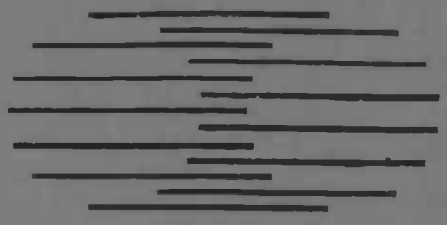
IDIAP COMMUNICATION

IDIAP
Martigny - Valais - Suisse



Minimal High Order Perceptron Construction

Robbert Visscher *

March 1997

* Cognitive Science and Engineering, University of Groningen

Foreword

This report contains an outline and papers concerning the research conducted at IDIAP, from May 1996 to February 1997, for the finalization of my studies of Cognitive Science and Engineering, at the University of Groningen. With this last work my studies have come to an end. During this time I have learned a great deal, not only through my studies, but also through my extra-curriculary activities. Therefore, I would like to thank several people who have helped me make my studies and my stay in Switzerland into a success.

My parents and André for supporting me, not only during my stay in Switzerland, but also, and most importantly, during my studies in Groningen. Without their moral and, of course, financial support it would have been impossible to round off my studies succesfully.

Furthermore, Emile Fiesler, from IDIAP, for having confidence in me, and giving me freedom in conducting the research and Tjeerd Andringa for giving me helpful feedback from the Netherlands. Hans, Perry, Georg, the rest of the people at IDIAP, and Christian for giving me a swell time in Switzerland.

Last but not least my friends in Groningen without whom studying in Groningen would not have been as much fun. Of these friends I would like to name two; Reinder for introducing me to the studies of Cognitive Science and Arieke for proof-reading my final report.

Groningen, 31 March 1997.

Robbert Martijn Visscher.

Abstract

High Order Perceptrons offer an elegant solution to the problem of finding the amount of hidden layers in multilayer perceptrons. High order perceptrons only have an input and an output layer, whose size is completely defined by the problem to be solved.

The major drawback of high order perceptrons is the exponential number of possible connections, which can even become infinite. The aim of this work is to find ways of restricting the amount of connections by verifying a restriction method on the order of the network and to identify a heuristic which can be used in an ontogenic method for the dynamical construction of the connectivity of the high order perceptron. Besides these two issues an answer is also found to whether rerandomization of the parameters is beneficial for the construction.

Keywords: ontogenic neural networks, pruning, generalization, high order perceptrons, partially connected networks, backpropagation neural networks, feature selection.

Contents

1 Introduction

This research on constructing minimal high order neural networks, was done as part of the curriculum for Cognitive Science and Engineering (TCW) at the University of Groningen in The Netherlands.

One of the topics in cognitive science and engineering is connectionist systems, and neural networks are an example of that. An important part of the curriculum involves the actual implementation of systems. This research into minimal networks enhances the practicle usage of neural networks in general.

The research was performed at the Dalle Molle Institute for Perceptive Artificial Intelligence (IDIAP) in Martigny, Switzerland. IDIAP researches are of both theoretical and applied nature in the domain of artificial intelligence and more specifically the study of perception, cognition and pattern recognition. The three main research groups are neural networks, speech recognition and computer vision. This research was done for the neural network research group.

A major research goal of the neural networks group at IDIAP is the development of compact and user-friendly neural networks. Large scale acceptance of neural networks has been hampered by the fact that they are user-*un*friendly. A large amount of expertise and training overhead is required for the selection of the topology and the training parameters.

To alleviate this problem a promising new alternative to multilayer perceptrons is introduced, the high order perceptrons. These networks are characterized by the fact that they only have an input and an output layer whose sizes are completely defined by the problem and hence no choice about the number of hidden layers and units per layer has to be made. The absence of hidden layers make these networks more efficient and, in combination with complexity reducing strategies, enhance their compactness. These strategies are based on partial connectivity, meaning that not all possible connections are used, and training methods that automatically modify the network during the training process, so-called *ontogenic methods*. These methods enhance the user-friendliness by reducing the amount of parameters (e.g. weights) by exploiting the network's capability to learn and self-organize.

Although high order perceptrons have a potentially unlimited amount of connections, several studies show that it is not needed to use all possible connections as partially connected networks perform very well [Lee-86]. In fact, a network can be found that is just big enough to map the data. In [Fiesler-93] these networks are called *minimal networks*.

Several complexity reducing strategies have been investigated at IDIAP. These methods can be roughly divided into initialization methods and ontogenic methods. The first are methods by which a final network is directly constructed from the data. Ontogenic methods also make use of information in the data, but the network is automatically modified during the training process, and only after the training process construction is finished.

Ontogenic methods generally fall into three categories: *growing methods* which add connections to a small network, *pruning methods* which remove connections from a large network, and a combination of growing and pruning. This research will concentrate on the combination of growing and pruning as a complexity reduction strategy.

First of all, some background will be given into the basic theories that underlie this research. High order perceptrons, and the reasons why they are more efficient and user-friendly than multilayer perceptrons, will be considered first. These networks have a drawback, however, and that is their potentially enormous amount of connections. As was mentioned above not all of these connections are needed and some background will be given on minimal neural networks. Besides minimality, another important issue is generalization, which will also be considered in relation to higher order perceptrons. Finally, ontogenic methods as a network reduction strategy will be considered.

In the following papers the specific research items will be introduced and discussed together with the results. In the first paper three simple heuristics for the construction of high order perceptrons will be introduced, together with the rerandomization of weights as a strategy for avoiding local minima. In the second paper a restriction on the order of the network will be compared to not restricting the order. By restricting the order a restriction on the possible infinite amount of connections can be obtained. In the third paper three further heuristics for the construction of high order perceptrons will be investigated.

2 High Order Perceptrons

High order perceptrons are a relatively new kind of neural networks that have advantages over multilayer perceptrons. An important problem for multilayer perceptrons is the fact that they require a lot of knowledge of the problem to be solved and knowledge of neural networks in general. One of the biggest problems is finding the number of hidden layers, and number of neurons per hidden layer. Problems like these can be overcome using so called high order perceptrons instead of multilayer perceptrons. The solutions lie in the fact that these high order perceptrons do not make use of hidden layers which means that the topology is completely defined by the problem. Having no hidden units also means that a simpler training algorithm can be used, which is more efficient than training the multi-layer network [Giles-87]. Furthermore, according to [Lee-86] the number of computational cycles is much lower than for networks using back-propagation. See figure 1 for the relation between a multilayer perceptron and it's higher order counterpart.

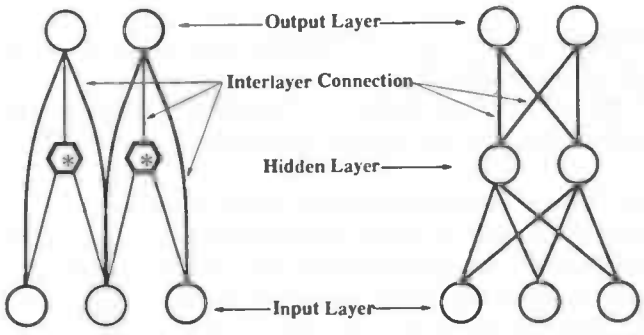


Figure 1: The relationship between high order perceptrons (right) and multilayer perceptrons (left)

As can be seen in the figure, the multilayer perceptron (right) connects exactly one *source* or *input neuron* to one *sink* or *output neuron*. These are all *first order connections* in a fully interconnected three layer perceptron. In the left of the two networks the *hidden layer* is discarded and higher order connections are introduced, that combine inputs using a so called *splicing function*. In the most simpel case this splicing function is a multiplication. [Rumelhart-86] calls these high order perceptrons *Sigma-Pi neural networks* and networks using more complicated splicing functions are introduced by [Pao-89] and called *functional link networks*.

The figure above of a high order perceptron can also be written in a formula for an output, y_i, as follows:

$$y_i = \varphi\Big(W_0 + \sum_i W_i x_i + \sum_{i,j} W_{ij} x_i x_j + \sum_{i,j,k} W_{ijk} x_i x_j x_k + \ldots\Big) \tag{1}$$

This equation resembles a *Taylor series expansion*, and is known in signal processing as a *Volterra filter*. In equation 1, φ is the activation or transfer function, y_i is the i-th output, W the weight assigned to a connection, and x_i is the i-th input. The first two terms of this equation are the same as for a standard perceptron, the bias W_0 and the first order connections $\sum_i W_i x_i$. The next terms are the second, third, and higher order connections. The order of a connection is determined by the amount of inputs combined by that connection and the order of a network, ω, is determined by the connection of the highest order.

2.1 Computational Burden

A high order perceptron can be used for equivalent problems as multilayer perceptrons and from formula 1 it can easily be seen that the computational burden of a higher order perceptron is smaller than that for the multi-layer perceptron. In the case of the first only the output neurons have an activation function $\varphi(.)$. In the case of the multilayer perceptron the hidden units also have an activation function $\varphi(.)$. The output for a three layer perceptron can be written as follows:

$$y_i = \varphi(\sum_k W_2 \varphi(W_0 + \sum_j W_1 x_j) \tag{2}$$

Whereas the output for a second order perceptron can be written as:

$$y_i = \varphi(W_0 + \sum_j W_1 x_j + \sum_{j,k} W_2 x_j x_k) \tag{3}$$

In the multilayer perceptron the activation function $\varphi(.)$ comes in twice compared to once in the higher order case. In the higher order case there is only a multiplication because of the choice of the splicing function used for the second order connections.

2.2 Learning

Besides the calculation of the output, the learning-rule for high order perceptrons is also simpler than for multilayer perceptrons. In two layered standard perceptrons the *delta-rule* is used as a learning-rule. The weight changes are determined using an errorsignal $e(n)$, which is the difference between the desired output $d(n)$ and actual output $y(n)$. The learning algorithm changes the free parameters, the weights, to minimize the error signal.

Multilayer perceptrons have hidden units, which means that not all weights are connected to the output. For connections to the output layer the new weights can easily be determined because the difference between desired and actual output is readily available. For weights connected to hidden layers however, that is not the case and for these weights an error function has to be calculated based on the error function determined for the connections to the output layer. This means that the error has to be fed backwards through the network to determine the changes for every layer of connections. For this reason the learning rule is called, back-propagation[1] [Rumelhart-86]. This learning rule is computationally more demanding than the delta rule and can easily get stuck in local minima.

As for standard perceptrons, higher order perceptrons do not have hidden layers. As a learning rule an augmented form of the standard delta-rule can be used as shown below. The delta-rule uses a linear activation function which in the back-propagation learning-rule, also known as the *generalized Delta-rule*, may be changed to a non-linear one. This can be introduced in the equations below, but for simplicity a linear activation function is assumed. For a standard perceptron with one output neuron the learning rule takes the form [Haykin-94]:

$$w'_{1(i,k)}(n+1) = w_{1(i,k)}(n) + \eta[d(n) - y(n)]x_i(n) \tag{4}$$

$$\begin{aligned}
x_i(n) &= i\text{-th input from } (p+1)\text{-by-1 input vector} \\
&= [-1, x_1(n), x_2(n), \ldots, x_p(n)]^T \\
w_{1(i,k)}(n) &= \text{weight connecting input } x_i \text{ to output } y_k \\
y(n) &= \text{actual output} \\
d(n) &= \text{desired output} \\
\eta &= \text{learning-rate parameter}
\end{aligned}$$

The updated weight $w'_{1(i,k)}$ after presentation of the input pattern $(n+1)$ is the weight $w_{1(i,k)}$ after presentation of pattern n plus the error signal $[d(n) - y(n)]$ multiplied by the input associated with the weight. This term is then multiplied by a constant η known as the learning-rate. Because there are no hidden

[1] Several terms are used in the literature to denote this learning algorithm, such as error back-propagation algorithm or just back-prop. This algorithm is also known as the Generalized Delta rule

layers the weight update can be calulated at once. The connection associated with the weight updated above is a first order connection, the connection is from one input to one output. The update rule for a second order connection can easily be derived from the update rule for a first order connection and takes the following form:

$$w'_{2(i,j,k)}(n+1) = w_{2(i,j,k)}(n) + \eta[d(n) - y(n)]x_i(n)x_j(n) \tag{5}$$

In equation 5 the input x_i has been replaced by two input variables x_i and x_j, hence the notation $w_{s(i,j,k)}$ where i, j and k denote the two input variables connected to the output variable respectively. The remaining part of the equation is unaltered. As for the weight update for a first order connection the weight update for a second order connection can also be calculated instantaniously because from the output neuron the high order connection is simply another input for which a weight has to be found so as to minimize the error.

2.3 Topology

In high order perceptrons the connections are the solution to the problem estimating the optimal number of hidden layers but the introduction of high order connections means that a large amount of connections are possible. Especially when all connections up to a certain order are present, a so called fully connected network. For a fully connected neural network of order Ω the amount of connections W is given by [Fiesler-92.3] :

$$W = N_2 \sum_{i=1}^{\Omega} \binom{N_1}{i} \tag{6}$$

The relation between number of weights W and the number of neurons in the input layer N_1 is exponential and can go to infinity for ever higher Ω. However, in this work a restriction on the order is investigated by not allowing all possible combinations of inputs. An upper bound for the order $\Omega = N_1$ is found if inputs are only allowed to combine with other inputs to form a high order connection.

Although these connections form a drawback for the use of high order perceptrons over multilayer perceptrons, in the next section the notion of *partially connected networks* and *minimal topologies* will be introduced.

3 Defining a Good Topology

Before going on to methods for limiting the amount of connections, an idea of what a good topology is will have to be introduced for later comparison of the different methods. Topologies of artificial neural networks consist of a framework of neurons and their interconnection structure. In high order perceptrons the framework of neurons is detemined by the problem, but the connectivity has still to be determined.

In defining a good topology, several issues play a key role. The network has to be easily implementable, in the sense that the network topology has to be as simple as possible. This simplicity will also help to understand the working of the network. As was seen in the previous section, high order perceptrons were already simpler than multilayer perceptrons, except for their connectivity. For a high order perceptron to be as simple as possible, a low connectivity is desired. But besides this issue the network also has to have a good performance.

The two issues are thus:

- The minimal topology.
- The performance of the network.

3.1 Minimal Topologies

In multilayer perceptrons a fully connected network is often taken because it supposedly has a greater robustness due to multiplication of information. For high order perceptrons, as was indicated above, this means a great computational load. However, research aimed at finding partially[2] connected architectures has been done. [Canning-88], [Gardner-89] and [Walker-90] wrote some theoretical papers on partially interconnected networks. Most work done on this topic however concerns multilayered networks and not high order networks. But also for high order perceptrons this fully connected network is not always the best network for the problem [Fiesler-92]. Several reasons are:

- All connections are often an overkill, resulting in an extra computational burden.
- Smaller networks tend to generalize better. Bigger networks can over-fit the data (see following paragraph).
- Smaller networks are more easily hardware implemented.

The optimum topology for a network is, according to [Fiesler-93], the smallest topology that solves the problem. In fact, there is a *minimal topology* defined by Fiesler as being the network that solves the problem adequately with a minimal computational complexity. Adequately is in this sense somewhat underspecified by the paper, but can be thought of as meeting a given performance criterion such as the *generalization* (see following section). This definition of minimality is of course only suitable for theoretical problems where the whole problem domain is known. For real world problems, where the total domain is unknown, this minimality principle generalizes to a smallest topology search.

For very simple theoretical problems an exhaustive search can be done to find the minimal topology. This is what is done for the XOR-problem [Rumelhart-86] and minimal topologies are found for multilayer perceptrons and higher order perceptrons. The minimal networks for higher order perceptrons are shown to have less neurons and connections for these specific problems than for multilayer perceptrons.

Minimal networks and partially connected networks are not the same, but through partial connectedness a minimal topology can be acquired. There are several ways of finding partially connected networks and some of these methods will be discussed in the next section of this report.

[2] Several different terms are used to denote partially connected neural networks. Sparsely connected, sparse or diluted to name some. In this paper the more objective term partially connected will be used.

3.2 Generalization Ability

When training a network, the error on the training set always decreases due to the learning rule. This means that for the given set of data, the network will try to find an ever better solution. If the network is trained on all possible instances, it would find the best possible solution. But, this is not the case for real world problems. The data on which the network is trained, is only a selection of what occurs in the real world. What might be the optimal solution for the given set of data, might not be the optimal solution for another set of possible data points. What is needed, is a network that performs well on the training set and performs well on another independent data set, the so-called test set. The performance on this independent data set is called the *generalization ability* of the network, or for short generalization because it gives an indication of the network performance on new data.

In some cases the test set is used in the training of the network, a third set of data is then used to verify the obtained network, the so called *validation set*. In the training of high order perceptrons the test set is not used and hence two data sets are sufficient.

In research on multilayer perceptrons there have been a number of investigations into generalization and the relation between the generalization ability and the training error. There seems to be a simple relation between the training error and the generalization, the training error keeps going down, but at some point the generalization reaches a minimum after which the generalization ability deteriorates. This effect is called *overtraining*, the network has learned the relations in the training set too well, it has learned relations that are present in the training set which are not present in the test set. See figure 2 for the relation.

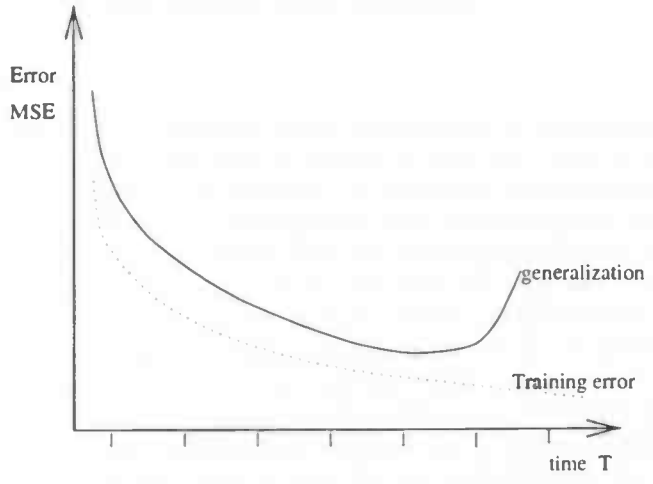


Figure 2: The training-error curve in comparison to the generalization ability.

For higher order perceptrons this relation is not as simple as for multilayer perceptrons. There has not been extensive research into this subject, but evidence shows that the generalization ability will not deteriorate when the training error curve is nearly flat.

3.3 Criterion defenition

In defining an optimal topology the above defined criteria have to be taken into account. In the first place there is the networks size, which should be as small as possible and still be able to solve the problem adequately. Solving the problem adequately can than be defined as having a good generalization ability.

In the following section methods for the construction of partially connected networks will be introduced.

4 Ontogenic Methods

One way of finding a minimal topology for a higher order network is letting networks find their own topology. These methods are often called ontogenic methods and networks constructed on this basis called ontogenic neural networks. The term *superceptron* is also used to describe ontogenic higher order perceptrons [Visscher-96.1].

The term ontogenic is derived from the english word "Ontogenesis" which means the origin and development of an individual. This ontogenesis can also be seen in the biological nervous system. The topology of biological nervous systems is not hard wired but changes with time, connections are made and deleted. This can also be done for high order perceptrons and hence methods that do just this are called ontogenic methods.

These ontogenic neural networks can be divided into three categories depending on were you start the process. In one case the network starts from a small network to become as big as necessery by adding components to a network, such methods are called *constructive* or *growing methods.* On the other hand, a fully connected network can be taken and connections or units can be deleted from the network, these methods are called *destructive* or *pruning methods.*

These methods can also be combined into algorithms that first add components to a small network. After the training is complete and the network has satisfied an error criterion the network then starts pruning the superfluous connections as long as this error criterion remains satisfied.

First of all some pruning methods will be discussed. Several references will be given for these methods and how they can be applied to high order perceptrons. Subsequently construction methods will be taken into account and how they can be used in higher order perceptrons.

4.1 Pruning Methods

Pruning methods are generally used to upgrade network performance by deleting components from a network. Deleting connections and thus making the network smaller is seen as a way to overcome overfitting of the data and enhancing the generalization of the network. In multilayer neural networks these components can be units as well as connections. In high order perceptrons only connections can be deleted from the network.

The difficulty for pruning methods is to decide when the pruning should start, how far the pruning should go and how many and which components have to be pruned from the network. There are quite a few methods for pruning neural networks, ranging from the very simple methods, to the difficult methods that use complex ways of deciding when to start pruning and which components to prune. In [Fiesler-97] and [Fiesler-94.2] a comparison is given of several methods of which an overview will be given here.

[Thodberg-90] starts with a fully connected network and proposes to randomly delete a connection and retrain the network, if the error is not significantly increased the pruning is made permanent. If the error does increase the old network is restored. This is done for all connections in the network. Seeing that that there might be a lot of connections in the network this scheme is very slow.

Another simple pruning method that is less time consuming, is a method that assumes that the importance is proportional to the weight of the connection. Weights that are smallest (close to zero) are pruned, and to reduce the induced error the *average contribution* of the removed connection is added to the corresponding bias.

[Sietsma-91] proposes a pruning method that removes units with small contribution variance on the training set. The contribution of a connection is the value available to the connection from the lower layer, multiplied by its weight.

[Karnin-90] proposes a method that estimates the sensitivity of a network to the removal of a weight by monitoring the sum of all weight changes during training. [Mozer-89] uses a pruning method that estimates the error induced by the removal of a connection based on a manipulation of the function that is to be minimized by backpropagation, the *objective function.*

Several other methods are Optimal Brain Damage (OBD) [LeCun-90], Optimal Brain Surgeon (OBS) [Hassibi-93], autoprune, a pruning method introduced by [Finnoff-93], and Lprune by [Prechelt-95].

4.2 Pruning in High Order Perceptrons

[Thimm-95] has compared several pruning methods to see how well they perform for higher order perceptrons. Both computational burden is taken into account and final found network sizes are taken into account. Furthermore, an idea of the dependancy of the generalization on the pruning proces is given.

Five pruning methods with low computational complexity were chosen. The smallest weight pruning method, the smallest contribution variance method proposed by [Sietsma-91], the method proposed by [Karnin-90], the method proposed by [Mozer-89] and finally autoprune proposed by [Finnoff-93].

Thimm concludes that the simple pruning methods (the smallest weight method and the smallest contribution variance method) perform best on networksize for high order perceptrons. These simple methods find the smallest networks. The generally excepted wisdom that pruning has a positive influence on the generalization capability of neural networks was not found by Thimm for higher order perceptrons. During the pruning phase the generalization performance does not increase steadily, but shows an erratic behaviour. There was also no significant difference in generalization for the different pruning methods. Thimm then concludes that if a good performance on generalization is required, a network with an non-optimal size is to be considered and several different pruning methods will have to be tried.

4.3 Growing Methods

Growing methods are seen as a method of avoiding local minima. The addition of extra units creates a higher dimensional error surface which might eliminate the local minima in a lower dimensional one [Fiesler-94.2]. In multilayer perceptrons these units can be hidden layers, neurons and/or connections but in higher order perceptrons only connections can be added.

There have also been a number of investigations into growing methods for neural networks and some of more well known ones will be discussed here. For an overview and comparison of these and other methods see [Fiesler-94.2], [Fiesler-97] and [Smieja-91]. In these papers methods are divided into two categories, the perceptron based methods and the tree-based methods. A well known growing method will be given as an example.

Dynamic Node Creation (DNC) is introduced by [Ash-89]. In this method a three layered network is constructed by adding a neuron to the hidden layer. This node is added every time the error curve has flattened at some error that is unacceptibly high. The newly added connection is then connected to all input and output units. The moment to stop growing is specified by two parameters set by the user and are the maximum and worse case error. According to Ash, the DNC method needs more training iterations than simply using a fully connected network, but DNC naively starts with one hidden unit while more are probably necessary. Ash also notes that DNC always converges where standard back-propagation training does not.

Other well known methods are the Tiling algorithm by [Mezard-89], a method by [Nadal-89] and the Upstart Algorithm by [Frean-90].

These different methods all have one thing in common. They all grow by adding hidden layers or neurons to hidden layers. Although it is done in totally different ways, these methods can not be used for high order perceptrons, because they do not have hidden units. In the case of high order perceptrons only connections can be added to the network.

In the next section methods for growing high order perceptrons are described which make use of the fact that adding connections means that more information becomes available to the network.

5 Ontogenic Methods for Growing High Order Perceptrons

As mentioned in the previous section, existing growing methods can not be used for higher order perceptrons, because they add units and hidden layers to the network, and thus build a multilayered perceptron. High order perceptrons only have connections that can be added which means that a totally new approach is needed to grow high order perceptrons.

The higher order perceptrons use inputs and combinations of inputs to model the output. The connections in high order perceptrons relate directly to the different inputs. This means that methods for growing high order perceptrons become, from a problem of determining which connections to use, a problem of determining which inputs and combinations of inputs to use.

To determine which inputs to use methods for input selection are used. Input selection, or more well known feature selection methods, select the features that are most informative. In the following sections, different stages in the construction of high order perceptrons are described together with possible feature selection methods.

Before feature selection is introduced, another issue has to be dealt with first. Feature selection methods leave the existing data unaltered. But, besides these selection methods, there are also feature extraction methods. Just as feature selection selects the best features from existing set of features, feature extraction creates a new set of features that are more informative. In this report attention will focus on feature selection methods.

5.1 Feature Selection in Higher Order Perceptrons

Feature selection is of general interest to a lot of different research fields, and a lot of different methods of selecting features exist. Although there are several ways of looking at feature selection, it is mostly used as a method for preprocessing. Before taking the data and train a network, it is always important to look at the data first. Check for very clear relationships between features or outliers. Using these methods, a subset of features can be found that give enough information to determine the solution and at the same time these methods eliminate noise. However, the way these methods determine whether features are needed or not can be very different. These differences boil down to determining a total set of features to evaluating each feature and choosing the best features on the basis of the evaluation. These differences mean that the feature selection methods can be used at different stages in the construction of high order perceptrons as will be discussed below.

In the construction of minimal neural networks there are three stages that are of interest. That is the stage before a network is constructed, the *pre-processing stage*, the stage in which a partailly connected network is created by some other means than ontogenic method, the *initializing stage*, and finally, the stage where a network is constructed by using a growing method, the *growing stage*.

Feature selection as a preprocessing stage, can be seen as a method of *subset selection* of features that describe the problem best. An evaluation of how good certain inputs are relative to others is not important, what is wanted is an elimination of noise features. There are a lot of these methods especially in statistics. All methods of feature selection can be used for this step, however not all will be as efficient and the quality of the selected features is an important criterion for determining which method to use. That this might be a problem, can be seen in the following; selecting certain features means discarding others. So, certain information is lost when the network is trained, because several combinations of inputs are no longer possible. But as a preprocessing step this method can make the use of neural networks more efficient. By eliminating certain inputs, more elaborate initializing and growing methods may become possible.

Forward selection and backward elimination are well known statistical subset selection algorithms used for preprocessing. These methods select a subset of features for which the mean square error is minimal. The selected features in no way reflect the relative importance of the selected inputs [Miller-90] and [Derksen-92].

Initialization methods need to be efficient, because they give an approximation of the partially connected network to use before the growing process. Such a method will have to be more efficient than growing and pruning, which as a combination is already a feature selector in itself. Using feature selection as a method

to find an initial topology before growing, a fast and efficient method is prefered. The fine tuning will be done by growing the network. This way more methods can be combined. An initialization method based on a different selection method may find different connections than the feature selection method on which the growing mehtod is based.

As an initialization method, certain tree-based methods could well be used. These methods determine a complete set of features to be used; also in high order combinations. Features that are most deterministic for a problem will be put at the top of the tree and less important features are put in the branches untill the problem is adequately solved. This tree structure can than be used as a basis for determining which features to use and how to combine them to model the problem. Examples of tree based methods are ID3 [Quinlan-83], and CART [Breiman-84]. A related method is MARS [Friedman-88].

The methods used for subset selection, like linear regression, tend to be usefull for initializing a network, but way it is used differs from the preprocessing stage. As a preprocessing step, it can be used for deleting non-necessary inputs. As an initializing method it can be used as a way to find an initial network topology, where the unused features are not discarded completely, but remain available for later growing.

Feature selection in growing higher order perceptrons is different from methods in the before mentioned stages. It has to give an estimate of how good a certain feature is to be able to make an ordering. This means that a lot of statistical subset selection methods are not useful. Some methods, such as certain tree-based methods and for instance MARS, give a total model, rendering it more useful as an initialization method, than as a growing method.

Next, several issues concerning growing superceptrons and feature selection will be discussed in more detail, and other issues to be dealt with are discussed.

5.2 Feature selection and growing Higher order perceptrons

In this work special attention is given to methods that are useful for growing high order perceptrons. Before useful methods are discussed, several issues have to be considered. First of all an initial topology has to be chosen. This can be a network that is constructed using an initialization method as discussed above or choosing all connections up to a certain order or only using a bias connection. Then the actual growing can take place using a growing method. Feature selection can be used in the growing method, but it is not the actual growing method. Feature selection is used as a heuristic, it evaluates a certain connection from its input(s) and output, the growing method uses this evaluation of the possible connections for determining when and if to add the connection. This distinction is very important.

The fact that feature selection is used as an evaluation method, means that feature selection methods like forward selection can not be used. These methods only select a group of inputs that are necessary, it does not give an evaluation of how important a certain input is. A good example of a method that depends on this evaluation for selecting a subset of inputs is for instance the *mutual information*. This is a measure introduced by the *information theory* [Shannon-49]. The mutual information measures the general dependency between two variables, e.g. input and output. This is much like another possible heuristic, the correlation coefficient, but the correlation measures only the linear dependency between two variables [Li-90]. In for instance [Battiti-94], [Bichsel-89] and [Bridle-90] this mutual information measure has been used as a feature selection method, and even as a feature extraction method, in neural networks.

There are several possible growing methods, but for determining whether a feature selection method performs well the most simple one will be used in this research. This is the growing method where the connections are evaluated according to a heuristic, and afterwards ordered in a list with the connection with the best evaluation at the top. The growing method adds connections to the network according to the ordering of the list. This is an *a priori* growing method because all connections are evaluated before growing. The reason for taking this very simple procedure is that this way a comparison between the different heuristics can be made. Taking a more complex method makes it unclear to determine whether the heuristic or the method itself was performing well, it might even be possible that the combination of method and heuristic is the reason for the good performance.

Using such a method for growing, implies that an evaluation will have to be made for every possible connection, otherwise an ordering will be impossible. Calculating the evaluation for every possible connection

in a high order perceptron means that the feature selection method will have to be very efficient. Ideally another growing method should be used when a good heuristic for evaluating a connection is determined. A possible method could be to pick a possible connection at random and calculate its evaluation. If the evaluation exceeds a threshold the connection can be added. If not, it will be discarded and a new connection will be chosen. When such a method for growing is used, the heuristic does not need to be very efficient, rather a very good evaluation method is needed.

References

[Ash-89] T. Ash. Dynamic Node Creation in Backpropagation Networks. *Connection Science*, vol. 1, no. 4, pp. 365-375, 1989.

[Battiti-94] R. Battiti. Using Mutual Information for Selecting Features in Supervised Neural Net Learning. *IEEE Transactions on Neural Networks*, vol. 5, no. 4, July 1994.

[Bichsel-89] M. Bichsel and P. Seitz. Minimum Class Entropy: A Maximum Information Approach to Layered Networks. *Neural Networks*, vol. 2, 133-141, 1989.

[Breiman-84] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone. *Classification and Regression Trees*, Wadsworth Belmont, CA, 1984.

[Bridle-90] J. S. Bridle. Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters. *Advances in Neural Information Processing Systems*, vol. 2, pp. 211-217, Morgan Kaufmann, San Mateo, CA, USA, 1990.

[Canning-88] A. Canning and E. Gardner. Partially Connected Models of Neural Networks. *Journal of Physics A: Math. Gen.*, vol. 21, pp. 3275-3284, 1988.

[Deffaunt-90] G. Deffaunt. Neural Units Recruitment Algorithm for Generation of Decision Trees. *Proceedings of IJCNN '90*, vol. 1, pp. 637-642, San Diego, USA, 1990.

[Derksen-92] S. Derksen and H. J. Keselman. Backward, Forward and Stepwise Automated Subset Selection Algorithms: Frequency of Obtaining Authentic and Noise Variables. *British Journal of Mathematical and Statistical Psychology*, vol. 45, pp. 265-282, The British Psychological Society, Great Britain, 1992.

[Fiesler-92] E. Fiesler. Partially Connected Ontogenic High Order Neural Networks. Tech-Report 92-02, IDIAP, Martigny, Switzerland, 1992.

[Fiesler-93] E. Fiesler, Minimal and High Order Neural Network Topologies. *Proc. of the Fifth Workshop on Neural Networks*, pp. 173-178, San Diego, California, 1993.

[Fiesler-94.1] E. Fiesler, Neural Network Classification and Formalization. In J. Fulcher (ed.), *Computer Standards & Interfaces*, vol. 16, num. 3, special issue on Neural Network Standardization, pp. 231-239. North-Holland/Elsevier, 1994. ISSN: 0920-5489

[Fiesler-94.2] E. Fiesler, Comparative Bibliography of Ontogenic Neural Networks. *Proc. of the International Conference on Artificial Neural Networks (ICANN 94)*, pp. 793-796, Sorrento, Italy, 1994.

[Fiesler-97] E. Fiesler and R. Beale, *Handbook of Neural Computation*. Institute of Physics and Oxford University Press, New York, New York, 1997. ISBN: 0-7503-0312–3 and 0-7503-0413-8.

[Finnoff-93] W. Finnoff, F. Hergert, and H. G. Zimmermann. Improving Model Selection by Nonconvergent Methods. *Neural Networks*, vol. 6, pp. 771-783, 1993.

[Frean-90] M. Frean. The Upstart Algorithm: A Method for Constructing and Training Feedfroward Neural Networks. *Neural Computation*, vol. 2, pp. 198-209, 1990.

[Friedman-88] J. H. Friedman. Multivariate Adaptive Regression Splines, Technical Report 102, Stanford University Lab for Computer Statistics, 1988.

[Gardner-89] E. Gardner. Optimal Basins of Attraction in Randomly Sparse Neural Network Models. *Journal of Physics A: Math. Gen.*, vol. 22, pp. 1969-1974, 1989.

[Giles-87] C. L. Giles and T. Maxwell. Learning, Invariance, and Generalization in High-Order Neural Networks. *Applied Optics*, vol. 26, no. 23, pp. 4927-4978, 1987.

[Hassibi-93] B. Hassibi and D. G. Stork. Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. *Advances in Neural Information Processing Systems*, vol. 5, Morgan Kaufmann, San Mateo, CA, USA, 1993.

[Haykin-94] S. Haykin. *Neural Networks; A Comprehensive Foundation*. MacMillan College Publishing Company, New York, New York, USA, 1994. ISBN: 0-02-352761-7

[Judge-85] G. G. Judge, W. E. Griffiths, R. Carter Hill, and T.-C. Lee. The Theory and Practice of Econometrics. Wiley Series in Probability and Mathematical Sciences. John Wiley and Sons, 2nd edition, 1985.

[Karnin-90] E. D. Karnin. A Simple Procedure for Pruning Back-Propagation Trained Neural Networks. *IEEE Transactions on Neural Networks*, vol. 1, num. 2, pp. 239-242, 1990.

[LeCun-90] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal Brain Damage. *Advances in Neural Information Processing Systems*, vol. 2, pp. 598-605, Morgan Kaufmann, San Mateo, CA, USA, 1990.

[Lee-86] Y. C. Lee, G. Doolen, H. Chen, T. Maxwell, H. Lee, and C. L. Giles. Machine Learning Using a Higher Order Correlation Network. *Physica D: Nonlinear Phenomena*, volume 22, pages 276-306, 1986. ISSN: 0167-2789.

[Li-90] W. Li. Mutual Information Versus Correlation functions. *J. of Statistical Physics*, vol. 60, no.5/6, pp. 823-837, 1990.

[Mezard-89] M. Mezard and J.-P. Nadal. Learning in Feedforward Layered Networks: The Tiling Algorithm. *Journal of Physics A: Math. Gen.* vol. 22, pp. 2191-2203, 1989.

[Miller-90] A. J. Miller. *Subset Selection in Regression*, St. Edmundsbury Press Ltd, Bury St Edmunds, Suffolk, Great Britain, EU, 1990. ISBN: 0-412-35380-6.

[Mozer-89] M. C. Mozer and P. Smolensky. Using Relevance to Reduce Network Size Automatically. *Connection Science*, vol. 1, num. 1, pp. 3-16, 1989.

[Nadal-89] J.-P. Nadal. Study of Growth Algorithm for a Feedforward Network. *interantional Journal of Neural Systems*, Vol. 1, No. 1, pp. 55-59, 1989.

[Pao-89] Y. Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, USA, 1989. ISBN: 0-201-12584-6.

[Prechelt-95] L. Prechelt. Adaptive Parameter Pruning in Neural Networks. Tech. Report 95-009, International Computer Science Institute, Berkeley, California, 1995.

[Quinlan-83] J. R. Quinlan. Learning Efficient Classification Procedures and Their Application to Chess and Games. *Machine Learning: An Artificial Intelligence Approach*, chapter 15, pp. 463-482, Tioga P., Palo Alto, 1983.

[Rumelhart-86] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* The MIT Press, Cambridge, Mass., 1986. ISBN: 0-262-18120-7.

[Shannon-49] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* Urbana, IL. University of Illinois Press, 1949.

[Sietsma-91] J. Sietsma and R. J. F. Dow. Creating Artificial Neural Networks that Generalize. *Neural Networks*, vol.4, num.1, pp.67-69, 1991.

[Smieja-91] F. J. Smieja, Neural Network Constructive Algorithms: Trading Generalization for Learning Efficiency?, *German National Research Center for Computer Science*, November 22, 1991.

[Thimm-95] G. Thimm and E. Fiesler. Evaluating Pruning Methods. *1995 International Symposium on Artificial Neural Networks (ISANN'95)*, pp. 20-25, 1995.

[Thodberg-90] H. H. Thodberg. Improving Generalization of Neural Networks through Pruning. *International Journal of Neural Systems*, vol. 1, pp. 317-326, 1990.

[Visscher-96.1] R. M. Visscher, E. Fiesler and G. Thimm. Superceptron Construction. *Proceedings of SIPAR Workshop '96*, University of Geneva, Geneva, Switzerland, 1996.

[Walker-90] C. C. Walker. Attractor Dominance Patterns in Sparcely Connected Boolean Nets. *Physica D: Nonlinear Phenomena*, vol. 45, pp. 441-451, 1990.

6 Papers

Superceptron Construction Methods

R. M. Visscher and E. Fiesler

IDIAP, CP 592, CH-1920 Martigny, Switzerland

E-mail: Robbert@IDIAP.CH

http://www.idiap.ch/nn.html

Abstract

Superceptrons are higher order perceptrons constructed by ontogenic methods. These superceptrons offer an elegant solution to the problem of finding the amount of hidden layers in multilayer perceptrons because they only have an input and an output layer, whose size is completely defined by the problem to be solved. The power of Superceptrons lies in the use of high order connections which render them superior in functionality with respect to back-propagation based neural networks.

The aim of this paper is to identify a so called ontogenic method for a dynamical construction of the connectivity of the Superceptron. More precisely, an answer is found to whether rerandomization of the parameters is beneficial for the construction, and to which ontogenic methods are candidates for adaptively building the network topology.

Keywords: ontogenic neural networks, growing, pruning, generalization, high order perceptrons, partially connected networks, backpropagation neural networks, rerandomization, superceptrons.

1 Introduction

A *Superceptron* is a higher order perceptron constructed using so called *ontogenic* methods. This isa relatively new kind of neural network that have advantages over the popular multilayer perceptrons. An important problem for the practical application of multilayer perceptrons is that they require knowledge of the problem to be solved and knowledge of neural networks. One of the biggest problems is finding the number of hidden layers and number of neurons per hidden layer. This problem makes the usage of multilayer perceptrons very user-unfriendly.

A promising way of selecting the number of hidden layers and neurons per hidden layer is to make use of the network's ability to adapt and to let it find the topology itself. The hidden layers are constructed by either adding units to the network or deleting units from the network whenever necessary. These methods are called *ontogenic* methods [Fiesler-97] and often find smaller and thus more efficient networks. However the ideal ontogenic method is not found yet. Moreover, these ontogenically constructed multilayer perceptrons still have hidden layers which make it hard to analyse the network's performance. Another type of network, the high order perceptron, does not make use of hidden layers but combines inputs by so-called high order connections. A high order connection combines inputs using a *splicing function*, which, in our case, is a multiplication. This way of modeling the problem resembles a *Taylor series expansion* and is known in signal processing as a *Volterra filter*. The output of the network, y_i, can be written as given in equation 1:

$$y_i = \varphi\Big(W_0 + \sum_i W_i x_i + \sum_{i,j} W_{ij} x_i x_j + \sum_{i,j,k} W_{ijk} x_i x_j x_k + \ldots\Big) \quad (i \neq j) \tag{1}$$

In this equation, φ is the activation or transfer function, y_i is the i-th output, W the weight assigned to a connection, and x_i is the i-th input. The total number of possible combinations is limited as high order connections need not use the same input more than once [Visscher-97.2]. The first two terms of this equation are the same as for a standard perceptron, the bias W_0 and the first order connections $\sum_i W_i x_i$. The next terms are the second, third, and higher order connections.

In these high order perceptrons only an input and an output layer are required, the sizes of which are completely defined by the problem. This makes these networks much easier to construct than multilayer perceptrons. Having no hidden units also means that a simpler learning rule can be used. The learning rule for standard perceptrons is the delta-rule which has been extended to the generalized delta-rule, or backpropagation learning

rule, for multilayer perceptrons. The higher order perceptrons do not make use of hidden units which means that the delta-rule can be used as a learning rule. However, there is a tradeoff, because for high order perceptrons the order of the problem has to be estimated.

Using higher order connections has the disadvantage that an enormous amount of connections are possible because in a fully connected network there is an exponential relationship between the number of connections and the number of neurons in the input layer [Fiesler-97]. However a fully connected network is not needed, as partially connected networks perform very well [Lee-86]. In fact, a network is needed that is just big enough to map the data; in [Fiesler-93] these networks are called *minimal networks*. For small, theoretical problems such as the XOR-problem [Rumelhart-86], the minimal topology can be found and proven to be the smallest. For real world problems used in this paper, the minimality of a topology can not be proven, so the topologies sought are to be as small as possible.

As with multilayer perceptrons, the construction of these networks can be done using ontogenic methods, letting the network adapt and learn the problem-specific network topology. Existing growing methods for multilayer perceptrons primarily add units to the hidden layers, which means that these methods are not useful for higher order perceptrons. On the other hand, pruning methods for multilayer perceptrons delete units and/or connections from the network which means that they can also be used for pruning higher order perceptrons [Thimm-95].

In this paper three simple growing methods are introduced that determine the best connections to add. Besides these three growing methods a weight randomizing method during the growing phase will also be investigated. This weight randomizing method might improve the average network size and performance by avoiding local minima. Both heuristics and rerandomization will be explained in subsequent sections.

2 Description of the Ontogenic Methods

The process of constructing a network using ontogenic methods starts with defining an initial topology. Several different possibilities exist ranging from a topology with only the bias connection(s) to all connections up to a certain order and adding higher order connections or replacing lower order connections with higher order ones. In this case a bare topology with only bias connections is used. This bare topology is too small to learn the input-output combinations and connections have to be added. The addition of connections means that the network gets more information which reduces the squared error in the trainingset.

Connections are added according to a growing method which calculates the a value for a connection using a certain heuristic. This is an a priori computation and the connections are ordered to ensure that the best (relative to the heuristic) connections are added first. The amount of connections means that the methods used should ideally be as simple as possible and therefore three heuristics have been used based on the variance of the input, the correlation between the input and output and a combination of correlation and variance. As a reference random adding of connections is used.

The variance heuristic calculates the variance of the input data for a given connection:

$$V_{x \to y} = VAR(X) \qquad V_{x_1, x_2 \to y} = VAR(X_1 * X_2) \tag{2}$$

The first of the two equations calculates the variance for a first order connection with input x connected to output y. The capital X denotes the data corresponding with input x. The second calculates the variance for a second order connection with inputs x_1 and x_2. For a second order connection the corresponding input variables are first multiplied, because the splicing function is a multiplication, and the outcome of this multiplication is used to compute the variance.

The largest value for the variance is put at the top of the list because the hypothesis is that the larger the variance is the more information might be contained in the data. Or the other way around: if an input variable has zero variance this means that the input is always the same irrespective of the output and is thus bound to convey little information.

For the correlation heuristic the correlation coefficients between the input and output variables of the data, and hence the corresponding connections, are calculated. For first order connections this comes down to the correlation between the corresponding input variable and output variable. For a second order connection the

corresponding input variables are first multiplied, and the result of the multiplication is used to calculate the correlation coefficient.

$$r_{x \rightarrow y} = CORR(X, Y) \qquad r_{x_1, x_2 \rightarrow y} = CORR(X_1 * X_2, Y) \tag{3}$$

Here the $r_{x \rightarrow y}$ denotes the correlation coefficient for a first order connection from x to y, the capital X denotes the data for input x and equivalently Y denotes the data for the output of the connection y. Similarly $r_{x_1, x_2 \rightarrow y}$ is the correlation coefficient for a second order connection with inputs x_1 and x_2. The correlation coefficient ranges between $[-1, 1]$, but for a neural network a negative correlation is the same as a positive correlation save a different sign for the associated weight. Hence an absolute value for the correlation coefficient will be taken.

The correlation computes the linear dependency of the input and the output. A correlation coefficient that approaches 1 indicates that there is a large dependency between the input and the output. In the same way a zero correlation coefficient indicates that the variables are linearly independent. Higher absolute values for the correlation coefficient might be an indication that the input(s) associated with that connection give more information about the output than a connection with a lower value.

After viewing the results for size and generalization of the correlation and variance heuristic a combination of the two heuristics was constructed. The calculations are exactly the same for the variance and correlation given above but the two values are combined in such a way that connections that have a large variance *and* a large correlation are added first. The reason for using the combined method is that besides finding a minimal network it is also important to have a robust heuristic. A heuristic is sought that performs well without having a lot of outliers. Although it might not always find the smallest solution it will always find a good solution. Note that rerandomizing of the weights only takes place during the growing phase.

After the network training has reached a certain error criterion, the growing process is stopped and the pruning process starts. Pruning is done because the network might have added too many connections and the pruning process might be able to remove these superfluous connections. It is generally thought that pruning these connections improves the generalization ability of the networks because superfluous connections deteriorate the network's performance by adding non-informative data to the network. These extra connections can cause the network to over-fit the training data which means that it might be optimal for the given training set but not for the unseen test set. Using less connections also reduces the dimensionality and thus the possibilities of the network to over-fit the training set. For pruning, the *smallest variance method* is used [Sietsma-91]. This is a very simple method, shown to perform very well for high order perceptrons [Thimm-95].

3 Weight Rerandomization

During the growing phase of the high order perceptrons, connections are added whenever the decrease in the error curve becomes smaller than a predetermined threshold and the given error criterion is not reached (see section 4 for further details). Every connection that is added has to be given a random weight when it is introduced otherwise the network will be biased in a certain direction. Furthermore, the newly added connection is added to a network that is already in a state in which the weight setting has been determined. This means that the network might be biased towards the original weight setting, which might not be optimal for the new network.

A solution to this problem might be to assign a higher learning rate to the new connection which decreases during the learning process. This implies that every connection added is assigned its own variable learning rate, making the total computation of the network more difficult and less transparant. An easier solution that might help to overcome this problem is to rerandomize all weights when a new weight is introduced. This ensures that the whole process starts anew and a new weight setting can be found in an unbiased way.

This rerandomizing of the weights might help in finding smaller networks because the networks can find the optimum for every number of connections without being biased towards a certain solution because of previous weight settings. This might also hold true for the generalization capabilities for the networks constructed in this way.

4 Simulations

The construction of a higher order perceptron starts with initializing a network with bias connections only. The training starts, and after a certain amount of training cycles when a *mean squared error* criterion for the training error is not reached, an extra connection is added according to one of the heuristics discussed above. The point in time to add a connection is determined by a minimal decrease in the error slope[1] which is calculated over a certain amount of training iterations. This process is continued until the convergence criterion is reached (see table 1 for the error criterion for each of the datasets). When the criterion is reached the pruning process starts using the smallest variance method as discussed above. Connections are removed and a check is made to see if the error criterion is still satisfied. If this is the case more connections are removed. However if the criterion is no longer satisfied, training takes place until the criterion is satisfied again. The pruning stops when the training error does not reach the criterion and the error slope is smaller than the minimal error slope.

Six real-world data sets were chosen, most of which were obtained (if not stated otherwise) from an anonymous-ftp server at the University of California [Murphy-94], and which are described below. The name of the data set is followed by the number of input and output variables of the problem, which also determines the number of input and output units of the network.

Solar (12,1) contains sunspot activity for the years 1700 to 1990. The task is to predict the sun spot activity for one of the years, given the preceding twelve years. The real-valued input and output data are scaled to the interval [0, 1].

Glass (15,1) consists of 8 scaled weight percentages of certain oxides found in the glass, the ninth input is a 7-valued code for the type of glass (eg.tableware, head lamps etc.). The input is scaled to [-1, 1]. The output is the refractive index of the glass, scaled to [0, 1].

Wine (13,3) is the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The 13 real-valued input values are scaled to the interval [-1, 1], the output values are boolean and scaled to [-1, 1].

Servo (12,1) was created by Karl Ulrich (MIT) in 1986 and contains a very non-linear phenomenon: predicting the rise time of a servomechanism in terms of two (continuous) gain settings and two (discrete) choices of mechanical linkages. The input is coded into two groups of five boolean values each, and two discrete inputs, one assuming four, the other five values. The output is real-valued, and like all real-valued inputs, scaled to the interval [0, 1].

Auto-mpg (7,1) concerns city-cycle fuel consumption of cars in miles per gallon, to be predicted in terms of three multi-valued discrete and 4 continuous attributes. Input data is scaled to the interval [-1, 1], output to [0,1].

British Vowels (10,11) was created by Tony Robinson (CMU) and concerns speaker independent recognition of the eleven steady state vowels of British English using a specified training set of lpc derived log area ratios. Input data type is real and scaled to the input interval [-1, 1], output is boolean and scaled to [-1, 1].

Parameter settings were all taken from [Thimm-96] and are summarized in table 1. The benchmark name is given followed by the number of connections in a fully connected second order network. For the maximum order 2 is taken because of computational constraints and in [Thimm-96] 2-nd order networks performed very well for these benchmarks. For the training error criterion either the mean square error or a percentage of wrong classifications tolerated on the training set is taken, depending on the kind of benchmark. The initial weight distribution is the same for all data sets: uniform with initial weight variance 10^{-5} and therefore not listed in the table. According to [Thimm-96], for higher order perceptrons a very small value for the initial variance is better than a bigger value.

[1] The decrease in the error slope is determined by calculating the mean error over 20 iterations and comparing it to the mean of the previous 20 iterations. Is the difference between the two smaller than a defined minimal decrease, a connection is added.

benchmark	full 2-nd	learning rate	momentum	activation function	error criterion
Solar	79	0.2	0.5	linear	0.05
Glass	121	0.5	0	stand. sigmoid	0.03
Wine	273	2.5	0	tanh	10%
Servo	79	4.5	0	stand. sigmoid	0.05
Auto-MPG	28	0.1	0.3	linear	0.06
Vowels	616	0.005	0.3	tanh	10%

Table 1: Parameters for each dataset.

Seven datasets were selected for experementing but the seventh dataset, CES, performed poorly due to its small size. This dataset was subsequently removed from the experiments.

5 Results and Discussion

In table 2 the results of the experiments on minimizing the size are shown. Each experiment consists of 50 simulations and the resulting network sizes are considered to be normally distributed. For each experiment a 95% confidence interval is calculated to enhance comparison. The first column shows the name of the dataset. The second column states if all weights were rerandomized after introducing a connection. The subsequent columns give the outcome of the different heuristics with the network size after growing has taken place and the network size after pruning has also taken place. The format is: the mean size of the networks found for that specific experiment followed by the confidence interval. Boldfont indicates the smallest network after growing and after pruning for either with rerandomizing and without rerandomizing, for a benchmark. Bold font is only used when there is no overlap in the confidence intervals.

bench-mark	rerand. weights	heuristic							
		network size after growing				network size after pruning			
		random	correlation	variance	combined	random	correlation	variance	combined
Solar	yes	36.7± 3.0	**15.6**± 0.1	17.2± 0.2	16.1± 0.1	16.7± 1.6	**7.5**± 0.2	8.1± 0.2	11.0± 0.1
	no	36.9± 2.8	**16.1**± 0.1	17.1± 0.1	16.1± 0.1	19.4± 1.8	13.0± 0.0	**7.9**± 0.1	11.7± 0.3
Glass	yes	21.8± 2.1	**13.0**± 0.0	51.0± 0.0	19.0± 0.0	10.9± 1.2	**5.0**± 0.0	10.0± 0.7	7.6± 0.1
	no	24.3± 1.9	**13.0**± 0.0	51.9± 0.1	19.0± 0.0	12.1± 1.2	**7.1**± 0.8	16.2± 0.8	10.6± 0.6
Wine	yes	37.2± 2.0	20.3± 0.1	23.3± 0.1	**18.0**± 0.4	18.7± 1.3	**10.9**± 0.3	13.4± 0.2	11.9± 0.1
	no	29.5± 1.6	17.2± 0.0	21.3± 0.2	**16.4**± 0.2	19.5± 1.3	13.5± 0.3	17.5± 0.6	**13.0**± 0.2
Servo	yes	33.6± 2.8	**23.0**± 0.0	23.1± 0.1	24.0± 0.0	14.9± 1.2	**13.4**± 0.2	20.8± 0.7	14.7± 0.4
	no	35.3± 2.6	23.0± 0.0	**22.0**± 0.0	24.0± 0.0	18.4± 1.4	**10.0**± 0.0	17.2± 0.1	10.2± 0.1
Auto-MPG	yes	20.9± 1.2	18.6± 0.2	**12.3**± 0.1	**12.3**± 0.1	11.5± 1.0	**9.3**± 0.2	10.8± 0.3	10.6± 0.3
	no	21.7± 1.1	19.0± 0.2	**12.6**± 0.1	13.1± 0.2	12.1± 1.0	10.7± 0.2	**10.5**± 0.2	11.3± 0.2
Vowels	yes	306.4± 2.8	295.4± 1.8	**287.7**± 2.4	307.7± 6.5	275.6± 3.6	**182.3**± 11.8	194.8± 1.9	212.6± 15.4
	no	436.6± 21.1	**292.4**± 8.9	392.2± 4.3	313.6± 3.7	229.2± 23.1	**210.2**± 9.1	245.0± 8.1	222.9± 11.4

Table 2: Summary of the results on size for each dataset.

The discussion of the results will be divided into two parts. First of all the results for the different heuristics will be discussed, after which the rerandomization will be taken into consideration. After the growing phase all heuristics find smaller networks than the random adding of connections except for the variance heuristic on the *Glass* dataset and the combined heuristic on the *Vowels* dataset with rerandomization of the weights. The correlation heuristic finds the most networks that are the smallest in size when compared to the other heuristics.

The results after the pruning phase shows an increase in the amount of networks found by the correlation heuristics that are the smallest in size. The networks found by the correlation heuristic are also always smaller than the random adding of connections. This contrasts with the variance heuristic which also finds bigger networks than the random adding of connections. The combined heuristic only finds the smallest network once, but never finishes pruning with more connections than the random case.

Overall the correlation heuristic performs the best. It generally finds smaller networks after growing and pruning. Correlation grown networks also benefit pruning, because the correlation heuristic finds more smallest networks after pruning than after the growing phase.

Besides using different heuristics, all weights have been randomized after the addition of a connection. To see if this gives better results, the mean sizes of these networks are compared with mean network sizes when no rerandomization has taken place, for every benchmark.

After the growing, the difference between rerandomization or no rerandomization is minimal. Of the twentyfour (four possible heuristics and six benchmarks) experiments, with rerandomizing gives smaller networks in twelve experiments, compared to eight for without rerandomizing. Rerandomizing the weights during the growing phase seems to facilitate the pruning process because after pruning the experiments with weight rerandomization perform significantly better than experiments without weight reranomization.

Taking the different heuristics into account it is seen that specifically for the correlation heuristic rerandomization performs better. For five out of six benchmarks with rerandomizing the weights finds the smaller networks. When comparing the different heuristics for rerandomizing, the correlation heuristic finds the smallest network for all benchmarks. The same applies when rerandomization is not taken into consideration, the correlation heuristic finds the smallest networks possible for all benchmarks.

In table 3 the results for the generalization performance are given. The results are the *mean square errors* except for the *Wine* and *Vowels* benchmarks indicated by a *, where it is the amount of correctly classified instances. The resulting generalization performances of the simulations are considered to be normally distributed and a 95% confidence interval is given to enhance comparison. Bold font indicates the best performance for that benchmark for both with and without rerandomization. Bold font is only used when there is no overlap in the confidence intervals.

bench-mark	rerand. weights	heuristic			
		Random	Correlation	Variance	Combined
Solar	yes	0.0865± 0.0028	0.0794± 0.0009	**0.0732**± 0.0005	0.0785± 0.0007
	no	0.0918± 0.0031	0.0816± 0.0005	**0.0731**± 0.0003	0.0796± 0.0007
Glass	yes	0.0378± 0.0014	**0.0367**± 0.0003	0.0444± 0.0008	0.0431± 0.0003
	no	0.0359± 0.0013	**0.0307**± 0.0003	0.0426± 0.0009	0.0468± 0.0006
Wine *	yes	0.749± 0.017	0.793± 0.009	**0.808**± 0.011	0.759± 0.009
	no	0.735± 0.016	0.743± 0.003	0.730± 0.004	**0.771**± 0.005
Servo	yes	0.0791± 0.0032	0.0816± 0.0005	**0.0629**± 0.0006	0.0684± 0.0006
	no	0.0921± 0.0004	0.0852± 0.0004	**0.0653**± 0.0006	0.0813± 0.0005
Auto-MPG	yes	**0.0595**± 0.0007	0.0623± 0.0002	0.0623± 0.0002	0.0623± 0.0003
	no	**0.0598**± 0.0009	0.0628± 0.0003	0.0653± 0.0003	0.0631± 0.0003
Vowels *	yes	0.458± 0.003	0.465± 0.003	**0.519**± 0.003	0.498± 0.003
	no	0.375± 0.007	0.453± 0.005	0.489± 0.003	**0.496**± 0.004

Table 3: Summary of the results for the generalization of the final network after pruning.

For the results of the generalization the following has to be taken into account. In [Thimm-95] it was found that the smallest network is not always the network with the best generalization because the generalization performance shows an erratic behaviour during the pruning phase. Therefore it is possible that the method A that performs best on the size criterion might find a worse generalization compared to a method B, that finds bigger networks after pruning. However a slightly bigger network for method A, buts still smaller than the one for

method B, might give a comparable, or better generalization than the network found by method B. If, however, the smaller network also finds the better generalization a better conclusion can be taken whether or not one heuristic performs better than another taking both network size and generalization into account.

When looking at the results for the generalization performance for the experiments it can be seen that rerandomization of the weights also seems to benefit the performance on generalization. For the different heuristics however the outcome is not that clearcut but the variance heuristic generaly performs better on generalization. However the variance heuristic also finds bigger networks which makes a conclusion difficult.

6 Conclusions

In this paper several heuristics for the ontogenic construction of high order perceptrons were introduced, together with the rerandomization of weights. All the heuristics perform significantly better than the random adding of connections, which justifies their use. There exist however, considerable differences between the different heuristics. The variance heuristic for example is not very reliable because it sometimes grows very big networks which can be very hard to prune. The correlation heuristic seems to be the best because it most often results in the smallest networks found and does not have the problems of the variance heuristic. The combined heuristic also performs reasonably well, sometimes even better than both of the heuristics that comprise it, but not when the overall results of the correlation heuristic are compared with it.

Rerandomizing does not decrease the size of the networks found after the growing phase, but it results in considerably smaller final networks. This might be because the rerandomizing, although not of much influence in the growing part, relaxes the networks in such a state that it benefits pruning. Hence, for finding the smallest networks the correlation heuristic should be used in combination with rerandomizing all the weights when adding a new connection.

Acknowledgements

This research was made possible, in part, thanks to funding by FORMITT. Special thanks to Georg Thimm for helpful comments and suggestions and for proofreading of the paper.

References

[Fiesler-93] E. Fiesler, Minimal and High Order Neural Network Topologies. *Proc. of the Fifth Workshop on Neural Networks*, pp. 173-178, San Diego, California, 1993.

[Fiesler-94.1] E. Fiesler, Neural Network Classification and Formalization. In J. Fulcher (ed.), *Computer Standards & Interfaces*, vol. 16, num. 3, special issue on Neural Network Standardization, pp. 231-239. North-Holland/ Elsevier, 1994. ISSN: 0920-5489

[Fiesler-94.2] E. Fiesler, Comparative Bibliography of Ontogenic Neural Networks. *Proc. of the International Conference on Artificial Neural Networks (ICANN 94)*, pp. 793-796, Sorrento, Italy, 1994.

[Fiesler-97] E. Fiesler and R. Beale, *Handbook of Neural Computation*. Institute of Physics and Oxford University Press, New York, New York, 1997. ISBN: 0-7503-0312-3 and 0-7503-0413-8.

[Lee-86] Y. C. Lee, G. Doolen, H. Chen, T. Maxwell, H. Lee, and C. L. Giles. Machine Learning Using a Higher Order Correlation Network.*Physica D: Nonlinear Phenomena*, volume 22, pages 276-306, 1986. ISSN: 0167-2789.

[Murphy-94] Data made available in 1994 by librarians P. M. Murphy and D. W. Aha from the UCI Repository of Machine Learning Databases, a machine-readable data repository accessible via anonymous-ftp: ftp://ftp.ics.uci.edu/pub/machine-learning-databases.

[Prechelt-95] L. Prechelt. Adaptive Parameter Pruning in Neural Networks. Tech. Report 95-009, International Computer Science Institute, Berkeley, California, 1995.

[Rumelhart-86] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group.*Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. The MIT Press, Cambridge, Mass., 1986. ISBN: 0-262-18120-7

[Sietsma-91] J. Sietsma and R. J. F. Dow. Creating Artificial Neural Networks that Generalize. *Neural Networks*, vol.4, num.1, pp.67-69, 1991.

[Thimm-95] G. Thimm and E. Fiesler. Evaluating Pruning Methods. *1995 International Symposium on Artificial Neural Networks (ISANN'95)*, pp. 20-25, 1995.

[Thimm-96] G. Thimm and E. Fiesler. Weight Initialization in Higher Order and Multi-Layer Perceptrons. Accepted for publication in *IEEE Transactions on Neural Networks*, 1996.

[Visscher-96.1] R. M. Visscher, E. Fiesler, and G. Thimm, Superceptron Construction. *Proc. of SIPAR '96*, pp. , Geneva, 1996.

[Visscher-97.2] R. M. Visscher and E. Fiesler, Order Restriction in High Order Perceptrons, RR 97-02, IDIAP, Martigny, Switzerland, 1997.

Order Restriction in High Order Perceptrons

R. M. Visscher and E. Fiesler

IDIAP, CP 592, CH-1920 Martigny, Switzerland

E-mail: Robbert@tcw2.ppsw.rug.nl

http://www.idiap.ch/nn.html

Abstract

The amount of possible connections in high order networks has an exponential relation to the number of neurons in the input layer, and can even become infinite when the order of the network is unrestricted. Clearly this is not desired and hence the effect of restricting the order is considered here.

Keywords: ontogenic neural network, growing, pruning, generalization, high order perceptron, partially connected network, backpropagation neural network, rerandomization.

1 Introduction

High order perceptrons are a relatively new kind of neural network that have advantages over the popular multilayer perceptrons. An important problem for the practical application of multilayer perceptrons is that they require knowledge of the problem to be solved and knowledge of neural networks. One of the biggest problems is finding the number of hidden layers and number of neurons per hidden layer. This problem makes the usage of multilayer perceptrons very user-unfriendly.

A promising way of selecting the number of hidden layers and neurons per hidden layer is to make use of the network's ability to adapt and to let it find the topology itself. The hidden layers are constructed by either adding units to the network or deleting units from the network whenever necessary. These methods are called *ontogenic* methods [Fiesler-97] and often find smaller and thus more efficient networks. However the ideal ontogenic method is not found yet. Moreover, these ontogenically constructed multilayer perceptrons still have hidden layers which make it hard to analyse the network's performance. Another type of network, the high order perceptron, does not make use of hidden layers but combines inputs by so-called high order connections. A high order connection combines inputs using a *splicing function*, which, in our case, is a multiplication. This way of modeling the problem resembles a *Taylor series expansion* and is known in signal processing as a *Volterra filter*. The output of the network, y_i, can be written as given in equation 1:

$$y_i = \varphi\Big(W_0 + \sum_i W_i x_i + \sum_{i,j} W_{ij} x_i x_j + \sum_{i,j,k} W_{ijk} x_i x_j x_k + \ldots\Big) \quad (i \neq j) \tag{1}$$

In this equation, φ is the activation or transfer function, y_i is the i-th output, W the weight assigned to a connection, and x_i is the i-th input. The first two terms of this equation are the same as for a standard perceptron, the bias W_0 and the first order connections $\sum_i W_i x_i$. The next terms are the second, third, and higher order connections.

In these high order perceptrons only an input and an output layer are required, the sizes of which are completely defined by the problem. This makes these networks much easier to construct than multilayer perceptrons. Having no hidden units also means that a simpler learning rule can be used. The learning rule for standard perceptrons is the delta-rule which has been extended to the generalized delta-rule, or backpropagation learning rule, for multilayer perceptrons. The higher order perceptrons do not make use of hidden units which means that the delta-rule can be used as a learning rule. However, there is a tradeoff, because for high order perceptrons the order of the problem has to be estimated.

Using higher order connections has the disadvantage that an enormous amount of connections are possible because in a fully connected network there is an exponential relationship between the number of connections and the number of neurons in the input layer [Fiesler-97]. However a fully connected network is not needed, as partially connected

networks perform very well [Lee-86]. In fact, a network is needed that is just big enough to map the data; in [Fiesler-93] these networks are called *minimal networks*. For small, theoretical problems such as the XOR-problem [Rumelhart-86], the minimal topology can be found and proven to be the smallest. For real world problems used in this paper, the minimality of a topology can not be proven, so the topologies sought are to be as small as possible.

As with multilayer perceptrons, the construction of these networks can be done using ontogenic methods, letting the network adapt and learn the problem-specific network topology. Existing growing methods for multilayer perceptrons primarily add units to the hidden layers, which means that these methods are not useful for higher order perceptrons. On the other hand, pruning methods for multilayer perceptrons delete units and/or connections from the network which means that they can also be used for pruning higher order perceptrons [Thimm-95].

The problem for these kind of ontogenic methods is that when determining the merit of a connection, all connections will have to be taken into consideration. In fact the total amount of connections can be infinite, as will be discussed in the following section. In [Visscher-96.1] the amount of possible connections was restricted and in this paper the use of this restriction will be compared to not using it.

2 High Order Connections

As discussed in the introduction, high order connections combine inputs. This combining of inputs has the disadvantage that there is an exponential relationship between the number of possible connections and inputs. For problems with a lot of inputs this means a huge computational load. Although through the use of ontogenic methods the amount of connections in the final network can be restricted, the difficulty remains that growing methods based on an a priori calculation of the best connection to use will have a complexity proportional to the total amount of possible connections. These growing methods use an ordered list of connections, therefore the values have to be calculated for all connections before they can be sorted.

In [Visscher-96.1] a choice was made to restrict the number of connections through only combining different inputs. In equation 2, N_1 denotes the number of neurons in the input layer and N_2 the number of neurons in the output layer.

$$W = N_2 \sum_{i=1}^{\Omega} \binom{N_1}{i} \tag{2}$$

The relation between number of weights W and the number of neurons in the input layer N_1 is exponential but has an upper bound for the maximum order $\Omega = N_1$, if inputs are only allowed to combine with other inputs to form a high order connection. If this restriction is not applied, the order can go to infinity thus allowing an infinite amount of connections.

Not using this restriction on using an input more than once means an enormous potential increase in the computational load of calculating the best connection to add in an a priori growing method. The networks found without applying any restrictions on the possible combination will thus have to be significantly smaller than the networks found when this restriction applies to render it worthwhile.

3 Description of the Ontogenic Methods

The process of constructing a network using ontogenic methods starts with defining an initial topology. Several different possibilities exist ranging from a topology with only the bias connection(s) to all connections up to a certain order and adding higher order connections or replacing lower order connections with higher order ones. Here a bare topology with only bias connections was chosen. This bare topology is too small to learn the input-output combinations and connections have to be added. The addition of connections means that the network gets more information which reduces the squared error in the trainingset.

Connections are added according to an a priori growing method which gives an evaluation for a connection using a certain heuristic. The connections are then ordered to ensure that the best (relative to the heuristic) connections are added first. The number of possible connections means that the methods used should ideally be as simple as possible. Therefore three heuristics have been used based on the variance of the input, the correlation between the

input and output and a combination of correlation and variance. As a reference random adding of connections is used.

The variance heuristic calculates the variance of the input data for a given connection:

$$V_{x-y} = VAR(X) \qquad V_{x_1,x_2-y} = VAR(X_1 * X_2) \tag{3}$$

The first of the two equations calculates the variance for a first order connection with input x connected to output y. The capital X denotes the data corresponding with input x. The second calculates the variance for a second order connection with inputs x_1 and x_2. For a second order connection the corresponding input variables are first multiplied, because the splicing function is a multiplication, and the outcome of this multiplication is used to compute the variance.

The largest value for the variance is put at the top of the list because the hypothesis is that the larger the variance is the more information might be contained in the data. Or the other way around: if an input variable has zero variance this means that the input is always the same irrespective of the output and is thus bound to convey little information.

For the correlation heuristic the correlation coefficients between the input and output variables of the data, and hence the corresponding connections, are calculated. For first order connections this comes down to the correlation between the corresponding input variable and output variable. For a second order connection the corresponding input variables are first multiplied, and the result of the multiplication is used to calculate the correlation coefficient.

$$r_{x-y} = CORR(X, Y) \qquad r_{x_1,x_2-y} = CORR(X_1 * X_2, Y) \tag{4}$$

Here the r_{x-y} denotes the correlation coefficient for a first order connection from x to y, the capital X denotes the data for input x and equivalently Y denotes the data for the output of the connection y. Similarly r_{x_1,x_2-y} is the correlation coefficient for a second order connection with inputs x_1 and x_2. The correlation coefficient ranges between $[-1, 1]$, but for a neural network a negative correlation is the same as a positive correlation save a different sign for the associated weight. Hence an absolute value for the correlation coefficient will be taken.

The correlation computes the linear dependency of the input and the output. A correlation coefficient that approaches 1 indicates that there is a large dependency between the input and the output. In the same way a zero correlation coefficient indicates that the variables are linearly independent. Higher absolute values for the correlation coefficient might be an indication that the input(s) associated with that connection give more information about the output than a connection with a lower value.

After viewing the results for size and generalization of the correlation and variance heuristic a combination of the two heuristics was constructed. The calculations are exactly the same for the variance and correlation given above but the two values are combined in such a way that connections that have a large variance *and* a large correlation are added first. The reason for using the combined method is that besides finding a minimal network it is also important to have a robust heuristic. A heuristic is sought that performs well without having a lot of outliers. Although it might not always find the smallest solution it will always find a good solution.

As described in [Visscher-96.1], besides using the addition of connections to avoid local minima, rerandomizing all weights when a new connection is added might also aid in avoiding local minima and thus finding smaller networks. The results confirmed this and the same will be done here to see if it also applies to these heuristics. Note that rerandomization only takes place during the growing phase.

After the network training has reached a certain error criterion, the growing process is stopped and the pruning process starts. Pruning is done because the network might have added too many connections and the pruning process might be able to remove these superfluous connections. It is generally thought that pruning these connections improves the generalization ability of the networks because superfluous connections deteriorate the network's performance by adding non-informative data to the network. These extra connections can cause the network to over-fit the training data which means that it might be optimal for the given training set but not for the unseen test set. Using less connections also reduces the dimensionality and thus the possibilities of the network to over-fit the training set. For pruning, the *smallest variance method* is used [Sietsma-91]. This is a very simple method, shown to perform very well for high order perceptrons [Thimm-95].

4 Simulations

The construction of a higher order perceptron starts with initializing a network with bias connections only. The training starts, and after a certain amount of training cycles when a *mean squared error* criterion for the training error is not reached, an extra connection is added according to one of the heuristics discussed above. The point in time to add a connection is determined by a minimal decrease in the error slope[1] which is calculated over a certain amount of training iterations. This process is continued until the convergence criterion is reached (see table 1 for the error criterion for each of the datasets). When the criterion is reached the pruning process starts using the smallest variance method as discussed above. Connections are removed and a check is made to see if the error criterion is still satisfied. If this is the case more connections are removed. However if the criterion is no longer satisfied, training takes place until the criterion is satisfied again. The pruning stops when the training error does not reach the criterion and the error slope is smaller than the minimal error slope.

Six real-world data sets were chosen, most of which were obtained (if not stated otherwise) from an anonymous-ftp server at the University of California [Murphy-94], and which are described below. The name of the data set is followed by the number of input and output variables of the problem, which also determines the number of input and output units of the network.

Solar (12,1) contains sunspot activity for the years 1700 to 1990. The task is to predict the sun spot activity for one of the years, given the preceding twelve years. The real-valued input and output data are scaled to the interval [0, 1].

Glass (15,1) consists of 8 scaled weight percentages of certain oxides found in the glass, the ninth input is a 7-valued code for the type of glass (eg.tableware, head lamps etc.). The input is scaled to [-1, 1]. The output is the refractive index of the glass, scaled to [0, 1].

Wine (13,3) is the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The 13 real-valued input values are scaled to the interval [-1, 1], the output values are boolean and scaled to [-1, 1].

Servo (12,1) was created by Karl Ulrich (MIT) in 1986 and contains a very non-linear phenomenon: predicting the rise time of a servomechanism in terms of two (continuous) gain settings and two (discrete) choices of mechanical linkages. The input is coded into two groups of five boolean values each, and two discrete inputs, one assuming four, the other five values. The output is real-valued, and like all real-valued inputs, scaled to the interval [0, 1].

Auto-mpg (7,1) concerns city-cycle fuel consumption of cars in miles per gallon, to be predicted in terms of three multi-valued discrete and 4 continuous attributes. Input data is scaled to the interval [-1, 1], output to [0,1].

British Vowels (10,11) was created by Tony Robinson (CMU) and concerns speaker independent recognition of the eleven steady state vowels of British English using a specified training set of lpc derived log area ratios. Input data type is real and scaled to the input interval [-1, 1], output is boolean and scaled to [-1, 1].

Parameter settings were all taken from [Thimm-96] and are summarized in table 1. The benchmark name is given followed by the number of connections in a fully connected second order network. For the maximum order 2 is taken because of computational constraints and in [Thimm-96] 2-nd order networks performed very well for these benchmarks. For the training error criterion either the mean square error or a percentage of wrong classifications tolerated on the training set is taken, depending on the kind of benchmark. The initial weight distribution is the same for all data sets: uniform with initial weight variance 10^{-5} and therefore not listed in the table. According to [Thimm-96], for higher order perceptrons a very small value for the initial variance is better than a bigger value.

[1] The decrease in the error slope is determined by calculating the mean error over 20 iterations and comparing it to the mean of the previous 20 iterations. Is the difference between the two smaller than a defined minimal decrease, a connection is added.

benchmark	full 2-nd	learning rate	momentum	activation function	error criterion
Solar	79	0.2	0.5	linear	0.05
Glass	121	0.5	0	stand. sigmoid	0.03
Wine	273	2.5	0	tanh	10%
Servo	79	4.5	0	stand. sigmoid	0.05
Auto-MPG	28	0.1	0.3	linear	0.06
Vowels	616	0.005	0.3	tanh	10%

Table 1: Parameters for each dataset.

5 Results and Discussion

The results of the experiments using inputs multiplied with itself (in this case quadratic connections because the order is two) will be compared to results found for experiments not using quadratic connections. In table 2 and table 3 the results of the experiments on the size criterion are shown. Each experiment consists of 50 simulations and the results for the size of the network are considered to be normally distributed. For each experiment a 95% confidence interval is calculated to enhance comparison. The first column gives the name of the dataset. The second column states if all weights were rerandomized after introducing a connection. The subsequent columns give the outcome for the different heuristics used, first the outcome of the experiment without quadratic connections $Q-$ and subsequently with quadratic connections $Q+$. The format is: the mean size of the networks found for that specific experiment followed by the confidence interval. Bold face indicates the smaller network when comparing $Q+$ to $Q-$ for the given heuristic. Bold face is only used when there is no overlap in the confidence interval.

bench-mark	rerand. weights	heuristic							
		Random		Correlation		Variance		Combined	
		$Q-$	$Q+$	$Q-$	$Q+$	$Q-$	$Q+$	$Q-$	$Q+$
Solar	yes	36.7± 3.0	40.8± 3.2	**15.6**± 0.1	16.5± 0.1	17.2± 0.2	**15.0**± 0.2	16.1± 0.1	**12.2**± 0.2
	no	36.9± 2.8	38.8± 3.1	**16.1**± 0.1	17.0± 0.0	17.1± 0.1	**15.1**± 0.1	16.1± 0.1	**12.2**± 0.1
Glass	yes	21.8± 2.1	21.7± 2.3	**13.0**± 0.0	15.0± 0.0	**51.0**± 0.0	52.0± 0.0	**19.0**± 0.0	20.0± 0.0
	no	24.3± 1.9	23.0± 1.8	13.0± 0.0	**10.0**± 0.0	**51.9**± 0.1	53.0± 0.1	**19.0**± 0.0	20.0± 0.0
Wine	yes	37.2± 2.0	38.6± 2.2	**20.3**± 0.1	21.2± 0.2	23.3± 0.1	23.9± 0.3	**18.0**± 0.4	21.0± 0.3
	no	29.5± 1.6	32.4± 2.3	**17.2**± 0.0	19.6± 0.2	**21.3**± 0.2	22.7± 0.1	**16.4**± 0.2	19.2± 0.2
Servo	yes	33.6± 2.8	36.0± 2.8	**23.0**± 0.0	28.0± 0.0	**23.1**± 0.1	38.4± 0.6	**24.0**± 0.0	**31.0**± 0.0
	no	35.3± 2.6	38.1± 2.9	**23.0**± 0.0	28.0± 0.0	**22.0**± 0.0	33.0± 0.1	**24.0**± 0.0	**31.0**± 0.0
Auto-MPG	yes	20.9± 1.2	22.0± 1.4	**18.6**± 0.2	22.2± 0.1	**12.3**± 0.1	19.4± 0.3	**12.3**± 0.1	18.6± 0.9
	no	21.7± 1.1	21.4± 1.2	**19.0**± 0.2	22.2± 0.2	**12.6**± 0.1	21.1± 0.1	**13.1**± 0.2	23.9± 0.3
Vowels	yes	306.4± 2.8	326.0± 21.3	295.4± 1.5	**278.0**± 2.6	**287.7**± 2.4	376.0± 0.0	307.7± 6.5	**258.0**± 7.1
	no	436.6± 21.1	398.9± 40.4	292.4± 8.9	285.0± 14.1	**392.2**± 4.3	-/-	313.6± 3.7	**281.0**± 9.8

Table 2: Summary of the results for each dataset after growing has taken place.

The results of the experiments after the growing phase, table 2, show that using the quadratic connections have a negative impact on the size of the resulting networks. Overall the tendency is that not using quadratic connections produces smaller networks.

The few instances where $Q+$ gives better results using the correlation heuristic are for the *Glass* benchmark without rerandomizing the weights and for the *Vowels* benchmark. The variance heuristic gives better results for the *Solar* benchmark but the *Vowels* benchmark without using rerandomization of the weights gives no results. This is probably due to the fact that the combination of the quadratic connections with the variance heuristic and no rerandomizing of the weights allows the network to get trapped in a local minimum, the network has insufficient connections to get out of the minimum and reach a desired solution. For the combined heuristic smaller networks are found for the benchmarks *Solar* and *Vowels*.

bench-mark	rerand. weights	heuristic							
		Random		Correlation		Variance		Combined	
		$Q-$	$Q+$	$Q-$	$Q+$	$Q-$	$Q+$	$Q-$	$Q+$
Solar	yes	16.7± 1.6	15.6± 1.8	**7.5**± 0.2	9.2± 0.2	**8.1**± 0.2	9.9± 0.4	11± 0.1	**8.8**± 0.2
	no	19.4± 1.8	19.0± 2.2	**13.0**± 0.0	14.7± 0.4	**7.9**± 0.1	10.0± 0.4	11.7± 0.3	**9.0**± 0.1
Glass	yes	10.9± 1.2	10.3± 1.1	5.0± 0.0	6.2± 0.3	10.0± 0.7	9.5± 0.5	7.6± 0.1	7.8± 0.1
	no	12.1± 1.2	11.4± 1.1	7.1± 0.8	8.0± 0.1	16.2± 0.8	15.3± 0.5	10.6± 0.6	11.7± 0.5
Wine	yes	18.7± 1.3	18.4± 1.1	10.9± 0.3	11.1± 0.2	**13.4**± 0.2	15.8± 0.7	11.9± 0.1	12.0± 0.1
	no	19.5± 1.3	21.4± 1.6	13.5± 0.3	13.3± 0.3	17.5± 0.6	14.6± 0.7	13.0± 0.2	14.6± 0.3
Servo	yes	14.9± 1.2	15.3± 1.4	**13.4**± 0.2	14.0± 0.1	20.8± 0.7	**18.0**± 0.6	14.7± 0.4	**11.9**± 0.1
	no	18.4± 1.4	18.9± 1.6	**10.0**± 0.0	**9.8**± 0.1	17.2± 0.1	17.8± 0.7	10.2± 0.1	9.8± 0.4
Auto-MPG	yes	11.5± 1.0	11.6± 1.0	**9.3**± 0.2	12.8± 0.3	**10.8**± 0.3	12.3± 0.5	**10.6**± 0.3	11.5± 0.4
	no	12.1± 1.0	12.1± 0.8	**10.7**± 0.2	**10.6**± 0.4	**10.5**± 0.2	14.7± 0.2	**11.3**± 0.2	14.3± 0.2
Vowels	yes	275.6± 3.6	**206.0**± 17.7	**182.3**± 11.8	**172.2**± 24.0	**194.8**± 1.9	260.9± 21.1	212.6± 15.4	**200.6**± 27.3
	no	229.2± 23.1	229.6± 38.5	210.2± 9.1	194.6± 29.6	**245.0**± 8.1	$-/-$	222.9± 11.4	**195.8**± 10.4

Table 3: Summary of the results for each dataset after pruning has taken place.

Looking at the results for the experiments after the pruning phase, table 3, $Q-$ still tends toperform better than $Q+$ although the advantage is less than after the growing phase. The correlation and variance heuristic most often give the smallest networks when no quadratic connections are used. For the combined heuristic there is no difference, $Q-$ and $Q+$ both give the same amount of smallest networks.

In table 4 the results for the generalization are given for the experiments. For most datasets the results are the mean squared error on the test set. Datasets marked by a * use a percentage measure as error criterion on the test set. The simulation results on the generalization are considered to be normally distributed and a 95% confidence interval is given. The best generalization performance for a heuristic is printed in bold font. For data sets using the mean square error this means that the smaller the value the better. For data sets using a percentage the higher the value the better the performance is. Bold font is only used when the confidence intervals do not overlap.

bench-mark	rerand. weights	heuristic							
		Random		Correlation		Variance		Combined	
		$Q-$	$Q+$	$Q-$	$Q+$	$Q-$	$Q+$	$Q-$	$Q+$
Solar	yes	0.0865± 0.0026	0.0878± 0.0029	0.0794± 0.0009	0.0801± 0.0005	0.0732± 0.0005	0.0755± 0.0007	0.0785± 0.0007	0.0808± 0.0006
	no	0.0918± 0.0031	0.0937± 0.0031	0.0816± 0.0005	0.0814± 0.0005	0.0731± 0.0003	0.0758± 0.0006	0.0796± 0.0007	0.0809± 0.0005
Glass	yes	0.0378± 0.0014	0.0370± 0.0013	0.0367± 0.0003	0.0372± 0.0003	0.0444± 0.0008	0.0452± 0.0005	0.0431± 0.0003	0.0433± 0.0003
	no	0.0359± 0.0013	0.0370± 0.0014	**0.0307**± 0.0003	0.0339± 0.0004	0.0426± 0.0009	0.0422± 0.0009	0.0468± 0.0006	0.0470± 0.0005
Wine *	yes	0.749± 0.017	0.752± 0.014	0.791± 0.009	0.795± 0.006	0.808± 0.011	0.801± 0.009	0.759± 0.009	0.766± 0.006
	no	0.735± 0.016	0.739± 0.017	0.774± 0.003	0.752± 0.007	0.730± 0.004	0.798± 0.012	0.771± 0.005	0.781± 0.005
Servo	yes	0.0791± 0.0032	0.0824± 0.0044	0.0816± 0.0005	0.0789± 0.0006	**0.0629**± 0.0006	0.0698± 0.0008	0.0684± 0.0006	0.0651± 0.0006
	no	0.0921± 0.0004	0.0917± 0.0051	0.0852± 0.0004	0.0856± 0.0004	0.0653± 0.0006	0.0657± 0.0005	0.0813± 0.0005	0.0747± 0.0013
Auto-MPG	yes	0.0595± 0.0007	0.0588± 0.0008	0.0623± 0.0002	**0.0566**± 0.0002	0.0623± 0.0002	**0.0588**± 0.0002	0.0623± 0.0003	**0.0581**± 0.0003
	no	0.0598± 0.0009	0.0593± 0.0008	0.0628± 0.0003	0.0566± 0.0001	0.0653± 0.0003	0.0696± 0.0003	0.0631± 0.0003	0.0593± 0.0002
Vowels *	yes	**0.458**± 0.003	0.405± 0.020	0.465± 0.003	0.490± 0.012	0.519± 0.003	0.494± 0.014	0.498± 0.003	0.496± 0.006
	no	0.375± 0.007	0.417± 0.022	0.453± 0.005	**0.503**± 0.011	**0.489**± 0.003	–	0.496± 0.004	0.471± 0.008

Table 4: Summary of the results for the generalization performance of the final networks.

For the results of the generalization the following has to be taken into account. In [Thimm-95] it was found that the smallest network is not always the network with the best generalization because the generalization performance shows an erratic behaviour during the pruning phase. Therefore it is possible that the method A that performs best on the size criterion might find a worse generalization compared to a method B, that finds bigger networks after pruning. However a slightly bigger network for method A, buts still smaller than the one for method B, might give a comparable, or better generalization than the network found by method B. If, however, the smaller network also

finds the better generalization a better conclusion can be taken whether or not one heuristic performs better than another taking both network size and generalization into account.

For the generalization performance there is not much difference between the performance of the networks constructed using quadratic connections and networks not using quadratic connections. When quadratic connections are used, the generalization is better in 25 experiments and in 23 experiments when no quadratic connections are used.

Besides looking at these results it is also interesting to know if connections with connections of the same input are used in the final networks. First of all during the growing phase the quadratic connections are introduced in the network which results in bigger networks. Secondly after pruning has taken place the networks still contain these quadratic connections although without them better or at least similar results can be obtained. These quadratic connections thus seem to impair the networks performance in the growing phase and pruning phase and does not result in a better generalization performance.

6 Conclusions

In this paper the use of of a retriction on the order of a high order perceptron and thus on the total amount of possible connections was investigated. Without this restriction an infinite amount of connections would be possible, see section 2, whereas with this restriction the order is bound by the amount of inputs. The number of possible connections already is a problem for high order perceptrons and is the reason for finding methods to construct partially connected networks. For the restriction to be a viable alternative the networks constructed using this restriction will have to perform as well or better than networks constructed without using the restriction. For this paper a restriction to the order was taken to be two and hence the term quadratic connections is introduced.

The results show that the use of quadratic connections generally gives worse results on the network sizes. Although the differences are slight it clearly does not give smaller networks. The results on the generalization performance is similar which implies that overall it can be concluded that the use of quadratic connections is not necessary and that the restriction on the order can be used.

Acknowledgements

This research was made possible, in part, thanks to funding by FORMITT.

References

[Fiesler-92] E. Fiesler, Partially Connected Ontogenic High Order Neural Networks. Tech-Report 92-02, IDIAP, Martigny, Switzerland, 1992.

[Fiesler-93] E. Fiesler, Minimal and High Order Neural Network Topologies. *Proc. of the Fifth Workshop on Neural Networks*, pp. 173-178, San Diego, California, 1993.

[Fiesler-94.1] E. Fiesler, Neural Network Classification and Formalization. In J. Fulcher (ed.), *Computer Standards & Interfaces*, vol. 16, num. 3, special issue on Neural Network Standardization, pp. 231-239. North-Holland/Elsevier, 1994. ISSN: 0920-5489

[Fiesler-94.2] E. Fiesler, Comparative Bibliography of Ontogenic Neural Networks. *Proc. of the International Conference on Artificial Neural Networks (ICANN 94)*, pp. 793-796, Sorrento, Italy, 1994.

[Fiesler-97] E. Fiesler and R. Beale, *Handbook of Neural Computation.* Institute of Physics and Oxford University Press, New York, New York, 1997. ISBN: 0-7503-0312–3 and 0-7503-0413-8.

[Lee-86] Y. C. Lee, G. Doolen, H. Chen, T. Maxwell, H. Lee, and C. L. Giles. Machine Learning Using a Higher Order Correlation Network.*Physica D: Nonlinear Phenomena*, volume 22, pages 276-306, 1986. ISSN: 0167-2789.

[Murphy-94] Data made available in 1994 by librarians P. M. Murphy and D. W. Aha from the UCI Repository of Machine Learning Databases, a machine-readable data repository accessible via anonymous-ftp: ftp://ftp.ics.uci.edu/pub/machine-learning-databases.

[Prechelt-95] L. Prechelt. Adaptive Parameter Pruning in Neural Networks. Tech. Report 95-009, International Computer Science Institute, Berkeley, California, 1995.

[Rumelhart-86] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group.*Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* The MIT Press, Cambridge, Mass., 1986. ISBN: 0-262-18120-7

[Sietsma-91] J. Sietsma and R. J. F. Dow. Creating Artificial Neural Networks that Generalize. *Neural Networks*, vol.4, num.1, pp.67-69, 1991.

[Thimm-95] G. Thimm and E. Fiesler. Evaluating Pruning Methods. *1995 International Symposium on Artificial Neural Networks (ISANN'95)*, pp. 20-25, 1995.

[Thimm-96] G. Thimm and E. Fiesler. Weight Initialization in Higher Order and Multi-Layer Perceptrons. Accepted for publication in *IEEE Transactions on Neural Networks*, 1996.

[Visscher-96.1] R. M. Visscher, E. Fiesler, and G. Thimm, Superceptron Construction. *Proc. of SIPAR '96*, pp. , Geneva, 1996.

Heuristics for the Ontogenic Construction of High Order Perceptrons

R. M. Visscher and E. Fiesler

IDIAP, CP 592, CH-1920 Martigny, Switzerland

E-mail: Robbert@tcw2.ppsw.rug.nl

http://www.idiap.ch/nn.html

Abstract

The major drawback of high order perceptrons is the exponential number of possible connections. Partially connected networks offer an answer to this problem and can be constructed using ontogenic methods. In this paper three heuristics for determining which connection to add to the network are investigated. These heuristics are based on maximizing the information content by starting with adding connections from an input that might minimize the error most. The heuristics are the correlation coefficient, mutual information and a method based on the Wiener-Hopf equations.

Mutual information measures the general dependency between two variables, contrary to only the linear dependencies for the correlation coefficient. The heuristics based on the Wiener-Hopf equations not only take into account the correlation between the input and the output of a connection, but also between the inputs of the different connections.

Keywords: high order perceptrons, growing, pruning, generalization, ontogenic neural networks , partially connected networks, input selection, rerandomization, Mutual Information, Wiener approximation .

1 Introduction

High order perceptrons are a relatively new kind of neural network that have advantages over the popular multilayer perceptrons. An important problem for the practical application of multilayer perceptrons is that they require knowledge of the problem to be solved and knowledge of neural networks. One of the biggest problems is finding the number of hidden layers and number of neurons per hidden layer. This problem makes the usage of multilayer perceptrons very user-unfriendly.

A promising way of selecting the number of hidden layers and neurons per hidden layer is to make use of the network's ability to adapt and to let it find the topology itself. The hidden layers are constructed by either adding units to the network or deleting units from the network whenever necessary. These methods are called *ontogenic* methods [Fiesler-97] and often find smaller and thus more efficient networks. However the ideal ontogenic method is not found yet. Moreover, these ontogenically constructed multilayer perceptrons still have hidden layers which make it hard to analyse the network's performance. Another type of network, the high order perceptron, does not make use of hidden layers but combines inputs by so-called high order connections. A high order connection combines inputs using a *splicing function*, which, in our case, is a multiplication. This way of modeling the problem resembles a *Taylor series expansion* and is known in signal processing as a *Volterra filter*. The output of the network, y_i, can be written as given in equation 1:

$$y_i = \varphi\Big(W_0 + \sum_i W_i x_i + \sum_{i,j} W_{ij} x_i x_j + \sum_{i,j,k} W_{ijk} x_i x_j x_k + \ldots\Big) \quad (i \neq j) \tag{1}$$

In this equation, φ is the activation or transfer function, y_i is the i-th output, W the weight assigned to a connection, and x_i is the i-th input. The total number of possible combinations is limited as high order connections need not use the same input more than once [Visscher-97.2]. The first two terms of this equation are the same as for a standard perceptron, the bias W_0 and the first order connections $\sum_i W_i x_i$. The next terms are the second, third, and higher order connections.

In these high order perceptrons only an input and an output layer are required, the sizes of which are completely defined by the problem. This makes these networks much easier to construct than multilayer perceptrons.

Having no hidden units also means that a simpler learning rule can be used. The learning rule for standard perceptrons is the delta-rule which has been extended to the generalized delta-rule, or backpropagation learning rule, for multilayer perceptrons. The higher order perceptrons do not make use of hidden units which means that the delta-rule can be used as a learning rule. However, there is a tradeoff, because for high order perceptrons the order of the problem has to be estimated.

Using higher order connections has the disadvantage that an enormous amount of connections are possible because in a fully connected network there is an exponential relationship between the number of connections and the number of neurons in the input layer [Fiesler-97]. However a fully connected network is not needed, as partially connected networks perform very well [Lee-86]. In fact, a network is needed that is just big enough to map the data; in [Fiesler-93] these networks are called *minimal networks*. For small, theoretical problems such as the XOR-problem [Rumelhart-86], the minimal topology can be found and proven to be the smallest. For real world problems used in this paper, the minimality of a topology can not be proven, so the topologies sought are to be as small as possible.

As with multilayer perceptrons, the construction of these networks can be done using ontogenic methods, letting the network adapt and learn the problem-specific network topology. Existing growing methods for multilayer perceptrons primarily add units to the hidden layers, which means that these methods are not useful for higher order perceptrons. On the other hand, pruning methods for multilayer perceptrons delete units and/or connections from the network which means that they can also be used for pruning higher order perceptrons [Thimm-95].

In earlier investigations three simple heuristics, among which the correlation coefficient, for determining which connections to add have been used as a basis for a simple a priori growing method [Visscher-96.1]. The heuristics used are based on the fact that connections that are expected to decrease the error of the network the most need to be added to the network first. The heuristics investigated in this paper are based on the same principle and are the Mutual Information measure and a method derived from a linear approximation of the higher order perceptron, the *Wiener filter*. The basics of these heuristics will be discussed first. Next, the results for the different heuristics will be discussed in relation to the size of the network found; the smallest one being the best. The size however is not the only important criterion. The goal of a neural network is to be able to work well with new data and to apply the learned relationships to new situations. This ability is known as the *generalization ability* and is measured by the error of the network on so called test data.

2 Correlation and Mutual Information

A well known statistical measure to quantify the dependency between two variables is the coefficient of correlation. This coefficient gives the proportion to which changes in the first variable can be attributed to the second variable. The coefficient is always between -1 and 1 where -1 indicates that if the first variable increases the second variable decreases, 1 indicates that when the first variable increases the second variable also increases. Zero correlation indicates that the second variable remains constant when the first variable is changed. The coefficient of correlation ρ is defined as:

$$\rho = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{n(\sum x^2) - (\sum x)^2}\sqrt{n(\sum y^2) - (\sum y)^2}} \tag{2}$$

An input that has a high correlation with the output, is likely to give more information concerning the output than an input with zero correlation. In the neural network context, this means that a connection connecting an input that has a high correlation with its output is preferred over a connection with a lower correlation. For neural networks however, a high but negative value for the correlation coefficient has the same effect as a high positive value, because the weights assigned to the connections in a network can have negative values.

However, correlation only computes the linear dependency between two variables where there might be a nonlinear dependency. A measure of dependency which takes this nonlinearity into account is the *mutual information*. This is a measure from information theory which deals with the question of the amount of information associated with an event [Shannon-49].

The mutual information measure has been used in neural networks before, especially in the preprocessing stage of input reduction. In [Battiti-94] the mutual information as a method for input reduction is discussed and compared to correlation. Since mutual information measures a more general dependency between two variables, this implies that when two variables are linearly independent, they might still have a more general nonlinear dependency. A description of the mutual information will be given next.

The idea behind information theory is that rare events convey much information and common events a small amount. The amount of information conveyed by an event i is considered to be $log(1/P(i))$, where $P(i)$ is the probability of that event occurring. When there are a finite number of output classes for a certain problem and the probability of the different output classes $c = 1, ..., N$ of C occurring is $P(c)$, then the initial amount of uncertainty about the output class is known as the *entropy*:

$$H(C) = -\sum_{c=1}^{N} P(c)log(P(c)) \tag{3}$$

When a certain input feature $f = 1, ..., N$ of F is known the remaining uncertainty is defined by the *conditional entropy* $H(C|F)$. This conditional entropy can be written as a combination of the *joint entropy* $H(C, F)$ and the entropy $H(F)$. The definition of the joint entropy is the same as for the entropy, only for a combination of two variables.

$$H(C|F) = H(C, F) - H(F) \tag{4}$$

Furthermore the joint entropy never exceeds the sum of the individual entropies:

$$H(C, F) \leq H(C) + H(F) \tag{5}$$

When combining equations 4 and 5 it can be seen that the remaining uncertainty about C when F is known never exceeds the initial uncertainty of C, $H(C|F) \leq H(C)$. The mutual information $I(C, F)$ is then defined as the amount by which the knowledge provided by the input feature decreases the initial uncertainty $H(C)$.

$$I(C, F) = I(F, C) = H(C) - H(C|F) \tag{6}$$

Which can also be written as:

$$I(C, F) = I(F, C) = \sum_{c} \sum_{f} P(c, f)log\frac{P(c, f)}{P(c)P(f)} \tag{7}$$

Where $P(c)$ is the probability density for class c of C and $P(c, f)$ is the joint probability density for class c and input feature f.

3 Wiener Approximation

The basic idea from the heuristics discussed above is that a connection to be added to the network should provide as much information as possible. For this reason the correlation and the mutual information are introduced. However, these methods do not take into account the possibility that two features might convey the same or a large part of the same information. Introducing connections that add similar features does not add much extra information to the network. Hence, besides providing a lot of information a feature should also add new information to the network, which means that the dependencies between the features will also have to be taken into consideration.

The intuitive idea discussed above can also be found in the field of *linear adaptive filters* using a linear neuron model. These networks consist of input nodes, which represent the different input features $\{x_i\}_{i=1...N}$ and are connected to an Adder node by a weight. The Adder node sums the weighed inputs to produce an output y. The relation between input and output is given in equation 8. By changing the weights of this network the filter can decrease the difference between the desired response d and the system output y. The goal is to minimize this

difference and hence obtain the optimal settings of the weights. An optimal solution to this problem is known as the Wiener filter.

$$y = \sum_{i=1}^{N} W_i x_i \quad \text{(1-st order)} \tag{8}$$

This filtering problem can easily be extended to higher order perceptrons with a linear activation function. Equation 8 can be viewed as a first order connection of a high order perceptron as described in equation 1. A higher order connection can be viewed as an artificial input constructed from the real inputs x_i . The relation between input and output for a second order connection is given in equation 9.

$$y = \sum_{i,j=1}^{N} W_{i,j} x_i x_j \ \ (i \neq j) \quad \text{(2-nd order)} \tag{9}$$

A high order connection is a multiplication of two or more input features resulting in an alternative input feature for the network. Equation 9 can simply be seen as a special case of equation 8 where the subscript i is changed to incorporate all the new higher order features or connections. $i = 1, .., .N$ is changed to $i = 1, ...N, N+1, ...M$. W_i can thus be either a weight belonging to a first order connection or a weight belonging to a higher order connection. For x_i the same applies, it can be a first order feature or high order feature comprising two or more first order features features. The inequality $i \neq j$ is the restriction applied to high order connections [Visscher-97.2].

The error signal is defined as the difference between the desired response d and the actual response y:

$$e = d - y. \tag{10}$$

To be able to change the weight in order to decrease the difference between the desired response and the system output, a performance measure has to be introduced. As a performance measure the *mean-squared error* is defined:

$$J = \frac{1}{2} E[e^2], \tag{11}$$

Where E is the statistical expectation function.

Finding the smallest mean-squared error for this linear approximation of high order networks can be done using the Wiener-Hopf equations [Haykin-94]. Therefore, the equations 8, with the summation extended to M, and 10 are substituted into equation 11:

$$J = \frac{1}{2} E[d^2] - \sum_{i=1}^{M} W_i E[x_i d] + \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} W_i W_j E[x_i x_j]. \tag{12}$$

The summation and weights can be taken out of the expectations because the expectation is a linear operator. The last part of the equation should not be confused with a higher order connection comprising two inputs. It is rather a combination of two input features to be compared which could be either a first order input feature or a higher order one.

Equation 12 can be rewritten after taking following definitions into account. $E[d^2]$ is the mean-square value of the desired output and denoted by $s(d)$. $E[x_i d]$ is defined as $1/N \sum_{n=0}^{N-1} x_i(n) d(n)$, which is the cross-correlation between the desired output and an input and denoted by $r(d, x_i)$ and $E[x_i x_j]$ is then the auto-correlation between two input signals and denoted by $r(x_i, x_j)$.

$$J = \frac{1}{2} s(d) - \sum_{i=1}^{M} W_i r(d, x_i) + \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} W_i W_j r(x_i, x_j) \tag{13}$$

The minimum value for J is determined by a gradient descent with respect to the free parameters in the equation, the weights. For this value for J the optimal settings of the weights will thus be found.

When growing a higher order perceptron, not all input features are connected to the output and hence are not used in the calculation of the mean squared error. This means that besides the weights W_i the features also become free parameters and optimal features can thus be defined. From equation 13 it can be seen that, irrespective of the weights, a feature that has a high cross-correlation with the output will decrease J more than a feature with a smaller value for the cross-correlation. However, the last term indicates that a large auto-correlation with other features in the network has an increasing effect on J. Hence for a feature to have maximal effect it should be correlated with the output and uncorrelated with the features already in the network.

4 Description of the Ontogenic Methods

The process of constructing a network using ontogenic methods starts with defining an initial topology. Several different possibilities exist ranging from a topology with only the bias connection(s) to a topology which contains all possible connections up to a certain order. In this paper, a bare topology with only bias connections is chosen as an initial topology. This bare topology is too small to learn the input-output combinations, connections have to be added. The addition of connections means that the network uses new information which reduces the squared error in the training set. To be as efficient as possible, a connection is needed that adds as much new information to the network as possible.

Connections are added according to an a priori growing method which gives an evaluation of a connection using a certain heuristic. The connections are ordered such that the best (relative to the heuristic) connections are added first. Hence, the heuristics need to give an indication of the amount of information contained in an input that will be connected to the output for a certain connection. This evaluation will have to be done for all connections and because the total amount of possible connections is enormous, three simple heuristics were used in [Visscher-96.1]. Namely based on the variance of the input, the correlation between the input and the output, and a combination of these two. The conclusion from that paper was that the best performing heuristic was the one based on correlation. The heuristics above were all compared to the reference random adding of connections. In this paper the correlation heuristic will be used besides the random adding of connections.

In this paper three new heuristics will be introduced: one based on the mutual information and two based on the Wiener-Hopf equations; see sections 2 and 3. The implementation of these methods will be discussed here together with the implementation of the correlation heuristic.

Correlation

For the correlation heuristic the correlation coefficients between the input and output variables of the data, and hence the corresponding connections, are calculated see equation 2. For first order connections this is equal to the correlation between the corresponding input variable and output variable. For a second order connection the corresponding input variables are first multiplied, and the result of the multiplication is used to calculate the correlation coefficient:

$$\rho_{x \to y} = CORR(X, Y) \qquad \rho_{x_1, x_2 \to y} = CORR(X_1 * X_2, Y). \tag{14}$$

In equation 14 the $r_{x \to y}$ denotes the correlation coefficient for a first order connection from x to y, the capital X denotes the data for input, x, and Y denotes the data for the output of the connection, y. Similarly $r_{x_1, x_2 \to y}$ is the correlation coefficient for a second order connection with inputs x_1 and x_2.

Mutual Information

The calculation of the mutual information for a connection is done in a similar fashion as described above for the correlation heuristic. The implementation of the mutual information is similar to the implementation described in [Battiti-94] which also contains a proof for the reliability of this approach.

For the calculation of the mutual information, see section 2, the probability densities, $P(c), P(f)$ and $P(c, f)$ are needed. These probability densities will have to be estimated from samples contained in the data sets and can be obtained using histograms. If there are N samples, the number of times that a sample belongs to a certain

interval is counted and a value n is determined for that interval. This is done for all intervals for both input and output and from these values an approximation of the probability density can easily be calculated by dividing the number of instances in an interval by the total amount of samples $P = n/N$.

To give a good approximation using histograms, a certain amount of intervals need to be taken. In [Battiti-94] good results were obtained using 10 intervals which is also taken for this implementation. The input and output range are divided into 10 equal parts and the amount of data points falling into each interval is determined. For the input features the intervals can be denoted by $n_f; f = 1...10$ where n_f denotes the amount of instances in a certain interval. Similarly for the output classes n_c. The value for the combined occurrences n_{cf} is calculated in a similar way but for a two dimensional histogram. The probability densities are simply calculated by dividing by the total amount of samples N, $P_f = n_f/N, P_c = n_c/N$, and $P_{cf} = n_{cf}/N$.

The calculation of the Mutual Information is straightforward once the different probability densities have been calculated. The summations in equation 7 are taken over the ten different input and output intervals.

Wiener Approximation

For the implementation of the heuristics based on the linear Wiener filter, see section 3, the cross- and auto-correlations have to be calculated for the different connections. However, when comparing correlations, normalization is necessary and hence for the practical implementation of these heuristics the correlation coefficients will be used. The calculation of these correlation coefficients is discussed above. Furthermore it has to be used in an a priori growing method, constraining the implementation possibilities. An approximation will therefore be used here.

The calculation of a value for a connection according to the Wiener heuristic comprises several steps. First of all the best connection to add to the network has to be found. For this reason the connections will first be sorted according to the correlation heuristic ensuring that the most informative connections will be added first. To verify whether connections add new information, their correlation with the preceding connections in the list is calculated. For every connection a new value has to be determined based on the how much it correlates with the preceding connections. For that reason a punishment term is introduced that is defined as the mean of the correlation coefficients between the input of the connection in question and all its preceding connections divided by two, see equation 15.

$$W_{x_i \to y} = \rho_{x_i \to y} - \frac{1}{2} \sum_j \rho(X_i, X_j) \quad \text{(j<i)} \tag{15}$$

This value $\frac{1}{2}$ is directly taken from the equation 13. A connection that is highly correlated with preceding connections will thus get a high punishment term. This punishment term is then subtracted from the value of the correlation between input and output of the connection in question. After this procedure is done for all connections the list is reordered.

In fact two strategies can and have been used for calculating the punishment term. In calculating the punishment term all correlations can be taken, another possibility is to take only the positive correlations between the inputs. The reason for exploiting both these methods is that a connection that has a negative correlation with another connection might give new information, which is still desired for the network. The two different methods will be indicated with *Wiener* approximation and *Wienerpos* approximation respectively.

As described in [Visscher-96.1], besides using the addition of connections to avoid local minima, rerandomizing all weights when a new connection is added might also aid in avoiding local minima and thus finding smaller networks. The results confirmed this and the same will be done here to see if it also applies to these heuristics. Note that rerandomization only takes place during the growing phase.

After the network training has reached a certain error criterion, the growing process is stopped and the pruning process starts. Pruning is done because the network might have added too many connections and the pruning process might be able to remove these superfluous connections. It is generally thought that pruning these connections improves the generalization ability of the networks because superfluous connections deteriorate the network's performance by adding non-informative data to the network. These extra connections can cause

the network to over-fit the training data which means that it might be optimal for the given training set but not for the unseen test set. Using less connections also reduces the dimensionality and thus the possibilities of the network to over-fit the training set. For pruning, the *smallest variance method* is used [Sietsma-91]. This is a very simple method, shown to perform very well for high order perceptrons [Thimm-95].

5 Simulations

The construction of a higher order perceptron starts with initializing a network with bias connections only. The training starts, and after a certain amount of training cycles when a *mean squared error* criterion for the training error is not reached, an extra connection is added according to one of the heuristics discussed above. The point in time to add a connection is determined by a minimal decrease in the error slope[1] which is calculated over a certain amount of training iterations. This process is continued until the convergence criterion is reached (see table 1 for the error criterion for each of the datasets). When the criterion is reached the pruning process starts using the smallest variance method as discussed above. Connections are removed and a check is made to see if the error criterion is still satisfied. If this is the case more connections are removed. However if the criterion is no longer satisfied, training takes place until the criterion is satisfied again. The pruning stops when the training error does not reach the criterion and the error slope is smaller than the minimal error slope.

Five real-world data sets were chosen, most of which were obtained (if not stated otherwise) from an anonymous-ftp server at the University of California [Murphy-94], and which are described below. The name of the data set is followed by the number of input and output variables of the problem, which also determines the number of input and output units of the network.

Solar (12,1) contains sunspot activity for the years 1700 to 1990. The task is to predict the sun spot activity for one of the years, given the preceding twelve years. The real-valued input and output data are scaled to the interval [0, 1].

Glass (15,1) consists of 8 scaled weight percentages of certain oxides found in the glass, the ninth input is a 7-valued code for the type of glass (eg.tableware, head lamps etc.). The input is scaled to [-1, 1]. The output is the refractive index of the glass, scaled to [0, 1].

Wine (13,3) is the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The 13 real-valued input values are scaled to the interval [-1, 1], the output values are boolean and scaled to [-1, 1].

Servo (12,1) was created by Karl Ulrich (MIT) in 1986 and contains a very non-linear phenomenon: predicting the rise time of a servomechanism in terms of two (continuous) gain settings and two (discrete) choices of mechanical linkages. The input is coded into two groups of five boolean values each, and two discrete inputs, one assuming four, the other five values. The output is real-valued, and like all real-valued inputs, scaled to the interval [0, 1].

Auto-mpg (7,1) concerns city-cycle fuel consumption of cars in miles per gallon, to be predicted in terms of three multi-valued discrete and 4 continuous attributes. Input data is scaled to the interval [-1, 1], output to [0,1].

Parameter settings were all taken from [Thimm-97] and are summarized in table 1. The benchmark name is given followed by the number of connections in a fully connected second order network. For the maximum order 2 is taken because of computational constraints and in [Thimm-97] 2-nd order networks performed very well for these benchmarks. For the training error criterion either the mean square error or a percentage of wrong classifications tolerated on the training set is taken, depending on the kind of benchmark. The initial weight distribution is the same for all data sets: uniform with initial weight variance 10^{-5} and therefore not listed in

[1] The decrease in the error slope is determined by calculating the mean error over 20 iterations and comparing it to the mean of the previous 20 iterations. Is the difference between the two smaller than a defined minimal decrease, a connection is added.

the table. According to [Thimm-97], for higher order perceptrons a very small value for the initial variance is better than a bigger value.

benchmark	full 2-nd	learning rate	momentum	activation function	error criterion
Solar	79	0.2	0.5	linear	0.05
Glass	121	0.5	0	stand. sigmoid	0.03
Wine	273	2.5	0	tanh	10%
Servo	79	4.5	0	stand. sigmoid	0.05
Auto-MPG	28	0.1	0.3	linear	0.06

Table 1: Parameters for each dataset.

6 Results and Discussion

In table 2 the results of the experiments on minimizing the size are shown. Each experiment consists of 50 simulations and the resulting network sizes are considered to be normally distributed. For each experiment a 95% confidence interval is calculated to enhance comparison. The first column of table 2 shows the name of the dataset. The second column states if all weights were rerandomized after introducing a connection. The subsequent columns give the outcome of the different heuristics with the network size after growing has taken place and the network size after pruning has also taken place. The format is: the mean size of the networks found for that specific experiment followed the confidence interval. The different heuristics are *corr.* indicating the correlation heuristic, *MI* indicating the mutual information heuristic, *Wienerpos* indicating the Wiener heuristic only using positive correlations between the inputs for the punishment term and *Wiener* finally indicates the Wiener heuristic using all correlations between the inputs to calculate the punishment term. Bold font indicates the smallest network after growing and after pruning for either with rerandomizing or without rerandomizing, for a benchmark. Bold font is only used when there is no overlap in the confidence intervals.

bench-mark	rerand. weights	heuristic									
		network size after growing					network size after pruning				
		random	corr.	MI	Wienerpos	Wiener	random	corr.	MI	Wienerpos	Wiener
Solar	yes	36.7±3.0	**15.6±0.1**	16.0±0.2	16.1±0.1	20.4±0.2	16.7±1.6	7.5±0.2	10.0±0.2	9.8±0.2	11.7±0.4
	no	36.9±2.8	**16.1±0.1**	16.3±0.2	16.1±0.1	19.1±0.1	19.4±1.9	13.0±0.0	11.4±0.3	9.1±0.1	11.1±0.3
Glass	yes	21.8±2.1	13.0±0.0	15.0±0.0	**7.0±0.0**	12.0±0.0	10.9±1.2	5.0±0.0	5.0±0.0	5.0±0.0	5.0±0.0
	no	24.3±1.9	13.0±0.0	15.0±0.0	**7.1±0.1**	12.0±0.0	12.1±1.2	7.1±0.1	10.0±0.0	4.0±0.0	4.1±0.2
Wine	yes	37.2±2.0	20.3±0.1	32.0±0.0	**15.8±0.4**	16.7±0.3	18.7±1.3	13.7±0.1	11.6±0.2	10.4±0.1	12.0±0.2
	no	29.5±1.6	17.2±0.0	24.4±0.2	**12.0±0.0**	12.5±0.1	19.5±1.3	12.1±0.2	16.4±0.5	9.5±0.1	10.0±0.1
Servo	yes	33.6±2.8	23.0±0.0	22.0±0.1	**12.0±0.0**	15.0±0.0	14.9±1.2	13.4±0.2	11.1±0.2	10.0±0.0	9.0±0.0
	no	35.3±2.6	23.0±0.0	22.1±0.1	**12.0±0.0**	15.0±0.0	18.4±1.4	10.0±0.0	12.0±0.5	10.0±0.0	8.6±0.2
Auto-MPG	yes	20.9±1.2	18.6±0.2	15.0±0.2	17.2±0.2	18.0±0.4	11.5±1.0	9.3±0.2	9.2±0.1	12.4±0.5	13.4±0.4
	no	21.7±1.1	19.0±0.2	15.9±0.2	18.0±0.3	18.5±0.4	12.1±1.0	10.7±0.2	11.1±0.1	12.5±0.4	14.2±0.2

Table 2: summary of the results on size for each dataset.

First of all, the results on the network sizes for the mutual information heuristic will be compared to the random adding of connections and the correlation heuristic. In table 2 it can clearly be seen that the network sizes after the growing phase are all smaller than the networks constructed by randomly adding connections. However when comparing to the correlation heuristic there is not much difference in the network sizes. For *Glass* and *Wine* the correlation heuristic performs better and for *Servo* and *Auto-mpg* the Mutual information produces smaller networks. The differences are only slight except for the *Wine* benchmark.

The performance of mutual information on the final network sizes after pruning is also comparative to the correlation heuristic and no clearly better method can be determined. Hence, although the mutual information can define nonlinear dependencies, it does not seem to give clearly better results than the linear dependencies given by the correlation.

After the growing phase, the Wienerpos and Wiener heuristics always find smaller networks than randomly adding connections. When comparing to the correlation and mutual information heuristic the networks found with the heuristics based on the Wiener-Hopf equations are significantly smaller for *Glass, Wine* and *Servo*. For *Solar* the networks are comparable, for *Auto-mpg* they are comparable to correlation but slightly bigger than for the mutual information heuristic. The results of the heuristic that only uses the positive correlations between the inputs for determining the punishment term, Wienerpos, clearly outperforms the heuristic that takes all correlations into account. It always finds smaller networks.

After the pruning phase the Wiener heuristics still outperforms the correlation and Mutual information heuristics. The advantage has slightly decreased, however the Wiener heuristics still find considerably smaller networks for 6 experiments, mainly for the same benchmarks as after growing. For the *Solar* benchmark without rerandomizing all weights the Wiener heuristic using only positive correlations also did considerably better than either correlation or mutual information. However the results for the *Auto-mpg* benchmark are very poor contrary to the other results obtained. Overall, the Wienerpos heuristic is able to find the smaller networks after growing as well as after pruning for these data sets.

The advantage found for rerandomizing of weights in [Visscher-96.1] and [Visscher-97.2] for the correlation heuristic was not found for the Wiener heuristics. There seems to be a slight advantage for not rerandomizing the weights. However the results for the Mutual Information do show an advantage for rerandomizing the weights.

In table 3 the results for the generalization performance are given. The results are the *mean square errors* except for the *Wine* benchmark indicated by a *, where it is the amount of correctly classified instances. The resulting generalization performances of the simulations are considered to be normally distributed and a 95% confidence interval is given to enhance comparison. Bold font indicates the best performance for that benchmark for both with and without rerandomization and italics indicates the second best performance on generalization. Bold font and italics are only used when there is no overlap in the confidence intervals.

bench-mark	rerand. weights	heuristic				
		Random	Correlation	MI	Wienerpos	Wiener
Solar	yes	0.0865±0.0028	**0.0794**±0.0009	**0.0798**±0.0004	*0.0835*±0.0007	0.0876±0.0009
	no	0.0918±0.0031	*0.0816*±0.0005	**0.0806**±0.0006	0.0826±0.0007	0.0921±0.0009
Glass	yes	0.0378±0.0014	*0.0367*±0.0003	0.0380±0.0001	**0.0307**±0.0001	*0.0364*±0.0002
	no	0.0359±0.0013	*0.0307*±0.0003	0.0398±0.0004	*0.0307*±0.0001	**0.0296**±0.0001
Wine *	yes	0.749±0.017	0.791±0.009	0.701±0.006	**0.841**±0.007	*0.825*±0.006
	no	0.735±0.016	0.774±0.003	0.738±0.006	*0.879*±0.004	**0.886**±0.004
Servo	yes	0.0791±0.0032	0.0816±0.0005	**0.0679**±0.0002	*0.0742*±0.0002	0.0782±0.0001
	no	0.0921±0.0004	0.0852±0.0004	**0.0717**±0.0009	*0.0748*±0.0003	0.0771±0.0003
Auto-MPG	yes	**0.0595**±0.0007	0.0623±0.0002	0.0625±0.0002	*0.0615*±0.0003	*0.0619*±0.0004
	no	**0.0598**±0.0009	0.0628±0.0003	0.0624±0.0003	0.0621±0.0004	*0.0613*±0.0004

Table 3: Summary of the results for the generalization performance of the final network after pruning.

For the results on generalization the following has to be taken into account. In [Thimm-95] it was found that the smallest network is not always the network with the best generalization because the generalization performance shows an erratic behaviour during the pruning phase. It is therefore possible that a method A that performs best on the size criterion might find a worse generalization as compared to a method B that finds bigger networks after pruning. However, a slightly bigger network for method A, but still smaller than the one for method B, might give a comparable or better generalization than the network found by method B. If, however, the smaller network also finds a better generalization, a better conclusion can be reached whether or not one

heuristic performs better than another taking both network size and generalization into account.

The results for mutual information on the generalization are mixed when compared to the correlation heuristic. Although it gives best performance on generalization in four experiments, in three of those experiments the network found is also bigger than when using correlation. In all other experiments the results are poorer and furthermore the generalization performance is more often worse than the generalization performance for the random adding of connections, besides *Auto-mpg* also for the *Glass* and *Wine* benchmarks.

The two methods using the Wiener approximations show satisfactory results for the generalization performance. In four of the five experiments that the Wienerpos heuristic finds the smallest networks it also finds the best or second best generalization. This is for the benchmarks *Glass* and *Wine*. The results for the Wiener heuristic is comparable to the Wienerpos results for generalization but the Wiener heuristic performs extremely poor on the *Solar* benchmark, the network size is the biggest and the generalization performance is even worse than for the random adding of connections. On generalization the Wienerpos heuristic seems to perform best of all five heuristics.

The smallest benchmark, *Auto-mpg*, gives some odd results. The random addition of connections gives smaller networks than the Wiener heuristics and the generalization for the random addition of weights is the best overall. One reason could be that the maximum second order network of this benchmark is not much bigger than the network size after growing using the random heuristic. The fact that the random adding of connections lets the network grow quite big as compared to the total amount of connections possible, increases the chance of better connections getting into the network. During pruning, these 'better' connections are kept in the network and a good performance on size and generalization is ensured. For bigger problems the results for the random adding of connections are far worse.

7 Conclusions

In this paper, several heuristics were reviewed for their merit as a basis for a growing algorithm for higher order neural networks. In earlier papers by Visscher *et al.* it was concluded that the correlation heuristic performs satisfactory and in this paper this heuristic has been compared to some more elaborate ones. There is however a trade off between the computational complexity of the growing heuristic and its usefulness for a neural network growing method. A more complex method has to perform better than a simple method for it to be an acceptable alternative. In this paper a simple method of a priori computation of the ordering of the connections was used to get a good idea of how the different methods perform. Depending on the final growing method implemented one can choose either an efficient heuristic as correlation or a less efficient but better performing heuristic like Wiener approximation.

The first heuristic to be compared to the correlation heuristic was the mutual information heuristic which also takes the nonlinear dependencies between different variables into account. This method is however more complex than the simple method of correlation which means that if it is to be an alternative it has to increase performance on the size of the final network found and the generalization capability of this final network. There is however no clear sign of an increase in performance even though the method is more elaborate.

The heuristics based on the Wiener-Hopf equations improve performance considerably, especially the heuristic that only uses the positive correlations for the calculation of the punishment term. First of all, after the growing phase the networks are considerably smaller, making the growing procedure more efficient. Secondly, the final networks after pruning are smaller than for both correlation and mutual information. Lastly, this method also performs well on the generalization performance. The drawback is that the method is considerably more complex than either correlation or mutual information. As a heuristic for an a priori growing method it might therefore not be very suitable, but as a heuristic for an efficient growing method it is.

For further investigations into heuristics for growing methods, a Wiener heuristic based on the mutual information instead of the correlation looks interesting. As mentioned earlier one of the problems with the correlation heuristic is the fact that it only takes linear dependencies into account. The mutual information heuristic however, did not increase performance as compared to the correlation heuristic, but a Wiener heuristic based on

mutual information might prove to work even better than a Wiener heuristic based on correlation.

Furthermore, a growing method is needed that is not based on the a priori calculation of the value for all weights. A possible method might be to randomly choose a connection and verify that the heuristic value for this connection is above a certain threshold. If so, the connection can be added to the network, otherwise take another connection and do the same thing. Such a growing method is far more efficient and would allow for computationally complex heuristics.

Acknowledgements

This research was made possible, in part, thanks to funding by FORMITT.

References

[Battiti-94] R. Battiti. Using Mutual Information for Selecting Features in Supervised Neural Net Learning. *IEEE Transactions on Neural Networks*, vol. 5, no. 4, July 1994.

[Fiesler-93] E. Fiesler, Minimal and High Order Neural Network Topologies. *Proc. of the Fifth Workshop on Neural Networks*, pp. 173-178, San Diego, California, 1993.

[Fiesler-94.1] E. Fiesler, Neural Network Classification and Formalization. In J. Fulcher (ed.), *Computer Standards & Interfaces*, vol. 16, num. 3, special issue on Neural Network Standardization, pp. 231-239. North-Holland/Elsevier, 1994. ISSN: 0920-5489.

[Fiesler-94.2] E. Fiesler, Comparative Bibliography of Ontogenic Neural Networks. *Proc. of the International Conference on Artificial Neural Networks (ICANN 94)*, pp. 793-796, Sorrento, Italy, 1994.

[Fiesler-97] E. Fiesler and R. Beale, *Handbook of Neural Computation*. Institute of Physics and Oxford University Press, New York, New York, 1997. ISBN: 0-7503-0312–3 and 0-7503-0413-8.

[Haykin-94] S. Haykin. *Neural Networks; A Comprehensive Foundation*. MacMillan College Publishing Company, New York, New York, USA, 1994. ISBN: 0-02-352761-7.

[Lee-86] Y. C. Lee, G. Doolen, H. Chen, T. Maxwell, H. Lee, and C. L. Giles. Machine Learning Using a Higher Order Correlation Network. *Physica D: Nonlinear Phenomena*,

[Prechelt-95] L. Prechelt. Adaptive Parameter Pruning in Neural Networks. Tech. Report 95-009, International Computer Science Institute, Berkeley, California, 1995.

[Murphy-94] Data made available in 1994 by librarians P. M. Murphy and D. W. Aha from the UCI Repository of Machine Learning Databases, a machine-readable data repository accessible via anonymous-ftp: ftp://ftp.ics.uci.edu/pub/machine-learning-databases.

[Rumelhart-86] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. The MIT Press, Cambridge, Mass., 1986. ISBN: 0-262-18120-7.

[Shannon-49] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois, 1949.

[Sietsma-91] J. Sietsma and R. J. F. Dow. Creating Artificial Neural Networks that Generalize. *Neural Networks*, vol.4, num.1, pp.67-69, 1991.

[Thimm-95] G. Thimm and E. Fiesler. Evaluating Pruning Methods. *1995 International Symposium on Artificial Neural Networks (ISANN'95)*, pp. 20-25, 1995.

[Thimm-97] G. Thimm and E. Fiesler. Weight Initialization in Higher Order and Multi-Layer Perceptrons. *IEEE Transactions on Neural Networks*, vol. 8, num. 2, 1997.

[Visscher-96.1] R. M. Visscher, E. Fiesler, and G. Thimm, Superceptron Construction. *Proc. of SIPAR '96*, pp. , Geneva, 1996.

[Visscher-97.2] R. M. Visscher and E. Fiesler, Order Restriction in Higher Order Perceptrons, RR 97-02, IDIAP, Martigny, Switzerland, 1997.