FLEXIBILITY AND ACCURACY ENHANCEMENT TECHNIQUES FOR NEURAL NETWORKS

LI PENG
(Master of Engineering, NUS)

A THESIS SUBMITTED FOR

THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2003

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Associate Professor Guan Sheng Uei, Steven. His continuous guidance, insightful ideas, constant encouragement and stringent research style facilitate the accomplishment of this dissertation. His amiable support in my most perplexed time made this thesis thus possible.

Further thanks to my parents for their endless support and encourage throughout my life. Their upbringing and edification is the foundation of all my achievements, in the past and future. My thanks also go to my friends in Digital System and Application Lab. Their friendship always encourages me in my research and life.

Finally, I would like to thank the National University of Singapore for providing me research resources.

Contents

Summary

This thesis focuses on techniques that improve flexibility and accuracy of Multiple Layer Perceptron (MLP) neural network. It covers three topics of incremental learning of neural networks in terms of output attributes, task decomposition based on incremental leaning and feature selection for neural networks with task decomposition.

In the first topic of the thesis, the situation of adding a new set of output attributes into an existing neural network is discussed. Conventionally, when new output attributes are introduced to a neural network, the old network would be discarded and a new network would be retrained to integrate the old and the new knowledge. In this part of my thesis, I proposed three Incremental Output Learning (IOL) algorithms for incremental output learning. In these methods, a new sub-network is trained under IOL to acquire the new knowledge and the outputs from the new sub-network are integrated with the outputs of the existing network when a new output is added. The results from several benchmarking datasets showed that the methods are more effective and efficient than retraining.

In the second topic, I proposed a hierarchical incremental class learning (HICL) task decomposition method based on IOL algorithms. In this method, a K-class problem is divided into K sub-problems. The sub-problems are learnt sequentially in a hierarchical structure. The hidden structure for the original problem's output units is decoupled and the internal interference is reduced. Unlike other task decomposition methods, HICL can also maintain the useful correlation within the output attributes of

a problem. The experiments showed that the algorithm can improve both regression accuracy and classification accuracy very significantly.

In the last topic of the thesis, I propose two feature selection techniques – Relative Importance Factor (RIF) and Relative FLD Weight Analysis (RFWA) for neural network with class decomposition. These approaches involved the use of Fisher's linear discriminant (FLD) function to obtain the importance of each feature and find out correlation among features. In RIF, the input features are classified as relevant and irrelevant based on their contribution in classification. In RFWA, the irrelevant features are further classified into noise or redundant features based on the correlation among features. The proposed techniques have been applied to several classification problems. The results show that they can successfully detect the irrelevant features in each module and improve accuracy while reducing computation effort.

List of Tables

List of Figures

Chapter 1
Introduction

An Artificial Neural Network, or commonly referred to as Neural Network (NN), is an information processing paradigm that works in an entirely different way compared to modern digital computers. The original paradigm of how neural network works is inspired by the way biological nervous systems processes information, such as the human brain. In this paradigm, the information is processed in a complex novel structure, which is composed of a large number of highly interconnected processing elements (neurons) working in unison. The bionic structure permits neural networks to adapt itself to the surrounding environment, so that it can perform useful computation, such as pattern recognition or data classification. This adaptation is carried out by a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true for neural networks as well.[1] Thus, the following definition can be offered to a neural network viewed as an adaptive machine [2]:

> *A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in tow respects:*
> 1. *Knowledge is acquired by the network from its environment through a learning process.*
> 2. *Interneuron connection strengths, known as synaptic weights, are sued to store the acquired knowledge.*

Neural networks process information in a self-adaptive, novel computational structure, which offers some useful properties and capabilities, compared to conventional information processing systems:

- *Nonlinearity.* A neural network, which is composed by many interconnected nonlinear neurons, is nonlinear itself. This nonlinearity is distributed throughout the network and makes neural network suitable for solving complex nonlinear problems, such as nonlinear control functions and speech signal processing.
- *Input-output Mapping.* In supervised learning of neural networks, the network learns from the examples by constructing an input-output mapping for the problem. This property is useful in model-free estimation [3].
- *Adaptivity.* Neural networks have built-in capability to adapt their synaptic weights to changes in the surrounding environment.
- *Evidential Response.* In pattern classification, a neural network can be designed to provide information about the confidence in the decision made, which can be used to reject ambiguous patterns.
- *Contextual Information.* In neural networks, knowledge is represented by the very structure and activation state of a neural network. Because each neuron can be affected by the global activity of other neurons, hence, the contextual information is represented naturally.
- *Fault Tolerance.* If a neural network is implemented in hardware form, its performance degrades gradually under adverse operating conditions, such as damaged connection links, since the knowledge is distributed in the structure of the NN [4].
- *VLSI Implementability.* Because of the parallel framed nature of neural network, it is suitable for implementation using very-large-scale-integrated (VLSI) technology.
- *Uniformity of Analysis and Design.* The learning algorithm in every neuron is common.

- *Neurobiological Analogy.* It is easy for engineers to obtain new ideas from biological brain to develop neural network for complex problems.

Because of the useful properties, neural networks are more and more widely adopted for industrial and research purposes. Many neural network models and learning algorithms have been proposed for pattern recognition, data classification, function approximation, prediction, optimization, and non-linear control. These models of neural networks belong to several categories, such as Multiple Layer Perceptron (MLP), Radial Basis-Function (RBF) [5], self-organizing maps (SOM) [6] and Supported Vector Machine (SVM), etc. Among them, the MLP is the most popular one. In my thesis, I will focus on MLP neural networks only.

The major issues of present neural networks are flexibility and accuracy. Most of neural networks are designed to work in a stable environment. They may fail to work properly when environment changes. As non-deterministic solutions, accuracy of neural networks is always an important problem and has a great room for improvement. In order to improve the flexibility and accuracy of a MLP network, there are three factors that should be considered: (1) the network should be able to adapt itself to the environment changes; (2) the proper network structure should be selected to make maximum use of the information contained in the training data; (3) the training data should be preprocessed to filter out the irrelevant information. In this thesis, I will discuss the issues in detailed.

1.1 Changing Environment – Incremental Output Learning

Usually, a neural network is assumed to exist in a static environment in its learning and application phases. In this situation, the dimensions of output space and input space are fixed and all sets of training patterns are provided prior to the learning of neural network. The network adapts itself to the static environment by updating its link values. However, in some special applications the network can be exposed into a dynamic environment. The parameters may change with time. Generally, the dynamic environment can be classified into the following three situations.

a) Incomplete training pattern set in the initial state: New training patterns (knowledge) are introduced into the existing system during the training process[8][9][10][28].
b) Introduction of new input attributes into the existing system during the training process: it causes an expansion of the input space [26][27].
c) Introduction of new output attributes into the existing system during the training process: it causes an expansion of the output space.

Traditionally, if any of the three situations happens to a neural network, the network structure that is already learnt will be discarded and a new network will be reconstructed to learn the information in the new environment. This procedure is referred to as *retraining* method. There are some serious shortcomings with this retraining method. Firstly, this method does not make use of the information already learnt in the old network. Though the environment has changed, a large portion of the learnt information in the old network is still valid in the new environment. Relearning of this portion of information requires long training time. Secondly, the neural network

cannot provide its service during the retraining, which is unacceptable in some applications. Hence, it is necessary to find a solution to enable it to learn the new information provided incrementally without forgetting the learnt information. Many researchers have proposed such incremental methods for the problems in the first and the second categories, which will be discussed in section 2.1.

During the library research, I cannot find any solutions proposed in literature for the problems in the third category. In fact, such category of problems can be further divided into two groups. If the new output attributes are independent with the old ones, the incremental learning needs only to acquire the new information, since the learnt information is still valid in the new environment. However, if there are conflicts between the new and old output attributes, the learnt information must be modified to meet the new environment while the new information is being learnt. In this thesis, problems belong to this category will be discussed in detail and several solutions will be proposed.

1.2 Network Structure – Task Decomposition with Modular Networks

The most important issue on the performance of a neural network system is its ability to generalize beyond the set of examples on which it was trained. This issue is grievous in some applications, especially in dealing with real-world large-scale complex problems. Recently, there has been a growing interest in decomposing a single large neural network into small modules; each module solves a fragment of the original problem. These modular techniques not only improve the generalization

ability of a neural network, but also increase the learning efficiency and simplify the design [11]. There are some other advantages [12] [13] including: 1) Reducing model complexity and making the overall system easier to understand. 2) Incorporating prior knowledge. The system architecture may incorporate a prior knowledge when there exists an intuitive or a mathematical understanding of problem decomposition. 3) Data fusion and prediction averaging. Modular systems allow us to take into account data from different sources and nature. 4) Hybrid systems. Heterogeneous systems allow us to combine different techniques to perform successive tasks, ranging, e.g., from signal to symbolic processing. 5) They can be easily modified and extended.

The key step of designing a modular system is how to perform the decomposition – using the right technique at the right place and, when possible, estimating the parameters optimally according to a global goal. There are many task decomposition methods proposed in literature, which roughly belong to the following classes.

- *Domain Decomposition*. The original input data space is partitioned into several sub-spaces and each module (for each sub-problem) is learned to fit the local data on each sub-space [11][14]-[17][39][40].
- *Class Decomposition*. A problem is broken down into a set of sub-problems according to the inherent class relations among training data [18][19][42].
- *State Decomposition.* Different modules are learned to deal with different states in which the system can be [20][21][43][44].

In most of the proposed task decomposition methods, each sub-network is trained in parallel and independently with all the other sub-networks. The correlation between

classes or sub-networks is ignored. A sub-network can only use the local information restricted to the classes involved in it. The sub-networks cannot exchange with other sub-networks information already learnt by them. Though the harmful internal interference between the classes is avoided, the global information (or dependency) between the classes is neglected as well. This global information is very useful in solving many problems. Hence, it is necessary to find a new method that utilizes the information transfer between sub-networks while keeping the advantages of a modular system.

1.3 Data Preprocessing – Feature Selection for Modular Neural Network

In section 1.2, I showed that most of task decomposition methods, such as Class Decomposition, split a large scale neural network into several smaller modules. Every module solves a subset of the original problem. Hence, the optimal input feature space that contains features useful in classification for each module is also likely to be a subset of the original one. The input features that are useless for a specified module contained in the original data set can disturb the proper learning of the module. For the purpose of improving classification accuracy and reducing computation effort, it is important to remove the input features that are not relevant to each module. A natural approach is to evaluate every feature and remove those with low importance. This procedure is often referred to as feature selection technique.

In order to evaluate the importance of every input feature in a data set, many researchers have proposed their methods from different perspectives. Roughly, these methods can be classified into the following categories.

1. Neural network performance perspective. The importance of a feature is determined based on whether it helps improve the performance of neural network [22].

2. Mutual information (entropy) perspective. The importance of a feature is determined based on mutual information among input features and input and output features[23][59].

3. Statistic information perspective. The importance of a feature can be evaluated by goodness-score functions based on the distribution of this feature [24][25][60].

A common problem of the existing feature selection techniques is that they need excessive computational time, which is normally longer than training the neural network actually used in application. It is not acceptable in some time-critical applications. It is necessary to find a new technique that utilizes reasonable computation time while removing the irrelevant input features.

1.4 Contribution of the Thesis

In order to improve the performance of the existing neural networks in terms of accuracy, learning speed and network complexity, I have researched in the areas introduced by section 1.1 to 1.3. The research results discussed in this thesis covers the topics of automatic adaptation of the changing environment, task decomposition and feature selection.

In the discussion of automatic adaptation, I proposed three *incremental output learning (IOL)* methods, which were completed newly developed by us. The motivation of these IOL methods is to make the existing neural network automatically adapts to the output space changes, while keeping proper operation during the adaptation process. IOL methods construct and train a new sub-network using the added output attributes based on the existing network. They have the ability to train incrementally and allow the system to modify the existing network without excessive computation. Moreover, IOL methods can reduce the generalization error of the problem compared to conventional retraining method.

In the discussion of task decomposition, a new task decomposition method of *hierarchical incremental class learning* (HICL) is proposed, which is developed based on one of the IOL methods. The objective is to facilities information transfer between classes during training, as well as reduces harmful interference among hidden layers like other task decomposition methods. I also proposed two ordering algorithms of MSEF and MSEF-FLD to determine the hierarchical relationship between the sub-networks. HICL approach shows smaller regression error and classification error than some widely used task decomposition methods.

In the discussion of feature selection, I propose two new techniques that are designed specially for neural networks using task decomposition (class decomposition). The objective is to detect and remove irrelevant input features without excessive computation. These two methods, namely Relative Importance Factor (RIF) and Relative FLD Weight Analysis (RFWA), need much less computation than other

feature selection methods. As an additional advantage, they are also able to analyze the correlation between the input features clearly.

All the methods and techniques proposed in this thesis are designed, developed and tested by the student under the guidance of the supervisor.

In brief, in the thesis, I proposed several new methods and techniques in nearly every stage of neural network development, from pre-processing of data, choosing proper network structure to automatic adapting of environment changes during operation. These methods and techniques are proven to improve the performance of neural network systems significantly with the experiments conducted with real world problems.

1.5 Organization of the Thesis

In this chapter, I have briefly introduced some background information and motivations of my researches, which covers the area of automatic adaptation of the changing environment, task decomposition and feature selection. In chapter 2, I will introduce the IOL methods and prove their validity by experiments. In chapter 3, HCIL method will be introduced. It is proven to have better performance than some other task decomposition methods by experiments. In chapter 4, I will introduce RIF and RFWA feature selection techniques and prove their performance by experiments. The conclusion of the thesis and some suggestions to the future work are given in chapter 5.

Chapter 2

Incremental Learning in Terms of Output Attributes

2.1 Background

Conventionally, the environment in which a neural network is being trained during its learning phase can be assumed to be static, wherein the input and output space together with the training patterns are assumed to be fixed before training. In such an environment, the learning process takes place in the form of “the neural network updating its parameters or by updating its network structure according to the given problem” [26].

However, in the real world, neural networks are often exposed to dynamic environments instead of static ones. Most likely a desiner do not know exactly in which type of environment a neural network is going to be used. Therefore, it would be attractive to make neural network more adaptive, capable of combining knowledge learned in the previous environment with new knowledge acquired in the changed environment [27] automatically. A natural approach to this kind of problems is keeping the main structure of existing neural network unchanged to preserve the learnt information and building additional structures (hidden units or sub-networks) to acquire new information. Because the existing neural network looks like increasing its

structure to adapt it to the changed environment during the process, this approach is often referred as incremental learning.

Changing environment can be classified into three categories:

a) Incomplete training pattern set in the initial state: New training patterns (knowledge) are introduced into the existing system during the training process.
b) Expansion of input space: New inputs are introduced into the existing system.
c) Expansion of output space: New outputs are introduced into the existing system.

Many researchers have come out with incremental learning methods under the first category. Fu et al. [9] presented a method called “Incremental Back-Propagation Learning Network”, which employs bounded weight modification and structural adaptation learning rules and applies initial knowledge to constrain the learning process. Bruzzon et al. [10] proposed a similar method. [8] proposed a novel classifier based on the RBF neural networks for remote-sensing images. [28] proposed a method to combine an unsupervised self-organizing map with a multilayered feedforward neural network to form the hybrid Self-Organizing Perceptron Network for character detection. These methods can adapt network structure and/or parameters to learn new incoming patterns automatically, without forgetting previous knowledge.

For the second category, Guan and Li [26] proposed “Incremental Learning in terms of Input Attributes (ILIA)”. It solves the problem via a “divide and conquer” approach. In

this approach, a new sub-network is constructed and trained using the ILIA methods when new input attributes are introduced to the network. [27] proposed Incremental Self Growing Neural Networks (ISGNN), which implements incremental learning by adding hidden units and links to the existing network.

In the research, I focused on the problems of third category, where one or more new output attributes must be added into the current systems. For example, the original problem has N input attributes and K output attributes. When another output attribute needs to be added into the problem domain, the output vector will contain K+1 elements. Conventionally, the problem is solved by discarding the existing network and redesigning a new network from scratch based on the new output vector and training patterns. However, this approach would waste the previously learnt knowledge in the existing network, which may still be valid in the new environment. The operation of the neural network also has to be broken during the training of new network, which is unacceptable in some applications, especially real-time applications. If self-adapted leaning can be performed quickly and accurately without affecting the operation of the existing network, it will be a better solution compared to merely discarding the existing network and retraining another network [26].

Self adaptation of a neural network with new incoming output attributes is a new research area and I cannot find any methods being proposed in literatures. Through the research, I find that it can be achieved by either external adaptation or internal adaptation. In external adaptation, the problem in a changing environment is decomposed into several sub-problems, which are then solved by sub-networks individually. While the environment is changing, knowledge that is new to the trained

network is acquired by one or more new sub-networks. The existing network remains unchanged during adaptation. The final output is obtained by combining the existing outputs and new outputs (the sub-networks) together. In internal adaptation, the structure of the existing network is adjusted to meet the needs of the new environment. This structural adjustment may include insertion of hidden units or links and change of link weights, etc. In this chapter, I propose three Incremental Output Learning (IOL) methods based on external adaptation.

The rest of the chapter is organized as follows. In section 2.2, details of the IOL methods are introduced. In section 2.3, I present the experiments and results. In section 2.4, I discuss observations made from the experiments. In section 2.5 I summarize my research work in this area.

2.2 External Adaptation Approach: IOL

The external adaptation approach for incremental output learning solves the problem of self adaptation to the changing environment in a “divide and conquer” way. The basic structure is similar to the Modular Neural Networks (MNN) [29] model. This approach divides the changing environment problem into several smaller problems: discarding out-of-date or invalid knowledge, acquiring new knowledge from the incoming attributes and reusing valid learnt knowledge. These sub-problems are then solved with different modules. During the last stage, sub-solutions are integrated via a multi-module decision-making strategy.

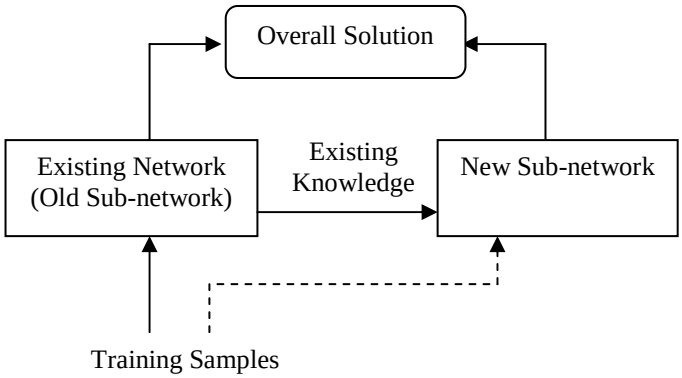


Figure 2.1 The External Adaptation Approach – an Overview

In the proposed IOL methods, the existing network (or old sub-network) is kept unchanged during self-adaptation. This existing sub-network is designed and trained before the environmental change. Its inputs, outputs and training patterns are left untouched as what they were before the environmental change. Reuse of valid learnt knowledge is achieved naturally.

If all the information leant in the existing network is still valid in the changed environment, it can be fully reused in the new structure. In this case, a new sub-network is designed and trained to acquire the new information only. The inputs, outputs and training patterns must cover what are changed at least. However, if some of the learnt information in the existing network is not valid in the new environment, it may make the outputs of the existing network different from what are desired in the new environment. In others words, it may disturb the proper leaning of new information. In this case, it can be considered that there is a "conflict" between the learnt information and new information and the new sub-net work must be able to discard the invalid information while acquiring new information. The inputs, outputs and training patterns should cover not only those are new after environmental change,

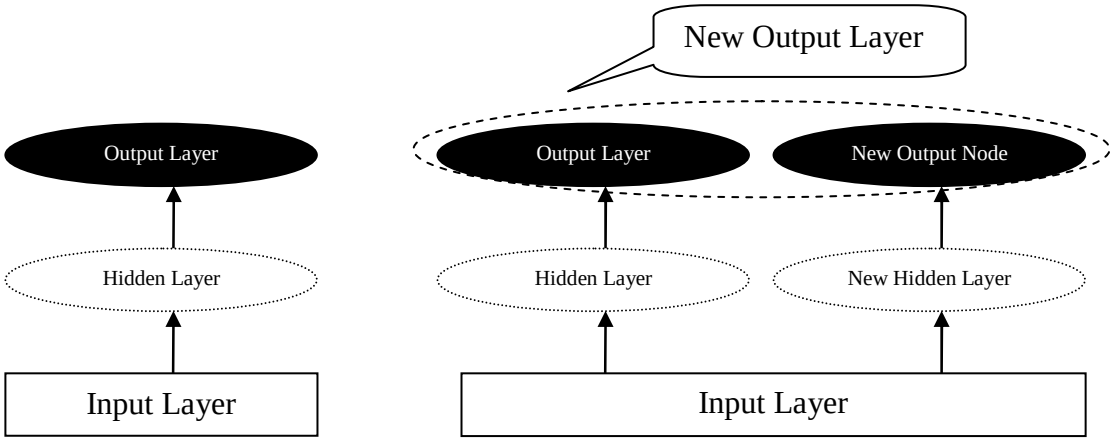
but also some of the original ones before the change, so that it is able to know what learnt information should be discarded. The design of new sub-network is based on the Rprop learning algorithm with one hidden layer and a fixed number of hidden units.

2.2.1 IOL-1: Decoupled MNN for Non-Conflicting Regression Problems

If there is no conflict between the new and learnt knowledge, a regression problem with an increased number of output attributes can be solved using a simple variation of decoupled modular networks.

The network structure of IOL-1 is shown in Figure 2.2. If the new knowledge carried by the new output attribute and training patterns does not bear any conflict with the learnt knowledge, the learnt knowledge in the old sub-network will still be valid under the new environment and does not need any modification. Therefore, the sub-problem of discarding out-of-date or invalid knowledge is avoided. In IOL-1, there is no knowledge exchange between the sub-networks. The new sub-network is trained independently with the old sub-network for the incoming output attribute with all available training patterns. In another word, the new sub-network contains all input attributes and one output attribute. The outputs of the old and new sub-networks together form the complete output layer for the changed environment. When a new input sample is presented at the input layer, the old sub-network and new sub-network work in parallel to generate the final result.

The structure of IOL-1 is very simple because it does not need the multi-module decision-making step as required in normal MNN.



a. Existing Network **b. Integrated Network**

Figure 2.2 IOL-1 Structure

The IOL-1 algorithm is composed of two stages. The procedure is as follows.

Stage 1: the existing network is retained as the old sub-network, as shown in Figure 2(a).

Stage 2: construct and train the new sub-network.

Step 1: Construct an MLP with one hidden layer as the new sub-network. The input layer of the new sub-network receives all input features available and the output layer contains only one output unit representing the incoming output attribute.

Step 2: Use the Cross-Validation Model Selection algorithm [2] to find out the optimal number of hidden units for the new sub-network.

Step 3: Train the new sub-network obtained in step 1.

Because the outputs from the existing network are still valid in the changed environment, they can be used as part of the new outputs directly. The other part of the new outputs that reflects the new information can be obtained directly from the new sub-network. Hence, there is no need to integrate the old and new networks together with any additional process, because they are integrated naturally.

IOL-1 is a variation of the traditional decoupled modular neural networks. It has the advantages of decoupled MNN naturally. For example, it avoids possible coupling among the hidden layer weights and hence reduces internal interference between the existing outputs and the incoming output [26] [30]. Because the old and new sub-networks process input samples in parallel, the input-output response time will not be affected much after adaptation. Another advantage is that the old sub-network (existing network) can continue to carry out normal work during the adaptation process, since the new sub-networks is being trained independently. The last two advantages make IOL-1 perfect for real-time applications.

Though IOL-1 has many advantages, its usage is limited. Because the old sub-network and the new sub-network are independent from each other, the learnt knowledge in the existing network that is no longer valid in the changed environment cannot be discarded by the new sub-network. Therefore, IOL-1 can be used only when there are no conflicts between the new and learnt knowledge. In most regression problems, there are few conflicts so that IOL-1 is suitable. However, in classification problems there are likely conflicts among the new and learnt classification boundaries. It should be noted that in the existing network, each input sample has to be assigned with one

out of the many old class labels. If an input sample meant for the incoming class is presented to IOL-1, both the new and old network will assign a different class label to it. This will be a problem for IOL-1. Hence, IOL-1 is not suitable for classification problems.

2.2.2 IOL-2: Decoupled MNN with Error Correction for Regression and Classification Problems

In order to handle the sub-problem of discarding invalid knowledge in the existing network, IOL-2 is developed from IOL-1 based on an "error generation and error correction" model. In such a model, the old sub-network will produce a solution based on the learnt knowledge when a sample associated with the new output attribute is presented at the input layer. This solution will not be accurate because the existing output attributes do not have the knowledge carried by the incoming attribute. Hence, there is always an error between the existing output and the new desired output in the changed environment. In IOL-2, this error is "corrected" by a new sub-network that runs in parallel with the old sub-network. In another word, a new sub-network is trained to minimize the error between the combined solution from the old and new sub-networks and the desired solution for each input sample.

IOL-2 is composed of two stages. The procedure is as follows.

Stage 1: the existing network is retained as the old sub-network, as shown in Figure 2.3.

Stage 2: construct and train the new sub-network.

Step 1: Construct an MLP with one hidden layer as the new sub-network. The input layer of the new sub-network receives all input features available and the output layer contains K+1 units, where K is number of output units in the existing network.

Step 2: Use the Cross-Validation Model Selection algorithm to find out the optimal number of hidden units for the new sub-network.

Step 3: Train the new sub-network obtained in step 1 to minimize the difference between the desired solutions and the combined solutions from the old and new sub-networks when training samples are presented at the input layer.

In IOL-2, the output layer of the new sub-network integrates the output form old network and new information obtained in the hidden layer of the new sub-network. Learnt information that is invalid in the changed environment from the old network is also discarded by this output layer.

IOL-2 has the same advantages as IOL-1. The existing network can work normally when adapting to the changed environment. The network depth will not be changed. It is suitable for real-time applications.

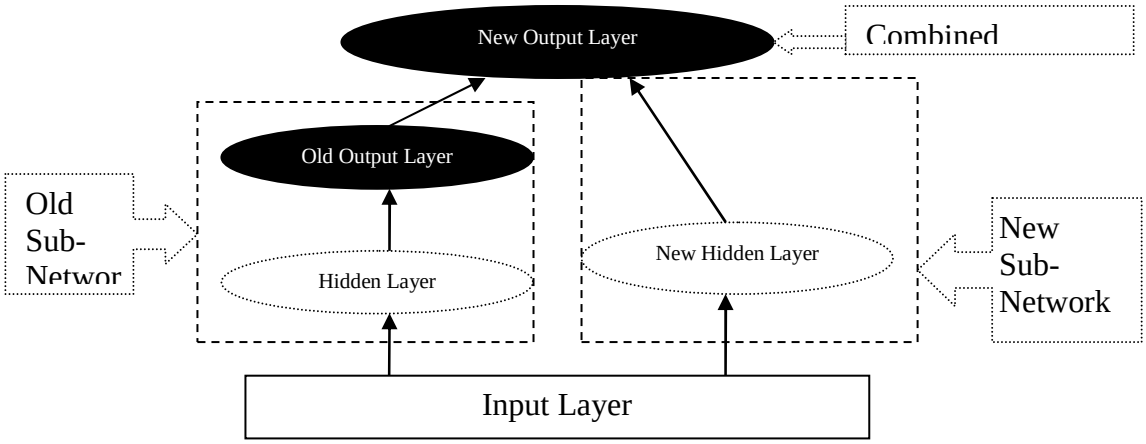


Figure 2.3 IOL-2 Structure

2.2.3 IOL-3: Hierarchical MNN for Regression and Classification Problems

In IOL-1, the sub-problem of discarding invalid learnt knowledge is avoided. In IOL-2, this sub-problem is solved by modifying the objective function of the new sub-network to minimize the error of the combined solution of the old and new networks. In IOL-3, I try to solve this sub-problem together with new knowledge acquiring in the same new sub-network.

Unlike IOL-1 and IOL-2, IOL-3 is implemented with a hierarchical neural network [31]. The new sub-network is sitting “on top of” the old sub-network instead of sitting in parallel with it, which is shown in figure 2.4.

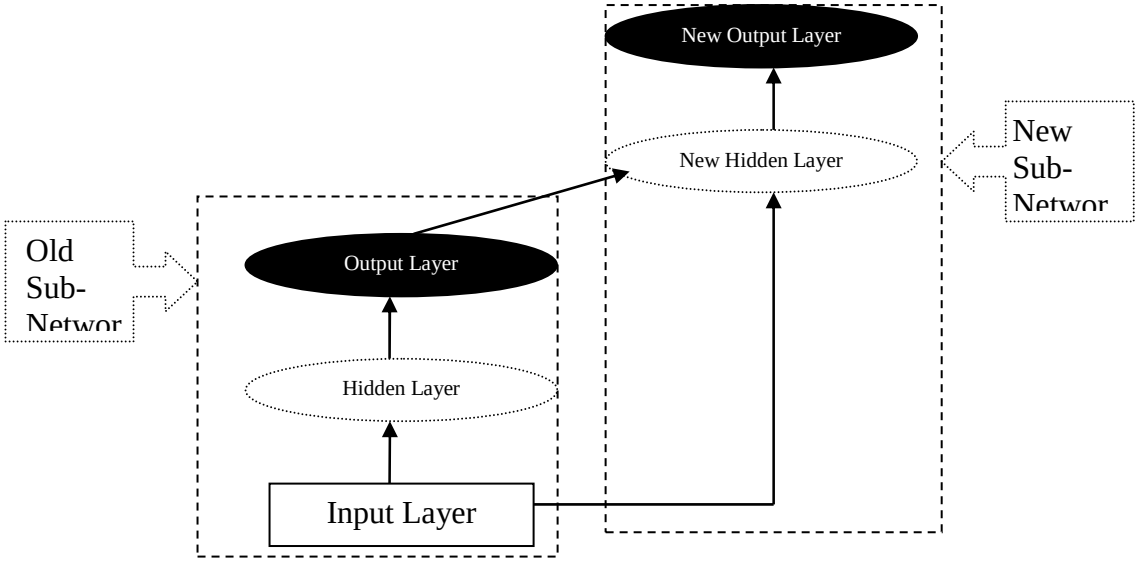


Figure 2.4 IOL-3 Structure

IOL-3 is composed of three stages. The procedure is as follows.

The first stage of IOL-3 is the same as IOL-1.

Stage 2 of IOL-3 is as follows:

Step 1: Construct a new sub-network with K+N input units and K+1 output units, where K is the number of existing output attributes and N is number of input attributes of the original problem.

Step 2: Feed input samples to the existing network; combine the outputs of the existing network together with the original inputs to form as new inputs to the new sub-network. Train the new sub-network with the patterns presented.

In stage 2, when an unknown sample is presented to the input layer, it should be fed into the existing network first. Then the output attributes of the existing network

together with the original inputs will be fed into the new sub-network as inputs. The output attributes of the new sub-network produce the overall outputs.

The new sub-network in IOL-3 not only acquires the new information in the changed environment, but also integrates the outputs from the old sub-network with the new information and discards any invalid information carried by the old network.

In IOL-3, the old sub-network acts as an input data pre-processing unit. It presents to the new sub-network pre-classified (in classification problems) or pre-estimated input attributes (in regression problems), so that the new sub-network can use this knowledge to build its own classification boundaries or make its own estimates of the output attributes. The knowledge passed between the two sub-networks is direct forward in a serial manner. The new sub-network solves all the three sub-problems of discarding invalid knowledge, acquiring new knowledge from the incoming output attributes and retaining valid knowledge at the same time.

Compared with IOL-1 and IOL-2, the cooperation between the old and new sub-networks in IOL-3 is better and efficient. The training time of the new sub-network can be significantly reduced. However, the network depth is increased as the depth of the new sub-network is added on top of the existing network. This may be undesirable for real time applications. The existing network can also continue with its work during the adaptation process in IOL-1 and IOL-2.

2.3 Experiments and Results

Three benchmark problems, namely Flare, Glass and Thyroid, are used to evaluate the performance of the proposed IOL methods. The first problem is a regression problem and the other two are classification problems. All the three problems are taken from the PROBEN1 benchmark collection [32].

2.3.1 Experiment Scheme

The simulation of IOL methods is implemented in the MATLAB environment with the Rprop [33] learning algorithm.

The stopping criteria can influent the performance of an MNN significantly. If training is too short, the network cannot acquire enough knowledge to obtain a good result. If training is too long, the network may experience over-fitting. In over-fitting, a network simply memorizes the training patterns, which will lead to poor generalization performance. In order to avoid this problem, early stopping with validation is adopted in the simulation. In the thesis, the set of available patterns is divided into three sets: a *training set* is used to train the network, a *validation set* is used to evaluate the quality of the network during training and to measure over-fitting, and a *test set* is used at the end of training to evaluate the resultant network. The sizes of the training, validation, and test are 50%, 25% and 25% of the problem's total available patterns respectively.

There are three important metrics when the performance of a neural network system is evaluated. They are accuracy, learning speed and network complexity. As to accuracy, I use regression or classification error of the test patterns as the most important metric. I also use error of the test patterns to measure the generalization ability of the system.

When dealing with the learning speed, it should be considered that there is significant difference between the number of hidden units in each sub-problem of IOL and retraining. As a result, the computation time of each epoch in the sub-networks varies significantly. Hence, each solution (each IOL method or retraining) should be taken as a whole and independent with the structure and complexity of networks. In order to achieve that, I emphasize on adaptation time instead of training time, which means the time needed for each method to achieve its best accuracy after the environmental change. Since the old sub-network is treated as existed before performing IOL, the adaptation time of IOL should be measured by the training time of the new sub-network only. When network complexity is concerned, I use the number of newly added hidden units as a metric.

The experimental results of IOL methods were compared to the results of retraining method, which is the only known way to solve the changing output attributes problem besides IOL methods in literatures.

The structure of new sub-networks and retraining networks are determined by the Cross-Validation Model Selection technique. To simplify the simulation, the old sub-network is simulated with a fixed structure with a single hidden layer and 20 hidden units.

2.3.2 Generating Simulation Data

In nature, incremental leaning of output attributes can be classified into two categories. In the first category, the incoming output attribute and the new training patterns contains completely new knowledge. For example, a polygon classifier was trained to

classify squares and triangles. Now, we need it to classify a new class of diamonds besides previously learnt classes. There is no clear dependency or conflict between the existing output attributes and the new one. In the second category, the incoming output attribute could be a sub-set of one or more existing attributes, which is normally referred to as reclassification. For example, the classifier discussed above is required to classify equilateral triangles from all triangles. The proposed IOL methods are suitable for both categories[1]. However, I only adopt the first category of problems in the experiments for IOL because reclassification problems have been well studied already.

The simulation data for incremental output learning is obtained from several benchmark problems. Since the benchmark problems are real world problem, it would be difficult to generate new data to simulate a new incoming output attribute ourselves in order to reflect the true nature of the dataset. To simulate the old environment before inserting the incoming output attribute, training data for the existing network is generated by removing a certain output attribute from all training patterns in the benchmark problem. The original data of the benchmark problem without any modification is used to simulate the new environment after inserting a new output attribute.

2.3.3 Experiments for IOL-1

As stated in section 2.2.1, IOL-1 is suitable for regression problems only. Hence, the experiments are conducted with the Flare problem using each different output attribute as the incoming output attribute. This problem predicts solar flares by trying to guess

[1] Please refer to section 2.4.2 for detailed discussions.

the number of solar flares of small, medium, and large sizes that will happen during the next 24-hour period in a fixed active region of the Sun surface. Its input values describe previous flare activity and the type and history of the active region. Flare has 24 inputs (10 attributes), 3 outputs, and 1066 patterns.

Table 2.1 shows the generalization performance of IOL-1 with different number of hidden units in the new sub-network and different output attribute being treated as the incoming output. Also listed is the generalization performance of retraining with different number of hidden units. This data is used for cross-validation model selection.

Table 2.1 Generalization Error of IOL-1 for the Flare Problem with Different Number of Hidden Units

Number of hidden units	1st output as the incoming output	2nd output as the incoming output	3rd output as the incoming output	Retraining with old and new outputs
1	**0.0028**	0.0029	**0.0029**	0.003
3	0.0028	**0.0028**	0.0031	0.003
5	0.0028	0.0033	0.003	**0.0029**
7	0.0033	0.003	0.0034	0.003
9	0.0031	0.0031	0.0033	0.003
11	0.0033	0.0032	0.0033	0.003
13	0.0036	0.0036	0.0039	0.0029
15	0.0036	0.0034	0.0036	0.003
17	0.0037	0.0035	0.0039	0.003
19	0.0039	0.0036	0.0038	0.0028
21	0.0038	0.0037	0.0038	0.003
23	0.0038	0.0036	0.0038	0.0029
25	0.0039	0.004	0.0039	0.0032
27	0.0043	0.004	0.004	0.0028
29	0.0042	0.004	0.0038	0.0028

Notes:
1. *Numbers in the first column stand for the numbers of hidden units for the new sub-networks in IOL-1 and numbers of hidden units for the overall structures in retraining.*
2. *The number of hidden units for the old sub-networks is set to 20 always.*
3. *The values in the table represent regression errors of the overall structures with different number of hidden units.*

We can find that the new sub-networks require only one or three hidden units to obtain good generalization performance. However, the generalization performance of IOL-1 drops rapidly due to the problem of over-fitting, when the number of hidden units in the new sub-network increases. The generalization performance of retraining remains stable with various numbers of hidden units. The new sub-network is trained to solve a sub-problem with single output attribute, which is much simpler than the retraining problem with 3 output attributes. Because of the simplicity of the problem being solved, the new sub-network turns to memorize the training patterns instead of acquiring valid knowledge from the patterns. This is why the over-fitting problem of IOL-1 is more serious than retraining.

Table 2.2 shows the performance of IOL-1 (test error) and retraining with properly selected structures in the last step. In this table, I choose 1 hidden unit for the new sub-network when the 1st or 3rd output is used as the incoming output, 3 hidden units for the new sub-network when the 2nd output is used as the incoming output and 5 hidden units for retraining.

Table 2.2 Performance of IOL-1 and Retraining with the Flare Problem

	Test error	Adaptation time	No. of hidden units
IOL-1 with 1st output as incoming output	0.0028	0.789 (22.75%)	1
IOL-1 with 2nd output as incoming output	0.0029	0.8492 (16.86%)	3
IOL-1 with 3rd output as incoming output	0.0028	0.9014 (11.75%)	1
Retraining	0.0029	1.0214	5

Notes:

1. *The number of hidden units measured in IOL methods is for the new sub-network only.*
2. *Adaptation time shows the time needed for each methods to provide its most accuracy solution in the changed environment respectively. It equals to the*

training time of new sub-network for IOL methods and the training time for retraining method.

3. *The number in '( )' is adaptation time reduction in percentage compared to retraining.*

In this experiment, the accuracy of IOL-1 is slightly better than retraining. Compared to retraining, IOL-1 needs much fewer new hidden units to adapt itself to the changed environment, which directly results in less adaptation time. The adaptation time of IOL-1 is 22.75% less than retraining.

2.3.4 Experiments for IOL-2

IOL-2 contains a generalized decoupled MNN structure and is suitable for both regression and classification problems. The experiments are conduced with the Flare, Glass and Thyroid problems for it.

- **Flare Problem**

Table 2.3 shows the generalization performance of IOL-2 with different number of hidden units in the new sub-network and each output attribute being treated as the incoming output. Also listed is the generalization performance of retraining with different number of number of hidden units.

Table 2.3 Generalization Error of IOL-2 for the Flare Problem with Different Number of Hidden Units

Number of hidden units	1st output as the incoming output	2nd output as the incoming output	3rd output as the incoming output	Retraining with old and new outputs
1	0.0247	0.04	0.1593	0.003

3	**0.0031**	**0.0032**	**0.0028**	0.003
5	0.0031	0.003	0.0031	**0.0029**
7	0.0033	0.0036	0.0031	0.003
9	0.0035	0.0034	0.0036	0.003
11	0.0039	0.0036	0.0039	0.003
13	0.004	0.0036	0.0045	0.0029
15	0.0042	0.0046	0.0044	0.003
17	0.0054	0.0044	0.0051	0.003
19	0.0046	0.0047	0.0044	0.0028
21	0.0053	0.0044	0.005	0.003
23	0.0051	0.0053	0.0049	0.0029
25	0.0049	0.0058	0.0053	0.0032
27	0.0055	0.0064	0.0051	0.0028
29	0.0055	0.0055	0.0056	0.0028

Notes:
1. *Numbers in the first column stand for the numbers of hidden units for the new sub-networks in IOL-2 and numbers of hidden units for the overall structures in retraining.*
2. *The Number of hidden units for the old sub-networks is set to 20 always.*
3. *The values in the table represent the regression errors of the overall structures with different number of hidden units.*

The number of hidden units in each new sub-problem is selected as 3 for each output used as the incoming output. Table 2.4 shows the performance of IOL-2 when such configuration is used.

Table 2.4 Performance of IOL-2 and Retraining with the Flare Problem

	Test error	Adaptation time	No. of hidden units
IOL-2 with 1st output as incoming output	0.003	1.0214 (0%)	3
IOL-2 with 2nd output as incoming output	0.003	1.0676 (-4.5%)	3
IOL-2 with 3rd output as incoming output	0.0028	0.9154 (10.38%)	3
Retraining	0.0029	1.0214	5

Notes: 1-3. refer to notes under Table 2.2

Compared to retraining, IOL-2 needs 1.96% less adaptation time in average. The test error is very close to retraining. The differences between the test errors of IOL-2 and retraining are within the rage of ±0.0001, or 3.5%.

- **Glass Problem**

This data set is used to classify glass types. The results of a chemical analysis of glass splinters (percentage of 8 different constituent elements) plus the refractive index are used to classify a sample to be either float processed or non-float processed building windows, vehicle windows, containers, tableware, or head lamps. This task is motivated by forensic needs in criminal investigation. This data set contains 9 inputs, 6 outputs, and 214 patterns.

Since the Glass problem is a classification problem, classification error is used instead of regression in the last problem to conduct cross-validation model selection. Table 2.5 shows the classification error of IOL-2 with different number of hidden units in the new sub-problem and retraining.

Table 2.5 Classification Error of IOL-2 for the Glass Problem with Different Number of Hidden Units

Number of hidden units	1st output as the incoming output	2nd output as the incoming output	3rd output as the incoming output	4th output as the incoming output	5th output as the incoming output	6th output as the incoming output	Retraining with old and new outputs
1	0.4755	0.566	0.4528	0.3925	0.5132	0.5774	0.7434
3	0.4226	0.5245	0.3208	0.283	0.317	0.3358	0.4
5	0.4151	0.4679	0.3132	0.3132	0.3472	0.3547	0.3849
7	0.4151	0.5019	0.3094	0.3057	0.3434	0.3057	0.3547
9	0.3698	0.4302	0.317	0.3396	0.3321	0.3057	0.3283
11	0.3283	0.4	**0.2906**	0.3358	0.3057	0.3283	0.3509
13	0.4189	0.3736	0.317	0.2868	0.3283	0.3208	0.317
15	0.3245	**0.3019**	0.3208	**0.283**	0.3358	0.2943	0.317

17	0.3283	0.3321	0.3132	0.3094	0.3208	0.3019	0.3132
19	0.3887	0.3472	0.3132	0.2981	**0.3019**	0.2981	0.3358
21	0.3472	0.3358	0.317	0.3245	0.3057	**0.2868**	**0.3019**
23	0.3208	0.3396	0.3094	0.3094	0.3019	0.3132	0.3358
25	0.3396	0.3509	0.3019	0.3019	0.3358	0.3283	0.3094
27	0.3283	0.3396	0.317	0.3019	0.3321	0.3283	0.3208
29	**0.317**	0.3283	0.3132	0.3057	0.3358	0.2943	0.3132

Notes: 1. Numbers in the first column stand for the numbers of hidden units for the new sub-networks in IOL-2 and numbers of hidden units for the overall structures in retraining.
2. The number of hidden units of the old sub-networks is set to 20 always.
3. The values in the table represent the classification errors of the overall structures with different number of hidden units.

The number of hidden units in the new sub-networks is 29, 15, 11, 15, 19 and 21 respectively when different output is used as incoming output. The network used for retraining requires 21 hidden units. Table 2.6 shows the performance (classification error of test set) of IOL-2 compared with retraining.

Table 2.6 Performance of IOL-2 and Retraining with the Glass Problem

	Test classification error	Adaptation time	No. of hidden units
IOL-2 with 1st output as incoming output	0.3094	0.9936 (-3.5%)	29
IOL-2 with 2nd output as incoming output	0.3395	0.9232 (3.79%)	15
IOL-2 with 3rd output as incoming output	0.3170	0.931 (2.98%)	11
IOL-2 with 4th output as incoming output	0.3358	0.9458 (1.44%)	15
IOL-2 with 5th output as incoming output	0.3208	1.0156 (-5.8%)	19
IOL-2 with 6th output as incoming output	0.2868	0.913 (4.9%)	21
Retraining	0.3396	0.9596	21

Notes: 1-3. refer to notes under Table 2.2

- **Thyroid Problem**

Thyroid diagnoses whether a patient's thyroid has overfunction, normal function, or underfunction based on patient query data and patient examination data. Thyroid has 21 inputs (21 attributes), 3 outputs, and 7200 patterns.

Table 2.7 shows the classification error under cross-validation model selection.

Table 2.7 Classification Error of IOL-2 for the Thyroid Problem with Different Number of Hidden Units

Number of hidden units	1st output as the incoming output	2nd output as the incoming output	3rd output as the incoming output	Retraining with old and new outputs
1	0.0343	0.1828	0.047	0.0628
3	0.0232	0.0292	0.0227	0.0244
5	0.019	0.0298	0.0262	0.021
7	0.0221	0.0237	0.0188	0.0208
9	0.019	0.0217	0.02	0.0203
11	**0.0189**	0.0204	0.0206	0.0201
13	0.0213	**0.0202**	0.0194	0.0183
15	0.0212	0.0221	0.0201	0.0211
17	0.0201	0.0217	0.0188	0.0196
19	0.0217	0.0266	0.0193	0.0199
21	0.0236	0.0238	**0.0177**	0.0192
23	0.0224	0.0238	0.0193	**0.0181**
25	0.0208	0.0204	0.0188	0.0184
27	0.0223	0.0226	0.0192	0.0189
29	0.0224	0.0223	0.019	0.0193

Notes:
1. *Numbers in the first column stand for the numbers of hidden units for the new sub-networks in IOL-2 and numbers of hidden units for the overall structures in retraining.*
2. *The number of hidden units of the old sub-networks is set to 20 always.*
3. *The values in the table represent the classification errors of the overall structures with different number of hidden units.*

The number of the new sub-networks with each output as the incoming output is set to 11, 13 and 21 respectively. The number of hidden units for retraining is set to 23. The results of IOL-2 with properly selected structures are shown in Table 2.8.

Table 2.8 Performance of IOL-2 and Retraining with the Thyroid Problem

	Test classification error	Adaptation time	No. of hidden units
IOL-2 with 1st output as incoming output	0.0214	9.9158 (47.41%)	11
IOL-2 with 2nd output as incoming output	0.0217	35.6856 (-89.26%)	13
IOL-2 with 3rd output as incoming output	0.019	25.951 (-37.63%)	21
Retraining	0.0191	18.8554	23

Notes: 1-3. refer to notes under Table 2.2

From the results of these three problems, we can find that IOL-2 provides reasonable generalization accuracy with slightly shorter adaptation time compared to retraining in most cases. However, adaptation time is problem dependent. If an incoming class is hard to be classified in nature, the adaptation time will be much longer. For example, IOL-2 needs 89.26% and 37.63% more adaptation time than retraining, when the 2nd or 3rd class is used in Thyroid as the incoming class. The complexity of the new sub-network is lower than the network used for retraining.

2.3.5 Experiments for IOL-3

IOL-3 is developed to overcome the disadvantages of IOL-2. It needs much less adaptation time than IOL-2.

- **Flare Problem**

Table 2.9 shows the regression error of the Flare 1 problem under cross-validation model selection.

Table 2.9 Generalization Error of IOL-3 for the Flare Problem with Different Number of Hidden Units

Number of hidden units	1st output as the incoming output	2nd output as the incoming output	3rd output as the incoming output	Retraining with old and new outputs
1	0.0032	0.0033	0.0033	0.003
3	0.003	**0.0027**	0.0031	0.003
5	0.0029	0.0029	0.0031	**0.0029**
7	**0.0028**	0.0029	**0.0028**	0.003
9	0.0029	0.0028	0.003	0.003
11	0.0028	0.0029	0.0029	0.003
13	0.0029	0.003	0.0028	0.0029
15	0.0029	0.0029	0.0028	0.003
17	0.0031	0.0031	0.0031	0.003
19	0.0028	0.003	0.0029	0.0028
21	0.0029	0.003	0.0028	0.003
23	0.003	0.0029	0.003	0.0029
25	0.0029	0.0031	0.0029	0.0032
27	0.0029	0.003	0.0031	0.0028
29	0.0029	0.003	0.003	0.0028

Notes:
1. *Numbers in the first column stand for the numbers of hidden units for the new sub-networks in IOL-3 and numbers of hidden units for the overall structures in retraining.*
2. *The number of hidden units of the old sub-networks is set to 20 always.*
3. *The values in the table represent the regression errors of the overall structures with different number of hidden units.*

From Table 2.9, the number of hidden units of the new sub-networks when the 1st, 2nd and 3rd output is used as the incoming output is set to 7, 3 and 7 respectively. Table 2.10 shows the results when such a configuration is used.

Table 2.10 Performance of IOL-3 and Retraining with Flare Problem

	Test Classification Error	Adaptation Time	No of Hidden units
IOAL-3 with 1st output as incoming output	0.0029	0.7642 (25.18%)	7
IOAL-3 with 2nd output as incoming output	0.003	0.813 (20.4%)	3
IOAL-3 with 3rd	0.003	0.807	7

output as incoming output		(20.99%)	
Retraining	0.0029	1.0214	5

Notes: 1-3. refer to notes under Table 2.2

- **Glass Problem**

Table 2.11 shows the classification error used for cross-validation model selection.

Table 2.11 Classification Error of IOL-3 for the Glass Problem with Different Number of Hidden Units

Number of hidden units	1st output as the incoming output	2nd output as the incoming output	3rd output as the incoming output	4th output as the incoming output	5th output as the incoming output	6th output as the incoming output	Retraining with old and new outputs
1	0.6868	0.6717	0.683	0.5774	0.6151	0.6528	0.7434
3	0.4792	0.366	0.3094	0.3962	0.3321	0.3774	0.4
5	0.3887	0.3472	0.366	0.3094	0.3321	0.3472	0.3849
7	0.3132	0.3472	0.3057	0.4	0.317	**0.2868**	0.3547
9	0.3132	0.3698	0.2981	0.3208	0.3396	0.317	0.3283
11	0.3094	0.3208	0.3245	0.3057	0.3094	0.3057	0.3509
13	0.3472	**0.317**	0.3094	0.317	0.3208	0.3245	0.317
15	0.3283	0.3698	0.3208	0.3358	**0.3019**	0.3208	0.317
17	0.3094	0.3509	0.3208	**0.3019**	0.317	0.3057	0.3132
19	0.3585	0.3396	0.3094	0.3019	0.3208	0.3208	0.3358
21	0.366	0.3208	0.2981	0.3057	0.3245	0.3094	**0.3019**
23	0.3132	0.3321	0.3057	0.3132	0.3132	0.3132	0.3358
25	0.3472	0.3434	**0.2906**	0.3057	0.3019	0.3245	0.3094
27	**0.2981**	0.3321	0.2981	0.3019	0.3057	0.3019	0.3208
29	0.3396	0.3396	0.3396	0.317	0.3321	0.3208	0.3132

Notes:
1. *Numbers in the first column stand for the numbers of hidden units for the new sub-networks in IOL-3 and numbers of hidden units for the overall structures in retraining.*
2. *The number of hidden units of the old sub-networks is set to 20 always.*
3. *The values in the table represent the regression errors of the overall structures with different number of hidden units.*

The number of hidden units for new sub-networks is set to 27, 13, 25, 17, 15 and 7 respectively when different outputs are used as incoming output. The results with such a configuration are shown in table 2.12.

Table 2.12 Performance of IOL-3 and Retraining with the Glass Problem

	Test classification error	Adaptation time	No. of hidden units
IOL-3 with 1st output as incoming output	0.3132	0.799 (16.74%)	27
IOL-3 with 2nd output as incoming output	0.3019	0.7534 (21.5%)	13
IOL-3 with 3rd output as incoming output	0.3123	0.735 (23.4%)	25
IOL-3 with 4th output as incoming output	0.2981	0.8514 (11.3%)	17
IOL-3 with 5th output as incoming output	0.3094	0.779 (18.8%)	15
IOL-3 with 6th output as incoming output	0.3094	0.7992 (16.7%)	7
Retraining	0.3396	0.9596	21

Notes: 1-3. refer to notes under Table 2.2.

- **Thyroid Problem**

Table 2.13 shows the classification error used in cross-validation model selection.

Table 2.13 Classification Error of IOL-3 for the Thyroid Problem with Different Number of Hidden Units

Number of hidden units	**1st output as the incoming output**	**2nd output as the incoming output**	**3rd output as the incoming output**	**Retraining with old and new outputs**
1	0.06	0.0554	0.0467	0.0628
3	0.0233	0.0206	0.0196	0.0244
5	**0.0181**	0.0187	0.0176	0.021
7	0.0231	0.0189	0.0179	0.0208
9	0.0211	0.0196	0.0188	0.0203
11	0.0229	0.0203	0.018	0.0201
13	0.0204	**0.0177**	0.0204	0.0183
15	0.0204	0.0193	0.0192	0.0211
17	0.0184	0.0187	0.0193	0.0196
19	0.0204	0.0198	0.0207	0.0199

21	0.0194	0.0179	0.0206	0.0192
23	0.0183	0.0182	0.0183	**0.0181**
25	0.0183	0.0209	**0.017**	0.0184
27	0.0224	0.0211	0.0192	0.0189
29	0.0196	0.0216	0.0186	0.0193

Notes: *1. Numbers in the first column stand for the numbers of hidden units for the new sub-networks in IOL-3 and numbers of hidden units for the overall structures in retraining.*
2. The number of hidden units of the old sub-networks is set to 20 always.
3. The values in the table represent the regression errors of the overall structures with different number of hidden units.

Numbers of hidden units in the new sub-networks when different output is used as incoming output are set to 5, 13 and 25 respectively. Table 2.14 shows the results with such a configuration.

Table 2.14 Performance of IOL-3 and Retraining with the Thyroid Problem

	Test classification error	Adaptation time	No. of hidden units
IOL-3 with 1st output as incoming output	0.0197	5.262 (72.1%)	5
IOL-3 with 2nd output as incoming output	0.02	9.113 (51.7%)	13
IOL-3 with 3rd output as incoming output	0.0197	5.1392 (72.7%)	25
Retraining	0.0191	18.8554	23

Notes: 1-3. refer to notes under Table 2.2

The experiments of the three problems show that IOL-3 has good performances for both regression and classification problems. It has significantly reduced the adaptation time (up to 72.7% reduction), while achieving similar or better accuracy compared to retraining.

2.4 Discussions

2.4.1 The IOL Methods

As mentioned before, IOL decomposes the problem of incremental output learning for an existing network into sub-problems of discarding invalid old knowledge, reusing existing knowledge and acquiring new knowledge through the cooperation of the old (existing) and new sub-networks. They show great advantages over retraining methods, which was the only known solution when the number of outputs increased. The difference between IOL-1, IOL-2 and IOL-3 lies in the flow of knowledge/information between the old and new sub-networks.

In IOL-1, the old and new sub-networks are completely independent. The new sub-network cannot affect the results of old sub-network. Hence, the knowledge in the old sub-network cannot be discarded. On the other hand, the old sub-network cannot contribute in training of the new sub-network either. However, the new sub-network only needs to solve a simple problem with one output attribute instead of solving the whole problem in retraining. It benefits from the nature of a decoupled MNN. IOL-1 reduces adaptation time and slightly improves the performance compared to retraining.

In IOL-2, the knowledge flow from the new sub-network to the old sub-network is enabled by changing the objective of the new sub-network as minimizing the error produced by the old sub-network in the changed environment. It is possible for the new sub-network to discard knowledge that is no longer valid in the changed environment. This feature makes IOL-2 suitable for both regression and classification problems. However, the old and new sub-networks work under the “error generation and error correction” model. The new network needs to put down much effort to

correct the errors produced by the old sub-network, which might be difficult due to the fuzzy nature of the old sub-network. In another word, the new objective of IOL-2 might be difficult to achieve in some problems. In the experiments conducted, IOL-2 reduces adaptation time in most cases. However, in some extreme cases, it needs longer adaptation time than retraining.

In IOL-3, two one-directional knowledge flows are enabled between the two sub-networks by using a hierarchical MNN. The old sub-network supplies the learnt knowledge directly to the new sub-network as part of the inputs to the new sub-network. The new sub-network determines whether the knowledge supplied to it is valid and discards it when necessary during the learning process. The two sub-networks work in a cooperative manner. Compared to IOL-2, the training of the new sub-network is much easier with the help of the old sub-network. The adaptation time of IOL-3 is the shortest among the proposed methods. It also gives better classification accuracy than the other methods. However, the real-time response of IOL-3 is the worst among the three, due to its increased network depth.

The greatest advantage of the proposed IOL methods is that the original neural network provides non-disturbed service when adapting itself to the environmental changes, which is important for real world applications, especially some real-time systems. IOL-1 and IOL-3 also significantly reduced adaptation time while keeping high accuracy when compared to retraining methods.

Although no automatic adaptation methods other than IOL are proposed for changing output attributes problems in literatures, some prior work for automatic adaptation of

changing input attributes have been done, such as ILIA [26] proposed by Guan & Li. The proposed IOL methods follow the pioneer track of ILIA. The idea of incremental learning can also be applied to intelligent systems other than neural networks, for example, genetic algorithms (GA). In [63], Guan and Zhu suggested an incremental output learning algorithm for GA, which is proven to be faster and more accurate than retraining for many problems.

In addition, although experiments was conducted with one incoming output only in the research, the IOL methods can be extended easily to accommodate multiple incoming outputs by repeating the learning steps of the methods described.

2.4.2 Handling Reclassification Problems

IOL-2 and IOL-3 are also suitable for reclassification problems. In reclassification, a new output attributes may be formed as a subset of one or more existing output attributes, which is shown in figure 2.5.

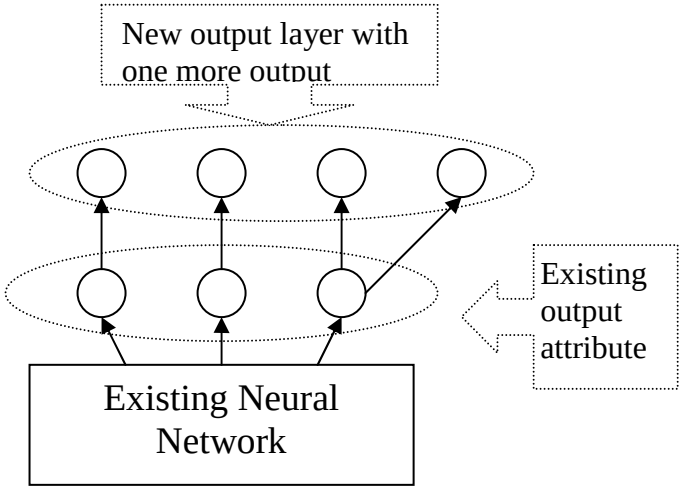


Figure 2.5 Illustration of Reclassification

The reclassification problem can also be decomposed into the sub-problems of discarding invalid learnt knowledge, acquiring new knowledge and retaining valid learnt knowledge. The only difference between reclassification and a completely new class problem is that there exists clear conflict between the existing outputs and the new output. Both IOL-2 and IOL-3 can fully solve the sub-problems in reclassification and handle the conflict between the old and new outputs. Therefore, IOL-2 and IOL-3 can be used in reclassification.

2.5 Summary of the Chapter

In this chapter, I proposed three incremental output learning methods based on modular neural networks. These methods allow a neural network to learn incrementally with incoming output attributes. They use a "divide and conquer" way to decompose learning in the changing environment into several sub-problems. When a new output attribute is to be learnt, a new module is combined with the existing neural network to solve the sub-problems. The experiment result shows that the proposed methods can get similar or better results compared to traditional retraining in terms of accuracy. The learnt knowledge that is still valid in the changed environment is retained in the learning process of new knowledge.

The proposed methods show some advantages over retraining. Firstly, they provide continuous work in the adaptation process and smooth handover between the existing neural network and the upgraded neural network. Secondly, they need less adaptation time in most cases. IOL-3 can reduce adaptation time up to 72.7% in the experiments.

Thirdly, the existing network can be reinstalled at any time after adaptation, since the existing network is kept unchanged as the old sub-network in the methods.

Chapter 3

Task Decomposition with Hierarchical Structure

3.1 Background

Multiple layer perceptron (MLP) neural network suffers from several drawbacks [34] when applied to complex behavioral problems. [35] and [36] stated that learning a complex behavior requires bringing together several different kinds of knowledge and processing, which is impossible to be achieved for global NN like MLP. For the "stability-plasticity dilemma" problem, [37] argued that when two tasks have to be learnt consecutively by a single network, the learning of the second task will interfere with the previous learning. Another common problem for multiple task NN is the "temporal crosstalk" problem [38], which means that a network tends to introduce high internal interference because of the strong coupling among their hidden-layer weights when several tasks have to be learnt simultaneously.

A widely used approach to overcome these shortcomings is to decompose the original problem into sub-problems (modules) and perform local and encapsulated computation for each sub-problem. There are various task decomposition methods that have been

proposed in the literature [14]-[21] [39]-[42]. These decomposition methods can be based in the characteristics on input data space and/or output space.

One category of decomposition methods based on the characteristics of input data space is *Domain Decomposition*. [11] suggested that the original input data space can be partitioned into several sub-spaces and each module (for each sub-problem) is learnt to fit the local data in each sub-space to improve the effectiveness of training. There are many such methods proposed in the literature. In [39], the training set is divided into subsets recursively using hyper planes till all the subsets become linearly separable. [40] described that neural networks where the first unit introduced on each hidden layer can be trained on all patterns and further units on the layer are trained primarily on patterns not already correctly classified. [14] suggested that in the mixture of experts architecture, expert networks can be used to learn sub-spaces and then cooperate via a gating network. For example, in the hierarchical mixture of expert architecture, the input space is partitioned recursively into sub-spaces [15]. Similar recursive partition is also used in neural trees structure [16]. Another decomposition method of this category is proposed in the multi-sieving neural network [17]. In this method, patterns are classified by a rough sieve in the beginning and they are re-classified further by finer ones in subsequent stages.

Another category of decomposition methods based on the characteristics of output space is *Class Decomposition*. [18] split a K-class problem into K two-class sub-problems. One sub-network is trained to learn one sub-problem only. Therefore, each sub-network is used to discriminate one class of patterns from patterns belonging to the remaining classes, and there are K modules in the overall structure. The method

proposed in [19] divided a K-class problem into $\binom{K}{2}$ two-class sub-problems. Each of the two-class sub-problems is learnt independently while the existence of the training data belonging to the other $K-2$ classes is ignored. The final overall solution is obtained by integrating all of the trained modules into a min-max modular network. A powerful extension to the above class decomposition method, *output parallelism*, is proposed in [42]. Using output parallelism, a complex problem can be divided into several sub-problems as chosen, each of which is composed of the whole input vector and a fraction of the output vector. Each module (for one sub-problem) is responsible for producing a fraction of the output vector of the original problem. These modules can be grown and trained in parallel.

Besides these two categories, there are some other decomposition methods. In [43], different functional aspects in a task are modeled independently and the complete system functionality is obtained by the combination of these individual functional models. In [44], the original problem is decomposed into sub-problems based on different states in which the system can be in at any time.

Class decomposition methods reduce the internal interference among hidden layers, consequently, improve performance and accuracy. However, there is a shortcoming of this approach. In these methods, each sub-network is trained independently from all the other sub-networks. The correlation between classes or sub-networks is ignored. A sub-network can only use the local information restricted to the classes involved in it. The sub-networks cannot exchange with other sub-networks information already learnt

by them. The global information between classes that can be positive to the learning of sub-networks is missing as well as internal interference between them.

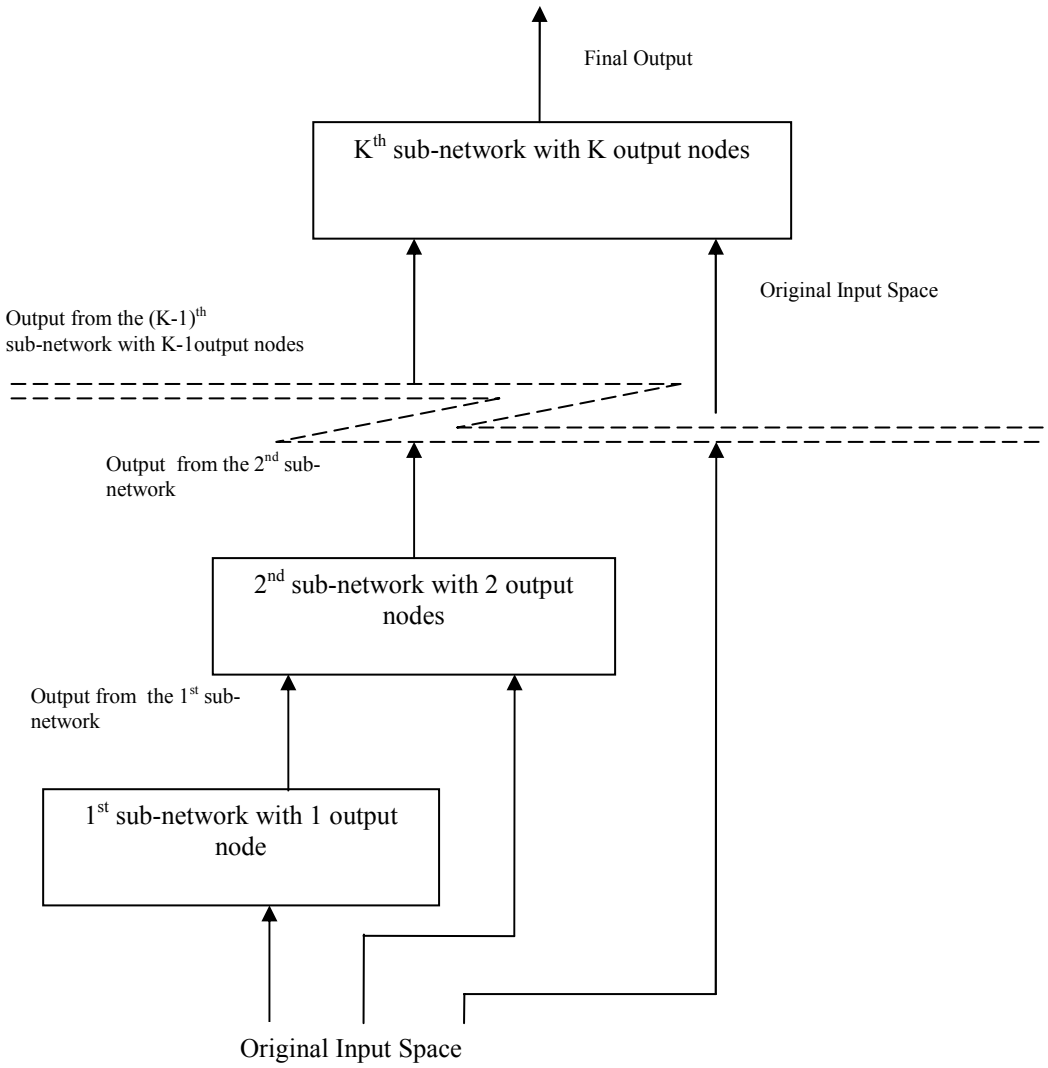


Figure 3.1 Overview of Hierarchical MNN with Incremental Output

In this chapter, I propose a new task decomposition approach namely *hierarchical incremental class learning* (HICL). In this approach, a K-class problem is divided into K sub-problems. The sub-problems are learnt sequentially in a hierarchical

structure with K sub-networks. Each sub-network takes the output from the sub-network immediately below it as well as the original input as its input. The output from each sub-network contains one more class than the sub-network immediately below it, and this output is fed into the sub-network above it as Fig 3.1. The overall structure of HICL is an extension of IOL-3 discussed in section 2.2.3. This method not only reduces harmful interference among hidden layers, but also facilitates information transfer between classes during training as described in section 2.4.1. It shows more accurate classification performance than traditional class decomposition methods.

The chapter is organized as follows. In section 3.2, the structure of HICL is introduced. In section 3.3, the ordering problem of HICL is discussed and two ordering methods are proposed. Section 3.4 discusses the experimental results of HICL. Section 3.5 summarizes the work.

3.2 Hierarchical MNN with Incremental Output

In the proposed method, the original K-class problem is solved using a hierarchical modular neural network (HMNN) consisting of K sub-networks. After a sub-network is constructed and trained, a new sub-network is constructed on top of it. The new sub-network accepts the output from the old sub-network, together with the original input as its input. The output space of the new sub-network is one dimension larger than that of the old sub-network. For classification problems, this means the output space of the new sub-network includes one more class than the old sub-network.

The proposed HICL decomposition method is composed of the following steps.

Step 1: Determine the order of the classes (output attributes) to be inserted into the hierarchical MNN structure. The output attributes are then sorted into a list based on this order. This stage is essentially important to achieve high accuracy, which will be discussed in detail in section 3.3. Set the trained sub-network index counter to *index*=1.

Step 2: Construct a sub-network with only one output node. The input data space is the same as the original problem before decomposition. The output space contains only the first output node in the sorted list generated in Step 1. Train the network till convergence. Increment *index* by1.

Step 3: If *index* is not equal to the number of output attributes in the original output space, construct a new sub-network on top of the structure that has been constructed.

The input space for the newly constructed network is formed by merging the output space of the sub-networks below it with the original input space. When an input training sample is presented to the structure, the output attributes from the structure below the new sub-network together with the original input attributes form the input for the new sub-network. Hence, to the new sub-network, there are $index + n$ input attributes, where n is the number of input attributes in the original input data space. The output space of the new sub-

network contains all the output attributes (classes) that were trained in the sub-networks below it, together with the $index^{th}$ output attribute in the sorted list generated in Step 1. Hence, there are $index+1$ output attributes (classes) for the new sub-network.

The new sub-network is trained until it converges. Increment the trained sub-network index counter, $index = index+1$.

This step is repeated until *index* is equal to the number of output attributes in the original output space.

Step 4: Test the overall structure and evaluate the performance.

The functionality of the first sub-network is to classify the training samples belonging to the first output attribute (class) in the list generated in Step 1. This is a localized computation associated with the output attribute representing the specified class only, which is the same as one single module in class decomposition. Because internal interference is removed, the output from this sub-network tends to be more accurate.

The functionalities of sub-networks other than the first one are more complex. Because each sub-network needs to deal with more than two classes simultaneously, the correlation between different classes is taken into consideration automatically. There are two functions for each sub-network.

- The minor function is to perform reclassification to the classes learnt previously. If the lower sub-networks produce no error, they provide the present sub-network linear-separable inputs. Due to the strong bias of these inputs, the reclassification process is most likely to follow the decision boundaries delineated by the lower sub-networks and simply repeats the results of it.

- The major function is to classify samples belonging to the newly added class from all the other class. This function is a local computation relative to the new class in the sub-network, which is the same as a sub-network in class decomposition. However, it should be noted that some of the classes are already classified in the lower sub-networks from the newly added class. Again, if this pre-classified information contains no error, it takes no effort to classify the new class from the classes learnt in lower sub-networks.

In HICL, learning processes of different classes are decomposed logically, instead of being decomposed physically in most task decomposition methods. Figure 3.2 illustrates the leaning of a three-class problem with HICL in an ideal situation. A, B and C stand for the three classes in the problem. In this condition, the real task of the 2^{nd} sub-network is classifying class B from class C logically, because class B and C have already been classified from class A. The 2^{nd} sub-network deals with only 2 classes in fact.

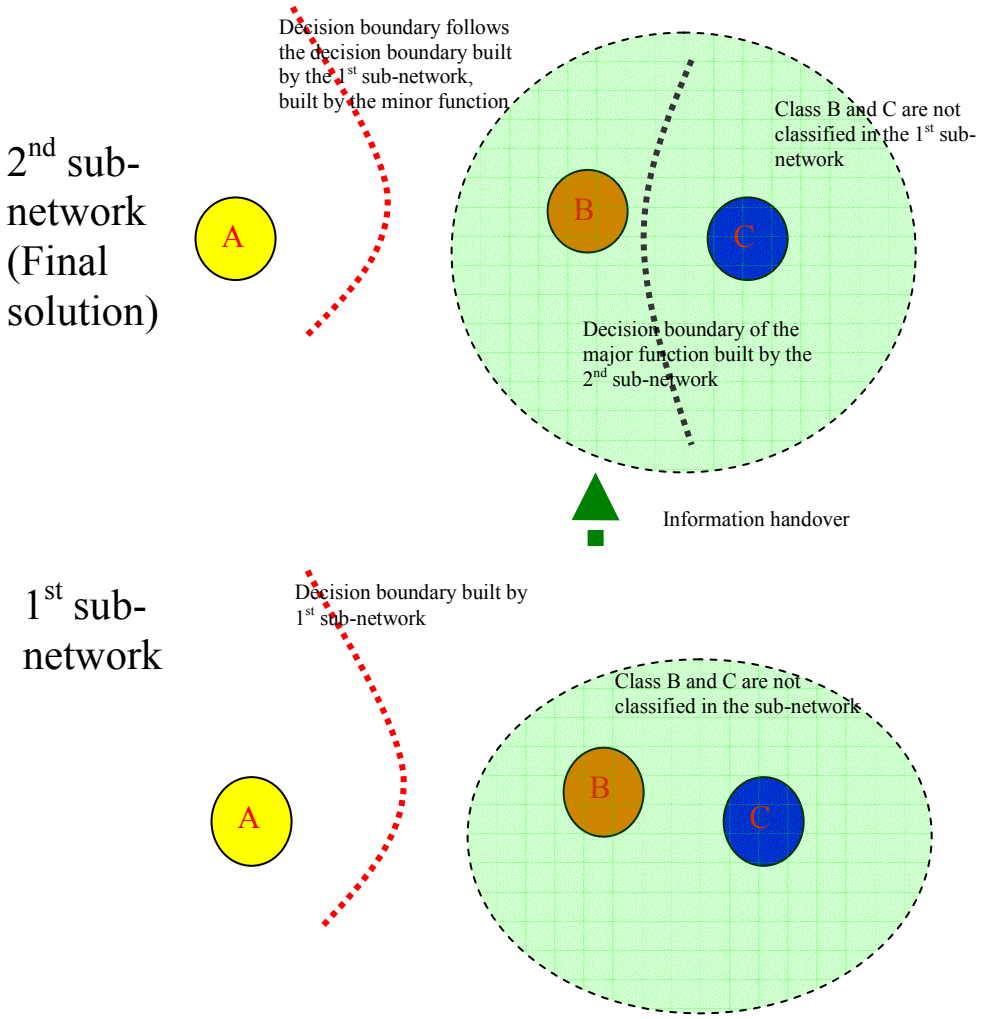


Figure 3.2 A three classes problem solved with HICL

Figure 3 shows how the three class problem is solved in class decomposition, which contains three sub-networks in total. From figure 3.2 and figure 3.3, HICL has two advantages over class decomposition. Firstly, problem being solved by the 2nd network with HICL is simpler than the one in class decomposition since it deals with 2 classes only. This advantage becomes greater when there are more sub-networks. For example, the k^{th} sub-network with HICL deals with $k-1$ fewer classes than class decomposition. Because the problem being solved is simpler, the k^{th} sub-network with HICL tends to be more accurate than class decomposition. Secondly, HICL requires one less sub-

network than class decomposition, which simplifies overall structure and improves accuracy.

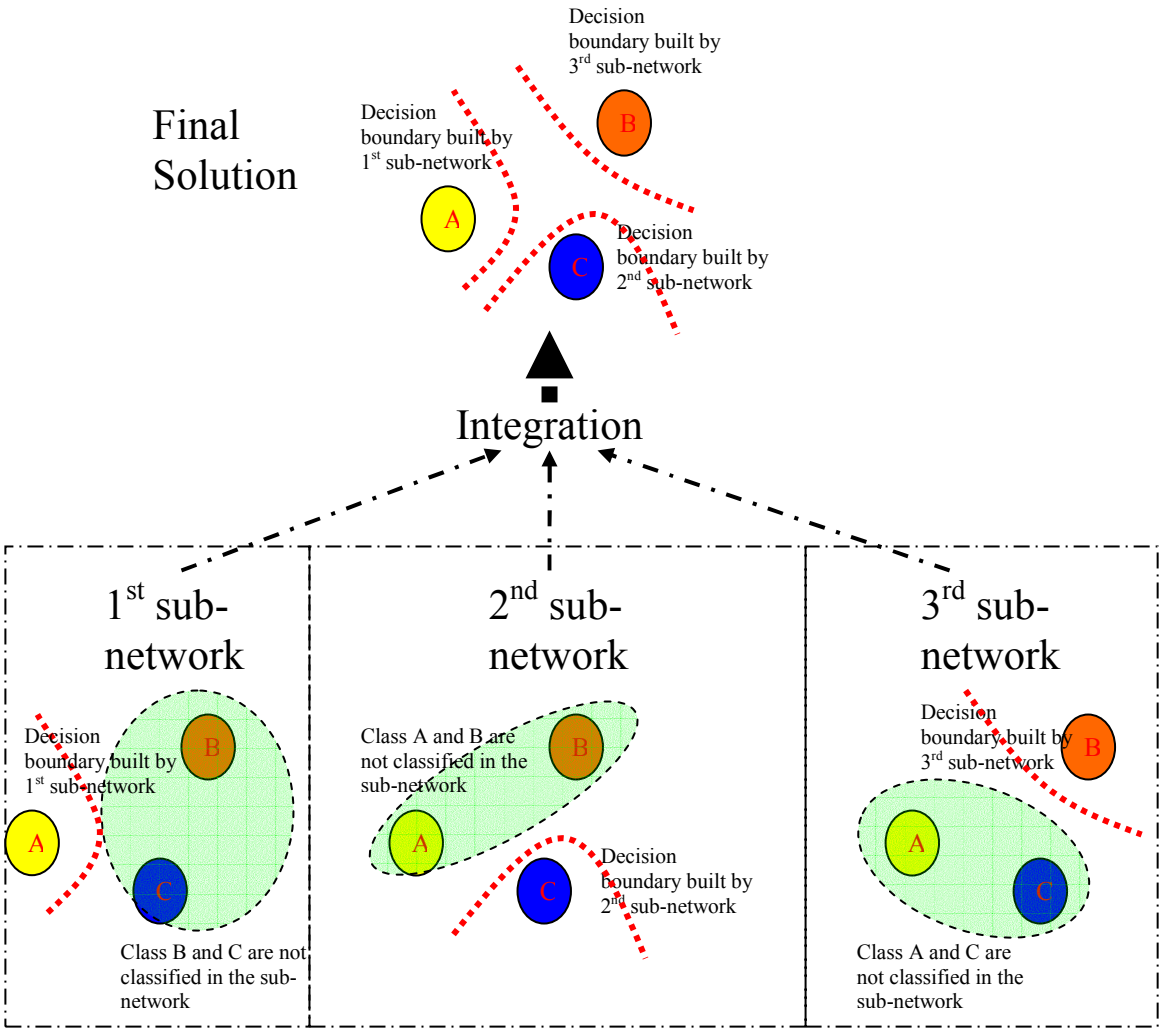


Figure 3.3 A three classes problem solved with class decomposition

3.3 Determining Insertion Order for the Output Attributes

In HICL, the output attributes are inserted into a network following some predetermined order, which is a key factor to improve the overall accuracy of the network. In this section, two ordering methods that lead to good accuracy will be introduced.

3.3.1 MSEF-CDE Ordering

In Section 2, we found that HICL have great advantages over class decomposition if there is no error in lower sub-networks. However, the errors can hardly be avoided in practice. These errors may mislead learning of upper sub-networks and downgrade the advantages of HICL. Due to the hierarchical structure used, the earlier a class is trained, the more its associated sub-network will affect the overall performance. In this section, the Minimal-Side-Effect-First (MSEF) ordering method based on Class Decomposition Error (CDE) is introduced to minimize the negative effect of possible errors, which in turn maximize the advantages of HICL.

3.3.1.1 Simplified Ordering Problem of HICL

The major function of a sub-network in HICL can be viewed as a 2-class problem logically which is similar to a module in class decomposition. The first class ω_1 is the one being extracted. The second class ω_2 is the complement of ω_1 in the entire output space. The overall error E of the major function in a sub-network can also be decomposed as:

$$E = e + \bar{e} \tag{3.1}$$

where e *is the error produced by the training samples belonging to* ω_1, $\bar{e}$ *is the error produced by the samples belonging to* ω_2.

The error of a module for the class in class decomposition is a good approximation for the value of correspondence E in equation (3.1). In this paper, I use a stepwise optimal approach to solve this problem., which simplifies the ordering problem to a 2-step problem:

(1) Find the error of each module in class decomposition and use it as an approximation for the correspondence major function in HICL. Find the portion of each error that may bring negative effect to proper learning of upper sub-networks.

(2) Order the classes based on this portion of error belonging to each major function, from the smallest to the greatest.

Based on this simplified model, it is necessary to identify which portion of major function error in a sub-network may affect the proper learning of sub-networks upper to it.

In the k[th] sub-network of the neural network solution for an n-class problem, there are $\omega_1 = C_k$ and $\omega_2 = C_1 + C_2 + \cdots + C_{k-1} + C_{k+1} + \cdots + C_n$, where C_i stands for the i^{th} class in the original output space.

There are two possible types of error events for the major function of this sub-network:

1. A sample belonging to ω_2 is misclassified into ω_1, which is an event of $\bar{e}$.

   In this case, the present sub-network indicates to the upper sub-networks that the sample belongs to C_k. If the misclassified sample belongs to C_i (where $i > k$) in the original data space, there will be a clear conflict between the information passed from the k^{th} sub-network and the information contained in the original input space when it is being extracted in the i^{th} sub-network. This conflict may cause interference to the proper learning of the major function of i^{th} sub-network, so that it may misclassify some more samples belonging to the i^{th} class. Hence, $\bar{e}$ is the portion of error that needs to be considered in deciding the order of sub-networks.

2. A sample belonging to ω_1 is misclassified into ω_2, which is an event of e.

   In this case, the present sub-network indicates to the upper sub-networks that the sample does not belong to C_k. In the i^{th} sub-network, the information passed from the k^{th} sub-network indicates that the sample does not belong to the k^{th} class, which is independent with whether the sample belongs to the i^{th} class or not. Hence, the major functions of the following sub-networks will not be disturbed by the error event. The ordering can be made independent of this type of error.

From the above analysis, the ordering is dependent on the accumulated error of samples belonging to the complementary class ω_2, which is $\bar{e}$. Step 1 in the proposed solution is further simplified as: finding $\bar{e}$ of the major function in each sub-network.

3.3.1.2 Calculating the Order

A 2-class problem is normally solved by a neural network with a single output attribute. Theoretically, if a neural network is perfectly trained and produces no error in the entire output space, it outputs 1 when the input sample belongs to ω_1 and 0 when the input sample belongs to ω_2. The decision boundary that differentiates ω_1 from ω_2 is simply a threshold of 0.5. Figure 4 illustrates the distribution of the desired output for a 2-class problem.

However, in practical applications a neural network can hardly be trained perfectly due to interference, existence of local minima, overfitting, and distribution of samples in the data space and so on. Hence, errors will occur in the output of samples. In general, the samples have almost equal probability to be interfered. The errors introduced by the interference are most probably to be Gaussian distributed with a mean of 0 and variance of σ^2. If any error is larger than 0.5, the specified sample will be misclassified. Hence, the probability of misclassification, which is represented by P_{error}, is identical for all the samples in the data space. Figure 3.5 illustrates the real output of a neural network for the same problem in figure 3.4.

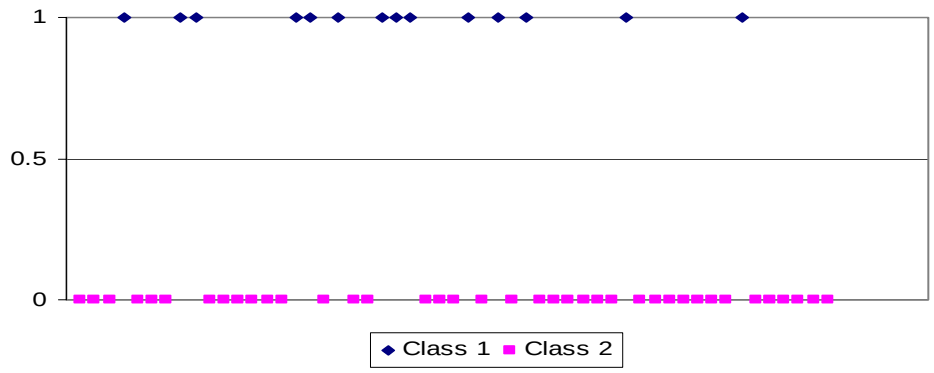


Figure 3.4 Desired Output for a 2-Class Problem

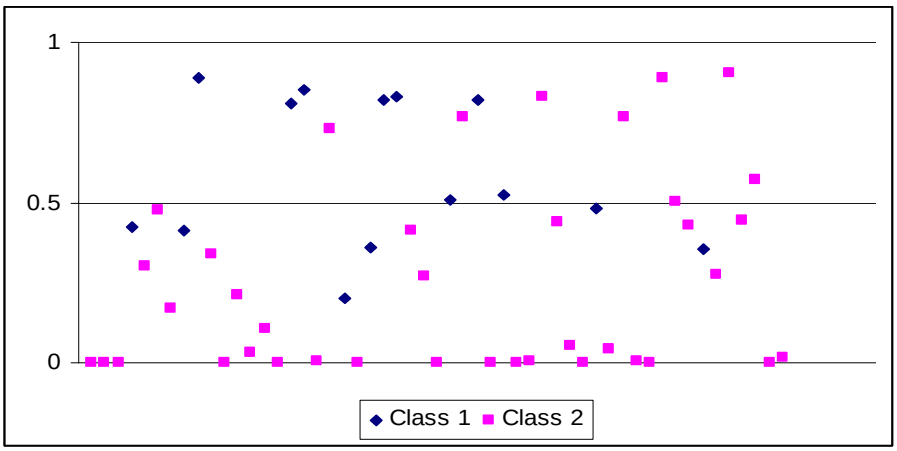


Figure 3.5 Real Output for a 2-Class Problem

Assume P_{error} is identical for all samples, it can be derived that:

$$\frac{\bar{e}}{e} = \frac{P_{error} \times N_{com}}{P_{error} \times N_{spe}} = \frac{N_{com}}{N_{spe}} \tag{3.2}$$

where N_{spe} *is the number of samples belonging to* ω_1 *and* N_{com} *is the number of samples belonging to* ω_2.

From equation (3.1) & (3.2), the portion of error that may affect the proper learning of the other classes can be calculated as:

$$\bar{e} = E\frac{N_{com}}{N_{spe}+N_{com}} = E\frac{N_{com}}{N} \tag{3.3}$$

Hence, this portion of error caused by the k^{th} class of a *N-class* problem is

$$\overline{e_k} \approx E_k \frac{N-N_k}{N} \tag{3.4}$$

where N is the number of samples in the entire data space, N_k is the number of samples belonging to the k^{th} class in the original data set and E_k is the error of the network used to extract C_k from the other classes in class decomposition.

Based on the simplified problem descript in section 3.3.1.1, the MSEF-CDE ordering procedure can be summarized as:

1. Train a network based on class decomposition, record the error for each class as $E_1, E_2, \cdots, E_{n-1}, E_n$
2. Calculate the portion of error that may affect the proper learning of the other classes $\overline{e_1}, \overline{e_2}, \cdots, \overline{e_{n-1}}, \overline{e_n}$ for each class using equation (3.4)
3. Sort the classes in order by the value of this portion of error for each class, from the smallest to the largest and store them in a list, as described in Step 1, Section 2.

The proposed MSEF-CDE ordering method estimates the order that minimizes the overall interference in stepwise. From the experiment results, this ordering method is shown to be effective and improves the accuracy of HICL significantly. However,

finding the error for each class using class decomposition requires computation. As a pre-processing step of HICL, the computation may be unaffordable. In the next section, another ordering method that requires much less computation is proposed.

3.3.2 MSEF-FLD Ordering

Linear pattern recognition techniques, such as Fisher's Linear Discrimenent (FLD) [45], provide simples ways to estimate the accuracy of a classification problem. In this section, the method of Minimal Side-Effect Ordering (MSEF) based on Fisher's Linear Discriminant (FLD) is proposed. The idea behind this method is similar to the MSEF-CDE method proposed previously, which is to order the sub-networks (classes) based on the portion of error caused by learning the specified class alone that may affect the proper learning of other classes. Hence, the problem is to find $\bar{e}$ for each class and perform ordering based on it, which is the same as the one given in section 3.3.1.1. Instead of using the classification error E of each class obtained by class decomposition, the MSEF-FLD method uses Fisher's criteria function $J(w)$ as a goodness score for each value of E.

FLD projects a *d-dimensional* feature space into a *c-1* dimensional feature space, where *d* is the number of features and *c* is the number of classes, by the transformation function $y_i = w^t x_i$. Hence, for a 2-class problem, the projected feature space will be *one*-dimensional (projected on one line).

Let a set of m training patterns be $X = [x_1, x_2, \cdots x_i, \cdots x_m]^t$, where $x_i \epsilon R^n$, *i=1,2...m*. These patterns belong to two classes ω_1 and ω_2. Mathematically, FLD can be described as follows:

Let m_i *(i=1 or 2)* be the d-dimensional sample means of ω_1 and ω_2 as given by $m_1 = \frac{1}{n_1}\sum_{x \in X_1} x$ and $m_2 = \frac{1}{n_2}\sum_{x \in X_2} x$ respectively, where X_1 and X_2 represent the set of samples belonging to classes ω_1 and ω_2 respectively, n_1 and n_2 represent the numbers of samples in X_1 and X_2 respectively. The sample means for the projected points is given by $\tilde{m}_i = \frac{1}{n_i}\sum_{y \in Y_i} y = w^t x = w^t m_i$, where *i=1,2* are the symbols of the two classes respectively, and Y_1, Y_2 are samples belonging to class 1 and class 2 in the projected space respectively. It is simply the projection of m_i. If we define the scatter for the projected samples of class *i* as $\tilde{s}_i = \sum_{y \in Y_i} (y - \tilde{m}_i)^2, \quad i = 1,2$, the within-class scatter of $S_w = \sum_{i=1}^{2} S_i$ can be calculated. This within-class scatter is a measure of how close the patterns in the same class are distributed. Similarly, the between-class scatter can be calculated as $S_B = \sum_{i=1}^{2} n_i (m_i - m)(m_i - m)^t$, where $m = \frac{1}{n}\sum_{x \in X} x$ is the mean of all patterns in the feature space. Fisher's linear discriminant employs that linear function $w^t x$ for which the Fisher's criterion function

$$J(w) = \frac{w^t S_B w}{w^t S_W w} \tag{3.5}$$

is maximized and independent of ||w||. The optimal projection can be computed by solving the eigenvector problem: $(S_B - \lambda_i S_W) w_i = 0$, where λ_i 's are the non-zero

eigenvalues and w_i's are the corresponding eigenvector. The larger the value of $J(w)$, the easier the classification. Hence, the accuracy of classification is increasing with $J(w)$, and the error is decreasing with $\frac{1}{J(w)}$. From equation (3.4) and (3.5), the portion of error $\overline{e_k}$ caused by extracting the k^{th} class can then be expressed in the following form:

$$\overline{e_k} = \frac{N - N_k}{J_k(w)N} \qquad (3.6)$$

The MSEF-FLD procedure is summarized as follows:

1. Calculate the value of Fisher's criteria function for each class and its complementary class in the data space as $J_1(w), J_2(w), \cdots, J_{N-1}(w), J_N(w)$.

2. Calculate the portion of errors that may affect the proper learning of the other classes $\overline{e_1}, \overline{e_2}, \cdots, \overline{e_{n-1}}, \overline{e_n}$ for each class using equation (3.6).

3. Sort the classes in order by the value of this portion of error for each class, from the smallest to the largest and store them in a list, as described in Step 1, Section 3.2.

3.4 Experiments and Analysis

3.4.1 Experiment Scheme

In order to optimize the performance for each module, constructive neural network is used in the experiments. The constructive learning algorithms include the *Dynamic Node Creation* (DNC) method [46], *Cascade-Correlation* (CC) [47] algorithm and its variations [48]-[50], *Constructive single-hidden-layer network* [51], and *Constructive Backpropagation* (CBP) algorithm [52], etc. I adopt the CBP algorithm. Please refer to [42] for details of the CBP algorithm and parameter settings.

The RPROP algorithm is used to minimize the cost functions. In the set of experiments undertaken, each problem was conducted with 20 runs. The RPROP algorithm used the following parameters: $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_0 = 0.1$, $\Delta_{\max} = 50$, $\Delta_{\min} = 1.0e-6$, with initial weights from –0.25 … 0.25 randomly. In the experiments, the hidden units and output units all use the sigmoid activation function. When a hidden unit needs to be added, 8 candidates are trained and the best one is selected. All the experiments are conducted 10 times and the results are averaged.

The test error measure E_{test} used in this chapter and chapter 4 is *the squared error percentage* [61], derived from the normalization of the mean squared error to reduce the dependency on the number of coefficients in the problem representation and on the range of output values used:

$$E_{test} = 100 \cdot \frac{o_{\max} - o_{\min}}{K \cdot P} \sum_{p=1}^{P} \sum_{k=1}^{K} (o_{pk} - t_{pk})^2 \qquad (3.7)$$

where $o_{\max}$ and $o_{\min}$ are the maximum and minimum values of output in the problem data. P and K are total number of test patterns and number of outputs. o_{pk} and t_{pk} are the desired (the value in original test data) and real output from neural network of the k^{th} output in p^{th} pattern in test data.

3.4.2 Segmentation Problem

The data set of segmentation problem consists of 18 inputs, 7 outputs, and 2310 patterns. It is more complex compared to the Thyroid and Glass problems. The experimental results of ordering obtained by the MSEF-CDE and MSEF-FLD methods, random ordering, retraining, and class decomposition are listed in Table 3.1 below.

Table 3.1 Results of HICL and Other Algorithms with Segmentation Problem

Method	Ordering	Training Time	Test Error	Classification Error
HICL (MSEF-CDE)	7261345	Value: 5357.8	Value: 0.852246	Value: 3.604851
		Reduction (retraining): -452%	Reduction (retraining): 33.8%	Reduction (retraining): 38.8%
		Reduction (class decomposition): -369%	Reduction (class decomposition): 29%	Reduction (class decomposition): 32%
HICL (MSEF-FLD)	7261354	Value: 5979	Value: 0. 836863	Value: 3.450446
		Reduction (retraining): -516%	Reduction (retraining): 35%	Reduction (retraining): 41.4%
		Reduction (class decomposition): -423%	Reduction (class decomposition): 30.2%	Reduction (class decomposition): 34.9%
HICL (Random)	7613542	4194.2	1.020443	3.89948
HICL (Random)	1234567	2620.8	1.020746	4.19411
HICL (Random)	4531627	1507.7	1.236225	4.800696
HICL (Random)	1436972	2560.6	1.20976	4.67938
HICL (Random)	2164357	5792.8	1.12248	4.33276
HICL (Random)	2761354	4012	0.922908	3.81282
HICL (Random)	4136572	2172.2	0.869713	4.33276
HICL (Random)	5346127	1688.6	1.22915	4.85269

HICL (Random)	5436712	1611.5	1.2824	4.8007
HICL (Random)	6275434	3174	0.930261	4.15945
Retraining		970	1.2869	5.89255
Class Decomp		1143	1.2	5.3

Notes:

1. *Each row shows the experiment result obtained from the ordering stated in the cell of "ordering" column. This column shows the insertion orders of the experiments. The digit represents the index of the class (output attribute) to be added and the order of the digits represents the sequence of inserting the class (output attributes). For example, 1234567 means the first class in the original data is inserted into the HICL structure, followed by the second class. The seventh output is the last one to be inserted into the HICL structure.*
2. *The row starting with " Retraining" shows the result of training the problem using standard CBP without task decomposition.*
3. *The row starting with " Class Decomp" shows the result of training the problem using class decomposition [15].*
4. *A cell stating " Reduction(retraining)" shows the percentage value reduction of the specified method compared to the value in retraining. It is calculated by function* $(currentVlaue - retrainingValue) \div retrainingVlaue \times 100\%$ *. Negative percentage indicates value increase instead of reduction.*
5. *A cell stating " Reduction(class decomposition)" shows the percentage value reduction compared to the value in class decomposition. It is calculated by function* $(CurrentVlaue - ClassDecompValue) \div ClassDecompVlaue \times 100\%$ *. Negative percentage indicates value increase instead of reduction.*
6. *Because there are 5040 possible orderings, which are hard to be tested completely, only a random selected small portion of them are tested in the experiments.*
7. *The training time for Class Decomposition is calculated based on the module which needs the longest training time.*

In this problem, the linear estimator in the MSEF-FLD method obtained sufficient information from the data set to make accurate estimation of each module's performance. As a result, MSEF-CDE and MSEF-FLD give very close orderings. From the experimental results, we can find that both of the orderings lead to very small classification errors (38.3% and 41.4% error reduction compared to retraining respectively) and generalization errors (33.8% and 35% error reduction compared to class decomposition respectively). It also shows great advantage over class decomposition when accuracy is emphasized. However, as a tradeoff, both orderings need very long training time.

3.4.3 Glass Problem

The experimental results of ordered training obtained by the MSEF-CDE and MSEF-FLD methods, random ordering, retraining, and class decomposition are listed in Table 3.2 below.

Table 3.2 Results of HICL and Other Algorithms with Glass Problem

Method	Ordering	Training Time	Test Error	Classification Error
HICL (MSEF-CDE)	543612	Value: 66.5	Value: 8.936928	Value: 31.69813
		Reduction (retraining): -349%	Reduction (retraining): 12%	Reduction (retraining): 9.7%
		Reduction (class decomposition): -200%	Reduction (class decomposition): -0.1%	Reduction (class decomposition): 19.6%
HICL (MSEF-FLD)	614523	Value: 73.2	Value: 8.60229	Value: 32.45286
		Reduction (retraining): -484.6%	Reduction (retraining): 15.33%	Reduction (retraining): 7.5%
		Reduction (class decomposition): -231%	Reduction (class decomposition): 3.6%	Reduction (class decomposition): 17.7%
HICL (Random)	123456	87.8	8.511716	33.5849
HICL (Random)	132456	101	8.880506	32.45286
HICL (Random)	132456	83	9.497634	34.717
HICL (Random)	325146	90.4	9.066676	35.09434
Retraining		14.8	10.15961	35.09436
Class Decomp		22.1	8.92708	39.434

Notes: 1-5.Refer to notes under table 3.1

6. *Because there are 720 possible orderings, which are hard to be tested completely, only a randomly selected small portion of them are tested in the experiments.*
7. *The training time for Class Decomposition is calculated based on the module which needs the longest training time.*

Glass problem is a special case in the data sets used in the experiments. Because it contains a very small number of patterns, which is 214 in total, there is insufficient information for linear analysis techniques like FLD to predict the performance of each sub-network in HICL. In this problem, MSEF-FLD method fails to predict the ordering obtained by MSEF-CDE. The two methods give very different orderings.

The ordering obtained with the MSEF-CDE method shows much smaller test error and generalization error compared to retraining or class decomposition, but longer training time. It also leads to the most accurate result in the different orderings that have been tested. The result using the ordered training obtained from the MSEF-FLD method is not as accurate as what obtained with MSEF-CDE. However, the errors are still much less than the errors in retraining and class decomposition.

3.4.4 Thyroid Problem

The orders obtained with MSEF-CDE and MSEF-FLD are both 3? 1? 2, which stands for learning the sub-network associated with the third class first in HICL, followed by the sub-network associated with the first class, and then followed by the sub-network associated with the second class. The experimental results of ordered training obtained by the MSEF-CDE and MSEF-FLD methods, random ordering, retraining, and class decomposition are listed in Table 3.3 below.

Table 3.3 Results of HICL and Other Algorithms with Thyroid Problem

Method	Ordering	Training Time	Test Error	Classification Error
HICL (MSEF-CDE)	312	Value: 840.6	Value: 0.94121	Value: 1.666668
		Reduction (retraining): 29.9%	Reduction (retraining): 11.5%	Reduction (retraining): 13.3%
		Reduction (class decomposition):49.2%	Reduction (class decomposition): 9.4%	Reduction (class decomposition):9.43%
HICL (MSEF-FLD)	312	Value: 840.6	Value: 0.94121	Value: 1.666668
		Reduction (retraining): 29.9%	Reduction (retraining): 11.5%	Reduction (retraining): 13.3%
		Reduction (class decomposition):49.2%	Reduction (class decomposition): 9.4%	Reduction (class decomposition):9.43%
HICL	123	1509	1.203448	2.144446

(random)				
HICL (random)	132	605.2	1.102526	2.033336
HICL (random)	213	1672.6	0.984035	1.755556
HICL (random)	231	1500.6	1.093764	1.944444
HICL (random)	321	1353	0.89799	1.544444
Retraining		1198.4	1.063898	1.92222
Class Decomposition		1656.2	1.038454	1.84015

Notes: 1-5.Refer to notes under table 3.1

From the experimental results, we can find the HICL approach with ordering obtained from the MSEF-CDE and MSEF-FLD methods gives much smaller classification error and generalization error (test error) compared to retraining the problem and class decomposition. It also requires much less computation time. Because MSEF-CDE and MSEF-FLD are developed with a simplified model of HICL structure, both of them did not give the ordering that leads to the exactly minimal error. In this problem the ordering of 321 gives slightly less classification error and generalization error, but much longer training time. However, the ordering given by MSEF-CDE and MSEF-FLD still leads to much less error than the average of all the orderings.

It is clear that HICL with MSEF-CDE and MSEF-FLD ordering methods is more accurate than retraining and class decomposition. However, it usually needs longer training time. Between the two ordering methods, MSEF-CDE is more general. It is suitable for small problems with insufficient information (or lack of samples). However, when the data set is large, MSEF-CDE may become time expensive in its pre-processing. If the data set contains sufficient information, MSEF-FLD can always give very similar, if not better, ordering compared to MSEF-CDE. In some cases like the Segmentation problem, MSEF-FLD gives even better ordering than MSEF-CDE, because it is a deterministic method and does not require the experiment result of each

module as in MSEF-CDE. As a result, MSEF-FLD avoids the possible error in the experiment results required by MSEF-CDE.

3.5 **Summary of the Chapter**

In this chapter, I proposed a new task decomposition approach namely *hierarchical incremental class learning* (HICL) to grow and train neural network in a hierarchal manner. A neural network can be divided into several sub-networks, each sub-network takes the output from the sub-network immediately below it as well as the original input as its input. The output from each sub-network contains one more class than the sub-network immediately below it, and this output is fed into the sub-network above it. In order to reduce the error, two ordering methods, namely MSEF-CDE and MSEF-FLD are further developed based on class decomposition error and linear analysis technique respectively.

The suggested HICL with the MSEF-CDE and MSEF-FLD ordering methods is compared with one of the newest task decomposition techniques, Output Parallelism [42]. The experimental results of Glass problem with different task decomposition methods are shown in table 3.4

Table 3.4 Compare of Experimental Results of Glass Problem

Method	Test Error	Classification Error
HICL-MSEF-CDE	8.936928	31.69813
HICL-MSEF-FLD	8.60229	32.45286
Output Parallelism	9.233	34.906

From the results, it is clear that the HICL method with MSEF-CDE or MSEF-FLD ordering has better accuracy than Output Parallelism. I have compared the results of some other problems and HICL is more accurate than Output Parallelism in most of the cases.

In some task decomposition techniques such as Class Decomposition and Output Parallelism, the outputs of different sub-networks are assumed to be independent and isolated from each other. There is no information flow between the output attributes. However, this is not true in some real world applications. The proposed method not only reduces harmful interference among hidden layers, but also facilities information transfer between classes during training. The later sub-networks can obtain information learnt from the earlier sub-networks. With the hierarchical relationship (ordering) obtained from the MSEF-CDE and MSEF-FLD, the HICL approach shows smaller regression error and classification error than the class decomposition and retraining methods.

Chapter 4

Feature Selection for Modular Neural Network Classifiers

4.1 Background

As what is discussed in section 3.1, neural networks suffers from the interference between outputs when it is applied to large-scale problems [14] [29]. In order to overcome this shortcoming, many task decomposition techniques are developed. Among these techniques, Class Decomposition [42] is the one most widely used. It splits a K-class problem into K two-class sub-problems and each module is trained to learn a two-class sub-problem [19]. Therefore, each module is a feedforward network which is used to discriminate one class of patterns from patterns belonging to the remaining classes. Each module solves a subset of the original problem. Hence, the optimal input feature space that contains features useful in classification for each module is also likely to be a subset of the original one. From section 3.2, we can find that the HICL also decomposes the original problem into two-class sub-problem. Hence, it has the same problem as Class Decomposition[1]. For the purpose of improving classification accuracy and reducing computation effort, it is important to remove the input features that are not relevant to each module. A natural approach is

[1] In this chapter, the discussion and experiments are based on Class Decomposition instead of HICL, because it is more widely used in practice.

to use a feature selection technique to find the optimal subset for each module. There are several feature selection techniques developed from the following perspectives [22]-[25] [53]-[60].

- Neural network performance perspective

The importance of a feature is determined based on whether it helps to improve the performance of neural network. Setiono and Lui [22] proposed a feature selection technique based on the neural network performance. In this technique, the features of the original feature space are excluded one by one and the neural network is retrained repeatedly. If the overall performance of the neural network is improved when a feature is excluded, the feature is removable from the input feature space. Techniques from this perspective have many attractive attributes but they basically require a large amount of processing on retraining neural networks. Besides, the performance of neural network classifiers depends on many parameters, for example, the initial link weights and neural network structure, etc. In order to obtain a reliable result for each combination of features, a neural network should be retrained several times with different initial link weights and the results averaged. This clearly makes the computation workload less acceptable. In order to overcome this shortcoming, faster learning algorithms and better search algorithms, such as RPROP and genetic algorithm, are used. However, it nevertheless requires considerable computation effort.

- Mutual information (entropy) perspective

Shannon's information theory provides a measure to the mutual information among input features and input and output features. The ideal greedy feature selection technique was developed based on the joint entropy between input and output features.

It can detect the features that are irrelevant to classification, but faces problems dealing with the features carrying redundant information. In order to overcome this shortcoming, Battiti [23] proposed a mutual information feature selector (MIFS) based on the joint entropy between not only inputs and outputs, but also different inputs. Up till now, researchers have developed some modified versions based on this technique, such as MIFS-U [59], to handle redundant features better. However, performance of these techniques can be largely degraded due to the large error in estimating the mutual information using the training data.

- Statistic information perspective

The importance of a feature can be evaluated by goodness-score functions based on the distribution of this feature. Fisher's linear discriminant (FLD) is the most popular goodness-score function. It is simple in computation and does not need strict assumptions in the distribution of features. Generally, all combinations of features in the original feature space can be evaluated with the goodness-score function by excluding some features in the feature space. The combination with a good balance of a large goodness-score and a small number of input features will be considered as the optimal input space for neural networks. Because all possible combinations of the features should be tried, the computation effort of such techniques is very high. In order to reduce computation time, some search algorithms are developed, such as knock-out [24], backtrack tree [25] and genetic algorithm [60].

The shortcomings of the above feature selection techniques can be summarized as: 1) most techniques require huge amount of computation; 2) most of them cannot analyze the correlation among features in a clear manner.

In this chapter, I propose two new feature selection techniques: Relative Importance Factor (RIF) and Relative FLD Weight Analysis (RFWA) based on the optimal transformation weights from Fisher's linear discriminant function. The RIF technique can detect features that are irrelevant to the classification problem and remove them from the feature space to improve the performance of each module in terms of accuracy and network complexity. The RFWA technique can further classify the irrelevant features into noise features and redundant features. In section 4.2, I give a brief introduction to modular neural networks with class decomposition and Fisher's linear discriminant. Then, the RIF and RFWA techniques are depicted in details in section 4.3. The experiments and results of the proposed techniques are analyzed in section 4.4. Section 4.5 summarizes the research on this topic.

4.2 Modular Neural Networks with Class Decomposition

When neural network classifiers are used to solve large scale real world problems, their structures tend to be large to match with the complex decision boundaries of the problems. Large networks tend to introduce high internal interference because of the strong coupling among their hidden-layer weights. Internal interference exists during the training process, whenever updating the weights of hidden units, the influence (desired outputs) from two or more classes cause the weights to compromise to non-optimal values due to the clash in their weight update directions.

In order to avoid such interference, the original network can be decomposed into several modules or sub-problems (Guan, 2002). A common decomposition method used in classification problems is to split a K-class problem into K two-class sub-problems (Figure 4.1) and each module is trained to learn a "yes or no" problem for one class. Each two-class sub-problem is learned independently. Hence, each sub-problem forms a module that is independent from the others. The final overall solution is obtained by integrating all the trained modules' solutions together.

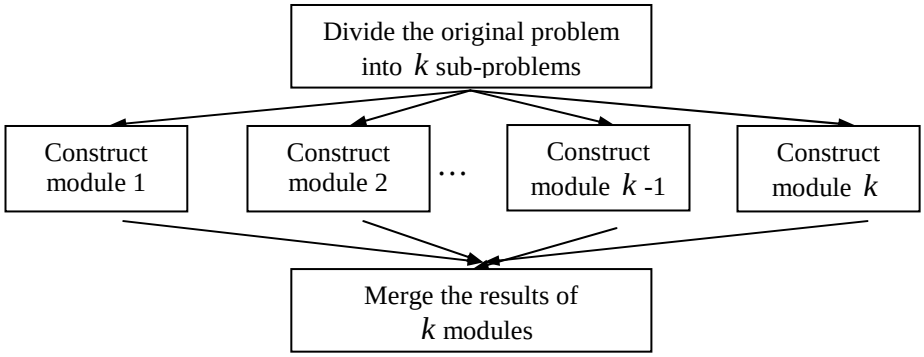


Figure 4.1 Modular Network

In a modular neural network classifier, the occurrences of irrelevant input features are more serious than that in a non-modular neural network classifier. Each module of the modular network is trained independently to solve a "yes or no" problem for one class. Some input features supplied to the original problem may only be useful in classifying certain classes, but irrelevant to the other classes. This suggests that a feature selection process should be applied to each module independently to minimize any undesirable effects. Such a feature selection process can further reduce the internal interference within the modular network to obtain higher classification accuracy.

4.3 RFWA Feature Selector

In this section, two feature selection techniques are presented. The discussion start from the classification of different types of input features. In the next subsection, the design goals of the proposed techniques are given. After that, the RIF feature selection technique based on Fisher's transformation matrix w is proposed. Then, the RFWA feature selection technique based on the RIF is introduced.

4.3.1 Classification of Features

In order to distinguish features that contribute to solve a sub-problem from features that do not contribute or contribute little, the features in the original feature space should be classified into the following two classes.

1. *Relevant Features:* The relevant features of a certain module carry significant useful information for correct classification.

2. *Irrelevant Features:* The irrelevant features of a certain module carry little useful information for correct classification. In another word, irrelevant features make little or no contribution for correct classification. Irrelevant features can be further classified into noise and redundant features.
    - *Noise Features:* Noise features are purely random noise to the module. They do not carry classification information to the module.
    - *Redundant Features:* Redundant features contain classification information overlapping with the other features and their classification information can be fully represented by other relevant features.

The optimal input feature space of a module should contain the relevant features only. If noise features are present in the input feature space, the classifier may end up building unnecessarily complex decision boundaries in these feature dimensions under training. This will make the neural network harder to converge and lose generalization ability. If redundant features are present in the input feature space, they cannot contribute to classification either, because the useful information carried by them can be fully covered by relevant features. The noise carried by redundant features is harmful to the accurate classification of the neural network.

4.3.2 Design Goals

The aim of feature selection is to improve the performance of the classifier. For neural networks, there are three key measurements of the performance, which are generalization error, learning speed and network complexity. In the proposed RIF and RFWA feature selection techniques, a good balance between the three goals is desired in our research.

- *Design Goal 1:* The performance of a neural network classifier should be improved after the feature selection process. The test/classification error and network complexity should be reduced and the leaning speed should be increased significantly.

- *Design Goal 2:* The feature selection technique should be able to detect redundant features as well as noise features.

- *Design Goal 3:* The feature selection technique should not require too much computation.

A common shortcoming of available feature selection techniques based on statistic analysis, such as knock-out techniques, is that they do not consider the correlation among features in a clear manner. Hence, they will face problems when handling highly correlated features. The proposed RFWA technique suggests a clear way of detecting correlated features. The computation workload is another important consideration in the design. The proposed feature selection technique in this chapter is very attractive in computation time.

4.3.3 A Goodness Score Function Based on Fisher's Transformation Vector

Fisher's linear discriminant[2] algorithm projects a d-dimensional feature space to a c-1 dimensional feature space by the function $y_i = w^t x_i$, in the direction w that maximizes the function $J(w) = \frac{w^t S_B w}{w^t S_W w}$, where d is the number of features and c is the number of classes.

For each module in a modular neural network described in Figure 4.1, the projected feature space is one-dimensional (projected on a line). Hence, the transformation matrix w that maximizes the criteria function J(w) is a vector

[2] Refer to section 3.3.2 for reference of Fisher's linear discriminant.

$w = [w_1 \quad w_2 \quad \cdots \quad w_d]^t$. After transformation, an input vector $x_i = [x_{i1} \quad x_{i2} \quad \cdots \quad x_{id}]^t$ will become

$$y_i = w^t x_i = [w_1 \quad w_2 \quad \cdots \quad w_d][x_{i1} \quad x_{i2} \quad \cdots \quad x_{id}]^t = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id} \tag{4.1}$$

This optimal transformation vector w which maximize $J(w)$ can be computed by solving the eigenvector problem: $(S_B - \lambda S_W)w = 0$, where λ is the non-zero eigenvalue and w is the corresponding eigenvector. S_B and S_W can be computed as discussed on page 61 of section 3.3.2.

The elements in the transformation vector w can be viewed as weights for different features in the original feature space respectively. Because w represents the best transformation direction, w_i , $i = 1,2 \cdots d$ shows how much classification information the i^{th} feature in the original feature space carries.

Based on the above analysis and experiment results from several benchmark problems, an observation can be made: *in an optimal transformation vector w of the Fisher's linear discriminant, a larger* w_i *represents that the* i^{th} *feature is more likely to be relevant to the module and a smaller* w_i *represents the* i^{th} *feature is less likely to be relevant to the module.* This observation forms the basis of the proposed RIF and RFWA techniques. In order to show this observation is valid, experiments were conducted using the knock-out technique (Lerner, 1994) with Fisher's function $J(w) = \dfrac{w^t S_B w}{w^t S_W w}$ as a goodness-score function on several benchmark problems.

In the experiments, the features in the original feature space are removed one at a time

and the Fisher's value with respect to all the remaining features is calculated. If the Fisher's value after removing a feature changes little compare to the Fisher's value with respect to all features, the removed feature is likely to be irrelevant. The experiment results confirm with the observation. The experiment results of RIF and RFWA also show that the observation is correct, which will be discussed in the next section.

The proposed goodness score shows some advantages compared with some traditional goodness scores, such as Fisher's function *J(w)*. Firstly, it requires much less computation time. Assume there are d input features in the original feature space. In order to obtain the relative importance of each feature, we need d FLD computations with d-1 features included each time using the traditional knock-out techniques. With the proposed goodness score, the relative importance of each feature in the module can be obtained in one FLD computation with all d features included. Secondly, from the experiment results, it is found that the proposed goodness score can easily handle highly correlated features. Assume there are two duplicated features with one carrying more noise information than the other. In order to remove the one with more noise, the traditional knock-out goodness score requires at least d+2 FLD computations. The proposed goodness score can automatically handle this situation without extra computation. In the experiments, it is observed that if two features in the original feature space carry almost the same classification information, the proposed goodness score will assign high importance to the one with less noise and very low importance to the other one with more noise.

4.3.4 Relative Importance Factor Feature Selection[3] (RIF)

From section 4.3.3, we know that the proposed goodness score can measure the relative importance of a specified set of features. If the weights of some features in the transformation vector w are less than some threshold value T_1, these features can be considered as irrelevant features. Otherwise, they are relevant features of the problem.

However, the weights obtained directly from the transformation vector are not normalized. In another word, the weights obtained from one set of features are not comparable with weights obtained from another set of features. Hence, the value of T_1 may vary from problem to problem.

In order to overcome this problem, I introduce a *Relative Importance Factor (RIF)*, $r = [r_1 \quad r_2 \quad \cdots \quad r_d]^t$, instead of using the transformation vector *w* directly in feature selection. The RIF is obtained from the transformation vector w through the following two steps of normalization.

1. **Normalize the length of the transformation vector w.**

   Since we are looking for the relative importance between features, we are more interested in the relative weights of the features formed from the transformation vector *w*, which can be obtained through normalization:

$$w' = \frac{w}{\sqrt{\sum_{i=1}^{d} (w_i)^2}}, \tag{4.2}$$

[3] In the discussions in section 4.3.4 and 4.3.5, the original feature space is assumed to be d-dimensional.

where w_i is the weight of the i^{th} feature and w' is the normalized transformation vector.

2. Make the importance factor independent from the number of features.

Different problems have different numbers of features in their feature spaces. In order to make the importance values obtained from different problems comparable, it is necessary to make them independent with the number of features in the feature space. This is achieved by the following equation

$$r = \frac{d}{\sum_{i=1}^{d} |w'_i|} w' \tag{4.3}$$

If we combine the first and second steps together, the Relative Importance Factor can be obtained from the transformation vector w directly as:

$$r = \frac{d}{\sum_{i=1}^{d} \left| \frac{w_i}{\sqrt{\sum_{i=1}^{d} (w_i)^2}} \right|} \frac{w}{\sqrt{\sum_{i=1}^{d} (w_i)^2}} = \frac{d}{\sum_{i=1}^{d} |w_i|} w \tag{4.4}$$

The elements of *r* represent the normalized importance of different features, which are independent from the magnitude of *w* and the number of features in the feature space. If the d features carries equal classification information and they are independent from each other, all the elements of *r* will have the value of 1. Hence, the RIF value obtained from different problems are comparable, and a threshold value T_1 may be

found that can be applied to various problems. In the research, I adopt $T_1 = 0.1$ as the threshold value base on the experiment results of several benchmark problems. This threshold value will be used through out the rest of the chapter.

The exact value of this threshold can be varied by the user. In most cases, if a larger threshold value is used, more features can be removed and training time and complexity of the neural network can be further reduced. However, too large a threshold value may cause information loss, so that the classification accuracy can be affected. On the other hand, if the threshold value is too small, there are few features that can be selected as irrelevant. In the problems I have worked on, there is significant feature reduction and no undesirable affect to the classification accuracy when 0.1 is used.

The RIF feature selection technique can be summarized as the following.

1. Calculate the Fisher's transformation vector w with respect to all features in the input feature space.

2. Calculate the Relative Importance Factor for each feature by normalizing the transformation weight of each feature.
    i. If the RIF value of a feature is larger than 0.1, it can be considered as a relevant feature.
    ii. If the RIF value of a feature is less than 0.1, it can be considered as an irrelevant feature and can be removed from the input feature space.

3. Repeat step 1 and 2 for each module in the modular neural network classifier.

Though the RIF feature selection technique can classify irrelevant features from the original input feature space, it cannot tell whether a detected feature is a noise feature or it carries classification information that can be represented by other features. To resolve this, the RFWA feature selection technique is further developed based on RIF. Not only it can distinguish between relevant and irrelevant features, but also able to classify irrelevant features into noise and redundant features.

4.3.5 Relative FLD Weight Analysis (RFWA) Feature Selection

If the classification information carried by one feature in a module can be fully represented by another feature, it is a redundant feature and its RIF value is small based on the analysis in section 3.3. However, when one of the features that carry similar classification information as the redundant feature is removed from the input feature space, the information of the redundant feature becomes more important than before. Hence, its RIF value increases significantly if the remaining features in the input feature space cannot fully represent the information carried by it any longer. On the other hand, if a feature in the original feature space does not carry any classification information, its RIF value will not be affected much with whichever feature being removed. This observation suggests a solution to distinguish between the noise and redundant features.

The proposed RFWA feature selection technique uses the RIF feature selection technique as the first step. In this step, the RIF value of each input feature with respect to all input features is obtained. If the RIF value of a feature is less than the threshold T_1, this feature will be labeled as irrelevant feature, which can be either noise or redundant.

In the next step of RFWA, one relevant feature is removed from the input feature space and the RIF values with respect to the remaining *d-1* features are calculated again. Hence, each irrelevant feature gets one more RIF value, which is called as Cross Relative Importance Factor (CRIF). Repeat this process by restoring the previously removed feature back to the input feature space and removing another relevant feature, till every relevant feature has been removed once and the corresponding CRIF values have been computed.

Until now, *d-N+1* RIF values for each feature have been obtained through the two steps. One RIF value with respect to all input features and *d-N* CRIF values, where N is the number of irrelevant features detected in the previous step. If one of the *d-1* CRIF values of an irrelevant feature increases significantly so that it exceeds a pre-defined threshold value T_2 after some other feature is removed, the feature can be considered as a redundant feature. In the research, I have adopted $T_2 = 0.6$ as the threshold value, based on the experiment results from various benchmark problems. Otherwise, that irrelevant feature can be considered as a noise feature.

In summary, the RFWA technique can be described as the following.

1. Calculate the RIF value of each feature, select those features whose RIF values are less than 0.1 as irrelevant features, and place them in a list for further selection. Initially set counter M=1.

2. If the M^{th} feature is a relevant feature, remove it from the input feature space and calculate the CRIF values with respect to all the remaining features. Restore the M^{th} feature, M=M+1. Repeat 2. If the M^{th} feature is an irrelevant feature, M=M+1 and repeat 2.

3. Perform the following procedure to classify each irrelevant feature in the list:
    - If the CRIF value of a feature in the list exceeds 0.6, the feature is a redundant feature to the module. Remove it from the list.

4. The features remaining in the list are noise features.

4.4 Experiments and Analysis

In this section, the same learning algorithm and parameter settings as described in section 3.4.1 is adopted.

4.4.1 Diabetes Problem

The Diabetes problem diagnoses diabetes of Pima Indians. It has 8 inputs, 2 outputs, and 768 patterns. All inputs are continuous.

Because there are only 2 classes in the problem, it is a "yes or no" problem itself for each class. There is only one module in the modular neural network classifier, which is the original problem.

The RIF and CRIF values for each feature are obtained as in Table 4.1

Table 4.1 RIF and CRIF Values of Each Feature

	RIF	**CRIF 1**	**CRIF 2**	**CRIF 3**	**CRIF 4**
Feature 1	0.8290		0.7966	0.8177	0.7301
Feature 2	2.8040	2.5776		2.7679	2.4643
Feature 3	0.6736	0.6040	0.5234		0.5888
Feature 4	**0.0368**	**0.0363**	**0.4081**	**0.0750**	
Feature 5	0.3216	0.3853	0.7843	0.3419	0.3036
Feature 6	2.1046	1.9403	2.6555	1.9135	1.8664
Feature 7	0.8165	0.7005	0.9677	0.8357	0.7221
Feature 8	0.3726	0.7561	0.8647	0.2472	0.3253
	CRIF 5	**CRIF 6**	**CRIF 7**	**CRIF 8**	
Feature 1	0.7744	0.8930	0.7568	0.9017	
Feature 2	2.4884	3.2881	2.7295	2.5881	
Feature 3	0.6147	0.4058	0.6635	0.5327	
Feature 4	**0.0843**	**0.6162**	**0.1196**	**0.0086**	
Feature 5		0.4685	0.2691	0.3458	
Feature 6	1.9529		2.067	1.8712	
Feature 7	0.7294	0.9659		0.7518	
Feature 8	0.3549	0.3618	0.3941		

Notes: *1. Row of the table stands for the RIF and different CRIF measures of each specified feature with respect to the index number specified in the first column. The row highlighted means that the specified feature is detected as irrelevant.*

2. Column of the table stands for RIF values for all features and CRIF values with different features removed from the input features space, with respect to the index specified in the first row. The number following "CRIF" means the index of feature removed. For example, CRIF 2 means that the CRIF is measured with the second feature removed from the input feature space.

3. The CRIF values with respect to irrelevant features removed are also listed in this table.

From the RIF feature selection technique, feature 4 is detected as irrelevant, because its RIF value is far less than the threshold T_1, which is set as 0.1 in the experiment. We can also find that when feature 6 is removed from the input feature space, the CRIF value of feature 4 rises to 0.6162, which is larger than the threshold T_2 in RFWA. Hence, from RFWA, feature 4 is a redundant feature rather than noise feature and it has redundant relations with feature 6. The experiment results for the original problem and the problem after removing feature 4 are listed in Table 4.2.

Table 4.2 Results of the Diabetes Problem

	Epochs	**Training Time (s)**	**Hidden Units**	**Test Error**	**Classification Error**
Original Problem	3456	3	11.6	16.52402	25
Feature4 Removed	5127	5.4	16.6	15.6787(5.1%)	22.39582(10.4%)
Feature 4, 5, 8 Removed	1371	1	4	16.5006(0.14%)	24.79168(0.8%)

Notes:
1. The values in brackets show the percentage reduction of the specified parameter obtained in the modified feature space compared to the one obtained in the original feature,
2. Training time is measured in seconds,
3. The column starting with "Hidden Units" shows number of hidden units in the neural network when training is finished. Because the results listed are average value of ten experiments, there are decimal parts in the results,
4. "Test Error" means the regression error obtained from test patterns and the "Classification Error" means classification error obtained from test patterns.

Based on table 4.2 the performance of the neural network classifier is improved significantly in terms of classification accuracy and training time After removing the irrelevant feature. From the experiment, I also find that using 0.1 as the value of threshold T_1 is a very strict condition. If a larger value, e.g. 0.4, is used, there are some "boundary features" that can be detected. The features have some contribution to the accurate classification of the module, but the contribution is limited. Normally, these

boundary features will make the neural network harder to converge. For example, in Table 4.1 feature 5 and feature 8 have RIF values of 0.3216 and 0.3726 respectively, which are larger than the threshold value we discussed earlier. However, they are still far less than 1, which means they do not carry classification information as much as the other features. In Table 4.2, the test error and classification error raise a little, but the training time and number of hidden units (network complexity) drop to a very small value after removing both boundary features. The exact value of T_1 can be varied from problem to problem and 0.1 is only a heuristic based on the experiment results that normally reduces the classification error as much as possible in many problems. If the user focuses on simplifying the input feature space as much as possible while keeping the classification accuracy in an acceptable range, he can always use a larger T_1, for example, 0.4 in this problem. However, normally this value should not be greater than 0.5, from the experiment results.

4.4.2 Thyroid Problem

This problem is divided into three modules in a modular neural network classifier, because there are three classes, one module for each class. The RIF and CRIF values of the features in the three modules are listed in Table 4.3, 4.4 and 4.5 respectively.

Table 4.3 RIF and CRIF of Features in the First Module of the Thyroid Problem

	RIF	CRIF 15	CRIF 17	CRIF 18	CRIF 19	CRIF 20	CRIF 21
Feature 1	0.0343	0.0331	0.0007	0.063	0.0397	0.0385	0.0387
Feature 2	0.0131	0.0126	0.0209	0.0178	0.0167	0.0233	0.0152
Feature 3	0.0164	0.0157	0.0402	0.0265	0.0103	0.0172	0.0177
Feature 4	0.034	0.0338	0.0376	0.0389	0.0388	0.0436	0.037
Feature 5	0.0041	0.0039	0.0184	0.022	0.0039	0.0051	0.0055
Feature 6	0.0501	0.0481	0.0538	0.0545	0.0641	0.0665	0.0567
Feature 7	0.0155	0.0147	0.0661	0.0213	0.0008	0.0233	0.0119
Feature 8	0.0174	0.0166	0.0154	0.0179	0.01	0.0135	0.0169

Feature 9	0.0228	0.0219	0.0101	0.0221	0.0373	0.0373	0.027
Feature 10	0.0236	0.0227	0.0286	0.0306	0.0312	0.0326	0.0267
Feature 11	0.0288	0.0276	0.0264	0.0252	0.0363	0.038	0.0329
Feature 12	0.0641	0.0614	0.0734	0.0801	0.0749	0.0812	0.0717
Feature 13	0.0465	0.0446	0.0693	0.0546	0.0529	0.0551	0.0514
Feature 14	0.0375	0.0359	0.0157	0.0361	0.0474	0.0504	0.042
Feature 15	0.1445		0.227	0.1507	0.1395	0.1497	0.1548
Feature 16	0.0158	0.0151	0.0219	0.0205	0.0214	0.0225	0.0181
Feature 17	9.6481	9.2572		11.4325	12.3852	12.9401	10.873
Feature 18	3.1858	3.054	3.8771		3.8452	3.3785	3.3528
Feature 19	3.3733	3.2336	5.9632	3.9016		1.2145	2.7078
Feature 20	3.2921	3.1545	6.5621	3.2757	0.5642		2.4422
Feature 21	0.9324	0.8929	2.8722	0.7083	2.5802	1.7691	

Table 4.4 RIF and CRIF of Features in the Second Module of the Thyroid Problem

	RIF	**CRIF 1**	**CRIF 2**	**CRIF 3**	**CRIF 7**	**CRIF 8**	**CRIF 10**	**CRIF 12**
Feature 1	0.2375		0.2245	0.2599	0.2253	0.2311	0.2415	0.2269
Feature 2	0.1798	0.1631		0.16	0.1646	0.1664	0.1833	0.1728
Feature 3	0.4693	0.4224	0.4095		0.4303	0.447	0.4102	0.4521
Feature 4	0.0172	0.013	0.0118	0.0164	0.0074	0.0178	0.0035	0.0148
Feature 5	0.0707	0.0732	0.0502	0.0817	0.0679	0.0593	0.0909	0.0701
Feature 6	0.048	0.0302	0.0447	0.0094	0.0429	0.0443	0.0368	0.0468
Feature 7	0.2858	0.2763	0.2544	0.2886		0.2603	0.2891	0.2775
Feature 8	0.4409	0.4025	0.3788	0.4781	0.4		0.4267	0.427
Feature 9	0.0323	0.0092	0.0298	0.1245	0.0261	0.0302	0.0016	0.0327
Feature 10	0.4533	0.4101	0.4257	0.3954	0.4176	0.4305		0.4369
Feature 11	0.0343	0.0296	0.0446	0.0605	0.0244	0.0293	0.0429	0.0324
Feature 12	0.1399	0.1159	0.1136	0.1251	0.1311	0.1381	0.1265	
Feature 13	0.3513	0.3356	0.3177	0.3673	0.3291	0.3249	0.3434	0.3411
Feature 14	0.007	0.0132	0.0219	0.0115	0.0195	0.0068	0.0259	0.0084
Feature 15	0.4741	0.4875	0.5087	0.4966	0.4178	0.4392	0.4554	0.46
Feature 16	0.1439	0.1452	0.1603	0.1069	0.1314	0.1308	0.1424	0.1348
Feature 17	4.5771	4.0426	4.2602	4.1837	4.244	4.3329	4.4384	4.3993
Feature 18	10.0926	10.1074	9.6549	7.8248	9.3103	9.4393	10.05	9.6816
Feature 19	0.0342	0.107	0.0114	1.7402	0.2099	0.2672	0.1251	0.0033
Feature 20	0.176	0.3899	0.8326	0.924	0.725	0.4386	0.0405	0.145
Feature 21	2.7348	2.426	2.2447	2.3453	2.6755	2.766	2.526	2.6366
	CRIF 13	**CRIF 15**	**CRIF 16**	**CRIF 17**	**CRIF 18**	**CRIF 20**	**CRIF 21**	
Feature 1	0.2362	0.2318	0.2428	0.2082	0.4259	0.2287	0.1944	
Feature 2	0.1725	0.1752	0.1787	0.1766	0.2662	0.1727	0.1464	
Feature 3	0.4508	0.4569	0.4459	0.4218	0.6424	0.4513	0.3837	
Feature 4	0.0169	0.0126	0.0251	0.0068	0.0284	0.0165	0.0118	
Feature 5	0.064	0.0688	0.0612	0.0766	0.1671	0.068	0.056	
Feature 6	0.0435	0.0467	0.0415	0.0577	0.0515	0.0461	0.0383	
Feature 7	0.2816	0.2778	0.2732	0.3045	0.4201	0.2767	0.2462	
Feature 8	0.4196	0.4291	0.4179	0.4163	0.6197	0.4245	0.3675	
Feature 9	0.0289	0.0314	0.0275	0.0339	0.064	0.0313	0.0295	

Feature 10	0.4361	0.4413	0.437	0.4337	0.658	0.4361	0.3718	
Feature 11	0.0363	0.0334	0.0395	0.0366	0.0156	0.033	0.0269	
Feature 12	0.1379	0.1363	0.1195	0.1122	0.1815	0.1345	0.1147	
Feature 13		0.3418	0.3309	0.3542	0.5011	0.3382	0.29	
Feature 14	0.0062	0.0068	0.0015	0.0131	0.0406	0.0068	0.0058	
Feature 15	0.4471		0.4481	0.5285	0.6026	0.4577	0.4047	
Feature 16	0.1351	0.14		0.1423	0.2126	0.1383	0.1173	
Feature 17	4.4247	4.4586	4.4195		6.8482	4.3923	3.6287	
Feature 18	9.6801	9.8168	9.7743	10.5802		9.7406	8.7537	
Feature 19	0.083	0.0424	0.0041	2.3446	0.3069	0.093	2.2768	
Feature 20	0.2516	0.1843	0.1123	2.6373	2.5483		2.5356	
Feature 21	2.6481	2.668	2.5995	1.1148	5.3991	2.5136		

Table 4.5 RIF and CRIF of Features in the Third Module of the Thyroid Problem

	RIF	**CRIF 3**	**CRIF 8**	**CRIF 10**	**CRIF 13**	**CRIF 15**	**CRIF 17**	**CRIF 18**	**CRIF 19**	**CRIF 20**
Feature 1	0.0955	0.0959	0.0944	0.0956	0.095	0.0924	0.0453	0.1679	0.1004	0.1021
Feature 2	0.062	0.0538	0.0587	0.0628	0.0596	0.0599	0.055	0.0859	0.0669	0.0733
Feature 3	0.1217		0.1187	0.1037	0.1171	0.1174	0.0578	0.1456	0.1377	0.1386
Feature 4	0.0221	0.0214	0.0208	0.025	0.0211	0.0235	0.0293	0.027	0.0213	0.024
Feature 5	0.0235	0.0249	0.0203	0.03	0.0211	0.0227	0.0315	0.0634	0.0245	0.0261
Feature 6	0.0537	0.0417	0.0515	0.0478	0.0505	0.0517	0.0563	0.0618	0.0587	0.0603
Feature 7	0.0944	0.0898	0.0879	0.0951	0.0935	0.0908	0.1193	0.1294	0.0876	0.0764
Feature 8	0.1404	0.1417		0.1355	0.1335	0.1354	0.102	0.1789	0.1428	0.1504
Feature 9	0.0089	0.0139	0.0087	0.0189	0.0093	0.0085	0.0009	0.0027	0.0155	0.0148
Feature 10	0.149	0.1266	0.1446		0.1435	0.1437	0.1164	0.2003	0.1612	0.1672
Feature 11	0.0328	0.0376	0.0307	0.0343	0.0328	0.0316	0.0294	0.0264	0.0354	0.0367
Feature 12	0.0108	0.0145	0.0085	0.0121	0.009	0.0104	0.0358	0.0232	0.0081	0.0101
Feature 13	0.1379	0.1345	0.1309	0.1336		0.1329	0.1326	0.1803	0.1445	0.1499
Feature 14	0.0278	0.0312	0.0268	0.0199	0.0269	0.0268	0.01	0.0213	0.0303	0.0318
Feature 15	0.2512	0.2439	0.2397	0.2391	0.2381		0.2987	0.292	0.2418	0.2536
Feature 16	0.0539	0.042	0.0504	0.0529	0.0506	0.0519	0.0484	0.0741	0.059	0.0613
Feature 17	8.993	8.4933	8.6893	8.5184	8.6479	8.6741		11.9233	9.89	10.2119
Feature 18	5.4337	4.6057	5.2164	5.2881	5.2163	5.2357	5.4357		5.7449	5.5265

Feature 19	2.6749	2.9937	2.5074	2.5751	2.5486	2.5748	5.3698	3.3731		0.8522
Feature 20	2.5695	2.7289	2.3935	2.492	2.4346	2.4712	5.9213	2.2291	0.4289	
Feature 21	0.0433	0.0651	0.1007	0.0201	0.0511	0.0447	2.1045	0.7941	2.6004	2.0329

In the module of class one, fifteen out of twenty-one features are found to be irrelevant to the module by RIF. After applying RFWA, the entire fifteen features are classified to be noises to the module, which are feature 1-14 and feature 16. Table 4.6 shows the experiment results before and after removing the fifteen noise features from the input feature space.

Table 4.6 Results of the First Module of the Thyroid Problem

	Epochs	**Training Time**	**Hidden Units**	**Test Error**	**Classification Error**
Original Problem	753	52.2	2.1	0.801002	1.578%
Noise Features Removed	867	28.8	2.5	0.655246 (18.2%)	1.350% (14.44%)

In the module of class two, feature 4, 5, 6, 9, 11, and 14 are detected as noise features by RIF and RFWA. Feature 19 is detected as redundant. The experiment results are shown in table 4.7.

Table 4.7 Results of the Second Module of the Thyroid Problem

	Epochs	**Training Time**	**Hidden Units**	**Test Error**	**Classification Error**
Original Problem	13165	1187	12.66667	1.37877	1.833%
Noise Features Removed	11303	943.2	8.8	0.944492 (31.5%)	1.144% (37.6%)

In the module of class three, feature 1, 2 4-7, 9, 11, 12, 14, 16 and 21 are detected as irrelevant by RIF. From RFWA, feature 21 is found to be a redundant feature and all the other features are noises. Table 4.8 shows the experiment results before and after removing the irrelevant features.

Table 4.8 Results of the Third Module of the Thyroid1 Problem

	Epochs	**Training Time**	**Hidden Units**	**Test Error**	**Classification Error**
Original Problem	3936.25	338.5	7.75	1.55297	1.722225
Noise Features Removed	2568	135.8	5.2	1.508322 (2.88%)	1.61111 (6.45%)

Based on the experiment results listed above, the performance of the neural network improves a lot. The training time and network complexity are reduced significantly, while the classification accuracy improved more or less. In the second module of the problem, the classification error is reduced up to 37.6%.

So experiments with some other benchmark problems, such as the Glass problem[4] (Table 9 – Table 11), are also conducted. From the experiment results, the feature selection techniques shows great performance for problems with multiple classes and large input feature spaces. For example, in the Thyroid problem, there are nearly half of the features in each module detected as irrelevant features by RIF. After removing the irrelevant features, the generalization accuracy, learning speed and network complexity improved dramatically. As an addition, it is observed that most of the irrelevant features are noise features in the experiments.

[4] Only the experiment results of module 1, 2 and 3 of the Glass problem are listed in this section. The other three modules have very small classification error and no feature selection is performed.

Table 4.9 Results of the First Module of the Glass Problem

	Epochs	**Training Time**	**Hidden Units**	**Test Error**	**Classification Error**
Original Problem	856.67	1.556	1.444	19.67771	35.6%
Feature 1, 9 Removed	438	0.6	0.3	18.81184 (4%)	35.4% (0.56%)

Table 4.10 Results of the Second Module of the Glass1 Problem

	Epochs	**Training Time**	**Hidden Units**	**Test Error**	**Classification Error**
Original Problem	3476	6.6	3.7	21.11	33.02%
Feature 1, 9 Removed	9683.5	19.7	6.5	18.52 (12.27%)	23.4% (29.134%)

Table 4.11 Results of the Third Module of the Glass1 Problem

	Epochs	**Training Time**	**Hidden Units**	**Test Error**	**Classification Error**
Original Problem	3519.5	6.1	2.5	6.35	7.92%
Feature 7, 9 Removed	1494	2.4	1.2	6.94 (-9%)	7.92%

An interesting observation in table 4.11 is that the test error is higher than original problem while the classification is reduced after the irrelevant features are removed. It does not mean the feature selection fail in this problem, but reflects that they are designed for classification problems instead of regression problems. Because the test error is not in the consideration of the techniques, the input features that are relevant to regression but irrelevant to classification may also be removed.

To understand it clearly, it should be noted that the classification error is not necessarily be an increasing function of test error. From equation 3.7 in chapter 3, we know that the test error $E_{test} = 100 \cdot \frac{o_{\max} - o_{\min}}{K \cdot P} \sum_{p=1}^{P} \sum_{k=1}^{K} (o_{pk} - t_{pk})^2 = \alpha \sum_{p=1}^{P} \sum_{k=1}^{K} (o_{pk} - t_{pk})^2$

where α is a constant, is a linear function of the square of real distance between the

desired output position and the real output position in the output space. However, the classification error is a step function of the distance based on the decision boundary of the classification problem. There is no clear dependency between the two errors. For example, a two class problem has 5 data samples, three belonging to class 1 and the other 2 belonging to class 2. Class 1 has an output value of 1 and Class 2 has an output value of 0. The decision boundary is the output value of 0.5. There are two possible situations as shown in figure 4.2 and figure 4.3:

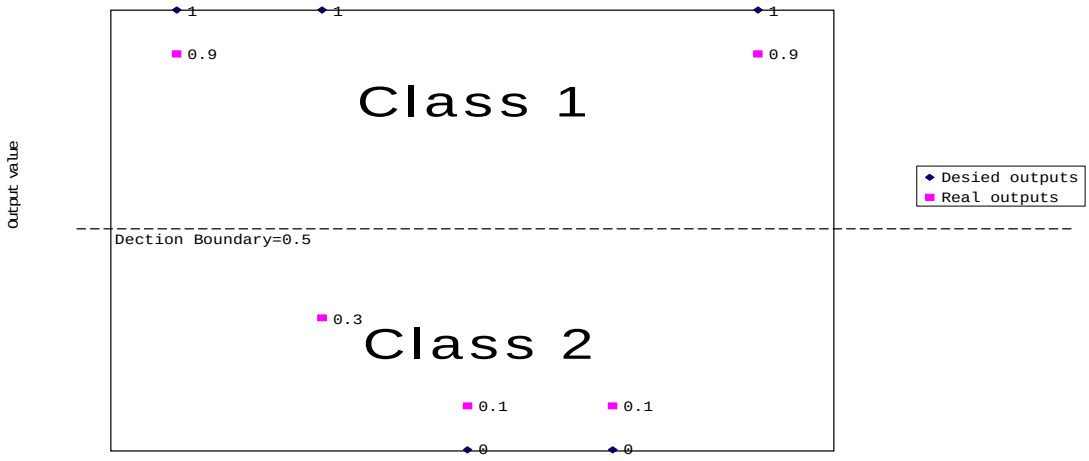


Figure 4.2 Situation 1 of a Two-Class problem

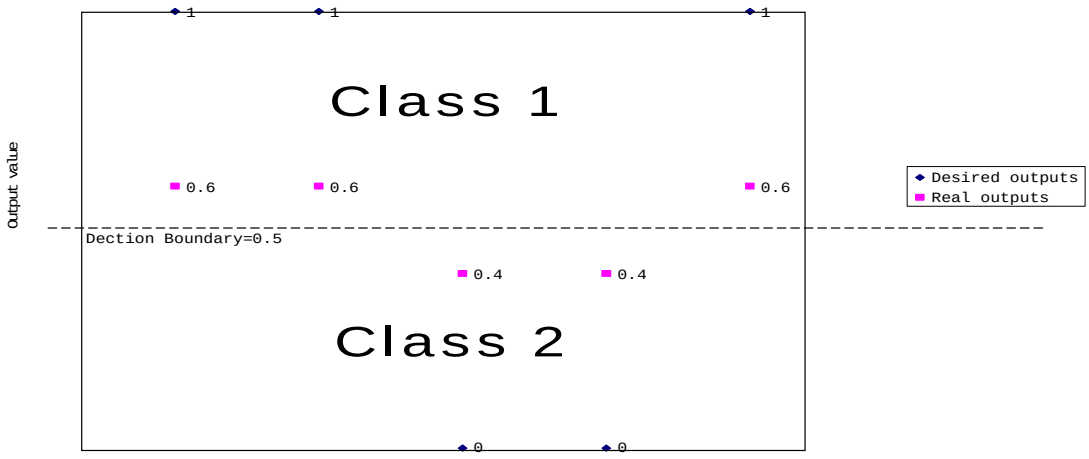


Figure 4.3 Situation 2 of a Two-Class problem

In figure 4.2, the real outputs have values of 0.9, 0.1, 0.1, 0.1 and 0.9 respectively. The classification error is increased to $E_{Class2} = 1/5 = 20\%$, while the regression error (test error) is $E_{test2} = \left(4 \times 0.1^2 + 0.7^2\right)\alpha = 0.53\alpha$. In figure 4.3, after some input features are removed, the real outputs have values of 0.6, 0.6, 0.4, 0.4 and 0.6 respectively. The classification error is 0 because all the real outputs are in the correct side of the decision boundary. The regression error (test error) is $E_{test1} = 5 \times 0.4^2 \alpha = 0.8\alpha$. Clearly, the classification error increases and the regression error (test error) decreases just like what is observed in the experiment.

4.5 Summary of the Chapter

In this chapter, I proposed two new feature selection techniques, RIF and RFWA for modular neural network classifiers. RIF classifies input features into relevant and irrelevant features based on the amount of classification information carried by the features. The irrelevant features detected are then removed from the input feature space of the module to improve the accuracy and/or reduce the training time. Based on the results of RIF, RFWA further classifies irrelevant features into noise and redundant features based on the correlation among features.

RIF and RFWA techniques are specially designed for modular neural networks for modular network with class decomposition. They also show some unique characteristics compared to other feature selection techniques. Table 4.12 shows the performance of the proposed feature selection methods and ADHOC [62], NNFS [22], and GADistAl [60] with Diabetes1 problem.

Table 4.12 Performance of Different Techniques in Diabetes1 Problem

Technique	Features Removed	Classification Error
RIF and RFWA (Strict)	1	22.39582
RIF and RFWA (Loose)	3	24.79168
ADHOC	5	26.8
NNFS	6	23.2
GADistA1	5.97	25.7

From the table, it is clear that though RIF and RFWA removes less irrelevant features than other proposed techniques, they significantly reduced the classification error of neural network classifier. As linear analysis based feature selection techniques, RIF and RFWA may not be as accurate as neural network performance based techniques like NNFS. However, they are much simpler and faster than performance based techniques.

Compared to other feature selection techniques, the advantages of RIF and RFWA can be summarized as follows:

1. Both RIF and RFWA require relatively small computation cost.

   Both techniques are based on the statistic distribution of features in the input feature space. It dose not retrain the network repeatedly as the other techniques based on the network performance perspective, or go through complex steps to obtain mutual information as the techniques based on the mutual information perspective. RIF needs only one calculation of optimal FLD transformation weights to detect irrelevant features, whether they are noise features or redundant features. It is even less expensive in computation than some other techniques based on the statistic distribution of features, such as the knock-out technique. For example, when detecting irrelevant features in Diabetes1 problem, NNFS needs to training neural networks more than 8 times, which

needs more than one hundred seconds in the same test machine as the one used to test the proposed methods. ADHOC and GADistA1 need to utilize complex genetic algorithms, which are even more computational expensive than NNFS. All the feature selection techniques need more computation time than training the neural network used to solve the problem, which is not acceptable in some time critical applications. In contrary, the proposed techniques need less than 1 second only.

2. It analyzes highly correlated features in a clear manner.

   RIF can detect both noise features and redundant features due to the nature of Fisher's transformation vector. Through RFWA, the relationship among features can be obtained. It provides a way to detect highly correlated features with relative small amount of computation and gives us a clear image of the internal relationship among the input features. None of the three techniques mentioned above can perform this kind of work.

3. It is independent with the learning algorithms used in the neural network.

   No matter what learning algorithm is adopted, better performance can always be achieved. In order to achieve good performance, different modules can even use different learning algorithm to train the modular neural network. In NNFS, the leaning algorithm used in feature selection and in training of the problem-solving network should be the same, though the author did not mention it.

Though RIF and RFWA are designed for modular neural network classifiers, they can be applied to other classifiers as well, such as Bayes classifiers, because RIF and RFWA are independent with the types of classifiers.

Chapter 5
Conclusion and Future Works

In the thesis, the techniques to improve the flexibility and accuracy of neural network are proposed and discussed. These techniques belongs to three related research topics of neural network, which are incremental learning in dynamic environment, task decomposition and feature selection.

The research started from investigating network structures that can adapt themselves when new output attributes are introduced into the existing system. How to integrate learnt knowledge in the existing neural network with the new incoming knowledge to form a new neural network is the primary interest. The Incremental Output Learning (IOL) methods take the advantages of modular neural network to preserve learnt knowledge while leaning the new knowledge. They can provide continuous work in the adaptation process and smooth handover between the existing neural network and the upgraded neural network, which is very useful in industrial applications. They are also proven to be very efficient and accurate.

Based on one of the structures developed in the incremental learning research, a new task decomposition of hierarchical incremental class learning (HICL) was developed. Because of the hierarchical relationship between its sub-networks, HICL not only avoids interferences between output attributes, but also facilitates the favorable information flow between its sub-networks. Hence, it is more accurate compared to many other task decomposition techniques, such as class decomposition. HICL is also

very flexible to environmental changes. It adapts new output attributes automatically due to its structure.

In order to improve the efficiency and accuracy of modular neural networks, I developed two feature selection techniques of Relative Importance Factor (RIF) and Relative FLD Weight Analysis (RFWA). These techniques make use of the optimal transformation weights from Fisher's linear discriminant function. RIF technique can detect features that are irrelevant to the classification problem. The RFWA technique can further classify the irrelevant features into noise features and redundant features. Compared to other feature selection techniques in literacy, RIF and RFWA require relatively small computation cost and independent with the leaning algorithm used in a neural network.

In summary, several techniques and methods have been proposed in this thesis to enhance the flexibility and accuracy of neural networks. These techniques and methods are proven to be effective and practical by experiments. They can be easily applied to practical neural network applications.

There are some ideas in all my three research topics that need to be developed and tested in the future research. In the topic of incremental output learning, there is no methods being proposed based on internal adaptation, which adapts the output change with inserting new neurons and adjust the existing link weights between neurons. If the researcher can find the way to make use of positive correlation between the neurons of the network, the internal adaptation methods may give better performance than external ones. In the topic of task decomposition with hierarchy structure, the

MSEF-CDE and MSEF-FLD ordering focus only on accuracy. However, the high accuracy is in the cost of long training time. In the future research, the researcher should try to find an ordering method that balances high accuracy with reasonable training time. In the topic of feature selection, the proposed feature selection methods have a limitation that they can only work for classification problems with class decomposition. How to extend it into normal neural network without decomposition is still a problem. A possible solution is to find a balanced overall goodness score for each input feature from the RIF and CRIF values of the input feature obtained in each individual class.

References:

[1] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, London: Pretice-Hall, 1999.

[2] Aleksander I., and H. Morton, *An Introduction to Neural Computing*, London: Chapman and Hall, 1990.

[3] Geman S., E. Bienenstock, R. DOursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1-58, 1992.

[4] Kerlirzin P., F. Vallet, "Robustness in multilayer perceptrongs," *Neural Computation*, vol. 5, pp. 473-482, 1993.

[5] Light W., "Some aspects of radial basis function approximation," *Approximation Theory, Spline Functions and Applications*, NATO ASI vol. 256, pp. 163-190, Boston: Kluwer Academic Publishers, 1992.

[6] Kohonen T., "The self-organizing map," in *Proceedings of the institute of Electrical and Electronics Engineers*, vol. 78, pp. 1464-1480, 1990.

[7] Cotes C., V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273-297, 1995.

[8] J. -F. Hebert, M. Parizeau and N. Ghazzali, "Cursive character detection using incremental learning," in *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pp. 808 – 811, 1999.

[9] L. M. Fu, H. -H. Hsu and J. C. Principe, "Incremental backpropagation learning networks," *IEEE Transactions on Neural Networks*, vol. 7, pp. 757-761, 1996.

[10] L. Bruzzon and P. D. Fernandez, "An incremental-learning neural network for the classification of remote - sensing images," *Pattern Recognition Letters*, vol. 20, pp. 1241-1248, 1999.

[11] A. J. C. Sharkey, "Modularity, Combining and artificial neural nets," *Connection Science*, vol. 9, no. 1, pp.3-10, 1997.

[12] P. Gallinari, "Modular neural net systems, training of, " in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA: MIT Press, 1995, pp. 582-585.

[13] T. Hrycej, *Modular Learning in Neural Networks*, Chichester: John Wiley, 1992.

[14] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, no. 1, pp.79-87, 1991.

[15] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and EM algorithm," *Neural Computation*, vol. 6, no. 2, pp.181-214, 1994.

[16] K. Chen, L. Yang, X. Yu and H. Chin, "A self-generating modular neural network architecture for supervised learning," *Neurocomputing*, vol. 16, pp. 33-38, 1997.

[17] B. L. Lu, H. Kita, and Y. Nishikawa, "A multisieving neural-network architecture that decomposes learning tasks automatically," in *Proceedings of IEEE Conference on Neural Networks*, Orlando, FL, pp. 1319-1324, 1994.

[18] B. L. Lu and M. Ito, "Task decomposition and module combination based on class relations: a modular neural network for pattern classification," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1244 – 1256, 1999.

[19] R. Anand, K. Mehrotra, C. K. Mohan and S. Ranka, "Efficient classification for multiclass problems using modular neural networks," *IEEE Transactions on Neural Networks*, vol. 6, no.1, pp. 117 – 124, 1995.

[20] V. Petridis and A. Kehagias, *Predictive Modular Neural Network: Applications to Time Series*, Boston: Kluwer Academic Publishers, 1998.

[21] S. Kumar and J. Ghosh, "GAMLS: A generalized framework for associative modular learning systems," in *Proceedings of the Applications and Science of Computational Intelligence II*, Orlando, FL, pp. 24-34, 1999.

[22] Setiono R. and Liu H., "Neural network feature selector," *IEEE Transactions on Neural Networks*, vol. 8, pp. 654-662, 1997.

[23] Battiti R., "Using mutual information for selecting features in supervised neural net learning," *IEEE Transaction on Neural Networks*, vol. 5, pp. 537-550, 1994.

[24] Lerner B., Levinstein M., Rosenberg B., Guterman H., Dinstein L. and Romem Y., "Feature selection and chromosome classification using a multilayer perceptron neural network," *IEEE International Conference on Neural Networks*, vol. 6, pp. 3540-3545, 1994.

[25] Souza J. C. S., Rodrigues M. A. P., Schilling M. T. and Do Coutto Filho M.B. "Fault location in electrical power systems using intelligent systems techniques," *IEEE Transaction on Power Delivery*, vol. 16, pp. 59-67, 2001.

[26] Sheng-Uei Guan and Shanchun Li, "Incremental Learning with Respect to New Incoming Input Attributes," *Neural Processing Letters*, vol. 14, issue 3, pp. 241-260, 2001.

[27] Li Su, Sheng-Uei Guan and Y. C. Yeo, "Incremental Self-Growing Neural Networks with Changing Environment," *Journal of Intelligent Systems*, vol. 11, issue 1, pp. 43-74, 2001.

[28] A. Blum, R. L. Rivest, "Training a 3-node Neural Network is NP-complete," *Neural Networks*, vol. 5, pp. 117-128, 1992.

[29] Auda G., Kamel, M., Raafat H, "Modular Neural Network Architectures for Classification," *Neural Networks, IEEE International Conference on* , vol. 2, pp. 1279 –1284, 1996.

[30] R. Jacobs, M. Tai, and A Reynolds, "An Art2-bp Supervised Neural Net," In *World Congress on Neural networks*, San Diego, USA, vol. 3, pp. 619-624, 1994.

[31] E. Corwin, S. Greni, A. Logar, and K. Whitehead, "A Multi-stage Neural Network Classifier," In *World Congress on Neural networks*, San Diego, USA, vol. 3, pp. 198-203, 1994.

[32] L. Prechelt, "PROBEN1: A Set of Neural Network Benchmark Problems and Benchmarking Rules," *Technical Report 21/94*, Department of Informatics, University of Karlsruhe, Germany, 1994.

[33] M. Riedmiller, H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: the RPROP Algorithm," in *Proceedings of the IEEE International Conference on Neural Networks*, pp. 586-591, 1993.

[34] G. Auda, M. Kamel and H. Raafat, "Modular neural network architectures for classification," in *IEEE International Conference on Neural Networks*, vol. 2, pp.1279-1284, 1996.

[35] J. Feldman, "Neural representation of conceptual knowledge," in Nadel and al. (Eds.). *Neural connections, mental computation*. Cambridge, MA.: MIT Press,1989.

[36] H. Simon, *The sciences of the artificial.* Cambridge, MA: MIT press, 1981.

[37] G.A. Carpenter, and S. Grossberg, "The art of adaprive pattern recognition by a self organizing neural network," in *IEEE-CS Computer*, vol. 21, no. 3, pp.77-88, 1988.

[38] R.A. Jacobs, and M.I. Jordan, "A competitive modular connectionist architecture", in *Neural Information Processing System 3*, vol. 3, pp. 767-773, 1991.

[39] P. Liang, "Problem decomposition and subgoaling in artificial neural networks," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, Los Angeles, CA. 1990, pp.178-181.

[40] S. G. Romaniuk and L. O. Hall, "Divide and conquer neural networks," *Neural Networks*, vol. 6, pp.1105-1116, 1993.

[41] S. -U. Guan and S.C. Li, "An approach to parallel growing and training of neural networks," in *Proceedings of 2000 IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS2000)*, Honolulu, Hawaii, 2000.

[42] S. -U. Guan and S.C. Li, "Parallel growing and training of neural networks using output parallelism," in *IEEE Transaction on Neural Networks*, vol. 13, pp. 542 -550, 2002.

[43] R. E. Jenkins and B. P. Yuhas, "A simplified neural network solution through problem decomposition: the case of the truck backer-upper," *IEEE Transactions on Neural Networks*, vol. 4, no. 4, pp. 718 – 720, 1993.

[44] V. Petridis and A. Kehagias, *Predictive Modular Neural Network: Applications to Time Series*, Boston: Kluwer Academic Publishers, 1998.

[45] Duda R. O., and P.E. Hart, *Pattern Classification and Scene Analysis*, New York: Academic Express, 1973.

[46] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Science*, vol. 1, no. 4, 1989, pp.365-375.

[47] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing systems II*, D. S. Touretzky, G. Hinton, and T. Sejnowski, Eds. San Mateo, CA: Morgan Kaufmann Publishers, 1990, pp.524-532.

[48] L. Prechelt, "Investigation of the CasCor family of learning algorithms," *Neural Networks*, vol. 10, no. 5, pp.885-896, 1997.

[49] S. Sjogaard, "Generalization in cascade-correlation networks," in *Proceedings of the IEEE Signal Processing Workshop*, pp.59-68, 1992.

[50] S. -U. Guan and S. Li, "An approach to parallel growing and training of neural networks," in *Proceedings of 2000 IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS2000)*, Honolulu, Hawaii, 2000.

[51] D. Y. Yeung, "A neural network approach to constructive induction," in *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, Illinois, U.S.A, 1991.

[52] M. Lehtokangas, "Modelling with constructive backpropagation," *Neural Networks*, vol. 12, pp.707-716, 1999.

[53] Priddy, K. L., "Bayesian selection of important features for feed-forward neural networks," *Neurocomput*ing, vol. 5, pp.91-93, 1993.

[54] Belue L. M. and Bauer K. W., "Methods of determining input features for multilayer perceptrons," *Neural Comput*ing, vol. 7, pp. 111-121, 1995.

[55] Steppe J. M., Bauer K. W. Jr., and Rogers S. K., "Integrated feature and architecture selection," *IEEE Transaction on Neural Network*s, vol. 7, pp. 1007-1014, 1996.

[56] Yeung, D. Y., "A neural network approach to constructive induction," in *Proceedings of the Eighth International Workshop on Machine Learning,* Evanston, Illinois, 158-164. 1991.

[57] Li, Q. and Tufts, D. W., "Principal feature classification," *IEEE Transaction on Neural Network*s, vol. 8, pp. 155-160, 1997.

[58] Gonzalez A. and Perez R., "Selection of relevant features in a fuzzy genetic learning algorithm," *IEEE Transaction on Neural Networks,* vol. 48, pp. 417-425, 2001.

[59] Kwak Nojun and Choi Chong-Ho, "Input feature selection for classification problems," *IEEE Transaction on Neural Networks,* vol. 13, pp.143-159, 2002.

[60] Jihoon Yang, Vasant Honavar, "Feature Subset Selection Using a Genetic Algorithm", *IEEE Intelligent Systems,* vol. 13, no. 2, pp. 44-49, 1998.

[61] C. S. Squires and J. W. Shavlik, "Experimental analysis of aspects of the cascade-correlation learning architecture," *Machine Learning Research Group Working Paper 91-1,* Computer Science Department, University of Wisconsin-Madison, 1991.

[62] M. Richeldi and P. Lanzi, "Performing effective feature selection by investigating the deep structure of the data," *Proceedings of the Second International Conference on Knowledge Discovery and Data Minin*g, pp. 379-383, 1996.

[63] Sheng-Uei Guan and Fangming Zhu, “Incremental Learning of Collaborative Classifier Agents with New Class Acquisition – An Incremental Genetic Algorithm Approach,” *International Journal of Intelligent Systems*, vol. 18, no. 11, pp. 1173-1192, 2003

Author's Recent Publications

[1] Guan Sheng-Uei and Peng Li, "Feature Selection for Modular Neural Network Classifiers," *Journal of Intelligent Systems*, vol. 12, no. 3, 2002.

[2] Guan Sheng-Uei and Peng Li, "A Hierarchical Incremental Learning Approach to Task Decomposition," *Journal of Intelligent Systems*, vol. 12, no. 3, 2002.

[3] Guan Sheng-Uei and Peng Li, "Incremental Learning in Terms of Outputs," Accepted by *Journal of Intelligent Systems* for future publication.