Boolean Logic Inspired High Order Perceptron Construction

A. De Pol, G. Thimm, and E. Fiesler

IDIAP, Case postale 592, CH-1920 Martigny, Switzerland

Abstract

This paper describes and evaluates sevaral new methods for the construction of high order perceptrons. The methods are based on approximating real valued data by Boolean values and to use these as a base for the construnction of a high order neural network. The methods are evaluated for their effectiveness in reducing the network size by comparing them to fully connected N_1-th order networks and their performance is evaluated by testing the generalization capabilities of the resulting networks.

1 Introduction

A first order perceptron consists of an input and an output layer of neurons with interlayer connections between them. Their applicability is limited since they can only solve linearly separable problems. To overcome this limitation, High Order Perceptrons (HOP) can be used [Thimm-94.1]. HOPs are a generalization of perceptrons. This type of neural network has, in addition to the commonly used *first order connections*, *high order connections* where two or more input values are combined by means of a *splicing function* whose result is propagated to the output layer. Multiplication is chosen as splicing function in this publication. A HOP with a single output can be written in the following manner:

$$y = \varphi\left(\sum_{i=1}^{N_1} w_i x_i + w_{N_1+1} x_1 x_2 + w_{N_1+2} x_1 x_3 + \ldots + w_{2^{N_1}-1} x_1 x_2 x_3 \ldots x_{N_1} - \theta\right). \tag{1}$$

In this equation N_1 denotes the input layer size, w_r $(r = 1, 2, \ldots, 2^{N_1} - 1)$ the weights of the connections, θ the threshold, φ the activation function, and y the output. The order of a HOP is defined as the maximum number of inputs combined by any of the splicing function operations. This means that if the polynomial in equation (1) includes products of up to k input terms, the network is a kth-order HOP ($k \leq N_1$). The number of connections in a fully interlayer connected HOP W_{max} increases exponentially with the order of the HOP (counting only connections combining different inputs neurons):

$$W_{max} = N_2 \sum_{m=1}^{k} \left(\begin{array}{c} N_1 \\ m \end{array} \right). \tag{2}$$

This combinatoric explosion of connections can be bound by constructively adding connections to a initially small topology. Such a constructive HOP algorithm for binary problems is described in [Redding-93]. It determines a HOP with minimal order for an arbitrary binary or bipolar mapping problem, allowing perfect recognition of the training set. This method constructs a set of polynomials based upon their relevance to (that is, linear independence upon) the particular pattern set. However, this algorithm is computationally expensive and cannot be applied to real-valued problems.

Muselli describes a constructive algorithm for binary neural networks and applies it to real valued classification problems [Muselli-95]. The real data is first converted into binary data using Gray encoding, which maps proximate numbers to similar binary strings. Further, a preprocessing technique called hamming clustering is proposed for improving the generalization[1] ability. A disadvantage of this analog-to-digital conversion is the increased number of input and output elements, which implies a very large increase in the number of connections.

2 The methods

The three methods described in the following sections allow the construction of network topologies for problems with real valued inputs and outputs. The first method transforms a real valued problem into a Boolean problem without increasing the number of input and output elements. In a first step every input and output value is replaced by a Boolean value, which is *True* if the input pattern value is greater or equal than the threshold and *False* if not. The transformed data set is then used to construct a network. Finally, the resulting network is trained with the real valued problem.

The second method is based upon the first, but instead of all terms, only those with the lowest order are chosen for constructing the HOP.

The third method transforms a real valued data set into a data set with binary inputs where each input is represented by more than one bit, in the aim of extracting more information from the data. The transformed data set can be seen as the values a function assumes on the vertices of a hypercube. Next, a polynomial is constructed agreeing with the values this function assumes on the vertices of the hypercube. Finally, only the lowest order terms of this polynomial are used, as in the second method.

[1] The generalization ability is the capacity of the neural network, after being trained on a number of examples (training patterns), to interpolate and extrapolate on unseen patterns.

2.1 First method

In the first step of this method, the real-valued patterns set is transformed into Boolean values. Every pattern value is replaced by a Boolean value, *True* if the input value is bigger than a threshold and *False* if not. For example the two patterns:

Data			Target		Data			Target
0.1	0.2	$\rightarrow$	0.6	are transformed into	*False*	*False*	$\rightarrow$	*True*
0.55	0.4	$\rightarrow$	0.2		*True*	*False*	$\rightarrow$	*False*

In this example the threshold value for the transformation is chosen to be 0.5, but other thresholds are imaginable, like the mean value for each element calculated over the whole pattern set. It is possible that the transformation maps two or more different patterns onto patterns with the same input but different target values. In such cases the new patterns do not define a function. To obtain a function, only the patterns with target *True* are taken into account. The set of Boolean patterns define a Boolean function, which can be expressed in an standard algebraic form, for example as disjunctive normal form. Note that basing the function only on the patterns with a *True* target produces a result where all other function values are automatically zero.

For example the XOR function is described by:

$$XOR(X,Y) = X\overline{Y} + \overline{X}Y.$$

The disjunctive normal form can be transformed into a polynomial by applying the following rewriting rules [Rumelhart-86.2]:

1. Replacing *True* by 1 and *False* by 0.
2. Replacing the disjunction operator by addition and the conjunction operator by multiplication.
3. Replacing the negation operator by subtraction from 1.
4. Replacing X by x for every variable (X represents a Boolean variable and x represents a real valued variable).
5. Simplification of the terms by multiplication.

The XOR function of the above example is transformed into the polynomial

$$xor(x,y) = x(1-y) + (1-x)y$$

and then after multiplying

$$xor(x,y) = x + y - 2xy.$$

Next, the polynomial is used to build the associated network in the following way:

- each variable corresponds to one input
- each term of the polynomial represents a connection and the number of variables of a term represents the order of the connection
- the coefficient of the term represents the weight of this connection.

The resulting network is then trained with the original real valued pattern set using the backpropagation algorithm [Rumelhart-86.2].

2.2 Second method

In order to find smaller HOP topologies, only the term of the polynomial expression with the smallest number of variables is used. The resulting HOP does not guarantee the solvability of the Boolean function resulting from the first method, since it is only a rough appoximation.

Example:

$$0\ 0\ 0\ 0 \quad \rightarrow \quad \overline{X}_1\overline{X}_2\overline{X}_3\overline{X}_4 \qquad 1\ 0\ 0\ 1 \quad \rightarrow \quad X_1\overline{X}_2\overline{X}_3X_4$$

$$f(x_1,x_4) = 0.6 + 0.2 * x_1x_4$$

The terms of the polynomial represent the connections used for building the HOP. The resulting network is then trained with the original real valued pattern set as in the first method.

2.3 Third method

The input transformation in the first method uses an imprecise, one bit quantization. As this might be an insufficient precision, a quantization with more bits can be used.

Using two bit precision, the patterns in the first example are transformed into new patterns:

Data			Target
0 0	0 0	$\rightarrow$	0.6
1 0	0 1	$\rightarrow$	0.2 .

The transformed data set can be regarded as the values a function assumes at the vertices of a hypercube. An interpolating polynomial on these vertices is then constructed by building for each vertex of the hypercube the corresponding Boolean conjunct. Then, as in the first method, each of these conjuncts are transformed into a polynomial term. Finally, the sum of all these terms, each weighted by the value of the function at the corresponding vertex, is taken.

For example, the corresponding Boolean conjuncts for the Boolean patterns of the previous example are:

$$0\ 0\ 0\ 0 \quad \rightarrow \quad \overline{X}_1\overline{X}_2\overline{X}_3\overline{X}_4 \qquad 1\ 0\ 0\ 1 \quad \rightarrow \quad X_1\overline{X}_2\overline{X}_3X_4.$$

They are transformed into the polynomial

$$f(x_1, x_2, x_3, x_4) = 0.6 * (1 - x_1)(1 - x_2)(1 - x_3)(1 - x_4) + 0.2 * x_1(1 - x_2)(1 - x_3)x_4$$

Based on this polynomial, the HOP topology is constructed like it is in the first method, but the resulting network is trained with the transformed pattern set.

This method has a disadvantage: the number of input variables for the mapping problem increases with the number of bits in the quantization. Therefore, the calculation of the polynomial is sometimes infeasible because of the huge number of terms. A way to handle this problem is to consider for each pattern only the term of the polynomial expression with the smallest number of variables, as in the second method.

3 Results

In order to determine the effectiveness of the methods presented above, experiments are performed using three different benchmarks: Solar, Servo, and Auto-MPG [data-server].

Of special interest in this experiments are:

- The number of connections W (which is equal to the number of terms in the polynomial obtained N_t) found by the methods,in comparison to the number of connections in a fully interlayer connected high-oder network $W_{max} = 2^k - 1$ (considering only Boolean inputs), where k represents the order of the network. In the following k is equal to N_1.
- The generalization ability G(calc) of the network (mean square error), using the weights calculated by the method in question
- The mean generalization ability $\overline{G}$(ran) of the network trained by the backpropagation algorithm with random initial weights. This value is the average over 10 simulations.

In the tables below, the following conventions are used [Fiesler-94]: P is the number of patterns in the training set, N the number of inputs, T is the threshold used for the conversion to boolean values, and E_t the mean square error tolerated on the training set. 'n.c.' stands for 'no convergence'.

3.1 First method

Tables 1 and 2 show the outcome of two series of experiments. The first series was done using a fixed threshold during the conversion of the real data into Boolean data. The results in table two are produced by applying thresholds that represent the mean value of each element of the data set and target set respectively.

pattern set	P	N	T	$N_t = W$	W_{max}	E_t	$\overline{G}$(ran.)	G(calc.)
Solar	209	12	0.5	3200	4096	0.07	0.0823	0.0875
Servo	84	12	0.5	1704	4096	0.13	n.c.	0.19
Auto-MPG	294	7	0.5	48	128	0.07	0.0644	0.066

Table 1: Performance with $T = 0.5$

pattern set	P	N	T	$N_t = W$	W_{max}	E_t	$\overline{G}$(ran.)	G(calc.)
Solar	209	12	mean	2827	4096	0.07	0.0982	0.11
Servo	84	12	mean	3744	4096	0.1	0.172	0.163
Auto-MPG	294	7	mean	66	128	0.07	0.07	0.067

Table 2: Performance with $T = mean$

The tables show, that the number of connections depends highly and in a undetermined manner on the threshold value. Equally for the generalization of the networks with the calculated, respectively by backpropagation obtained weights, no preference for one of the methods can be determined. Interestingly, one of the networks did not learn the problem after its weights were randomized.

Compared with the smallest networks found using pruning algorithms [Thimm-95], these networks are very big (for all three data sets the minimal topology is smaller than 10 connections), whereas the generalization is comparable.

3.2 Second method

In tables 3 and 4 it can be seen that the omission of all but the lowest order connections significantly decreases the sizes of the networks, which are now very close to the results obtained by the pruning of 'big' HOPs [Thimm-95]. However, the generalization of the networks trained with random initial weights is worse than that of the pruned ones. In comparison to the first method, the smaller networks do not always reach the same precision on the training set, and the generalization of the networks with the calculated connections is sometimes better sometimes worse.

pattern set	P	N	T	$N_t = W$	W_{max}	E_t	$\overline{G}$(ran.)	G(calc.)
Solar	209	12	0.5	15	4096	0.07	0.07923	0.0942*
Servo	84	12	0.5	11	4096	0.14	starts with 0.18	0.18
Auto-MPG	294	7	0.5	6	128	0.1	0.0955	0.0896

* HOP trained to (E_t = 0.06)

Table 3: Performance with $T = 0.5$

pattern set	p	N	T	$N_t = W$	W_{max}	E_t	$\overline{G}$(ran.)	G(calc.)
Solar	209	12	mean	54	4096	0.08	0.0931	0.169
Servo	84	12	mean	16	4096	0.14	starts with 0.18	0.18
Auto-MPG	294	7	mean	19	128	0.08	0.0677	0.067

Table 4:Performance with $T = mean$

3.3 Third method

Table 5 shows the results of using only the terms with the minimal number of variables in the polynomial expression.

pattern set	P	Q	$N_t = W$	E_t(rand)	$\overline{G}$(rand)	E_t(calc)	G(calc)
Solar	209	4	209	0.005	0.2113	0.005	0.315
Servo	84	3	84	0.005	0.2373	0.005	0.25
Auto-MPG	294	7	294	0.03	0.1663	0.005	0.297

Table 5: Performance

And, as expected, the third method produces networks of a high precision and a medium size, which are often smaller than those of the first method. Unfortunately the generalization is significantly worse, both for the networks with calculated weights and with weights obtained by training.

4 Conclusions

Three methods have been presented for finding initial High Order Perceptron configurations with a reduced connectivity. These configurations are used to train HOPs while avoiding the exponential number of connections in normal fully interlayer connected HOPs.

The first method results in a considerable reduction in both the number of connections (22% to 63%, for $T = 0.5$), as well as in the time spent to train the HOPs, as compared to fully interlayer connected N_1-th order HOPs. However, the resulting HOP topologies are still large and a further refinement of the method has to be considered. The generalization performance of the first method is close to that of pruning methods.

The second method, which is based on the first method, produces near minimal topologies at the cost of a slight decrease in generalization performance.

The third method, which uses a quantized pattern set for training, gives (despite the bigger input vector) less connections than the first method, but no improvement over the results of the second method. However, the resulting networks show a bad generalization performance.

References

[Fiesler-94] E. Fiesler. "Neural Network Classification and Formalization." *Computer Standards & Interfaces*, volume 16, number 3, pages 231–239, June 1994.

[Muselli-95] Marco Muselli. "On Sequential Construction of Binary Neural Networks." *IEEE Transactions on Neural Networks*, volume 6, number 3, pages 422–431, May 1995.

[Redding-93] Nicholas J. Redding, Adam Kowalczyk, and Tom Downs. "Constructive Higher-Order Network Algorithm That Is Polynomial Time." *Neural Networks*, volume 6, number 7, pages 997–1010, 1993.

[Rumelhart-86.2] 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.

[Thimm-94.1] G. Thimm, R. Grau, and E. Fiesler. "Modular Object-Oriented Neural Network Simulators and Topology Generalizations." In Maria Marinaro and Pietro G. Morasso (editors), *Proceedings of the International Conference on Artificial Neural Networks (ICANN 94)*, (Sorrento, Italy; 26–29 May, 1994), (ISBN: 3-540-19887-3), volume 1, pages 747–750. Springer-Verlag, London, U.K., 1994.

[Thimm-95] G. Thimm and E. Fiesler. "Evaluating Neural Network Connection Pruning MethodsN." Submitted to ISANN'95.

[data-server] P. M. Murphy and D. W. (Librarians) Aha. "*UCI Repository of machine learning databases [Machine-readable data repository], ftp access* `ics.uci.edu: pub/machine-learning-databases`." 1994.