CHAPTER 7

Committee Machines

7.1 INTRODUCTION

In the previous three chapters we describe three different approaches to supervised learning. The MLP trained with the back-propagation algorithm, discussed in Chapter 4, relies on a form of global optimization for its design. The RBF network, discussed in Chapter 5, relies on local optimization for its design. The support vector machine, discussed in Chapter 6, exploits VC-dimension theory for its design. In this chapter we discuss another class of methods for solving supervised learning tasks. The approach used here is based on a commonly used engineering principle: divide and conquer.

According to the *principle of divide and conquer,* a complex computational task is solved by dividing it into a number of computationally simple tasks and then combining the solutions to those tasks. In supervised learning, computational simplicity is achieved by distributing the learning task among a number of *experts,* which in turn divides the input space into a set of subspaces. The combination of experts is said to constitute a *committee machine*. Basically, it fuses knowledge acquired by experts to arrive at an overall decision that is supposedly superior to that attainable by any one of them acting alone. The idea of a committee machine may be traced back to Nilsson (1965); the network structure considered therein consisted of a layer of elementary perceptrons followed by a vote-taking perceptron in the second layer.

Committee machines are universal approximators. They may be classified into two major categories:

1. *Static structures.* In this class of committee machines, the responses of several predictors (experts) are combined by means of a mechanism that does *not* involve the input signal, hence the designation "static." This category includes the following methods:
   - *Ensemble averaging,* where the outputs of different predictors are linearly combined to produce an overall output.
   - *Boosting,* where a weak learning algorithm is converted into one that achieves arbitrarily high accuracy.
2. *Dynamic structures.* In this second class of committee machines, the input signal is directly involved in actuating the mechanism that integrates the outputs of the

individual experts into an overall output, hence the designation "dynamic." Here we mention two kinds of dynamic structures:

- *Mixture of experts,* in which the individual responses of the experts are nonlinearly combined by means of a single gating network.
- *Hierarchical mixture of experts,* in which the individual responses of the experts are nonlinearly combined by means of several gating networks arranged in a hierarchical fashion.

In the mixture of experts, the principle of divide and conquer is applied just once, whereas in the hierarchical mixture of experts it is applied several times, resulting in a corresponding number of levels of hierarchy.

The mixture of experts and hierarchical mixture of experts may also be viewed as examples of modular networks. A formal definition of the notion of *modularity* is (Osherson et al., 1990):

> A neural network is said to be modular if the computation performed by the network can be decomposed into two or more modules (subsystems) that operate on distinct inputs without communicating with each other. The outputs of the modules are mediated by an integrating unit that is not permitted to feed information back to the modules. In particular, the integrating unit both (1) decides how the outputs of the modules should be combined to form the final output of the system, and (2) decides which modules should learn which training patterns.

This definition of modularity rules out the static class of committee machines since there is no integrating unit at the output that has a decision-making role.

Organization of the Chapter

This chapter is organized in two parts. The class of static structures is covered in the first part, encompassing Sections 7.2 through 7.5. Section 7.2 discusses the method of ensemble averaging, followed by a computer experiment in Section 7.3. Section 7.4 discusses the boosting technique, followed by a computer experiment in Section 7.5.

The class of dynamic structures is covered in the second part of the chapter, encompassing Sections 7.6 through 7.13. Specifically, Section 7.6 discusses the mixture of experts (ME) as an associative Gaussian mixture model. Section 7.7 discusses the more general case, namely the hierarchical mixture of experts (HME). This latter model is closely related to standard decision trees. Then Section 7.8 describes how a standard decision tree may be used to solve the model selection problem (i.e., number of gating and expert networks) for HME. In Section 7.9, we define some *a posteriori* probabilities that assist us in the formulation of learning algorithms for the HME model. In Section 7.10 we lay the foundation for solving the parameter estimation problem by formulating the likelihood function for the HME model. Section 7.11 presents an overview of learning strategies. This is followed by a detailed discussion of the so-called EM algorithm in Section 7.12 and its application to the HME model in Section 7.13.

The chapter concludes with some final remarks in Section 7.14.

7.2 ENSEMBLE AVERAGING

Figure 7.1 shows a number of differently trained neural networks (i.e., experts), which share a common input and whose individual outputs are somehow combined to produce an overall output y. To simplify the presentation, the outputs of the experts are assumed to be scalar-valued. Such a technique is referred to as an *ensemble averaging method*[1]. The motivation for its use is two-fold:

- If the combination of experts in Fig. 7.1 were replaced by a single neural network, we would have a network with a correspondingly large number of adjustable parameters. The training time for such a large network is likely to be longer than for the case of a set of experts trained in parallel.
- The risk of overfitting the data increases when the number of adjustable parameters is large compared to cardinality (i.e., size of the set) of the training data.

In any event, in using a committee machine as depicted in Fig. 7.1, the expectation is that the differently trained experts converge to different local minima on the error surface, and overall performance is improved by combining the outputs in some way.

Consider first the case of a single neural network that has been trained on a given data set. Let $\mathbf{x}$ denote an input vector not seen before, and let d denote the corresponding desired response (representing a class label or numerical response); $\mathbf{x}$ and d represent realizations of the random vector $\mathbf{X}$ and random variable D, respectively. Let $F(\mathbf{x})$ denote the input–output function realized by the network. Then, in light of the material on the bias/variance dilemma presented in Chapter 2, we may decompose the mean-square error between $F(\mathbf{x})$ and the conditional expectation $E[D|\mathbf{X} = \mathbf{x}]$into its bias and variance components as follows:

$$E_{\mathcal{D}}[(F(\mathbf{x}) - E[D|\mathbf{X} = \mathbf{x}])^2] = B_{\mathcal{D}}(F(\mathbf{x})) + V_{\mathcal{D}}(F(\mathbf{x})) \tag{7.1}$$

where $B_{\mathcal{D}}(F(\mathbf{x}))$ is the bias squared:

$$B_{\mathcal{D}}(F(\mathbf{x})) = (E_{\mathcal{D}}[F(\mathbf{x})] - E[D|\mathbf{X} = \mathbf{x}])^2 \tag{7.2}$$

and $V_{\mathcal{D}}(F(\mathbf{x}))$ is the variance:

$$V_{\mathcal{D}}(F(\mathbf{x})) = E_{\mathcal{D}}\left[(F(\mathbf{x}) - E_{\mathcal{D}}[F(\mathbf{x})])^2\right] \tag{7.3}$$

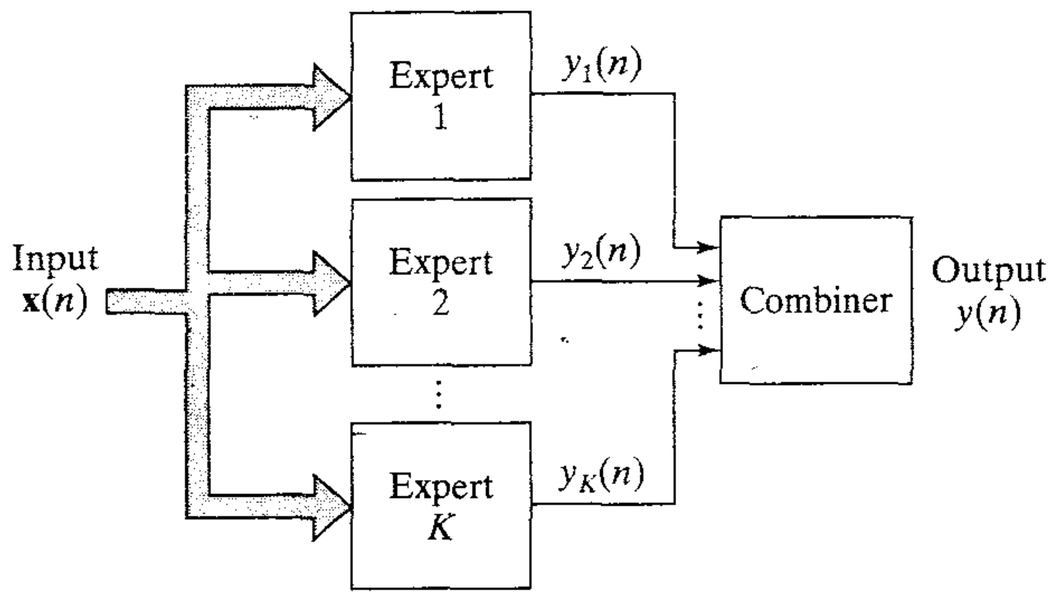


FIGURE 7.1 Block diagram of a committee machine based on ensemble-averaging.

The expectation $E_{\mathcal{D}}$ is taken over the space $\mathcal{D}$, defined as the *space encompassing the distribution of all training sets (i.e., inputs and target outputs) and the distribution of all initial conditions*.

There are different ways of individually training the expert networks in Fig. 7.1 and also different ways of combining their outputs. For the discussion presented here, we will consider the situation where the expert networks have an identical configuration, but they are trained starting from different initial conditions. For the combiner at the output of the committee machine in Fig. 7.1, we will use a simple *ensemble averager*.[2] Let $\mathcal{I}$ denote the *space of all initial conditions*. Let $F_I(\mathbf{x})$ denote the average of the input–output functions of the expert networks in Fig. 7.1 over a "representative" number of initial conditions. By analogy with Eq. (7.1), we may write

$$E_{\mathcal{I}}[(F_I(\mathbf{X}) - E[D\,|\mathbf{X} = \mathbf{x}])^2] = B_{\mathcal{I}}(F(\mathbf{x})) + V_{\mathcal{I}}(F(\mathbf{x})) \tag{7.4}$$

where $B_{\mathcal{I}}(F(\mathbf{x}))$ is the squared bias defined over the space $\mathcal{I}$:

$$B_{\mathcal{I}}(F(\mathbf{x})) = (E_{\mathcal{I}}[F_I(\mathbf{x})] - E[D\,|\mathbf{X} = \mathbf{x}])^2 \tag{7.5}$$

and $V_{\mathcal{I}}(F(\mathbf{x}))$ is the corresponding variance:

$$V_{\mathcal{I}}(F(\mathbf{x})) = E_{\mathcal{I}}[(F_I(\mathbf{x}) - E_{\mathcal{I}}[F(\mathbf{x})])^2] \tag{7.6}$$

The expectation $E_{\mathcal{I}}$ is taken over the space $\mathcal{I}$.

From the definition of space $\mathcal{D}$, we may view it as the product of the space of initial conditions, $\mathcal{I}$, and the *remnant space* denoted by $\mathcal{D}'$. Accordingly, we may write, again by analogy with Eq. (7.1):

$$E_{\mathcal{D}'}[(F_I(\mathbf{x}) - E[D\,|\mathbf{X} = \mathbf{x}])^2] = B_{\mathcal{D}'}(E_I(\mathbf{x})) + V_{\mathcal{D}'}(F_I(\mathbf{x})) \tag{7.7}$$

where $B_{\mathcal{D}'}(F(\mathbf{x}))$ is the squared bias defined over the remnant space $\mathcal{D}'$:

$$B_{\mathcal{D}'}(F_I(\mathbf{x})) = (E_{\mathcal{D}'}[F_I(\mathbf{x})] - E[D\,|\mathbf{X} = \mathbf{x}])^2 \tag{7.8}$$

and $V_{\mathcal{D}'}(F_I(\mathbf{x}))$ is the corresponding variance:

$$V_{\mathcal{D}'}(F_I(\mathbf{x})) = E_{\mathcal{D}'}[(F_I(\mathbf{x}) - E_{\mathcal{D}'}[F_I(\mathbf{x})])^2] \tag{7.9}$$

From the definitions of spaces $\mathcal{D}$, $\mathcal{I}$, and $\mathcal{D}'$, we readily see that

$$E_{\mathcal{D}'}[F_I(\mathbf{x})] = E_{\mathcal{D}}[F(\mathbf{x})] \tag{7.10}$$

It follows therefore that Eq. (7.8) may be rewritten in the equivalent form:

$$\begin{aligned} B_{\mathcal{D}'}(F_I(\mathbf{x})) &= (E_{\mathcal{D}}[F(\mathbf{x})] - E[D\,|\mathbf{X} = \mathbf{x}])^2 \\ &= B_{\mathcal{D}}(F(\mathbf{x})) \end{aligned} \tag{7.11}$$

Consider next the variance $V_{\mathcal{D}'}(F_I(\mathbf{x}))$ of Eq. (7.9). Since the variance of a random variable is equal to the mean-square value of that random variable minus its bias squared, we may equivalently write

$$\begin{aligned} V_{\mathcal{D}'}(F_I(\mathbf{x})) &= E_{\mathcal{D}'}[(F_I(\mathbf{x}))^2] - (E_{\mathcal{D}'}[F_I(\mathbf{x})])^2 \\ &= E_{\mathcal{D}'}[(F_I(\mathbf{x}))^2] - (E_{\mathcal{D}}[F(\mathbf{x})])^2 \end{aligned} \tag{7.12}$$

where in the last line we have made use of Eq. (7.10). Similarly, we may redefine Eq. (7.3) in the equivalent form

$$V_{\mathscr{D}}(F_I(\mathbf{x})) = E_{\mathscr{D}}[(F(\mathbf{x}))^2] - (E_{\mathscr{D}}[F(\mathbf{x})])^2 \tag{7.13}$$

Note that the mean-square value of the function $F(\mathbf{x})$ over the entire space $\mathscr{D}$ is destined to be equal to or greater than the mean-square value of the ensemble-averaged function $F_I(\mathbf{x})$ over the remnant space $\mathscr{D}'$. That is,

$$E_{\mathscr{D}}[F(\mathbf{x})^2] \geq E_{\mathscr{D}'}[(F_I(\mathbf{x}))^2]$$

In light of this inequality, comparison of Eqs. (7.12) and (7.13) immediately reveals that

$$V_{\mathscr{D}'}(F_I(\mathbf{x})) \leq V_{\mathscr{D}}(F(\mathbf{x})) \tag{7.14}$$

Thus, from Eqs. (7.11) and (7.14) we draw two conclusions (Naftaly et al., 1997):

1. The bias of the ensemble-averaged function $F_I(\mathbf{x})$, pertaining to the committee machine of Fig. 7.1, is exactly the same as that of the function $F(\mathbf{x})$ pertaining to a single neural network.
2. The variance of the ensemble-averaged function $F_I(\mathbf{x})$ is less than that of the function $F(\mathbf{x})$.

These theoretical findings point to a training strategy for reducing the overall error produced by a committee machine due to *varying initial conditions* (Naftaly et al., 1997): The constituent experts of the machine are purposely *overtrained*, the use of which is justified on the following grounds. Insofar as the individual experts are concerned, the bias is reduced at the cost of variance. Subsequently, however, the variance is reduced by ensemble averaging the experts over the initial conditions, leaving the bias unchanged.

7.3 COMPUTER EXPERIMENT I

In this computer experiment on the ensemble-averaging method, we revisit the pattern classification problem considered in the previous three chapters. The problem pertains to the classification of two overlapping two-dimensional Gaussian distributions. The two distributions have different mean vectors and different variances. The statistics of distribution 1 (class $\mathscr{C}_1$) are

$$\boldsymbol{\mu}_1 = [0, 0]^T$$
$$\sigma_1^2 = 1$$

The statistics of distribution 2 (class $\mathscr{C}_2$) are

$$\boldsymbol{\mu}_2 = [2, 0]^T$$
$$\sigma_2^2 = 4$$

Scatter plots for these two distributions are shown in Fig. 4.13.

The two classes are assumed to be equiprobable. The costs for misclassifications are assumed to be equal, and the costs for correct classifications are assumed to be zero. On this basis, the (optimum) Bayes classifier achieves a probability of correct

classification $p_c = 81.51$ percent. Details of this calculation are also presented in Chapter 4.

In the computer experiment described in Chapter 4 we were able to achieve a probability of correct classification close to 80 percent using a multilayer perceptron with two hidden neurons and trained using the back-propagation algorithm. In this experiment we study a committee machine composed as follows:

- Ten experts.
- Each expert made up of a multilayer perceptron with two hidden neurons.

All the experts were individually trained using the back-propagation algorithm. The parameters used in the algorithm were

Learning-rate parameter, $\eta = 0.1$
Momentum constant, $\alpha = 0.5$

The size of the training sample was 500 patterns. All the experts were trained on the same data set, but were initialized differently. In particular, the initial values of the synaptic weights and thresholds were picked at random from a uniform distribution inside the range $[-1, 1]$.

Table 7.1 presents a summary of the classification performances of the 10 experts trained on 500 patterns using the test set. The probability of correct classification obtained by simply taking the arithmetic average of the 10 results presented in Table 7.1 is $p_{c,\text{av}} = 79.37$ percent. On the other hand, by using the ensemble-averaging method, that is, by simply summing the individual outputs of the 10 experts and then computing the probability of correct classification, we obtained the result: $p_{c,\text{ens}} = 80.27$ percent. This result represents an improvement of 0.9 percent over $p_{c,\text{av}}$. The improvement of $p_{c,\text{ens}}$ over $p_{c,\text{av}}$ was maintained in all the trials of the experiment. The classification results were all computed using 32,000 test patterns.

TABLE 7.1 Classification Performances of Individual Experts Used in a Committee Machine

Expert	Correct classification percentage
Net 1	80.65
Net 2	76.91
Net 3	80.06
Net 4	80.47
Net 5	80.44
Net 6	76.89
Net 7	80.55
Net 8	80.47
Net 9	76.91
Net 10	80.38

Summarizing the results of this experiment, we may say: The classification performance is improved by overtraining the individual multilayer perceptrons (experts), summing their individual numerical outputs to produce the overall output of the committee machine, and then making a decision.

7.4 BOOSTING

As mentioned in the introduction, boosting is another method that belongs to the "static" class of committee machines. Boosting is quite different from ensemble averaging. In a committee machine based on ensemble averaging, all the experts in the machine are trained on the same data set; they may differ from each other in the choice of initial conditions used in network training. By contrast, in a boosting machine the experts are trained on data sets with entirely different distributions; it is a general method that can be used to improve the performance of *any* learning algorithm.

Boosting[3] can be implemented in three fundamentally different ways:

1. *Boosting by filtering.* This approach involves filtering the training examples by different versions of a weak learning algorithm. It assumes the availability of a large (in theory, infinite) source of examples, with the examples being either discarded or kept during training. An advantage of this approach is that it allows for a small memory requirement compared to the other two approaches.
2. *Boosting by subsampling.* This second approach works with a training sample of *fixed* size. The examples are "resampled" according to a given probability distribution during training. The error is calculated with respect to the fixed training sample.
3. *Boosting by reweighting.* This third approach also works with a fixed training sample, but it assumes that the weak learning algorithm can receive "weighted" examples. The error is calculated with respect to the weighted examples.

In this section we describe two different boosting algorithms. One algorithm, due to Schapire (1990), belongs to approach 1. The other algorithm, known as AdaBoost due to Freund and Schapire (1996a, 1996b), belongs to approach 2.

Boosting by Filtering

The original idea of boosting described in Schapire (1990) is rooted in a *distribution-free* or *probably approximately correct (PAC) model of learning*. From the discussion of PAC learning presented in Chapter 2, we recall that a *concept* is a Boolean function in some domain of instances that contains encodings of all objects of interest. In PAC learning, a learning machine tries to identify an unknown binary concept on the basis of randomly chosen examples of the concept. To be more specific, the goal of the learning machine is to find a hypothesis or prediction rule with an error rate of at most ϵ, for arbitrarily small positive values of ϵ, and this should hold uniformly for all input distributions. It is for this reason that the PAC learning model is also referred to as a *strong learning model*. Since the examples are of a random nature, it is likely that the learning

machine will be unable to learn anything about the unknown concept due to the presentation of a highly unrepresentative example. We therefore require the learning model to succeed only in finding a good approximation to the unknown concept with a probability of $1 - \delta$, where δ is a small positive number.

In a variant of the PAC learning model, called a *weak learning model*, the requirement to learn an unknown concept is relaxed dramatically. The learning machine is now required to find a hypothesis with an error rate only slightly less than 1/2. When a hypothesis guesses a binary label in an entirely random manner on every example, it can be right or wrong with equal probability. That is, it achieves an error rate of exactly 1/2. It follows therefore that a weak learning model has to perform only slightly better than random guessing. The notion of weak learnability was introduced by Kearns and Valiant (1989), who posed the *hypothesis boosting problem* that is embodied in the following question:

Are the notions of strong and weak learning equivalent?

In other words, is any concept class that is weakly learnable also strongly learnable? This question, which is perhaps surprising, was answered in the affirmative by Schapire (1990). The proof presented therein was constructive. Specifically, an algorithm was devised for directly converting a weak learning model into a strong learning model. This is achieved by modifying the distribution of examples in such a way that a strong learning model is built around a weak one.

In boosting by filtering, the committee machine consists of three experts or subhypotheses. The algorithm used to train them is called a *boosting algorithm*. The three experts are arbitrarily labeled "first," "second," and "third." The three experts are individually trained as follows:

1. The first expert is trained on a set consisting of N_1 examples.
2. The trained first expert is used to *filter* another set of examples by proceeding in the following manner:
   - Flip a fair coin; this in effect simulates a random guess.
   - If the result is *heads*, pass new patterns through the first expert and discard correctly classified patterns until a pattern is misclassified. That misclassified pattern is added to the training set for the second expert.
   - If the result is *tails*, do the opposite. Specifically, pass new patterns through the first expert and discard incorrectly classified patterns until a pattern is classified correctly. That correctly classified pattern is added to the training set for the second expert.
   - Continue this process until a total of N_1 examples has been filtered by the first expert. This set of filtered examples constitutes the training set for the second expert.

By following this coin flipping procedure, it is ensured that if the first expert is tested on the second set of examples, it would have an error rate of 1/2. In other words, the second set of N_1 examples available for training the second expert has a distribution entirely different from the first set of N_1 examples used to train the first expert. In this way, the second expert is forced to learn a distribution different from that learned by the first expert.

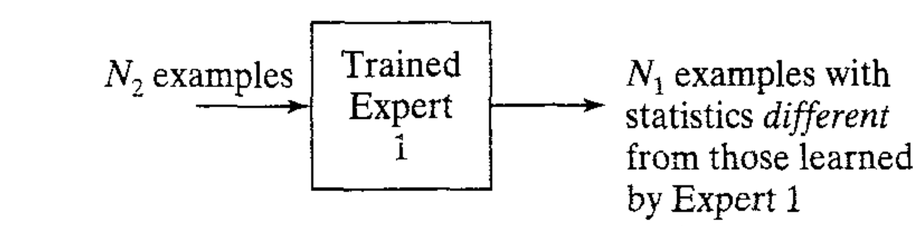
3. Once the second expert has been trained in the usual way, a third training set is formed for the third expert by proceeding in the following manner:
   - Pass a new pattern through both the first and second experts. If the two experts agree in their decisions, discard that pattern. If, on the other hand, they disagree, the pattern is added to the training set for the third expert.
   - Continue with this process until a total of N_1 examples has been filtered jointly by the first and second experts. This set of jointly filtered examples constitutes the training set for the third expert.

   The third expert is then trained in the usual way, and the training of the entire committee machine is thereby completed.

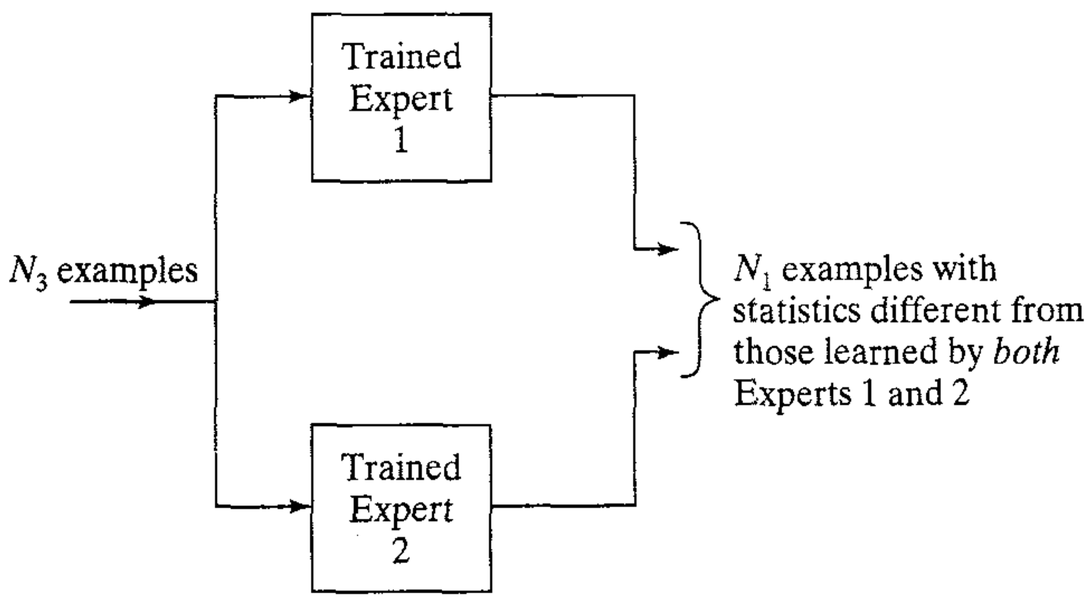
This three-point filtering procedure is illustrated in Fig. 7.2.

Let N_2 denote the number of examples that must be filtered by the first expert to obtain the training set of N_1 examples for the second expert. Note that N_1 is fixed, and N_2 depends on the generalization error rate of the first expert. Let N_3 denote the number of examples that must be jointly filtered by the first and second experts to obtain the training set of N_1 examples for the third expert. With N_1 examples also needed to train the first expert, the total size of data set needed to train the entire committee machine is $N_4 = N_1 + N_2 + N_3$. However, the computational cost is based on $3N_1$ examples because N_1 is the number of examples actually used to train each of the three experts. We may therefore say that the boosting algorithm described herein is indeed "smart" in the sense that the committee machine requires a large set of examples for its operation, but only a subset of that data set is used to perform the actual training.

Another noteworthy point is that the filtering operation performed by the first expert and the joint filtering operation performed by the first and second experts make the second and third experts, respectively, focus on "hard-to-learn" parts of the distribution.



(a) Filtering of examples performed by Expert 1



(b) Filtering of examples performed by Experts 2 and 3

FIGURE 7.2 Illustration of boosting by filtering.

In the theoretical derivation of the boosting algorithm originally presented in Schapire (1990), simple *voting* was used to evaluate the performance of the committee machine on test patterns not seen before. Specifically, a test pattern is presented to the committee machine. If the first and second experts in the committee machine agree in their respective decisions, that class label is used. Otherwise, the class label discovered by the third expert is used. However, in experimental work presented in Drucker et al. (1993, 1994), it has been determined that *addition* of the respective outputs of the three experts yields a better performance than voting. For example, in the optical character recognition (OCR) problem, the addition operation is performed simply by adding the "digit 0" outputs of the three experts, and likewise for the other nine digit outputs.

Suppose that the three experts (i.e., subhypotheses) have an error rate of $\epsilon < 1/2$ with respect to the distributions on which they are individually trained; that is, all three of them are weak learning models. In Schapire (1990), it is proved that the overall error rate of the committee machine is *bounded* by

$$g(\epsilon) = 3\epsilon^2 - 2\epsilon^3 \tag{7.15}$$

The bound $g(\epsilon)$ is plotted versus ϵ in Fig. 7.3. From this figure we see that the bound is significantly smaller than the original error rate ϵ. By applying the boosting algorithm recursively, the error rate can be made arbitrarily small. In other words, a weak learning model, which performs only slightly better than random guessing, is converted into a strong learning model. It is in that sense that we may say that strong and weak learnability are indeed equivalent.

AdaBoost

A practical limitation of boosting by filtering is that it often requires a large training sample. This limitation can be overcome by using another boosting algorithm called *AdaBoost* (Freund and Schapire, 1996a, 1996b), which belongs to boosting by resampling. The sampling framework of AdaBoost is the natural framework of batch learning; most importantly, it permits the training data to be reused.

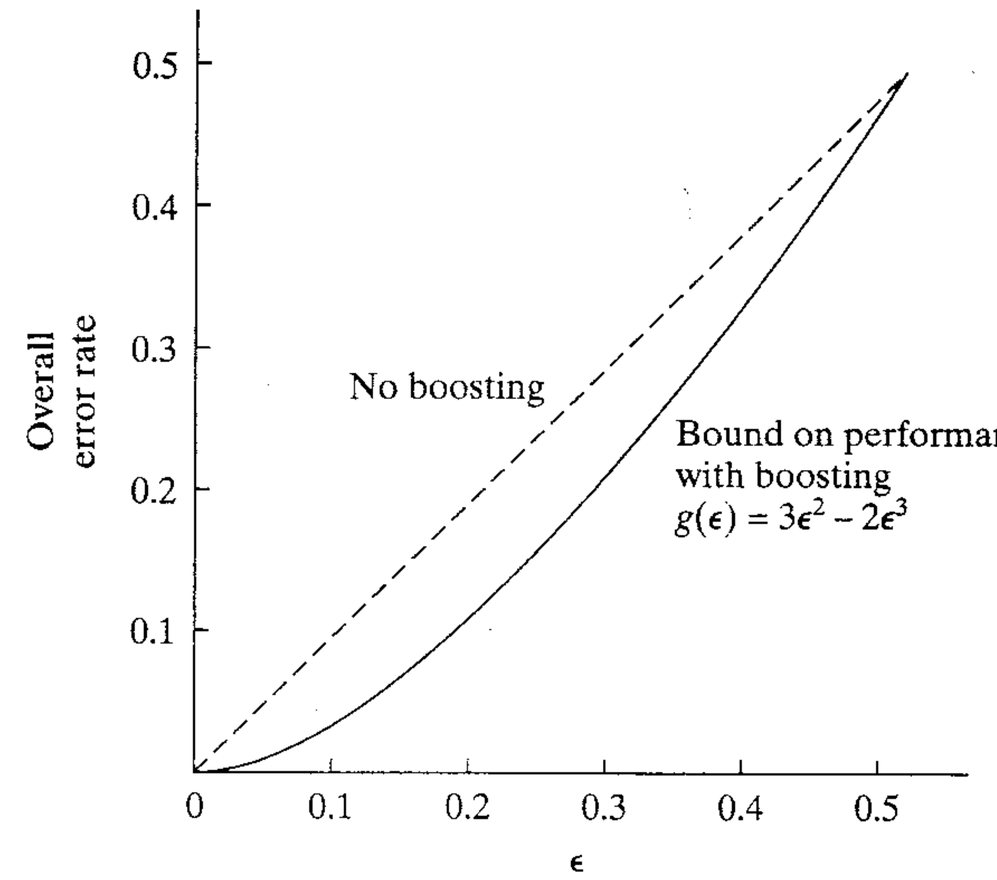


FIGURE 7.3 Graph of Eq. (7.15) for boosting by filtering.

As with the boosting by filtering algorithm, AdaBoost has access to a weak learning model. The goal of the new algorithm is to find a final mapping function or hypothesis with low error rate relative to a given distribution $\mathcal{D}$ over the labeled training examples. It differs from other boosting algorithms in two respects:

- AdaBoost adjusts *adaptively* to the errors of the weak hypothesis returned by the weak learning model, hence the name of the algorithm.
- The bound on performance of AdaBoost depends only on the performance of the weak learning model on those distributions that are actually generated during the learning process.

AdaBoost operates as follows. On iteration n, the boosting algorithm provides the weak learning model with a distribution $\mathcal{D}_n$ over the training sample $\mathcal{T}$. In response, the weak learning model computes a hypothesis $\mathcal{F}_n : \mathbf{X} \rightarrow Y$ that correctly classifies a fraction of the training examples. The error is measured with respect to the distribution $\mathcal{D}_n$. The process continues for T iterations, and finally the boosting machine combines the hypotheses $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_T$ into a single final hypothesis $\mathcal{F}_{\text{fin}}$.

To calculate (1) the distribution $\mathcal{D}_n$ on iteration n, and (2) the final hypothesis $\mathcal{F}_{\text{fin}}$, the simple procedure summarized in Table 7.2 is used. The initial distribution $\mathcal{D}_1$ is uniform over the training sample $\mathcal{T}$, as shown by

$$\mathcal{D}_1(i) = \frac{1}{n} \qquad \text{for all } i$$

Given the distribution $\mathcal{D}_n$ and weak hypothesis $\mathcal{F}_n$ on iteration n of the algorithm, the next distribution $\mathcal{D}_{n+1}$ is computed by multiplying the weight of example i by some number $\beta_n \in [0, 1)$ if $\mathcal{F}_n$ classifies the input vector $\mathbf{x}_i$ correctly; otherwise, the weight is left unchanged. The weights are then renormalized by dividing by the normalizing constant Z_n. In effect, the "easy" examples in the training set $\mathcal{T}$ that are correctly classified by many of the previous weak hypotheses are given lower weights, whereas the "hard" examples that are often misclassified are given higher weights. Thus the AdaBoost algorithm focuses the most weight on those examples that appear to be hardest for it to classify.

As for the final hypothesis $\mathcal{F}_{\text{fin}}$, it is computed as a weighted vote (i.e., weighted linear threshold) of the weak hypotheses $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_T$. That is, for a given input vector $\mathbf{x}$, the final hypothesis $\mathcal{F}_{\text{fin}}$ outputs the label d that maximizes the sum of the weights of the weak hypotheses predicting that label. The weight of hypothesis $\mathcal{F}_n$ is defined to be $\log(1/\beta_n)$, so that greater weight is given to hypotheses with lower error.

The important theoretical property of AdaBoost is stated in the following theorem (Freund and Schapire, 1996a):

> Suppose the weak learning model, when called by AdaBoost, generates hypotheses with errors $\epsilon_1, \epsilon_2, \ldots, \epsilon_T$, where the error ϵ_n on iteration n of the AdaBoost algorithm is defined by
>
> $$\epsilon_n = \sum_{i:\mathcal{F}_n(\mathbf{x}_i) \neq d_i} \mathcal{D}_n(i)$$
>
> Assume that $\epsilon_n \leq 1/2$, and let $\gamma_n = 1/2 - \epsilon_n$. Then the following upper bound holds on the error of the final hypothesis:
>
> $$\frac{1}{N} \left| \{ i : \mathcal{F}_{\text{fin}}(\mathbf{x}_i) \neq d_i \} \right| \leq \prod_{n=1}^{T} \sqrt{1 - 4\gamma_n^2} \leq \exp\left(-2 \sum_{n=1}^{T} \gamma_n^2 \right) \tag{7.16}$$

TABLE 7.2 Summary of AdaBoost

Input:
- Training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$
- Distribution $\mathcal{D}$ over the N labeled examples
- Weak learning model
- Integer T specifying the number of iterations of the algorithm

Initialization: Set $\mathcal{D}_1(i)=1/N$ for all i

Computation: Do the following for $n = 1, 2, \ldots, T$:

1. Call the weak learning model, providing it with the distribution $\mathcal{D}_n$.
2. Get back hypothesis $\mathcal{F}_n : \mathbf{X} \to Y$
3. Calculate the error of hypothesis $\mathcal{F}_n$:

$$\epsilon_n = \sum_{i:\mathcal{F}_n(\mathbf{x}_i) \neq d_i} \mathcal{D}_n(i)$$

4. Set $\beta_n = \dfrac{\epsilon_n}{1 - \epsilon_n}$
5. Update the distribution $\mathcal{D}_n$:

$$\mathcal{D}_{n+1}(i) = \frac{\mathcal{D}_n(i)}{Z_n} \times \begin{cases} \beta_n & \text{if } \mathcal{F}_n(\mathbf{x}_i) = d_i \\ 1 & \text{otherwise} \end{cases}$$

where Z_n is a normalization constant (chosen so that $\mathcal{D}_{n+1}(i)$ is a probability distribution).

Output: The final hypothesis is

$$\mathcal{F}_n(\mathbf{x}) = \arg\max_{d \in \mathcal{D}} \sum_{n:\mathcal{F}_n(\mathbf{x})=d} \log \frac{1}{\beta_n}$$

This theorem shows that if the weak hypotheses constructed by the weak learning model consistently have error only slightly better than 1/2, then the training error of the final hypothesis $\mathcal{F}_{\text{fin}}$ drops to zero exponentially fast. However, this does not mean that the generalization error on test data is necessarily small. Experiments presented in Freund and Schapire (1996a) indicate two things. First, the theoretical bound on the training error is often weak. Second, the generalization error tends to be much better than what the theory would suggest.

Table 7.2 presents a summary of AdaBoost for a binary classification problem.

When the number of possible classes (labels) is $M > 2$, the boosting problem becomes more intricate because the probability that random guessing gives the correct label is $1/M$, which is now smaller than $1/2$. For boosting to be able to use any hypothesis that is slightly better than random guessing in such a situation, we need to somehow change the algorithm and the definition of what is meant to be a "weakly learning" algorithm. Ways of invoking that change are described in Freund and Schapire (1997) and Schapire (1997).

Error Performance

Experiments with AdaBoost reported in Breiman (1996b) show that when the training error and test error are plotted as a function of the number of boosting iterations, we often find that the test error continues to decrease after the training error has reduced

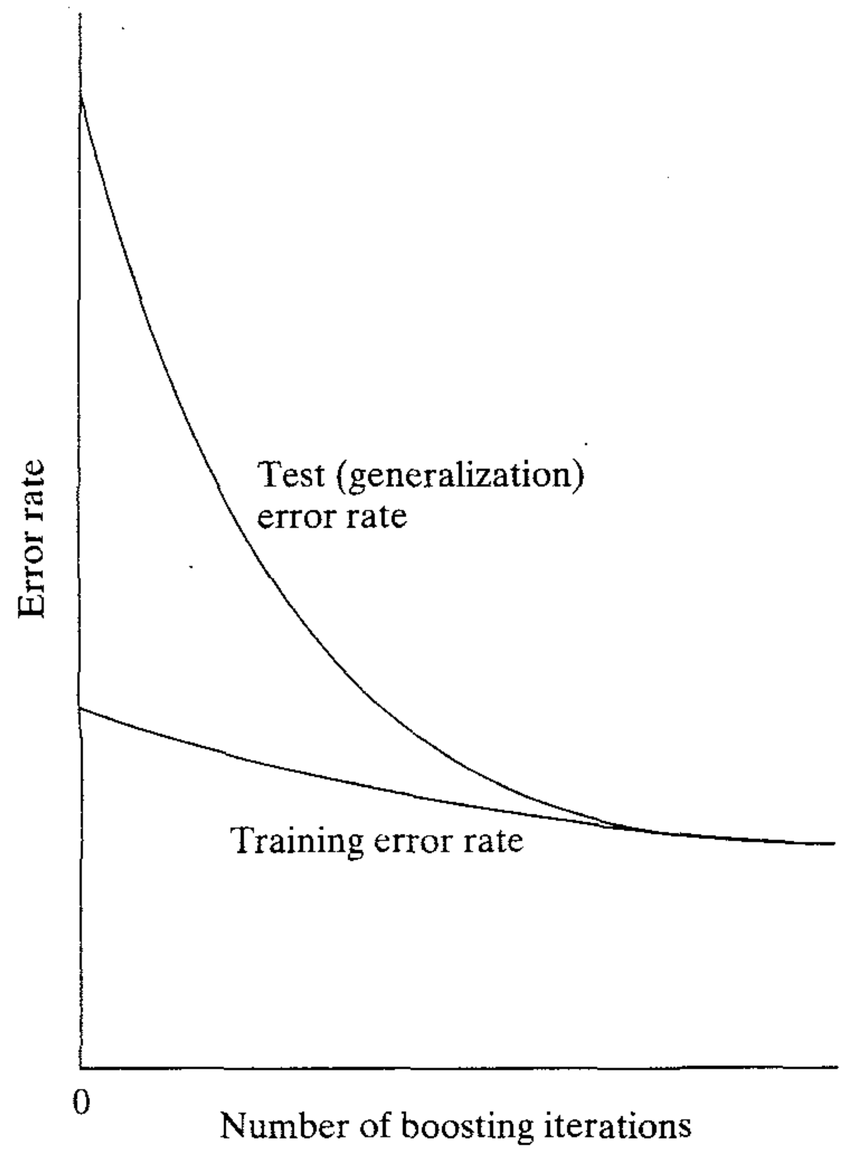


FIGURE 7.4 Conceptualized error performance of the AdaBoost algorithm.

essentially to zero. This phenomenon is illustrated in Fig. 7.4. A similar result was reported earlier in Drucker et al. (1994) for boosting by filtering.

The phenomenon displayed in Fig. 7.4 is very surprising in light of what we know about the generalization performance of a single neural network. From Chapter 4 we recall that in the case of a multilayer perceptron trained with the back-propagation algorithm, the error on test (validation) data decreases, reaches a minimum, and then increases due to overfitting; see Fig. 4.20. The behavior displayed in Fig. 7.4 is remarkably different in that as the networks become more and more complex through increased training, the generalization error continues to decrease. Such a phenomenon appears to contradict *Occam's razor*, which states that a learning machine should be as simple as possible in order to achieve a good generalization performance.

In Schapire et al. (1997), an explanation is given for this phenomenon as it applies to AdaBoost. The key idea of the analysis presented therein is that in evaluating the generalization error produced by a boosting machine, not only the training error should be considered but also the *confidence* of classifications. The analysis presented therein reveals a relation between boosting and support vector machines; support vector machines are considered in the previous chapter. In particular, the classification *margin*, for example, is defined as the difference between the weight assigned to the correct label pertaining to that example and the maximal weight assigned to any single incorrect label. From this definition, it is easy to see that the margin is a number in the range $[-1, 1]$ and that an example is correctly classified if and only if its margin is positive. Thus Schapire et al. show that the phenomenon observed in Fig. 7.4 is indeed related to the distribution of margins of the training examples with respect to the generated voting classification error. It should be emphasized again that

the margin analysis presented in Schapire et al. (1997) is specific to AdaBoost and does not apply to other boosting algorithms.

7.5 COMPUTER EXPERIMENT II

In this experiment we explore the boosting by filtering algorithm to solve a fairly difficult pattern classification task. The classification problem is two-dimensional, involving nonconvex decision regions, as shown in Fig. 7.5. One class of patterns consists of data points lying inside the region labeled $\mathscr{C}_1$, and the other class of patterns consists of data points inside the region labeled $\mathscr{C}_2$. The requirement is to design a committee machine that decides whether a test pattern belongs to class $\mathscr{C}_1$ or to class $\mathscr{C}_2$.

The committee machine used to solve this problem consists of three experts. Each expert consists of a 2-5-2 multilayer perceptron that has two input nodes, five hidden neurons, and two output neurons. The back-propagation algorithm was used to perform the training. Figure 7.6 displays the scatter plots of the data used to train the three experts. The data shown in Fig. 7.6a were used to train expert 1. The data shown in Fig. 7.6b were filtered by expert 1 after its training was completed; this data set was used to train expert 2. The data shown in Fig. 7.6c were filtered by the combined action of trained experts 1 and 2; this data set was used to train expert 3. The size of training sample for each expert was $N_1 = 1000$ patterns. Examining these three figures we observe:

- The training data for expert 1 in Fig. 7.6a are uniformly distributed.
- The training data for expert 2 in Fig. 7.6b exhibit concentrations of data points in areas labeled A and B that are seemingly difficult for the first expert to classify. The number of data points in these two regions is equal to the number of the correctly classified points.

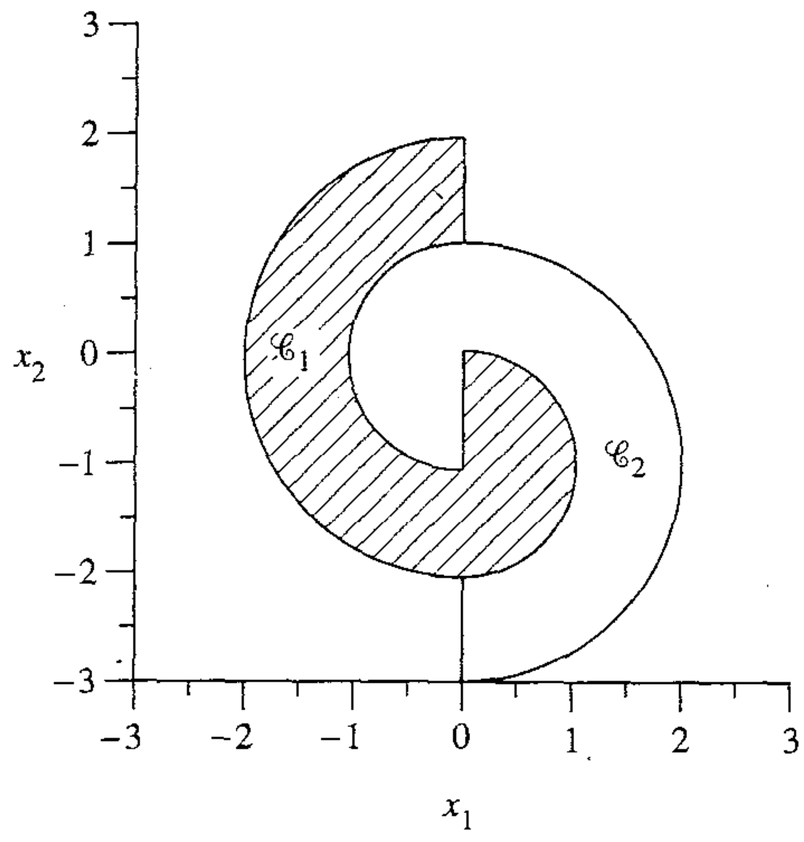
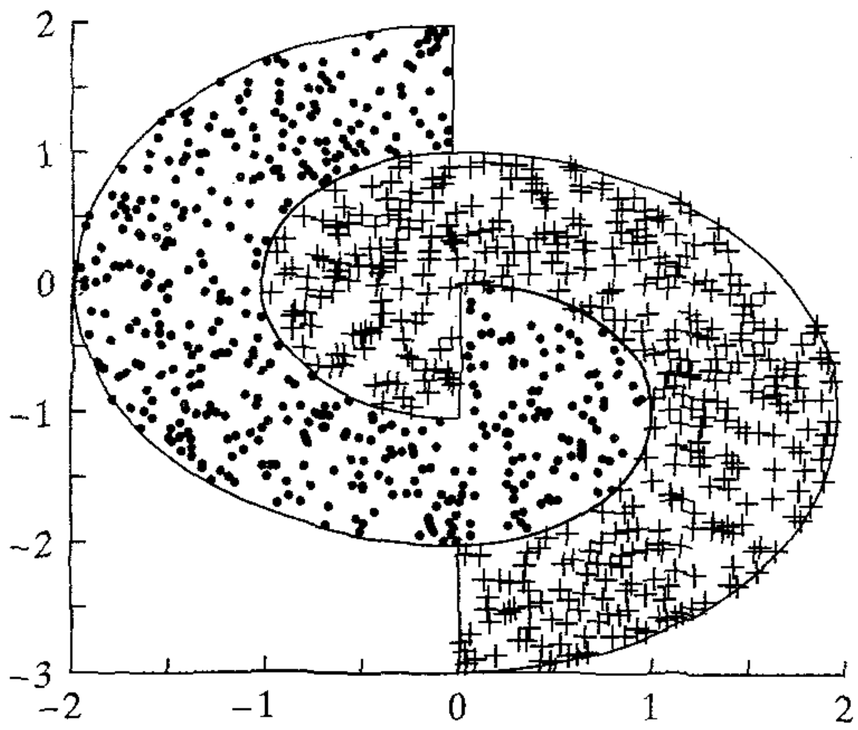
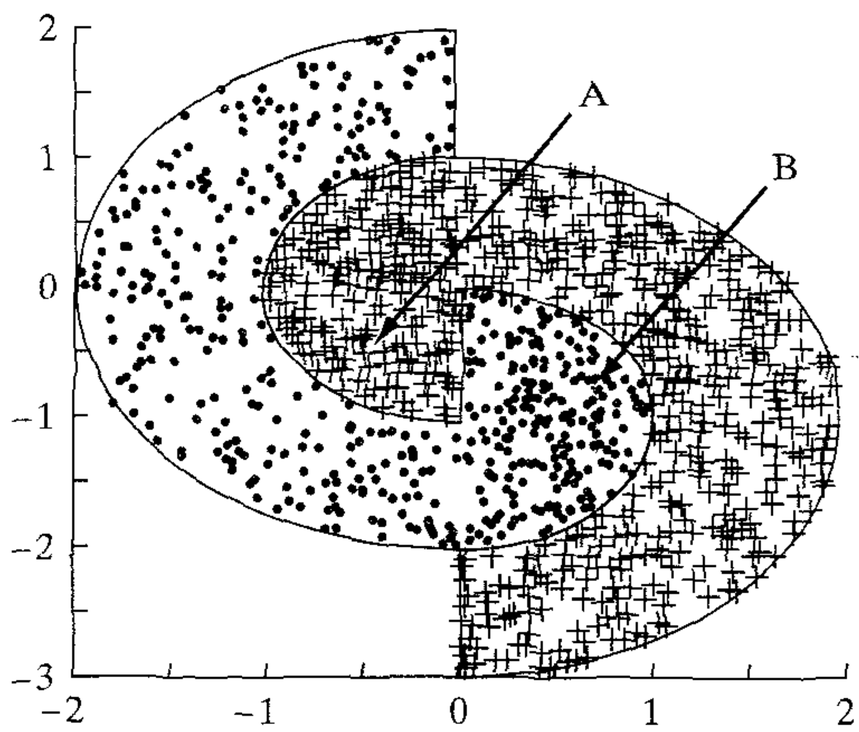


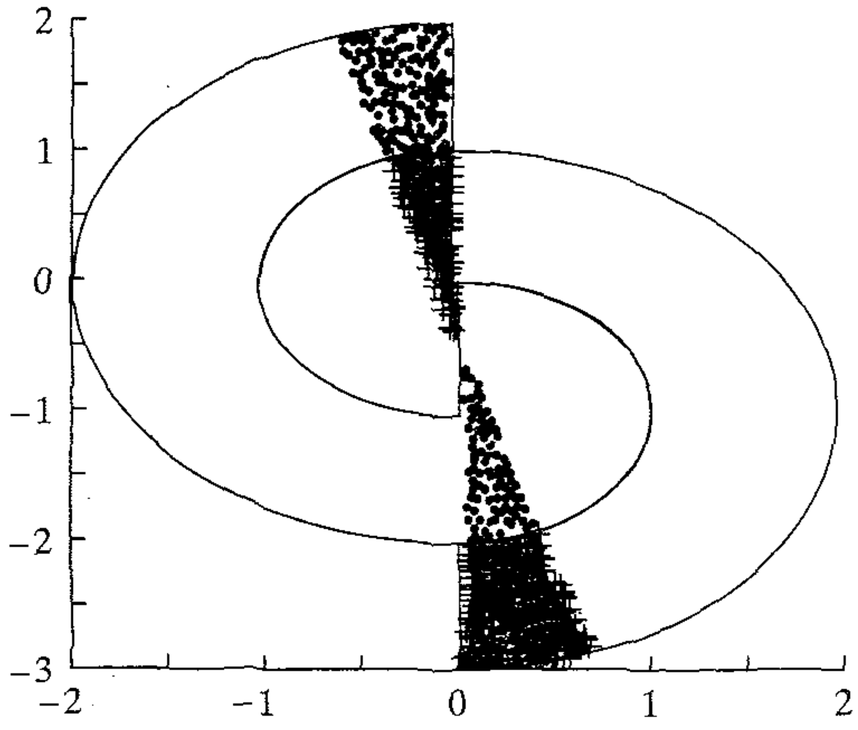
FIGURE 7.5 Pattern configurations for experiment on boosting.



(a)



(b)



(c)

FIGURE 7.6 Scatter plots for expert training in computer experiment on boosting: (a) Expert 1. (b) Expert 2. (c) Expert 3.

- The training data for expert 3 in Fig. 7.6c exhibit an even greater concentration of data points seemingly difficult for both experts 1 and 2 to classify.

Figures 7.7a, 7.7b, and 7.7c display the decision boundaries formed by experts 1, 2, and 3, respectively. Figure 7.7d displays the overall decision boundary formed by the combined action of all three experts, which is obtained by simply summing their individual outputs. Note that the difference between the decision regions of Figs. 7.7a and 7.7b pertaining to experts 1 and 2 defines the distribution of data points in Fig. 7.6c used to train expert 3.

The probabilities of correct classification for the three experts on test data were:

Expert 1 : 75.15 percent
Expert 2 : 71.44 percent
Expert 3 : 68.90 percent

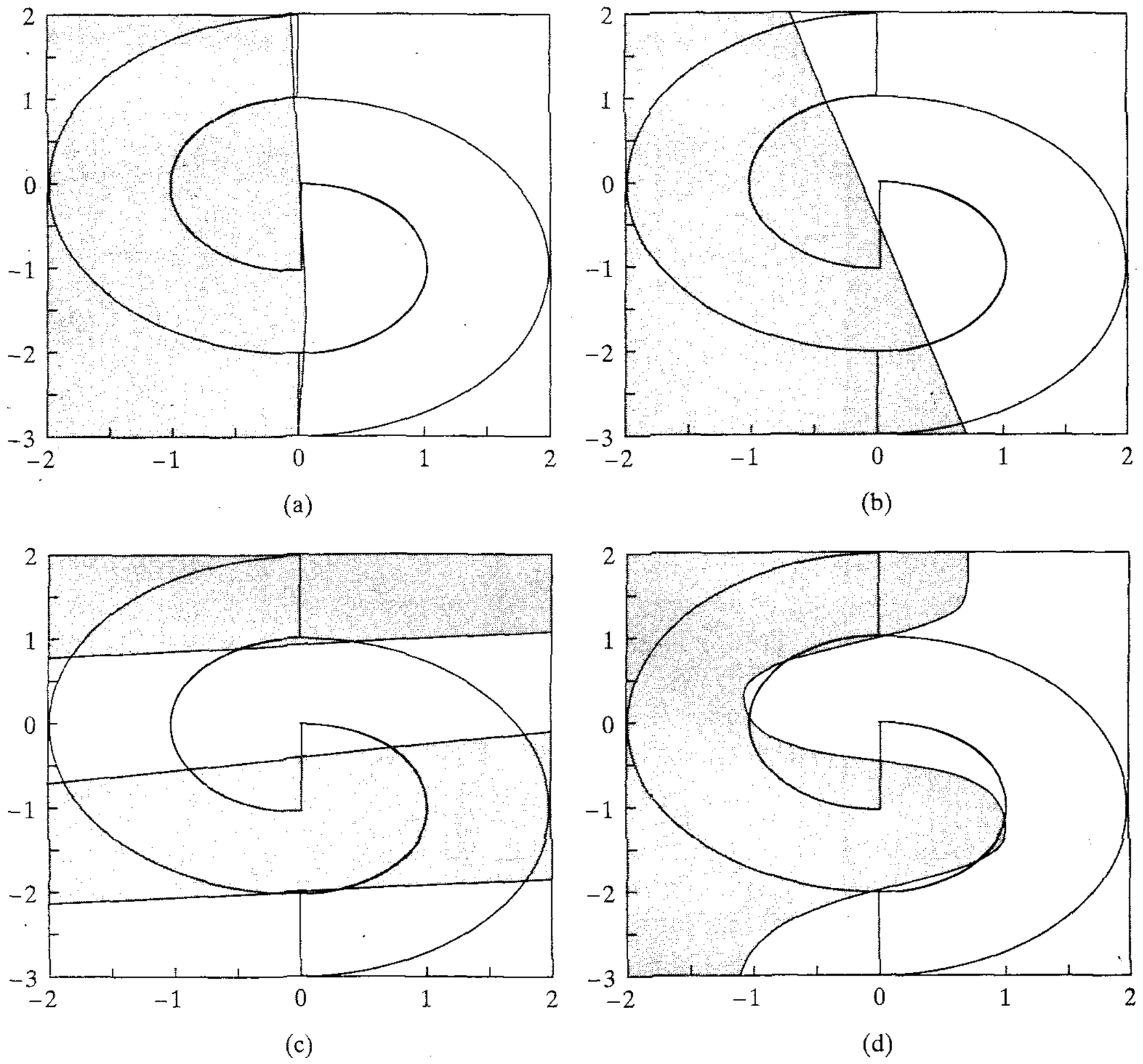


FIGURE 7.7 Decision boundaries formed by the different experts in the boosting experiment. (a) Expert 1. (b) Expert 2. (c) Expert 3. (d) Entire committee machine.

The overall probability of correct classification for the entire committee machine was 91.79 percent, which was computed using 32,000 patterns for test data. The overall decision boundary constructed by the boosting algorithm for the three experts shown in Fig. 7.7d is further evidence for this good classification performance.

7.6 ASSOCIATIVE GAUSSIAN MIXTURE MODEL

In the second part of the chapter, beginning with this section, we study the second class of committee machines, namely dynamic structures. The term "dynamic" is used here in the sense that integration of knowledge acquired by the experts is accomplished under the action of the input signal.

To begin the discussion, consider a modular network in which the learning process proceeds by fusing self-organized and supervised forms of learning in a seam-

less fashion. The experts are technically performing supervised learning in that their individual outputs are combined to model the desired response. There is, however, a sense in which the experts are also performing self-organized learning; that is, they self-organize to find a good partitioning of the input space so that each expert does well at modeling its own subspace, and as a whole group they model the input space well.

In the learning scheme just described, there is a point of departure from the schemes considered in the previous three chapters in that a specific model is assumed for generating the training data.

Probabilistic Generative Model

To fix ideas, consider a *regression* problem in which a regressor $\mathbf{x}$ produces a response denoted by random variable D; a realization of this random variable is denoted by d. Without loss of generality, we have adopted a scalar form of regression, merely to simplify the presentation. Specifically, we assume that the generation of response d is governed by the following probabilistic model (Jordan and Jacobs, 1995):

1. An input vector $\mathbf{x}$ is picked at random from some prior distribution.
2. A particular rule, say the kth rule, is selected in accordance with the conditional probability $P(k|\mathbf{x}, \mathbf{a}^{(0)})$, given $\mathbf{x}$ and some parameter vector $\mathbf{a}^{(0)}$.
3. For rule k, $k = 1, 2, \ldots, K$, the model response d is linear in $\mathbf{x}$, with an additive error ϵ_k modeled as a Gaussian distributed random variable with zero mean and unit variance:

$$E[\epsilon_k] = 0 \qquad \text{for all } k \tag{7.17}$$

and

$$\text{var}[\epsilon_k] = 1 \qquad \text{for all } k \tag{7.18}$$

Under point 3, the unity variance assumption is made just for didactic simplicity. In general, each expert can have a different output variance that can be learned from the training data.

The probabilistic generation of D is determined by the conditional probability $P(D = d|\mathbf{x}, \mathbf{w}_k^{(0)})$, given $\mathbf{x}$ and some parameter vector $\mathbf{w}_k^{(0)}$, for $k = 1, 2, \ldots, K$. We do not require that the probabilistic generative model just described must have a direct correspondence to a physical reality. Rather, we simply require that the probabilistic decisions embodied therein represent an *abstract* model, which with increasing precision specifies the location of the *conditional mean of response d on a nonlinear manifold* that relates the input vector to mean output (Jordan, 1994).

According to this model, the response D can be generated in K different ways, corresponding to the K choices of label k. Thus, the conditional probability of generating response $D = d$, given input vector $\mathbf{x}$, is equal to

$$P(D = d|\mathbf{x}, \boldsymbol{\theta}^{(0)}) = \sum_{k=1}^{K} P(D = d|\mathbf{x}, \mathbf{w}_k^{(0)})\, P(k|\mathbf{x}, \mathbf{a}^{(0)}) \tag{7.19}$$

where $\boldsymbol{\theta}^{(0)}$ is the *generative model parameter vector* denoting the combination of $\mathbf{a}^{(0)}$ and $\{\mathbf{w}_k^{(0)}\}_{k=1}^{K}$ The superscript 0 in $\mathbf{a}^{(0)}$ and $\mathbf{w}_k^{(0)}$ is intended to distinguish the generative model parameters from those of the mixture of experts model considered next.

Mixture of Experts Model

Consider the network configuration of Fig. 7.8, referred to as a *mixture of experts (ME) model*.[4] Specifically, it consists of K supervised modules called *expert networks* or simply *experts*, and an integrating unit called a *gating network* that performs the function of a mediator among the expert networks. It is assumed here that the different experts work best in different regions of the input space in accordance with the probabilistic generative model described, hence the need for the gating network.

With the regression problem assumed to be scalar, each expert network consists of a linear filter. Figure 7.9 shows the signal-flow graph of a single neuron constituting expert k. Thus, the output produced by expert k is the inner product of the input vector $\mathbf{x}$ and synaptic weight vector $\mathbf{w}_k$ of this neuron, as shown by

$$y_k = \mathbf{w}_k^T \mathbf{x}, \qquad k = 1, 2, \ldots, K \tag{7.20}$$

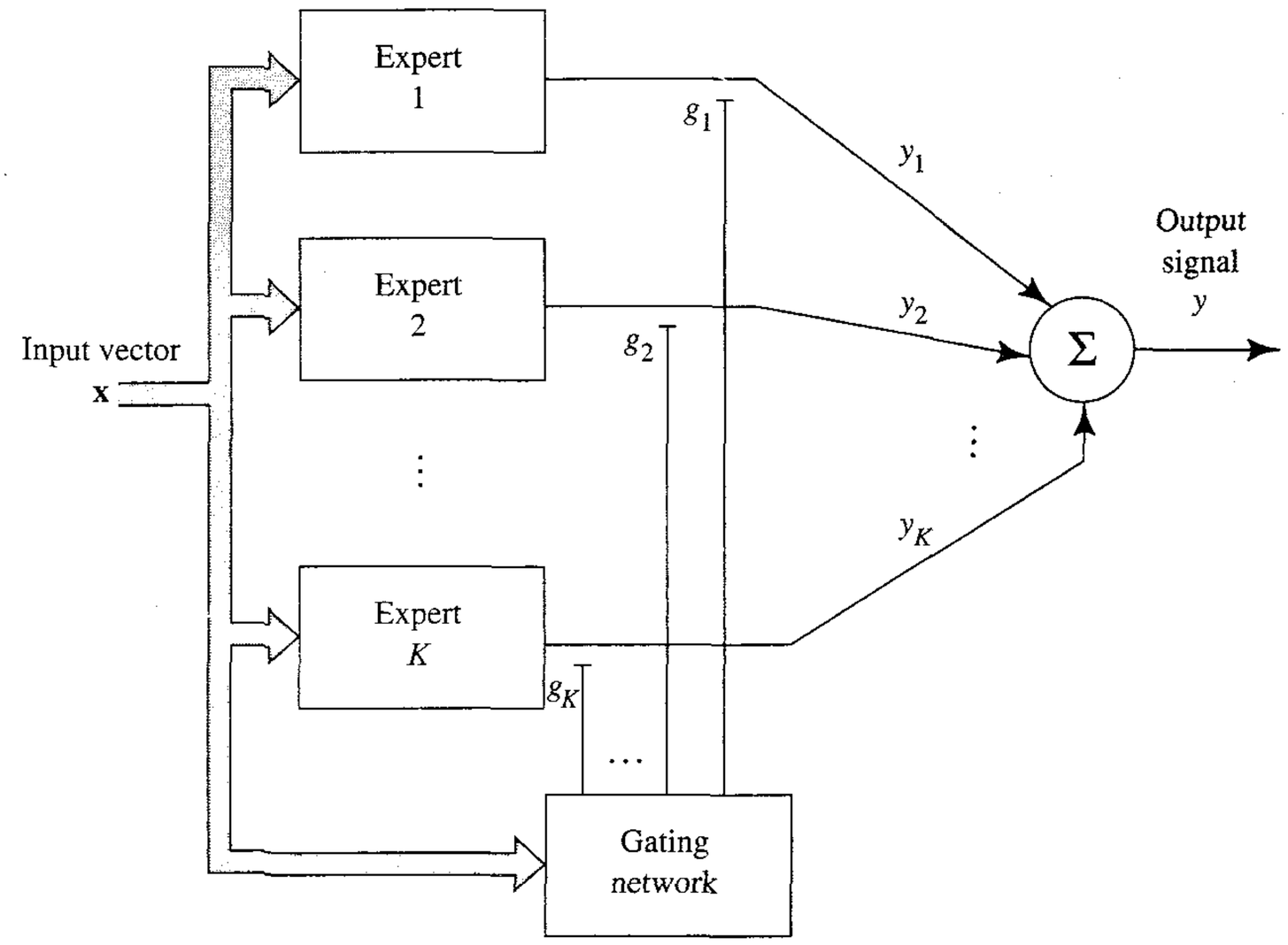


FIGURE 7.8 Block diagram of the ME model; the scalar outputs of the experts are mediated by a gating network.

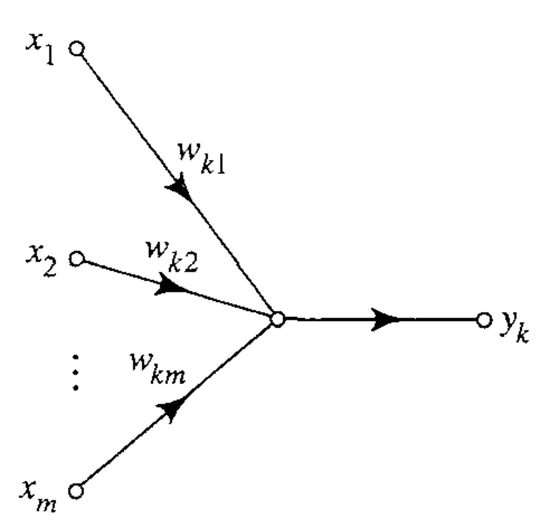


FIGURE 7.9 Signal-flow graph of a single linear neuron constituting expert k.

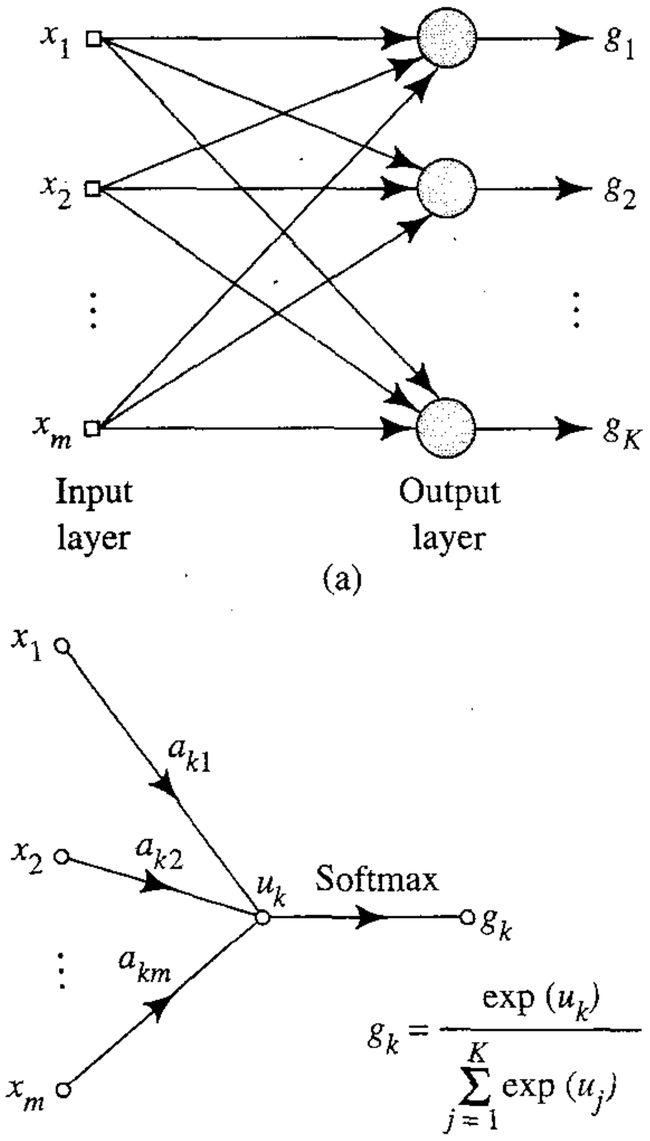


FIGURE 7.10 (a) Single layer of softmax neurons for the gating network. (b) Signal-flow graph of a softmax neuron.

The gating network consists of a single layer of K neurons, with each neuron assigned to a specific expert. Figure 7.10a shows the architectural graph of the gating network, and Fig. 7.10b shows the signal-flow graph of neuron k in that network. Unlike the experts, the neurons of the gating network are nonlinear, with their activation functions defined by

$$g_k = \frac{\exp(u_k)}{\sum_{j=1}^{K} \exp(u_j)}, \qquad k = 1, 2, \ldots, K \tag{7.21}$$

where u_k is the inner product of the input vector $\mathbf{x}$ and synaptic weight vector $\mathbf{a}_k$; that is,

$$u_k = \mathbf{a}_k^T \mathbf{x}, \qquad k = 1, 2, \ldots, K \tag{7.22}$$

The "normalized" exponential transformation of Eq. (7.21) may be viewed as a multi-input generalization of the logistic function. It preserves the rank order of its input values, and is a differentiable generalization of the "winner-takes-all" operation of picking the maximum value. For this reason, the activation function of Eq. (7.21) is referred to as *softmax* (Bridle, 1990a). Note that the linear dependence of u_k on the input $\mathbf{x}$ makes the outputs of the gating network nonlinear functions of $\mathbf{x}$.

For a probabilistic interpretation of the gating network's role, we may view it as a "classifier" that maps the input vector $\mathbf{x}$ into *multinomial probabilities* so that the different experts will be able to match the desired response (Jordan and Jacobs, 1995).

Most importantly, the use of softmax as the activation function for the gating network ensures that these probabilities satisfy the following requirements:

$$0 \leq g_k \leq 1 \qquad \text{for all } k \tag{7.23}$$

and

$$\sum_{k=1}^{K} g_k = 1 \tag{7.24}$$

Let y_k denote the output of the kth expert in response to the input vector $\mathbf{x}$. The overall output of the ME model is

$$y = \sum_{k=1}^{K} g_k y_k \tag{7.25}$$

where, as pointed out earlier, g_k is a nonlinear function of $\mathbf{x}$. Given that rule k of the probabilistic model is selected and the input is $\mathbf{x}$, an individual output y_k is treated as the conditional mean of the random variable D, as shown by

$$\begin{aligned} E[D \,|\mathbf{x},k] &= y_k \\ &= \mathbf{w}_k^T \mathbf{x}, \qquad k = 1, 2, \ldots, K \end{aligned} \tag{7.26}$$

With μ_k denoting the conditional mean of D, we may write

$$\mu_k = y_k, \qquad k = 1, 2, \ldots, K \tag{7.27}$$

The variance of D is the same as that of the error ϵ_k. Thus, invoking the use of Eq. (7.18), we may write

$$\text{var}[D \,|\mathbf{x},k] = 1, \qquad k = 1, 2, \ldots, K \tag{7.28}$$

The probability density function of D, given the input vector $\mathbf{x}$ and given that the kth rule of the probabilistic generative model (i.e., expert k) is selected, may therefore be described as:

$$f_D(d \,|\mathbf{x}, k, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(d - y_k)^2\right), \qquad k = 1, 2, \ldots, K \tag{7.29}$$

where $\boldsymbol{\theta}$ is a parameter vector denoting the parameters of the gating network and those of the experts in the ME model. The probability density function of D, given $\mathbf{x}$, is the *mixture* of the probability density functions $\{f_D(d \,|\mathbf{x},k,\boldsymbol{\theta})\}_{k=1}^{K}$, with the mixing parameters being the multinominal probabilities determined by the gating network. We may thus write

$$\begin{aligned} f_D(d \,|\mathbf{x}, \boldsymbol{\theta}) &= \sum_{k=1}^{K} g_k f_D(d \,|\mathbf{x}, k, \boldsymbol{\theta}) \\ &= \frac{1}{\sqrt{2\pi}} \sum_{k=1}^{K} g_k \exp\left(-\frac{1}{2}(d - y_k)^2\right) \end{aligned} \tag{7.30}$$

The probability distribution of Eq. (7.30) is called an *associative Gaussian mixture model.* Its nonassociative counterpart is the traditional Gaussian mixture model (Titterington et al., 1985; McLachlan and Basford, 1988), which is briefly described in

Chapter 5. An associative model differs from a nonassociative model in that the conditional means μ_k and mixing parameters g_k are *not* fixed; rather, they are all functions of the input vector $\mathbf{x}$. The associative Gaussian mixture model of Eq. (7.30) may therefore be viewed as a generalization of the traditional Gaussian mixture model.

The important aspects of the ME model shown in Fig. 7.8, assuming that it is properly tuned through training, are :

1. The output y_k of the kth expert provides an estimate of the conditional mean of the random variable representing the desired response D, given $\mathbf{x}$ and that rule k of the probabilistic generative model holds.
2. The output g_k of the gating network defines the multinomial probability that the output of expert k matches the value $D = d$, on the basis of knowledge gained from $\mathbf{x}$ alone.

Working with the probability distribution of Eq. (7.30) and given the training sample $\{(\mathbf{x}_i, d_i)\}_{i=1}^N$, the problem is to *learn* the conditional means $\mu_k = y_k$ and the mixing parameters g_k, $k = 1, 2, \ldots, K$, in an optimum manner, so that $f_D(d|\mathbf{x}, \boldsymbol{\theta})$ provides a good estimate of the underlying probability density function of the environment responsible for generating the training data.

Example 7.1 Regression Surface

Consider an ME model with two experts, and a gating network with two outputs denoted by g_1 and g_2. The output g_1 is defined by (see Eq. (7.21))

$$\begin{aligned} g_1 &= \frac{\exp(u_1)}{\exp(u_1) + \exp(u_2)} \\ &= \frac{1}{1 + \exp(-(u_1 - u_2))} \end{aligned} \tag{7.31}$$

Let $\mathbf{a}_1$ and $\mathbf{a}_2$ denote the two weight vectors of the gating network. We may then write

$$u_k = \mathbf{x}^T \mathbf{a}_k, \qquad k = 1, 2$$

and therefore rewrite Eq. (7.31) as:

$$g_1 = \frac{1}{1 + \exp(-\mathbf{x}^T(\mathbf{a}_1 - \mathbf{a}_2))} \tag{7.32}$$

The other output g_2 of the gating network is

$$\begin{aligned} g_2 &= 1 - g_1 \\ &= \frac{1}{1 + \exp(-\mathbf{x}^T(\mathbf{a}_2 - \mathbf{a}_1))} \end{aligned}$$

Thus, both g_1 and g_2 are in the form of a logistic function, but with a difference. The orientation of g_1 is determined by the direction of the difference vector $(\mathbf{a}_1 - \mathbf{a}_2)$, whereas the orientation of g_2 is determined by the direction of the difference vector $(\mathbf{a}_2 - \mathbf{a}_1)$, that is the negative of that for gate g_1. Along the *ridge* defined by $\mathbf{a}_1 = \mathbf{a}_2$, we have $g_1 = g_2 = 1/2$, and the two experts contribute equally to the output of the ME model. Away from the ridge, one or the other of the two experts assumes the dominant role.

■

7.7 HIERARCHICAL MIXTURE OF EXPERTS MODEL

The ME model of Fig. 7.8 works by dividing the input space into different subspaces, with a single gating network responsible for distributing information (gathered from the training data) to the various experts. The *hierarchical mixtures of experts (HME) model,* illustrated in Fig. 7.11, is a natural extension of the ME model. The illustration is for an HME model of four experts. The architecture of the HME model is like a *tree,* in which the gating networks sit at the various nonterminals of the tree and the experts sit at the leaves of the tree. The HME model differs from the ME model in that the input space is divided into a *nested* set of subspaces, with the information combined and redistributed among the experts under the control of several gating networks arranged in a *hierarchical* manner.

The HME model of Fig. 7.11 has two *levels of hierarchy* or two *layers of gating networks*. By continuing with the application of the principle of divide and conquer in a manner similar to that illustrated, we may construct an HME model with any number of levels of hierarchy. Note that according to the convention described in Fig. 7.11, the numbering of gating *levels* starts from the output node of the tree.

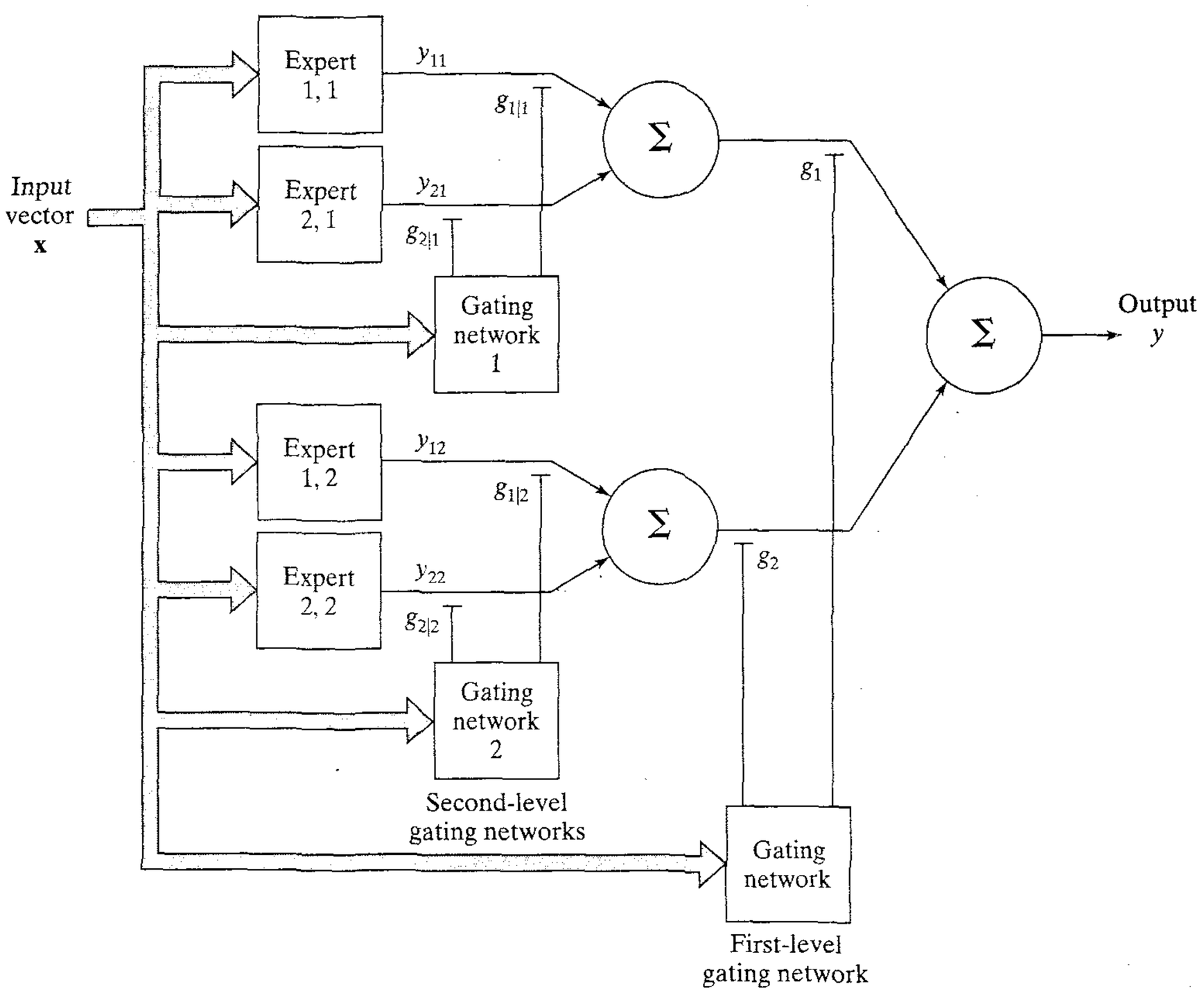


FIGURE 7.11 Hierarchical mixture of experts (HME) illustrated for two levels of hierarchy.

The formulation of the HME model of Fig. 7.11 can be viewed in two ways (Jordan, 1994):

1. *The HME model is a product of the divide and conquer strategy.* If we believe that it is a good strategy to divide the input space into regions, then it is an equally good strategy to divide each of those regions into subregions. We can continue recursively in this manner until we reach a stage where the complexity of the approximating surfaces is a good fit to the "local" complexity of the training data. The HME model should therefore perform at least as well as, and often better than, the ME model for this reason: A higher level gating network in the HME model effectively combines information and redistributes it to the experts in the particular subtree controlled by that gating network. Consequently, each parameter in the subtree in question shares strength with other parameters contained in the same subtree, thereby contributing to possible improvement in the overall performance of the HME model.
2. *The HME model is a soft-decision tree.* According to this second viewpoint, the mixture of experts is just a one-level decision tree, sometimes referred to as a *decision stump*. In a more general setting, the HME model is viewed as the probabilistic framework for a decision tree, with the output node of the HME model referred to as the *root* of the tree. The methodology of a *standard* decision tree constructs a tree that leads to *hard* (e.g., yes-no) decisions in different regions of the input space. This is to be contrasted with the soft decisions performed by an HME model. Consequently, the HME model may outperform the standard decision tree for two reasons:
   - A hard decision inevitably results in loss of information, whereas a soft decision tries to preserve information. For example, a soft binary decision conveys information about distance from the decision boundry (i.e., the point at which the decision is 0.5), whereas a hard decision cannot. We may therefore say that unlike the standard decision tree, the HME model adheres to the *information preservation rule*. This empirical rule states that the information content of an input signal should be preserved in a computationally efficient manner until the system is ready for final decision-making or parameter estimation (Haykin, 1996).
   - Standard decision trees suffer from a *greediness* problem. Once a decision is made in such a tree, it is frozen and never changes thereafter. The HME model lessens the greediness problem because the decisions made throughout the tree are continually altered. Unlike the standard decision tree, it *is* possible in the HME model to recover from a poor decision somewhere further along the tree.

The second viewpoint, that is, a soft decision tree is the preferred way to think about an HME model. With the HME viewed as the probabilistic basis for a decision tree, it allows us to calculate a likelihood for any given data set, and maximize that likelihood with respect to the parameters that determine the splits between various regions in the input space. Thus, by building on what we already know about standard decision trees, we may have a practical solution to the model selection problem as discussed in the next section.

7.8 MODEL SELECTION USING A STANDARD DECISION TREE

As with every other neural network, a satisfactory solution to the parameter estimation problem hinges on the selection of a suitable model for the problem at hand. In the case of an HME model, the model selection involves the choice of the number and arrangement of the decision nodes in the tree. One practical solution to this particular model selection problem is to run a standard decision tree algorithm on the training data, and to adopt the tree so obtained as the initializing step for the learning algorithm used to determine the parameters of the HME model (Jordan, 1994).

The HME model has clear similarities with standard decision trees, such as the *classification and regression tree* (CART) due to Breiman et al., (1984). Figure 7.12 shows an example of CART, where the space of input data, $\mathscr{X}$ is repeatedly partitioned by a sequence of binary *splits* into *terminal nodes*. Comparing Fig. 7.12 with Fig. 7.11, we readily see the following similarities between CART and HME:

- The rules for selecting splits at intermediate (i.e., nonterminal) nodes of CART play an analogous role to gating networks in the HME model.
- The terminal nodes in CART play an analogous role to expert networks in the HME model.

By starting with CART for a classification or regression problem of interest, we take advantage of the *discrete* nature of CART to provide an efficient search among alternative trees. By using a tree so chosen as the initializing step in the learning algorithm for parameter estimation, we take advantage of the *continuous* probabilistic basis of the HME model to yield an improved "soft" estimate of the desired response.

The CART Algorithm

In light of what has just been said, a brief description of the CART algorithm is in order. The description is presented in the context of regression. Starting with the training data $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$, we may use CART to construct a binary tree T for least-squares regression, by proceeding as follows (Breiman et al., 1984):

1. *Selection of splits.* Let a node t denote a subset of the current tree T. Let $\bar{d}(t)$ denote the average of d_i for all cases $(\mathbf{x}_i, d_i)$ falling into t, that is,

$$\bar{d}(t) = \frac{1}{N(t)} \sum_{\mathbf{x}_i \in t} d_i \tag{7.33}$$

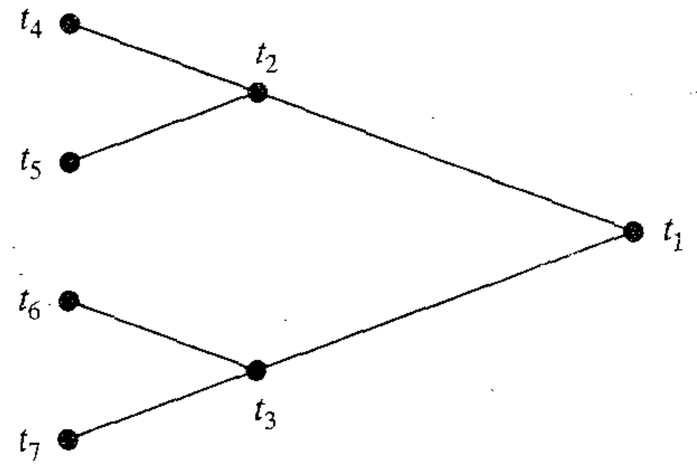


FIGURE 7.12 Binary decision tree, described as follows:
- Nodes t_2 and t_3 are *descendents* of node t_1.
- Nodes t_4 and t_5 are descendents of node t_2; and likewise for t_6 and t_7 in relation to t_3.

where the sum is over all d_i such that $\mathbf{x}_i \in t$ and $N(t)$ is the total number of cases in t. Define

$$\mathscr{E}(t) = \frac{1}{N} \sum_{\mathbf{x}_i \in t} (d_i - \bar{d}(t))^2 \tag{7.34}$$

and

$$\mathscr{E}(T) = \sum_{t \in T} \mathscr{E}(t) \tag{7.35}$$

For node t, the sum $\sum_{\mathbf{x}_i \in t}(d_i - \bar{d}(t))^2$ represents the "within node sum of squares"; that is, it is the total squared deviations of all the d_i in t from their average $\bar{d}(t)$. Summing these deviations over $t \in T$ gives the total node sum of squares and dividing it by N gives the average.

Given any set of splits S of a current node t in T, the best split s^* is that split in S that most decreases $\mathscr{E}(T)$. To be more precise, suppose that for any split s of node t into t_L (a new node to the left of t) and t_R (another new node to the right of t), we let

$$\Delta\mathscr{E}(s,t) = \mathscr{E}(T) - \mathscr{E}(t_L) - \mathscr{E}(t_R) \tag{7.36}$$

The best split s^* is then taken to be the particular split for which we have

$$\Delta\mathscr{E}(s^*,t) = \max_{s \in S} \Delta\mathscr{E}(t,s) \tag{7.37}$$

A regression tree so constructed is designed to maximize the decrease in $\mathscr{E}(T)$.

2. *Determination of a terminal node.* A node t is declared a terminal node if this condition is satisfied:

$$\max_{s \in S} \Delta\mathscr{E}(s,t) < \beta \tag{7.38}$$

where β is a prescribed *threshold*.

3. *Least-squares estimation of a terminal node's parameters.* Let t denote a terminal node in the final binary tree T, and let $\mathbf{X}(t)$ denote the matrix composed of $\mathbf{x}_i \in t$. Let $\mathbf{d}(t)$ denote the corresponding vector composed of all the d_i in t. Define

$$\mathbf{w}(t) = \mathbf{X}^+(t)\mathbf{d}(t) \tag{7.39}$$

where $\mathbf{X}^+(t)$ is the pseudoinverse of matrix $\mathbf{X}(t)$. The use of $\mathbf{w}(t)$ yields a least-squares estimate of $d(t)$ at the output of terminal node t. Using the weights calculated from Eq. (7.39), the split selection problem is solved by looking for the least sum of squared residuals (errors) with respect to the regression surfaces, rather than with respect to the means.

Using CART to Initialize the HME Model

Suppose that the CART algorithm has been applied to a set of training data, resulting in a binary decision tree for this problem. We may describe a split produced by CART as a multidimensional surface defined by

$$\mathbf{a}^T\mathbf{x} + b = 0$$

where $\mathbf{x}$ is the input vector, $\mathbf{a}$ denotes a parameter vector, and b denotes a bias.

Consider next the corresponding situation in an HME model. From Example 7.1 we note that the regression surface produced by a gating network in a binary tree may be expressed as:

$$g = \frac{1}{1 + \exp(-(\mathbf{a}^T\mathbf{x} + b))} \tag{7.40}$$

which defines a split, particularly when $g = 1/2$. Let the weight vector (difference) $\mathbf{a}$ for this particular gating network be written as

$$\mathbf{a} = \|\mathbf{a}\| \cdot \frac{\mathbf{a}}{\|\mathbf{a}\|} \tag{7.41}$$

where $\|\mathbf{a}\|$ denotes the length (i.e., Euclidean norm) of $\mathbf{a}$, and $\mathbf{a}/\|\mathbf{a}\|$ is a normalized unit-length vector. Using Eq. (7.41) in (7.40), we may thus rewrite a parameterized split at a gating network as:

$$g = \frac{1}{1 + \exp\left(-\|\mathbf{a}\|\left(\left(\frac{\mathbf{a}}{\|\mathbf{a}\|}\right)^T\mathbf{x} + \frac{b}{\|\mathbf{a}\|}\right)\right)} \tag{7.42}$$

where we see that $\mathbf{a}/\|\mathbf{a}\|$ determines the *direction* of the split and $\|\mathbf{a}\|$ determines the sharpness of the split. From the discussion presented in Chapter 2, we observe that the length of vector $\mathbf{a}$ acts effectively as the *reciprocal of temperature*. The important point to note from Eq. (7.42) is that a gating network made up of a linear filter followed by a softmax form of nonlinearity is able to mimic a split in the style of CART. Moreover, we have an additional degree of freedom, namely, the length of parameter vector $\mathbf{a}$. In a standard decision tree, this additional parameter is irrelevant because a threshold (i.e., hard decision) is used to create a split. In contrast, the length of $\mathbf{a}$ has a profound influence on the split sharpness produced by a gating network in the HME model. Specifically, for a synaptic weight vector $\mathbf{a}$ of fixed direction, we may state that:

- when $\mathbf{a}$ is long (i.e., the temperature is low), the split is sharp, and
- when $\mathbf{a}$ is short (i.e., the temperature is high), the split is soft.

If, in the limit we have $\|\mathbf{a}\| = 0$, the split vanishes and $g = 1/2$ on both sides of the vanished (fictitious) split. The effect of setting $\|\mathbf{a}\| = 0$ is equivalent to pruning the nonterminal node from the tree, because the gating network in question is no longer splitting. In the very extreme case when $\|\mathbf{a}\|$ is small (i.e., the temperature is high) at every nonterminal node, the entire HME model acts like a single node; that is, the HME is reduced to a linear regression model (assuming linear experts). As the synaptic weight vectors of the gating networks start to grow in length, the HME starts to make (soft) splits, thereby enlarging the number of degrees of freedom available to the model.

We may thus initialize the HME by proceeding as follows:

1. Apply CART to the training data.
2. Set the synaptic weight vectors of the experts in the HME model equal to the least-squares estimates of the parameter vectors at the corresponding terminal nodes of the binary tree resulting from the application of CART.

3. For the gating networks:
   (a) set the synaptic weight vectors to point in directions that are orthogonal to the corresponding splits in the binary tree obtained from CART, and
   (b) set the lengths (i.e., Euclidean norms) of the synaptic weight vectors equal to *small* random vectors.

7.9 A PRIORI AND A POSTERIORI PROBABILITIES

The multinomial probabilities g_k and $g_{j|k}$ pertaining to the first-level and second-level gating networks, respectively, may be viewed as *a priori* probabilities, in the sense that their values are solely dependent on the input vector (stimulus) $\mathbf{x}$. In a corresponding way, we may define *a posteriori* probabilities $h_{j|k}$ and h_k whose values depend on both the input vector $\mathbf{x}$ and the responses of the experts to $\mathbf{x}$. This latter set of probabilities is useful in the development of learning algorithms for HME models.

Referring to the HME model of Fig. 7.11, we define the *a posteriori* probabilities at the nonterminal nodes of the tree as (Jordan and Jacobs, 1994):

$$h_k = \frac{g_k \sum_{j=1}^{2} g_{j|k} \exp\left(-\frac{1}{2}(d - y_{jk})^2\right)}{\sum_{k=1}^{2} g_k \sum_{j=1}^{2} g_{j|k} \exp\left(-\frac{1}{2}(d - y_{jk})^2\right)} \tag{7.43}$$

and

$$h_{j|k} = \frac{g_{j|k} \exp\left(-\frac{1}{2}(d - y_{jk})^2\right)}{\sum_{j=1}^{2} g_{j|k} \exp\left(-\frac{1}{2}(d - y_{jk})^2\right)} \tag{7.44}$$

The product of h_k and $h_{j|k}$ defines the *joint a posteriori* probability that expert (j, k) produces an output y_{jk} that matches the desired response d, as given by

$$\begin{aligned} h_{jk} &= h_k h_{j|k} \\ &= \frac{g_k g_{j|k} \exp\left(-\frac{1}{2}(d - y_{jk})^2\right)}{\sum_{k=1}^{2} g_k \sum_{j=1}^{2} g_{j|k} \exp\left(-\frac{1}{2}(d - y_{jk})^2\right)} \end{aligned} \tag{7.45}$$

The probability h_{jk} satisfies the following two conditions

$$0 \leq h_{jk} \leq 1 \qquad \text{for all } (j, k) \tag{7.46}$$

and

$$\sum_{j=1}^{2} \sum_{k=1}^{2} h_{jk} = 1 \tag{7.47}$$

The implication of Eq. (7.47) is that credit is distributed across the experts on a competitive basis. Moreover, we note from Eq. (7.45) that the closer y_{jk} is to d, the more