A Neural Networks Construction Method based on Boolean Logic

Georg Thimm and Emile Fiesler
IDIAP, P.O. Box 592, CH-1920 Martigny, Switzerland.
Email: Georg.Thimm@idiap.ch

Short version accepted for publication in the
IEEE International Conference on Tools with Artificial Intelligence
proceedings,
Toulouse, France, 1996

Abstract

A neural network construction method for problems specified for data sets with in- and/or output values in the continuous or discrete domain is described and evaluated. This approach is based on a Boolean approximation of the data set and is generic for various neural network architectures. The construction method takes advantage of a construction method for Boolean problems without increasing the dimensions of the in- or output vectors, which is a strong advantage over approaches which work on a binarized version of the data set with an increased number of in- and output elements.

Further, the networks are pruned in a second phase in order to obtain very small networks.

Keywords: construction of networks, pruning, generalization, optimality criteria, high order perceptrons, backpropagation neural networks.

1 Introduction

A major problem in applying of neural networks is the choice of a *(minimal) topology* [5]: a considerable amount of architectures and methods for the construction of (almost) *optimal* neural networks have been proposed (see [7], [13], and [1] for overviews). Some of those approaches are aimed at improving well-known architectures by training a network which is expected to be big enough to solve the given problem and subsequently remove (*prune*) units [15] [11]. The main problem with this approach is that *big enough* is an *a priori* indeterminable quantity and therefore the training of a network involves a lot of trial-and-errors. The number of trials is often further increased, it is unclear whether the network does not learn, due to either a bad choice of a network topology, or an unlucky choice of learning parameters; and the user tunes the wrong parameter.

The type of neural network used in this research is the so called *high order perceptron* [14], which is basically a set of polynomials with a sigmoidal function applied to their outputs. Each connection in the network and its weight corresponds to a polynomial term and its coefficient. The order of a connection is equal to the order of the corresponding polynomial term. An example for a second order high order perceptron with two inputs and one output is:

$$net(x_1, x_2) = tanh(w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2)$$

where the hyperbolic tangent serves as sigmoidal function, the w_i are the weights and the x_i represent the neural network inputs. The training is performed with the standard backpropagation algorithm.

In the case of a training failure, the user is tempted to insert the connections order-wise, which means that from trial to trial the network grows exponentially to the number of inputs. This behavior is especially bad for data sets with a big number of input elements, as for example images, where already second order high order perceptrons are sensibly oversized and as well as the required memory and CPU time are painful. It is therefore important to construct neural networks of a reasonable size which are able to learn a data set, whereas it does not matter if the network is slightly too big.

Even though this publication focuses on high order perceptrons, the method presented below is generic in the sense that it is applicable to other neural network topologies, as many neural network architectures permit to construct a network in a straight forward manner for Boolean problems and can be trained iteratively.

2 The Approach

It is relatively easy to construct higher order perceptrons or other types of neural networks for Boolean problems, as shown for example in [3], [6] [4], [16], [10], and [2]. The networks produced by the algorithms presented in these papers differ mainly in properties like generalization performance, size, depth, weight range, etc. Unfortunately, these algorithms take advantage of the data being Boolean and are therefore not directly applicable to many applications.

A straight forward way to make these algorithms applicable to non-Boolean data is to discretise the real-valued data elements and to code each value by a Boolean vector. This approach has two major disadvantages: it requires a network significantly increased in size as more in- and outputs are necessary and therefore also the probability for a poor generalization performance increases. Consequently, the neural network construction algorithm should keep the number of in- and outputs unchanged.

A "human-like" way of doing so, is to search a set of simple logic rules in the aim to explain the basic relations between input and target vectors. These quasi-logic rules have most likely the form **if value A is big and value B is small then the output is big**. This means that the developer instinctively first applied a sort of fuzzy threshold function and tried then to find a kind of Boolean rules based on this approximation of the data. If a set of rules, which explains most of the data elements, is found, these rules are quantified and finally refined. The, from this approach inspired and following described "machine-like" method, tries to imitate this. Figure 1 shows the basic steps, described in more details below:

① the real- or discrete-valued data are transformed into a Boolean data set by thresholding the elements of the input, respectively output vectors.

② This Boolean data set is used to construct a Boolean expression, which

③ is translated into a high order perceptron. As a network constructed for the Boolean data will perform poorly on the original, real valued data set,

④ this network is trained using the backpropagation algorithm and

⑤ finally pruned.

Note that the Boolean data is only involved in the construction phase, the training is done using the original data set.

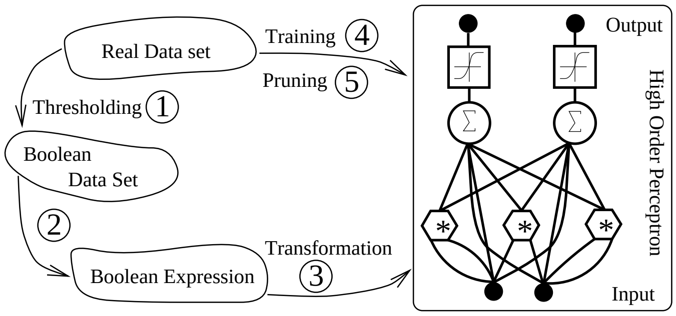

Figure 1: Construction and training scheme.

① Thresholding the Real-Valued Data Set

First, the continuous or discrete valued data set is thresholded, in order to obtain a Boolean data set which is similar to the real valued data set. The threshold value is chosen to be the mean value of each input element x_i (similar for the outputs y_j):

$$b_i = \begin{cases} true & if\ x_i > \sum x_i/N \\ false & otherwise \end{cases}$$

This threshold value has a high probability to create different Boolean vectors for different input vectors: assumed that the different input vector elements are statistically independent, the number of the original data mapped onto the same Boolean vector is approximately the size of the data set divided by two to the power of the elements in the input vector. This threshold value is not necessarily optimal in terms of creating the maximal number of different Boolean vectors in regard to a specific data set, but an optimal value is expensive to calculate[1].

It occasionally happens that different continuous input vectors are mapped onto the same Boolean vector, but do not have the same target vectors and therefore the Boolean data set does not define a function. In order to obtain a sound Boolean data set one of each pair of those contradicting data elements is omitted from the data set used for the calculation of the Boolean expression. For this research, the data for which the target vector is most contradictory to the data with similar inputs is omitted. Note that the omitted data remain in the original data set used during the training phase and there is no reason why they should not be learnable by the constructed network.

② Calculation of the Boolean Expression

A for this approach suitable algorithm was initially developed by Y. Crama *et al.* [3] and further developed, as well as implemented by E. Mayoraz [9]. This algorithm is in principle very similar to other algorithms for the calculation of Boolean expressions on the base of some Boolean vectors given. This program offers, besides being optimized in calculation time, some useful features which allow tuning the size of the initial network in a more reasonable way than simply increasing the order of the network:

- The number and order of connections can be reduced at the cost of misclassifications: the extraction of the minimal basic pattern set[2] may be done in a way that only a percentage of training patterns are well classified.

[1] This optimal threshold value is also subject to future research.

[2] A basic pattern is a conjunction of some Boolean variables or their negations.

- The order of the terms can be hard limited (by limiting the number of literals per basic pattern).
- It can be specified that each pattern in the Boolean data set is explained by at least a certain number of basic patterns.
- The Boolean expression is optimized by using the don't-care terms, which are implicitly defined by the Boolean vectors onto which no data element is mapped during step ①.

③ Creation of the network

In this phase the Boolean expressions (respectively the sets of basic patterns) found for each output are transformed into polynomials. A Boolean expression can be translated into a polynomial of the corresponding real-valued variables and coefficients which can be calculated from the precise form of the conjunction [12]. This calculation is not done here, for three reasons:

- In consideration of the crude approximation done in step ①.
- As a sigmoidal function is applied to the output of each polynomial, it is likely that some terms are superfluous (higher order terms can be pruned in the presence of lower order terms).
- In order to save computation time required for the calculation of these terms.

More precisely, each basic pattern corresponding to a certain output is translated into a polynomial with all combinations of possible terms of the involved literals and for the moment undetermined coefficients. These polynomials are then added and duplicate terms removed (sums of coefficients are replaced by a single, new coefficient). The terms in these polynomials correspond to the connections of the high order perceptron, and the coefficients to the weights. Hence, only sigmoidal functions have to be applied to each output in order to obtain a high order perceptron.

Example: suppose the minimal basic pattern set and the corresponding output vectors are:

$$\begin{array}{llllll} \bar{x}_4 & x_5 & \rightarrow y_1 & \quad x_1 & x_3 & \rightarrow y_2 \\ x_4 & \bar{x}_5 & \rightarrow y_2 & & & \end{array}$$

Then the per-pattern polynomials are (the w_i are the coefficients, respectively weights):

$$\begin{array}{ll} w_0 + w_1 x_4 + w_2 x_5 + w_3 x_4 x_5 & \rightarrow y_1 \\ w_4 + w_5 x_4 + w_6 x_5 + w_7 x_4 x_5 & \rightarrow y_1 \\ w_8 + w_9 x_1 + w_{10} x_3 + w_{11} x_1 x_3 & \rightarrow y_2 \end{array}$$

And the high order perceptron has the following form (the weights are renumbered and $tanh()$ is the sigmoidal function):

$$\begin{array}{rl} y_1(\mathbf{x}) = & tanh(w_0 + w_1 x_4 + w_2 x_5 + w_3 x_4 x_5) \\ y_2(\mathbf{x}) = & tanh(w_4 + w_5 x_1 + w_6 x_3 + w_7 x_1 x_3) \end{array}$$

④ Training and ⑤ Pruning

The training of the neural networks is performed using the backpropagation algorithm without further consideration of the fact that the network is constructed in a special way. As the algorithm described above does not give any default values

	conv.	$W_{con.}$	W_{2nd}	$W_{con.,\ pr.}$	$W_{2nd,\ pr.}$	$E_{gen.\ con.}$	$E_{gen.\ 2nd}$	Pr. M.
Solar	0.05 MSE	26	78	15.7$\pm$1	6.8$\pm$2	0.106$\pm$0 MSE	0.0810$\pm$1 MSE	$min(\sigma)$
Auto-mpg	0.06 MSE	27	36	6.06$\pm$1	8.02$\pm$1	0.059$\pm$0 MSE	0.060$\pm$0 MSE	$min(w)$
Wine r=4	100% Rec.	113	274	22$\pm$0	20$\pm$1	20.1$\pm$1 %	17.3$\pm$0 %	$min(w)$
Wine r=6	100% Rec.			19.7$\pm$2		16.1$\pm$1 %		$min(w)$
Servo	0.1 MSE	45	91	9.1$\pm$1	8.2$\pm$1	0.144$\pm$0 MSE	0.123$\pm$0 MSE	$min(w)$
Digits	99% Rec.	1,260	20,490	292$\pm$2	160$\pm$2	7.9$\pm$0 %	9.0$\pm$0 %	$min(\sigma)$
Glass	0.03 MSE	17	121	5.0$\pm$0	9.7$\pm$0	0.035$\pm$0 MSE	0.044$\pm$0 MSE	$min(\sigma)$

conv.	Convergence condition (MSE = mean square error; % Rec. = percent recognition)
$W_{con.}$ (W_{2nd})	Number of connections in constructed (2nd order) networks
$W_{con.,\ pr.}$ ($W_{2nd,\ pr.}$)	Average size of constructed (2nd order) and pruned networks
$E_{gen.\ con.}$ ($E_{gen.\ 2nd}$)	Generalization error of pruned, constructed (2nd order) networks

Table 1: Experiments

for the weights, they are initially randomized. The networks are pruned using the pruning algorithm found to produce the smallest networks in [15], which was either the smallest weight[3] or the smallest connection variance method[4], depending on the data set.

As it turns out, that the networks constructed in step 3 are not always capable to learn the task, the maximal order of the network (initially set to 2) and the coverage of the patterns (initially set to 1) are increased one by one until the training in step 4 is successful.

3 Experiments

The main aim of this section is to show that the generalization capabilities of the constructed networks are comparable to the approach with full nth-order networks but that the initial network size is considerably smaller. The required experiments are performed using six real world data sets, which are discussed in [14]. Each experiment consists of at least 50 simulations. The notation $\pm X$ indicates the statistical error, where X is the maximal difference to the last digit (assumed a normal distribution and a confidence of 95%).

As can be seen in table 1, the constructed networks have only between $frac116$ and $frac23$ of the number of connections as compared to the fully connected second order networks, justifying the approach. These networks can be further pruned with success and the final size of the network is close to the size obtained by pruning the full second order network.

The generalization of the pruned networks is sometimes worse, sometimes better as compared to the full second order networks for the same training parameters (this concerns especially the learning rate). But a different choice can change the outcome of the experiments greatly: for the wine data experiments show that a different choice of the learning rate, the proposed method produces up with networks of an on average better generalization performance and the same size than shown in the table for the full second order perceptron. Unfortunately, a complete scanning of the domain of learning parameters is hardly feasible due to a irrealisic hight demand of CPU time. It is therefor unclear whether the networks obtained by applying the proposed method or the pruning of a full second order neural network give better results for an optimal choice of the learning parameters. A comparison on the base of identical learning parameters does not permit a conclusive statement, as the different initial size an the presence of connections of an order bigger than two

[3] Which is equivalent to optimal brain damage [8] in the case of (high order) perceptrons.
[4] This method removed connections having the smallest variance for the training set.

changes the behavior of a high order perceptron and its optimal settings for the learning rate. The extreme case of that for the same choice on learning parameters the constructed high order perceptron converges, but not the full second order high order perceptron is imaginable. Therefore, the experiments performed show only that final network size and generalization performance are similar and that the new method is not greatly inferior to the native approach which prunes full second order networks.

4 Conclusion

The proposed neural network construction method is shown to significantly reduce the number of connections in the initial network (as compared to the full second order or third order network). It can be expected that this is especially true for data sets with very high dimensional feature vectors. However, the networks are not minimal and pruning the networks is advantageous. A comparison of pruned full second order networks with the pruned constructed networks showed that the sizes as well as the generalization performance of the pruned networks are comparable.

The general idea of using a Boolean approximation of the continuous valued data set to construct an initial network is an generic approach which can be applied to other neural network architectures and is beneficial if its construction from a Boolean expression has a low complexity and an iterative learning rule for the adaption of the weights like backpropagation can be used. Equally, other Boolean methods for the construction of a initial neural network may produce better results, which is subject to future research.

References

[1] Ethem Alpaydın. Gal: Networks that Grow When They Learn and Shrink When They Forget. Technical Report TR 91-032, International Computing Science Institute, 1947 Center Street Suite 600, Berkeley CA 94704-1105 USA. 1991.

[2] Valeriu Beiu and John G. Taylor. Area-efficient constructive learning algorithm. In *Proceedings of CSCS-10: the 10th International Conference on Control System and Computer Science*, vol. 5, pp. 293–310. Bucharest, Rumania, 1995.

[3] Yves Crama, Peter L. Hammer, and Toshhihide Ibaraki. Cause-Effect Relationships and Partially Defined Boolean Functions. Technical Report # 39-88, Rutcor, Rutgers University, New Brunswick, N.J. 08903, USA, 1988.

[4] G. Fahner, N. Goerke, and R. Eckmiller. Structural Adaptation of Boolean Higher Order Neurons: Classification with Parsimonious Topologies for Superior Generalization. I. Alexander *et al.* (ed), *Proceedings of the 1992 International Conference on Artificial Neural Networks*, vol. 1, pp. 285–288, Amsterdam, The Netherlands, 1992. Elsevier Science Publishers B.V.

[5] E. Fiesler. Minimal and High Order Neural Network Topologies. *Proceedings of the Fifth Workshop on Neural Networks: Academic/Industrial/NASA/Defense*, number 2204 in SPIE Proceedings, pp. 173–178, San Diego, California, 1993. Simulation Councils, Inc. / The Society for Computer Simulation.

[6] Donald L. Gray and Anthony N. Michel. A training algorithm for binary feedforward neural networks. *IEEE Transactions on Neural Networks*, vol. 3, num. 2, pp. 176–194, March 1992.

[7] Tin-Yau Kwok and Dit-Yan Yeung. Constructive Feedforward Neural Networks for Regression Problems: A Survey. Technical Report HKUST-CS95-43, Dept. of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, 1995.

[8] Y. Le Cun, J.S. Denker, and S. A. Solla. Optimal Brain Damage. *Neural Information processing Systems*, 2:598–605, 1989.

[9] Eddy Mayoraz. C++ Tools for Logical Analysis of Data. Rutcor Technical Report RTR 1-95, Rutgers University's Center for Operations Research, P.O. Box 5062, New Brunswick, NJ 08903-5062. July 1995.

[10] Andrea De Pol, Georg Thimm, and Emile Fiesler. Boolean Logic Inspired High Order Perceptron Construction. Nicolas Droux (ed.), *SIPAR Workshop'95 Parallel and Distributed Systems*, SIPAR SI Group for Parallel Systems, Biel School of Engineering, Computer Science Dept., Rue de la Source 21, CH-2501 Biel, Switzerland.

[11] Russell Reed. Pruning Algorithms – A Survey. *IEEE Transactions on Neural Networks*, vol. 4, num. 5, pp. 740–747, 1993.

[12] David E. Rumelhart, James L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. The MIT Press, Cambridge, Massachusetts, 1986.

[13] Yoan Shin and Joydeep Ghosh. Ridge Polynomial Networks. *IEEE Trans. Neural Networks*, 1995. to appear.

[14] Georg Thimm and Emile Fiesler. High Order and Multilayer Perceptron Initialization. Submitted to IEEE Transactions on Neural Networks. An extended version is available as IDIAP Technical Report 94-07, 1994.

[15] Georg Thimm and Emile Fiesler. Evaluating Pruning Methods. *1995 International Symposium on Artificial Neural Networks* , pp. A2 20–25. National Chiao-Tung University, Hsinchu, Taiwan, ROC, 1995.

[16] Rei-Yao Wu and Wen-Hsiang Tsai. A Single-Layer Neural Network for Parallel Thinning. *Int. Journal of Neural Systems*, vol. 3, num. 4, pp. 395–404, 1992.