# 7 Integration of Heterogeneous Sources of Partial Domain Knowledge

Pedro Romero *promero@eecs.wsu.edu*
Zoran Obradović *zoran@eecs.wsu.edu*
Justin Fletcher *jfletche@xkl.com*

School of Electrical Engineering and Computer Science
Washington State University, Pullman, Washington, 99164-2752

**Abstract**

*This article explores the possibility of achieving better classification results using systems that integrate prior knowledge and learning from examples into a hybrid knowledge-based neurocomputing system. This integration is first discussed as a transformation of either the original problem's domain or its range. A domain transforming neural network model that starts from a single source of prior knowledge and grows incrementally as needed is introduced next. In this model two integration techniques are explored: (1) converting the expert system rule base into a corresponding neural network and then extending this network by constructive learning; and (2) embedding the pre-existing expert system directly into a constructive neural network learning system. Domain transforming and range transforming methods for integrating multiple prior-knowledge sources simultaneously are also considered. Various experiments carried out on the two-spirals and a financial advising classification problems showed that prior knowledge can indeed help improve the results obtainable by learning from examples alone. When integrating multiple sources of prior knowledge, a competitive neural network based integration technique significantly outperformed the individual classifiers, a symbolic based method and a cooperative neural network based technique.*

## 7.1 Introduction

Automatically classifying data into categories is an important problem in many real life domains. When data is two-dimensional, classification tasks can be simple for humans but still quite difficult for automatic systems (e.g. determining whether a point is inside or outside of nested spirals is a well-known benchmark problem for classification systems (Fahlman and Lebiere, 1990)). Higher-dimensional data

classification is typically challenging for humans too, and sometimes more accurate results are achievable by automatic classifiers (e.g. determining protein disorder from amino acid sequence (Romero et al., 1997)).

There are two traditional ways for a computer system to acquire knowledge required to perform classification tasks. The *knowledge based* approach translates information obtained from human domain experts into a form that is interpretable by a computer system (Hayes-Roth et al., 1983). This knowledge base is the core of an *expert system*, which emulates human decision-making skills. The alternative approach, called *machine learning*, is an attempt to extract knowledge directly from data (Dietterich and Michalski, 1983).

Each of these two approaches has its advantages and disadvantages. An expert system represents knowledge in symbolic form allowing relatively easy manipulation and incorporation of newly obtained knowledge. In addition, its classification engine may be readily interpreted by humans. A machine learning system is less dependent on human understanding of the phenomena and can be, in principle, applied to any domain with sufficient amount of available data. However, both approaches are based on strong modeling assumptions. Expert systems assume human understanding of the phenomena and availability of an expert capable of explaining domain knowledge to a computer programmer. The knowledge used to build an expert system is typically acquired from the expertise of many individuals and thus, it can be inconsistent and incomplete. Similarly, a data set used to build a machine learning system can be noisy, conflicting and sparse. Even if these problems are not present in a given data set, extracting complex nonlinear relationships directly from data through machine learning can still be a difficult nonlinear optimization task.

*Hybrid intelligent systems* (Michalski and Tecuci, 1994) that integrate knowledge extraction from data and use of existing alternative sources of domain specific knowledge have had considerable practical success (Drossu and Obradović, 1996; Fletcher and Obradović, 1993; Towell et al., 1990). Typical systems of this type use domain-specific rules, stochastic analysis, nonlinear dynamics, genetic algorithms, fuzzy logic or other approaches to complement limited training data information and create more accurate prediction systems.

An inspection of recent literature indicates that the most popular approaches for integrating multiple learning components are: (1) combining expert modules through various averaging scenarios; and (2) selecting the most competent local expert for any given example. The combining approach can potentially be used to reduce the variance of an unstable predictor without increasing its bias (Breiman, 1996). This is very attractive when applying neural network modeling in complex domains where these models are very likely to be unstable (Chan et al., 1996; Chenoweth and Obradović, 1996; Shimshoni and Intrator, 1996). In the selection approach each expert module tends to learn only a subset of the training data, thus devoting itself to a sub-region of the input space (Jacobs et al., 1991). This showed quite promising results when forecasting a non-stationary time series (Weigend et al., 1995).

In particular, current research has been directed towards systems in which neural network machine learning techniques are combined with expert systems in order to complement and enhance their capabilities (Kandel and Langholz, 1992; Medsker and Bailey, 1992). This combination has been carried out in several directions including:

**Transformational models.** Here, one type of system is transformed into another, i.e. either an expert system is transformed into a neural network or vice versa (Gallant, 1988; Samad, 1988). Neural nets are transformed into expert systems whenever knowledge documentation and justification facilities are needed. Conversely, an expert system can be transformed into a neural network when speed, adaptability and robustness are a priority.

**Fully-integrated models.** Here several systems share data structures and knowledge representation. The most common variation of this model is the connectionist expert system, in which symbolic nodes are connected by weighted links (Gallant, 1988). These systems are robust and have improved problem solving characteristics, but the complexity of the system makes it difficult to develop and maintain.

**Loosely-coupled models.** This is an integrational model in which neural network and knowledge based systems interact through shared data files. Examples include one system serving as a preprocessor, post-processor, co-processor or interface for the other (Benachenhou et al., 1990). These systems are easier to develop than more integrated models, allowing for the use of commercial software for both expert systems and neural networks, but at the cost of slower operation, redundancy in computational capabilities, overlap in data input requirements and high communication cost.

**Tightly-coupled models.** As in loosely-coupled models, this type of architecture uses independent neural net and expert systems. The difference is that here the interaction is by means of memory resident data structures, so communication and operation velocities are vastly improved. System development is not much more difficult than that for loosely-coupled systems, but redundancy is also here an issue (Gutknecht et al., 1991; Hanson and Brekke, 1988; Hendler and Dickens, 1991).

The approaches just explained are used in this work to develop knowledge-based neurocomputing systems that integrate neural network learning from examples with sources of partial domain knowledge obtained from human symbolic reasoning (henceforth called *experts*). This integration has been carried out in two ways:

**Embedding transformed experts.** This is a transformational approach in which the expert is converted into a neural network that serves as a starting point for learning from examples.

**Embedding experts directly.** In this approach the experts are embedded without modifications in the hybrid classification system, which means that no knowledge of their internal mechanisms is required. Thus, in this case, the definition of

an expert can be expanded to include any system used to encapsulate pre-existing knowledge: traditional expert systems, statistical prediction systems, nearest neighbor algorithms, inductive algorithms, computer programs, fuzzy logic, genetic algorithms, neural networks and the like.

In addition, the hybrid systems discussed in this study are organized into:

**Single expert expansion systems.** In this approach, all expert knowledge comes from a single source, which is further expanded through neural network learning.

**Multiple experts integration systems.** Here multiple, possibly heterogeneous, experts are integrated into a hybrid classification system.

The integration of partial domain knowledge sources can be viewed as a transformation of either the problem domain or its range, as illustrated in Section 7.2. Section 7.3 describes three single expert expansion systems, two being transformational, and one fully-integrated model that embeds an expert directly. Three multiple experts integration techniques that embed prior knowledge sources directly are presented in Section 7.4. Finally, Section 7.5contains experimental comparison of these models on two classification problems.

## 7.2 Experts Integration: Domain or Range Transformation Dilemma

A classification problem can be viewed as identification of an appropriate mapping from a given *domain* to a *range* of categories. For example, a classification problem that takes a pattern composed of $n$ real numbers as input and tries to classify it as belonging to one of two classes, defined as "0" or "1", can be seen as a mapping:

$$\Re^n \rightarrow \{0, 1\}$$

where $\Re^n$ is the domain of the mapping and {0,1} represents its range.

When integrating prior knowledge sources, the classification problem is hopefully simplified by transforming its corresponding mapping. This can be achieved by modifying either the problem domain or its range as discussed in this section.

### 7.2.1 Domain transformation

When using the outputs of several experts as inputs to a new classifier (called a *combiner*), the original input space is transformed to an entirely new one, its dimensionality being defined by the number of experts used and their output representation.

For example, suppose $K$ experts are to be combined to solve an $n$-input, $m$-class classification problem. Assuming that each expert uses $m$ outputs, the combining classifier will have $mK$ inputs, thus transforming the problem domain from $n$

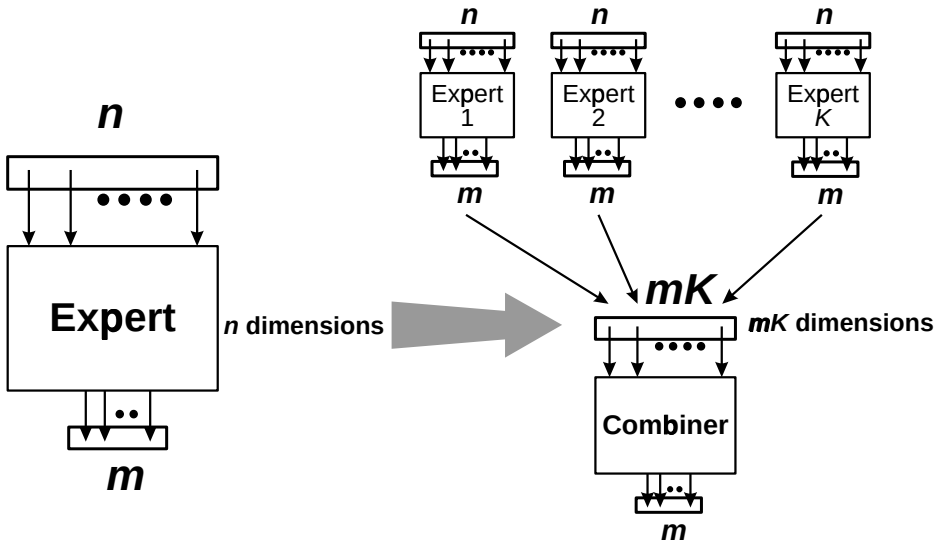



**Figure 7.1** Integrating $m$ experts: domain transformation from $n$-dimensional to $mK$-dimensional.

dimensions onto $mK$ dimensions. Figure 7.1 shows schematically how this process works.

When transforming the problems domain as explained, the hope is that it will be easier to identify an appropriate mapping using the new problem domain. This can be the result of one or more of the following reasons:

**Reduced domain dimensionality.** This results in models with less parameters, which, due to the curse of dimensionality, can be a significant advantage when designing a model from a limited data set (Bishop, 1995).

**Input pre-processing.** The domain transformation can be regarded as a feature extraction process (Fukunaga, 1990). This can help eliminate the effect of irrelevant or noisy input variables.

**Simpler decision regions.** Even if the original dimensionality is not significantly reduced through a transformation, classes in the new domain may be easier to separate due to more favorable clustering of the patterns.

It is important to observe that information can be lost when transforming the domain, resulting in a poor classifier. This is especially true for experts with low-resolution output representations (e.g. one bit representation for each output dimension). This is illustrated in Figure 7.2, where two experts, whose outputs are one-bit numbers, are to be combined. In that figure, class 0 and class 1 patterns are represented as circles and crosses respectively. The domain is partitioned into regions according to the experts' outputs, shown as pairs $(a, b)$ for each region.

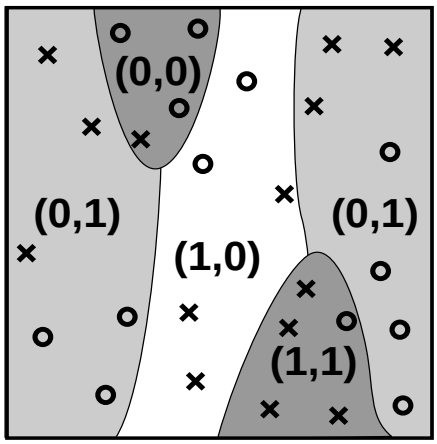



**Figure 7.2** Decision regions for two experts with binary outputs.

Notice that each pair $(a, b)$ represents an input vector for the combiner. In this example, the transformed domain is not faithful in any of the regions, meaning that for each region there are examples that belong to different classes and that are mapped to the same input vector for the combiner. This information loss makes it impossible for a combiner to resolve the problem completely.

### 7.2.2 Range transformation

In this case, a classifier is trained to integrate the various experts by partitioning the domain and assigning disjoint regions to individual experts. Here, the original problem domain does not change, but the integrating classifier has a different class range defined by the number of experts.

Suppose that $K$ experts are to be integrated in this manner. The problem is reduced to assigning each expert exclusively to its "region of expertise", that is, a sub-domain where a given expert's classification performance is superior to that of the other experts. So, the original $m$-class problem is transformed into a new problem with $f(K)$ classes.

For $K = 2$, the classifier could be trained to recognize the following possibilities:

| | |
|---|---|
| Class 0 | None of the local experts classifies this example correctly; |
| Class 1 | Expert 1 alone classifies the example correctly; |
| Class 2 | Expert 2 alone classifies the example correctly; |
| Class 3 | Both local experts classify the example correctly. |

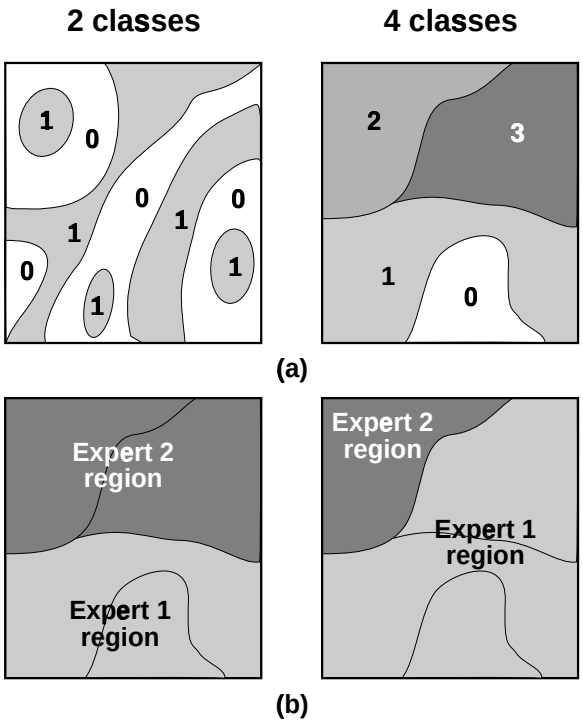



**Figure 7.3** **(a)** Hypothetical range transformation for the integration of two experts in a 2-class problem. **(b)** Two of the possible distributions of the input space between the two experts.

Figure 7.3(a) shows a schematic view of the input space of a 2-class problem and a possible range transformation into a 4-class problem according to the method just explained for $K = 2$ experts. Notice that in this case the number of classes is *increased*, but this is done anticipating that the new decision regions are simpler in the transformed problem than in the original one. Whether or not this will be the case depends on the characteristics of the problem and the quality of the experts used.

This transformation leads to $f(K) = 2^K$, so the number of classes grows exponentially with the number of experts used. A better alternative for this example would be to use only 2 classes:

Class 1: Use expert 1 to classify the example;

Class 2: Use expert 2.

However, the new problem here is that of assigning the patterns for which both experts are correct (or incorrect). Figure 7.3(b) depicts two of the possible ways to distribute the input space between the two experts. Notice that any decision boundary laying within the region where both experts are correct (class 3 on the right hand side of Figure 7.3(a)) is acceptable for this distribution: Figure 7.3(b) just shows the extreme cases, that is, assigning the whole "class 3" region to either

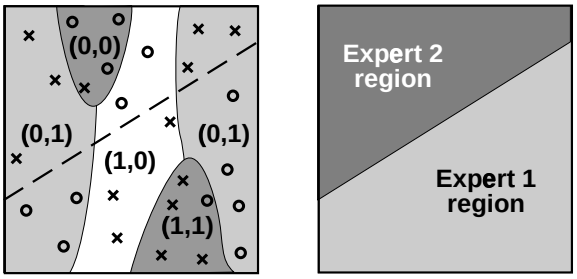



**Figure 7.4** Range transformation for the Figure 7.2 problem.

expert. Similarly, the "class 0" region could have been assigned to expert 2 or even distributed randomly between both experts. In any event, the decision regions thus created would have been a lot more complex. In a problem with more dimensions or experts, this assignment task gets even more complicated.

Hybrid systems relying on range transformation can provide various advantages over original classifiers, specifically the following:

**Simpler decision regions.** As in the domain transformation case, it is anticipated that the transformed problem will have simpler decision regions than the original one, thus being easier to solve and requiring a less complex classifier to do the job.

**Restricted domain for local training.** Assigning a portion of the domain to a specific classifier allows local training of machine learning based experts on the remaining region or regions. Examples of this approach are studied in Sections 7.3 and 7.4.

**Insensibility to output representation.** Range transformation works in the same way regardless of the experts' output representation. Indeed, the technique can even work with experts whose output representations differ from one another. Figure 7.4 shows the same hypothetical problem shown in Figure 7.2, but this time the range transformation produces two distinct and easily separable regions corresponding to the "areas of expertise" of each expert. Remember that this problem was impossible to solve appropriately when transforming the domain.

## 7.3 Incremental Single Expert Expansion

In (Towell et al., 1990), the authors have proposed a knowledge-based artificial neural network approach called *KBANN*, that generates neural networks from hierarchically structured rules. In these neural networks units correspond to rules, while connection weights and thresholds correspond to rule dependencies (as will be demonstrated on a financial advising example in the results section). Each layer is then fully connected with the weights of the new links set to small random values. Finally, the initial knowledge is refined by neural network training from examples

using the backpropagation algorithm (Werbos, 1995).

The "hyperplane determination from examples algorithm" (HDE) proposed at (Fletcher and Obradović, 1995) belongs to the class of *constructive algorithms*. These algorithms, in addition to optimizing the model's parameter values, also search for an appropriate neural network topology by growing hidden units in a greedy optimization manner. In contrast to the well known cascade correlation technique (Fahlman and Lebiere, 1990), which grows neurons in depth, HDE is used to construct a 2-layer neural network starting from a single hidden unit and adding new hidden units as necessary.

This algorithm can be used to build a hybrid classification system by using an expert to build the initial neural network and then applying HDE to add new hidden units to improve classification performance. Similar to the KBANN technique, the expert can be converted into a rule base and then transformed into a neural network (Fletcher and Obradović, 1993). Alternatively, the source of prior knowledge can be used in a "black box" fashion by treating it as a single hidden unit (called an *expert unit*). A more detailed explanation of this domain transforming hybrid model is provided in this section.

### 7.3.1 The HDE algorithm

An interesting iterative construction of hidden units in a feed-forward neural network with a single hidden layer was proposed at (Baum, 1991; Baum and Lang, 1991). This algorithm constructs a two layer neural network given examples and the ability to query an oracle for the classification of specific points within the problem domain. The algorithm is very efficient, but in practice the required oracle may either be too expensive or not available.

Inspired by this work on learning from queries, our HDE algorithm constructs the hidden units in a feedforward neural network from examples alone (Fletcher and Obradović, 1995). Construction of the HDE neural network is performed in three phases:

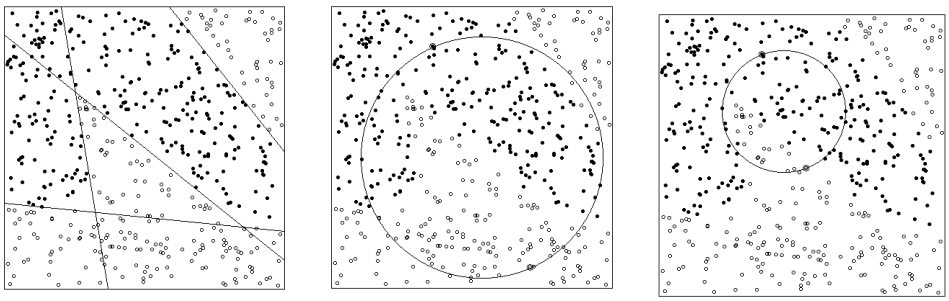
1. Determination of points on the decision boundary;

2. Generation of a pool of candidate hyperplanes from the obtained points; and

3. Selection of the final separating hyperplanes from the candidate pool and creation of hidden units from selected hyperplanes.

These phases will be described using as an example the construction of a neural network approximating the ideal decision boundary shown in Figure 7.5(a). For simplicity of explanation, we will assume that the examples from the training set $T$ shown in the figure belong to two classes $T_1$ and $T_2$.

For all pairs of training examples belonging to different classes, a search for corresponding points on the boundary separating those examples is performed. Approximations to points on the decision boundary are determined by repeatedly interpolating between example points of the classes $T_1$ and $T_2$. The interpolation



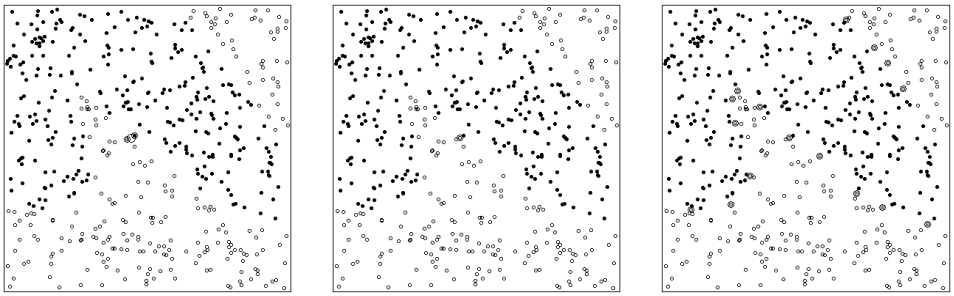
(a) (b) (c)

**Figure 7.5** HDE algorithm example: (a) ideal decision region; (b) initial unknown region, (c) next unknown region.

begins by selecting two examples $m \in T_1$, $n \in T_2$. The *unknown region* between $m$ and $n$ is defined as the circle centered at the midpoint of $m$ and $n$ with a diameter of the distance between $m$ and $n$, as shown in Figure 7.5(b). The unknown region between $m$ and $n$ is then searched for the training example nearest to the midpoint of $m$ and $n$. If such an example $q$ is found and $q \in T_1$ ($T_2$) the search is then repeated in the smaller unknown region between $q$ and $n$ ($m$). The next unknown region is shown in Figure 7.5(c).

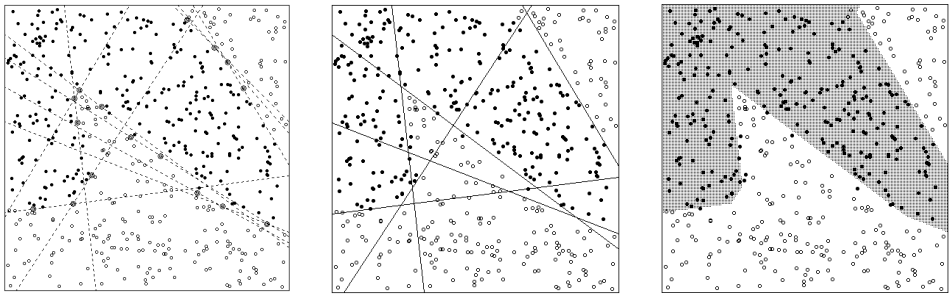
If no point from $T$ is found in the current unknown region (Figure 7.6(a)), its midpoint is the closest approximation to a point on the decision boundary (Figure 7.6(b)). If the radius of this known region is within a specified tolerance, the boundary point is stored providing it has not been previously determined. Boundary points continue to be generated until a pre-determined number have been found or a number of data points have been examined without finding a new point on a decision boundary. The resultant boundary points are shown in Figure 7.6(c).

Once the points on the decision boundary have been found, their $k$-1 nearest



(a) (b) (c)

**Figure 7.6** HDE algorithm example: (a) last unknown region; (b) a point on the decision boundary; (c) more points on the decision boundary.



(a) (b) (c)

**Figure 7.7** HDE algorithm example: (a) candidate hyperplanes; (b) selected hyperplanes; (c) resultant decision boundary.

boundary points are determined. As previously, $k$ is the domain dimensionality. A pool of hyperplanes is then determined through solution of the equation system defined by each set of the $k$ boundary points. Figure 7.7(a) shows such a candidate pool with their associated boundary points.

The first hidden unit is created from the candidate hyperplane which best classifies the training data. This hyperplane is then removed from the candidate list. Each remaining hidden unit is created by evaluation of the remaining candidate hyperplanes in conjunction with the previously created hidden units. This is accomplished by creating a hidden unit and iteratively setting the input layer connection weights to the corresponding equation of each of the candidate hyperplanes. The output layer weights for a candidate for the next intermediate network are then determined by learning from the training examples using the ratcheted pocket algorithm (Gallant, 1990). This procedure continues until no candidate hyperplane results in a significant improvement in classification on the training set.

A final selection of hidden layer hyperplanes is shown in Figure 7.7(b) with the resultant decision boundary depicted in Figure 7.7(c).

### 7.3.2 Embedding of transformed prior knowledge

Here, a symbolic expert is transformed into a neural network as in the KBANN method. However, in a neural network obtained from the original rule-base, the KBANN stage in which the layers are fully connected is omitted. Instead, the HDE algorithm is used to add neurons to the last layer of the initial neural network. The output weights of the starting network are modified after each new hidden unit is added through HDE construction.

The obtained hybrid system is a transformational model in which a symbolic rule base is required to build the original neural network using the KBANN technique. This means that in this case it is not possible to use an arbitrary type of expert since the knowledge source has to be a known AND/OR structured rule base.

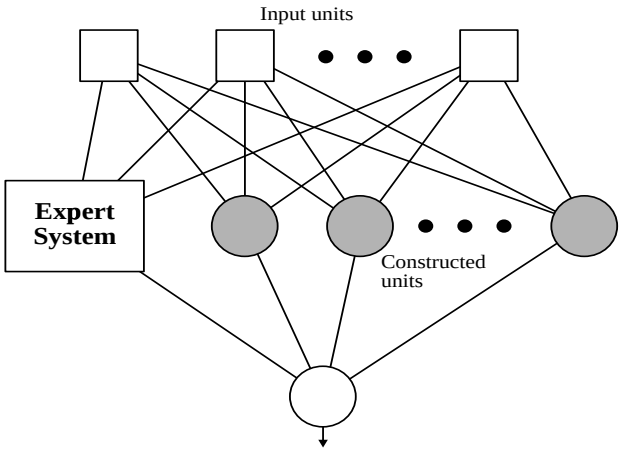


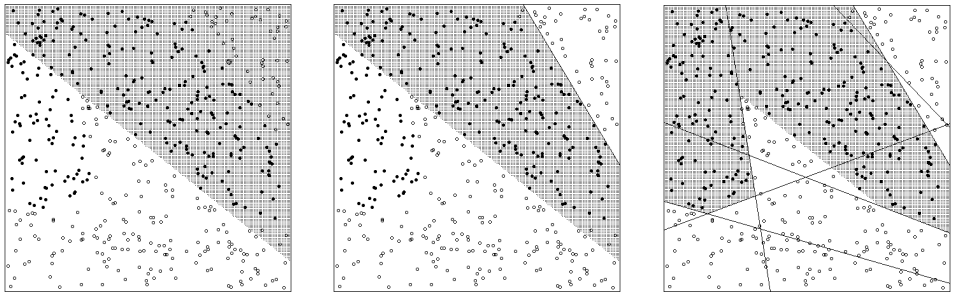

**Figure 7.8** Direct integration of an expert into HDE construction.

### 7.3.3 Direct integration of prior knowledge

In this approach, no transformation of the expert system is required. Instead, a hybrid architecture is constructed with the expert system directly embedded into the neural network as shown in Figure 7.8.

The three-phase process of the HDE algorithm is followed with certain small exceptions. An initial network consisting of the input layer, a single hidden unit and an output unit is created. The hidden unit is designated as an *expert unit*, which, instead of computing the usual activation function, calls the expert system to determine the unit's output. The expert unit, then, acts as the initial hidden unit, contributing with its decision boundary, such as the simple one shown in Figure 7.9(a).

As before, a set of candidate hyperplanes is built, and they are tested as new hidden units in conjunction with the original one (the expert unit) in order to create an intermediate network. The first hyperplane added in this example is shown in



(a) (b) (c)

**Figure 7.9** Direct expert integration example: (a) first modification of the decision boundary; (b) next decision boundary; (c) final decision boundary.

Figure 7.9(b).

Any remaining hidden units are then created by evaluating each of the remaining candidate hyperplanes with the intermediate hybrid network. This process continues until either the candidate pool is exhausted or no significant improvement is gained by integrating any candidate hyperplane. The final output layer weights are again determined through use of the pocket algorithm. A final decision boundary is shown in Figure 7.9(c).

In contrast to this simple example, an expert can contribute with a more complex decision boundary that helps reduce the number of hidden units needed in the integrated system as compared to that of a simple neural network constructed similarly. This process has the same effect as an input space reduction through range transformation as explained in Section 7.2.2, that is, the generation of new hidden units is carried out only in the regions of the input space where classification based on prior knowledge is unsatisfactory, effectively reducing the input space to be solved. On the other hand, the hidden units act as experts, producing a domain-transformed problem for the output layer to solve.

This classifier is an example of a fully integrated hybrid system with direct expert embedding. Since the expert is not modified in any way, any kind of classifier can be used as an expert neuron.

## 7.4 Multiple Experts Integration

Three techniques designed to integrate several sources of prior knowledge are discussed in this section.

### 7.4.1 Cooperative combination of heterogeneous experts

The domain transformation through cooperative combination of multiple experts has been successfully applied to a number of real-life classification problems (e.g., protein structure prediction (Zhang et al., 1992)).

In this article, a feedforward neural network is used as a combiner for $K$ *local experts* as shown in Figure 7.10. Here, each example for the combiner training process is constructed from an input vector $\mathbf{x}$ and a desired response vector $\mathbf{y}^*$. First, a vector $\mathbf{z} = [\mathbf{y}_1 \mathbf{y}_2 \ldots \mathbf{y}_K]$ is assembled using the output vectors $\mathbf{y}_1, \mathbf{y}_2, \ldots$, $\mathbf{y}_K$ obtained from the $K$ experts when presented with the input vector $\mathbf{x}$. Second, vector $\mathbf{z}$ is fed to the combining neural network, which finally produces the output vector $\mathbf{y}$.

The combining module is trained by means of the backpropagation learning algorithm (Werbos, 1995), using the $\mathbf{z}$ vectors as inputs, and the desired response vectors ($\mathbf{y}^*$) as targets. Once the combiner is trained, the whole system can be used, as shown in Figure 7.10, to classify new patterns.

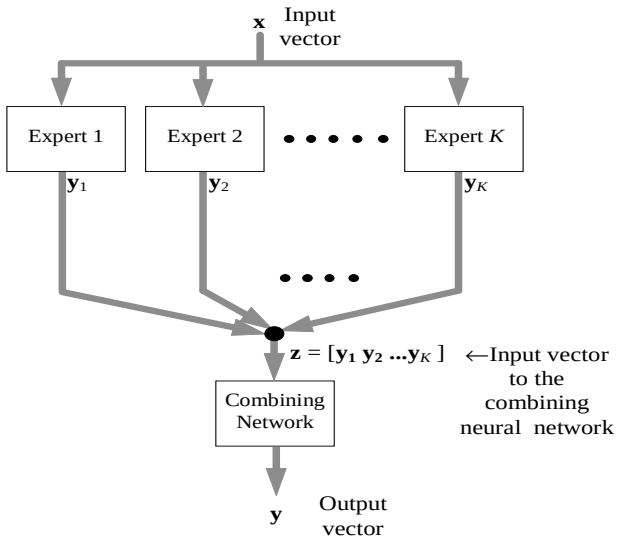



**Figure 7.10** Neural network based cooperative combination.

### 7.4.2 Symbolic integration using decision trees

This technique, considered in (Romero and Obradović, 1995), trains a decision tree to select among the $K$ local experts to be integrated. Because there is no simple way to decide how to assign the patterns for which none, all or several experts make a correct classification, the decision tree has to learn a $2^K$-class problem. This corresponds to a range transformation, as explained in section 7.2.

Figure 7.11 shows an implementation of the proposed system. Again, each training pattern consists of an input vector $\mathbf{x}$ and a desired response vector $\mathbf{y}^*$. Each local expert outputs a response $\mathbf{y}_i$ which is fed to a selection routine. This class information is used for the decision tree generation.

The selection routine compares the local experts' outputs ($\mathbf{y}_i$) to the desired response $\mathbf{y}^*$. Then, it assigns the input $\mathbf{x}$ to a desired class $\mathbf{z}^*$. Thus, the decision tree is generated from modified patterns, as illustrated in Table 7.1.

| **Classifier** | **input vector** | **output (class) vector** |
|---|---|---|
| Expert | $\mathbf{x}$ | $\mathbf{y}$ |
| Decision tree | $\mathbf{x}$ | $\mathbf{z}$ |

**Table 7.1** Range transformation through decision tree integration.

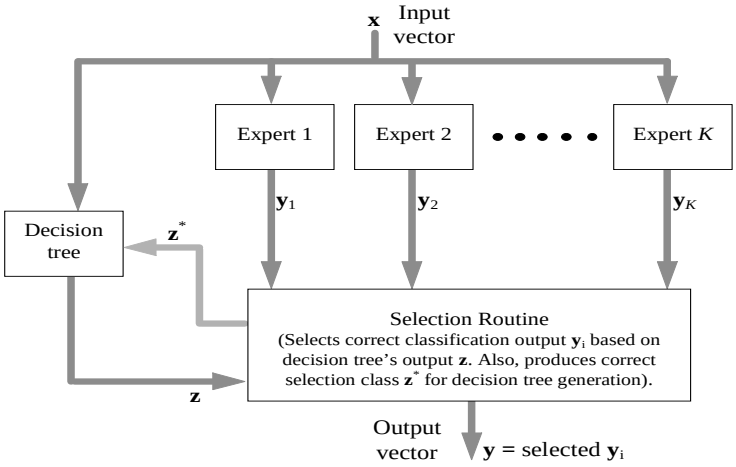



**Figure 7.11** Decision tree symbolic integration.

After the three is generated, the system works as follows: The input vector $\mathbf{x}$ is fed to the decision tree and to all local experts. The decision tree produces a response $z$ and each expert $i$ outputs a response $y_i$. All these responses are given to the selection routine, which, based on the value of $z$, selects one of the $y_i$'s. The selected value is then output as the system's response.

When all experts are wrong, the system's output can be generated randomly or inferred from the data. For example, when solving a binary classification (2-class) problem using 2 experts that are both incorrect on a given example, the system has to output the opposite class from that selected by both experts.

The construction of the decision tree can be carried out using various decision tree generation methods. In this study, GID3* (Fayyad, 1994) was used. This technique groups together irrelevant attribute values before generating the three using the ID3 algorithm (Quinlan, 1986). The classification problem addressed in this paper has real valued attributes which have to be discretized before using GID3*. This is achieved by using a multiple-interval discretization technique proposed in (Fayyad and Irani, 1993). The common approach is to discretize the attributes at each node of the decision tree. For the problem studied here, we found that a single discretization step performed on each input variable before tree generation always achieved better generalization, and so the results reported here were obtained using the latter technique.

### 7.4.3 Competitive integration of heterogeneous experts

The competitive integration of heterogeneous experts, proposed in (Romero and Obradović, 1995), is an extension of the mixture of local experts architecture (Jacobs et al., 1991) which uses a *gating network* to integrate the responses of

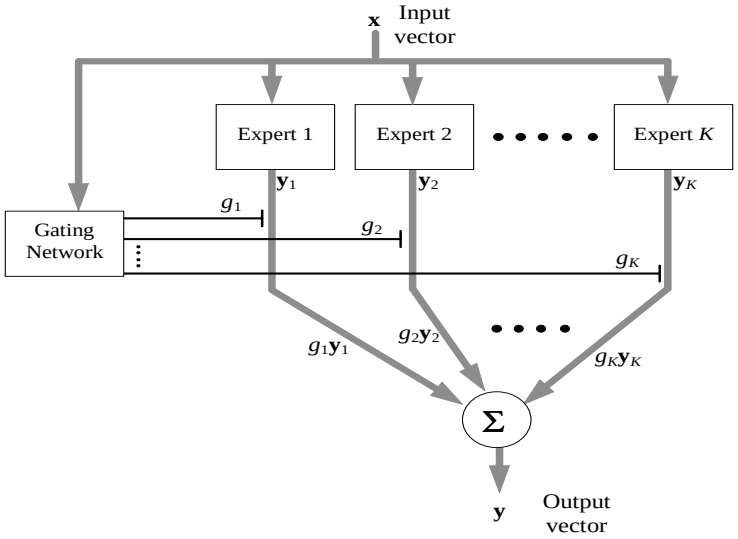



**Figure 7.12** Competitive neural network based integration.

all local experts, selecting the most competent local expert for any given example (see Figure 7.12).

In the original architecture from (Jacobs et al., 1991) all $K$ local experts are backpropagation neural networks. A supervised learning process is carried out using a set of training examples, each consisting of an input vector $\mathbf{x}$ and a desired response vector $\mathbf{y}^*$. In the basic model, the input vector $\mathbf{x}$ is applied to both the local expert networks and the gating network. The gating network used in this study is a one-layer feedforward neural network whose output units use a *softmax* activation function

$$g_i = \frac{e^{s_i}}{\sum_{j=1}^{K} e^{s_j}}$$

where $s_i$ is the weighted input sum of the $i$th output unit. This activation function ensures that the system's output $\mathbf{y} = \sum_{i=1}^{K} g_i \mathbf{y}_i$ corresponds to a weighted average of the individual expert's outputs $\mathbf{y}_1, \ldots, \mathbf{y}_K$. It is interesting to notice that the softmax function is a continuous (differentiable) version of the "winner take all" selection criteria, suitable for use in a gradient descent technique such as backpropagation.

All experts use a special error function

$$E = -\ln L = -\ln \sum_{i=1}^{K} l_i = -\ln \sum_{i=1}^{K} g_i e^{-\frac{1}{2\sigma_i^2}\|\mathbf{y}^* - \mathbf{y}_i\|^2}$$

where $\ln L$ represents the log likelihood of generating a response vector $\mathbf{y}^*$, and $\sigma_i^2$ is a scaling term. The summation term is called $l_i$ for clarity purposes. The

system is trained as to minimize $-\ln L$ (maximize the log likelihood), which allows a competitive learning process by training only the most competent local expert(s) on a given example. This is best understood when examining the last hidden layer weight update term for the $i$th expert network:

$$(\Delta w_{jk})_i = \eta \delta_{ji}(g_i) o_k$$

where $w_{jk}$ is the weight of the connection between hidden layer unit $k$ and output unit $j$, $\eta$ is the learning rate, $o_k$ is the output of hidden unit $k$, and $\delta_{ji}(g_i)$ represents the back-propagated error for output unit $j$ of expert $I$ ($\delta_{ji}$), which is a function of $g_i$. It can be seen that the $i$th expert network weight change is dependent on $g_i$, so only the networks selected for a given example (those with $g_i$ greater than 0) will have their weights updated, i.e. will "learn" that example.

The weights' update for the gating network is given by

$$\Delta \mathbf{u}_i = \eta(g_i - h_i)\mathbf{x}$$

where $\mathbf{u}_i$ represents the weight vector associated to output unit $I$, and $h_i = \frac{l_i}{L}$. In a statistical sense, the $g_i$s can be regarded as *prior* probabilities of selecting the $i$th expert, while the $h_i$s represent *posterior* probabilities of expert $i$ generating the desired output vector. Thus, as the gating network learns, the prior probabilities of selecting an expert move towards the posterior probability of that expert generating the desired response.

In an extended model, the gating network can receive an additional input $\mathbf{x}'$ either in conjunction with, or instead of the expert networks' input $\mathbf{x}$. Using an additional input for the gating network could be useful as a "hint" on the correct distribution of experts. For example, the gating network might work better if provided with the sex of the speaker in a vowel recognition problem (Nowlan and Hinton, 1991).

In the competitive integration system used in this paper we assume that the local experts can be not only neural networks, but also various sources of prior knowledge (Romero and Obradović, 1995). We will also assume that only the gating network and any neural network components do the learning as explained above, while the other expert components are fixed and can only be used to respond to the input patterns.

## 7.5 Results

The systems discussed in previous sections are evaluated in the context of two quite different benchmark problems. The first problem has a 2-dimensional domain, but it requires generation of an extremely complex decision region, whereas the second one is six-dimensional, but it can be solved by using a classifier with a simpler decision region. Classification results of various hybrid systems applied to these two problems are summarized in this section.

Some common parameters have been defined in order to standardize comparisons.

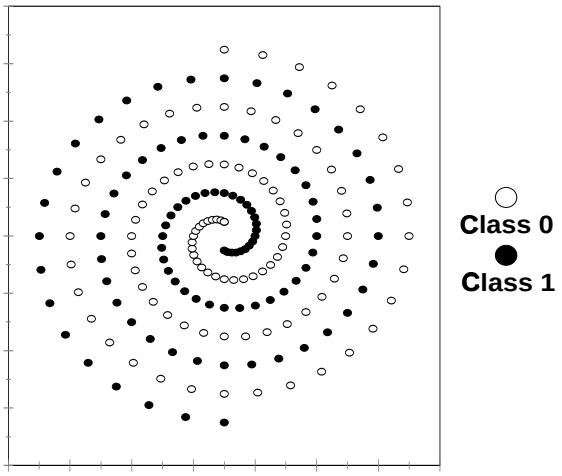



**Figure 7.13** The two-spirals problem.

For the HDE-based approaches' default configuration, the maximum number of points on the decision boundary is set to ten times the dimensionality of the input space, or stopped if one thousand example pairs are examined without determining a new boundary point. Gallant's parameter recommendations for the pocket algorithm, used for determination of the final output layer weights, are followed (ten thousand initial iterations, increased by fifty percent if the pocket is updated in the final eighty percent of the iterations). The learning rate for the pocket algorithm is standardized at 30 percent, and no additional hyperplanes are selected if the overall classification improvement is less than 0.5 percent. The gating neural networks used on the competitive integration experiments are all single layer networks, i.e., with no hidden units. Gating networks can in principle be multi-layered, but the principal idea was to show how problem transformation can simplify a classification mapping so that a basic classifier is able to achieve good integration.

### 7.5.1 The two-spirals problem

The two-spirals problem was proposed by A. Wieland of MITRE Corporation as a task constructed to test the ability of neural networks to deal with complex decision regions. This well known benchmark (Fahlman and Lebiere, 1990) differs from many others by testing only the memorization ability, rather than the ability to generalize over the problem domain. The input space consists of two dimensional data points arranged into two spirals on the x-y plane. This is a 2-class problem: all points on a given spiral are of one class, while the points on the other spiral belong to the opposite class, as shown in Figure 7.13.

#### 7.5.1.1 Sources of partial domain knowledge

Both the incremental construction and the competitive integration approaches were applied to this problem in order to illustrate the effects of using sources of partial domain knowledge to construct improved classifiers. To simulate partial domain knowledge, it is assumed that a human expert is under the impression that the class of a given point depends on its polar radius, that is, on its distance to the origin. Using this assumption, several experts were developed. Based a distance metric, the experts classify a given data point as follows:

$$Class = \begin{cases} 0 & \text{if } dist \bmod 2 < 1 \\ 1 & \text{otherwise} \end{cases}$$

Each of the various experts used in this work employs a different distance metric $dist$, as defined in Table 7.2, where $x$ and $y$ refer to the pattern's coordinates in the 2-dimensional input space. As an example, Figure 7.14 shows the decision regions for Expert 0 from Table 7.2. Although the global classification rate for this expert is only 50%, it actually contains information. In fact, this expert correctly classifies points lying above the $x$ axis (horizontal line in Figure 7.14). Similarly, for experts 1, 2 and 3 it is also possible to identify regions of the input space where they are reasonably good classifiers.

| **Experts** | **Distance metric** | Success rate |
|---|---|---|
| Expert 0 | $dist = \sqrt{x^2 + y^2}$ | 50.00% |
| Expert 1 | $dist = \sqrt{(x + 0.5)^2 + (y + 0.5)^2}$ | 32.99% |
| Expert 2 | $dist = \sqrt{x^2 + y^2} + 0.5$ | 53.61% |
| Expert 3 | $dist = \sqrt{x^2 + y^2} + 1.0$ | 53.09% |

**Table 7.2** Distance metrics for the two-spirals experts used in this work.

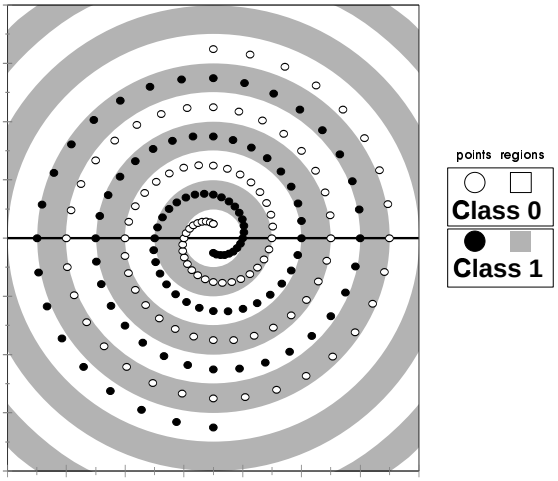
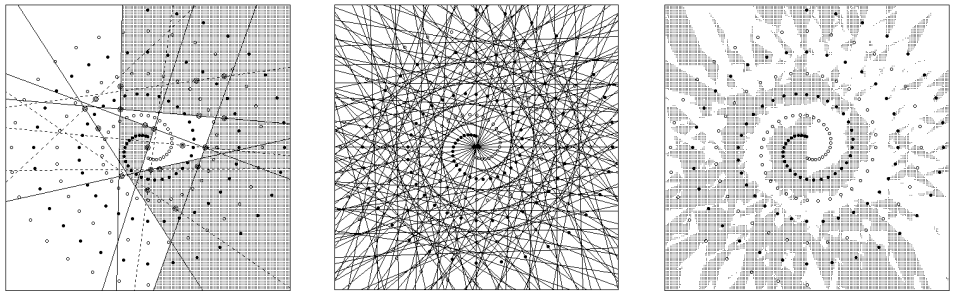



**Figure 7.14** Two-spirals: decision regions for expert system "0" from Table 7.2.

#### *7.5.1.2 Incremental model construction*

The results summarized in Table 7.3 and the corresponding decision boundary shown in Figure 7.15(a) were obtained by the HDE algorithm using the default learning parameters. This may be viewed as rather dismal results especially if compared to cascade-correlation, which reports 100% classification using between 12 and 19 units (Fahlman and Lebiere, 1990). However, these results may be somewhat improved if three additional steps are taken. First, the number of decision boundary points is not limited to ten times the problem dimensionality but instead continue to be generated until one thousand pairs are examined without generating a new boundary point. This results in a significantly larger candidate hyperplane pool (Figure 7.15(b)). Second, the hyperplane selection phase is eliminated as a hidden



(a) (b) (c)

**Figure 7.15** HDE algorithm application on the two-spirals problem: (a) decision boundary using default learning parameters; (b) candidate pool of decision boundaries; (c) final decision boundary.

| | Average | Min. | Max. |
|---|---|---|---|
| Boundary Points | 20 | 20 | 20 |
| Candidate Hyperplanes | 15.7 | 18 | 18 |
| Hidden Units | 4.3 | 1 | 7 |
| Accuracy | 61.49 | 56.19 | 65.98 |

**Table 7.3** Two-spirals: default HDE network construction.

| | Average | Min. | Max. |
|---|---|---|---|
| Boundary Points | 159.6 | 157 | 163 |
| Candidate Hyperplanes | 152.1 | 150 | 156 |
| Hidden Units | 152 | 152 | 152 |
| Accuracy | 99.84 | 98.45 | 100.0 |

**Table 7.4** Two-spirals: modified HDE network construction.

unit is constructed for each candidate hyperplane. Finally, the initial number of iterations of the pocket algorithm during final output layer weight training is increased from ten thousand to twenty-five thousand. Table 7.4 shows the new results with a representative decision boundary shown in Figure 7.15(c).

While this results in near-perfect classification (one of the ten experiments resulted in 98.45% accuracy), the algorithm generated a very large network architecture, which is not likely to generalize well on new data. A preferred approach would be to integrate an existing knowledge base in order to reduce the complexity of the classifier.

As shown in Table 7.2, Expert 0 has a success rate of 50%. If we embed this expert, hyperplanes are selected in such a fashion as to take advantage of the areas where the expert successfully classifies the input space (the region above the horizontal line on Figure 7.14). Table 7.5 shows an improvement in classification ability of over ten percent when integrating Expert 0 as compared to the original HDE algorithm (Table 7.3). The default learning parameters were used in both cases.

A number of items are made apparent by this benchmark. The basic approaches

| | Average | Min. | Max. |
|---|---|---|---|
| Boundary Points | 20 | 20 | 20 |
| Candidate Hyperplanes | 15.7 | 13 | 18 |
| Hidden Units | 3.5 | 3 | 6 |
| Accuracy | 74.43 | 68.56 | 76.29 |

**Table 7.5** Two-spirals: Expert 0 + default HDE hybrid network construction.

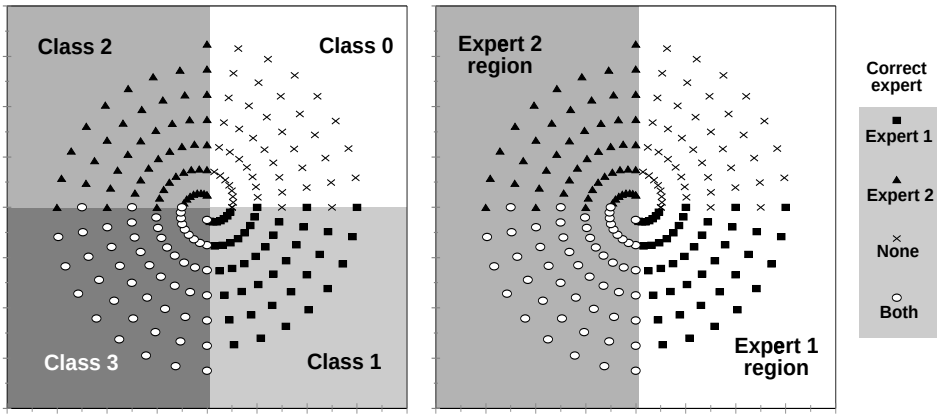



**Figure 7.16** Integration of experts 0 and 2 for the two-spirals problem: (left) 4-class decision region; (right) one of the possible experts' assignment.

to determining decision boundary points and constructing the candidate hyperplane pool appear to be appropriate, but the proper number of decision boundary points will vary with the problem. It is also apparent that process parameters may need to be adjusted for each individual problem. However, an important point made by this benchmark is that the integration of two techniques which do not perform well independently may result in improved classification when combined into a hybrid system.

#### *7.5.1.3 Multiple experts integration*

The experts shown in Table 7.2 can be combined in order to obtain a better classifier. Due to the fact that these experts have single one-bit outputs, this example is not suitable for domain transformation approaches as explained in Section 7.2. (Figure 7.2). Indeed, the very nature of the decision regions generated by these experts (see Figure 7.14) guarantees that different patterns from each of the classes will be transformed into identical input vectors for the combiner.

On the other hand, range transformation approaches can work very well on this problem. Figure 7.16, corresponding to the integration of Expert 0 and Expert 2, shows graphically how the use of local experts can simplify the decision regions on the input space. As explained in Section 7.2, the integration of two local experts can result in the input space being partitioned in up to four different decision regions, shown in the left picture of Figure 7.16. These regions correspond to the ones shown in Figure 7.3(a), that is: (0) data points misclassified by both experts; (1) data points correctly classified only by the first expert; (2) data points correctly classified only by the second expert; and (3) data points correctly classified by both experts. The right hand side of Figure 7.16 shows one possible domain partitioning between the two experts in the manner illustrated in Figure 7.3(b).

| Local experts | Success rate on training data | |
|---|---|---|
| | Upper bound | Integrated system |
| 0+1 | 66.49% | 66.49% |
| 0+2 | 76.80% | 76.80% |
| 0+3 | 100.00% | 100.00% |

**Table 7.6** 2-Spirals Problem: Success rates for local experts and their combinations.

The competitive integration technique for multiple experts integration was tested on this problem. The experiments summarized in Table 7.6 were performed by integrating Expert 0 with one of the other experts shown in Table 7.2. The upper bounds for the success rate are measured as the maximum accuracy obtainable by combining experts perfectly, that is, always selecting the best expert for the job. The difference from 100% corresponds to the examples that neither expert classified correctly. As it can be seen in the table, the competitive integration system always achieved the maximum possible success rate, which means that the system always performed an optimal integration. Figure 7.17 depicts the decision region found by the gating network for the integration of experts 0+2, which can be compared to that shown on the right hand side of Figure 7.16. Notice that both solutions are equally effective for integrating these experts, since they keep the areas where only one of the experts is correct on opposite sides of the decision boundary, while distributing the regions where both experts perform identically.

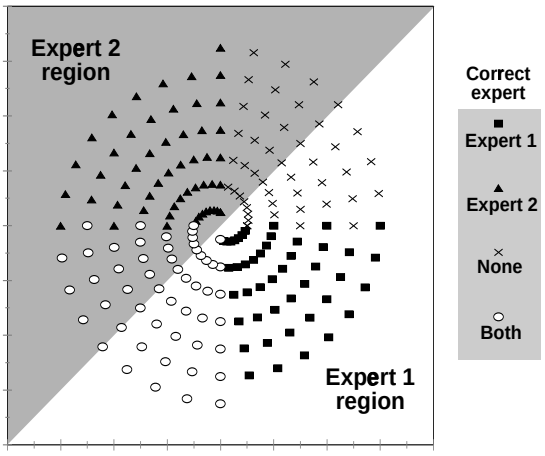



**Figure 7.17** Gating network experts' assignment for the integration of experts 0 and 2 shown in Figure 7.16.

### 7.5.2 A financial advising problem

This problem is a modified version of the simple financial advisor from (Luger and Stubblefield, 1989). The task is to advise whether an individual should invest capital in additional savings or in the stock market. Although the rule based model shown in Table 7.7 is extremely simplified, it illustrates issues involved in realistic financial advising.

The system's input consists of six real variables, shown in italics in Table 7.7: annual income, if the income source is steady, current assets, current savings, annual debt payments and the number of dependents. The output variable, **invest_stocks**, can have two possible values, corresponding to advising "yes" or "no". The AND/OR graph corresponding to the rule base from Table 7.7 is shown in Figure 7.18.

#### *7.5.2.1 Sources of partial domain knowledge*

Pruned versions (i.e. with one or more rules missing) of the rule-base were used to create imperfect local expert systems with diverse performances, which were used as models of real-life, rule-based financial advising systems developed using incomplete knowledge. The experts used to test the constructive integration approach

| Label | Rule | |
|---|---|---|
| (1) | if (savings_ok and income_ok) then | **invest_stocks** |
| (2) | if dependent_savings_ok then | savings_ok |
| (3) | if assets_high then | savings_ok |
| (4) | if (dependent_income_ok and *earnings_steady*) then | income_ok |
| (5) | if debt_low then | income_ok |
| (6) | if (*savings* $\geq$ *dependents* $\times$ 5000) then | dependent_savings_ok |
| (7) | if (*income* $\geq$ 25000 + 4000$\times$ *dependents* ) then | dependent_income_ok |
| (8) | if (*assets* $\geq$ *income* $\times$ 10) then | assets_high |
| (9) | if (*annual_debt* $<$ *income* $\times$ 0.30) then | debt_low |

**Table 7.7** Financial advisor rule base.

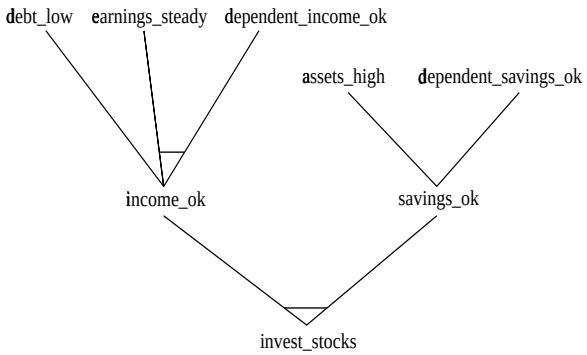



**Figure 7.18** Financial advisor AND/OR graph.

are shown in Table 7.8. This table identifies each expert by its pruning point. As an example, the elimination of the savings_ok rule and its antecedents dependent_savings_ok and assets_high is indicated by a prior knowledge pruning point of savings_ok. The expert system rule base of Table 7.7 is used to generate example data.

The expert systems shown in Table 7.9 were used to test the multiple experts integration approaches. The pruned rules for each expert are designated with rule numbers corresponding to those used in Table 7.7. A fixed, previously trained neural network, named "NN" was also used as a local expert. The output representations for both symbolic and neural experts were treated as real numbers in the range [0, 1]. Notice that these experts can be separated into three classes: pessimistic, optimistic and mixed. A pessimistic expert's errors are always false negative predictions, that is, errors in which the output is 0 (recommending to stay out of stocks) when it should be 1 (recommending to invest). On the other hand, an optimistic expert's errors are all false positive predictions (it outputs 1 when it should say 0). A mixed

| **Prior knowledge pruning point** | **Size** (hidden units) | **Generalization** | |
|---|---|---|---|
| | | **rules alone** | **rules + examples** |
| no pruning | 0 | 100% | 100% |
| no prior knowledge | 4.1 | n/a | 81.06% |
| dependent_savings_ok | 3.4 | 74.95% | 86.19% |
| assets_high | 0 | 93.36% | 93.36% |
| dependent_income_ok | 0 | 95.2% | 95.2% |
| earnings_steady | 0 | 95.6% | 95.6% |
| debt_low | 5.7 | 61.76% | 82.22% |
| savings_ok | 0.4 | 90.18% | 91.17% |
| income_ok | 4.7 | 67.54% | 85.02% |

**Table 7.8** Individual experts vs. hybrid systems classification accuracy.

| System | Success Rate | False prediction | | Pruned rules | Expert type |
|---|---|---|---|---|---|
| | | negative | positive | | |
| Expert 1 | 65% | 100% | 0% | (2),(6) | pessimistic |
| Expert 2 | 69% | 100% | 0% | (5),(9) | pessimistic |
| Expert 3 | 82% | 100% | 0% | (3),(4),(7),(8) | pessimistic |
| Expert 4 | 73% | 0% | 100% | (7),(4) | optimistic |
| NN | 87% | 57% | 43% | not applicable | mixed |

**Table 7.9** Local experts used in multiple experts integration approaches.

expert makes both kinds of errors.

#### *7.5.2.2 Incremental model construction*

As already explained, the expert system rule base of Table 7.7 is used to generate example data. Five hundred training examples and five thousand test examples were randomly generated consistent with the full rule base.

For these experiments, pruned versions of the AND/OR graph from Figure 7.18 were transformed into neural networks as in the KBANN technique, but without fully connecting the network. Then, the HDE algorithm was used to add units to the last hidden layer. As an example, Figure 7.19 illustrates the initial neural network obtained by transforming the original AND/OR graph from Figure 7.18 with no pruning.

Average results of five experiments are shown in Table 7.8. Observe that the hybrid system's performance was always equal or superior to those of the rule based experts and learning from examples alone. Also, note that when learning without the debt_low rooted subtree of the rule base, the constructive algorithm

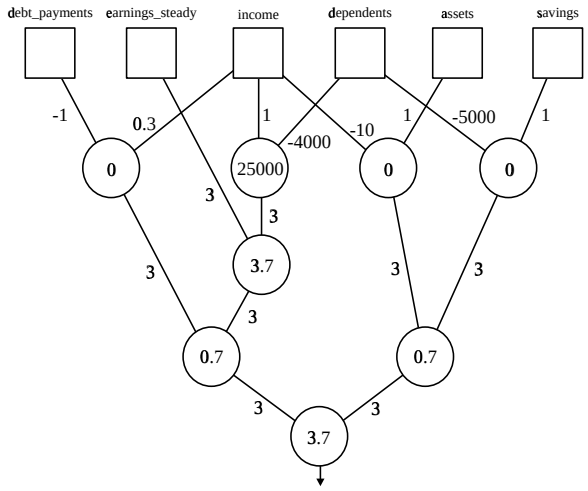



**Figure 7.19** Initial network.

| Expert systems | Generalization | | | |
|---|---|---|---|---|
| | Upper bound | Decision tree | Cooperative network | Competitive network |
| Experts 1 + 2 | 86.05% | 67.75% | 86.05% | 85.78% |
| Experts 1 + 3 | 90.37% | 68.60% | 90.37% | 90.02% |
| Experts 2 + 3 | 95.56% | 70.00% | 95.56% | 94.53% |
| Experts 1 + 4 | 100.00% | 68.60% | 58.62% | 90.60% |

**Table 7.10** Financial Advising: integration of two experts.

showed an impressive increase in prediction quality. Predictive quality of 61.76% from rules alone increased to 82.22% for rules and examples. In comparison, the knowledge refinement technique of the extended KBANN rule-based network using additional connections and backpropagation as in (Towell et al., 1990) provided an increase to only 64.64%. It is also important to observe that when the expert was able to classify the sample data well no hidden units were constructed.

#### *7.5.2.3 Multiple experts integration*

In the experiments carried out with the neural network combiners the training and testing sets were generated independently, with 1,000 examples in the training set and 10,000 in the test set. For the decision tree approach different training sets were used, ranging from 120 to 500 examples, and the testing set was the same one used for the neural network based integration techniques. Experiments were performed by applying the three integration techniques discussed in Section 7.4 to different combinations of expert systems and/or neural networks. For comparison purposes, an upper bound on the success rate of each combination was computed for the given test data set. As in the two-spirals problem (Table 7.6), this upper bound represents the maximum possible success rate achievable by always following the correct advice, i.e., selecting the expert that best classifies the given example. Also, a bound below 100% means that there are cases where all local experts classification responses are wrong, and so it is impossible to output a correct answer either by selecting one of them, or by combining their outputs. Notice that this upper bound is by no means tight. For example, a pair of "dumb" experts, in which one of the modules always outputs 0 and the other always outputs 1, has an upper bound of 100% (they are never both wrong), but the information they provide is nil. Thus, the cooperative combiner network can only achieve a very limited performance, while the gating network used in the competitive approach is left with the task of learning to classify the patterns by itself. Actually, the gating network has the advantage of being fed the original input in addition to the outputs of the "dumb" experts, and so it can learn to some extent how to classify the patterns. For the financial advising problem, a gating network combining two "dumb" experts achieved a test set success rate close to 72%.

The results of integrating several pairs of expert systems are shown in Table 7.10. The table presents the computed upper bound on the accuracy of each combination and the generalization (testing data) success rates obtained by implementing the decision tree, cooperative and competitive network, respectively. The results shown for the decision tree approach are averaged over twelve training sets of different sizes (120, 200, 300 and 500 examples). It is interesting to note here that the monostrategy decision tree approach (i.e., using a decision tree to solve the original classification problem) gave a better result (74.75%) than all the decision tree integrated systems tested here. Notice that, when combining pairs of pessimistic experts, both neural network-based methods produce excellent results. This is so because it is very easy to combine either two or more pessimistic or two or more optimistic experts by feeding their outputs to an AND gate (if the experts are optimistic) or an OR gate (if the experts are pessimistic). Obviously, both problems can be learned by a single neuron.

In contrast, the combination of pairs of experts of different types (1+4) proves to be much more difficult for the cooperative combiner approach. In this case, the competitive network achieves around 90% accuracy, and the decision tree does a better job than the cooperative combiner. This is another example of the domain transformation's shortcomings when dealing with low resolution outputs, as explained in Section 7.2.

The results of combining two of the expert systems with a neural network are summarized in Table 7.11. The upper bound for the success rate is measured using the symbolic expert and the fixed neural network. The table shows two different implementations of the competitive network. In the fixed neural network case, the neural network shown in Table 7.9 was used as an expert system, i.e. it only responded to the inputs, with no further training. In the dynamic learning method, the neural network local expert was trained at the same time as the gating network.

Notice that, in this case, the cooperative combiner's performance when combining "mixed" experts improves significantly over that on Table 7.10. This is caused by the fact that the output from one of the experts (NN) is now a real number, instead of a one-bit value. This increase in resolution facilitates the generation of a more adequate decision region on the transformed domain. The gating network, on the other hand does a very good job on integrating these systems, especially when the local expert neural network is allowed to learn simultaneously with the gating network. The decision tree approach managed to outperform the cooperative

| **Expert systems** | **Generalization** | | | | |
|---|---|---|---|---|---|
| | Upper bound | Decision tree | Cooperative network | Competitive network | |
| | | | | Fixed NN | Dynamic NN |
| Expert 3 + NN | 94.11% | 74.05% | 72.10% | 89.00% | 93.24% |
| Expert 4 + NN | 96.62% | 66.10% | 78.21% | 91.45% | 95.33% |

**Table 7.11** Financial Advising: integration of one expert and a neural network.

combiner in one of the cases, but its results continued to be very poor.

## 7.6 Conclusions

Several approaches to the development of knowledge-based neurocomputing classification systems integrating existing classifiers and learning from examples are discussed and compared on two domains. It was demonstrated that incremental single expert expansion can provide generalization improvement over both the expert and learning from examples alone. Also, it was evident that some of the multiple experts integration techniques can take advantage of multiple heterogeneous sources of partial domain knowledge. In particular, the competitive neural network approach was found to be superior to the other multiple experts integration methods studied, and to each of the sources of prior knowledge.

It is important to observe that single expert expansion and multiple experts integration are not mutually exclusive approaches. Once heterogeneous experts are efficiently integrated, the obtained system can be used as prior knowledge for single expert expansion. Conversely, in multiple experts integration systems, a single expert extended through incremental learning can be treated as one of several sources of partial domain knowledge. Further research is needed to characterize which approach is more appropriate for specific problem classes.

# References


Baum, E. B., 1991. Neural Net Algorithms That Learn in Polynomial Time from Examples and Queries. *IEEE Transactions on Neural Networks* 2, no. 1:5–19.

Baum, E. B. and Lang, K. J., 1991. Constructing Hidden Units using Examples and Queries. In *Advances in Neural Information Processing Systems*, eds. R. P. Lippmann, J. E. Moody, and D. S. Touretzky, vol. 3, pp. 904–910. Denver 1990: Morgan Kaufmann, San Mateo.

Benachenhou, D., Cader, M., Szu, H., Medsker, L., Wittwer, C., and Garling, D., 1990. Neural Networks for Computing Invariant Clustering of a Large Open Set of DNA-PCR Primers Generated by a Feature-Knowledge Based System. In *Proc. International Joint Conference on Neural Networks*, vol. 2, pp. 83–89. San Diego, CA.

Bishop, C., 1995. *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press.

Breiman, L., 1996. Bagging Predictors. *Machine Learning* 24, no. 2:123–140.

Chan, P., Stolfo, S., and Wolpert, D., 1996. *Working Notes of The 1996 AAAI Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms, held in conjunction with National Conference on Artificial Intelligence AAAI. WWW location http://cs.fit.edu/∼imlm/*. Portland, OR.

Chenoweth, T. and Obradović, Z., 1996. A Multi-Component Nonlinear Prediction System for the S&P 500 Index. *Neurocomputing* 10, no. 3:275–290.

Dietterich, T. and Michalski, R., 1983. A comparative review of selected methods for learning from examples. In *Machine Learning*, eds. J. C. R.S. Michalski and T. Mitchell, pp. 41–82. San Mateo: Morgan Kaufmann.

Drossu, R. and Obradović, Z., 1996. Rapid Design of Neural Networks for Time Series Prediction. *IEEE Computational Science and Engineering* 3, no. 2:78–89.

Fahlman, S. and Lebiere, C., 1990. The Cascade-Correlation Learning Architecture. In *Advances in Neural Information Processing Systems*, ed. D. S. Touretzky, vol. 2, pp. 524–532. Denver 1989: Morgan Kaufmann, San Mateo.

Fayyad, U., 1994. Branching on attribute values for decision tree generation. In *Proc. of the 12th National Conference on Artificial Intelligence*, pp. 601–606.

Fayyad, U. and Irani, K., 1993. Multi-interval discretization of continuous-valued

attributes for classification learning. In *Proc. of the International Joint Conference on Artificial Intelligence, IJCAI-93*, pp. 1022–1027.

Fletcher, J. and Obradović, Z., 1993. Combining Prior Symbolic Knowledge and Constructive Neural Network Learning. *Connection Science* 5, no. 3,4:365–375.

Fletcher, J. and Obradović, Z., 1995. A Discrete Approach to Constructive Neural Network Learning. *Neural, Parallel and Scientific Computations* 3, no. 3:307–320.

Fukunaga, K., 1990. *Introduction to statistical pattern recognition*. San Diego: Academic Press.

Gallant, S. I., 1988. Connectionist Expert Systems. *Communications of the ACM* 31, no. 2:152–169.

Gallant, S. I., 1990. Perceptron-Based Learning Algorithms. *IEEE Transactions on Neural Networks* 1, no. 2:179–191.

Gutknecht, M., Pfeifer, R., and Stolze, M., 1991. Cooperative Hybrid Systems. Tech. rep., Universitat Zurich.

Hanson, M. and Brekke, R., 1988. Workload Management Expert System - Combining Neural Networks and Rule-Based Programming in an Operational Application. In *Proc. Instrument Society of America*, vol. 24, pp. 1721–1726.

Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., eds., 1983. *Building Expert Systems*. Reading, MA: Addison-Wesley.

Hendler, J. and Dickens, L., 1991. Integrating Neural Network and Expert Reasoning: An Example. In *Proc. AISB Conf. on Developments of Biological Standardization*.

Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G., 1991. Adaptive Mixtures of Local Experts. *Neural Computation* 3:79–87.

Kandel, A. and Langholz, G., eds., 1992. *Hybrid Architectures for Intelligent Systems*. Boca Raton: CRC Press, Inc.

Luger, G. F. and Stubblefield, W. A., 1989. *Artificial Intelligence and the Design of Expert Systems*. Redwood City, CA: Benjamin/Cummings.

Medsker, L. and Bailey, D., 1992. Models and Guidelines for Integrating Expert Systems and Neural Networks. In *Hybrid Architectures for Intelligent Systems*, eds. A. Kandel and G. Langholz. Boca Raton: CRC Press, Inc.

Michalski, R. and Tecuci, G., eds., 1994. *Machine Learning. A Multistrategy approach*, vol. 4. Morgan Kaufmann.

Nowlan, S. and Hinton, G., 1991. Evaluation of Adaptive Mixtures of Competing Experts. In *Advances in Neural Information Processing Systems*, eds. R. Lippmann, J. Moody, and D. Touretzky, vol. 3, pp. 774–780. San Mateo, CA: Morgan Kaufmann.

Quinlan, J., 1986. Induction of Decision Trees. *Machine Learning* 1:81–106.

Romero, P. and Obradović, Z., 1995. Comparison of Symbolic and Connectionist Approaches to Local Experts Integration. In *IEEE Technical Applications*

*Conference at Northcon/95*, pp. 105–110. Portland, OR.

Romero, P., Obradović, Z., Kissinger, C., Villafranca, J., and Dunker, A., 1997. Identifying disordered regions in proteins from amino acid sequence. In *Proc. IEEE International Conference on Neural Networks*, vol. 1, pp. 90–95. Houston, TX.

Samad, T., 1988. Towards Connectionist Rule-Based Systems. In *Proceedings of the IEEE International Conference on Neural Networks*, vol. 2, pp. 525–532. San Diego.

Shimshoni, Y. and Intrator, N., 1996. On the Integration of Ensembles of Neural Networks: Application to Seismic Signal Classification. In *Working Notes of the 1996 AAAI Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms, held in conjunction with National Conference on Artificial Intelligence AAAI, WWW location http://cs.fit.edu/~imlm/*, eds. P. Chan, S. Stolfo, and D. Wolpert. Portland, OR.

Towell, G. G., Shavlik, J. W., and Noordwier, M. O., 1990. Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 861–866. Boston, MA.

Weigend, A., Mangeas, M., and Srivastava, A., 1995. Nonlinear Gated Experts for Time Series: Discovering Regimes and Avoiding Overfitting. *Int. J. of Neural Systems* 6:373–399.

Werbos, P., 1995. *Beyond regression: New tools for predicting and analysis in the behavioral sciences. Harvard University, Ph.D. Thesis, 1974*. Wiley and Sons (Reprinted).

Zhang, X., Mesirov, J. P., and Waltz, D. L., 1992. Hybrid System for Protein Secondary Structure Prediction. *Journal of Molecular Biology* 225:1049–1063.

**Pedro Romero** graduated in 1981 as a Chemical Engineer from Universidad Simón Bolívar, Caracas, Venezuela. He received his M.S.E. degree in Chemical and Biochemical Engineering from the University of Pennsylvania in 1985 and is currently a Ph.D. candidate in Computer Science at Washington State University, doing his research on the applications of artificial neural networks in biocomputing and data mining under the supervision of Dr. Zoran Obradović.

**Zoran Obradović** received his B.S. degree in Applied Mathematics, Information and Computer Sciences in 1985, his M.S. degree in Mathematics and Computer Science in 1987, both from the University of Belgrade and his Ph.D. degree in Computer Science from the Pennsylvania State University in 1991. He is currently an associate professor in the School of Electrical Engineering and Computer Science, Washington State University, and an adjunct research scientist at the Mathematical Institute of the Serbian Academy of Sciences and Arts. The objective of his research is to explore the applicability of neural network technology to large scale classification and time series prediction problems in very noisy domains.

**Justin Fletcher** received his B.S. in Interdisciplinary Studies in 1980 and his M.S. in Computer Science in 1983 from the University of Idaho. From 1982 until 1991 he worked as software engineer, consultant and software manager in industry. He received his Ph.D. degree in Computer Science from Washington State University in 1994 specializing in artificial neural networks and is currently a member of the technical staff at XKL LLC.