VALIDATION AND VERIFICATION OF EMBEDDED NEURAL SYSTEMS

PAVEN Mickaël, José, Stéphane

MSc by research in Pattern Analysis and Neural Networks

THE UNIVERSITY OF ASTON IN BIRMINGHAM

September 1996

The University of Aston in Birmingham

PAVEN Mickaël, José, Stéphane

MSc by research in Pattern Analysis and Neural Networks

Thesis summary

In a safety critical environment every element of a system must be dependable. Although neural network technology could be applied to solve various problems in safety critical environments, there is currently no established method to validate and verify systems embedding neural network technology. This deficiency is mainly due to the fact that neural network technology corresponds to a very different way of viewing software, and that validation and verification methodologies developed for conventional software do not take into account these differences.
This thesis is aimed at showing that even if at first sight a neural network function may look like a very compact 'black box', it is in fact not true and, therefore, the validation and the verification of neural systems are possible.
Firstly, a short presentation of the business context in which this project has been placed is given. Secondly, an overview of neural network technology, which includes in particular a description of the main principles on which it is based, is presented. Thirdly, a set of good practice rules to develop a successful neural network application is suggested. From these good practice rules, guidelines to assess a neural network system have been derived and tested against two case studies. The main findings are discussed. Finally, a set of research issues and suggestions for future research are provided.

This document is necessarily limited in its scope. Indeed, there is a considerable variety of models related to neural network technology, and each of them has its own characteristics. A general framework to validate and verify all these models would probably be too general and, therefore, might be unable to detect some significant mistakes specific to the model considered. Similarly, we cannot consider each model individually. We have, therefore, chosen to restrict the scope of this document to the 2 main types of models currently used, namely, the radial basis function network and the multi-layer perceptron. We have excluded, for instance, unsupervised techniques and recurrent neural networks that raise notoriously difficult problems which could not be addressed within the time available. However, whatever the model used, they are related in that they are data-driven. Consequently, issues relative to the data discussed in this thesis are central and generally applicable to each of these models.

Key words:
Safety critical, feed-forward neural networks, dependable neural networks, life cycle, good practice rules.

To my mother

Acknowledgements

I would like to thank all Lloyd's register staff that I met during this project for having shared with me a little of their experience, and especially Clive Lee (Lloyd's register project manager).

Last but not least, I would like to thank Dr Ian Nabney first, for having supervised this project, but also because, throughout this project, he made me benefit from his knowledge and experience in both software engineering and neural network technology. Finally, I would like to thank him for his precious help in writing the introductory chapters for this thesis: rephrasing the first third, and writing most of the chapter 3.

List of contents

List of figures

MAIN TEXT

APPENDIX C

Note: figures 6, 10 and 13.a have been inspired by similar figures appearing in [3], figures 1, 5 and 9 have been inspired by similar figures appearing in [4].

Foreword

- This project has been developed in collaboration with Lloyd's register (LR).

- This document is intended for anyone interested in neural network (NN) technology, and in particular for any potential developer of NN applications.
  To read this document no prior knowledge of NN technology is required, but it is assumed that the reader has studied mathematics to, at least, a A-level standard. Finally, note that experience in conventional software development or in the assessment conventional software applications may be helpful.

- Appendix A contains a list of technical definitions. In order not to overload the main text of this thesis, references to these definitions have not been systematically included. The reader is therefore invited to consult this appendix if need be.

1. BUSINESS CONTEXT

1.1 Lloyd's Register

1.1.1. Overview

Lloyd's Register (LR) is an international company, whose work can be summarised by the following statement: LR provides its customers with services intended to help them to comply with safety, quality and reliability standard criteria. LR provides commercial safety and quality assessment, and in the case that compliance requirements are fulfilled LR can award certification. Finally, LR is active in research, which allows LR to play a part in the definition and the improvement of new international safety and quality standards.

Nowadays there is an increasing appreciation of the importance of safety and quality. LR is a financially stable group that has always belonged to the leading group of classification societies. Moreover, LR is one of the pioneers in the domain[1], and hence is now highly renowned.

The range of sectors within which LR operates is considerable, and while its main sector of activity remains the marine, LR is present in numerous others. For instance, LR has clients in the chemical, oil, transport, and software systems industries.

LR is arranged in 3 main divisions and 1 group:

- the ship division: which, in 1994, represented 57% of LR income,
- the industrial division: 21% of LR income in 1994,
- the offshore division: 9% of LR income in 1994,
- the Engineering services group: this group can be seen primarily as an internal service provider for LR, since most of the services it provides are intended for the other divisions of LR. That said, it also deals directly with external clients.

1.1.2. Engineering service group

This group, which is the initiator of the project, has two types of departments:

- the plan appraisal departments, which operate in various fields (such as piping, machinery design) and provide expertise in these fields.
- the assessment departments, which provide assessment in the same fields as the Plan appraisal departments.

SIRM (System Integrity and Risk Management) which has commissioned the project, belongs to the latter group of departments, and is itself arranged in 2 groups: "system integrity group" and "risk management group". These have a common theme the safety of software systems.

---

1 LR has provided certification services since 1760.

Most of the staff concerned with the project, belong to the software integrity group.

The system integrity group provides 2 different commercial services: "assurance" and "certification".
Assurance means: ensuring that a system complies with certain criteria in order to answer the question "is this system appropriate for a particular use?". The quality standard criteria are principally concerned with process rather than product: i.e. checking that appropriate methods have been used to develop the software.

One of the main differences between the certification and the assurance processes, is that the certification process may lead to the award of a certificate, whereas the assurance process does not. Nevertheless, assurance and certification can both cover any phase of the software life cycle, and are available to any user or supplier of software. Usually these services are requested when the software forms part of a high-integrity system.

1.2. Validation and verification of embedded neural system

1.2.1. Why neural networks?

Commercial neural network (NN) applications have been in existence for more than 5-10 years, but now the technology is moving from research to products in more safety related areas.

Their power and their efficiency are becoming more widely known, and in a few years, safety related systems including software based on neural network technology will probably start to be widely developed.

For LR, this means a potential new market. Indeed, neural network technology has already been applied in several fields relevant for LR's business[2], and many others will surely be found in the future.

Furthermore, in the case that LR were to be the first to propose the assessment of safety related systems including neural network technology, that would be an important advantage over competitors: that is the advantage of pro-activeness.

NNs are a new way of viewing software. Indeed, the classical view of software development is mainly based on the implementation of an algorithm. This classical approach is not valid with NNs, since using NN consists of building a model that is based on data. For instance, this means an absence of rigorous specification to describe the NN functionality. But above all, this means for LR, that most of the usual software assessment methods no longer cover all aspects of system safety.

Hence, LR would like to have available methods to assess safety related systems including NN technology. As NN technology is currently not mastered by LR, Aston University Neural Computing Research Group is to develop such methods in this project.

This project is important, since its results will be used on the international market. The whole project is planned to last 3 years. The purpose of the MSc project was, first, to provide a first set of guidelines to assess NN applications, and secondly, to raise the main research issues that will have to be solved to complete the overall project. This phase of the project has lasted one year.

1.2.2. Project definition

A safety critical application must be an example of rigour, and "nearly right" is not acceptable, since, by definition, if there is any problem with this kind of application, the

2 for instance, neural networks are used in medical instrumentation, and currently, pilot studies are being carried out in the naval vessel and car motor ignition fields.

consequences may be catastrophic. Hence, in safety critical systems, it is important to be able to show that the probability of failure is very low.

If a serious problem occurred with a safety critical application certified by LR, and this problem occurred in conditions for which the application had been certified, then LR's credibility would be affected and LR could be liable.

However, as NNs are probabilistic models, a set of questions arises:

- Are there any limitations, or specific context for their use?
- What are the phases of the system development life cycle that cannot be taken into account by using conventional software assessment?
- What are the key safety properties that need to be assessed?

Consequently, the main task of this MSc project was to identify the key safety properties, to find a set of criteria to assess them, and thus provide assessors with a tool, in the form of guidelines. These guidelines must have a very practical aspect[3]. They must enable LR assessors, first, to answer the crucial question "is a system safe for the use for which it has been designed?", and secondly, to understand what are, in terms of safety, the underlying safety requirements. Moreover, they must be relevant for the typical scale of LR projects. Finally, the assessment of systems is based on a top-down view, and the assessment of a NN component should fit into that process. Consequently, these guidelines must be applicable in a top-down approach[4].
The last stage of the MSc project has consisted of testing these guidelines through two case studies, in order to correct and refine them.

In order to do that, assessment techniques used by LR in conventional software assessment, and neural NN knowledge provided by Aston University, have been used in synergy.

---

3 since they are intended for software assessors and not neural network specialists, implementors, nor system designers.
4 Some procedures such as HAZOPS are bottom-up processes but these tend to be used during design and safety case development. As such, they fall outside the scope of the initial MSc project.

1.3. Summary

For this MSc project, a very practical approach was needed. Thus, technical problems did not always need to be solved, but, on the other hand, the key questions had to be raised.

Lastly, the guidelines must contain principles of assessment rather than narrowly applicable prescriptive rules.

2. Introduction

This project takes place in the more general framework of safety related systems. In this deliverable, we shall suppose that the reader is familiar with the terminology and concepts defined in that framework. The reference document we have used is: "draft IEC 1508: Functional safety: safety related systems" [1].

Even if Neural Networks (NNs) are implemented in software on conventional computers, NNs correspond to a very different way of viewing computer programs, so it is not obvious that the classical methods used to develop and assess software are applicable to them. Therefore, each phase of the life cycle (specification, design, implementation, validation, operation and maintenance)[5] may have peculiarities compared to the ones defined with classical software applications, and consequently, assessment techniques also differ. A NN application can be naturally inserted in the "IEC framework" as the software part of an E/E/PES[6] safety related system.

2.1. Objectives

This document identifies the technical issues relevant to the assessment of NN applications. It addresses the question of where safety problems arise and deals with issues such as:

- what aspects of performance assessment are not understood?
- under what circumstances are the current assessment techniques (such as ISO 9001 and the MISRA guidelines) not applicable?

This document should be read as a companion to the Assessment Guidelines (deliverable D3). It contains the following sections:

- an overview of the differences between neural computing and conventional programming;

[5] A more precise definition is given in [1].
[6] E/E/PES: Electrical/Electronic/Programmable Electronics System, the exact definition is given in [1] part 4.

- a description of neural computing and its relationship with statistical pattern recognition;
- the key issues that should be addressed in a neural computing applications. This section explains the reasoning behind many of the detailed points in the Assessment Guidelines.

2.2. Scope

This document is necessarily limited in its scope. We only address certain sorts of problem and certain NN architectures. It is assumed that the applications of most relevance to LR's business involve the use of a model of a dynamical system (such as an engine) for the purposes of control or monitoring. Such applications are of *classification* or *regression* type, and often *time series* data is used to develop such models. Usually, *feedforward* NN architectures (such as *MLP* and *RBF* networks) are applied to such problems. At an approximate estimate, about 70-80% of applications in the process modelling and control industries fall within the scope of this document.

Aspects of neural computing that are not addressed in this document include:

- model development for non-stationary time series. This raises difficult issues for assessment which have not yet been solved. The problem is discussed further in section 4.2.3.
- neural controllers. Currently, in live applications, NNs are more often used as a system model within a more conventional model based control framework than as a controller in their own right. There are many difficult issues concerned with the training and validation of neural controllers that there is insufficient time to address in the current scope[7].
- recurrent networks. A network architecture used in time series analysis. Again, there are a lot of open research questions involved in the validation of such networks.

[7] Indeed, stability results are only available for neural controllers with some special cases of non-linear systems.

- unsupervised networks (such as the Kohonen network). The principal use of such networks is for clustering and visualisation, and they are therefore rarely used in applications of relevance to LR's business.

Future work on the project could cover this material: the first and third areas are of most potential relevance.

3. Neural Computing and Inductive Programming

3.1. Data Modelling

Neural computing is a form of *inductive programming*: a task is performed by a general model which is trained using data that represents the task. Such an approach is particularly appropriate when applied to problems that involve modelling complex systems. We can compare this with a conventional approach by considering the problem of developing a system model for a marine engine.

A conventional approach would involve determining the physical processes involved in the engine, analysing these to generate mathematical equations that represent the system, and then programming some software to simulate these equations (and their solution). In principle, this method could be used without the use of an actual engine.

An inductive approach involves gathering data from an existing engine, and training a model (a NN, for example) to reproduce the same relationships as are in the data and determine a good approximation to the underlying function that has generated the data. The training process consists of adjusting some variable parameters in the model using the data. In principle, this method involves no software development, as the software to run and train the model is independent
of the application and data.

In practice, the distinction between the two development approaches may not be so clear cut. When modelling any complex real world system, some data from the actual system is nearly always required, if only for model calibration and validation. Most of the issues discussed in this document are relevant wherever real world data is used for such purposes, as can be seen in the next section. Equally, neural computing may only represent part of the solution to a problem.

It is clear that neural computing represents a very different approach to the conventional view of software development where an algorithmic solution can be specified in advance of writing the software. Because the performance and precision of the model cannot be determined in advance, the use of neural computing is confined to applications where efficient algorithmic solutions are impossible or impractical. Such applications are typically complex, poorly understood, and imprecise.

Understanding speech, reading hand-written documents, and modelling and controlling non-linear systems are all areas where neural computing and other statistical techniques outperform algorithmic methods.

Although there is little or no software development in a neural computing application, there are other important tasks to be carried out. These are discussed more fully later in the document, but the key tasks are:

- data collection. Adequate quantities of relevant data are essential.
- data pre-processing. It is rare that the best performance is achieved when the data is presented in its raw form.
- network training. This has to be carefully monitored to ensure that a good solution is reached.
- performance assessment. The key question is not how well the network performs on the data it was trained on, but how well it generalises to unseen data.

Another consequence of the fact that NN software is the same for different applications is that it is the parameters in the model that are the only thing which is specific to the problem. Thus these parameters represent the problem-specific 'algorithmic processing'. The interpretation of these parameters is considerably more difficult and less precise than the interpretation of the algorithmic source code produced by a conventional approach to software. This implies that the assessment of such systems is necessarily statistical in nature. This is not necessarily a drawback, as safety cases typically are written in terms of probability of failure.

3.2. Life cycle model

Most phases of a neural computing application development life cycle will be familiar to people who know about life cycles for conventional software development. The key differences arise for three reasons:

- a precise functional specification (i.e. as for conventional software) is not possible at the start of development;
- the use of data means that extra tasks must be carried out;

- development is necessarily iterative.

Although there is no definitive life cycle (just as for conventional software development), the following is an approach which has been successfully used in practice:

1. Specification: What are the aims? How can you measure success?
2. Data collection: data sampled from the system to be modelled, cost measures, prior information (for example, how smooth the function should be, operational constraints on variables).
3. Data analysis: visualisation for understanding the data, feature extraction, missing and corrupt data, pre-processing.
4. Model development: model design is based on prior knowledge and data analysis. A range of models should be trained: simple models as benchmarks, more complex models to attempt to capture more features of the problem and to improve accuracy.
5. Model validation: check the compliance with the requirements specification. Compare models with benchmarks and test model assumptions.
6. Integration: interface with other software.
7. Operation and maintenance: ongoing model validation and retraining.

4. Neural Networks and Statistical Pattern Recognition

4.1. Introduction

Three of the main reasons that explain the current emergence of NN technology are:

- Generality. NNs are universal approximators[8] that can learn a function from a set of examples. NNs have been used to solve a wide range of notoriously difficult problems.
- Speed. In operation, running a NN consists of computing a relatively small and fixed number of floating operations. (Note: this is true for feed-forward NNs, which are the only NNs we shall consider in that project).
- Accuracy. The high performance (in terms of accuracy) possible with a well designed neural computing application can make it the only viable solution in safety critical contexts.

In this document we shall consider only *supervised* problems: in such problems each data point consists of a set of input values from which a set of target output values are to be predicted. For example, when developing a model of a distillation process, the inputs could be temperature measurements and collected volumes, and the outputs could be the proportions of chemical compounds in the distillate. To train the NN, both inputs and outputs must be available in the data. In operation, only the inputs need to be measured; the role of the NN is to predict the (unknown) values of the outputs.

There are two major classes of supervised application:

- classification. The target values belong to a small set of possible classes. Each data point belongs to a given class. The task is to predict the class of an example (or more generally, to predict the probability that the example belongs to a given

[8] A feedforward NN is able to model with arbitrary accuracy any continuous function defined on a finite domain.

class). For example, in a condition monitoring application, the task might be to determine whether a bearing is faulty based on vibration sensor measurements.

- regression. The target values belong to a set of continuous numeric values. The task is to predict the continuous numeric values associated with an example. An example is the identification of a state model for an engine, where the states are physical measurements on a continuous scale.[9]

The data sample used to train a statistical model is usually one of two types:

- static data. Each data point is sampled independently of the other data points and is considered to be uncorrelated with the rest of the data.
- time series. Time series data consists of parameters measured over time. In this case, we expect that data later in the time series will be dependent on data measured earlier: i.e. there is an ordering to the data that must be respected.

In order to implement these applications, many different architectures are available, the two main ones being the multi-layer perceptron (MLP) and the radial basis function networks (RBF). Other architectures exist[10] but, we shall not consider them in this document.

Some example applications:

- TOKAMAK plasma control
- Steel blast furnace control (Kobe Steel works, Japan): a hybrid NN/knowledge based system
- Microwave oven control (Sharp)
- EEG Analysis to determine sleep states (Oxford Instruments)

Most NN applications are implemented in software, yet successful hardware implementations of NN do exist. The main advantage of hardware implementations is the potential increase of the performance in terms of speed, since for instance, they allow the implementation to benefit from the intrinsic parallel structure of the NNs. We will not specifically address hardware implementations further in this document,

---

[9] A NN used as a controller represents a special type of regression problem where the desired outputs are not in the data gathered.

[10] For instance: unsupervised techniques (e.g. Kohonen nets) or recurrent NNs.

since the problems they pose are either common to those set by software implementations of NN, or belong to the usual problems related to hardware applications and hence are not specific to the NN context.

4.2. Problem types

4.2.1. Regression problems

In a regression task the aim is to learn a function that maps inputs to a set of continuous numeric values. The data set consists of pairs of vectors in the form (input, target) where the target is a vector of real numbers.

A regression model is usually written in the following form:

$$\text{target} = h(\text{input}) + \varepsilon,$$

where ε is the noise and is drawn from a zero-mean probability distribution (usually chosen to be a normal distribution). With this model, the NN is trained using a sum of squares error function. For this error function, the optimal solution is for h to have the value of the conditional mean of the data at each input point. The variance of the conditional mean (that is, the value predicted by the NN) can also be determined to provide error bars around the conditional mean values (see fig. 1).

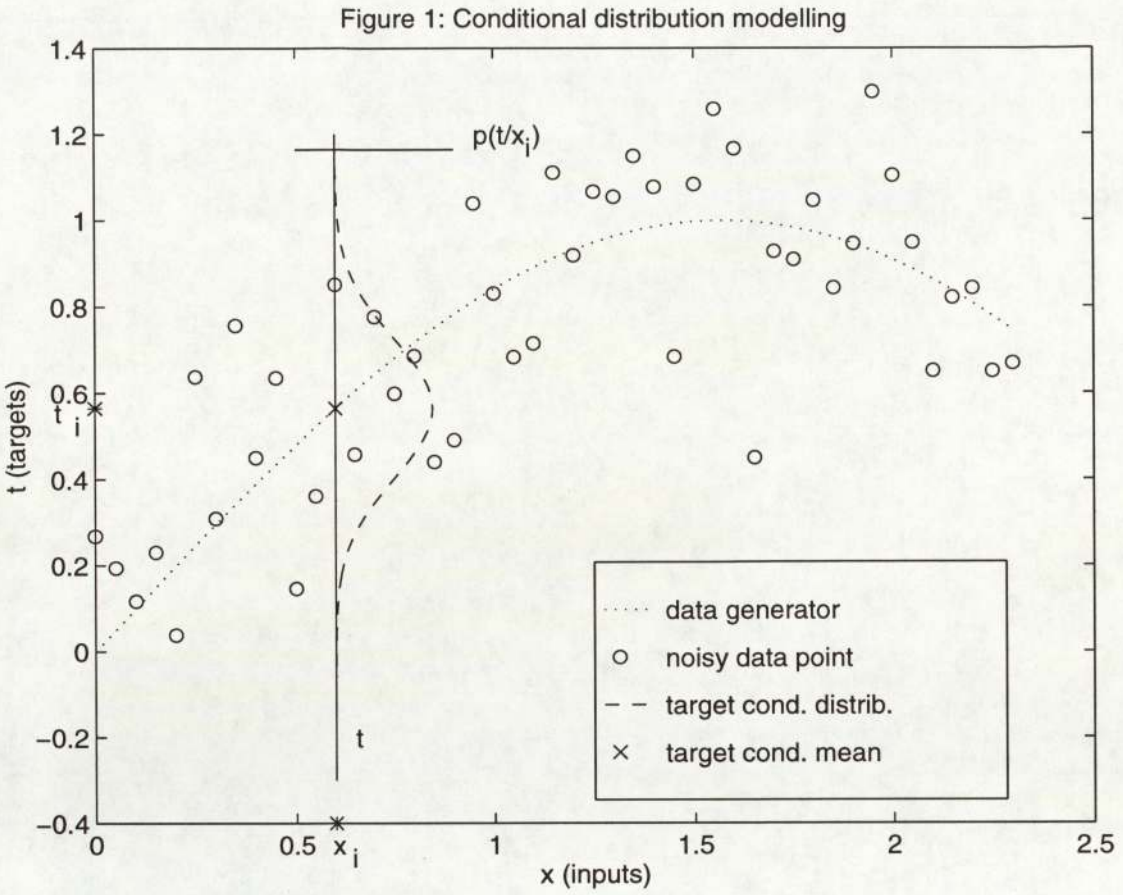

Figure 1: Illustration of the notion of conditional distribution of the target values. Given an input value (x_i), the NN predicts the corresponding target value by, in a way, computing an average, of the targets of the training data points lying in the vicinity of x_i. This average value is the conditional mean and is the value predicted by the NN for the input x_i. Note that a second set of Cartesian co-ordinates is represented. In these Cartesian co-ordinates, the conditional distribution of the targets is plotted (this distribution is conditioned by the 'position of x_i').

This conditional distribution of the targets expresses the uncertainty concerning the output value produced by the NN. Roughly speaking, this uncertainty has two main origins: the uncertainty on the weights (due to a lack of training data points in the vicinity of x_i) (see [3] pp. 399-401) and the uncertainty due to the noise on the targets. On this figure to simplify the problem we have considered the uncertainty on the weights to be negligible compared to the uncertainty due to the noise. Consequently, the conditional distribution has a Gaussian form and has a variance approximately equal to the variance of the noise (see also section 5.6.1.1).

TOKAMAK plasma control is an example of a regression problem. Here the task is the prediction of the energy confinement time (a continuous numeric value) from 4 input variables.

4.2.2. Classification problems

In a classification task the aim is to learn a function that maps inputs to a small set of classes. The data set consists of pairs of the form (input vector, class label), where the class label is a discrete value identifying the class that the point represented by the input vector belongs to.

It is often more useful to map inputs to the conditional probability of the classes. The reason for this involves decision theory. The classification produced by the NN should be such that it minimises the risk of misclassification error. It can be proved that classifying the input to the class corresponding to the largest Bayesian posterior probability[11] minimises the misclassification risk. With a suitable choice of model (e.g. involving 'softmax' outputs) and error function (cross-entropy), a minimum error solution corresponds to the Bayesian conditional probabilities.

The decision may involve a loss matrix which defines the cost associated with each type of misclassification. A typical example is that of health screening: It is less grave to diagnose a cancer when the patient is healthy than to state that the patient is healthy when he does have a cancer. The loss matrix can be then used to weight the conditional probabilities in a simple way to arrive at the optimal decision. The class with the smallest expected cost (given by the product of the conditional probability and its associated losses) is the best prediction.

Fault detection is an example of a classification problem. The task is to determine the status of a system (i.e. is it functioning normally, or is it in some malfunctioning state), given certain physical measurements from the system.

4.2.3. Time series

Time series data is mostly used for regression problems, but can be used for classification problems. Time series data captures the temporal behaviour of a system. Because of this, the order of data points is significant, as we expect there to be correlations between data gathered at different points in time. In 'static' data sets, we

[11] See appendix A; definition of 'Bayes' theorem'.

assume that the data points are independent. A typical time series problem consists of predicting future states of a system given past data: this can be done precisely because of the time-based correlations. Much of the analysis of time series data is concerned with determining the time window over which the correlations act for different variables, and the form of the functions that relate different data points.

A time series is said to be *stationary* if the underlying generator of the time series remains constant. Obviously, over long periods of time, this assumption may break down. For example, wear of physical components means that engine response changes over time. In this document we shall not consider the development of models for non-stationary time series, since this involves on-line training methods, where the model parameters are adjusted *after* installation. This raises difficulties for assessment which are not yet solved. However, we will address the issue of detection of non-stationarity, so that a system may alert the user to the fact that the assumptions under which it was trained are no longer valid.

4.3. Feed-forward NN architectures[12]: MLP, RBF

4.3.1. Multi-Layer Perceptron networks (MLP)

The Multi-Layer Perceptron (MLP) is the most commonly used NN in applications. Most MLPs consist of three layers of nodes: the input nodes (which just have the input data values), the hidden nodes and the output nodes. Each hidden node computes a linear combination of the input values and passes this through a non-linear activation function. The output nodes compute a linear combination of the hidden node values and pass the values through an activation function which depends on the type of problem (typically the identity for a regression problem, and 'softmax' for a classification problem).

Advantages:

- Universality. With sufficient hidden nodes, such a network can approximate any continuous function on a bounded domain to arbitrary accuracy.

[12] An example of feed-forward NN is given in fig. 2.

- Global interpolation properties. Because of the mathematical form of the network, it tends to interpolate well between data points, even in regions of relatively low data density.

Drawbacks:

- The training process is usually computationally very demanding, and there is no assurance that the optimal solution (see section 4.4) will be found.

Figure 2

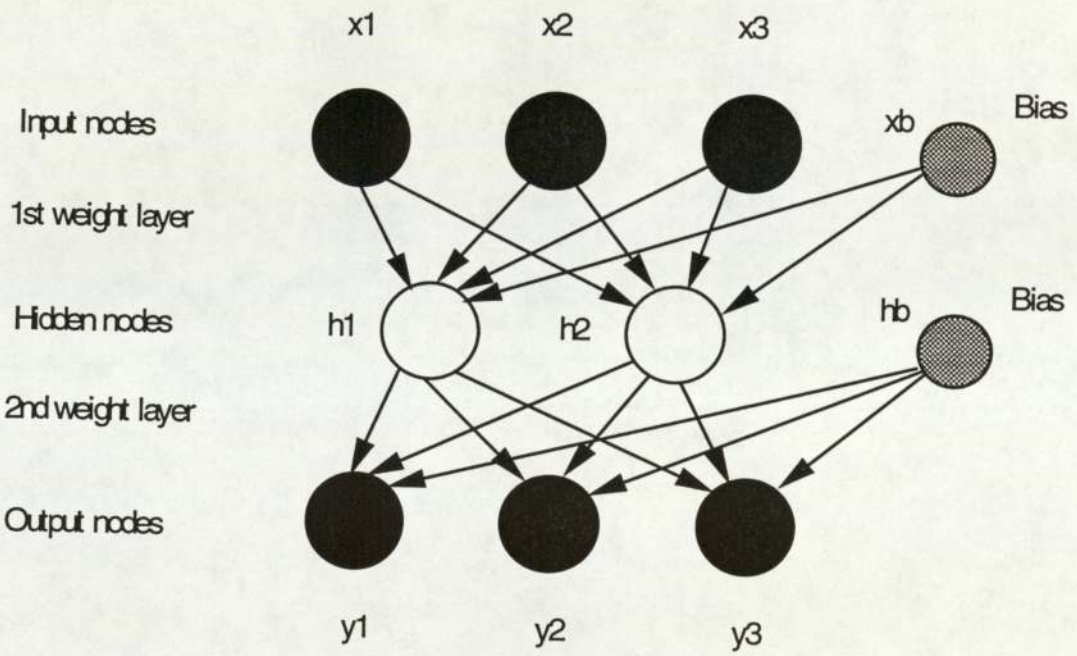

Figure 2: A feed-forward NN architecture.

4.3.2. Radial Basis Function network (RBF)

The principle underlying this form of NN is that any function can be approximated by a linear combination of localised[13] functions (see fig. 3). The name of the network comes about since the 'basis' functions chosen are radially symmetric so that their value depends only on the distance of a data point from the 'centre' or 'position' of the function.

[13] This means that the function value is zero (or near zero) except in a small region.

Figure 3

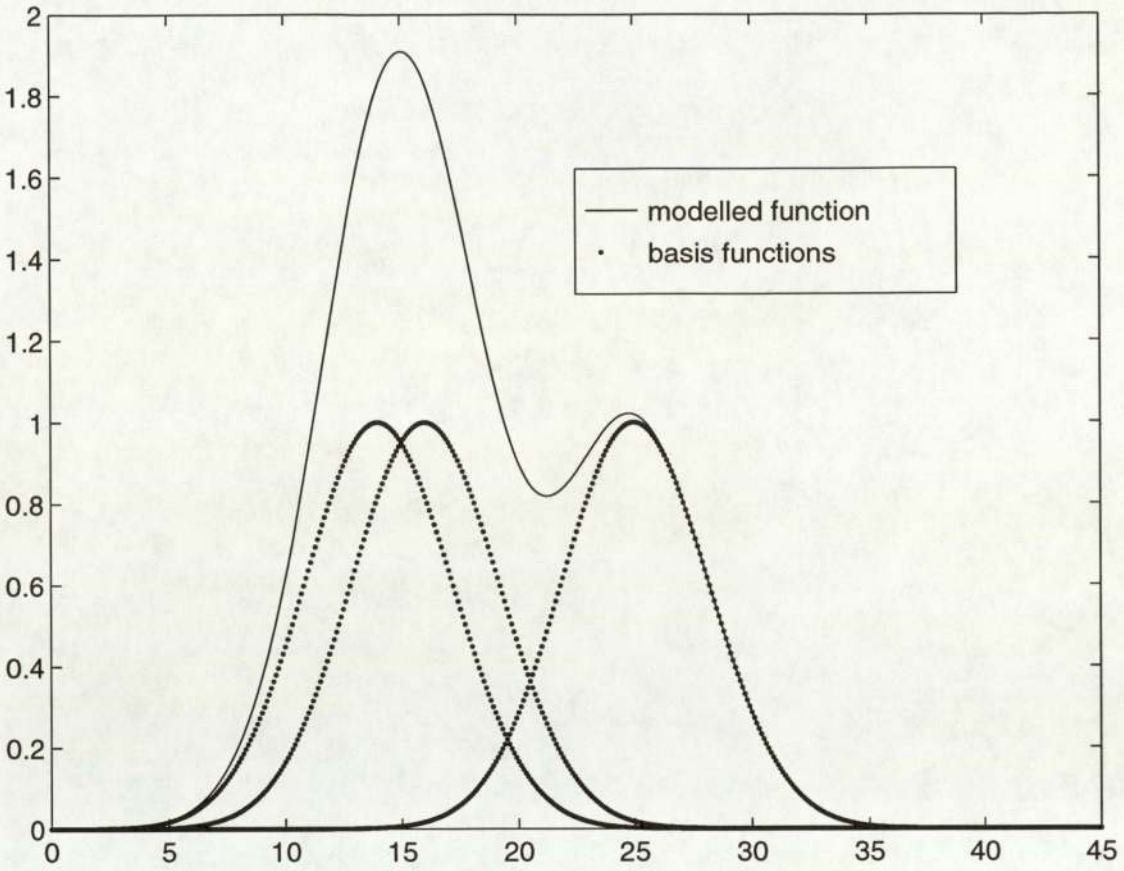

Figure 3: illustration of the idea of linear combination of basis functions to approximate a function. In this figure the function is modelled by a linear combination of three basis functions (the weighting coefficients of the linear combination are all equal to 1).

RBFs also use one layer of hidden nodes. The usual training process for a RBF consists of two steps: first, the parameters of the basis functions (i.e. position and width) are determined often by some clustering process, and secondly the weights of the second layer are determined by a matrix inversion.

Advantages:

- Universality. With sufficient hidden nodes, such a network can approximate any continuous function on a bounded domain to arbitrary accuracy.
- This two stage procedure is much faster than trying to optimise all the weight at once by using costly iterative non-linear methods.
- The basis function centres have a statistical interpretation: they model the input data density

Drawbacks:

- Owing to the localised aspect of the basis functions, RBFs are very sensitive to the problems related to the curse of the dimensionality (see section 4.7), and thus the number of basis functions needed may increase exponentially with the number of input variables. This, of course, results in greatly increased computational effort required.
- The localised response means that the RBF may not interpolate well to regions of low data density. However, this problem can be detected, as the response of all the basis functions will be low in such circumstances. This can be used to alert the rest of the system that the response at this data point is unreliable, given the data that the network was trained on.

Note: It is also possible to use non-local basis functions in RBFs with good practical results (see [18]).

4.4. Learning and generalisation

4.4.1. Principles

Learning for a NN means adjusting parameters (the weights) to approximate an unknown function, based on a data set sampled from that function. As we only consider supervised learning, this data is in the form of pairs of vectors (input value, target value).

The weights are adjusted during the training process by minimising an error function. This error function is a global measure of the discrepancy between the target values and the values predicted by the NN.

One of the main difficulties in the training process stems from the fact that the error function is often very complex (since it depends on the weights in a highly non-linear way). Thus, finding the minimum of this error function is not an easy problem. To find such a minimum, optimisation algorithms are used. Current algorithms are mainly based on the fact that the global minimum of a function is a point where all the derivatives of the function are equal to zero. However, having zero derivatives at a point is a necessary but not sufficient condition to ensure that this point is the global minimum of a function. There may be numerous points where the derivatives are equal to zero: these points can be local minima (i.e. at this point the function is at its

minimum value for some small region around the point), local maxima, or 'saddle points' (which are 'flat' regions but are neither maxima or minima).

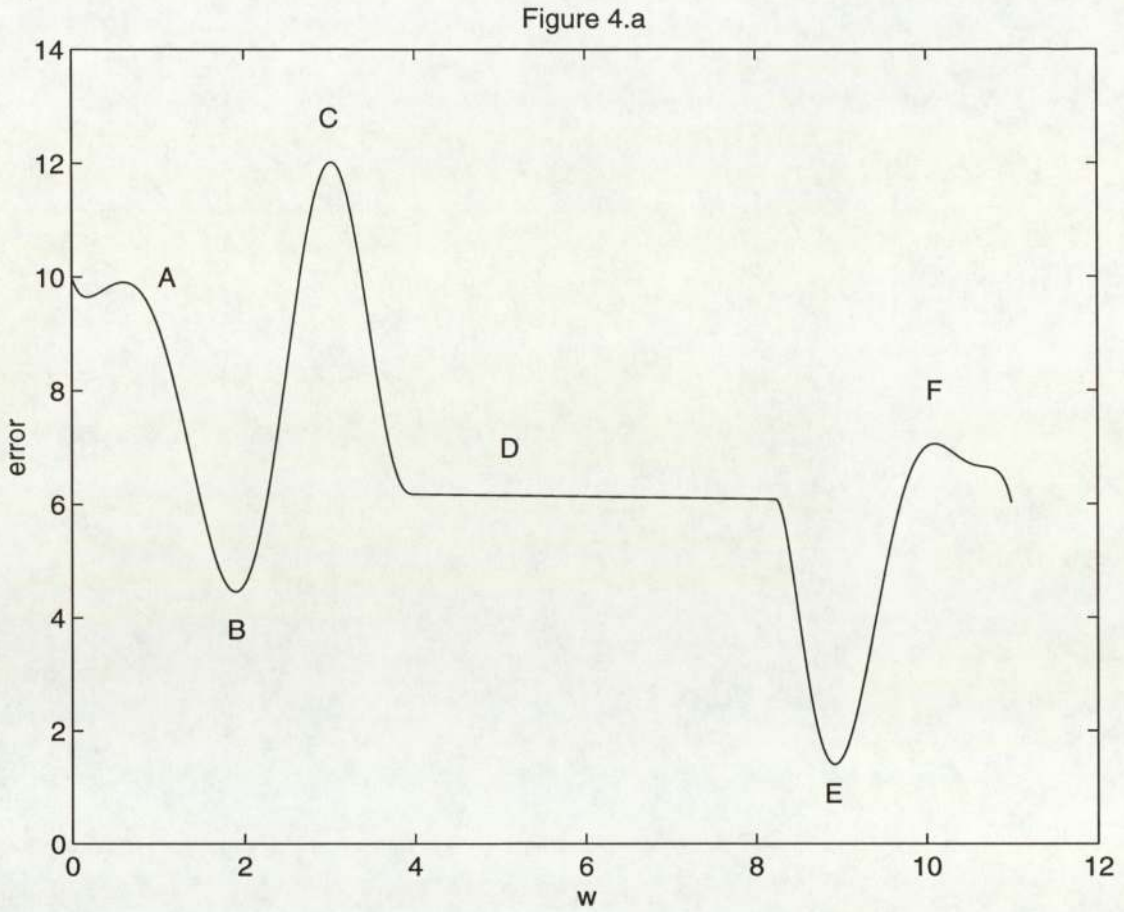

Figure 4.a: Illustration of the difference between local and global minimum: *B* is a local minimum, *C* is the global maximum, *D* is a very flat region, *E* is the global minimum, *F* is a local maximum. All these points are characterised by a zero or near zero (e.g. *D*) derivative. *A* can be, for instance, the starting position at the beginning of the training process. Typically in this case, most algorithms would be trapped in the local minimum *B*, and would ignore the global minimum *E*.

Figure 4.b

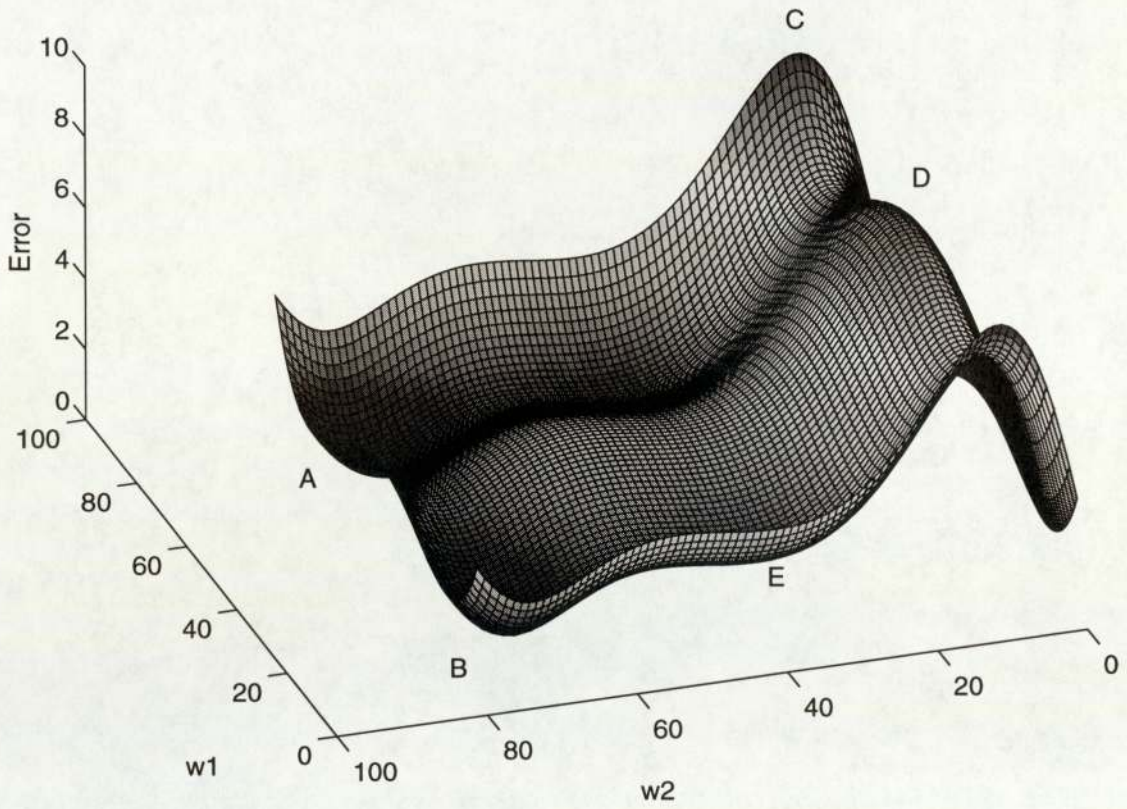

Figure 4.b: This figure is very similar to figure 4.a, but the weight space is a two dimensional space. Note, that for a neural network, we can typically expect spaces with a dimensionality greater than 20. In this figure *A* is the global minimum, *B* is a 'good ' local minimum, *C* is the global maximum, *D* a local maximum, and *E* is a local minimum (probably a bad local minimum).

Being 'trapped' in a *bad* local minimum implies that the solution produced by the NN is a sub-optimal one, which may result in non-compliance with the specification.

One of the main difficulties of the training process is to avoid being trapped in such bad local minima and to find the global minimum or at least a good local minimum. Most optimisation algorithms find the local minimum nearest to the point in parameter space they start[14]. While, none of them is better at finding global minima than all the others, there are some which are more efficient in their search for minima. For example, algorithms that make use of higher order gradient information (such as quasi-Newton and conjugate gradient algorithms) are usually much more efficient than those that do not (such as gradient descent, often called 'back-propagation'[15]).

---

[14] Although this picture is a bit simplistic, it gives a good idea of what happens.

[15] This terminology is improper, for further information see appendix A; definition of 'back-propagation'.

Another major problem is that the data the NN is provided with, is corrupted by noise (e.g. owing to the imprecision of measuring instruments that have provided the data). Hence, the aim of the training process is to avoid the NN learning this noise. Rather than merely storing the training data (which is noisy) the NN is supposed to learn the main structure of the underlying function generating the data.

So, if a NN has learned the training data and fits perfectly with it, this NN is said to *overfit* the training data. Generally overfitting leads to very poor generalisation capability and therefore this may result in non-compliance with the specification. This problem can be detected by using the error on an independent test set (see fig. 5 and fig. 6).

Figure 5.a

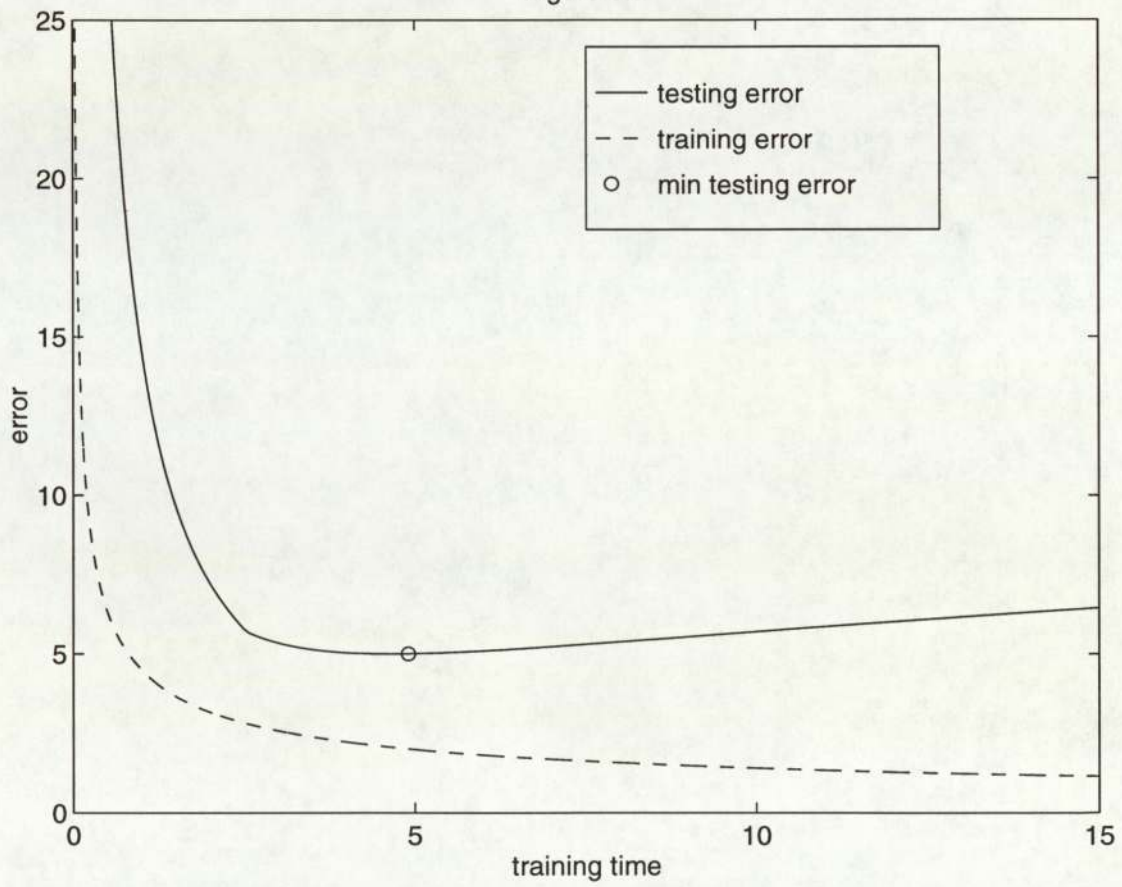

Figure 5.a: Illustration of the difference between the training error curve and testing error curve as a function of the training time. The training error is a monotonically decreasing function of the training time. When the testing error starts to increase, the NN starts to learn the noise in the data and so starts to overfit the training data. Note that with real NN applications stochastic aspects intervene, and hence the curves obtained look more chaotic (see [4] fig. 22).

Figure 5.b

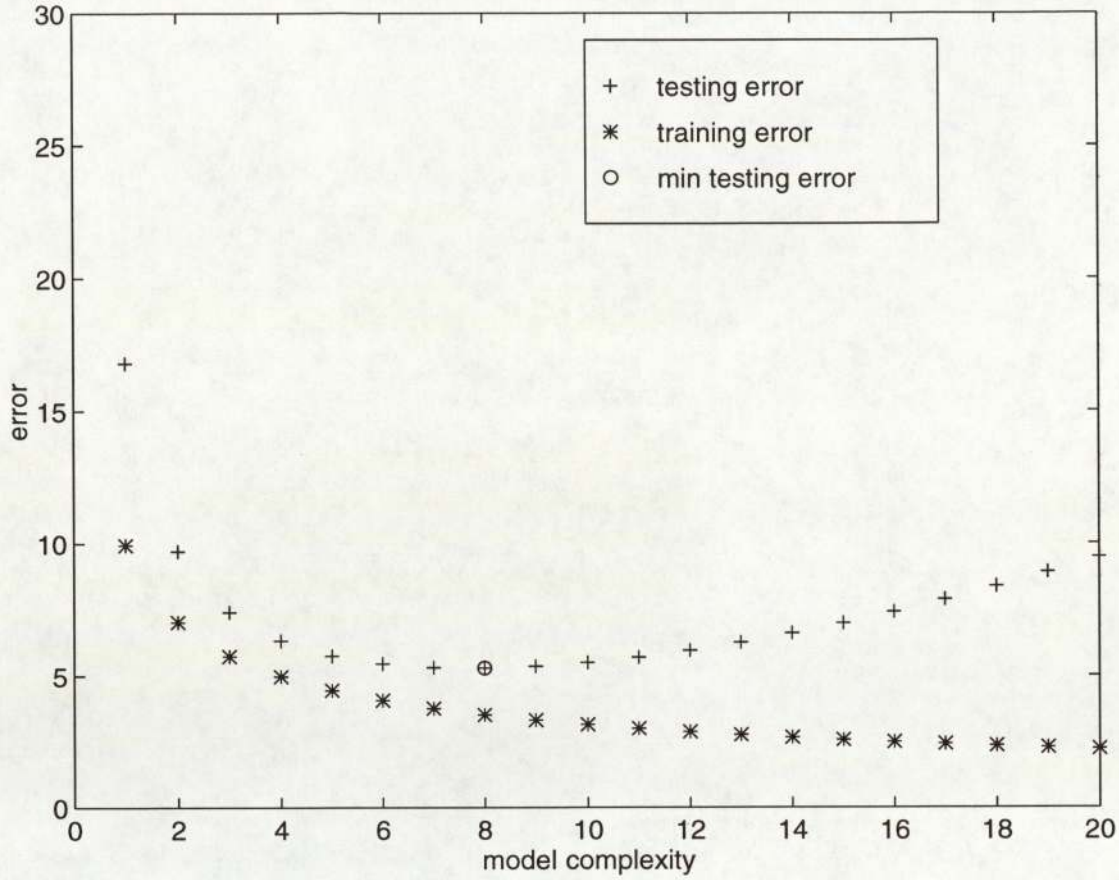

Figure 5.b: Given a problem, this figure gives an idea of the type of relationship there is between the training error, the testing error and the complexity of the models trained on this problem. The more complex the model is, the more likely it is to overfit the data. However, the models must be complex enough to be able to reproduce the main trends in the data.

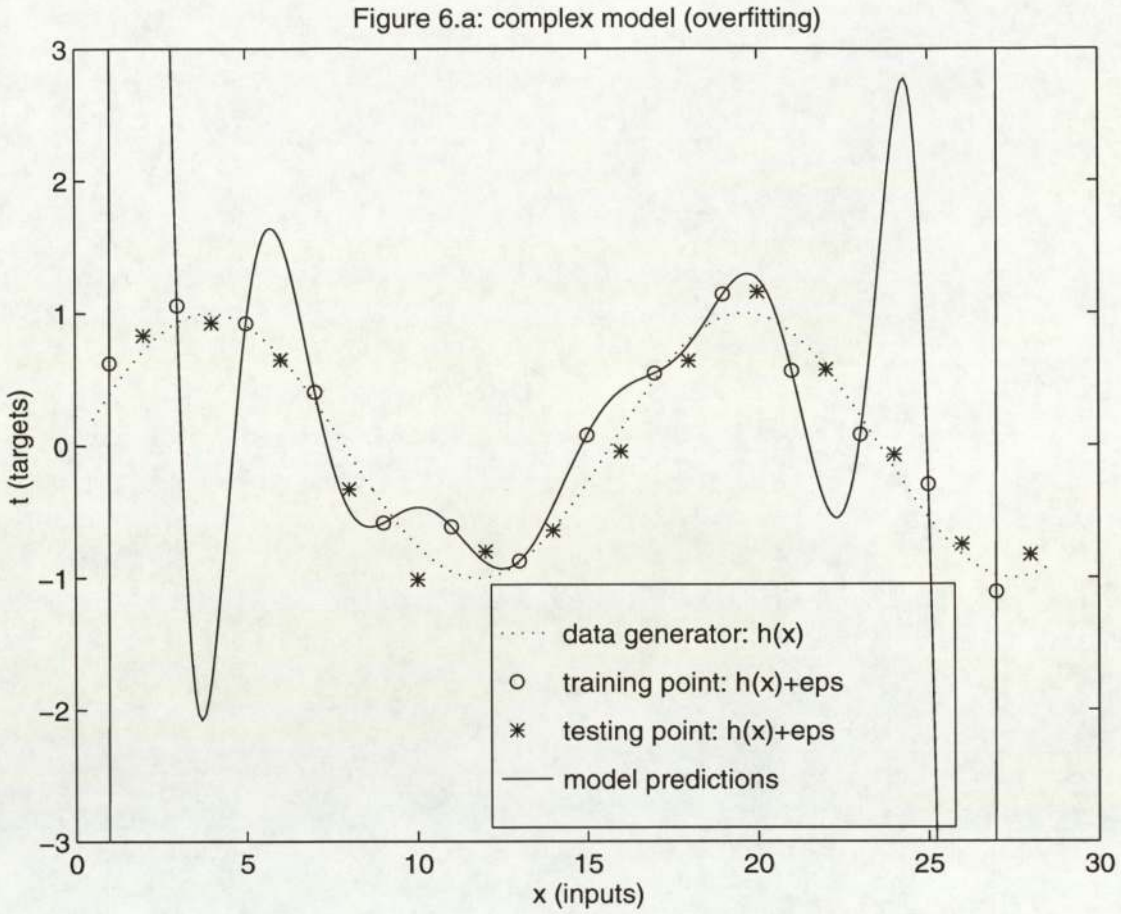

Figure 6.a: This figure illustrates the overfitting phenomenon. A complex model has been trained to fit a (comparatively) simple problem.

In this figure the training error is zero which means that the model fits perfectly the training data points. However, the testing error is high (i.e. the deviation from the testing points is sometimes very large). This is synonymous with a poor generalisation ability. The model overfits the training data.

Note: In figure 6.b the same problem is considered but the model used is significantly less complex.

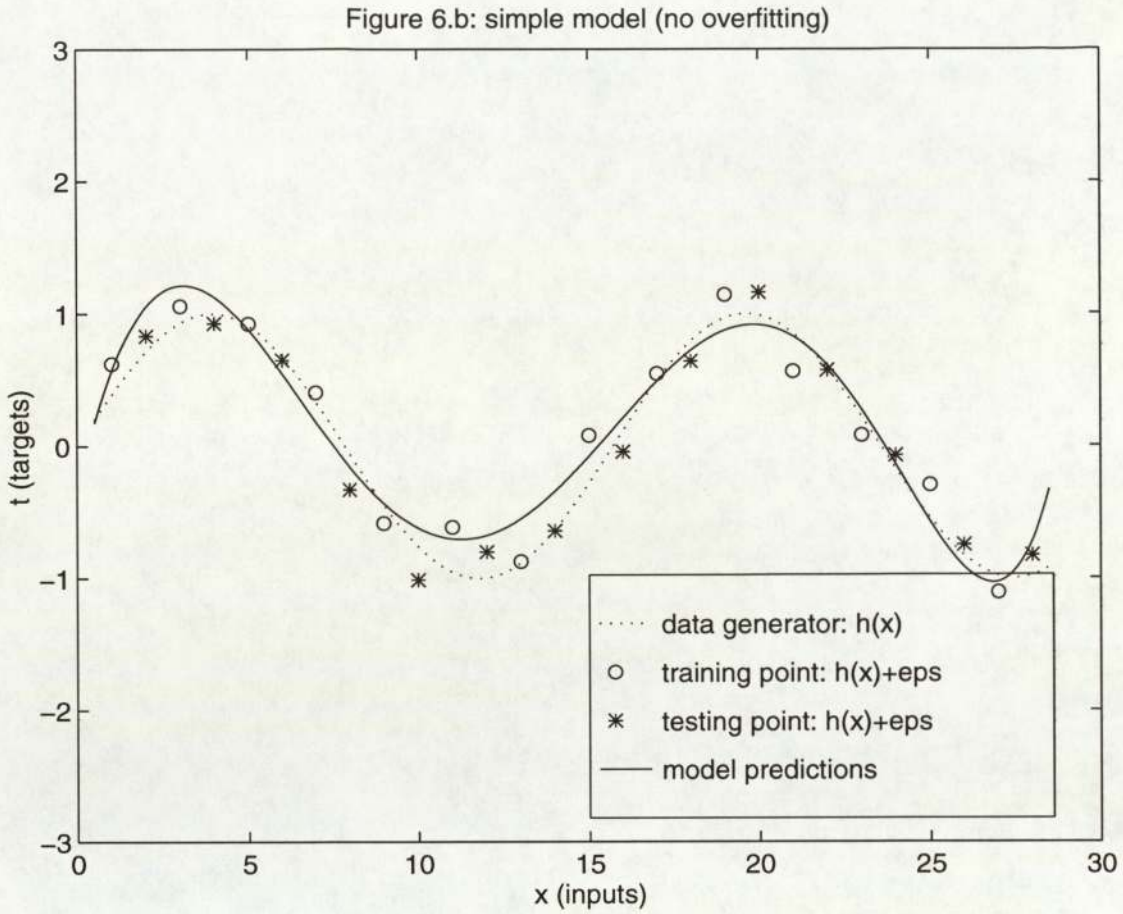

Figure 6.b: In this figure the problem to be solved is the same as in figure 6.a, but the model used is less complex and therefore less flexible.

This reduction of the model complexity reduces the risk of overfitting, which results (in this figure) in a more suitable solution. Indeed, although the results obtained on the training data are less good than for the complex model (see fig. 6.a), the testing error for this model is dramatically smaller, and is *not* significantly greater than the training error.

Finally, note that this model provides a much smoother solution than the complex model that sometimes oscillates wildly.

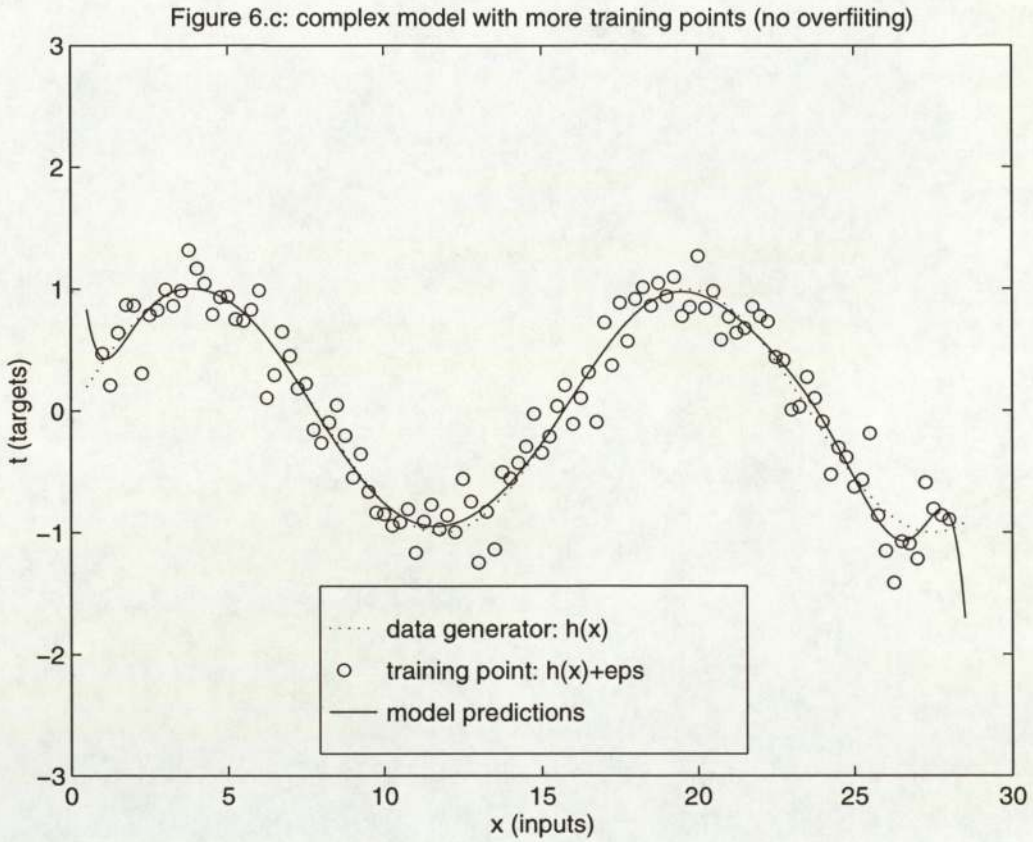

Figure 6.c: This figure shows that increasing the number of training data points enables a better control of the complexity of the model and reduces the overfitting phenomenon. With this increase in the number of training points the complex model gives a very good approximation within the interpolation area and outperforms the simple model (see fig 6.b).

For continuous inputs, although the training data is sampled at discrete points in the input space, the underlying function has values everywhere in the input domain. We usually have some prior expectation that the function will vary in a relatively smooth way between the training data points. This prior knowledge can be incorporated in the model and will improve generalisation.

These remarks lead us to say that a good model of the function should be such that the function generated by the NN passes smoothly close to the training data, and reproduces well the main trends of this data.

Lastly, let us note that a NN can provide good interpolation results, but cannot be used as a reliable extrapolation tool. So in the NN context, we shall say that generalisation ability is the capacity to give reliable interpolation predictions (see fig. 7).

Note: The opposition between interpolation and extrapolation is discussed further in the next section.

Figure 7

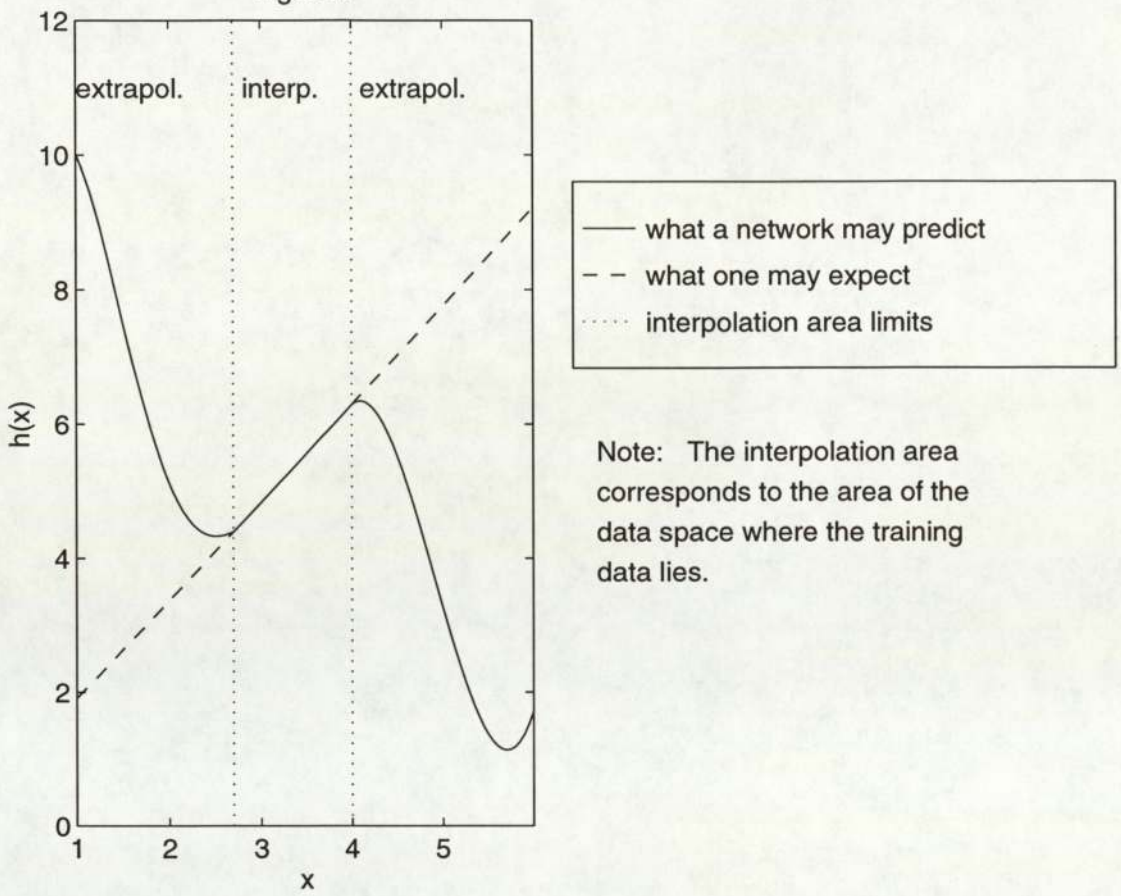

Figure 7: This figure shows the difference between the generalisation that a NN may produce and the generalisation one could expect. Both are comparable within the interpolation domain (area where the training data lies) but may be dramatically different outside the interpolation domain (extrapolation domain). Note that for this figure the data generator is a linear function. Finally, note that this figure is a bit simplistic since it considers a sharp boundary between the interpolation and extrapolation areas. This problem is discussed further in section 4.4.2.

4.4.2. Interpolation/extrapolation and data density

The understanding of the distinction between interpolation and extrapolation is fundamental in the NN technology context. As said in the previous section, NNs can be efficient interpolation tools but usually give very poor results when forced to extrapolate. Nevertheless, giving a precise definition of both notions is not easy.

Let us consider a simplistic example. Consider a one-dimensional input data space and two training data points P_A and P_B whose co-ordinates are respectively

(X_A,Y_A) and (X_B,Y_B), where X_A and X_B are the one-dimensional input co-ordinates of P_A and P_B, and Y_A and Y_B the values of the function to be modelled at these input points. Consider now that we fit a model 'M' on these two training data points. Then, the following definitions of interpolation and extrapolation are often given:

Consider a third point P_C whose input co-ordinate is X_C, if:

. X_C is between X_A and X_B (i.e. $X_A \leq X_C \leq X_B$ or $X_B \leq X_C \leq X_A$), then M is said to interpolate if it is used to predict the value of the function at X_C.

. X_C is outside the interval defined by X_A and X_B (i.e. $X_C < X_A$ and $X_C < X_B$, or $X_C > X_A$ and $X_C > X_B$), then M is said to extrapolate if it is used to predict the value of the function at X_C.

Despite these definitions are commonly used, they are too simplistic. For instance, if the dimensionality of the input space is greater than 1, then it is not obvious to define when a point is between two other points. Moreover, as said previously, one of the main motivations to distinguish interpolation from extrapolation is that when a NN is forced to extrapolate its predictions are often very poor, whereas when it interpolates they are generally much more accurate. The simplistic definition given above is not always consistent with this idea. For example, suppose that X_A and X_B are very far from each other, then the prediction of M may be significantly more accurate for an input point $X_{C'}$ that lies outside the interval defined by X_A and X_B but is very close to X_A or X_B than for an input point $X_{C''}$ located in the middle of this interval (see fig. 8.a).

In [24] (pp. 105-108) the authors refine a little the simplistic definitions and consider (for one dimensional input spaces only) that if an input is "outside the range of tabulated values by more than the typical spacing of tabulated points then it can be considered as an extrapolation point". Even if this definition is sensibly better it only partially defines what is extrapolation (i.e. it gives a sufficient condition but not a necessary condition) and is only applicable to one dimensional input spaces.

As this is a crucial issue in the NN context, a more precise definition has to be given. A better way to distinguish interpolation from extrapolation is probably to

consider the notion of data density, and to say that the *areas of the data space where the training data density is low are extrapolation areas whereas areas where the training data density is high are interpolation areas*[16]. Intuitively, the idea is: in the areas where the model has information (i.e. data points) about the behaviour it is supposed to adopt, the model 'knows' how to behave. On the other hand, if it has no information it behaves freely and can do anything (see fig. 7). Even if this definition remains slightly ambiguous (we have not defined what 'low' and 'high' mean), this ambiguity can probably not be completely suppressed since 'interpolation' and 'extrapolation' are continuous notions: that is given a data point P, and a model M, whatever the area of the data space considered, the influence of P on M is never nil (even if it is often negligible). Accordingly, a simple way to solve this problem could consist of changing the terminology and instead of distinguishing 'interpolation' and 'extrapolation', speaking of 'degree of interpolation'.

In order to simplify the understanding of the basic problems existing in the NN context, we will not use explicitly this notion of degree of interpolation, but rather use the classical sharp distinction between both notions, and implicitly consider that areas where the training data density is low (e.g. below a given threshold empirically chosen) are areas of extrapolation, whereas areas of high training data density are interpolation areas (see fig. 8.b and fig 8.c).

[16] This interpretation may be arguable. However, it is quite natural and well established results seem to be consistent with it (e.g. for regression problems it has been shown that the amplitude of the error bars is approximately inversely proportional to the data density of the training set).

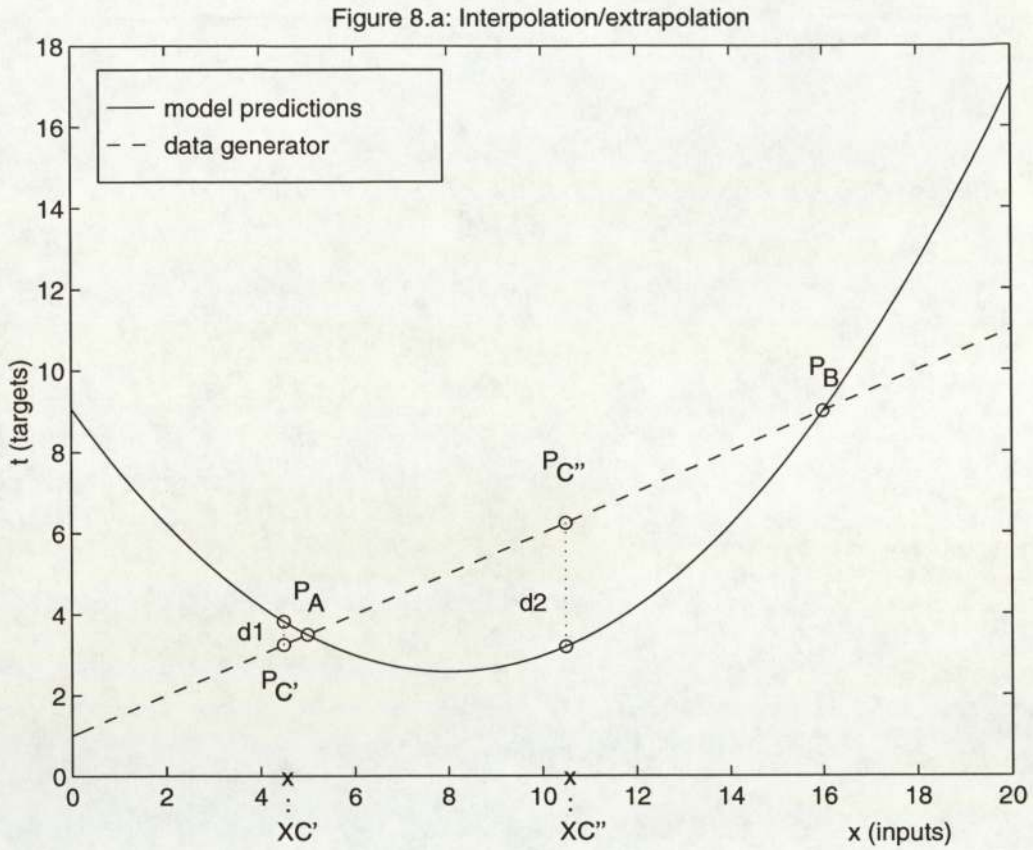

Figure 8.a: According to the accuracy expected the point $X_{C''}$ can be considered as an extrapolation point whereas $X_{C'}$ may be considered as an interpolation point. Indeed d_1 is significantly smaller than d_2.

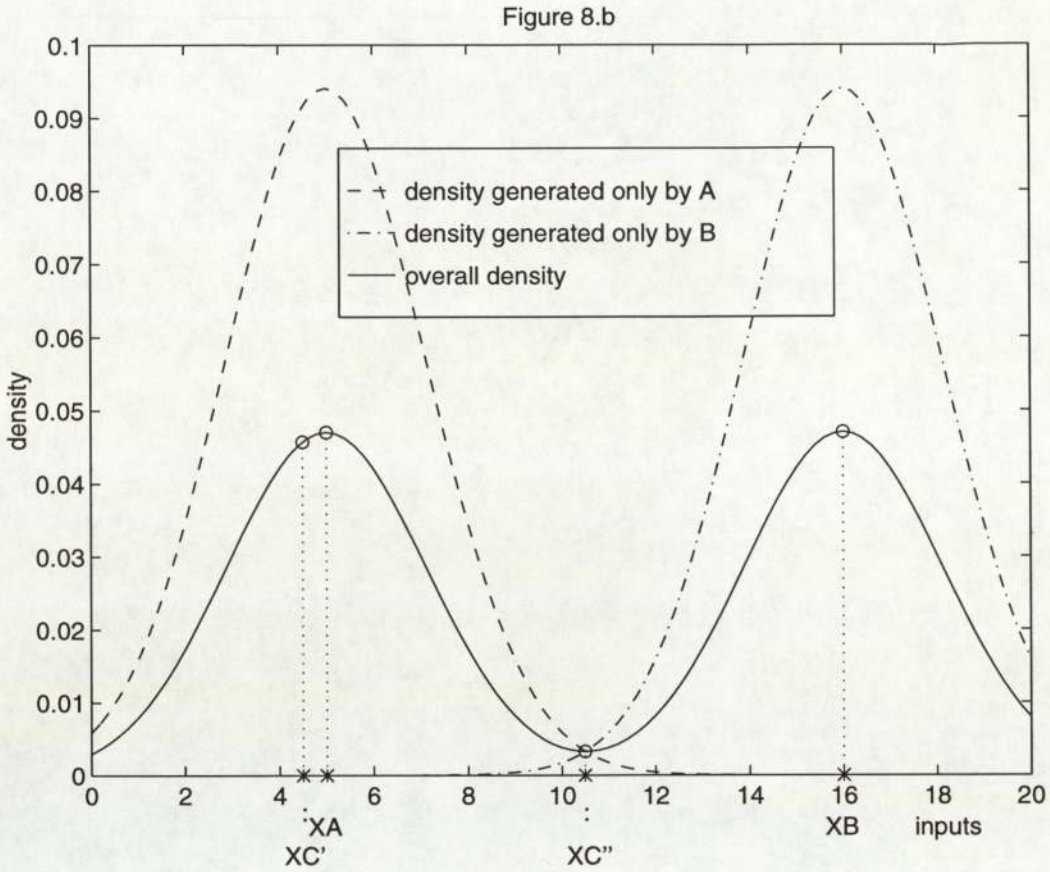

Figure 8.b: This figure corresponds to the same problem as in figure 8.a. It represents the data densities in the input space. It also illustrates what has been said for the figure 8a: The density at $X_{C''}$ (between X_A and X_B) is significantly less than the density at $X_{C'}$ (just outside the interval defined by X_A and X_B), and as said previously (see fig 8.a) the accuracy of the NN prediction is lesser.

Note that the overall density is normalised and, therefore, is a weighted sum of the densities generated by P_A and by P_B (the same weighting coefficients have been used for the densities from P_A and P_B, namely 0.5).

For completeness, note also that the density models used in fig 8.b and fig. 8.c are Gaussian mixtures with equal priors for each component and with a constant width (for these simplistic examples we have arbitrarily chosen a width equalling 3).

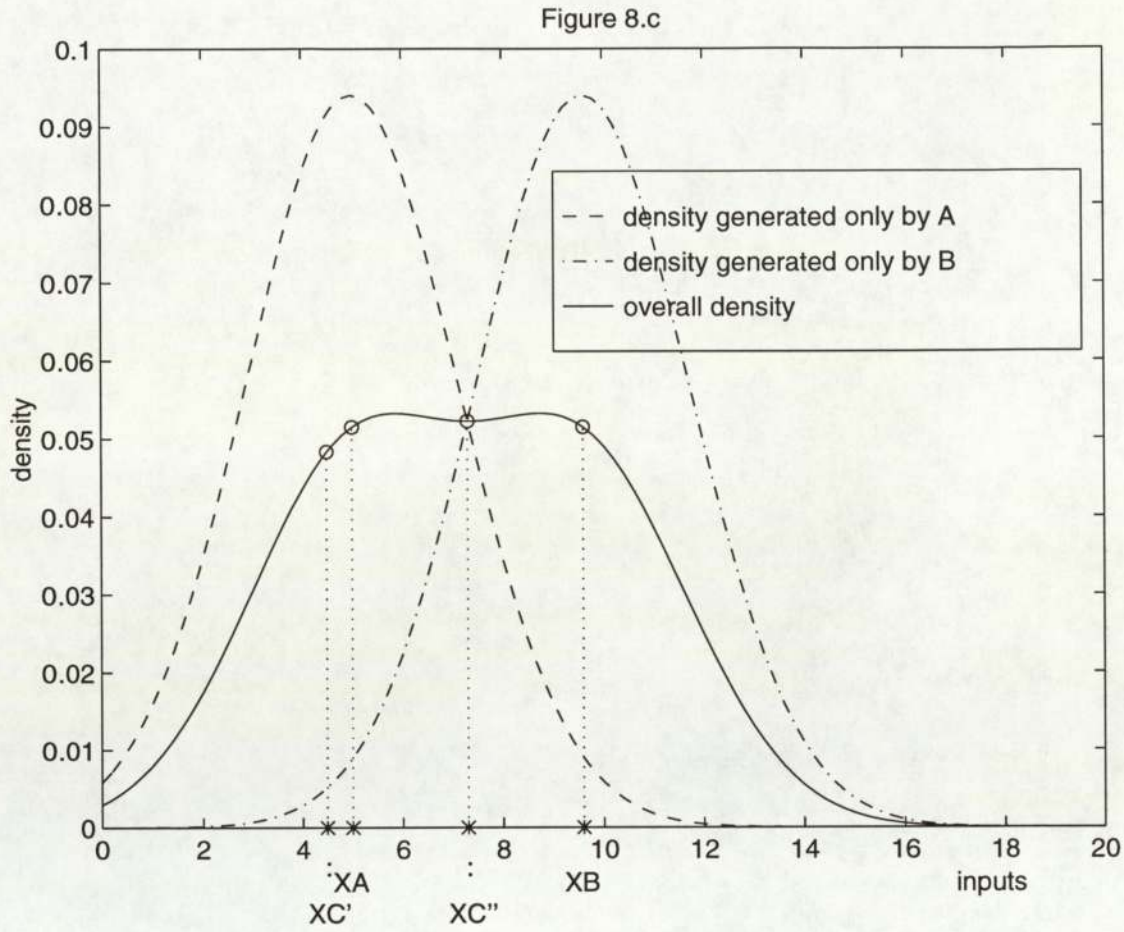

Figure 8.c: This figure is very similar to fig. 8.b but it does not correspond to the problem represented in fig. 8.a. Indeed, in this figure P_A and P_B are closer in the input space.

Although fig 8.b might suggest that for an input point X_C the distance to the nearest data point explains the input density value at X_C, this view is too simplistic and that is what figure 8.c shows. For instance, the shortest distance between $X_{C'}$ and any training input point is smaller than the shortest distance for $X_{C''}$, yet the density at $X_{C''}$ is higher than at $X_{C'}$. This is due to the fact that P_A and P_B are close enough to each other for both to contribute significantly in generating the density at $X_{C''}$. On the other hand for $X_{C'}$ only the contribution from P_A is significant (This effect partially explains the extensive use of the simplistic definitions)

4.5. Pre and Post-processing

With a NN application, each embedded NN function can roughly be divided into three units: a pre-processing unit, a NN unit (which is the actual NN component of the function[17]), and a post-processing unit. The pre-processing and post-processing units

[17] A NN unit may be composed of several NNs (e.g. committee of NN, mixture of experts).

are responsible for some of the operations the NN function is supposed to perform (see fig. 9).

Figure 9

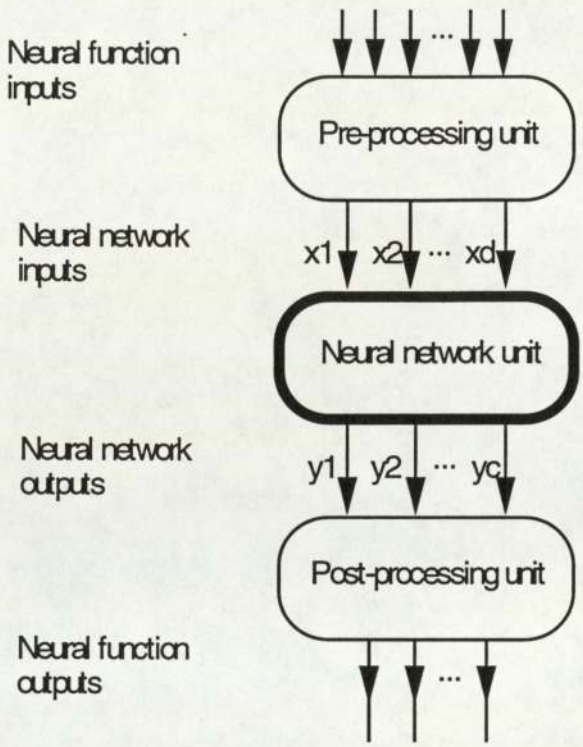

Figure 9: This figure shows the arrangement of the three units for a NN function, that is, the pre-processing unit, the NN unit, and the post-processing unit.

Note that, by virtue of its universal approximator properties, the NN unit could theoretically carry out all the operations the NN function is supposed to perform. However, the proof of this result shows only that an approximation exists, and says nothing about how to find it. Learning an accurate approximation to a function sampled at a limited number of data points is generally a hard task. This said, we have the following intuitive idea: if the number of data points is fixed, then the simpler the problem to be modelled, the greater the accuracy of the solution is likely to be. This is another example of the phenomenon noted in section 3.1: if an efficient algorithmic solution exists, then it should be used rather than learned from data.

The distribution of the initial problem between the three units (NN, pre and post-processing units) is based on this intuitive idea: the pre and post-processing units make the task of the NN unit easier and thus increase its performance. In the simplest case,

the pre and post-processing units apply a linear transformation to scale the input and output values to some desired range: for example, to make them have zero mean and unit variance.

A good example of the importance of pre-processing is the implementation of *invariant properties* of the modelled system through the pre-processing unit. For example, character recognition should be translation invariant, in that it should not be affected by a letter's position in the field of view. This can be ensured by pre-processing the data so that the character is always in the centre of the field of view. Indeed, by doing so, performances of the NN function may be significantly increased, since:

- the NN unit does not have to learn these invariant properties, since they are computed by the pre and post-processing units. So the function the NN unit has to learn is simpler, and is more likely to be learned more efficiently.
- invariant properties can be satisfied exactly instead of being approximated by the NN unit using the data set.
- if the property is an invariant transformation of the inputs then a smaller training set can be used since inputs which differ only by the invariant transformation can be considered as equivalent, and so, can be represented by a single pattern in the training set.
- generalisation is improved, since the NN is able to extrapolate to new input data if this is equivalent to training data modulo the invariant transformation.

The alternative to pre-processing the data would be to train a NN classifier to have this translation invariance property by presenting all example characters at all positions in the input field. This is a harder task which requires many more pieces of the data to be presented to the network.

As we shall see later, there are several other reasons to pre-process the input data. For instance:

- to suppress possible scaling effects by normalising the data (see section 5.3.3.2).
- to perform feature extraction in order to reduce the dimensionality of the data space (see section 4.7 and 5.3.3.2).

4.6. Evaluation of the generalisation performance

In a safety critical context, the slightest incident can have serious consequences, and hence safety critical NN applications must be dependable. Consequently, the risk of failure of the application and therefore the dependability and the quality of the outputs produced by the NN should be assessed. Because the underlying function is unknown and is being approximated from data samples, any results in this area *must* be probabilistic (or statistical) in nature. We cannot, even in principle, ensure that the outputs of the NN correspond exactly to the values of the function we are modelling. Various techniques exist to evaluate the variance of the NN predictions, and such techniques exist for the two main classes of problems: classification and regression problems

- Regression problems:

  Techniques exist that give error bars[18] on the output predicted by the NN in operation. We shall call such techniques '*in operation error bar techniques*'. These error bars are a measure of *model variance*, the likely variation of the actual target value around the value predicted by the NN.

  These techniques stem from well-founded frameworks and numerous results have been mathematically proved (an indispensable condition for safety critical applications). One of these results that is particularly relevant for the safety critical application context is the following. Note: It has been shown that, for regression problems[19], the amplitude of the error bars in certain regions of the data space is approximately inversely proportional to the data density in this area[20] (see [7] and [30]).

  These techniques can be used with the two principal NN architectures (MLP and RBF), and have already been successfully applied to various projects.

---

[18] The role of an error bar is to quantify the variance of the output value at a given input point. This is the variance of the noise. If the distribution of the noise is known, then we can determine an interval around the predicted in which the true value lies with any probability p. If p is increased, then the error interval becomes larger.

[19] We have found nothing in the literature concerning a similar result for classification problems. For further information see appendix C, section C.6.

[20] In fact, this result has been proved for Generalised Linear Regression models (RBFs belong to this class of models), and empirically shown for MLPs.

Examples of techniques: Bayesian error bars, use of an additional NN to model the error of the NN predictions.

Such error bars do not ensure that the true value of the function to be modelled lies within these error bars, but that there is a probability P that it lies within them (P is generally called the 'p-error'). For instance if the error bars are defined as two standard deviations around the NN prediction, then there is a probability of 95% that the true value of the function lies within the error bars. The main drawback of such techniques is that they provide error bars only for the input data point considered, and so, they do not provide an overall upper bound of the error for all the relevant input data space.

- For classification problems, two main classes of techniques exist:

  - Probably Approximately Correct (PAC) learning techniques[21]:

  These techniques provide stronger results, since unlike the in-operation error bars used with regression problems, they give an upper bound for the error for all the relevant data space of the considered problem. Furthermore, they can provide practical help to design a NN when an error upper bound for the whole relevant data space has been specified in advance (see [2] and [15]). For example, the number of training data points can be related to the generalisation error required.

  We now describe a typical result proved using these techniques. First, consider a two-class classification problem defined in an n-dimensional input space and that we are using a feed-forward NN architecture A_r . Let us also assume that we require the generalisation error rate (i.e. the probability of misclassification) to be no more than ε (where $0 < \varepsilon \leq 1/4$). Then there exists a value κ depending only on ε and A_r, such that if:

  1) A_r can learn at least a fraction $1-\varepsilon/2$ of κ randomly drawn training examples from a distribution Π (i.e. the error rate on the training data is no more than $\varepsilon/2$),

[21] Such techniques stem from Computational Learning Theory and VC dimension.

2) All future examples are also randomly drawn from the same distribution as the training set,

then there is a probability close to 1 that the actual generalisation error of the NN is at most ε. The value of κ is provided by the PAC learning framework and can be calculated. This means that if a specification requires a certain error rate we can achieve this (with high probability) by making sure that our training set has at least κ examples, and that our training error rate is less than half ε.

Although these results are very powerful, there are difficulties with applying them to practical problems. Firstly, these results are currently only applicable to classification problems with two target classes. Secondly, these techniques provide very pessimistic bounds for the error which is of value if a high safety integrity level is required, but can be too extreme for many real world problems. This is because they assume the worst possible sampling of training examples from the underlying function and make no assumptions about the nature of the function generating the data (which could be very 'unsmooth'). Most of the time, the number of training data samples required (κ) to guarantee the desired error upper bound is very large, and it may be extremely difficult and expensive to gather such large quantities of data.

However, research in this field is very active, and in the near future these restrictions may be overcome.

- The second class of techniques enables to put error bars over the generalisation error value obtained on the testing set[22], two cases must be considered:

. First case, the data set is large[23]:

Error bars for the true generalisation error can be given from the size of the testing set and the generalisation error obtained for this set (see [29] pp. 44-46).

[22] Unlike in-operation error bars for regression problem, but like PAC learning results, these results apply globally to the entire relevant data space.

[23] The definition of "large" is problem dependent. For example, a small set of high quality data may be preferable to a larger one whose data quality is poor.

More precisely, there is a mathematical formula that defines the standard deviation of the generalisation error estimates as a function of the size of the testing set and the generalisation error estimate obtained on this set. This estimate of the standard deviation, is used to define error bars.

. Second case, the testing data set is not large enough:

Error bars cannot be directly given. The idea is then to use resampling techniques to estimate the variance of the generalisation error from the different generalisation error estimates obtained on the different testing sets. Then, as above, given this variance estimate, error bars for the true generalisation error can be provided.

4.7. The curse of dimensionality

The main purpose of NNs is to model functions from data generated by real world systems. Modelling problems can be partially characterised by their input variables. When a value is assigned to each of these inputs, the combination of values obtained represents a point in a data space. The number of input variables is called the *dimensionality* of the data space.

The goal of training a NN is to allow it to learn a function which maps points in the data space to the output space. That is, to develop a NN that, for all input data points produces as an output a good approximation of the value of the function at that point. To learn the function, the NN is provided with a finite number of data points generated by the function. Unfortunately, the "size" of a data space depends exponentially on the dimensionality of this space. This exponential relationship is a big problem, since it implies that there is an exponential relationship between the number of data points required and the input dimension.

To illustrate this point, let us consider the problem of modelling, using a histogram, a data density. Let us suppose that each dimension of the data spans the interval [0, 1]. If the dimension of this data space is 1, then the data space is merely the interval [0, 1]; if the dimension of this space is 2 then the space is a unit square; if the dimension is 3 then the data space is a unit cube, and so on. Now suppose that the width of the histogram boxes is 0.1. If the dimension of the space is 1 then there are 10 histogram

boxes, each of which is an interval of width 0.1; if the dimension is 2 then there are 100 boxes which are square with sides of length 0.1, and so on. If we sample points from the data space and fill the boxes of the histogram with these points, the question arises "how many points are needed so that the histogram gives a good approximation of the actual density of the points in the data space ?". A reasonable answer is "at least as many points as the number of boxes of the histogram". This implies that in a 10 dimensional space at least 10^{10} points will be required.

Thus, even if the dimensionality of the space is relatively modest, the number of data points required is enormous, and obviously gets much worse as the dimensionality increases. This is a severe practical problem since real world system identification problems commonly have about 10 inputs, but to be able to provide 10^{10} examples is impractical. When a NN application is developed, the number of training examples is in general "small" (less than 10,000).

That is why, as we shall see later on, to improve the performance of a NN, the greatest care should be taken in order to reduce the dimensionality of the problem considered. Indeed, even if reducing the dimensionality causes a loss of information, it may also be synonymous with a significant reduction in the complexity of the problem to be solved, hence the NN function performance may increase.

5. Developing a NN application: Key issues

5.1. Are NN the solution?

First of all, as emphasised previously, if an efficient algorithmic solution exists that solves the problem, then it should be preferred to a data-driven approach.

Secondly, as for every project, a feasibility study is necessary. Indeed, even if theoretically NNs are universal approximators that does not mean that this theory is always applicable for real world problems. For instance, among the main limitations, there are:

- the computational requirements[24],
- the problems of data availability. This can occur when high accuracy is required, little relevant prior knowledge is available, and it is difficult to collect sufficient quantities of data samples.

Developing a NN application is not easy. One should be aware that embedding a NN component in a software application is not a magic wand which solves all the conceptual problems. On the contrary, developing a successful NN application requires expertise, hence problems of feasibility could emerge according to the available financial resources or the technical knowledge of the developers.

5.2. Specification

5.2.1. The problem

The specification for a NN application is analogous to a Requirements Specification in a conventional application. The specification should define clearly and unambiguously the required characteristics of the application. Wherever possible, these should be expressed in terms of performance limits together with some testable probabilistic statement about acceptable failure rates. (For example, the statement 'there should be at most one major mode failure per 100 million hours of operation' is a suitable specification of failure rate) The specification is the reference throughout the

[24] For example, certain training algorithms require considerable computing resources (of the order of several hours training on high performance workstations).

development of the application as well as for the validation and maintenance operations[25].

However, a detailed functional specification for a NN application is not possible, since the exact function to be modelled is not known. Besides, although the exact function is defined on a continuous space, this function is learned by the NN using a finite set of (possibly) noisy data points, and so the NN can only provide an approximation of this ideal function.

5.2.2. Good practice rules

Currently there is no well-founded standard method of specifying a NN application. Nevertheless, the following information is the minimum that should be contained in the specification:

- A high-level functional specification in natural language (e.g. recognise hand-written characters with a probability of error less or equal to 1%).
- Concerning the inputs:

  Whenever possible, the type and the range of possible values should be given for each input variable. In general, it is very difficult to specify the relevant data space, since it is a multidimensional space, and we should expect correlation between the input variables. In addition, certain regions of the space may correspond to impossible states of a system. Consequently the actual space covered by the input data may be much smaller than the space defined as the Cartesian product of the different input ranges.

  Finally, input values that can lead to an unsafe state for the equipment under control (EUC)[26] should be specified (the specification of the EUC should be the starting reference point).
- Concerning the outputs:

  Even if it is not possible to provide a formal definition of each output, a definition in natural language should be given (e.g. in case of hand-written

[25] A more thorough definition of the specification can be found in [1].
[26] See [1], part 4.

recognition system, the definition of an output might be the probability of that the input letter is a 'w').

In addition, since NNs are probabilistic models, the specification should define what accuracy is expected for the outputs. Thus, the maximum allowable error bounds and failure rates should be specified. Note that these values need not be the same for the entire data space. At safety critical operating points, we would expect the allowable error bounds to be much smaller than in regions of normal operation (e.g. the amplitude of the error bars must not exceed 1% of the predicted value for inputs corresponding to a temperature greater than 100°C, and 5% otherwise).

All the identified ranges of values of the outputs capable of leading (directly or indirectly) the EUC to an unsafe state should be specified.

If the NN function is safety related[27] then the safety function should be defined, together with the expected safety integrity level. The implications of this for the required accuracy of the NN predictions should be defined.

- The required speed and timing performance should be specified, if applicable. Note that for the speed and timing performance, the specification methods used in conventional software are still applicable with NN applications.
- Every invariant property identified should be included in the specification.
- Every other specific property the NN that should satisfy, should be stated in the specification (e.g. self-checking properties such as the control of the validity of the inputs in operation).

5.2.3. Difficulties/Questions:

- What could be done to make this specification as formal as possible? Could new formal or semi-formal methods be defined?
- One of the main practical difficulties is to determine what are the failure modes of the EUC and translate these failure modes into requirements in terms of error

[27] See [1], part 4.

bars on NN outputs. This is made more difficult if the NN outputs are propagated through other functions (which may not be statistical in nature).

5.3. Data issues

One of the most important differences between NN applications and conventional software applications is that the NN model is data driven, which implies that when developing a NN application great care should be spent on addressing the data issues. Indeed in the NN context more than ever, the old computer science saying "*Garbage In Garbage Out*" can be applied.

Thus, when developing a NN application, one should expect to spend 30-40% of the overall time development on the data issues.

5.3.1. Data collection

5.3.1.1. The problem

There are mainly two types of data to be collected: the data sampled from the system to be modelled, and the prior knowledge concerning this system.

The quality of the data is in general closely related to its origin and the way it has been collected.

One should not expect to develop a successful NN application by only using the data points collected and presenting them in their raw form to the NN to be trained. To develop a successful application, the data usually needs to be pre-processed before being presented to the NN unit and sometimes specific NN architectures have to be developed (e.g. shared weights to capture invariance). Prior knowledge is a key factor to enable an efficient pre-processing of the data, or to choose an appropriate architecture.

5.3.1.2. Good practice rules

- The greatest care should be taken to collect some relevant prior knowledge.
- The data origin (prior knowledge and data samples) should be reliable.
- If the data has been collected on a long period of time, the history of the collection process should be studied. For instance, in order to check that the data is homogeneous (e.g. a change in the measurement methods can alter the data homogeneity). It should be ensured that the data is homogeneous, in the sense that,

all the data points have been collected in the same conditions. If this is not the case, the data samples are not necessarily comparable. Moreover, these collection conditions should be the conditions in which the NN application will be used.

- If the data quality is high enough, then the more data that is available the better. The idea is that the more complex is a NN, the more flexible it is, and the more complex functions it can model. However, this greater flexibility must be controlled to avoid overfitting, and this is the role of the training data. The more data there is available, the greater the NN complexity can be without overfitting, and hence the greater the potential accuracy.
- Suitable means should be used to extract a representative data set of the problem from the data samples collected (or at least to enable a representative test of the NN function). This means for instance, that the data set should include data points corresponding to every relevant failure mode. This also means that the data density corresponding to the system in operation can be determined.

  To ensure a good coverage of the relevant data space, techniques such as failure mode analysis can be helpful. To model the data density of the system in operation, the simplest (but not always possible) solution is to collect data from the system in operation condition during a 'sufficient' period of time. The data collected can then be used to fit a data density model.

5.3.2. Data analysis

5.3.2.1. The problem

Good quality data is a rare commodity. Accordingly, it is of fundamental importance to understand the characteristics of the available data in order to use it efficiently. Moreover, if the quality of the data is not high enough, the performance of the NN is likely to be poor. So the purpose of a data analysis or *audit* of the data is to evaluate the data quality and to provide the means to select and correct the data, and thus improve its quality.

The data analysis must be undertaken before the design phase, since the results of this audit may significantly influence the design.

As in other phases of the life cycle, the methods appropriate to time series data are different from those appropriate to static data. This is because the data set is ordered

and we cannot make the assumption that the patterns are all independently sampled from the underlying data generator.

Finally, this data analysis is necessary since, as with classical software, it is much more cost effective to detect problems[28] at an early stage of the life cycle than, for instance, at the testing stage.

5.3.2.2. Good practice rules

- It should be checked that the data is representative of the problem. For example, the coverage of the data space can be assessed by using visualisation techniques. This can, for instance, enable to detect possible holes in the data space coverage
- The function to be modelled may be multi-valued. If this is not detected then an inappropriate model will be used which will necessarily have low accuracy. Indeed, if the NN has been designed to model a single-valued function (both MLP and RBF regression networks are of this type), at each input it will compute the mean of the output values, which may differ dramatically from a possible output value. Multi-valued functions may arise in *inverse* problems[29] (see fig. 10).

[28] For instance the function to be modelled may be multi-valued, see section 5.3.2.2.

[29] Specific NN architectures can be used to deal with inverse problems (e.g. mixture density network; see [3] pp. 212-222, and [7]).

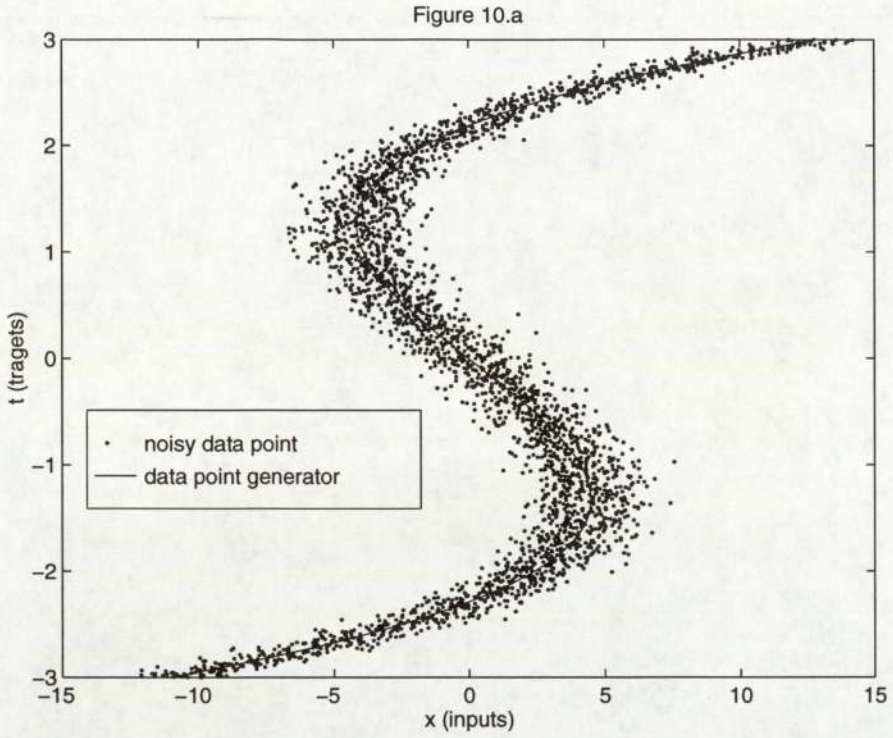

Figure 10.a: Example of a multi-valued function.

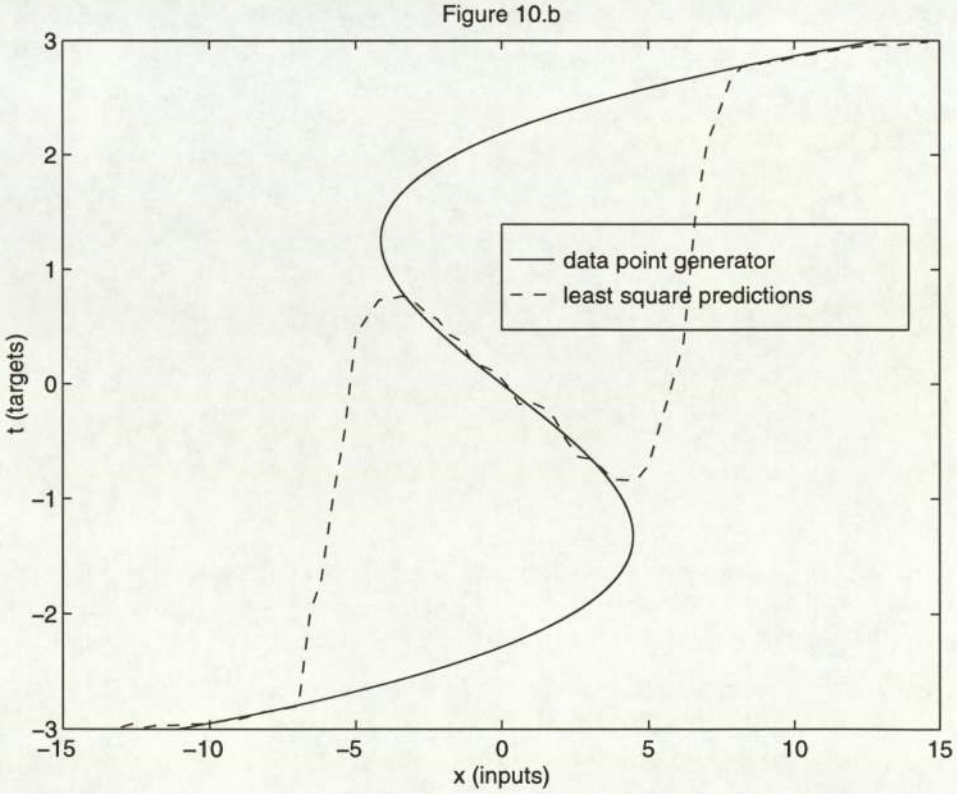

Figure 10.b: This figure illustrates the typical result coming from a learning process that uses a single-valued mapping to model a multi-valued function. For each input, the NN predicts the mean of the target values, and thus gives poor predictions.

- Whenever the dimensionality of the input data space is not small compared to the number of data samples available, feature extraction should be carried out. The main reason for doing this is the curse of dimensionality: in high dimensions, large numbers of data points are needed to model the density accurately (see section 4.7).

  The goal of feature extraction is to reduce the dimensionality of the input data space by selecting a set of new input variables for the NN (the features) whose size is lower than the dimensionality of the initial input data space.

  The simplest method consists of suppressing the less relevant input variables (e.g. variables with no or little relevance to the problem, almost constant variables, or variables with small correlation with the outputs).

  More elaborate techniques construct new variables (the 'features') from subsets of the initial variables. For speed and simplicity, linear techniques, such as Fisher's discriminant (also known as canonical variates), Principal Component Analysis, and factor analysis, are commonly used.

  However, whatever the technique used to select the features, it should include at least two components: a criterion to compare the candidate sets of features and select the best ones, and a systematic procedure to search through the candidate sets of features (see [3] pp. 105-112).

  Note 1: feature extraction is typically the kind of operation that can be performed by a pre-processing unit.

  Note 2: feature extraction cannot increase the amount of information contained in the data but can decrease it. So, a compromise must be found between the loss of information generated by the feature extraction and the greater data density resulting from the reduction of the dimensionality. Often, the only way to find the best compromise is to train NNs on several different feature sets and compare their performance.

- Problems of missing data should be considered with the greatest care. Several options exist when dealing with missing data.

The simplest method is to discard all data with missing field(s). This poses two major problems. First, if little data is available it may be impractical to discard data. Second, the cause of these missing values may be correlated with the data itself, hence discarding data could mean discarding relevant information. An example of the latter case would be a sensor that did not return a reading if the variable it was measuring was greater than a certain value.

Other techniques try to complete the data. However, no method can ensure a perfect reconstitution of the data. One of the simplest methods consists of filling in the values of the missing fields by the average values of the corresponding field; this average value being computed using the data points for which this value is present. This approach may lead to very poor estimates of the actual values of the missing fields and, therefore, may introduce "errors" in the data, and consequently "errors" in the training process. These errors may not be detected if the same approach has been applied to the validation and testing data sets. More principled techniques are based on modelling the input data density and imputing the missing values given the known values for a data point.

Whatever technique is applied, the problem of missing data is critical, and the knowledge of a domain expert should be used to assist and validate the operations.

Lastly, note that there are several types of missing data[30] and that, the relevance of the methods used depends on these types (see [17]).

- Variable types and encodings should be determined.

  There are three main types of variables: continuous, ordinal and categorical[31].

  A variable is said to be *continuous*, if it can be measured as a real number and applying arithmetic operations (such as addition and multiplication) makes sense. For example, the yearly quantity of oil transported by a tanker is a continuous variable. On the other hand, if an object's colour is measured as 'blue', 'green' or 'red' and then encoded as values '0', '1' and '2', the colour is not a

[30] For example: missing at random data, missing completely at random data, observed at random data. See appendix A for further information.

[31] In the literature, a wide variety of terminology is used to refer these different types of variables.

continuous variable, since it does not make sense to add two values. Usually, continuous variables arise directly from objective measurements. Feed-forward NN models use real valued inputs and outputs, and hence continuous variables cause no special problems.

An *ordinal* variable takes on a finite set of possible values, usually small (less than 20). These values are ordered, but the normal arithmetic operations and do not apply. Typically, such variables arise through the discretization of an underlying continuous variable (for example, the Beaufort scale for wind speed). They are usually mapped onto numeric values with the correct ordering. Because NNs are non-linear models, the exact values chosen in this mapping are not critical.

However, it should be noted that, as such variables result from a discretization of a continuous value, they are by definition approximate. The coarser is the discretization, the greater is the approximation. This problem should not be neglected, since using an extra variable as an input of a NN, implies an increase in the dimensionality. If this variable does not carry enough relevant information, its use may be synonymous with a decrease in the performance (see section 4.7).

Such variables may also result from a human evaluation and then can be very subjective (e.g. to appraise the wear of an equipment the human operator can have to choose between the following set of values: 'no wear', 'light wear', 'normal wear', 'serious wear', 'critical wear'). Consequently, before being used, such variables should be carefully analysed and their utility should be evaluated.

Even, if in general these problems are not critical, they exist.

A *categorical* variable is one where there is no order between the different values that it takes (e.g. the variable 'colour' above mentioned). Owing to their nature, categorical variables can be the cause of serious problems when used with data-driven models. For instance, they often stem from a subjective human evaluation (e.g. 'purple' may be classified as 'red', or 'blue'), which means that they are often intrinsically approximate, and therefore, not always suitable when a high accuracy is required.

Another problem with categorical variables, is that they require specific encoding techniques. For instance, let us suppose a categorical variable having a numeric form. If this variable is presented to the NN without being encoded, the NN will interpret the numeric values of the variable as actual numbers and then will assign an order on these values. But this order is completely artificial and should not be used as an information during the NN training.

Accordingly, encoding problem should not be neglected with categorical variables. One of the most common and suitable ways to deal with this problem is to isolate the modalities of the variable by creating a new variable for each modality. These variables are Boolean, and disjunctive one from another. This procedure is called 'disjunctive coding'[32].

- Outliers should be identified and discarded. For instance, if the sum of squares error function is used then, as this error function is very sensitive to outliers, a single outlier may affect severely the training process and so the performance. Accordingly, to detect outliers, data density modelling techniques should be used in conjunction with prior knowledge concerning the data (e.g. expert opinion).

  However, it is advisable to be cautious. For instance, data points corresponding to failure modes occurring rarely may have all the characteristics of outliers, yet they should not be considered as such and should not be discarded. Indeed, if the NN is not trained on these cases, it will not be able to recognise them in operation and will extrapolate and then provide unreliable predictions. Under such circumstances prior knowledge should be used to decide which data points to discard.

  Note that the detection of outliers should not been done on pre-processed data, since the pre-processing function may map outliers to 'non-outliers' values (e.g. if a dimensionality reduction has been carried out).

- It should be checked that the data is not too noisy, and hence reliable.

  Here again, the number of available data points is a fundamental parameter. To illustrate this, let us consider an over-simple case. Suppose that the function to

[32] This encoding is often called '1-of-c coding'. For further information, see [3] p. 300, and [25] pp. 217-218.

be modelled is the straight-line defined by $h(x) = x$ (x being the input variable), that the training data set is constituted of two data points corrupted by a zero-mean Gaussian noise with a unit standard deviation, and that the two data points are[33] (-1,0) and (1,0).

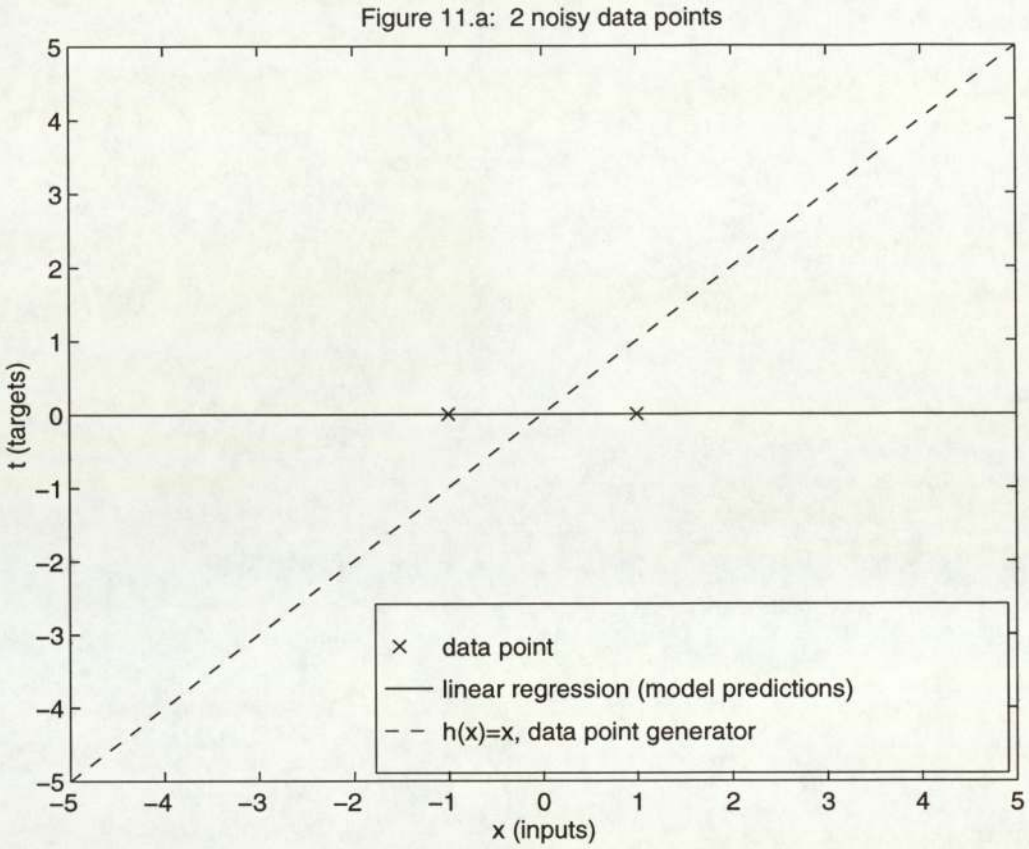

Figure 11.a: This figure shows that if too few data points are available compared to the magnitude of the noise then, the model is unable to capture the linear relationship between the inputs and the targets.

If a linear relationship is sought for this particular data set, then the result will be that there is no relationship between the input and the target values (see fig. 11.a), since the target values are equal to the constant zero whatever the value of x. This is due to the fact that too few data points are available compared to the magnitude of the noise. If, for the same noise magnitude, the data set used is large enough, then the model becomes able to capture the underlying linear relationship between the inputs and the targets (see fig. 11.b and fig. 11.c).

[33] The first component of each pairs is the input, and the second the target (that is, $h(x)+\varepsilon$).

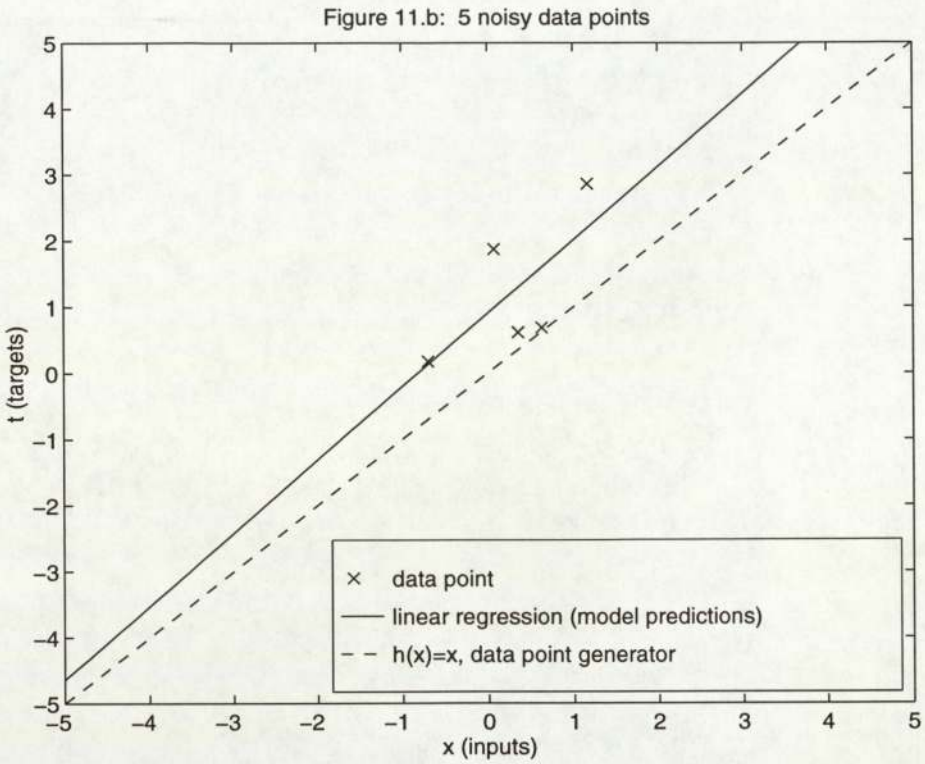

Figure 11.b: This figure shows if more data points are available, then the effect of the noise can be reduced.

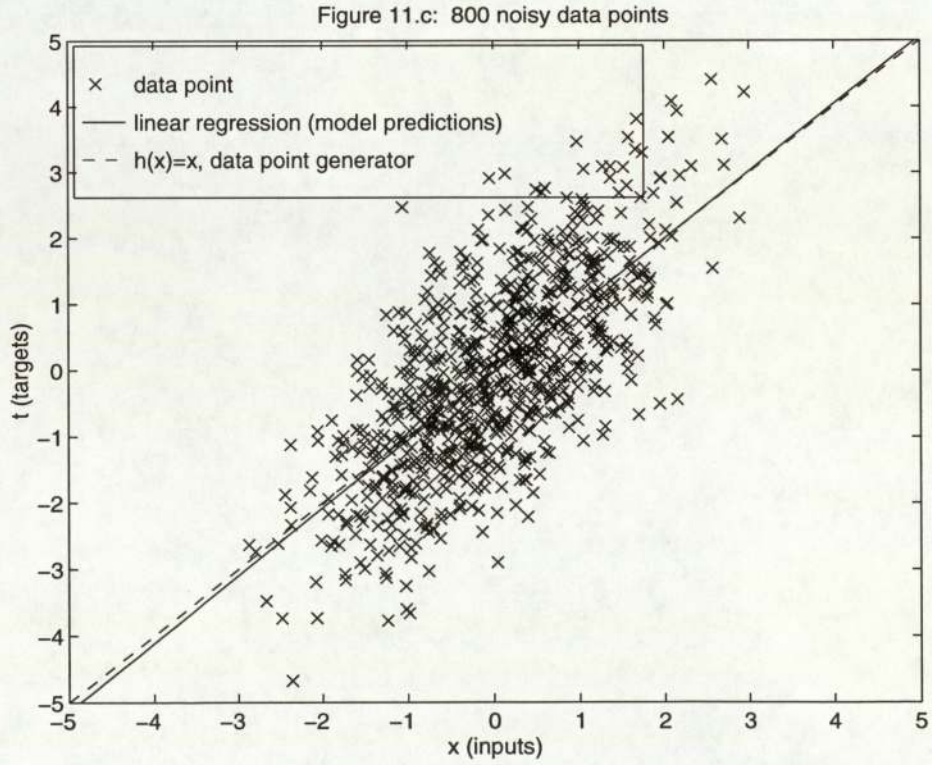

Figure 11.c: As figure 11.b, this figure shows that the effect of the noise can nearly be suppressed by a sufficient increase in the size of the data set.

- Time series data:

Most of the issues related to static data are still applicable to time series data. However, it should be noted that time series data has two specific features: First, it has a temporal aspect, secondly, fewer data samples are generally available than with static data sets. This raises extra problems:

- Outliers and time series:

It often happens (independently of measurement or typing mistakes) that there are outliers in a time series.

These outliers cannot be discarded as easily as with static data. As the data points are not independent and supposed to be equally spaced in time, suppressing one of them would create a kind of hole in the time series, and all the data points later than the point suppressed would be shifted back in time. This could severely alter the time series structure. On the other hand, leaving an outlier in the data set when this data set is a small one, can result in a poor prediction performance.

Hence, with time series, outliers should be considered as missing data that must not be discarded, and usual techniques to complete missing data can be used to replace outlier values by more relevant ones.

- Time spacing:

The temporal nature of time series data raises specific extra problems concerning its homogeneity. It is implicitly assumed that the time interval between two consecutive observations is constant. With real-world data this requirement may introduce extra difficulties.

For instance, if the time series represents the monthly values of a given variable, then it is advisable not to use these values directly to model the time series. For example, values for the month of March (i.e. 31 days) may not be comparable with the ones for February (i.e. 28 or 29 days).

These problems should not be neglected, since they can significantly distort the time series structure and, therefore, affect the NN performance.

- Detrending:

The time series may have an underlying trend, such as a steady increase. If that trend is not removed, then, when predicting on inputs beyond the training data, the NN will be forced to extrapolate[34], and therefore, will have a poor performance.

Such a problem can be reduced by trying to model the trend by using a simple model (e.g. a linear model), and subtracting the values predicted by this model from the data. The NN is then trained on the transformed data. This procedure is called *detrending*.

Note that if a detrending technique is applied to train the NN unit, the same detrending technique must be applied in operation to pre-process the data.

An underlying trend in a time series should be detected in the early stages of the development (e.g. by plotting the time series or by using statistical tools).

5.3.2.3. Difficulties/Questions:

- Uncertainty in the data:

When missing data problems occur, whatever the method that is used to reconstitute a "complete data set", approximations are made. Consequently, if such methods are used we can wonder:

. whether corrected data should not systematically be excluded from the testing sets. Indeed, if a testing set includes such data, in what extent is the generalisation error obtained on this test set reliable?

. whether the approximations made to complete the data could not be taken into account and incorporated as uncertainty on the input data (e.g. by using the concept of noisy input data[35]).

The same type of problem of approximation exists with categorical variables, and to a lesser extent with ordinal variables.

---

[34] Indeed, in the case of a steady increase, the values of the time series after the ones of the training set, are, roughly speaking, greater than those of this training set. Consequently, they do not lie in the training data space area, which is a typical case of extrapolation. Time series with that type of trend are special cases of non-stationary time series (see section 5.7).

[35] Usually only noisy outputs are considered. Currently, the problem of noisy inputs is at a research stage.

The higher the safety integrity level required, the more sensitive this kind of problem becomes.

- Noise:

  We can wonder whether a rule of thumb could not be found to estimate the maximum tolerable level of noise. For instance, this maximum could be, defined as a function of the data mean value, the data standard deviation and the size of the training set.

5.3.3. Constitution of the training, validation and testing sets

5.3.3.1. The problem

A NN learns from data samples, consequently if the data used to train the NN is not representative of the function to be modelled, then the results may be very different from the expected ones (see fig. 7). That is why a good training set is crucial.

Usually, several different NN configurations are trained and the best ones are selected according to their generalisation capabilities. In other words they are selected according to their ability to give accurate predictions on data distinct from the training data. Hence, to evaluate the generalisation capacities of the different configurations, an independent data set of the training set is used. This data set is called the validation set[36].

Finally, a third data set, the testing set, is used to validate the selected configurations. The use of this third independent set is required since the validation set has already been used to select the best configurations and, therefore, is not independent of them.

As these three sets have different functions, the criteria to be considered for their constitutions are different.

Note: in the Bayesian framework this distinction between the training and the validation set does not exist.

5.3.3.2. Good practice rules

- The distribution of the data points, in terms of quantity, between the three sets should be carefully considered. Indeed, the more training data is provided, the

[36] In [22], the author uses an other terminology: the 'validation set' is called the 'acceptance set', and the 'testing set' is called the 'validation set'.

better the NN can learn the function to be modelled[37]. Nevertheless, enough data should be allocated to the validation and testing sets in order them to be representative of the problem and enable a dependable test of the NN.

Note: To guide this distribution, rules of thumb can be used. For instance, for two-class classification problems, the Widrow's rule of thumb gives a lower bound for the size of the training set (N_{min}) required if a MLP is used: $N_{min}=\frac{W}{\varepsilon}$, where W is the total number of weights in the NN, and ε the maximum percentage of misclassification tolerated on the testing set (see [2] p. 152, see also [3] p. 380 and [14] p. 156).

- This second level of data issues, the selection of the models to be trained, and the selection of the training techniques should be addressed conjointly:

  - The number of data points available should influence the choice of the complexity for the models to be trained. For instance, for classification problems, the rule of thumb given just above can be used to determine an upper bound for the NN complexity.

  - The model and the training techniques selected should influence the splitting of the initial data set into training, validation and testing sets. For example, cross-validation can be used if the model is complex compared to the number of data points available. In this case, the splitting criteria are radically different.

- As the accuracy of the NN predictions in an area of the input data space is inversely proportional to the data density in this area, if a higher accuracy is required in specific areas of the data space, then the training set can be provided with more data points in these areas.

- The testing set is the ultimate reference to evaluate the generalisation ability of the system in operation. Accordingly, this set should be representative of the system in operation: that is, representative in terms of data space coverage, but as well, in terms of data density. If this last criterion is not fulfilled, and if no adjustment of the generalisation error evaluation is undertaken, then the generalisation error obtained

[37] Furthermore if biased error bar techniques are used, increasing the size of the training set has an interesting side effect, since it reduces the bias (see appendix A; definition of 'biased estimate').

on the testing set can differ significantly from the true generalisation error of the NN in operation and, therefore, cannot be considered reliable (see also [22]).

Note: to compensate for the possible difference between the testing data density, and the data density of the system in-operation the way the generalisation error is computed can be changed. The simplest method is to weight each testing point by the ratio between the data density (at the considered point) for the system in operation and the data density (at the considered point) for the testing set.

- The validation set should be representative of the problem to be modelled. Indeed, as with the testing set its role is to test the generalisation capabilities of the NN in operation conditions.
- This distinction among the training, validation and testing sets should always be done[38].
- The training, validation and testing sets should be independent, that is, (i) the training, validation and testing set should have a null intersection, (ii) a suitable randomisation method should be used to split the initial data set.

  As a caricatural example consider the case where only one data set has been used simultaneously as a training, validation and testing set. In this case overfitting cannot be detected.
- For classification problems, the training data set should approximately contain the same proportion of data points from each class.

  For instance, consider a two-class classification problem. Suppose that the proportion of training data points of the first class (healthy patients) is much higher than for the second class (patients with a tumour). If no specific error function is used, the data points corresponding to healthy patients will have a higher contribution in the value of the error than the points for patients with a tumour. In consequence, the NN will tend to learn to classify all the data points as corresponding to healthy patient instead of learning the characteristics of the discriminant information between both classes[39].

---

[38] Exceptions to this rule exist; e.g. in the Bayesian framework no validation set is needed to select the best models.

[39] For regression problems, a similar recommendation is often given: that is, the variables should be deskewed. However, it seems that this recommendation is more based on empirical results than on well established theoretical results.

This artificial distribution of the different classes into the training set may be dramatically different of the natural distribution. This 'distortion' of the distribution should be corrected. Provided the prior probabilities of each class (i.e. the natural proportion represented by each class) are known, it is a simple matter of using Bayes' theorem to rectify the posterior probabilities provided by the NN.

- Data should be normalised before being presented to the NN unit. For example:

  - this suppresses scaling effects that can stem, for instance, from the difference of units associated with each variable,

  - this reduces rounding problems, and therefore, enables to work with a better numerical accuracy,

  - this can simplify weight initialisation.

- Categorical variables (see section 5.3.2.2) should be encoded properly.
- If little data is available, resampling techniques should be considered (e.g. cross validation).

  Note that, if cross-validation is used, then the data set used should be representative of the system in operation, or, suitable means should be used to ensure that the generalisation error obtained is representative. This is due, first, to the way the generalisation error is computed with cross-validation, and secondly, to the fact that the validation error should be representative of the system in operation.

- Time series:

  - One of the major problems with time series is to choose, first, the lag (that is the time increment between each input), and secondly the number of previous inputs to be considered

  For example, consider a time series representing the daily turnover of a shop. If, for instance, the problem is to predict the turn over for the following Saturday, it may be more efficient to take into account the turnover of the previous Saturday (that is, a lag of 7 days) than the turnover of the previous day (i.e. Friday, that is, a lag of one day). Similarly, it may be more efficient to consider the two previous Saturdays than only the previous one.

  Even if determining these two parameters usually requires empirical optimisation, several techniques may be used as a guidance. For instance, a linear model can be

fitted and these parameters selected on the basis of the best fit (e.g. fitting an autocorrelation function for different lags, see [16] pp. 54-75).

Nevertheless, these techniques only give a rough approximation of what could be the values of the parameters. Accordingly, these values should be refined on the NN model.

- One can wonder whether it may be interesting to add extra variables (external factors) as inputs to the NN. Indeed, providing the NN with extra information, may help it in its predictions. However, evaluating the quality of this type of variables and their correlation with the time series is often difficult. Furthermore, usually with time series, little data is available. Therefore, the rule should be: if little data is available, and if the quality of the considered extra variable is not very high, then it should not be used as an input to the network (see section 4.7, section related to the curse of dimensionality).

- Two main strategies exist to split the data set into a training, validation and testing sets:

. selecting the data randomly.

. splitting the data set into three contiguous sequences of data points: one sequence for the training set, one for the validation set, and one for the testing set.

In principle, the second method is better since data points that are temporally close are more likely to be correlated. Moreover, this ensures that the sets are as independent as possible.

If this solution is retained, another problem remains, which sequence should be associated with the training, the validation and the testing set. There are several approaches. The first one considers that is better to test the NN on the most recent data points, since it is more likely to provide a better estimate of how well the NN will perform with future data. On the other hand, if the NN is trained with the most recent data, it is more likely to be more appropriate to future data.

Note: This problems mostly exists, because there is no real-world problem that is perfectly stationary.

- Smoothing:

With a time series application it may be necessary to smooth the data in order to obtain a good performance. This problem is closely related to the size of the

available data set. Indeed, if enough data is available, a NN is able to cancel the noise of the data. With time series, data points are often few and far between, hence, the NN is not necessarily able to cancel the noise properly. Nevertheless, it is advisable to be extremely cautious when applying smoothing techniques. Indeed, smoothing can introduce a bias (see [24] p. 650), or suppress relevant structure in the data (i.e. oversmoothing).

5.4. Design issues

5.4.1. Classical linear methods

If the project is considered as feasible, then before trying to use NN technology, it should be verified that classical statistical techniques (e.g. linear techniques such as linear regression) are not adequate. Indeed, we have the following (simplified) relationship: the more complex is the software solution provided to solve a problem, the more difficult it will be to prove the correctness of this solution[40], and the more it will cost to develop such a solution. Linear models are easy and fast to train, they can constitute good benchmarks, and sometimes they can provide suitable solutions.

If linear models were not a suitable solution, most of the work which would have been carried out (e.g. data analysis, see section 5.3.2) would not be lost and would be reusable with NNs, since this kind of work applies to every data-driven model and therefore to NNs.

5.4.2. Modularity

5.4.2.1. The problem

The choice of the architecture is another very important issue in order to develop a successful NN application. In addition, as a classical software application, a good NN application should be modular, since:

- As the problem to be solved by each module is simpler than the original one, it can be done more easily.

  This modularization may lead to a significant decrease of the dimensionality, since generally, each module has fewer inputs than for the initial problem. This means for

[40] e.g. proving that the solution allows the fulfilment of the safety integrity level required.

instance more possibilities to improve the accuracy of the results, since the power of the NN can be concentrated on a less difficult problem, and by doing so the accuracy can be improved.

This decomposition of the approximation problem into several less complex ones corresponds to the well-known principle of "*divide and conquer*". Intuitively this corresponds to the idea it is easier to understand 2 problems of difficulty δ, than one problem of difficulty 2δ.

- The decrease in the complexity of the problems to be modelled means that the training process is facilitated[41].
- Each module is defined over a smaller data space and, therefore, is easier to test.
- All the above advantages are again met when maintenance operations are needed, in particular the diagnosis and the correction of the problems are easier.

Modularity in the NN context provides most of the advantages it provides in the classical software development context. Nevertheless, one of the main problems raised by the concept of modularization is to find an efficient decomposition of the problem, since in the NN context this does not depend on the same criteria as in classical software development context.

5.4.2.2. Good practice rules

The first rule to respect should be that any part of the problem that can be tackled easily with classical programming techniques should be tackled with classical programming techniques (e.g. input normalisation). Indeed, a NN only provides an approximation of the function modelled, and when a high accuracy is required every unnecessary approximation should be avoided. Furthermore, this makes the work easier for the NN and allows it to concentrate its power on the actual approximation problems.

A good decomposition of the initial problem should ensure that as little information as possible is lost. Several approaches are possible to achieve a good modularization of the problem:

---

[41] The models used, and the error functions to be minimised are simpler. We conjecture that the training time required is far from being merely proportional to the number of weights of the NN, since the minimum of the error function is searched in a weight space whose size increases exponentially with the number of weights in the NN (see section 4.7).

- Using prior knowledge (see next section 5.4.3). Indeed, the domain knowledge may suggest a natural decomposition of the problem, according to the relationships existing between the variables in the "real-world". For instance, there may be a natural decomposition of the initial NN function into several ones.
- Using mixture of experts: partitioning the data space into subspaces and assigning one expert NN to each subspace.
- With classification problems, a multistage recognition approach can be used. That is, using a cascade of NNs that carries out successive refinements of the classification of the inputs.

  Such an approach has been applied successfully to solve a problem of hand-written character recognition [9]. For this application two levels of networks were used; the first level performed a broad classification of the characters, whereas the second level refined this classification.

  Note: In a way, these techniques of successive refinements can be viewed as a special case of a mixture of experts.
- With classification problems posterior probabilities should be provided. These probabilities can be used to explain the decision (e.g. a probability of having cancer equalling 50.1 % is different from a probability of 99.9%). Such posterior probabilities can provide a rejection option. Consequently, the discriminant function should be separate from the NN unit.

  There are other advantages to this modular approach. For example, if a loss matrix is used and it is decided to change this loss matrix the modifications needed will be minor, and no retraining of the NN will be needed.
- Using stacked generalisation[42] or a committee of NNs: the idea being to have more NNs but with a lower input dimensionality (see also 5.4.3.2).
- Except if it is shown appropriate, outputs that are not mutually related[43] should not be gathered in the same NN unit. For instance, let us consider a MLP with one hidden node layer, and whose outputs are not mutually related. In this case, the result of the optimisation process for the first weight layer, is a compromise that tries to simultaneously minimise the error for all the outputs. As, these outputs may

[42] See appendix A; definition of 'stacked generalisation'.
[43] Output variables corresponding to probabilities related to a same problem are examples of mutually related outputs.

have completely contradictory requirements, this compromise may be synonymous with a significant loss of accuracy.

- When real time requirements must be met, before starting the training of a modularised architecture, an estimation of its speed should always be carried out. Indeed, a modularization may imply a degradation of the speed performance of the NN function in operation.

Note: This notion of modularization is closely related to the notion of feature extraction (see section 5.3.2.2).

5.4.2.3. Difficulties/Questions:

We can wonder what the impact of the modularity is on the computation of possible error bars.

For example, let us consider a NN application with two NN modules m_1 and m_2, whose outputs are inputs to a third NN module M (i.e. M is a kind of mixing NN). Given that m_1 and m_2 provide error bars on their predictions, we can wonder how error bars for M can be computed, as, for instance, nothing guarantees that M maps a monotonic function between the error bars.

Once again, this raises the problem of what could be done with noisy inputs (see section 5.3.2.3).

Note: A possible solution to this problem is discussed in appendix C.

5.4.3. Incorporation of prior knowledge

5.4.3.1. The problem

The incorporation of prior knowledge is a fundamental issue that can, if tackled properly, dramatically increase the performance of NN applications. The main idea is that it is easier to solve a problem when relevant hypotheses have been made, than to solve the same problem without any hypothesis.

Prior knowledge can be used advantageously throughout the life cycle. For instance:

- It can be used as a guidance during the data collection, to ensure, for instance, that data has been collected from all the relevant areas of the data space (e.g. data corresponding to every faulty state known).
- During the data analysis, for instance, it can be used:

- to differentiate outliers to be discarded from atypical points that should not be discarded (e.g. rare faulty states),

- as a guidance to facilitate feature extraction.

The following simple example illustrates how prior knowledge can be used to reduce the dimensionality. If we consider a NN with 5 input variables, and if we know that there is an invariant property defined by: the sum of the 5 inputs always equals a constant value, then, only 4 inputs of them are actually required, since the 5^{th} can always be computed from the other 4. For instance, this can occur if the input variables are probability measures that always sum to 1.

- During the design stage:

  - an invariant property is a typical example of prior knowledge that, if suitably incorporated in the design, may lead to a dramatic increase in the NN performance (see section 4.5),

  - prior knowledge may help to achieve an efficient decomposition of the problem into sub-problems (see section 5.4.2).

More generally, the incorporation of prior knowledge is a central issue that may mean that a NN application that was impracticable becomes practicable if prior knowledge is suitably taken into account.

5.4.3.2. Good practice rules

A thorough study of the system to be modelled should be done to collect prior knowledge:

- The specification of the system should be the first source of information.
- Experts of the system (e.g. its main users) should be consulted[44].
- All the relevant prior knowledge (e.g. invariant properties) should be taken into account, and incorporated in the NN application.

  Note: Some invariant properties can be built into the pre-processing unit or in the NN structure itself (e.g. by using shared weights techniques[45]).
- Prior knowledge can be viewed as a special case of hypothesis, and as such, its correctness should be checked (see section 5.6.1).

---

[44] Brainstorming techniques should be considered since being always a good way to gather knowledge.

[45] Such an approach has already been successfully applied in a hand-written ZIP code recognition application, see [3] pp. 324-326, and [14] pp. 139-141.

5.4.3.3. Difficulties/Questions:

It would be probably interesting to develop a standard and systematic methodology to collect prior knowledge, and in particular to identify the invariant properties of a system.

5.4.4. Diversity

5.4.4.1. The problem

Diversity means using simultaneously and independently different means to perform the same function[46]. In the NN context, these different means can typically be different NNs, differentiated for instance by their types (MLP or RBF), the number of hidden units, the type of the training data (e.g. rational data[47], or random data[48]). Such a combination of NNs is usually called a committee.

Diversity is a simple and powerful means to improve the performance of a NN function. This notion of diversity is very similar to the notion of synergy. That is, if the members of the committee are complementary (i.e. roughly speaking, there is at least always one committee member that provides an accurate prediction), then the average performance of the committee can be better than the average performance of any of its member. Accordingly, the key factor to build an efficient committee is to ensure that the probability of simultaneous failure of the committee members is low. This remark is perfectly illustrated in [22] where the author considers a classification problem, explains the principle of majority voting for three NNs, and defines as the central parameter of the voting the probability that two out of three NNs fail simultaneously to classify an input .

The same study [22] has provided the following empirical results: 500 different configurations of NN were trained to solve a classification problem. These configurations differed from one another in, at least, one of the following: the type of NN used (RBF or MLP), the number of hidden units, or the type of training set (rational or random). Among these 500 configurations the best 15 had been selected

---

[46] This idea is used with hardware to limit the probability of hardware failure (see [1] parts 2 and 6).

[47] A rational training data set is a data set that is focused on sensitive areas of the data space. For instance, for a classification problem, a rational training set would typically include a large proportion of data points near to the decision boundaries.

[48] A random data set, by opposition to a rational data set, is a set for which there is no particular concentration of the data in the sensitive areas of the data space.

according to their generalisation performance[49]. These 15 were all MLPs. Then, it has been shown experimentally that if the 5 best ones were replaced by 5 RBF configurations, the average of the individual generalisation ability performances fell from 97.90% to 95.08% *but* the majority vote performance increased from 98.48% to 98.51%.

Two main ideas arise from these results:

- Majority voting improves the generalisation performance. Indeed the generalisation performance of the "best" NN was 98.40%, whereas using majority voting the generalisation: was 98.51%.
- Even if at first sight these results do not seem impressive, they become very important when a very high accuracy for the prediction is required and difficulties to comply with the requirements specification are met. Indeed, these results show that, without trying to optimise the different NN architectures (which may be very costly, since it may be very difficult to overcome the last 100th of per cent of error), the performance can be improved only by combining non-optimal NN architectures.

These results show that diversity constitutes a very simple and powerful extra means to improve the accuracy of the results.

5.4.4.2. Good practice rules

- Diversity should always be considered when difficulties to fulfil the specification in terms of accuracy occur.
- For regression problems, the prediction of a committee should correspond to a weighted average of the predictions of the committee members. In [3] (pp. 365-369) the author gives a method to determine the weighting coefficients that takes into account the possible correlations between the committee members (i.e. the correlation of the residuals).

  Note that techniques exist to compute the error bars associated with a committee prediction (see [28] p. 18).
- For classification problems: The prediction of a committee should be based on voting principles. Various strategies exist [22].

[49] In that case the generalisation performance was the percentage of correct classification on new data.

- Using stacked generalisation. The idea of stacked generalisation is very similar to the one of committee, except that the averaging of the predictions of the committee members is performed by an extra NN[50].

Note: If there is no real time requirement, Bayesian techniques are an interesting solution since they enable a very natural implementation of diversity.

5.4.4.3. Difficulties/Questions

We can wonder:

- whether PAC learning results that are currently applicable to a single network, could be extended to a committee of NNs.
- whether committees consisting of feed-forward NNs and other technologies (e.g. fuzzy logic model) fall in the scope of this project.
- whether an intermediate solution between the committee of NNs and stacked generalisation could not be propounded. This solution would take into account the correlation between the committee members and, to improve the accuracy of the predictions, would also take into account the amplitude of the error bars of each committee member for the considered input point. In this way the weighting coefficients would not be constant but would adapt themselves the input data points.

5.4.5. Other good practice rules related to the design

- With classification problems, in doubtful cases (i.e. input data point very near to decision boundary), in order to minimise the risk of misclassification, the NN should have the possibility of answering "I do not know". Thereby, this gives a reject option to a possible human operator. For instance this reject option principle is widely used with NN application related to medical diagnostic.

  Note that this requirement may be impossible to satisfy and depends on the application considered (e.g. this may be impossible when real time requirements exist).
- With regression problems, the NN function should produce error bars. Moreover, if the error bar techniques used are biased[51], a solution should be provided to detect

[50] See appendix A; definition of 'stacked generalisation'.

[51] For example, error bars based on an estimate of the variance stemming from the maximum likelihood approach are biased.

when the error bars should not be considered as dependable. For example, a natural solution is to model the training data density, and provide simultaneously the error bars and the training data density for the considered input[52].

- The choice of the error function is problem dependent. Thus, for regression problems the sum of squares error function is usually used, whereas for classification problems cross-entropy error function is more appropriate. Nevertheless, these error functions implicitly assume, for instance, that the data points are not correlated, yet it may not be the case (e.g. with time series data) and, therefore, specific error functions may have to be considered.

5.5.Training and model selection issues

5.5.1. The problem

Even if the quality of the NN architecture and the data set are high, if the training issues are neglected, the performance of the NN may be poor and the specification requirements impossible to fulfil. An appropriate architecture and high quality data, only provide the potential to obtain an efficient NN function, but nothing more.

To perform an efficient training, two main pitfalls must be avoided:

- being trapped in a bad local minimum,
- overfitting,

No algorithm is able to always avoid these two pitfalls. Worse than that, there is no *best* method to train a NN, and the optimal method to be used is problem dependent.

Different types of optimisation algorithms exist: some can escape from local minima, others cannot. Some converge very quickly, others converge much slower and can even diverge. Unfortunately, algorithms that generally converge the fastest are also those that cannot escape from local minima.

Generally, several distinct NN architectures are trained[53] and the best ones are selected. This selection is mainly based on the generalisation capabilities of the NN architectures.

---

[52] When the error bars are based on a variance estimate stemming from the maximum likelihood approach, the bias decreases as the training data density increases.

[53] With pruning and growing techniques there is one architecture whose complexity evolves during training.

5.5.2. Good practice rules

- All the training tools and more generally the whole conventional software used in the NN function should be validated before being used (e.g. the back-propagation procedure).
- Suitable techniques should be used to avoid overfitting. The most commonly used are: regularisation techniques (i.e. modifying the error function in order to penalise NN mappings with too a high curvature), early stopping (i.e. stopping the training process when the generalisation error starts to increase), pruning techniques[54].
- Avoiding being trapped in a bad local minima:

  There is no optimisation algorithm that ensures that the global minimum of the error function is found at the end of the training process. The solution found by the algorithm depends on the initial starting conditions (e.g. initial values of the weights). Consequently, whatever algorithm is used, the training process should always be repeated with different initial conditions and include a stochastic component (e.g. multiple random initialisations of the weights).
- Great care should be taken concerning numerical problems (e.g. rounding errors). If neglected they can introduce approximations that can lead to a very sub-optimal solution. For example, with RBFs, to determine the second layer of weights, instead of using classical matrix inversion it is better to use singular value decomposition[55] that can overcome some of the problems related with near-singular matrices.
- As the notion of best optimisation algorithm is problem dependent, several optimisation algorithms should be tried.
- If RBFs are trained, suitable techniques should be used to determine the basis function parameters (e.g. clustering techniques) (see section 4.3.2).
- Selection of the model(s):

  - several different NN configurations should always be tried.

---

[54] All these techniques are more precisely described in appendix A.

[55] See appendix A.

- the main criterion should be the generalisation ability of the model, and so, the choice of the model should only depend on the results obtained on the validation set[56].

5.5.3. Difficulties/Questions

It might be interesting to give approximate rankings of the different existing main optimisation algorithms. These ranking should be based on different criteria such as: the quality of the solutions found (i.e. the generalisation capability of the trained model) or the convergence speed. They should compare the different algorithms for a wide range of problems: from toy problems to real-world problems. At the moment such benchmarks do not exist [23].

5.6. Testing issues

5.6.1. Correctness of the hypotheses made

5.6.1.1. The problem

As a logician would say "false implies true".

During the development of a NN function, various assumptions are made (e.g. distribution of the noise, smoothness of the underlying function). Therefore, the validity of the NN function obtained relies on the validity of these assumptions. Hence, these assumptions should be shown as valid, or at least suitably justified.

To illustrate this, consider the two following examples related to the hypotheses concerning the noise:

(i) A usual assumption is to consider a zero-mean Gaussian noise, but in a safety critical context such an assumption should be verified. For instance, the sum of squares error function allows to provide an estimation of the mean and the variance of the conditional distribution for the output variables. Given these two parameters (mean and variance) error bars can be determined. Nevertheless, for instance, according to the noise distribution, the error bars can be different: if a Gaussian noise is assumed then the predicted error bars are symmetric around the mean value, but if the true distribution of the noise is a Poisson distribution, then the true error bars may

[56] The Bayesian framework does not need any validation set to select the best model; the selection is based on the notion of evidence (see [19] and [28] p. 3).

be significantly asymmetric around the mean value. In this case, the predicted error bars are not reliable.

(ii) in-operation error bar techniques generally assume that the variance of the noise is constant all over the data space. However, this assumption can be false and, consequently, the error bars not reliable[57].

5.6.1.2. Good practice rules

Every important assumption made should be checked, or at least, justified, the rule being "the sooner the better":

- Hypotheses concerning the noise should be checked. For instance, a possible solution is to perform several samplings for each input value and observe the distribution of the targets. Note: Observing the distribution of the residuals can also be useful.
- It should be checked that the assumptions the model is based on are actually verified. For example, in the Bayesian framework, it has been shown that when the number of weights is greater than the third of the number of training points, then the Gaussian approximation, that is one of the main approximations made in this framework, breaks down[58] [28].
- The validity of the prior knowledge should be checked[59]. For instance, it should be checked that the possible invariant properties are really invariant and not almost invariant.

Note: Sensitivity analysis is a very general technique that can be used to evaluate the relative importance of the assumptions made (e.g. it could be used to determine in what extent the choice made concerning the possible missing data can be affect the model).

---

[57] Sometimes, specific techniques can be considered. For example, if the noise is proportional to the target values, a solution is to pre-process the data by taking the logarithm of the target values. The variance of the noise that corrupts the logarithm of the targets can be considered as constant.

[58] This statement applies to the implementations of Bayesian techniques using a Gaussian approximation to the posterior distribution of the weights (see appendix A, 'Bayesian techniques').

[59] This may require the knowledge of a domain expert.

5.6.2. Correctness of the implementation

5.6.2.1. The problem

The correctness of the implementation is at least as important as with conventional software: on one hand, to train the NN unit conventional software tools are used, on the other hand, the trained NN function may contain a significant amount of conventional software (e.g. in the pre-processing unit).

5.6.2.2. Good practice rules

- It should be ensured that the tools that have been used to deal with the data issues have been correctly implemented (e.g. algorithms used to normalise the variables, algorithms to deal with the possible missing values).
- It should be ensured that the optimisation algorithm and each of its components have been correctly implemented (e.g. back propagation procedure, error function)
- It should be ensured that the prior knowledge has been correctly incorporated in the design. For instance it should be checked that the possible invariant properties are satisfied.
- Each unit (i.e. pre-processing unit, NN and post-processing units) should be individually tested.
- Every interface should be tested. That is, interfaces between the different units of the NN function, but as well interfaces between the NN function and some possible conventional software.

All the above aspects are classical software testing aspects, and consequently validation and verification techniques used with classical software are still applicable.

- It should be checked that the training, validation and testing data sets have the expected properties. For instance: (i) they should be independent (see section 5.3.1.2 and section 5.3.3.2), (ii) the validation and testing set should be representative of the problem.

5.6.3. Compliance with the specification

5.6.3.1. The problem

The fact that the different algorithms used to train the NN have been implemented correctly and that the data issues have been properly addressed does not necessarily

imply that the performance specification requirements (in terms of speed and accuracy) are fulfilled.

5.6.3.2. Good practice rules

- For regression problems, when "in-operation error bars" are used[60], it should be checked that these error bars stay in the limits fixed by the specification for all the relevant data space.

  - To test the validity of the error bars, it should be checked on the data available that the targets "usually" lie within the error bars (e.g. two standard deviation error bars should contain targets 95% of the time).

  - The compliance of the error bars amplitude with possible specification requirements (see 4.2.2) can be tested on the data points available. Note: In appendix C, we propose a procedure that may be used to perform a more stringent test if a high safety integrity level is required.

- If the NN performance (speed or accuracy) does not comply with the specification, then a precise diagnostic of the problem should be given. This diagnostic should identify precisely what are the specification requirements that are not fulfilled. Hence, this implies that the testing stage should be completed (even if a problem is detected from the outset of this testing stage). The findings from this diagnostic should then be used to select the appropriate stage of the life cycle to go back to (e.g. it may be decided to come back to the data collection stage).

  Note: If the speed requirements are not met, it should then be tried to simplify the architecture. If not sufficient, then the only remaining solution is probably to use hardware components.

- Normalised error functions should be used during the testing phase, since they allow an interpretation of the generalisation error value obtained (e.g. relative error).

- Testing data should not be used before the testing stage, and should be considered as a means to provide a ultimate validation of the NN function.

- Every specific property the NN function is to satisfy should be shown to be satisfied (e.g. novelty detection property).

- Time series:

[60] Error bars for classification problems raise extra difficulties that are currently at a research stage.

When testing a NN function used to model a time series, another important aspect should be taken into account: the temporal aspect of the time series.

Thus, it is important to know how the quality of the NN predictions evolves through the time.

Mainly, two families of techniques exist:

- Testing the evolution of the residuals (that is the differences between the prediction and the actual value) through the time. Among the aspects that may be useful to consider, there are:

. the mean of the residuals (one should expect them to have zero-mean)

. the correlation of their amplitude with time. Such a correlation can be detected for instance by performing tests of randomness[61] or by trying to fit a simple model on the residuals (e.g. a linear model). An increasing amplitude means that it is likely that the predictions will deteriorate with time[62].

. their distribution (it can be estimated for instance by constructing an histogram). If it is not symmetric, that may indicate that the NN has a consistent bias, and so will have a trend to over or under estimation.

- A more stringent test consists of using the NN as a multiple step-ahead forecaster. That is, to make it predict values and reuse these values, as if they were actual values, to predict the next ones, and so on... Usually the error increases very quickly since errors are accumulated. Note: When it is required to predict a long time in the future, it is better to consider a model with a larger lag.

5.6.3.3. Difficulties/Questions

- Currently, no standard technique exists to test the compliance of "in-operation error bars" with specification in the whole of the relevant data space. A procedure is propounded in appendix C. It is based on the fact that the relevant area of the data space where such tests should be done, is 'more or less' the area where the available data points lie (this implicitly assumes that the data set is representative of the problem in terms of coverage of the data space).

[61] Ideally, if a model has perfectly learned the underlying function the only error in the prediction should be the one due to the noise.

[62] If this trend cannot be suppressed, a more frequent retraining of the network will be needed.

- When developing a NN function, data density modelling techniques can have various applications (e.g. to test if the validation and testing sets can be considered as representative of the problem, see section 5.3.3.2). Nevertheless, these techniques are by definition unsupervised techniques. Consequently, even if the quality of the processes that have been used to develop the model can be assessed, testing the quality of the results obtained using these techniques, is a hard issue. This is discussed further in appendix C.
- How to compute an overall failure rate for regression problems? As no framework seems to address this problem, we propose a solution in appendix C.

5.7. Stationarity issues

5.7.1. The problem

Roughly speaking, a process is said stationary, if the function that describes it, does not involve in time.

For the class of NNs we consider in this project, the NN function is fixed and has no adaptive capability. So if the underlying function that is modelled is not stationary then the predictions provided by the NN may become incorrect[63]. That is why the greatest care should be taken to ensure that the system to be modelled is stationary.

Note 1: problems of non-stationarity can for instance arise owing to the wear of the equipment.

Note 2: The goal of this project is not to provide solutions to solve the problems raised by the non-stationarity, but to provide criteria to assess the measures used to detect if the underlying function is non-stationary.

5.7.2. Good practice rules

The problems of non-stationarity can be minimised by combining preventive measures. For instance:

- A suitable on-line monitoring mechanism should be used in-operation to control that:

  - The data density remains relatively constant.

[63] The possible error bars do not take into account such problems of non-stationarity.

- The performance of the NN remains in compliance with the specification. For instance, by monitoring the value of the error bars and check that the actual values of the function modelled lie within the error bars of the NN in conformity with the specification.

  Note: It may also be interesting to monitor the residuals: e.g. to detect possible trends towards under- or over-estimation in the predictions. Such trends mean that the function computed by the NN does not correspond to the function associated with the system modelled, and therefore, that errors have been made during the development of the NN application.

- The wear of the equipment should be controlled regularly.
- Systematic impact analysis should be undertaken if any modification or renewal of the EUC or the input suppliers for the NN were to be performed.
- If the system is shown as being weakly stationary, and if the degree of control over the data collection is low, then continuous on-line data collection techniques should be considered.
- It might be interesting to check that the conditions when the data has been collected are the same as those conditions just before the commissioning of the E/E/PES.

5.8. Documentation issues

5.8.1. The problem

Throughout the development of the application, special care should be taken concerning the documentation of the data since, as for classical software, it constitutes a crucial issue that must be carefully considered throughout the development of the application.

A good documentation can be advantageously be used throughout the life cycle of a NN application:

- It is useful during the development phase. Formalising problems improves their understanding and thus can enable an earlier detection of possible mistakes. Moreover owing to the iterative nature of a NN function development, it is likely that problems faced in an earlier stage of the development have to be faced again. Under such circumstances, it is useful to know exactly what the problems are, what

approaches had been tried to solve them, what the motivations for these approaches were, as well as the possible assumptions that had been made.

- It makes information accessible to anybody (e.g. this can be vital if the project team loses members).
- It is a central element in the verification, or the validation of a NN function,
- It is crucial to maintain the NN function.

5.8.2. Good practice rules

A good documentation should:

- possess all the usual qualities required for a documentation: it should be accurate, concise, easy to understand, easy to access and maintain[64].
- be suitable for audit trails.

  For instance: it should include a report of the data analysis, it should also mention every relevant prior knowledge information that has been used and show the way it has been used.

  More generally, every major decision should be reported in the documentation, the rule being; "the sooner the better".
- include the list of all the hypotheses made (e.g. hypotheses concerning the noise) as well as the procedures used to check these hypotheses.
- enable each major version of the NN function to be easily rebuilt. In particular, it should be possible to retrain any major NN configuration and reproduce the results. This implies, first, that each relevant configuration should be clearly described in the documentation (e.g. the type of NNs used should be stated as well as the arrangement of these NNs). Secondly, the re-execution of the algorithms used to train these configurations should lead to exactly the same results. For instance, this means that all the parameters of these algorithms should be recorded (e.g. learning rate, stopping criteria) and also that the possible stochastic aspects should be mastered (e.g. recording of the random initialisations of the weights, see section 5.5).
- include the training, validation and testing sets.

---

[64] Further information is given in [1] annexe G.

- include an estimation of the density function of the data sets used (i.e. training, validation and testing sets), and estimations of the actual data density functions for the system in operation.
- report all serious numerical problems that have occurred during the training phase (e.g. ill-conditioning).
- provide information concerning the different algorithms that have been used to develop the NN function. This includes the algorithms used to analyse the data, to split the initial data set into the training, validation and testing sets and, the algorithms to train and to test the NN.

  This documentation is comparable to the type of documentation that is expected when dealing with conventional software. This implies that this documentation should also specify the version identification of the algorithms used, include the code of these algorithms, as well as all the parameters used for these algorithms (e.g. the learning rate in the case of the gradient descent optimisation algorithm).
- describe how the NN function outputs are used by the rest of the system. For example, it should describe possible human factors and answer, for instance, to questions such as: What are the roles of the possible human operators in the monitoring of the neural system? What kind of control decision (having a link with the NN application) may they have to take?
- if a high safety integrity level is required, the working machine precision should be mentioned in the documentation (e.g. precision used for the weights values).

5.9. Other issues

In safety critical context, security problems should not be neglected. For instance, a malevolent modification of the NN function parameters (e.g. the weights values) can change dramatically the function produced by the network.

- Reusability.

  The algorithms used to develop the NN solution are generally reusable (e.g. optimisation algorithms).
- Replication.

  The replication of a NN function is problematic since, as said previously, the class of NNs we consider has no adaptive property. Consequently, if the environment

where the replication is planned is not the same as the original, then the NN prediction may not be reliable. This means that before replicating a NN function, it should be checked that the destination environment can be considered as identical with the original. If in doubt, a retraining and 'retesting' of the NN component should be performed.

- Maintenance issues.

  The NN unit may need to be retrained at least for the two following reasons:

  a) new relevant data points have been encountered in operation. Under such circumstances, this new data and the old data together can be used to retrain the NN.

  b) the system may be slightly non-stationary, which may require periodic retraining of the NN. In this case, data that is too old should not be used to retrain the NN.

5.10. Summary

Owing to the problem of curse of dimensionality (see section 4.7), the task a NN has to solve is a generally a very hard one. Hence, everything should be tried to simplify this task.

The keyword is: *the data*, and everything should be done to have a better understanding of the data.

If the data is not understood a NN function is a kind of black box with all the drawbacks that represents.

If the data is better understood, then a NN function looks more like a glass box.

6. Results and future work

6.1. Results

6.1.1 Introduction

From the good practice rules discussed in the previous chapter and from the life cycle proposed in the section 3.2, guidelines to assess NN applications have been developed. The efficiency of these guidelines has been tested against two cases studies. Some understandable problems of confidentiality (see section 1.2.1) prevent us from including the guidelines in this thesis. For the same reasons we have not included the detailed results of the application of these guidelines to the case studies. Nevertheless, thereafter we report the main findings for both case studies, furthermore, an extract of the report for one of them is given in the appendix B.

The first project has been developed by Oxford University in collaboration with Oxford Instruments. This project was aimed at enabling sleep disorder detection. It was completed and in a commercialisation phase. The second project is currently developed by Aston University in collaboration with SAGEM and is related to ignition timing calibration. At the moment of the case study, this project was only in its early stages, that is, its feasibility had been shown on a sub-problem.

As with conventional software, providing a formal proof of the correctness of a NN application is in general impossible. However, both case studies suggest that if a NN application is developed in a sensible way and properly tested, then it should be possible to validate and verify it with an efficiency at least comparable to what is obtained in conventional software.

6.1.2. Findings

6.1.2.1. CONCERNING THE PROJECTS ASSESSED

- Both case studies have shown that during the development of a NN function, two types of issues seem to be generally neglected: the documentation and specification

issues. For instance, for both applications no quantitative target was defined in the specification. With conventional software, these issues are crucial.

The 2 main reasons are probably that:

- as NN technology corresponds to a novel and very different way of viewing software, no standard currently exists. For instance, there is no clear definition of what should appear in a specification for a NN function.

- Although NN technology is currently moving from research to products, most NN applications are developed by academics. While these latter generally have a very good understanding of the technology, they usually do not have the same objectives and the same experience in software development as software houses.

Consequently, we can wonder whether these problems, and more generally, the research nature of NN applications development, are relevant problems or temporary problems due to the relative youth of NN technology. If we glance through conventional software history, the second interpretation is the more likely.

The case studies have emphasised the highly iterative nature of NN application development. Thus, for both projects a quick loop through the life cycle has been performed to evaluate the project feasibility. This is mainly due to the fact that there is currently no technique that enables to foresee the accuracy of the prediction of an NN function before testing it. Therefore, developing a NN function is a 'trial and error' process.

6.1.2.2. CONCERNING THE GUIDELINES

- The cases studies have enabled the detection of minor deficiencies of the guidelines. They have been corrected.
- Globally, the draft guidelines, were adequate to assess both projects.
- The guidelines have enabled to the discovery of technical mistakes committed during development of both projects (e.g. see appendix B).

6.1.2.3. PRINCIPLES OF VALIDATION AND VERIFICATION FOR NEURAL SYSTEMS

Both case studies appear to demonstrate that the basic principles used to validate and verify conventional software are applicable to NN technology, that is:

- checking that the NN function has been developed in a sensible way (i.e. quality of the processes and methodology used).
- checking that the NN function has successfully passed the tests and complies with its specification.

It should be noted that none of these two aspects, if used alone, is sufficient to validate and verify a NN function. However, when used together, they are complementary and we can expect the validation and verification of a NN function to be at least as stringent as with conventional software.
Intuitively, this means that if the NN function has been properly tested, and has passed the tests, then if the development has been rigorously carried out, it is very likely that the NN function does comply with its specification.

6.2. Future work

For completeness, note that for some of the good practice rules given in chapter 5, no standard technique enabling their application has been found in the literature. In appendix C, we propose procedures that we have created to enable their application. We do not pretend to give definitive procedures but rather an idea of the feasibility of such procedures and a possible basis for further investigations.

Note that the good practice rules given in chapter 5 consider safety critical NN functions. For non-safety critical NN functions the development can be less stringent. Anyway, the idea is: the more stringent the development, the more likely the NN function is to be safe for use and to operate in conformity with its specification.

Basically, the future work consists of refining what has been done during this MSc project and extending it to:

- NN as controllers,
- unsupervised techniques (e.g. data density modelling, mixture density models),
- non-stationary time series.

CONCLUSION

As was the case in the early days of conventional software development, there is in the NN technology context a need for standardisation. The use of standard methodologies such as the life cycle proposed in section 3.2 could be an important step in this direction.

There are several motivations for standardisation:

1. Providing guidelines to develop successful NN applications makes NN technology more accessible.

2. The development of an application is easier to control if it is done in a systematic way. Moreover, a better control over development is synonymous with a higher dependability.

3. A standardised development methodology enables standardised verification and validation methodologies. The findings of the case studies appear to demonstrate that validation and verification of neural systems are possible and can probably be as stringent as with conventional software.

The lack of standardisation in NN technology is probably the main obstacle in the way of its mass generalisation. Let us hope that the small contribution made by this project will help it to progress in the right direction.

List of references

[1] "Draft IEC 1508: Functional safety: safety related systems", (June 1995)

[2] E B Baum, D Haussler, "What Size Net Gives Valid Generalization ?", Neural Computation 1 pp. 151-160, (1989)

[3] C M Bishop, "Neural Networks for Pattern Recognition", Oxford University Press, (1995)

[4] C M Bishop, "Neural Networks and their applications", Review of Scientific Instruments Vol. 65 pp. 1803-1832, (1994)

[5] C M Bishop, M Svensén, C K I Williams, "EM optimisation of Latent-Variable Density Models", Advances in Neural Information Processing Systems 8, MIT Press, (1996)

[6] C M Bishop, "Mixture Density Networks", Neural Computing Research Group Report NCRG/94/004, (1994)

[7] C M Bishop, "Novelty detection and Neural Networks Validation", IEE Proc. Vis. Image and Signal Process 141 pp. 217-222, (1994)

[8] C M Bishop, M Svensén, C K I Williams, "GTM: A Principled Alternative To The Self-Organizing Map", Neural Computing Research Group Report NCRG/96/015, (1996)

[9] P A Brierton, M R Lynch, "Application of Linear Weight Neural Networks to Recognition of Hand Print Characters", UK IEE Conference Publication No 409 pp. 143-147, (1995)

[10] A N Burgess, "Non-Linear Model Identification and Statistical Significance tests and their Applications to Financial Modelling", UK IEE Conference Publication No. 409 pp. 312-317, (1995)

[11] J B Copas, "Regression, Prediction and Shrinkage (with discussion)", J.R. Statist.Soc B, 45(3), pp. 311-354, (1983)

[12] J De Lagarde, "Initiation a l'Analyse des Données", Dunod, (1993), (in French)

[13]Y M Enab, "Genetic Algorithms for Identifying Self-generating Radial Basis Neural Networks", UK IEE Conference Publication No. 409 pp. 65-70, (1995)

[14] J Hertz, A Krogh, R G Palmer, "Introduction to the Theory of Neural Computation", Addison-Wesley Publishing Company, (1991)

[15] S B Holden, M Niranjan, "On the Practical Applicability of VC Dimension Bounds", Neural Computation 7 pp. 1265-1288, (1995)

[16] Sir M Kendall, J K Ord, "Time Series (third ed.)", Edward Arnold, (1990)

[17] R J A Little, D B Rubin, "Statistical Analysis with Missing Data", Willey, (1987)

[18] D Lowe, "On the Use of Nonlocal and Non Positive Definite Basis Functions in Radial Basis Function Networks", UK IEE Conference Publication No. 409 pp. 206-211, (1995)

[19] D J C MacKay, chapter 6: "Bayesian Methods for Back-propagation Networks" in "Models of Neural Networks III", Springer-Verlag, (1994)

[20] R M Neal, "Bayesian Learning for Neural Networks", Ph.D. thesis, University of Toronto, Canada, (1995)

[21] J Pardey, S Roberts, L Tarassenko, J Stradling, "A new approach to the analysis of the Human Sleep-Wakefulness Continuum", J. Sleep Res., vol.5, no.6, (1996)

[22] D Partridge, W B Yates, "Engineering Reliable Neural Networks", UK IEE Conference Publication No 409 pp. 352-357, (1995)

[23] L Prechelt, "A Quantitative Study of Experimental Neural Network Learning Algorithm Evaluation Practices", UK IEE Conference Publication No. 409 pp. 223-227, (1995)

[24] W H Press, S A Teukolsky, W T Vetterling, B P Flannery, "Numerical Recipes in C: The Art of Scientific Computing (Second ed.)", Cambridge University Press, (1992)

[25] G Saporta, "Probabilités, Analyse des Données et Statistique", Technip, (1990), (in French)

[26] L Tarassenko, P Hayton, N Cerncaz and M Brady, "Novelty Detection for Identification of Masses in Mammograms", UK IEE Conference Publication No. 409 pp. 442-447, (1995)

[27] C K Tham, "On-line Learning Using Hierarchical Mixtures of Experts", UK IEE Conference Publication No. 409 pp. 347-351, (1995)

[28] H H Thodberg, "A review of Bayesian Neural Networks with an Application to Near Infrared Spectroscopy", IEEE Trans of Neural Networks, Vol. 7, 1, pp. 56-72, (1996)

[29] S M. Weiss, C. A. Kulikowski, "Computer Systems that Learn", Morgan Kaufmann Publishers pp. 41-49, (1991)

[30] C K I Williams, C Qazaz, C M Bishop, H Zhu, "On the Relationship Between Bayesian Error Bars and the Input Data Density", UK IEE Conference Publication No. 409 pp. 160-165, (1995)

[31] D H Wolpert, "Stacked generalization", Neural Networks 5 (2), pp. 241-259, (1992)

Note: The "UK IIE Conference Publication No. 409" is the publication associated with the fourth international conference on Artificial Neural Networks organised by the Electronics Division of the Institution of Electrical Engineers.

Date: 26-28 June 1995,

Place: Churchill College, University of Cambridge, UK

Published by the Institution of Electrical Engineers, London.

Bibliography

- I S Helliwell, M A Turega, R A Cottis, "Accountability of Neural Networks Trained with 'Real World' Data", UK IEE Conference Publication No. 409 pp. 218-222
- S Klyne, "Validating Connectionist Implementations", UK IEE Conference Publication No. 409 pp. 228-233
- G Morgan, J Austin, "Safety Critical Neural Networks", UK IEE Conference Publication No. 409 pp. 212-217
- I T Nabney, C M Bishop, C Legleye, "Modelling Conditional Probability Distributions for Periodic Variables", Neural Computing Research Group Report NCRG/95/010, (1995)

Note: The "UK IIE Conference Publication No. 409" is the publication associated with the fourth international conference on Artificial Neural Networks organised by the Electronics Division of the Institution of Electrical Engineers.

Date: 26-28 June 1995,

Place: Churchill College, University of Cambridge, UK

Published by the Institution of Electrical Engineers, London.

Appendix A: Definitions

Foreword

This appendix is aimed at:

- defining the technical vocabulary used in the main text of this thesis,
- giving a further insight into a few important notions or techniques,
- providing references to source of further information.

So far, in the NN technology context, no genuine standardisation effort has been made concerning the terminology. Accordingly, the terminology used is not always consistent and may be a source of ambiguities. For example, although [3] and [14] are two references in the NN literature, some discrepancies between them exist concerning the terminology related to basic notions.

In this appendix we have tried to use a terminology consistent with the one used in [3].

Definitions

- **Activation function** (also called 'gain function', 'transfer function' or 'squashing function')

One activation function is associated with each hidden node, and each output node. The activation function computes a transformation (linear or non-linear) of the input

values entering the node[65]. The result of this computation (which is called the *activation of the node*) is either used as an output of the NN (if the node is an output node) or used as an input value to the following layer if the node is a hidden node (see [3] pp. 82).

The non-linear modelling properties of NNs, and hence their power, stem from their activation functions that are their only non-linear components.

Note: Usually, the same function is used for every node of a node layer, thus most of the time, with MLPs, the same sigmoidal activation function (see 'sigmoidal activation function') is generally used for every hidden node, and the same function is used for the output nodes (generally the identity function for regression, and for classification, according to the number of outputs nodes, either the logistic or the softmax activation function).

- **Activation of a node**

See 'Activation function'.

- **AR model** (or 'AutoRegressive model')

See 'autoregressive model'.

- **Autocorrelation**

The autocorrelation for a lag k and a time series S_t is defined as the linear correlation coefficient between the data points S_{t_i-k} and the data points S_{t_i} for $i > k$ (See [16] pp. 52, see also section 5.3.3.2).

- **Autoregressive model** (or 'AR model')

[65] Input nodes are a bit peculiar since they do not compute any transformation of the single input they receive, and transmit it unchanged to the next node layer.

The autoregressive model is a linear model used for stationary times series modelling (see section 4.2.3). The basic idea is that the value to predict at time t (S_t) can be computed using a linear combination of the n previous values of the time series. That is, $S_t=a_1.S_{t-1}+a_2.S_{t-2}+...+a_n.S_{t-n}$, where the a_i's are constant coefficients (i.e. independent of t), and n is the order of the model (see [16] pp. 62-63, pp. 65-68 and [24] p. 573).

- **Back-propagation**

Most non-linear optimisation algorithms (see 'non-linear optimisation techniques') used to train MLPs (see 'multi-layer perceptron') use gradient information (that is, the derivatives of the error function with respect to the weights, see 'gradient'). The Back-propagation procedure is an efficient means of computing an accurate approximation of this gradient[66] (see [3] pp. 140-148). The term 'back-propagation' stems from the fact that a step of the procedure consists of propagating information backwards through the NN (that is, from the output nodes to the input node).

Alternative approaches to back-propagation exist, but are less efficient. One of the simplest consists of using finite differences (see 'finite differences' and [3] p. 147).

There are several limitations to the use of back-propagation:

- Back-propagation is not normally used to train RBFs, other techniques exist (see section 4.3.2, see also 'clustering techniques' and 'singular value decomposition').
- Back-propagation cannot be applied directly to recurrent NNs and must before be extended. It is then called '*back-propagation through time*'.
- Back-propagation assumes that the activation function is differentiable (e.g. it cannot be applied when threshold activation functions are used, see 'threshold activation function').

Note: The term 'back-propagation' is often used to refer to the gradient descent algorithm. Although this practice is very common, it is improper. Indeed, even if the gradient descent algorithm uses the back propagation procedure to compute the gradient information, the back-propagation procedure can also be applied with any first or second order optimisation algorithm.

---

[66] Back-propagation procedure can also be used to compute the Hessian (see 'Hessian matrix').

- **Back-propagation framework**

We term back-propagation framework, the one in which the classical approach of NN development is applied, that is, using a non-linear optimisation algorithm based on the back-propagation procedure.

Note: Other frameworks exist to train NNs (e.g. the Bayesian framework, see 'Bayesian framework'), however so far, the back-propagation framework has been used the most commonly.

- **Batch learning**

When applying a learning algorithm to determine the parameters of a model, we speak of batch learning if the parameters are updated only once all the training data points have been presented (see also 'sequential learning' and 'gradient descent').

- **Bayesian posterior probability**

See 'Bayes' theorem'.

- **Bayesian framework**

Assume that we have some data generated according to a probabilistic model whose parameter values are unknown. To determine the parameter of this model, there are 2 main classes of statistical approach: the frequentist approach and, the Bayesian approach.

A frequentist approach consists of estimating a single value for the parameter by using the data. For example, if we consider the problem of tossing a coin, the parameter to be determined could be the probability of obtaining head. Given a set of coin tosses, to estimate this probability, a frequentist typically evaluates the proportion of heads and uses this proportion as an estimate of the probability.

By contrast, a Bayesian approach does not try to obtain a single estimate of the parameter but a distribution over all the possible values of the parameter. The principle is rather simple: the first step consists of defining a prior distribution over the possible values of the parameter. If we have a belief about the likely possible values for the parameter, this can be expressed through this prior (e.g. for the problem of tossing a coin, if the coin seems balanced and if we do not have any additional information concerning the coin, it is natural to choose a prior distribution for the probability of a head symmetric around 0.5 and with a maximum at 0.5). If we do not have any belief at all about the likely value of the parameter then this can be expressed by using a non-informative prior (e.g. for the problem of tossing a coin, a uniform distribution on [0,1]). This prior can be used as a way of incorporating prior knowledge in the model (see section 5.4.3).

The second step consists of observing the results of several experiments and using these results to update the prior distribution, that is, giving a posterior distribution. Intuitively, this means that we had an belief concerning the possible values of the parameter, and that we have updated our belief after having observed the results of experiments. This update is done according to the laws of probability, and is performed by applying Bayes' theorem (see 'Bayes' theorem'). This Bayesian view of learning is generally referred as '*Bayesian inference*'.

One of the main advantages of the Bayesian approach over the frequentist approach is that the result is a distribution over the possible values for the parameter. Indeed, the frequentist approach tries to estimate the expectation (i.e. a single value) of the parameter by using a finite number of experiments. This can lead to erroneous conclusions. For example, let us consider again the problem of tossing a coin, and let us suppose that in two tosses we obtain two heads. If the probability of obtaining heads is evaluated using a frequentist approach, then the conclusion will be: the probability of obtaining head is equal to 1, which is likely to be wrong.
On the other hand, if a Bayesian approach and a non-informative prior are used, then the resulting distribution will be peaked around 3/4 and the shape of the distribution obtained will be relatively smooth (see [20] pp. 6-7). This distribution is a more

realistic result since it expresses the uncertainty in the result due to the low number of experiments. Moreover, this means that no value (and in particular the true value of the parameter) is discarded and that, roughly speaking, a degree of belief is associated to each possible value.

One of the main criticisms of the Bayesian approach is the use of the prior distribution to initiate the learning. Indeed this prior influences the results, whereas it may be chosen more or less arbitrarily.
Nevertheless, note that for an infinite number of data examples both approaches converge to the same result.

With the Bayesian approach, to make predictions, all the possible values of the parameter are taken into account. The value predicted is a weighted average of the predictions for all the possible parameter values and the weighting coefficients are the degree of belief associated with each parameter value. This is formally expressed by: $p(y|D) = \int p(y|w)p(w|D)dw$, where D is the data, $p(y|w)$ is the prediction made by the model for the parameter w and $p(w|D)$ is the posterior distribution of w (i.e. the degree of belief in w given the data). This integration is usually referred as *marginalisation*. Note that thanks to this principle of marginalisation, Bayesian techniques are a very natural implementation of diversity (see section 5.4.4)[67] since even the least likely models are taken into account to provide predictions.

If we apply these principles to NN technology, the parameters to be determined are the variables of the NN unit, that is, the NN architectures to be used and the associated weight values. Here again the same distinction between the frequentist and the Bayesian approaches can be made. Roughly speaking, the frequentist approach corresponds to the selection of a single NN configuration in the maximum likelihood framework (i.e. selecting the best architecture with the best set of weights with respect to the generalisation error), whereas for the Bayesian approach the goal is to find the distribution over the sets of weights, and over the architectures considered.

[67] In the NN context it can be related to the notion of committee of NNs (see 'committee of NNs').

There are two main implementations of Bayesian techniques. These implementations correspond to two ways of carrying out marginalisation. The first one is based on a Gaussian approximation for the posterior distribution of the parameters (see [3] pp. 397-398, [19], [28]) whereas the second one uses Monte Carlo methods (see [20] pp. 61-109).

Last but not least, note that the main drawback of the Bayesian techniques is probably that they may be computationally demanding in operation. This can make them inapplicable when a high safety integrity level is required and real time requirements exist (see also 'Bayesian error bars').

- **Bayesian error bars**

We define Bayesian error bars to be error bars used in the Bayesian framework (see 'Bayesian framework').

Usually, in the maximum likelihood framework (see 'maximum likelihood principle', see also 'back-propagation framework'), to provide error bars over the NN predictions, the variance V of the NN predictions is estimated[68], and then, assuming that the distribution of the targets is Gaussian (with a mean equalling the NN prediction and a variance equalling V), error bars are provided.

In principle, in the Bayesian framework the approach is different. As the application of Bayes' theorem gives the posterior distribution for the weights and, as in theory, the distribution of the weights being known the distribution of the targets can be determined, "exact" error bars can be given. However, in practice the things are not so simple. Indeed:

- the quality of the approximation of the posterior distribution of the weights to the true posterior depends on the initial prior distribution that has been chosen and on the number and quality of the data points used.

- it is generally impossible to determine "exact" error bars analytically,

[68] It is often carried out by using the Hessian and gradient information at a given input point (see 'Hessian' and 'gradient') or by using predictive error bars (see 'predictive error bars').

- Although numerical sampling techniques can be used to avoid analytical problems, as the dimensionality of the weight space is generally high (greater than 20), serious problems of accuracy and computation time exist.
Hence, this potential can generally not be exploited fully.
To avoid these problems, some techniques try approximate the posterior distribution of the weights by a function easy to handle. One of the most popular is the Gaussian approximation, that is, approximating posterior distribution by a Gaussian at the peak of the distribution (see [3] pp. 397-398, [19] and [28]).

Finally, note that in the back-propagation framework and for classification problems, assigning error bars to the posterior probabilities produced by the NN unit is highly problematic, two of the main reasons are that:
1) The activation functions associated to the output nodes are often highly non-linear. For example, softmax activation function or, sigmoidal function,
2) The Gaussian assumption for the distribution of the targets does not hold (since the distribution over the posterior probability is generally not symmetric).
In principle, in the Bayesian framework this is a natural problem, which could be solved by using sampling techniques (see appendix C, section C.6).

- **Bayes' theorem**

The Bayes' theorem is defined as: $P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$, where $P(A)$ and $P(B)$ are the prior probabilities, $P(B|A)$ is the conditional probability of B (conditioned by A), and $P(A|B)$ is the conditional probability of A (conditioned by B) and is called the Bayesian posterior probability (see [3] pp. 17-23 and [25] pp. 12-14).
Note: This theorem is a central element in the Bayesian framework (see 'Bayesian framework').

Note: Bayes' theorem can also be applied with density function (instead of probabilities), for example, typically for a classification problem we have:

$$P(C_k|x) = \frac{p(x|C_k)P(C_k)}{p(x)}$$

where $P(C_k)$ is the prior probability that a point belongs to the class C_k, $p(x)$ is the unconditional probability (the distribution density of the input data), $p(x|C_k)$ is the conditional distribution of x, $P(C_k|x)$ is the conditional probability of C_k (conditioned by x) and is called the Bayesian posterior probability.

- **Biased estimate**

An estimate of a parameter is said to be biased when the expectation of this estimate is different from the value of the parameter. In other words, if each experiment provides an estimate of the parameter and if an infinite number of experiments is performed, then, the average value of these estimates will be different from the value of the parameter (see [3] p. 41 and [25] p. 286).

For example, the variance estimate stemming from the maximum likelihood approach is biased. One of the most important implications of this in NN context, is that error bars based on this estimate of the variance are also biased. Nevertheless, one important feature of this bias is that it fades away when the number of data points increases. Thus, if a large number of data points is used to train the NN, the bias can be neglected[69].

The bias for this estimate of the variance is expressed by:

$\varepsilon(\sigma_{ML}^2) = \frac{N-1}{N}\sigma^2$, where σ_{ML}^2 is the estimate of the variance obtained by the maximum likelihood approach, $\varepsilon(.)$ is the expectation operator, N is the number of data points, and σ^2 is the true value of the variance[70].

- **Bias node** (also called 'bias unit', 'threshold node' or 'threshold unit')

---

[69] To be more rigorous; as the amplitude of the error bars depends on the input point considered, if the number of training data points is large, then the bias can be neglected in the area of the data space where the training data density is "high enough" (see section 4.4.2 and 'density function').

[70] When this variance estimate is used to provide error bars on the NN prediction, N is the number of data points in "the neighbourhood of the input point considered (see section 4.4.2 and 'density function').

A bias node is a node that does not receive any connection from other nodes. However this kind of node is fundamentally different from an input node, since it is not associated with any input variable and its value is usually fixed to the constant '1' (see fig. 2, see also 'input node', 'hidden node' and 'output node').

- **Bias weight** (also called 'threshold weight')

A bias weight is a weight connecting a bias node to either a hidden or an output node (see fig. 2, see also 'bias node', 'hidden node' and 'output node'). Unlike a bias node, a bias weight is an adjustable parameter whose value can vary during training.
Note: In case of linear output units, the bias weights of the last weight layer compensates for the difference there is between the mean of the targets and the mean of the NN predictions that would have been obtained without these bias weights (see [3] p. 342).

- **Bootstrapping techniques**

See 'resampling techniques'.

- **Broyden-Fletcher-Goldfarb-Shanno procedure** (also called 'BFGS procedure')

The BFGS procedure is a quasi-Newton optimisation method (see 'quasi-Newton methods', see also [3] p. 288 and [24] p. 397, pp. 426-430).

- **Cascade correlation**

Cascade correlation is a special case of a growing technique (see 'growing techniques', see also [3] pp. 357-359 and [14] p. 160).

- **Categorical variable** (also called 'nominal variable')

A categorical (or 'nominal') variable is one for which there is no order between the different values that it takes (see section 5.3.2.2, see also 'continuous variable' and 'ordinal variable').

Owing to their nature, categorical variables can be the cause of serious problems when used with data-driven models. For instance, (i) they often stem from a subjective human evaluation, which means that they are often intrinsically approximate, and therefore, not always suitable when a high accuracy is required (see section 5.3.2.2, [3] p. 300 and [24] pp. 628-629, see also 'continuous variable' and 'ordinal variable'), (ii) they require specific encoding techniques (see 'disjunctive coding'). For instance, let us suppose a categorical variable having a numeric form. If this variable is presented to the NN without being encoded, the NN will interpret the numeric values of the variable as actual numbers and then will assign an order to these values. But this order is completely artificial and should not be used as an information during the NN training.

- **Clustering techniques**

Generally the distribution of a data set in the data space is far from being uniform and clusters of data points exist. The goal of clustering techniques is to allow the localisation of these clusters (see [3] pp. 187-189, see also 'K-means algorithm').

Note: Clustering techniques are often used to assign RBF centres (see 'radial basis functions'). The idea being that it is better to have the best response of the basis functions where there are the most number of data points.

- **Committee of NNs**

The basic idea of a committee of NNs is that instead of selecting only the best model to provide predictions, several models are selected. The underlying principle is that if the members of the committee are complementary (i.e. roughly speaking, there is at least always one committee member that provides an accurate prediction) then the average performance of the committee will be superior to the average performance of the best model (see [3] pp. 364-369 and pp. 422-424, see also section 5.4.4).

- **Complexity of a NN**

The complexity of a NN can be defined as the number of its free parameters[71] and so, broadly speaking, the number of its weights.

To be rigorous, it should be noted that usually, correlations exist between the weights (e.g. the weights of the second layer are not independent from the weights of the first layer). Hence the actual number of free parameters is in fact lower than the number of weights (see [10] for a further discussion).

Note: Sometimes, the complexity is defined as the number of weights that are significantly different from zero. Then, for a given architecture, the complexity is not constant and varies during the training process (see 'early stopping' and [3] pp. 343-345).

- **Conditional distribution**

See Bayes' theorem.

- **Conditional probability**

See Bayes' theorem.

- **Confidence interval** (also called 'error bars')

See 'error bars'.

- **Confusion matrix**

Confusion matrices are used with classification problems. Such a matrix enables an interpretation of the overall misclassification rate obtained. It gives the percentage obtained for each type of classification and misclassification. The labels for the rows

[71] The free parameters are sometimes also called degrees of freedom.

and the columns of the matrix are the same and correspond to the different classes of the problem.

For example, the element at the i^{th} row and j^{th} column represents the percentage of points of class 'i' that have been (mis)classified as belonging to class 'j', whereas the element at the i^{th} row and i^{th} column indicates the percentage of points of class 'i' that have been correctly classified.

Note: There is no established convention concerning the respective roles of the rows and the columns. Thus, the opposite convention is often used, that is, the element at the i^{th} row and the j^{th} column represents the percentage of points of class 'j' that have been (mis)classified as belonging to class 'i'.

- **Conjugate gradients**

Conjugate gradients algorithm is a second order optimisation method (see 'non-linear optimisation techniques'). One interesting aspect of this algorithm is that it does not require an explicit computation of the Hessian matrix (while being a second order method). Several variations of this method exist and experience seems to indicate that the 'Polak-Ribiere version' is generally superior (see [3] pp. 274-282, [14] pp. 125-127 and [24] pp. 396-397).

- **Contingency table analysis of two distributions**

Contingency table analysis is a statistical technique to detect and measure the (linear or non-linear) correlation between two variables. In theory this type of technique is only applicable to pairs of ordinal or categorical variables, however it can easily be adapted to continuous variables (by discretizing them) (see [12] pp. 29-30, p. 54, [24] pp. 628-636 and [25] pp. 150-157).

Continuous variable (also called 'quantitative variable')

A variable is said to be continuous (or quantitative), if it can be measured as a real number and applying arithmetic operations (such as addition and multiplication) makes sense.

For example, the yearly quantity of oil transported by a tanker is a continuous variable. On the other hand, if the colour of an object colour is measured as 'blue', 'green' or 'red' and then encoded as values '0', '1' and '2', this is not a continuous variable, since it does not make sense to add two values. Usually, continuous variables arise directly from objective measurements. Feed-forward NN models use real valued inputs and outputs, and hence continuous variables cause no special problem (see section 5.3.2.2, [24] pp. 628-629, see also 'categorical variable' and 'ordinal variable').

- **Cost matrix** (also called 'loss matrix')

See 'loss matrix'.

- **Cross-entropy error function**

The cross-entropy error function is in principle the most appropriate for classification problems. It is defined as: $E = -\sum_{i=1}^{N} \left(t_i \ln y(\underline{w}^*, x_i) + (1 - t_i) \ln\left(1 - y(\underline{w}^*, x_i)\right) \right)$, where N is the size of the data set, $\underline{w}^*$ the NN weight vector, $y(\underline{w}^*, x_i)$ the prediction for the i^{th} input data point, and t_i the corresponding target (see [3] pp. 230-240).

- **Cross-validation**

Cross validation is a special case of resampling technique (see 'resampling techniques'). It consists of splitting the data set into N disjoint subsets. For each subset S_i train a model on the remaining N-1 subsets and test the model on S_i. The generalisation performance of a model trained on all the data is then the average test error on S_1, ..., S_N (see [3] pp. 372-375).

Cross-validation is especially convenient when little data is available, as it uses the same data set both to train the NN and to evaluate its generalization performance. On the other hand, its main drawback is that it requires the training to be repeated *N* times, and therefore, may be computationally very demanding.

- **Curvature driven smoothing**

Curvature driven smoothing is a regularisation technique (see 'regularisation'). The basic idea is that overfitting is generally synonymous with high curvature[72] for the function produced by the NN. Hence, for this technique the regularizer is a measure of the curvature (see [3] pp. 345-346).

- **Data density modelling**

The purpose of the data density modelling is to approximate the true unconditional data density using a model and data samples.

There are broadly three main classes of techniques : the non-parametric (e.g. histograms, kernel based methods, K-nearest neighbours), the parametric (e.g. the single Gaussian model, whose parameters are its mean and its variance) and the semi-parametric (e.g. Gaussian mixture models) (see [3] pp. 33-74, see also 'density function').

The main advantage of non-parametric techniques is that they are extremely flexible and, therefore, can potentially give a good representation of the true density function, on the other hand they are generally computationally very intensive in operation and require large storage capacities (see [3] pp. 49-59). Parametric techniques are computationally less demanding but are not very flexible and hence may give a poor representation of the true data density (see [3] pp. 34-39). Semi-parametric techniques (e.g. the Gaussian mixture model) usually represent an advantageous compromise between parametric and non-parametric techniques (see [3] pp. 59-73).

[72] The curvature of a function is given by the second derivatives of the function with respect to its inputs.

- **Data normalisation**

The most common form of data normalisation is to transform the data set so that each variable has zero mean and unit variance. To perform such a normalisation the mean and the standard deviation of each variable are first computed. Then, for each sample point, the mean value is subtracted and the result is divided by the standard deviation value (see also 'data whitening').

There are several advantages to normalise the data, for instance:

- it reduces rounding problems and, therefore, enables to work with a better numerical accuracy (see [24] p. 353),
- it suppresses scaling effects (see section 5.3.3.2 and [3] p. 298),
- it simplifies weight initialisation (see 'weight initialisation').

Note: In principle, the data presented to the NN unit should be normalised (see sections 4.5 and 4.3.3.2).

- **Data space**

The data space is the space spanned by the input variables.

- **Data whitening** (also called 'whitening' or 'sphering')

Data whitening is a special case of data normalisation (see 'data normalisation'). It consists of making variables zero mean and unit variance. However, it is slightly more sophisticated than the classical data normalisation, since it takes into account the possible correlations between data examples. That is, the data is transformed in such a way that the input variables are independent from one another and have unit variance (i.e. unit covariance matrix) (see [3] pp. 299-300).

- **Decision boundary**

When considering classification problems the decision boundary is the area of the data space where it is impossible to decide to which class the data points belong since their

probabilities of belonging to the most probable classes are equal (e.g. equal to 0.5 in a two-class classification problem) (see [3] pp. 23-27).

- **Density function**

By definition, the value of the density function at a data point, measures the data density in the area of this data point. In other words, if the data density function is known then the distribution of the data in the data space is also known (see [25] pp. 261-283, see also 'distribution of a continuous variable').

The integration of the data density over an area of data space gives the probability that a data point belongs to this area. As a density function is normalised, its integration over the whole data space is equal to one. This corresponds to the natural result that all the data points of the data space lie in the data space.

Finally, note that a low data density in an area of the data space does not necessarily mean that there are very few data points in this area, but rather the number of data points in this area is very small compared to the total number of data points available (see also 'data density modelling').

Note 1: The Gaussian density function is one of the most popular.

Note 2: Typically when training a NN, data space areas corresponding to a high data density in the training set are favoured by the training process. Indeed the error function is a sum of terms, such that each term is associated with the prediction error related to one training data point. Consequently, areas of the space with more data points in the training set have a greater contribution in the value of the error function than areas with fewer data points. Accordingly, the minimisation of the training error is mainly focused on these areas of high density[73]. This gives an intuitive explanation to the fact that the amplitude of the error bars in an area is approximately inversely proportional to the training data density in this area.

[73] This can be annoying when very little data is available in the areas of the data space where the required accuracy is the highest (e.g. areas corresponding to risks of entering a faulty state). Measures can be taken to suppress this effect (e.g. by modifying the error function, and weighting each term by a function of the training data density in the corresponding area, see also section 5.3.3.2).

- **Detrending** (also called 'trend removal')

See section 5.3.2.2, see also [16] pp. 32-34, pp. 43-45 and p. 173.

- **Dimensionality of a space**

The dimensionality of a space is the number of independent vectors that span the space.

In the NN context, the dimensionality of the space considered is usually the dimensionality of the data space, that is, the number of input variables (see also 'intrinsic dimensionality of a data space').

Note: When considering the problem of minimising an error function, it is generally referred to the dimensionality of the weight space, that is, the number of weights in the considered NN.

- **Disjunctive coding** (also called '1-of-c coding')

Disjunctive coding is used to encode categorical variables (see 'categorical variable'). Given a categorical variable V , the disjunctive coding consists of creating a new variable for each value of V. These variables are Boolean, such that for each value of V there is one and only one Boolean variable that is set to '1' (see [3] p. 300 and [25] pp. 217-218).

- **Distribution of a continuous variable**

The distribution of a continuous variable is closely related to the notion of density function (see 'density function'). The basic distinction between both is that a density function takes as an input a value of the variable and provides as an output a measure of the density for this value. If we see the distribution of a variable as a function, we can say that the distribution takes as an input a variable and produces as an output a density function for this variable. For instance, when a variable is said to have a Gaussian distribution, this means that the density function associated is a normalised Gaussian function.

- **Early stopping**

Early stopping is an alternative to regularisation techniques to reduce overfitting problems (see 'overfitting'). The intuitive idea is to stop the NN training process before the NN starts to learn too accurately the training data (i.e. when the generalisation ability of the NN starts to decrease, that is, when the validation error starts to increase). This can be achieved by monitoring the evolution of the generalisation error (see fig 5.a and [3] pp. 343-346).

A more precise explanation relies on the fact that during the training process, for a given NN architecture, the actual number of free parameters (that is, roughly speaking, the number of weights whose value is significantly different from 0) increases and so do the model complexity and the risk of overfitting. Hence, the training process should be stopped before this number becomes too large.

Note: Often the term 'cross-validation' (see 'cross-validation') is used to refer to 'early stopping', yet both techniques are different.

- **EM algorithm** (or 'Expectation Maximisation algorithm')

The EM algorithm is an iterative procedure to solve non-linear optimisation problems. To use the EM algorithm, the problem to be solved is first expressed in terms of a set of coupled recursive equations. Then, the EM algorithm can be applied to find an approximate solution of the problem (i.e. an approximation of the fixed point of the equations).

The algorithm alternates calculation of posterior probabilities and maximisation steps and is guaranteed to converge to a local minimum. It is efficient when the maximisation step is easy (e.g. solution of linear equations) models. Finally, note that it does not use gradient information, and that it is often used to fit Gaussian mixture (see [3] pp. 65-72 and p. 301, see also [5]).

Finally, note that the EM algorithm is a very general and very important technique to determine maximum likelihood estimate to complete data (see section 5.3.2.2 and [17]).

- **Error bars** (also called 'confidence interval')

When an estimate of a parameter is given, error bars for this estimate can sometimes be determined. These error bars constitute a range of values around the estimate where the true value of the parameter is likely to be. This does not ensure that the true value lies within the error bars, but gives a probability that the true value lies within them.

For instance, in the case of Gaussian error bars, if we consider two standard deviation error bars around the estimate (i.e. the interval $[Y_e-2.\sigma, Y_e+2.\sigma]$, where Y_e is the estimate of Y, and σ the standard deviation of this estimate), then there is a probability of approximately 0.95 that the true parameter value (Y) lies inside the error bars.

Note: For regression problems, in-operation error bars characterise the conditional distribution of the targets and express the uncertainty concerning the output values produced by the NN. Roughly speaking, this uncertainty has two main origins: the uncertainty on the weights (due to a lack of training data points in the vicinity of the input point) (see [3] pp. 399-401) and the uncertainty due to the noise on the targets[74].

- **Expectation Maximisation algorithm** (or 'EM algorithm')

See 'EM algorithm'.

- **Extrapolation**

See section 4.4.2.

- **Fast Fourier Transform** (or 'FFT')

A computationally more efficient version of the Fourier transform (see 'Fourier transform', [16] pp. 174-175 and [24] pp. 496-608).

---

[74] In fact the distinction between both is not so clear cut since the noise makes the task of the NN harder, and thus contributes in the weight uncertainty (see also fig. 11)

- **Feature extraction techniques**

When data is collected, the number of inputs variables is often large. However, owing to the curse of dimensionality (see section 4.7), not all these input variables should be used directly as such by the NN unit.

Accordingly to develop a successful NN function the number of inputs has to be reduced. This can be done either by discarding input variables judged as less relevant, or creating a smaller set of new input variables from the initial one. This selection of the variables to be used as NN input variables is called feature extraction (see also 'principal component analysis', 'Fisher's discriminant', 'Fourier transform', section 5.3.2 and [3] pp. 304-318).

- **Feed-forward NN**

A feed-forward NN is a NN whose nodes can be ordered in such a way that, first, there is no connection joining node 'j' to node 'i' if 'j' appears after 'i' in the order, and secondly, there is no connection joining any node to itself.

Intuitively this means that there is no (direct or indirect) loop in the NN architecture.

Note: An example of feed-forward NN is given in figure 2. The NN presented has two layers of weights, three input nodes, two hidden nodes, three output nodes and two bias nodes.

- **Finite differences**

The use of finite differences is an alternative to back-propagation procedure (see 'back-propagation') to evaluate gradient information (see 'gradient'). Its main drawback is its relative inefficiency compared to back-propagation. On the other hand its implementation is straight-forward. In consequence, it can provide a powerful means to check the correctness of any implementation of the back-propagation procedure.

There are mainly two forms of finite differences; the simple form, and the symmetrical one. Generally the symmetrical form provides more accurate results and is defined as:

$$\frac{\partial E}{\partial W_{ij}} = \frac{E(W_{ij}+\varepsilon) - E(W_{ij}-\varepsilon)}{2\varepsilon} + O(\varepsilon^2)$$, where $O(\varepsilon^2)$ is the accuracy order

(see [3] p. 147, p. 158 and [6] p. 13).

- **First layer of weights**

In the case of a multi-layer NN, the first weight layer is the one that connects the input nodes to the first layer of hidden-nodes.

According to the type of NN considered, the first weight layer has a different functional interpretation: For a RBF, the weight values of this layer represent the positions of the basis functions (see 'formal definition of the outputs of a RBF'), whereas for a MLP, this weight layer is an ordinary one and hence constitutes a set of linear weighting coefficients (see 'formal definition of the outputs of a MLP' and section 4.3.1).

- **First order methods**

See 'local quadratic approximation'.

- **Fisher's discriminant**

See 'canonical variates'.

- **Canonical variates**

Canonical variates is a technique aimed at reducing the dimensionality of the data, in such a way that as little information as possible is lost. This technique is specifically intended to classification problems.

As PCA (see 'principal component analysis'), this is a linear technique that finds a set of linear components that minimise the loss of information. However, unlike PCA, the

useful information to be preserved is not the variance of the data, but the discriminant information between classes. Consequently, to select the best linear components, the Canonical variates uses a criterion based on the overlapping between classes (see [3] pp. 105-112, p. 227).

Canonical variates is principally used for high dimensional data visualisation (see 'visualisation techniques') and feature extraction (see 'feature extraction').

Note that when canonical variates is applied to two-class problems it is often termed *Fisher's discriminant.*

- **Formal definition of the outputs of a MLP**

$y_k = h\left(\sum_j w_{kj} g\left(\sum_i^d w_{ji} x_i\right)\right)$, where y_k is the kth output of the NN, h and g are activation functions, the w_{ji}'s the weights of the first weight layer, the w_{kj}'s the weights of the second weight layer, and $\underline{x}$ an input vector in the form $(x_1 x_2 \ldots x_d)$ (see also fig. 2).

Note that d is called the dimensionality of the data space (see 'dimensionality of a data space'). Note also that g is generally a sigmoidal activation function (see 'sigmoidal activation function') and h is usually the identity function for regression problems and the logistic activation function for classification problems.

- **Formal definition of the outputs of a RBF**

$y_k = \sum_j w_{kj} \Phi_j(\underline{x})$, where y_k is the kth output of the NN, the w_{kj}'s the weights of the second weight layer, and $\underline{x}$ is an input vector, and Φ_j a basis function.

- **Fourier transform**

The Fourier transform is based on the fact that any time series can be approximated to arbitrary accuracy by a linear combination of sinusoidal functions (e.g. in electricity

pulse signals can be approximated by linear combinations of sinusoidal functions) (see [16] p. 161 and [24] pp. 496-608, see also 'fast Fourier transform').

When considering stationary time series the Fourier transform can be used as a feature extraction means (see 'feature extraction') or as a smoothing technique (see 'smoothing').

- **Genetic algorithms**

The main idea is based on an analogy with the evolution theory where the species that survive are those the most adapted to the environment. In the NN context, the configurations (i.e. the set of weight values) are coded as binary strings (the chromosomes). The best chromosomes are then selected (according to the error for the corresponding NNs) and pairs of chromosomes are combined to form new chromosomes, and so on... Thus in principle, the population is gradually improved.

The main advantage of genetic algorithms is that they discount the gradient information and, therefore, are not easily fooled by the problem of local minimum (see section 4.4 and section 5.5). Nevertheless, genetic algorithms require a large storage capacity, and as they discount the gradient information, they are computationally demanding. Hence, genetic algorithms are often used in combination with techniques taking into account the gradient information. In [14] the authors consider that "an initial genetic search followed by a gradient method might be an appropriate compromise" (see [13] and [14] pp. 128-129, p. 157).

- **Gradient** (also called 'jacobian matrix')

The gradient of a function is the matrix of its first partial derivatives with respect to each of its input variables. In the NN context, the function is generally the error function and the "input variables" are the weights of the NN.

Note that most of the multi-dimensional non-linear optimisation methods (see 'non-linear optimisation techniques') make use of the gradient information. The basic idea being that at the minimum of a function the gradient is equal to zero. Hence, these

methods try to find the minimum of the error function by cancelling out the gradient (see section 4.4).

Finally, note that if the NN unit has a single output this matrix is a vector, and is often called 'gradient vector' or 'jacobian vector'.

- **Gradient descent**

Gradient descent algorithm has been, and still is, one of the most popular optimisation algorithms.

The principle of gradient descent is, at each step, to move in the direction of the negative gradient information[75] (i.e. the direction of steepest descent[76]). The length of the move is determined by a learning rate.

Usually the learning rate is a constant. If this learning rate is too high then the algorithm may diverge (overshooting the solution), whereas if it is too small the algorithm may need a tremendous number of steps to converge to a minimum (since at each iteration the NN makes very little progress). Hence, when applying gradient descent algorithm, the main difficulty is the choice of this learning rate.

Several variations of the standard gradient descent have been proposed to reduce these problems (e.g. momentum, adaptive learning rate, see [3] pp. 263-272), however, none of them significantly improves the efficiency. Second order methods are generally preferred since being usually much more efficient (see ' non-linear optimisation techniques').

Note 1: Two ways of applying the gradient descent exist: the sequential version performs an update of the weights after the presentation of each data point, whereas the batch version performs a global update of the weights once all the points of the data sets have been presented to the NN (see [3] p. 263 and [14] p. 119). In [14] the authors consider that the sequential version is generally the most efficient of the two.

---

[75] Most of the time the gradient information is computed using back-propagation procedure (see 'back-propagation').

[76] In the literature there are sometimes terminology problems concerning the steepest descent. Indeed, sometimes it is considered as another name for gradient descent [3], at other times it is a gradient descent procedure where there is no constant learning rate anymore but with a 'built-in line search optimisation procedure' [14] (see also 'steepest descent').

- **Growing algorithms**

Growing algorithms attempt to optimise the network architecture by starting with a relatively small NN and adding new weights or nodes during the training process (see [3] pp. 353-364), see also 'pruning algorithms').

Note: More advanced techniques are usually much more efficient.

- **GTM algorithm** (also called 'Generative Topographic Mapping')

The GTM algorithm is based on the same idea as the Kohonen map (see 'Kohonen topographic mapping'), that is, trying to represent the data of a high dimensional space into a low dimensional space (the latent space). The GTM algorithm overcomes most of the deficiencies of the Kohonen map (see [8], see also [5]). For instance: (i) as the GTM algorithm is based on the EM algorithm, it is guaranteed to converge, (ii) unlike the Kohonen map, the GTM algorithm can be used as a data density modelling technique (see 'data density modelling').

Note: As the GTM algorithm is often used as a visualisation technique (see 'visualisation techniques'), the dimensionality of the latent space is generally two and sometimes three (see also 'dimensionality of a data space' and 'intrinsic dimensionality of a data space').

- **Hessian matrix** (also called 'Hessian')

The Hessian is the matrix of the second partial derivatives (see [3] pp. 150-160). In the NN technology context, it is generally the second partial derivatives of the error function with respect to the weights, that is: $H(W_{ij}, W_{kl}) = \frac{\partial E}{\partial W_{ij} . \partial W_{kl}}$

The Hessian matrix has various important applications in NN technology, for instance: (i) it is essential for second order optimisation algorithm (see 'non-linear optimisation techniques' and 'linear quadratic approximation'), (ii) its inverse can be used to assign in-operation error bars (see section 4.6) to the prediction provided by a NN.

- **Hidden node** (also called 'hidden unit')

A hidden node is a node that is neither an input nor an output node and, therefore, hidden nodes exist only for multi-layer NNs (see fig. 2, see also 'input node', 'output node', and 'bias node').

- **Hyperbolic tangent activation function**

See 'sigmoidal activation function'.

- **Input node**

Each input node corresponds to one input variable. Such a node simply propagates the values of its associated input variable unchanged (see fig 2, see also 'hidden node', 'output node', and 'bias node').

- **Interpolation**

See section 4.4.2.

- **Intrinsic dimensionality of a data space**

One should distinguish the dimensionality of a data space (see 'dimensionality of a data space') and the intrinsic dimensionality of this data space. The intrinsic dimensionality of a data space can be defined as the minimum number of variables needed to specify non-ambiguously any data point in that space. The intrinsic dimensionality of a data space is no greater than the dimensionality of the data space (see [3] pp. 313-314).

For instance, let us consider a circle in a three dimensional space. In that case, the intrinsic dimensionality is the number of variables needed to specify any data point on this circle. The dimensionality of the data space is three, yet, if we consider that this circle is entirely defined in a plane, we can define this circle using only two variables (two co-ordinates in this plane). Furthermore, given that the centre of the circle is a

constant point in the plane, if we choose another constant point belonging to the circle as a reference point, then we can define any point P of the circle using only one variable: the angle between the lines joining the centre and the reference point and the centre and P.

Thus we see that although the dimensionality of the data space is three, the intrinsic dimensionality of the circle is in fact one.

Note: This notion of 'intrinsic dimensionality' is closely related to the one of 'feature extraction' (see 'feature extraction techniques').

- **Inverse problem**

A problem is said to be inverse, if, given a function (f) and an output value (y) for this function, the problem consists of finding what is (or what are) the input(s) x_{yi} such that $f(x_{yi}) = y_i$. Typically, for a single value of y there may be several possible values for x_{yi}. Modelling inverse problems with a NN requires extra precautions (see section 5.3.2.2).

- **Kohonen topographic mapping** (also called 'Kohonen map', 'Kohonen network', 'SOM', 'self organising map', or 'self organising feature map')

The Kohonen map tries to find a representation of high dimensional data in a lower dimensional space (the latent space), in such a way that this representation preserves the data structure as much as possible (see [14] p. 218, pp. 236-246).

The Kohonen map is often used as a visualisation technique (see 'visualisation techniques'), hence the dimensionality of the latent space is generally two (sometimes three).

Nevertheless, the Kohonen map suffers from various deficiencies. For example, (i) it can not be used as a data density modelling technique (see 'data density modelling techniques') (ii) there is no guarantee of convergence, (iii) the smoothness of the map has to be arbitrarily chosen (e.g. by using prior knowledge).

Note: the Kohonen map is also sometimes used as a clustering technique (see 'clustering techniques') to assign the centres of RBFs (see [3] pp. 188).

Note 2: The GTM algorithm (see 'GTM algorithm') is a principled alternative to Kohonen map that overcomes most of its deficiencies. Note that it has been shown that the Kohonen map can be seen as an approximation of the GTM algorithm.

- **Linear unit**

We term linear unit, any unit (i.e. input, hidden, or output unit) having a linear activation function (in general the identity function) (see also 'activation function').

- **Line search**

Line search means: optimisation of a function in a one input dimensional space and a one dimensional target space (see [3] pp. 272-274 and [24] pp. 394-412).

- **Local quadratic approximation**

The local quadratic approximation of a function in the vicinity of a point in its input space, corresponds to a Taylor expansion of order two in the vicinity of this point.

Such a Taylor expansion is supposed to provide a good approximation of the behaviour of the function in the vicinity of the point considered. When dealing with non-linear functions this approximation is very convenient. Indeed a Taylor expansion has a polynomial form, hence it benefits from all the analytical properties of polynomials (e.g. finding the minimum of a second order polynomial is straightforward) (see [3] pp. 257-259 and [24] pp. 425-426).

This approximation of the real function may be very rough. That is why practical optimisation algorithms are not entirely based on it.

Note 1: For NNs the function is generally the error function, and the input space is the weight space.

Note 2: Optimisation methods using Taylor expansion of order two are called *second order methods*, whereas methods using an expansion of order 1 are called first order

methods (see 'non-linear optimisation techniques'). Thus, a second order method uses both gradient and Hessian information, whereas *first order methods* use only gradient information (see also 'Hessian matrix', and 'gradient').

- **Logistic activation function**

See 'sigmoidal activation function'.

- **Loss matrix** (also called 'cost matrix')

Loss matrices are used with classification problems. Such a matrix define the loss associated with each type of misclassification. Indeed, the different types of misclassification may have dramatically different consequences. For instance for health screening, it is less grave to diagnostic a cancer when the patient is healthy than to state the patient is healthy when he does have a cancer. The loss matrix can be then used to weight the conditional probabilities in a simple way to arrive at the optimal decision. The class with the smallest expected loss (given by the product of the conditional probability and their associated misclassification loss) is the best prediction (see [3] p. 22, see also section 5.4.2.2).

Note: The elements of a loss matrix are examples of prior knowledge.

- **MAR data** (or 'Missing At random Data')

See 'missing at random data'.

- **Maximum likelihood principle**

Given a data set D and a probabilistic model having θ as a parameter, the likelihood of θ ($L(\theta)$), is a function of θ and D and is defined as $L(\theta) = P(D|\theta)$. That is, the likelihood that the model whose parameter is θ has generated D.

Then, the maximum likelihood principle consists of finding the value of θ that maximises the likelihood function (see [3] pp. 39-42, p. 195 and [25] pp. 301-302).

For NN technology this typically means trying to find the best architecture with the best set of weights.

Note: Most usual error functions (e.g. sum of squares error function, cross-entropy error function, see 'sum of squares error function' and 'cross-entropy error function') are motivated by the maximum likelihood principle.

- **MCAR data** (or 'Missing Completely At Random data')

See 'missing completely at random data'.

- **Misclassification**

For classification problems, a misclassification corresponds to assigning an input data to a wrong class (see also 'loss matrix' and 'confusion matrix').

- **Missing At Random data** (or 'MAR data')

We speak about MAR data when the probability of missing value is dependent on the input variables but not on the output variables (see [17] pp. 14-17).

- **Missing Completely At Random data** (or 'MCAR data')

We speak about MCAR data when the probability of missing value is independent of the input variables and the output variables. In this case the observed values for the outputs form a random subsample of the sampled values of the outputs (see [17] pp. 14-17).

- **Mixture density networks**

The mixture density network model is a NN model that has been designed to represent general conditional probability density and in particular multi-modal distributions. This makes it suitable to model of multi-valued functions. This model is the combination of

a NN and a parametric density model (e.g. Gaussian mixture model, see 'data density modelling techniques'). The role of the network is, for each input data point, to predict the appropriate values of the density model parameters. Given these values, the density model predicts the conditional distribution of the outputs for the considered input data point (see [6]).

Note: As the predictions of the NN function are given by a data density model (that is by definition trained by using unsupervised techniques), we have considered mixture density networks as falling outside the scope of this MSc project (see section 2.3, see also section 4.1).

- **Mixture of experts**

The characteristics of a function (e.g. its smoothness) may significantly differ according to the area of the input space considered. The basic idea of the mixture of experts is to assign one NN (an expert) to each of the these areas. In this way, instead of having a single NN that tries to satisfy contradictory requirements, for each characteristic area of the data space, there is one specialised NN. This enables to improve the accuracy (see [3] p. 214 and pp. 369-371, see also section 5.4.2.2).

Note that generally there is one (or several) NN (called the *gating NN*) that, given an input, selects the appropriate NN (i.e. the appropriate expert) to predict the value of the function for the considered input (see [27]). This decomposition of the input space is not determined by hand but found automatically during the training process.

- **MLP** (or 'Multi-Layer Perceptron')

See 'multi-layer perceptron'.

- **Momentum**

To improve gradient descent algorithm (see 'gradient descent') a momentum term is added. This momentum increases the inertia of the algorithm and, thereby, reduces the

number of useless oscillations. This results in a greater efficiency (see [3] pp. 267-268 and [14] p. 123).

- **Multi-layer perceptron** (or 'MLP')

See section 4.3.1, [3] pp. 116-163 and [14] pp. 115-162, see also 'formal definition of the outputs of a MLP'.

- **Newton's method**

Newton's method is a second order optimisation method (see 'non-linear optimisation techniques'). It is not very much used because it is computationally very demanding, since it requires the computation of the Hessian and its inverse (see [3] pp. 285-287).
Note: Quasi-Newton methods are computationally more efficient versions of the Newton's method (see 'quasi-Newton methods').

- ***n*-layer NN**

A n-layer NN, is a feed-forward NN with n layers of weights.

Note: Sometimes in the literature, 'n' corresponds to the number of layers of nodes. This convention is arguable, since it considers a simple perceptron (see 'perceptron') as a multi-layer NN.

- **NN function**

See section 4.5, see also 'NN unit'.

- **NN unit**

A NN function is constituted of three units: the pre-processing unit, the post-processing unit and the NN unit (see section 4.5). The NN unit is the unit that

contains the NN technology, that is, one or several NNs (e.g. committee, mixture of experts, see 'committee of NNs' and 'mixture of experts').

- **Non-linear optimisation techniques**

Non-linear optimisation techniques are aimed at finding the minimum (or the maximum) of a non-linear function (e.g. an error function). They can be roughly be divided in three categories: first order methods, second order methods (see [3] pp. 253-292, [14] pp. 125-129 and [24] pp. 394-455), and others (e.g. genetic algorithms, see 'genetic algorithms').

Even if second order methods are generally more efficient than first order methods, this is not always the case and it is very problem dependent. In general second order methods converge more quickly, however they cannot escape from local minima whereas gradient descent (for instance) can. In addition, there are currently very few quantitative results measuring the relative efficiency of the different optimisation methods for real-world problems. This problem has not yet been studied thoroughly and, this finds expression in very few relevant related publications [23].

Examples of first order methods: all the versions of gradient descent (see 'gradient descent').

Examples of second order methods: conjugate gradients, scaled conjugate gradients, Newton's methods, quasi-Newton methods, the Levenberg-Marquardt algorithm (see 'conjugate gradients', 'scaled conjugate gradients', 'Newton's methods', 'quasi-Newton methods' and 'Levenberg-Marquardt algorithm').

Note: These methods are generally used for MLP training. For RBFs (see 'radial basis functions') specific techniques exist.

- **Novelty detection**

The reliability of the predictions made by a NN function depends on the novelty of the input considered. In other words, if the input is novel compared the data points the NN function has been trained on, then NN function is forced to extrapolate and,

therefore, is not dependable. Hence, a novelty detection system is aimed at detecting novel inputs (see [7], [26], see also section 4.4.2.).

- **Number of symmetries in a NN**

Let us consider a fully connected two layer NN, with M hidden units. As usually all the hidden units have the same activation function and that this function is odd, it can be easily shown (see [3] pp. 132) that a two layer NN with M hidden units has a symmetry factor of $2^M.M!$ That is, for each weight configuration there are $\left(2^M.M!-1\right)$ other different weight configurations that are equivalent.

Consequently, for a given problem and a two-layer NN with M hidden units there are at least $2^M.M!$ equivalent global minima for the error function.

Note: For instance for a two-layer NN with 5 hidden units, there are at least 3840 equivalent global minima!

- **OAR data** (or 'Observed At random Data')

See 'Observed at random data'.

- **Observed At Random data** (or 'OAR data')

We speak about OAR data when the probability of missing value is independent of the input variables (see [17] pp. 14-17).

- **Ordinal variable**

An ordinal variable takes on a finite set of possible values[77]. These values are ordered, but the normal arithmetic operations and comparisons do not apply. Typically, such variables arise through the discretization of an underlying continuous variable (for example, the Beaufort scale for wind speed). They are usually mapped onto numeric

[77] This set is usually small: less than 20 different values.

values with the correct ordering (see section 5.3.2.2, [24] pp. 628-629, see also 'categorical variable' and 'continuous variable').

- **Outlier**

Roughly speaking, an outlier is a data point which has an atypical position is the data space with respect to the other data points. In general outliers are not generated by the system to be modelled, and result for instance, from errors during data acquisition. Note that giving an exact definition of what is an outlier is a difficult problem; the same kind of problem that exists when trying to distinguish between interpolation and extrapolation (see section 4.4.2).

- **Output node** (also called 'output unit')

An output node is a node whose outputs is an output of the NN (see fig. 2).

- **Overfitting**

Typically a NN is said to overfit the training data when the error obtained on the training set is very low, whereas the testing error is high. Overfitting is synonymous with poor generalisation capabilities (see section 4.4.1).

- **Oversmoothing**

See 'smoothing'.

- **PAC learning theory**

See section 4.6.

- **Partition**

A partition of a set S is a set of subset of S, such that the reunion of the subsets is equal to S, and every pair of subsets has null intersection.

- **Perceptron**

A perceptron is a NN with a single layer of weights. In consequence, it can only be used to solve linear problems (see also 'multi-layer perceptron').

- **Polak-Ribiere's formula**

The Polak-Ribiere's formula provides a variation of conjugate gradients algorithm (see 'conjugate gradients'). Experience has shown that this variation is sometimes significantly better than the others (e.g. Hestenes-Stiefel's or Fletcher-Reeves' variation) (see [3] pp. 280-282, [14] p. 126 and [24] p. 396, p. 422).

- **Posterior probability**

See 'Bayes' theorem'.

- **Post-processing**

See section 4.5 and [3] pp. 296.

- **Predictive error bars**

We term 'predictive error bars' error bars produced by an extra NN that has been trained to predict the standard deviation of the error for the predictions provided by the NN function.

This can be carried out by measuring for each input point x^{test}_i of the test set, the absolute value of the error (e^{abs}_i) for the corresponding prediction. Then, another NN

is trained on the data set formed by the pairs (x^{test}_i, e^{abs}_i) (the x^{test}_i's being used as inputs and the e^{abs}_i's as targets).

- **Pre-processing**

See section 4.5, [3] p.6, pp. 296-298 and [14] p. 144, p. 181, see also 'NN unit'.

- **Principal Component Analysis** (or PCA)

PCA is a technique aimed at reducing the dimensionality of high dimensionality data in such a way that as little information as possible is lost. PCA is a linear technique, therefore, it only captures linear structure in the data.

In the NN context, the two main applications of PCA are high dimensional data visualisation (see 'visualisation techniques') and feature extraction (see 'feature extraction') (see [3] pp. 310-313, pp. 454-456 and [25] pp. 159-186).

Note: PCA is generally used for regression problems. For classification problems specific techniques exist (e.g. Fisher's discriminant, see 'Fisher's discriminant').

- **Pruning algorithms**

One of the fundamental problems when designing a NN function, is to choose an appropriate complexity. The traditional method consists of testing several NN architectures corresponding to different complexities, and select the best one among them. This technique is computationally very demanding, since it must be performed as many training processes as there are architectures tested. Pruning algorithms start from a relatively complex NN (i.e. a NN with a large number of nodes and weights) and allow the suppression of weights and nodes during the training process (see [3] pp. 353-364, see also 'growing algorithms'). Obviously, criteria are required to determine which weights or nodes should be deleted (see [3] pp. 359-364, see also [10]).

Note: Sometimes pruning algorithms and weight decay techniques are combined (see 'weight decay' and [14] pp. 157-158).

- **Quantitative variable** (also called 'continuous variable')

See 'continuous variable'.

- **Quasi-Newton methods**

Quasi-Newton methods are second order optimisation techniques (see 'local quadratic approximation'). They rely on the same principle as Newton's method (see Newton's method'), but they use an approximation of the inverse of the Hessian that do not require the explicit computation of the Hessian and its inverse. Hence, they are much more efficient that Newton's method (see [3] pp. 287-290, and [24] p. 397, pp. 425-430).

- **Radial basis functions** (or 'RBF')

See section 4.3.2, [3] pp. 164-193 and [14] pp. 143, pp. 248-250, see also 'formal definition of the outputs of a RBF', 'clustering techniques' and 'singular value decomposition'.

- **Random training data set**

A random data set, by opposition to a rational data set (see 'rational data set'), is a set for which there is no particular concentration of the data in the sensitive areas of the data space (see section 4.7.2, section 5.4.4.1 and [22]).

- **Rational training data set**

A rational training data set is a data set that is focused on sensitive areas of the data space. For instance, for a classification problem, a rational training set would typically include a large proportion of data points near to the decision boundaries (see section 4.7.2, section 5.4.4.1 and [22], see also 'random data set').

- **RBF** (or 'Radial Basis Functions')

See 'radial basis functions'.

- **Recurrent NNs**

In contrast to a feed-forward NN, a recurrent NN is a NN such that whatever the order of the nodes, there is always at least, one connection joining node 'j' to node 'i' such that either 'i' and 'j' are the same, or 'j' is after 'i' in the order.

Intuitively this means that, there is at least one (direct or indirect) loop in the NN architecture, and that values do not flow in only one direction through the NN (see [14] pp. 163-196).

Note: Recurrent NNs pose specific and difficult problems for validation (e.g. training techniques that are used with feed-forward cannot be applied with recurrent NNs, and owing to the circular aspect of a recurrent NN architecture, the techniques to be used have to cope with stability problems).

- **Regularisation**

Regularisation is a technique aimed at reducing overfitting problems. It incorporates some prior knowledge concerning the smoothness of the function to be modelled into the error function. Thereby, the training process is somewhat reoriented and automatically controls the smoothness of the function produced by the NN.

Such an incorporation is achieved by adding a penalty term (called a *regularizer*) expressing this prior knowledge to the error function.

Thus, if E is the 'unregularised' error function (e.g. sum of squares error function) then the regularised error function is defined as: $E_R = E + \nu.\Omega$, where Ω is the regularizer and ν is a multiplicative coefficient used to control the relative importance of the regularizer[78]. This multiplicative coefficient plays an important role. If its value is too small then the contribution of the regularisation term in the error is

[78] In the back-propagation framework (see 'back-propagation framework') this coefficient is empirically adjusted. In the Bayesian framework (see 'Bayesian framework'), this coefficient is automatically adjusted.

negligible and, therefore, some overfitting risks appear again. If its value is too high then the values of the weights are too constrained, the model is not flexible enough and, therefore, gives a poor representation of the function to be modelled (see [3] pp. 171-175).

Examples of techniques related to regularisation: weight decay, training with noise added to the inputs, soft weight sharing, curvature driven smoothing (see 'weight decay', 'training with noise on the inputs', 'soft weight sharing' and 'curvature driven smoothing').

Note 1: For a two-layer MLP one should always distinguish between the different types of weights (weight of the input layer, weights of the output layer and bias weights) and assign a specific coefficient to each of these weight subsets (see [3] pp. 338-343 and p. 396).

Note 2: As the minimisation of the error function is mainly based on the cancellation of its derivatives, the derivatives of this penalty term should be efficiently computable (see [3] pp. 338-353).

- **Relative error function**

By opposition to, for instance, the sum of squares error function, the relative error function is normalised. That is, its value does not increase with the number of data points and spans $[0,1]$. One important consequence is that its value is interpretable. Thus, if the relative error is nil then this means that the NN predicts perfectly the data set, whereas if it is equal to 1 then the NN always predicts the mean value of the targets of the data set[79] (see [3] pp. 197-198[80], see also section 5.6.3.2).

[79] Every value between zero and one is possible.

[80] In [3] the relative error function is improperly called '*root-mean-square error*'.

It is defined as: $E_{REL} = \frac{\sum_{I=1}^{N} \left\| y(x_i; w^*) - t_i \right\|^2}{\sum_{I=1}^{N} \left\| t_i - \bar{t} \right\|^2}$, where N is the size of the data set, $\underline{w}^*$ the NN weight vector, $\underline{x}_i$ the ith input data point, $y(\underline{w}^*, \underline{x}_i)$ the value predicted by the NN function at $\underline{x}_i$, t_i the target value at $\underline{x}_i$, and $\bar{t}$ the average value for the targets.

- **Relevant data space**

We term relevant data space the area of the space where the data lies.

- **Resampling techniques** (also called 'bootstrapping techniques', or 'recutting techniques')

Given a data set, if data points have to be sampled from the data set to carry out an experiment, then instead of performing only a single sampling and a single experiment, several samplings and several experiments are carried out. From these multiple samplings and experiments various information can be extracted.

The two most interesting ones are the average of the results obtained for the different experiments and their variance.

For instance: by using the variance and the average values, error bars can be computed concerning the results of the experiment (see section 4.6). Resampling techniques can also be used to avoid overfitting through the estimation of the generalisation error (see 'cross-validation').

- **Residual**

A residual is the difference between the prediction provided by the model at a given input point and the target for this input point.

- **Scale conjugate gradients**

Scale conjugate gradients algorithm is a version of conjugate gradients (see 'conjugate gradients') that does not use a linear search. Experiments indicate that the algorithm can sometimes give a significant improvement in speed on the conventional conjugate gradient algorithm (see [3] pp. 282-285).

- **Second order methods**

Second order methods are optimisation methods based on a local quadratic approximation of the error function (see 'local quadratic approximation' and 'non-linear optimisation techniques').

- **Self organising map** (or 'SOM' also called 'self organising feature map', 'Kohonen topographic mapping', 'Kohonen map' or 'Kohonen network')

See 'Kohonen topographic mapping'.

- **Sensitivity analysis**

Sensitivity analysis is a very general method to test the importance of the hypotheses made to develop a model. The basic idea is to see in what extent small changes on the hypotheses affects the results.

If, for an hypothesis, the results are significantly affected, then this means that the model is very sensitive to the hypothesis. This implies that the validity of this hypothesis is particularly important and should be checked.

- **Sequential learning**

We speak of sequential learning when applying a learning algorithm to determine the parameters of a model, the parameters are updated after the presentation of each training data point (see also 'batch learning' and 'gradient descent').

- **Sigmoidal activation function**

There are mainly two kinds of sigmoidal activation functions: the hyperbolic tangent activation function and the logistic activation function (see [3] pp. 126-132 and [14] pp. 27-28). One of the most important features of these functions is that they are continuously differentiable and hence back-propagation procedure can be used (see 'back-propagation'). Finally, note that they are termed 'sigmoidal functions' owing to their S-shapes (see also 'activation function').

Note: Kanter and Le Cun have shown that using hyperbolic tangent activation function speeds up the training process compared to the logistic activation function.

- **Sigmoidal unit**

We term sigmoidal unit, any unit (i.e. hidden or output unit) having a sigmoidal activation function (see 'activation function').

- **Simulated annealing**

Simulated annealing is used to reduce the problem of 'bad' local minima (see [14] pp. 75-76, p. 122, p. 130, p. 168 and [24] pp. 444-455).

Simulated annealing principle stems from an analogy with thermodynamics theory. The idea is to add a random component to the training process. This random component allows the training process to sometimes go uphill, and thus sometimes escape from local minima. The effect of this random component is progressively decreased to zero (by analogy with a temperature) as the training process progresses.

- **Singular value decomposition** (or 'SVD')

Singular value decomposition is a technique generally used with RBFs to determine the second layer of weights. Singular value decomposition is a method used for matrix inversion that overcomes some of the numerical problems related with near singular matrices (see [24] p. 33 and p. 59, see also section 5.5.2.).

- **Smoothing**

Smoothing techniques are aimed at reducing the irregularities of a function. Such irregularities can, for instance, stem from noise that corrupts the function (see [16] pp. 168-179 and [24] pp. 650-655). In this case, these irregularities should be suppressed since they are, by definition, unpredictable and spoil the quality of the data.

There is no 'universal smoothing technique' and the result of the application of a smoothing technique is largely conditioned by smoothing parameters. The value of these parameters has to be chosen in such a way that the function is sufficiently smoothed, but not too much (oversmoothing). Indeed, oversmoothing could suppress some relevant structure in the data (see [3] fig. 2.8 and fig. 2.9).

Note 1: Data density modelling requires the adjustment of smoothing parameters (see 'data density modelling').

Note 2: When a NN is trained by using a regularisation technique, the regularizer can be seen as a smoothing parameter.

- **Softmax function**

The softmax function is used when modelling conditional probabilities. It ensures that the outputs of the network behave like probabilities, that is, they are positive and sum to 1 (see [3] p. 215, pp. 238-240 and section 4.2.1, see also 'activation function').

- **Soft weight sharing**

Soft weight sharing can be seen as a regularisation technique (see 'regularisation') and is, therefore, aimed at reducing overfitting problems. Overfitting appears when the model used has too many free parameters comparatively to the complexity of the problem to be modelled. The basic idea of this technique is therefore to reduce the number of free parameters by encouraging some *groups* of weights to have similar values. This is achieved by using a specific regularizer (see [3] pp. 349-353).

- **Stacked generalisation**

Stacked generalisation is closely related to the one of committee of NNs (see 'committee of NNs'). For stacked generalisation, there is a first level of NNs (i.e. a committee of NNs) that are trained on the problem to be solved. Then, the outputs of the NNs of this first level are used as inputs to a kind of "gathering NN" that performs a kind of "non-linear average" of these inputs. In contrast to a standard committee of NNs the weighting coefficients to average the predictions of the committee members are not constant and depend on the inputs. The result of this average (i.e. the output of the gathering NN) is then used as the output of the NN unit (see 'NN unit').

Wolpert [31] suggests, first, that an important feature of the committee should be its diversity, and secondly, that the gathering NN should not be very complex and thereby provide a relatively smooth mapping (see [3] pp. 375-376 and [31], see also section 5.4.2.2). Various means exist to ensure the diversity of the committee members. One of them can be to gather inputs that are correlated in the same NNs. To detect (linear or non-linear) correlations between the inputs, a method can be to study the correlation of every pair of inputs by using statistical tests (e.g. contingency table analysis).

Finally note that to train such an architecture a technique close to 'cross-validation' is used (see 'cross-validation').

- **Standard deviation**

The standard deviation is the average deviation of the data points from the mean of the data set. It is defined as: $\sigma = \sqrt{\left(\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2\right)}$, where σ is the standard deviation, N is the size of the data set, $\bar{x}$ its mean, and x_i its i^{th} data point.

- **Static data** (also called 'cross-sectional data' or 'sampled data')

We speak of static data by opposition to time series data (see section 5.3.2.2). Each data point is sampled independently of the other data points and is considered to be uncorrelated with the rest of the data.

- **Steepest descent**

Steepest descent is another example of ambiguity of terminology used in the NN technology context. There is at least one common point between the various definitions: steepest descent algorithm is related to the gradient descent algorithm (see 'gradient descent'). However, sometimes 'steepest descent' is merely considered as a synonymous of 'gradient descent' (see [3] p. 263). At other times, it is considered as an improved version of the gradient descent algorithm. That is, instead of moving along the gradient direction on a distance conditioned by a more or less arbitrary constant (i.e. the learning rate), a line search (see 'line search') is performed to move directly to the minimum of the error function along the gradient direction (see [14] pp. 125-126 and [24] p. 421).

The best way to suppress this ambiguity is probably to distinguish the 'standard gradient descent' from the 'gradient descent using a line search procedure'.

- **Sum of squares error function**

It is the sum of the squared differences between the target values and the values predicted by the NN. The sum of squares error function is mostly used with regression problems. It may be used with classification problems, but more suitable error functions exist for this latter type of problems (see 'cross-entropy error function').

The sum of squares error function is formally defined as: $E = \sum_{i=1}^{N} \left(y\left(\underline{w}^*, \underline{x}_i\right) - t_i \right)^2$,

where N is the size of the data set, $\underline{w}^*$ the NN weight vector, $\underline{x}_i$ the i^{th} input data point, $y\left(\underline{w}^*, \underline{x}_i\right)$ the value predicted by the NN function at $\underline{x}_i$, and t_i the target value at $\underline{x}_i$.

Note: This error function is very sensitive to outliers and can be source of problems if used without any precaution with multi-valued functions (see section 5.3.2.2).

- **Supervised learning**

See section 4.1, [3] p. 10 and [14] p. 10, see also 'unsupervised learning'.

- **Testing error**

The testing error is the value of the error function obtained on the testing data set. This error is a measure of the global discrepancy between the predictions made by the NN for the inputs of the testing set and the targets of this testing set. The testing error is aimed at evaluating the generalisation capabilities of the NN.

- **Threshold activation function** (also called 'heaviside function' or 'step-function')

The threshold activation function is an activation function (see 'activation function') formally defined as: $g(a) = 0$, when $a < 0$, and 1 otherwise. As this function is not continuous, it is not differentiable and, therefore, cannot be used with back-propagation (see 'back-propagation').

- **Time series data**

Time series data consists of parameters measured over time. In this case, we expect that data later in the time series will be dependent on data measured earlier: i.e. there is an ordering to the data that must be respected (see also section 4.2.3 and 'static data').

- **Training error**

The training error is the value of the error function obtained on the training data set. This error is a measure of the global discrepancy between the predictions made by the NN for the inputs of the training set and the targets of this training set. The different

optimisation algorithms are aimed at minimising the training error (see 'non-linear optimisation techniques').

- **Training process**

The training process is the process that uses data to adjust the NN parameters.

With the type of NNs we consider in this project, the training process is iterative and consists of minimising an error function by cycling through the data of the training set (see training error').

A good training process should lead to good generalisation abilities for the NN having been trained. To achieve this objective, the training process must be able to avoid two main difficulties: overfitting, and being trapped in bad local minima (see also 'non-linear optimisation techniques' and section 4.4).

- **Training with noise on the inputs**

The basic idea is to add a little noise on each input presented to the NN. In this way, even if each input is presented several times to the NN during the training process, this input will be different each time. Thus, it is much harder for the NN to learn the data set, which may prevent it from overfitting.

Note: It can be shown that learning with noise is closely related to regularisation techniques and requires specific training algorithms (e.g. double back-propagation, see also 'regularisation' and [3] pp. 346-349).

- **Unbiased estimate**

An estimate of a parameter is said to be unbiased when the expectation of this estimate is equal to the value of the parameter (i.e. if an infinite number of experiments was performed then the average value of the estimates obtained for each of these experiments would be equal to the value of the parameter). For example the mean estimate stemming from maximum likelihood approach is unbiased (see [25] p. 286, see also 'biased estimate').

- **Unsupervised learning**

The goal of unsupervised learning is to adjust the parameters of a model to find a new representation of the data that captures its underlying structure (see [3] p. 10, pp. 318-319 and [14] p. 10, pp. 217-250). In contrast to supervised techniques (see 'supervised learning') no target is provided to guide the learning process[81].

The meaning of 'structure' is very problem dependent. For example a simplistic model to describe the data structure could be the pairs formed by the mean and the standard deviation of the data. A much more interesting example in the NN context, is the evaluation of the distribution of the data in the whole data space, that is, the data density.

- **Validation set** (sometimes also called 'acceptance set')

The validation set is independent of the training set and is aimed at measuring the generalisation ability of the model trained (see section 5.3.3).

Note: In [22] the author uses a different terminology: the 'validation set' is called the 'acceptance set', and the 'testing set' is called the 'validation set'. This terminology is probably more appropriate, since the role of the validation set is then to validate the final configuration. However the 'terminology validation/testing set' is now widely accepted, that is why we have kept it. Note: This constitutes another example of ambiguity in the terminology.

- **Visualisation techniques**

In the NN technology context, the dimensionality of the input space is generally high (i.e. greater than 3). If the data is not transformed, it is generally impossible to visualise the data without loosing important information concerning its structure. The goal of visualisation techniques is to transform the data to enable its representation in a low dimensional space (generally a two dimensional space). This transformation must

[81] For this reason it is difficult to evaluate the quality of the modelling.

be such that as little information as possible is lost (see also 'principal component analysis', 'Fisher's discriminant', 'GTM algorithm' and 'Kohonen topographic mapping').

- **W_{ij}**

W_{ij} represents a weight from the node 'i' to the node 'j'.

Note: sometimes in the literature, the opposite convention is used (e.g. in [3]).

- **Weight**

The weights are the adjustable parameters of a NN.

For a MLP, the different weights have all the same function, that is, being weights in the linear combination of the node activations (see 'activation of a node' and 'formal definition of the outputs of a MLP').

For a RBF, there are 2 categories of weights: the weights of the first layer and the weights of the second layer. The weights of the first layer are the co-ordinates of the RBF centres. The weights of the second layer, are weights in the linear combination of the activations of the hidden nodes (see 'activation of a node' and 'formal definition of the outputs of a MLP' and fig. 2).

- **Weight decay**

Weight decay is a regularisation technique (see 'regularisation'). Overfitting is generally synonymous with high curvature. To obtain high curvature large weight values are required. Consequently, the regularizer (i.e. the penalty term added to the error function) is the sum of the NN weights squared.

Nevertheless, this approach is often too simplistic. Thus, for instance, when both input and output variables have been normalised, the regularizer should be a little more sophisticated and, for instance, distinguish between the different weight layers (see [3] pp. 340-342).

- **Weight initialisation**

The weight initialisation can be defined as affecting to the weights an initial value in order to enable the optimisation procedure to start.

As there is no optimal optimisation procedure to train a NN and as these procedures are very sensitive to the initial conditions, the weight initialisation should not be neglected. Several techniques exist. In general these techniques include a stochastic component. In [3] (pp. 261-262), the author gives a method to initialise the weights when the data has been normalised in such a way that it has zero mean and unit variance (see 'data normalisation', see also 'non-linear optimisation techniques' and section 5.5.2).

Note: genetic algorithms (see 'genetic algorithms') can sometimes be considered as a weight initialisation procedure.

- **Weight vector**

A weight vector is a vector whose size is equal to the number of weight in the considered NN, and such that the value of each its component corresponds to the value of a weight in the NN.

- **Whitening** (also called 'data whitening')

See 'data whitening'.

- **Widrow's rule of thumb**

For classification problems, the Widrow's rule of thumb gives a lower bound for the size of the training set (N_{min}) required if a MLP is used: $N_{min}=\frac{W}{\varepsilon}$, where W is the total number of weights in the NN, and ε the maximum percentage of misclassification tolerated on the testing set (see section 5.3.3.2 and [2] p. 152, see also [3] p. 380 and [14] p. 156).

APPENDIX B: Case study (Detection of sleep disorders)

B.1. Objectives of the case study

The main objectives were:

- evaluating the efficiency of the draft guidelines for assessment, by testing them on a real NN application developed independently of them.
- using this experience to refine the guidelines.
- performing technology transfer from Aston University to LR,
- providing the (Oxford University) project team with an independent assessment of the quality of the application.

B.2. Project assessment

B.2.0. Adequacy level of the project for assessment

- The project was not a toy problem, but a live application involving real-world data. Accordingly, every difficulty related to NN application development may have occurred, which made the project interesting to be assessed.

However:

- The project has not been developed with a view to be assessed and certified.
- At the beginning, this project was considered as a research project: its feasibility and its targets were not completely known.

B.2.1. Project description

B.2.1.1 PROJECT PRESENTATION

The project has been developed by Oxford University and financed by Oxford Instruments. The goal of this project was to develop a sleep analysis tool to assist clinicians in the detection of sleep disorders. Generally such analysis is performed by

using hypnograms. Hypnograms give a representation of the evolution of the sleep through the time. Six main sleep states corresponding to progressively deeper sleep stages are usually distinguished: wakefulness, rapid-eye-movement (REM), stage 1, stage 2, stage 3, stage 4. Given an electroencephalogram (EEG) and other complementary information [21], rules are used to determine the sleep stage to be associated with the epoch considered and thereby hypnograms are built. These rule-based hypnograms have various drawbacks:

. their resolution in time and accuracy are insufficient and not adapted to the notion of sleep continuum (e.g. some disorders , such as micro-arousal lasting a few seconds cannot be detected) and consequently they give a coarse representation of the sleep wakefulness continuum.

. subjectivity when applying the rules. For instance: if two clinicians get to a consensus 80% of the time, then this is considered as very good.

Accordingly, the objective of this project is to produce a continuous (in time and sleep state) version of hypnogram, by (i) benefiting from the speed of a computerised version (manual classification requires a lot of time) to improve the time resolution (1 second epoch instead of 30 second epochs, 1 second epoch having been considered as sufficient to detect any possible disorder and hence the time scale can then be considered as continuous) (ii) benefiting from the interpolation properties of NNs to provide a continuous sleep state resolution.

Note: for further information see [21].

B.2.1.2 CHARACTERISTICS OF THE SELECTED ARCHITECTURE

- Pre-processing: to reduce the dimensionality of the problem, feature extraction has been performed by using a linear model: the AR model [21]. Thus for each second of EEG the coefficients of the corresponding AR model are computed and used as features. These coefficients are normalised before being presented to the NN unit. Note: In fact two types of AR models are used (tenth and third order), this is related to robustness issues (see next bullet point)
- NN unit: to improve the robustness[82] of the NN function, an elementary mixture of experts is used. This mixture is constituted by two experts (both MLPs); the first

[82] For instance, the EEG data can sometimes be corrupted by muscular artefacts.

expert has ten inputs (the coefficients of a tenth order AR model), six hidden units and is responsible for the classification of normal EEG data, the other one has three inputs (the coefficients of a third order AR model) and is responsible for the classification of corrupted EEG data (e.g. the corruption can stem from muscular artefacts). Both experts have three outputs corresponding to the probability of being in each of the relevant sleep stages (among the six main sleep stages, three have been shown as sufficient to provide a good explanation of the sleep wakefulness continuum [21]. These stages are: wakefulness, stage 4, and REM). The gating of the mixture is carried out by a third NN and can be seen as a binary switch between both experts (i.e. according to the input data the gating NN selects the outputs of one of the two experts).

- Post-processing unit: the NN unit outputs are transformed to produce as an output to the NN application the continuous hypnogram.
- The whole software related to the NN technology used in this application has been developed by the Oxford University project team. Note: This represents a total estimated at 700 lines of code.

B.2.2. Outcomes of the project assessment

B.2.2.1. MAIN QUALITIES OF THE PROJECT

- The necessity of using a NN component has been suitably justified.
- Globally the project has been developed in a sensible way, and no grave mistake has been discovered during the assessment.
- The developer seemed to have a suitable knowledge of the field relevant to the application (e.g. he was aware of the risks associated with a NN prediction failure) and sensible means had been used to have a better understanding of the data and check the validity of the prior knowledge (e.g. to understand the dynamics of EEG during sleep and compare it with prior knowledge, the data has been visualised by using Kohonen mappings [21]).
- Prior knowledge seems to have been correctly used and incorporated in the design to facilitate the development of the application and improve the performance. For instance: it is known that the age, sex or health state of a person does not influence the fundamental characteristics of the EEG relevant for the application. This

knowledge has been used to simplify the data collection: the data has been obtained from only young healthy and voluntary women.

- Great care has been taken with the pre-processing of the data in order to reduce the problems related to the curse of the dimensionality (e.g. AR model for feature extraction).
- The testing (unit and integration testing) seems to have been carefully carried out. Note: At least for the final version, the results are reproducible (e.g. the random initialisation of the weights has been recorded, consequently the NN unit can be retrained in the same conditions).
- Several models have been tried for the NN unit (MLPs, RBFs and linear models). Note: no RBF configuration appears in the final architecture, because as RBFs needed a larger number of hidden units they were significantly slower than MLPs and were too slow to fulfil the speed performance required.
  Note: For the pre-processing unit several different options have also been tried (AR models, Fourier transforms, band-pass filtering).
- Management of the project: There were monthly meetings to evaluate the project progress. These meetings involved: the project manager and the main developer of Oxford University, and some specialists from Oxford instruments.

B.2.2.2. MAIN DEFICIENCIES OF THE PROJECT

- A relatively systematic recording of the information has been carried out, but a significant lack of structure and standardisation of this recording has been noticed: e.g. (i) very few cross-references, (ii) the information is spread out in the documentation, (iii) no real version management has been used.
- A significant lack of documentation concerning the prior knowledge used has also been noticed (e.g. some prior knowledge used to develop the application is not stated, and the origin of the prior knowledge stated is not systematically mentioned).
- A specification for this application exists[83] but it is a functional specification in which no performance target for the accuracy and timing aspects, is numerically stated.

---

[83], "Questar functional specification, version 1.0", this document has been written by M J Laister.

- The absence of quantitative target can make one wonder whether the project was "under control". For example, during the assessment the main developer has asserted that the specification had not been used much.
- Even if at first sight the application (sleep disorder detection) did not seem to be safety critical, it should have been considered as such. Indeed, somebody suffering from sleep disorders often also suffers from drowsiness problems in the daytime, and this can be fraught with consequences (e.g. when driving).
- Different recorders are used to collect the data, but no test has been done to evaluate the measurement variations between the different types of recorders nor between the recorders of a same type. In terms of NN implications, this means that as for a same input the measurements made may differ significantly, there is a risk of inaccuracy when predicting in operation (see section 5.3.3.2).

  Note: Error bar techniques do not take into account the fact that the training data does not capture all the potential variation. Consequently, such a problem can remain undetected in operation while altering the predictions.
- The NN function does not include any real novelty detection system to prevent it from extrapolating.
- With classification problems it is preferable to have an equal proportion of data points from each of the class. This rule has been applied for the project. Nevertheless, as this distorts the true data distribution, this should be rectified. If the prior probability are known then to obtain the correct posterior probability, it is a simple matter of applying Bayes' theorem. This has not been done. An acceptable justification has been given, but it does not appear in the documentation[84].
- Two possible improvements of the architecture can be suggested:

  - It would be preferable to use a cross-entropy error function instead of a sum of squares error function. Indeed the former is more appropriate for classification problems (see [3] pp. 230-236).

  - It would be preferable to use hyperbolic tangent activation functions instead of logistic activation functions. It has been shown (Kanter and Le Cun) that using the

[84] The idea was to implicitly use a loss matrix.

hyperbolic tangent generally speeds up the training process compared to using the logistic function (see [3] p. 127).

B.3. Adequacy level of the draft guidelines

- The guidelines raised the key problems related to the development of the application assessed.
- Their level of detail seemed to be appropriate for this application.
- This case study has enabled the detection of minor deficiencies in the guidelines, and, therefore, enabled to refine them.

APPENDIX C: Applicability of the good practice rules

C.1. Testing the error bars amplitude

C.1.0. The problem

Some requirements concerning the amplitude of the error bars may be stated in the specification: e.g. the amplitude of the error bars must not exceed 1% of the predicted value for inputs corresponding to a temperature greater than 100°C, and 5% otherwise (see section 5.2.2).

If the data available is representative of the problem in terms of data space coverage, then it can be checked on this data that the amplitude requirements are fulfilled. However if the data set is small, or if a high safety integrity level is required, a more stringent test may be needed. In the next section we propose a procedure to perform such a test over the entire relevant data space.

C.1.1. Proposed procedure

Note that this procedure is not aimed at checking the validity of the error bars, but only checking that their amplitude complies with the specification requirements. Accordingly, we assume that the validity of the error bars has been previously checked, and, therefore, the target values are not required for the test (see section 5.6.3.2). The basic idea of this procedure is that the relevant area of the data space is the one where the data lies. The procedure requires 3 steps:

1. Let S_R be a set of input points representative of the problem in terms of data space coverage,
2. A large number of points is randomly generated in the vicinity of every input point of S_R.
3. For each point thereby generated, it is checked that the error bars produced by the NN function comply with the specification requirements.

To randomly generate a point in the vicinity of a data point P_{SR} belonging to S_R (see step 2 above), we have defined the following algorithm:

2.1. Identify the N_p nearest points (in terms of Euclidean distance) to P_{SR}. Let us call S_N the set constituted by P_{SR} and its N_p nearest neighbours[1].

2.2. For each input variable, the average spacing between the points of S_N is computed.

2.3. A point can then be randomly generated from P_{SR} by adding to each input co-ordinate of P_{SR}, a random value drawn from a zero-mean Gaussian distribution having a standard deviation equals to the half of the average spacing for the corresponding input variable (see step 2.2).

C.1.2. Discussion

- As the target values are not required, the amplitude of the error bars can be tested for an arbitrarily large number of data points in the input data space.
- This procedure requires the data set to be representative of the problem in terms of data space coverage, but it does not need to be representative in terms of data density.
- An alternative solution could be to use an estimate of the data density to generate the points at which error bars amplitude can be tested. Nevertheless various problems arise. For instance, this solution implicitly assumes that the data density estimate is dependable, which is a harder condition to ensure than that the data set (S_R) correctly covers the input data space. Moreover, the probability of generating a point lying in an area of low data density is (by definition) low. As, these areas are typically the one for which the amplitude of the error bars is the highest, such a test is likely to be very optimistic. This can be problematic when low density areas are in the vicinity of failure mode areas.
- The choice of a Gaussian distribution may be arguable (see step 2.3). However, the main problem is to have a means of generating data points in the vicinity of another data point, in such way that the region of the data space defined by P_{SR} and its *Np*

[1] *Np* is an arbitrary and small and integer constant.

nearest neighbours is correctly covered by the test, and that points belonging to areas outside the relevant data space are unlikely to be sampled.

C.2. Overall failure rate determination for regression problems

C.2.0. The problem

In a safety critical context, one of the most important measures is the overall failure rate of a system. This applies with equal force to conventional software applications and, therefore, should also apply to NN applications. Currently no technique seems to be available to determine the overall failure rate of a NN function for regression problems.

This failure rate is, roughly speaking, the probability that the EUC enters into a failure mode without the NN function having enabled the detection of this failure mode. In other words the error bars of the NN function lied entirely into the normal mode subspace whereas the true value of the function lies in the failure mode subspace (see fig. 12.c and fig. 12.d). In this section we propose a method to compute such a failure rate.

To develop such a procedure the error bars produced by the NN function (see fig. 12.a, fig. 12.b and fig. 12.c) are essential.

C.2.1. Example and principles

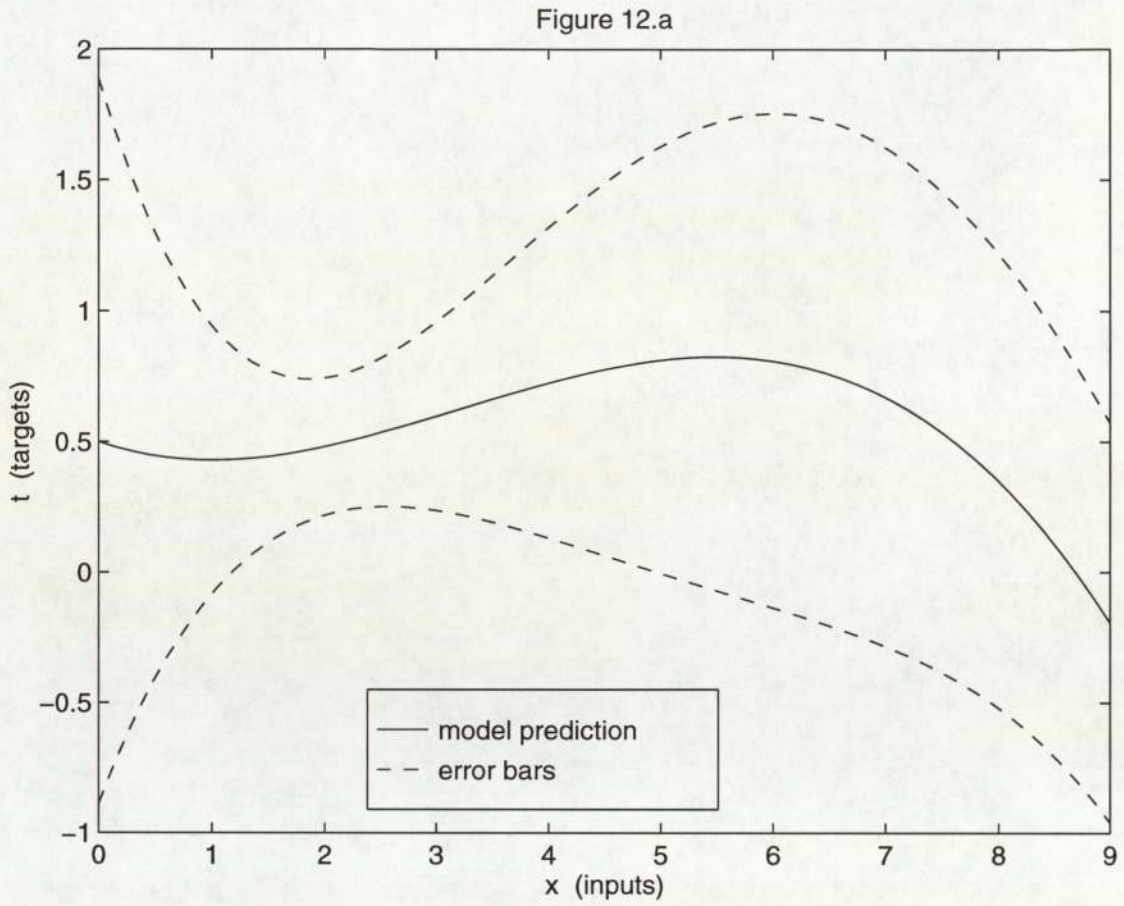

Figure 12.a: This Figure illustrates the notion of error bars and shows how the amplitude of these error bars can vary in the input data space. On this figure the error bars are defined as ± 2 standard deviations around the NN prediction. Hence, if the error bars are valid then the true values of the function lie within the error bars approximately 95% of the time.

Figure 12.b: conditional distribution of the target values

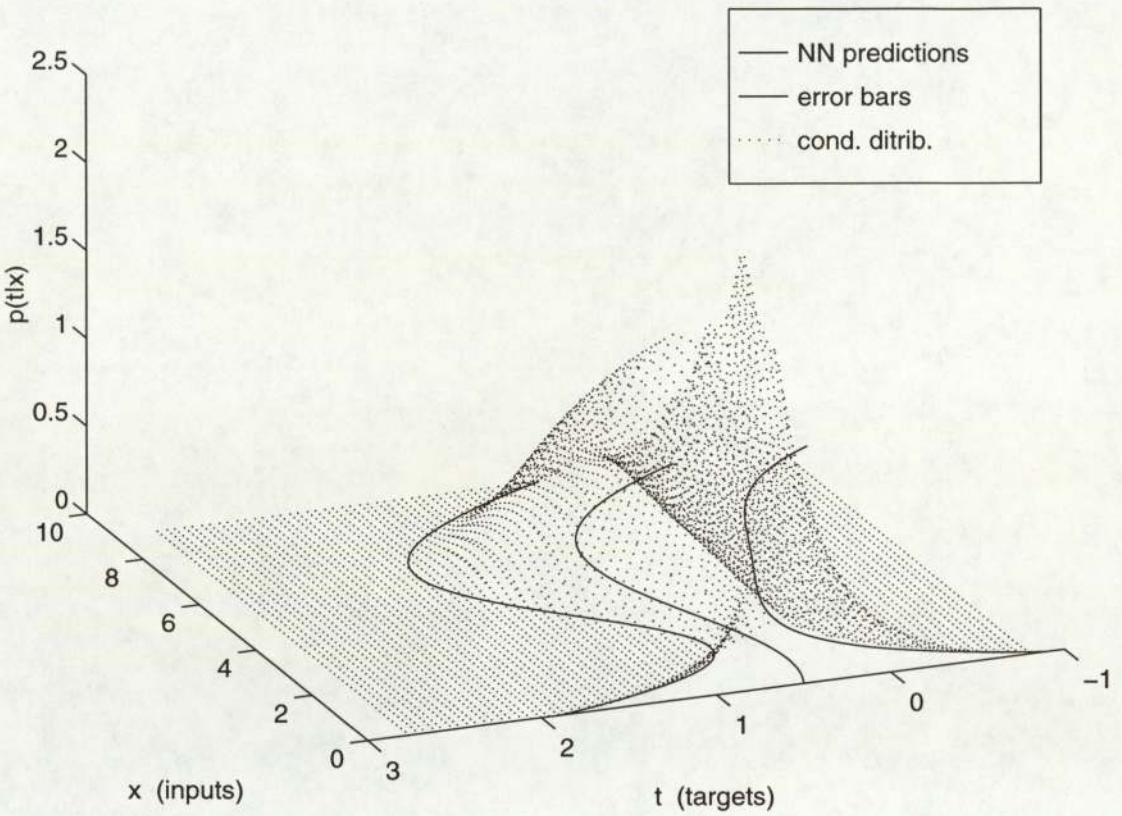

Figure 12.b: This figure gives an equivalent three dimensional representation of figure 12.a, and explicitly represents the conditional distribution of the targets. In this case, for each x, the target distribution (i.e. p(t|x)) is Gaussian (see fig. 12.c).

Note: The perspective effects are slightly misleading: The middle line (i.e. the NN predictions) does go along the peak.

Figure 12.c: conditional distribution of the target values

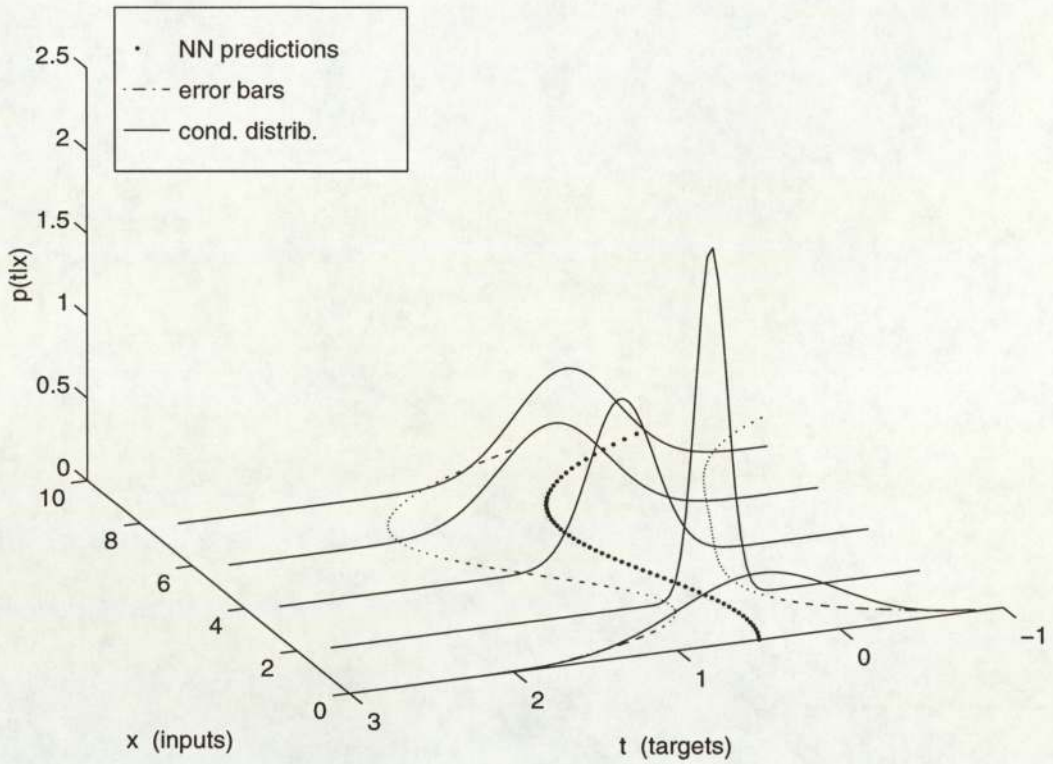

Figure 12.c: This figure is the same as fig. 12.b except that, to facilitate the visualisation, the conditional distribution of the targets has been draw for fewer (5) input points (see fig 12.b).

Figure 12.d: conditional distribution of the target values

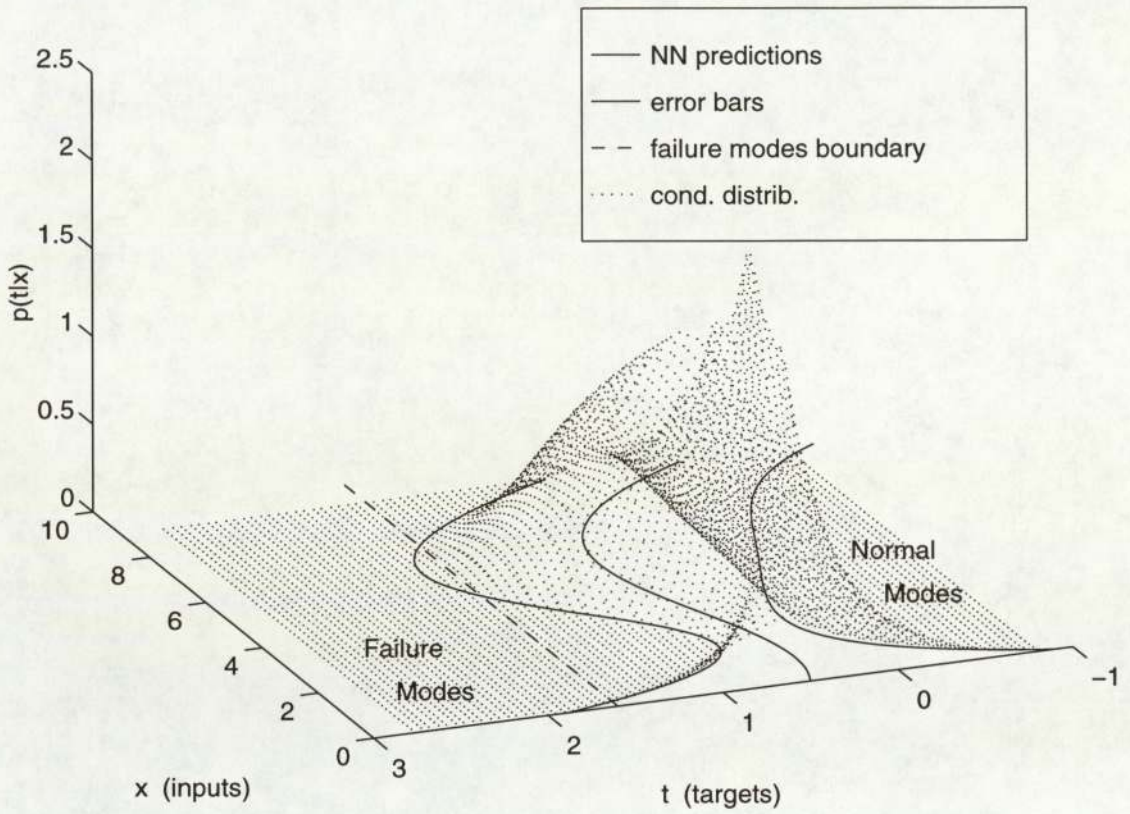

Figure 12.d: This figure shows the same conditional distribution, but the joint space (i.e. the two dimensional space spanned by the input and the target variables) has been partitioned into two subspaces: one corresponding to the normal modes of operation of the EUC, and one to the failure modes. For this example, a point belongs to the failure mode subspace if the value of its target is greater than 1.6 (an example of such a constant threshold could be, the maximum number of rpm that an engine can endure before breaking down, since it is independent of the gear engaged).

Figure 12.e: failure rate determination (undetected failure modes)

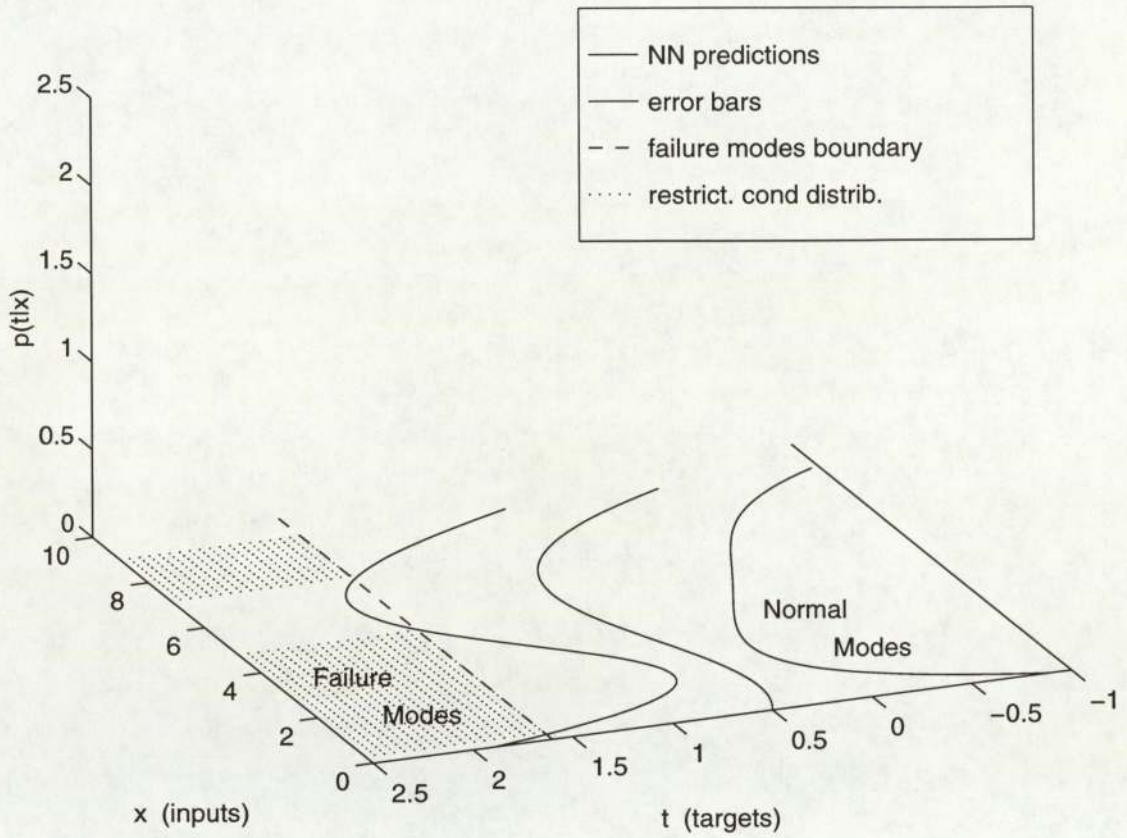

Figure 12.e: This figure considers the same conditional distribution but restricted to the points corresponding to possible undetected failure modes (the dashed area). Note that a failure mode is undetected if the error bars lie entirely within the normal mode subspace. Finally, note that this restricted distribution is referenced as $p(t_f|x)$ thereafter.

Figure 12.f: failure rate determination (zoom)

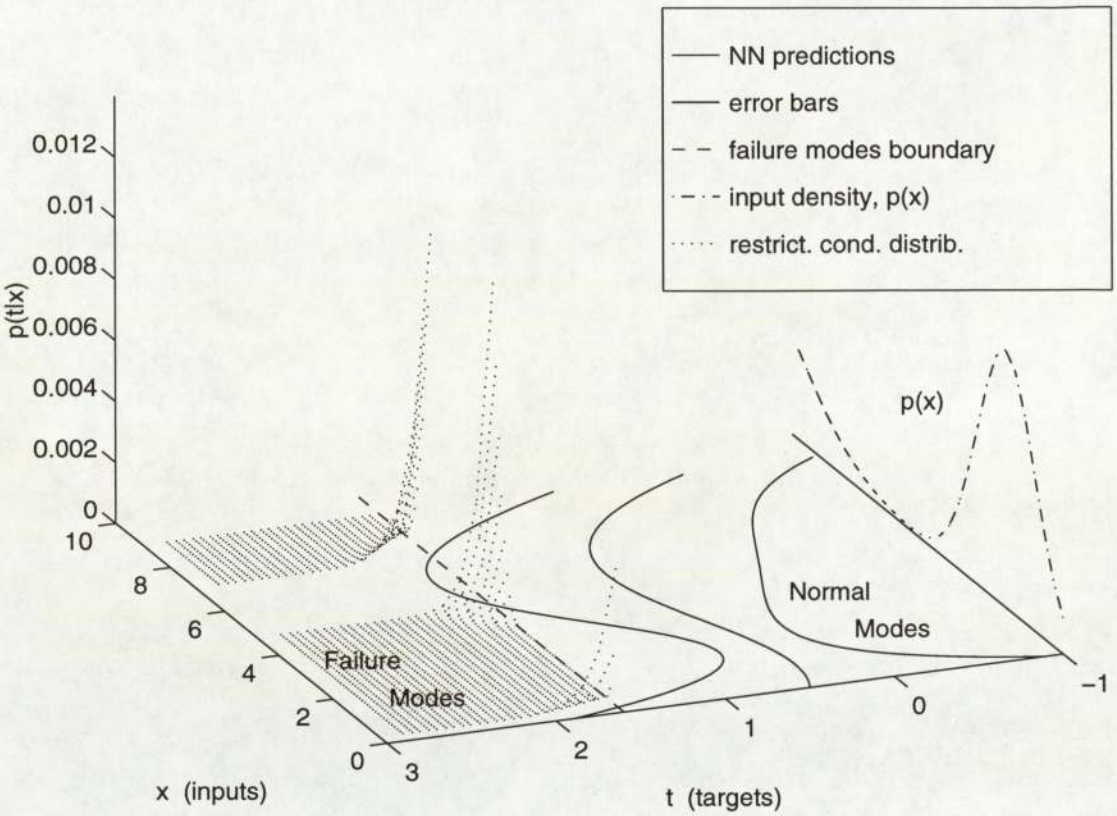

Figure 12.f: This figure is the same as figure 12.e, except that the input data density is represented on this figure and that the scale for p(t|x) has been changed to enable a better visualisation.

The restricted conditional distribution of the targets and the input data density are the two key elements to determine the probability of failure undetected by the NN (see figure 12.f). This failure rate can be obtained by integrating, over all the failure mode subspace, the conditional probabilities of the undetected failure points, weighted by the input data density. Formally it is defined as:

$$\iint p(t_f|x)\, p(x)\ d\, t_f\ .\ d\, x,$$

where t_f is a variable that spans all the possible target values t such that: (x,t) corresponds to an undetected failure mode ((x,t) being a point of the joint space).

In order to understand the idea behind the mathematics, the following can be considered:

- A failure mode is not detected if the error bars lie entirely inside the normal mode subspace. On the other hand, if the error bars lie (at least) partially in the failure mode subspace, the (possible) failure mode is detected (see fig. 12.d and fig. 12.e).
- To compute the probability of undetected failure for an input value (x_i), we have to consider all the undetected failure mode points for this input, and sum their probabilities of occurrence (namely $p(t|x_i)$).
- This has to be done for each possible input value. Then, the global probability of undetected failure is computed by performing a weighted sum of the 'local probabilities of undetected failure' (i.e. the probability of undetected failure for a particular input data point). This sum is weighted by the probability of occurrence of the different input data points (namely, p(x)) (see also fig. 12.f).

Note: The terminology used in the above explanation was deliberately not very rigorous to facilitate the understanding (e.g. 'probability' has been used instead of 'density').

C.2.2. Proposed procedure

The next stage is to pass from the mathematical expression of the failure rate to a practical procedure.

To create such a procedure, the main problem is to adapt the formula defined on a continuous input space to an input space represented by a finite number of data points. Several solutions exist. A natural one is to use a set of input points representative of the input space (e.g. the testing set) and then, for each point of this data set, compute the probability of undetected failure (i.e. $\int p(t_f|x_i)\ \mathrm{d}\,t_f$)[2]. As the data set is representative, it is also representative in terms of density. Hence, a simple average of the probability of undetected failure obtained for each input point gives the overall probability of undetected failure. This is formally defined by:

$P_{\text{undetected failure}} = \frac{1}{N}\sum_{i=1}^{N} \int p(t_f|x_i)\ \mathrm{d}\,t_f$, where N is the size of the data set, x_i is the i[th] input of this data set, and t_f a variable that spans all the possible target values t such that the point (x_i, t) corresponds to an undetected failure mode.

---

[2] This integral does not pose major problems, and can be performed by using classical numerical technique or mathematical tables for the Gaussian distribution (see [25] p. 448).

C.2.3. Discussion

- We have only considered one-dimensional target spaces, which should expect to be the most common case for regression problem (see section 5.4.2.2). For a higher dimensionality the integration $\int \ldots \int p(t_f^1,\ldots,t_f^n|x_i)\, dt_f^1 \ldots dt_f^n$ ($\underline{t_f}$ being a vector of n target values) may be more problematic.
- Here again (see section C.1), the target values of the data points are not needed. An important consequence is that the application of this procedure is not constrained by the quantity of data points available, which makes it very flexible. For example, we can constitute the data set by sampling an arbitrarily large number of input points from the input data density.
- The adaptation of the procedure to compute, for instance, the probability of undetected normal mode is straight forward. Even if this is of lesser interest, there may be possible applications, since a 'dummy faulty state detection' may have an important associated cost.
- The procedure can readily be extended to take into account more sophisticated boundaries between the normal mode and the failure mode subspaces (e.g. the failure mode boundary for the targets might be defined as a linear function of the input value).

C.3. Multi-valued function detection

C.3.0. The problem

Trying to fit a NN model to a multi-valued function generally leads to very poor results (see section 5.3.2.2). Sometimes prior knowledge of the problem to be solved is enough to detect such multi-valued functions before the beginning of the design phase, yet this is not always the case. In this section, an idea of procedure to detect multi-valued functions is provided.

Up to two dimensional input space, plotting the function to be modelled generally enables the detection of multi-valued functions (see fig. 13.a). If the dimensionality is greater than 2, then using such visual techniques is no longer possible. An alternative solution is to observe the residuals associated with a trained NN; if they are multi modal then the function is likely to be multi-valued.

This means that if the dimensionality of the input space is greater than 2 (i.e. most of the time), then the multi-valued aspect of the function cannot be detected before the training has been completed. In such a case, the time spent to design the architecture (e.g. design of the pre-processing unit) and the time used to train the network is lost. A solution has to be found. Basically, the idea we propose reduces the dimensionality of the space to enable a visual detection of multi-valued functions.

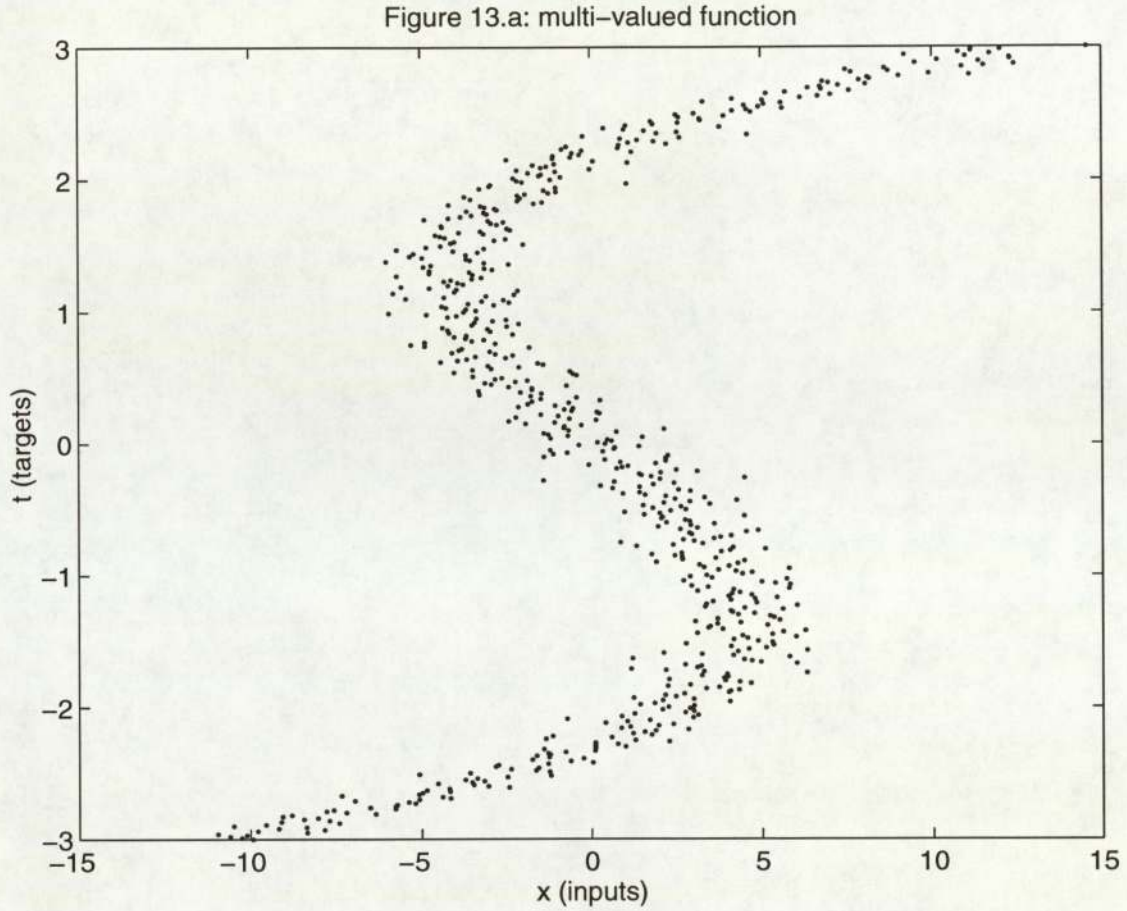

Figure 13.a. Illustration of a multi-valued function.

C.3.1. Terminology

- The only distance we consider thereafter is the Euclidean distance.
- Let S be a set of input data points. Let P be an element of S. Let P_n be the nearest neighbour of P in S. Then, we define as the Δ *value for* P the distance between the target of P and the target of P_n.

C.3.2. Example and principles

- If the data set is representative of the function to be modelled, and if this function is single-valued and relatively smooth, then we can expect the Δ values to be mainly due to the noise (on the targets). Accordingly, they are likely to be small and their distribution likely to be strongly dependent on the noise distribution.

  Indeed, if the distribution of the noise is known then it should be possible to evaluate the expected distribution for the Δ values: e.g. if the noise if Gaussian, and if we consider a one dimensional target space, then we can expect that the distribution of the Δ values will be half the shape of a Gaussian distribution for positive input values (see fig. 13.b, fig. 13.c and fig. 13.d).

Figure 13.b: single–valued function

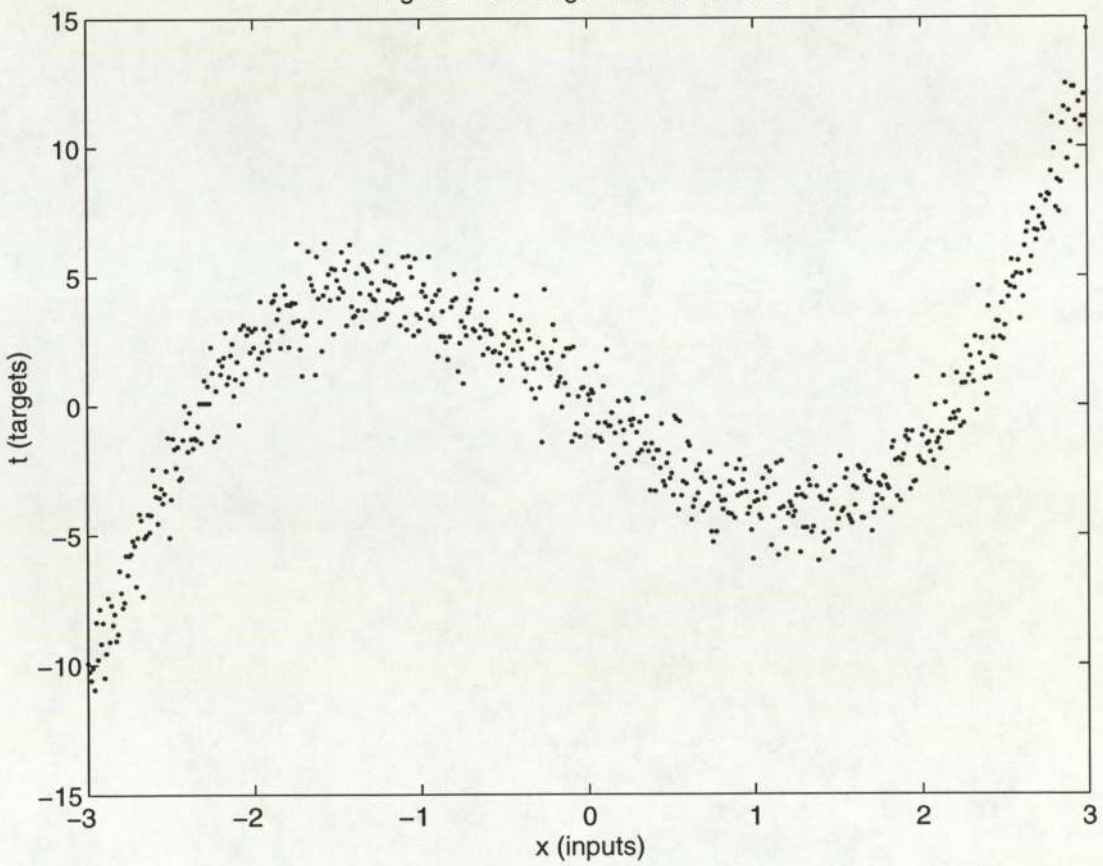

Figure 13.b: Illustration of a single-valued function.

Note: The slightly multi-valued nature of the function in this figure is owing to the noise, and therefore this function can be considered as single-valued.

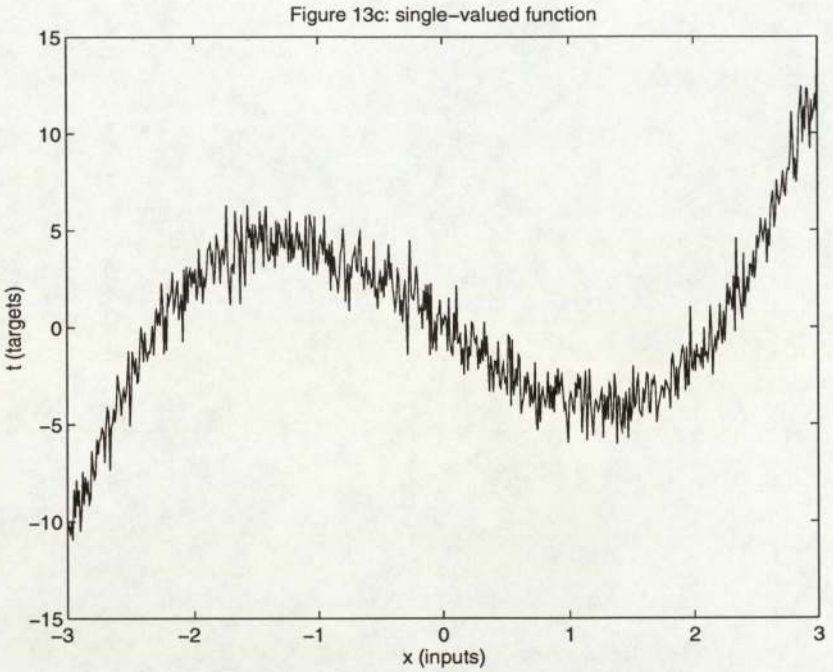

Figure 13.c: This figure corresponds to the function represented in figure 13.b. In this figure, the length of the lines joining the points corresponds approximately to the Δ values. Here, as the function is single-valued, the Δ values are relatively small in the whole input space.

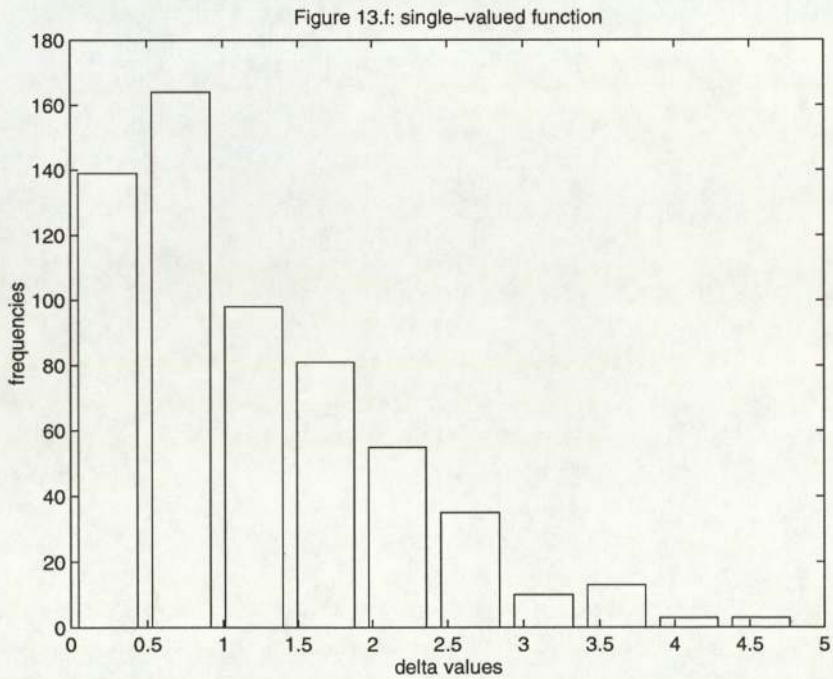

Figure 13.d: This histogram represents the distribution of the delta values for the single-valued function whose Δ values are represented in fig. 13.c. As the noise on the target is Gaussian, the shape of this histogram should be half the shape of a Gaussian distribution for positive input values, and the magnitude of the Δ values should be comparable to the standard deviation of the noise. As this is the case, this gives a good confidence in the single-valued nature of the function.

- If the function is multi-valued, then even if it is relatively smooth on each branch and the data set is representative, we can expect that a large number of points will have a high Δ value (since their nearest neighbours in the input space may have their targets on the another branch of the function).

  Thus, typically, we can expect (at least) two main ranges of Δ values: The ones related to the noise in the data (low Δ values) and the one related to the multi-valued aspect of the function (high Δ values) (see fig. 13.e). More generally we can expect the distribution of the Δ values to be different from the expected one (see fig 13.f).

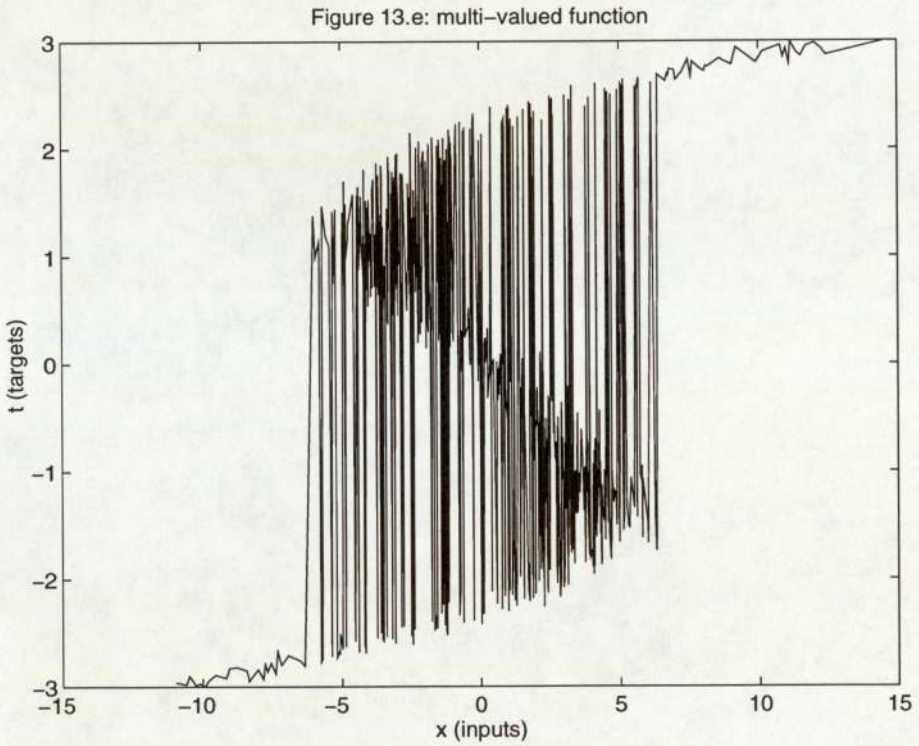

Figure 13.e: This figure corresponds to the multi-valued function represented in figure 13.a. As in figure 13.c, the length of the lines joining the points corresponds approximately to the Δ values. For the inputs where the function is single-valued the Δ values are small, whereas for the inputs where the function is multi-valued the Δ values can be large.

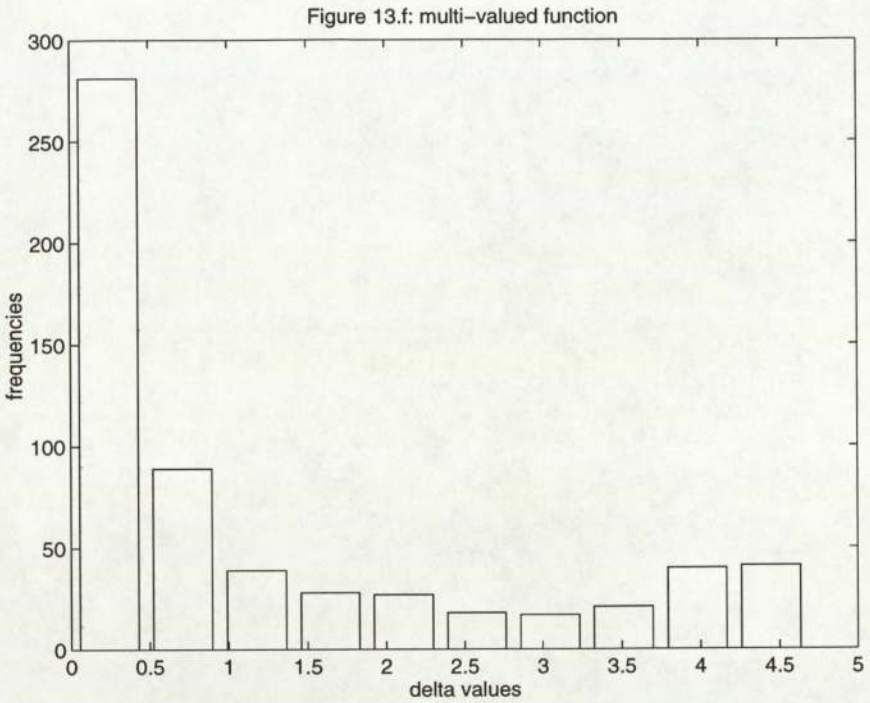

Figure 13.f: This histogram represents the distribution of the delta values for the multi-valued function represented in fig. 13.a. As the noise on the target is Gaussian, the shape of this histogram should be half the shape of a Gaussian distribution for positive input values. As this is not the case, the function is very likely to be multi-valued.

C.3.3. Proposed procedure

The main idea consists of computing the Δ values, and comparing their distribution with the expected one (e.g. by using histograms or statistical tests[3]).

If both distributions differ from each other, or if the Δ values are abnormally large compared to the standard deviation of the noise, then the function is likely to be multi-valued.

If the standard deviation of the noise is not known, then only using the shape of the distribution can be misleading. Indeed, there are probably multi-valued functions such that the shape of the distribution of the Δ values is near to the expected one. Such functions would fool the test. This problem is mainly due to the fact that the histograms give an overall distribution of the Δ values, but do not consider the local distribution of the Δ values (i.e. the distribution of the Δ values in the vicinity of a particular input point). For example, for the figure 13.g we could expect to obtain

[3] See [25] pp. 331-341.

locally 2 clusters of Δ values; one related to the noise, and one related to the multi-valued aspect of the function. On fig. 13.h we can see that if we consider a global distribution of the Δ values this information is lost (i.e. the distribution is not bimodal). Furthermore, a function may be multi-valued in only a small area of the input space, which may not distort significantly the shape of the Δ values distribution.

This loss of information can be reduced by plotting the distribution of the Δ values as a function of the inputs.

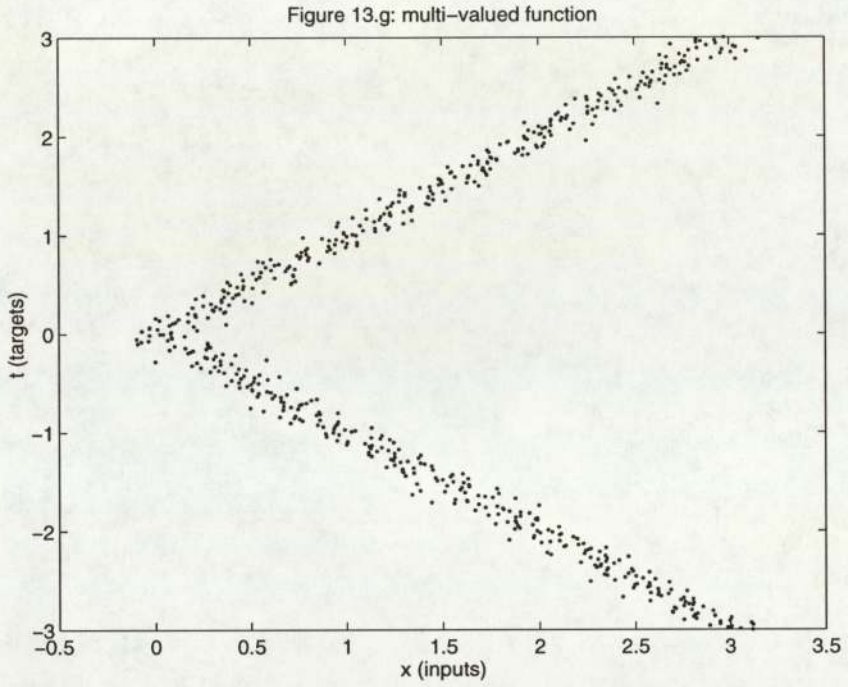

Figure 13.g: Another example of multi-valued function. In this figure, most of the time, for a given area of the input space, there are 2 ranges of Δ values. Hence, we may expect the distribution of the Δ values to be bimodal. However, this is not the case (see fig. 13.h).

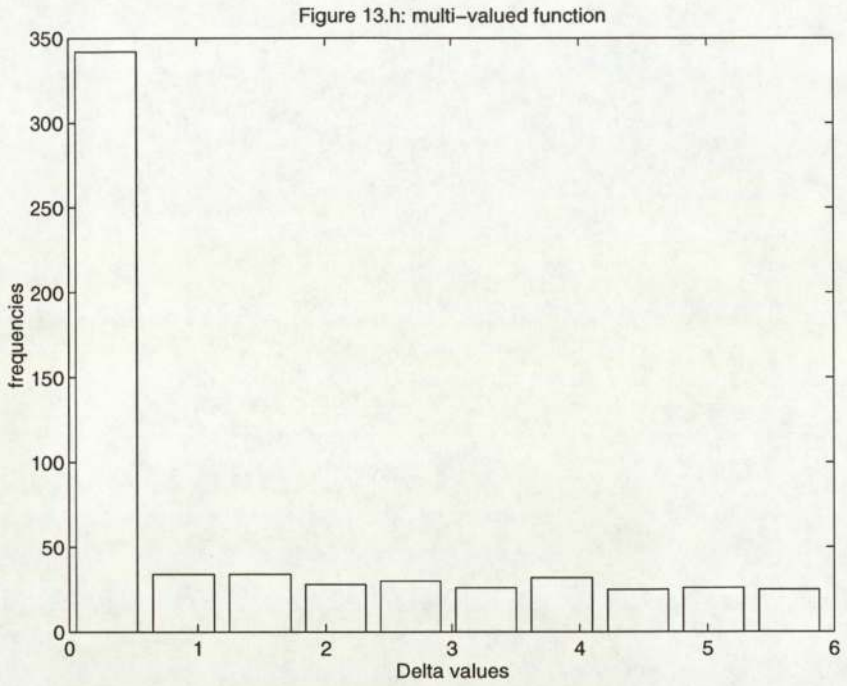

Figure 13.h: Although the Δ values are locally bimodal, this does not appear on the histogram. This loss of information is owing to the fact that this histogram is not input dependent and gives a global distribution of the delta values. Typically, the solution that uses principal component analysis and that is discussed later on, would enable to reduce this problem dramatically.

To enable an easy visualisation of such a distribution, a means of reducing the dimensionality of the inputs has to be found. The idea we propose is to perform a principal component analysis (PCA), and project the input points onto the first principal component. In this way the distribution of the Δ values is a function of the inputs, this distribution can easily be visualised since involving only 2 input dimensions (the Δ values and the co-ordinates of the inputs on the first principal component).

C.3.4. Discussion

- Although the use of PCA may improve significantly the procedure, it can still be fooled, since information is lost when projecting on the first component[4].
- Very few tests have been performed on this procedure. Moreover, the tests performed concerned toy problems involving low dimensional synthetic data. So we can wonder whether the procedure is robust enough to be efficient with real-world data.

[4] The percentage of variance explained by the first principal component can be used to evaluate the quality of the projection.

- We have not tested the version of the procedure using PCA.
- Even if this procedure may be unable to detect some multi-valued functions, it can have other applications. For instance, points with an abnormally large Δ value should be considered with great care: a large Δ value may be synonymous with outlier presence (see section 5.3.2.2) or holes in the input data distribution (see section 5.3.2.2).
- This procedure should not be computationally very demanding
- This procedure is not supposed to be the ultimate one but only to give an idea of the feasibility of the problem.

C.4. Data density modelling and representative data sets

C.4.0. The problem

In chapter 5 the need of data density modelling is repeatedly emphasised. This is a crucial issue, especially if high a safety integrity level is required, since the compliance with the specification is based on the representative aspect of the testing set. The main difficulty with this kind of techniques is their unsupervised aspect.

C.4.1. Procedure

To model a data density, the life cycle defined in section 3.2 is still applicable and there are various common points with the development of a NN function.

As for a NN function, several models of different complexity should be tried. However the goal of the training process is not to minimise an error function[5] but to maximise the likelihood of the model parameters with respect to the data set.

As for NN function, problems of overfitting and local minimum exist, and the main quality criteria for a model should be its generalisation ability, that is, the likelihood of the model with respect to a testing set should be relatively high.

Some of the techniques used are also quite similar to those used with NN models. For instance, to reduce the risk of overfitting and select the complexity of the model cross-validation can be used.

[5] i.e. a global measure of the discrepancies between the NN predictions and the target values.

C.4.2. Discussion

- Provided great care has been taken concerning overfitting and local the problems of minimum, a suitable modelling of the data density can be performed (see [3] pp. 33-73).
- Currently data density modelling techniques do not fall into the scope of the guidelines developed. Adapting these guidelines to enable the assessment of data density modelling techniques should not be very difficult, but could raise extra problems that time constraints have prevented us from addressing: e.g. we can wonder whether confidence measures (e.g. error bars) could be associated with a data density modelling.
- Provided the true data density is known and the relevant data space is correctly covered by the data, building a representative data set of the system, or, simulating a representative data set[6], should not pose major problems. The main problem is to get the first representative data set from which the true data density can be modelled:

  - the problem of coverage of the relevant data space can partially be solved by using techniques such as 'failure mode analysis'. Nevertheless, it is in general impossible to ensure that data for *every* failure mode is present in the data set.

  - the most difficult problem is probably to ensure that the data set is representative in terms of density, that is, the natural distribution of the data points is respected by the data set.

  Anyway, to model perfectly the data density an infinite number of data points would be needed. In consequence, the best we could do is probably to develop a methodology for the data collection that maximises the representative aspect of the data set collected.

[6] As said in section 5.3.3.2, the way the generalisation error is computed can be changed by weighting each testing point by the ratio between the data density (at the considered point) for the system in operation and the data density (at the considered point) for the testing set.

C.5. Error bars and modularity

C.5.1 The problem

- As said in section 5.4.2.3, modularity may pose problems to the computation error bars (see section 5.4.2.3). For example, we can wonder how to compute error bars on the predictions of the NN function when stacked generalisation is used.

C.5.2. The procedure

Among the different error bars techniques, predictive error bars are probably currently the only ones that enables to overcome this problem.

Predictive error bars are determined by using an extra NN to model the variance of the standard deviation of the NN unit residuals. As to carry out such a modelling, only the target values and the overall NN unit predictions are required, error bars for the whole NN unit can be given without having to give error bars for the outputs for every NN of the NN unit.

C.5.3. Discussion

- This solution relies on the quality of the modelling produced by the extra NN, which introduced a possible extra source of error.
- No extra specific theoretical result is needed.

C.6. Error bars and classification problems

C.6.1. The problem

As with regression problems the accuracy of the prediction decreases with the training data density, therefore, one should be able to provide a measure of the confidence over the NN predictions. It has been shown that when the training data density is low, a NN tends to gives overconfident predictions (see [3] pp. 405-407), whereas one should expect the NN to give probabilities near to uncertainty (e.g. near to 0.5 for a two-class problem). Intuitively this means that instead of answering "this point belongs to class *B*" a 'reasonable NN' should answer "I think this point belongs to class *B*, but I have not enough information to be sure of it".

No practical technique has been found in the literature, during this project, to provide error bars for classification problems.

C.6.2. Ideas of solutions

- As the accuracy of the NN predictions is positively correlated with the training data density, a simple solution could consist in providing both: the NN prediction and the training data density (or any appropriate novelty measure) in the area of the input data point.

  While this correlation does exist, this solution does not give any means to convert the data density into an interpretable accuracy measure. Accordingly, if such a solution were to be retained, the conversion should probably be determined empirically, which may be incompatible with high safety integrity levels.
- More principled solutions try to reduce the over-confidence in low data density area. Mainly there are 2 ways of addressing this problem: using Bayesian techniques or down-weighting all predictions uniformly by an empirically determined factor (see [11] and [19]). Note that even if these techniques provide more dependable predictions for low data density, they do not give any confidence interval concerning these predictions.
- A natural, but computationally demanding solution, can be to use sampling techniques in the Bayesian framework to evaluate the distribution of the posterior probability. That is, sampling sets of weights according to the weight posterior distribution and predicting the posterior probability using each different set of weights. Thus, the posterior probability distribution can then be approximated.

C.6.3. Discussion

Let us recall that global failure measures can be provided (see section 4.6), and therefore, this problem of confidence over the prediction may be a complementary aspect that may be relevant only for high safety integrity levels.