Preprocessing Perceptrons

Lena Kallin Westin
kallin@cs.umu.se
©Lena Kallin Westin, 2004, except

Paper II, ©Austrian Society for Cybernetic Studies, 1996

Print & Media, Umeå University, 2004:403009

UMINF-04.10 ISSN-0348-0542 ISBN 91-7305-645-6
To Frida Sjöberg
- you encouraged and inspired me!
Abstract

Reliable results are crucial when working with medical decision support systems. A decision support system should be reliable but also be interpretable, i.e. able to show how it has inferred its conclusions. In this thesis, the preprocessing perceptron is presented as a simple but effective and efficient analysis method to consider when creating medical decision support systems. The preprocessing perceptron has the simplicity of a perceptron combined with a performance comparable to the multi-layer perceptron.

The research in this thesis has been conducted within the fields of medical informatics and intelligent computing. The original idea of the production line as a tool for a domain expert to extract information, build decision support systems and integrate them in the existing system is described. In the introductory part of the thesis, an introduction to feed-forward neural networks and fuzzy logic is given as a background to work with the preprocessing perceptron. Input to a decision support system is crucial and it is described how to gather a data set, decide how many and what kind of inputs to use. Outliers, errors and missing data are covered as well as normalising of the input. Training is done in a backpropagation-like manner where the division of the data set into a training and a test set can be done in several different ways just as the training itself can have variations. Three major groups of methods to estimate the discriminance effect of the preprocessing perceptron are described and a discussion of the trade-off between complexity and approximation strength are included.

Five papers are presented in this thesis. Case studies are shown where the preprocessing perceptron is compared to multi-layer perceptrons, statistical approaches and other mathematical models. The model is extended to a generalised preprocessing perceptron and the performance of this new model is compared to the traditional feed-forward neural networks. Results concerning the preprocessing layer and its connection to multivariate decision limits are included. The well-known ROC curve is described and introduced fully into the field of computer science as well as the improved curve, the QROC curve. Finally a tutorial to the program trainGPP is presented. It describes how to work with the preprocessing perceptron from the moment when a data file is provided to the moment when a new decision support system is built.


I avhandlingen visas genom exempel hur den preprocessande perceptronen fungerar jämfört med andra analysmetoder. Det beskrivs också hur man kan utöka idén och göra en ny analysmetod där man förutom neurala nätverk också använder sig av fuzzy logik för att bygga ett beslutsstödssystem. Det finns beskrivningar, tankar, idéer och teorier kring hela vägen från insamlandet av data fram till tolkningen och utvärderingen av det färdiga systemet.
Preface

This thesis consists of the five papers below and an introductory part. In the introductory part, a background to and a description of the preprocessing perceptron is given together with a summary of the papers.


In addition to the papers included in the thesis, other papers have been produced in relation to the Ph.D. studies [Kallin Westin, 2004a, Kallin Westin, 2001a, Eklund and Kallin Westin, 2001, Eklund et al., 1999, Kallin and Eklund, 1998, Kallin, 1998, Bohlin et al., 1998]. During the Ph.D. studies the author has had the opportunity to orally present her thesis work at four international conferences and by a poster at one conference.
Acknowledgement

Patrik Eklund has been my supervisor and was the one that introduced the research area to me. He is my co-author on several papers and has never stopped encouraging me during my work. Almost every time we have met he has been full of new ideas and questions which have inspired me in my work.

There are so many people that have meant a lot to me during the work with this thesis. It is impossible to mention all of you by name but I give my sincere thanks to all colleagues at the department! The (sometimes very long) "fika" breaks have been the breather-space needed to be able to continue work with the same enthusiasm and energy over the years.

I still want to mention a few names. When I started to work at the department Niclas Börlin and I shared office and he has been a good discussion partner since then. Lena Palmquist was one of the few female computer scientists at that time and she has given me support and valuable advice which helped me adjust to the atmosphere at the department.

Fredrik Georgsson has been a keen reader of my papers and a true discussion partner concerning the contents of some of the papers. Sometimes he has been like a co-supervisor and pointed out alternative research directions.

During the last months I have been working full-time finishing the thesis. Marie Nordström has rearranged her working schedule just to help me get more time for my thesis work. She has also taught me a lot about writing and thinking scientifically. We have been writing papers within computer science didactics together which has given me insights and knowledge that I have used in this thesis.

Fredrik Georgsson, Thomas Hellström, and Gunilla Wikström have all read parts of my earlier drafts of this thesis and I am very grateful for their comments. Their discussions over the subject have been inspiring and have improved the thesis. I also acknowledge the comments on my papers from the anonymous referees.

Finally I thank Mats for being my partner for life and helping me see what is important in life. He has also read parts of my thesis and supported me in many ways during my work. Thanks!

Financial support has been provided by the Department of Computing Science and the Faculty of Science and Technology, Umeå University.

Umeå, April, 2004
Lena Kallin Westin
### Contents

1 Introduction ........................................... 1
   1.1 Medical informatics .................................. 1
   1.2 Intelligent computing ................................. 2
       1.2.1 Decision support systems ......................... 2
   1.3 The production line .................................. 3
       1.3.1 The extraction phase .............................. 4
       1.3.2 The analysis phase ............................... 6
       1.3.3 The synthesis phase .............................. 9

2 Neural networks and fuzzy logic ......................... 11
   2.1 The perceptron ...................................... 12
       2.1.1 Learning algorithm ............................... 13
   2.2 The multi-layer perceptron ............................ 13
       2.2.1 Learning algorithms .............................. 14
   2.3 Fuzzy logic: Basic concepts .......................... 16
       2.3.1 $t$-norms and $t$-conorms ....................... 17
       2.3.2 Combinations of neural networks and fuzzy systems ... 20

3 The preprocessing perceptron ............................ 23
   3.1 Adding a preprocessing layer ......................... 23
       3.1.1 Generalising the preprocessing perceptron ......... 25
   3.2 Inputs to the system ................................ 27
       3.2.1 Pre-processing of the data ....................... 27
       3.2.2 Preparing a data set for training ................ 30
   3.3 Training the preprocessing perceptron ................ 32
       3.3.1 Batch vs. incremental learning ..................... 32
       3.3.2 Learning rates and training ....................... 33
   3.4 Outputs from the system ............................. 33
       3.4.1 Estimation and comparing decision support systems .. 34
       3.4.2 Complexity vs. approximation strength .......... 38

4 Summary of papers .................................... 39
   4.1 Aim of thesis ...................................... 39
   4.2 Results and contributions ........................... 40
4.2.1 Does the preprocessing perceptron work? .......... 40
4.2.2 Can the preprocessing perceptron be generalised? .... 40
4.2.3 How can the results be analysed? .................. 41
4.2.4 Can the preprocessing layer be interpreted? ......... 41
4.2.5 How to create and train a preprocessing perceptron? ... 42
4.3 Future work ........................................... 42

<table>
<thead>
<tr>
<th>Papers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper I</td>
<td>51</td>
</tr>
<tr>
<td>Paper II</td>
<td>63</td>
</tr>
<tr>
<td>Paper III</td>
<td>81</td>
</tr>
<tr>
<td>Paper IV</td>
<td>109</td>
</tr>
<tr>
<td>Paper V</td>
<td>125</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The first four chapters of the thesis form an introductory part. This part gives an introduction to the research area and the ideas behind the preprocessing perceptron. The first chapter sets the framework for the research and explains where the preprocessing is placed on the research map. The second chapter describes the preliminaries needed in neural networks and fuzzy logic. The preprocessing perceptron is described in detail and compared to other analysis methods in the third chapter. The introductory part is ended with a chapter containing a summary of the five papers that forms the second part of this thesis.

During the work with this thesis, the focus has been on the preprocessing perceptron. It is a method that is used to build decision support systems. Most experiments have been from the medical domain. This is due to the fact that the preprocessing perceptron is easy to interpret yet has discrimination capability that is comparable to more complex models.

This chapter describes the background for the main research field of the thesis, i.e. Medical Informatics and Intelligent Computing. The last section describes the production line, the original idea for the surroundings of the preprocessing perceptron, thereby answering some of the questions regarding where it is supposed to function and how.

1.1 Medical informatics

Medical informatics is an interdisciplinary field where usage, processing and creation of information and knowledge within the field of medicine and health care are concerned. The usage of the term "medical informatics" began in the end of the 1970s, other terms have also been used such as e.g. "medical information science" or "health informatics". Greenes and Shortliffe gave the following definition of Medical informatics [Greenes and Shortliffe, 1990]:

"Medical informatics is the field that concerns itself with the cognitive, information processing, and communication tasks of medical"
practice, education, and research, including the information science and the technology to support these tasks.

In the preface of [van Bemmel and Musen, 1997] it is stated:

> In medical informatics we develop and assess methods and systems for the acquisition, processing and interpretation of patient data with the help of knowledge that is obtained in scientific research.

Some examples of what the field of Medical Informatics might contain are¹ Health and clinical Management, Patient Records, Health Information Systems, Signal Processing, Image Processing, Knowledge Processing and Decision Support, Computer Supported Education, and Bioinformatics. For a thorough introduction to the area see e.g. [van Bemmel and Musen, 1997]. Work in this thesis is primary within the part concerning knowledge processing and decision support although the work touches several other areas as well.

1.2 Intelligent computing

Intelligent computing is a research field that has emerged from Artificial Intelligence. Sometimes the term Computational Intelligence is used as well. There are no sharp borders between the two fields and often conferences and journals are aimed at both at once. One way to make a distinction is that, within the field of artificial intelligence, the focus lies on logic and symbolic computation while within intelligent computing the focus is on numerical computations and methods such as neural networks and statistical methods.

1.2.1 Decision support systems

One of the major application areas within intelligent computing is systems that aid users to a decision or even make the decision for them. In the earliest days of artificial intelligence, the term expert system was used. It is somewhat misleading and is now often replaced by the term knowledge based system. When using the term expert system, it automatically raises the claim that the system is built on expert knowledge and it gives the impression that only hard, objective, and scientific facts have been used [Adam, 1998]. With e.g. medical expert systems in mind, it is obvious that a diagnosis is built not only on hard, objective facts but also on uncertain facts that are difficult to describe, common sense, and the experience of the clinician.

Knowledge based systems can be divided in two main groups of systems; the decision support systems and the decision making systems. A decision making system is designed to assist people with missing qualifications and knowledge when making decisions that requires expert knowledge. In this case the system actually makes the decision for the user. A decision support system on the other

¹This list is actually the table of contents of the Yearbook of medical informatics 2003 [Haux and Kulikowski, 2003].
1.3. The production line

hand provides information and advice to the user within a specific domain, e.g. medicine. This information and advice can be used by the user to make better and more accurate decisions but the system itself makes no decisions.

The main idea behind a decision support system is that the system should be based on the knowledge of one or more experts and be able to give the same advice, as a human expert would do. Furthermore, it is an advantage if the system does not require more time for its decisions than a human does. Often, users of a decision support system want it, in order for the system to be trusted, to be able to describe how it has reached its conclusions.

Traditionally most decision support systems were built on rules dictated by one or more experts, e.g. in the case of medicine, clinicians. The rules were mostly written in some, either formal or informal, language of logic. The drawbacks with this method are that it is very difficult to extract the knowledge and experience of a clinician and perhaps even more difficult to express them in logical form without losing any information in the process. Furthermore, the common knowledge in medicine changes and rules incorporated in a system may be outdated.

If existing data masses, containing many patients (obtained from patient record systems), are used and the system is able to learn from them, the problem of transforming information and experience from human experts to logic becomes less important. The information and experience is believed to be hidden implicitly in the journals containing symptoms and their corresponding diagnosis. The data in the journal must be transformed in some way to a representation acknowledged by the computer but there is no information explicitly represented. The problem with outdated knowledge is easier to overcome with this kind of system because it is easier to build the system automatically and therefore also easier to rebuild and update it.

Researchers have been working with expert systems/knowledge based systems since the late 50s. One of the most famous medical expert systems is MYCIN [Shortliffe, 1976], which was developed during the 70’s. MYCIN could diagnose certain kinds of bacterial infections and recommended administration of appropriate drug therapies. It is a rule-based system with a backward-chaining inference engine (see also Section 1.3.2). This engine was later made as a separate program which led to the possibility to use the same rule-based syntax applied on new medical areas. INTERNIST-I is a system developed during the 80’s that made diagnosis within internal medicine. It has been further developed into the commercial program QMR, which handles more than 600 diseases and 4500 findings [van Bemmelen and Musen, 1997].

1.3 The production line

One of the underlying concepts presented in this thesis is the production line for building new decision support systems, depicted in Figure 1.1. This system is trying to "eliminate" the system developer from the development of decision support systems. The idea is that the production line is part of a software
system containing many decision support systems. When the medical expert wants to create a new decision support system, there are three steps necessary: There must be facts available and the extraction phase gives the expert tools that easily combine journals from different places and retrieve information from them. In the analysis phase there are several methods available for building decision support systems. It should also be possible to combine two or more methods into hybrid systems. Finally, when a decision support system has been created it must be incorporated into the existing software system in a seamless way. The production line has also been discussed in [Eklund et al., 1999, Kallin, 1998].

![Diagram of the production line for decision support systems.](image)

**Figure 1.1:** The production line for decision support systems.

### 1.3.1 The extraction phase

The first step in the production line, the extraction phase, extracts facts from the vast medical databases and transforms the data into a form that can be managed by a computer. Medical databases are mainly constructed to fulfil two tasks: administrate data concerning the staff and equipment at the hospital and administrate data concerning individual patients. In Sweden, databases are used by larger hospitals and many of the smaller clinics. The problem with the databases is that they are developed over a long period, the format of the data may have changed, and the files are spread over different servers. In most cases, each health care centre or clinic keep their own record over a patient. This implies that a single patient might have several records. The data stored regarding a patient is not the same at a local clinic as at a large university hospital and the database management systems may also be different. All these aspects make information regarding patients difficult to compare and/or combine.

There are virtually no databases constructed for clinical research. Since
the clinician mainly wants information regarding one patient at a time, the databases are constructed to make such searches easier and faster. When clinical research is performed, there is no need to know the identity of a specific patient but rather to gather large amounts of data from a group of patients and combining e.g. laboratory tests, medications, and diagnoses. That is, collecting information regarding a specific disease or laboratory test or even medication at a time instead of information regarding a specific patient. One way to accomplish this is to create large data pools containing information from several databases.

In order for data pools to work, all patient records at all care units must be computerised. However, in Sweden there are regulations that stipulate that patient information may not be registered or used in research in any form without permission of the patient [Knoll, 1996]. Furthermore, the information stored in a data pool may not be comparable since laboratories at different hospitals may use slightly different equipment and/or calibrate it differently. The data may be inaccurate due to mistakes when typing it in or when interpreting measurements.

One prototype for the extraction phase has been implemented for and used by Turku University Central Hospital (TUCH) [Selén, 1996]. Although the system is tightly coupled with the specific laboratory information system of TUCH, the implementation is general and that makes it possible to generalise the program to handle a data pool as described above.

**Patient record systems**

The data stored regarding a patient can be divided into three types: text, measurements, and signals of different dimensions. A large part of the information regarding a patient is text such as the patient’s own description of the disease, the preliminary diagnosis, and notes regarding patient’s progress during a hospital stay. In order to find structures in the data masses, the notations and concepts must be standardised. This is not an easy task considering that there are about 35 000 different diseases and syndromes where the notations sometimes overlap [Jacobson, 1995]. Measurements are values that can easily be stored as a number, e.g. blood pressure, temperature, pulse etc, and some examples of signals are ECG-curves, ultrasound, x-rays, and CT/MR volumes.

Records on paper take an enormous amount of space. In Sweden, it is estimated that conventional patient record systems need two running kilometres per million inhabitants plus five running kilometres x-rays per million inhabitants. Patient records often disappear, 20% of the records are not found when searched for the first time, and about 5% are never found [Jacobson, 1995]. Other drawbacks with patient records on paper are that the record can only be read by one user at the same time, it is difficult to do systematic searches, and a substantial amount of time is spent on updating, copying, and adding information to the record. The computerised patient record has shorter access time and the material stored can be organised and grouped depending on the user’s demands. Even if computers and software may be expensive initially, it is an economic
way to store large amounts of data. Some drawbacks with computerised patient record systems should be mentioned: it is difficult to browse the patient record and get the same overview on a terminal as on paper. If the system uses too much structured information and standardised terms, nuances in information may get lost.

Secrecy

All personnel in Swedish health care have the obligation to observe professional secrecy regarding the contents in a patient record. The Secret Act says that the information in a patient record must be kept secret from everyone except the care provider and the patient. When working with computerised patient record systems it is important not to undermine the confidence of patients in the way information is handled. The secrecy issues together with the fact that each single patient must give her/his permission before data is used may give a very pessimistic view concerning the data pools. However, if data pools were built, the data would be totally unidentifiable in the sense that it is impossible to work information back to patients’ identities. This may imply that the pools could be built without breaking any secrecy rules or raise any integrity issues.

It must be emphasised that the patient data referred to, in this thesis and used in the case studies, are gathered from patients by clinicians following the rules and secrecy issues described above.

1.3.2 The analysis phase

In the next step, the analysis phase, the goal is to find a suitable analysis method, which is used to build a decision support system for a particular disease. Instead of making interviews with experts as traditionally done, the decision support systems described in this thesis try to get knowledge out of existing facts in data masses, e.g. patient records. The decision support systems should produce the same answers as the clinician, i.e. if the patient has a positive diagnosis the system should produce a high risk-value (close to ‘true’) and a low value (close to ‘false’) otherwise.

The analysis phase can be represented by a kind of a black box that transforms the inputs to an output value. The inputs $x_1, x_2, \ldots, x_n$ and the diagnosis $y$ are known and the goal is to find the function $y = f(x_1, x_2, \ldots, x_n)$ that is the best approximation of the data given. The black box consists of analysis methods, such as logical, statistical, and neural methods, used to mimic the clinician’s reasoning. The preprocessing perceptron is one of these analysis methods and will be described in detail in the Chapter 3.

Inputs to the system

As mentioned before, information about a patient can be of three types, text, measurements, and signals. Signal processing and analysis is a large research field and will not be mentioned further here. For a decision support system to
be able to use the data, it must be transformed to a representation that can be interpreted by a computer. Measurements are easy to handle, often they consist of numerical values that can be interpreted directly by the computer or interpreted after a simple transformation.

Textual information is more difficult to interpret by a computer, especially if there are no standards for terms used in free-text fields. It exists an standard called "International Statistical Classification of Diseases and Related Health Problems" [World Health Organization, 2003] where diagnoses and symptoms are coded in a very detailed manner. However, the codes are not always present in a journal and they can be accompanied by more free-text information which is usually not standardised in a formal way. It is difficult to convert the text "the patient has headache" to a number for example. Of course one may use a binary number, 0 standing for not headache and 1 for headache but often it is useful to interpret nuances in the symptoms. Headache may be very different from person to person and from time to time.

Sometimes a single measurement is not enough, the clinician must consider changes in values over time. One example of this is the myocardial infarction, were the change of one significant marker, CK-MB, during the first day is very useful for diagnosis. The actual interpretations of the symptoms do also differ. A numerical value of 37.7 standing for body temperature may mean one thing when considering the diagnosis pneumonia and another when considering appendicitis. With all data in computerised patient records (as well as in written records) there is the problem of mistakes when putting them into the system. Somebody may put 1 instead of 10 as a measurement or forget to write the data altogether. A decision support system learning from examples must be able to handle these anomalies in a good way.

Analysis methods

In this section, examples of analysis methods will be given. The trend seems to be shifting from logically founded methods, common in the early years of artificial intelligence, to hybrids of two or more types of methods.

**Statistical methods** are used to estimate risks. The knowledge is obtained by collecting data and looking at the distribution of ill and healthy patients. The risk for a specific patient is then calculated from these distributions according to special formulas. The Bayes decision theory is the ground for many statistical methods and e.g. Bayesian belief networks are popular. The main drawback with these methods are that they require information regarding conditional densities [Duda et al., 2001] which often are unknown.

**Logic** is one of the oldest traditions within the field of artificial intelligence. The thought of imitating the real world by using logic to manipulate symbols representing real world objects has pervaded the research in artificial intelligence ever since its creation during the 50’s. Logic has traditionally often been used to represent knowledge, especially the predicate logic has been popular. However, the predicate logic has only two truth-values, true and false, and is focused on propositional knowledge. Therefore, predicate logic in particular and traditional
logic in general is not able to capture uncertainties in a good way. This means that traditional logic has large difficulties handling procedural knowledge and common sense [Adam, 1998].

The inference is done by manipulating the facts according to certain logical rules. The medical expert system MYCIN was built on logic and in the introduction of Shortliffe’s book regarding MYCIN [Shortliffe, 1976] he talks about deduction, induction, and abduction. In the book, Shortliffe uses the following example to distinguish between the three:

Consider, for example, the three statements:

1. If a person has pneumonia, then he has fever,
2. John has pneumonia,
3. John has fever.

Deductive logic allows us to derive 3 from 1 and 2, i.e. "since people with pneumonia have fever, and since John has pneumonia, John must have a fever". Induction, on the other hand, uses one or more observations of people for whom 2 and 3 hold in order to infer that 1 is true. I.e. "since I have observed several people with pneumonia, all of whom have fever, it is perhaps generally true that people with pneumonia have fever". Abduction is the remaining combination, namely using 1 and 3 to infer 2, i.e. "since people with pneumonia have fever and since John has fever, perhaps it is true that John has pneumonia."

MYCIN was built using abduction as inference rule and it can be seen from the example above that it seems to be the most natural way to handle logical facts concerning medicine. However, there is a problem with the truth-values. If there is a logical rule, stating that a patient with a temperature of \(38^\circ C\) or higher is ill, a patient with the temperature \(37.9^\circ C\) will be considered as healthy. Furthermore, it is difficult to translate the knowledge of a clinician to logical formulas. The knowledge has traditionally been given through interviews and it is difficult to translate linguistic terms as large, medium, and small into sufficient numerical values.

Fuzzy logic is an approach trying to handle linguistic terms in a logical way. This makes it possible to express the knowledge of a domain expert in a more natural way when building a decision support system. In fuzzy logic, the truth-value of a statement is given by a function and the truth-values can be any number in the continuous interval \([0, 1]\). As in the case of traditional logic, the inference engine uses specific logical rules. The fuzzy logic connectives and their truth-values are different from those in two-valued logic and e.g. the implication function can be expressed in more than one way. Section 2.3 will give some of the basic concepts and references to more reading on the subject.

When using neural networks the knowledge lies in the weights of the network. The idea behind neural networks is to imitate the neurons in the human
1.3. The production line

brain. Through mathematical formulas and a special training phase, the network "learns" to make the right diagnosis. See Chapter 2 for a more detailed description of neural networks. The training phase uses large data masses with examples of healthy and ill patients. The network "learns" by trying to make a diagnosis according to a specific case and then correcting itself if the diagnosis is wrong. After the training phase is done, the network is the inference engine. Given input, it produces outputs representing a risk value.

Outputs from the system

Outputs from a decision support system often lie in the interval \([0, 1]\) (except when using predicate logic where the results are either 0 or 1). The number 0 stands for a negative diagnosis, i.e. disease is not present, and 1 stands for a positive diagnosis, i.e. disease is present. This number could be presented directly to the user or transformed in some way before presentation. If a larger decision support system is built that tries to distinguish between diagnoses the system may also come with suggestions of other inputs to use or tests to be made.

The analysis methods described in this section are suitable for applications with well-defined inputs and where the output is a kind of a classification as ill/healthy, high/low risk etc. This approach applies very nicely to areas like virology, infectious diseases, clinical chemistry, and endocrinology. When the inputs are time-dependent and/or the outputs are something else than risk values, e.g. the areas of x-rays and ECG, other methods may be more successful. Those methods are however often built of smaller modules. Some of those modules may be decision support systems of the type described in this thesis.

1.3.3 The synthesis phase

The goal for the last step, the synthesis phase, is to integrate the new system into a clinician’s workbench. In this way, the clinician is able to build and evaluate the system. Who can better decide if a system is working or not?

Prototypes implementing the production line have been built using the World Wide Web as the workbench. After the decision support system is created it is presented as a web page where a clinician can give symptoms as inputs and receives a risk value as output [Eklund et al., 1999]. In Figure 1.2, a web page for the decision support system for Polycystic Ovary Syndrome is shown. In a master’s thesis [Karlsson, 1998], another prototype for the synthesis phase is built and described. This prototype also contains a part of the analysis phase, only able to use traditional neural networks. When the program starts, the user gives the necessary facts for training the network together with directory names for the resulting web pages.

Let us summarise some of the main points of the production line. The production line is supposed to aid clinicians when building decision support systems on their own. The goal of the extraction phase is to create a large file with comparable data regarding a specific disease from both ill and healthy patients.
Chapter 1. Introduction

Figure 1.2: The decision support system for the Polycystic Ovary Syndrome.

Data pools must be created for an extraction program to work smoothly. There are some problems when building a data pool, the data from different patient records may be difficult to compare and there are secrecy issues to handle. The main advantage with data pools is that it is easy to extract data regarding a specific disease or measurement as opposed to extract data regarding a single patient.

The data file from the extraction phase is used as input to the analysis phase where the clinician searches for a suitable analysis method. In this section, examples have been given of different analysis methods such as statistical, logical, neural, and fuzzy logical methods. The search for a method is done with the help of software where the clinician tries and tests different methods. When the analysis method is selected, and if needed, training is performed, the method and its corresponding parameters are sent to the synthesis phase.

The synthesis phase takes the chosen method and the parameters and merges them with the surrounding software. The result is presented as a natural part of the clinician’s workbench.
Chapter 2

Neural networks and fuzzy logic

In this chapter, a description of neural networks, especially perceptrons and multi-layer perceptrons\(^1\) will be given together with a short discussion regarding fuzzy logical connectives. The main purpose of this chapter is to set the preliminaries for Chapter 3 where the preprocessing perceptron and its generalisation, which can be seen as a fuzzy-neural hybrid, will be described.

The perceptron [Rosenblatt, 1958], was one of the first attempts to find a model for the neuron in the human brain. The brain is composed of about \(10^{11}\) neurons that are connected in a large network [Hertz et al., 1991]. A multi-layer perceptron imitates the brain by connecting the perceptrons and putting weights on the connections to symbolise the strength in the impulses. All incoming values are summarised in a perceptron and if the resulting value is above a special threshold, an impulse is sent on through the network.

Multi-layer neural networks are characterised by four aspects: a large number of neurons, weighted connections between these neurons (the weights can change and this symbolises learning), parallel computing, and focus on applications within automatic learning and pattern recognition. It can be shown that multi-layer perceptrons are universal approximators. Any function can be learned with an arbitrary accuracy by a three-layer network and every bounded continuous function can be learned with a small error by a two-layer network [Hornik, 1989]. Drawbacks with multi-layer neural networks are that the operator must decide what all input and output stands for logically. It is often difficult to understand what happens during the training of a neural network and someone must interpret the results. The knowledge lies implicitly in the network.

\(^1\)For descriptions of other types of neural network see e.g. [Hertz et al., 1991, Ripley, 1996].
2.1 The perceptron

In feed-forward neural networks, the neurons (or units) are organised into layers with feed-forward connections in between. Note that the inputs do not form a layer since they do not perform any computations. In Figure 2.1, an example of a simple perceptron with four inputs and three output nodes in the output layer is shown. The notations for the nodes and weights are also shown.

Figure 2.1: A (simple) perceptron with four inputs and three outputs.

The perceptron calculates a weighted sum of its inputs $x_i$ and outputs a value within a defined interval, usually $[0, 1]$ or $[-1, 1]$, according to a non-linear function $\sigma(x)$ called the activation function. The weight $w_{ji}$ represents the strength of the synapse on the connection from neuron $i$ to neuron $j$. The parameter $\theta_j$ is the threshold for unit $j$, the weighted sum must be at least as large as the threshold for the neuron to fire. The notion of a neuron firing is easiest seen when a step function is used as an activation function. The step function returns zero for all values below the threshold and one for all other values. The resulting function for the perceptron becomes

$$o_j = \sigma \left( \sum_{i=1}^{n} w_{ji} x_i - \theta_j \right) = \sigma \left( \sum_{i=0}^{n} w_{ji} x_i \right). \tag{2.1}$$

Note that the threshold can be omitted from the first expression in Equation 2.1 by treating it as a weight to an extra input with the constant value -1. That is, let $x_0$ have the fixed value -1 and then $\theta_j = w_{j0}$. The activation function can, for example, be the step function, the sign function, the tanh function or the sigmoid function defined by

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.2}$$

The inputs to the network are usually linearly normalised to the $[0, 1]$-interval in order to reduce the changes in magnitudes between the different kind of inputs.
2.2. The multi-layer perceptron

2.1.1 Learning algorithm

The goal of training a neural network is to find a set of weights such that the error between the desired output and the actual output is minimised. The delta rule [Rumelhart et al., 1986] is based on the gradient descent algorithm. An error measurement for a perceptron is defined as

$$E = \frac{1}{2} \sum_{k=1}^{P} \sum_{j=1}^{M} (t^k_j - o^k_j)^2 = \frac{1}{2} \sum_{k=1}^{P} \sum_{j=1}^{M} (t^k_j - \sigma(z^k_j))^2$$

$$= \frac{1}{2} \sum_{k=1}^{P} \sum_{j=1}^{M} \left( t^k_j - \sigma \left( \sum_{i=0}^{n} w_{ji} x^k_i \right) \right)^2,$$

where $P$ is the number of patterns, $M$ the number of outputs, $n$ the number of inputs, $o^k_j$ the $j$th output of the network for pattern $k$, and $t^k_j$ is the desired output for pattern $k$. $E$ is positive and gets smaller as the performance of the net gets better. The gradient descent algorithm tells us to change the weights $w_{ji}$ by an amount $\Delta w_{ji}$, a sum of changes each proportional to the gradient of $E$ at the present location ($\eta$ is called the learning rate):

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = \eta \sum_{k=1}^{P} (t^k_j - o^k_j) \sigma'(z^k_j) x^k_i.$$

For some choices of activation function $\sigma$, the derivative is particularly simple to compute. If the sigmoid (Equation 2.2) is used, the derivative can be shown to be $\sigma'(z) = \sigma(z)(1 - \sigma(z))$. As can be seen, only the value of the function is needed in order to compute the derivative.

The idea is to present a pattern to the network, calculate $\Delta w_{ji}$, update the weights, present a new pattern again and so on, until the error measurement is small enough. Sometimes batch training is used, i.e. all patterns are presented to the network before updating the weights.

Minsky and Papert [Minsky and Papert, 1969] showed that the one-layer perceptron can only be applied to structures that are linearly separable and therefore can not solve e.g. the XOR-problem.

2.2 The multi-layer perceptron

The multi-layer perceptron has a greater representation power and is able to represent any boolean function (including XOR) [Hertz et al., 1991] with just one hidden layer. An example of a multi-layer perceptron is shown in Figure 2.2. The layers between the input values and the output layer are called hidden layers. Each node in the first hidden layer produces an output value for a specific pattern, $net^1_j = \sigma \left( \sum_{i=0}^{n} w_{ji} x_i \right)$. This value is used as input to the nodes on the second layer and their output become $net^2_k = \sigma \left( \sum_{j=0}^{n^2} w_{kj} net^1_j \right)$ and, finally,
for the outputs from a three-layered net (as the example in Figure 2.2) we have
\((n_i)\) is the number of values in layer \(i\)

\[ o_l = \sigma \left( \sum_{k=0}^{n_3} w_{lk} \text{net}_k^2 \right) = \sigma \left( \sum_{k=0}^{n_3} w_{lk} \sigma \left( \sum_{j=0}^{n_2} w_{kj} \sigma \left( \sum_{i=0}^{n_1} w_{ji} x_i \right) \right) \right). \tag{2.3} \]

The threshold has been taken care of with the help of an extra unit as described in the previous section. From this example it is easy to see how the output values for an arbitrary multi-layer perceptron are calculated.

### 2.2.1 Learning algorithms

The delta rule can be generalised to networks with more than one layer and is the base of the famous backpropagation algorithm, see e.g. [Rumelhart et al., 1986, Hertz et al., 1991]. The delta-rule summarises the errors from all patterns and updates the weight according to the sum. The backpropagation algorithm is usually used on one pattern at a time but can be used for batch training as well. In this section, an outline for the case of a single pattern will be showed. To make things easier we will use the network structure in Figure 2.2 in our example.

The error function used is \( E = \frac{1}{2} \sum_{i=0}^{M} (t_i - o_i)^2 \) where \( o_i \) is given by Eq. 2.3. We want to find the values of \( \Delta w_{lk}, \Delta w_{kj}, \) and \( \Delta w_{ji} \) and we therefore need three derivatives as shown below.

\[ -\frac{\partial E}{\partial w_{lk}} = (t_l - o_l) \sigma'(z_l) \text{net}_k^2 = \delta_l \text{net}_k^2, \]
2.2. The multi-layer perceptron

where \( \delta_l = (t_l - o_l)\sigma'(z_l) \) and \( z_l = \sum_{k=0}^{n_3} w_{lk}net_k^l \).

\[
-\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k^l} \frac{\partial net_k^l}{\partial w_{kj}} = \sum_{l=0}^{M} (t_l - o_l)\sigma'(z_l)w_{lk}\sigma'(z_k)net_j^l
\]

\[
= \sum_{l=0}^{M} \delta_l w_{lk}\sigma'(z_k)net_j^l = \delta_k net_j^l
\]

where \( \delta_k = \sigma'(z_k) \sum_{l=0}^{M} \delta_l w_{lk} \) and \( z_k = \sum_{j=0}^{n_2} w_{kj}net_j^l \).

\[
-\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial net_j^l} \frac{\partial net_j^l}{\partial w_{ji}}
\]

\[
= \sum_{l=0}^{M} (t_l - o_l)\sigma'(z_l) \sum_{k=0}^{n_3} w_{lk}\sigma'(z_k)w_{kj}\sigma'(z_j)x_i
\]

\[
= \sum_{l=0}^{M} \delta_l \sum_{k=0}^{n_3} w_{lk}\sigma'(z_k)w_{kj}\sigma'(z_j)x_i
\]

\[
= \sum_{k=0}^{n_3} \delta_k w_{kj}\sigma'(z_j)x_i = \delta_j x_i
\]

where \( \delta_j = \sigma'(z_j) \sum_{k=0}^{n_3} \delta_k w_{kj} \) and \( z_j = \sum_{i=0}^{n_1} w_{ji}x_i \). The update rule is, as in the delta-rule, \( w_{new} = w_{old} + \eta \Delta w \).

As we have seen, the backpropagation algorithm works by presenting a pattern to the network, calculating the error and then propagating it backwards in the network. Errors obtained on a higher level (the \( \delta \)-values) are used to calculate errors on lower levels.

One drawback of the backpropagation algorithm is that the learning may stop in local minima. Another drawback is that the learning is rather time-consuming. A reason for this is that in order to avoid oscillation in the change of the parameters, the learning rate must be rather small. Several variations of the backpropagation algorithm exist that try to make it faster, avoid local minima and increase the generalisation ability [Hertz et al., 1991].

The resilient propagation rule [Riedmiller and Braun, 1993] is a way to handle the problem with the size of the partial derivative. Only the sign of the derivative is considered and the sign is used to indicate the direction of the weight update. Every time the partial derivative of the error changes its sign, the last update was too big and a (local) minimum was jumped over. In this case, the update-value is decreased. If the partial derivative does not change its sign, the update-value can be increased in order to speed up the training.

The size of the change \( \Delta w_{ji} \) is decided by an 'update-value' \( \Delta_i \):

\[
\Delta w_{ji} = \begin{cases} 
-\Delta_i & \text{if } \frac{\partial E}{\partial w_{ji}} > 0, \\
+\Delta_i & \text{if } \frac{\partial E}{\partial w_{ji}} < 0, \\
0 & \text{otherwise.}
\end{cases}
\]
\[ \frac{\partial E_t}{\partial w_{ji}} \] is the sum of the partial derivatives for all patterns at time \( t \). The start value of \( \Delta_t \) should be chosen according to the initial values of the weights themselves, for example \( \Delta_0 = 0.1 \) (default setting). The choice of this value is rather uncritical, since it is adapted as learning proceeds [Riedmiller and Braun, 1993].

Every time the partial derivative of the error changes its sign, the last update was too big and a (local) minimum was jumped over. In this case, the update-value is decreased. If the partial derivative does not change its sign, the update-value can be increased in order to speed up the training.

\[
\Delta_t = \begin{cases} 
\eta^+ \Delta_{t-1} & \text{if } \frac{\partial E_t}{\partial w_{ji}} \cdot \frac{\partial E_{t-1}}{\partial w_{ji}} > 0, \\
\eta^- \Delta_{t-1} & \text{if } \frac{\partial E_t}{\partial w_{ji}} \cdot \frac{\partial E_{t-1}}{\partial w_{ji}} < 0, \\
\Delta_{t-1} & \text{otherwise.} 
\end{cases}
\]

The weight is updated according to the formula
\[ w_{ji}^{t+1} = w_{ji}^t + (\Delta w_{ji})^t, \]
except when a change of sign has occurred. In that case
\[ (\Delta w_{ji})^t = - (\Delta w_{ji})^{t-1}. \]

In order to avoid a double punishment of the update-rule there should be no update of the parameter in the next step. This is achieved by setting \( \frac{\partial E_{t-1}}{\partial w_{ji}} = 0 \) in the rule above.

The value of \( \eta^+ \) and \( \eta^- \) are set to fixed values, the values proposed by Riedmiller are \( \eta^- = 0.5 \) and \( \eta^+ = 1.2 \) [Riedmiller and Braun, 1993]. There are two other parameters needed \( \Delta_{\text{min}} \) and \( \Delta_{\text{max}} \). Usually \( \Delta_{\text{min}} = 1e^{-6} \) and \( \Delta_{\text{max}} = 50.0 \).

Resilient propagation is not used in any of the papers in this thesis but has been used in experiments where the results have been encouraging. For more descriptions of experiments and comparisons between backpropagation and resilient propagation see [Riedmiller and Braun, 1993].

### 2.3 Fuzzy logic: Basic concepts

In traditional logic a proposition can either be true or false (have truth-value 1 or 0). Either the object belongs to the given set or it does not. However, some things can not be described as either true or false. It is difficult to draw a line between the property tall and the property short. If someone says that all people above 1.70 m are tall, does that mean that a person of length 1.69 m is short? In fuzzy logic, the truth-value is continuous in the interval \([0, 1]\). This means that the person can be long with truth-value 0.6 and short with truth-value 0.4. Fuzzy logic makes it possible to express the knowledge in linguistic terms and interpret them in a logical way.
This section is intended to give the notations needed for the discussion in the rest of the thesis. A description of fuzzy controllers and fuzzy systems is given by [Kruse et al., 1994]. For a deeper discussion on fuzzy logic and \( \text{t-norms} \), see for example [Kruse et al., 1994, Mizumoto, 1989]

A fuzzy set \( A \) in \( X \) is characterised by a membership function

\[ \mu_A(x) : X \to [0, 1], \]

from the reference set \( X \) to the unit interval. The value of \( \mu_A(x) \) represents the "grade of membership" of \( x \) in \( A \). In our example with lengths, the reference set is the possible length in metres and the fuzzy set \( A \) may be the linguistic term "short". In that case \( \mu_A(1.70) = 0 \) says that a person with length 1.70 m is not short while \( \mu_A(1.69) = 0.1 \) states that 1.69 m is short with grade of membership 0.1. \( F(X) \) denotes the set of all fuzzy sets of \( X \). In our example \( F(X) \) may contain the fuzzy sets representing the linguistic terms "short", "medium", "tall", "very tall" etc.

The membership functions can be defined in many different ways and may be based on different types of functions. The membership function may be discrete or continuous, be triangular, trapezoidal, Gaussian or sigmoidal. Often families of membership functions are used for a set of linguistic terms.

The usual set operations can be defined on fuzzy sets as well. The values of e.g. the union \( C \) between the fuzzy sets \( A \) and \( B \) is defined point by point using a function satisfying certain conditions. The function defining the union must fulfil the conditions for \( \text{t-norms} \) and can for example be defined by the formula

\[ \mu_C(x) = \max\{\mu_A(x), \mu_B(x)\}. \]

The intersection can be defined by the function \( \min\{\mu_A(x), \mu_B(x)\} \). The function defining the intersection between two sets must fulfil the conditions for \( \text{t-norms} \). The complement of the set \( A \) is defined by \( 1 - \mu_A(x) \). The definition of \( \text{t-norms} \) and \( \text{t-conorms} \) are given in the next section.

### 2.3.1 \( \text{t-norms} \) and \( \text{t-conorms} \)

In order to be able to use logic on fuzzy sets, the logic connectives must be defined for them. For the cases of conjunction and disjunction, the set operations earlier defined for intersection and union can be used, and the negation is implemented using the complement function. The grade of membership now denotes the truth-value. For more details regarding \( \text{t-norms} \) and \( \text{t-conorms} \) see [Schweizer and Sklar, 1961].

A function \( T : [0, 1]^2 \to [0, 1] \) is called a \( \text{t-norm} \) if the following holds:

1. \( T(a, 1) = a \) (unit element),
2. \( a \leq b \Rightarrow T(a, c) \leq T(b, c) \) (monotonicity),
3. \( T(a, b) = T(b, a) \) (commutativity),
4. \( T(a, T(b, c)) = T(T(a, b), c) \) (associativity).
Chapter 2. Neural networks and fuzzy logic

T is monotonic non-decreasing in both arguments and \( T(a, 0) = 0 \). A function \( S : [0, 1]^2 \to [0, 1] \) is called a \( t \)-conorm if and only if the following holds:

1. \( S(a, 0) = a \) (unit element),
2. \( a \leq b \Rightarrow S(a, c) \leq S(b, c) \) (monotonicity),
3. \( S(a, b) = S(b, a) \) (commutativity),
4. \( S(a, S(b, c)) = S(S(a, b), c) \) (associativity).

From the definition above we get that \( S \) is monotonic non-decreasing and that \( S(1, x) = 1 \). If a \( t \)-norm \( T \) is given, the \( t \)-conorm \( S \) can be found by the formula

\[
S(x, y) = 1 - T(1 - x, 1 - y).
\]

The \( t \)-norm \( \min(a, b) \) is often used to represent \( a \lor b \) and the \( t \)-conorm \( \max(a, b) \) is often used to represent \( a \land b \). The pair of \( \min(a, b) \) is the \( t \)-norm giving the highest values and \( \max(a, b) \) is the \( t \)-conorm giving the lowest values [Kruse et al., 1994].

If a \( t \)-norm \( T \) is continuous and \( T(x, x) < x, \forall x \in (0, 1) \) it will be called Archimedian. Furthermore, it will be called strict if and only if \( T \) is strictly increasing in \((0, 1)^2\). In the same way a \( t \)-conorm \( S \) will be called Archimedian if and only if \( S \) is continuous and \( S(x, x) > x, \forall x \in (0, 1) \). An Archimedian \( t \)-conorm will be called strict if and only if \( S \) is strictly increasing in \((0, 1)^2\).

Every Archimedian \( t \)-conorm can be represented by a continuous and increasing function \( f : [0, 1] \to [0, \infty) \) with \( f(0) = 0 \) such that

\[
S(x, y) = f^{-1}(f(x) + f(y))
\]

where \( f^{-1} \) is the pseudo inverse of \( f \), defined by

\[
f^{-1}(y) = \begin{cases} f^{-1}(y) & y \in [0, f(1)], \\ 0 & y \in (f(1), \infty). \end{cases}
\]

The function \( f \) is called an additive generator of \( S \). A \( t \)-conorm is strict if and only if it is generated by an additive generator \( f \) with \( f(1) = \infty \) (otherwise it is generated by an additive generator such that \( f(1) = 1 \) [Mizumoto, 1989].

As an example \( f(x) = \sqrt{1 - x^2} \) is a continuous function describing a half-circle in the \([-1, 1]\)-interval. Let \( f' \) denote the derivative with respect to \( x \). Let \( S(x, y) = (-f')^{-1}((-f'(x)) + (-f'(y))) \) where

\[
-f'(x) = \frac{x}{\sqrt{1 - x^2}}
\]

and

\[
(-f')^{-1}(x) = \frac{x}{\sqrt{x^2 + 1}}
\]
then $S(x,y)$ is a strict Archimedean $t$-conorm. $S(x,y)$ is Archimedean since $S(x,x) > x, \forall x \in (0,1)$ and it is strict since $-f'(1) = \infty$. The inverse function can also be seen as an activation function in a neural network since it is strict increasing from $\mathbb{R}$ to $(-1,1)$. The plots of $-f'(x)$ and $(-f')^{-1}(x)$ for the half-circle function $f(x)$ is shown as plots a) and b) in Figure 2.3.

$$S(x; y) = \tanh(\tanh^{-1}(x) + \tanh^{-1}(y))$$ (2.4)

$tanh^{-1}$ takes values from the interval $[-1,1]$ and the result lies in $(-\infty, \infty)$, see plots c) and d) in Figure 2.3. In the case of $t$-conoms, however, the input always lies in $[0,1]$ and therefore the result lies in $[0,\infty)$. Given values from that interval, $\tanh$ returns values in $[0,1)$. As have been seen, $\tanh^{-1}$ fulfils the
conditions for an additive generator and therefore the result of Equation 2.4 is
a \( t \)-conorm.

The half-circle, in our first example, is a natural choice as an additive gen-
erator for a \( t \)-conorm and each \( t \)-conorm has a corresponding activation function
as shown in the example above. In the same way, the \( \tanh \) function is a natural
choice for an activation function and has a corresponding \( t \)-conorm. Notable is
that the \( t \)-conorms seldom exist as activation functions in the neural network
literature and the activation functions are seldom found as \( t \)-conorms in the
fuzzy logic literature.

2.3.2 Combinations of neural networks and fuzzy systems

Earlier in this thesis, it has been shown that neural networks are one way to
build a decision support system that learns from numerical data. It has also
been shown that fuzzy logic can be used to represent knowledge in linguistic
form. In this section, different ways to combine these methods are described.
The aim is to combine the benefits from the characteristics of fuzzy systems
with those of neural networks, see Figure 2.4.

![Figure 2.4: Fuzzy systems and neural networks can be combined into hybrid systems.](image)

There are at least two terms for this combination depending on the research
focus. Researchers within fuzzy systems that incorporate neural networks call
it a neuro-fuzzy system, see e.g. [Ojala, 1994, Bossley, 1995, Hayashi, 2000]. If
the focus is neural networks and fuzzy logic is incorporated, it is called a fuzzy-
neural network. See e.g. [Buckley and Hayashi, 1994, Gupta and Rao, 1994,
Pedrycz, 1993].

**Neuro-fuzzy systems**

One way to build a neuro-fuzzy system is to use a neural network to approxi-
mate a fuzzy system. It is known that neural networks are universal approxi-
mators [Funahashi, 1989]. That is, given a continuous function \( f : [0, 1]^n \to \mathbb{R} \)
there is a neural net that can approximate \( f \) to any degree of accuracy. This
might, however, require an infinite number of hidden nodes in the network.
2.3. Fuzzy logic: Basic concepts

In [Buckley and Hayashi, 1993], it is shown that given a continuous (discrete) fuzzy expert system, or a continuous fuzzy controller, there is a neural net that can approximate it to any degree of accuracy.

When approximating a fuzzy system, the network is not used for learning - only for computation. This means that learning algorithms are not considered for this type of hybrid networks. The reason for approximating a fuzzy system is to take advantage of the fact that neural networks are computationally fast parallel methods [Buckley and Hayashi, 1993].

Another way to introduce neural nets in a fuzzy system is to use a neural net to tune a fuzzy system. One of the steps in building a fuzzy system involves identifying the main control parameters. For each linguistic term, one must determine a set of terms. This set must be at the right level of granularity to describe the domain in a satisfactory way. In our earlier example of lengths (Section 2.3) the set of terms {short, medium, long} may not be satisfactory and the larger set {very short, short, medium, long, very long} should be used instead. Furthermore, each term is represented by a membership function containing parameters. These parameters must be tuned in order for the fuzzy system to work well and this training is sometimes done with the help of a neural net [Berenji and Khedkar, 1992, Zhou, 1996].

The network represents the set of logical rules in the fuzzy system and gives the same output, as the fuzzy system would have done. If the responses to given inputs are known, a supervised learning mechanism is used. The output is compared to the expected one and any errors are used to change the parameters in the membership functions. If the responses are not known, as in most real-world applications, reinforcement learning is used. In that case, the only response sent back to the network is if the output is good or bad.

Fuzzy-neural networks

In fuzzy-neural networks, the ideas from fuzzy logic are incorporated in the neural networks. This can be done by combining a neural part and a fuzzy part into one system. The fuzzy part can be used as an preprocessing interface to the network, translating linguistic terms to a form understandable for the network. In this setup, the parameters in the network are the only entities changing during a learning phase. The parameters in the fuzzy part are not changed. The fuzzy part can also be applied after a neural network, the result from a network is run through a fuzzy system, and the resulting values are used to update the weights in the network.

Another example of how fuzzy ideas are incorporated into neural networks is to merge neural and fuzzy ideas into a hybrid system. One way to build a hybrid is to have inputs and outputs as real numbers while the operations are fuzzy. Instead of using the weighted sum as the semantics for the implication function and the summation together with the activation function as semantics for the disjunction function, fuzzy logic connectives are used. This approach is used in the preprocessing perceptron described in Chapter 3 and will be further described there.

21
If the operations are neural, while the inputs and weights are fuzzy we have yet another way to create fuzzy-neural networks. In this case, there are three possible types of networks. The first type is using real input and fuzzy weights, the second fuzzy inputs but real weights and the third uses both fuzzy input and fuzzy weights. Regardless of the type of hybrid system, a learning algorithm for the system must often be developed.
Chapter 3

The preprocessing perceptron

In the two previous chapters, the foundation was built for the preprocessing perceptron. The main ideas behind feed-forward neural networks and fuzzy logic connectives have been described. In this chapter, the preprocessing perceptron and the generalised preprocessing perceptron are described. The preparation of a data set to be used with the preprocessing perceptron is discussed and methods for handling missing data are also treated. A discussion regarding the interpretation of the outputs from a preprocessing perceptron is also included together with an overview of different methods to get good estimations of the error function of the method.

3.1 Adding a preprocessing layer

In traditional perceptrons, the inputs are usually linearly normalised to the \([0, 1]\)-interval, see also Section 3.2.2. Normalising in this way means that the inputs are presented to the network in exactly the same way disregarding the approximation task at hand. In the case of a medical decision support system, the inputs are symptoms and the output is the diagnosis. In this case, linearly normalising might not be an optimal approach since the value of the symptoms may have different interpretations depending on the specific diagnosis.

The preprocessing perceptron has an extra preprocessing layer in the network. This preprocessing layer shows the network how a specific symptom behaves for a certain disease, by use of a preprocessing function \(g\). Often the sigmoid function with parameters \(\alpha\) and \(\beta\) is used. This function is described in Figure 3.1 and by the following equation,

\[
g(\alpha, \beta)(x) = \frac{1}{1 + e^{-\beta(x-\alpha)}} \tag{3.1}
\]

where \(\alpha\) is the \(x\)-value at the inflexion point and \(\beta\) describes the slope at the inflexion point.\(^1\)

\(^1\)Please note that the sigmoid activation function given by Equation 2.2 is in fact Equation 3.1 with \(\alpha = 0\) and \(\beta = 1\)
3. The preprocessing perceptron

Figure 3.1: Example of the sigmoid function with $\alpha = 0.5$ and $\beta = 10$.

A typical example of the difference between interpretations of an input is the erythrocyte sedimentation rate (ESR). ESR measures how quickly red blood cells fall to the bottom of a test tube of unclotted blood. Rapidly descending cells (an elevated rate) indicate inflammation in the body. Patients with an ear inflammation or patients having tonsillitis typically have a large ESR value while patients with stomach flu usually have smaller ESR values. If linear normalisation is used, the value, e.g., six, is transformed into the $[0, 1]$-interval in the same way, say to the number 0.4. If a non-linear preprocessing function is used, the differences between the diseases can be shown by the function. If a sigmoid function is used, the limit between two classes, in this example a healthy and an ill patient, is symbolised by the inflexion point on the curve. The slope of the curves shows if the inflexion point is a sharp limit (steep slope) or if the limit between healthy and ill is more vague. In general, values near zero stand for a healthy patient and values near one stand for an ill patient with respect to this particular measurement. If the symptom’s value is high for a healthy patient and low for an ill, the curve is simply reversed.

The function for the preprocessing perceptron is given by

$$o = \sigma \left( \sum_{i=0}^{n} w_i g_i(\omega_i)(x_i) \right)$$

(3.2)

The index for the output is omitted since usually only one output is used in this kind of decision support system. As can be seen, each one of the inputs in the function in Equation 3.2 can have its own individual preprocessing function. An example of a preprocessing perceptron is shown in Figure 3.2.

The set of parameters used in the preprocessing function $g_i$, is denoted $\omega_i$, e.g., in the case of the sigmoid $\omega_i = \{\alpha_i, \beta_i\}$. The parameters are trained using the gradient descent algorithm as described in the section regarding the delta rule (on page 13 in Section 2.1.1).
3.1. Adding a preprocessing layer

PSfrag replacements

x
x
x
x

\[ o \]

\[ w_1 \]

\[ w_4 \]

\[ x_1 \]

\[ x_2 \]

\[ x_3 \]

\[ x_4 \]

preprocessing layer

input values

output layer

Figure 3.2: A preprocessing perceptron with four inputs, sigmoid preprocessing functions and one output.

3.1.1 Generalising the preprocessing perceptron

The generalised preprocessing perceptron is a hybrid network using real inputs, weights and outputs, and fuzzy logical connectives as functions in the net instead of the weighted sum. To explain the introduction of fuzzy logical connectives, the description of the preprocessing perceptron is used. The preprocessing layer is used to view relationships between symptoms and diagnosis to the network. This is done by applying a preprocessing function on the input, thereby obtaining \( g_i(x_i) \). Each one of the symptoms is considered by the clinicians when making a diagnosis. In other words, \( g_i(x_i) \rightarrow o \) with different degrees of certainty. The weight \( w_i \) in the network represents this uncertainty. The product \( w_i \cdot g_i(x_i) \) stands for the semantics of the implication \( g_i(x_i) \rightarrow o \). All indications given by the symptoms must form the ground for the diagnosis. This means that the implications have to be joined together to the conclusion \( o \) with some kind of disjunction function.

The generalised preprocessing perceptron (GPP) with one output is represented by the function \( \Phi : [0, 1]^M \rightarrow [0, 1] \) given by the equation

\[
    o = (\Phi^\wedge_{i=1}^M)(\varphi_i(\psi_i)(w_i, g_i(x_i))).
\]  

(3.3)

In Equation 3.3, \( g_i(x_i) \) is a preprocessing function as described in Section 3.1, \( \varphi_i(\psi_i)(w_i, g_i(x_i)) \) represents the semantics of the implication \( g_i(x_i) \rightarrow o \), and, finally, \( \Phi^\wedge_{i=1}^M(\varphi_i(\psi_i)(w_i, g_i(x_i))) \) corresponds to a fuzzy logical disjunction that calculates an increment of evidence. Typically, \( \Phi \) has a generator function, and is an assembly of \( t \)-conorms.

Note that the implication function always has at least one parameter, the weight \( w_i \), but there may be more than one, shown by the set \( \psi_i \). As implication function, e.g. Yager’s implication function defined by the formula

\[
    \varphi_i(w_i, g_i(x_i)) = w_{1}^{g_i(x_i)}{w_i},
\]

can be used. Other examples of disjunction functions are shown in Paper II.
The idea of a generalised preprocessing perceptron may be taken further into a generalised multi-layer perceptron where the generalised perceptrons are built together to form larger networks. This idea has, however, not been tested.

Learning algorithm

When training a generalised preprocessing perceptron there are more parameters than just the weights to update. The learning algorithm used is a version of the backpropagation algorithm described in Section 2.2.1. The general formula for updating a parameter \( \chi \) is \( \chi_{\text{new}} = \chi_{\text{old}} + \eta \Delta \chi \), where \( \Delta \chi \) is a measure of the error in the network, \( \Delta \chi = -\frac{\partial E}{\partial \chi} \), and \( \eta \) is the learning rate.

Since \( \Delta \chi \) usually can have values from the interval \((-\infty, \infty)\), the cases when the parameters to be updated are supposed to be in the \([0, 1]\)- or the \([0, \infty)\)-interval have to be handled in some way. This is often the case when fuzzy logic functions are used since the values correspond to truth-values. Special transformation rules have to be derived and a new kind of addition, \( \oplus \), be defined.

In the example of the sigmoid function (Equation 3.1), there are two parameters to update, \( \alpha \) and \( \beta \). Since normalised inputs are used, \( \alpha \) should be kept in the \([0, 1]\)-interval. Therefore \( \oplus_{[0,1]} \) must be defined. \( \beta \) on the other hand must be kept in the positive real domain and \( \oplus_{\mathbb{R}_+} \) need to be defined. When considering Yager’s function, the weight must be in the \([0, 1]\)-interval and \( \oplus_{[0,1]} \) is needed again.

An additive generator (see Section 2.3) is used to implement the addition functions with the help of two transformation functions, \( h \) and \( h^{-1} \). In the case of \( \oplus_{[0,1]} \), \( h_{[0,1]} \) must be a bijective and increasing function, \( h_{[0,1]} : (0, 1) \to \mathbb{R} \) and \( h_{[0,1]}(0.5) = 0 \).

The half-circle function will be used, with \( h_{[0,1]}(x) \) as the negation of the derivative and \( h_{[0,1]}^{-1}(x) \) as the inverse of the derivative of the half-circle function.

\[
h_{[0,1]}(x) = -\frac{x - 0.5}{\sqrt{0.25 - (x - 0.5)^2}} \quad \text{and} \quad h_{[0,1]}^{-1}(x) = 0.5 + \frac{0.5x}{\sqrt{1 + x^2}}
\]

\( \chi_{\text{new}} = \chi_{\text{old}} \oplus_{[0,1]} \eta \Delta \chi \) is then calculated in the following way:

1. Calculate \( \Delta \chi \) (\( \in \mathbb{R} \)).
2. Calculate \( a := \eta \Delta \chi \) (\( \in \mathbb{R} \)).
3. Apply \( h_{[0,1]} \), \( b := h_{[0,1]}(\chi_{\text{old}}) \) (\( \chi_{\text{old}} \in [0, 1] \) and \( b \in \mathbb{R} \)).
4. \( c := a + b \) (\( \in \mathbb{R} \)).
5. Apply \( h_{[0,1]}^{-1} \), \( \chi_{\text{new}} := h_{[0,1]}^{-1}(c) \) (\( \in [0, 1] \)).

The same procedure is used also for \( h_{\mathbb{R}_+} \), for example \( h_{\mathbb{R}_+} \) can be defined as follows:

\[
h_{\mathbb{R}_+}(x) = \begin{cases} \log(x) & \text{if } x > 0, \\ -\infty & \text{otherwise} \end{cases}
\]
and \( h_{\mathbb{R}^+}^{-1}(x) = \exp(x) \)

\[
\begin{align*}
  y &= 1 \\
  b &= 0.2959 \\
  a &= \Delta \chi = 1.1 \\
  c &= a + b
\end{align*}
\]

\( \chi_{\text{old}} = 0.4 \) \( \chi_{\text{new}} = 0.7559 \)

Figure 3.3: Using an additive generator to add two numbers in the \([0, 1]\)-interval.

The properties of an additive generator can also be described geometrically. Figure 3.3 shows the half-circle and the line \( y = 1 \) together with the values mentioned in this example. Suppose that we want to add \( \chi_{\text{old}} = 0.4 \in [0, 1] \) and \( \Delta \chi = 1.1 \in \mathbb{R} \) respectively. For simplicity choose \( \eta = 1 \) so that \( a \) in the second step above has the same value as \( \Delta \chi \). \( \chi_{\text{old}} \) lies in \([0, 1]\) and when \( h_{[0,1]} \) is applied the result \( b \) lies in \( \mathbb{R} \). The two values are added and the result \( c \) also lies in \( \mathbb{R} \). It is transformed back to the \([0, 1]\)-interval by applying \( h_{[0,1]}^{-1} \).

## 3.2 Inputs to the system

In Section 1.3.1, the extraction of data from journals into a data set is described. Different aspects of patient record systems and different types of inputs are mentioned. In this section, pre-processing, i.e. the steps from a raw data file to a file ready to be used by the preprocessing perceptron are described.

### 3.2.1 Pre-processing of the data

There are several steps that must be taken into account before a data file is ready to be used in order to build a decision support system. These steps are often referred to as pre-processing of the data. It is important to realise that these steps still have to be taken even though a preprocessing perceptron is going to be used.

Pre-processing of data is needed since real world data contains errors and outliers. Furthermore, when collecting a data set from scratch, decisions must be made concerning the number of inputs and what type of inputs to include in the set. The data is often incomplete, i.e. there is data missing in some of the
inputs. When using neural network models, the distribution of the data among the different classes is important. Normalisation of the input is also useful for neural network-like methods and this will be discussed in a later subsection.

**Input selection and detecting outliers**

Depending on the application area, the selection of inputs, sometimes referred to as feature selection, can be a trivial task or something that requires a substantial amount of time and effort. The skill and expertise of the person performing the selection are crucial. The goal is to find a set of inputs, or features, that are large enough to be a good description of the specific problem area without being too large thereby increasing the number of parameters in the model. Principal component analysis [Duda et al., 2001] is one example of a method often used to reduce the number of inputs. Another example is transformation of data, meaning combination of raw data inputs into a single input. This combination can for example be computing the difference or taking the ratio between inputs.

When the number of inputs is decided, the data must be inspected in order to find outliers and/or missing data. Outliers must be handled with care and often an inspection by a human expert is needed. The suspected outlier might not be an outlier but an error. Let us use the input "waist-measurement" as an example, if a value of 5 cm is detected among the inputs it is certainly an error. No human can have such a small waist unless it is a measurement of a foetus. Even if the suspected outlier is not an error, it still might not be an outlier but rather a very rare sample from the distribution. One statistical method for detecting outliers is e.g. [Grubbs, 1969].

**Missing data**

Missing data are usually divided into three categories [Little and Rubin, 1987]. The first category is **MCAR or Missing Completely at Random**. MCAR occurs when cases with complete data are indistinguishable from cases with incomplete data, i.e. the missingness is not related to the variables in the system. The input representing the BMI value\(^2\) is used as an example. When collecting a data set from a large data pool of medical journals created over time, the missingness of this value is more connected to the routines at different hospitals than to the other data in the collection of the diagnosis. Therefore the missingness is MCAR.

**MAR or Missing at Random** occurs when cases with missing data are different from complete data cases but the pattern of data missingness is traceable or predictable from other variables in the database rather than being due to the specific variable on which the data is missing. In other words, the cause of the missing data is due to external influence and not to the variable itself.

Missing data is **nonignorable** when the missingness is explainable - and only explainable - by the very variables on which the data are missing. If the BMI

\(^2\)BMI — Body Mass Index is calculated as the weight in kilograms divided by the square of a person’s height in meters and can be used e.g. to decide if a patient is obese or not.
3.2. Inputs to the system

value mentioned earlier would be missing for only obese persons, i.e. with a large BMI, the missing data is nonignorable.

Usually, the assumption is that the missing data is MAR. Since the reason for missingness often is unknown for users of the data set, they can not be sure that the MAR assumption holds. The problem is that there is no way to test the MAR hypothesis since that would require knowledge of the missing data themselves. However, the use of MAR methods is common and that usage is recommended [Schafer and Olsen, 1998] until more available methods for non-ignorable data are present. Furthermore, they state that MAR methods such as multiple imputation will perform better than ad hoc methods such as list-wise deletions or imputation of means.

There are several different methods to handle missing data and a few of them will be mentioned here. The methods are presented in an order where the first methods are naive ways to handle missing data while the last one, multiple imputation, is the state of the art. It is important to note that this is said in the light of prediction. Most literature on missing data deals with estimation rather than prediction. In estimation it might be possible to throw away cases with missing data while in prediction, these particular cases might be very important. See e.g. [Sarle, 1998] for a discussion of prediction with missing values.

One method to handle missing data is to simply ignore it. Usually the missing data is marked with a number such as -1 or 0. This number can be used directly in the file. As mentioned in [Kallin Westin, 2004a], it is not unusual that people, not familiar with the background regarding the collection of the data set, do not realise that there is missing data present in the set.

Another common method is to remove the problem, i.e. the missing data. This might work if there is a very small amount of missing data and if the missingness itself does not have meaning, i.e. the data is MCAR. The method is easy and the data used is the data given and nothing else. However, if the data set is not MCAR, the resulting data set is often biased and it is always a bad idea to throw away information when building a decision support system. Furthermore, the problem resides, since nothing is done to handle missing data in the future use of the decision support system either.

Removing missing data can be done in several ways. If the proportion of missing data is not too large, one might consider to remove all cases containing missing data from the set. If a specific input has a large proportion of missing data one might consider to remove the input from the data set. The problem with that is that in some cases this input is a very clear sign. For example, in the case of myocardial infarction, four blood samples are collected at four different times. If all four samples are collected it means that the patient has survived 24 hours after the possible infarction. On the other hand, if data are missing this is probably due to the fact that the patient has deceased, and thereby the missing data might be a very clear sign of myocardial infarction.

Obviously, one can combine the two approaches above and remove both inputs and cases from the data set.

Another way to handle missing data is to try to replace the missing values with a plausible value. Two major methods for imputations are mentioned in
3. The preprocessing perceptron

this section but there are more approaches available, e.g. hot deck imputation and regression imputation. See e.g. [Little and Rubin, 1987] for a description of these methods and more examples.

When mean imputation is used, missing values are replaced by the means of the set of observed data. This method is known to lead to biased variances [Little and Rubin, 1987]. The bias increases as the number of missing data rises. A variant of mean imputation is also possible. Two mean values are calculated, one for the healthy cases and one for the ill cases. These two means are then used to fill in the missing values in the training set. If the value is missing for a healthy case, the healthy mean of that input is imputed and the ill mean is imputed otherwise. If a test set is used, it is imputed with the total mean of the training set since it is not possible to use information in the test set to see if a specific case is healthy or ill.

Multiple imputation [Rubin, 1987, Schafer, 1997] is a procedure where each missing value is replaced by not only one but \( m \) different values. The values are drawn from an implicit model of the posterior predictive distribution of the missing values. The \( m \) imputations form \( m \) different data sets. Each of these data sets are analysed as if there were no missing data and finally the results from these experiments are combined into a single result. Multiple imputation usually requires only a small \( m \), i.e, 3 to 5 to get good results. This is due to the fact that uncertainty due to missing data is incorporated in the model. In order to generate imputations for the missing values, one must impose a probability model on the complete data (observed and missing values). Details of these models are given by [Schafer, 1997]. The program NORM can be used to create the imputed data sets. NORM can be downloaded from J. L. Schafer’s web page at the address http://www.stat.psu.edu/~jls/misoftwa.html.

3.2.2 Preparing a data set for training

When a data set is collected and errors and outliers are removed, the choice of analysis method must be made. There exist many different methods, some are discussed in Section 1.3.2. If all inputs are numerical, or can be transformed to a numerical form in an easy way, a neural network or a preprocessing perceptron might be useful. Neural networks and the preprocessing perceptron are similar in the way that they have a training phase and that the data usually are normalised before training. In order to be able to evaluate the results of these methods, the data set collected is usually divided into a training and a test set.

The organisation of the data set

The data in the data file form a matrix. Usually, each row of the matrix corresponds to one case. The inputs and targets form columns in the matrix and the targets are often placed furthest to the right in the matrix. In this thesis, almost every data set has only one target, the diagnosis. The order of the cases in the data sets is not fixed. In some data files, the cases are ordered by the output
value and in others their order are completely random. In Section 3.3, the order of the cases and its influence on the result of the training will be discussed.

Normalising the data

The perhaps most common way to normalise the data is to use min-max normalisation to the \([0, 1]\)-interval as described by the formula

\[
x_i = \frac{x_i^\ast - m_i}{M_i - m_i},
\]

where \(m_i\) is the minimum value among the inputs \(x_i^\ast\) and \(M_i\) is the maximum value. This is the recommended way to normalise inputs, if the sigmoid is used as a preprocessing perceptron. If \(\tanh\) is used, it might be a better choice to normalise to the \([-1, 1]\)-interval.

Another way to normalise the data is to let the normalised data have zero mean and unit standard variation. This is accomplished by the formula, assuming there are \(n\) inputs:

\[
x_i = \frac{x_i^\ast - \bar{x}}{s},
\]

where

\[
\bar{x} = \frac{\sum_i x_i^\ast}{n}
\]

and

\[
s = \sqrt{\frac{\sum_i (x_i^\ast - \bar{x})^2}{n - 1}}.
\]

If the data set is supposed to be divided into a training, test, and, possibly a validation set, the training set must be normalised using information from the training set only and not the whole set. Otherwise, information from the test set is used while creating the training set and that should be avoided. When the validation and test files are normalised, only information from the training set is used. This might lead to values ending up being outside the intended interval. In these cases values outside the interval are truncated into it, e.g. values below zero are set to zero and values above one are set to one.

Division of the data set

In order to be able to evaluate the resulting decision support system, the data used for training a network must be divided into at least two parts. One part is used to train the preprocessing perceptron, the \textit{training set}, and the other is used for testing, the \textit{test set}. The way the division into different data sets is performed is sometimes a result of the estimation method that will be used. In Section 3.4.1, different methods for estimating the resulting decision support system is described.

If two sets are used, the network is trained using the training data and no information from the test set is used during the training. The test set is used to measure the network performance after the training. The set of training data
must not be too small or overfitting may occur. Overfitting means that the network has been trained on a too small set of training data or for too long time such that it has learned all details in the training data but lost the ability to generalise and therefore will perform badly on the test set. Overfitting may occur every time a network is trained. However, if the set of training data is too small, the risk of overfitting is higher.

Instead of using two sets, the training data can be divided further into one training set and one validation set. The validation set is used during training in order to evaluate the quality of the network. Such an evaluation helps to reduce the problem with overfitting. Early stopping occurs when the training of a neural net stops, not when the error of the training set is minimal but when the validation set has minimum error. The validation set can never be used as a test set since it has been used during training.

In practice, it seems as if the preprocessing perceptron is rather insensitive to overfitting and experiments with and without a validation set show small differences in generalisation performance. This suggests that there often is no need for a validation set when training the preprocessing perceptron. It is better to use a larger training set instead.

The division into test and training set is crucial. If there is a great variation in the number of cases corresponding to each class, there might be a slower convergence and difficulties in learning the classes with a low number of cases [Lawrence et al., 1998]. The ideal training set contains equal proportions of the different output classes and contains cases that are easy to separate as well as cases very similar to each other belonging to different classes. Similarly, in order for the validation set to work as expected and the test set to give a good estimation of the error, the creation of these data sets is important too.

### 3.3 Training the preprocessing perceptron

As has been described earlier in this chapter, the preprocessing perceptron is trained in a backpropagation-like manner. Patterns are presented to the network and the resulting output is compared to the target value. The error is then sent backwards in the net updating the parameters according to the generalised delta rule. There are many different ways to describe how the training is done. One characterisation is if incremental or batch-training is used.

#### 3.3.1 Batch vs. incremental learning

In batch-learning, all cases in the training file are presented to the network and an average error is computed and propagated backwards. When incremental training is used, one case is presented to the network at a time and the parameters in the network are updated before the next case is presented to the network. There are also hybrids of these two methods where two or more cases are presented to the network each time.
When incremental training is used, the order of the cases in the training set might influence the results. If the training data set is sorted such that all cases belonging to one case are placed first in the file, followed by the other class(es) there is a risk of getting a biased training. This can be avoided by having the cases in random order in the file or by choosing a random case in each turn.

In the program \texttt{trainGPP}, described in Paper V, it is possible for the user to choose any number of cases from one to the whole data set to be run each turn of the algorithm. It is also possible to decide if the cases should be drawn from the data set in random or in consecutive order.

### 3.3.2 Learning rates and training

In traditional algorithms for network training, all parameters in the network are updated in each run of the algorithm. It is common to have the same learning rate for all parameters (weights and thresholds). However, usually the learning rates change during the course of the training.

When training a preprocessing perceptron, the learning rates are different for different types of parameters. If a standard preprocessing perceptron is considered, with sigmoids in the preprocessing layer and the ordinary weighted sum and activation function as implication and disjunction functions, there are three types of learning rates. The first one is for the weights and the threshold, the other for the slope ($\beta$) in the sigmoid and the third is for the inflexion point ($\alpha$). This is discussed further in Paper IV where the following rule of thumb is given: Let $\eta_w = \frac{1}{10} \cdot \eta_a = \frac{1}{100} \cdot \eta_\beta$, with $\eta_\beta = 1$ or less.

It has been shown in experiments that it is also might be useful to regularise training in the sense that parameter classes are trained separately. Heuristically, it is suitable to start with the weight values, with $\alpha$-values, initialised to the centre of respective data intervals. Weights can be retrained each time $\alpha$-values have gone through a training sequence. Even if this way of training have proven to be successful in practice it must be noted that this is a procedure that might lead to slower convergence and local minimums.

### 3.4 Outputs from the system

The output from the preprocessing perceptron lies in the interval $[0, 1]$. If it is used in a medical setting, a low value corresponds to a negative diagnosis and a high value to a positive diagnosis. There are several ways to estimate the performance of a decision support system and this section will describe a few of them.

The output value can be used as it is and be compared to a target value in order to estimate the magnitude of the total error value. It is also possible to introduce a cut-off value and decide that all output values below it will be treated as zeros and all values above as ones. When this is done, the total accuracy, the sensitivity, and the specificity of the decision support system can
be calculated. If several cut-off values are chosen, a ROC and/or a QROC-curve can be plotted. At the end of this section, there will be a discussion of the complexity of a decision support system compared to its capacity to generalise and adapt to new data.

3.4.1 Estimation and comparing decision support systems

When training a decision support system there are three types of errors present.

- The unknown true error will occur when the decision support system is used in practice.

- The error obtained on the training data set. The goal is to have this error as small as possible without destroying the generalisation effects of the network. The training error is often optimistic, it is sometimes possible to reach a value of zero on this error.

- The error obtained when the trained structure is run on the test data set. This error is an estimation of the unknown true error and the estimation is more accurate if the test set is larger.

The optimal scenario would be if all possible cases are used for training. In real applications this is almost never possible. Instead, there is a subset of all possible cases collected in a data set. This data set must then be divided and used for training and testing the accuracy of the decision support system.

Cross validation

One popular group of methods is cross validation described in many textbooks on neural networks and pattern classification, see e.g. [Duda et al., 2001, Ripley, 1996]. The simplest form of cross validation, often called holdout procedure, has already been described. The data set is divided into two sets, the training and the test set. The decision support system is built on the training set and then tested using the test set. The disadvantage of this method is that the results depend on distribution of the cases between the two sets. If the data set is very large, this is the easiest method. It is also often possible to use a stratified holdout where the sampling of the test and training set is done in such a way that each class is represented in both sets according to their proportions in the total set [Ripley, 1996].

A way to handle the disadvantages mentioned and to cope with small data sets is the k-fold cross validation. The data set is divided into k different subsets. Then k − 1 sets form a training set and the remaining set forms the test set. This method is repeated k times, each time with a new test set. Each case in the original data set is trained k − 1 times and tested once. Finally the results of these k different trainings are combined to a single estimation. The variance of the error estimation decreases as k increases but the computational effort does also increase. Usually the value of k lies somewhere between three and ten.
3.4. Outputs from the system

Going to the limit, i.e. using \( k = N \), the number of cases in the data set, results with the \textit{leave-one-out cross validation} or \textit{jackknife estimation}. The training is performed \( N \) times, each time leaving out one case, training all the rest and then testing the single case. This means a very high computational cost since \( N \) networks have to be trained.

\textbf{Bootstrap}

Another famous method to reuse data and still be able to estimate bias and variance of the error is the \textit{bootstrap} method [Duda et al., 2001, Ripley, 1996]. These techniques uses sampling with replacement. Assume, as before, that the original data set consists of \( N \) cases. Form the training set by drawing \( N \) samples with replacement from the original data set. When this is done, let the cases in the original data set that never were selected form the test set. This implies that the size of the test set can be different from time to time. Do this forming of training and test sets \( k \) times.

Due to the replacement of cases, the variance of cases in each one of the \( k \) sets increases as compared to cross-validation. Sampling with replacement preserves the a priori probabilities of the different classes in the data set.

\textbf{Sensitivity, specificity and accuracy}

The two methods mentioned in the most recent subsections estimate the error of a decision support system. Sometimes, especially in medical decision support systems, a low error rate is not the only goal. A short example to illustrate the idea: Suppose there is a need for a decision support system for diagnosing a rare disease that occurs only among one percent of the population. An easy way to build a decision support system is to state "patient healthy" no matter the input. This system will have an error rate of only one percent but will be useless as a decision support system. In this section, other measures of the results of a decision support system will be described. For a more thorough description see e.g. Paper III and [Kraemer, 1992].

Table 3.1 illuminates the possible outcomes of comparing an output from a decision support system with a target value (i.e a previously made diagnosis). If both target and output are positive, it is called a \textit{true positive} (TP). The probability of a TP to occur is estimated by counting the true positives in the output set and divide by the size of the output set. If the target is positive and the output is negative it is called a \textit{false negative} (FN). \textit{False positive} (FP) and \textit{true negative} (TN) are defined similarly.

Let \( p_i \) be the probability that case \( i \) will get a positive target (i.e., the patient corresponding to the case is diagnosed to have the given disease) and \( q_i \) be case \( i \)'s probability of a positive output from the decision support system. The \textit{prevalence}, \( P \), of the positive diagnosis in the population is estimated by \( P = \text{mean}(p_i) \). The \textit{level} of the test, \( Q \), is \( Q = \text{mean}(q_i) \).

\textit{Sensitivity}, SE, is the probability of having a positive output among the
Table 3.1: Possible outcomes when comparing the output from a decision support system with a previously made diagnosis.

<table>
<thead>
<tr>
<th>Target</th>
<th>Output</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>Negative</td>
<td>FP</td>
<td>TN</td>
</tr>
<tr>
<td>Q</td>
<td>Q'</td>
<td>1</td>
</tr>
</tbody>
</table>

cases who have a positive target.

\[
SE = \frac{TP}{(TP + FN)} = \frac{TP}{P}.
\]

Specificity, \( SP \), is the probability of having a negative output among the cases who have a negative target.

\[
SP = \frac{TN}{(FP + TN)} = \frac{TN}{P'}.
\]

Since sensitivity and specificity both are related to the level \( Q \) of the test, their ideal value of 100% might not be achievable. Two quality measurements are derived in [Kraemer, 1992] and their values lies in the interval \([0,1]\).

\[
\kappa(1,0) = \frac{(SE - Q)/Q'}{Q'}
\]
\[
\kappa(0,0) = \frac{(SP - Q')/Q}{Q}
\]

Furthermore, these quality measurements are weighted together and efficiency is considered in a third measurement, the optimal efficiency as given by [Kraemer, 1992].

\[
\kappa(0.5,0) = \frac{PQ'\kappa(1,0) + P'Q\kappa(0,0)}{PQ' + P'Q}
\]

A decision support system gives output in the interval \([0,1]\) where 0 denotes a negative and 1 denotes a positive diagnosis. By introducing a cut-off somewhere in the interval, the output is binarised and compared to the true diagnosis given by a clinician. Each cut-off corresponds to a table as shown in 3.1 and results in sensitivity-specificity pair. This pair is a point on a **ROC curve**. The ROC curve has the sensitivity plotted vertically and the reversed scale of the specificity on the horizontal axis. The scale of the horizontal axis is also called the **false positive rate**. The sensitivity and specificity, and therefore the performance of the system, vary with the cut-off. If several tests are performed with the same samples, e.g. several decision support systems built, ROC curves can be used to compare their performance. The correspondence between cut-off and performance can be shown by an example.
Assume that the result of the decision support system forms two normal distributions, one for the healthy patients and one for the ill patients. The threshold is placed at different positions to divide the distributions. The sensitivity and specificity are calculated for each position and the resulting points are plotted as a ROC curve. The curve gives a picture of the performance of the system. In Figure 3.4, three examples with ten different decision thresholds are shown. The overlap between the distributions is largest in the first example and smallest in the third. When plotting the sensitivity versus 1 - specificity values for these cut-offs we get the curves shown in Figure 3.4. The point marked by an arrow corresponds to the sensitivity and specificity at the cut-off value 0.222 in the distributions with the largest overlap.

Figure 3.4: Three pairs of distributions with large, medium, and small overlap respectively.

Figure 3.5: The corresponding ROC-curves for three pairs of distributions shown in Figure 3.4.

The complete curve, or more correctly the area beneath it, gives an overview of the overall performance of a decision support system. When comparing ROC-curves of different decision support systems, good curves lie closer to the top left corner and the worst case is a diagonal line (shown as a dashed line in Fig-
3. The preprocessing perceptron

Figure 3.4). There are methods for estimating confidence intervals for ROC curves as well. For more information see e.g. [Reiser and Faraggi, 1997, Schäfer, 1994, Zou et al., 1997].

A QROC curve has the quality measurement $\kappa(0,0)$ on the horizontal axis and $\kappa(1,0)$ on the vertical. In a QROC curve, one can immediately see which test that has the optimal sensitivity. It is the marker at the highest position. The rightmost marker is the one with the optimal specificity. All legitimate tests have quality values that are positive. The point $(0,0)$ corresponds to a random test and the point $(1,1)$ corresponds to an ideal test. Any point inside the QROC (i.e. not on one of the axes) will be statistically significant, if the sample size is large enough, for details see [Kraemer, 1992].

3.4.2 Complexity vs. approximation strength

In order to understand the differences between different methods, the surfaces of example networks are shown in Figure 3.6. Furthest to the left, a perceptron with a sigmoid activation function but no preprocessing layer is shown. The plot in the middle is a preprocessing perceptron with sigmoids as preprocessing and activation functions. The plot to the right is a multi-layer perceptron with two hidden layers and a large total number of neurons.

We have a trade-off between a simple model which is easy to understand and a more complex model. If a model is too simple, it will not be able to do the acquired approximation. If a model is too complex it will lose some of its ability to generalise and moreover it will need much larger data sets in order to learn the acquired approximation. The goal is to find a large enough structure to solve the problem but small enough for the structure to be able to generalise beyond the training data. This trade-off is also described as the structural risk minimisation principle [Vapnik, 1998].

![Figure 3.6: Comparison of example networks for the perceptron, the preprocessing perceptron and the multi-layer perceptron respectively.](image)
Chapter 4

Summary of papers

4.1 Aim of thesis

The aim of the thesis is to explore the possibilities to use the preprocessing perceptron as a computational method to build decision support systems. Often a thesis has some kind of historic line or "red thread" telling a story of the author’s way from a starting point to a stopping point which is the final dissertation. The work with this thesis could be described as a snowflake\(^1\) as done in Figure 4.1.

\(^1\)The web page http://www.snowflakebentley.com/ tells the story of Wilson "Snowflake" Bentley who spent his whole life taking photos of snowflakes. The goal was to find two identical snowflakes - he took over 5000 pictures and found no duplicates.

Figure 4.1: The work behind this thesis shown as a snowflake. The preprocessing perceptron is placed in the middle and different aspects of it around it.
4. Summary of papers

Instead of being done in a linear fashion, the work has evolved in many different directions, some routes leading to dead ends and others to results. However, all work has evolved around the preprocessing perceptron and research questions about it. Even though a snowflake is known to always have six branches, this research has more than six areas to discover and this snowflake will probably grow into a bigger structure in the future.

4.2 Results and contributions

For each of the five questions attached to a branch of the snowflake in Figure 4.1 a paper was written in order to find and document an answer. The sixth branch together with the smaller parts in the figure represents papers connected to, but not included in, this thesis as well as work done in other areas during the work leading up to this thesis. In this section, each paper will be shortly introduced together with a summary of the results and contributions.

4.2.1 Does the preprocessing perceptron work?

The aim of Paper I, [Kallin et al., 1998], is to explore if adding a preprocessing layer to a perceptron creates a method whose performance is comparable to existing methods. In the paper, the preprocessing perceptron is compared to a statistical formula (multiple Gaussian formula) and a multi-layer perceptron. The case study used is the Down’s syndrome with three inputs and one output.

The results of Paper I are that the performance of the preprocessing perceptron is indeed comparable to the multi-layer perceptron and is significantly better than the multiple Gaussian formula. The fact that the difference is significant is not explicitly shown in this paper. However it is shown in the paper [Kallin Westin, 2001a]. Another result from Paper I is that the parameters in the preprocessing network all had a logical meaning that could be explained to a domain expert. An example of this is the cut-off values in the preprocessing perceptron, they are similar to the medical decision limits being used within the medical domain.

The main contribution of Paper I is that it has been shown that it is possible to use a simple model such as the preprocessing perceptron, where each parameter can be explained to the domain expert, and still get a performance comparable to a multi-layer perceptron.

4.2.2 Can the preprocessing perceptron be generalised?

The aim of Paper II, [Anttila et al., 1996], is to explore if it is possible to replace the weighted sum in a preprocessing perceptron with fuzzy logic connectives and still have comparable results. The multiplication was replaced by Yager’s implication and the activation and summation with a Schweizer-Sklar t-norm. The case study used is the Polycystic Ovary Syndrome with nine inputs and one output and the results of the generalised preprocessing perceptron is compared
4.2. Results and contributions

to the preprocessing perceptron. The paper also presents the production line for decision support systems.

Even though the data set used in this article is small (64 cases) it serves its purpose as data set when comparing the preprocessing perceptron with its generalised counterpart. The correctness rates are very high for the methods (>95% in both cases). This is of course partly due to the small data set but the data set is also easier to classify than the Down data set used in the first paper. Another result from Paper II is that when the preprocessing layer is compared for the two methods, which is built of sigmoid functions in both cases, the cut-off values proves to be similar. In fact, they are almost identical for the inputs with the highest weights.

The contribution of Paper II is the presentation of a generalisation of the preprocessing perceptron and showing that their performances are comparable. Another contribution is that it seems that the preprocessing layer is, to some extent, independent of the network structure. See also [Kallin and Eklund, 1998], where different preprocessing functions are discussed and also fuzzy logic connectives that can be used instead of the weighted sum.

4.2.3 How can the results be analysed?

The aim of Paper III, [Kallin Westin, 2001b], is to give an overview of the Receiver Operating Characteristic (ROC) analysis within medicine and computer science. The theory behind ROC analysis comes from statistical decision theory and was originally used during World War II. During the late 60’s the ideas were spread into the field of medicine and in recent years the tool has also been used within computer science as a tool to evaluate discrimination effects among different types of methods. The paper presents the theory behind the ROC and QROC curves and shows how to perform analysis of the performance of decision support systems. Several case studies are used as examples to illustrate the use of the analysis.

The contribution of Paper III is the further introduction of ROC and QROC analysis into the field of computer science. A comparison of the similarities and differences in ROC terminology between medicine and computer science is presented. The notion of QROC analysis and its advantages over ROC analysis is believed to be almost unknown within computer science and this paper is a way to introduce the analysis method. The paper is an expansion and a refinement of the material found in [Kallin Westin, 2001a].

4.2.4 Can the preprocessing layer be interpreted?

Paper IV, [Eklund and Kallin Westin, 2003], includes a thorough description of the preprocessing perceptron and important comparative results. The paper also investigates the role of the preprocessing layer. It has been seen earlier that some of the $\sigma$s in the sigmoid preprocessing function are similar to decision limits within medicine. In this paper, it is argued that the preprocessing layer produces and represents multivariate decision limits.
4. Summary of papers

The work in this paper is partly based on [Eklund and Kallin Westin, 2001] where the differences between a univariate and multivariate approach were first mentioned. In [Ellenius et al., 1997], the preprocessing layer was hand-tuned before training in such a way that the $\alpha$s in the sigmoid were placed where 95% on each individual symptom was reached. During training the preprocessing parameters were not trained. This approach, where the $\alpha$s are seen as univariate decision limits, were compared in the paper to a multivariate approach where all parameters of the preprocessing perceptron were trained.

The results and contributions of Paper IV, are that the training of all parameters improves classification rates and in several cases there are significant differences. Furthermore, when training the $\alpha$-values it is shown that the final values were not the same as the univariate $\alpha$s chosen for the fixed preprocessing layer. This implies that the $\alpha$s in the preprocessing layer represents multivariate decision limits. Since the paper is a survey of the preprocessing perceptron, a comparison is included of the results of a preprocessing perceptrons and a multi-layer perceptron for three different data sets. The preprocessing perceptron has the same or better results than the multi-layer perceptron on all three data sets.

4.2.5 How to create and train a preprocessing perceptron?

The aim of Paper V, [Kallin Westin, 2004b], is to provide a user’s manual for the program trainGPP that is developed by the author in Matlab. The paper is built in a tutorial-like manner and describes how to use the preprocessing perceptron from the moment when a data file is provided until the moment when an existing decision support system is built.

4.3 Future work

There are many new paths to take during the journey of making a map over the land of the preprocessing perceptron. One thing worth investigating is closeness between preprocessing perceptrons. If several different preprocessing perceptrons are trained on the same data set, how close are they to each other? What should decide that, the value of the error function or the values of the parameters or perhaps something else?

It would also be interesting to investigate different error functions and how they influence the training algorithm. Another idea worth exploring is the use of the preprocessing perceptron in an algorithm for imputing missing data.

This thesis does not include any formal proof of the extent of the discriminance effect of the preprocessing perceptron. It is evident from experiments done that the structure is capable of more than just linear discrimination, but how much more? Intuitively, the preprocessing perceptron seems to work well when approximating monotonic functions.

The original idea in this thesis, the production line [Eklund et al., 1999], would be really interesting to realise. A prototype was made for the web already in 1995-96.
Bibliography


Bibliography


Bibliography


