}_n, \mathbf{w})$. Given these definitions we obtain the following cost function:

$$J(\mathbf{w}) = -\sum_n \sum_i t_{n,i} \ln y_{n,i}, \tag{19}$$

which again has the form of a cross entropy.

We now turn to density estimation as exemplified by Gaussian mixture modeling. The probabilistic model in this case is that given in Eq. 2. Assuming Gaussian component densities with arbitrary covariance matrices, we obtain the following cost function:

$$J(\mathbf{w}) = -\sum_n \ln \sum_i \pi_i \frac{1}{|\mathbf{\Sigma}_i|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_i)\right\}, \tag{20}$$

where the parameters $\mathbf{w}$ are the collection of mean vectors $\boldsymbol{\mu}_i$, the covariance matrices $\boldsymbol{\Sigma}_i$, and the mixing proportions π_i. A similar cost function arises for the generalized mixture models (cf. Eq. 12).

3.2 Gradients of the cost function

Once we have defined a probabilistic model, obtained a cost function and found an efficient procedure for calculating the gradient of the cost function, the problem can be handed off to an optimization routine. Before discussing optimization procedures, however, it is useful to examine the form that the gradient takes for the examples that we have discussed in the previous two sections.

The i^{th} output unit in a layered network is endowed with a rule for combining the activations of units in earlier layers, yielding a quantity that we denote by z_i, and a function that converts z_i into the output y_i. For regression problems, we assume linear output units such that $y_i = z_i$. For binary classification problems, our earlier discussion showed that a natural output function is the logistic: $y_i = 1/(1+e^{-z_i})$. For multi-way classification, it is possible to generalize the derivation of the logistic function to obtain an analogous representation for the multi-way posterior probabilities known as the *softmax function* [cf. Bishop, 1995]:

$$y_i = \frac{e^{z_i}}{\sum_k e^{z_k}}, \tag{21}$$

where y_i represents the posterior probability of category i.

If we now consider the gradient of $J(\mathbf{w})$ with respect to z_i, it turns out that we obtain a single canonical expression of the following form:

$$\frac{\partial J}{\partial \mathbf{w}} = \sum_i (t_i - y_i)\frac{\partial z_i}{\partial \mathbf{w}}. \tag{22}$$

As discussed by [Rumelhart, et al. 1995], this form for the gradient is predicted from the theory of Generalized Linear Models [McCullagh and Nelder, 1983], where it is shown that the linear, logistic, and softmax functions are (inverse) *canonical links* for the Gaussian, Bernoulli, and multinomial distributions, respectively. Canonical links can be found for all of the distributions in the exponential family, thus providing a solid statistical foundation for handling a wide variety of data formats at the output layer of a network, including counts, time intervals and rates.

The gradient of the cost function for mixture models has an interesting interpretation. Taking the partial derivative of $J(\mathbf{w})$ in Eq. 20 with respect to $\boldsymbol{\mu}_i$, we find:

$$\frac{\partial J}{\partial \boldsymbol{\mu}_i} = \sum_n h_{n,i} \boldsymbol{\Sigma}_i(\mathbf{x}_n - \boldsymbol{\mu}_i), \tag{23}$$

where $h_{n,i}$ is defined as follows:

$$h_{n,i} = \frac{\pi_i |\boldsymbol{\Sigma}_i|^{-1/2} \exp\{-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_i)\}}{\sum_k \pi_k |\boldsymbol{\Sigma}_k|^{-1/2} \exp\{-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k)\}}. \tag{24}$$

When summed over i, the quantity $h_{n,i}$ sums to one, and is often viewed as the "responsibility" or "credit" assigned to the i^{th} component for the n^{th} data point. Indeed, interpreting Eq. 24 using Bayes rule shows that $h_{n,i}$ is the posterior probability that the n^{th} data point is generated by the

i^{th} component Gaussian. A learning algorithm based on this gradient will move the i^{th} mean $\boldsymbol{\mu}_i$ toward the data point $\mathbf{x}_n$, with the effective step size proportional to $h_{n,i}$.

The gradient for a mixture model will always take the form of a weighted sum of the gradients associated with the component models, where the weights are the posterior probabilities associated with each of the components. The key computational issue is whether these posterior weights can be computed efficiently. For Gaussian mixture models, the calculation (Eq. 24) is clearly efficient. For decision trees there are a set of posterior weights associated with each of the nodes in the tree, and a recursion is available that computes the posterior probabilities in an upward sweep [Jordan and Jacobs, 1994]. Mixture models in the form of a chain are known as hidden Markov models, and the calculation of the relevant posterior probabilities is performed via an efficient algorithm known as the Baum-Welch algorithm.

For general layered network structures, a generic algorithm known as "backpropagation" is available to calculate gradient vectors [Rumelhart, et al., 1986]. Backpropagation is essentially the chain rule of calculus realized as a graphical algorithm. As applied to layered networks it provides a simple and efficient method that calculates a gradient in $O(W)$ time per training pattern, where W is the number of weights.

3.3 Optimization algorithms

By introducing the principle of maximum likelihood in Section 1, we have expressed the problem of learning in neural networks in terms of the minimization of a cost function $J(\mathbf{w})$ which depends on a vector $\mathbf{w}$ of adaptive parameters. An important aspect of this problem is that the gradient vector $\nabla_{\mathbf{w}} J$ can be evaluated efficiently (for example by backpropagation). Gradient-based minimization is a standard problem in unconstrained nonlinear optimization, for which many powerful techniques have been developed over the years. Such algorithms generally start by making an initial guess for the parameter vector $\mathbf{w}$ and then iteratively updating the vector in a sequence of steps:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{25}$$

where τ denotes the step number. The initial parameter vector $\mathbf{w}^{(0)}$ is often chosen at random, and the final vector represents a minimum of the cost function at which the gradient vanishes. Due to the nonlinear nature of neural network models, the cost function is generally a highly complicated function of the parameters, and may possess many such minima. Different algorithms differ in how the update $\Delta\mathbf{w}^{(\tau)}$ is computed.

The simplest such algorithm is called *gradient descent* and involves a parameter update which is proportional to the negative of the cost function gradient $\Delta = -\eta\nabla E$ where η is a fixed constant called the learning rate. It should be stressed that gradient descent is a particularly inefficient optimization algorithm. Various modifications have been proposed, such as the inclusion of a *momentum* term, to try to improve its performance. In fact much more powerful algorithms are readily available, as described in standard textbooks such as [Fletcher, 1987]. Two of the best known are called *conjugate gradients* and *quasi-Newton* (or *variable metric*) methods. For the particular case of a sum-of-squares cost function, the *Levenberg–Marquardt* algorithm can also be very effective. Software implementations of these algorithms are widely available.

The algorithms discussed so far are called *batch* since they involve using the whole data set for each evaluation of the cost function or its gradient. There is also a *stochastic* or *on-line* version of gradient descent in which, for each parameter update, the cost function gradient is evaluated using just one of the training vectors at a time (which are then cycled either in order or in a random sequence). While this approach fails to make use of the power of sophisticated methods such as conjugate gradients, it can prove effective for very large data sets, particularly if there is significant redundancy in the data.

3.4 Hessian matrices, error bars and pruning

After a set of weights have been found for a neural network using an optimization procedure, it is often useful to examine second-order properties of the fitted network as captured in the Hessian matrix $H = \partial^2 J/\partial \mathbf{w} \partial \mathbf{w}^T$. Efficient algorithms have been developed to compute the Hessian matrix in time $O(W^2)$ [Bishop, 1995]. As in the case of the calculation of the gradient by backpropagation, these algorithms are based on recursive message passing in the network.

One important use of the Hessian matrix lies in the calculation of error bars on the outputs of a network. If we approximate the cost function locally as a quadratic function of the weights (an approximation which is equivalent to making a Gaussian approximation for the log likelihood), then the estimated variance of the i^{th} output y_i can be shown to be:

$$\hat{\sigma}_{y_i}^2 = \left(\frac{\partial y_i}{\partial \mathbf{w}}\right)^T H^{-1} \left(\frac{\partial y_i}{\partial \mathbf{w}}\right), \tag{26}$$

where the gradient vector $\partial y_i/\partial \mathbf{w}$ can be calculated via backpropagation.

The Hessian matrix is also useful in pruning algorithms. A pruning algorithm deletes weights from a fitted network to yield a simpler network that may outperform a more complex, overfitted network (see below), and may be easier to interpret. In this setting, the Hessian is used to approximate the increase in the cost function due to the deletion of a weight. A variety of such pruning algorithms are available [cf. Bishop, 1995].

3.5 Complexity control

In previous sections we have introduced a variety of models for representing probability distributions, we have shown how the parameters of the models can be optimized by maximizing the likelihood function, and we have outlined a number of powerful algorithms for performing this minimization. Before we can apply this framework in practice there is one more issue we need to address, which is that of model complexity. Consider the case of a mixture model given by Eq. 2. The number of input variables will be determined by the particular problem at hand. However, the number M of component densities has yet to be specified. Clearly if M is too small the model will be insufficiently flexible and we will obtain a poor representation of the true density. What is not so obvious is that if M is too large we can also obtain poor results. This effect is known as *overfitting* and arises because we have a data set of finite size. It is illustrated using a simple example of mixture density estimation in Figure 4. Here a set of 100 data points in one dimension has been generated from a distribution consisting of a mixture of two Gaussians (shown by the

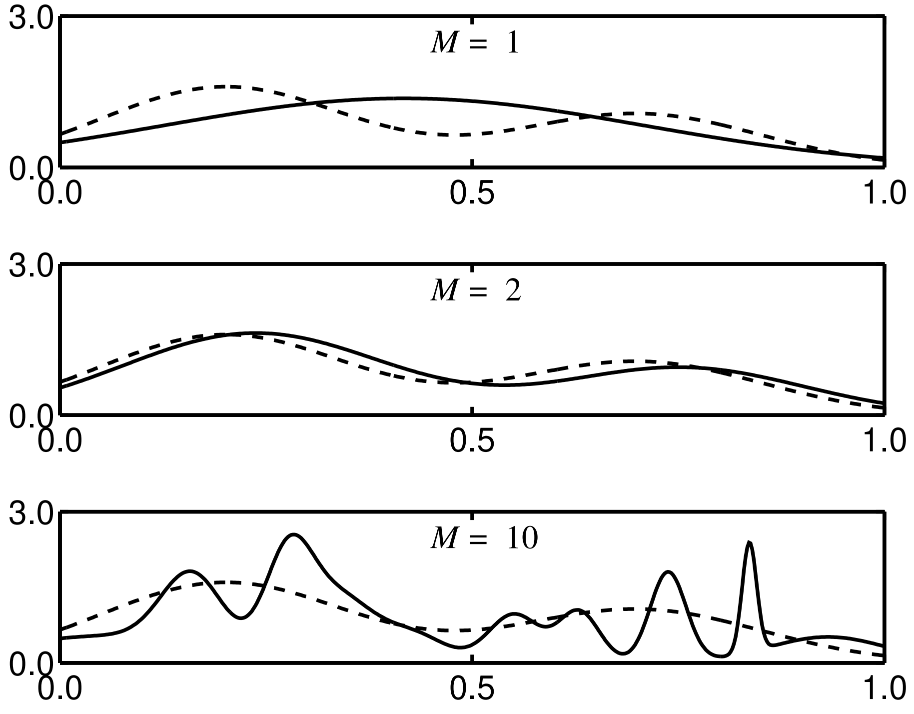

Figure 4: Effects of model complexity illustrated by modeling a mixture of two Gaussians (shown by the dashed curves) using a mixture of M Gaussians (shown by the solid curves). The results are obtained for 20 cycles of EM.

dashed curves). This data set has then been fitted by a mixture of M Gaussians by use of the EM algorithm. We see that a model with 1 component ($M = 1$) gives a poor representation of the true distribution from which the data was generated, and in particular is unable to capture the bimodal aspect. For $M = 2$ the model gives a good fit, as we expect since the data was itself generated from a two-component Gaussian mixture. However, increasing the number of components to $M = 10$ gives a poorer fit, even though this model contains the simpler models as special cases.

The problem is a very fundamental one and is associated with the fact that we are trying to infer an entire distribution function from a finite number of data points, which is necessarily an ill-posed problem. In regression for example there are infinitely many functions which will give a perfect fit to the finite number of data points. If the data are noisy, however, the best generalization will be obtained for a function which does not fit the data perfectly but which captures the underlying function from which the data were generated. By increasing the flexibility of the model we are able to obtain ever better fits to the training data, and this is reflected in a steadily increasing value for the likelihood function at its maximum. Our goal is to model the true underlying density function from which the data was generated since this allows us to make the best predictions for new data. We see that the best approximation to this density occurs for an intermediate value of M.

The same issue arises in connection with nonlinear regression and classification problems. For example, the number M of hidden units in an MLP network controls the model complexity and must be optimized to give the best generalization. In a practical application we can train a variety of different models having different complexity, and compare their generalization performance using an independent validation set, and then select the model with the best generalization. In fact the process of optimizing the complexity using a validation set can lead to some partial overfitting to the validation data itself, and so the final performance of the selected model should be confirmed using a third independent data set called a *test* set.

Some theoretical insight into the problem of overfitting can be obtained by decomposing the error into the sum of *bias* and *variance* terms [Geman, et al., 1992]. A model which is too inflexible is unable to represent the true structure in the underlying density function and this gives rise to a high bias. Conversely a model which is too flexible becomes tuned to the specific details of the particular data set and gives a high variance. The best generalization is obtained from the optimum trade-off of bias against variance.

As we have already remarked, the problem of inferring an entire distribution function from a finite data set is fundamentally ill-posed since there are infinitely many solutions. The problem only becomes well-posed when some additional constraint is imposed. This constraint might be that we model the data using a network having a limited number of hidden units. Within the range of functions which this model can represent there is then a unique function which best fits the data. Implicitly we are assuming that the underlying density function from which the data were drawn is relatively smooth. Instead of limiting the number of parameters in the model, we can encourage smoothness more directly using the technique of *regularization*. This involves adding a penalty term Ω to the original cost function J to give a total cost function $\widetilde{J}$ of the form:

$$\widetilde{J} = J + \nu\Omega \tag{27}$$

where ν is called a regularization coefficient. The network parameters are determined by minimizing $\widetilde{J}$, and the value of ν controls the degree of influence of the penalty term Ω. In practice Ω is typically chosen to encourage smooth functions. The simplest example is called *weight decay* and consists of the sum of the squares of all the adaptive parameters in the model:

$$\Omega = \sum_i w_i^2 \tag{28}$$

Consider the effect of such a term on the MLP function (Eq. 9). If the weights take very small values then the network outputs become approximately linear functions of the inputs (since the sigmoidal function is approximately linear for small values of its argument). The value of ν in Eq. 27 controls the effective complexity of the model, so that for large ν the model is over-smoothed (corresponding to high bias) while for small ν the model can overfit (corresponding to high variance). We can therefore consider a network with a relatively large number of hidden units and control the effective complexity by changing ν. In practice, a suitable value for ν can be found by seeking the value which gives the best performance on a validation set.

The weight decay regularizer (Eq. 28) is simple to implement but suffers from a number of limitations. Regularizers used in practice may be more sophisticated and may contain multiple regularization coefficients [Neal, 1994].

Regularization methods can be justified within a general theoretical framework known as *structural risk minimization* [Vapnik, 1995]. Structural risk minimization provides a quantitative measure of complexity known as the *VC dimension*. The theory shows that the VC dimension predicts the difference between performance on a training set and performance on a test set; thus, the sum of log likelihood and (some function of) VC dimension provides a measure of generalization performance. This motivates regularization methods (Eq. 27) and provides some insight into possible forms for the regularizer Ω.

3.6 Bayesian viewpoint

In earlier sections we discussed network training in terms of the minimization of a cost function derived from the principle of maximum a posteriori or maximum likelihood estimation. This approach can be seen as a particular approximation to a more fundamental, and more powerful, framework based on Bayesian statistics. In the maximum likelihood approach the weights $\mathbf{w}$ are set to a specific value $\mathbf{w}_{\mathrm{ML}}$ determined by minimization of a cost function. However, we know that there will typically be other minima of the cost function which might give equally good results. Also, weight values close to $\mathbf{w}_{\mathrm{ML}}$ should give results which are not too different from those of the maximum likelihood weights themselves.

These effects are handled in a natural way in the Bayesian viewpoint, which describes the weights not in terms of a specific set of values, but in terms of a probability distribution over all possible values. As discussed earlier (cf. Eq. 13), once we observe the training data set $\mathcal{D}$ we can compute the corresponding *posterior* distribution using Bayes' theorem, based on a *prior* distribution function $p(\mathbf{w})$ (which will typically be very broad), and a *likelihood* function $p(\mathcal{D}|\mathbf{w})$:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}. \tag{29}$$

The likelihood function will typically be very small except for values of $\mathbf{w}$ for which the network function is reasonably consistent with the data. Thus the posterior distribution $p(\mathbf{w}|\mathcal{D})$ will be much more sharply peaked than the prior distribution $p(\mathbf{w})$ (and will typically have multiple maxima). The quantity we are interested in is the predicted distribution of target values $\mathbf{t}$ for a new input vector $\mathbf{x}$ once we have observed the data set $\mathcal{D}$. This can be expressed as an integration over the posterior distribution of weights of the form:

$$p(\mathbf{t}|\mathbf{x}, \mathcal{D}) = \int p(\mathbf{t}|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|\mathcal{D})\, d\mathbf{w} \tag{30}$$

where $p(\mathbf{t}|\mathbf{x}, \mathbf{w})$ is the conditional probability model discussed in the introduction.

If we suppose that the posterior distribution $p(\mathbf{w}|\mathcal{D})$ is sharply peaked around a single most-probable value $\mathbf{w}_{\mathrm{MP}}$, then we can write Eq. 30 in the form:

$$\begin{aligned} p(\mathbf{t}|\mathbf{x}, \mathcal{D}) &\simeq p(\mathbf{t}|\mathbf{x}, \mathbf{w}_{\mathrm{MP}}) \int p(\mathbf{w}|\mathcal{D})\, d\mathbf{w} && (31) \\ &= p(\mathbf{t}|\mathbf{x}, \mathbf{w}_{\mathrm{MP}}) && (32) \end{aligned}$$

and so predictions can be made by fixing the weights to their most probable values. We can find the most probable weights by maximizing the posterior distribution, or equivalently by minimizing its negative logarithm. Using Eq. 29, we see that $\mathbf{w}_{\mathrm{MP}}$ is determined by minimizing a regularized cost function of the form in Eq. 27 in which the negative log of the prior $-\ln p(\mathbf{w})$ represents the regularizer $\nu\Omega$. For example, if the prior consists of a zero-mean Gaussian with variance ν^{-1} then we obtain the weight-decay regularizer of Eq. 28.

The posterior distribution will become sharply peaked when the size of the data set is large compared to the number of parameters in the network. For data sets of limited size, however, the posterior distribution has a finite width and this adds to the uncertainty in the predictions for $\mathbf{t}$ which can be expressed in terms of error bars. Bayesian error bars can be evaluated using a local Gaussian approximation to the posterior distribution [MacKay, 1992]. The presence of multiple maxima in the posterior distribution also contributes to the uncertainties in predictions. The capability to assess these uncertainties can play a crucial role in practical applications.

The Bayesian approach can also deal with more general problems in complexity control. This can be done by considering the probabilities of a set of alternative models, given the data set:

$$p(\mathcal{H}_i|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{H}_i) p(\mathcal{H}_i)}{p(\mathcal{D})}. \tag{33}$$

Here different models can also be interpreted as different values of regularization parameters as these too control model complexity. If the models are given the same prior probabilities $p(\mathcal{H}_i)$ then they can be ranked by considering the *evidence* $p(\mathcal{D}|\mathcal{H}_i)$ which itself can be evaluated by integration over the model parameters $\mathbf{w}$. We can simply select the model with the greatest probability. However, a full Bayesian treatment requires that we form a linear combination of the predictions of the models in which the weighting coefficients are given by the model probabilities.

In general, the required integrations, such as that in Eq. 30, are analytically intractable. One approach is to approximate the posterior distribution by a Gaussian centered on $\mathbf{w}_{\mathrm{MP}}$ and then to linearize $p(\mathbf{t}|\mathbf{x}, \mathbf{w})$ about $\mathbf{w}_{\mathrm{MP}}$ so that the integration can be performed analytically [MacKay, 1992]. Alternatively, sophisticated Monte Carlo methods can be employed to evaluate the integrals numerically [Neal, 1994]. An important aspect of the Bayesian approach is that there is no need to keep data aside in a validation set as is required when using maximum likelihood. In practical applications for which the quantity of available data are limited, it is found that a Bayesian treatment generally outperforms other approaches.

3.7 Pre-processing, invariances and prior knowledge

We have already seen that neural networks can approximate essentially arbitrary nonlinear functional mappings between sets of variables. In principle we could therefore use a single network to transform the raw input variables into the required final outputs. However, in practice for all but the simplest problems the results of such an approach can be improved upon considerably by incorporating various forms of pre-processing, for reasons which we shall outline below.

One of the simplest and most common forms of pre-processing consists of a simple normalization of the input, and possibly also target, variables. This may take the form of a linear rescaling of each input variable independently to give it zero mean and unit variance over the training set. For some applications the original input variables may span widely different ranges. Although a linear rescaling of the inputs is equivalent to a different choice of first-layer weights, in practice the optimization algorithm may have considerable difficulty in finding a satisfactory solution when typical input values are substantially different. Similar rescaling can be applied to the output values in which case the inverse of the transformation needs to be applied to the network outputs when the network is presented with new inputs. Pre-processing is also used to encode data in a suitable form. For example, if we have categorical variables such as 'red', 'green' and 'blue', these may be encoded using a 1-of-3 binary representation.

Another widely used form of pre-processing involves reducing the dimensionality of the input space. Such transformations may result in loss of information in the data, but the overall effect can be a significant improvement in performance as a consequence of the curse of dimensionality discussed in Section 3.5. The finite data set is better able to specify the required mapping in the lower-dimensional space. Dimensionality reduction may be accomplished by simply selecting a subset of the original variables, but more typically involves the construction of new variables consisting of linear or nonlinear combinations of the original variables called *features*. A standard technique for dimensionality reduction is principal component analysis [Anderson, 1984]. Such methods, however, make use only of the input data and ignore the target values, and can sometimes be significantly sub-optimal.

Yet another form of pre-processing involves correcting deficiencies in the original data. A common occurrence is that some of the input variables are missing for some of the data points. Correction of this problem in a principled way requires that the probability distribution $p(\mathbf{x})$ of input data be modeled.

One of the most important factors determining the performance of real-world applications of neural networks is the use of *prior knowledge* which is information additional to that present in the data. As an example, consider the problem of classifying hand-written digits discussed in Section 1. The most direct approach would be to collect a large training set of digits and to train a feedforward network to map from the input image to a set of 10 output values representing posterior probabilities for the 10 classes. However, we know that the classification of a digit should be independent of its position within the input image. One way of achieving such *translation invariance* is to make use of the technique of *shared weights*. This involves a network architecure having many hidden layers in which each unit takes inputs only from a small patch, called a *receptive field*, of units in the previous layer. By a process of constraining neighboring units to have common weights, it can be arranged that the output of the network is insensitive to translations of the input image. A further benefit of weight sharing is that the number of independent parameters is much smaller than the number of weights, which assists with the problem of model complexity. This approach is the basis for the highly successful US postal code recognition system of [LeCun, et al., 1989]. An alternative to shared weights is to enlargen the training set artificially by generating "virtual examples" based on applying translations and other transformations to the original training set [Poggio and Vetter, 1992].

4 Graphical models

Neural networks express relationships between variables by utilizing the representational language of graph theory. Variables are associated with nodes in a graph and transformations of variables are based on algorithms that propagate numerical messages along the links of the graph. Moreover, the graphs are often accompanied by probabilistic interpretations of the variables and their interrelationships. As we have seen, such probabilistic interpretations allow a neural network to be understood as a form of probabilistic model, and reduce the problem of learning the weights of a network to a problem in statistics.

Related graphical models have been studied throughout statistics, engineering and AI in recent years. Hidden Markov models, Kalman filters, and path analysis models are all examples of graphical probabilistic models that can be fitted to data and used to make inferences. The relationship between these models and neural networks is rather strong; indeed it is often possible to reduce one kind of model to the other. In this section, we examine these relationships in some detail and provide a broader characterization of neural networks as members of a general family of graphical probabilistic models.

Many interesting relationships have been discovered between graphs and probability distributions [Spiegelhalter, et al., 1993]; [Pearl, 1988]. These relationships derive from the use of graphs to represent conditional independencies among random variables. In an undirected graph, there is a direct correspondence between conditional independence and graph separation—random variables X_i and X_k are conditionally independent given X_j if nodes X_i and X_k are separated by node X_j (we use the symbol "X_i" to represent both a random variable and a node in a graph). This statement remains true for sets of nodes (see Figure 5(a)). Directed graphs have a somewhat different semantics, due to the ability of directed graphs to represent "induced dependencies." An induced dependency is a situation in which two nodes which are marginally independent become conditionally dependent given the value of a third node (see Figure 5(b)). Suppose, for example, that X_i and X_k represent independent coin tosses, and X_j represents the sum of X_i and X_k. Then X_i and X_k are marginally independent but are conditionally dependent given X_j. The semantics of

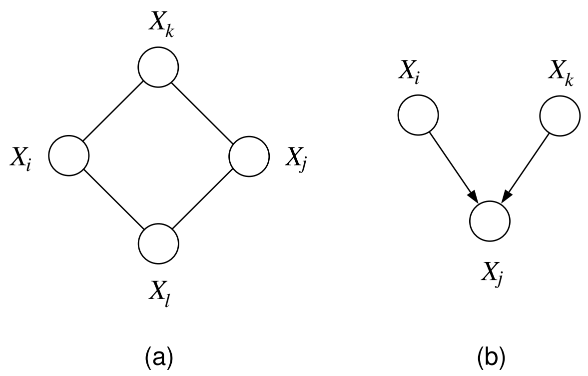

Figure 5: (a) An undirected graph in which X_i is independent of X_j given X_k and X_l, and X_k is independent of X_l given X_i and X_j. (b) A directed graph in which X_i and X_k are marginally independent but are conditionally dependent given X_j.

independence in directed graphs is captured by a graphical criterion known as *d-separation* [Pearl, 1988], which differs from undirected separation only in those cases in which paths have two arrows arriving at the same node (as in Figure 5(b)).

Although the neural network architectures that we have discussed until now have all been based on directed graphs, undirected graphs also play an important role in neural network research. Constraint satisfaction architectures, including the Hopfield network [Hopfield, 1982] and the Boltzmann machine [Hinton and Sejnowski, 1986], are the most prominent examples. A Boltzmann machine is an undirected probabilistic graph that respects the conditional independency semantics described above (cf. Figure 5(a)). Each node in a Boltzmann machine is a binary-valued random variable X_i (or more generally a discrete-valued random variable). A probability distribution on the 2^N possible configurations of such variables is defined via an *energy function* E. Let J_{ij} be the weight on the link between X_i and X_j, let $J_{ij} = J_{ji}$, let α index the configurations, and define the energy of configuration α as follows:

$$E_\alpha = -\sum_{i<j} J_{ij} X_i^\alpha X_j^\alpha. \tag{34}$$

The probability of configuration α is then defined via the Boltzmann distribution:

$$P_\alpha = \frac{e^{-E_\alpha/T}}{\sum_\gamma e^{-E_\gamma/T}}, \tag{35}$$

where the *temperature* T provides a scale for the energy.

An example of a directed probabilistic graph is the hidden Markov model (HMM). An HMM is defined by a set of *state variables* H_i, where i is generally a time or a space index, a set of output variables O_i, a *probability transition matrix* $A = p(H_i|H_{i-1})$, and an *emission matrix* $B = p(O_i|H_i)$. The directed graph for an HMM is shown in Figure 6(a). As can be seen from considering the separatory properties of the graph, the conditional independencies of the HMM are defined by the

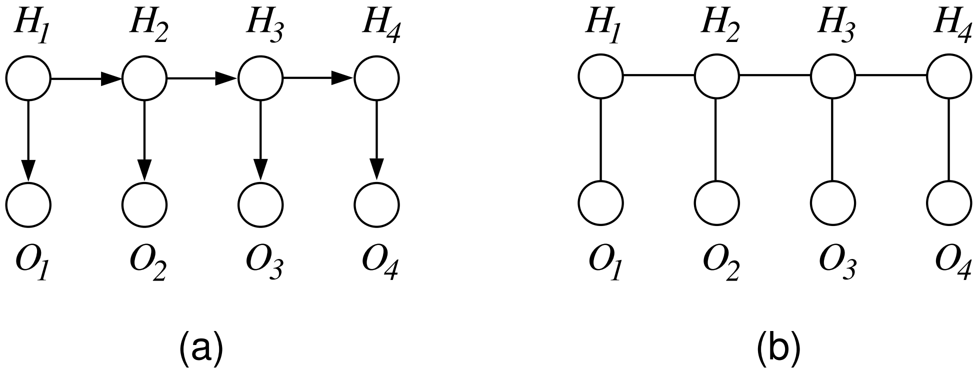

Figure 6: (a) A directed graph representation of an HMM. Each horizontal link is associated with the transition matrix A, and each vertical link is associated with the emission matrix B. (b) An HMM as a Boltzmann machine. The parameters on the horizontal links are logarithms of the entries of the A matrix, and the parameters on the vertical links are logarithms of the entries of the B matrix. The two representations yield the same joint probability distribution.

following Markov conditions:

$$H_i \perp \{H_1, O_1, \ldots, H_{i-2}, O_{i-2}, O_{i-1}\} | H_{i-1}, \qquad 2 \leq i \leq N \tag{36}$$

and

$$O_i \perp \{H_1, O_1, \ldots, H_{i-1}, O_{i-1}\} | H_i, \qquad 2 \leq i \leq N, \tag{37}$$

where the symbol $\perp$ is used to denote independence.

Figure 6(b) shows that it is possible to treat an HMM as a special case of a Boltzmann machine [Luttrell, 1989]; [Saul and Jordan, 1995]. The probabilistic structure of the HMM can be captured by defining the weights on the links as the logarithms of the corresponding transition and emission probabilities. The Boltzmann distribution (Eq. 35) then converts the additive energy into the product form of the standard HMM probabilility distribution. As we will see, this reduction of a directed graph to an undirected graph is a recurring theme in the graphical model formalism.

General mixture models are readily viewed as graphical models [Buntine, 1994]. For example, the unconditional mixture model of Eq. 2 can be represented as a graphical model with two nodes—a multinomial "hidden" node which represents the selected component, a "visible" node representing $\mathbf{x}$, and a directed link from the hidden node to the visible node (see below for the hidden/visible distinction). Conditional mixture models [Jacobs, et al., 1991] simply require another visible node with directed links to the hidden node and the visible nodes. Hierarchical conditional mixture models [Jordan and Jacobs, 1994] require a chain of hidden nodes, one hidden node for each level of the tree.

Within the general framework of probabilistic graphical models, it is possible to tackle general problems of inference and learning. The key problem that arises in this setting is the problem of computing the probabilities of certain nodes, which we will refer to as *hidden nodes*, given the observed values of other nodes, which we will refer to as *visible nodes*. For example, in an HMM, the variables O_i are generally treated as visible, and it is desired to calculate a probability distribution on the hidden states H_i. A similar inferential calculation is required in the mixture models and the Boltzmann machine.

Generic algorithms have been developed to solve the inferential problem of the calculation of posterior probabilities in graphs. Although a variety of inference algorithms have been developed, they can all be viewed as essentially the same underlying algorithm [Shachter, Andersen, and Szolovits, 1994]. Let us consider undirected graphs. A special case of an undirected graph is a *triangulated graph* [Spiegelhalter, et al., 1993], in which any cycle having four or more nodes has a chord. For example, the graph in Figure 5(a) is not triangulated, but becomes triangulated when a link is added between nodes X_i and X_j. In a triangulated graph, the cliques of the graph can be arranged in the form of a *junction tree*, which is a tree having the property that any node that appears in two different cliques in the tree also appears in every clique on the path that links the two cliques (the "running intersection property"). This cannot be achieved in non-triangulated graphs. For example, the cliques in Figure 5(a) are $\{X_i, X_k\}$, $\{X_k, X_j\}$, $\{X_j, X_l\}$, and it is not possible to arrange these cliques into a tree that obeys the running intersection property. If a chord is added the resulting cliques are $\{X_i, X_j, X_k\}$ and $\{X_i, X_j, X_l\}$, and these cliques can be arranged as a simple chain that trivially obeys the running intersection property. In general, it turns out that the probability distributions corresponding to triangulated graphs can be characterized as *decomposable*, which implies that they can be factorized into a product of local functions ("potentials") associated with the cliques in the triangulated graph.[1] The calculation of posterior probabilities in decomposable distributions is straightforward, and can be achieved via a local message-passing algorithm on the junction tree [Spiegelhalter, et al., 1993].

Graphs that are not triangulated can be turned into triangulated graphs by the addition of links. If the potentials on the new graph are defined suitably as products of potentials on the original graph, then the independencies in the original graph are preserved. This implies that the algorithms for triangulated graphs can be used for *all* undirected graphs; an untriangulated graph is first triangulated (see Figure 7). Moreover, it is possible to convert *directed* graphs to undirected graphs in a manner that preserves the probabilistic structure of the original graph [Spiegelhalter, et al., 1993]. This implies that the junction tree algorithm is indeed generic; it can be applied to any graphical model.

The problem of calculating posterior probabilities on graphs is NP-hard; thus, a major issue in the use of the inference algorithms is the identification of cases in which they are efficient. Chain structures such as HMM's yield efficient algorithms, and indeed the classical forward-backward algorithm for HMM's is a special, efficient case of the junction tree algorithm [Heckerman, Jordan, and Smyth, 1996]. Decision tree structures such as the hierarchical mixture of experts yield efficient algorithms, and the recursive posterior probability calculation of [Jordan and Jacobs, 1994] described earlier is also a special case of the junction tree algorithm. All of the simpler mixture model calculations described earlier are therefore also special cases. Another interesting special

[1] An interesting example is a Boltzmann machine on a triangulated graph. The potentials are products of $\exp(J_{ij})$ factors, where the product is taken over all (i, j) pairs in a particular clique. Given that the product across potentials must be the joint probability, this implies that the partition function (the denominator of Eq. 35) must be unity in this case.

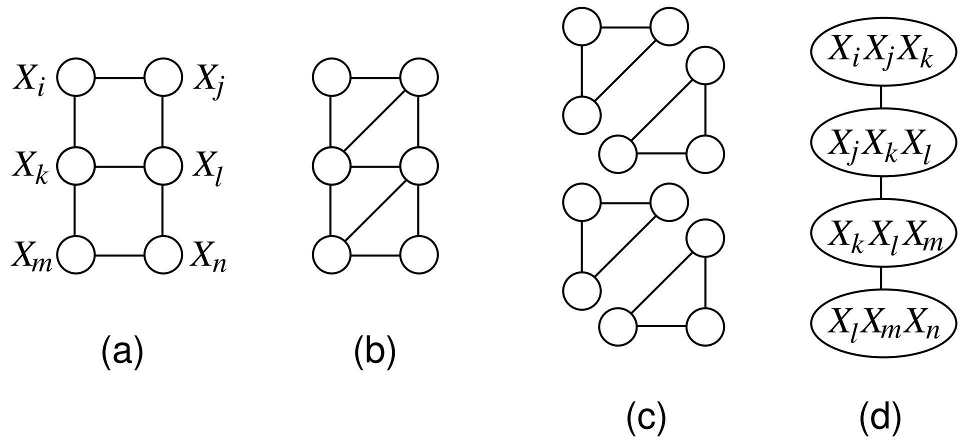

Figure 7: The basic structure of the junction tree algorithm for undirected graphs. The graph in (a) is first triangulated (b), then the cliques are identified (c), and arranged into a tree (d). Products of potential functions on the nodes in (d) yield probability distributions on the nodes in (a).

case is the state estimation algorithm of the Kalman filter [Shachter and Kenley, 1989]. Finally, there are a variety of special cases of the Boltzmann machine which are amenable to the exact calculations of the junction tree algorithm [Saul and Jordan, 1995].

For graphs that are outside of the tractable categories of trees and chains, the junction tree algorithm often performs surprisingly well, but for highly connected graphs the algorithm can be too slow. In such cases, approximate algorithms such as Gibbs sampling are utilized. A virtue of the graphical framework is that Gibbs sampling has a generic form, which is based on the notion of a *Markov boundary* [Pearl, 1988]. A special case of this generic form is the stochastic update rule for general Boltzmann machines.

Our discussion has emphasized the unifying framework of graphical models both for expressing probabilistic dependencies in graphs and for describing algorithms that perform the inferential step of calculating posterior probabilities on these graphs. The unification goes further, however, when we consider learning. A generic methodology known as the Expectation-Maximization (EM) algorithm is available for MAP and Bayesian estimation in graphical models [Dempster, Laird, and Rubin, 1977]. EM is an iterative method, based on two alternating steps: an *E step*, in which the values of hidden variables are estimated, based on the current values of the parameters and the values of visible variables, and an *M step*, in which the parameters are updated based on the estimated values obtained from the E step. Within the framework of the EM algorithm, the junction tree algorithm can readily be viewed as providing a generic E step. Moreover, once the estimated values of the hidden nodes are obtained from the E step, the graph can be viewed as fully observed, and the M step is a standard MAP or ML problem. The standard algorithms for all of the tractable architectures described above (mixtures, trees and chains) are in fact instances of this general graphical EM algorithm, and the learning algorithm for general Boltzmann machines is a special case of a generalization of EM known as GEM [Dempster, et al., 1977].

What about the case of feedforward neural networks such as the multilayer perceptron? It is in fact possible to associate binary hidden values with the hidden units of such a network (cf. our earlier discussion of the logistic function; see also [Amari, 1995]) and apply the EM algorithm directly. For N hidden units, however, there are 2^N patterns whose probabilities must be calculated in the E step. For large N, this is an intractable computation, and recent research has therefore begun to focus on fast methods for approximating these distributions [Hinton, et al., 1995]; [Saul, et al., 1995].

References

Amari, S. 1995. The EM algorithm and information geometry in neural network learning. *Neural Computation*, 7(1):13–18.

Anderson, T. W. 1984. *An Introduction to Multivariate Statistical Analysis.* John Wiley, New York.

Bengio, Y. 1996. *Neural Networks for Speech and Sequence Recognition.* Thomson Computer Press, London.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford University Press.

Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.J. 1984. *Classification and Regression Trees.* Wadsworth International Group, Belmont, CA.

Buntine, W. 1994. Operations for learning with graphical models. *Journal of Artificial Intelligence Research* 2:159–225.

Dempster, A.P., Laird, N.M., and Rubin, D.B. 1977. Maximum-likelihood from incomplete data via the EM algorithm. *J. of Royal Statistical Society, B39*:1–38.

Fletcher, R. 1987. *Practical Methods of Optimization*, 2nd ed. John Wiley, New York.

Geman, S., Bienenstock, E., and Doursat, R. 1992. Neural networks and the bias/variance dilema. *Neural Computation*, 4:1–58.

Heckerman, D., Jordan, M. I., and Smyth, P. 1996. *Conditional independence graphs for hidden Markov probability models.* Tech. Rep., Center for Biological and Computational Learning, Massachusetts Institute of Technology.

Hertz, J., Krogh, A, and Palmer, R. G. 1991. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA.

Hinton, G. E., Dayan, P., Frey, B., and Neal, R. 1995. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1161.

Hinton, G. E. and Sejnowski, T. 1986. Learning and relearning in Boltzmann machines. In *Parallel distributed processing: Volume 1*, ed., D. E. Rumelhart and J. L. McClelland, p. 282–317. MIT Press, Cambridge, MA.

Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J., and Hinton, G.E. 1991. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87.

Jordan, M.I. and Jacobs, R.A. 1994. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214.

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

Luttrell, S. 1989. The Gibbs machine applied to hidden Markov model problems. *Royal Signals and Radar Establishment: SP Research Note 99*, Malvern, UK.

MacKay, D. J. C. 1992. A practical Bayesian framework for back-propagation networks. *Neural Computation*, 4:448–472.

McCullagh, P. and Nelder, J.A. 1983. *Generalized Linear Models*. Chapman and Hall, London.

Neal, R. M. 1994. *Bayesian Learning for Neural Networks*. Unpublished PhD thesis, Department of Computer Science, University of Toronto, Canada.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo, CA.

Poggio, T. and Vetter, T. 1992. *Recognition and structure from one 2D model view: Observations on prototypes, object classes and symmetries*. AI Memo 1347, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Rabiner, L.R. 1989. A tutorial on Hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286.

Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. 1995. Backpropagation: The basic theory. In *Backpropagation: Theory, Architectures, and Applications*, ed. Y. Chauvin, and D.E. Rumelhart, p. 1–35. Lawrence Erlbaum, Hillsdale, NJ.

Saul, L. K., Jaakkola, T., and Jordan, M. I. 1995. Mean field learning theory for sigmoid belief networks. Computational Cognitive Science Tech. Rep. 9501, MIT, Cambridge, MA.

Saul, L. K. and Jordan, M. I. 1995. Boltzmann chains and hidden Markov models. In *Advances in Neural Information Processing Systems 7*, ed., G. Tesauro, D. Touretzky, and T. Leen. Cambridge, MA, MIT Press.

Sandler, D. G., Barrett, T. K., Palmer, D. A., Fugate, R. Q., and Wild, W. J. 1991. Use of a neural network to control an adaptive optics system for an astronomical telescope, *Nature*, 351:300–302.

Shachter, R., Andersen, S., and Szolovits, P. 1994. Global conditioning for probabilistic inference in belief networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, p. 514–522. Seattle, WA.

Shachter, R. and Kenley, C. 1989. Gaussian influence diagrams. *Management Science*, 35(5):527–550.

Spiegelhalter, D., Dawid, A., Lauritzen, S., and Cowell, R. 1993. Bayesian analysis in expert systems. *Statistical Science*, 8(3):219–283.

Tarassenko, L. 1995. Novelty detection for the identification of masses in mammograms. *Proceedings Fourth IEE International Conference on Artificial Neural Networks*, 4:442–447.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York.

Further information

In this chapter we have emphasized the links between neural networks and statistical pattern recognition. A more extensive treatment from the same perspective can be found in [Bishop, 1995]. For a view of recent research in the field, the proceedings of the annual NIPS (Neural Information Processing Systems; MIT Press) conferences are highly recommended.

Neural computing is now a very broad field and there are many topics which have not been discussed for lack of space. Here we aim to provide a brief overview of some of the more significant omissions, and to give pointers to the literature.

The resurgence of interest in neural networks during the 1980's was due in large part to work on the statistical mechanics of fully connected networks having symmetric connections (i.e. if unit i sends a connection to unit j then there is also a connection from unit j back to unit i with the same weight value). We have briefly discussed such systems; a more extensive introduction to this area can be found in [Hertz, et al., 1991].

The implementation of neural networks in specialist VLSI hardware has been the focus of much research, although by far the majority of work in neural computing is undertaken using software implementations running on standard platforms.

An implicit assumption throughout most of this chapter is that the processes which give rise to the data are stationary in time. The techniques discussed here can readily be applied to problems such as time series forecasting, provided this stationarity assumption is valid. If, however, the generator of the data is itself evolving with time then more sophisticated techniques must be used, and these are the focus of much current research [see Bengio, 1996].

One of the original motivations for neural networks was as models of information processing in biological systems such as the human brain. This remains the subject of considerable research activity, and there is a continuing flow of ideas between the fields of neurobiology and of artificial neural networks. Another historical springboard for neural network concepts was that of adaptive control, and again this remains a subject of great interest.

Defining terms

Classification A learning problem in which the goal is to assign input vectors to one of a number of (usually mutually exclusive) classes.

Boltzmann machine An undirected network of discrete valued random variables, where an energy function is associated with each of the links, and for which a probability distribution is defined by the Boltzmann distribution.

Cost function A function of the adaptive parameters of a model whose minimum is used to define suitable values for those parameters. It may consist of a likelihood function and additional terms.

Decision tree A network that performs a sequence of classificatory decisions on an input vector and produces an output vector that is conditional on the outcome of the decision sequence.

Density estimation The problem of modeling a probability distribution from a finite set of examples drawn from that distribution.

Discriminant function A function of the input vector which can be used to assign inputs to classes in a classification problem.

Hidden Markov model A graphical probabilistic model characterized by a state vector, an output vector, a state transition matrix, an emission matrix and an initial state distribution.

Likelihood function The probability of observing a particular data set under the assumption of a given parametrized model, expressed as a function of the adaptive parameters of the model.

Mixture model A probability model which consists of a linear combination of simpler component probability models.

Multilayer perceptron The most common form of neural network model, consisting of successive linear transformations followed by processing with nonlinear activation functions.

Overfitting The problem in which a model which is too complex captures too much of the noise in the data, leading to poor generalization.

Radial basis function network A common network model consisting of a linear combination of basis functions each of which is a function of the difference between the input vector and a center vector.

Regression A learning problem in which the goal is to map each input vector to a real-valued output vector.

Regularization A technique for controlling model complexity and improving generalization by the addition of a penalty term to the cost function.

VC dimension A measure of the complexity of a model. Knowledge of the VC dimension permits an estimate to be made of the difference between performance on the training set and performance on a test set.