1. Multi-Net Systems

Summary.

This chapter provides an introduction to the main methods and issues in the combination of Artificial Neural Nets. A distinction is made between ensemble and modular modes of combination, and the two are then treated separately. The reasons for ensemble combination are considered, and an account is provided of the main methods for creating and combining ANNs in ensembles. This account is accompanied by a discussion of the relative effectiveness of these methods, in which the concepts of *diversity* and *selection* are explained. The review of modular combination outlines the main methods of creating and combining modules, depending on whether the relationship between the modules is co-operative, competitive, sequential or supervisory. An overivew of the chapters in the book forms the conclusion section.

The honeymoon is officially over, and neural computing has moved beyond simple demonstrations to more significant applications. There is a growing realisation that such applications can be facilitated by the development of multi-net systems. Multi-net systems can result in solutions to tasks which either cannot be solved by a single net, or which can be more effectively solved by a system of modular neural net components. Similarly, better performance can be achieved when ANNs, as unstable predictors, are redundantly combined. Arguably, there are few neural net applications accomplished by means of a single net where better performance could not be achieved if this single net were replaced by a multi-net system. As well as performance improvement, there are other advantages to decomposing a task into modular components. For example, a modular system can be easier to understand and to modify. And modularity is almost necessarily implicated in any brain or biological modelling.

It seems likely that multi-net systems will be an important component of future research in neural computing. There are a number of areas from which inspiration and guidance about the construction of such systems can be gained. Clearly we can expect a major contribution from statisticians, and from the wider machine learning community, in terms, for instance, of explanations of the relative effectiveness of different methods for creating and combining ensemble members. Although the focus of concern here is on the combining of artificial neural nets in particular, research on combining other kinds of unstable predictors (e.g. decision trees, see Breiman, Chap-

ter 2, Drucker, Chapter 3) is also relevant, and there is no reason why the members of an ensemble should not consist of a variety of predictors. Insights about combining could potentially be gained from consideration of other areas as well, such as the modelling of biological systems which also make use of redundant and modular elements. And the concept of reliability through redundancy is one that is familiar in a number of different areas, such as software engineering (see Eckhardt [1], for example).

The aim of this chapter is to provide a review of the main methods that have been proposed for combining artificial neural net modules and ensembles, and to examine the principal motivations for creating multi-net systems. However, we shall first turn our attention to a consideration of the distinction between ensembles and modules, and of the ways in which they can be combined to form multi-net systems.

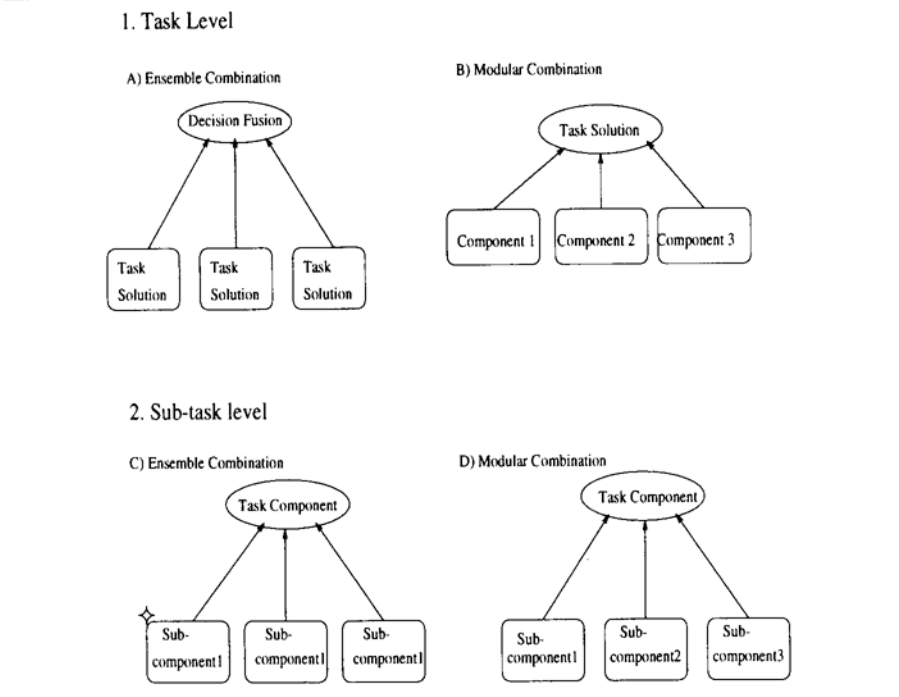

Fig. 1.1. Ensemble and modular multi-net systems, at task and sub-task levels

1.0.1 Different Forms of Multi-Net System

It is useful to make a distinction between ensemble and modular combinations of artificial neural nets [2]. The term 'ensemble' is the one commonly used for

combining a set of redundant nets (e.g. [3]), although the term 'committee' [4] or 'committee machine' has also been used for the same purpose. In an *ensemble* combination, the component nets are redundant in that they each provide a solution to the same task, or task component, even though this solution might be obtained by different means. By contrast, under a *modular* approach, the task or problem is decomposed into a number of subtasks, and the complete task solution requires the contribution of all of the several modules (although individual inputs may be dealt with by only one of the modules). Both ensemble and modular combinations can exist at either a task, or a sub-task level, as shown in Figure 1.1.

- Task level: An ensemble could consist of a number of different solutions to an entire task, or problem (Figure 1.1a). Similarly, a task solution might be constructed from a combination of a number of decomposed modules (Figure 1.1b).
- Sub-task level. When a task or an application is decomposed into component modules, each modular component could itself consist of an ensemble of nets, each of which provided a solution for that same modular component (Figure 1.1c). Alternatively, each module could be further subdivided into yet more specialist modules (Figure 1.1d).

At both levels in Figure 1.1, the distinction between an ensemble or modular combination depends on the presence or absence of redundancy; note the redundant components (several versions of Subcomponent 1) in the ensemble sub-task example, (Figure 1.1c) as compared to the lack of redundancy (Subcomponents 1, 2 and 3) in the modular sub-task example (Figure 1.1d). It should be noted that the modular examples (Figure 1.1b and d) at both levels could either result from the decomposition of a task into smaller components, or could represent 'bottom-up' fusion of information from distinct sensors, which provides a link to the quite considerable literature on sensor fusion [see [5] for a review]. Here, rather than decomposing in order to simplify the task, the modular structure can arise as a consequence of the available inputs.

Ensemble and modular combinations should not be thought of as mutually exclusive. It should be noted that Figure 1.1 is designed to show building blocks from which a multi-net system could be constructed: an actual multi-net system could consist of a mixture of ensemble and modular combinations at different levels. As an illustration, Figure 1.2 shows a hypothetical multi-net system which consists of both ensemble and modular components. At the top level, the system consists of an ensemble combination of three task solutions. At the sub-task level however, one of the task solutions is arrived at as the result of a modular combination of distinct components. The three task solutions are produced in different ways. The first is computed on the basis of data from one of three sensors. The second is computed on the basis of a cooperative combination of the output of three sensors. And the third is assembled from the modular combination of three subcomponents, each of which relies on input from a single sensor. Although this figure assumes the

existence of three distinct sensors, a similar modular decomposition could be based on an equivalent partitioning of a data set, or on the extraction of different features from the data.

In this chapter, a 'module' is assumed to be a self-contained or autonomous processing unit (see Fodor's account of informationally encapsulated modules, [6]). Under this definition, it is apparent that the component nets in an ensemble are themselves modules; however the point remains that their manner of combination is that of an ensemble as opposed to a modular combination, and that they are themselves redundant versions of the same task or task component.

In the following sections, ensemble combinations and modular combinations will be considered in turn. An overview of methods for ensemble creation and combination will be presented, followed by a consideration of how a choice might be made about which of the several methods should be used. Following this, in Section 1.2, an account of the methods for creating and combining modules is presented. The chapter is concluded with a brief description of the contents of the other chapters in the book.

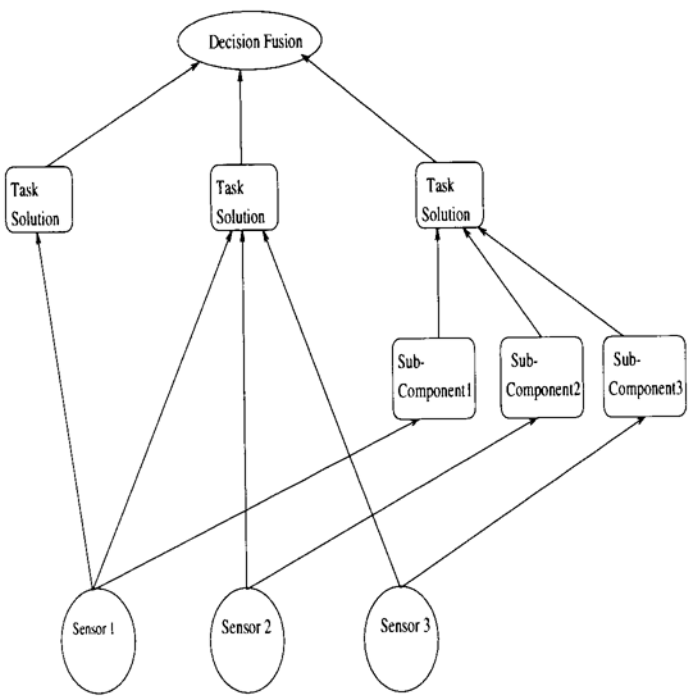

Fig. 1.2. An example of a multi-net system containing ensemble and modular combinations

1.1 Ensembles

1.1.1 Why Create Ensembles?

The main motivation for combining nets in redundant ensembles is that of improving their generalisation ability, or to put it another way, to guard against the failure of individual component nets. The reason for expecting individual nets to sometimes "fail", or make errors on some inputs, is based on the assumption that they will have been trained on a limited set of training data, and required, on the basis of that data, to *estimate* the target function. Such estimates will inevitably not be identical to the target function (unless the training set is perfectly representative and/or the function is so simple that interpolation between the training points results in perfect generalisation.

The idea of combining estimators in order to achieve better performance is one that has a long history, and has emerged independently in a number of different areas. For instance, in the context of democracy models, the Condorcet Jury model proposed in 1786 was designed to study the conditions under which a democracy as a whole is more effective than any of its constituent members [7]. In 1956, von Neumann was writing about the 'synthesis of reliable organisms from unreliable components' [8]. In the area of forecasting, it has been shown that better results can be achieved by combining forecasts than by choosing the best one [9]. Researchers of sensor fusion have looked at the best ways of combining sensors that are subject to probabilistic errors [10]. In software engineering a standard method of increasing reliability is through the incorporation of redundancy, or multiple versions [11], and in hardware, a standard approach to increasing reliability is that of Triple-Modular Redundancy.

Combining a set of imperfect estimators can be thought of as a way of *managing* the recognised limitations of the individual estimators; each component net is known to make errors, but they are combined in such a way as to minimise the effect of these errors. A consideration of the reasons for redundant ensembles can be clarified by an examination of the likely effect of such combining in terms of the statistical concepts of bias and variance. Much has been made recently of the fact that the error of a predictor can be expressed in terms of the *bias* squared plus the *variance* (see [14] for a detailed presentation of these concepts, and for further discussion, see [4]; [13]; [14] [45]).

A net can be trained to construct a function $f(\mathbf{x})$, based on a training set $(x_1, y_1), ..., (x_n, y_n)$ for the purpose of approximating y for previously unseen observations of x, (the following account and discussion applies to regression problems; for a formulation appropriate to classification see Chapter 2). Following [14] we shall indicate the dependence of the predictor f on the training data by writing $f(\mathbf{x}; D)$ instead of $f(\mathbf{x})$. Then the mean squared error of f as a predictor of y may be written

$$E_{\mathcal{D}}[(f(\mathbf{x}; \mathcal{D}) - E[y|\mathbf{x}])^2]$$

where $E_{\mathcal{D}}$ is the expectation operator with respect to the training set $\mathcal{D}$, (i.e. the average of the set of possible training sets), and $E[y|\mathbf{x}]$ is the target function. Now the bias/variance decomposition gives us,

$$E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D}) - E[y|\mathbf{x}])^2] = (E_{\mathcal{D}}[f(\mathbf{x};\mathcal{D})] - E[y|\mathbf{x}])^2 \quad \text{"bias"}$$
$$+E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[f(\mathbf{x};\mathcal{D})])^2] \quad \text{"variance"}$$

The bias and variance of a predictor can be estimated when the predictor is trained on different sets of data sampled randomly from the entire possible set. The bias of a net can be intuitively characterised as a measure of its ability to generalise correctly to a test set once trained (reflecting the average output over the set of possible training sets). The variance of a net can be similarly characterised as a measure of the extent to which the output of a net is sensitive to the data on which it was trained, i.e. the extent to which the same results would have been obtained if a different set of training data were used.

There is a tradeoff between bias and variance in terms of training nets; the best generalisation requires a compromise between the conflicting requirements of small variance and small bias. It is a tradeoff because attempts to decrease the bias (i.e. taking more account of the data) are likely to result in higher variance, whilst efforts to decrease the variance (i.e. taking less notice of the data) usually result in increased bias. What is required of a net that is to generalise well following training on noisy or unrepresentative data[1] is to take sufficient account of the data, but to avoid overfitting (low variance, low bias).

The bias and variance can be approximated by an average over a fixed number of possible training sets. Krogh and Vedelsby [16] provide an account of the bias and variance in an ensemble, expressing the bias-variance relation in terms of an ensemble average, instead of an average over possible training sets (which means that the ensemble members could be created by a variety of methods, see Section 1.1.2, as well as by varying the training set). Krogh and Vedelsby's account is made use of by Opitz and Shavlik in Chapter 4. In terms of an ensemble of nets, the bias measures the extent to which the ensemble output averaged over all the ensemble members differs from the target function, whilst the variance is a measure of the extent to which the ensemble members disagree (Krogh and Vedelsby use the term 'ambiguity' to refer to this disagreement).

The performance improvement which can arise from ensemble combinations is is usually the result of a reduction in variance, rather than a reduction in bias, since the usual effect of ensemble averaging is to reduce the variance

[1] If the data were not noisy, and were sufficiently representative of the test set to permit good generalisation, then there would be no problem with overfitting.

of a set of nets, whilst leaving the bias unaltered.[2] Therefore, an effective approach is to create and/or select a set of nets that exhibits high variance, but low bias, since the variance component can be removed by combining. In other words, it makes sense to take steps to reduce bias (taking more account of the available data) since the increased variance that results from these steps will be removed by combining. Combining can thus provide a way of circumventing, or at least reducing, the bias-variance tradeoff described above.

An ensemble which exhibits high variance should also show a low correlation of errors. One of the main determinants of the effectiveness of an ensemble is then the extent to which the members are 'error-independent' [44], in the sense that they make different errors (or to put it another way, show different patterns of generalisation). Wolpert [54] for instance points out that '..the more each generalizer has to say (which isn't duplicated in what other generalizers have to say), the better the resultant stacked generalization..' And for Jacobs [20], '..The major difficulty with combining expert opinions is that these opinions tend to be correlated or dependent..' This same point is made in other areas, such as software engineering ([1]; [21], and forecasting [22]). The ideal, in terms of ensembles of artificial neural nets, would be a set of nets which did not show any coincident errors. That is, each of the nets generalised well (low bias component of error), and when they did make errors on the test set, these errors were not shared with any other nets (high variance component of error).

Rather than just considering the relative contribution of bias and variance to the total error, or measuring the error correlation, it is also possible to distinguish different types of error patterns that an ensemble may exhibit when tested. Sharkey and Sharkey [23] [24] present an account of four different levels of error pattern which may be exhibited by an ensemble with respect to a validation test set, (although they use the term 'diversity'). These range from Level 1 to the minimal requirement for diversity, Level 4. In Level 1 Diversity, there are no coincident errors, and when errors occur on one net they are not shared with any other ensemble member. In Level 2 Diversity there are coincident errors, but the majority is always correct. In Level 3 Diversity, the majority is not always correct, but the correct output is always produced by at least one net. In Level 4 Diversity, the majority is not always correct, and there are some inputs which fail on all the ensemble members, but there is some difference between the errors made by different nets that could be exploited by combining.

An advantage of defining levels of diversity is that it makes it possible to quantify the level of error independence achieved by an ensemble. The method is more clearly applicable where binary outputs are used (although an error threshold could be used to convert continuous outputs to binary

[2] Although some forms of stacking, i.e. the use of a level 1 generaliser, may reduce bias, (eg [17]).

outputs). An interesting question is the extent to which it would be better to include nets in an ensemble which clearly make different errors on a test set, or whether it would be better to include nets, none of which make errors on the test set. In the first case, there is evidence that the nets exhibit a degree of diversity, whilst in the second although they might appear to result in better performance, it is possible that they show identical patterns of generalisation, and that they would make identical errors if tested on further examples beyond the test set (i.e. *Better the devil you know than the devil you don't know*).

1.1.2 Methods for Creating Ensemble Members

Since the main reason for combining nets in ensembles is to improve their performance, there is clearly no advantage to be gained from an ensemble that is composed of a set of identical nets; identical that is, in that they *generalise in the same way.* The emphasis here is on the similarity or otherwise of the pattern of generalisation. In principle, a set of nets could vary in terms of their weights, the time they took to converge, and even their architecture (eg the number of hidden units) and yet constitute essentially the same *solution*, since they resulted in the same pattern of errors when tested on a test set.

The aim then is to find nets which generalise differently. There are a number of training parameters which can be manipulated with this goal in mind. These include the following: initial conditions, the training data, the typology of the nets, and the training algorithm. We can provide an overview of the main methods which have been employed for the creation of ensemble members, whilst providing more information about methods which involve varying the data, since that is the approach which has most commonly been taken.

- Varying the set of initial random weights: A set of nets can be created by varying the initial random weights from which each net is trained whilst holding the training data constant.
- Varying the topology: A set of nets can be created by varying the topology or architecture, and training with a varying number of hidden units whilst holding the training data constant. An interesting possibility is that of varying the topology in a more radical manner, such that each ensemble member itself consists of a different modular system. The errors made by two modular systems with different internal modular structures might well be uncorrelated.
- Varying the algorithm employed: The algorithm used to train the nets could be varied whilst holding the data constant. Our concern here is with the use of Artificial Neural Nets, but the members of an ensemble could be created using a variety of statistical techniques.
- Varying the data: The methods which seem to be most frequently used for the creation of ensembles are those which involve altering the train-

ing data. There are a number of different ways in which this can be done which include: sampling data, disjoint training sets, boosting and adaptive resampling, different data sources, and preprocessing. These are considered individually below, although it should be noted that ensembles could be created using a combination of two or more of these techniques (e.g. sampling *plus* preprocessing, or sampling, noise injection and weight decay, see Raviv and Intrator, in Chapter 7).

Sampling data: A common approach to the creation of a set of nets for an ensemble is to use some form of sampling technique, such that each net in the ensemble is trained on a different subsample of the training data. Resampling methods which have been used for this purpose include cross-validation [16], bootstrapping [25], and smooth bootstrapping [26]; although in statistics the methods are better known as techniques for estimating the error of a predictor from limited sets of data. In bagging [25] a training set containing N cases is perturbed by sampling with replacement (bootstrap) N times from the training set. The perturbed data set may contain repeats. This procedure can be repeated several times to create a number of different, although overlapping, data sets. Such statistical resampling techniques are particularly useful where there is a shortage of data.

Disjoint training sets: A similar method to the above is the use of disjoint, or mutually exclusive training sets, i.e. sampling without replacement (e.g. [27]). There is then no overlap between the data used to train different nets. The problem is that, as noted by [28], the size of the training set may be reduced, and this may result in deteriorated performance.

Boosting and Adaptive resampling: Schapire [29] showed that a series of weak learners could be converted to a strong learner as a result of training the members of an ensemble on patterns that have been filtered by previously trained members of the ensemble. A number of empirical studies (e.g. [30], support the efficacy of the boosting algorithm, although a problem with this method is that it requires large amounts of data. Freund and Schapire [12] have proposed an algorithm, Adaboost, that largely avoids this problem, although it was developed in the context of boosting. Essentially the basis of this algorithm is that training sets are adaptively resampled, such that the weights in the resampling are increased for those cases which are most often misclassified. Drucker, in Chapter 3 compares the effectiveness of boosting to bagging, finding generally superior performance from boosting algorithms. Similarly, Breiman, [32] and Chapter 2 explores some of the differences between the Freund and Schapire algorithm and bagging; concluding, on the basis of empirical and analytic evidence, that Freund and Schapire's algorithm is more successful than bagging at variance reduction.

Different data sources: Another method of varying the data on which nets are trained is to use data from different input sources. This is possible under circumstances in which, for instance, more than one sensor is used, and it

is particularly applicable where the sensors are designed to pick up different kinds of information. For example, picking up fuel injection faults in a diesel engine using either a measure of engine cylinder pressure, or engine cylinder temperature [27].

Preprocessing: The data on which nets are trained can also be varied by using different preprocessing methods. For example, different signal processing methods might be applied to the data, or different feature sets extracted. Alternatively, the input data for a set of nets could be distorted in different ways; for example by using different pruning methods (see [28]), by injecting noise (see Raviv and Intrator, Chapter 7), or by using non-linear transformations [24].

1.1.3 Methods for Combining Nets in Ensembles

Once a set of nets has been created, an effective way of combining their several outputs must be found. There are several different methods of combining, and since a number of reviews of the topic already exist, (e.g. [20]; [33]; [34]; [35]), I shall do no more than briefly outline some of the more common methods.

Averaging and weighted averaging: Linear opinion pools are one of the most popular aggregation methods, and refer to the linear combination of the outputs of the ensemble members' distributions with the constraint that the resulting combination is itself a distribution (see [20]). An single output can be created from a set of net outputs via simple averaging, (e.g. [41]), or by means of a weighted average that takes account of the relative accuracies of the nets to be combined (e.g. [41]; [19] [20] [39]).

Non-linear combining methods: Non-linear combining methods that have been proposed include Dempster-Shafer belief-based methods, (e.g. [44]), combining using rank-based information (e.g. [40]), voting (e.g. [3]), and order statistics [41] and Tumer and Ghosh, Chapter 6).

Supra Bayesian: Jacobs [20] contrasts supra Bayesian with linear combinations. The underlying philosophy of the supra Bayesian approach is that the opinions of the experts are themselves data. Therefore the probability distribution of the experts can be combined with its own prior distribution.

Stacked generalisation: Under stacked generalisation [54] a nonlinear net learns how to combine the networks with weights that vary over the feature space. The outputs from a set of level 0 generalisers are used as the input to a level 1 generaliser, which is trained to produce the appropriate output. The term 'stacked generalisation' is used by Wolpert [54] to refer both to this method of stacking classifiers, and also to the method of creating a set of ensemble members by training on different partitions of the data. It is also possible to view other methods of combining, such as averaging, as instances of stacking with a simple level 1 generaliser. The same idea has

been adapted to regression tasks, where it is termed 'stacked regression', [9]. A comprehensive exploration of stacking is reported in [43].

1.1.4 Choosing a Method for Ensemble Creation and Combination

In the previous sections, Section 1.1.2 and Section 1.1.3 , an account has been provided of the different ways in which ensemble members can be created and combined. However, since more than one method for creating ensembles exists, clearly it would be helpful to have some guidance about which method is likely to produce the best results, or results that are better than that obtained by means of choosing the best net from a set of available alternatives. As argued earlier (Section 1.1.1) the effectiveness of an ensemble depends on the extent to which its members make different errors, or are 'error-independent' [44]. Once the importance of the error correlation between the nets has been recognised, the main approaches which can be adopted are are:

1. *Taking account of the dependency between nets when choosing a method of combining.*
2. *Creating nets for effective combination.*
3. *Selecting nets for effective combination*

Taking account of the dependency between nets. Methods of combining which take into account the dependency between nets have been proposed. Hashem [19] [20] [39] provides an account of methods of finding optimal linear combinations of the members of an ensemble; combinations which take into account the relative accuracy of the component nets as opposed to using equal combination weights. An interesting alternative approach to this is that presented by Rosen [45], whereby nets are forced to be decorrelated with one another by means of a training algorithm that incorporates an error decorrelation penalty term designed to encourage nets to make errors which are decorrelated from those made by other nets.

The extent to which the outputs of a set of nets are correlated gives a strong indication about how they should be combined. For example, if on a classification problem, an ensemble does not exhibit any coincident failures with respect to a validation set (Level 1 Diversity), then combining the nets by means of a simple majority vote will produce good results. Good results will also be obtained if a simple majority vote is used to combine nets which do share coincident errors, but where the majority is always correct (Level 2 Diversity). Where there are overlapping errors, more complex methods of combination, such as stacked generalisation are likely to be appropriate, or some form of weighted averaging (e.g. optimal linear combinations, [19] [20] [39]; [41]).

Creating nets for effective combination. One approach to creating ensembles is to consider the relative merits of methods of creating ensemble members, and to choose and apply one which is likely to result in nets which

are diverse. Wolpert ([54]) described the available guidance on the choice of methods for generating ensemble members (or level 0 generalisers in his terms), as a 'black art'. However, a number of researchers have conducted empirical investigations into the effectiveness of different methods of ensemble creation, (e.g. [13]; [27]; [28]) and the consensus emerging from the field is that it is methods of creating nets by varying the data in some way that are more likely to result in nets which make different errors. And as is apparent from the outline above, the main methods of varying the data are using different sampling methods, varying the input-output relationships within the data, and adaptive resampling of some form (see Breiman, Chapter 2).

Varying the data on which a set of nets are trained is more likely, it appears, to result in a set of nets that can be combined effectively than varying for instance the set of initial conditions from which they are trained, or their topology. The conclusion about the relative ineffectiveness of varying the initial conditions is supported by the results of [13], and [27]. It has been claimed that backpropagation is sensitive to initial conditions [44], but the available evidence suggests that although variations in initial conditions may affect the speed of convergence, or whether or not a net converges, the resulting differences in generalisation are likely to be slight. It seems that unless the neural net being trained is low in complexity, often only one function that is compatible with a set of data is found. Therefore, regardless of the initial set of weights, the algorithm used for training, or its topology, a net that has learned a particular set of data is likely to show the same pattern of generalisation. Of course, it is difficult to argue conclusively against the possibility that altering the initial conditions of a net could result in significant changes in the pattern of generalisation, but the evidence suggests varying the initial conditions is likely to be less effective than training nets on data sets that are different in some way.

Selecting nets for effective combination. There are two questions that can be looked at here: (i) why should selection be undertaken, and (ii) how should it be carried out?

Why select?. There are a number of different ways in which the concept of selection could be incorporated into the construction of effective ensembles. However, before listing these, it is important to make the case in favour of selection of any kind. The point of selecting is to reduce the number of shared failures that a set of nets will produce. As has been argued (e.g. [20] and Chapter 5; [41]), the presence of 'harmful collinearity' or correlation between the errors made by the component nets in an ensemble will reduce the effectiveness of the ensemble itself. Even though the argument can be made that certain methods of creating ensemble members are more likely to be effective than others, it is still the case that any such methods are best combined with some form of testing and selection of ensemble members, for it cannot be assumed that adopting a particular approach ensures that error independence will be achieved. We shall illustrate this point with respect to

the notion of using disjoint training sets. Although varying the data might be expected to be an effective way of producing nets which generalise differently, this is not necessarily the case as a consideration of the notion of training set representativeness makes apparent. It is still important to test the resulting nets and establish the extent to which they constitute identical solutions, or make different errors.

The argument is that *disjoint training sets will not necessarily result in low error correlations.* This point can be explained with reference to the concept of training set *representativeness* (see [45] and [46] for further discussion of the notion of training set representativeness). A representative training set is one which leads to a function being inferred which is similar, or identical, to that which generated the test set. A representative training set will therefore lead to good generalisation. The problem is however, that two representative training sets, even if the data that defined them did not overlap at all, could still lead to very similar functions being inferred, with the result that their pattern of errors on the test set will be very similar. For instance, think of a simple classification determined by a boundary (i.e. a square wave boundary) where the output is 1 on one side of the boundary, and 0 on the other. There is a very large, or unbounded number of different combinations of data points which could be chosen as boundary conditions, but which would yield the same, or nearly the same pattern of generalisation. In the same way, the data points which make up a training set should not overlap with those in a test set, but it is to be hoped that they result in almost the same function being inferred.

On the other hand, if a candidate set of nets were trained using unrepresentative training sets, the resulting generalisation performance would be poor. The nets might each infer quite different functions, and show different patterns of generalisation to the test set, but as the amount of errors increases so does the probability that the errors that they make on the test set will overlap. It also follows that the smaller training sets that can result from using disjoint samples are also likely to be less representative; and result in the poorer performance noted by [28]. There is therefore a delicate balance between training set representativeness and error correlation. What is needed is several training sets, all of which are representative and lead to good generalisation, but which exhibit a minimum number of coincident failures. The extent to which they exhibit coincident failures (or the determination of the type of diversity they exemplify) can only be determined through a process of testing the performance of selected ensembles.

In the immediately preceding text, the examination of the notion of training set representativeness has been used to argue the case for the importance of testing and selecting nets for effective combining. However, the notion of the representativeness of training sets has further implications for effective combining. It has been argued (e.g. Breiman, Chapter 2), that nets trained for combining should be under-regularised, such that they fit the training

data closely, since the variance that results from this will be removed by combining. But, the issue is complicated if the data available for training is of poor quality (i.e. noisy and/or unrepresentative), since it is then likely to be important to avoid overfitting, and to use some form of regularisation. Thus Raviv and Intrator, Chapter 7, get better results when they incorporate a form of regularisation (weight decay) into the training of their component nets on noisy data. An interesting discussion of the role of overfitting in ensemble combination can be found in [47].

How to select. Having argued the case for the principle of selecting nets for inclusion in an ensemble, we can now turn to a consideration of the methods by which such selection will be accomplished. One approach is to create a pool of nets and then to use selection criteria to pick the best ensemble from amongst these. That is the approach taken, for instance, by Perrone and Cooper [41]. Perrone and Cooper suggest a heuristic selection method whereby the population of trained nets are ordered in terms of increasing mean squared error, and an ensemble is created by including those with lowest mean squared error. The process can be further refined by constructing a small ensemble and then only adding a new net if it results in a lower mean squared error for that ensemble. Hashem [20] also considers the selection of nets for effective combining and in his chapter, Chapter 5, compares the effectiveness of two alternative selection algorithms.

The same idea of selecting nets can be expanded (i) by applying selection procedures to a set of nets which have been created through the use of methods designed to promote diversity, and (ii) by continuing the process of generation and selection until a stopping criterion is reached. The second of these two possibilities is explored by Opitz and Shavlik ([48] and Chapter 4), who present a method which uses genetic algorithms to actively search for ensemble members which generalise well, but which disagree as much as possible. The standard genetic operators, crossover and mutation, are used to create new individuals from an initial set. The most fit members (in terms of generalisation and disagreement, or diversity) then form the next generation, and the process is repeated until 'a stopping criterion is reached' [48]. Once found, the ensemble members are combined using weighted averaging. Opitz and Shavlik ([48], and Chapter 4) are not explicit about the stopping criterion; a suggestion [2] is to use a stopping criterion such as Level 2 Diversity. The rigour of the stopping criterion would depend on the conflicting demands of time taken to search for an ensemble that fulfils it, and the demands for accuracy; clearly in a safety-critical domain it would make sense to use a stringent stopping criterion even if it took a long time to fulfil it.

1.2 Modular Approaches

1.2.1 Why Create Modular Systems?

There are a number of possible motivations for adopting a modular approach to a particular task or problem. Modular decomposition can be undertaken for the purposes of improving performance. In other words, a task could be solved with a monolithic net, but better performance is achieved when it is broken down into a number of specialist modules. One reason for better performance when a task is decomposed is that it makes it possible to switch to the most appropriate module, or blend of modules, depending on the current circumstances. The divide and conquer approaches that are exemplified by the mixture-of-experts approach (see below, Section 1.2.2) provide an example of the improved performance that can result from a modular system. Switching has also been discussed in the control literature [49] [50], and a similar exploitation of switching of control can be found in the literature on behaviour-based robotics [51].

In addition to performance improvement, there are other reasons for decomposing a problem. It might not be possible to accomplish the task in question unless the problem is simplified by decomposing it. Thus the 'divide and conquer' principle, whereby the task is divided into a number of sub-problems, can be used to extend the capabilities of a single net. Each sub-problem could then be solved with a different neural net architecture or algorithm, making it possible to exploit specialist capabilities. For example, [52] reports a solution to a robotics problem that was only obtained as a result of decomposing the problem into three separate components. Similarly, in [53], a solution to a language parsing problem (mapping from syntactically ambiguous sentences to a disambiguated syntactic tree) was only obtained when the problem was decomposed into three modules, each consisting of a different connectionist architecture.

Each component in a modular system can take the form of an artificial neural net. However, as in ensemble combination, there is no reason in principle why some of these components could not make use of non-neural computing techniques. Thus Catfolis and Meert [54] provide an account of the hybrid combination of a knowledge-based system and a neural net. Similarly, in the speech recognition literature, the use of hybrid system architectures (e.g. ANNs and hidden Markov models) is common [55]. The preprocessing of ANN inputs before training, whether this is accomplished with neural nets, or other systems, can also be viewed as a form of modular decomposition for the purposes of simplifying the problem.

There are other possible motivations for adopting a modular approach to a problem. As suggested earlier, sometimes the issue is one of recombining rather than decomposing, as is the case when the input information comes from a number of independent sources or sensors, and the potential for modularity is inherent in the task itself. In addition, a modular approach is often

more coherent in terms of biology/cognition/neurophysiology - for instance there are clear justifications for particular subdivisions when the aim is to model brain function, and it is reasonable to suppose that the processing of information, particularly sensory information, involves modularity (even when, as is the case in models of language processing, there is disagreement about what those modules are, and the extent to which they interact, [56]). Redundant systems in the brain could provide protection against damage, and could also offer flexibility, with different systems being used in different contexts. The advantages of 'multiple, partial representations of the world' have been discussed in the emerging area of behaviour-based artificial intelligence (c.f. [57]). For instance, if several partial representations confirm the same hypothesis, it is reinforced. Similarly, inconsistent hypotheses could provide 'back-up', to be used if a preferred hypothesis is shown to be inappropriate.

Another reason for adopting a modular approach is that of reducing model complexity, and making the overall system easier to understand, modify, and extend. This justification has often been noted (e.g. [58]; [59]) and is common to engineering design in general. Training times can be reduced as a result of modular decomposition ([60]), and prior knowledge can be incorporated in terms of suggesting an appropriate decomposition of a task [58].

1.2.2 Methods for Creating Modular Components

The main determinant of the form of a modular system is the way in which the component modules may have arrived, or been arrived at. A task may be decomposed into modules, or alternatively the input to the system might come from a number of independent sources such that the question is how to combine these to form an overall solution (i.e. sensor fusion). We shall consider decomposition and sensor fusion in turn. The decomposition of a problem into modular components may be accomplished automatically, explicitly, or by means of class decomposition [61]. Where the decomposition into modules is explicit, this usually relies on a strong understanding of the problem. The division into sub-tasks is known prior to training (eg [62]), and improved learning and performance can result (eg [63]). Similarly, specialist modules might be developed for particular purposes. Sometimes the modules may be specialist solutions to the same task, such that the best performance on the task will be obtained when the most appropriate module, given the circumstances, is selected. For instance, Ohno-Machado and Mussen [64] developed specialist modules for particular years, in a task where the aim was to predict AIDS survival. And Baxt [6] separately optimised neural net modules to either reduce the number of false positive errors, or the number of false negative errors.

Class decomposition involves breaking a problem into sub-problems based on the class relationships. The method, as proposed by Anand et al, [66] involves dividing a k-class classification problem into k two-class classification problems, whilst using the same number of training data for each two class

classification as the original k-class problem. A further refinement of this approach is reported in [61].

An alternative approach is one in which *automatic decomposition* of the task is undertaken, characterised by the blind application of a data partitioning technique. Automatic decomposition is more likely to be carried out with a view to improving performance, whilst explicit decomposition might either have the aim of improving performance, or that of accomplishing tasks which either could not be accomplished using a monolithic net, or could not be accomplished either as easily, or as naturally.

Automatic decomposition of a task for the purposes of improved performance is an approach which is closely related to the ensemble-based one we have already considered. Under the *divide and conquer* approach of Jacobs and Jordan ([67]; [68]; [69]) complex problems are automatically decomposed into a set of simpler problems. One approach to the automatic decomposition of a problem into modular components is presented by Luttrell, (Chapter 10). Mixtures of experts ([67]; Jacobs and Tanner, this volume) and Hierarchical mixtures of experts [68] partition the data into regions and fit simple surfaces to the data that fall in each region. Expert nets learn to specialise onto sub-tasks and to cooperate by means of a Gating net. The regions have 'soft' boundaries, which means that data points may lie simultaneously in multiple regions. The mixtures of experts model consists of a number of expert networks, combined by means of a Gating network which identifies the expert, or blend of experts, most likely to approximate the desired response. In Chapter 11, Jacobs and Tanner set the mixtures of experts approach in the wider context of other mixture models. The hierarchical extension of the mixtures of experts model is a tree-structured model which recursively divides each region into sub-regions. Such decomposition ensures that the errors made by the expert nets will not be correlated, for they each deal with different data points.

There are similarities between the mixtures of experts approach, and an ensemble-based one; the underlying aim of both is the improvement of performance, and both can involve linear combinations of their components. However, the approaches are distinct, in that the mixture-of-experts approach assumes that each data point is assigned to only one expert (mutual exclusivity) whereas ensemble combination makes no such assumption, and each data point is likely to be dealt with by all the component nets in an ensemble. In electronic discussions, Jordan has suggested that mixtures of experts are best thought of as another kind of statistical model, such as hidden Markov models. Thus, one of the members of an ensemble could be a mixtures of experts approach to a particular task, whilst other members were trained on the task using other techniques.

Sensor fusion can be seen as the (near) equivalent flipside to decomposition. As previously discussed, sensor fusion is the term used when the issue is one of how to combine information from independent sources, rather than

how to decompose. Here the modularity emerges naturally, and choices have to be made about how to recombine information (i.e. signal fusion, medium level fusion, decision fusion) instead of how to decompose the task. Although sensor fusion does not necessarily involve ANNs, consideration of the literature on sensor fusion is relevant to questions about multi-net systems. In sensor fusion it is possible to identify three main levels [70]. At the lowest level of pixel or signal fusion [70], raw data from disparate sensors is fused to form a common representation. At the intermediate level of mid-level fusion, features are extracted from several sensors and processed independently. And at the level termed "decision fusion" a number of locally made decisions are recombined. This could take the form of separately computing a solution to the entire task based on the input from each sensor, and then recombining these solutions - an approach which could be viewed as an example of ensemble combination but one in which the ensemble components will exhibit an increased independence of generalisation. The chapter by Fine and Jacobs (Chapter 8) considers alternative models of visual cue combination, in an approach which is closely related to the concept of sensor fusion.

1.2.3 Methods for Combining Modular Components

A consideration of the ways in which component modules, as distinct from ensembles, can be combined, should be aided by an outline of the main forms that such combination can take. It is possible to identify (at least) four different modes of combining component nets. Figure 1.3 shows four different modes of combining modules; co-operative, competitive, sequential, and supervisory.

A distinction is made here between co-operative and competitive combination, although the two could be classed together as in [2]. The main difference, as defined here, is that in co-operative combination it is assumed that all of the elements to be combined will make some contribution to the decision, even though this contribution may be weighted in some way; whereas in competitive combination, it is assumed that for each input the most appropriate element (task, component or sensor) will be selected. In sequential combination, the processing is successive; the computation of one module depending on the output of a preceding module. Such combination is found, for example, when inputs are sequentially processed by means of a number of nets. Modelling of the processing carried out by the brain, for instance the processing of visual information, is likely to involve sequential combination of modules (e.g. Fine and Jacobs, this volume). In a supervisory relationship, one module is used to supervise the performance of another module. For instance, McCormack [71] describes a system in which one module was trained to select the parameters of a second net, on the basis of observations of the effect of various parameter values on the performance of that net. A related, but different, supervisory relation is found in [17], where a supplementary

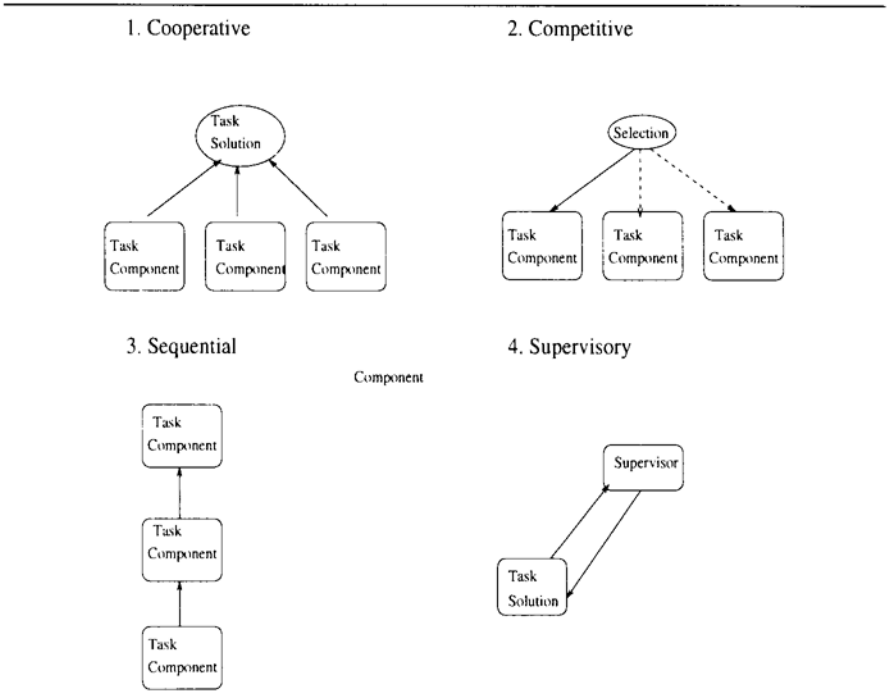

Fig. 1.3. Four different methods of combining ANN modules

network is trained to predict the error of a main ANN using input features and the output of the main ANN module.

It seems likely that ensembles will be involved in co-operative combination, and since such co-operative combination has already been considered under in Section 1.1 it is not examined further here. However, it is admitted that when separate task components are co-operatively combined to achieve a task solution, more complex methods of combination than those previously considered here might be required (see for example the notion of *cue promotion* discussed by Fine and Jacobs in Chapter 8, and the literature on sensor fusion e.g. [72]). For the present purposes, it is assumed that modular components are more likely to require competitive, sequential or supervisory combinations, depending on the task in hand. We shall briefly consider each of these in turn.

Competitive combination. In some modular combinations, the best performance is obtained when the most appropriate module is selected depending on the particular circumstances (circumstances in this case corresponding to either the inputs or outputs of the modules). There are two main mechanisms for accomplishing this selection; under the first, the system learns to allocate examples to the most appropriate module (i.e. Gating, see below). Under the second (Rule-based switching, see below), the switching is accomplished by means of an more explicit mechanism. One of the ensemble-combining methods outlined earlier, stacked generalisation, can also be trained to assign weights to different components depending on the current input. However, the method of stacked generalisation is based on the assumption that all the component nets will make *some* contribution, and our empirical observation has been that better results are obtained through the use of a more explicit rule-based switching between modules.

1. Gating: Here expert modules are combined by means of a Gating net (e.g. [67] [68]). An auxillary Gating network is used to output a set of scalar coefficients that serve to weight the contributions of the various inputs; coefficients that vary as a function of the input.
2. Rule-based Switching: A form of rule-based switching between modules can be found in Brooks' account of goal subsumption, [51]. Here the switching between modules is triggered on the basis of the input. For instance, the detection by sensors of an obstacle in a robot's path would trigger a switch from a 'wander' modules to an 'obstacle avoidance' module. A related, but different form of rule-based switching can be found in [6] where control is switched between modules depending on the output of one of the modules. Two nets are separately optimised, one to make as few false positive errors as possible, and one to make as few false negative errors as possible. The output of the first net is used, unless it exceeds an empirically defined threshold, in which case the output of the second modules is used. (See [73] for a version of this approach, adapted for use with two distinct ensembles).

Sequential combination. Here it seems most likely that the mode of combination would be via the inputs and outputs of the nets, such that the output of an earlier net forms the input for the next. However this is not the only means of combination, and a number of studies (e.g. [74]; [52]; [53]) have exploited the potential of hidden unit representations, where, for example, a net is trained to map the hidden unit representation of one net onto the hidden unit representations of another net.

Supervisory combination. An example of one net supervising another can be found in [71], where one ANN module is trained to select the parameters of a second net, on the basis of observations of the effect of various parameter values on the performance of that net. Another supervisory relationship can be found in a paper by Kim and Bartlett [17], where a supplementary network is trained to predict the error of a main network using the input features and the output of the main ANN module.

1.3 The Chapters in this Book

The chapters in this book address several of the issues which have been reviewed here; part of the point of this review being to place the chapters in a wider context. The content of each chapter will be briefly summarised below.

Five of the chapters in the book are concerned with ensemble approaches to combining. The chapters by Breiman, and by Drucker, consider the ensemble combining approaches of bagging and boosting, comparing their effectiveness, and in Breiman's case in particular examining possible explanations for the greater effectiveness of adaptive resampling. These contributions are followed by two chapters, one by Opitz and Shavlik, and one by Hashem, which consider the idea of selecting neural nets for effective combination in an ensemble. In all of these chapters, an important subtext is the detrimental effect of error correlations on combining. This subtext is also evident in the chapter by Raviv and Intrator in which they present a method for creating nets that promotes diversity by means of a combination of noise injection and bootstrapping.

The next four chapters are all concerned with modular approaches. The first two provide an indication of some of the different circumstances in which modular systems might be developed; the first being concerned with psychological modelling, and the second with the application of speech recognition. The chapter by Fine and Jacobs compares alternative accounts of the combination of visual cues in human depth perception. The following chapter by Furlanello provides an account of a modular speech recognition system within which bootstrap error estimation methods are used to select the most appropriate model for a normalisation component module. The last two chapters both consider the automatic decompositon of a task into modular components. A chapter by Luttrell provides an account of the development of a

modular system by means of self-organisation, and compares its effectiveness to a non-modular approach. The final chapter, by Jacobs and Tanner consists of a review of mixture models, setting a detailed account of the modular Mixtures of Experts method in a wider statistical context.

Chapter 2. L. Breiman: Combining Predictors. In this contribution, Breiman considers a number of methods for combining predictors. As such, he is concerned only with ensemble combinations, and not modular combinations as defined above. His chapter is written from a wider perspective than that of neural computing, and considers the combination of *unstable* predictors in general, of which neural nets are one example, and decision trees another. Unstable predictors are so called because they are sensitive to small changes; perturbing the learning set results in a different predictor that will show different patterns of generalisation. Breiman begins with an account of bagging, and then discusses more recent alternatives such as boosting and adaptive resampling. He reports comparative results which demonstrate the advantages of Adaboost [12] over bagging, and explores possible explanations for this improvement.

Chapter 3. H. Drucker: Boosting Using Neural Networks. In this chapter, a detailed account is provided of a number of boosting algorithms, together with an account of an alternative method for constructing ensembles, namely bagging. The two types of boosting algorithm are first, boosting through filtering, and secondly the more recent adaptive boosting algorithm, Adaboost. Different versions of the Adaboost algorithm are described, the choice of version depending on whether the problem is one of classification or regression, and if classification, whether there are more than two classes. The chapter focuses on using neural networks to implement ensembles, but it also contains an account of their implementation in terms of decision trees, and a consideration of the relative merits of neural networks and decision trees. The detailed account of ensemble techniques is followed by reports of a number of empirical comparisons of boosting and bagging, using either neural nets, or trees, based on a number of different data sets.

Chapter 4. D.Opitz and J.Shavlik: A Genetic Algorithm Approach for Creating Neural-Network Ensembles. Opitz and Shavlik present an algorithm called ADDEMUP that uses genetic algorithms to search actively for ensemble members which generalise well, but which disagree as much as possible. The standard genetic operators, cross-over and mutation, are used to create new individuals from an initial set. The most fit members, (in terms of generalisation and disagreement, or diversity) then form the next generation, and the process is repeated until a 'stopping criterion' is reached. Once found, the ensemble members are recombined using weighted averaging. The algorithm can incorporate prior knowledge in order to create a more effective ensemble. Opitz and Shavlik report experiments on four real-world domains, in which the performance of ADDEMUP is compared to selecting the best network, and other ensemble methods (Bagging and Adaboost).

The results indicate that even though ADDEMUP does not involve varying the training set (the training set is held constant), the performance of the ADDEMUP algorithm is comparable to that of Bagging and Adaboost, and even outperforms them when use is made of prior knowledge.

Chapter 5. S.Hashem: Treating Harmful Collinearity in Neural Network Ensembles. Like Opitz and Shavlik, Hashem is also concerned with selecting a set of nets which will form an effective ensemble. However, his focus is on effective selection from a given pool of nets, rather than on the generation of nets which exhibit diversity. He explores the harmful effects that collinearity, or linear dependence, among the members of an ensemble may have on the effectiveness of an ensemble. The proposed 'treatment' for these effects is to select nets using an algorithm that reduces the collinearity between the ensemble members. Two selection algorithms are evaluated, one of which is based on examining the outputs of potential ensemble members, and one of which is based on examining their errors. The results indicate that when an ensemble is created on the basis of one of the selection algorithms, and is combined by means of optimal linear combinations, better performance is obtained than than choosing the best network, or taking a simple average of the component nets.

Chapter 6. K.Tumer and J.Ghosh: Linear and Order Statistics Combiners for Pattern Classification. Tumer and Ghosh provide an analytical (Bayesian) framework within which they examine the improvements that result from combining nets in ensembles. Their emphasis in this chapter is on ways of combining component nets, as opposed to creating nets for effective combining. They examine two methods of combining; linear and order statistics. Their analysis emphasises the effect on combining of the correlation between the component nets, and considers the effect of terms of the bias/variance of the decision boundaries obtained with respect to the Bayes optimal decision boundary. The authors present experimental results on a number of data sets, confirming the benefits of combining, illustrating the effect of correlation between the combiners, and showing that combining by means of order statistics produces results that are at least comparable to those of simple averaging.

Chapter 7. Y.Raviv and N.Intrator: Variance Reduction via Noise and Bias Constraints. Raviv and Intrator are concerned with the creation of nets for effective combining in an ensemble. They present an algorithm, termed 'Bootstrap Ensemble with Noise' (BEN), which is designed to foster the independence of the estimators to be combined. A variable amount of noise is added to the data set before using bootstrap sampling to assemble training sets. Nets are trained on these training sets, using a weight decay factor. The BEN algorithm is applied to the highly non-linear two spirals problem, and shown to produce good results. The relative contributions of noise, weight decay and ensemble averaging are considered, and the authors conclude that the best results are obtained when all three of these components

are used together. The results are presented and interpreted in the context of their relationship to the bias/variance error decomposition.

Chapter 8. I.Fine and R.Jacobs: A Comparison of Weak, Modified Weak, and Strong Fusion Models for Integrating Cues to Visual Depth and Shape. This contribution can be characterised as an example of a modular, as opposed to an ensemble approach. In this chapter, Fine and Jacobs are concerned with evaluating alternative models of visual cue combination. The alternative models differ in terms of the amount of modularity they involve, or the level at which the visual cues are fused - an issue which is closely related to the idea of sensor fusion. In human depth perception, visual cues can be assumed to be combined in a variety of ways. The three models simulated here are a weak fusion model; a modified weak fusion model; and a strong fusion model. In the strong fusion model it is assumed that the available visual cues are fused in an unconstrained manner, whereas in the weak fusion model and the modified weak fusion model a greater degree of modularity is incorporated, such that in the weak fusion model separately estimated predictions are combined (c.f. *decision fusion* above). The modularity involved here is one that arises as a consequence of the input data, as opposed to automatic or explicit decomposition, although as is apparent from the presentation of three models, there is more than one way in which such modularity could be structured. The three models are evaluated both in terms of the performance that results from them, and also in terms of their consistency with the known experimental results. The chapter provides an illustration of the way in which alternative modular systems could be used to perform a given task, and uses a different metric to assess their performance, namely the consistency with experimental data.

Chapter 9. C.Furlanello, D.Giuliani, S.Merler and E.Trentin: Model Selection of Combined Neural Nets for Speech Recognition. Another modular system is examined in this chapter, which considers the problem of model selection in the context of speech recognition applications. Model selection is required for the development of a normalisation module to be used to map the acoustic data spoken by a new speaker to corresponding observations in the training data of the recognition system. Such normalisation is used to avoid the decrease in performance that could otherwise result from an acoustic mismatch between training data and the data involved in actual use of a speech recogniser system. Such normalisation is accomplished by creating a number of local regressors (created either automatically, or) and combining them to obtain an effective global regressor. The chapter focuses on the use of bootstrap error estimation for finding the best net combination; the advantages of bootstrap error estimation being that it enables the selection of the best net combination without requiring additional acoustic material. The effectiveness of the approach is demonstrated in two speech recognition tasks, and results are compared for linear, Radial Basis Functions, and Multi-Layer Perceptron architectures.

Chapter 10. S.Luttrell: Self-Organised Modular Neural Networks for Encoding Data. In this chapter, an account is presented of the development of a modular system by means of self-organisation. As such, Luttrell's contribution provides an example of the modular structure that can arise from automatic decomposition, where communication between the modules is co-operative. The system is evaluated in terms of the level of performance that results from implementing a factorial encoder, where high dimensional data is separately encoded as a number of low-dimensional subspaces. The performance of a factorial encoding is compared to that of a joint encoding approach, by which the network acts as a single encoder module, encoding the entire input space. The circumstances under which better results can be expected for a factorial encoding are described.

Chapter 11. R.Jacobs and M.Tanner: Mixtures of X. This chapter provides a review of mixture models. The Mixtures of Experts model discussed earlier is an example of a modular approach, since it is based on the assumption that better results will be obtained as the result of a 'divide and conquer' approach whereby different regions of the data are dealt with by different experts. Jacobs and Tanner's review includes a detailed account of the Mixtures of Experts model, setting it in the statistical context of other mixture models, which include mixtures of exponential distributions, hidden Markov models, mixtures of marginal models, mixtures of Cox models, mixtures of factor models, and mixtures of trees.

The chapters, and the reviewed literature, in this book illustrate the vastness of the topic of multi-net systems. Whether a researcher is concerned with getting good neural network performance in an application domain, or understanding the behaviour of a combination of nets, or developing a model of a biological process, their work is likely to involve the development of some form of multi-net system. The chapters contained in this book offer insights into several of the major issues involved in such development.

Acknowledgement

Preparation of this chapter, and this edited book, was supported by EPSRC Grant No GR/K84257.