Learning with Discrete Multi-Valued Neurons

Zoran Obradovic[1] and Ian Parberry

Department of Computer Science,
Penn State University,
University Park, Pa. 16802.

Abstract

Analog neural networks of limited precision are essentially k-ary neural networks. That is, their processors classify the input space into k regions using $k-1$ parallel hyperplanes by computing k-ary weighted multilinear threshold functions. The ability of k-ary neural networks to learn k-ary weighted multilinear threshold functions is examined. The well known perceptron learning algorithm is generalized to a k-ary perceptron algorithm with guaranteed convergence property. Littlestone's winnow algorithm is superior to the perceptron learning algorithm when the ratio of the sum of the weights to the threshold value of the function being learned is small. A k-ary winnow algorithm with a mistake bound which depends on this value and the ratio between the largest and smallest thresholds is presented.

[1]On leave from the Mathematical Institute, Belgrade, Yugoslavia.

1 Introduction

In Obradovic and Parberry [8] it was shown that analog neural networks of limited precision are essentially k-ary neural networks (that is, their processors classify $\mathbf{R}^n$ into k regions using $k-1$ parallel hyperplanes) and their computing power was examined. Here, we investigate their learning power. One of the results from that reference was that there is no canonical set of threshold values for a k-ary perceptron when $k > 3$, although they exist for binary and ternary neural networks. This indicates that learning algorithms for k-ary neural networks which modify only the weights are not necessary convergent. Here we show that matters can be improved by learning both the thresholds and the weights. A preliminary version of the results from this paper appear in Obradovic and Parberry [9].

The main body of this paper is divided into three sections. The first section sketches definitions of the k-ary neural network model and learning in that model. The second section contains a k-ary perceptron learning algorithm (derived from the binary perceptron learning algorithm) and its convergence proof. The third section contains a k-ary winnow algorithm (derived from Littlestone's winnow algorithm [2,3]) and its mistake bound proof.

2 A General Framework for Learning

Let $k \in \mathbf{N}$, and $\mathbf{Z}_k = \{0, \ldots, k-1\}$. A *$k$-ary neural architecture* is a k-ary neural network with the weights, thresholds and initial activation levels left unspecified. That is, it is a 4-tuple $A = (k, V, I, O)$, where:

$k \in \mathbf{N}$ is the number of logic levels,

V is a finite set of *processors*, or *gates*,

$I \subseteq V$ is a set of *input processors*,

$O \subseteq V$ is a set of *output processors*.

$A(a, w, h)$ denotes the *k-ary neural network* (k, V, I, O, a, w, h), where

$a : V - I \to \mathbf{Z}_k$ is a set of *initial activation levels*,

$w : V \times V \to \mathbf{R}$ is a *weight assignment*,

$h : V \to \mathbf{R}^{k-1}$ is a *threshold assignment*.

The processors of a k-ary neural network are relatively limited in computing power. Processor $v \in V$ has $k-1$ thresholds $h_1(v), \ldots, h_{k-1}(v)$, and if its weighted input sum is between $h_i(v)$ and $h_{i+1}(v)$, it has i for output.

More formally, a *k-ary function* is a function $f : \mathbf{Z}_k^n \to \mathbf{Z}_k$. Let F_k^n denote the set of all n-input k-ary functions. Define $\Theta_k^n : \mathbf{R}^{n+k-1} \to F_k^n$ by $\Theta_k^n(w_1, \ldots, w_n, h_1, \ldots, h_{k-1}) : \mathbf{R}_k^n \to \mathbf{Z}_k$, where

$$\Theta_k^n(w_1, \ldots, w_n, h_1, \ldots, h_{k-1})(x_1, \ldots, x_n) = i \quad \text{iff} \quad h_i \leq \sum_{i=1}^{n} w_i x_i < h_{i+1}.$$

Here and throughout this paper, we will assume that $h_1 \leq h_2 \leq \ldots \leq h_{k-1}$, and for convenience define $h_0 = -\infty$ and $h_k = \infty$. The set of *k-ary weighted multilinear threshold functions* is the union, over all $n \in \mathbf{N}$, of the range of Θ_k^n. Each processor of a k-ary neural network can compute a k-ary weighted multilinear threshold function of its inputs.

Let $A = (A_1, A_2, \ldots)$ and $f = (f_1, f_2, \ldots)$, where $A_n = (k, V_n, I_n, O_n)$ is a neural architecture with $\|I\| = n$, and $f_n : \mathbf{R}^n \rightarrow \mathbf{Z}_k$. A *learning algorithm* for f on A is a relativized algorithm L with an oracle for f which on input n outputs a series of distinct initial activation, weight, and threshold assignments $\langle a_0, w_0, h_0 \rangle$, $\langle a_1, w_1, h_1 \rangle, \ldots, \langle a_t, w_t, h_t \rangle$ such that the neural network $M_n = A_n(a_t, w_t, h_t)$ computes f_n. We will consider learning algorithms for k-ary weighted multilinear threshold functions on *neural circuits* (that is, layered neural networks without feedback).

Resources of interest include those of M_n, and those of L. The former include the size (number of processors), depth (number of layers), and weight (sum of all the weights) of the circuit. The latter include the latency and the mistake bound, defined as follows. The *latency* of learning algorithm is the worst case running time between the output of one set of assignments and the next. We will measure *unit-cost* latency, that is, we will assume that L is implemented on a digital computer with word-size large enough that each elementary arithmetic and logic operation can be implemented in constant time. The *mistake bound* is the worst case total number of distinct assignments output.

If f is a k-ary weighted multilinear threshold function, we say that $(w_1, \ldots, w_n, t_1, \ldots, t_{k-1}) \in \mathbf{R}^{n+k-1}$ is a *representation* of f iff $f = \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$. Note that each k-ary weighted multilinear threshold function has many representations.

We will consider the problem of learning k-ary weighted multilinear threshold functions, and for the most part be concerned with learning them on the minimal architecture consisting of a single k-ary processor. That is, we will be learning a representation for a k-ary weighted multilinear threshold function f, given only an oracle for f. All of our learning algorithms will be expressed in a high-level pseudocode. Initial activation levels will always be zero.

We will consider two new resources, called the *height* and *width* of a representation, which give some indication of the relationships between the weights and the thresholds, and between the thresholds themselves (respectively), to be defined later.

3 A k-ary Perceptron Learning Rule

The *perceptron learning problem* is the problem of learning binary weighted linear threshold functions on a binary neural network consisting of a single processor (called a *perceptron* for historical reasons). There is a well-known algorithm for the perceptron learning problem which uses the so-called *perceptron learning rule* to derive successive weights. The algorithm is described in Figure 1.

Theorem 3.1 (The Perceptron Convergence Theorem) *The perceptron learning algorithm for learning n-input binary weighted linear threshold functions on a single n-input perceptron described in Figure 1 terminates.*

Proof: See, for example, Duda and Hart [1], Minsky and Papert [4], Nilsson [6] or Novikoff [7]. □

The initial weights can be set to any value in the for-loop on line 1 in Figure 1. The members of $\mathbf{Z}_2^n$ can be used in any order in the for-loop on line 3, provided every member is used an infinite number of times in the algorithm. Also, the value added to w_i in the for-loop of *perceptron_update* procedure can be multiplied by some constant $c_j \in \mathbf{R}^+$ at the j^{th} call of that procedure provided

$$\lim_{m \rightarrow \infty} \sum_{i=1}^{m} c_i = \infty$$

```
procedure perceptron(n)
  for i := 1 to n do $w_i := 0$ ;
  repeat
    for each $x = (x_1, \ldots, x_n) \in \mathbf{Z}_2^n$ do  $p := \Theta_2^n(w_1, \ldots, w_n, 0)(x)$;
      if $f(x) \neq p$ then perceptron_update(x, p);
      Output $\langle w_1, \ldots, w_n \rangle$
  until  $f(x) = \Theta_2^n(w_1, \ldots, w_n, 0)(x)$  for all $x \in \mathbf{Z}_2^n$.

procedure perceptron_update(x, p)
  if $f(x) > p$ then sign := 1
    else sign := −1;
  for i := 1 to n do $w_i := w_i + \text{sign} * x_i$;
```

Figure 1: The Perceptron Learning Algorithm.

and

$$\lim_{m\to\infty} \frac{\sum_{i=1}^{m} {c_i}^2}{(\sum_{i=1}^{m} c_i)^2} = 0.$$

Furthermore, the algorithm will learn any weighted linear threshold function whose domain is some finite subset of $\mathbf{R}^n$. We will make use of this fact later. The latency of the algorithm is clearly linear in n. The worst-case mistake bound appears no better than exponential in n.

The *k-ary perceptron learning problem* is the problem of learning k-ary weighted multilinear threshold functions. The minimal architecture for learning n-input k-ary weighted multilinear threshold functions is a single k-ary weighted multilinear threshold gate with n inputs, which we will call a *k-ary perceptron.* A k-ary perceptron which computes n-input k-ary weighted multilinear threshold function $\Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$ will be depicted as in Figure 2. It was shown in Obradovic and Parberry [8] that there is no canonical set of threshold values for a k-ary perceptron when $k > 3$. This suggests that the thresholds $t_1, \ldots, t_{k-1}$ must be learned in addition to the weights $w_1, \ldots, w_n$.

Even if the threshold values are known in advance, many obvious extensions to the perceptron learning rule for the k-ary perceptron learning problem (such as that shown in Figure 3) which modify only the weights do not necessary terminate for all choices of ordering of sample inputs in line 4. For example, suppose $k = 3$ and $n = 2$ (similar examples can be found for arbitrary n and k using the same principles). Consider $f = \Theta_2^2(4, 3, 7, 8)$. Suppose we use the algorithm described in Figure 3 on a 2-input 3-ary perceptron with thresholds $t_1 = 7$ and $t_2 = 8$ to find weights w_1, w_2 such that $\Theta_2^2(w_1, w_2, 7, 8) = f$. After considering points $(2, 0)$ and $(2, 2)$, we have weights $(w_1, w_2) = (4, 2)$. All points are correctly classified using these weights except for the point $(1, 1)$. Thus there is no change to the weights until point $(1, 1)$ is considered, at which time the new weights become $(5, 3)$. Once again, all points are correctly classified using these weights except for

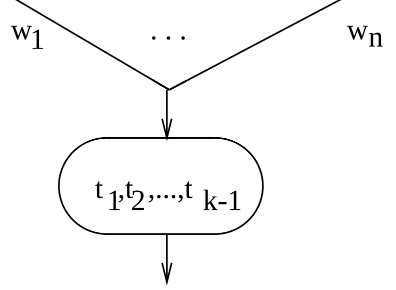

Figure 2: An Arbitrary n-input k-ary Perceptron

```
procedure thresholdperceptron$(\mathrm{n}, \mathrm{k}, \mathrm{t}_1, \ldots, \mathrm{t}_{\mathrm{k}-1})$
  for i := 1 to n do $\mathrm{w}_\mathrm{i} := 0$;
  repeat
    for each $\mathrm{x} \in \mathbf{Z}_\mathrm{k}^\mathrm{n}$ do  $\mathrm{p} := \Theta_\mathrm{k}^\mathrm{n}(\mathrm{w}_1, \ldots, \mathrm{w}_\mathrm{n}, \mathrm{t}_1, \ldots, \mathrm{t}_{\mathrm{k}-1})(\mathrm{x})$
      if $\mathrm{f}(\mathrm{x}) \neq \mathrm{p}$  then thresholdperceptron_update(x, p);
      Output $\langle \mathrm{w}_1, \ldots, \mathrm{w}_\mathrm{n} \rangle$
  until  $\mathrm{f}(\mathrm{x}) = \Theta_\mathrm{k}^\mathrm{n}(\mathrm{w}_1, \ldots, \mathrm{w}_\mathrm{n}, \mathrm{t}_1, \ldots, \mathrm{t}_{\mathrm{k}-1})(\mathrm{x})$  for all $\mathrm{x} \in \mathbf{Z}_\mathrm{k}^\mathrm{n}$.

procedure thresholdperceptron_update(x, p)
  if $\mathrm{f}(\mathrm{x}) > \mathrm{p}$ then sign := 1
    else sign := −1;
  for i := 1 to n do $\mathrm{w}_\mathrm{i} := \mathrm{w}_\mathrm{i} + \mathrm{sign} * \mathrm{x}_\mathrm{i}$;
```

Figure 3: A Trial k-ary Perceptron Learning Algorithm for Known Thresholds.

the point $(1,1)$. Thus there is no change to the weights until point $(1,1)$ is reconsidered, at which time the new weights are again $(4,2)$. Thus the weights cycle between $(4,2)$ and $(5,3)$ without ever reaching an acceptable solution. Matters are not improved by making obvious changes to Figure 3, for example, instead of adding a multiple of x_i in the for loop of *thresholdperceptron_update* procedure, substituting one if $x_i > 0$ and zero otherwise.

However, matters can be improved by learning both the thresholds and the weights. It can be shown from first principles that the k-ary perceptron learning algorithm for learning n-input k-ary weighted multilinear threshold functions on a single n-input k-ary perceptron described in Figure 4 terminates. Termination can more easily be proved as a corollary of the Perceptron Convergence Theorem as follows.

Definition. If f is an n-input k-ary weighted multilinear threshold function, the *orthogonal slice function* for f is a binary weighted linear threshold function g such that for all $x \in \mathbf{Z}_k^n$ and all $i \in \mathbf{Z}_k$,

$$f(x) \geq i \Leftrightarrow g(x, y(i)) = 1,$$

where $y : \{1, \ldots, k-1\} \rightarrow \{0,1\}^{k-1}$ is defined by

$$y(i) = (y_1, \ldots, y_{k-1}) \text{ with } y_j = 1 \text{ iff } j = i.$$

Lemma 3.2 *$f = \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$ iff $\Theta_2^{n+k-1}(w_1, \ldots, w_n, -t_1, \ldots, -t_{k-1}, 0)$ is the orthogonal slice function for f.*

Proof: Follows immediately from the definition of the orthogonal slice function □

Theorem 3.3 (The k-ary Perceptron Convergence Theorem) *The k-ary perceptron learning algorithm for learning n-input k-ary weighted multilinear threshold functions on a single n-input k-ary perceptron described in Figure 4 terminates.*

Proof: (Sketch) The learning algorithm, instead of learning f, learns the orthogonal slice function for f using the binary perceptron learning algorithm shown in Figure 1. The orthogonal slice function for f is guaranteed to exist by (the "only-if" part of) Lemma 3.2. Once the orthogonal slice function has been learned, f can be reconstructed using (the "if" part of) Lemma 3.2. The algorithm is guaranteed to terminate by the Perceptron Convergence Theorem. It is clear that the algorithm realizes Figure 4. □

The latency of the k-ary perceptron learning algorithm is $O(n)$, and the mistake bound is no worse than for the binary perceptron learning algorithm on $n + k$ inputs.

A second candidate architecture for learning k-ary weighted multilinear threshold functions consists of $\lceil \log k \rceil$ binary perceptrons, the i^{th} of which learns the i^{th} bit of the output value, together with a single k-ary weighted multilinear threshold gates with exponentially increasing weights which converts the binary output of these gates into the corresponding member of $\mathbf{Z}_k$. Unfortunately this cannot work because the last binary perceptron is expected to learn the least significant bit of the k-ary output, which is not necessarily a binary weighted threshold function.

A third candidate architecture for learning k-ary weighted multilinear threshold functions consists of a depth 2 circuit of size k. The first layer consists of $k-1$ binary perceptrons, each connected

```
procedure multiperceptron(n, k)
   for i := 1 to n do $w_i := 0$;
   for i := 1 to k − 1 do $t_i := 0$;
   repeat
      for each $x = (x_1, \ldots, x_n) \in \mathbf{Z}_k^n$ do  $p := \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})(x)$;
         if $f(x) \neq p$  then multiperceptron_update(x, p);
         Output $\langle w_1, \ldots, w_n, t_1, \ldots, t_{k-1} \rangle$
   until  $f(x) = \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})(x)$  for all $x \in \mathbf{Z}_k^n$.

procedure multiperceptron_update(x, p)
   if $f(x) > p$
      then $t_{p+1} := t_{p+1} - 1$; sign := 1
      else $t_p := t_p + 1$; sign := −1;
   for i := 1 to n do $w_i := w_i + \mathrm{sign} * x_i$;
```

Figure 4: The *k*-ary Perceptron Learning Algorithm.

```
procedure netperceptron(n, k)
   for  i := 1 to k − 1 do
      for j := 1 to n do $w_{i,j} := 0$;
   repeat
      for each $x = (x_1, \ldots, x_n) \in \mathbf{Z}_k^n$ do
         for each $i \in \mathbf{Z}_k$ do neuron_update(x, i)
      Output $\langle w_{1,1}, \ldots, w_{k-1,n} \rangle$
   until $(f(x) \geq i) \Leftrightarrow (\Theta_2^n(w_{i,1}, \ldots, w_{i,n}, 0)(x) = 1)$
            for all $x \in \mathbf{Z}_k^n$  and  $1 \leq i \leq k - 1$.

procedure neuron_update(x, i)
   $p := \Theta_2^n(w_{i,1}, \ldots, w_{i,n}, 0)(x)$;
   if $(f(x) \geq i)$ and $(p = 0)$ then sign := 1
      else if $(f(x) < i)$ and $(p = 1)$  then sign := −1
            else sign := 0;
   for j := 1 to n do $w_{i,j} := w_{i,j} + \mathrm{sign} * x_i$;
```

Figure 5: The *k*-ary Perceptron Learning Algorithm for a Depth 2 Circuit.

to all of the inputs. The second layer consists of a single k-ary perceptron, connected to all of the gates in the first layer. The thresholds of the first layer are all zero. The thresholds of the k-ary perceptron are $1, 2, \ldots, k-1$. The weights of the connections from the first layer to the second are all one. The weights of the connections from the inputs to the first layer will be learned.

Let $w_{i,j}$ denote weight from the j^{th} input to the i^{th} gate on the first layer, where $1 \leq i \leq k-1$ and $1 \leq j \leq n$. Suppose we are to learn a k-ary weighted multilinear threshold function f. The first level essentially computes the orthogonal slice function for f, and the second level converts this to a value from $\mathbf{Z}_k$. More precisely, the i^{th} gate on the first level, $1 \leq i \leq k-1$, will output one on input x iff $f(x) \geq i$. This implies that exactly $f(x)$ of the gates in the first layer will be active. The output gate sums the number of active gates in the first layer.

It is clear that by performing the binary perceptron learning rule in parallel for all $k-1$ gates in the first layer, the network will learn arbitrary k-ary weighted multilinear threshold functions. The learning algorithm is described in Figure 5. It has a latency of $O(nk)$. Its mistake bound may be better than that of Figure 4 in practice since it learns arbitrary separating hyperplanes, rather than parallel ones. However, the worst case mistake bound remains apparently exponential.

4 A k-ary Winnow Algorithm

A representation $(w_1, \ldots, w_n, t_1, \ldots, t_{k-1}) \in \mathbf{R}^{n+k-1}$ of a k-ary weighted multilinear threshold function is *positive* iff $w_i \geq 0$ for all $1 \leq i \leq n$. A k-ary weighted multilinear threshold function is *positive* iff it has a positive representation.

A positive representation $(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$ has *separation* $\lambda \in \mathbf{R}^+$, $0 < \lambda \leq 1$, if for all $x = (x_1, \ldots, x_n) \in \mathbf{Z}_k^n$, and all $i \in \mathbf{Z}_k$, $i < k-1$,

$$f(x) \leq i \quad \text{iff} \quad \sum_{j=1}^{n} w_j x_j \leq (1-\lambda) t_{i+1}.$$

The *width* of a representation $(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$ is the ratio t_{k-1}/t_1. Note that all representations of a binary weighted linear threshold function have width one. A k-ary weighted multilinear threshold function has *width* (at most) d iff it has a representation of width d.

The *height* of a representation $(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$ of width d is the ratio

$$\sum_{i=1}^{n} \frac{|w_i|}{t_{k-1}} + 1 - \frac{1}{d}.$$

A k-ary weighted multilinear threshold function has *height* (at most) h iff it has a representation of height h. A k-ary weighted multilinear threshold function is (λ, h, d)*-separable* if it has a positive representation of separation $\lambda > 0$, height h, and width d. Since all binary weighted linear threshold functions have width 1, we will write (λ, h)-separable when $k = 2$.

When learning weighted linear threshold functions, we can without loss of generality restrict ourselves to learning positive ones. If we need to learn a weighted linear threshold function with negative weights, we can substitute a positive function of the same height, and perform a minimal amount of pre-processing of the inputs:

Lemma 4.1 *For each representation $(v_1, \ldots, v_n, t)$ of height h there exists a positive representation $(w_1, \ldots, w_n, r)$ of height h and a function $g : \mathbf{Z}_2^n \to \mathbf{Z}_2^n$ of the form $g(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ where for $1 \leq i \leq n$, , either $y_i = x_i$ or $y_i = \overline{x}_i$, such that for all $x \in \mathbf{Z}_2^n$,*

$$f(x) = \Theta_2^n(w_1, \ldots, w_n, r)(g(x)).$$

```
procedure winnow(n, $\alpha$)
  for i := 1 to n do $w_i := 1$;
  repeat
    for each $x = (x_1, \ldots, x_n) \in \mathbf{Z}_2^n$ do  $p := \Theta_2^n(w_1, \ldots, w_n, n)(x)$;
      if $f(x) \neq p$  then winnow_update(x, p, $\alpha$);
      Output $\langle w_1, \ldots, w_n, n \rangle$
  until  $f(x) = \Theta_2^n(w_1, \ldots, w_n, t)(x)$  for all $x \in \mathbf{Z}_2^n$.

procedure winnow_update(x, p, $\alpha$)
  if $f(x) > p$ then $\delta := \alpha$
    else $\delta := 1/\alpha$;
  for i := 1 to n do $w_i := w_i\ \delta^{x_i}$;
```

Figure 6: The Winnow Learning Algorithm.

Proof: We make use of an elementary technique due to Muroga [5] (see also Theorem 4.5.2 of Parberry [10]). Suppose $f(x) = \Theta_2^n(v_1, \ldots, v_n, t)$. Then

$$w_i = |\ v_i\ | \quad \text{for} \quad 1 \leq i \leq n,$$

and

$$r = t + \sum_{i=1}^{n} (|\ v_i\ | - v_i)/2,$$

and

$$y_i = 0.5 + (x_i - 0.5)v_i/|\ v_i\ |.$$

The new representation has height at most h since its denominator is larger than that of the original representation, whilst its numerator is the same. □

The threshold value in a positive representation is not important.

Lemma 4.2 *For each positive representation $(v_1, \ldots, v_n, t)$ of height h and separation λ with $t > 0$, and all $r \in \mathbf{R}^+$, there is a positive representation $(w_1, \ldots, w_n, r)$ of height h and separation λ such that*

$$\Theta_2^n(v_1, \ldots, v_n, t) = \Theta_2^n(w_1, \ldots, w_n, r).$$

Proof: Set $w_i = v_i r/t$ for $1 \leq i \leq n$. □

Littlestone [2,3] proposed a learning algorithm, called the *winnow algorithm* (see Figure 6) for learning n-input binary, positive, (λ, h)-separable functions on a single n-input perceptron. The algorithm takes as parameter a constant α, and learns a positive representation with threshold value n (such a representation exists, by Lemma 4.2). The latency of the binary winnow algorithm is clearly linear in n. The mistake bound is given by the following theorem.

Theorem 4.3 *If $\alpha = 0.5\lambda + 1$, then the number of mistakes made by the winnow algorithm in Figure 6 learning an n-input, positive, (λ, h)-separable weighted linear threshold function on a single n-input perceptron is at most*

$$\left(\frac{14 \log n}{\lambda^2} + \frac{5}{\lambda}\right) h + \frac{8}{\lambda^2}.$$

Proof: See Littlestone [2]. □

Later we will use the fact (Littlestone [3]) that the winnow algorithm will learn any n-input, positive, (λ, h)-separable weighted linear threshold function whose domain is some finite subset of $\{\{0\} \bigcup [\delta, 1]\}^n$. In that case, the mistake bound from Theorem 4.1 depends on $\log(n/\delta)$ instead of $\log n$.

The mistake bound is a significant improvement over the binary perceptron learning algorithm for weighted linear threshold functions with large separation and small height. In contrast, the best known mistake upper bound for the perceptron learning algorithm (see, for example, Duda and Hart [1]) is polynomial in the weight of the best representation (if it is sufficiently large). It is known (see, for example, Muroga [5]; Parberry [10]) that there are weighted linear threshold functions for which the weight of the best representation is at least exponential in n, and it can be deduced that there are functions with exponential weight, polynomial height and inverse-polynomial separation. Whilst the perceptron learning algorithm appears to make exponentially many mistakes for these functions, the winnow learning algorithm makes only polynomially many mistakes. If λ and h are constant, only $O(\log n)$ mistakes are made.

In the light of Lemma 4.1, the definition of (λ, h)-separability can be extended to non-positive weighted linear threshold functions as follows. A representation $(w_1, \ldots, w_n, t)$ has *separation* $\lambda \in \mathbf{R}^+$, $0 \leq \lambda \leq 1$, if for all $x = (x_1, \ldots, x_n) \in \mathbf{Z}_2^n$,

$$f(x) \leq 0 \quad \text{iff} \quad \sum_{j=1}^{n} | w_j | x_j \leq (1 - \lambda)(t + \sum_{j=1}^{n} (| w_j | - w_j)/2).$$

Hence we have:

Corollary 4.4 *Any n-input (λ, h)-separable weighted linear threshold function can be learned on a neural circuit of depth 2 and size at most n with latency $O(n)$ and mistake bound*

$$\left(\frac{14 \log n}{\lambda^2} + \frac{5}{\lambda}\right) h + \frac{8}{\lambda^2}.$$

Proof: The result is an immediate consequence of Lemma 4.1 and Theorem 4.3. □

We extended the winnow algorithm to k-ary weighted multilinear threshold functions. The *k-ary winnow algorithm* for learning n input, k-ary, positive, (λ, h, d) separable function on a single k-ary perceptron with n inputs is described in Figure 7.

The latency of the algorithm is $O(n + k)$. To prove the mistake bound we will use a new slice function which preserves positiveness.

```
procedure multiwinnow_learning($\mathrm{n}, \mathrm{k}, \alpha$)
   for $\mathrm{i} := 1$ to $\mathrm{n}+\mathrm{k}-2$ do $\mathrm{w_i} := 1$;
   $\mathrm{t_{k-1}} = (\mathrm{k}-1)(\mathrm{n}+\mathrm{k}-2)$;
   for $\mathrm{i} := \mathrm{k}-2$ downto 1 do $\mathrm{t_i} = \mathrm{t_{i+1}} - 1$;
   repeat
      for each $\mathrm{x} = (\mathrm{x_1}, \ldots, \mathrm{x_n}) \in \mathbf{Z}_\mathrm{k}^\mathrm{n}$ do  $\mathrm{p} := \Theta_\mathrm{k}^\mathrm{n}(\mathrm{w_1}, \ldots, \mathrm{w_n}, \mathrm{t_1}, \mathrm{t_2}, \ldots, \mathrm{t_{k-1}})(\mathrm{x})$;
         if $\mathrm{f(x)} \neq \mathrm{p}$ then  multiwinnow_update($\mathrm{x}, \mathrm{p}, \alpha^{1/(\mathrm{k}-1)}$);
         Output $\langle \mathrm{w_1}, \ldots, \mathrm{w_n}, \mathrm{t_1}, \mathrm{t_2}, \ldots, \mathrm{t_{k-1}} \rangle$.
   until  $\mathrm{f(x)} = \Theta_\mathrm{k}^\mathrm{n}(\mathrm{w_1}, \ldots, \mathrm{w_n}, \mathrm{t_1}, \ldots, \mathrm{t_{k-1}})(\mathrm{x})$  for all $\mathrm{x} \in \mathbf{Z}_\mathrm{k}^\mathrm{n}$

procedure multiwinnow_update($\mathrm{x}, \mathrm{p}, \alpha$)
   for $\mathrm{i} := 1$ to n do $\mathrm{z_i} := \mathrm{x_i}$;
   for $\mathrm{i} := 1$ to $\mathrm{k}-2$ do $\mathrm{z_{n+i}} := 0$;
   if $\mathrm{f(x)} > \mathrm{p}$ then $\delta := \alpha$;  ind $:= \mathrm{p}+1$
      else $\delta := 1/\alpha$;  ind $:= \mathrm{p}$;
   for $\mathrm{i} :=$ ind to $\mathrm{k}-2$ do $\mathrm{z_{n+i}} := 1$;
   for $\mathrm{i} := 1$ to $\mathrm{n}+\mathrm{k}-2$ do $\mathrm{w_i} := \mathrm{w_i}\delta^{\mathrm{z_i}}$;
   for $\mathrm{i} := \mathrm{k}-2$ downto 1 do $\mathrm{t_i} = \mathrm{t_{i+1}} - \mathrm{w_{n+i}}$;
```

Figure 7: The k-ary Winnow Learning Algorithm.

Definition. If f is an n-input k-ary weighted multilinear threshold function, the *unary slice function* for f is a binary weighted linear threshold function g such that for all $x \in \mathbf{Z}_k^n$ and all $i \in \mathbf{Z}_k$,

$$f(x) \geq i \Leftrightarrow g(x, y(i)) = 1,$$

where $y : \{1, \ldots, k-1\} \to \{0,1\}^{k-1}$ is defined by

$$y(i) = (y_1, \ldots, y_{k-2}) \text{ with } y_j = 1 \text{ iff } j \geq i.$$

Lemma 4.5 *If $(v_1, \ldots, v_n, t_1, \ldots, t_{k-1})$ is a positive representation of height h, width d, and separation λ of a k-ary weighted multilinear threshold function f, then $(w_1, \ldots, w_n, t_2-t_1, t_3-t_2, \ldots, t_{k-1}-t_{k-2}, t_{k-1})$ is a positive representation of height h and separation λ/d of the unary slice function for f.*

Proof: Follows immediately from the definition of the unary slice function. □

Theorem 4.6 *If $\alpha = 1+\lambda/(2d)$, then the number of mistakes made by the k-ary winnow algorithm in Figure 7 learning an n-input, positive, (λ, h, d)-separable k-ary weighted multilinear threshold* *function on a single n-input k-ary perceptron is bounded above by*

$$\left(\frac{14d^2\log((k-1)(n+k-2))}{\lambda^2}+\frac{5d}{\lambda}\right)(k-1)h+\frac{8d^2}{\lambda^2}.$$

Proof: (Sketch) By Lemma 4.5 the learning algorithm, instead of learning a positive representation $(w_1,\ldots,w_n,t_1,t_2,,\ldots,t_{k-1})$ of height h and separation λ for a k-ary weighted multilinear threshold function f , can learn a positive representation $(w_1,\ldots,w_n,t_2-t_1,t_3-t_2,,\ldots,t_{k-1}-t_{k-2},t_{k-1})$ of height h and separation $\lambda^*=\lambda/d$ of the unary slice function for f. Since inputs $(x_1,\ldots,x_n)$ for function f are from $\mathbf{Z}_k^n$, we cannot apply the binary winnow learning algorithm directly to learn the unary slice function for f on inputs $(x,y(i))=(x_1,\ldots,x_n,y_1,\ldots,\ y_{k-1})$. But, we can use the binary winnow learning algorithm with learning parameter α and threshold $t=n+k-2$ to learn a modified unary slice function $(w_1,\ldots,w_n,t_2-t_1,t_3-t_2,,\ldots,t_{k-1}-t_{k-2},t_{k-1}/(k-1))$ on compressed inputs $(x_1/(k-1),\ldots,x_n/(k-1),y_1/(k-1),\ldots,y_{k-2}/(k-1))$ from $\{\{0\}\bigcup[1/(k-1),1]\}^n$. Finally, observe that

$$\alpha^{z_i/k-1}w_i=(\alpha^{1/(k-1)})^{z_i}w_i$$

and also

$$\sum_{i=1}^{n}w_i\frac{x_i}{k-1}+\sum_{i=1}^{k-2}w_{n+i}\frac{y_i}{k-1}<(n+k-2)$$

iff

$$\sum_{i=1}^{n}w_ix_i+\sum_{i=1}^{k-2}w_{n+i}y_i<(k-1)(n+k-2).$$

So, for learning we can actually use binary winnow algorithm with learning parameter $\alpha^{1/(k-1)}$ and threshold $t=(k-1)(n+k-2)$ on the original inputs $(x_1,\ldots,x_n,y_1,\ldots,y_{k-1})$. It is easy to see that this algorithm realizes Figure 7. Substituting $\alpha=1+\lambda^*/2$, $t=n+k-2$ and $\delta=1/(k-1)$ in the Theorem 4.3 we obtain the mistake bound from the claim of this theorem. □

The mistake bound of the k-ary winnow algorithm is a significant improvement over the k-ary perceptron learning algorithm for (λ,h,d)-separable functions when λ is large and h and d are small.

A slightly better mistake bound can be obtained if the input $x\in\mathbf{Z}_k^n$ is encoded in binary as $x^*\in\mathbf{Z}_2^{n\log k}$ and the k-ary winnow algorithm is used on a k-ary perceptron with $n\log k$ binary inputs instead of n k-ary inputs. Instead of learning k-ary weighted multilinear threshold functions, we can substitute *binary-to-k-ary weighted multilinear threshold functions,* which are simply k-ary weighted multilinear threshold functions whose domain is restricted to $\mathbf{Z}_2^n$, and perform a small amount of pre-processing of the inputs. Without loss of generality, we will henceforth assume that k is a power of two. Define function $Encode:\mathbf{Z}_k^n\to\mathbf{Z}_2^{n\log k}$, which simply encodes a k-ary input in binary, by $Encode(x)=(y_1,\ldots,y_{n\log k})$ where $x=(x_1,\ldots,x_n)\in\mathbf{Z}_k^n$ and

$$x_i=\sum_{j=1}^{\log k}2^{j-1}y_{j+(i-1)\log k}.$$

Lemma 4.7 *For every n-input, positive, k-ary weighted multilinear threshold function f of height h and depth d there exists an $(n\log k)$-input binary-to-k-ary weighted multilinear threshold function g of height $(k-1)h$ and width d such that for all $x\in\mathbf{Z}_k^n$, $f(x)=g(Encode(x))$.*

Proof: If $f = \Theta_k^n(w_1, \ldots, w_n, t_1, \ldots, t_{k-1})$, then g is the function $\Theta_k^n(w_{1,1}, \ldots, w_{n,\log k}, t_1, \ldots, t_{k-1})$ with domain restricted to $\mathbf{Z}_2^{n \log k}$, where $w_{i,j} = 2^{j-1} w_i$ for $1 \leq i \leq n$, $i \leq j \leq \log k$. □

Lemma 4.8 *If $(w_1, \ldots, w_{n+k-2}, t)$ is the representation of the unary slice function of a binary-to-k-ary weighted multilinear threshold function f, then $(w_1, \ldots, w_n, t_1, , \ldots, t_{k-1})$ is a representation of f, where*

$$t_i = t - \sum_{j=n+i}^{n+k-2} w_j.$$

Proof: Follows immediately from the definition of the unary slice function. □

Theorem 4.9 *Any n-input, positive, (λ, h, d)-separable binary-to-k-ary weighted multilinear threshold function can be learned on a k-ary perceptron with latency $O(n + k)$ and mistake bound*

$$\left(\frac{14d^2 \log{(n + k - 2)}}{\lambda^2} + \frac{5d}{\lambda} \right) h + \frac{8d^2}{\lambda^2}.$$

Proof: (Sketch) The domain of binary-to-k-ary weighted multilinear threshold function is restricted to $\mathbf{Z}_2^n$. So, the learning algorithm, instead of learning f, can learn the unary slice function for f using the binary winnow learning algorithm. The threshold is chosen equal to $n + k - 2$ by Lemma 4.2. The unary slice function for f is guaranteed to exist by Lemma 4.5. Once the unary slice function has been learned, f can be reconstructed using Lemma 4.8. The mistake bound is given by Theorem 4.3. □

Theorem 4.10 *Any n-input, positive, (λ, h, d)-separable k-ary weighted multilinear threshold function can be learned on a k-ary neural circuit of depth 4 and size $O(nk)$ with latency $O(n \log k + k)$ and mistake bound*

$$\left(\frac{14d^2 \log(n \log k + k - 2)}{\lambda^2} + \frac{5d}{\lambda} \right) (k - 1)h + \frac{8d^2}{\lambda^2}.$$

Proof: (Sketch) For k-ary input $(x_1, \ldots, x_n)$ function $Encode(x_1, \ldots, x_n) = (y_1, \ldots, y_{n \log k})$ can be computed using the subcircuit of depth 3 and size $O(nk)$. Then the k-ary winnow algorithm from Theorem 4.9 can be used to learn $(n \log k)$-input binary-to-k-ary weighted multilinear threshold function on a single $(n \log k)$-input k-ary perceptron. See Figure 8 for detailed construction of the circuit. The i^{th} block $(1 \leq i \leq n)$ in layers $1 - 2$ of the circuit determines whether the i^{th} digit x_i of the input is equal to $0, 1, \ldots,$ or $k - 1$. The test whether the digit x_i is equal to j $(1 \leq j \leq n)$ is easy to realise using 3 gates in 2 layers because $(x_i = j)$ iff $(x_i \geq j \ \& \ -x_i \geq -j)$. The weights of the connections from the second to the third layer are all equal to one. A circle with symbol 'V' inside in the third layer denotes an OR gate (which is an n-input threshold gate with weights 1, first threshold equal to 1 and all other thresholds equal to $n + 1$). The i^{th} block of $\log k$ OR gates in the third layer $(1 \leq i \leq n)$ outputs the binary encoding $y_{1+(i-1)\log k}, \ldots, y_{i \log k}$ of the digit x_i. The weights $w_1, \ldots, w_{n \log k}$ of the connections from the third layer to the k-ary gate in the fourth layer, and the thresholds $t_1, \ldots, t_{k-1}$ of the output gate in the fourth layer can be learned using Theorem 4.9. Since the output k-ary gate has $n \log k$ inputs, latency is $O(n \log k + k)$. The mistake bound easily follows from Lemma 4.7 and Theorem 4.9. □

The definition of (λ, h, d)-separability can be extended to non-positive weighted multilinear threshold functions as follows. A k-ary weighted multilinear threshold function $f : \mathbf{Z}_k^n \to \mathbf{Z}_k$ is (λ, h, d)*-separable* iff its binary encoded equivalent $f^* : \mathbf{Z}_2^{n \log k} \to \mathbf{Z}_k$ is $(\lambda, (k-1)h, d)$-separable, where separation is $\lambda \in \mathbf{R}^+$, $0 < \lambda \leq 1$, if for all $x \in \mathbf{Z}_2^{n \log k}$ and all $i \in \mathbf{Z}_k$, $i < k-1$,

$$f^*(x) \leq i \quad \text{iff} \quad \sum_{j=1}^{n \log k} |w_j| x_j \leq (1-\lambda)(t_{i+1} + \sum_{j=1}^{n \log k} (|w_i| - w_i)/2)).$$

Then we have the extension of the result to the non-positive case:

Corollary 4.11 *Any n-input (λ, h, d)-separable k-ary weighted multilinear threshold function can be learned on a k-ary neural circuit of depth 4 and size $O(nk)$ with latency $O(n \log k + k)$ and mistake bound*

$$\left(\frac{14d^2 \log(n \log k + k - 2)}{\lambda^2} + \frac{5d}{\lambda} \right) (k-1)h + \frac{8d^2}{\lambda^2}.$$

Proof: (Sketch) Suppose f is a (λ, h, d)-separable k-ary weighted multilinear threshold function. Then it has an $(n \log k)$-input $(\lambda, (k-1)h, d)$-separable binary-to-k-ary weighted multilinear threshold function f_1 by Lemma 4.7. By Lemma 4.5, f_1 has an $(n \log k + k - 2)$-input $(\lambda/d, (k-1)h)$-separable unary slice function f_2, which can be replaced by an $(n \log k + k - 2)$-input positive $(\lambda/d, (k-1)h)$-separable unary slice function f_3 by Lemma 4.1. A depth 3, size $O(nk)$ k-ary threshold circuit can compute function Encode as in Theorem 4.10. The winnow algorithm is used to learn a representation for f_3. By Theorem 4.3, the latency is $O(n \log k + k)$ and the mistake bound is

$$\left(\frac{14d^2 \log(n \log k + k - 2)}{\lambda^2} + \frac{5d}{\lambda} \right) (k-1)h + \frac{8d^2}{\lambda^2}.$$

A careful analysis gives the required mistake bound. $\square$

The k-ary winnow algorithm can also be used to learn k-ary weighted multilinear threshold functions on a network of size k and depth 2 by using essentially the same techniques as were used for the perceptron learning algorithm in Section 3. The details are left for the interested reader.

5 Conclusion

The study of k-ary neural networks was justified by the observation that they are closely related to analog neural networks of bounded precision. We have seen two learning results for k-ary neural networks. Firstly, we have demonstrated a k-ary perceptron learning rule with guaranteed convergence. Secondly, Littlestone's winnow algorithm, which learns binary weighted linear threshold functions with a mistake bound dependent on their height has been extended to a k-ary winnow algorithm whose mistake bound depends on the height and width of the k-ary weighted multilinear threshold function being learned.

6 Acknowledgements

The financial support of the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant numbers AFOSR 87-0400 and AFOSR 89-0168 and NSF grant CCR-8801659 to Ian Parberry is gratefully acknowledged.

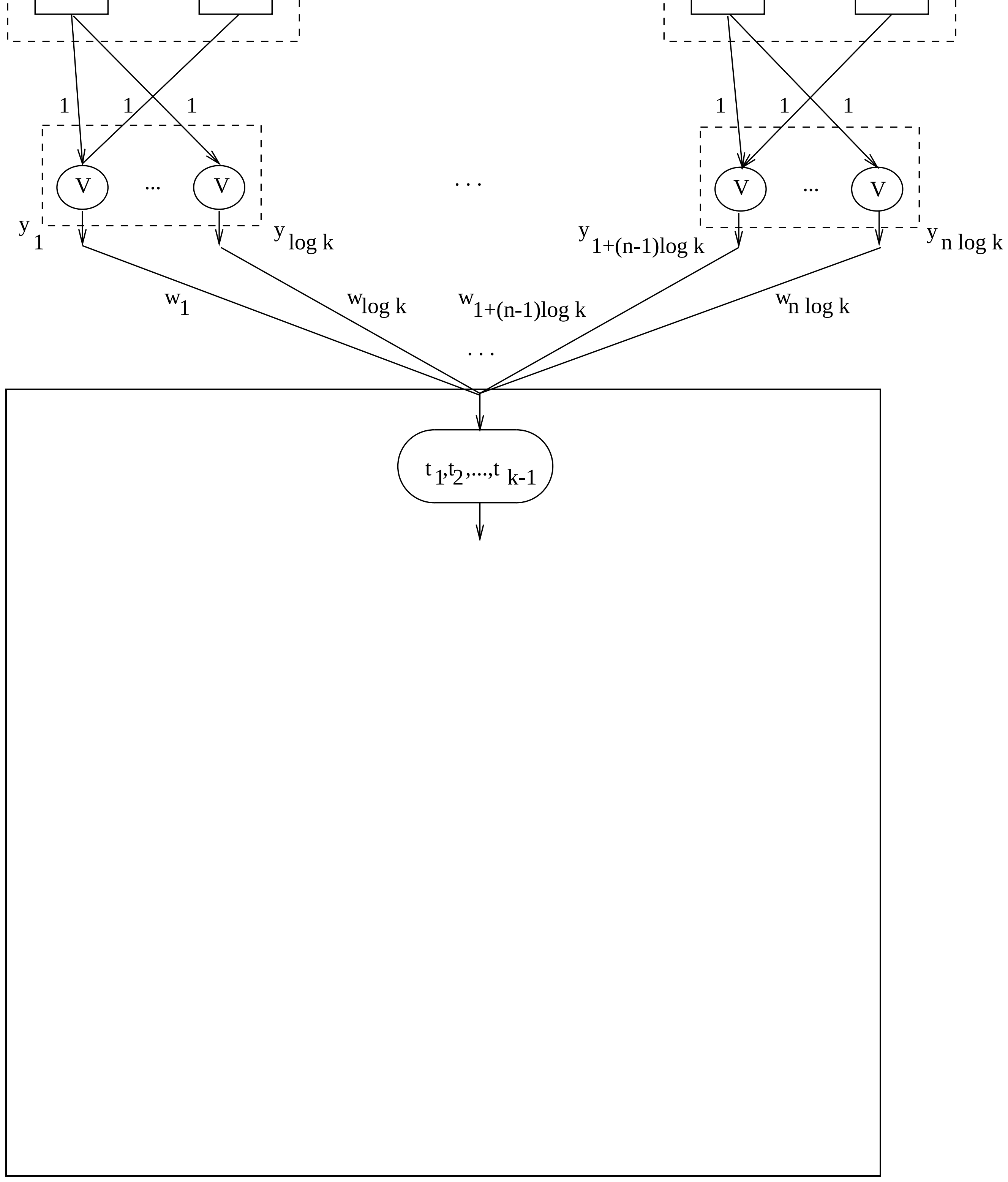

Figure 8: A depth 4, size $O(nk)$ k-ary neural circuit from Theorem 4.10.

7 References

1. Duda, R.O., and Hart, P.E., *Pattern classification and scene analysis,* John Willey, New York, 1973.

2. Littlestone, N., "Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm," *Machine Learning,* vol. 1, no. 2, pp. 285-318, 1987.

3. Littlestone, N., "Mistake bounds and logarithmic linear-threshold learning algorithms," *Technical Report CRL-89-11,* University of California at Santa Cruz, 1989.

4. Minsky, M., and Papert, S., *Perceptrons,* MIT Press, 1969.

5. Muroga, S., *Threshold Logic and its Applications,* Wiley Interscience, New York, 1971.

6. Nilsson, N.J., *Learning Machines,* McGraw-Hill, New York, NY, 1962.

7. Novikoff, A., "On convergence proofs for perceptrons," *Proc. Symposium on Mathematical Theory of Automata,* New York, NY, pp. 615-622, 1962.

8. Obradovic, Z., and Parberry, I., "Analog neural networks of limited precision I: Computing with multilinear threshold functions," in *Advances in Neural Information Processing Systems II,* ed. D.S. Touretzky, Morgan-Kaufmann, pp. 702-709, 1990.

9. Obradovic, Z. and Parberry, I., "Learning with Discrete Multi-Valued Neurons," *Proc. Seventh International Conference on Machine Learning*, Austin, Texas, 1990.

10. Parberry, I., "A Primer on the Complexity Theory of Neural Networks," in *Sourcebook of Formal Methods in Artificial Intelligence,* ed. R. Banerji, North-Holland, pp. 217-268, 1990.