International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 19, No. 2 (2005) 249–269

World Scientific
www.worldscientific.com

MODULAR NEURAL NETWORK DESIGN FOR THE PROBLEM OF ALPHABETIC CHARACTER RECOGNITION

BRENT FERGUSON*, RANADHIR GHOSH and JOHN YEARWOOD
University of Ballarat, PO Box 663, Ballarat, Victoria 3353, Australia
**b.ferguson@students.ballarat.edu.au*

This paper reports on an experimental approach to find a modularized artificial neural network solution for the UCI letters recognition problem. Our experiments have been carried out in two parts. We investigate directed task decomposition using expert knowledge and clustering approaches to find the subtasks for the modules of the network. We next investigate processes to combine the modules effectively in a single decision process. After having found suitable modules through task decomposition we have found through further experimentation that when the modules are combined with decision tree supervision, their functional error is reduced significantly to improve their combination through the decision process that has been implemented as a small multilayered perceptron. The experiments conclude with a modularized neural network design for this classification problem that has increased learning and generalization characteristics. The test results for this network are markedly better than a single or stand alone network that has a fully connected topology.

Keywords: Artificial modular neural network; task decomposition; alphabetic character recognition.

1. Introduction

Artificial neural networks that have fully connected feed forward topologies have in the past had successful application in the problem areas of classification and regression. However their success in part favors databases where the data predominantly describes its concepts as a clear function of its features. It has also been established in the studies of Quinlan[11] and Collier[4] that these networks also require conditions of high feature independence to learn optimally. In recent times the spiralling accumulation of complex and high-dimensional databases are overwhelming the capabilities of these networks by introducing two conditions that adversely affect their training and post training performance. These conditions are scalability and data complexity.

The scalability problem arises when the artificial neural network is applied to large databases. Where these databases have large numbers of feature attributes and multiple classes, this often requires the network to have high numbers of neurons

and connections. These large networks when fully connected, require lengthy training times and are susceptible to the condition of overfitting.[13] This is where the network learns too well the specifics of its training set so that it does not generalize very well over new or unseen data. Although there are a number of ways to overcome overfitting that include weight decay and early stopping during training, these techniques apply better to smaller networks. Large databases often have complex interrelationships between data that confuse training algorithms. One problem that these interrelationships can cause is the problem of interference.

There are two known causes of interference, being spatial crosstalk and temporal crosstalk.[8] Temporal crosstalk occurs where functions are learned by the network consecutively and results with one function that has already been learned and forgotten while it learns another.[12,17] Spatial crosstalk is where two or more functions are learned at the same time causing sharing problems with the neurons of the hidden layer. Both problems of scalability and data complexity can be resolved to a greater extent by structuring a neural networks topology.[3,7,13]

In recent times, there has been growing interest in a different kind of neural network known as the modular neural network. These networks have the potential to overcome the problems experienced by fully connected networks or monolithic networks. The modular neural network is designed to separate the functions that cause interference problems by having independent modules assigned to each function. Modularized networks have other advantages. Modules can be trained in parallel which reduces training times substantially and additional modules can be added without the need to retrain the others.

Current research for the development of modular neural networks is divided into two areas. According to Auda and Kamel,[1] one area concerns itself with the decomposition of the problem into smaller subtasks while the other area of development concerns itself with the coupling of the modules in such a way that they are appropriately coordinated to provide an optimal solution for the problem. Decomposing the problem into subtasks is a process generally referred to as task decomposition and is usually directed by expert knowledge. Although, in recent studies there appears to be a trend towards automating this process.[5,6,10]

Task decomposition for a problem is a straightforward process where there is knowledge of the problem and the relationships in data. In this circumstance, modular structuring follows the structure of knowledge.[1] However, where knowledge is vague or nonexistent, the process becomes frustrated with little resolve from analytical approaches and it becomes unclear how to proceed and go about the process. This can be the case with high-dimensional data emerging from research programs such as those that are based in genetics and imaging.

In recent times, the research for task decomposition in the absence of knowledge is moving to find the functional distribution between modules[16] for a given problem. In an approach by Wan *et al.*[16] functional distinction between modules proceeds by developing modules on disjoint subsets of the feature set. The disjoint subsets

develop from a process of measuring the relative interaction between features with the application of a second-order derivative to two features at a time. Features accumulated within a subset have higher interaction between them than with features in other subsets. In the study by Liu *et al.*,[5] functional distinction was referred to as the speciality of a module. The speciality between the modules of their ensembles was arrived at through an evolutionary process. Negatively correlating modules were evolved in populations of modules learning in parallel using a correlation penalty term in the error function. In this case, the interactivity between modules was exploited to encourage the development of module speciality or expertness in separate areas of the problem.

In the second area of research the problem of how to combine the subtasks for modules is still relatively unclear given a diversity of reported methods. According to Sharkey,[14] these methods fall into one of four categories.

1. Competitive — Where modules are selected on the basis of their expertness using rule-based switching, for example.
2. Cooperative — Where all modules contribute towards the main task. An example involves using a gating process.[9] This is where the outputs of the modules are weighted with a gating vector and summed together.
3. Sequential — Where modules are arranged serially with the output of one serving as an input to another.
4. Supervisory — This is where one module learns to supervise the behavior of another. For instance, a neural network can learn to predict another networks[1] error response for a given circumstance.

The majority of methods are based on either the competitive or cooperative categories.

Our study is concerned with finding a modularized neural network solution for the problem of alphabetic character recognition. This is a large classification problem where fully connected or monolithic networks are known to have trouble training and performing this task. We apply an experimental approach to task decomposition to find suitable subtasks for the modules and to find a suitable combining process for the decision module. This paper is organized into the following sections. Section 2 details our approach to modular neural network design. Section 3 describes the conditions for the experiments, the software used, the details of the dataset and its preparation. Section 4 outlines the experimental approach for design, followed by the results and discussion in Sec. 5. Section 6 discusses the causes for error in the design experiments. Section 7 outlines further experiments to combine the modules with supervision, followed by the results for this section and further discussion in Sec. 8. Section 9 summarizes the conclusions reached in our study.

2. Approach to Modular Design

In this work, we consider the two components for designing a modular neural network; task decomposition and the combination of modules. The proposed architecture for this network that is illustrated in Fig. 1 has a number of independent subtasking modules that contribute cooperatively to a common output through a decision module. A number of task decomposition processes are experimented with in order to partition the data for the problem of character classification to learn the subtasks that would occur in a parallel sense. Each subtask would be responsible for a unique portion of the data space. There are two considerations that arise with an approach to task decomposition:

— how to decompose a problem where little or no knowledge exists?
— how many subtasks are sufficient for adequate solution of the problem?

We acknowledge these considerations by determining the criteria for a suitable subtask in that it must be highly discriminating for its data partition over other partitions. That is, a neural network trained in a subtask would feature high classification for test data belonging to its partition. This would also indicate the granularity of decomposition to decide how many subtasks. That is to say that further iterations of the decomposition method would not result in any appreciable gains in test accuracy. We explore two approaches to task decomposition with one based

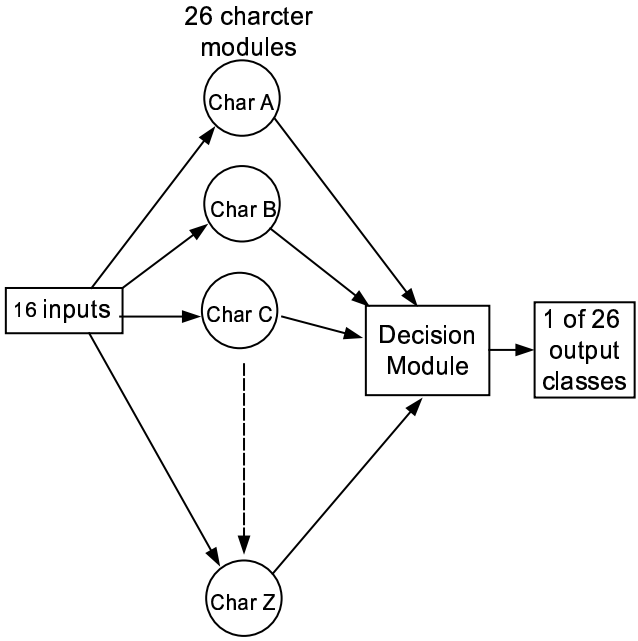

Fig. 1. Proposed architecture.

on applied expert knowledge and the other using machine automation to assist this process. In our experiments we try clustering using a self-organizing map (SOM) where the nearest neighbor principle is used to cluster data partitions.

Having found appropriate subtasks through task decomposition we next seek to understand how simultaneously occurring subtasks can be combined to perform the overall task, where the combined result would be superior to that obtained with a single fully connected neural network. For this purpose we explore the effectiveness of decision based on, learning the decision process using a neural network and learning the process using a decision tree.

3. Experimental Conditions

3.1. *The UCI dataset and its preparation*

The UCI Letters dataset is a classification problem for the 26 uppercase characters of the English alphabet. This dataset consists of 20,000 character examples. Each character is represented by a black-and-white rectangular pixel display and are based on 1 of 20 different fonts. The examples for each character have 16 primitive numerical attributes, being statistical moments and edge counts which were scaled to fit into a range of integer values from 0 to 15. This dataset was chosen for our experiments because it is a large classification problem that presents a challenge for different types of classifier.

For the purposes of our study, it was necessary to resize the dataset from 20,000 to 3,900. This size represents the 150 examples for each of the 26 characters that were chosen from the population at random. The self-organizing map used in the clustering experiments had a maximum limit of 4,000 for the dimensional length of examples that it could process. The reduction in the size of the dataset was necessary to accommodate this limitation so that the proposed experiment for clustering the 16 features of the UCI dataset could be carried out. In this experiment the dataset was transposed so that each feature had a dimension of 3,900.

In addition to resizing, the dataset was normalized to between 0 and 1 by dividing all values by the highest, which was 15. The output training vector was prepared so that each example indicated its associated subtask with a value of 0.9 and 0.1 for no association. This allowed the delta rule that was used to train the backpropagation neural networks, which had sigmoid transfer functions, to work more efficiently.

3.2. *Software details*

All experiments were performed on a PC Microsoft windows platform. The PC used had a 2.2 GHz Pentium 4 microprocessor. Programs used for implementing algorithms were:

— Matlab 6.5 release 13 with neural network toolbox 4.0.1 for neural network components.

— SOM_PAK developed by the Helsinki University of Technology Laboratory of Information and Computer Science for clustering procedures.
— Weka-3-4 developed at the University of Waikato New Zealand for its command line decision tree utilities. The decision tree algorithm used was J48 being the equivalent of C4.5 version 8.

3.3. *Module implementation and training*

Modules for the subtasks found by our decomposition approaches have been implemented as small fully connected three layered neural networks. The module learns its subtask which is to distinguish the examples that define the subtask from all other examples. The training set for the module is therefore composed of all examples allocated for training with the training output for each example being indicated by 0.9 if it is associated with the subtask and 0.1 if it is not associated. Character modules were trained in sequence for 10 trials. In each successive trial the examples for the training, validation and test sets were drawn at random with replacement. The validation set worked with the training set to grow the hidden layer of the neural networks until there was no further improvement to be observed in the validation set classification accuracy. All neural networks in this study used the validated training cycle described in Fig. 2. The stopping criteria for the networks or the point where training stops before each increment in hidden neurons used Matlabs's net training parameters. Three parameters were used and set accordingly, epochs = 2,000, min_gradient = 1e-5 and validation_fail = 30. The training, validation and test sets were allocated 85%, 5% and 10% of the examples in the reduced dataset.

4. Methodology for Designing the Modular Neural Network

4.1. *Task decomposition experiments*

Our experiments began with finding parallel subtasks for this problem. Two approaches were tried to partition the data. The first approach was to direct decomposition through applying expert knowledge or knowledge of the problem that is external to the dataset. In this, we apply something of what we know about the structure of the data. In the second approach we applied an algorithm to find the subtasks automatically. In this approach, we assume we know nothing of how the data is structured. For this purpose we used a SOM self-organizing map where the learning procedure is based upon a Euclidean Distance measure.

4.2. *Subtasks for individual characters*

In this experiment we used knowledge of the class distribution of character examples to create subtasks. Each subtask is the simple recognition of a particular character. The modules that corresponded to each subtask were trained to distinguish or recognize a particular character's examples from all the other examples.

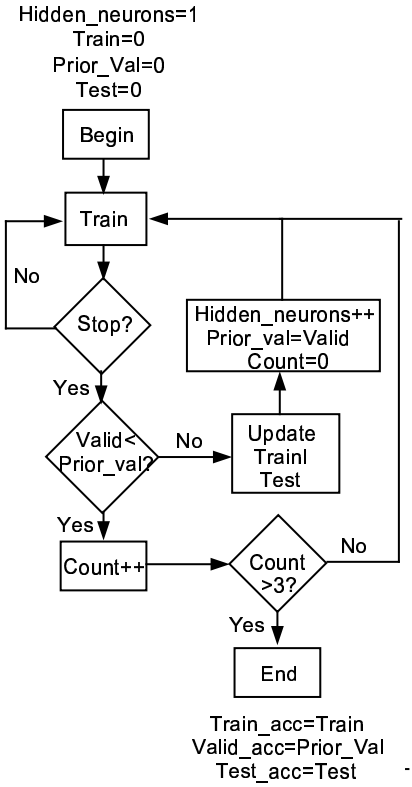

Fig. 2. ANN training cycle.

Expert knowledge approach
- subtasks for individual characters
- subtasks for character shapes

Automated approach - using clustering
- subtasks from example clusters
- subtasks from feature clusters
- subtasks from feature clusters within example clusters

Fig. 3. Task decomposition approaches.

4.3. *Subtasks for character shapes*

For these experiments we used knowledge of the shape of characters. We organized subtasks to recognize groups of characters having similar graphical characteristics. Each of the following three experiments have different groupings of characters according to different specifications.

4.3.1. *Experiment with two groups*

Group 1 — Characters predominantly composed of straight edges A,E,F,H,I,J,K, L,M,N,T,U,V,W,X,Y,Z.
Group 2 — Characters having a curve within their shape B,C,D,G,O,P,Q,R,S.

4.3.2. *Experiment with three groups*

Group 1 — Characters having an open vertical shape U,V,W.
Group 2 — Characters that are closed loops Q,O.
Group 3 — All other characters A,B,C,D,E,F,G,H,I,J,K,L,M,N,P,R,S,T,X,Y,Z.

4.3.3. *Experiment with nine groups*

Group 1 — Horizontal top line E,F,J,T,Z.
Group 2 — Horizontal bottom line E,L,Z.
Group 3 — Horizontal center line A,E,F,H.
Group 4 — Vertical left line B,D,E,F,H,I,K,L,M,N,P,R.
Group 5 — Left curve C,G,S.
Group 6 — Right curve D,O,Q.
Group 7 — Right small curve top B,P,R.
Group 8 — Open shape at top U,V,W.
Group 9 — All remaining characters X,Y.

4.4. *Decomposition using clustering to partition data*

In these experiments we used an automated approach to find the subtasks by applying a clustering technique. A subtask was associated with data having similar properties. For this purpose we clustered data having a similar Euclidean distance from a point that was determined by a self-organizing map SOM. The purpose of these experiments was to observe the effectiveness of an automatic means to perform the decomposition process. The SOM found an optimal number of clusters by having the learning rate adjusted until the number of clusters were found to limit and at which point the number of data items per cluster began to stabilize with repeat iterations of the SOM. The data was clustered in three ways:

— subtasks from example clusters,
— subtasks from feature clusters,
— subtasks from feature clusters within example clusters.

4.5. *Subtask combination experiments*

Following the experimental stage for task decomposition, the training results for modules found through this process were surveyed for each approach. The modules found to have the most ability for their subtask or those modules that were highest

in performance were chosen to be combined in a decision process which effectively combined each of their contributions for the main task. Each module output was combined in parallel for input to the decision module. The decision module was trained using different algorithms to learn to coordinate the combined outputs to one of 26 character classes. Three different algorithms were tried for the decision module.

— Linear perceptron — Using Matlab's linear perceptron with hard limiting transfer function.
— Multi layered perceptron (ANN) — Using a three-layered neural network, the same being used for the subtask modules.
— Decision Tree — Using a decision tree algorithm.

5. Results and Discussion for Design Experiments

5.1. *Task decomposition results*

To coincide with these experiments we have determined our own benchmark result for test set classification using a single fully connected neural network. This value was averaged at 70.60% having used the same experimental conditions outlined in Sec. 2 and using the same training and validation sets intended for training the modules. This value serves to make comparisons with the experimental results to assess our proposed methods for finding modules.

The proposed neural network solution for the problem of character classification is illustrated in Fig. 1. This illustration shows a simple structure of parallel character modules that function concurrently where each module provides an input to a single combining or decision process. As we have already discussed there are two issues with modular networks; finding modules and combining them. It is unknown whether a general solution exists for all problems and assuming that they can be decomposed, what determines the level of granularity of decomposition. In other words to what extent do we decompose and what indicates the limits of decomposition. Part of this study concentrates on finding the modules so that something can also be learned from the approaches to task decomposition.

We made assumptions where subtasking was concerned in that the function of a subtask is to be unique to the functions of other subtasks. We assumed that correct subtasking would train a neural network module in a subtask of classification that would produce superior performance to a monolithic multiple classifier. From this we can assume that there exists for a decomposable problem, a group of modules that will individually classify well for their part of the problem, but when combined will produce a high multiple classification performance that is similar to that obtained at the module level.

On the basis of our assumptions we have found through experiment that highly accurate modules are associated with those methods that were directed by external

knowledge assisting the decomposition process. In Table 1 for character level modularization and in Tables 2–4 for modularization based on character shape similarity, all of which are derived from knowledge-based decomposition, the average module test result far exceeded our benchmark test result.

Character level modularization gave us best result of all the experiments within the knowledge-based approach with an average test result of 98.16%. In addition, this figure is comparable to the average training accuracy of 98.66% indicating on average that the modules are able to learn their subtask well enough to generalize

Table 1. Character level training results.

Char	Neurons	Inputs	Connections	Train	Valid	Test
A	5	16	91	99.73	100	98.72
B	3	16	55	98.49	98.97	98.46
C	3	16	55	99.03	98.97	97.95
D	2	16	37	98.46	98.97	97.69
E	1	16	19	97.29	96.92	95.64
F	8	16	145	98.88	98.97	98.72
G	1	16	19	96.14	96.41	96.15
H	1	16	19	97.25	95.38	97.95
I	1	16	19	99.25	98.46	97.95
J	1	16	19	98.64	97.95	99.23
K	3	16	55	98.55	97.44	98.46
L	2	16	37	99.34	98.46	99.23
M	2	16	37	99.67	100	98.97
N	2	16	37	99.34	98.46	98.72
O	9	16	163	98.82	99.49	98.46
P	1	16	19	98.73	98.97	98.72
Q	3	16	55	98.52	98.97	97.95
R	1	16	19	97.22	97.44	95.64
S	4	16	73	97.62	96.41	96.92
T	5	16	91	99.22	98.97	99.49
U	6	16	109	99.19	98.97	97.95
V	3	16	55	99.43	98.97	99.49
W	1	16	19	99.13	100.0	99.23
X	3	16	55	98.70	98.97	98.46
Y	6	16	109	99.46	99.49	97.18
Z	4	16	73	99.28	100.0	98.97
Avg.	3.07	16	56.38	98.66	98.53	98.16
Max	9	16	163	99.73	100.00	99.49
Min	1	16	19	96.14	95.38	95.64

Table 2. Two character group training results.

Group	Neurons	Inputs	Connections	Train	Valid	Test
1	6	16	109	91.13	87.18	88.46
2	8	16	145	91.40	88.72	89.23
Avg.	7	16	127	91.26	87.95	88.84
Max	8	16	145	91.40	88.72	89.23
Min	6	16	109	91.13	87.18	88.46

Table 3. Three character group training results.

Group	Neurons	Inputs	Connections	Train	Valid	Test
1	4	16	73	97.71	99.49	97.18
2	4	16	73	97.47	98.46	97.69
3	4	16	73	94.72	95.38	93.33
Avg.	4	16	73	96.63	97.77	96.06
Max	4	16	73	97.71	99.41	97.69
Min	4	16	73	94.72	95.38	93.33

Table 4. Nine character group training results.

Group	Neurons	Inputs	Connections	Train	Valid	Test
1	9	16	163	95.17	93.74	91.79
2	8	16	145	97.65	95.26	95.90
3	6	16	109	95.07	92.41	91.54
4	4	16	73	87.43	85.96	86.67
5	6	16	109	96.28	94.12	94.62
6	5	16	91	96.38	94.88	94.36
7	5	16	91	96.45	95.83	96.41
8	5	16	91	98.26	97.91	96.67
9	9	16	163	97.92	98.48	95.13
Avg.	6.3	16	115	95.62	94.28	93.67
Max	9	16	163	98.26	98.48	96.67
Min	4	16	73	87.43	85.96	86.67

over test data. A similar observation can be made for the best of the graphical similarity experiments where for the three group experiment the test result was 96.06% and the train result was 96.63%. With respect to these experiments the three-character group faired better than the nine-character group test result of 93.67% and much better than 88.84% for the two-group experiment. This indicates that an optimal result is possible that is less than 9 modules by applying what is known of the characteristics of letter characters. This to say that for a general multiclass problem we could apply something of what is known of the relationships between classes to the decomposing process to effectively create the modules of a modular neural network.

In the clustering experiments, that were designed to automatically perform task decomposition, the results show a deteriorating classifying performance at the module level when compared with the benchmark. The results for the clustering experiments, clustering of character examples in Table 5, clustering of the 16 features for each example in Table 6 and clustering of the features of examples within the example clusters in Table 7, all show low average module test results. There is a much wider variance in individual module accuracy than found with the knowledge-based experiments. This consequence is dependant upon the characteristics in the data for a given problem. For the problem of character classification, it appears unsupported to form modules from clusters of character examples and their features. In conclusion, it has been discovered that the data can be organized in at least two

Table 5. Example clustering average training results.

Statistic	Clusters	Neurons	Inputs	Train	Valid	Test
Average	7	12.31	16	66.57	69.77	54.70
Max	7	18.2	16	94.06	98.95	90.17
Min	7	6.8	16	40.85	48.57	32.85

Table 6. Feature clustering training results.

Statistic	Clusters	Neurons	Inputs	Train	Valid	Test
Average	6	3.97	2.66	14.00	20.44	14.02
Max	6	7.33	6.00	31.82	43.66	34.33
Min	6	2.33	1.00	4.00	7.00	4.66

Table 7. Example and feature clustering training results.

Statistic	Clusters	Neurons	Inputs	Train	Valid	Test
Average	51	2.98	2.50	56.28	61.74	55.65
Max	51	6.8	7.00	96.05	100	100
Min	51	1.2	1.00	2.88	11.25	1.54

Table 8. Classification results for the decision module processes.

Classifier	Train	Test
Benchmark	73.35	70.60
Linear perceptron	42.82	38.40
Multilayered perceptron	82.05	77.69
Decision tree	96.62	86.49

different ways to result in highly predicting modules and that the characteristics can be better managed with applied expert knowledge.

The question of granularity for task decomposition is also answered by use of applied knowledge as decomposition follows a plan. The low test results for the automated approach indicate additional decomposition may be necessary and that the extent of this cannot be determined.

5.2. *Subtask combination*

The modules most suited for combination are the character level modules. There are 26 modules that correspond to the subtasks of individual character discrimination. Referring to Table 8, the comparison of test results for the decision process for these modules indicates that the decision tree with a result of 86.49% exceeds the other decision approaches with the result for the linear perceptron being the worst at 38.40%. The result for the multilayered perceptron at 77.69% was relatively only a small improvement on our benchmark fully connected test result of 70.60%. Of the algorithms used for decision it is clear that the linear perceptron is highly unsuitable for this purpose. In deciding between decision tree and the multilayered perceptron,

even though the decision tree provided a better test result of 86.49%, the difference between the training and test result was greater than for the perceptron. This indicates that the perceptron could generalize better and therefore learn the decision task more capably. However the test result for the multilayered perceptron did not compare reasonably with the average module test result of 98.16% in Table 1. We speculate that if a decision module were to be based on the multilayered perceptron and if trained with inputs that were the same as its output training vector, in the ideal sense, it will yield a test result of 100%. However, when trained with the combined outputs of the modules, the test result was low at 77.69% even though the normalized combined output vector in comparison was 98% the same as the training vector. It therefore should be possible to use a multilayered perceptron for the decision process and the following section discusses causes for its current low result. We further extend our experimentation to improve the decision result using a multilayered perceptron.

6. The Problem of Module Combination

The question that needs to be answered is that given the high classification accuracies of the individual modules what could possibly be the output accuracy of the decision module. We expect that the decision module accuracy should at least reflect the behavior of its input modules. As this has not been the case, we conjecture that some error propagation from these modules to the decision level has taken place and we need to understand the nature of this propagation. In this section, we will explain the relationship between the decision module and the individual modules with a simple example and offer a solution to the problem with slight variations in the module design. The aim of this section is to find

$$\text{Acc}_d = f(m_1, m_2, m_3) \tag{1}$$

where Acc_d = accuracy of the decision module, m_i = accuracy of modules $1, \ldots, n$.

Let us take the example of creating a classifier for a 2 bit odd parity and even problem. Only 2 input bits are considered so that we may illustrate this example, Fig. 4. Also let us assume that this is a multiclass problem, in this Case, 2 class. We modularize, using task-based modularization and hence obtain two modules — one for solving the odd parity problem and another for solving the even parity problem. We also assume that no hidden layer is used for both modules for simplistic purposes. Now consider the following cases

Case 1: Propagation of errors for nonintersecting training examples. The first figure shows the decision boundary after the training of the ANN module. As can be seen, the classification accuracy for that module is 75% and the training example for which the error occurs is the vector (0,1). Similarly from Fig. 2, we can find that the module accuracy for the odd parity module is also 75% and the training example for which the error occurs is the vector (1,1).

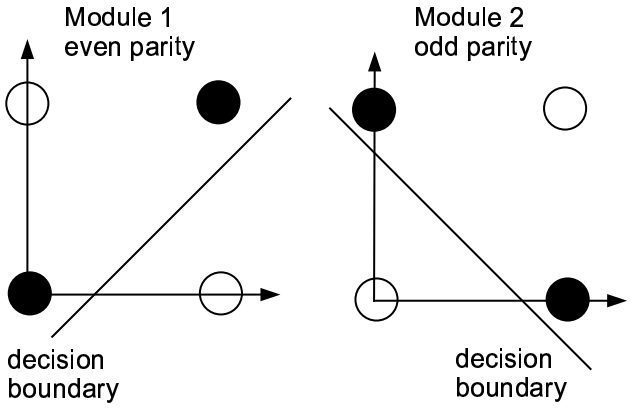

Fig. 4. Modules — odd and even parity.

The above circumstance leads to the decision module having a classification error of at least 50%, assuming we keep the threshold for both the modules same. The threshold issue will be discussed in the next case. So, if the training examples for which modules give error are disjoint, the error will be reflected in the decision module as an additive term. Similarly it can be concluded that if the intersection of the training examples for which the modules generate error is any value greater than zero then this can be used to reduce the error of the decision module from the summation of errors for all the modules. Following this, an expression can be developed for the error term of the decision module.

$$\sum_{i=1}^{n} me_i - I_1 \cap I_2 \cdots \cap I_n \tag{2}$$

where me_i is the error for module i and

$$I_i = \bigcup_{j=1}^{k} l_{ij} \tag{3}$$

which represents the union of all the examples for an individual module which produce errors for module i.

It is quite interesting to observe that because of the nature of this intersection, it is possible to reduce the error in the decision module. For example, by rotating the decision hyperplane for the second module, it is possible to increase the number of intersecting training examples. For example, in the above scenario after the training of one module, we provide a higher penalty term for the training of the second module for those examples which are not disjoint from the examples for the training of the first module. Thus by using different penalty terms for different example sets we can adjust the weights for the second module in such a way that the disjoint example sets have less overlap between the modules. In training the

modules using backpropagation of errors, for instance, the weight adjustment will be as follows

$$\delta w \propto Px \tag{4}$$

where P is the new penalty term depending on the input vector x.

Also it is worth mentioning that the accuracy of the individual module remains unchanged. However, although there is a solution, it can be quite complex to find some optimum value when the number of modules are many.

Case 2: Considering threshold and interfering effects of the modules. If we can class the errors of individual modules as either false positive or false negative, this leads to an interesting occurrence. Let us consider the case for a false negative error from the first module. For a multiclass problem the output class of the decision module can be considered as

$$\text{class} = \max(O_1, O_2, \ldots, O_n) \tag{5}$$

and

$$O_i = \{1if \geq t_1, 0if < T_i\}. \tag{6}$$

O_i is the output of module i and T_i is the threshold.

For a false negative error in module 1, if all the threshold values for other modules are less than the actual output, then some modification of the output for module 1 can remove this error. In other words if

$$T_i < O_1 i\epsilon 2, \ldots, n \tag{7}$$

then

$$O_1 > T_1. \tag{8}$$

Changing the module output values will need a good choice of threshold for individual modules to improve collective learning for the whole task. This is again, a difficult task if not impossible.

For a false positive error for module 1, and also if any other module produces the false positive error for the same input, the actual module for the class should have a lower threshold value, which is less than all the other modules producing the errors and thus needs adjusting thereafter.

Adjustment of the module outputs based on the type of class error leads to the idea of using a decision tree or some rule induction algorithm, which may simplify this process. The other solution is to choose an optimum vector for the threshold values for all the modules but the determination of this vector however, would expectedly incur high training times and learning complexity.

From the above case discussions, the classification of the decision module for combining n modules can be generalized as

$$\mathrm{Err}_d = \sum_{i=1}^{n} me_i - I_1 \cap I_2 \cdots \cap I_n - g(T_1, T_2, \ldots, T_k) - h(T_1, T_2, \ldots, T_k) + me_d. \quad (9)$$

g and h represent the functions for threshold adjustment for false negative and false positive errors for all the modules $k = n$, and me_d is the error after the training of the decision module itself.

So in the above formulation it is interesting to note that propagation of errors from the module level to the decision level can be quite high. However, it also suggests that there are possibilities in which these errors can be minimized, by rotating the decision planes, by adjusting the weights or by finding optimum threshold values for the individual modules.

7. Methodology for Improved Modular Neural Network Design

The following experiments have been designed to improve the performance of the decision module that has been implemented as a multilayered perceptron. These experiments observed the effect of error propagation from the module level and attempted to eliminate or significantly reduce them. As we have reasoned, the errors at this level being due to those examples that did not classify correctly, propagated to the decision module to adversely affect its performance. Two of three experiments were proposed to verify this while the third experiment, which was composed of two parts, used a combination of decision trees to supervise the modules and adjust deficiencies in the module's behavior. The details of the experiments:

1. Identify and isolate from the training set those examples that a character module classified correctly. Train a neural network module to perform the decision process with these examples only but include all examples for the test set.
2. Train a neural network module to perform the decision process with those examples that did not classify correctly and determine test result for all test examples.

3.1. Train a decision tree to detect the character module's unclassified examples from classified examples using only the training set. With the error examples that were correctly classified by the decision tree, label them as false positive and false negative error types. Train a second decision tree to detect the error types from the unclassified error examples (use three labels).

3.2. With the error examples that were not correctly classified, label them as false positive and false negative error types. Train a second decision tree to detect the error types from the classifiable error examples (use three labels).

Figure 5 illustrates the supervision process for each module. The purpose of the first decision tree is to detect the examples that cause classification error in the module. The second decision tree detects the false positive errors from the false

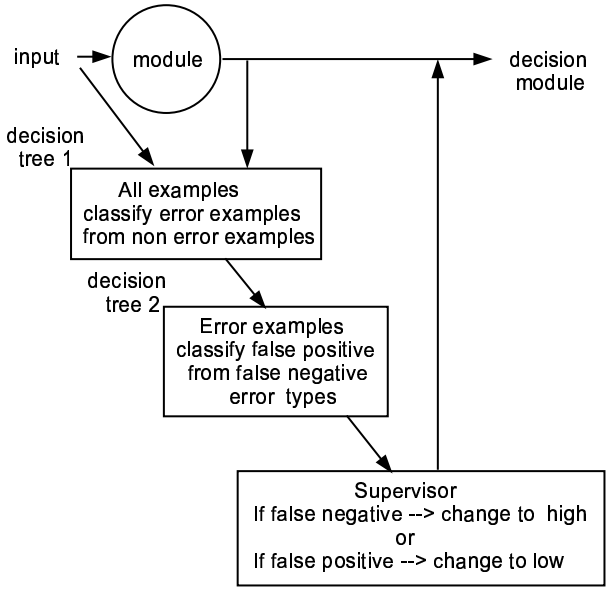

Fig. 5. Decision tree supervision process for modules.

negative errors in 1 of 2 scenarios in experiment 3 — those errors that the decision tree can classify and those that it cannot. In either case, the supervision process proceeds with correcting the module output. If the second decision tree detects an example as being a false positive then the module output is adjusted low or set to 0.1. If the example is a false negative the output is adjusted high or 0.9.

8. Results and Discussion for the Improved Design

The results for training the decision module appear in Table 9. Each case corresponds to a condition that has been applied to the training set as outlined in the steps of the previous section.

Case 1: Training set having classifiable examples only.
Case 2: Training set having unclassifiable examples only.
Case 3.1: Training set using all examples with the unclassifiable examples that the decision tree process can detect, being modified.
Case 3.2: Training set using all examples with the unclassifiable examples that the decision tree process cannot detect, being modified.

From Table 9, it can be seen that the test result of 76.66% is an improvement over our benchmark result of 70.60%. The result for training with unclassifiable examples clearly worsens the result. The test result of 69.23% in Case 3.1, gained by modifying the module outputs where the decision tree correctly classified examples causes error, surprisingly, was found to be much less than that of the benchmark.

Table 9. Decision module result.

Training Set Conditions	Train	Test
Case 1	98.53	76.66
Case 2	51.50	23.85
Case 3.1	73.57	69.23
Case 3.2	93.63	92.66

However, in Case 3.2, the test result of 92.66% exceeded the benchmark for modifying the error examples that the first decision tree in Fig. 5 did not classify.

In the third part of these experiments, it was noted that the first decision tree could detect examples that did not cause error, almost completely. But it could detect a good portion of the examples causing error. In the case of the second decision tree, it did a better job in discriminating between the error types. But where the test result improved dramatically using the second decision tree to train on error examples that the first decision tree could not classify, needs some further investigation for explanation.

The results support a decision process based upon a multilayered perceptron that is assisted to train by decision trees modifying its inputs. The behavior of the modules respond to varying characteristics that appear in the character data. For the examples belonging to a particular character, a portion is learned by the neural network and what it does not learn is learned by the decision tree.

Bench-Capon[2] observed from experiments with an artificial data set that a neural network that learned patterns in the data were not affected by the addition of irrelevant or noisy attributes provided that they had no functional relationship with the output. The neural network was able to learn unimpeded. From this, we could speculate that the neural network learns only character data having certain properties or only that data which is suitable for learning. The studies of Quinlan[11] and Collier and Waugh[4] support this. They observed from experiments with artificial data, that neural networks learned patterns from data where attributes have high independence and decision trees perform better in circumstances where neural networks do not.

The application of decision trees in these experiments in sequence has partitioned the data in an advantageous way, to permit the second decision tree to learn better in the region where the first could not. It could be that the regions of data where the first decision tree could and could not learn are associated with different levels of dependence between the features. We could reason therefore that the unclassifiable examples that do make a difference when modified, have some increasing feature dependence that is favorable for decision tree learning.

An interestingly observation made throughout these experiments, is that the difference between training and test results reduces markedly for the neural network and decision tree combination at the decision level whereas, consulting Table 8, this has not been the case when a neural network or decision tree has been used on its own.

Our test result for the decision module compares with and exceeds in most cases, the test results from other neural network based methods that qualify as stand alone and not ensembled. Currently the best result has been reported by Schwenk and Bengio[15] at 93.9%. We propose further work for our decision module with tests on other multiclass problems to assess its suitability for broader application.

9. Conclusion

Two conclusions regarding task decomposition are drawn from our study. Decomposing a task into subtasks where there is knowledge of the structure of the data can produce modules having a higher performance for their subtask than a single full network does for its one task. The result of task decomposition can be equally effective for different sets of modules found by directing the decomposing process in terms of what is required from it. We have found for the UCI letters recognition problem that task decomposition can be guided by knowledge of the similarity in shape between characters or by class distribution of character information. This suggests that other multiclass problems can be learned more effectively by modular neural network where expert knowledge can be applied to find the modules.

An improved design for a modularized neural nework has emerged from our studies on task decomposition and subtask combination for the overall classification task of alphabetic characters. This design incorporates the modules found through knowledge assisted task decomposition and combines them through a single neural network-based decision module. Our design uses decision trees to learn from the trained state of each character module, to recognize training examples that cannot be classified correctly by the module. The rules induced can be used to clarify the outputs of all the modules that will result in increased classification by the decision module. The addition of a decision tree to supervise the function of each module reduces the additive error effect of the modules and the detriment this has on training the decision module.

Our approach provides an effective and simple solution for the combination of separate subtasks of a modular neural network where the subtasks are highly defined and concurrent. The performance of the modular neural network in our study greatly improves upon the use of a fully connected network for this problem. The test result for the fully connected network in our case was 70.60% classification and the test result for the modular design was 92.66%.

The combination of neural network and decision tree has also been shown to improve with the application of each on its own. In the case of the single fully connected network the training result was 73.35% and test was 70.6%. In the case of the decision tree, the training result was 94.70% and test 77.43%. Both learning processes generalized poorly over test cases when compared with the combined result for training which was 93.63% and for the test 92.66%. The similarity of test result to the training result indicates that effective learning has taken place. Our method for the design of a modularized neural network and its suitability for learning the

recognition of alphabetic characters supports its application on other large multi-class problems while preserving their known advantages over other approaches by reducing computational complexity and training times through parallelism.

References

1. G. Auda and M. Kamel, Modular neural networks: a survey, *Int. J. Neural Syst.* **9**(2) (1999) 129–151.
2. T. Bench-Capon, Neural networks and open texture, in *Proc. Fourth International Conference on Artificial Intelligence and Law* (ACM Press, Amsterdam, 1993).
3. E. J. W. Boers and H. Kuipers, Biological metaphors and the design of modular artificial neural networks, Masters thesis, Leiden Univesity, Netherlands (1992).
4. P. Collier and S. Waugh, *Characteristics of Data Suitable for Learning with Connectionist and Symbolic Methods*, *Proc. Seventh Australion Joint Conf. Artificial Intelligence (AI94)*, Armidale (World Scientific, 1994).
5. Y. Liu, X. Yao and T. Higuchi, Evolutionary ensembles with negative correlation learning, *IEEE Trans. Evolut. Comput.* **4**(4) (2000) 380–387.
6. S. Guan and S. Li, Parallel growing and training of neural networks using output parallelism, *IEEE Trans. Neural Networks* **13**(3) (2002) 542–550.
7. B. Happel and J. M. Murre, Design and evolution of modular neural network architectures, *Neural Networks* **7** (1994) 985–1004.
8. R. Jacobs and M. Jordan, Task decomposition through competition in a modular connectionist architecture: the what and where vision tasks, *Cogn. Sci.* **15** (1991) 219–250.
9. R. Jacobs, M. Jordan, S. Nowlan and G. Hinton, Adaptive mixtures of local experts, *Neural Comput.* **3** (1991) 79–97.
10. S. Pal and S. Mitra, Rough fuzzy MLP: modular evolution, rule generation and evaluation, *IEEE Trans. Knowl. Data Engi.* **15** (2003) 14–25.
11. J. Quinlan, Comparing connectionist and symbolic learning methods, *Computational Learning Theory and Natural Learning Systems*, eds. S. Hanson, G. Drastal and R. Rivest, Constraints and Prospects, Vol. 1 (MIT Press, 1994), pp. 445–456.
12. A. Robins, Catastrophic forgetting, rehearsal and pseudorehearsal, *Connection Sci.* **7** (1995) 123–146.
13. A. Schmidt, A modular neural network architecture with additional generalisation abilities for high dimensional input vectors, Masters thesis, Manchester Metropolitan University (1996).
14. A. Sharkey, *Combining Artificial Neural Nets: Ensemble and Modular Multi Net Systems* (Springer-Verlag, 1999).
15. H. Schwenk and Y. Bengio, Boosting neural networks, *Neural Comput.* **12** (2000) 1889–1900.
16. H. Wang, S. Mukhopadhyay and S. Fang, Feature decomposition architectures for neural networks: algorithms, error bounds, and applications, *Int. J. Neural Syst.* **12**(1) (2002) 69–81.
17. S. Weaver, L. Baird and M. Polycarpou, On the localization of feed forward networks, in *Proc. American Control Conf.*, 1995, Seattle, Washington.

Brent Ferguson studied computer systems and electrical engineering concurrently and obtained Associate Diplomas with distinction for each in 1995 from the School of Mines and Industries of Ballarat, Australia. In 2002 he received the B.Comp. (Hons) from the University of Ballarat, Australia and is currently undertaking a Ph.D. in computing science at this university.

His research interests are based in artificial intelligence. The primary interest is in developing a schema for prediction based on patterns learned from datasets using neural network classifiers. Another interest is in the area of knowledge recovery and the extraction of patterns from databases.

Ranadhir Ghosh, after completing his B.Sc. in computer science and engineering from Bangalore University, India, obtained his M.Sc. in IT from Bond University, and later obtained a Ph.D. from Griffith University in 2003, with an academic excellence award. He is currently a lecturer in the school of ITMS at the University of Ballarat. His expertise is in evolutionary neural learning and its applications. He has many publications in various international journals, conferences, and book chapters.

John Yearwood is Associate Professor of Information Technology at the University of Ballarat. He is leader of the Data Mining and Informatics Research Group in the Center for Informatics and Applied Optimization. Dr. Yearwood has published widely in the areas of decision support and data mining.

His current research interests include narrative models, reasoning and argumentation, ontologies, artificial neural networks and data mining.