A comparison between modular and non-modular neural network structures for classification problems

author: Jan Rekker
supervisor: J.A.G. Nijhuis

RuG

Technische Informatica

Afstudeerverslag

A comparison between modular and non-modular neural network structures for classification problems

author: Jan Rekker
supervisor: J.A.G. Nijhuis

Rijksuniversiteit Groningen
Technische Informatica
Postbus 800
9700 AV Groningen

September 2000

Abstract

Using a multi–layer perceptron as an implementation of a classifier can introduce some difficulties in the design process. When a lot of classes need to be identified traditional neural networks tend to become very large due to the monolithic concept. Because of the high internal connectivity of such a network, changing some weights during learning may negatively affect other weights. Training such a network can take a lot of time and may not lead to optimal performance. Also, it is very well possible that only a couple of classes produce a relatively high error which cause a decrease of the total performance. It would be useful to have the ability to retrain only the part of a network that is responsible for high errors, especially when working with large networks.

To solve several of the before mentioned issues one has the option to use a modular neural networks to build a neural classifier instead of using a monolithic concept. Modular neural networks consist of several (independent) modules that can be arranged according to several different structures. The most easy to design structure consist of placing several modules in parallel where each module is responsible for just one class. Each of these modules consists of a small neural network with just one output. This is the structure that is used in later experiments.

The main goal for us is to determine the learning robustness of a modular neural network and its modules and to compare the performance of the resulting modular network with a non–modular network. First we will look at the learning behavior of a single module. For that purpose several neural modules were trained using different values for certain design parameters. Choosing values for these parameters appears not to be very critical when kept within reasonable ranges. The overall learning process of a module is robust.

To test the performance of a modular neural network, several of these networks were trained for different kinds of classification problems. For each problem a non–modular network was also trained to use as a reference. The first problem consist of identifying three non overlapping classes of a synthesized dataset. As the classes are non overlapping, classifiers for this problem should show a relatively high performance. The modular as well as the non–modular network indeed perform very well and the learning process of both structures appear to be robust.

Two other datasets that are part of the ELENA benchmarks have also been used to train a modular and a non–modular network. Both networks perform according to the benchmarks and show robust learning behavior. The last problem consists of identifying 22 classes using a very large dataset. Therefore 22 separate modules were successfully trained and the resulting modular network showed a very high performance. The non–modular reference network showed similar results.

When looking at the results from the different experiments it is clear that the performance of a modular neural network is about the same as a non–modular networks performance. Training modules for a modular network appears to be a robust process. However, when the classification problem at hand has a lot of different classes we need to train a lot of modules, which means the total training time will increase. So when designing a classifier for a large problem one can use the simple design concept of a modular network but this will lead to longer total training times.

Samenvatting

Het kiezen van een multi–layer perceptron als implementatie van een classificator kan zorgen voor een aantal ontwerpproblemen. Indien we veel verschillende klasses willen onderscheiden zal het resulterende netwerk erg groot worden vanwege het monolitsche concept. Door de vele interne connecties kan het leren bemoeilijkt worden doordat verschillende gewichtsaanpassingen elkaar negatief beinvloeden. Het resultaat is dat het trainen van een netwerk veel meer tijd kost en dat de uiteindelijke prestatie niet optimaal is. Tevens is het mogelijk dat maar een klein gedeelte van het netwerk verantwoordelijk is voor de totale fout waardoor we graag de mogelijkheid zouden willen hebben om alleen dat gedeelte opnieuw te trainen.

Als men de hiervoor genoemde problemen op wil lossen kan men kiezen voor een modulair ontwerp concept in plaats van een monolitisch concept. Een modulair neuraal netwerk bestaat uit verschillende (onafhankelijke) modules die volgens verschillende structuren gerangschikt kunnen worden. Het meest simpele concept bestaat uit het parallel plaatsen van modules, waarbij elke module verantwoordelijk is voor het identificeren van slechts een enkele klasse. Elk van deze modules bestaat uit een klein neuraal netwerk met een output. Deze structuur zal bij latere experimenten gebruikt worden.

Het belangrijkste doel is om the bepalen of het leerproces van een modulair netwerk robuust is en wat de performance is ten opzichte van een niet–modulair netwerk. Eerst zal het leergedrag van een module nader bekeken worden. Om dit gedrag te bepalen zijn meerder modules ontworpen, waarbij een aantal verschillende leerparameters gevarieerd werd. Indien men deze parameters binnen redelijke grenzen kiest blijken ze niet echt kritisch te zijn. Het uiteindelijke leergedrag van een module blijkt robuust te zijn.

Om de performance van een modulair netwerk te testen, zijn er netwerken ontworpen voor vier verschillende soorten classificatie problemen. Voor elk probleem werd zowel een modulair als een niet–modulair netwerk ontworpen. Het eerste probleem bestaat uit het identificeren van drie niet–overlappende klasses van een gegenereerde dataset. Doordat de klasses niet overlappen zou een classificator voor deze set een hoge performance moeten kunnen halen. Zowel het modulaire als het niet–modulaire netwerk haalden inderdaad een zeer goede performance waarbij het leerproces robuust bleek.

Twee andere sets die veel gebruikt worden om neurale classificatoren te evalueren komen uit de ELENA benchmarks. Zowel het modulaire als het niet–modulaire netwerk presteerde zoals verwacht kon worden uit de benchmarks en toonden robuust leergedrag voor beide problemen. Het laatste probleem bestaat uit het identificeren van 22 klasses aan de hand van een grote dataset. Hiervoor werden 22 modules ontworpen die samen een goede prestatie leveren. Het niet–modulaire netwerk liet een soortgelijke prestatie zien.

Als we de resultaten van de verschillende experimenten bekijken dan zien we dat de performance van de modulaire netwerken ongeveer gelijk is aan dat van een niet–modulair netwerk. Het trainen van een module voor een modulair netwerk blijkt een robuust proces te zijn. Wanneer we echter veel modules moeten trainen betekent dit dat de totale trainingstijd in vergelijking met een niet–modulair netwerk veel hoger ligt. Dus het ontwerpen van een classificator voor een omvangrijk probleem zal een afweging zijn tussen het eenvoudige ontwerp concept van een modulair netwerk en de kortere trainingstijden van een niet–modulair netwerk.

1 Introduction

Neural networks are commonly used for all kinds of classification tasks. A great disadvantage of common neural networks is their relative big size that can make training and testing the network very time consuming. Therefore the idea of building a neural network out of several smaller modules is becoming very popular and will be investigated in this thesis. First we will give an introduction to classification problems and neural networks followed by a discussion of several modular structures.

1.1 Classification

Classification can be described as the task of choosing a class from a fixed number of classes to which a given sample belongs. Most of the classification problems can be characterized as waveform classification or the classification of geometric figures. Consider for instance a production machine equipped with a microphone for detecting unwanted noise that could indicate a malfunction. Analysis of these recordings consist of labelling a sampled waveform as normal operation or abnormal operation. Character recognition however, is based on classifying a sampled geometric figure as a certain character.

1.1.1 Pre–processing

Before we can do any classification we first have to decide how we will use the available data for solving the task, also known as *pre–processing*. When looking at the recognition of characters, the available data will most likely consist of (digital) images representing the character as illustrated by the left of Figure 1–1. We can decide to use all pixel information for the classification, but we can also reduce this information by projecting the image on the horizontal axis as shown on the right of Figure 1–1. The sample is then represented as a vector with each component defining the number colored pixels in the corresponding column and thereby reducing the number of inputs (i.e. 20 instead of 20x30=600). This process of reducing the number of inputs is called *feature extraction*.

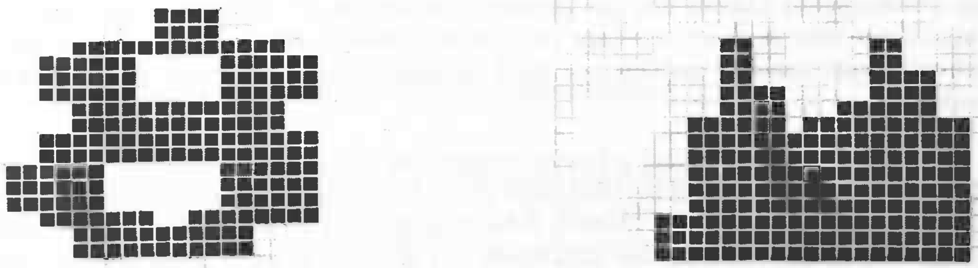


Figure 1–1: Example of feature extraction. (left) An image of the character 8. (right) Projection of the character 8 on the horizontal axis.

1.1.2 A classifier

Within classification the input data is represented as vectors, or *samples*, in a n–dimensional space with the complete dataset forming a distribution as shown in Figure 1–2 for the machine behavior model.

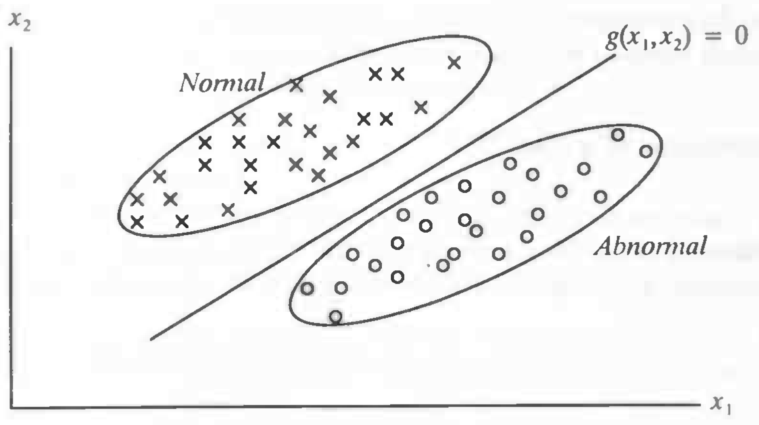


Figure 1–2: Distribution of samples from normal and abnormal machines.

The boundary between the distributions for normal and abnormal operation is represented by $g(x_1,x_2)$=0 where $g(x_1,x_2)$ is a so–called *discriminant function*. A network that detects the sign of $g(x_1,x_2)$ as shown schematically in Figure 1–3, is called a pattern *recognition network*, a *categorizer*, or a *classifier* [9]. So, when designing a classifier one has to look at the characteristics of the input data for each class and then find a proper discriminant function. This process is called *training* or *learning*.

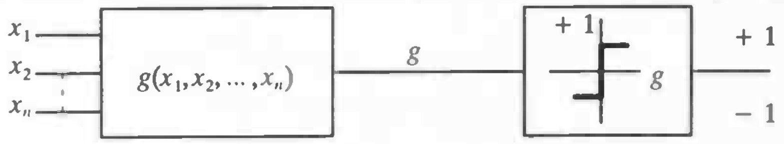


Figure 1–3: Block diagram of a classifier. (left) The discriminant function *g* on the input vector *x*. (right) Sign detector for the function *g*.

1.1.3 Learning

With the elements from the previous sections, we now have the base for a *learning machine*, with a given classifier structure and a number of unknown parameters, as shown in Figure 1–4. The input data,

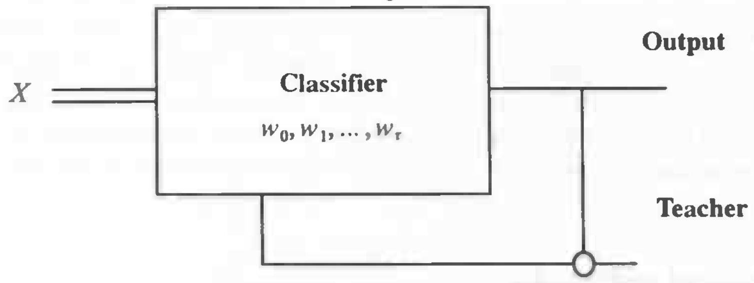


Figure 1–4: A learning machine

or *train set*, is sequentially fed to the machine which performs a classification on each sample. Every input and corresponding output is supervised by a teacher that notifies a wrong output to the classifier which then adjusts the parameters according to a given *learning algorithm*. This sequence is repeated until the classification accuracy reaches a desired level.

1.1.4 Performance of a classifier

The accuracy or *performance* of a classifier is defined as the percentage of correctly labelled samples and gives an indication how well the classifier performs during learning. Analog to this definition is the definition of the *error–rate*, which is defined as the percentage of misclassified samples.

$$recognition\ rate = \frac{\#correctly\ classified\ samples}{\#samples}$$

$$error\ rate = \frac{\#misclassified\ samples}{\#samples}$$

Another way of representing a classifiers performance is done with a *confusion matrix*. Here a table is constructed where each column represents a target class and each row represents the classifiers output. Consider for example a three class problem with classes A, B and C. A confusion matrix for the corresponding classifier that has been presented 100 samples for each class could look like table 1–1.

	Target A	Target B	Target C
output A	95	0	17
output B	0	100	0
output C	5	0	83

Table 1–1: Confusion matrix for a three class classification problem.

The table shows that 95 out of 100 samples with target class A were classified as class A. It also shows that 5 samples were misclassified as output C. Similar to this, class B has been classified without errors and 17 samples from class C were misclassified. So a confusion matrix gives not only the overall performance but also the kind of errors the classifier has made. When looking at Table 1–1 it seems that class A and C are somehow overlapping because of the relative high error–rate.

1.2 Neural networks

To build a classifier as described in the previous paragraph one can decide to use neural networks. This is often done because neural networks perfectly fit the classifier structure as described above. Neural networks are modelled after the human brain where several neurons cooperate which causes the ability to learn. Within neural network design, various architectures can be used for building such a network. However, we will concentrate on one of the most popular structures. But first we will give a short introduction on the basic building blocks of neural networks.

1.2.1 The neuron model

First, have a look at the neuron model. As shown in Figure 1–5 the structure of the model is equivalent to a classifier's structure as shown in Figure 1–3. It contains the following components:

- Synapses or connecting links, with each link having its own weight. A signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}.
- An adder or summing element for summing the input signals weighted by the respective synapses of the neuron. The operations constitute a linear combiner
- An activation function to limit the output amplitude of the neuron. Typically, the amplitude range of the output of a neuron is written as the closed unit interval [0,1] or [–1,1].

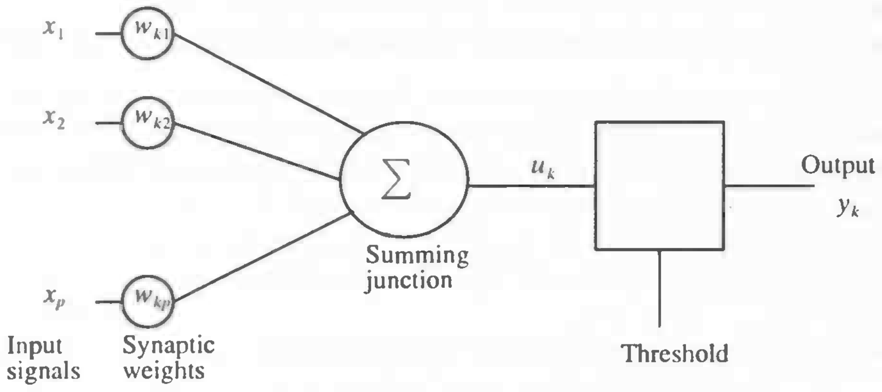


Figure 1–5: Non–linear model of a neuron

A neuron k can mathematically be described as the following equations:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

and

$$y_k = \varphi(u_k - \theta_k)$$

where x_1, x_2, ..., x_p are the input signals; w_{k1}, w_{k2}, ..., w_{kp} are the synaptic weights of neuron k; u_k is the linear combiner output; θ_k is the threshold; $\varphi(\cdot)$ is the activation function; and y_k is the output signal of the neuron. The use of threshold θ_k has the effect of applying an affine transformation to output u_k of the linear combiner in the model of Figure 1–5 as shown by

$$v_k = u_k - \theta_k$$

We can divide the type of activation function in three categories:

- *Threshold Function*. This type of activation function is described by

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

- *Piecewise–Linear Function.* This function (see Figure 1–6) is defined as

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq \frac{1}{2} \\ v & \text{if } -\frac{1}{2} > v > -\frac{1}{2} \\ 0 & \text{if } v \leq -\frac{1}{2} \end{cases}$$

  Where the amplification factor inside the linear region of operation is assumed to be unity.

- *Sigmoid Function*. The sigmoid function (Figure 1–6) is widely used for the construction of neural networks. It is defined as an increasing function that inhibits smoothness and asymptotic behavior. An example of the sigmoid is the *logistic function* defined by

$$\varphi(v) = \frac{1}{1 + exp(-av)}$$

  where a is the *slope parameter* of the sigmoid function.

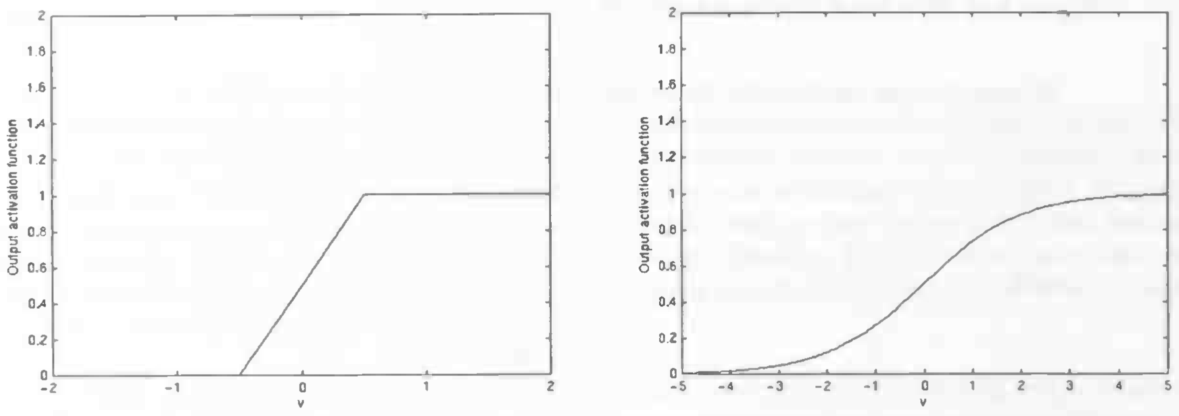


Figure 1–6: Examples of activation functions. (left) Piecewise–linear function. (right) Sigmoid function

1.2.2 Multi–layer perceptrons (MLP)

The neuron model as described in the previous section is also known as the *perceptron*. When we combine several of these perceptrons into a multi–layer network structure we have a so–called multi–layer perceptron, as illustrated in Figure 1–7.

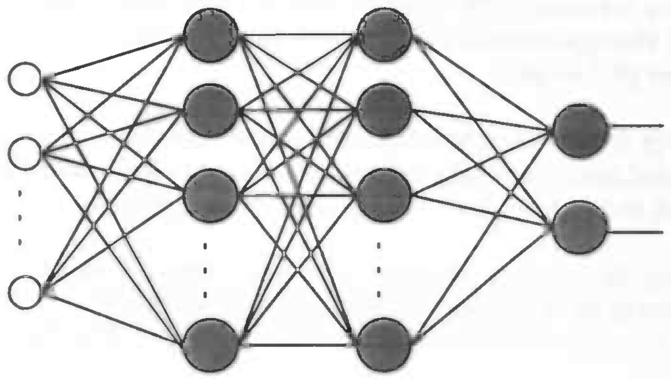


Figure 1–7: A multi–layer perceptron with *n* inputs, 2 hidden layers with *m* neurons each and an output layer consisting of two outputs. All neurons are fully connected.

A multi–layer perceptron has three distinctive characteristics:

- *Nonlinearity* at the output end. This non–linearity is mostly accomplished by using the before mentioned logistic function. Using such a smooth function is important because, otherwise, the input–output relation of the network could be reduced to that of a single layer perceptron. Also, considered from a biological view, it closely matches the modeling of real neurons.
- One ore more layers of *hidden neurons*. These hidden layers are responsible for extracting features from input patterns and thus enable the network to learn.
- *Connectivity*. The network has a high degree of connectivity, determined by the synapses of the network.

These characteristics make this network very successful and powerful but also cause the drawbacks. For example, the high connectivity makes a theoretical analysis very difficult while determining the number of hidden neurons is also a though problem. However, theoretically. a MLP network with just one hidden layer is capable of approximating any function with any accuracy.

Choosing the size of a network is highly dependant on the problem to be solved. Research has shown that a network must not be to small in size because it might then not be able to learn all features of the problem. Using a too large network can cause the network to act like a look–up table which causes undesired result with unseen input data.

1.2.3 Learning process

Training a MLP is often done using the popular *error–back propagation algorithm* (EBP)[6]. The algorithm uses a set of input vectors and the corresponding desired output vectors. First, a sample is fed to the network, then the generated output is compared to the desired output. The error is then propagated back into the network, from output to input, causing the network to change the weights according to the error. Within the whole process of neural network design this learning is a big issue. There are two important aspects involved in learning:

- *Learning robustness*. The learn process has to be very reliable or *robust*. This ensures that several identical networks trained with the same dataset show the same learning behavior and the same performance.
- *Learning speed*. When building large neural networks a single epoch can require a huge amount of computational power, especially when large datasets are used. Therefore the neural network should learn fast to reduce the total design time.

To determine the learning robustness of a neural network we can look at the corresponding *learncurves*. Learncurves show the mean train and test error over time and should ideally look like Figure 1–8, showing a fast learning network reaching a stable mean train and test error after *n* epochs.

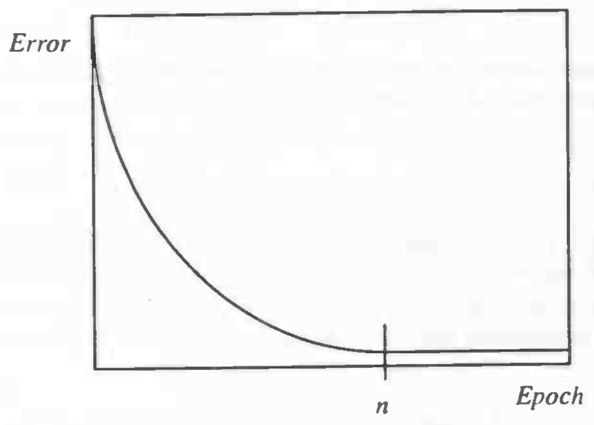


Figure 1–8: Determining learning robustness using learn curves. Smooth learn curve with a stable error after *n* epochs.

When learning is robust the learncurves of several identical trained networks will approximately look the same.The learning speed is indicated by *n*, showing the number of epochs needed to achieve a steady (minimal) error. At this point the network does not learn anymore so training can be stopped. This point is also called the *stop criteria*. When *n* is about the same for each network trained, this also indicates robust learning. Figure 1–9 shows an example of non–robust learning. Here two identical trained networks show a different learning curve with different learning speeds and minimum error. Therefore, the learning process is not robust.

Another way of evaluating the performance of a neural network used for classification is to use the recognition rate, as described in a previous section. The recognition rate gives an indication of the performance of a particular classifier while a learncurves visualizes the learn process. However, we can also decide to build a learncurve with the error rate instead of the mean error and use that to determine learning speed and robustness.

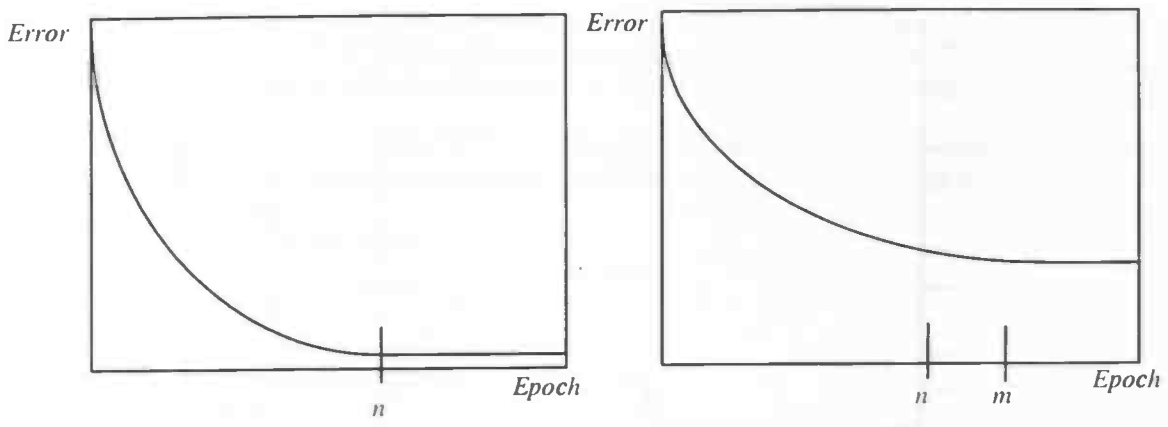


Figure 1–9: Example of a non–robust learning process. (left) Smooth learn curve reaching a small error after *n* epochs. (right) Same network trained again showing a smooth curve reaching a much higher error after *m* epochs.

1.2.4 Drawbacks of MLP's

Besides all the powerful features of MLP's mentioned in the previous sections, there are also some major drawbacks when designing neural classifiers. Solving large classification problems with neural networks can cause the networks to become very big and complex because of the monolithic concept of a MLP, and thus making training and testing very time consuming. Also, to successfully train a network one should use a large train and testset which slow down the learning process. Another drawback of the high internal connectivity of a MLP is that weight changes can be interfering leading to longer training times [4].

1.3 Modular neural networks

To overcome the drawbacks of traditional MLP classifiers, one can decide to split a large network into smaller and less complex subnetworks. These networks are called *modular neural networks* and each of the subnetworks is called a *module*. There are several other reasons why one wants to build modular neural networks instead of traditional non–modular networks ranging from biological viewpoints to hardware considerations. For us the most important motivations are:

- *Traditional design concept.* Most of the traditional design concepts are based on task decomposition, splitting a large complex tasks into several smaller manageable tasks. This concept is widely used in all kinds of disciplines and is becoming very popular in neural network research.
- *Incremental design.* Using modular design one can easily extend an existing network with more functionality. Take for instance a speech recognition system. When one has successfully trained a network to recognize for instance five different phrases, recognizing more phrases will consist of extending the existing network with functionality for the new phrases. Another option is to build a separate network for the new phrases and then combining the two networks.

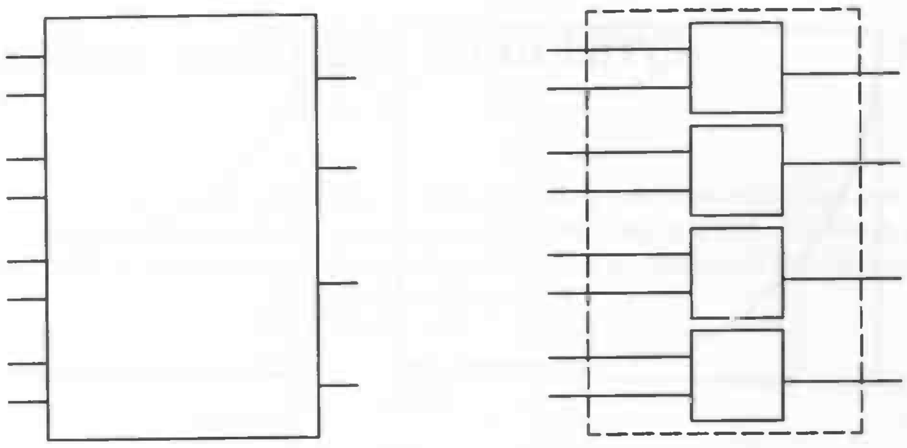


Figure 1–10: An example of a modular neural network. (left) From the outside the modular network looks like a traditional non–modular network with a certain number of inputs and outputs. (right) Inside the network several independent small modules are combined.

- *Partial retraining.* Modular design makes it easy to find the part of a neural network that is responsible for reducing the overall performance by tracing the modules outputs. When a module is found to be less accurate this module can be retrained and thus eliminates the need to retrain the entire network.
- *Faster learning.* Within modular networks not all neurons are fully connected anymore which reduces the possible interfering weight changes. This could lead to shorter training times.

1.4 This thesis

In this thesis the performance and learning robustness of modular neural networks used for classification problems will be examined. A comparison will be made between modular and non–modular structures by building both a non–modular and modular classifiers for different kinds of classification problems.

A number of different modular architectures will be presented along with each architectures advantages and disadvantages. The results from previous research on those structures together with their specific features will be discussed as well as the ease of designing such networks.

A modular neural network consists of several modules which can be build using small neural networks. The internal structure of such a single neural module will be discussed chapter 2. As we will see, one can use all kinds of internal structures to build a module while the interface to these modules are all the same.

Then, in chapter 3, we will look at a modules learn behavior and the influence of the variation of certain learn and network parameters. The focus will be on one module structure that will be used within several modular neural networks designed for different kinds of classification problems.

In chapter 4 we will discuss the results of some more experiments carried out wtih modular and non–modular networks. Here we will use different kinds of classification problems to find out how a modular network performs compared to a non–modular network. Each of these problems have specific features for which we can expect certain results. The learning robustness of both network structures for these problems will also be compared. Based on the results, a summary of the overall behavior will be given.

Finally, in chapter 5, conclusions will be presented as well as a couple of other interesting topics dealing with modular networks that are not covered in this thesis but might be worth investigating.

2 Existing modular structures

The idea of dividing a neural network into separate smaller sub–networks is not new. One has proposed several different modular concepts as shown below. We think of a neural network as a black box with a given number of inputs and outputs (see Figure 2–1). The content of the black box can be one

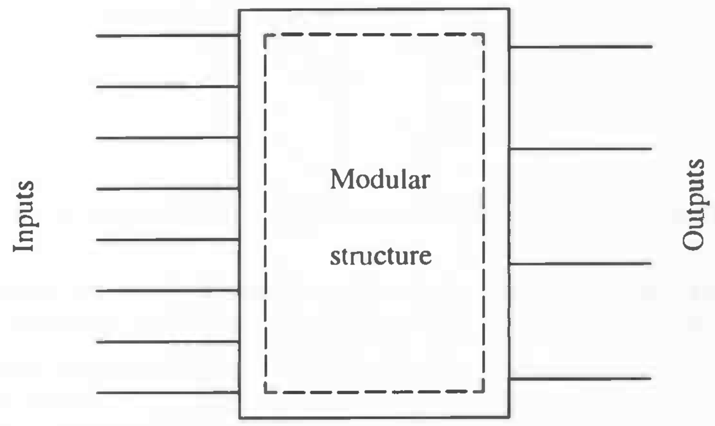


Figure 2–1: A modular neural network presented as a black box.

of several structures. We can divide those structures in two global classes:

- Single step structures. This class represents the structures in which inputs are fed to a module that is directly coupled to the outputs. The modules are placed parallel to eachother and have no mutual connection.
- Multi–step structures. This is the class of structures in which an input signal propagates through multiple modules before reaching the outputs. Some of the individual modules can thus be connected.

As we will see in the following paragraphs each of these two classes has its own advantages and disadvantages.

2.1 Single–step networks

One type of modular neural networks are the single–step networks. These networks consist of a number of parallel modules with the modules' output forming the network output. None of the modules are connected to eachother so each of the modules can be trained independently. If we divide the hidden layer of the non–modular MLP network presented in the previous chapter into several independent layers and each of the layers is connected to a certain number of outputs, we have the basic concept of a single step modular neural network. As we will see, the following structures are a variation on this concept.

2.1.1 One neural network per class

A one per class (OPC) network [1][2][3] consists of a number of independent modules, or *decoupled modules* [4], equal to the number of classes, so that each module is responsible for the classification of just one class. These so–called decoupled networks operate in parallel as illustrated in Figure 2–1.

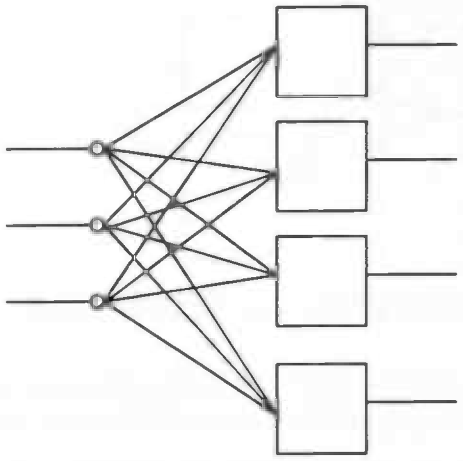


Figure 2–1: A modular structure with a single neural network for each class

Some interesting features of this type of network are:

- Each individual network converges quickly because it has to learn just a simple function
- Faster learning due to less interfering weight changes
- Resulting networks are simple and therefore more easy to understand
- Task decomposition is straightforward

Experiments have shown that this network performs as good as a non modular network [2]. Depending on the classification problem this network can outperform a non modular network. It must be noted that these results were accomplished with a modified EBP algorithm. Another application [1] showed that this modular structure is superior to non–modular networks.

Other experiments [5] have shown an decreased performance because each module would have a different perception of the input space, leading to worse overall generalization. Also, when training one of these modules we encounter the problem of imbalance of the dataset because there will be a lot more samples from the other classes. Using an imbalanced dataset to train a module with the EBP algorithm appears to be a very slow process [8]. Therefore we have to consider modifying the learning algorithm or the datasets if we want to achieve optimal results.

2.1.2 One neural network per cluster of classes

A variation on the OPC structure is the one per cluster of classes (OPCC) structure. The difference is that some classes are clustered into one sub–network and thus sharing a hidden layer. All modules are still independent of eachother Figure 2–2.

Here, the task decomposition is not obvious and we have to decide which classes are put together. We can use the OPC structure as an initial design, and based on the results decide to put certain classes

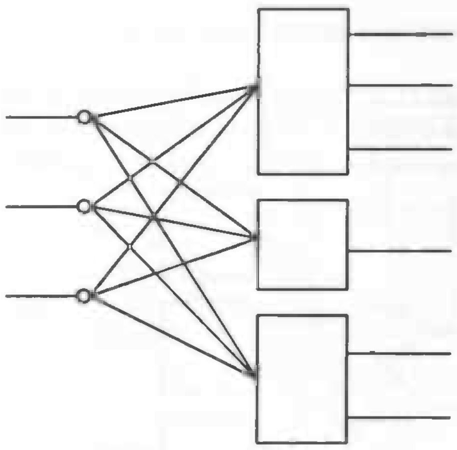


Figure 2–2: Modular structure with a neural network for each cluster of classes.

together in one module. However, it is not to say that doing this leads to better performance or faster training but it makes designing such a modular classifier much easier and logical.

As with the OPC structure, we also encounter the problem of an imbalanced dataset. If we want to modify the dataset we have to consider what classes are present and based on that, decide how the modification should be done. Research has shown that the OPCC structure extended with error correcting output bits gives good performance [3].

2.2 Multi–step network

This section describes structures cascading multiple neural network modules. This implies that some modules are dependant of other modules which means that in case of retraining a module we have to retrain several modules. The performance of such a module is also dependant on other modules performance.

2.2.1 Hierarchical structure

Using this concept one clusters the available classes and for each cluster a separate neural network is built. The outputs of these networks serve as the inputs of the following neural networks which define subclasses. In this way a hierarchical tree–structure of neural networks, like Figure 2–3, is built.

One important property of this type of network is that the performance of a module can never be optimal if its predecessor is not performing well [8]. This problem is inherent to this structure. Research on this type of network shows that it performs better than a non–modular network [8]. However, there must be noted that [8] deals just with a two–level structure. Depending on the classification problem less weight changes and training cycles are needed for good performance, resulting in decreasing learning time. Also, when looking at this structure from a designers view, it is a great example of top–down design and therefore an interesting concept. We will, however, not focus on this type of network.

2.2.2 Ensemble network

Here a number of identical neural networks is solving the whole classification task, see Figure 2–4. The network with the highest output is then chosen (majority vote) as the winner. Actually this structure

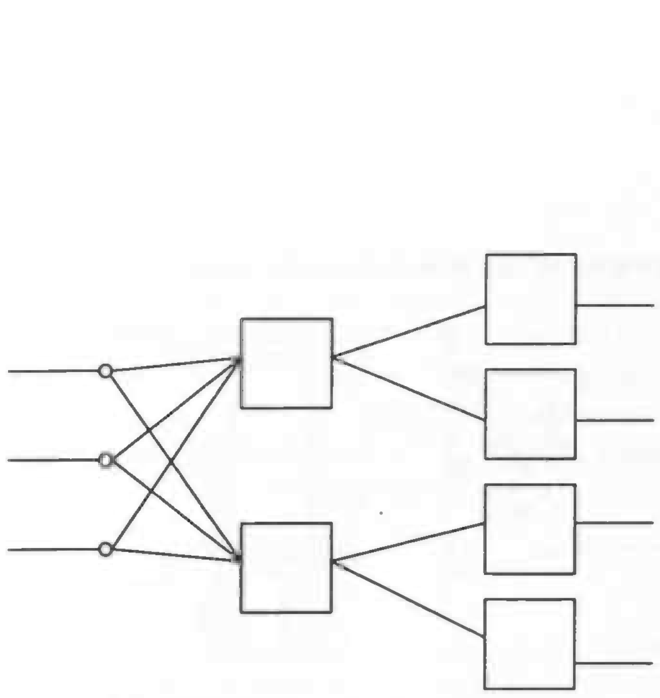


Figure 2–3: A hierarchical modular structure. Each square represents a neural network.

can be seen as a single step network but the voting module makes it a multi–step network. This structure is also the least interesting because it does not change the design concept at all.

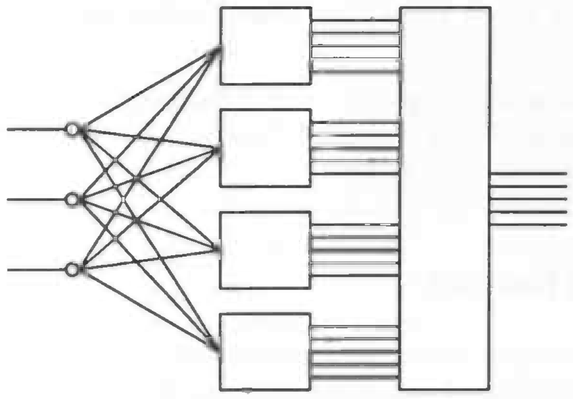


Figure 2–4: Modular structure using several neural networks in an ensemble.

The idea of creating several different initialized networks will lead to different approaches to the problem, but each network still has the drawbacks of a non–modular network. Another reason why this structure is less interesting is because one has to construct several (large) neural networks which means there will be a lot more computational power required for all the training and testing to be done. Experiments have shown that this structure is performing less than a conventional network which is caused by the majority vote mechanism [4].

2.2.3 Cooperative networks

A cooperative neural network as described in [5] consists of a number of neural networks with class– and group outputs that cooperate by using the class output for determining if a given sample belongs to their class and a group output indicating that classification should be done by another sub–network (Figure 2–5). The group outputs are fed to a voting module that determines to which class the sample should belong. The outputs of this module are then used together with the class outputs for making the final decision.

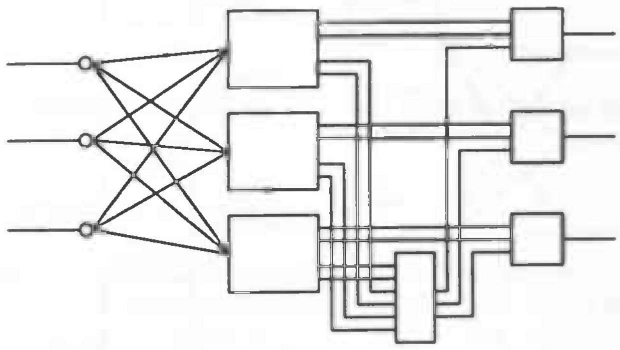


Figure 2–5: Modular structure using several cooperative neural networks

This structure uses an automatic task decomposition mechanism based on the *Adaptive Resonance Theory* (ART) [4]. This mechanism divides the data into several classes depending on its *vigilance parameter* that indicates how clearly classes should be separated. Each class is then assigned to a single module.

Experiments have shown that the performance of such a network is equally good and sometimes a lot better than a non–modular network. However, this structure is rather complex and because of the automatic task decomposition of less interest for us.

2.3 Summary of features

As described in the previous section, one can build modular neural networks using all kinds of structures. A summary of the features of each of the presented structures is given in Table 2–1.

Structure	Incremental design	Partial retraining	Complexity	Claimed performance	# learning epochs
MLP [6]	no	no	high	good	high
OPC [1,2,3]	easy	yes	low	well	low
OPCC [3]	easy	yes	low	well	low
Hierarchical [4,8]	easy	yes/no	low	good	medium
Ensemble [4]	no	no	high	good	high
CMNN [5]	hard	no	high	good	low

Table 2–1: Summary of the features of different kinds of modular structures. Note: All characteristics are derived from literature.

The table shows that only three structures are suitable for incremental design. We must note here that an ensemble network can be designed incrementally by repeatedly adding trained modules. With incremental design however, we mean the design implied by the particular classification problem and its classes. This item is directly related to the possibility to partially retrain a modular network. Therefore, partially retraining an ensemble network is not possible. Retraining a part of a hierarchical network can also mean that several other dependant modules have to be retrained and therefore is not straightforward.

When we compare the characteristics of a MLP with a modular OPC structure we can see that the OPC structure has some interesting design advantages. Therefore we will use the OPC structure in further experiments and compare it to a non–modular MLP network.

2.4 Module structures

As we have seen in the previous paragraphs we can combine a number of neural network modules on several different ways. Figure 2–6 shows how a module structure fits in the total concept of a modular neural network. A couple of internal architectures that can be used to build a module will be discussed

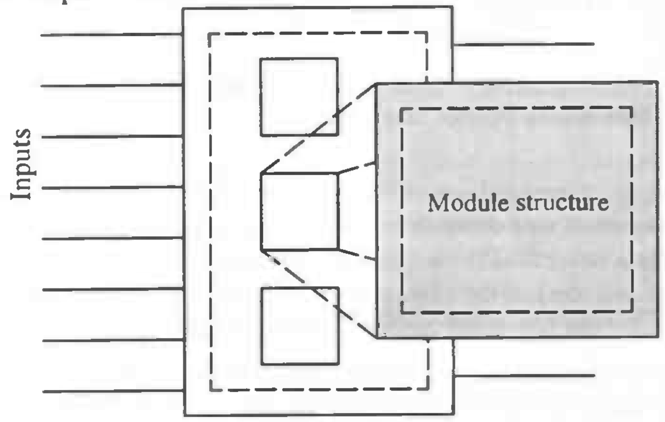


Figure 2–6: A module placed within a modular structure.

in this section. The focus will be on structures that are used within OPC concept, as this is the structure we will be using in later experiments. Note that the internal structure of a module is only an issue during the training of a module, so it is possible to use different module structures within a modular network. However, in later experiments we will use the same internal structure for each module.

2.4.1 One output per module

The most basic module structure consists of a MLP network with just one output, the target class output, and all the available inputs (Figure 2–7). It might be possible to use only certain inputs for the desired output, but as we do not know at forehand what kinds of inputs are responsible for the target class all available inputs are used. This also means that in case of a lot of inputs, there might be the need of a relatively high amount of hidden neurons to capture all features from the inputs.

When several of these modules are combined into a complete network we simply feed all the modules' outputs to the modular network outputs. The datasets used for training and testing are constructed out of the original datasets by deleting all target columns, except the selected class target column. This is the structure we will use in later experiments.

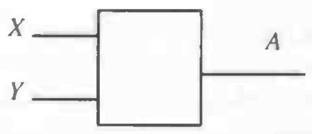


Figure 2–7: Module structure with two inputs *X* and *Y*, using a single class output *A*.

2.4.2 Two outputs per module

When we extend the module structure from the previous section with an output for the complement of the target class we have a dual output module (Figure 2–8). Using this extra output we do not need

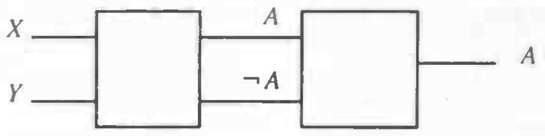


Figure 2–8: Module structure using a class and a not class output. When this module is used in a modular network we can ignore the not class output.

a threshold to determine the recognition rate as was the case with a single output module. But it is also possible that this recognition rate depends mostly on the not–class output because this class contains a lot more samples. When several of these modules are combined into a complete network we can omit the other class outputs and use just the class outputs. Therefore, the class output might show a high error that was compensated by the not–class output during training.

One can also decide to or use both outputs as inputs for some decision rule with its output fed to the network output. Note that in other modular structures like CMNN, the other output is also used in decision making.

The train and testsets are constructed by deleting all but the modules target class columns, and then generating a second column with the complement of the first. Evaluating the performance of such a module can easily be done by determining the recognition rate of the class output.

2.4.3 Multiple outputs per module

The last presented module structure consists of a MLP network with an output for each target class. When combining the modules only the selected class output is used. This is the least interesting method because for *n* classes it requires training *n* large networks and thereby ignoring several motivations for using modular networks. However, it does not introduce imbalanced datasets as the previous described structures do and when viewing a module as a black box with all the inputs and just one output it perfectly fits in a modular structure.

2.5 Module evaluation

When analyzing the performance of a classification module with a single output we have to find a suitable error measure if we want to judge the classifiers performance on the recognition rate. Therefore

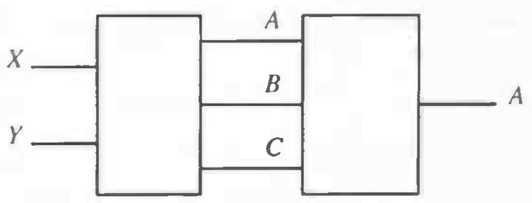
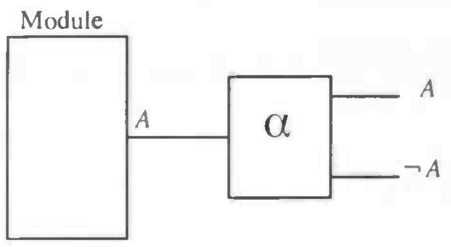


Figure 2–9: Module structure using all class outputs. After training only the class A output is used.

we will also use the structure given in Figure 3–10 when analyzing the performance of a module, where the module output A is fed to a threshold ALPHA determining if the module output should be interpreted as class A or one of the other classes. The most reasonable value for ALPHA will be 0.5, as it is exactly the middle of the targets, 0.0 and 1.0.



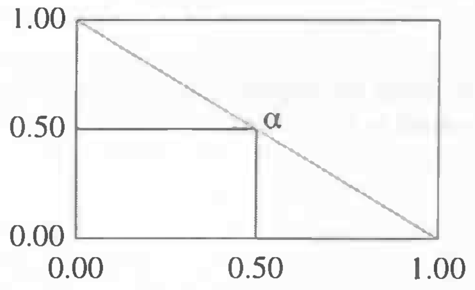


Figure 2–10: Definition of the threshold ALPHA. (left) Schematic diagram. (right) ALPHA as function of the high and low target values.

As we are dealing with single output modules we can also use *output histograms* for learning evaluation. The output histograms show the output level of a single output. For a single output this would mean that a perfect histogram for a three class classification problem with 100 samples per class would look like Figure 2–11, showing that each of the 100 class samples produces an output in the range 0.9–1.0 and the other 200 samples producing an output in the range 0.0–0.1. The histogram on the right shows a wider spread on this particular output level, which indicates a much less confident classifier.

2.6 Research topics

To tell which structure one should use with a given classification problem is not easy. However, my research will be concentrated on the OPC method because it is the most simple concept that comes closest to the traditional design concepts. The most important research topic is how a modular neural network performs compared to a non modular network. Another thing of interest is the robustness of the learning process per module. That is, training each module should be a reliable process.

To compare a modular network with a non–modular network we will first introduce the non–modular reference network. This network is a standard (large) MLP network as depicted in Figure 2–12. This network consist of an input layer, one hidden layer and an output layer that are fully connected. The number of hidden neurons within the hidden layer is assumed to be optimal. Supervised training is accomplished

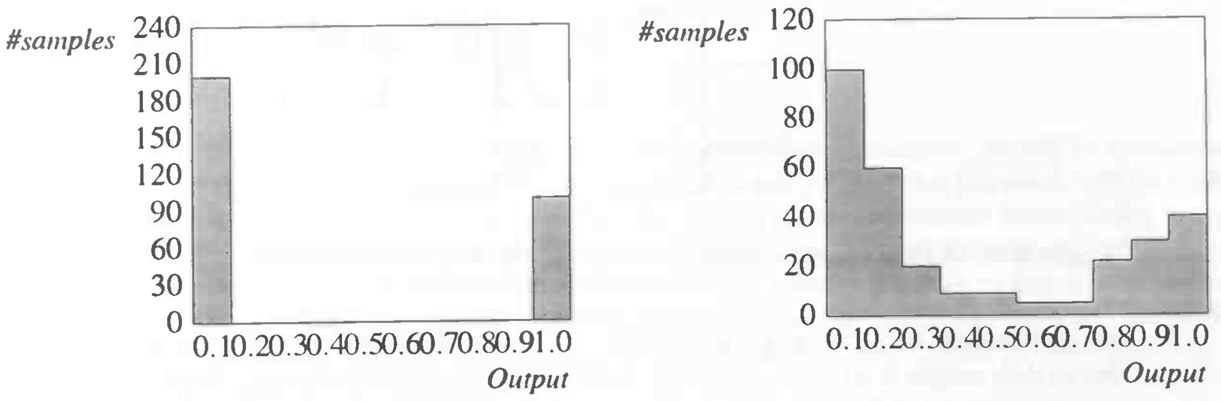


Figure 2–11: Determining learning robustness using learn curves. (left) Smooth learn curve with a stable error after *n* epochs. (right)

by using the popular error back propagation (EBP) algorithm. The parameters of this algorithm are assumed to be optimal.

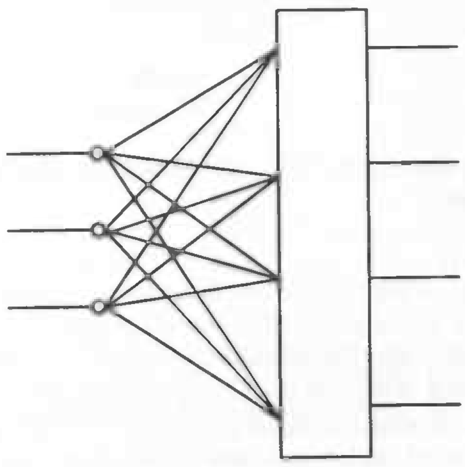


Figure 2–12: A standard MLP network that will be used as a reference for modular networks.

Now that we have defined the reference network we have to decide what kind of internal architecture we will use for a single module. Each of the modules structure will consist of a single output network, as described in paragraph 2.4.1. As we want to compare a modular network with a reference network consisting of a MLP and because MLP networks have been successfully used for all kinds of classification problems we will also use this structure for each module. The learning parameters and the number of hidden neurons should not affect the module learning robustness very much, therefore we will investigate the effect of a slight variation in these parameters.

Summarizing, our interests are:

- What is the performance of a modular neural network compared to a non–modular network?

- How robust is the learning process of a modular network compared to a non–modular network?

We will make the following assumptions:

- We are considering modular neural network consisting of several MLP networks because this type of network has been successfully used in all kinds of applications [6]
- Each of the modules in a modular neural network will consist of an input layer, one hidden layer and an output layer. The number of hidden neurons will be determined by some experiments.
- As learning method we will use standard Error Back Propagation. The number of learning cycles will be determined experimentally.
- The momentum and learning rate parameters of the EBP algorithm are supposed to be given or determined by experiments and are considered optimal.
- The non–modular reference network will be a MLP network with the number of hidden neurons being optimal. The learning parameters are also assumed optimal.

3 Module learning

As modular neural networks consist of several smaller neural network modules, we will first examine the learning behavior of a single module. The modules are build using a MLP that is trained with the EBP algorithm. Therefore we have to choose reasonable value for the number of neurons in the hidden layer of the MLP as well as deciding what values will be used for the learnrate and momentum parameters. Therefore, experiments were carried out with different kinds of learning parameters and several variations in the module structures. We are only considering a single output module because we will use such modules to build a modular network. The dataset used for these experiments consist of three non–overlapping classes as described in Appendix B.

3.1 Typical module behavior

When training modules for this particular three class problem it appears that one of the two classes shows extremely fast learning compared to the other two modules. Look for instance at the learn curve in Figure 3–1. Several identically trained class B modules showed this typical learning behavior. The

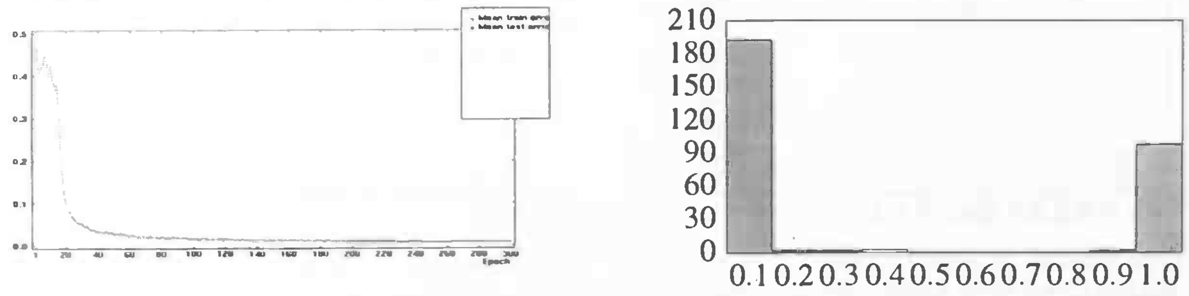
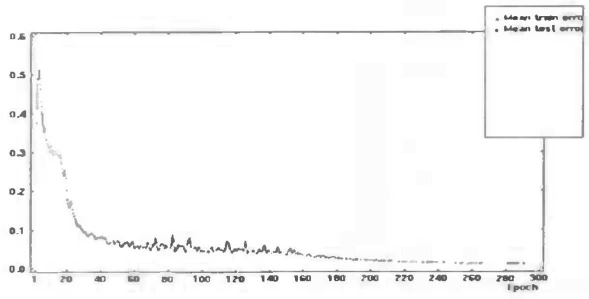


Figure 3–1: Typical learn behavior of a class B module. (left) Very smooth learn curve reaching its minimum error level after a short period of time. (right) Output histogram after100 epochs showing clear class separation. With ALPHA=0.5 the recognition rate of this module is 100%

corresponding output histogram shows no surprises as the small error showed by the learn curve implies good classification performance. Thus training this module would consist of training just 50 epochs after we will have a high performance module.

Now look at the two learncurve in Figure 3–2, representing the learning behavior of two identically trained class A modules. The curve on the left shows a period with a very instable error, after which the error decreased towards it's minimum level. The right curve also shows such a period, only much longer. It should be obvious that this is a situation we want to avoid, we would rather see an indentical learncurve for each module as this implies a reliable learning process.

Training class C modules showed learning behavior similar to the class A module, so just one of the three modules showed the desired behavior. This also indicates that the temporary unstableness is not a consequence of the modular concept. However, we are dealing with just one module structure and one set of learning parameters so there might be some influence of these on the modules' behavior. Therefore, in the next section several module an learning parameters are varied to measure the influence on a modules' learning. Considering the results showed above, the focus will be on the class A module because



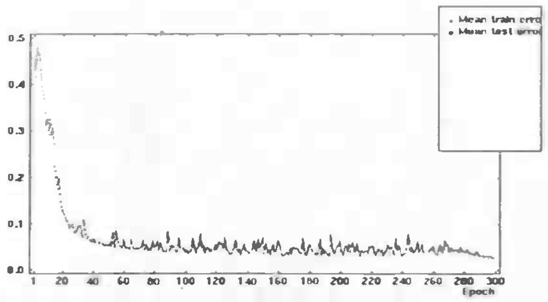
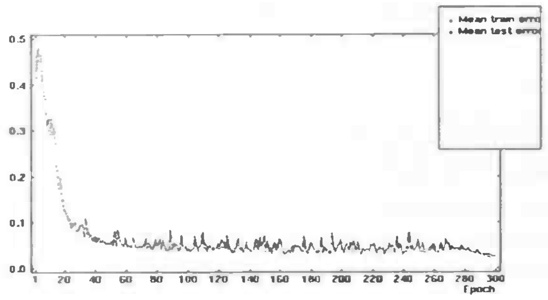


Figure 3–2: Typical learn behavior of a class A module. (left) Learn curve with a temporary unstable error. After 200 epochs the curve goes toward its minimum level. (right) Learn curve with an even longer unstable period. After 300 epochs the curve is going towards its minimum level.

of its unreliable learning behavior. First we will examine how the number of hidden neurons affects the module learning. Then the learning rate and momentum will be varied to measure their influence.

3.2 Number of hidden neurons

To overcome the problem of temporal unstableness as described in the previous section, a module with 5 hidden neurons was trained several times until a small and stable error was reached. The learning rate and momentum were set to 0.6 and 0.2 respectively. When we look at a typical learncurve (Figure 3–3) we can see that after about 250 epoch the mean error is decreasing towards this stable level. The corresponding output histogram shows that most samples are being classified either in the range 0.0–0.1 or in the range 0.9–1.0. Therefore when using ALPHA=0.5 the recognition rate lies around 99%.



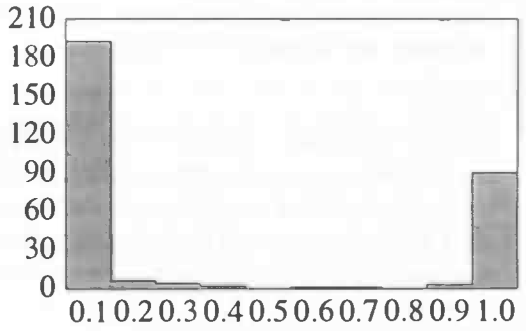


Figure 3–3: Typical learn behavior of a module with 5 hidden neurons during 300 epochs. (left) Unstable learn curve with high error level. (right) Output histogram showing non–optimal class separation.

When we use 10 hidden neurons the module behavior slightly changes. The learncurve (Figure 3–4) stays at a mean error in the range 0.05–0.08 for a while before decreasing to it's minimum level. After enough epochs the error will always go towards 0.01, and with ALPHA =0.5 results in 100% recognition which is slightly better than a module with 5 hidden neurons.

Increasing the number of hidden neurons to 15 shows quite different behavior than the situations described above. Now the learncurve shows that the mean error is stuck at 0.2 after just 50 epochs, and there-

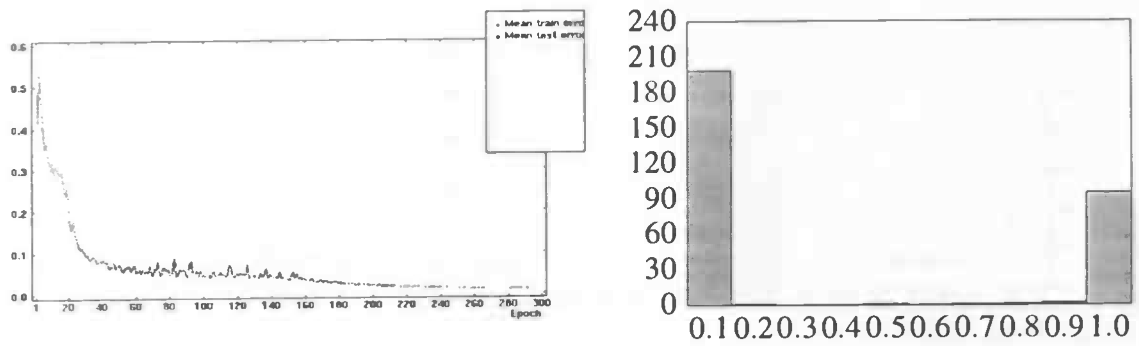


Figure 3–4: Typical learn behavior of a module with 10 hidden neurons during 300 epochs. (left) Short temporal instability is following by a low mean error. (right) Output histogram showing almost optimal class separation.

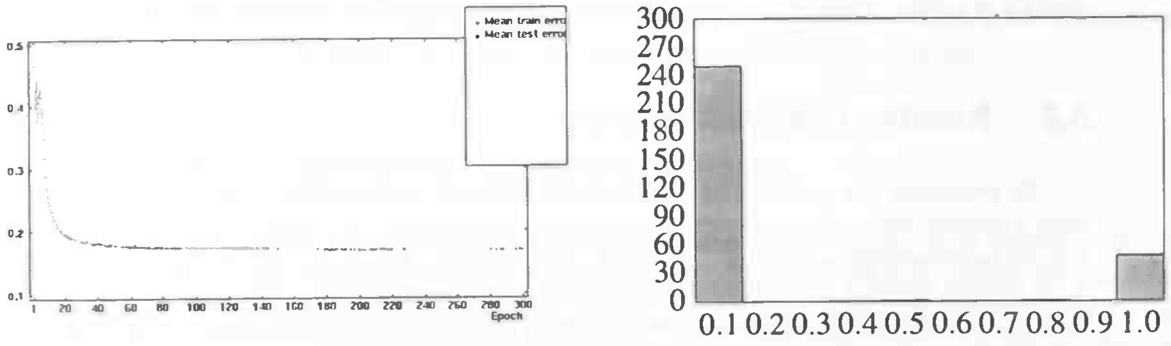


Figure 3–5: Typical learn behavior of a module with 15 hidden neurons during 300 epochs. (left) Smooth learncurve with high mean error. (right) Output histogram showing that only about 50 of the 100 class samples are correctly classified.

fore the recognition rate does not become higher than 83%. Note however that the learncurve is very smooth. When we look at the corresponding output histogram we can see that about 250 out of 300 patterns are classified as other class, indicating that the network is stuck in a local minimum This happens after about 50 epochs. This behavior indicates that the module is too capable and therefore using 10 hidden neurons in a module is the most acceptable value.

3.3 Learning rate

To determine the learning rate a module with 10 hidden neurons was trained several times for each learning rate. The momentum was set to 0.2. Looking at a typical learncurve of a module with the learning rate set to 0.4, we can see that after about 100 epochs the network is not learning anymore. Only when trained longer than 300 epochs the error will slowly decrease towards a reasonable value. This behavior is not unlikely when we look at the other learncurves presented before. Most of these curves show a temporally 'unstable' curve after which the error will decrease to a more stable (minimum) level. When low-

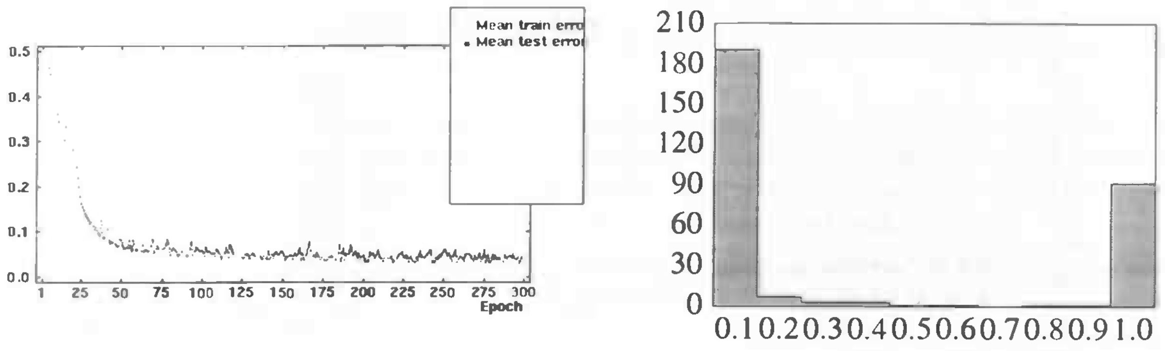


Figure 3–6: Learning rate =0.4

ering the learnrate we are slowing down the learnproces leading to longer temporal unstableness. Setting the learning rate to 0.6 gives better performance and faster learning (see previous section).

When setting the learning rate at 0.8 we can see that the temporal unstable part is smaller when

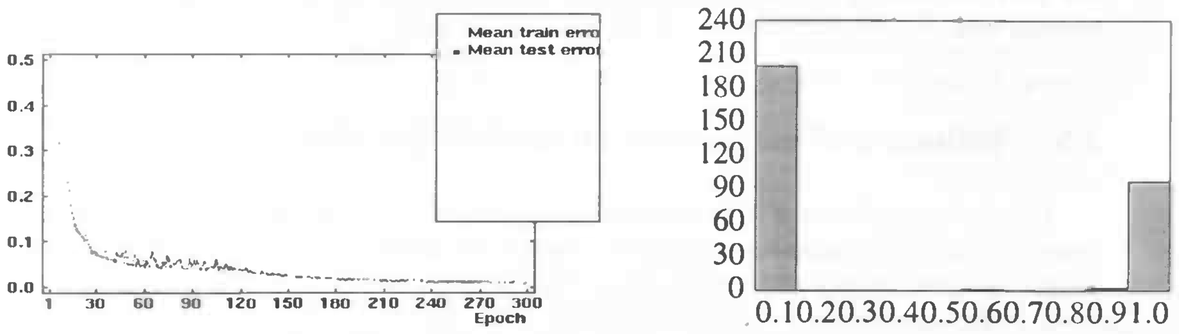


Figure 3–7: learnrate =0.8

compared to the other learncurves. This is reasonable, as a higher learning rate speeds up the learnproces. Therefore, a learning rate of 0.8 is the most useful setting because it shows the fastest learning without getting stuck in a local minimum or showing high errors.

3.4 Momentum

The second parameter of the EBP algorithm is the momentum. In the previous section this parameter was set to 0.2. When we lower the momentum and set it to 0.1 we can see a slower learning network, as illustrated by Figure 3–8. Here the error level stays relatively high and only after training much longer than 300 epochs the error goes to its minimum level. This is also illustrated by the corresponding output histogram that shows a number of samples are generating outputs in the range 0.2–0.9. Therefore, if we use ALPHA=0.5 this module has a recognition rate of 97%. When we look at the previous sections we saw a module recognizing 100% of all samples after 300 train epochs with the momentum set to 0.2. Therefore lowering the momentum does not improve the learn process.

Setting the momentum to a higher value, in this case to 0.4, does not improve learning. This setting shows a behavior similar to the module with the momentum set to 0.1 (Figure 3–8). Also, the recognition

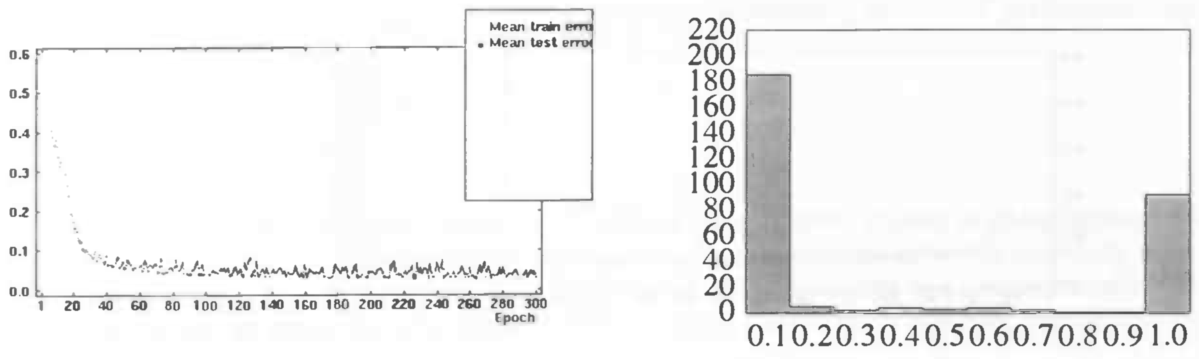


Figure 3–8: Learn behavior of a module with momentum set to 0.1. (left) After 300 epochs the network still has not reached its minimum error level. (right) The output histogram (generated after 300 epochs) shows a number of samples being classified in the range 0.2–0.9, indicating a moderately confident classifier.

rate after 300 epochs is 97%. Therefore it seems that setting the momentum to 0.2 leads to the fastest training with the best performance. So we will use this value in further experiments.

3.5 Influence of parameters on module learning

Comparing the different learn parameters as discussed above, we can see that they influence the learn behavior just a little with a few exceptions. Some of the more extreme parameter values show a radical decrease of performance and learning reliability. Varying parameters within reasonable ranges does not cause performance boost or enhanced reliability either. Therefore, we will assume that varying these parameters on modules for other kinds of data will also have minor influence.

In the previous section we have set the EBP parameters to fixed values but it is not unusual to vary the parameters over time according to a training scheme. But as we are interested in the total learning behavior of simple to design modules the focus will be on fixed values. When there is the need to further improve the modules learning behavior one can decide to use a certain training scheme or even use a modified EBP algorithm.

In the next chapter the role of the single modules' behavior within a complete modular network will be investigated. A couple of different classification problems will be solved using modular networks with the same module structure as presented above.

Summarizing, when designing modules for learning the above mentioned dataset we will use the most reliable network parameters based on the results described above, that is:

- 10 hidden neurons per module
- learning rate = 0.8
- momentum = 0.2

4 Experimental results

This chapter describes the results of some experiments performed on modular and non–modular neural classifiers. For each dataset as described the appendices B, C, D and E we trained a respective number of modules to build a complete modular network. A non–modular network was also trained for each dataset. This was done several times to determine the typical learn process of each classifier.

The first dataset is the three class dataset as described in appendix B, and has the following features:

- Small number of classes
- Small number of inputs
- No overlapping classes
- Synthesized dataset

Based on these features, training a good performing classsifier should be easy to do. Because of the small number of inputs and classes training should be fast. The size of the dataset is no problem either as we have the possibility to generate a dataset as large as we want.

The next step is to use a small dataset that has been used in other experiments so we are able to compare our classifiers performance with others. Therefore, Fishers Iris database (Appendix C) was used. This set has the following characteristics:

- Small number of classes (3)
- Benchmarks available
- One class is linear separatable, the other two classes are overlapping

Using this set, we should see that one module of the modular network learns fast and performs very well. The other two modules should show some confusion resulting in a relatively high error. The non–modular network should show similar results.

Another set often used in experiments is the Texture database (Appendix D). This set has some interesting features:

- High number of classes (11)
- High number of inputs (40)
- Benchmarks available

Because of the size of this dataset, using a modular concept becomes interesting. Also, according to benchmarks we should be able to build a high performance classifier for this problem.

The last set used is the number plates database as described in Appendix E. This set has the follwing features:

- High number of classes (22)
- High number of inputs (30)
- Good pre–processing

Because of these features we should be able to build a fast learning, high performance classifier for this particular problem.

To design a modular network, several INTERACT tools were built, as shown in Appendix F. First one has to train and evaluate ech single module. Then these modules are merged into a complete modular neural network. This resulting network is then evaluated by determining the overall performance and generating a confusion matrix.

4.1 Three class data

The first classification problem consists of classifying three different classes in a two dimensional input space. The dataset is generated to test the performance of modular neural networks on an easy seperatable dataset without any overlap of the individual classes. With the ability to generate enough class samples and due to the before described features we can expect classifiers for this data to perform very well.

4.1.1 Modular

Each module was trained several times, for 50 epochs. The modules for class A and C often show an instable learning curve, whereas class B always shows a very stable curve. However, if we train much longer than 50 epochs, we can see that the class A module also reaches a stable minimum error level (see Figure 4–1, 300 epochs). Typical output histograms after 50 epochs show less clearly separated classes. This might be a consequence of the modular design, but it is very likely that an individual output from a non–modular network shows the same kind of behavior.

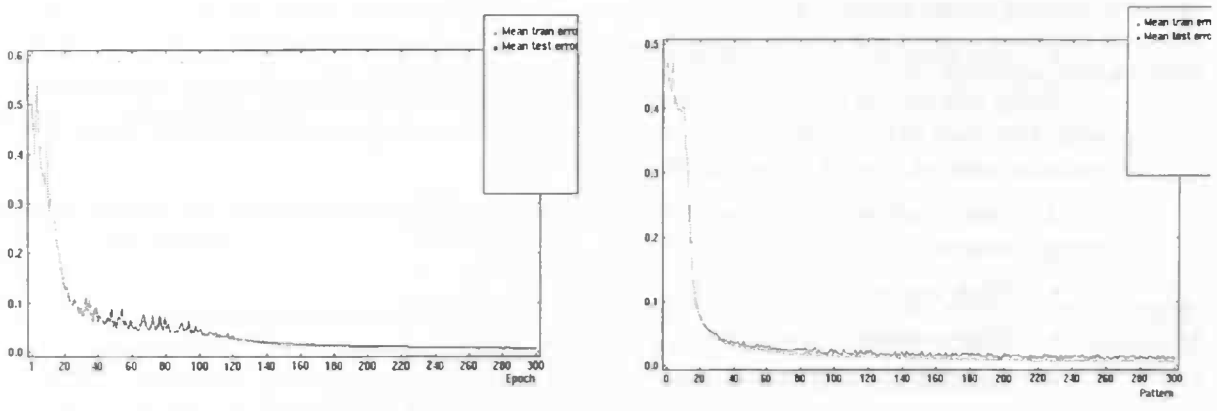


Figure 4–1: Two typical learncurves. The left curve represents the class A modules and the right curve is for the class B module. The unstable part in the first curve is clearly visible here, after 70 epochs the curve is getting smoother and after 200 epochs the minimum error is reached. The second curve does not show such a unstable part, after 50 epochs the curve is around its minimum error level.

When using α=0.5 we can see a recognition rate around 99–100% for each module. See table 2 for results of the class A module. We must take into account that several modules will be combined so preferably the class separation should be near optimal. When training a module long enough, the module error will eventually go to a stable (low) level, leading to relatively clear class separation.

When the separate modules are merged the unstable learning and the resulting high error of some of the modules does not have a catastrophic effect on the overall performance, so our restriction that each modules class separation should be optimal is not mandatory here. Therefore, training was stopped after 50 epochs which still leads to good overal performance. Also, the overall learning process appears to be robust. The merged network shows a recognition rate of 99%.

Epoch	Mean error	Rec. rate(%)
10	0.29	82.67
20	0.15	97.67
30	0.08	98.67
40	0.07	97.33
50	0.05	98.33

Table 4–2: The mean output error and it's corresponding recognition rate with ALPHA=0.5.

Class	A	B	C
A	97	2	1
B	0	100	0
C	0	0	100

Table 4–3: A typical confusion matrix for the entire modular network. As expected the overall performance is quite high. This particular matrix represents a network with a recognition rate of 99.00%.

4.1.2 Non–modular

Training a non–modular network with this three class dataset is very robust. The performance of the network on this particular dataset is high, typical recognition rates lie at about 99–100%. This was to be expected with such clearly separated classes. A typical confusion matrix also shows the abilities of the network.

Class	A	B	C
A	100	0	0
B	0	100	0
C	0	1	99

Table 4–4: A typical confusion matrix for the non–modular network. As expected the overall performance is also quite high. This matrix represents a network with a recognition rate of 99.67%, slightly better than the modular network.

4.1.3 Comparison

Comparing the modular network with the non–modular, the former appears to be able to learn equally well as the latter. When looked at the single modules output we can see some relatively high error. However, the overall performance rivals that of a non–modular network indicating that such individual errors are smoothened by the other outputs.

With this particular three class dataset the non–modular network outperforms a modular network, both in recognition accuracy and learning time. However, the difference is small which indicates that with some tweaking of certain parameters the performance of a modular network might reach the performance of a non–modular network.

4.2 Iris data

A dataset often used for evaluating neural classifiers is Fischers Iris database, as described in appendix C. This is a small dataset and it is known that there are not enough samples for optimal training of a classifier. Therefore the expected performance is not optimal and we should see some particular output behavior according to the ELENA benchmarks.

4.2.1 Modular

The three modules for classifying iris data were trained for 50 epochs. When examining the modules learn behavior it is obvious that class 1 is easy seperatable from the other two classes. After just 10 learn epochs the modules recognition rate is at 100%. The output histogram shows that classes are clearly separated. This was to be expected because it is known that this class is linear separatable. The other two classes are less clearly seperatable, as showed by each modules learn curve and error rate. The confusion matrix for the merged network (Table 4–5) also shows this behavior. This was to be expected as previous research on non–modular networks using this database showed similar results. The resulting error rate lies at about 7% and therefore the modular network performs as expected.

Class	1	2	3
1	24	0	0
2	0	23	3
3	0	2	23

Table 4–5: A typical confusion matrix for the entire modular network. As expected, class 1 is perfectly classified while the other two classes show some recognition errors. The recognition rate of the corresponding network is 93.33%.

4.2.2 Non–modular

Typical learn curves for non–modular network trained with the iris database show a very low train error and a relatively big test error. This means that after a while the network does not learn anymore because of the small train error. Therefore when using the testset the overall recognition rate lies around the 93%. This was to be expected as the ELENA benchmarks also show this performance.

Class	1	2	3
1	24	0	0
2	0	22	4
3	0	1	24

Table 4–6: A typical confusion matrix for the non–modular network. As expected, class 1 is again perfectly classified while the other two classes show some recognition errors. The recognition rate of this particular network is 93.33%.

4.2.3 Comparison

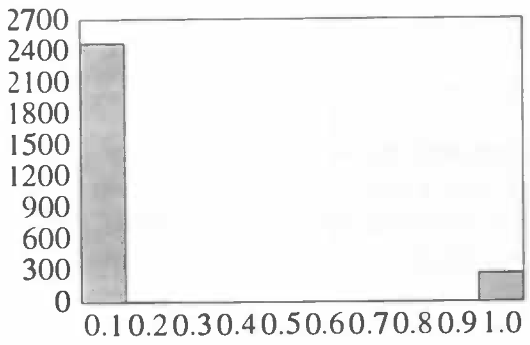
Both the modular and the non–modular network perform equally well. Each of the modules learning as well as the non–modular networks learning appears to be robust. The linear seperatability of class 1 from the others is clearly visible with both networks as is the relatively high confusion between the classes 2 and 3. So in this case, the modular network has the advantage to first build a perfect classifier for class 1. Then one can concentrate on building and optimizing classifiers for the other two classes. However, it is questionable if this really is an advantage because training a non–modular network always shows a perfect classification of class 1 and the network structure as well as the used dataset is not very large.

4.3 Texture data

The texture database as described in appendix D has some interesting features. This is a relatively big dataset with a lot of targets. According to the ELENA benchmarks we should see a high performance and the corresponding confusion matrix should indicate a clear class separation.

4.3.1 Modular

Some modules, for instance 4,5 and 9, are able to classify very clearly, as showed by the output histograms (see Figure 4–2) as well as in the confusion matrix (Figure 4–7). Other modules stay at a relatively high error level resulting in less precise classification, see for instance the modules 6 and 8. If we however, look at the performance of the resulting modular network the overall recognition rate is around 99%.



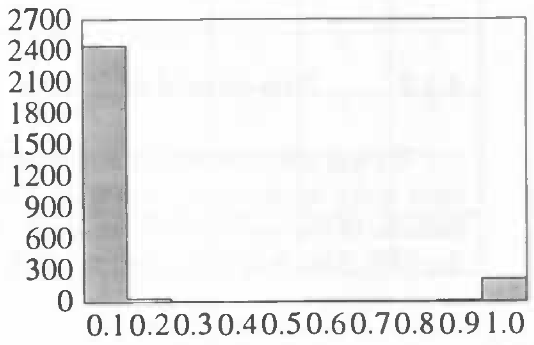


Figure 4–2: Class output histograms for some of the modules trained with the texture database. (left) Output histogram for class 5 showing a clear separation. (right) output histogram for class 6 showing some output values in the range 0.2–0.8.

When compared to results gained with non–modular networks in the ELENA benchmarks, the modular network is performing equally well. However, training 11 separate modules takes 11 times the amount of training time of a non–modular network. With a total of 2500 samples with 40 inputs each this introduces much longer total training times. But for this particular problem the possibility of a modular design concept may be more important than the total training time because one can easily retrain some of the less accurate modules.

Class	1	2	3	4	5	6	7	8	9	10	11
1	250	2	0	0	0	1	0	0	0	0	0
2	0	241	4	0	0	0	0	0	0	0	0
3	2	0	245	0	0	0	0	0	0	0	0
4	0	0	0	247	0	0	0	0	0	0	0
5	0	0	0	0	268	0	0	0	0	0	0
6	0	0	0	0	0	239	0	5	0	0	1
7	2	0	0	0	0	0	252	1	0	0	0
8	1	0	0	0	0	0	0	240	0	0	0
9	0	0	0	0	0	0	0	0	253	0	0
10	0	0	0	0	0	1	0	0	0	254	0
11	0	1	0	0	0	0	0	0	0	0	236

Table 4–7: A typical confusion matrix for the modular network. The matrix is almost similar to the ones found in the ELENA benchmarks. The overall recognition rate of this particular network is 99.09%.

4.3.2 Non–modular

The non–modular network for classification of the texture database was trained several times for 50 epochs. Each of the networks showed similar learning behavior so the learning process seems to be robust. The classification error of the non–modular network lies around 1 %. This result was to be expected as the ELENA benchmarks also show this performance. The resulting confusion matrix also closely matches the matrix given by earlier experiments.

Class	1	2	3	4	5	6	7	8	9	10	11
1	250	0	1	0	0	2	0	0	0	0	0
2	1	240	0	0	0	0	0	0	0	0	0
3	2	0	247	0	0	0	0	0	0	0	0
4	0	0	0	247	0	0	0	0	0	0	0
5	0	0	0	0	268	0	0	0	0	0	0
6	6	0	0	0	0	234	0	4	0	0	1
7	4	0	0	0	0	0	251	0	0	0	0
8	1	0	0	0	0	4	0	244	0	0	0
9	0	0	0	0	0	0	0	0	253	0	0
10	0	0	0	0	0	0	0	0	0	254	1
11	0	0	0	1	0	0	0	0	0	0	236

Table 4–8: A typical confusion matrix for the non–modular network. The matrix is almost similar to the ones found in the ELENA benchmarks. The overall recognition rate of this particular network is 99.05%.

4.3.3 Comparison

When comparing the results from both the non–modular and the modular network we can see that both structures perform equally well. The learn process of both structures appears to be very robust, but training the modular network takes approximately 10 times the time needed to train a non–modular network. This also implies that in the event of retraining a network the potential advantage of retraining a couple of modules instead of a complete network may be very small. But deciding if a modular design is feasible for this problem depends on several design considerations so it is not to say at forehand.

4.4 License plate

This database (Appendix E) contains a lot of samples and many classes. Previous results on this classification problem showed fast and robust learning with very high performance due to the good pre–processing. Using a modular structure with such a dataset has potential advantages over a non–modular structure because retraining implies only retraining some small modules instead of a complete new network. Also, with a lot of classes the concept of incremental design makes the problem more manageable.

4.4.1 Modular

Each of the modules were trained for 10 epochs, because most of the modules reached their minimum error level already after 5 epochs. All of the modules show a reliable and fast learning behavior. Some of the modules are able to make a perfect classification while some other modules make some small error, see Table 4–9. The overall performance for corresponding network is 99.19%.

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	358	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	394	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
3	1	0	191	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	166	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	405	0	0	0	3	0	0	0	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	175	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	426	0	0	0	0	0	0	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	0	180	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	212	0	0	0	0	0	0	2	0	1	0	0	0
11	1	0	0	0	0	0	0	0	0	0	197	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	227	0	0	0	0	0	0	0	0	0	0
13	2	0	0	0	0	0	0	0	0	0	2	0	208	0	3	0	0	0	0	1	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	61	0	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	255	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	240	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	225	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	262	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	219	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	329	0	3
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	169	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	96

Table 4–9: A typical confusion matrix for the modular network. The matrix shows very accurate classification, resulting in a recognition rate of 99.19%.

4.4.2 Non–modular

Due to the good pre–processing and the amount of available data the network is able to learn very fast. After just 5 epochs the recognition rate lies around 99%. Therefore training was stopped after 10 epochs. A typical confusion matrix is shown in Table 4–10.

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	354	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	1	0	0	0	0	0
2	0	393	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
3	1	0	192	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	165	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	404	3	0	0	4	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	176	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	427	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	2	178	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	211	2	0	1	0	0	0	2	0	0	0	0	0
11	1	0	0	0	0	0	0	0	0	0	197	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	227	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	1	0	0	0	0	0	0	209	0	2	3	0	0	0	1	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	1	0	0
15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	255	1	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	238	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	224	1	0	0	0	0
18	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	261	0	0	0	0
19	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	217	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	330	0	1
21	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	169	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	96

Table 4–10: A typical confusion matrix for the non–modular network. The matrix is almost similar to the one found with the modular network. The overall recognition rate of this particular network is 98.96%.

4.4.3 Comparison

Looking at the confusion matrix for both the modular and non–modular network we can see a lot of similarities. Also, the performance and the learning robustness of the modular network equals that of the non–modular. Therefore we can conclude that training a modules for this problem is a robust process and the resulting modular network is performing well.

4.5 Summary of results

In the previous sections we have seen the results of experiments carried out on several different kinds of datasets with some specific features. Table 4–11 shows these results for the modular networks, while Table 4–12 shows the non–modular network results.

Database	#Train epochs (total)	Overall recognition rate (%)	Module learn robustness
Three class	150	99	robust
Iris	150	93	robust
Texture	550	99	robust
License plate	220	99	robust

Table 4–11: Summary of the results gained with a modular network for different kinds of classification problems.

	#Train epochs	Recognition rate (%)	Learn robustness
Three class data	50	99	robust
Iris database	50	93	robust
Texture database	50	99	robust
License plate database	10	99	robust

Table 4–12: Summary of the results gained with a non–modular network for different kinds of classification problems.

When we look a the performance we can see that both modular and non–modular perform equally well on all datasets. All networks perform as we had expected, according to specific features of each dataset and some benchmarks. Also, the overall learning is robust. Note that some modules need longer training time to reach a desired minimum error level, but if we ignore the train time we have robust learning modules. Therefore we can conclude that a modular neural networks performance equals that of a non–modular network and training a modular neural network is a robust process. The only disadvantage of modular neural network is the total training time which increases linearly with the number of classes within classification problems.

5 Conclusions and recommendations

5.1 Conclusions

Using modular neural networks for different kinds of classification problems gives the same results as non–modular networks designed for the same problems. Therefore we can conclude that modular networks perform equally well as non–modular networks.

The learn process of a modular neural network is robust, at least when training a non–modular network is robust as was the case in our experiments. Varying the learnrate and the momentum parameters of the EBP algorithm does not influence the learning behavior of a module very much. Changing the number of hidden neurons within a module affects learning a bit more, but overall these settings are not to critical.

There are some specific classification problems not covered in this thesis that can hardly be tackled by non–modular networks. Probably, modular networks will also not be able to perform well on these problems as they are build of several non–modular networks with just one output.

Training a complete modular neural network takes much more time as training a non–modular, especially when the datasets used are large. The training time increases linearly with the number of classes that have to be identified. So when designing a neural classifier for many classes using large datasets, opting for a modular approach will be a trade between fast training and ease of design.

5.2 Recommendations

Within this thesis a couple of module structures are examined but only one structure (MLP) is used within the complete modular networks. Therefore it might be worth investigating how other module structures, not necessarily MLP networks, perform within a modular network. Also, mixing different structures within a network might give improved learning or simpler design. It might even be possible to use only relevant inputs for a certain module and thereby simplifying a module. But determining what inputs are relevant to a certain class output makes designing a module even harder.

Another thing of interest might be a method to improve a modules performance or learn behavior. As proposed in literature, one can use a modified learning algorithm to accomplish this. The initial learn steps should push a modules behavior into a desired direction. Also, using certain training schemes with learn parameters being varied over time might help to improve a modules design and performance. When one uses a certain module structure all kinds of other optimalizations methods for that structure can also be used. However, we must take into consideration that without any optimalisation techniques used a modular neural network performs very well as we have seen in the previous chapters.

During a modules design it would be useful to determine the learning time needed by a certain module. In our experiments we have simply performed enough training cycles to determine the learn behavior but it might not always be possible or desired to train that long. Therefore defining a suitable stop criteria is an issue as it is with non–modular neural networks.

One thing we must note is that the structure used for each of the modules is the same as the complete non–modular network, a MLP with one hidden layer containing 10 hidden neurons. The only difference is the number of outputs and thus the number of connections between the hidden and output layer. Therefore training just one single module takes almost the amount of computational power needed to train a non–modular network. Besides that, the number of epochs needed to train a non–modular network is also the number needed to train a single module of a modular network. So training a lot of modules might takes considerably more time than training a non–modular network, indicating the need for a measure to determine when a modular network is preferred over a non–modular.

A References

- [1] Application of a Multi–Structure Neural Network (MSNN) to Sorting Pistachio Nuts, A. Ghazanfari, A. Kusalik, J. Irudayaraj, International Journal of Neural Systems, Volume 8, Number 1, February 1997 Special Issue on Neural Networks for Computer Vision applications, pp. 55–61
- [2] Efficient classification for multiclass problems using modular neural networks, R. Anand, K. Mehrotra, C.K. Mohan, S. Ranka, IEEE transactions on neural networks, vol. 6, no 1, January 1995, pp. 117–124
- [3] Output coding and modularity for multi–class problems, A. Pastoors, T. Heskes, Neural Networks: Artificial Intelligence and industrial applications, Proceedings of the third annual SNN symposium on Neural Networks, 15–15 september 1995, pp. 221–224
- [4] Modular Neural Network Classifiers: A Comparative Study, G.Auda, M. Kamel, Journal of Intelligent and Robotics Systems, Volume 21, 1998, pp. 117–129.
- [5] CMNN: Cooperative Modular Neural Networks for Pattern Recognition, G. Auda, M. Kamel, Neurocomputing, Volume 20, 1998, number 1-3, pp. 189–207
- [6] Neural Networks, A comprehensive foundation, S. Haykin, MacMillan, 1994.
- [7] Partial Retraing: A new Approach to Input Relevance Determination, P. van der Laar, T. Heskes, S. Gielen.
- [8] Cooperation of Neural Nets and Task Decomposition, M. de Bollivier, P. Gallinari, S. Thiria, International Joint Conference on Neural Networks, volume 2, pp. 573–576, Seattle, USA, 1991
- [9] Introduction to Statistical Pattern Recognition, K. Fukanaga, Second Edition, Academic Press Limited, 1990

B Three class database

The datasets used for the experiments are generated datasets containing three non–overlapping classes with two inputs *x* and *y*. Each class consist of two areas in the input space as illustrated in the dataplot below. These sets are generated by using a normal distribution with a mean and a certain variance. For each sample the corresponding target output is set to 1.0, the other two are set to 0.0.

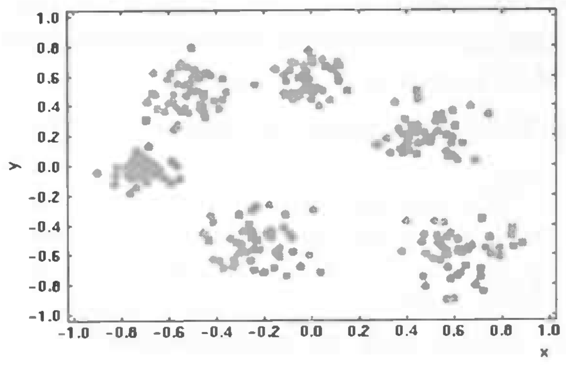


Figure B–1: Dataplot of the three class dataset.

The generated three class datasets contain 100 samples per class, making a total of 300 patterns per dataset. This is done to have enough data available but also keeping the amount of calculations per learnepoch at a reasonable level. When training a module the whole dataset is used, but only just one target column, the other two are just ignored.

C Iris database

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. Predicted attribute: class of iris plant. This is an exceedingly simple domain.

Original source: Anderson, E. (1935) "The Irises of the Gaspe Peninsula", Bulletin of the American Iris Society, 59, 2–5. Made famous by R. A. Fisher.

Class Distribution: number of instances per class

- class 1: 50 – 33.3%
- class 2: 50 – 33.3%
- class 3: 50 – 33.3%

Confusion matrix obtained with the k_NN classifier on the iris_CR.dat database (with the Leave_One_Out test method). k was set to 7 in order to reach the minimum error rate : 3.3 %.

Class	1	2	3
1	100.0	0.0	0.0
2	0.0	96.0	4.0
3	0	2.0	98.0

D Texture database

This database was generated by the Laboratory of Image Processing and Pattern Recognition (INPG–LTIRF) in the development of the Esprit project ELENA No. 6891 and the Esprit working group ATHOS No. 6620.

The aim is to distinguish between 11 different textures (Grass lawn, Pressed calf leather, Handmade paper, Raffia looped to a high pile, Cotton canvas, ...), each pattern (pixel) being characterized by 40 attributes built by the estimation of fourth order modified moments in four orientations: 0, 45, 90 and 135 degrees.

Original source: P. Brodatz "Textures: A Photographic Album for Artists and Designers", Dover Publications,Inc.,New York, 1966.

The data set contains 11 classes of 500 instances and each class refers to a type of texture in the Brodatz album. The database dimension is 40 plus one for the class label. The 40 attributes were build respectively by the estimation of the following fourth order modified moments in four orientations: 0, 45, 90 and 135 degrees: mm4(000), mm4(001), mm4(002), mm4(011), mm4(012), mm4(022), mm4(111), mm4(112), mm4(122) and mm4(222).

The following confusion matrix was obtained with the k_NN classifier on the iris_CR.dat database (with the Leave_One_Out test method). k was set to 1 in order to reach the minimum error rate : 1.0+/–0.8%.

Class	1	2	3	4	5	6	7	8	9	10	11
1	97.0	1.0	0.4	0.0	0.0	0.0	1.6	0.0	0.0	0.0	0.0
2	0.2	99.0	0.0	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.4
3	1.0	0.0	98.8	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	99.4	0.0	0.0	0.0	0.6	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	98.6	0.0	1.4	0.0	0.0	0.0
7	0.4	0.0	0.2	0.0	0.0	0.2	98.8	0.4	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	1.4	0.0	98.6	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	99.8	0.2
11	0.0	0.4	0.0	0.0	0.0	0.4	0.0	0.0	0.2	0.0	99.0

E Number plate database

This database is often used within the department of Computing Science of the University of Groningen. It consists of images of characters taken from numberplates that were preprocessed using Principal Component Analysis. The goal is to recognize each image from 22 classes total. The high number of images per class and the used preprocessing leads to very high performance classifiers.

• Inputs:	30
• Classes:	22
• #train samples:	5191
• #test samples:	5191

F Software/tools description

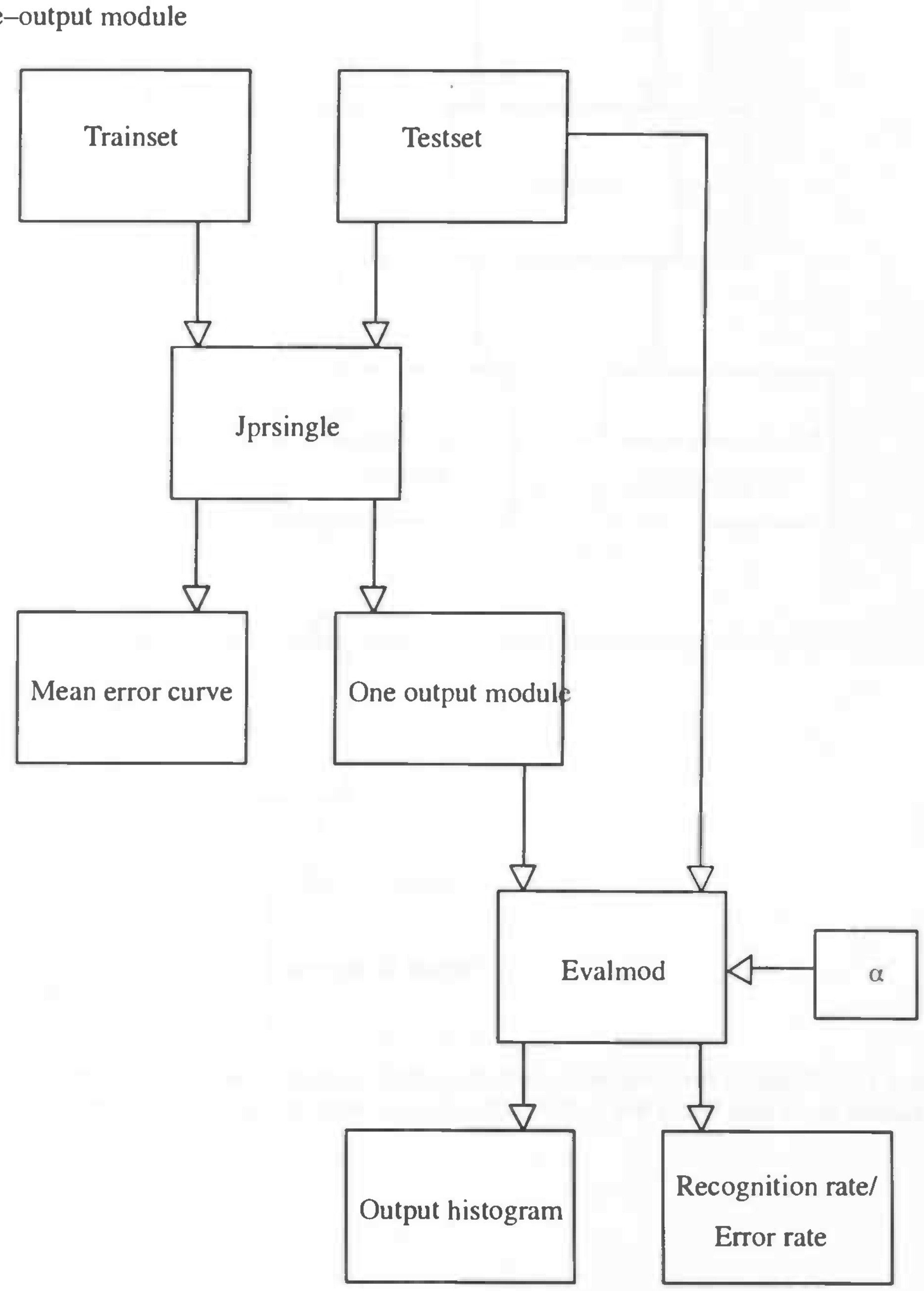


Figure F–1: Training and evaluating a single–output module. The inputs of the train and testset should be scaled to –1 and 1 and the outputs should be scaled to high and low target in the range 0.0 – 1.0.

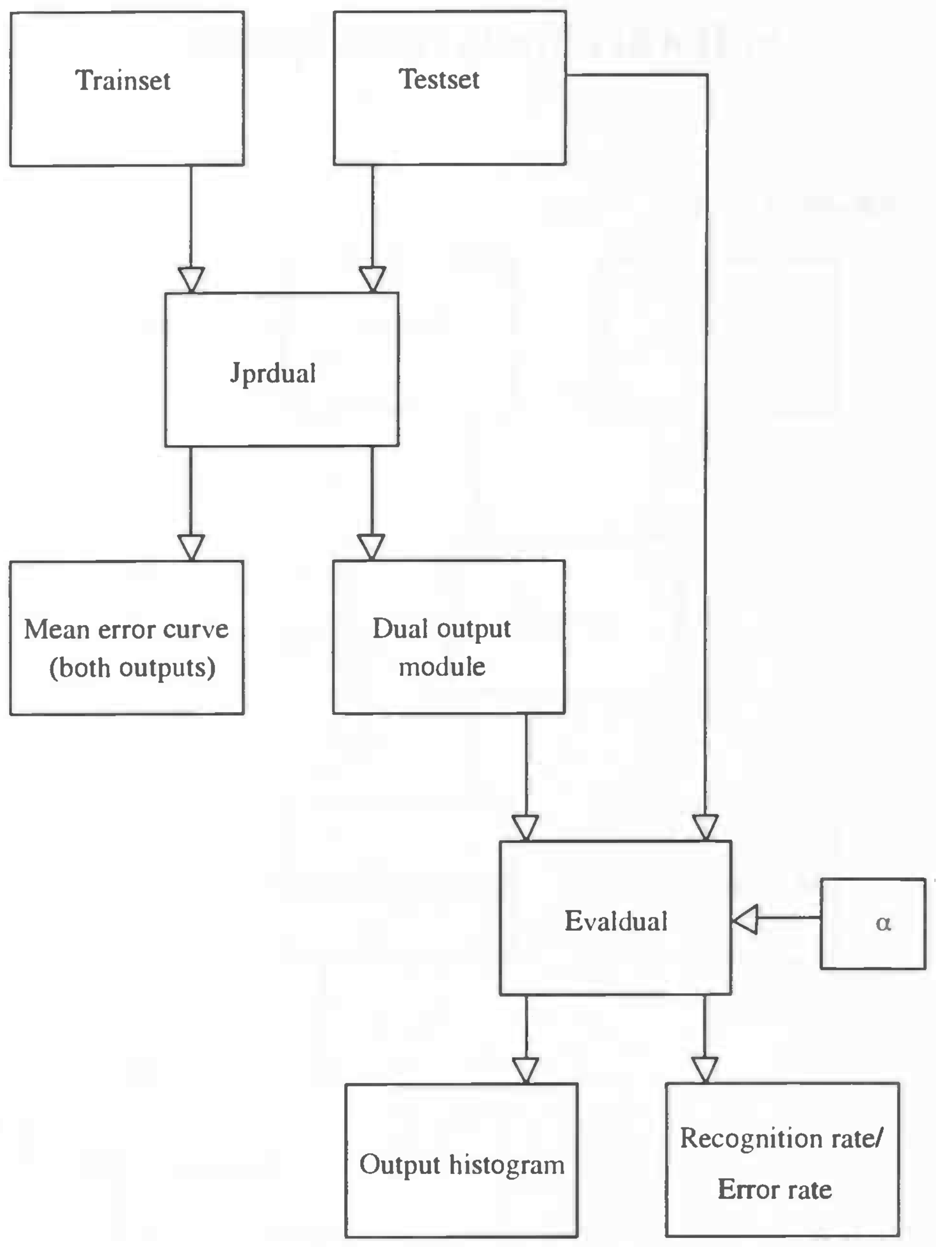


Figure F–2: Training and evaluating a dual–output module. The inputs of the train and testset should be scaled to –1 and 1 and the outputs should be scaled to high and low target in the range 0.0 – 1.0.

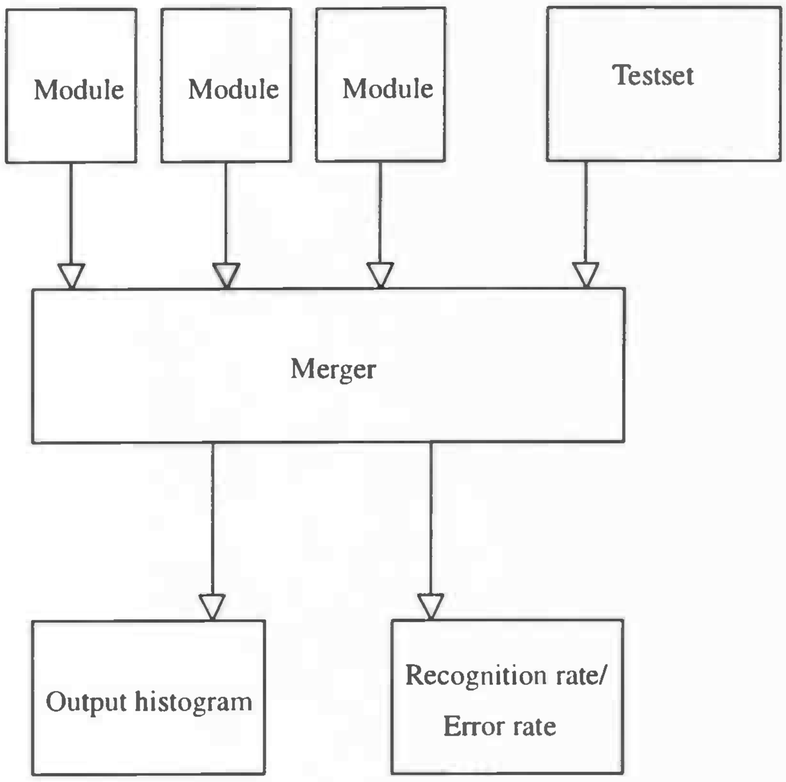


Figure F–3: Evaluating a modular neural network consisting of several modules. The testset is the test-set used for testing each module.