CENTER FOR
PARALLEL OPTIMIZATION

MATHEMATICAL PROGRAMMING IN NEURAL NETWORKS

by

O. L. Mangasarian

Computer Sciences Technical Report #1129

December 1992

Mathematical Programming in Neural Networks

O. L. Mangasarian*

December 11, 1992

Abstract

This paper highlights the role of mathematical programming, particularly linear programming, in training neural networks. A neural network description is given in terms of separating planes in the input space that suggests the use of linear programming for determining these planes. A more standard description in terms of a mean square error in the output space is also given, which leads to the use of unconstrained minimization techniques for training a neural network. The linear programming approach is demonstrated by a brief description of a system for breast cancer diagnosis that has been in use for the last four years at a major medical facility.

1 What is a Neural Network?

A neural network is a representation of a map between an input space and an output space. A principal aim of such a map is to discriminate between the elements of a finite number of disjoint sets in the input space. Typically one wishes to discriminate between the elements of two disjoint point sets in the n-dimensional real space R^n. In this case the input space is R^n and the output space is the binary set $\{0, 1\}$. The neural network consists of neuron-like units with threshold values, connected by weighted arcs. These units are similar to the human neuron in that they fire when their input exceeds their threshold. The simplest and earliest neural network is the linear threshold unit (LTU), first proposed by McCulloch and Pitts in 1943 [30] and for which Rosenblatt [39, 38] proposed his iterative perceptron training algorithm in 1957. Such an LTU represents the following (nonlinear) step function y from the n-dimensional real space R^n into $\{0, 1\}$:

$$y(x) = s(wx - \theta) := \begin{cases} 1 \;\; if \;\; wx > \theta \\ 0 \;\; if \;\; wx \leq \theta \end{cases} \tag{1}$$

Here w is some fixed weight vector in R^n and the threshold θ is a fixed real number. Figure 1 depicts the neural network representing this function. Geometrically an LTU can be represented by the plane $wx = \theta$ in R^n. Since such a plane divides the space into the disjoint halfspaces $\{x \mid wx \leq \theta\}$ and $\{x \mid wx > \theta\}$, it is immediately obvious that an LTU can be used to completely discriminate between two disjoint sets $\mathcal{A}$ and $\mathcal{B}$ in R^n, each of which lying in one of these two halfspaces. We term such sets **linearly separable.**

Among the earliest algorithms for obtaining such a separating plane or an LTU was Rosenblatt's perceptron algorithm, which turns out to be a version of the Motzkin-Schoenberg iterative algorithm [33] for solving a system of linear inequalities. This algorithm terminates in a finite number of steps

*Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, WI 53706, email: *olvi@cs.wisc.edu.* This material is based on research supported by Air Force Office of Scientific Research Grant AFOSR-89-0410 and National Science Foundation Grant CCR-9101801.

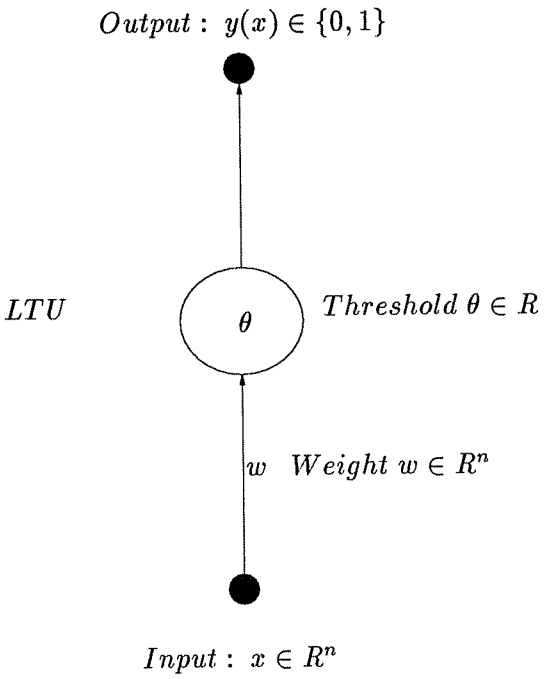

Figure 1: The simplest neural network, a perceptron or linear threshold unit (LTU), characterized by the weight vector w in R^n and threshold θ in R.

if the sets $\mathcal{A}$ and $\mathcal{B}$ are linearly separable [32, pages 164-175], [35, Chapter 5], [9]. However if $\mathcal{A}$ and $\mathcal{B}$ are linearly inseparable the algorithm need not terminate but the iterates are bounded [32, pages 181-187], [9] and hence merely have an accumulation point. By contrast consider the following simple linear programming formulation

$$\max_{w,\theta,\zeta} \{\zeta \mid Aw \geq e(\theta + \zeta),\ Bw \leq e\theta,\ \zeta \leq 1\} \tag{2}$$

Here A and B are $m \times n$ and $k \times n$ matrices, representing m and k n-dimensional points of the sets $\mathcal{A}$ and $\mathcal{B}$ respectively, and e is a vector of ones of appropriate dimension. It is easy to see that (2) is always solvable because $w = 0,\ \theta = 0,\ \zeta = 0$ is feasible and the objective function ζ is bounded above by 1. Furthermore, the maximum is 1 if $\mathcal{A}$ and $\mathcal{B}$ are linearly separable, and is 0 if not. Obviously the linear program (2) can be solved in a finite number of steps to yield a separating plane or equivalently an LTU, if one exists. If none exists then, unfortunately, the null solution: $w = 0,\ \theta = 0,\ \zeta = 0$, is an optimal point that does not furnish any information other than the fact that the sets $\mathcal{A}$ and $\mathcal{B}$ are linearly inseparable. This is the case even when the sets $\mathcal{A}$ and $\mathcal{B}$ are "nearly" linearly separable. However this difficulty can be easily circumvented, in order to obtain some "approximate" error-minimizing linear separation, by considering a slightly different linear program (6) as shown by Theorem 2.1 below. The first linear programming formulations for the linearly separable case were given in 1964 and 1965 [11, 27], but they also suffered from the null-solution difficulty for the linearly inseparable case. In order to handle the linearly inseparable case one has to employ a more complex map than that provided by an LTU. This was made evident in the early days of neural network development by Minsky and Papert in 1969 [32] when they presented their now-classical exclusive-or (XOR) counterexample which is not linearly separable and hence for which no LTU will work. (See Figure 2). This essentially brought the early development of neural networks to a halt until it was realized [47, 41] that a more complex function than that represented by an LTU was needed to correctly map these simple four points into the set $\{0, 1\}$. Curiously enough, however, it should be noted that even before Minsky-Papert proposed their classical XOR counterexample, a linear-programming-based piecewise-linear separator was proposed in 1968 [28] that could easily and correctly handle this problem, and which in fact can be represented as a neural network [7]. (See Figure 14 and discussion following Algorithm 2.2 below.) We shall now use this example to motivate a general multisurface method (MSM) for separating the sets $\mathcal{A}$ and $\mathcal{B}$ of the XOR example or any other disjoint sets $\mathcal{A}$ and $\mathcal{B}$ in R^n and will present a neural network representation of this separation.

As the name implies, MSM uses more than one surface. (Typically planes are used, but any other surface that is linear in its parameters, such as a quadratic surface, can be used.) For the XOR example, Figure 3 shows complete separation by two planes which can be described analytically as follows:

$$\begin{array}{l} w^1 x > \theta^1 \ \ or \ \ w^2 x > \theta^2 \ \ for \ \ x \in \mathcal{A} = \{(1,0),(0,1)\} \\ w^1 x \leq \theta^1 \ \ and \ \ w^2 x \leq \theta^2 \ \ for \ \ x \in \mathcal{B} = \{(0,0),(1,1)\} \end{array} \tag{3}$$

By considering each of the two planes of Figure 3 as an LTU as indicated, we can represent the separation mapping achieved by those two planes as a neural network depicted in Figure 4.

In the language of neural networks, this is a feedforward neural network (i.e. no information from a unit is fed back to preceding units) with one hidden layer of 2 LTU's and one output LTU. The planes $w^i x = \theta^i,\ i = 1, 2$, are each represented by an LTU of the hidden layer LTU's with threshold θ^i and incoming arc weight vector $w^i,\ i = 1, 2$. For a fixed x in R^n, the collective output of the 2 hidden LTU's constitutes a vertex of the unit cube $C^2 := \{r \mid r \in R^2,\ 0 \leq r \leq e\}$ in R^2.

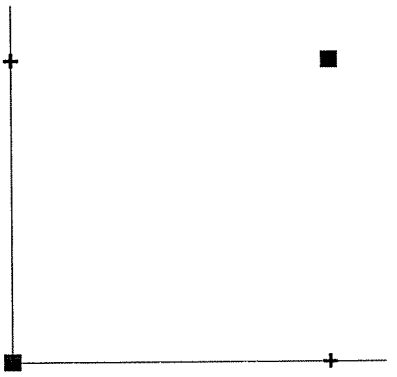

Figure 2: The sets $\mathcal{A} = \{(1, 0), (0, 1)\}$ and $\mathcal{B} = \{(0, 0), (1, 1)\}$ of the XOR example that cannot be mapped into $\{0\}$ and $\{1\}$ respectively by an LTU, or equivalently cannot be separated by a single plane: $wx = \theta$.

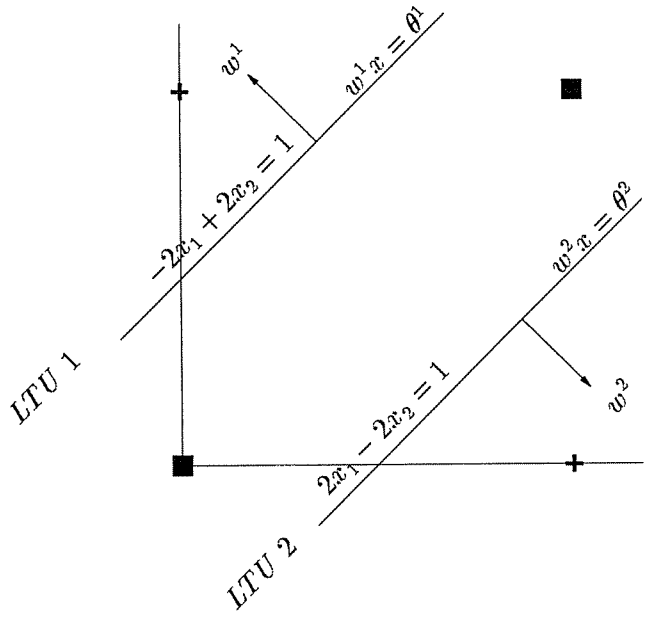

Figure 3: Separation of the XOR example by two planes.

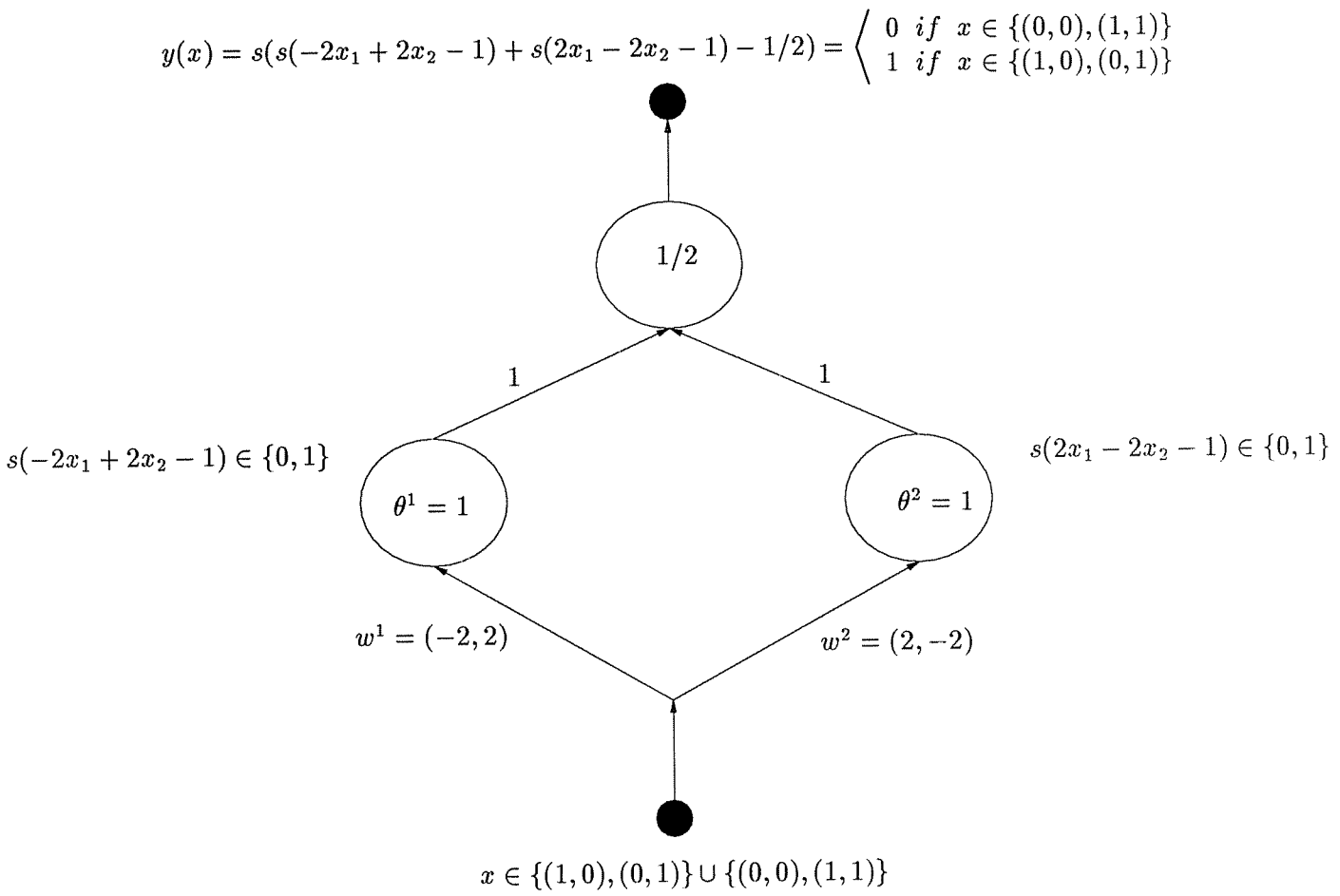

Figure 4: A neural network with 3 LTU's (2 hidden LTU's and 1 output LTU) representing the multisurface separation of Figure 3 for the XOR example.

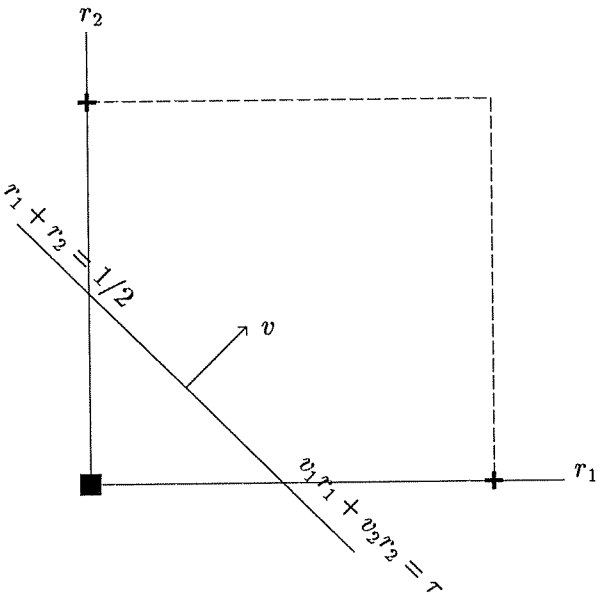

Figure 5: Mapping of the sets $\mathcal{A} = \{(1, 0), (0, 1)\}$ and $\mathcal{B} = \{(0, 0), (1, 1)\}$ by the output of the hidden layer of the neural network of Figure 4 as vertices of the unit cube in R^2, together with the separating plane representing the output unit and its incoming arcs.

The scalar weights (1 and 1 here) of the outgoing arcs from the 2 hidden units and the threshold $1/2$ of the output LTU constitute a plane in R^2 which strictly separates the vertices of C^2 representing $\mathcal{A}$ from those representing $\mathcal{B}$. This plane in R^2 and the vertices of C^2 to which $\mathcal{A}$ and $\mathcal{B}$ are mapped for the XOR example are shown in Figure 5.

The term "hidden layer" refers to units hidden from both the input and output layers. The presence of such a hidden layer is crucial and endows a neural network with a complexity that is not possessed by a single layer of LTU's. In fact, it can be shown [24, 29] that such a neural network can separate any two disjoint sets in R^n given a sufficient number of hidden units. (See Theorem 2.2 below.) This is equivalent to the multisurface method separating such disjoint sets given a sufficient number of planes [28].

Hopefully, it should be clear now that the mapping that we are after from R^n into $\{0, 1\}$ can be rather complex depending on the sets $\mathcal{A}$ and $\mathcal{B}$. The two representations that have been described, neural networks and multisurface separation, are both valid representations of this mapping. Mathematical programming techniques can be employed to obtain the parameters determining either representation. In Section 2 of this paper we shall describe how linear programming can be used to generate a sequence of planes that will separate any two disjoint point sets in R^n, while Section 3 will describe an unconstrained optimization formulation for obtaining the parameters of a feedforward neural network with one hidden layer. In Section 4 we shall describe a very effective application of neural network training via linear programming applied to breast cancer diagnosis. Section 5 concludes the paper.

2 Neural Network Training via Linear Programming

In this section we indicate how linear programming can be used to train neural networks. We begin by setting up a linear program that works not only for linearly separable sets, but also for sets that are not, that is linearly inseparable sets. For the latter case, what we expect, of course, is some error-minimizing linear separation that does not suffer from the shortcoming of the linear program (1.2), that is a null solution that does not provide any error-minimizing separation. For this purpose, we utilize the linear program introduced recently in [8] and which has the following desirable features not all of which are possessed by any other previous linear programming formulation [11, 27, 28, 45, 20, 19]:

(i) A strict separating plane (that is neither set lies on the separating plane) for linearly separable sets $\mathcal{A}$ and $\mathcal{B}$

(ii) An error-minimizing plane is obtained when the sets $\mathcal{A}$ and $\mathcal{B}$ are linearly inseparable.

(iii) No extraneous constraints are used to exclude the null solution for linearly inseparable sets. (Such constraints inevitably fail on certain problems.)

The proposed linear programming formulation is based on the fact that when the sets $\mathcal{A}$ and $\mathcal{B}$ are linearly separable, there exists a "dead zone" consisting of an open slab: $\{x \mid \theta - 1 < wx < \theta + 1\}$ surrounding the separating plane $\{x \mid wx = \theta\}$ which contains no points from either set $\mathcal{A}$ and $\mathcal{B}$. (The numbers +1 and -1 can be replaced by any other positive and negative numbers. However, a simple rescaling gives back +1 and -1.) Thus, for linear separation we have the following inequalities satisfied by the separating plane $\{x \mid wx = \theta\}$:

$$\begin{aligned} Aw &\geq e(\theta + 1) \\ Bw &\leq e(\theta - 1) \end{aligned} \tag{4}$$

Recalling that A represents the m points of the set $\mathcal{A}$, and B the k points of $\mathcal{B}$, an obvious error-minimizing problem one wishes to consider would be:

$$\min_{w,\theta} \frac{1}{m} \left\| (-Aw + e(\theta + 1))_+ \right\|_1 + \frac{1}{k} \left\| (Bw - e(\theta - 1))_+ \right\|_1 \tag{5}$$

where $\|\cdot\|_1$ denotes the 1-norm and $(z)_+$ denotes $((z)_+)_i = \max\{z_i,\ 0\},\ i = 1, \ldots m$, for $z \in R^m$. Here the average error of violating (4) is minimized, and a zero minimum is obtained if and only if the sets $\mathcal{A}$ and $\mathcal{B}$ are linearly separable. It is easy to show that (5) is equivalent to the linear program [8]

$$\min_{w,\theta,y,z} \left\{ \frac{ey}{m} + \frac{ez}{k} \mid Aw - e\theta + y \geq e,\ -Bw + e\theta + z \geq e,\ y \geq 0,\ z \geq 0 \right\} \tag{6}$$

Important properties of the linear program (6) are summarized in the following theorem.

Theorem 2.1 ***Exact and approximate separation of sets by linear programs*** *[8]. Let $\mathcal{A}$ and $\mathcal{B}$ be represented by the $m \times n$ and $k \times n$ matrices A and B respectively.*

(a) The sets $\mathcal{A}$ and $\mathcal{B}$ are linearly separable if and only if the linear program (6) has a zero minimum in which case $\{x \mid wx = \theta\}$ is a separating plane, where $(w,\ \theta,\ y,\ z)$ is any solution of (6).

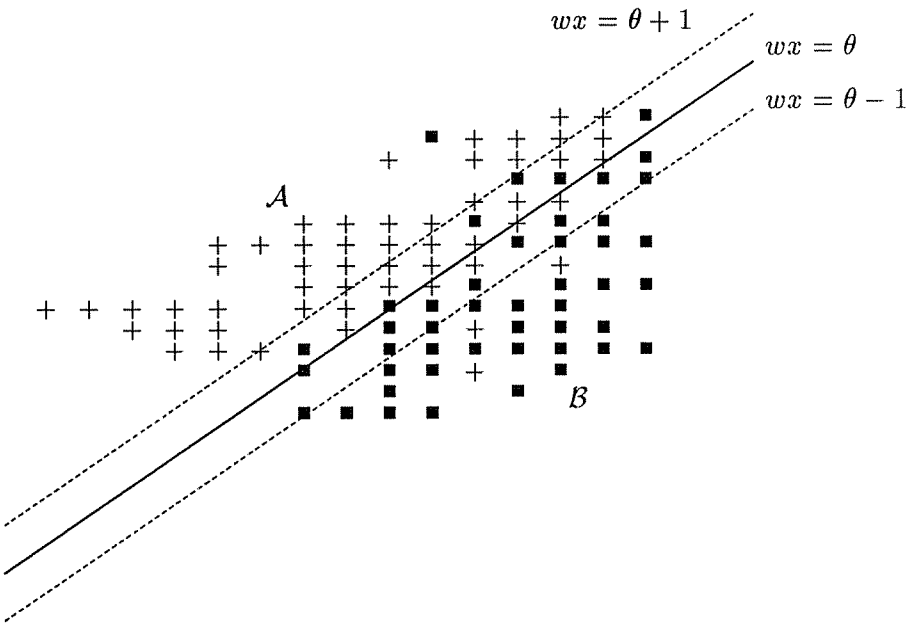

Figure 6: An optimal separator $wx = \gamma$ for linearly inseparable sets: $\mathcal{A}$ and $\mathcal{B}$.

(b) If $\mathcal{A}$ and $\mathcal{B}$ are linearly inseparable, the linear program (6) always provides an error-minimizing plane $\{x \mid wx = \theta\}$ that approximately separates $\mathcal{A}$ from $\mathcal{B}$, where (w, θ, y, z) is a solution of (6) with $w \neq 0$. The point $(w = 0, \theta, y, z)$ is a solution of (6) if and only if $\frac{eA}{m} = \frac{eB}{k}$, in which case it is never unique in $w = 0$. That is, there always exists a solution to (6) with a nonzero w.

Figure 6 depicts an actual approximate linear separation obtained by solving the linear program (6) for two linearly inseparable sets in R^2 shown in Figure 6. In this case the dead zone $\{x \mid \theta - 1 < wx < \theta + 1\}$ does contain points from both sets because the sets are not linearly separable. The importance of obtaining an approximate separating plane lies in the fact that such plane is a computational building block for a multisurface method of pattern separation which, as shown in the previous section for the XOR example, is equivalent to a neural network with a hidden layer. We shall demonstrate this application by obtaining the multisurface separation depicted in Figure 3 for the XOR example and hence the equivalent neural network of Figure 4. By letting

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

and solving the linear program (6) using MINOS 5.3 [34] we obtain the solution

$$w = (-2 \quad +2),\ \theta = 1,\ y = (4 \quad 0),\ z = (0 \quad 0) \tag{7}$$

(We note in passing that

$$w = (0 \quad 0),\ \theta = 0,\ y = (1 \quad 1),\ z = (1 \quad 1) \tag{8}$$

is also a solution but is neither unique in w nor is it the one given by MINOS.) The multisurface method tree (MSMT) [4] can be employed to obtain the complete separation of Figure 3 as follows. Note that the plane $wx = \theta$ for this problem

$$-2x_1 + 2x_2 = 1, \tag{9}$$

obtained from the solution (7) and depicted in Figure 3 as $w^1x = \theta^1$, separates R^2 into halfspaces one of which contains the point $(0, 1) \in \mathcal{A}$ only while the other halfspace contains points from both $\mathcal{A}$ and $\mathcal{B}$. MSMT now repeats the application of the linear program (6) to the halfspace containing the unseparated points: $(1, 0) \in \mathcal{A}$, $(0, 0) \in \mathcal{B}$ and $(1, 1) \in \mathcal{B}$ by solving (6) with

$$A = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$$

and obtaining the optimal solution

$$w = (2 \quad -2),\ \theta = 1,\ y = 0,\ z = (0 \quad 0) \tag{10}$$

This generates the plane $wx = \theta$:

$$2x_1 - 2x_2 = 1 \tag{11}$$

depicted in Figure 3 as $w^2x = \theta^2$. This plane completely separates the remaining points: $(1, 0)) \in \mathcal{A}$ from $\{(0, 0), (1, 1)\} = \mathcal{B}$. As was shown in Section 1, the two-plane separation depicted in Figure 3 is equivalent to the neural network of Figure 4.

More generally, we can think of a feedforward neural network with a single hidden layer of h LTU's as a representation of h planes in R^n that divide the space into p polyhedral regions, $p \leq \sum_{i=0}^{n} \binom{h}{i}$ [18], each containing elements of only one set $\mathcal{A}$ or $\mathcal{B}$. These planes, which are represented by the vector-weighted incoming arcs to the hidden LTU's together with their threshold values, map each polyhedral region into the vertices of the unit cube $C^h = \{r \mid r \in R^h,\ 0 \leq r \leq e\}$. The scalar-weighted outgoing arcs from the hidden LTU's to the output LTU together with its threshold, represent a plane in R^h which separates vertices of C^h to which $\mathcal{A}$ has been mapped from those to which $\mathcal{B}$ has been mapped. Figure 7 depicts an example in R^2 of two disjoint point sets $\mathcal{A}$ and $\mathcal{B}$ which have been compartmentalized into 7 polyhedral regions by 3 LTU's labeled 1 to 3. Each region is tagged by a 3-digit binary number determined (from left to right) by whether the region lies on the 1-side or 0-side of the LTU planes 1 to 3, as determined by the normals to these planes. Figure 8 depicts the 7 vertices of the unit cube to which the 7 polyhedral regions have been mapped as well as the plane $3r_1 + r_2 + r_3 = 1.5$ that separates the points $(1, 1, 0)$, $(0, 1, 1)$ $(1, 0, 1)$, $(1, 0, 0)$ representing $\mathcal{A}$ from the points $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$ representing $\mathcal{B}$. Figure 9 depicts the feedforward neural network corresponding to the composite mapping of Figures 7 and 8.

We describe now in more detail the greedy linear-programming-based algorithm MSMT (multisurface method tree), for constructing polyhedral regions in R^n similar to those of Figure 7, that will discriminate between two disjoint point sets $\mathcal{A}$ and $\mathcal{B}$ in R^n [4]. The essence of the method consists of applying the linear program (6) first to points contained in R^n and then in succession to points contained in appropriate intersections of complementary halfspaces. These intersections of halfspaces are generated by the planes $w^{ij}x = \theta^{ij}$, $i = 0, \ldots, \lambda$, $j = 1, \ldots, 2^i$, where $\lambda + 1$ is the number of levels in the decision tree representing groups of successive splits, i is the tree level and j is the specific node at level i. The splitting of an intersection of halfspaces terminates when

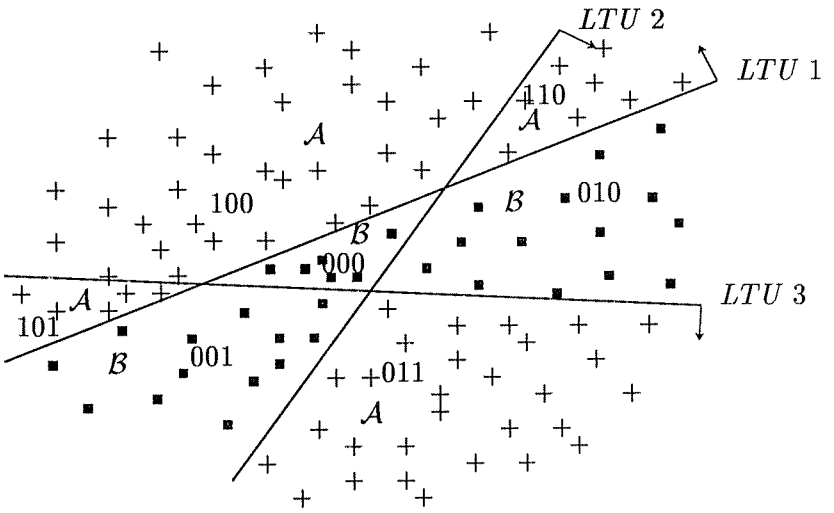

Figure 7: Compartmentalization of R^2 by 3 LTU's: $w^1x = \theta$, $w^1x = \theta^2$ and $w^3x = \theta^3$ into seven polyhedral regions, each containing elements of only one set $\mathcal{A}$ or $\mathcal{B}$. Each polyhedral region is tagged by a binary number denoting whether the ith digit of which the region is on the 1-side or 0-side of LTUi, $i = 1, 2, 3$.

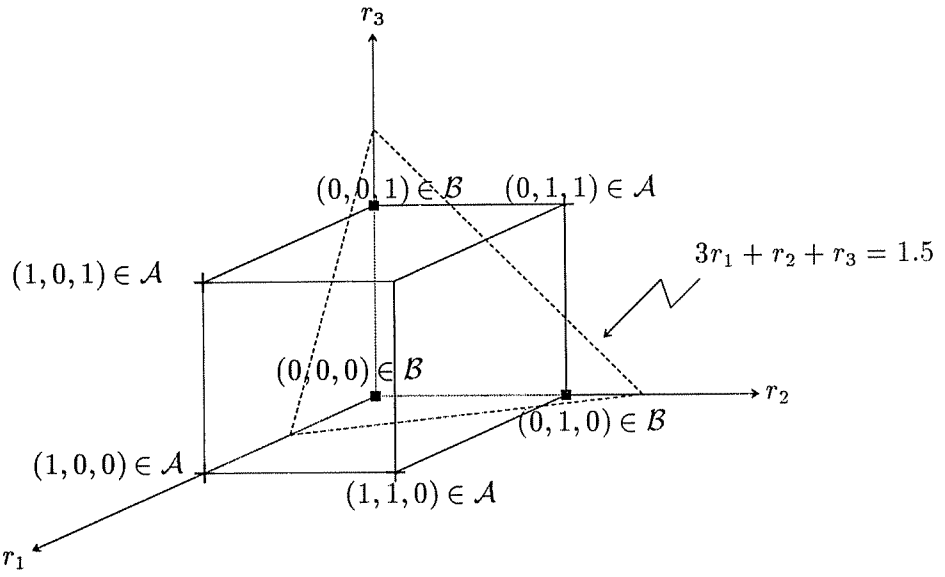

Figure 8: The vertices of the unit cube into which the sets $\mathcal{A}$ and $\mathcal{B}$ of Figure 7 are mapped, together with the plane separating them.

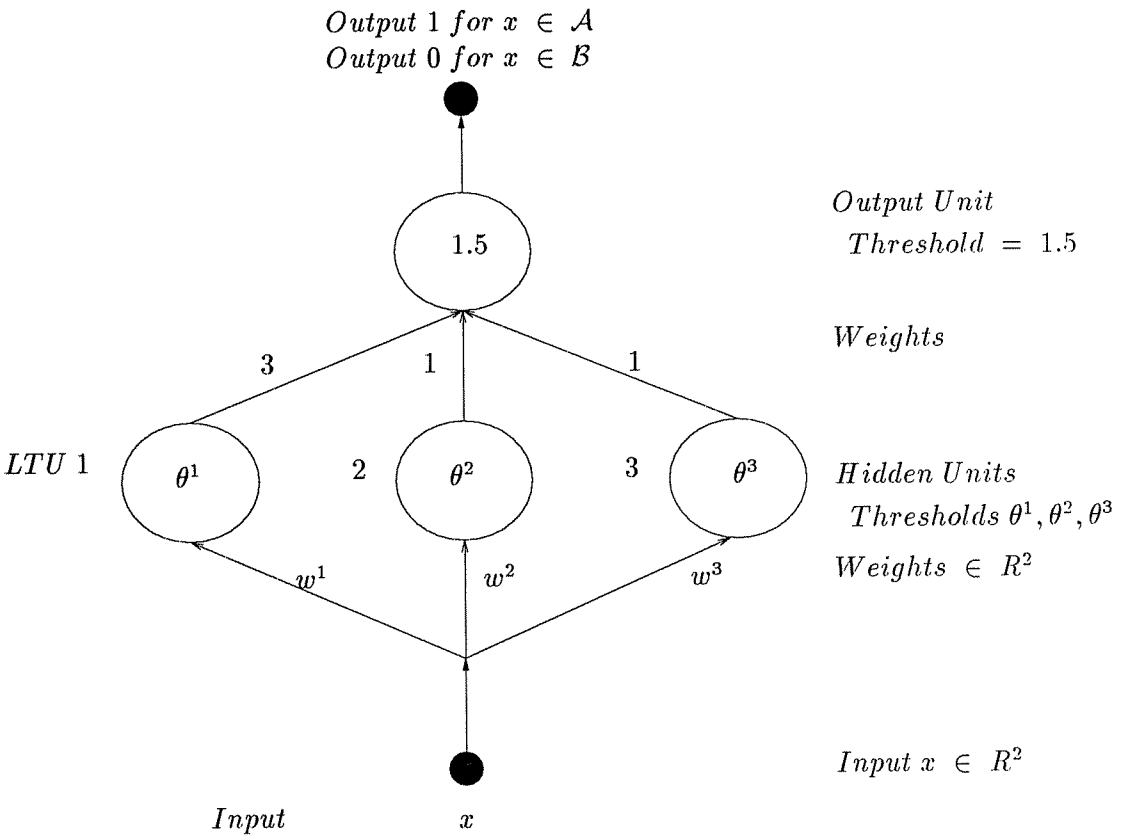

Figure 9: The feedforward neural network corresponding to the composite mapping of Figures 7 and 8.

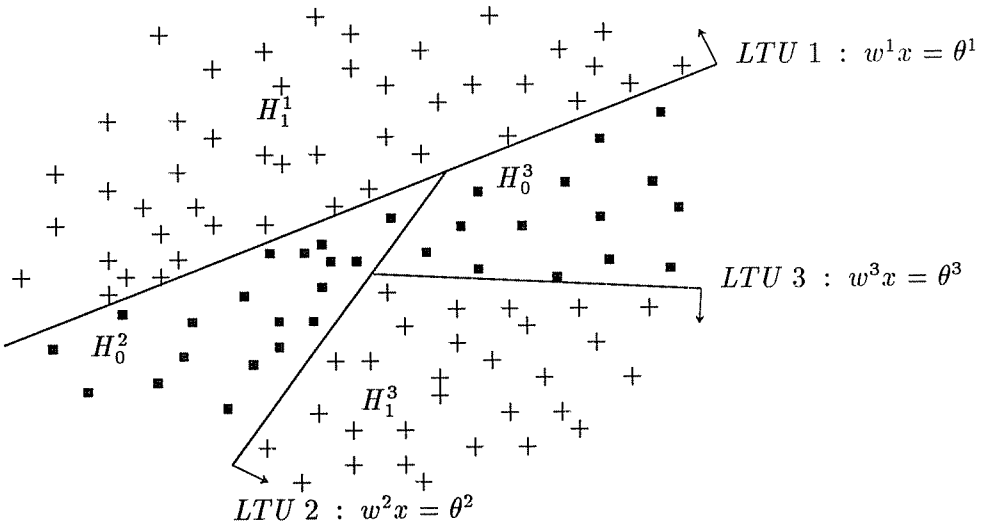

Figure 10: Splitting of R^2 by the greedy LP-based algorithm MSMT.

the intersection contains mostly elements of one set only to some desired percentage. Figure 10 depicts how MSMT is applied to the example of Figure 7. First the space R^2 is split into the complementary halfspaces $H_1^1 = \{x \mid w^1x > \theta^1\}$ and $H_0^1 = \{x \mid w^1x \leq \theta^1\}$. Since H_1^1 contains points of $\mathcal{A}$ alone, it is not split further. H_0^1 is split into H_0^2, which contains only points of $\mathcal{B}$, and H_1^2. The latter is finally split into H_0^3 containing points of $\mathcal{B}$ only and H_1^3 containing points of $\mathcal{A}$ only. The decision tree corresponding to this splitting of R^2 is shown in Figure 11. We now formally describe MSMT. We use the notation $A^{ij} \approx \phi$ to denote that the matrix A^{ij} is empty or nearly empty according to some prescribed tolerance, that is it contains no rows or very few rows relative to m, the number of rows in A. Here A^{ij} is the submatrix of rows of A representing the remaining points of the set $\mathcal{A}$ (at node j at level i of the decision tree, $i = 0, \ldots, \lambda$, $j = 1, \ldots, 2^i$) that need to be separated from the corresponding remaining points of the submatrix B^{ij} of B.

Algorithm 2.1 ***MSMT (Multisurface Method-Tree)*** *[4]. Let $\mathcal{A}$ and $\mathcal{B}$ be disjoint points in R^n, represented by the $m \times n$ and $k \times n$ matrices A and B respectively. Denote the linear program (6) by LP(A, B) and its solution set by* arg min *LP(A, B).*

(a) Initialization: Set tree level $i = 0$, $A^{01} = A$, $B^{01} = B$

(b) At tree level i: For $j = 1, 2, \ldots, 2^i$:

- Stop if $A^{ij} \approx \phi$ *or* $B^{ij} \approx \phi$ for all j
- Solve at most 2^i LP's: LP(A^{ij}, B^{ij}) for all j for which $A^{ij} \not\approx \phi$ & $B^{ij} \not\approx \phi$
- Let $(w^{ij}, \theta^{ij}, y^{ij}, z^{ij}) \in$ arg min LP(A^{ij}, B^{ij}) for all j
- Define the 2 "descendents" of each A^{ij} and B^{ij} such that $A^{ij} \not\approx \phi$ & $B^{ij} \not\approx \phi$:

$$A^{(i+1)(2j-1)} := \{A_\ell^{ij} \mid A_\ell^{ij} w^{ij} \leq e\theta^{ij}\}, \quad B^{(i+1)(2j-1)} := \{B_\ell^{ij} \mid B_\ell^{ij} w^{ij} \leq e\theta^{ij}\}$$
$$A^{(i+1)(2j)} := \{A_\ell^{ij} \mid A_\ell^{ij} w^{ij} > e\theta^{ij}\}, \quad B^{(i+1)(2j)} := \{B_\ell^{ij} \mid B_\ell^{ij} w^{ij} > e\theta^{ij}\}$$

where A_ℓ^{ij} denotes row ℓ of A^{ij}

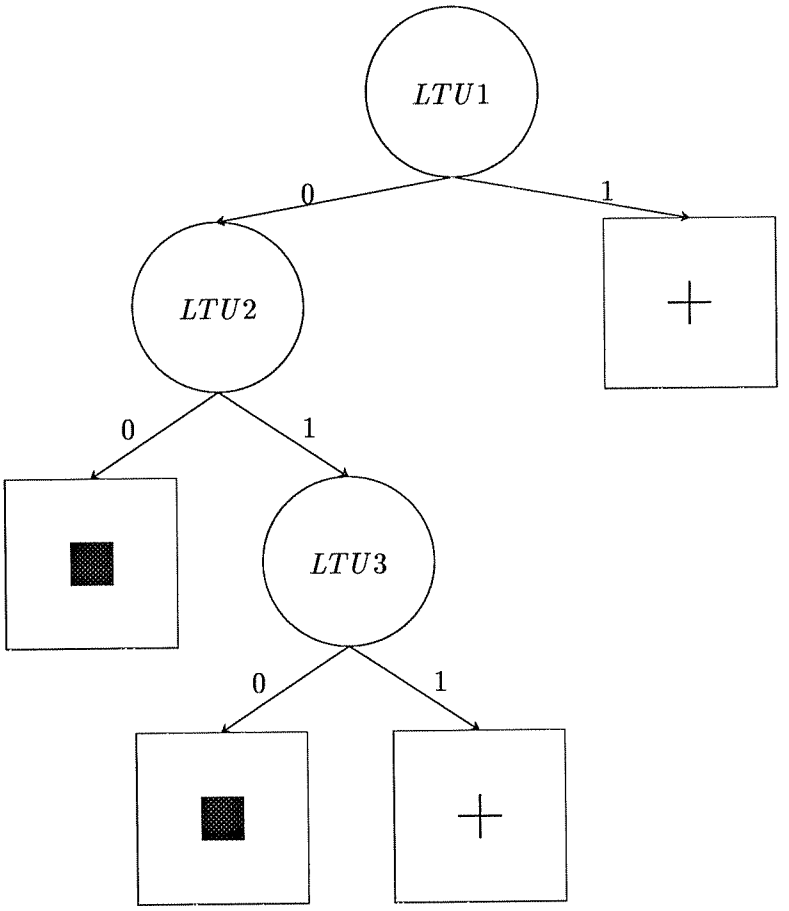

Figure 11: The 3-level decision tree corresponding to the splitting of R^2 by three LTU's.

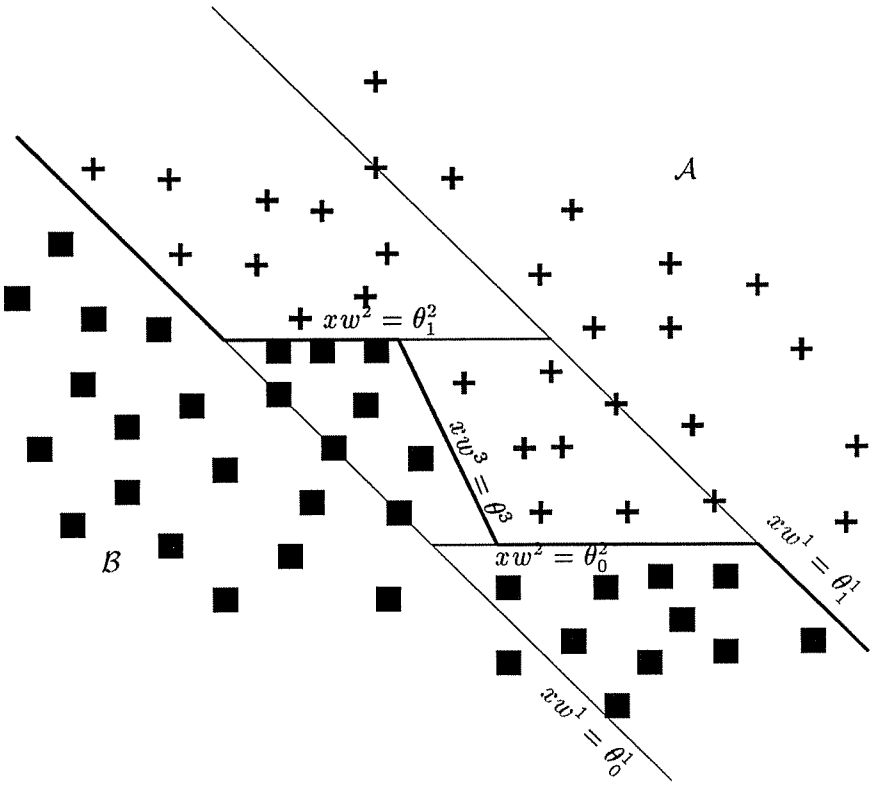

Figure 12: Separation of $\mathcal{A}$ and $\mathcal{B}$ by a piecewise-linear surface generated by MSM.

(c) Increment tree level: $i + 1 \rightarrow i$ & go to (b).

We describe now another multisurface method that is a variant of the original MSM method [28, 29], which is based on solving the single linear program (6) at each step instead of solving $2n$ linear programs as originally proposed [28, 29]. We first give a geometric description of the method and refer to Figure 12. The method obtains the piecewise-linear separator, depicted by a heavy line in Figure 12, as follows. First, the pair of parallel planes $xw^1 = \theta_1^1$ and $xw^1 = \theta_0^1$ are constructed from the solution of the linear program (6) such that $\theta_1^1 > \theta_0^1$ and the halfspace $\{x \mid xw^1 > \theta_1^1\}$ contains points from $\mathcal{A}$ only while the halfspace $\{x \mid xw^1 < \theta_0^1\}$ contains points from $\mathcal{B}$ only. The separated points in these halfspaces are now discarded and the process is repeated for the points between the planes $xw^1 = \theta_1^1$ and $xw^1 = \theta_0^1$, thus obtaining the parallel planes $xw^2 = \theta_1^2$ and $xw^2 = \theta_0^2$, which again separate part of $\mathcal{A}$ from part $\mathcal{B}$. Discarding these separated points again, the final remaining points between the planes $xw^2 = \theta_1^2$ and $xw^2 = \theta_0^2$ are separated by the plane $xw^3 = \theta^3$. The piecewise-linear separation of Figure 12 is depicted as a decision tree in Figure 13. We now formally describe the MSM algorithm, omitting a rarely needed antidegeneracy procedure [28, 29] that guards against the cases when some pair of planes does not separate any points, that is $A^{i+1} = A^i$ **and** $B^{i+1} = B^i$ for some i in the terminology of the MSM algorithm below.

Algorithm 2.2 ***MSM (Multisurface Method)*** *[28, 29] Let $\mathcal{A}$ and $\mathcal{B}$ be disjoint point sets in R^n, represented by the $m \times n$ and $k \times n$ matrices A and B respectively. Denote the linear program*

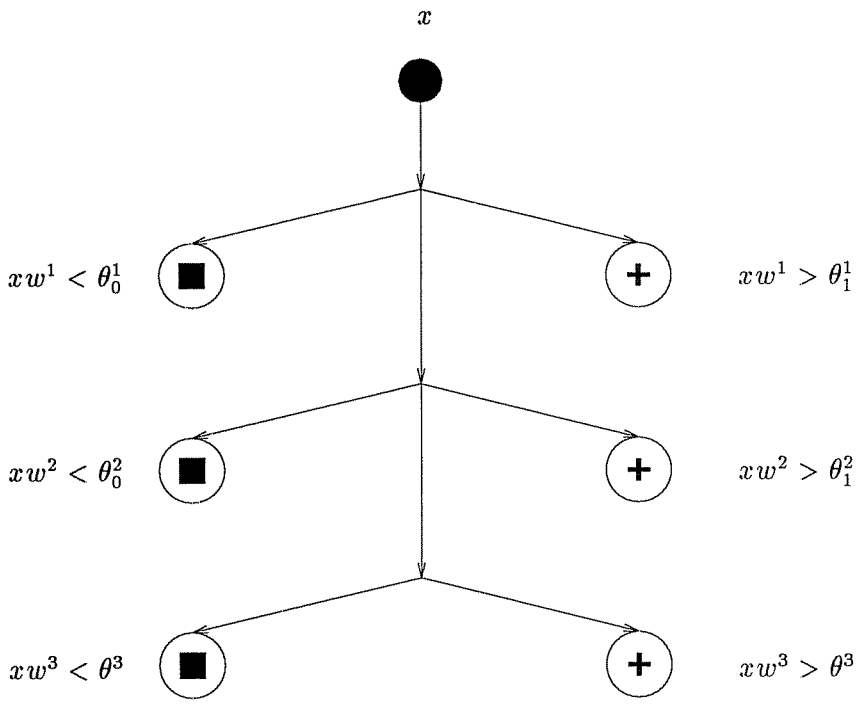

Figure 13: Decision tree representation of the MSM piecewise-linear separation of Figure 12.

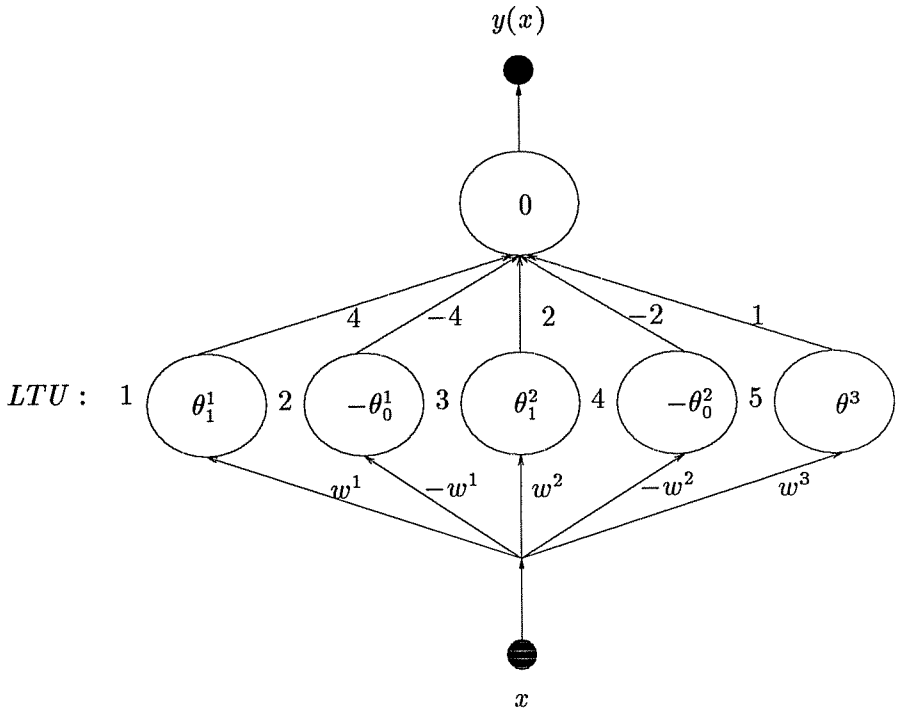

Figure 14: Neural network representation of the MSM piecewise-linear separation of Figure 12.

(6) by LP(A, B) and its solution set by arg min *LP(A, B).*

(a) Initialization: Set tree level $i = 0$, $A^0 = A$, $B^0 = B$

(b) At tree level i:

- Stop if $A^i \approx \phi$ *or* $B^i \approx \phi$
- Solve LP(A^i, B^i) & let $(w^i, \theta^i, y^i, z^i) \in \arg\min \text{LP}(A^i, B^i)$. Define $\theta_0^i := \min_j A_j^i w^i$, $\theta_1^i := \max_j B_j^i w^i$
- Define the descendents of A^i and B^i:
- $A^{i+1} := \{A_\ell^i \mid A_\ell^i w^i \leq \theta_1^i\}$, $\quad B^{i+1} := \{B_\ell^i \mid B_\ell^i w^i \geq \theta_0^i\}$

(c) Increment tree level: $i + 1 \rightarrow i$ & go to (b).

We remark that the separation achieved by the MSM algorithm and depicted in Figure 12 and 13 can also be represented as a single-hidden layer neural network [7] that is depicted in Figure 14, which we interpret now. The first pair of hidden LTU's with their incoming arcs represent the planes $w^1 x = \theta_1^1$ and $w x^1 = \theta_0^1$. Because $\theta_1^1 > \theta_0^1$, only one of them can be activated by a specific x. Hence when LTU1 fires, LTU2 does not, and since $4 > 2 + 1$, it follows that $x \in \mathcal{A}$ no matter what the other LTU's do. Similarly when LTU2 fires it follows that $x \in \mathcal{B}$. When neither LTU1 or 2 fire, the decision is relegated to LTU3 and LTU4 which again determine in a similar manner whether $x \in \mathcal{A}$ or $x \in \mathcal{B}$. If neither fires, the decision is relegated to LTU5 which fires if $x \in \mathcal{A}$ and does not

if $x \in \mathcal{B}$. Note that in the more general case, it is possible that $\theta_1^1 = \theta_0^1$ in which case LTU1 and LTU2 can fire simultaneously and thus canceling each other in which case the decision is relegated to LTU3 and LTU4. A similar remark applies to θ_1^2 and θ_0^2. See [7] for more explanation of this neural network representation of MSM.

It should be pointed out here that the MSM Algorithm 2.2 is stated in its simplest form above for exposition purposes. In order to cover possible degenerate cases that are not usually encountered in practice, one has to provide for the possibility of both $A^i = A^{i+1}$ and $B^i = B^{i+1}$ for some i [28, 29]. One such procedure is given below.

2.4 Degeneracy Procedure for MSM Algorithm Insert before step (c) of MSM Algorithm 2.2 the following step:

(b1) If $A^{i+1} = A^i$ and $B^{i+1} = B^i$, find **new** w^i, θ_0^i, θ_1^i satisfying

$$A^i w^i \geq e\theta_0^i,\ B^i w^i \leq e\theta_1^i,\ \theta_0^i < \theta_1^i$$

and

$$A^{i+1} \neq A^i \ \ or \ \ B^{i+1} \neq B^i$$

and go to (c).

We note that step (b) of the Degeneracy Procedure can be achieved in a number of ways, the simplest of which is to construct a plane $xw^i = \theta_1^i$ that chops off one or more points from A^i but no points from B^i, or to construct a plane $xw^i = \theta_0^i$ that chops off one or more points from B^i but no points from A^i.

With the Degeneracy Procedure in place, one can assert, as was done in [28, 29] that MSM can discriminate between any two disjoint point sets in R^n, provided that a sufficient number of planes are used. In terms of neural network terminology this is equivalent to the following.

Theorem 2.2 ***Neural Network as Universal Separator*** *[24, 29] A neural network with a single hidden layer and sufficient number of hidden LTU's can completely separate any two disjoint points sets in R^n.*

We note that MSM is a greedy algorithm that generates various pieces of a separating surface between the sets $\mathcal{A}$ and $\mathcal{B}$ sequentially. Although MSM has produced very effective practical results [49, 29, 7], optimality of the separating surface, as measured by the number of planes constituting it, cannot be guaranteed. Unfortunately, the problem of deciding whether two sets are separable by as few planes as two is NP-complete [31, 10]. Although there are effective bilinear programming algorithms for solving the bilinear separability problem [5], there are no methods for directly obtaining piecewise-linear separators other than the proposed greedy MSM algorithms [28, 29, 8]. We note that in these greedy MSM algorithms, each piece of the separating surface can be nonlinear as long as it is linear in its parameters such as a quadratic surface for instance. The separation can still be achieved by solving a linear program [27, 40] for each piece.

We turn now to the case of multicategory discrimination, that is, discriminating between the elements of k disjoint point sets in R^n, and show how it can be set as a single linear program. Smith [46] proposed solving k linear programs separating each set from the remaining $k-1$ sets. However, in [6] a **single** linear program is solved to obtain a convex k-piece piecewise-linear surface that exactly separates k disjoint sets under certain conditions, otherwise an error-minimizing approximate separation is obtained. Separation is achieved by having the ith linear piece of the surface exceed in value all other linear pieces over the set $\mathcal{A}^i$, for $i = 1, \ldots, k$. We give this linear programming formulation below.

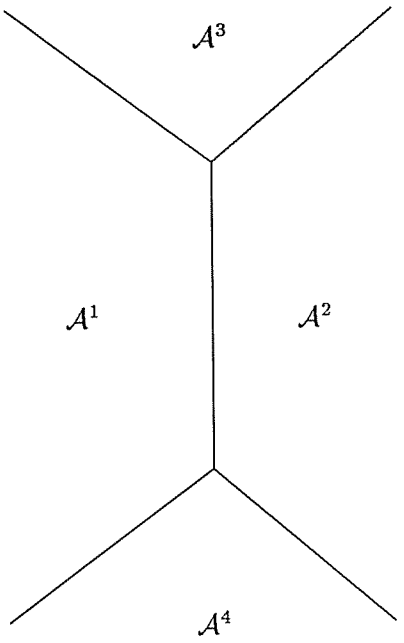

Figure 15: Separation of the sets A^1, A^2, A^3 *and* A^4 by a piecewise-linear surface.

Theorem 2.3 ***(Multicategory Separation)*** *The disjoint sets A^i in $R^n, i = 1, \ldots, k$, represented by the $m^i \times n$ matrices A^i, $i = 1, \ldots, k$, are piecewise-linear separable if and only if the solvable linear program*

$$\min_{w,\gamma^i,y^{ij}} \left\{ \sum_{i=1}^{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} \frac{ey^{ij}}{m^i} \mid y^{ij} \geq -A^i(w^i - w^j) + e(\gamma^i - \gamma^j) + e,\ y^{ij} \geq 0,\ i \neq j,\ j = 1, \ldots, k \right\} \quad (12)$$

has a zero minimum in which case, any solution $(w^i,\ \gamma^j\ y^{ij})$, $i, j = 1, \ldots, k$, $i \neq j$, provides a piecewise-linear separation as follows:

$$A^i w^i - e\gamma^i \geq A^i w^j - e\gamma^j + e,\ i, j = 1, \ldots, k,\ i \neq j \quad (13)$$

Figure 15 depicts a typical separation of 4 sets A^1, A^2, A^3 and A^4 by a piecewise-linear surface obtained by solving the single linear program (12). For more details see [6].

3 Neural Networks as Unconstrained Minimization Problems

In this section we cast the problem of determining the weights and thresholds of a feedforward neural network with a single hidden layer as an unconstrained optimization problem, and relate this problem to the standard backpropagation algorithm [41, 44, 23] for training such a neural network.

We consider the neural network depicted in Figure 16 with an input vector x in R^n, h hidden LTU's with threshold values $\theta^i \in R$, incoming arc weights $w^i \in R^n$, outgoing arc weights $v^i \in$

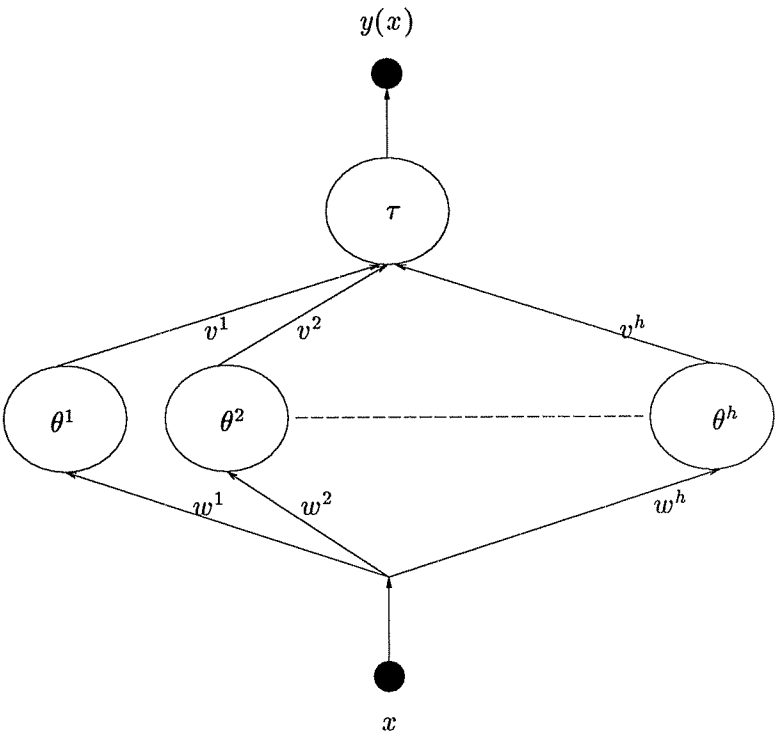

Figure 16: A typical feedforward neural network with a single layer of h hidden LTU's, input $x \in R^n$ and output $y(x) \in \{0, 1\}$.

R, $i = 1, \ldots, h$, and an output LTU with a threshold $\tau \in R$ and output $y(x) \in \{0, 1\}$. We define our problem as follows.

Problem 3.1 ***Neural Network Training Problem*** *Given the disjoint point sets $\mathcal{A}$ and $\mathcal{B}$ in R^n, determine a positive integer h, $w^i \in R^n$, $\theta^i \in R$, $v^i \in R$, $i = 1, \ldots, h$, and $\tau \in R$, such that the output $y(x)$ of the neural network of Figure 16 satisfies $y(x) = 1$ for $x \in \mathcal{A}$ and $y(x) = 0$ for $x \in \mathcal{B}$.*

Theorem 2.2 tells us that for sufficiently large h, Problem 3.1 is completely solvable. However, it is not easy to determine what h should be, nor is it desirable to make h large enough to completely solve Problem 3.1. This latter point has its analog in approximation theory where one always tries to approximate a given set of data points by the simplest possible function, typically a lowest order polynomial. In fact this point touches on a broad topic in the machine learning field that goes under the name of "generalization", that is how well a trained neural network does on unseen data. One approach is to obtain a least value of h for which an approximate solution to Problem 3.1 is satisfactory in the sense that $y(x)$ has a tolerable error in it [26, 23]. Note that both of our Algorithms MSMT 2.1 and MSM 2.2 essentially take this approach. In the sequel, however, we shall assume that h is fixed. An interesting empirical study of generalization in machine learning has been carried out on several real world data sets in [43].

For the neural network depicted in Figure 16 we can define the mean square error for a given set of weights and thresholds $w^i \in R^n$, $v^i \in R$, $\theta^i \in R$, $i = 1, \ldots, h$, $\tau \in R$ as follows:

$$f(w,\theta,v,\tau) := \sum_{i=1}^{k} \left(s\left(\sum_{j=1}^{h} s(x^i w^j - \theta^j) v^j - \tau \right) - t^i \right)^2 \tag{14}$$

where

h = fixed integer number of hidden LTU's

k = fixed integer number of given samples x^i in R^n

$t^i = 0$ or 1 target value for x^i, $i = 1, \ldots, k$

τ = real number threshold of output LTU

v^j = real number weights of outgoing arcs from hidden LTU's, $j = 1, \ldots, h$

θ^j = real number thresholds of hidden LTU's, $j = 1, \ldots, h$

w^j = n-vector weights of incoming arcs to hidden LTU's, $j = 1, \ldots, h$

x^i = given n-dimensional vector samples, $i = 1, \ldots, k$

$s(\zeta) = 1$ if $\zeta > 0$ else 0

$s(\zeta) \cong \sigma(\zeta) = \dfrac{1}{1 + e^{a\zeta}}$ for some $a > 0$

The differentiable activation **sigmoid function** $\sigma(\zeta)$ is typically used as an approximation of the discontinuous step function $s(\zeta)$ in order to render the function f differentiable. We shall assume from now on that $s(\zeta) = \sigma(\zeta)$. Once this is done the problem reduces to that of finding a stationary point of the nonconvex but differentiable function f. The classical **backpropagation (BP)** method [41, 23] for solving this problem is a gradient-type method applied to its k components sequentially (**online BP**) or to the whole function f (**batch BP**). If we define the ith component of f as f_i, that is

$$f_i(w,\theta,v,\tau) := \left(s\left(\sum_{j=1}^{h} s(x^i w^j - \theta^j) v^j - \tau \right) - t^i \right)^2, i = 1, \ldots, k \tag{15}$$

and if we let

$$z = (w, \theta, v, \tau) \tag{16}$$

then a step of the **online BP** consists of

$$z^{\ell+1} = z^\ell - \varepsilon^\ell \nabla f_i(z^\ell),\ i = \ell \bmod k,\ \ell = 1, 2, \ldots \tag{17}$$

whereas a step of the **batch BP** consists of

$$z^{\ell+1} = z^\ell - \varepsilon^\ell \nabla f(z^\ell),\ \ \ell = 1, 2, \ldots \tag{18}$$

The stepsize ε^ℓ, referred to as the **learning rate**, is adjusted heuristically and typically held at some small value or allowed to approach zero. Since the online BP direction $-\nabla f_i(z^\ell)$ may not even be a descent direction, that is $\nabla f(z^\ell) \nabla f_i(z^\ell) \geq 0$, there are, to the best knowledge of the author, no published convergence proofs for the method. This is a curious fact in view of the wide

acceptability and success of the method. Recently, however, Grippo [21] has proposed a promising convergence proof under certain assumptions and based on line search techniques similar to those of [22]. Batch BP on the other hand can be treated as an ordinary unconstrained minimization gradient method, and all the machinery of search methods [12] can be applied to it. In fact, by considering the whole objective function f instead of its components f_i separately, one can apply a whole variety of first and second order methods to the problem of minimizing f. First and second order methods for various BP algorithms are given in [2, 3, 36].

More recently Gaivoronski [16] gave a convergence proof for online BP (17) under minimal conditions using stochastic gradient ideas [15] in which he established convergence of $\{\| \nabla f(z^\ell) \|\}$ to zero as well as the stationarity of each accumulation point $\overline{z}$ of $\{z^\ell\}$, that is $\nabla f(\overline{z}) = 0$. The principal assumption of the stochastic gradient proof is that the learning rate sequence $\{\varepsilon^\ell\}$ of (17) goes to zero at a sufficiently slow rate, that is

$$\varepsilon^\ell \to 0, \quad \sum_{\ell=0}^{\infty} \varepsilon^\ell = \infty, \quad \{\varepsilon^{\ell+1}/\varepsilon^\ell\} \to 1 \tag{19}$$

This condition appears to be one of the simplest conditions under which a rigorous mathematical convergence of the online BP has been established. The complete result will appear in [17]. We note also that, in a little known paper, Kibardin [25] used similar conditions to (19) for establishing convergence of (17) for convex $f_i(z)$, $i = 1, \ldots k$. Unfortunately the convexity assumption does not hold here.

We can relate the unconstrained minimization of the error function f of (14) to the linear programming approach of the previous section by considering the neural network of Figure 9 and the mapping it represents as generated by the planes of Figures 7 and 8. Here the number of hidden units h is 3. The three planes in $R^n : w^i x = \theta^i$, $i = 1, 2, 3$ of Figure 7, together with the plane in $R^3 : v^1 r_1 + v^2 r_2 + r^3 r_3 = \tau$ of Figure 8, will generate a global minimum of zero for the error function f of (14). One can therefore think of f as a measure of the failure of a particular set of weights and thresholds in achieving complete linear separation between the vertices of a unit cube in R^h into which the two categories have been mapped. We turn our attention now to an application of neural networks trained via linear programming.

4 Breast Cancer Diagnosis via Linear Programming

Neural networks have been applied to diverse classes of problems such as robot control, character and speech recognition, finance problems such as bank failure prediction and credit evaluation, oil drilling as well as medical and prognosis problems. See for example [44, 23], references therein and [43, 13]. Most of these applications use neural networks trained by backpropagation or variants thereof. We shall describe here a successful linear programming-based application to breast cancer diagnosis that is in current operation at University of Wisconsin Hospitals. Its cumulative correctness rate has been 98% over the past four years [49, 29, 7].

The diagnosis system consists of an LP-trained neural network made up of seven hidden units that was originally trained on 369 samples, each consisting of a 9-dimensional vector, and retrained once. The weights of the incoming arcs to the hidden units and the thresholds of the hidden units correspond respectively to the normals and distance from the origin of 4 pairs of parallel planes in R^9 and their relative distance from the origin [29, 7]. The first pair of planes separates some malignant points from some benign points but leaving a mixture of benign and malignant points between the planes. The second pair of planes repeats the process for the points between the first pair of

planes, and the third pair repeats it for the points between the second pair. The fourth and last pair of planes which coalesce into a single plane, completely separate the mixture of points between the third pair of planes [28], thus giving a compete separation of the training set. These planes which generate a piecewise-linear separator, were obtained by a linear programming approach [28, 29] slightly different from MSM described in Section 2. The 9-dimensional vector consists of 9 cellular attributes measured microscopically by a surgical oncologist on a needle aspirate taken from the patient's breast. Malignant diagnosis is confirmed by subsequent biopsy of breast tissue. Benign diagnosis is similarly confirmed if the patient so desires, otherwise it is confirmed by subsequent examination. Although this system has been quite successful, it requires the services of an experienced oncologist for making the measurements. An automated system has been developed, and recently put into use, that completely eliminates the subjective assessment by the oncologist [48]. Instead vision techniques are used to draw boundaries around the nuclei of a few cells from which 30 numerical features are extracted. Three of these features enable a neural network, as simple as a single LTU, to diagnose between benign and malignant samples with an average accuracy of 96% both on a training set as well as on a randomly selected testing set not included in the training set. We are also applying the automated system to the more difficult problem of breast cancer prognosis and plan further applications to other medical diagnosis and prognosis problems.

5 Conclusion

We have shown how a fundamental tool of machine learning, a neural network, can be trained using mathematical programming techniques. There are basically two approaches to this problem: one based on linear programming where a succession of planes is used to divide the input space into polyhedral regions each of which containing points of mostly one category. The other approach reduces the problem to an unconstrained minimization problem of a nonconvex but differentiable error function that can be optimized by gradient-type methods that use either first or second order information. We also gave a brief description of a real-world application of the linear-programming-based approach to an important medical diagnosis problem. We conclude that optimization theory and algorithms play a significant role in the algorithmic and applied development of the burgeoning neural network field of machine learning [42]. Whether the converse is also true, is an interesting but not completely settled question [23, pages 76–79], even though there have been a number of interesting applications of neural networks to optimization problems, for example [37, 1, 14] and [23, pages 71–87].

Acknowledgement

I wish to thank my colleagues Yann le Cun, Renato De Leone, Laurence Dixon, Greg Madey, Jim Noyes, Boris Polyak, Steve Robinsosn and Jude Shavlik for helpful references that they furnished, some of which have been cited. Thanks also to my student Kristin Bennett for the many ideas on neural networks that we have collaborated on and for help in preparing the figures, and to my colleague Jude Shavlik and students Mikhail Solodov and Nick Street for reading the manuscript and making constructive suggestions.

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzman Machines.* John Wiley & Sons, Chichester, 1990.

[2] R. Battiti. First- and second-order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, 4:141–166, 1992.

[3] S. Becker and Y. le Cun. Improving the convergence of backpropagation learning with second order methods. In D.S. Touretzky, editor, *Proceedings of 1988 Connectionist Models Summer School*, pages 29–37, San Mateo, California, 1988. Morgan Kaufmann.

[4] K. P. Bennett. Decision tree construction via linear programming. In M. Evans, editor, *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society Conference*, pages 97–101, 1992.

[5] K. P. Bennett and O. L. Mangasarian. Bilinear separation of two sets in n-space. Computer Sciences Department Technical Report 1109, University of Wisconsin, Madison, Wisconsin, 1992.

[6] K. P. Bennett and O. L. Mangasarian. Multicategory separation via linear programming. Computer Sciences Department Technical Report 1127, University of Wisconsin, Madison, Wisconsin, 1992.

[7] K. P. Bennett and O. L. Mangasarian. Neural network training via linear programming. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 56–67, Amsterdam, 1992. North Holland.

[8] K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

[9] H.D. Block and S.A. Levin. On the boundedness of an iterative procedure for solving a system of linear inequalities. *Proceedings of the American Mathematical Society*, 26:229–235, 1970.

[10] A. Blum and R.L. Rivest. Training a 3-node neural network is NP-complete. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 494–501, San Mateo, California, 1989. Morgan Kaufmann.

[11] A. Charnes. Some fundamental theorems of perceptron theory and their geometry. In J. T. Lou and R. H. Wilcox, editors, *Computer and Information Sciences*, pages 67–74, Washington, 1964. Spartan Books.

[12] J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, N.J., 1983.

[13] L. DeSilets, B. Golden, Q. Wang, and R. Kumar. Predicting salinity in the Chesapeake Bay using backpropagation. *Computers and Operations Research*, 19:277–285, 1992.

[14] L.C. Dixon and D.J. Mills. Neural nets for massively parallel optimisation. Technical Report Numerical Optimisation Centre Technical Report No. 259, Hatfield Polytechnic, Hatfield, Hertforshire, England, 1992.

[15] Yu. Ermoliev and R.J.-B. Wets (editors). *Numerical Techniques for Stochastic Optimization Problems*. Springer-Verlag, Berlin, 1988.

[16] A.A. Gaivoronski. Private communication, November 1992.

[17] A.A. Gaivoronski. Stochastic gradient methods for neural networks. In O.L. Mangasarian and R.R. Meyer, editors, *Parallel Optimization 3*, Philadelphia, 1994. SIAM. Proceedings of Symposium on Parallel Optimization 3, Madison July 7-9, 1993.

[18] G.M. Georgiou. Comments on hidden nodes in neural nets. *IEEE Transactions on Circuits and Systems*, 38:1410, 1991.

[19] F. Glover. Improved linear programming models for discriminant analysis. *Decision Sciences*, 21:771–785, 1990.

[20] R.C. Grinold. Mathematical methods for pattern classification. *Management Science*, 19:272–289, 1972.

[21] L. Grippo. Private communication, September 1992.

[22] L. Grippo, F. Lampariello, and S. Lucidi. Global convergence and stabilization of unconstrained minimization methods without derivatives. *Journal of Optimization Theory and Applications*, 56:385–406, 1988.

[23] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, California, 1991.

[24] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[25] V.M. Kibardin. Decomposition into functions in the minimization problems. *Automation and Remote Control*, 40:1311–1323, 1980.

[26] Y. le Cun, J.S. Denker, and S.A. Solla. Optimal brain damage. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems II (Denver 1989)*, pages 598–605, San Mateo, California, 1990. Morgan Kaufmann.

[27] O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.

[28] O. L. Mangasarian. Multi-surface method of pattern separation. *IEEE Transactions on Information Theory*, IT-14:801–807, 1968.

[29] O.L. Mangasarian, R. Setiono, and W.H. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. In T. F. Coleman and Y. Li, editors, *Proceedings of the Workshop on Large-Scale Numerical Optimization, Cornell University, Ithaca, New York, October 19-20, 1989*, pages 22–31, Philadelphia, Pennsylvania, 1990. SIAM.

[30] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133, 1943.

[31] N. Megiddo. On the complexity of polyhedral separability. *Discrete and Computational Geometry*, 3:325–337, 1988.

[32] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, Massachusetts, 1969.

[33] T. S. Motzkin and I. J. Schoenberg. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:393–404, 1954.

[34] B.A. Murtagh and M.A. Saunders. MINOS 5.0 user's guide. Technical Report SOL 83.20, Stanford University, December 1983.

[35] N. J. Nilsson. *Learning Machines.* MIT Press, Cambridge, Massachusetts, 1966.

[36] J.L. Noyes. Neural network optimization methods. In *Proceedings of the Fourth Conference on Neural Networks and Parallel Distributed Processing*, pages 1–12, Fort Wayne, Indiana, 1991. Indiana-Purdue University.

[37] K.E. Nygard, P. Juell, and N. Kadaba. Neural networks for selecting vehicle routing heuristics. *ORSA Journal on Computing*, 4:353–364, 1990.

[38] F. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, New York, New York, 1962.

[39] F. Rosenblatt. The perceptron–a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Itahca, New York, January 1957.

[40] A. Roy and S. Mukhopadhyay. Pattern classification using linear programming. *ORSA Journal of Computing*, 3:66–80, 1990.

[41] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing.* MIT Press, Cambridge, Massachusetts, 1986.

[42] J.W. Shavlik and T.G. Dietterich (editors). *Readings in Machine Learning.* Morgan Kaufman, San Mateo, California, 1990.

[43] J.W. Shavlik, R.J. Mooney, and G.G. Towell. Symbolic and neural network learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143, 1991.

[44] P.K. Simpson. *Artificial Neural Systems.* Pergamon Press, New York, 1990.

[45] F. W. Smith. Pattern classifier design by linear programming. *IEEE Transactions on Computers*, C-17:367–372, 1968.

[46] F. W. Smith. Design of multicategory pattern classifiers with two-category classifier design procedures. *IEEE Transactions on Computers*, 18:367–372, 1969.

[47] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behaviorial Sciences.* PhD thesis, Harvard University, 1974.

[48] W.N.Street, W.H. Wolberg, and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. In *IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology*, volume 105, San Jose, California, 1993.

[49] W. H. Wolberg and O.L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences, U.S.A.*, 87:9193–9196, 1990.