Learning Concepts by Synthesizing Minimal Threshold Gate Networks

Arlindo L. Oliveira and **Alberto Sangiovanni-Vincentelli**
Dept. of EECS
UC Berkeley
Berkeley CA 94720

Abstract

We propose a new methodology for the synthesis of two-level networks of threshold gates based on techniques related to the ones used in the logic synthesis of digital networks. The proposed approach starts with a large network that performs the desired mapping and reduces its size by applying transformations that preserve the functionality for all examples in the training set.

1 Introduction

In the standard model of learning from examples, the task of the learning algorithm is to induce, from a set of positive and negative instances (the training set), a rule that will accurately predict the class of future instances (the test set).

For non-trivial induction tasks, intermediate concepts have to be developed. Finding such intermediate concepts is in general a difficult task, because the merit of an intermediate concept is usually dependent on what other intermediate concepts are defined at the moment.

In feed-forward layered neural networks, intermediate concepts are identified with hidden units. Each hidden unit represents an intermediate concept, and its function is to be derived by some learning algorithm. In general, the smaller the network, the better the generalization performed, as long as it is large enough to learn the required mapping [Huyser & Horowitz, 1988]. Empirical evidence suggests that, in many cases, learning with the absolute minimum number of units is far more difficult than if some extra hidden units are allowed [Rumelhart & McClelland, Eds. 1986].

Logic synthesis techniques have been successfully used to generate compact implementations of logic networks. Unlike in most connectionist approaches, an architecture is derived from scratch to map the given input to the desired output, and is then modified to optimize one or more criteria. Therefore, a strong motivation exists to apply some of the techniques used in traditional circuit design to the definition of architectures for general networks.

While algorithms for the synthesis of threshold gate networks have been proposed before [Muroga, 1971] they are only applicable for functions of a very small number of variables and, in most cases, find solutions very far from the optimum. The approach presented in this paper is different and introduces some new concepts like pyramids and *M*-covers. We will show that a substantial subset of the concepts developed for the synthesis of two-level logic networks [Brayton et Al. 1984] can be extended to the synthesis of threshold gate networks.

2 Basic concepts

2.1 Concepts, hypothesis and boolean functions

We will use the usual setting for the problem of learning from examples in an attribute based description language. Let B_N be the set of N objects in the instance space. Concepts and hypothesis are subsets of B_N. The task of learning from examples can be described as follows: given m objects, chosen from B_N according to some arbitrary distribution and labeled positive or negative according to whether they belong or not to a given concept C, derive a hypothesis H which is an approximation as good as possible to the concept C.

If the attributes are boolean, the objects in the instance space are the vertices of a n dimensional hypercube, where n is the number of attributes and there is a one to one correspondence between concepts and boolean functions. In this paper, we will restrict ourselves to the case where the attributes are boolean, but we remark that the concepts and techniques described can be extended to the case where the attributes are not boolean by using multi-valued logic synthesis.

Let f be an incompletely specified function of n boolean variables, $\{x_1, ..., x_n\}$. Formally, f is a mapping from $\{0,1\}^n \rightarrow \{0,1,\star\}$, where the $\star$ is used to identify the input points with unspecified output values. Function f can be described by the following disjoint sets of hypercube vertices (or minterms): f_{ON}, f_{DC} and f_{OFF}. The f_{ON} (f_{OFF}) set specifies which minterms should turn the output *on* (*off*). Non trivial boolean functions should have

non-empty f_{ON} and f_{OFF} sets. The f_{DC} (don't care) set contains all minterms not contained in the previous sets. The value of the function is not specified for this minterms. When the f_{DC} is empty, the boolean function is completely specified and it corresponds to a concept. Therefore, there is a one-to-one correspondence between *completely specified boolean functions* and *concepts*, and we will use the terms interchangeably. A function g is compatible with f iff the value of g for every point in the f_{ON} and f_{OFF} sets if compatible with f, i. e., iff $f_{ON} \subseteq g_{ON} \subseteq f_{ON} \cup f_{DC}$.

From a set of positive and negative examples defining the f_{ON} and f_{OFF} sets of a function f, we propose to derive a compatible function g such that the size of the network needed to implement g is minimal.[1] In other words, points in the f_{DC} set get assigned the value needed to minimize the total size of the network needed to implement a function compatible with f.

2.2 Cubes and Pyramids

Let the concepts of interest be defined by n boolean variables and let a literal be either a variable or its negation. A cube (or monomial) is a conjunction of k literals ($1 \leq k \leq n$), where no two literals corresponding to the same variable appear.

A cube with n literals corresponds to a minterm or a vertex in the n dimensional hypercube that represents the input space. A cube c_1 is said to contain another cube c_2 if $c_2 \Rightarrow c_1$, i.e., if the truth values defined in c_2 make c_1 true. If $c_1 \neq c_2$, then such a containment is proper. The dimension of a cube c is the number of variables not present in c. The distance between two cubes, c_1 and c_2, $\delta(c_1, c_2)$ is the number of variables that appear negated in one cube and non-negated in the other. For example, $x_1\overline{x_3}x_4$ is at distance 2 from $\overline{x_1}x_2\overline{x_4}$. This distance can be viewed as the number of edges that one has to traverse to go from one cube to the other in the hypercube representation of the input space.

A cube is identified with the boolean function implemented by an *and* gate. Work in artificial and biological neural networks suggests that an extension of the synthesis procedure to include threshold gates may be interesting and extend the power of logic synthesis algorithms. Consider a threshold gate of k input variables with weights of value -1 or +1 and let the input variables assume the values 0 or 1. Let the threshold gate be *on* when $w_1x_1+w_2x_2+...+w_kx_k \geq V_{thr}$, where it is assumed, without loss of generality that the gate implements a function of the first k variables in an n-dimensional input space. Let V_{max} be the maximum value that the sum in the previous expression can assume and cube c be defined by the literals $c_1c_2...c_k$ where $c_i = x_i$ if $w_i = +1$ and $c_i = \overline{x_i}$ if $w_i = -1$. Consider now a minterm $m = m_1m_2..m_k..m_n$. This minterm will turn the gate *on* iff no more that $V_{max} - V_{thr}$ literals are different from the literals in c, i.e., if $\delta(c, m) \leq V_{max} - V_{thr}$. Let $h = V_{max} - V_{thr}$. We define a pyramid as a pair $(c : h)$, where c is a the apex cube and h is an integer value, the height. The minterms contained by a pyramid are at distance h or less from the apex cube. See figure 1 for a graphical representation of some of the concepts described above.

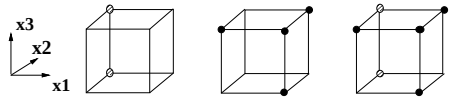

Figure 1: Cube $\overline{x_1}x_2$ and pyramid $(x_1\overline{x_2}x_3 : 1)$

The distance between a cube c_1 and a pyramid $(c_2 : h)$ is given by $\max(\delta(c_1, c_2) - h, 0)$. For example, cube $\overline{x_1}x_2$ and pyramid $(x_1\overline{x_2}x_3 : 1)$ (in the figure) are at distance 1.

A pyramid p is a prime pyramid, relatively to some boolean function f, iff there is no other pyramid contained in the $f_{ON} \bigcup f_{DC}$ set that properly contains pyramid p.

Any pyramid $p = (c : h)$ has a complement pyramid, $\overline{p} = (\tilde{c} : k-h-1)$, where k is the number of literals in c and $\tilde{c}$ is the cube obtained by complementing all such literals. It is trivial to show that $p\overline{p} = \emptyset$ and $p \cup \overline{p} = \{0,1\}^n$. This fact, together with a trivial generalization of the De Morgan laws leads to the following interesting result [Muroga, 1971]:

Lemma 1: If a function f can be implemented by a network with H threshold gates and the inputs are available in both negated and non-negated form, then function $\overline{f}$ can also be implemented by a network of H threshold gates.

2.3 Covers and *M*-covers

We are interested in the synthesis of a two-level network of threshold gates. We will generalize the concept of cube cover of a boolean function defined in [Brayton et Al. 1984] in two different directions. This generalization keeps the relation between a cover and a two-level implementation of f but allows for the use of general threshold gates in both the first and second levels.

A set of pyramids is a pyramid cover for a function $f = (f_{ON}, f_{DC}, f_{OFF})$ if every minterm in f_{ON} is contained in at least one pyramid and no minterm in the f_{OFF} set is contained in any pyramid. A pyramid cover for a given function leads to a direct implementation of that function: allocate one threshold gate for each pyramid and connect them all to an *or* gate in the second level.

The concept of cover can be extended to a more general one that leads to implementations where the second level gate is not forced to be an *or* gate. A bag of pyramids, B, is and *M*-cover ($M \geq 1$) for a function $f = (f_{ON}, f_{DC}, f_{OFF})$ iff all minterms in the f_{ON} set are covered by at least M pyramids and all minterms in the f_{OFF} set are covered by at most $M - 1$ pyramids in B. Given an *M*-cover for f, it is a trivial task to derive a two-level network of threshold gates that implements f, i.e., that performs the required

[1] We use the commonly accepted convention that a solution is minimal (or optimal) if there are no neighbors with smaller value and reserve the word minimum (or optimum) for the absolute minimum over the whole input space.

input-output mapping: simply allocate one threshold gate for every pyramid in B and connect them all to a gate in the second level with all $w_i = 1$ and $V_{thr} = M$. It is easy to see why the above network implements the correct input-output mapping. Every point in the f_{ON} set turns on M or more gates in the first layer and these will turn the output gate *on*. Points in the f_{OFF} set, however, will turn less than M of the first level gates *on* and this is not enough to switch the output gate to the *on* state.

The search for a network that minimizes the number of hidden nodes is therefore equivalent to the search of an *M*-cover for the specified function with a minimal number of pyramids in B.

Figure 2 shows how pyramids $p_1 = (\overline{x_1}\,\overline{x_2}\,\overline{x_3} : 2)$ and $p_2 = (x_1x_2x_3 : 1)$ are a 2-cover for the function f_1 defined by $f_{ON} = \{x_1x_2\overline{x_3}, x_1\overline{x_2}x_3, \overline{x_1}x_2x_3\}$, $f_{OFF} = \{0,1\}^3 \backslash f_{ON}$ and $f_{DC} = \emptyset$.

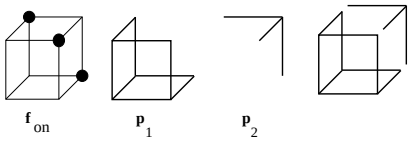

Figure 2: 2-cover for function f_1.

For this particular example, a 2-cover with only two pyramids exists, while the unique 1-cover for this function requires three pyramids. In general, the number of pyramids needed may be very different for different values of M.

2.4 Non-unitary weights

As described, a pyramid corresponds to a threshold gate with weights of +1 or -1. To see how this can be generalized to non-unitary weights consider a function f of n variables, $\{v_1, v_2 ... v_n\}$ defined by its f_{ON} and f_{OFF} sets. Let z be an integer and f^z be a function of zn variables defined by its f^z_{ON} and f^z_{OFF} sets. Every minterm in f^z_{ON} (f^z_{OFF}) is obtained by replicating z times every variable in the corresponding minterm of f_{ON} (f_{OFF}). For example, if $z = 2$ and $n = 3$ the minterm $x_1x_2\overline{x_3}$ is converted to $x_{1_1}x_{1_2}x_{2_1}x_{2_2}\overline{x_{3_1}}\,\overline{x_{3_2}}$. The following lemma shows how the same algorithm can be used to derive networks with arbitrary integer weights:

Lemma 2: Let f be a boolean function of n variables such that f is implementable by a single threshold gate with integer input weights $\{w_1, w_2 ... w_n\}$. Let z be the maximum value of $\mid w_1 \mid, \mid w_2 \mid ... \mid w_n \mid$. Then there exists a pyramid in the space of zn dimensions that covers all the minterms in the f^z_{ON} set and none in the f^z_{OFF}.

2.5 Expanding and reducing pyramids

The synthesis algorithm we propose in the next section is inspired in two-level synthesis algorithms for logic gate networks like, for example, ESPRESSO [Brayton et Al. 1984]. These algorithms work by performing successive expansion and reduction of cubes in the cover. Therefore, the ability to expand and reduce pyramids proves necessary to the implementation of similar algorithms in the current setting. The expand and reduction operations applied to pyramids can be described as follows:

- Expand: either
  - Expand the apex cube, by dropping one literal.
  - Reduce the apex cube, by adding one literal, but increase the pyramid height.
- Reduce: either
  - Reduce the apex cube, by adding one literal.
  - Expand the apex cube, by dropping one literal, but decrease the pyramid height.

The reader may verify that the expand (reduce) operation does indeed generate a pyramid that properly contains (is contained) in the original one. Furthermore, the expansion (reduction) of a pyramid corresponds to the reduction (expansion) of its complement pyramid.

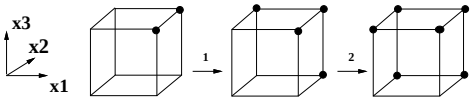

Figure 3: Expanding $(x_1x_3 : 0)$ to $(x_1x_2x_3 : 1)$ to $(x_2x_3 : 1)$

3 The synthesis procedure

3.1 Finding a small *M*-cover

Assume we have one solution, i.e., an *M*-cover for the function is given. If none other is given, the set of f_{ON} minterms is a 1-cover for the function.

A branch and bound algorithm is used to search for a more compact *M*-cover. A search tree is built with a predefined branching factor r, and, at each step, the most promising node in the tree is expanded. If the branching factor, r is greater than the number of pyramids in the cover, k, r is set equal to k.

The root corresponds to the initial solution. Every node n_i in the search tree corresponds to a bag B_i of pyramids and has a value given by $v_i = Cu - z$, where u is the number of f_{ON} minterms covered less than M times, z is the number of f_{ON} minterms covered more than M times and C some large constant. If v_i is non-positive, then node n_i represents a solution with one less pyramid than the root. The value of a node estimates how close it is to a solution. Minterms covered more that M times are weighted negatively because some pyramids that are currently covering these minterms will be free to cover other points in the space. A new node n_j is created by expanding a node in the tree. Bag B_j is obtained by modifying the bag B_i of the parent node applying one of the following two operations:

1) Removing one pyramid from the bag. 2) Changing one pyramid so that it covers a new minterm by applying the expand and reduce operations. Children of the root node are created applying operation 1. All other nodes (except the root) are created using operation 2. Operation 2, the expansion of node n_i by changing one of the pyramids in the cover is performed by the algorithm described below.

$m \leftarrow$ pick_one_uncovered_minterm(n_i);
for $j = 1$ to r do {
    $p_{old} \leftarrow$ choose_next_pyr(m);
    $(\mathcal{S}_{on}, \mathcal{S}_{off}) \leftarrow$ build_s_sets(m, n_i, p_{old});
    $p \leftarrow$find_max_red_pyr($\mathcal{S}_{on}, \mathcal{S}_{off}$);
    if $p \neq \emptyset$
        $n_j \leftarrow$ create_child($n_i, B_i \cup \{p\} \setminus \{p_{old}\}$); }

A minterm in the f_{ON} set that is not covered by M pyramids is chosen. The pyramids that are closer to this minterm are then sequentially selected by the choose_next_pyr procedure. Function build_s_sets creates the $\mathcal{S}_{on}$ set by adding minterm m to the list of f_{ON} minterms covered by p and not covered more than M times. The $\mathcal{S}_{off}$ set consists of all the minterms in the f_{OFF} set that are already covered by $M - 1$ pyramids. Procedure create_child simply adds one more child to node n_i, with a bag of pyramids appropriately modified.

Function find_max_red_pyr is the heart of the algorithm. It returns, if possible, the maximally reduced pyramid covering all minterms in the $\mathcal{S}_{on}$ set and none in the $\mathcal{S}_{off}$ set.

3.2 Finding the maximally reduced pyramid

The algorithm for finding the maximally reduced pyramid that covers all minterms in a given $\mathcal{S}_{on}$ set while not covering any minterms in the $\mathcal{S}_{off}$ set is based on the expand and reduce operations described in the previous section.

The algorithm starts by identifying $\mathcal{S}_{red}$, the set of minterms in $\mathcal{S}_{on}$ already covered by the pyramid. The pyramid is first reduced with respect to this set and then expanded until either all minterms in the $\mathcal{S}_{on}$ set are covered or a prime is obtained. If the second condition holds and the first doesn't, it reports failure. Otherwise, it returns the expanded pyramid.[2]

3.3 Changing the value of M

We start the algorithm with a 1-cover (M = 1) consisting of all the minterms in the f_{ON} set. This cover is then reduced as much as possible by successive expansion of pyramids. When the maximum size of the search tree is reached, the value of M is increased and a search for a smaller cover is performed.

In the outer loop of the algorithm, M is treated as a variable and increased when the search for a better solution fails. The procedure stops when a value of M is reached for which no solution is found. This procedure may fail to find the best M-cover, even if the inner loop is guaranteed to find the global optimum for a given M. In fact, we may fail to find a good solution for $M = M_{opt}$ if no good solution exist for $M = M_{opt} - 1$ because the algorithm will stop when no $(M_{opt} - 1)$-cover is found. Empirical validation, however, has shown that, in general, there is an optimal value of M, M_{opt}, and the cardinality of the solutions increases with $|M - M_{opt}|$.

A detailed analysis of the computational complexity of the procedures involved shows that, when an M-cover is to be found, we should expect an O($M(t + m)mn$) algorithm, where m is the number of minterms, n the number of variables and a search tree of no more than t nodes is used.

4 Experimental results

We used three families of concepts to test our algorithm on. These concepts are described on table 1. The limits on the weights (W_{max}) listed in column four are for the minimal realizations known. The performance of our approach was evaluated by comparing the number of hidden units (H) in the networks synthesized by our algorithm with known bounds (H_{min}) on the size of two-level layered networks required to implement these concepts [Rumelhart & McClelland, Eds. 1986]. The maximum weight allowed for each network was the weight needed to achieve the known minimum realization.

Table 1: Concepts description.

Concept	Description	H_{min}	W_{max}	M_{opt}
N-parity	1 if even # of bits is 1	N	1	$\lceil N/2 \rceil$
N-asym	1 if non-symmetric	2	$2^{N/2-1}$	1
N-mltpx	First N bits sel. one of 2^N bits	2^N	1	1

Table 2 summarizes the results obtained when the complete set of examples for each concept was used. Note that the number of variables in column 3 takes into consideration the expansion in the number of dimensions needed for the larger weight sizes required for the minimum realization in the asymmetry problems. All results were obtained in a DECstation 3100.

4.1 Discussion

Table 2 shows that we were able to obtain the minimum size realization for all but the two larger parity problems. The CPU times involved compare favorably with the ones described by Tesauro [Tesauro & Janssens, 1988] for the back-propagation method applied to the N-parity problems. He used 2N hidden units for an N-parity problem

[2] A detailed description of the procedure can be found in [Oliveira, 1991].

Table 2: Results.

Concept	m	n	H	T_{cpu}	T_{norm}
6-parity	64	6	6	4.3	6.5
7-parity	128	7	7	10.7	6.4
8-parity	256	8	8	61.9	9.9
9-parity	512	9	11	878.1	13.5
10-parity	1024	10	24	1759.8	8.4
6-asym	64	24	2	1.5	1.7
8-asym	256	64	2	23.2	1.9
10-asym	1024	160	2	408.6	1.6
12-asym	4096	384	2	11752.9	1.6
2-mltpx	64	6	4	0.2	0.9
3-mltpx	2048	11	8	160.8	2.8

and reported an average CPU time of 20000 sec. in a SUN workstation[3] for the 6-parity problem. Furthermore, these results were obtained after careful optimization of the parameters involved and he simply disregarded all cases where no convergence was obtained. Ash [Ash 89] reports superior convergence results with a constructive algorithm, but the larger examples reported are the 6-parity and 6-asymmetry, which are trivially simple for our algorithm. Furthermore, his algorithm failed to find solutions with the minimum number of units even for these smaller problems in some of the runs.

Our own experiments with a conjugate gradient method [Kramer & Sangiovanni, 1989] led us to conclude that, in general, it is difficult to load these concepts into networks when the minimum number of units is used in the hidden layer. For the larger problems the algorithm got consistently stuck in local minima. When larger networks were used for this method in order for the learning algorithm to succeed, the quality of the generalization obtained was consistently worst than the one obtained by applying our algorithm [Oliveira, 1991].

Finally, the last column in table 2 represents the CPU time normalized by the factor $M(t+m)mn$ and shows that the results are compatible with a O($M(t+m)mn$) algorithm.

5 Conclusions and future work

We have presented an algorithm for the synthesis of threshold gate networks that derives a compact realization compatible with a given set of examples. The resulting network is guaranteed to give the correct output for every example in the training set and, since it is of minimal size, will, in most cases, generalize better than networks obtained by alternative methods. The computational cost involved in the synthesis process compares favorably with the cost of loading the same set of examples into a fixed architecture using gradient descent techniques.

[3]The author does not specify which specific model of SUN workstation was used.

One of the strongest limitation of the algorithm is that it can be used only to generate two-level networks. While two-level networks of threshold gates are enough to implement any concept, compact two-level representations may not exist for a large family of interesting concepts. The use of multi-level synthesis techniques [Brayton et Al. 1989] is the natural solution for this problem and a study on the applicability of these techniques to this problem is a promising area for future research.

Acknowledgments

This work was supported by the Air Force Office of Scientific Research (AFOSR/JSEP) under Contract No. F49620-90-C-0029 and the Portuguese INVOTAN committee. The authors would like to thank Gregory Sorkin who suggested the use of threshold gates in logic synthesis, Jaijeet Roychowdhury who introduced the concept of pyramids and Alan Kramer by stimulating discussions and help in the comparisons with alternative approaches.

References

[Ash 89] Ash, Timur "Dynamic Node Creation in Back Propagation Networks", Connection Science 1:4, pp. 365-375, 1989.

[Brayton et Al. 1984] Brayton, R. K., G. D. Hachtel, C. T. McMullen & A. L. Sangiovanni-Vincentelli "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

[Brayton et Al. 1989] Brayton, R. K., G. D. Hachtel & A. L. Sangiovanni-Vincentelli "Multilevel Logic Synthesis", Proceedings of the IEEE, vol. 78:2, pp. 264-300, February 1990.

[Huyser & Horowitz, 1988] Huyser, K. A & Horowitz, M. A. "Generalization in Digital Functions", Abstract in Neural Networks 1:Suppl. 1, pp. 101, 1988.

[Kramer & Sangiovanni, 1989] Alan Kramer & A. Sangiovanni Vincentelli, "Efficient Parallel Learning Algorithms for Neural Networks", Proceedings of 1988 IEEE Conference on Neural Information Processing Systems, pp. 40-48, San Mateo, CA, 1989.

[Muroga, 1971] Muroga, Saburo "Threshold Logic and its Applications", Wiley-Interscience, 1971.

[Oliveira, 1991] Oliveira, Arlindo "Logic Synthesis Using Threshold Gates", Internal Report, in preparation, UC Berkeley, 1991.

[Rumelhart & McClelland, Eds. 1986] Rumelhart, D. E. & J. L. McClelland &, editors. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1, MIT Press, Cambridge MA, 1986.

[Roychowdhury, 1989] Roychowdhury, Jaijeet "Synthesis with M of N Gates", Internal Memo, UC Berkeley, 1989.

[Tesauro & Janssens, 1988] Tesauro, Gerald & Robert Janssens "Scaling relationships in back-propagation learning", Complex Systems, 2:39-44, 1988.