BackProp Task

Date:

2002-11-12

Authors:

David Öggesjö <it1ogda@ituniv.se>

Henric Thisell <it2thhe@ituniv.se>

Sebastian Edman <it2edse@ituniv.se>

Source Code:

source.zip

source.tar.gz

source.tar.bz2

Table of Contents

Chapter 1 - Introduction

This report is the result from a project at the IT-university in Gothenburg, Sweden. The project is a part of the course Adaptive Algorithms that was running in the beginning of Autumn 2002. The part of the course that this project is about was Neural Networks, in particular Back Propagation and viewing multi dimensional data sets. For the multidimensional viewing of data sets, the project described in this report is using the Kohonen's SOFM.

A short introduction to Back Propagation, Kohonen's SOFM and the Principal Component Analysis is to be found in the respective section of the report.

1.1 Problem description

The problem was divided into two parts. The first was to examine some data sets, the second was to train a feed forward network with two of the data sets using the back propagation algorithm. The first part was intended to visualize the problem, so that the difficulty of one particular data set could be evaluated. Then two data sets, one easy and one difficult, were to be chosen. Each of those two data sets should then be used to train a feed forward network. Both networks should be trained using the back propagation algorithm.
[toc | next | up]

1.2 Organization

The organization of this report is the following. Chapter one, is this short introduction. Chapter two and three presents the data analyze stage, where the Kohonen's SOFM and the Principal Component Analysis is described. The resulting program, its procedure and the results are described in detail as well. Chapter four and five present the Feed Forward network that was trained using the back propagation algorithm. Chapter six and seven present the conclusion, what modifications this program need to be usable in the future and possible applications for the program. The report is ended with a small glossary that explains the technical terms.
[toc | next | up]

Chapter 2 - Kohonen Data Analyzing

This chapter contains some brief information about the Kohonen's SOFM (Self Organizing Feature Map) and the choices that was made to get it work satisfactory. One obvious alternative of using Kohonen's SOFM for the viewing of multiminensional data sets is the Principal Component Analysis, which is explained very biefly. There is also a short description of the source code to the program that was made to accomodate the demands.

2.1 Introduction to Kohonen's SOFM

Kohonen's self organizing feature map is an extended competitive learning network. A competitive learning network is a feed forward network that consists of one layer of neurons. The neurons has one input for each dimension that is going to be analyzed and each neuron has one output that is supposed to represents a certain pattern in the input data. To map the patterns of the input data to a certain output neuron, the network has to be trained to recognize these patterns. That means that when an element of the data is analyzed there is one winning output neuron which is closest to recognize the pattern. The difference between Kohonen's SOFM and a competitive learning network is that in a Kohonen's SOFM there is a connection between the neurons. Because of this a certain pattern in the input data will seam to get a certain part of the output neurons as their winning neurons and there will be some clustering of the outputs that represents the classes of input.
[toc | previous | next | up]

2.2 Introduction to the Principal Component Analysis

The principal component analysis is a classical statistical method. The goal of the method is to decrease the number of dimensions in a multidimensional array. Basically only the axis with most variation is kept and the others are ignored, this can be calculated with statistical method. By doing this a data set with many dimensions can be reduced to two or three dimensions and plotted in a graph that can easily show if it is easy or hard to classify the data.
[toc | previous | next | up]

2.3 The Kohonen Choices

At the beginning of the course the Kohonen's SOFM seemed to be the better choice because it, for the first, did not contain any advanced mathematics. Second it was more in line with the course and felt more interesting to explore. This made the choice quite easy and it was decided to use the Kohonen's SOFM instead of the Principal Component Analysis in the beginning of the project.

When using the Kohonen's SOFM there is some parameters that needs to be addressed. The most obvious one is probably the size of the output layer. The size was chosen to be 13. The choice was made because it seemed to be a well balanced value (with respect to computational time and size). The computational time was also a big burden because there was a lot of squares and root squares (*RMS* calculating). This was reduced to a simple sum of the absolute values instead. More about this is described in chapter 3. The *Learning Rate* and the *Sigma value* was chosen to decrease over time so that an acceptable converging would be achieved.
[toc | previous | next | up]

2.4 Program Structure

The Kohonen program is divided in two parts and two files. One that is called *readfile.c* and one that is called *kohonen.c*. The first file reads the data files, which is used both in the Kohonen SOFM and in the back propagation program. The second file contains the Kohonen specific functions.
[toc | previous | next | up]

Readfile.c

This file takes care of the reading the input data files. Because all data files are a little different there is one function for each data file. This function reads one row of data at the time and saves it as an list of input and as the last element is the output. These functions are called *lRead<DataSetName>* and has to be executed before any data can be received. The last thing that happens is that the values get normalized between -1 and 1. During the operation two variables are set, *lWidth* and *lSize* which are the number of elements in one data row and the number of rows in the set. After the file has been read the function *lGetRow* can be called to get one row of data from 0 to *lSize*-1. The elements in the array that is returned is of the type floating-point

numbers and the output is located last in the array and is an integer value between 0 and the number of outputs.
[toc | previous | next | up]

Kohonen.c

This file contains all the Kohonen's SOFM functions. First, all the values are initialized and all the data from the appropriate file is read. The weights of the network are set to random values between 0.2 and 0.8. Then for each row of the set a winning neuron is calculated. The winning neuron's weights and its neuron neighbours weights are then updated. Which neuron that are close enough to be called neighbour is determined by the function *lambda*. When the whole set has been searched, the network goes back to the first row and goes on like this until it is decided that it does not need to be trained more. The program is generally terminated after about 2000 iterations. Before every new iteration starts, the sigma and the learning rate are calculated. These values are depending on time and are used in the lambda function and in the function that updates the weights. After a predefined number of iterations the program runs a function that calculates the winner for all input rows in the set and write the distribution for all the different output to files. These files can then be used to plot 3D graphs showing the distribution of the outputs. In this case the program *gnuplot* was used. These graphs can be seen in the data analyze section.
[toc | previous | next | up]

Changeable Parameters

In a Kohonen network there are some parameters that can be changed. The first thing is the number of neurons in the network and the neurons initial weights. Two other parameters that can be altered are the sigma value and the learning rate, they could be constant values or functions that changes during time. The sigma value determines how many of the neurons surrounding the winning neuron are moved and the learning rate determines how much they will move each time. Also the number of inputs to the network can be changed, often this is set to the same size as the input data set. There is also the possibility to use the more time consuming RMS calculation instead of the sum of absolute values.
[toc | previous | next | up]

Chapter 3 - Kohonen Data Analyzing Results

Even though the Kohonen's SOFM is pretty straight forward, there is still things that can go wrong. This is described in the first part of this chapter, the second part are some general graphs and the third part is the result of all the data sets. This is quite an amount of graphs, but many very interesting.

3.1 Initial Problems

Well, who said that problems do not exist? Here is the proof that they not even exist, they are also quite annoying.
[toc | previous | next | up]

How to Present the Result?

There where some difficulties how to represent the output of the network. First every neuron that was a winner for a certain output was coloured in a separate colour for this output. This showed a some interesting things, but because the outputs overwrite each other this didn't show quite what was expected. Instead the distribution of the output was written as a table and plotted as a 3D graph. This made it much easier to see the clusters in the output. Some different number of neurons in the output layer was tested, from 3x3 to 100x100 and 13x13 seemed to be suitable for this purpose.
[toc | previous | next | up]

Huge Data Sets

The Protein data set is huge, and the nettalk is even many times larger. This inflicts problems in memory management. First a simple linked list was made and to access each element (each element contains one row of the data set) of the list it was required to loop through all data. In the case of 21,000 elements this was quite tedious and took about 1/10 of a second for one element, this is because of the way a RISC-computer architecture is constructed (there is no way 35MB of data will fit into the processor cache). That means that to just perform a simple task to all of the elements in the list, it would take several minutes. This of course had to be changed, and a quick jumping table was programmed.
[toc | previous | next | up]

Computational Time

This is also one very interesting point. In the Kohonen Network there is the need to calculate winning output neuron. This is done by summering all the neuron's weights errors and compare them with each other. The method that is often used is the Root Mean Square method, which means that each weight error is squared and then the resulting value is the square root of the sum of all the squared errors.

$$comparableValue = \sqrt{\sum_{n=0}^{N} \left(weightError_n^{\,2} \right)}$$

The square and the square root functions actually take a considerably large time to execute. In the resulting Kohonen program we have therefore only used the sum of all the absolute value errors.

$$comparableValue = \sum_{n=0}^{N} \left(abs(weightError_n) \right)$$

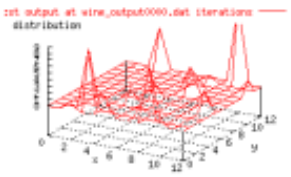
This makes the program execute seven times faster, which is several days of processor computing when organizing the protein data set.
[toc | previous | next | up]

3.2 Graphs

A collection of some general graphs that are common for all of the later graphs.

Animation of Kohonen's SOFM in Progress

To actually get a picture whether this Kohonen model works or not, this animated gif of one of the wine representations was made. It clearly shows that the data is moved in a satisfying manner.

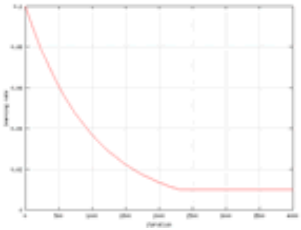


Animated gif over the change of Kohonen output layer in time.

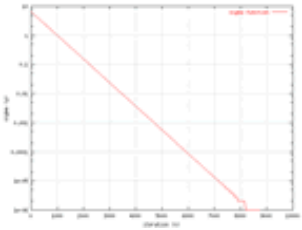
[toc | previous | next | up]

Learning Rate and Sigma Function

Because of some values changed in time, data from these values where collected. In the graphs below. The change of the learning rate and the sigma value from the wine data can be seen and they are the same in all sets.



The change of the learning rate in time for the wine output.

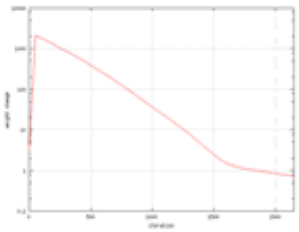


The change of the sigma value in time for the wine output.

[toc | previous | next | up]

Change in Weights

Another change in time that is relevant to look at is the change of the weights. This can be seen below, again from the wine data set and there is no big difference between some of the sets as well.



The change of the weights in time for the wine output.

[toc | previous | next | up]

What does the Graphs Show?

The difference of output values showed that it was rather hard to see if there was a good classification if there was many variations of the output. The difference can be seen in a comparison between the output of the mushroom data and the isolet data. Both are rather easy problems and the output of the mushroom data shows this, but the isolet data is much harder to tell if it's an easy or hard problem.
All the figures in the data set section below shows the distribution of the neurons in the output layer in the Kohonen network for each possible output after 2000 iterations.
[toc | previous | next | up]

3.3 Analyzed Data Sets

All the data sets are represented in different ways. This led to that each data files had to be treated in special way. This is discussed in the readfile.c section. The things that differs are the number of inputs, the number of outputs, the type of the elements (characters, integers or decimal numbers) and the number of variations of the output. All the data sets are briefly explained later. Because of the difference in the amount of data the time it took to run the program varied very much. The echo data only took a minute or so and the nettalk data took so long time to run that some data had to be deselected. To do this a random function with a possibility of 95 percent to skip one row the data was used. That led to a data set of 5 percent of the original, which even then took about twelve hours to run. All the sets were trained 2000 times with the training set and in a certain interval output data were collected. This gets an output like the one below that change during time and gets more and more stable.

All data was normalized before presented to the network, this means that all of the different dimensions will have the same impact on the system. The normalization range chosen for the Kohonen Network was 0 to 1.
[toc | previous | next | up]

Echocardiogram

This data set was donated by Steve Salzberg (salzberg@cs.jhu.edu) and shows echocardiogram information from 132 patients that suffered heart attacks. The problem is to predict if the patient will not survive one year after the attack. Part of the difficulty to do this is because of the size of the training set, it only contains 4 patients that did not survive the first year and 36 of the patients can not be part of the training because they wasn't followed one whole year. Each element in the set contains 13 numerical values, of which four are meaningless and two are used to get the output. That means that there are 7 inputs and one output which can be 0 or 1. Earlier work has shown that at least about 60 percent correctness is possible. The output of the Kohonen network below shows that there are only 4 outputs that are 0, but it also shows that they are relatively close which means that it is possible to a rough classification of the data, but it would probably be better with a larger training set.

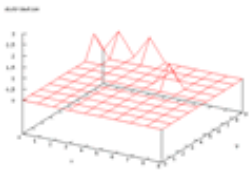


Figure 4 output layer from the Kohonen network that shows the distribution of patients that died during the first year after the heart attack

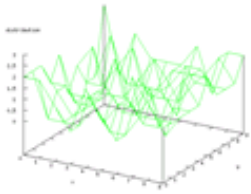


Figure 5 output layer from the Kohonen network that shows the distribution of the patients that survived at least one year after the heart attack

[toc | previous | next | up]

Heart-disease

The responsible for these sets are Andras Janosi, M.D at Hungarian Institute of Cardiology. Budapest, William Steinbrunn, M.D at University Hospital, Zurich, Switzerland, Matthias Pfisterer, M.D at University Hospital, Basel, Switzerland and Robert Detrano, M.D., PhD at V.A. Medical Center, Long Beach and Cleveland Clinic Foundation. The sets contain data from 4 databases concerning heart diagnostics. The set are divided in 14 variables of which the last is the presence of heart-disease where 0 is absence and 1, 2, 3 and 4 are presence. The problem is to find a presence or absence of heart-disease of the data. Earlier work has shown that at least about 75 - 80 percent correctness is possible. The output from the Kohonen network below shows that there is a higher density in the distribution of the two output possibilities, but it is not an absolutely clear distribution which means that there is some source of error that can be hard to predict.

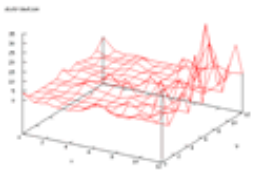


Figure 6 output layer from the Kohonen network that shows the distribution of absence of heart-disease

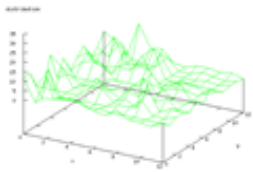


Figure 7 output layer from the Kohonen network that shows the distribution of presence of heart-disease

[toc | previous | next | up]

IR-spectra

This set was donated by John Stutz, stutz@pluto.arc.nasa.gov. The problem is to classify infra red spectra into 10 main classes. The inputs are 93 flux measurements from two different frequencies observed by the infra-red astronomy satellite. As can be seen below some classes are not very well represented in the set. Therefore some of the classes may be hard to distinguish

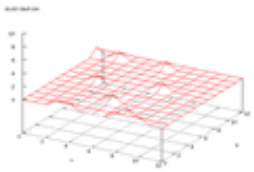


Figure 8 output layer from the Kohonen network that shows the distribution of the first basic class

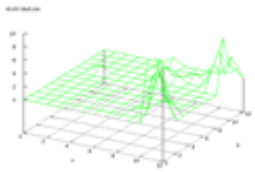


Figure 9 output layer from the Kohonen network that shows the distribution of the second basic class

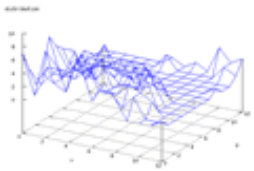


Figure 10 output layer from the Kohonen network that shows the distribution of the third basic class

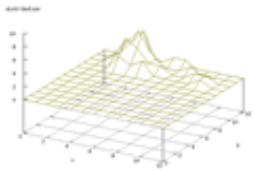


Figure 11 output layer from the Kohonen network that shows the distribution of the fourth basic class

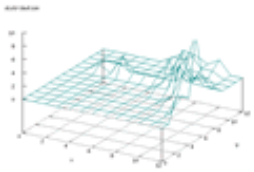


Figure 12 output layer from the Kohonen network that shows the distribution of the fifth basic class

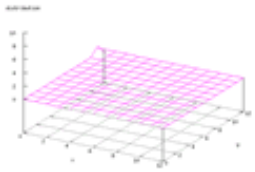


Figure 13 output layer from the Kohonen network that shows the distribution of the sixth basic class

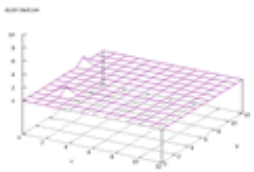


Figure 14 output layer from the Kohonen network that shows the distribution of the seventh basic class

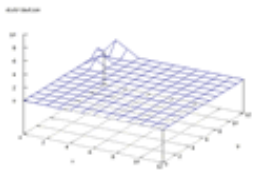


Figure 15 output layer from the Kohonen network that shows the distribution of the eighth basic class

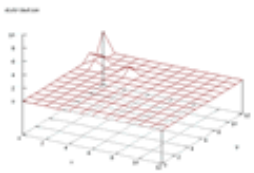


Figure 16 output layer from the Kohonen network that shows the distribution of the ninth basic class

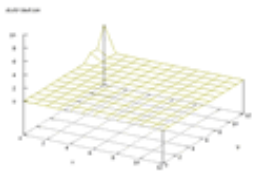


Figure 17 output layer from the Kohonen network that shows the distribution of the tenth basic class

[toc | previous | next | up]

Isolet

This set was donated by Tom Dietterich, tgd@cs.orst.edu. The set contains data from 150 subjects that spoke the name of each letter of the English alphabet twice. The problem is to predict which letter name was spoken. This is, according to the information on the set, a rather simple task, but because of its many different output signals it gets a little bit difficult to separate the outputs from the Kohonen network. Even tough it can be seen that the distribution of each output value are rather concentrated, it is not as clear as for example the mushroom data.

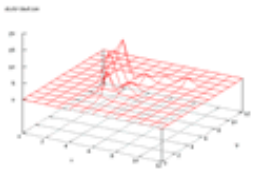


Figure 18 output layer from the Kohonen network that shows the distribution of the letter A

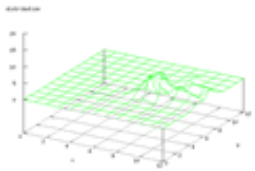


Figure 19 output layer from the Kohonen network that shows the distribution of the letter B

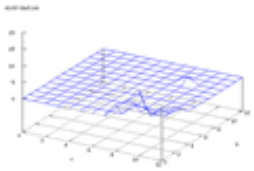


Figure 20 output layer from the Kohonen network that shows the distribution of the letter C

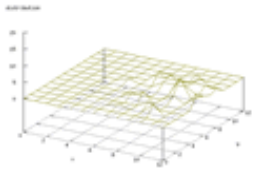


Figure 21 output layer from the Kohonen network that shows the distribution of the letter D

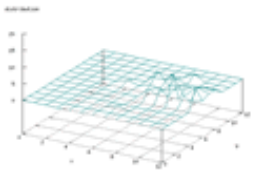


Figure 22 output layer from the Kohonen network that shows the distribution of the letter E

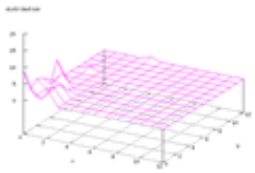


Figure 23 output layer from the Kohonen network that shows the distribution of the letter F

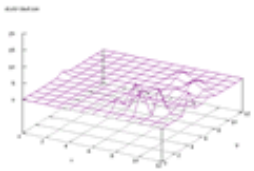


Figure 24 output layer from the Kohonen network that shows the distribution of the letter G

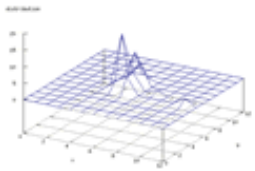


Figure 25 output layer from the Kohonen network that shows the distribution of the letter H

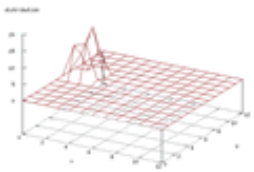


Figure 26 output layer from the Kohonen network that shows the distribution of the letter I

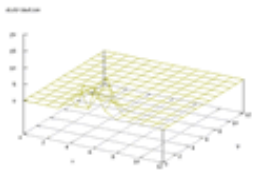


Figure 27 output layer from the Kohonen network that shows the distribution of the letter J

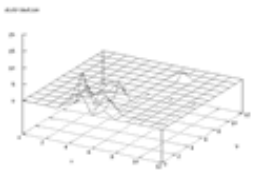


Figure 28 output layer from the Kohonen network that shows the distribution of the letter K

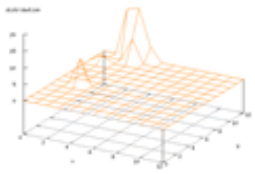


Figure 29 output layer from the Kohonen network that shows the distribution of the letter L

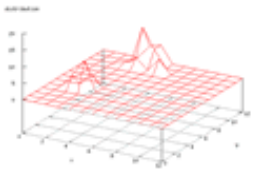


Figure 30 output layer from the Kohonen network that shows the distribution of the letter M

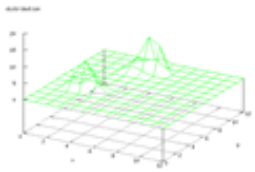


Figure 31 output layer from the Kohonen network that shows the distribution of the letter N

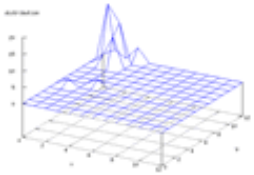


Figure 32 output layer from the Kohonen network that shows the distribution of the letter O

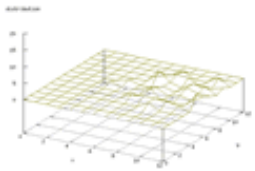


Figure 33 output layer from the Kohonen network that shows the distribution of the letter P

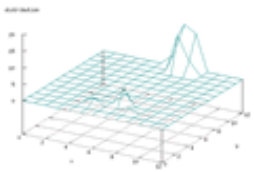


Figure 34 output layer from the Kohonen network that shows the distribution of the letter Q

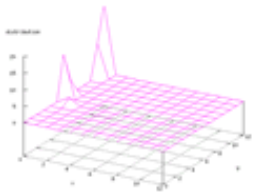


Figure 35 output layer from the Kohonen network that shows the distribution of the letter R

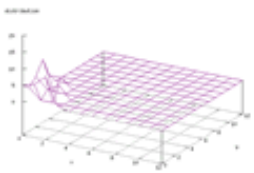


Figure 36 output layer from the Kohonen network that shows the distribution of the letter S

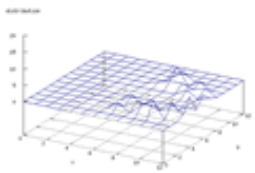
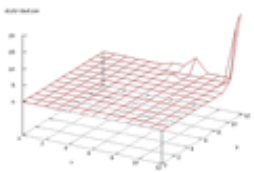


Figure 37 output layer from the Kohonen network that shows the distribution of the letter T



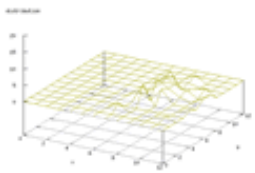


Figure 38 output layer from the Kohonen network that shows the distribution of the letter U

Figure 39 output layer from the Kohonen network that shows the distribution of the letter V

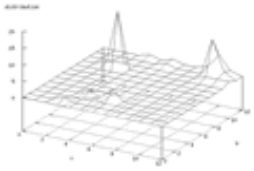


Figure 40 output layer from the Kohonen network that shows the distribution of the letter W

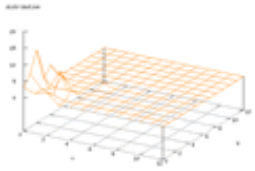


Figure 41 output layer from the Kohonen network that shows the distribution of the letter X

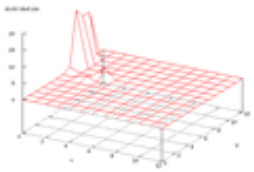


Figure 42 output layer from the Kohonen network that shows the distribution of the letter Y

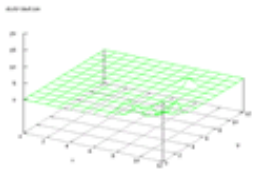


Figure 43 output layer from the Kohonen network that shows the distribution of the letter Z

[toc | previous | next | up]

Mushrooms

This set was donated by Jeff Schlimmer, jeffrey.schlimmer@a.gp.cs.cmu.edu, and contains data from 8124 mushrooms and consist of 22 nominally values and a class attribute (edible or poisonous). The problem is to predict if a mushroom is edible or poisonous. Earlier work has shown that it is possible to 95 percent classification accuracy and the output from the Kohonen network below shows that the two different classes are clearly distinguishable.

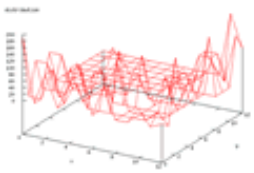


Figure 44 output layer from the Kohonen network that shows the distribution of the edible mushrooms

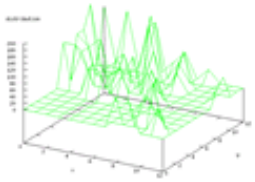


Figure 45 output layer from the Kohonen network that shows the distribution of the poisonous mushrooms

[toc | previous | next | up]

Nettalk

This set is allowed to use for non-commercial research purposes by Johns Hopkins University and contains the 1000 most common English words and how they are pronounced. The problem is to predict how each letter is pronounced. In earlier attempts to solve this problem a window of seven letters have been used where each letter is an array of 26 Booleans, one for each letter in the English alphabet, all together 182 inputs was used. There are two different outputs, one for which letter and one for how the letter should be

pronounced, that is 260 different variants of output. Because of this the Kohonen output would be very hard to analyze, so a different approach was used to do it. Only the difference in the pronouncing was written as output from the network as the graphs below shows.

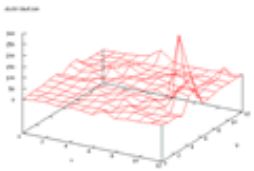


Figure 46 output layer from the Kohonen network that shows the distribution of > of the second output

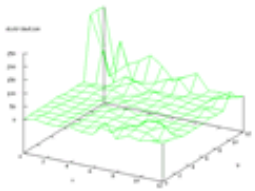


Figure 47 output layer from the Kohonen network that shows the distribution of < of the second output

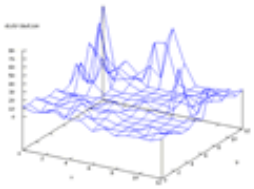


Figure 48 output layer from the Kohonen network that shows the distribution of 0 of the second output

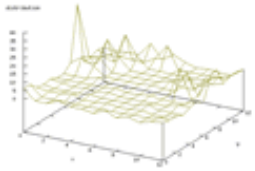


Figure 49 output layer from the Kohonen network that shows the distribution of 2 of the second output

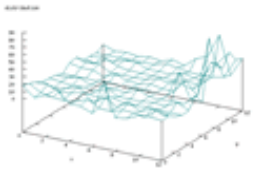


Figure 50 output layer from the Kohonen network that shows the distribution of 1 of the second output

[toc | previous | next | up]

Proteins

This set is allowed to use for non-commercial research purposes by Johns Hopkins University and contains data from a number of proteins, which consists of 20 amino acids with a following secondary structure. The secondary structure can be of three types, alpha-helix, beta-sheet and random-coil. It is not clear from the task which symbol that represents what secondary structure, so the symbol rather than the name is used later in the report. The problem is to predict the secondary structure given the amino acid sequence.

There was only minor corruption in the two files and the corruption is more like inconsistencies in the file format. Some of the 'end' tags were missing and the format of the 'end' tags were different in the two files. This was corrected in the data files before they were used as input to the program.

The training data consists of 18,105 amino acids with corresponding structure and the test data (used as validation and not used for training) consists of 3,520 amino acids with corresponding structure. Each data set row has a fixed number of previous and subsequent amino acids exactly like the nettalk approach. A window size (number of elements per row) of 7 and 21 was tried and also different representations for the

data was used. The first representation tried was to use one float for each row element or 7 and 21 inputs respectively. The other approach is to present the data with 20 Boolean values for each row element, this would result in 7*20 and 21*20 inputs respectively. The 420 input version was used to produce the following graphs.

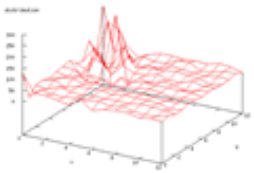


Figure 51 output layer from the Kohonen network that shows the distribution of the secondary structure _

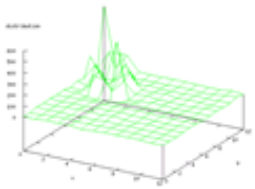


Figure 52 output layer from the Kohonen network that shows the distribution of the secondary structure e

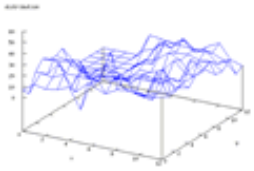


Figure 53 output layer from the Kohonen network that shows the distribution of the secondary structure h

This results really suggest that the secondary structure 'h' is very easy to distinguish from the rest of the data.

[toc | previous | next | up]

Sonar

The set contains data from sonar signals. The problem is to predict if the signal comes from a rock or a mine. The set contains of 208 rows of data and earlier work has shown that it is possible to classify the data very well, almost up to 90 percent. The output from the Kohonen network below shows that there are some distinctive differences in the outputs, but that there is also some overlapping in the distributions.

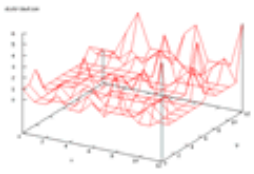


Figure 54 output layer from the Kohonen network that shows the distribution of rock output

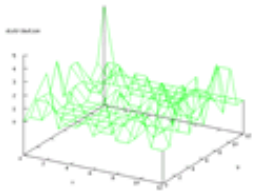
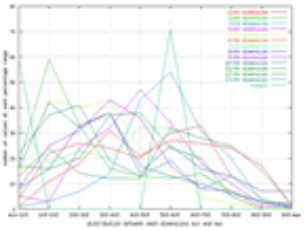


Figure 55 output layer from the Kohonen network that shows the distribution of mine output

[toc | previous | next | up]

Wine

This set comes from Stefan Aeberhard, stefan@coral.cs.jcu.edu.au, and contains data from a chemical analysis of 178 different wines from three different cultivars. The inputs are 13 numerical values and the problem is to predict which cultivar the wine belongs to. The output from the Kohonen network below shows that this is a very easy set to classify and earlier work has shown that it is possible to predict the cultivar to almost 100 percent.



The dimensional normalized distribution of the wine data set.

As can be seen from this graph, all the data dimensions are well distributed over the input space. This can be important to se so there is not one small value that is skewing all the data to one corner.

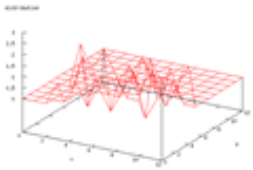


Figure 56 output layer from the Kohonen network that shows the distribution of the first class of wines

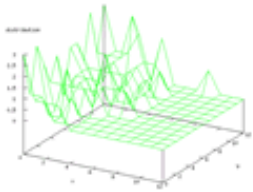


Figure 57 output layer from the Kohonen network that shows the distribution of the second class of wine

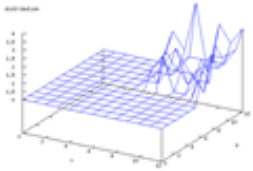
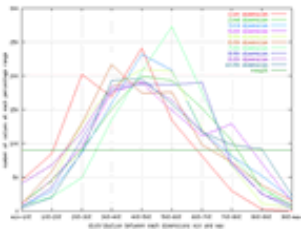


Figure 58 output layer from the Kohonen network that shows the distribution of the third class of wine

[toc | previous | next | up]

Vowels

The set was collected by David Deterding at the University of Cambridge, who recorded examples of the eleven steady state vowels of English spoken by fifteen speakers for a speaker normalization automatic speech recognition. The problem is to predict the eleven steady state vowels. The output from the Kohonen network below shows that there are some clear classifications, but they are all overlapping a little bit, this could make it hard to separate the different vowels.



The dimensional normalized distribution of the wine data set.

As can be seen from this graph, all the data dimensions are well distributed over the input space. The distribution is almost following the normal distribution this is a quite good sign, and suggests that the data is well represented..

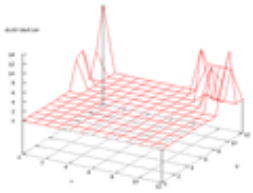


Figure 59 output layer from the Kohonen network that shows the distribution of the first of the vowels

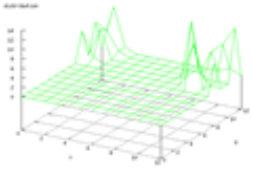


Figure 60 output layer from the Kohonen network that shows the distribution of the second of the vowels

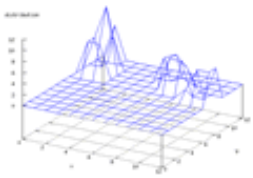


Figure 61 output layer from the Kohonen network that shows the distribution of the third of the vowels

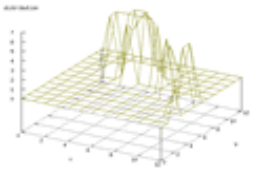


Figure 62 output layer from the Kohonen network that shows the distribution of the fourth of the vowels

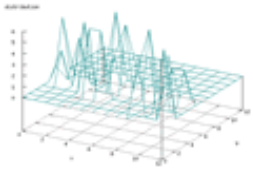


Figure 63 output layer from the Kohonen network that shows the distribution of the fifth of the vowels

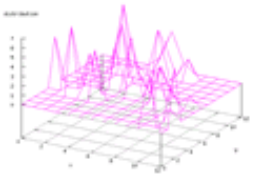
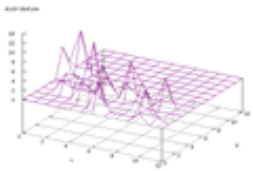


Figure 64 output layer from the Kohonen network that shows the distribution of the sixth of the vowels



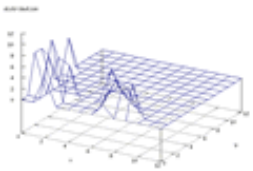


Figure 65 output layer from the Kohonen network that shows the distribution of the seventh of the vowels

Figure 66 output layer from the Kohonen network that shows the distribution of the eighth of the vowels

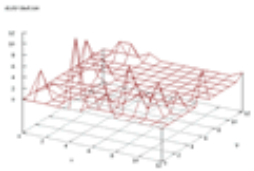


Figure 67 output layer from the Kohonen network that shows the distribution of the ninth of the vowels

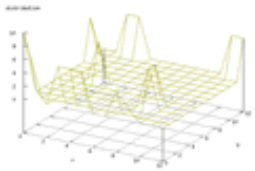


Figure 68 output layer from the Kohonen network that shows the distribution of the tenth of the vowels

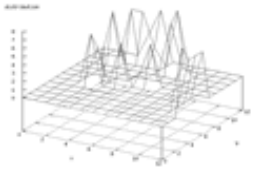


Figure 69 output layer from the Kohonen network that shows the distribution of the eleventh of the vowels

[toc | previous | next | up]

Chapter 4 - Back Propagation Network

This chapter describes the Back Propagation part of this report. The result of the program described in this chapter can be found in the next chapter. Here a quick description of the Back Propagation Algorithm can be found as well as the choices regarding our network. The program is described as well as the capabilities of it. And this chapter ends with a short description of the data sets that has been chosen.

4.1 Introduction to the Back Propagation Algorithm

The term 'Back Propagate' means just what it says. To propagate some signal backwards through the network, in this case it is the error and the reason we do that is to train the network. Basically this chapter explains shortly what a neural network is and how it is trained.
[toc | previous | next | up]

The Neuron

This is the fundamental building block of any neural networks. This neuron is programmed to imitate the behaviour of the neurons in our brain. It can be thought of as a black box with many inputs and one output, the axon. On each input there is a weight which regulates how much that input is going to affect the output. Inside the black box, all the inputs are added together and when the sum exceeds a threshold value, the axon fires. This makes a pattern recognition unit that can be made to fire on a specific pattern, and it can be taught to learn specific patterns by changing the weights.
[toc | previous | next | up]

Feed Forward Networks

A *Feed Forward Network* is made from *Neurons* and they are arranged in layers. Usually models of three layers are used, one input layer (that does not contain any neurons, but all the input signals), a hidden layer and an output layer which both are made from Neurons. It is the last two layers that actually does anything. All of the neurons inputs are connected to the previous layers outputs. This structure is called a Feed Forward Network.

Forward Propagation is when you input the signal vector to the network through the input layer and then check what outputs are generated on the output layer. When the network is not trained, it can present almost anything on the outputs. It is the possibility to train this network that makes it worth while.
[toc | previous | next | up]

Back Propagation Network

This is a process to train a Feed Forward Network, it is called Back Propagation because that is in essence what is done. The error from the outputs are propagated backwards through the network so that the weights can be updated to recognise the chosen pattern. This of course requires that there is a wanted output from the input vector. This is also called *Supervised learning*.
[toc | previous | next | up]

4.2 Back Propagation Choices

The network that is used consists of only one hidden layer and the weights are initialized to a random number between minus and plus one, divided with the square root of the number of neurons in the same layer. As shown in the following formula.

$$\left[-\frac{1}{\sqrt{p}},+\frac{1}{\sqrt{p}}\right]$$

The momentum and learning rate was chosen to be a fixed value rather than an adaptive one. They were both tested in different intervals and the result of that testing is described in the next chapter. In the initializing phase the layer sizes are set depending on the input data set. Only the hidden layer size can be altered for a specific data set. As the activation function the hyperbolic tangent function was used, because it makes it easy to compute the local gradient (*delta value*). The parameters A and B were both set to one in the beginning but was also tested in different intervals to see the effect they have on the performance. This formula shows how the parameters A and B were used in the network.

$$\varphi_j(v_j(n)) = a\tanh(bv_j(n)) \Rightarrow \delta_j(n) = \frac{b}{a}\left[d_j(n) - o_j(n)\right]\left[a - o_j(n)\right]\left[a + o_j(n)\right]$$

Also the choice to train the network after showing one input vector was chosen. The alternative is to accumulate the total error over all of the input vectors and then train the network. The training after each input vector should lead to quicker converging and it does seem to work.

[toc | previous | next | up]

4.3 Program Structure

The program consists of two source files and one header file. It is completely written in ANSI C and it is compileable under Linux. An accompanying *Makefile* has also been made to make the compiling process more simpler. It is important to notice that if the same *readfile.c* is used for both projects, a make clean must be issued before compiling the other program. This is done because the compile time option to normalize values must be set correctly. If this is not done, all data will be normalized in the other range, 0 to 1 instead of -1 to 1.

[toc | previous | next | up]

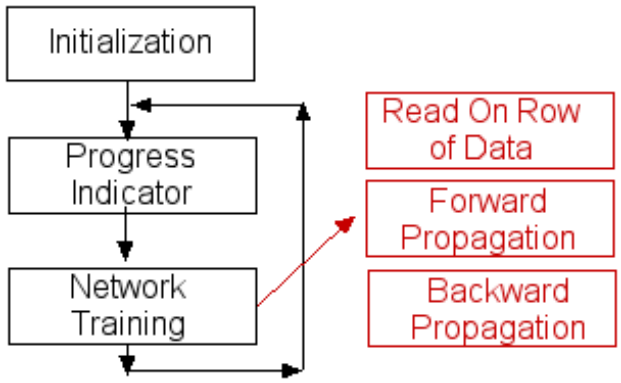
Readfile.c

This is the same file that was used for the Kohonen task. The only change is that the normalize function now normalizes the values between -1 and 1. This is because that is the range that is used in *backprop.c*.

[toc | previous | next | up]

Backprop.c

The program file can be divided into three main tasks, the first is initialization, the second is the progress indicator and the last is the actual network training. The main program loop starts with the progress indicator. The second part of the main loop does the actual training and it can be divided into three main tasks, read a row of data, present it to the network, and then train it using back propagation.



The first thing done in the initialization is to read the input data, this is done by calling a function in *readfile.c*. Then all data structures are allocated, the weights are initialized randomly between values that are relative to the size of the layer and all other data structures is cleared.

The progress indicator loops through the whole training set and computes an RMS error, and a correct percentage. The progress indicator loops through all rows of data and it is run twice for each program loop, once for the training set and once for the test set. In the end it presents a sequence of four values. The first value is the total RMS error, the second value is number of correct predictions, the third column is the total number of rows in the current set and the last column is the percentage of correct predictions. No training is done here, so the test set will not be contaminated.

The actual training is done in the second part of the main loop. Here three tasks are done. The first is to read new row of data, this is done by calling one of the readfile.c functions. The data is then presented to the network as two vectors, one for the input and one for the desired output. The second task is to forward propagate the input vector through the network to get the actual output. The actual output is then used in the third task, where it is subtracted from the desired output and the resulting value is back propagated through the network as *Delta Values*. When all the delta values are calculated the actual update of the weights are done.
[toc | previous | next | up]

Changeable Parameters

The program accepts command line parameters, and with them it is possible to specify most of the neural network and back propagation parameters. More parameters will be added as the program grows in complexity, but it is always possible to see a list of supported parameters by typing './backprop --help'.

The current program has the following parameters. *File* or data set, specify which data set that is used for training. *HiddenSize*, to specify the size of the hidden layer, if this is omitted or zero, the default size is chosen depending on the following formula.

$$hiddenLayerSize = \frac{inputLayerSize + outputLayerSize}{2}$$

LearnRate, specify the fixed learning rate. *Moment*, specify the fixed momentum term, set this to zero or omit to deactivate. *Parameter A and B*, specify the A and B parameters that are discussed in *Back Propagation Choices* above, if omitted the default value of 1 is set. Number of *iterations*, specify how many iterations that should be calculated, set this to zero or omit to run for infinity. There is also the possibility to add a prefix to the created filename, this is specified as *filePrefix*. The size of the input and output layers are updated automatically to adapt to the input data.
[toc | previous | next | up]

4.4 Chosen Data Sets

This is the different data sets that was chosen. For the easy choice we have chosen the Wine data set. For the hard choice we chose Proteins and the Vowel data set was used to test all parameters on. Because it is fairly simple and does have a lot of data to train on.
[toc | previous | next | up]

Wine

There not much to say about this set. The Kohonen revealed that it should be fairly simple so a small hidden layer was chosen. It was run several times with small variations in the parameters, and the final parameters were set to the following. The test set consists of 20% from the original data set, extracted in a random way. This means that the test set will vary from each time, making the scores from each run incomparable.

```
Input layer size: 13
Output layer size: 3
Training data set size: 142
Test data set size: 36
```
[toc | previous | next | up]

Proteins

This data set is quite big and consists of a specific test part. This makes it good to use as a reference how good the back propagation program really is. At first each row of the data was represented as seven numbers, but later it was decided to use 21 numbers where each number consists of 20 Boolean values as described in Chapter 3, Proteins. It is the 420 input model that is presented here.

```
Input layer size: 420
Output layer size: 3
Training data set size: 18,105
Test data set size: 3,520
```

[toc | previous | next | up]

Vowels

This data set was used to test various parameters on. At first this was the hard choice, but when more complex data sets were found, it was used for testing purposes. It has helped to understand the parameters a lot.

```
Input layer size: 10
Output layer size: 11
Training data set size: 528
Test data set size: 462
```

[toc | previous | next | up]

Chapter 5 - Back Propagation Network Results

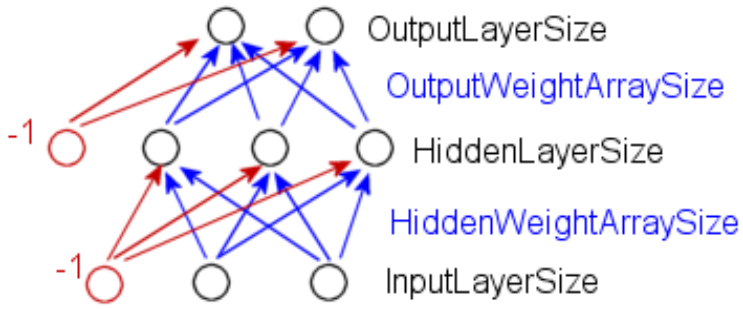
Since the Wine data set seemed to be very easy to categorize, it was the first data set presented to this back propagation program. The second set to be presented is the proteins data set. It is very complex and a nettalk like approach was needed. The third data set described is the Vowels data set. Because the Vowel data set seemed to be a slightly more complex data set than wine and was larger (more data to train on), it was trained many times with different parameter settings, so that each of the parameters influence could be monitored. All three data sets and the more complex Vowels data set are presented here with graphs and results. But first, some of the problems encountered will be described.

5.1 Initial Problems

Through this project some problems were encountered, a selected few are presented here. The reason for this is that it gives some insight of what was achieved, and maybe to prevent them from arising next time.
[toc | previous | next | up]

Extra '-1' Neuron

The problem of implementing this feature, it not big, its just irritating. It involves a lot of small changes everywhere in the code and it is very important to place the +1 count in the right places, otherwise a segment fault or overwriting memory could be the result. A good way to handle this is probably to program the code with this in the mind and use separate constants for the size of the layers and weight arrays as shown in the picture below.



This would mean that for the example above that has 2 inputs and 2 outputs, the *InputLayerSize* is 2, the *HiddenLayerSize* is 3 and the *OutputLayerSize* is 2. The *HiddenWeightArraySize* (this is the weight array size for each neuron in the hidden layer) should be set to one more than the size of the input layer, namely 3. The *OutputWeightArraySize* should be set to the *HiddenLayerSize* plus one. Then all these constants should be used instead of the +1 counts that is used in the current program. This would increase the structure, understanding and scalability of the code.
[toc | previous | next | up]

How to Present the Result?

This is very similar to the problem in the Kohonen chapters. How should the data be presented to the user? Since the *gnuplot* program was quite familiar at this point, the actual presenting of the data was not the issue. The real issue was probably the way the data was generated. In the beginning a redirect to a file was used quite commonly. Each programmer had their own format of the files and even if the files still exist there is very little comment on each file and the reason why it was computed was probably not remembered more than a couple of hours after it was generated. This has made a lot of information useless. Instead maybe it is a good idea to beforehand decide what structure and what data is to be stored

and then store it in different directories. With each directory there should be an information file, that describes when it was made and the changes since last time or the purpose of this generation. This, together with a small program that could automatically generate graphs from this directory would have made our time more well spent.
[toc | previous | next | up]

What is Considered to be a Correct Prediction?

Another completely different question is, between what values is considered right output? How much may the outputs deviate from the desired output and still be called right? In the resulting program, if the highest output is above zero and is the same as the desired then it is considered to be correct.
[toc | previous | next | up]

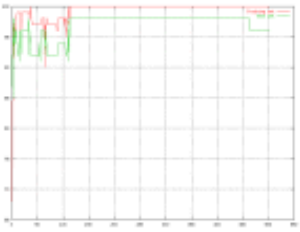
No Coding Structure

This was the main of our problems, and also the main cause for the late presentation of this report. The program was written by three different people, in three different ways. There was no initial discussion how the program was to be structured. The main reason for this was that parts of the group already was busy with another project. This mistake is probably very common in new projects, but should be avoided at all costs. The lack of collaboration in the beginning usually changes the outcome of the project in a not desired way.

In physical terms this problem revealed itself in countless number of hours debugging, because no one had the full knowledge of the code. Later though, when everyone had turned the code upside down for the tenth time. The general knowledge of the code was probably far greater and also quite good in the learning perspective, though it could also probably been accomplished by a thoroughly overhaul in the beginning of the project.
[toc | previous | next | up]

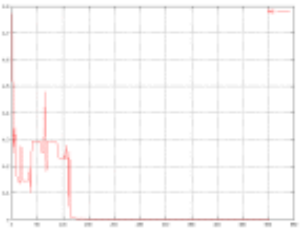
5.2 Wine, Graphs and Results

After implementing the back propagation algorithm to a neural network processing the wine data we could see some interesting behaviours. The most interesting thing to look at when evaluating a neural network is the possibility to classify the input correct, both for the training set and the test set. In the beginning of the training phase is the correctness fluctuating pretty much, but after (in this case) approximate 120 iterations it stabilizes.



The percent of correctness over time for the wine data set.

Another interesting thing to look at in a neural network is the RMS (root mean square) error for the nodes. The RMS error is a measurement for how correct the nodes are calculated compared to the correct output. The graph below shows that it takes a while before the error stops fluctuate and stabilizes. After it stops fluctuating it decreases very slowly.



The change of the RMS error over time.

It seems to have converged at 250 iterations or so, but is displaying very good results even at 100 iterations. This could probably be better if the learning rate and momentum were chosen differently.

Percentages

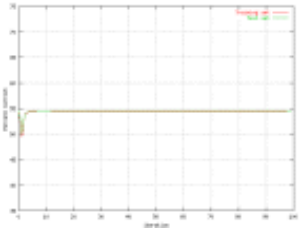
The average of the networks with 5 hidden units and up is for the test set, around 100%. This is with a random selection of the test set. The best percentage reached with the training set is 100.0% and it was performed by almost all networks from iteration 10 and up.
[toc | previous | next | up]

5.3 Proteins, Graphs and Results

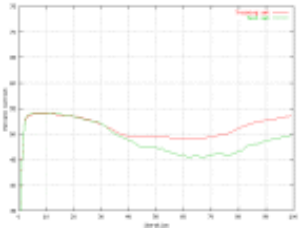
The secondary structures '_', 'e' and 'h' are in the graphs below labelled as their respective outputs, 0, 1 and 2.

Varying the Number of Hidden Neurons

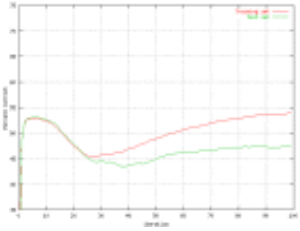
When a data set is first presented it is really hard to know how complex it is in number of neurons in the hidden layer. Especially if no previous experience is present. So this is the task of the following graphs, to show how many neurons that actually were needed for this data set.



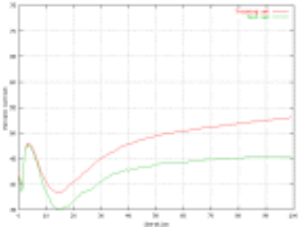
Percent correct predictions of the protein set with 10 hidden units.



Percent correct predictions of the protein set with 20 hidden units.



Percent correct predictions of the protein set with 30 hidden units.



Percent correct predictions of the protein set with 50 hidden units.

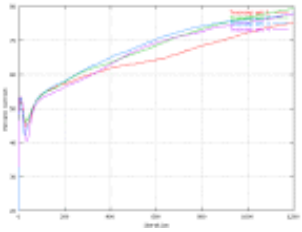
As can be seen the ten neuron graph is somewhat flat, it has converged this early. It was expected to be too small but the effect was not known. The 20 hidden neuron graph is a lot better than the first, but there is

still some 'slowness' about it if compared to the last two. The 30 and 50 neuron graphs seem to perform quite well. The 50 neuron graph seems to perform faster, but the computational time made most of the following graphs based on a hidden layer of 30 neurons.

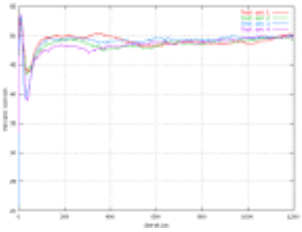
One really interesting thing about this is, that below 100 iterations the first network (the one with 10 hidden units) actually performs better than the other. Although in the long run it will probably not have a chance since the other outputs are still improving at the end of 100 iterations. One could also speculate about the generalization prediction of the first network. Would it really perform worse than the others if more data was fed into it? Hard to answer but the graphs above does suggest the possibility. The number of neurons has in spite of this been chosen to 30 for most of the other graphs.

Difference Between Runs

In order to actually see the effect of the randomness in the network, four batches on the same data set with the same parameters were run. This is the result.

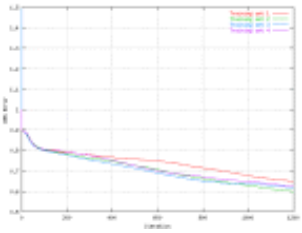


Percent correct predictions of the protein training set with 30 hidden units, generated four times.

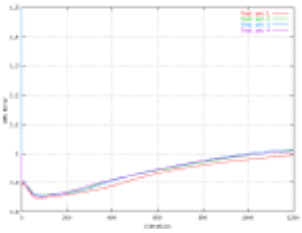


Percent correct predictions of the protein test set with 30 hidden units, generated four times.

As can be seen, they are not that different. This was a good and interesting thing to see though. Batch number one (the red) seems to deviate the most, but still shows the same charachteristics as the other. This means that the other graphs that show differences between different parameters could be taken quite literary since there was no real differences.



RMS Error of the protein training set with 30 hidden units, generated four times.

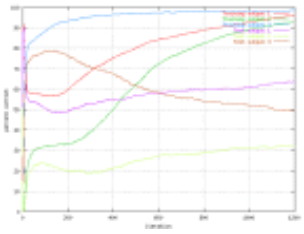


RMS Error of the protein test set with 30 hidden units, generated four times.

This is the RMS Error for the four runs, and they clearly show traits of similarity with the percentage graphs above. Yet again the red is the one most distingishable one and it performs slightly worse than the other here as well.

Neuron Output Specific Graphs

The Kohonen results presented in chapter 3 hinted that the secondary structure 'h' should be easy to recognize. The results below that show the percent of correct predictions in each of the secondary structures, has the ability to verify that.



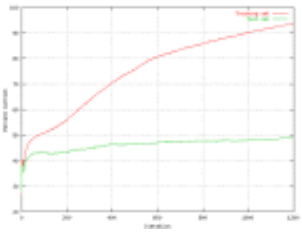
Percentage of correct predictions for each separate output of the protein set with 75 hidden units

This graph is with definite certainty the graph that shows the most in this report. Here we can se that the secondary structure 'h' (number 3 in the graph) is very easy to distinguish from the others and almost directly turn towards the 100% mark. The second proof that this set is easy it the test set curve. It starts dropping very early indicating that it is over trained. Maybe it is possible to make a run where the third output is not trained at every iteration, to make the match more evenly. Though that is beyond the resulting program at this point.

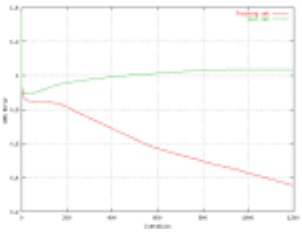
Also the first and second output (secondary structure '_' and 'e') seems to follow each other, this could suggests that the errors in one of them is predicted as the other. Then when the time progresses it learns how to distinguish them more accurately. As for the test sets for both of the outputs they seem not to display any hints of overtraining. The small dip in the beginning is probably just the result of the distribution changes that is going on. The rapid changes in one output might 'steal' predictions from the other outputs and result as dips in them.

Many Iterations

The Percentage correct and the RMS Error from the graph presented above is quite interesting to note.



Percent correct from a long iteration run of the protein set with 75 neurons in the hidden layer



RMS Error from a long iteration run of the protein set with 75 neurons in the hidden layer

The training error is is getting smaller all the time but the test error seem to stagnate after 800 to 1000 iterations. The test set percent correct is actually rising all the time and the question is when to stop training this network. Since the third output is quite over trained at 400 iterations when the other seem to present good results, it is tough to decide, but the prudent course of action is probably to stop around 350 iterations.

Percentages

The average of the networks with 30 hidden units and up is for the test set, around 47%. The best percentage reached with the training set is 93.7% and it was performed by the 75 hidden neuron network at iteration 1200.

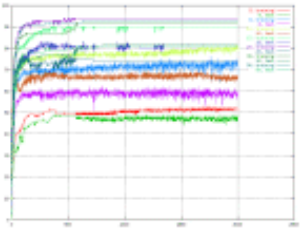
[toc | previous | next | up]

5.4 Vowels, Graphs and Results

The behaviours of the back propagation networks were interesting to study, because it showed expected and unexpected things, the variation of neurons needed in a well working network, the coupling between learning rate or momentum and the number of neurons in the hidden layer. When implementing the back propagation algorithm to analyze the vowel data set we wanted to see how changes in the network structure affected the behaviour of the network and the output result. To see the differences dependent on the variables (momentum, learning rate, number of neurons and parameters A and B) we plotted them out. Only one variable was changed at a time.
[toc | previous | next | up]

Number of Neurons

This test was done from as few as 5 neurons up to 75 neurons in different intervals (5, 8, 10, 15, 20, 30, 50, and 75). Out of this graph we could see that the 20 neuron network gave us the best results.

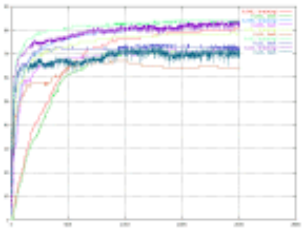


The percent of correctness over time for different number of neurons in the hidden layer.

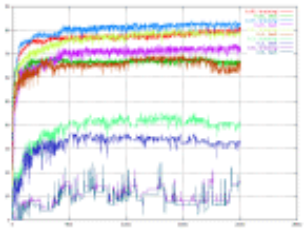
As can be seen by this, over the 15 neuron limit there was no noticeable increase or decrease in performance. It also shows that the fewer neurons that was tried the bigger was the fluctuation. It would suggest that a choice of 15 neurons is quite adequate for this data set. The change in other parameters could also influence some, so it is to early to say.
[toc | previous | next | up]

Learning Rate

For learning rate we tried a wide range of values, from 0.001 to 0.5. One thing that was speculated about was the influence of the learning rate with different sized data sets since the choice to train after each element of the data set was made.



The percent of correctness over time for different values for the learning rates.



The percent of correctness over time for different values for the learning rates.

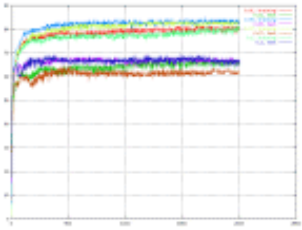
It was seen by these graphs that the fluctuation was connected to the size of the learning rate, the larger learning rate, the larger fluctuation. The fluctuation is also larger for the test set than for the the training set. The really big learning rates are not useful at all because of the low correctness and the huge

fluctuation. On the other hand, if the learning rate is very small (under 0.005) it takes to long time to train the network.

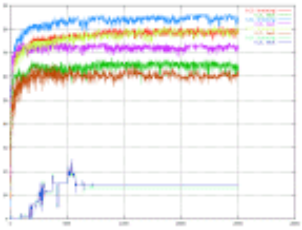
There was no real way to prove the learning rate behaviour on different sized data sets with just this information, but it is discussed in chapter 6.
[toc | previous | next | up]

Momentum

Momentum is an important thing in the neural network because it minimizes the chances to get stuck in a local minimum. Therefore the interest in how the momentum affects the behaviour of the output was big. We tried to find what would happen if it was way to large.



The percent of correctness over time for different values of the momentum.

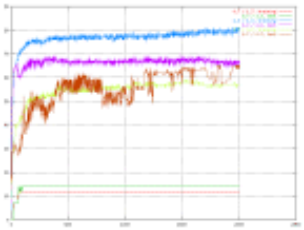


The percent of correctness over time for different values of the momentum.

When it was set to 0.9, really strange behaviour was observed. It was also observed that the fluctuation was related to the size of the momentum. The bigger momentum, the bigger fluctuation, except then the momentum was way to large. The best result was when the momentum was between 0.5 and 0.05.
[toc | previous | next | up]

Parameters A and B

Some of the less interesting changeable variables are the A and B parameters in the activation function. These are the parameters that are presented in chapter 4.

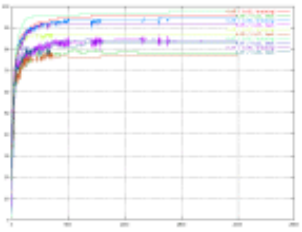


The percent of correctness over time for different values of the A and B constant.

When changing these some of the output was not even possible to use in any application. Either because the correctness of the output was very low or because it fluctuated too much. What is done is change the effect that the previous output has on the neuron and also the current output. For extreme values this would result in a very sensitive or a completely unsensitive network.
[toc | previous | next | up]

Optimized Network

From the previous results a mix of the best values was used to generate the following graphs. This is a way to get the best result possible without having adaptive momentum and/or learning rate. In the same time it gave us a possibility to see the behaviour dependent on the change in the different variables.

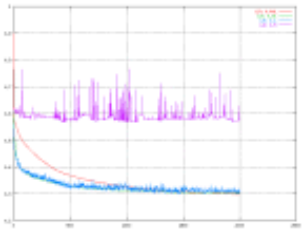


The percent of correctness over time for the optimized network using different values of the learning rate and momentum.

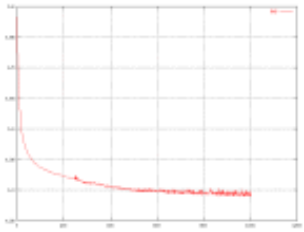
From this graph the following could be seen. The difference seems to change quite a lot when the different values are tried, but the performance is quite satisfactory.
[toc | previous | next | up]

RMS Error

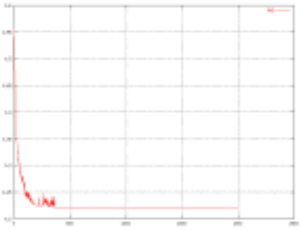
One other interesting thing, except the correctness of the output, is the RMS (root mean square) error. This shows how big the error between the output and the desired output is. This error is calculated here only on the training portion of the data set. In other words, the RMS error is almost always decreasing, even after the network is over trained.



The change in RMS over time for different values of the learning rate.



The change in RMS over time for the original network.



The change in RMS over time for the optimized network.

Most of the graphs have a point where there is a much larger amount of fluctuations than in the rest of the graph. This point appears usually just before the slope starts to flatten out.

Percentages

The average of the networks with 15 hidden units and up is for the test set, around 48%. This is with the

designated test set. The graphs above are made with a random selection of the test set. The best percentage reached with the training set is 97.0% and it was performed by the 20 hidden neuron network at iteration 300.

[toc | previous | next | up]

Chapter 6 - Conclusion

Many of the conclusions are embedded in chapter 3 and 5 which present the results of the Kohonen and the Back Propagation networks. Some chosen data is also presented here.

6.1 Kohonen's SOFM

The kohonen's SOFM is a quite simple and understandable method of a self organizing network. But it can really be complicated to implement and it really requires a lot of computational power to be able to compute anything.

Checking the Data Sets

This was actually a quite nice thing to do, it gave an extra advantage to tackle the problem. Especially the distribution of the input dimensions were good to have seen, so that errors could have been spotted right away. Also the search for question marks and missing commas in the data set made the data seem more familiar and made the handling of it somewhat easier.

Size of the Network

The size of the network is quite important. The size we chose was 13, and it was chosen because it seemed to be a nice compromise between the program execution time and the resolution of the graphs. Since the time to calculate a larger network increases in exponential manner, this was thought to be a good decision.

Calculations

There is a lot of calculations that need to be done in the Kohonen's SOFM. It is the calculation of who is the winning neuron that takes the most time. And as described in chapter 3, the use of the absolute value instead of the RMS calculation speeded up the program by more than seven times.

6.2 Back Propagation

There is a lot that can be done with a Back Propagation Network. But there are also a few things to keep in mind when programming them.

Number of Neurons

When choosing the number of neurons in the hidden layer it is better to have too many, if you have the computer power to run it, than to have to few. The number of neurons in the hidden layer is connected to the number in the output and input layer. It is also connected to the complexity of the input data. A good example of this is to compare the network for the wine (not so complex) with the network for the vowels (more complex).

Learning Rate Impact on Huge Data Sets

When comparing the two networks (wine and vowels) we realized that a good value of the learning rate is connected to the size of the network and the complexity of it. We can not say how big/little it should be under witch circumstances you have to test your way through it (or trust well trained feeling).

Since the network is trained on each data element rather than on the complete data set, it is important to

keep the learning rate small. Otherwise the data could fluctuate a lot and the results are jumping up and down because different elements tell it to go to a different direction. Based on the experience from this project a good rule of thumb for choosing the initial learning rate is 10/number of data elements in the data set. This will provide a quite accurate number that can be used for training.

Momentum

Because the momentum is used to overcome local minimums in the training of the network, it takes a very important part in the training. To have a big momentum makes it easier to overcome minimums and sometimes possible to overcome big local minimums. But, it makes the output fluctuate a lot and makes it harder to get a good result of the correctness. In the other hand using a small momentum makes it harder, if possible, to overcome local minimums. So what would be preferred is a momentum that is adaptive and in that way takes the best part from both worlds.

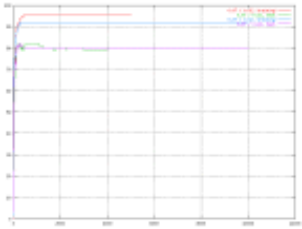
An adaptive momentum gains size when travelling downwards and is losing size when travelling upwards. This makes it more likely to overcome minimums.

Optimization

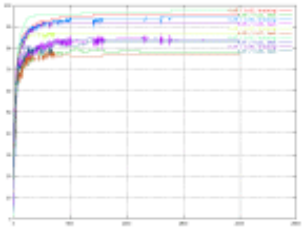
Optimization is something that almost every network has to go through to give good final result. One way of minimizing the effort is to let the learning rate, momentum and even the number of neurons be adaptive. This makes the network slower so it is an adjustment of time to do manual preparations and time to let the network train. The choice is connected to the size and complexity of the input data, the size of the network e.g. so the choice is very hard to predict but a way in the middle is to prefer.

Over Training

Over training a network is something that is not wanted, because it deteriorates the final results of a neural network. We tried to over train a network, just to see how it reacted. It was actually pretty hard to get some dramatically changes. We succeeded to get some of the networks over trained after many attempts. This shows that it is not always easy to over train a network, but it is very important that this does not happen.



A graph showing a network (the green) with a dip that can indicate an over trained network.



A graph showing some networks with early dips.

One really good example of overtraining is described in chapter 5, where the third output shows significant proof of being over trained.

Calculation Time

There must be about 2 or 3 whole days of computer processing time in these graphs presented in the report and at least an additional week processing time for testing. If this was 18 months ago it would be the double. It would be nice to know what results the next year students will produce, if they have access to better computers, or if it will be the same results.

One thing can really be said. The Kohonen's SOFM really take time to execute. The protein data set took

13 hours to compute and several other sets had to be cut down in order to being able to compute some output at all. To get this all running as fast as possible, there was a lot of optimizations done. Formulas were looked over to see if something could be calculated together and to use variables instead of functions in the really time critical part of the program.

Chapter 7 - Further work

It would be interesting to compare our result from the back propagation network with a identical network except to let the all weight changes after one iteration instead of, as today, change them one by one. Some other things to change are the momentum and learning rate, let them be adaptive. And then save all the data and plot it out to some graphs. In this way it would be possible to see how a network is evolving and changes over time. These changes/comparisons would give us a deeper look into the world of neural networks.

One important improvement of our program is the possibility to save the states of each neuron, i.e. all the data of the network. This would make it possible to train the network and later use the trained network in real application. If the program for training and testing the network is very dynamic, maybe because of that not so quick, it gives a good start to use it when developing real applications. To get all the states for each neuron and then easily implement it in a much faster version, for example in assembler or on a silicon chip.

It exists many applications where it would be very interesting to implement a neural network to improve the application. This makes it even possible to make new applications that were not possible before. One example is a system locating human bodies, it could be used in rescue missions, intruder alarms and even in some kind of interactive art systems. Another example is a system for searching and analysing data, for example it could give better search result on the web or finding life in space. Why not, build a system with a web camera and a projector, let the camera survey a room and show the people, just the people, moving around in the room with the projector. This could be an amusing and fascinating contribution to a dreary environment. A further project with a neural network in the central system could be a camera based human body interface to a computer. This would give some very interesting ways of controlling things, for example a robot or computer game. The number and types of applications possible to use an interface like that is enormous.

Chapter 8 - Glossary

42
: *The purpose of life.*

Back propagate
: *A way of processing data in a network, going from the output to the input.*

Delta Value
: *A local gradient between the actual output and the desired output in a neuron. See Chapter 4.2 and 4.3 for a more detailed description.*

Forward propagate
: *A way of processing data in a network, going from the input to the output.*

Gnuplot
: *A simple console based program for plotting graphs under both Linux and Windows.*

Hidden layer
: *The layer not visible from outside the network.*

Input layer
: *The input layer of neural network, it does not contain any active neurons.*

Kohonen's SOFM
: *A specific type of SOFM. See Chapter 2.1 for a more detailed description.*

Lambda
: *A neighbourhood function, that computes a value proportional to the distance. The value is large at small distances and small at far distances.*

Layer
: *A part of a network of neurons.*

Learning rate
: *A parameter to decide how quick the network should be able to learn.*

Momentum
: *A parameter used to overcome local minima.*

Network
: *In this report a system of neurons.*

Neuron
: *The building blocks containing all intelligent in the brain. See Chapter 4.1 for a more detailed description.*

Output layer
: *The layer of neurons that gives the output data.*

Overtraining

A behaviour that accurse if the network is trained to much. See Chapter 6.2 for a more detailed description.

Processor cache

An internal very fast and expensive memory in a processor.

RMS error

A way of calculating a type of mean error.

RISC-computer architecture

A type of processor architecture, the one used in PC's.

Sigma

A function used in the lambda neighbourhood function and it is an approximation of the normal distribution.

SOFM

Self organization feature maps. See Chapter 2.1 for a more detailed description.

Supervised Learning

A type of training when an input and the correct output are given to a network.

[toc | up]