On Constructing Threshold Networks for Pattern Classification

Martin Anthony

Abstract. This paper describes a method of constructing one-hidden layer feedforward linear threshold networks to represent Boolean functions (or partially-defined Boolean functions). The first step in the construction is sequential linear separation, a technique that has been used by a number of researchers [7, 11, 2]. Next, from a suitable sequence of linear separations, a threshold network is formed. The method described here results in a threshold network with one hidden layer. We compare this approach to the standard approach based on a Boolean function's disjunctive normal form and to other approaches based on sequential linear separation [7, 11].

1 Introduction

It is well known that any Boolean function can be represented by a feedforward linear threshold network with one hidden layer. The simplest way to see this is via the disjunctive normal form representation of the function (see later). Here, we discuss an alternative way of representing Boolean functions (or partially-defined Boolean functions, by which we mean restrictions Boolean functions to a specified domain). This alternative approach arises from considering a fairly natural way of classifying points by iterative or sequential linear separation.

The problem considered, to be more precise, is the following. Given disjoint subsets T and F of $\{0,1\}^n$, for some natural number n, we want to produce a feedforward linear threshold network whose output is 1 if its input is in T, and whose output is 0 if its output is in F. We refer to the pair (T,F) as a partially-defined Boolean function (pdBf), and if $T \cup F = \{0,1\}^n$, then the partially-defined Boolean function is simply a Boolean function (since its value is defined for all elements of $\{0,1\}^n$). The set T is called the set of true points, or those labelled 1; and the set

Martin Anthony
Department of Mathematics, London School of Economics and Political Science,
London WC2A 2AE, UK
e-mail: m.anthony@lse.ac.uk

L. Franco et al. (Eds.): Constructive Neural Networks, SCI 258, pp. 71–82.
springerlink.com

F is the set of false points, labelled with 0. We focus on two-class classification problems in the Boolean domain, but much of what we say can be generalised to deal with multi-class classification, or classification on more general domains (such as the whole of $\mathbb{R}^n$) [2].

We start by describing what we call the 'standard' approach, which is based on a disjunctive normal form representation of the Boolean function (or of a Boolean function that is consistent with a partially-defined Boolean function). Then we describe an approach in which we first find a *threshold decision list* that represents the pdBf (T,F) and, from this threshold decision list, produces a threshold network. We compare our method with other approaches. The threshold networks we produce have a single hidden layer of units, as do those resulting from the standard approach, but we show that there can be some advantages in the method we discuss here. Some previous work [7, 11, 12] also involved sequential linear separation, but resulted in networks with a different structure, in which there were many single-unit hidden layers, with connectivity between them.

2 Simple Threshold Networks Representing Boolean Functions: The Standard Approach

There is a very straightforward way in which to represent partially-defined Boolean functions by threshold networks having one hidden layer of units. This is based on the existence, for each Boolean function, of a *disjunctive normal form* for the function. We first briefly review key ideas on threshold networks and Boolean functions.

2.1 Threshold Functions and Threshold Networks

A function $t : \{0,1\}^n \to \{0,1\}$ is a *(Boolean) threshold function* if there are $w \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that

$$t(x) = \begin{cases} 1 \text{ if } \langle w,x\rangle \geq \theta \\ 0 \text{ if } \langle w,x\rangle < \theta, \end{cases}$$

where $\langle w,x\rangle$ is the standard inner product of w and x. Thus, $t(x) = \text{sgn}(\langle w,x\rangle - \theta)$, where $\text{sgn}(z) = 1$ if $z \geq 0$ and $\text{sgn}(z) = 0$ if $z < 0$. Given such w and θ, we say that t is represented by $[w,\theta]$ and we write $t \leftarrow [w,\theta]$. The vector w is known as the *weight vector*, and θ is known as the *threshold*.

A *threshold network* is formed when combine together *threshold units*, each of which computes a threshold function. More precisely, we have a directed graph, at each vertex of which is a 'unit', and with the arcs of the digraph representing the flows of signals between units. Some of the units are termed *input units*: these receive signals not from other units, but have their signals applied from outside. In our case, there will be n input units, each of which receives 0 or 1 as an input. In this situation, the set of all input patterns, or just 'inputs', is $\{0,1\}^n$. Units that do not transmit signals to other units are termed *output units*. We will be interested in networks with one output unit. The network is said to be a *feed-forward network*

if the underlying directed graph is acyclic (that is, it has no directed cycles). This feed-forward condition means that the units (both the input units and the threshold units) can be labeled with integers in such a way that if there is a connection from the unit labeled i to the unit labeled j then $i < j$. In any feed-forward network, the units may be grouped into *layers*, labeled $0, 1, 2, \ldots, \ell$, in such a way that the input units form layer 0, these feed into the threshold units, and if there is a connection from a threshold unit in layer r to a threshold unit in layer s, then we must have $s > r$. Note, in particular, that there are no connections between any two units in a given layer. The 'top' layer consists of output units. The layers that are not inputs or outputs are called hidden layers.

We will be primarily interested in linear threshold networks having just one hidden layer, and it is useful to give an explicit description in this case of the functionality of the network. Such a network will consist of n inputs and some number, k, of threshold units in a single hidden layer, together with one output threshold unit. Each threshold unit computes a threshold function of the n inputs. The (binary-valued) outputs of these hidden nodes are then used as the inputs to the output node, which calculates a threshold function of these. Thus, the threshold network computes a threshold function of the outputs of the k threshold functions computed by the hidden nodes. If the threshold function computed by the output node is described by weight vector $\beta \in \mathbb{R}^k$ and threshold ϕ, and the threshold function computed by hidden node i is $f_i \leftarrow [w^{(i)}, \theta^{(i)}]$, then the threshold network as a whole computes the function $f : \{0,1\}^n \to \{0,1\}$ given by

$$f(y) = 1 \Longleftrightarrow \sum_{i=1}^{k} \beta_i f_i(y) \geq \phi;$$

that is,

$$f(y_1 y_2 \ldots y_n) = \operatorname{sgn}\left(\sum_{i=1}^{k} \beta_i \operatorname{sgn}\left(\sum_{j=1}^{n} w_j^{(i)} y_j - \theta^{(i)} \right) - \phi \right),$$

where $\operatorname{sgn}(x) = 1$ if $x \geq 0$ and $\operatorname{sgn}(x) = 0$ if $x < 0$. The *state* of the network is the (concatenated) vector

$$\omega = (w^{(1)}, \theta^{(1)}, w^{(2)}, \theta^{(2)}, \ldots, w^{(k)}, \theta^{(k)}, \beta, \phi) \in \mathbb{R}^{nk+2k+1}.$$

A fixed network architecture of this type (that is, fixing n and k), computes a parameterised set of functions $\{f_\omega : \omega \in \mathbb{R}^{nk+2k+1}\}$. In state ω, the network computes the function $f_\omega : \{0,1\}^n \to \{0,1\}$.

2.2 Boolean Functions and DNF Representations

Any Boolean function can be expressed by a *disjunctive normal formula* (or DNF), using *literals* $u_1, u_2, \ldots, u_n, \bar{u}_1, \ldots, \bar{u}_n$, where the $\bar{u}_i$ are known as *negated literals*. A disjunctive normal formula is one of the form

$$T_1 \vee T_2 \vee \cdots \vee T_k,$$

where each T_l is a *term* of the form

$$T_l = \left(\bigwedge_{i \in P} u_i \right) \bigwedge \left(\bigwedge_{j \in N} \bar{u}_j \right),$$

for some disjoint subsets P,N of $\{1,2,\ldots,n\}$.

Given a disjunctive normal form for a Boolean function, there may be a number of ways of simplifying it. For two Boolean functions f and g, we write $f \leq g$ if $f(x) \leq g(x)$ for all x; that is, if $f(x) = 1$ implies $g(x) = 1$. Similarly, for two Boolean formulae ϕ, ψ, we shall write $\phi \leq \psi$ if, when f and g are the functions represented by ϕ and ψ, then $f \leq g$. A term T of a DNF is said to *absorb* another term T' if $T' \leq T$. For example, $T = \bar{u}_1 u_4$ absorbs the term $T' = \bar{u}_1 u_3 u_4$. That is, whenever T' is true, so is T. This means, for example, that the formula

$$\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3 \vee \bar{u}_1 u_3 u_4$$

is equivalent to $\bar{u}_1 u_4 \vee u_1 u_2 \bar{u}_3$. A term T is an *implicant* of f if $T \leq f$; in other words, if T true implies f true. The terms in any DNF representation of a function f are implicants of f. The most important type of implicants are the *prime implicants*. These are implicants with the additional property that there is no other implicant of f absorbing T. Thus, a term is a prime implicant of f if it is an implicant, and the deletion of any literal from T results in a non-implicant T' of f (meaning that there is some x such that $T'(x) = 1$ but $f(x) = 0$). If we form the disjunction of all prime implicants of f, we have a particularly important DNF representation of f.

2.3 From DNF to Threshold Network

Suppose that (T,F) is a partially-defined Boolean function and that the Boolean function f is some 'extension' of (T,F), meaning that $f(x) = 1$ for $x \in T$ and $f(x) = 0$ for $x \in F$. Let ϕ be a DNF formula for f. Suppose $\phi = T_1 \vee T_2 \vee \cdots \vee T_k$, where each T_i is a term of the form $T_i = \left(\bigwedge_{j \in P_i} u_j\right) \bigwedge \left(\bigwedge_{j \in N_i} \bar{u}_j\right)$, for some disjoint subsets P_i, N_i of $\{1,2,\ldots,n\}$. We form a network with k hidden units, one corresponding to each term of the DNF. Labelling these threshold units $1,2,\ldots,k$, we set the weight vector $w^{(i)}$ from the inputs to hidden threshold unit i to correspond directly to T_i, in the sense that $w_j^{(i)} = 1$ if $j \in P_i$, $w_j^{(i)} = -1$ if $j \in N_i$, and $w_j^{(i)} = 0$ otherwise. We take the threshold $\theta^{(i)}$ on hidden unit i to be $|P_i|$. We set the weight on the connection between each hidden threshold unit and the output unit to be 1, and the threshold on the output unit to be $1/2$. That is, we set β to be the all-1 vector of dimension k, and set the threshold ϕ to be $1/2$. It is clear that hidden threshold unit i outputs 1 on input x precisely when x satisfies the term T_i, and that the output unit computes the 'or' of all the outputs of the hidden units. Thus, the output of the network is the disjunction of the terms T_i, and hence equals f.

Note that this does not describe a unique threshold network representing the pdBf (T,F), for there may be many choices of extension function f and, given f, there may be many possible choices of DNF for f. In the case in which $T \cup F = \{0,1\}^n$, so that the function is fully defined, we could, for the sake of definiteness, use the particular DNF formula described above, the disjunction of all 'prime implicants'.

In general, a simple counting argument establishes that, whatever method is being used to represent Boolean functions by threshold networks, for most Boolean functions a high number of units will be required in the resulting network. Explicitly, suppose we have an n-input threshold network with one output and one hidden layer comprising k threshold units. Then, since the number of threshold functions is at most 2^{n^2} (see [1, 3], for instance), the network computes no more than $(2^{n^2})^{k+1}$ different Boolean functions, this being an upper bound on the number of possible mappings from the input set $\{0,1\}^n$ to the vector of outputs of all the $k+1$ threshold units. This bound, $2^{n^2(k+1)}$ is, for any fixed k, a tiny proportion of all the 2^{2^n} Boolean functions and, to be comparable, we need $k = \Omega(2^n/n^2)$. (This is a very quick and easy observation. For more detailed bounds on the sizes of threshold networks required to compute general and specific Boolean functions, see [10], for instance.)

It is easy to give an explicit example of a function for which this standard method produces an exponentially large threshold network. The parity function f on $\{0,1\}^n$ is given by $f(x) = 1$ if and only if x has an odd number of ones. It is well known that any DNF formula ϕ for f must have 2^{n-1} terms. To see this, note first that each term of ϕ must have degree n. For, suppose some term T_i contained fewer than n literals, and that neither u_j nor $\bar{u}_j$ were present in T_i. Then there are $x,y \in \{0,1\}^n$ which are true points of T_i, but which differ only in position j. Then, since T_i is a term in the DNF representation of the parity function f, we would have $f(x) = f(y) = 1$. But this cannot be: one of x,y will have an odd number of entries equal to 1, and one will have an even number of such entries. It follows that each term must contain n literals, in which case each term has only one true point, and so we must have 2^{n-1} distinct terms, one for each true point. It follows that the resulting network has 2^{n-1} threshold units in the hidden layer.

3 Decision Lists and Threshold Decision Lists

We now present a different approach to the problem of finding a threshold network representation of a partially-defined Boolean function. To explain this, we first discuss decision lists and threshold decision lists.

3.1 Decision Lists

We start by describing *decision lists*, introduced by Rivest [9]. Suppose that G is any set of Boolean functions. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be a *decision list* based on G if for some positive integer r, there are functions $f_1, f_2, \ldots, f_r \in G$ and bits $c_1, c_2, \ldots, c_r \in \{0,1\}$ such that f acts as follows. Given an example y, we first

evaluate $f_1(y)$. If $f_1(y) = 1$, we assign the value c_1 to $f(y)$; if not, we evaluate $f_2(y)$, and if $f_2(y) = 1$ we set $f(y) = c_2$, otherwise we evaluate $f_3(y)$, and so on. If y fails to satisfy any f_i then $f(y)$ is given the default value 0. The evaluation of a decision list f can therefore be thought of as a sequence of 'if then else' commands, as follows:

if $f_1(y) = 1$ then *set* $f(y) = c_1$
else if $f_2(y) = 1$ then *set* $f(y) = c_2$
...
...
else if $f_r(y) = 1$ then *set* $f(y) = c_r$
else *set* $f(y) = 0$.

We define $DL(G)$, the class of *decision lists based on G*, to be the set of finite sequences

$$f = (f_1, c_1), (f_2, c_2), \ldots, (f_r, c_r)$$

such that $f_i \in G$, $c_i \in \{0,1\}$ for $1 \leq i \leq r$. The values of f are defined by $f(y) = c_j$ where $j = \min\{i : f_i(y) = 1\}$, or 0 if there are no j such that $f_j(y) = 1$. We call each f_j a *test* (or, following Krause [6], a *query*) and the pair (f_j, c_j) is called a *term* of the decision list.

3.2 Threshold Functions and Threshold Decision Lists

We now consider the class of decision lists in which the tests are threshold functions. We shall call such decision lists *threshold decision lists*, but they have also been called *neural* decision lists [7] and *linear* decision lists [13]. Formally, a threshold decision list

$$f = (f_1, c_1), (f_2, c_2), \ldots, (f_r, c_r)$$

has each $f_i : \mathbb{R}^n \to \{0,1\}$ of the form $f_i(x) = \mathrm{sgn}(\langle w_i, x\rangle - \theta_i)$ for some $w_i \in \mathbb{R}^n$ and $\theta_i \in \mathbb{R}$. The value of f on $y \in \mathbb{R}^n$ is $f(y) = c_j$ if $j = \min\{i : f_i(y) = 1\}$ exists, or 0 otherwise (that is, if there are no j such that $f_j(y) = 1$).

3.3 A Geometrical Interpretation: Iterative Linear Separation

Threshold decision lists are, in fact, quite a natural way in which to classify points, and a useful geometrical motivation can be given. Suppose we are given a partially-defined Boolean function (T, F). We can use a hyperplane to separate off a set of points all having the same classification label (that is, all of which are from T, or all of which are from F). At least one point can always be separated off in this way. For, given any $x \in \{0,1\}^n$, x and $\{0,1\}^n \setminus \{x\}$ are linearly separable. To see this, we can suppose, without any loss of generality, that x is the origin. Then the hyperplane with equation $\sum_{i=1}^n x_i = 1/2$ achieves the required separation. (Note that this argument is contingent on the geometry of $\{0,1\}^n$. For more general subsets of $\mathbb{R}^n$, some additional properties, such as general position, would need to hold to make the argument

work.) The points that have been 'chopped off' can then be removed from consideration and the procedure iterated until no points remain. In general, we would hope to be able to separate off more than one point at each stage, but the argument given above establishes that, at each stage, at least one point can indeed be 'chopped off', so since the set of points is finite, the procedure does indeed terminate.

We may regard the chopping procedure as a means of constructing a threshold decision list consistent with the data set. If, at stage i of the procedure, the hyperplane with equation $\sum_{i=1}^{n} \alpha_i y_i = \theta$ chops off points all having label j, with these points in the half-space with equation $\sum_{i=1}^{n} \alpha_i y_i \geq \theta$, then we take as the ith term of the threshold decision list the pair (f_i, j), where $f_i \leftarrow [\alpha, \theta]$. Therefore, given any partially-defined Boolean function (T, F), there will always be some threshold decision list representing the pdBf.

3.4 A Related Approach

This sequential linear separation, or 'chopping', procedure is similar to one employed by Jeroslow [5], but at each stage in his procedure, only examples from T may be 'chopped off' (and one cannot choose instead to chop off a subset of points from F).

Note that if the 'chopping' method of constructing a threshold decision list is applied to the sequence of hyperplanes resulting from the Jeroslow method, a restricted form of decision list results, namely one in which all terms are of the form $(f_i, 1)$. But such a decision list is quite simply the disjunction $f_1 \vee f_2 \vee \cdots$. For Boolean functions, the problem of decomposing a function into the disjunction of threshold functions has been given substantial consideration by Hammer *et al.* [4] and Zuev and Lipkin [14]. Hammer *et al.* defined the *threshold number* of a Boolean function to be the minimum s such that f is a disjunction of s threshold functions, and they showed that there is an increasing function with threshold number $\binom{n}{n/2}/n$. (A function is increasing if, when $f(x) = 1$ and $x_i = 0$, then $f(x + e_i) = 1$ too, where e_i is the unit basis vector with ith entry equal to 1 and all other entries equal to 0.) Zuev and Lipkin showed that almost all increasing functions have this order of threshold number, and that almost all Boolean functions have a threshold number that is $\Omega(2^n/2)$ and $O(2^n \ln n/n)$.

We give an example for illustration, which demonstrates the advantages to be gained by the threshold decision list approach over the Jeroslow approach.

Example: Consider again the parity function f on $\{0,1\}^n$, given by $f(x) = 1$ if and only if x has an odd number of ones. We first find a hyperplane such that all points on one side of the plane are either positive or negative. It is clear that all we can do at this first stage is chop off one of the points since the nearest neighbours of any given point have the opposite classification. Let us suppose that we decide to chop off the origin. We may take as the first hyperplane the plane with equation $y_1 + y_2 + \cdots + y_n = 1/2$. We then ignore the origin and consider the remaining points. We can next chop off all neighbours of the origin, all the points which have precisely

one entry equal to 1. All of these are positive points and the hyperplane $y_1 + y_2 + \cdots + y_n = 3/2$ will separate them from the other points. These points are then deleted from consideration. We can continue in this way. The procedure iterates n times, and at stage i in the procedure we 'chop off' all data points having precisely $(i-1)$ ones, by using the hyperplane $y_1 + y_2 + \cdots + y_n = i - 1/2$, for example. (These hyperplanes are in fact all parallel, but this is not in general possible.) So we can represent the parity function by a threshold decision list with n terms. By contrast, Jeroslow's method requires 2^{n-1} iterations, since at each stage it can only 'chop off' one positive point: that is, it produces a disjunction of threshold functions (or a special type of threshold decision list) with an exponential number of terms.

3.5 Algorithmics

The chopping procedure as we have described it is in some ways merely a device to help us see that threshold decision lists have a fairly natural geometric interpretation. But the algorithmic practicalities have been investigated by Marchand *et al.* [7, 8] and Tajine and Elizondo [11]. Marchand *et al.* derive a greedy heuristic for constructing a sequence of 'chops', where the aim is to separate as large a set (all of the same class) as possible at each stage. This relies on an incremental heuristic for the NP-hard problem of finding at each stage a hyperplane that chops off as many remaining points as possible. Tajine and Elizondo consider batch and incremental and modular algorithms and also focus on greedy strategies.

4 Threshold Networks from Threshold Decision Lists

4.1 From a Threshold Decision List to a Threshold Network with One Hidden Layer

We now show how we can make use of the chopping procedure to find a threshold network representing a given Boolean function by giving an explicit way in which a threshold decision list can be represented by a threshold network with one hidden layer.

Theorem 1. *Suppose we have a threshold decision list*

$$f = (f_1, c_1), (f_2, c_2), \ldots, (f_k, c_k)$$

in which f_i is represented by weight vector $w^{(i)}$ and threshold $\theta^{(i)}$, so that $f_i \leftarrow [w^{(i)}, \theta^{(i)}]$. Consider a threshold network architecture having n inputs, k threshold units in a single hidden layer, and one output. Let ω be the state given as follows:

$$\omega = (w^{(1)}, \theta^{(1)}, w^{(2)}, \theta^{(2)}, \ldots, w^{(k)}, \theta^{(k)}, \beta, 1),$$

where

$$\beta = (2^{k-1}(2c_1-1), 2^{k-2}(2c_2-1), \ldots, 2(2c_{k-1}-1), (2c_k-1));$$

that is, $\beta_i = 2^{k-i}(2c_i-1)$. *Then* f_ω, *the function computed by the network in state* ω, *equals* f.

Proof: We prove the result by induction on k, the length of the decision list (and number of hidden threshold units in the network).

The base case is $k=1$. Since the default output of any decision list is 0, we may assume that f takes the form $f=(f_1,1)$ where $f_1 \leftarrow [w,\theta]$ for some $w \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$. Then, β is the single number $2^{1-1}(2c_1-1)=1$ and $\phi=1$. So

$$f_\omega(y_1y_2\ldots y_n) = \operatorname{sgn}\left(\operatorname{sgn}\left(\sum_{j=1}^{n} w_j^{(i)} y_j - \theta^{(i)}\right) - 1\right) = \operatorname{sgn}\left(\sum_{j=1}^{n} w_j^{(i)} y_j - \theta^{(i)}\right) = f_1(y_1y_2\ldots y_n),$$

so $f_\omega = f_1 = f$.

Now suppose that the result is true for threshold decision lists of length k, where $k \geq 1$. Consider a threshold decision list

$$f = (f_1,c_1),(f_2,c_2),\ldots,(f_k,c_k),(f_{k+1},c_{k+1}).$$

Let g denote the threshold decision list

$$g = (f_2,c_2),\ldots,(f_k,c_k),(f_{k+1},c_{k+1}).$$

Then, the inductive assumption implies that, for all y,

$$g(y) = \operatorname{sgn}\left(\sum_{i=1}^{k} 2^{k-i}(2c_{i+1}-1) f_{i+1}(y) - 1\right) = \operatorname{sgn}(G(y))),$$

say. What we need to prove is that for all y,

$$f(y) = \operatorname{sgn}(F(y)),$$

where

$$F(y) = \sum_{i=1}^{k+1} 2^{k+1-i}(2c_i-1) f_i(y) - 1.$$

Now,

$$\begin{aligned} F(y) &= 2^k(2c_1-1)f_1(y) + \sum_{i=2}^{k+1} 2^{k+1-i}(2c_i-1)f_i(y) - 1 \\ &= 2^k(2c_1-1)f_1(y) + \sum_{i=1}^{k} 2^{k-i}(2c_{i+1}-1)f_{i+1}(y) - 1 \\ &= 2^k(2c_1-1)f_1(y) + G(y). \end{aligned}$$

Now, suppose $f_1(y) = 0$. In this case, by the way in which decision lists are defined to operate, we should have $f(y) = g(y)$. This is indeed the case, since

$$\text{sgn}(F(y)) = \text{sgn}(2^k(2c_1 - 1)f_1(y) + G(y)) = \text{sgn}(0 + G(y)) = \text{sgn}(G(y)) = g(y).$$

Suppose now that $f_1(y) = 1$. In this case we have $f(y) = c_1$ and so we need to verify that $\text{sgn}(F(y)) = c_1$. We have

$$\begin{aligned}
(2c_1 - 1)F(y) &= 2^k(2c_1 - 1)^2 f_1(y) + (2c_1 - 1)G(y) \\
&= 2^k + (2c_1 - 1)\sum_{i=1}^{k} 2^{k-i}(2c_{i+1} - 1)f_{i+1}(y) - 1 \\
&\geq 2^k - \sum_{i=1}^{k} 2^{k-i} - 1 \\
&= 2^k - (2^k - 1) - 1 \\
&= 0.
\end{aligned}$$

That is, $(2c_1 - 1)F(y) \geq 0$, so $\text{sgn}(F(y)) = \text{sgn}(2c_1 - 1) = c_1$. This completes the proof.

4.2 Using Other Types of Threshold Network

Marchand *et al.* [7, 8] and Tajine and Elizondo [11] have also studied the construction of threshold networks through a consideration of how the points to be classified can be separated iteratively by hyperplanes. However, the threshold networks arising in [7] and (from the batch algorithm) in [11], have different architectures to those constructed above, in that there are connections between hidden units (making the networks have more than one layer). By contrast, like the standard representation based on DNF, our construction gives a network with only one hidden layer. A characteristic feature of decision lists which must be captured by the corresponding threshold networks is the 'if-then-else' nature of their definition: there is a *precedence* or hierarchy among the tests. The first test is conducted and, if passed, the output is determined. Only if the first test is failed, do we move on to the next test. In the construction of Theorem 1, the precedence structure is encoded into the network by the exponentially-decreasing weights in the β-vector: the output if the first hidden unit is weighted twice as much as that of the second, and so on. In [7, 11], the precedence structure is built in with lateral connections between hidden units. For instance, in [7], the network constructed has a 'cascade' structure: the hidden threshold units are labelled 1 to k and there are connections between unit i and unit j for all $j < i$. The weights on these connections are large enough to enable the output of unit i to dominate (or inhibit) that of unit j.

5 Comparison with an Approach Based on Disjunctive Normal Form

5.1 Comparing the DNF-Based Approach and the Threshold Decision List Approach

The parity function demonstrates that the representation arising from Theorem 1 can differ considerably from the one described earlier. For, we have seen that the parity function can be represented by a threshold decision list with n terms, and hence the network given by Thoerem 1 has only n hidden units. By contrast, as noted earlier, the standard DNF-based construction will, necessarily, have at least 2^{n-1} hidden units.

A useful observation in comparing the two approaches is the following: if T is any term of a DNF formula, then T can be represented by a threshold function. This is quite easy to see and, indeed, is implicit in our description of the standard construction of a network from a DNF. For, suppose that

$$T = \left(\bigwedge_{j\in P} u_j\right) \bigwedge \left(\bigwedge_{j\in N} \bar{u}_j\right),$$

where $P \cap N = \emptyset$. Then $T \leftarrow [w, |P|]$ where $w_j = 1$ if $j \in P$, $w_j = -1$ if $j \in N$, and $w_j = 0$ otherwise. So if $\phi = T_1 \vee T_2 \vee \cdots \vee T_k$ is a DNF representation of the function f, then f is also represented by the threshold decision list

$$(T_1, 1), (T_2, 1), \ldots, (T_k, 1).$$

Applying Theorem 1 now to this threshold decision list would give a threshold network representing f. That network would have exactly the same structure as the one obtained by using the standard DNF-based method, using DNF formula ϕ. (However, the weights from the hidden layer to the output would be different, with exponentially decreasing, rather than constant, values.) What this demonstrates is that, in particular, there is always a threshold decision list representation whose length is no more than that of any given DNF representation of the function. There may, as in the case of parity, be a significantly shorter threshold decision list. So the decision list approach (and application of Theorem 1) will, for any function (or partially-defined function), in the best case, give a network that is no larger than that obtained by the standard method.

6 Conclusions

We have shown that a natural approach to data classification by successive linear separation can be used to construct threshold networks of simple architecture to represent Boolean or partially-defined Boolean functions. Such an approach differs

from previous constructions which have also been based on iterative linear separation, in that the networks constructed have only one hidden layer. Furthermore, it can always produce a network that is no larger than that which follows from the standard translation from a Boolean function's disjunctive normal form into a threshold network.

References

1. Anthony, M.: Discrete Mathematics of Neural Networks: Selected Topics. Society for Industrial and Applied Mathematics, Philadeplhia (2001)
2. Anthony, M.: On data classification by iterative linear partitioning. Discrete Applied Mathematics 144(1-2), 2–16 (2004)
3. Cover, T.M.: Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. IEEE Trans. on Electronic Computers EC-14, 326–334 (1965)
4. Hammer, P.L., Ibaraki, T., Peled., U.N.: Threshold numbers and threshold completions. Annals of Discrete Mathematics 11, 125–145 (1981)
5. Jeroslow, R.G.: On defining sets of vertices of the hypercube by linear inequalities. Discrete Mathematics 11, 119–124 (1975)
6. Krause, M.: On the computational power of boolean decision lists. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 372–383. Springer, Heidelberg (2002)
7. Marchand, M., Golea, M.: On Learning Simple Neural Concepts: from Halfspace Intersections to Neural Decision Lists. Network: Computation in Neural Systems 4, 67–85 (1993)
8. Marchand, M., Golea, M., Ruján, P.: A convergence theorem for sequential learning in two-layer perceptrons. Europhys. Lett. 11, 487 (1990)
9. Rivest, R.R.: Learning Decision Lists. Machine Learning 2(3), 229–246 (1987)
10. Siu, K.Y., Rowchowdhury, V., Kalaith, T.: Discrete Neural Computation: A Theoretical Foundation. Prentice Hall, Englewood Cliffs (1995)
11. Tajine, M., Elizondo, D.: Growing methods for constructing Recursive Deterministic Perceptron neural networks and knowledge extraction. Artificial Intelligence 102, 295–322 (1998)
12. Tajine, M., Elizondo, D.: The recursive deterministic perceptron neural network. Neural Networks 11, 1571–1588 (1998)
13. Turán, G., Vatan, F.: Linear decision lists and partitioning algorithms for the construction of neural networks. In: Foundations of Computational Mathematics: selected papers of a conference, Rio de Janeiro, pp. 414–423. Springer, Heidelberg (1997)
14. Zuev, A., Lipkin, L.I.: Estimating the efficiency of threshold representations of Boolean functions. Cybernetics 24, 713–723 (1988); Translated from Kibernetika (Kiev) 6, 29–37 (1988)