

Prof. Dr. Andreas Mielke
Andreas-Hofer-Weg 47
D-69121 Heidelberg
Germany

www.andreas-mielke.de
e-mail info@andreas-mielke.de

Neuronal Networks

Andreas Mielke

March 4, 2011

Neural networks are a useful tool for simulations in various areas like medicine or economy. For a successful application one has to consider several points. In this document we answer the following questions: How does one plan a project in which neural networks are used? How does one construct a neural network? How can errors be analyzed, and how can one minimize them? Which techniques can be used to derive rules using neural networks? How can one optimize and control a neural network? At the end, we present an example of an application of neural networks in medicine.

Contents

1	Possibilities and limitations of neural networks	4
2	Planning	6
2.1	Introduction	6
2.2	The problem	6
2.3	The aim	7
2.4	Resources	7
2.5	Project personnel	8
2.6	Costs	8
2.7	Documentation	9
3	Data	9
3.1	Introduction	9
3.2	Coding	10
3.3	Quality and Quantity	10
4	The network	10
4.1	Introduction	11
4.2	Design of a multi-layer perceptron	11
4.3	Hidden layers	11
4.4	Training algorithms	12
5	Errors	12
5.1	Basic remarks	13
5.2	Sources of errors	13
5.3	Errors in classifications	13
5.4	Errors of continuous output data	14
5.5	Estimating errors of continuous data	14
5.6	Interpretation of errors	14
5.7	Special cases	15
6	Rules	15
6.1	Introduction	15
6.2	Method for rule extraction	16
6.3	Classification of rules	16
7	Optimization and control	17
7.1	Introduction	17
7.2	Refinement of a neural network	18
7.3	Optimization and control of the software	18
7.4	Optimization of the Hardware	18
8	Application	19
8.1	Description of the project	19
8.2	Preparatory work	19
8.3	Implementing a neural network	19

8.4 Outlook 20

9 Links to other sites providing information on neural networks 20

1 Possibilities and limitations of neural networks

Neural networks, or better artificial neural networks are software products or programs. In artificial neural networks one tries to imitate structures of the nervous system, but in a highly abstract way. In some aspects, a neural network is a program that processes data like (a part of) the nervous system. They have been developed as a route towards a better understanding of the brain. Later people used neural networks in various fields for cognitive tasks, such as learning and optimization. In the ideal case, the neural network learns from examples like the brain. In a somewhat anthropomorphic manner of speaking one can say: If one shows the neural network a sufficiently large set of examples, it learns to generalize these examples. This process of learning from examples is called training.

Neuronal networks can be used in various fields to model a given problem. Typical applications can be found in medical diagnostics, in the analysis of economical data, in the control of production processes, in robotics, etc. In all these applications, a neural network is used to obtain some information out of given data. Often one uses neural networks in situations where other methods are not available. Among other methods, one can use more general mathematical models. A text on general mathematical models <http://www.andreas-mielke.de/ms-en.html> discusses various aspects of such models, including their simulation.

In principle, neural networks can compute any function, i.e. they can do everything a normal computer can do. Neural networks are especially useful for classification problems and for function approximation problems which are tolerant of some imprecision, which have lots of training data available, but to which hard and fast rules (such as laws of nature) cannot easily be applied.

Before one starts a simulation using a neural network, one should try to find out, whether alternative methods exist and if the given task can be treated by a neural network. It is for instance not a good idea to try to predict the drawing of a lottery from the last five drawings using neural networks, because one cannot expect any correlations between the different drawings. A more interesting example would be a weather-forecast by a neural network using the weather conditions of the last five days. But in this case, better models are available and should be used instead of a neural network. In many situations one expects correlations between data but one does not know any rules or laws that could be used for a prediction. Such a situation is ideal for the application of a neural network. A typical example from medical diagnostics is a Risk estimation for prostate cancer <http://www.charite.de/ch/uro/schwerpunkte/pclass.html>: Several serum-based data are known to be important for the prediction of prostate cancer, but the detailed interplay between these data is unclear. The program estimates the risk for prostate cancer using a neural network. The network has been trained on a large set of samples and the prediction is by now a useful tool for a physician to find out whether a patient has prostate cancer. We present a more complex example in section 8 (Application). For other examples you may take a look at our collection of links (see section 9 (Links)).

An artificial neural network consists of a collection of nodes and lines between the nodes. Each node represents a neuron and the lines represent the connections between the different neurons. To each node one associates a state variable that describes the state of the neuron. Often the variable associated to a node takes two values, it is binary valued. To each line one associates a variable that describes the strength of the connection between the two neurons. The figure shows a sketch of a neural network.

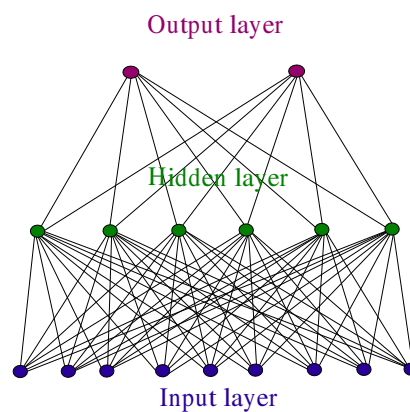


Figure 1: An artificial neural network

To fully describe an artificial neural network, further parameters are needed (bias, transfer functions), which are less important for the basic understanding. In most cases one uses a neural network that consists of various layers, an input layer, several hidden layers, and an output layer. A typical task for such a network is a classification: The input data feed into the input layer are classified by the discrete output. The medical example mentioned above is of that kind. The output of the network is the probability of the patient to have a certain disease.

Neural networks are trained with complete data sets. A complete data set consists of input and output data. One varies the parameter of the network, i.e. the couplings between the nodes. Typically one starts with a random configuration and calculates output data from the given input data. One compares the calculated output data with the output data of the complete data set and tries to minimize the error of the output data varying the couplings between the nodes. When the network has learned the complete data sets, one takes an independent collection of complete data sets to test the generalization capability of the network. Both collections of complete data sets must be large enough and correctly distributed within the range of possible data. Below, in the section 3 this will be made more precise. For the above medical example, this means that one needs the serum-based data from many patients, and for each patient one has to know whether or not he has the disease. After having trained the network with these data, the network is able to generalize from these examples. The output of the network may be a great help for a physician to find out whether or not a patient has the disease.

There are several steps which are important if one uses a neural network:

1. One has to check that no other, more efficient method is available and that the problem can be treated using an artificial neural network (feasibility).
2. One has to plan the project. This includes resources, personnel, costs, and documentation.
3. The next step is to setup standards for data collection and coding. After data are collected, one has to assure the quality of the data.
4. Now the network can be designed. Usually one needs several training cycles to obtain a optimal structure of the network.

5. After the network has been trained and designed, one has obtain precise estimates for the errors.
6. One can use the trained network to extract rules. Rules are important for optimization and control of the network.
7. The network can now be used in the production process. During the production process, one should control the network. Optimization of the network may be possible as well.

2 Planning

This section describes several aspects that should be taken into account when one starts a project in which neural networks are used.

2.1 Introduction

In many cases a neural network is used to calculate some output data from given input data. The connection between the output data and the input data must be unique: The output data are determined by a single valued function of the input data. The neural network has to learn this function. This is done by training the network with a collection of complete sets of input and output data. Further one needs a second collection of complete data sets to test the network. Since one has only a finite collection of complete data sets, the desired function is learned approximatively. On the other hand the function is not uniquely defined, so that the approximative aspect is not a real restriction. Nevertheless, the main problem is to get useful estimates on the errors of the output produced by the neural network. Often, this aspect is not taken into account. We will come back the this point in section 5 (Errors).

Since in typical applications of neural networks the function is not known, the procedure has to be planed accurately. In the following we discuss the main questions one has to take into account:

- Which problem has to be solved?
- What do we want to achieve?
- Which resources do we have?
- Who will contribute to the project?

2.2 The problem

Often one has several possibilities to reach the solution of a given problem. If some rules or principles are known, it may be inefficient to use a neural network. Probably a different method exists, which solves the problem more efficiently. In most cases it is not possible to implement rules in a neural network. On the other hand, a neural network can be used to obtain rules. We will come back to this point in section 6 (Rules). Further, rules can be used to check the output of the neural network.

2.3 The aim

When one uses neural networks it is very important to define what one wants to achieve. Without a fixed aim the simulation with a neural network is often useless. When one fixes the goal it must be clear that a neural network cannot generate information. All the information must be contained (in a hidden form) in the input data. This is obvious and holds for any simulation, but in the case of neural networks it is often overseen. When one knows what the network should do, a feasibility study can show whether one can reach the goal using a neural network. In many cases, some, probably incomplete preliminary data are available. Such data can be used to test whether a neural network can produce some useful output in simplified situations.

2.4 Resources

It is not very difficult to program a neural network. Neural networks can be run on any modern computer. The demand on the *hardware* is determined by the complexity of the problem. Some restrictions may occur when one wants to use special *software* products. It is not necessary to use special software, but it may be useful if the same software has been used already or if one knows that this special software has advantages for the given problem. This might be the case if special devices are used to produce e.g. the input data. An important point is the stability of the installation: Since typical projects run over several years, one has to be sure that hardware and software are well suited for that period or that the software can easily be ported to a new hardware after some time.

Concerning the hardware, one has to distinguish between the development platform and the production platform. The production platform often depends on existing hardware and several restrictions have to be taken into account. Such restrictions must be taken into account during the development of the neural network, even if the development platform does not have the same restrictions.

Programming a neural network can be done in any common programming language like C, C++, Java. It is clear that one needs a compiler for that language. Another possibility is to use a special program called *neural network simulator*. Many such products are available for many different platforms. Both methods, the direct programming of a network and the use of a neural network simulator have advantages. A simulator has usually a graphical user interface and the user does not need to know a programming language. Depending on the product it may be quite easy to work with such a program. A large number of different network architectures, learning strategies etc. are available. As a consequence the development of a neural network is easy and flexible. On the other hand, such programs usually need a large amount of memory and they are slower than a directly programmed neural network. Furthermore, in common languages large libraries for many different types of networks are available, so that it is not very difficult to write such a program. Often it is useful to design a neural network using a neural network simulator and to write a program of that network for the production process.

If one wants to use a neural network simulator, one has to clarify the following questions:

- *How much knowledge about neural networks and programming is needed to work with the simulator?* Many programs are very complex and offer a large amount of different possibilities. In many cases most of these possibilities are not needed. But often such a program can only be used by a specialist.

- *Is the program sufficient for the task?* In many cases one needs special properties, which a simple simulator does not offer. Therefore one must be sure that the program one wants to use can fulfill the task one has in mind.
- *Is a documentation available? How much time does one need to make oneself acquainted with the program?* Often the point is not taken into account seriously. A program is useful only if it is well documented. Otherwise, one needs too much time to learn how to work with it and additional costs are created.
- *What is the price of the simulator?* Neural network simulators may be very expensive, but there are also free programs available which are often very good. An overview may be found in the Neural Network FAQs <ftp://ftp.sas.com/pub/neural/FAQ.html> (Chapters 5 and 6).
- *Is it possible to use the program together with other software one needs in the project?* Often the input data are created or coded by existing software products. Often the output data are used by other existing programs. One has to be sure that the neural network simulator can be used together with these programs.
- *Does the program allow **un-supervised learning**?* If one treats a complex problem with long training cycles it is very useful if the neural network simulator is able to optimize the network by itself. Otherwise one needs more human resources, which is usually more expensive.
- *How old is the program? How long has it been developed? Does one know other people who gained positive experience with the program?* Very new programs are often not well tested or even unstable. On the other hand, modern tools are not available in an old program. A modern program with a high version number, which has been developed for many years might be useful, but may have a boring graphical user interface. Often it is very helpful if one knows people who already worked with such a program and know the advantages and disadvantages.

2.5 Project personnel

In a project where a neural network is used one usually needs a specialist who has a lot of experience with neural networks. Therefore it is important to find the right people for the project at the very beginning.

The project manager must be able to hunt up difficulties very early. Therefore he should have at least some experience with neural networks. Neural networks are extremely flexible. They can be used for very different tasks. But they should only be used if more efficient methods are not available. On the other hand, neural networks are new and the success of a simulation with neural networks is not guaranteed. It might always be that the available data are not sufficient and that the neural network cannot produce a useful result. The project manager must therefore be able to see whether or not the project runs well, and he must stop the project if it becomes clear that the neural network cannot be applied successfully to the given problem.

2.6 Costs

As usual the costs of the project are determined by the number of people working in the project, as well as by the hardware and software. Costs for data collection may be high. Furthermore,

the costs for the computing facilities may be higher than in other projects.

2.7 Documentation

This is an essential point. It is very important that each step in the project is well documented. Since the structure of a neural network is mostly developed experimentally, the documentation plays an important role. It is a necessary condition for a fast and reliable implementation of a neural network.

Since often many people are working in the same project, and since new people may join the project, the documentation must allow new coworkers to work in fast. To achieve that, it is useful to fix some standards for the documentation of each step. Especially if different groups are working in the project, a non-proprietary format for the documentation may be useful. XML <http://www.xml.org> offers very good possibilities for this task. One can fix a document type definition (DTD) for different steps that has to be used for the documentation. Furthermore, XML is very flexible and one can produce a readable documentation in very different formats (online, paper, etc) using XML.

The documentation also plays an important role for the control of the neural network in the production process.

3 Data

Data collection, controlling and coding are important aspects of neural network simulations. Especially the quality and quantity of the data has to be checked.

3.1 Introduction

A neural networks deals with input and output data. Artificial neural networks can work with data of very different kind, but the data must often be available in a special form. This means that data have to be coded. The form of the data and therefore the coding is given by the problem one wants to investigate. In some cases, data are already given in a special form and they have to be transformed into a form that can be used by a neural network. In other cases, one has to collect the data. In that case it is useful to fix the coding scheme before data collection is started. The reason is that the coding scheme tells one, which data are really needed. Often the data collection is very expensive, and it may be important to reduce the number of data to be collected. A typical example of coding is a rescaling or normalization of the input or output data. A more complex example is the Fourier transformation of a time series or the coding of several data in a single variable. In an example we will discuss later (see section 8 (Application)) such a more complicate coding scheme will be used.

Another important question concerns quality and quantity of data. Does one have enough data available to train and test the network? Which are the errors of the data? These questions have to be clarified, before long training cycles are started.

In this section, we discuss how the coding scheme is developed and how the output data can be recoded. Further we show how quality and quantity of the data can be checked.

3.2 Coding

The raw data are either continuously distributed in a given range or they are discrete. Furthermore they are given in certain units or they are scaled in different ways. A first step is to calculate simple statistical quantities like the mean or the variance of the data. In the case of discrete data one often needs the probability distribution of the data. These informations can then be used to rescale the data to obtain dimensionless quantities. Furthermore these informations can be used to remove outliers. This may be important for the training of the network. In special cases outliers are important and one needs a non-linear scaling function to treat them in an appropriate way. Neural network simulators often offer a large class of non-linear scaling functions which have been useful in practical applications.

Non-linear scaling is also useful if the data are not equally distributed or if the function the neural network has to learn has regions with a large curvature. Regions with a large curvature are usually more difficult to learn and one often needs more training data to obtain satisfactory results. In some cases, a suitably chosen non-linear scaling function ameliorates the behaviour of the network in such regions of the parameter space.

The neural network produces output data in a form given by the coding of the input data. For instance, the output value may take values between 0 and 1 and has to be rescaled to yield the desired value. Another possibility is to choose the coding such that the output data can be used directly, for instance as an index which has a prescribed meaning.

3.3 Quality and Quantity

The quality of the data depends essentially on the way they have been obtained. The data may have systematic or statistical errors. During the planning of the data collection one should try to avoid systematic errors. If this is not possible, one has to take these errors into account in a suitable coding scheme. Statistical errors in the data are always present and cannot be avoided. One can and should try to minimize them. Statistical tests can be useful to estimate the statistical errors.

Another questions concerns the quantity of the data. How much data are needed to be able to obtain the desired output? And how much complete data sets are needed to train and test the neural network? Often it is useful to work with a network that produces only few output data. It may also be useful to reduce the number of input data to reduce the complexity of the network. The complexity of the network grows exponentially with the number of input data.

A neural network cannot produce new information! This is an important point. All the information one wants to obtain must be present in the input data. For instance, it is not possible to produce a large set of independent output data from a small set of input data. It is also not possible to train a complex neural network with few complete data sets.

4 The network

The design of a neural network has often an experimental character. The network must not be too complex, since then it contains too many parameters which have to be determined in the training process. Such a network loses its ability to generalize. But the network must not be too

simple either, since then the correlations between the data are lost. In this section we discuss some important aspects concerning the design of a neural network.

4.1 Introduction

Finding the optimal structure of a neural network consists often of a series of essays which are ameliorated step by step. Often one can estimate the behaviour of a given network, but details have to be varied. Nevertheless, some aspects of the optimal design of a network can be planned. We shall see that the correct complexity of a neural network can be planned to some extent.

4.2 Design of a multi-layer perceptron

Most neural networks used in practical applications are so-called multi-layer perceptrons. We take the design of a multi-layer perceptron as an example, for other neural networks similar statements can be made.

A multi-layer perceptron consists of an input layer, one or few hidden layers and an output layer (see the figure). It is used to predict the output data using input data. Data processing is done in the hidden layers. An essential parameter of the multi-layer perceptron is the number of neurons. It determines the complexity of the network. The number of neurons in the input and output layers is fixed by the coding, the number of neurons in the hidden layers and the number of hidden layers can be varied.

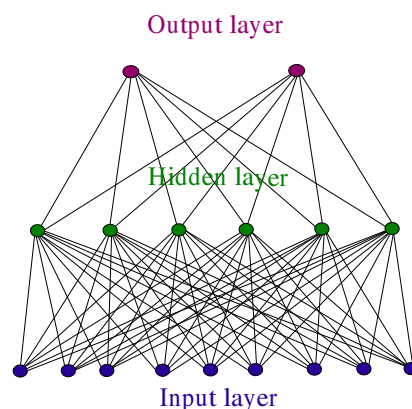


Figure 2: The design of a neural network

4.3 Hidden layers

The hidden layers are responsible for the properties of the artificial neural network. Especially the ability to generalize depends on the number of neurons in the hidden layer. If this number is too small, the network is not able to learn. If it is too large, the ability to generalize is lost.

Furthermore, the number of hidden neurons determines the complexity of the network. More hidden neurons mean more connections between neurons and therefore more parameters that have to be determined. As a consequence one needs larger training sets. This makes clear that there are different aspects that play a role for the design of the hidden layers.

In realistic situations one often has another difficulty: The input data contain redundant information which can not be eliminated by a suitable coding. In such a case the spurious data can be eliminated in the hidden layer, which means that the hidden layer yields a data compression. In such a situation, the hidden layer must contain less neurons than the input layer.

There have been several propositions how to calculate the necessary number of hidden neurons for a given problem. But these attempts yield contradictory results and are of no practical importance. In praxis, one starts with a given number of neurons in the hidden layers. This number is varied and one tries to optimize the behaviour of the network. The goal is to obtain a network with a high ability to generalize. This means that the error of the prediction of the test data has to be minimized. Although there is no unique algorithm that yields the optimal structure, some heuristic rules are available:

1. If the error of the output data during the training is small, but the error in the test-phase is large, the network does not generalize. Often this means that one has to reduce the number of neurons in the hidden layers.
2. If the error during the training is large, one should try to include more hidden neurons.
3. If all the weights for the connections between neurons are of the same order of magnitude, the number of hidden neurons is often too small.
4. One can always add hidden neurons to the network, but the errors may have a different cause. One can e.g. not expect a small error in the output data, if the input data have a large statistical or systematic error.

4.4 Training algorithms

There are several training algorithms for artificial neural networks. For a simple multi-layer perceptron, several standard algorithms are available. The most simple method is a gradient algorithm. One varies the couplings between the neurons in the direction of steepest decent. Popular methods use variants of the so called *error back-propagation*. Which algorithm is used depends often on the details of the problem one wants to solve. Neural network simulators offer many different training algorithms and describe the advantages and disadvantages in the documentation.

5 Errors

In this section we discuss the questions, why errors occur, how they can be minimized, and what one can learn from errors.

5.1 Basic remarks

In a project that uses artificial neural networks, it is not only important to calculate output data, but it is as well important to obtain correct estimates for the errors of the output data. Flawless output does not exist. Sources of errors are the non-perfect generalization of the network and errors of the input data. This means that the error estimate has to take into account different sources of errors.

Sometimes you may find statements telling you that in principle the error of a neural network computation can be made arbitrarily small. This is correct if arbitrary many precise training data are available and if the network can be made arbitrarily complex, which means that the function the network should learn is already known. For practical purposes such statements are meaningless.

5.2 Sources of errors

Systematic errors of the input data should be avoided during the data collection. If this is not possible, they can be reduced by suitable coding schemes.

Statistical errors in the input data are always present. An error-free measurement does not exist. In many cases the statistical errors play a minor role. But since the neural network represents a non-linear function, even a small error in the input data can produce large errors in the output data. Therefore one needs to know how errors propagate in the network. Strictly speaking, these errors are not errors of the network itself.

The main source of the errors is produced by the network itself. Large errors may have different reasons. It may of course happen that the network does not contain the information it needs to generalize. It may also be that the design of the network is not good enough. A third thing that may happen is that in some regions of the parameter space the network has been trained with too few complete data sets. Such errors can be avoided or eliminated by a suitable optimization or fine tuning of the network or by a better choice of the training data. If a network produces large errors although the design has been checked and sufficient training data were available, it may even be that the problem cannot be treated using a neural network.

If one assumes that all the above mentioned sources of errors have been eliminated, there is still a statistical error due to the lack of perfect generalization. The remaining job is to obtain good estimates for these statistical errors.

5.3 Errors in classifications

A typical task of a neural network is a classification. In that case the output data take few discrete values. An example is production control. One has to know whether the product is ready to be delivered or if there is something wrong with it. The classification of the products in these two classes could be done automatically using neural networks.

One obtains discrete output data if one introduces a threshold. If the continuous output lies above the threshold, it falls into one class, if it lies below the threshold, it falls into another class. Depending on the structure of the data, one can vary the threshold to get some information on

the error. Typically a variation of the threshold yields a variation of the error. In which region the threshold can be varied depends on the data.

Generically the classification of a given data set is less accurate if the output is close to the threshold. One can introduce an exclusion region around the threshold. One can count the portion of data sets falling into the exclusion region. Varying the magnitude and the position of this exclusion region, one obtains good estimates for the error of the classification.

5.4 Errors of continuous output data

For continuous output data one can estimate absolute or relative errors. Generically the errors depend on the region of the parameter space. The mapping of the input data to the output data is usually a non-linear function. If the training data are distributed homogeneously, the neural network learns the function less well in regions where the curvature of this function is large. Therefore, strongly inhomogeneous distribution of errors may indicate that in some regions the curvature is larger than in other regions.

5.5 Estimating errors of continuous data

The error estimate is usually based on the error one obtains for the test data. The collection of complete data sets for training and testing the network is divided into two parts. One part is used to train the network, the second part is used to test it. This is a dilemma: One needs many training data to guarantee a sufficient training of the network. But one also needs many test data to be able to estimate the error sufficiently well. How the division of the complete data sets into training and test data is done, depends on the details and may vary from one problem to another.

Another possibility to estimate the errors is error propagation. In a trained network, all the parameters of the network are known and one is able to calculate how errors propagate in the network. These estimates can be used to identify regions of large curvature.

A further possibility to obtain error estimates is to produce faulty data sets. One can use a random number generator to produce input and output data sets with a given error. The reaction of the network on such noisy data sets can be used to study error propagation in the network without investigating the detailed couplings of the network.

One can use a second neural network to predict the error of the output of a neural network. This is done as follows: During the test cycle of the first neural network, one records the errors of the output data. The errors are taken as the output data for the training cycle of the second neural network. The input data of the second neural network are the same as of the first. In this way the second neural network learns the error of the first neural network.

5.6 Interpretation of errors

Often good error estimates can be used to learn something about the behaviour of the neural network. We already saw that large inhomogeneities of errors indicate regions of large curvature. More generally, one can identify regions in the parameter space where the network has learned the data less efficiently. This information can be helpful if one wants to collect new

training data. One should try to obtain more training data in these regions of the parameter space.

A more general problem of error estimates is that generically the couplings within a network are not unique. Different training cycles with the same data yield different configurations of the couplings with comparable errors for the test data. This usually happens if the input data contain redundant information. One cannot avoid this problem. This makes a precise error estimate more difficult. A way out is to use an ensemble of neural networks of the same design, trained differently. To each of these networks one associates a weight that depends on the error estimate for this network. Now the output data and especially the errors are calculated by averaging over the ensemble of networks according to the weights. This procedure has often been used in medical applications, where not too many complete data sets are available and the input data themselves have large statistical errors.

5.7 Special cases

There are special tasks or variants of multi-layer perceptrons where the error itself has a special meaning. An example is the prediction of the input data out of the input data. At a first glance this is a stupid task. But if one introduces hidden layers with a smaller number of neurons than in the input layer, this task becomes non-trivial. A network of that kind can be used to estimate the amount of redundant information. In the hidden layers the information contained in the input data is stored in a compressed form. The network can reproduce the input data if the information density for all data sets is almost the same. It can therefore be used to identify input data which belong to the same class as the data it has been trained with. If a new input data set cannot be predicted it may have a larger information density and belongs to a different class.

6 Rules

In the last sections we discussed how a network is trained. Here we discuss possibilities to extract rules from trained neural networks.

6.1 Introduction

In the past a neural network has often been viewed as a black box. It worked but one was not able to understand how it worked. Today, various methods exist to extract information from a trained neural network. The perfect knowledge one could obtain from a trained neural network would be an explicit expression for the function the neural network presents. In practical cases this function can not be calculated. Since this function is complicated, one can not expect a simple, e.g. linear connection between the input and output data. The rules one can derive from a neural network represent some partial information.

Why does one want to derive rules from a trained neural network? If the neural network works, why should one want to know, how it works? There are several algorithms to extract rules from a trained neural network. Depending on the quality of the rules one wants to obtain they are time consuming and therefore expensive. Nevertheless there are some good reasons to obtain rules from a trained neural network:

- The user of the network better understands how it works.
- The ability of the network to generalize can be improved by some refinement techniques.
- Rules can be compared with other knowledge to check the network.
- Rules can be used in other areas of the project.
- Rules are important to explain the project to other people (e.g. credit providers).

The parameters of the network, especially the connections between the neurons, contain the information stored in the network. Rule extraction is mainly based on the analysis of these parameters and of the behaviour of the network.

6.2 Method for rule extraction

There are various methods which allow for the extraction of rules. They can be divided into analytic and synthetic methods.

Analytic methods try to directly analyze the couplings of the network. The network is divided into several parts, the behaviour of each part can be analyzed, and later the parts are glued together to understand the entire network. The analysis of a small part of the network can be done either by a direct analysis of the couplings or by numerical tests. Later one can try to optimize the structure of a part of the network or of the entire network using the knowledge one obtained. Then, the optimized network can again be used to obtain rules. In this way, the structure of the network and the rules are refined.

Synthetic methods regard the network as a black box, that yields the output data as a function of the input data. One may characterize this functions using the local derivatives. The matrix of local derivatives contains information about the local behaviour of the function. The derivatives may be obtained either analytically using the couplings of the network, or numerically studying small variations of the input data. Derivatives can only be used for continuous input and output data. Small variations of the input data can as well be used if the neural network has discrete output data, i.e. for classifications. It allows to extract regions of stable behaviour of the network. Studying the derivatives also allows to optimize the network. If for instance the derivative with respect to a single input variable is globally small, this variable is not very important and can probably be neglected.

6.3 Classification of rules

Having extracted rules one can apply several criteria to judge these rules. There are various criteria, some of them are opposed to each other. The following criteria are important:

Quality.

The rules should be correct. This seems to be trivial but in many cases it is not easy to verify the rules. Clearly, the rules must not contradict each other. One should also try to estimate the accuracy of the rules. Finally it may be important to know in which range of parameters the rules can be applied and if the rules generalize to regions, in which the network has not been trained.

Complexity.

Rules are extracted to use them. It makes no sense to create a set of very complex rules the application of which needs more powerful computational facilities than the network itself. On the other hand, one cannot expect to obtain simple rules for a complex problem.

Comprehensibility.

Comprehensibility and complexity depend on each other. A user who wants to apply the rules, e.g. to check the consistency of some output data must be able to understand the rules. Comprehensibility depends on how the rules are formulated and who must understand them. In some cases rules are used again by a computer, so that they have to be formulated in some formal language.

Applicability.

The range of applicability of a given rule may be restricted to parts of the parameter space. Rules can explain the behaviour of the network for all possible input data, or they explain the behaviour only in a restricted range of input data.

It is clear that some of these points contradict others. A higher quality of rules may cause a higher and unwanted complexity, which makes the rules incomprehensible. Therefore one has to know what one wants to achieve when one extracts rules from a trained neural network.

7 Optimization and control

Neural networks can be (and must be) controlled and optimized even in the production process. There are several possibilities to do that. Rules (see section 6 (Rules)) can be used to control the network.

7.1 Introduction

Various aspects are important if one wants to optimize and control a neural network:

Costs:

Running a neural network may be expensive. There are not only costs for the computational facilities and for the personnel, but also costs for the collection of the input data. In medical application it may be very expensive to obtain input data. We already saw, that eventually derivatives can be used to find unimportant input data, which can then be eliminated.

Quality:

The output data of a neural network always have an error. One goal of optimization can be to reduce the error. This can as well be done using the rules.

Speed:

In some applications the speed of the network can be a critical quantity and one wants to reduce the calculation time of the network. There are several possibilities to do that. One can use a special purpose program instead of a general *neural network simulator*, or one can use more powerful hardware.

As before, some of these aspects contradict others. One has to know which goal one wants to reach, before optimization is started.

A basic requirement for controlling is a sufficient documentation of the network and the whole project (see section 2.7 (Documentation)). This must be taken into account during the planning of the project and should be done using fixed standards.

7.2 Refinement of a neural network

Optimization of the network design, including the optimization of the number of neurons, the properties of the neurons, and the connections between the neurons is usually done during the development of the network. But even in the production process minor optimizations of that kind may be useful. During the production process one has to check, whether the input data lie in the region where the network was trained. Outliers can be found using statistical methods. Outliers may also contradict established rules. When such input data occur, one should check the original training set. It may be that the training of the network was not sufficient. It may be that the network treats outliers in a correct manner. Nevertheless, one should think about a new training cycle to improve the behaviour of the network.

Optimization of the speed of the network may be important. But one has to be sure that the quality of the network remains the same. In many cases it is not only useful to try to enhance the speed and to reduce the calculational costs, also costs for collecting input data are important. One may therefore try to minimize the number of input data. Again one has to be sure that this is possible without losing quality.

7.3 Optimization and control of the software

Neural networks are finally a software product. Concerning control they should be treated as other software products. It may happen that bugs or instabilities occur after some (long) time. This may happen if after that time input data occur which have not been taken into account earlier. A special point is that neural networks, in contrast to most other software, yield output data with some intrinsic error.

In many cases neural networks are developed using neural network simulators. (see section 2.4 (Resources)). These programs have important advantages. Often they offer special tools for analyzing the behaviour of the network. They also provide a graphical user interface which makes it easy to deal with the program. On the other hand, in the production process, these properties are often unnecessary. Therefore after the design and the training is finished, one should implement the neural network using a higher programming language and a fast compiler. This produces singular costs, but the network needs smaller resources in the production process and therefore one can save these costs easily. If the speed of the network is a critical quantity, this kind of optimization is essential. Furthermore, a smaller software package can be controlled more efficiently than a larger one.

7.4 Optimization of the Hardware

Neural networks can be developed and programmed on any modern hardware. When one wants to optimize the hardware, two aspects play an important role: Speed and stability. New

hardware can improve the speed of the network a lot, but an optimization of the software can increase the speed even more. Stability is important since the network should work on long time scales (several years). The hardware must allow that. It is also important that the network can easily be ported to newer hardware. The choice and the maintenance of the operating system are important as well.

8 Application

In this section we describe an application of a neural network in a medical project.

8.1 Description of the project

This project is part of a medical study on a specific active immunotherapy (ASI) <http://people.freenet.d> (pages in German) for cancer. The aim is to analyze clinical data and immune data of patients who got the therapy. During the therapy a patient is vaccinated three times. Before and after each vaccination one obtains immune data. It is very difficult to draw conclusions out of these data. The idea is to use a neural network to predict the progress of the therapy using clinical data and immune data. The final goal is to develop an index that tells the physician what progress can be expected if one uses the therapy for a given patient. The index will be based on clinical data and on immune data before vaccination, probably also on the immune data after a reaction of the patient on the first vaccination.

8.2 Preparatory work

Before one can start the development of a neural network one has to define the output data. This means that an index has to be defined based on the progress of patients who already got the therapy. The main problem is that different patients have been observed during very different intervals. It is clear that the observed progress depends on the length of the interval. This means that one has to assign different data sets a different weight, depending on the length of the interval. The definition of the index is a typical example for a complex coding of the output data (see section 3.2 (Coding)). Coding of the input data is trivial in this project, it consists of a simple normalization.

8.3 Implementing a neural network

When we started to work with neural networks in this project, data of about 200 patients were available. Unfortunately, not all of these data have been recorded systematically, so that some of the data sets were incomplete. The dilemma was that we could either work with a smaller number of complete data sets or with a larger number of incomplete data sets. Since a priori it was not clear, which data are really important, we decided to work with complete data sets only. But since the number of patients was not too large, we encountered relatively large statistical errors of about 10 percent. Using these preliminary studies we learned, that some of the data were not really important and we were able to take a part of the incomplete data sets into account. The result of these calculations was that a multi-layer perceptron with a single hidden

layer can be used to obtain satisfactory results. Unfortunately, the error of the results was still too large. The reason were statistical errors in the input data, which could not be eliminated.

In order to reduce the statistical error of the input data, the collection of the immune data is now done using standardized procedures. Since some time, a new trial with 600 patients has started. Since the data collection is now much better and since the number of patients is larger, we expect the new data can be used to obtain a working neural network and small enough errors of the output data. The experience made so far is clearly very important for the new study.

8.4 Outlook

The trial will last several years. Furthermore the patients have to be observed over a long period. But some first results may be available earlier, especially concerning the relevance or irrelevance of some input data.

9 Links to other sites providing information on neural networks

This list is very incomplete and will be updated regularly. Do you work on neural networks or do you provide informations on neural networks on some WWW-page? Then please contact us and we mention your site here.

- The risk estimation for prostate cancer <http://www.charite.de/ch/uro/html/prostatabiopsi> was already mentioned. On a german WWW-page <http://www.charite.de/ch/uro/schwerpunk> the authors provide additional information including literature about their method.
- The Neural Network FAQs <ftp://ftp.sas.com/pub/neural/FAQ.html> provide a good starting point. There one also finds links to other pages. Unfortunately, many of these links do no longer exist.
- NeuroDimension Inc. provides a page <http://www.nd.com/appcornr/purpose.htm> with several links to applications of neural networks. Unfortunately, many of these links do no longer exist.