A Study of Process Modelling using Artificial Neural Networks

Roland Orre and Anders Lansner

SANS – Studies of Artificial Neural Systems
NADA – Dept. of Numerical Analysis and Computing Science
Royal Institute of Technology, S–100 44 Stockholm, Sweden

Abstract
We model a part of a process in pulp to paper production using feed forward connected neural networks. A set of parameters related to paper quality is predicted from a set of process values. The predicted values are results from laboratory experiments which are time consuming. We check for irrelevant inputs and we manage with training sets that are considered small. The output vector is separated into single values which are predicted on different architectures adapted to each output. A strategy that continuously adapts the process model seems to be useful. In this work the backprop learning algorithm been used.

1 INTRODUCTION

The general problem is to model a process $\boldsymbol{P}$ as vector transformations. The actual process that is modelled has as inputs a known parameter vector $\boldsymbol{x}$, an unknown parameter vector $\boldsymbol{y}$ and a random disturbance vector $\boldsymbol{d}$. The process' output vector $\boldsymbol{o}$ is thus

$$\boldsymbol{o} = \boldsymbol{P}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{d})$$

The process $\boldsymbol{P}$ may have a memory and be a function also of its parameters and output history but for simplicity we assume that it is not. The problem is to make a model that can predict the process' output vector $\boldsymbol{o}$ when only the vector $\boldsymbol{x}$ is given. The actual vector $\boldsymbol{x}$ is however, also unknown. Due to physcial measurement errors, manual input errors and approximation errors the input vector $\boldsymbol{x}$ is also an estimate. The modeled vector output thus becomes

$$\hat{\boldsymbol{o}} = \hat{\boldsymbol{P}}_{\boldsymbol{model}}(\hat{\boldsymbol{x}})$$

The specific problem in this study was to investigate how some measures of pulp quality could be predicted. These values are results from laboratory experiments and should be predicted from a set of input parameters that are obtained from quick automatic experiments and some additional information about the process. The laboratory measurements that are predicted are *csf, density, elongation, tear* and *tensile* with the emphasis on *tear* and *tensile.*

In the following will a process *signal* refer to either a process *input* or *output* value.

2 METHOD

The computational method developed for this study uses *feed-forward artificial neural networks* with the *error back-propagation* learning algorithm. This algorithm, which is most often called just *back-propagation*, will not be described here. There are several text books available on this subject. A theoretical textbook that describes this and other neural network principles is for instance “Introduction to the Theory of Neural Computation” [HKP91]. General introductions to the field are [MRtPRG86], [RMtPRG86], [MRtPRG88], [Neu90], [Nie90] and [NI91].

The *back-propagation* learning method is just one of several artificial neural network algorithms which are available today. The reasons for selecting this method here is that it is both well understood and straightforward to use. It is also, for these reasons, the method that has become the most used neural network method in the industry today.

2.1 Tools and Environment

The tool that is used for the back-propagation network learning and execution is “Aspirin/ MIGRAINE” release V5.0 [Lei91]. This is a set of programming tools for neural network simulations which has been developed by an internally funded research effort at the MITRE Corporation. These tools include a compiler for a neural network description language “Aspirin” and a user interface “MIGRAINE”. The software is written in C and the Aspirin compiler generates C code as output that is automatically compiled and linked together with the MIGRAINE interface to create an application program that runs the specific network architecture(s) that is described in the Aspirin file.

The MITRE corporation decided to release this software March 1988 from V4.0 free of charge, publicy available. It may be freely used and modified for research and development purposes. The only requirement is that it is briefly acknowledged in any research paper or other publication where this software has made a significant contribution. If the software will be used for commercial gain the MITRE Corporation has to be contacted for further conditions of use.

The Aspirin/MIGRAINE software package is easily installed and runs on most UNIX systems. Tested systems are for instance Apollo, Convex, DecStation, IBM RS/6000, 486/386(System V), HP 9000, NeXT, News, Silicon Graphics and Sun. It also supports some coprocessor boards that can be added to a UNIX host, at the current writing these include i860 boards and iWarp Cells. During the current writing it was announced that Aspirin V6.0 is available.

For diagram generation the “GNUPLOT” V3.2 interactive plotting program has been used. This package is also free of charge and installs on all UNIX systems plus some others.

2.2 Method Development

The investigations performed in this report have resulted in the development of a method for preprocessing of data, doing an iterative search through different network architechtures and the removal of non significant network inputs. This code is written in the language scheme ([WC91]) and has been made so that this is the only interface needed to run the learning and prediction through MIGRAINE on a UNIX work station.

2.3 Specific Settings Used

To make it easier to repeat the experiments that have been done in this study, a few words about some specific control parameters, especially to the Aspirin/Migraine interface. A typical command line for learning would be “command -I 1000000 -l -d df -F net-name -t 10000 itok 0.05 -s 10000 -a 0.05 -F net-name save-name”. For other parameters like inertia, default values have been used.

command This is the name of the compiled and linked MIGRAINE code for a specific declaration file. For a two layer net with seven inputs, ten hidden and one output this will, with the syntax used here, be “..bp/7_10_1”.

-I 1000000 The upper limit for the number of learning iterations to run. This is not the same as *epochs*, which can be obtained by dividing the number of iterations with the size of the training set.

-t 10000 itok 0.05 Tells how often the test network will be tested and how many samples that have to be correct to see if it has reached the error limit. The "itok" has typically been set to the numbers of vectors in the training set. The error limit has typically been set to 0.05 when one want to learn the train set as good as possible within the iteration limit. For the continuous learning case the error limit has been set to a specific limit for each item to save time.

-a 0.05 The learning rate, "alpha", has typically been set lower or equal to the error limit. Alpha has typically been started on 0.1 for a limited number of iterations, like 100000, then decreased to 0.05.

3 DATA

3.1 Input and Output Data

The data that has been processed in this study is a set of vectors consisting of process input parameters and automatically and manually measured process outputs. The manually measured output values being predicted are *csf, density, elongation, tear* and *tensile.* Nine process inputs, most of them automatically measured, have been used. For proprietary reasons these inputs are called *par1* – *par9* here. The names and ranges of the input and output values used are shown in table 1.

parameter	range		
csf	250	···	449
density	523	···	665
elongation	3.9	···	6.3
tear	11.6	···	16.5
tensile	60.8	···	84.7
par1	0	···	6
par2	0	···	100
par3	0	···	100
par4	0	···	100
par5	29.2	···	45.9
par6	6.3	···	14.1
par7	22.2	···	30.9
par8	255	···	367
par9	51	···	75

Table 1: The set of signals used listed with their active ranges. For two of the input values, *par2* and *par6*, there were many values missing. Of this reason these two signals were skipped from the input data set.

3.2 Data Selection and Filtering

As the network learning algorithm will do a regression analysis of training data to create an input output mapping with as little error as possible, we need training data of as good quality as possible. To increase the quality some filtering of the data has to be done. There may be several reasons for filtering out data. Data may simply be missing, for instance due to some missed manual input, or data may be erroneous. In this study there has not been any possibility for any consistency check, *i.e.* , checking that each value in an input vector is reasonable considering all the other values. The only thing we could do was to remove data that obviously was outside some extreme limits. The limits for the acceptable range for each item was decided from histograms showing the distribution of each parameter. These histograms are shown in appendix A.

The total amount of data samples that have been available in this investigation is around 200 vectors from one process and around 50 vectors from another process. The set of 50 vectors was considered too small to make use of at the present time. Also the set of 200 vectors is a small set for some of the more complex nets that have been tested, why one should be cautious in drawing conclusions too far. After filtering the set of 200 vectors, by removing those vectors that obviously contained input values that were outside reasonable limits, we ended up with 110 vectors of good quality that could be used for training and testing. This may seem to be very few, but it was considered more important to have a good quality on those that were left than to risk learning of erroneous vectors. In the first experiments we tried to filter the data set selectively for each output parameter to not need to remove more samples than necessary but later on we removed all sample vectors containing some bad value.

3.3 Preprocessing and Scaling

In the preprocessing of data that is done before presenting it to a network, the first step consists of filtering out vectors where some of the signal values lie outside some accepted range. If too many values of a signal lie outside the accepted range then the whole column for that signal is removed and will not be used as input or output to the network. The data is then transformed to a form that is suitable as inputs to a network. In this study all signals have been construed as continuous valued data. The simpliest way to represent continuous valued variables as, for instance, *csf* above, is to let the analogue output value of a unit be proportional to the actual value. In that case only one neuron is needed to represent each value. The output value of a neuronal unit is normally, of practical reasons, limited. It is therefore necessary to scale the parameter values to fit within the limited output range. The scale has been chosen so that the parameter range is mapped to fit within an almost linear piece $(0.25 \cdots 0.75)$ of the sigmoid output function of the units (figure 1). There are not the same limitations of the input signals, but for practical reasons all inputs have been scaled to lie within the same range as the outputs, i.e., $0.25 \cdots 0.75$.

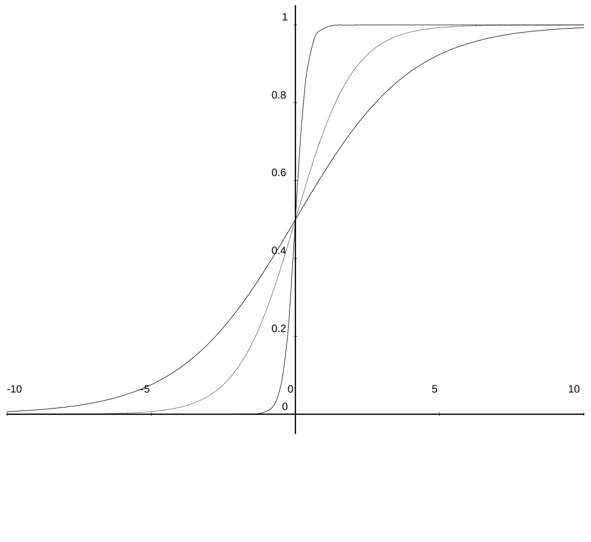

Figure 1: A sigmoid function, in this case $(\exp(-kx) + 1)^{-1}$, with three different k. This kind of squashing function is the most commonly used one to make the output of a neural unit non-linear. The output of a unit will show an almost linear response for a small input range, otherwise the output is limited, often to the interval 0. . .1.

4 RESULTS

To be able to classify how well a network predicts a parameter one needs some measure of quality. In the following we have used the average error (merr) and the standard deviation (sigma) for the difference between actual and predicted signal. The standard deviation and the average error are both expressed in percent of the dynamic range for the predicted signal. The average error is good measurement to see if there is some some offset or systematic error in the predicted values. The standard deviation is a well established error measurement, the formula used for its calculation here is

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \frac{\sum_{i=1}^{n} x_i}{n})^2}{n-1}}$$

4.1 Partitioning of Input Data

Before training a network, the data set has to be splitted into a training part and a test part. This has to be done to be able to measure the *generalization* capability of the network, *i.e.* , its performance on data it has not been trained on.

For some experiments the data has been splitted in four different ways, *first, last, random* and *split.* These are illustrated in figure 2. The partition “First 75%”, for instance, says that 75% of the total amount of data has been picked consecutively, starting from the oldest sample. “Last 75%” is similar but goes backwards starting from the most recent sample. For “Random 75%” the samples have been picked according to a rectangular distribution from the whole interval. The “Split 3-1” separates the data systematically, three to the train set, one to the test set and so on. These four ways of partitioning data are called “first”, “last”, “rand” and “split” in table 2 and “fi75”, “la75”, “rn75” and “sp31” when they occur in most other tables, like in appendix B. A column head like *csf.fi75*, tells us that this is the output signal *csf* predicted with the partition “First 75%”.

Too see what difference the type of data partition can make, look at table 2, which has been extracted from the tables in appendix B. What is shown is the prediction quality averaged over all architectures for four different partitions of the data set. As the “last” partition shows the highest error rate it indicates that the first 25% of the data set would behave different than the rest. This could be interpreted as that something has happened with the process during the first sampling period. Further, if we assume that the output signals and the input parameters would change slowly, but mainly caused by other factors than the input parameters, then the best guess would probably be that the new value is just close to the old one. In such a case it could be expected that the systematic “split 3-1” partitioning would perform best. For *csf* and *tensile* we actually get the best performance with the “split” partition. The difference may, however, not necessary be significant for a conclusion. For industrial process signals are, of course, predictions about the closest future the most interresting to make. Of this reason, most of the prediction experiments have been done by using the “First 75%” partition.

4.2 Finding a Good architechture

The network algorithm makes the input output mapping by adapting the weights of the network to the training data. If certain criteria of the training set are fulfilled, like that each input vector corresponds to a unique output vector or value, then the mapping function of the network can be done with arbitrary precision if we choose a network which is just complex enough. If the network get too complex, however, it will probably *generalize* badly. This problem can be

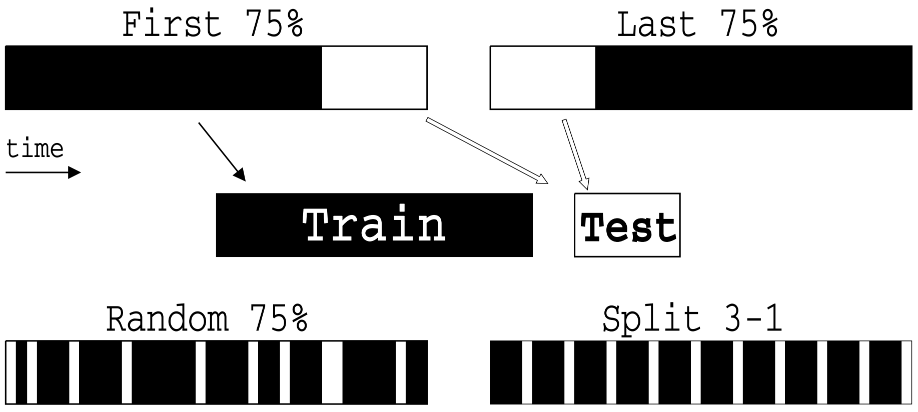

Figure 2: Four different partition types, which have been used. In all four cases have 75 % of the input data been used as training data.

	first	last	rand	split
csf	14.1	37.8	14.3	11.2
density	12.9	15.3	9.9	14.4
elongation	8.0	12.0	10.9	9.6
tear	14.0	26.6	12.3	13.9
tensile	15.7	16.5	13.2	12.9

Table 2: Prediction performance for different partititons, averaged over all tested architechtures. As “last” shows the largest error for all outputs it indicates some change in the process during the first sampling period.

thought of as similar to finding a polynomial approximation to some function. If the polynomial degree is too high it will probably approximate the actual function badly due to noise in the regression input data.

If there is a big amount of training data available the network can be allowed to be rather complex. In the case studied here we tired five different architectures from a simple one layer, i.e., where the output is just a linear combination of its inputs, to a three layer architechture. Three different two layer networks have been tested where the three layer network had about the same complexity as the two layer network with 20 units in its hidden layer. See figure 3 for a clarification of this. The number of inputs have varied but the first hidden layer has had 10, 20 and 30 units. The three layer net had 10 units in both hidden layers. The syntax used in result tables is like: “**7_20_1**” or “**7_10_10_1**” where the first means 7 inputs, 20 in hidden and 1 output and the second is 7 inputs, 10 hidden, 10 hidden and 1 output.

In figure 4 below, we can see how two different networks performs on a training and a test set of the parameter *elongation* (stretch in diagram). A one layer network, which corresponds to a linear system of equations, and a two layer network with 30 units in the hidden layer.

Performance on different architechtures

When doing a prediction of some output parameter we normally don’t know in advance how complex the problem is. It may be linear in the sense that each output can be written as a linear combination of its inputs or it may be so complex that it needs several hidden layers with a lot of units. To get a picture of how complex the problem is for each output parameter each of them has been been run separately on several different architechtures (Appendix B). Table 3 shows the average of the standard deviations of the prediction error over different partitions for the output parameters *csf, density, elongation, tear,* and *tensile* when these are learned on

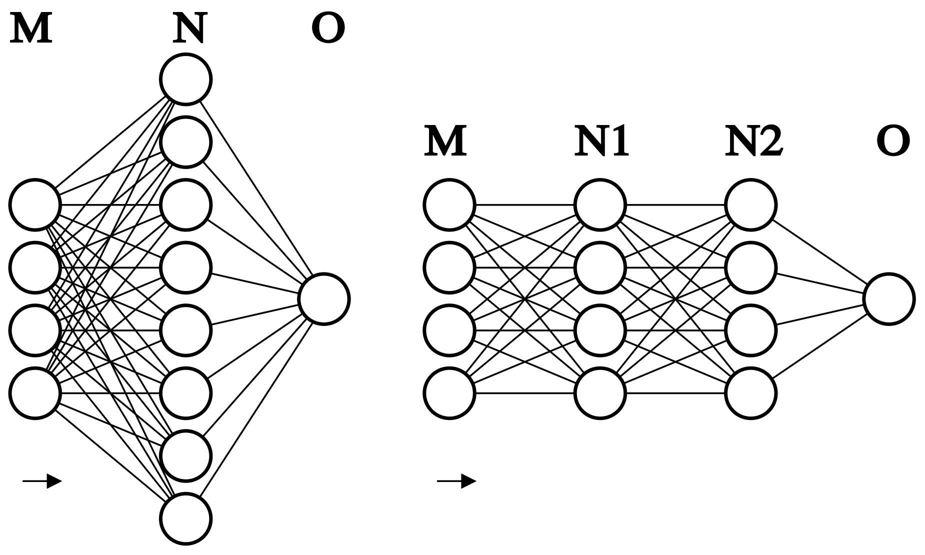

Figure 3: Two networks with about the same complexity. With the syntax used in this document to denote network architectures they would be described as M_N_O (4_8_1) and M_N1_N2_O (4_4_4_1) respectively. The number of weights in the left one is $((M+1)N)+(N+1)O=49$ and in the right one $(M+1)N1+(N1+1)N2+(N2+1)O=45$.

different architechtures. From this table it may be a preliminary proposal that *csf* and *tear* are best predicted using a linear combination of its inputs when *elongation* seems to do best with a two layer net and *density* and *tensile* seems to make best use of the properties of the three layer network.

TRAIN set	7_1	7_10_1	7_20_1	7_30_1	7_10_10_1
csf	10.0	6.8	6.8	6.7	6.9
density	11.2	9.4	9.2	9.3	9.7
elongation	12.6	7.8	7.9	8.0	8.4
tear	12.9	10.4	10.2	9.8	9.6
tensile	12.1	9.5	9.5	9.6	9.9
TEST set					
csf	11.7	13.4	21.9	20.9	18.2
density	13.8	13.9	13.6	13.1	11.3
elongation	11.4	9.8	9.6	9.5	10.4
tear	13.4	17.5	19.5	19.2	14.3
tensile	14.4	13.7	16.0	15.7	13.0

Table 3: A more complex architechture generally performs better on the training set but may cause a bad performance on the test set. Here it seems like *csf* and *tear* just need one layer but *elongation* needs two and *density* and *tensile* needs three layers.

4.3 Requirements on Cpu-time

The *backpropagation* learning algorithm is a numerical method that is similar to *gradient descent*, *i.e.* , a way to find an optimal solution (a minimum), versus some set of adjustable parameters, to some function under some boundary conditions. In the neural network case the adjustable

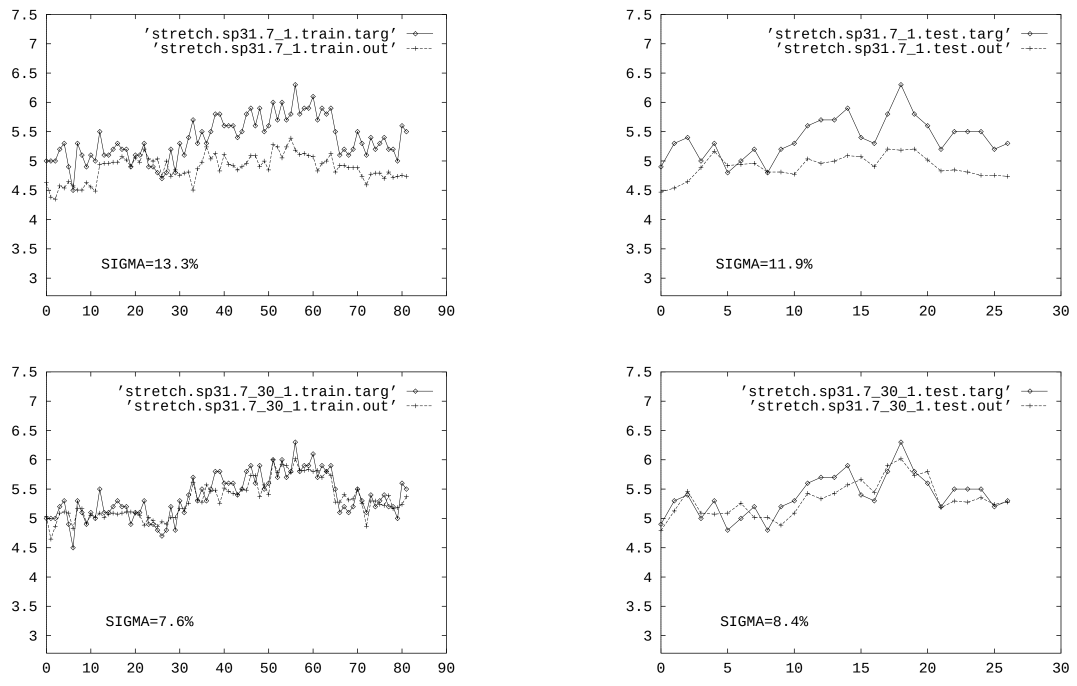

Figure 4: On the left is shown how two networks have adapted to a 75% training set for *elongation* (stretch in diag). The upper diagrams show a one layer, *i.e.* , a linear combination, network and the lower diagrams show a two layer network. The standard deviations for the errors are shown in the diagrams. On the right we see the prediction performance for the same nets on a 25% training set.

parameters are the weights and the function to minimize is the output error. The time to train a network is not a simple function of the number of weights, neither is it a simple function of the number of training samples. If there was just one optimal solution then the time to find the optimum could grow faster than polynomial in the number of weights. There may often, however, in a large network be multiple solutions that are equally good. Generally speaking, it takes a longer time to train a more complicated network. In the experiments performed in this study we have run the training until a specified error limit was reached, or until a maximum number of iterations has been performed. In the tables in, for instance, appendix B the table include the value “Kepo” that is the number of thousands of training epochs. One epoch is when all training samples have been used once. In figure 5 it is shown how “Kepo/hour” decreases when the number of weights increases. When the experiments have been run the iteration limit has been set so that the most complex net should come no further, which means that the less complex nets have run many more iterations than necessary. The nets that have been used here need no more than an hour to to retrain completely on a reasonable fast personal computer or a workstation. It should be noticed that a larger training set would not make the training time significantly longer. If the set of training samples is too small compared with the network size, then the training time would be rather small also for a complex net but with a bad generalization capability.

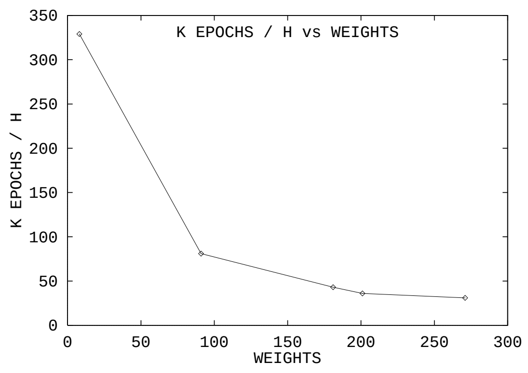

Figure 5: Number of K epochs, i.e., number of thousands epochs per hour, plotted versus the number of weights for five of the networks that have been used in this study. The timing which has been plotted here is for the networks 7_1, 7_10_1, 7_20_1, 7_10_10_1 and 7_30_1 with *stretch* as the output. It was measured on a DecStation 5100.

4.4 Selection of Significant Inputs

When we learn a function by examples it may be so that an output is much dependent on some of the inputs, but it can also be so that the actual function that we try to predict is not at all dependent on a specific input parameter. In the latter case when we adapt the network to the training data the algorithm will of course try to adapt the network also to those variables that the output is actually not a function of. When this network then is used on test data, these non significant inputs may act just as random disturbances.

If plotting just one variable versus another we can visually get a feeling that one input is clearly dependent on some other input as in figure 6 or that they are mostly independent as seems to be the case in figure 7. This is however, not easy to do visually when dealing with multi dimensional data.

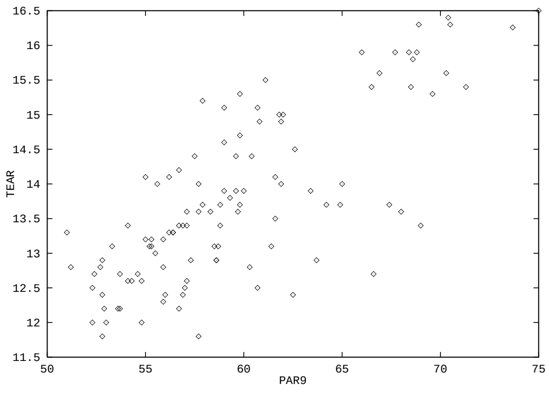

Figure 6: These two variables show a clear correlation.

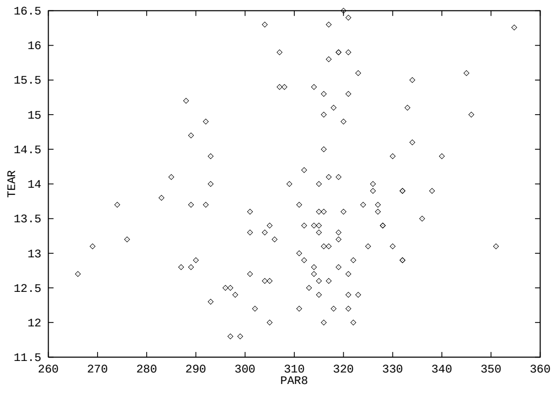

Figure 7: These two variables seem to be uncorrelated.

Input sensitivity test

The method used here to find the input parameters that have the greatest impact on an output value is to first train a network with all input parameters available and then test how much the prediction capability of the network is affected when each of the input parameters is “removed”. The removal of the input has been done here by replacing the input with its average value. The tables in appendix C show how each of the parameters have been predicted from all the others.

The tables are arranged from greatest impact to smallest. The table 4 below can also be found in appendix C. These two tables show how the prediction performance on the test sets for *tear* and *tensile* is affected when each of the other parameters are replaced with its average. A "—" signifies that none of the inputs were missing. An interpretation is that *tear* is mostly affected by the input *par3*, and *tensile* by the parameter *par4*. Another interpretation may be that *par1, csf, par7* and *density* may not be significant for *tear*. In the same way *density, tear* and *par3* appear to have a low impact on *tensile*. It may be noted that the error difference when an input is "removed" and when it is available in most cases here is to small to give an answer with high significance for the small input data sets that have bee used.

tear	%sigma	%merr
par3	22.3	6.1
par9	20.0	0.7
elongation	19.2	-1.0
par4	17.7	-0.9
par8	17.4	-1.1
par5	17.3	1.3
tensile	17.2	4.3
—	17.1	-0.3
par1	16.5	-1.9
csf	16.5	-0.3
par7	16.2	6.2
density	16.2	0.1

tensile	%sigma	%merr
par4	20.7	-19.2
csf	16.1	-7.6
elongation	15.7	-7.8
par9	15.4	-7.5
par5	15.4	-7.1
par1	15.3	-6.9
par7	15.3	-5.1
par8	15.2	-7.5
—	15.2	-7.5
density	15.1	-7.1
tear	14.2	-12.7
par3	13.8	1.3

Table 4: From appendix B. Prediction performance on the test set. The properties *par1, csf, par7 and density* seem, from this test set, somewhat less important for *tear* than pulp type and fiber length. Also for *tensile* it seems as pulp type has a high significance. The error differences here are probably to small to tell that some of these input signals are irrelevant for *tear* and *tensile*.

Removal of non significant inputs

The tables in appendix D show how the performance is affected when the inputs that in appendix C caused an improvement on the results of the test set when they were removed from the train and test sets. In some cases they have been run on more than one architechture, both the architechture as in appendix C and some other architechture where they previously showed a good result.

Removal of Lab Data inputs

In the results shown in appendix E the strategy for removal of inputs have been a little different. The inputs here do not include those that are time consuming to measure, *i.e.* , *csf, density, elongation, tear* and *tensile*. Further, if some input showed to be insignificant on the training it was removed. The tests have been performed as in appendix C.

These tests indicate that the pulp type, indicated by *par2, par3 and par4* is significant for both *tear* and *tensile*. We can see that the *par1* seem to bee of some significance for *csf, elongation, par9* and *tensile*.

4.5 Different Learning Strategies

Instant prediction

The prediction being performed in the previous examples is based on the instant input values only. The assumption is that the predicted process has no memory, *i.e.* , an output is a function of the momentarily input values only and does not depend on the history of an input or output.

As a simple example of a process that can not be predicted this way. Imagine that the process we model is the output voltage from a capacitor being charged with a current proportional to the input voltage. In such a case nothing can be said about the output values by just looking at the instantaneous input values.

Continuous learning

If the process which is modelled would change its characteristics over the time, then it could be expected that we could improve the prediction result by making the network a one step predictor. The idea is that we let the network learn the behaviour of the process during a limited time span. The process outputs are then predicted from the next input vector being measured. When the actual output vector, which was predicted, becomes available, this can be used as new training data for the network. The oldest sample is then thrown and replaced with the newest one. The figure 8 below shows how *tear* is predicted with a one layer and a two layer network using *par1, par8, par5, par9, , par4* and *par3* as inputs. The performance according to the standard deviation is about equally good with 12.0% for the **7_1** net and 12.3% for the **7_10_1** net. The latter net does, however, show a better error distribution with no offset why that one would be to prefer. In figure 9 it is shown how many learning iterations that has to be done for each new sample to reach the same error performance that was obtained on the first traning set. With "iterations" is here meant the number of separate training patterns that has to presented for the network. A low number shows that the network could be quickly retrained within the goal to keep the error level low. A high value here indicates that the network could not learn the new sample within a limited number of iterations.

In appendix F the one step predictor performance for the continuous learning case is listed for *csf, density, elongation, tear* and *tensile.* The columns "%sigma" and "%merr" there, as before, stand for percent standard deviation and percentage mean error. There is also a column "maxerr" that shows the largest error as $predicted - target$ in actual units. Diagrams for error distribution use the range of the neuron outputs. There are also diagrams for the number of iterations needed to relearn the network after each sample. In most cases when the whished error limit could not be reached quickly, the number of iterations have reached their limit. This is also a nice result in that sense that a much lower threshold on the number of the limit iterations can be set without affecting the performance.

Continuous learning, Tear and Tensile

In appendices G and H we have studied the *tear* and *tensile* in particular. In this case more samples have been used than in the previous examples. The original data set size is the same but here only those sample vectors containing bad values that would affect *tear* and *tensile* have been removed in the preprocessing. We then ended up with 179 useful sample vectors. The training approach was here a little different from the previous section. The network has been trained until no significant improvement could be obtained on the test set. Then, for each step forward, the network has been retrained but it has not been allowed to retrain fully. The number of iterations has been limited to about 200000. The idea behind this is to produce a kind of

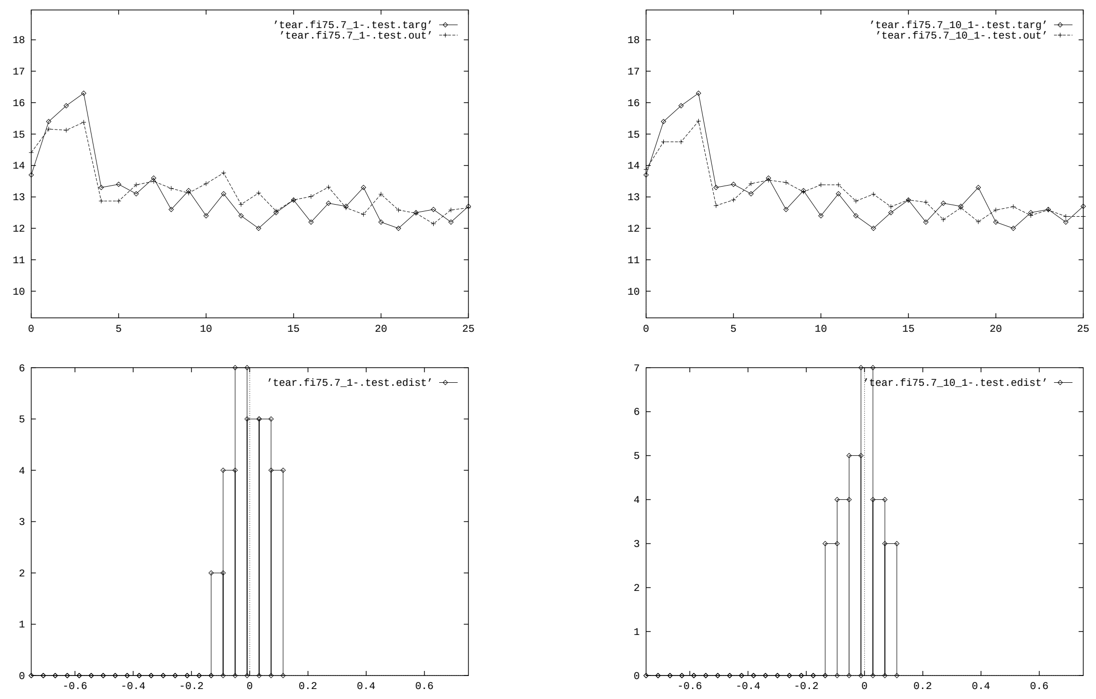

Figure 8: The property *tear* predicted with a one-layer (left) and a two-layer (right) net. They perform equally well but the two-layer net has a better error distribution with no offset, thus, the two-layer one would be preferred here.

inertia, i.e., a small variation in the input would not cause a drastic change of the network parameters.

The same length as in previous section , i.e., 82 samples has been used as the training set for prediction of the next sample. This means 46 % of the total set. We can see from the table in appendix G that the best result here is obtained from the linear, one layer network in the case with continuous learning. The *7_10_10_1* could probably have given a better result if it had been allowed to retrain fully as is indicated by the retrain iteration counts diagram.

When we look at the result for *tensile* in appendix H we see that we also here get the best result from the one layer network with continuous learning.

There is a great prediction failure between sample number 15 to 30 for all the multilayer networks where the one layer network performs rather well. This is probably an indication of

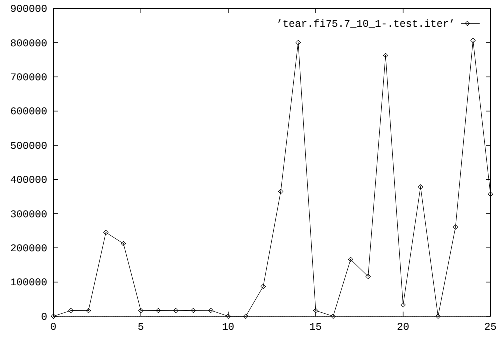

Figure 9: In the continuous learning case the number of learning iterations may vary widely for each new sample. A high iteration count here tells us that the whished error level not could be reached for a certain sample.

too few training samples. The *tensile* target is rather rugged as can be seen from the diagrams. The multilayer nets have tried to adapt to this shape. If the jaggy shape actually is there due to some distorsion or measurement errror then it could be expected that the more complex nets would try to adapt to this too quickly with too few training samples available.

Fixed learning, Tear and Tensile

To get a chance to compare the performance of the continuous learning case with the one where we have trained the net only on the beginning of the history for the same data as in the previous section, here 82 samples, these are shown in appendices I and J. The one layer network here seems to have no possibility to follow the tricky part between sample 15 to 30 that appear both in *tear* and *tensile.* There seems to be information about this, anyway, in the training part as all the multilayer nets react on this interval. Here one can see that the number of units in the hidden layer have a great influence on the capability to cope with the sample interval 15 to 30. One interresting result is of course that the three layer net **7_10_10_1** detects the interval perfectly but goes in the wrong direction both for *tear* and *tensile* in this interval.

Temporal prediction

The Aspirin tool which has been used here supports three forms of temporal coding. One is to have recurrent connections from delayed replicas of outputs that can be weighted in on any layer. The delay can be from one to four steps. Another form is to delay the inputs one or several steps and present these delayed inputs through weights for some layer. A third method is to do averaging over several input and present the average value through weights for the network. These methods can of course be combined. If one of these methods was going to be used here it is of course hard to say which method would be best. This would depend on the problem. The results here does not include any of these temporal codings. There are some problems with temporal coding that have not been solved. One problem is that the samples are not equidistant in time, thereby making it necessary to do some kind of extra preprocessing so that the samples become equidistant. One such way is to do linear or spline interpolation between the samples.

Limited history

In an implementation of this method the learning strategy will probably be some form of continuous learning. In such a case it may be important to address the problem of which size of the history that is preferrable to learn. The reason for this is that the process may change due to external environmental changes that are not measured. In such a case will the old history more act like disturbances.

4.6 Best Results

In table 5 below are the best results that were obtained in this study extracted. The table shows the signal name, the partition strategy, the learning strategy, the parameters standard deviation and its average error. The table does only include the outputs *csf, density, elongation, tear* and *tensile.* It may be noted that the table with best possible results, any partition, where the input information is restricted to quickly available parameters, shows better results than the table where the partition is restricted to the first one, without any restrictions on the input parameters. If this difference should be treated as being significant it is interresting, because, if the result depends on the partition then it would indicate that the prediction may be done better by, for instance, decreasing the size of the history etc.

The most interresting table may be the one that shows the continuous learning case. This is the case that would most correspond to the reality when doing prediction in real time, but the results may be possible to improve by, for instance, more samples over a shorter period, temporal and interval coding. See the section “FUTURE EXTENSIONS” for a discussion about this. The diagrams with test results for the continuous learning case are supplied in appendix F. There is also some statistics about the actual input values used in appendix K.

Best possible, process data only, any partition					
output	arch	part	sigma	merr	inputs
csf	7_1	sp31	6.8	-1.8	p1,p8,p5,p9,p7,p4,p3
density	7_10_10_1	la75	9.2	6.8	p1,p8,p5,p9,p7,p4,p3
elongation	7_20_1	fi75	7.1	8.2	p1,p8,p5,p9,p7,p4,p3
tear	7_1	rn75	11.7	-7.6	p1,p8,p5,p9,p7,p4,p3
tensile	7_10_10_1	la75	10.8	5.2	p1,p8,p5,p9,p7,p4,p3

Best with process data and lab data, first partition					
output	arch	part	sigma	merr	inputs
csf	8_10_1	fi75	9.0	7.9	p3,ten,p8,p7,den,p9,elo,p1
density	7_10_10_1	fi75	11.9	5.8	p3,tea,csf,p4,p7,p5,pl
elongation	7_20_1	fi75	7.1	8.2	p1,p8,p5,p9,p7,p4,p3
tear	7_1	fi75	11.9	7.5	p1,p8,p5,p9,p7,p4,p3
tensile	8_10_10_1	fi75	14.4	-4.4	p4,csf,elo,p9,p5,p1,p7,p8

Best with process data, first partition					
output	arch	part	sigma	merr	inputs
csf	7_30_1	fi75	13.5	0.1	p1,p8,p5,p9,p7,p4,p3
density	7_10_10_1	fi75	11.9	10.3	p1,p8,p5,p9,p7,p4,p3
elongation	7_20_1	fi75	7.1	8.2	p1,p8,p5,p9,p7,p4,p3
tear	7_1	fi75	11.9	7.5	p1,p8,p5,p9,p7,p4,p3
tensile	7_30_1	fi75	15.3	-6.9	p1,p8,p5,p9,p7,p4,p3

Best with process data, continuous learning					
output	arch	part	sigma	merr	inputs
csf	7_30_1	co75	11.9	-0.8	p1,p8,p5,p9,p7,p4,p3
density	7_20_1	co75	13.9	2.6	p1,p8,p5,p9,p7,p4,p3
elongation	7_20_1	co75	8.0	2.3	p1,p8,p5,p9,p7,p4,p3
tear	7_1	co75	12.0	2.8	p1,p8,p5,p9,p7,p4,p3
tensile	7_10_10_1	co75	14.3	-3.3	p1,p8,p5,p9,p7,p4,p3

Table 5: The best results that were obtained in this study. Abbreviations: den=*density*, elo=*elongation*, tea=*tear*, ten=*tensile*. *par1* – *par9* are written as *p1* – *p9*.

5 DISCUSSION OF RESULTS

The results of this study have showed that it is possible to do a prediction of certain laboratory measurements with a fairly good precision. Some important questions that arise are:

- Are the results good in comparision with other methods?
- Is the precision obtained high enough?
- Can the precision be improved?

Considering the first and second questions it is not possible to give a concise answer as we have not had results available from similar experiments with other methods on the same type and amount of data. Further, the data set underlying our investigation have most likely been too small to allow an answer with high significance.

5.1 Prediction Quality

We have reached a prediction quality that lies around 5-10% standard deviation, calculated relative to the dynamic range of the parameters. Under some circumstances this may be sufficient and under others it may not. For many of the parameters one should take into consideration that the actual ranges that have been used are often small compared to the absolute value of the parameters.

It must be clear, however, that no method can make a better result than what is possible due to disturbances and measurement errors in the training and test data. If the errors for these were measured as parts of the absolute values they would be much smaller than presented here. We did also observe that the prediction quality was rather much dependent on the type of partitioning into training and test set that was done as was shown in table 2. This indicates that something has happend in the process from the first to the last sample. The prediction quality can probably be increased by using more samples and by using more measurement values. More samples is always better as this allows us to use a more complex network and more measurement values, like for instance *temperatures* etc., may allow us to find other significant inputs.

The coding technique used is also important. By using, for instance interval coding, as is described in the section "FUTURE EXTENSIONS" below, it may be possible to gain some more robustness and reasonable solutions also when the data set is inconsistent and unreliable.

5.2 Data Availability

The data that has been available for training of the networks has been quite small compared to what common rules of thumb say. For some of the more complex nets, we should, considering these rules, have used at least ten times the amount of data that was actually available. This would be perfectly possible considering training times etc. This shortage of data has to be taken into account when the results are evaluated. Those cases where we got a poorer result with a larger hidden layer than with a smaller one, indicate that the data set was in fact too small. Some less intuitive results, such as the indication that density would not be significant for predicting tear and tensile may also be attributed to the relative lack of data.

6 FUTURE EXTENSIONS

Under the assumption that the neural network method developed and studied here is considered good enough, it may be used almost "as is", to do prediction of industrial process parameters. The method could, however, be extended to cover a wider range of usage than the cases studied here would indicate.

6.1 Measurement Estimation

One of the first things we did in this study was to throw away those samples containg data that could be easily detected as not being within reasonable limits. The reason for doing so is of course that if we fed this data into the learning network it would learn an illegal input output mapping. There may be several reasons for data to be outside allowed limits. The data may either be totally missed due to forgotten manual input or outside limits due to a bad probe. The reason may also be that laboratory data is missing due to some problems. It may also be that some tests are so complicated or expensive that they are preferably not done too often.

In all the cases above in which some data is missing it may not be necessary to throw the whole sample anyway. Under the assumption that one datum can be reasonably estimated from a subset of the rest of the data samples, then the missing datum can instead be replaced with its estimation. Each parameter can have an optimal associated network that runs on a timescale suited for the cost and prediction quality for that specific parameter, thus making all parameters instantly available without explicit measurements.

6.2 Consistency Check

When a datum has found to be within its allowed extreme limits this does however, not guarantee that it is valid. A manually fed value may look right but may still be 20 % from the correct value. A probe may be degraded or need calibration. Assume then, that a set of networks are used for prediction of each parameter from a subset of the other parameters. For each new sample that arrives, the network performs a prediction of each of the parameters. This prediction can then be compared with the value just fed in. Data can thus be checked sample by sample, datum for datum that it does not significantly deviate from its predicted value or that it does not cause an extreme value in some other parameter. This will of course be a great help for an operator doing manual input but it will also provide a possible way to give alarms about the necessity for recalibration or replacement of degraded probes.

6.3 Optimization of Learning Time

The method could probably, with some effort, be extended with optimization of the learning time. The learning iterations would then be stopped when the performance on the training set could not be improved and before the performance on the test set would possibly get worse due to overlearning. This optimization may be a hard problem to solve in a general way but by using statistics about required iterations versus number of connections this could probably be done in a "good enough" way. A great advantage with this would be that the required precision would not need to be set in advance. The network would do the best possible anyway.

6.4 Installation of online system

The method studied in this work for parameter estimation can be installed online on a workstation or on a reasonably fast personal computer, preferrably a UNIX work statiton.

6.5 Interval Coding Technique

An important coding technique that has not yet been tested is to use several units to code for one value. Such interval coding technique can be utilized for both logical (on/off) as well as for analogue values.

There are several reasons that speak for interval coding. We may first look at the input values. First it may be so that the precision needed is not rectangular distributed over the whole input range. An input value range may be distributed into several groups where all the values lie in clusters with almost no values in between. With an interval coding technique the available precision can be focused on these groups. One may also want to indicate the quality of a sample. With interval coding it is possible to lower the significance for certain samples.

The same reasons why interval coding is good on the inputs is also valid on the outputs. There is however, one more important reason to have interval coding on the outputs. As was mentioned previously, a multi-layer network with an architecture complicated enough, can in principle learn an input output mapping with arbitrary precision if each input vector corresponds to a unique output vector. If, however, one input vector value can result in several different output values in the training set, then this set can not be learned completely. By instead using interval coding on the outputs the possibility to learn the training set can be increased. It is also possible that the training vectors are nice but during prediction one may get conditions when the input vector would indicate two or several contradictory values on the outputs. This situation can also be resolved with the help of interval coding.

In cases with multiple valued outputs this can be interpreted as multiple hypothesis about the values.

6.6 Temporal Coding

As was mentioned in section 4.5, there are some temporal representation problems that have to be solved before a temporal coding of the inputs can be done. One method is to sample the process periodically, thus making the samples equidistant in time. Another method, that was previously mentioned, could be to do some kind of interpolation between the samples. If the process contains some kind of “memory”, then such coding technique may improve the results further.

7 SUMMARY

This study has focused on the problem of predicting various measures of paper quality from relevant input parameters. Typically, we have reached a prediction quality of 5-10% relative standard deviation. The results obtained do not give a complete answer to whether or not the artificial neural network technique can do better or at least equally good as other methods. It has, however, indicated that a quite good result can be obtained even though the amount of data available for training and evaluation of the networks was limited. Some ways of extending the use of this method to on-line verification and filling in of data has been proposed. A neural network based technique has an advantage in that it is self-organizing and capable of adapting to a process that may change with time.

8 Acknowledgments

STORA Teknik AB is acknowledged as being the sponsor of this project and supplier of data. We also want to thank the MITRE corporation for making their Aspirin/MIGRAINE software package available and Aubrey Jaffer at MIT AI-lab for making his scheme package *scm* available.