JOURNAL OF COMPUTER AND SYSTEM SCIENCES 46, 198–217 (1993)

Directed Drift: A New Linear Threshold Algorithm for Learning Binary Weights On-Line*

SANTOSH S. VENKATESH†

Department of Electrical Engineering, University of Pennsylvania, Philadelphia, Pennsylvania 19104

Received May 1, 1990; revised January 31, 1991

Learning real weights for a McCulloch–Pitts neuron is equivalent to linear programming and can hence be done in polynomial time. Efficient local learning algorithms such as Perceptron Learning further guarantee convergence in finite time. The problem becomes considerably harder, however, when it is sought to learn *binary* weights; this is equivalent to integer programming which is known to be NP-complete. A family of probabilistic algorithms which learn binary weights for a McCulloch–Pitts neuron with inputs constrained to be binary is proposed here, the target functions being majority functions of a set literals. These algorithms have low computational demands and are essentially local in character. Rapid (average-case) quadratic rates of convergence for the algorithm are predicted analytically and confirmed through computer simulations when the number of examples is within capacity. It is also shown that, for the functions under consideration, Preceptron Learning converges rapidly (but to an, in general, *non-binary* solution weight vector).

1. INTRODUCTION

We consider learning in the context of *linearly separable* functions. Given an arbitrary linearly separable dichotomy of a finite set of patterns, the perceptron training algorithm [1] guarantees convergence in *finite time* of an iteratively updated sequence of weight vectors to a *real* solution vector which separates the dichotomy. The problem becomes considerably harder, however, when we are required to learn *binary* weight for a linearly separable problem. The problem of learning real weights for a McCulloch–Pitts neuron is equivalent to linear programming, for which there exist polynomial time algorithms. (The preceptron learning rule is an on-line procedure which, as we will see in the sequel, can converge extremely rapidly under moderate conditions. Similar conclusions are also reported by Baum [2] under slightly different hypotheses.) Learning binary weights for a

* Presented in part at the "Workshop on Neural Networks for Computing, Snowbird, Utah, April 1989," and the "IEEE International Symposium on Information Theory, San Diego, California, January 1990."

† This research was supported in part by the National Science Foundation under Research Grant EET-8709198 and by the Air Force Office of Scientific Research under Research Grant AFOSR-89-0523.

0022-0000/93 $5.00

McCulloch–Pitts neuron is, however, equivalent to integer programming, which is known to be NP-complete [3].

Notwithstanding the apparent difficulty of the learning problem in this case, the potentially lower cost and simplicity of neural networks comprised of binary interconnections as opposed to real weights makes such circuits rather appealing practically. Recent theoretical results also bolster the usage of such circuits: the computational capacity of a neuron with binary weights remains comparable to that of a neuron with real weights [4. 5]. The development of efficient heuristics for learning binary weights (paralleling the development of such algorithms as back-propagation for neural circuits with real interconnections) is, hence, critical if the cost advantages promised by binary circuits are to realised.

We present here a new family of probabilistic algorithms which learn binary weights for a neuron in an on-line setting. The target functions here are weighted linear threshold functions with weights from $\{-1, 1\}$ which are defined on a domain of binary n-tuples, $\{-1, 1\}^n$. In particular, the target functions are majority functions of a set literals, that is, majority functions that may have any of their inputs complemented. Given a partial Boolean function defined on a subset of m points (patterns) from this class, or alternatively, given a dichotomy of m points in $\{-1, 1\}^n$ wich can be linearly separated with weights from $\{-1, 1\}$, the randomised algorithm described here iteratively adjusts the weights until a solution (binary) weight vector which separates the dichotomy is found. The principal advantage the proposed algorithm has over perceptron learning is that, not only is the solution vector generated by the procedure binary, but the weights remain confined to the domain $\{-1, 1\}$ throughout the entire learning process. The algorithm, as we will see, converges rapidly to a solution when the number of patterns to be dichotomised is within the computational capacity of the neuron. An interesting feature of the algorithm is that it is *local*, which makes it appealing from an implementation perspective.

In the next section we set up the learning protocol and describe the algorithm. We derive some preliminary results on the expected time of first passage of random walks to given boundaries in Section 3. In Section 4 we analyse the algorithm and show quadratic initial rates of convergence when the number of training examples is within the computational capacity of the threshold element. The analysis here is for the average case.[1] We also compare the results obtained with the rate of convergence of the perceptron training algorithm: we show that the perceptron algorithm converges in the worst case with a mistake bound $O(n^2)$ to a real solution vector under the constraint that there exists a binary solution vector within the solution space. We also present an average case analysis of a modification of the perceptron learning algorithm, in the spirit of the proposed directed drift algorithm,

[1] Directed drift is a randomised algorithm, and arbitrarily bad worst-case result are possible. The probability of such occurences is small, however, and is governed by the extreme tails of the underlying probability distribution.

wherein a single weight component is updated at a time. In Section 5 we present simulations and discussions of the algorithm.

Notation. We will use the symbol $\mathbb{B}$ to denote the set $\{-1, 1\}$. If $\mathbf{x} = (x_1, ..., x_n)$ and $\mathbf{y} = (y_1, ..., y_n)$ are points in real Euclidean n-space, we denote by $\langle \mathbf{x}, \mathbf{y} \rangle$ the inner product $\sum_{j=1}^{n} x_j y_j$. Following J. Riordan we use the word *epoch* to denote *points* on the time axis. A physical weight update may take some time, but we will assume updates are timeless and occur at epochs.[2] We define the function $\operatorname{sgn}: \mathbb{R} \to \mathbb{B}$ by $\operatorname{sgn} x = x/|x|$ if $x \neq 0$ and $\operatorname{sgn} 0 = 0$. All logarithms in the exposition are to base e. Finally, if $\{x_n\}$ and $\{y_n\}$ are positive sequences, we denote: $x_n = O(y_n)$ if there is a positive constant K such that $x_n/y_n < K$ for all n; $x_n \sim y_n$ if $x_n/y_n \to 1$ as $n \to \infty$.

2. Learning

2.1. *The setting*

We are given a set of *patterns*, $\mathcal{U} \subset \mathbb{R}^n$, and a function $f: \mathcal{U} \to \mathbb{B}$ which is linearly separable: specifically, there exists a *solution weight vector*, $\mathbf{w}^s \in \mathbb{R}^n$, such that

$$\operatorname{sgn}\{\langle \mathbf{w}^s, \mathbf{u} \rangle\} = f(\mathbf{u}) \tag{1}$$

for every choice of pattern $\mathbf{u} \in \mathcal{U}$. We call the function f the *target function* (also known as the *target concept* in the literature on learning theory). The target functions are, hence, partial Boolean functions corresponding to majority functions of a set of literals. Clearly, f realises a dichotomy of $\mathcal{U}$. Without loss of generality we assume that $f(\mathbf{u}) = 1$ for every patttern $\mathbf{u} \in \mathcal{U}$.[3]

An algorithm for *learning from examples* is a procedure where learning takes place in a sequence of trials. The protocol is as follows:

1. At epoch t the system is characterised by a weight vector, $\mathbf{w}[t]$, and receives an example pattern, $\mathbf{u}[t]$, drawn from $\mathcal{U}$.

2. The system produces a *response*, -1 or 1, according to the sign of $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle$.

3. A new weight vector, $\mathbf{w}[t+1]$, is generated based on the current response, weight vector, $\mathbf{w}[t]$, and example, $\mathbf{u}[t]$.

The procedure is carried out iteratively and is terminated if a solution weight vector is obtained. We call the sequence of examples, $\{\mathbf{u}[t]\}_{t=1}^{\infty}$, the *training sequence*, and the sequence of weight vectors, $\{\mathbf{w}[t]\}_{t=1}^{\infty}$, the *learning sequence*. If the proce-

[2] In his text, W. Feller credits J. Riordan with initiating the usage of the word epoch in such situations [7, p. 73].

[3] If $f(\mathbf{u}) = -1$ then $f(-\mathbf{u}) = 1$ as can be easily seen form (1). Replacing each pattern in $\mathcal{U}$ for which $f(\mathbf{u}) = -1$ by $-\mathbf{u}$ we obtain a corresponding set of patterns $\hat{\mathcal{U}}$; if $\mathbf{w}^s$ is any solution weight vector separating the dichotomy of $\mathcal{U}$ specified by f then all patterns $\hat{\mathcal{U}}$ lie on the same (positive) side of the hyperplane corresponding to $\mathbf{w}^s$, and conversely.

dure terminates in a finite time, we say that the learning algorithm has *learnt* the function f. We will be interested in the *mistake bound*—the number of classification mistakes the learning algorithm makes on the set of examples before it learns the given function. For our purposes, the mistake bound is equal to the number of updates of the weight vector before the function is learnt. We denote the mistake bound by T.

In the sequel, we will further restrict the set of patterns, $\mathcal{U}$, to be drawn from the vertices of the n-cube, $\mathbb{B}^n$, and require that there is a binary solution weight vector, $\mathbf{w}^s \in \mathbb{B}^n$, for f. A fourth restriction that we will require of any binary learning algorithm is:

4. The initial choice of weight vector, $\mathbf{w}[1]$, is arbitrary, subject only to its being chosen from $\mathbb{B}^n$, and the learning algorithm generates binary weight vectors, $\mathbf{w}[t] \in \mathbb{B}^n$, at each epoch of the learning process.

We will, thus, be constrained to looking at algorithms which make only bit changes in the weight vector at each epoch. Specifically, the weights are confined to the domain $\{-1, 1\}$ throughout the learning process. This situation may be compared to perceptron learning, where the weights typically grow in magnitude during the learning process.

2.2. *Directed Drift Algorithms*

We present here a family of probabilistic algorithms for binary learning. We call these algorithms *directed drift algorithms* because, as we shall see, they share some similarities with asymmetric random walks with a preferred direction toward a solution.

Let $\mathcal{U}$ be any subset of patterns from $\mathbb{B}^n$, and let $\{\mathbf{u}[t]\}$ be any training sequence such that each of the patterns in $\mathcal{U}$ appears infinitely often.[4] Let $\{\mathbf{w}[t]\}$ denote a binary learning sequence. For each epoch, t, we denote by $J[t]$ the subset of indices for which the corresponding components of $\mathbf{w}[t]$ and $\mathbf{u}[t]$ are opposite in sign:

$$J[t] = \{j : w_j[t] \neq u_j[t]\}.$$

Single bit updates. We begin with the simplest version of the algorithm where no more than a single component of the weight vector is updated per epoch.

BASE: $\mathbf{w}[1] \in \mathbb{B}^n$ is chosen arbitrarily.

ITERATION: The algorithm's response is predicated upon whether a correct or incorrect response is obtained at the current epoch, t.

- If $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle > 0$, then the weight vector is left unchanged: $\mathbf{w}[t+1] = \mathbf{w}[t]$.

[4] Note that $\mathcal{U} \subset \mathbb{B}^n$ is a finite set of patterns. If $\mathcal{U} = \{\mathbf{u}^1, ..., \mathbf{u}^m\}$ is an m-set of patterns, then we can, for instance, obtain valid training sequences by cycling through the patterns or choosing a pattern randomly at each epoch.

- If $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle \leqslant 0$, the an index $j[t]$ is picked at random from the set of indices, $J[t]$, of mismatched components. The new weight vector is now formed according to the following rule:

$$w_j[t+1] = \begin{cases} w_j[t] & \text{if } j \neq j[t] \\ -w_j[t] & \text{if } j = j[t]. \end{cases} \tag{2}$$

The intuition behind the algorithm is as follows. If a binary solution vector, $\mathbf{w}^s \in \mathbb{B}^n$, exists, then necessarily we must have $\langle \mathbf{w}^s, \mathbf{u} \rangle = \sum_{j=1}^n w_j^s u_j > 0$ for each pattern $\mathbf{u} \in \mathcal{U}$. As there is a contribution of $+1$ to the sum if two corresponding components of $\mathbf{w}^s$ and $\mathbf{u}$ have the same sign, and -1 if the signs are mismatched, it follows that the binary solution vector has more component sign matches than mismatches with *each* pattern in $\mathcal{U}$.

Now the algorithm updates the current estimate of the weight vector if and only if the current pattern from the training sequence is misclassified. A weight vector update results in a randomly chosen mismatched component of the weight vector being flipped to the sign of the corresponding pattern component. Since there is a probability better than a half that a randomly specified component of any pattern has the same sign as the corresponding component of the binary solution vector, it follows that at each epoch the a priori probability that the weight vector update is in the direction of the binary solution vector is better than a half. We will explore this more formally in the sequel.

Several bit updates. The algorithm can be simply extended to accommodate more than a single bit update per epoch. Let $\{N_t\}$ be a sequence of integers with $0 \leqslant N_t \leqslant n/2$.

BASE: $\mathbf{w}[1] \in \mathbb{B}^n$ is chosen arbitrarily.
ITERATION: As before, updates are made only if the current pattern from the training sequence is misclassified.

- If $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle > 0$, then the weight vector is left unchanged: $\mathbf{w}[t+1] = \mathbf{w}[t]$.
- If $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle \leqslant 0$, then N_t indices $j_1[t], \ldots, j_{N_t}[t]$ are picked at random from the set of indices, $J[t]$, of mismatched componets. The new weight vector is now formed according to the following rule:

$$w_j[t+1] = \begin{cases} w_j[t] & \text{if } j \notin \{j_1[t], \ldots, j_{N_t}[t]\} \\ -w_j[t] & \text{if } j \in \{j_1[t], \ldots, j_{N_t}[t]\}. \end{cases}$$

The sequence $\{N_t\}$ specifies the number of bits to be changed at each update epoch, and the proper choice of this sequence is clearly critical to the functioning of the algorithm. This is analogous to choosing an appropriate *cooling schedule* for *simulated annealing* [6].

2.3. *Perceptron Training Algorithm*

A geometrical appreciation of the directed drift algorithm can be obtained from a consideration of the classical perceptron training algorithm. Let $\{\mathbf{u}[t]\}$ be a training sequence of patterns, and let $\{\mathbf{w}[t]\}$ denote a learning sequence of *real* weight vectors.

Fixed increment Perceptron training. This is the simplest form of perceptron learning. Let $\beta > 0$ be fixed.

BASE: The initial choice of weight vector is arbitrary. For simplicity we take $\mathbf{w}[1] = \mathbf{0}$.

ITERATION: As before, weight vector updates are made only if a pattern is misclassified.

- If $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle > 0$, then the weight vector is left unchanged: $\mathbf{w}[t+1] = \mathbf{w}[t]$.
- If $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle \leqslant 0$, then set $\mathbf{w}[t+1] = \mathbf{w}[t] + \beta \mathbf{u}[t]$.

The perceptron training algorithm is known to converge to a real solution vector (if it exists) in finite time [1]. Geometrically speaking, the situation is as depicted in Fig. 1a. When a pattern from the training sequence is misclassified by the current estimate of the weight vector, the weight vector update is in the direction of the misclassified pattern. This idea of updating in the direction of the misclassified pattern is extended in the directed drift algorithms. The situation is as depicted schematically in Fig. 1b. The updates, being constrained to be binary, are not directly in the direction of the misclassified pattern; nevertheless, the update lies in the positive half space corresponding to the binary pattern vector so that the updated weight vector is more apt to classify the pattern correctly.

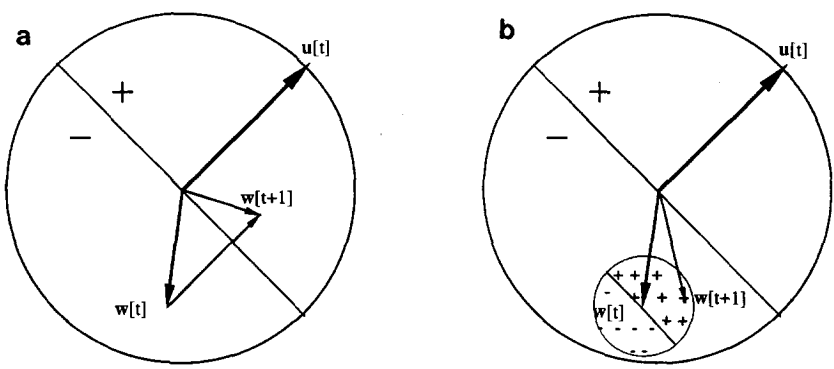

FIG. 1. (a) A shematic depiction of the incremental change in the current weight vector, $\mathbf{w}[t]$, made by the perceptron training rule. The change is in the direction of the misclassified pattern, $\mathbf{u}[t]$, and the resultant weight vector, $\mathbf{w}[t+1]$, is more apt to classify the pattern correctly. (b) The corresponding scenario for directed drift. Here the algorithm is constrained to make only single bit changes in the current (binary) weight vector. Bit changes in the positive hemisphere (in the direction of the misclassified pattern) improve the possibility of correct classification.

Single component perceptron training. The basic randomisation idea behind single bit update directed drift is easily extended to single component perceptron learning, where a single component of the weight vector is modified at each update epoch (as opposed to traditional perceptron learning where all components are modified at each update epoch).

For each epoch, t, let $I[t]$ denote the subset of indices for which the corresponding components of $\mathbf{w}[t]$ and $\mathbf{u}[t]$ are opposite in sign:

$$I(t) = \{i: u_i[t] \neq \operatorname{sgn} w_i[t]\}.$$

BASE: For simplicity, take $\mathbf{w}[1] = \mathbf{0}$.

ITERATION: The algorithm's response is predicated, as usual, upon whether a correct or incorrect reponse is obtained at the current epoch, t.

- If $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle > 0$, then the weight vector is left unchanged: $\mathbf{w}[t+1] = \mathbf{w}[t]$.
- If $\langle \mathbf{w}[t], \mathbf{u}[t] \rangle \leqslant 0$, then an index $i[t]$ is picked at random from the set of indices, $I[t]$, of mismatched components. The new weight vector is now formed according to the following rule:

$$w_i[t+1] = \begin{cases} w_i[t] & \text{if } i \neq i[t] \\ w_i[t] + u_i[t] & \text{if } i = [t]. \end{cases} \tag{3}$$

In this variation, a single bit is added (subtracted) from a randomly chosen component at each update epoch. Note that if a weighting factor β is not introduced, this constrains weights to $\{-1, 0, 1\}$. The mistake bound hence coincides with the number of component updates, as in single bit update directed drift. For fixed increment perceptron learning, of course, the number of component updates is n times the mistake bound.

3. Random Walks

An estimate of the rates of convergence of the randomised algorithms described above may be obtained by appealing to notions from random walks and the geometric theory of pahts.

Let $\{X_j\}$ be a sequence of Bernoulli trials with success probability $p_n = 1/2 + \beta_n$, $0 < \beta_n \leqslant 1/2$, depending on a parameter n:

$$X_j = \begin{cases} 1 & \text{with probability } p_n = 1/2 + \beta_n \\ -1 & \text{with probability } q_n = 1/2 - \beta_n. \end{cases}$$

Let $S_k = \sum_{j=1}^{k} X_j$ denote a random walk with positive drift, $\mathbf{E}S_k = 2k\beta_n$. We are interested in estimating the expected time of first passage of the random walk to some specified boundary, $B(n, k)$.

3.1. *Fixed Boundary*

Let us first consider the case of a fixed (one-sided) boundary at n. Define

$$T_1(n) = \inf\{k: S_k \geq n\}.$$

For this case, the theory of generating functions can be readily invoked to estimate the expected time of first passage to the boundary (cf. Feller [7, Chap. III]). A direct estimate using Wald's equation is also possible.) We have the following estimate:

PROPOSITION 3.1. $\mathbf{E}T_1(n) = n/2\beta_n$ *for every* n.

Proof. Let $\alpha_l(k)$ denote the probability that the random walk makes a first passage l units to the right of the starting point in k steps. We then have $\mathbf{E}T_1(n) = \sum_{k=0}^{\infty} k\alpha_n(k)$.

As a first passage through n must necessarily involve a prior first passage through $n-1$, we immediately have the convolutional relation

$$\alpha_n(k) = \sum_{j=1}^{k-1} \alpha_{n-1}(j)\,\alpha_1(k-j). \tag{4}$$

Let $A_l(s)$ denote the generating function for the probability distribution $\alpha_l(k)$; i.e.,

$$A_l(s) = \sum_{k=0}^{\infty} \alpha_l(k)\, s^k.$$

From Eq. (4) we hence have

$$A_n(s) = A_{n-1}(s)\, A_1(s) = [A_1(s)]^n, \tag{5}$$

with the latter equality following by induction.

To evaluate $A_1(s)$ we need to evaluate the probabilities, $\alpha_1(k)$, of a first transition one unit to the right in k steps. We note that

$$\alpha_1(0) = 0 \tag{6}$$

$$\alpha_1(1) = p_n. \tag{7}$$

Now, a first transition one unit to the right in $k \geq 2$ steps must necessarily involve an initial step to the left, followed by a first transition one unit to the right (back to the origin), and a final first transition one more unit to the right. Hence

$$\alpha_1(k) = q_n \sum_{j=1}^{k-2} \alpha_1(j)\,\alpha_1(k-j-1), \qquad k = 2, 3, \ldots. \tag{8}$$

Using Eqs. (6), (7), and (8) we now have

$$\begin{aligned} A_1(s) &= \sum_{k=0}^{\infty} \alpha_1(k)\, s^k \\ &= p_n s + \sum_{k=2}^{\infty} \alpha_1(k)\, s^k \\ &= p_n s + q_n \sum_{k=2}^{\infty} \sum_{j=1}^{k-2} \alpha_1(j)\, \alpha_1(k-j-1)\, s^k \\ &= p_n s + q_n s [A_1(s)]^2. \end{aligned}$$

Solving for $A_1(s)$ we finally obtain[5]

$$A_1(s) = \frac{1 - \sqrt{1 - 4p_n q_n s^2}}{2q_n s}.$$

Substituting in (5) we obtain

$$A_n(s) = \sum_{k=0}^{\infty} \alpha_n(k)\, s^k = \left[\frac{1 - \sqrt{1 - 4p_n q_n s^2}}{2q_n s}\right]^n.$$

We can now directly compute the expected time of first pasage n units to the right by

$$\mathbf{E}T_1(n) = A_n'(1) = \frac{n}{p_n - q_n}.$$

The substitution $p_n - q_n = 2\beta_n$ completes the proof. ∎

3.2. *Receding Boundary*

We now consider the case of a receding (two-sided) boundary at $\pm\sqrt{kn}$. Define

$$T_2(\sqrt{kn}) = \inf\{k : |S_k| \geq \sqrt{kn}\}.$$

It is difficult to get explicit closed-form expressions when the boundary is not fixed, and we will be satisfied with an asumptotic estimate for $\mathbf{E}T_2(\sqrt{kn})$ as $n \to \infty$. Our development follows Siegmund [8, Chap. IX], where a general analysis is presented in the context of nonlinear renewal theory.

Now $T_2(\sqrt{kn})$ (if finite) is the first integer k for which the random variable S_k^2/k exceeds n. Now, simple algebraic manipulations yield

$$\frac{S_k^2}{k} = \underbrace{4k\beta_n^2 + 4\beta_n(S_k - 2k\beta_n)}_{A_k} + \underbrace{\frac{1}{k}(S_k - 2k\beta_n)^2}_{B_k}.$$

[5] We discard the positive root as it grows without bound as $s \to 0$, and we require $A_1(0) = 0$.

By the strong law of large numbers we have

$$\begin{aligned} \frac{A_k}{k} &\to 4\beta_n^2 \qquad \text{with probability one} \\ \frac{B_k}{k} &\to 0 \qquad \text{with probability one.} \end{aligned} \tag{9}$$

Further, as an easy consequence of Kolmogorov's inequality, we have

$$\frac{1}{k} \max_{1 \leqslant j \leqslant k} B_j \to 0 \qquad \text{in probability.} \tag{10}$$

PROPOSITION 3.2. *The following assertions hold*:

(a) $\mathbf{P}\{T_2(\sqrt{kn}) < \infty\} = 1$ *for all* n;

(b) $T_2(\sqrt{kn})/n/4\beta_n^2 \to 1$ *in probability as* $n \to \infty$.

Proof. The observation (9) yields $S_k^2/k^2 \to 4\beta_n^2$ with probability one, so that part (a) of the proposition follows.

Now let $K_n = n/4\beta_n^2$. Fix $0 < \varepsilon < 1$ and set $K_n' = K_n(1+\varepsilon)$. Form (9) and (10) we have

$$\frac{1}{k} \max_{1 \leqslant j \leqslant k} \frac{S_j^2}{j} \to 4\beta_n^2 \qquad \text{in probability.}$$

Hence, as $n \to \infty$, we have

$$\begin{aligned} \mathbf{P}\left\{T_2(\sqrt{kn}) > \frac{n}{4\beta_n^2}(1+\varepsilon)\right\} &= \mathbf{P}\left\{\max_{1 \leqslant j \leqslant K_n'} \frac{S_j^2}{j} < n\right\} \\ &= \mathbf{P}\left\{\frac{1}{4\beta_n^2 K_n'} \max_{1 \leqslant j \leqslant K_n'} \frac{S_j^2}{j} < \frac{1}{1+\varepsilon}\right\} \to 0. \end{aligned}$$

A similar argument shows that $\mathbf{P}\{T_2(\sqrt{kn}) < (n/4\beta_n^2)(1-\varepsilon)\} \to 0$. ∎

PROPOSITION 3.3. $\mathbf{E}T_2(\sqrt{kn}) \sim n/4\beta_n^2$ *as* $n \to \infty$.

Proof. Let $K_n = n/4\beta_n^2$, as before. For $k \geqslant 2K_n$ we have

$$\begin{aligned} \mathbf{P}\{T_2(\sqrt{kn}) > k\} &\leqslant \mathbf{P}\left\{\frac{S_k^2}{k} < n\right\} \leqslant \mathbf{P}\{A_k < n\} \\ &= \mathbf{P}\left\{\sum_{j=1}^{k} (X_j - 2\beta_n) < -k\beta_n + \frac{n}{4\beta_n}\right\} \\ &\leqslant \mathbf{P}\left\{\sum_{j=1}^{k} (X_j - 2\beta_n) < -\frac{k\beta_n}{2}\right\}. \end{aligned}$$

The bound above is just the probability of an event in the left tail of the binomial distribution. An application of Chernoff's bound [9] yields

$$\mathbf{P}\{T_2(\sqrt{kn}) > k\} \leqslant e^{-kC_n},$$

where

$$C_n = -\left(\frac{1}{2}+\frac{3\beta_n}{4}\right)\log\left(1+\frac{\beta_n}{2+3\beta_n}\right)-\left(\frac{1}{2}-\frac{3\beta_n}{4}\right)\log\left(1-\frac{\beta_n}{2-3\beta_n}\right).$$

We now claim that

$$\sum_{k \geqslant 2K_n} \mathbf{P}\{T_2(\sqrt{kn}) > k\} \to 0 \qquad (n \to \infty). \tag{11}$$

If β_n is bounded away from zero this is clear: $C_n > D > 0$ for some absolute positive constant D and the sum in (11) is just a sum over the exponential tail (note that $K_n \to \infty$ as $n \to \infty$). Now consider the case where $\beta_n \to 0$. Using the Taylor series approximation

$$\log(1+x) = x - x^2/2 + O(x^3) \qquad (|x| \to 0),$$

it is easy to see that

$$\mathbf{P}\{T_2(\sqrt{kn}) > k\} \leqslant \exp\left[-\frac{k\beta_n^2}{8}\{1+O(\beta_n)\}\right].$$

It is now readily verified that the first term in the series in (11) decreases exponentially fast with n. This completes the proof of the claim.

The elementary observation $\mathbf{E}(T_2(\sqrt{kn}) \mid T_2(\sqrt{kn}) > 4K_n) \geqslant 4K_n$ together with the clain now yields

$$\begin{aligned}\mathbf{E}(T_2(\sqrt{kn}); T_2(\sqrt{kn}) > 4K_n) &\leqslant 2\mathbf{E}(T_2(\sqrt{kn}) - 2K_n; T_2(\sqrt{kn}) > 4K_n)\\ &\leqslant 2\mathbf{E}(T_2(\sqrt{kn}) - 2K_n; T_2(\sqrt{kn}) > 2K_n)\\ &\leqslant 2\sum_{k \geqslant 2K_n} \mathbf{P}\{T_2(\sqrt{kn}) > k\} \to 0 \qquad (n \to \infty).\end{aligned}$$

Hence, the random variables $(n/4\beta_n^2)^{-1}\, T_2(\sqrt{kn})$, $n \geqslant 1$, are uniformly integrable. In addition, by Proposition 3.2(b), $(n/4\beta_n^2)^{-1}\, T_2(\sqrt{kn}) \to 1$ in probability. The result follows. ∎

4. Analysis

4.1. *Directed Drift*

We consider single bit updates for simplicity. Assume that we have a finite set of patterns, $\mathscr{U} = \{\mathbf{u}^1, \ldots, \mathbf{u}^m\} \subset \mathbb{B}^n$, chosen independently with pattern components

drawn from a sequence of symmetric Bernoulli trials. Let $\{\mathbf{u}[t]\}$ be a training sequence, and $\{\mathbf{w}[t]\}$ the learning sequence specified by the rule (2). Let $\{t_k\}$ denote the subsequence of epochs at which patterns from the training sequence are misclassified; i.e.,

$$\langle \mathbf{w}[t_k], \mathbf{u}[t_k] \rangle \leqslant 0, \qquad k = 1, 2, \ldots.$$

We can write the weight vector updates of Eq. (2) in the form

$$\mathbf{w}[t_{k+1}] = \mathbf{w}[t_k] + \mathbf{v}[t_k],$$

where $\mathbf{v}[t_k]$ is a vector whose components satisfy

$$v_j[t_k] = \begin{cases} 0 & \text{if } j \neq j[t_k] \\ -2w_j[t_k] & \text{if } j = j[t_k]. \end{cases} \tag{12}$$

Assume that there is a binary solution vector, $\mathbf{w}^s \in \mathbb{B}^n$. Consider the estimate errors

$$\begin{aligned} \mathscr{E}_{k+1} &\triangleq \|\mathbf{w}[t_{k+1}] - \mathbf{w}^s\|^2 \\ &= \|\mathbf{w}[t_k] + \mathbf{v}[t_k] - \mathbf{w}^s\|^2 \\ &= \|\mathbf{w}[t_k] - \mathbf{w}^s\|^2 + \|\mathbf{v}[t_k]\|^2 + 2\langle \mathbf{w}[t_k] - \mathbf{w}^s, \mathbf{v}[t_k] \rangle \\ &= \mathscr{E}_k + 4 + 2\langle \mathbf{w}[t_k], \mathbf{v}[t_k] \rangle - 2\langle \mathbf{w}^s, \mathbf{v}[t_k] \rangle. \end{aligned}$$

Using (2) and (12) we hence obtain

$$\mathscr{E}_{k+1} = \mathscr{E}_k - 4w^s_{j[t_k]} u_{j[t_k]}[t_k].$$

Define the ± 1 random variables

$$X_i = w^s_{j[t_i]} u_{j[t_i]}[t_i].$$

By induction we obtain

$$\mathscr{E}_{k+1} = \mathscr{E}_1 - 4 \sum_{i=1}^{k} X_i.$$

Upper bounding $\mathscr{E}_1$ by $4n$, and setting $S_k = \sum_{i=1}^{k} X_i$ we finally obtain

$$0 \leqslant \mathscr{E}_{k+1} \leqslant 4(n - S_k).$$

The procedure terminates at the value of k for which the random sum S_k first exceeds n. The mistake bound, T, hence satisfies $S_T \geqslant n$, and $S_k < n$, for $k = 1, \ldots, T-1$. The mistake bound is infinite if there exists no such value of k, or if there exists no binary solution vector for the choice of patterns, $\mathscr{U}$.

The above is reminiscent of a random walk with a fixed boundary at n. In fact, if the m-set of pattern $\mathscr{U}$ is chosen independently, then the random variables X_i

corresponding to different patterns must be independent.[6] Let us assume that the training sequence is obtained by cycling through the patterns. For a random initial choice of weight vector a substantial number of patterns will be misclassified, so that a pattern will recur in the update sequence only after $\Omega(m)$ epochs. Through the inital progress of the algorithm, hence, the random sum $S_k = \sum_{i=1}^{k} X_i$ is a sum of independent, identically distributed, ± 1 random variables with

$$X_i = \begin{cases} 1 & \text{with probability } p_n = \frac{1}{2} + \beta_n \\ -1 & \text{with probability } q_n = \frac{1}{2} - \beta_n \end{cases}$$

for some $\beta_n \in (0, \frac{1}{2}]$. The specific value of the probability p_n—the relative frequency of the number of component matches between a binary solution vector and a pattern—depends on the size, m, of the set of patterns, $\mathcal{U}$. We clearly have

$$p_n \geq \frac{n/2+1}{n} = \frac{1}{2} + \frac{1}{n},$$

so that $\beta_n \geq 1/n$.

The above argument indicates that the expected time of first passage to n of a one-dimensional random walk, S_k, with positive drift, $2k\beta_n$, may be used as an estimate of the expected mistake bound for single bit update directed drift. Substituting $\beta_n \geq 1/n$ in Proposition 3.1 we then obtain the estimate $O(n^2)$ for the expected mistake bound when the number of patterns is within capacity.

4.2. *Perceptron Training*

It is instructive to compare the above convergence rates with rates that obtain for perceptron training. The classical proofs of the perceptron training procedure only guarantee that the procedure converges in finite time if a solution exists: convergence time, however, is strongly dependent on the distribution of (real) patterns, and in the worst case can be exponential in the number of bits needed to specify the pattern distribution. When constraints are placed on the allowable choices of patterns, however, convergence can be much more rapid. To compare mistake bounds with directed drift, let us consider perceptron training when the patterns are binary and under the condition that there exists a binary solution vector. (Note, however, that we only require that the perceptron training algorithm return a *real* solution vector.) We show first that the fixed increment perceptron training algorithm converges in the worst case with a mistake bound which increases no faster than quadratically in n; alternatively, the total number of component updates before convergence is $O(n^3)$. We follow this with an average case analysis, similar

[6] To facilitate ease of analysis for the nonce we assume that the number of patterns is within the *computational capacity* of a linear threshold element with binary weights. (The capacity is quite large—linear in n—and capacities of the order of $n/\log n$ can be easily achieved for rather simple algorithms [4, 5].) For a random choice of patterns we are then assured with arbitrarily high probability asymptotically that there exists a binary solution vector.

in flavour to the analysis for directed drift, for randomised, single component update perceptron training, which yields similar results.

Let $\mathscr{U} = \{\mathbf{u}^1, ..., \mathbf{u}^m\} \subset \mathbb{B}^n$ be a finite set of patterns, $\{\mathbf{u}[t]\}$ the training sequence, and $\{\mathbf{w}[t]\}$ the learnig sequence. As before, let $\{t_k\}$ denote the subsequence of epochs at which patterns from the training sequence are misclassified.

Fixed increment training. The weight vector updates result in

$$\mathbf{w}[t_{k+1}] = \mathbf{w}[t_k] + \beta \mathbf{u}[t_k], \qquad k = 1, 2, ...$$

Assume that there is a binary solution vector, $\mathbf{w}^s \in \mathbb{B}^n$. For a choice of parameter $c > 0$ to be specified shortly, consider now the estimate errors

$$\begin{aligned}\mathscr{F}_{k+1} &\triangleq \|\mathbf{w}[t_{k+1}] - c\mathbf{w}^s\|^2 \\ &= \|\mathbf{w}[t_k] + \beta \mathbf{u}[t_k] - c\mathbf{w}^s\|^2 \\ &= \mathscr{F}_k + \beta^2 \|\mathbf{u}[t_k]\|^2 + 2\beta \langle \mathbf{w}[t_k] - c\mathbf{w}^s, \mathbf{u}[t_k] \rangle.\end{aligned}$$

Since $\mathbf{w}^s$ is a binary solution vector we must have that $\langle \mathbf{w}^s, \mathbf{u}[t] \rangle \geqslant 1$ for each pattern in the training sequence. Furthermore, $\|\mathbf{u}[t]\|^2 = n$ for each pattern in the training sequence, and as $\mathbf{w}[t_k]$ misclassifies pattern $\mathbf{u}[t_k]$ by definition, we also have $\langle \mathbf{w}[t_k], \mathbf{u}[t_k] \rangle \leqslant 0$. Hence

$$0 \leqslant \mathscr{F}_{k+1} \leqslant \mathscr{F}_k + \beta^2 n - 2c\beta.$$

With the assumption that $\mathbf{w}[1] = \mathbf{0}$ we have $\mathscr{F}_1 = \|\mathbf{w}[1] - c\mathbf{w}^s\|^2 = c^2 n$. By induction on the above inequality we then obtain

$$0 \leqslant \mathscr{F}_{k+1} \leqslant c^2 n - \beta(2c - \beta n)\, k.$$

The procedure terminates with a mistake bound

$$T \leqslant \frac{c^2 n}{\beta(2c - \beta n)}.$$

Choosing $c = \beta n$ minimises this upper bound for the mistake bound. With this choice of c we obtain that the mistake bound for the fixed increment perceptron training rule is $T \leqslant n^2$ under the constraints of a binary pattern space and with the requirement that there exist a binary solution vector. Note that this bound is independent of the number of binary patterns and their distribution. The sole requirement for this estimate of convergence time for the perceptron training rule to hold is that there exist a vertex of the n-cube (a binary solution vector) within the convex polyhedral cone defined by the space of real solution vectors. While the mistake bound gives the number of weight updates before convergence, it must also be noted that in the perceptron training rule each update is a synchronous update in which each of the n components of the weight vector is modified (as opposed to

the single bit modifications in the simpler version of the directed drift algorithm) and each component modification requires the addition of a real scalar. Thus, in the worst case, the procedure terminates after no more than n^3 component updates.

Single components updates. The weight vector updates of Eq. (3) can be written in the form

$$\mathbf{w}[t_{k+1}] = \mathbf{w}[t_k] + \mathbf{x}[t_k],$$

where $\mathbf{x}[t_k]$ denotes the vector with components

$$x_i[t_k] = \begin{cases} 0 & \text{if } i \neq i[t_k] \\ u_i[t_k] & \text{if } i = i[t_k]. \end{cases}$$

Let $\mathbf{w}^s \in \mathbb{B}^n$ be a binary solution vector. We have the following bounds on the length-square of the weight vector estimates:

$$\begin{aligned} \mathscr{L}_{k+1}^2 \triangleq \|\mathbf{w}[t_{k+1}]\|^2 &= \|\mathbf{w}[t_k] + \mathbf{x}[t_k]\|^2 \\ &= \mathscr{L}_k^2 + \|\mathbf{x}[t_k]\|^2 + 2w_{i[t_k]}[t_k]\, u_{i[t_k]}[t_k] \\ &\leqslant \mathscr{L}_k^2 + 1. \end{aligned}$$

We hence have $\mathscr{L}_{k+1}^2 \leqslant k$ as a consequence of the choice $\mathbf{w}_1 = 0$. Also, as a consequence of the Cauchy–Schwarz inequality, we have

$$\begin{aligned} \mathscr{L}_{k+1}^2 &\geqslant \frac{|\langle \mathbf{w}[t_{k+1}], \mathbf{w}^s \rangle|^2}{\|\mathbf{w}^s\|^2} = \frac{1}{n} |\langle \mathbf{w}[t_k], \mathbf{w}^s \rangle + \langle \mathbf{w}^s, \mathbf{x}[t_k] \rangle|^2 \\ &= \frac{1}{n} \left| \sum_{h=1}^{k} \langle \mathbf{w}^s, \mathbf{x}[t_h] \rangle \right|^2 = \frac{1}{n} \left| \sum_{h=1}^{k} w^s_{i[t_h]} u_{i[t_h]}[t_h] \right|^2. \end{aligned}$$

Define the ± 1 random variables

$$X_h = w^s_{i[t_h]} u_{i[t_h]}[t_h],$$

and let $S_k = \sum_{h=1}^{k} X_h$. We then have

$$\frac{|S_k|}{\sqrt{n}} \leqslant \mathscr{L}_{k+1} \leqslant \sqrt{k}.$$

The algorithm terminates at the first instant k for which $|S_k|$ exceeds $\sqrt{kn}$. Arguing as for directed drift, we use as an estimate for the expected mistake bound the expected time of first passage of a random walk with positive drift $2k\beta_n \geqslant 2k/n$ to the two-sided boundary at $\pm\sqrt{kn}$: Proposition 3.3 then yields the asymptotic estimate $O(n^3)$ for the expected mistake bound as $n \to \infty$.

5. Simulations

Computer simulations indicate that the rapid convergence times predicted by analysis hold when the number of patterns to be loaded lies within the capacity. Mistake bounds are plotted as a function of n in Fig. 2. In each plot mistake bounds for each choice of m and n were averaged over 1000 runs of the single bit update directed drift algorithm. In each run of the algorithm an indpendent set of patterns was drawn from a standard pseudo-random binomial number generator. (To ensure the existence of a binary solution weight vector, a binary n-tuple was selected at random as the solution vector, and those patterns lying in the negative half space of the solution vector were reflected.) A random initial binary weight vector was selected as the initial estimate of the weight vector presented to the single bit update directed drift algorithm with the training sequence obtained by cyclically presenting the patterns. At convergence the number of adaptations of the weight vector were stored as the estimate of the mistake bound. The expected mistake bound for the choice of m and n was estimated by averaging the number of adaptations before convergence over 1000 independent runs (each on an independently chosen data set).

In Figs. 2a and b the number of patterns, m, was fixed within capacity (at $m = n/4$ and $m = n/2$, respectively) and very rapid convergence times are seen. Expected mistake bounds increase significantly around capacity as illustrated in Fig. 2c with $m = n$. Mistake bounds finally saturate above capacity at around $2n$ (plotted in Fig. 2d). When the number of patterns is well above capacity it is not clear whether the algorithm still converges polynomially fast, and this is under investigation. Convergence is still, however, several orders of magnitude faster than an exhaustive search through the vertices of the n-cube. (For instance, when $n = 20$ and $m = 40$, exhaustive search requires of the order of 10^6 steps while directed drift converges in about 10^4 steps.) Early results indicate that order of magnitude improvements in mistake bound may be obtained by updating several bits at a time with an appropriate choice of cooling schedule in the algorithm. We do not have good heuristics at this time, however, for choice of good cooling schedules. Batch learning versions also result in enormous improvements in running time (Fang and Venkatesh [10]).

Figures 3 and 4 show a plot of the expected mistake bound, T, for single bit update directed drift as the number of patterns, m, increases for a fixed value of n. We observe that the expected mistake bound saturates to a fixed value depending on n when the number of patterns exceeds approximately $3n$ in the range of n we considered in our simulations. Note also the rather abrupt threshold behaviour when the number of patterns exceeds the capacity (n). We conjecture that there is a threshold function for the expected mistake bound around the capacity.

The saturation of the mistake bound when the number of patterns exceeds capacity is caused essentially by the shrinkage in the solution space—when the number of patterns exceeds roughly $3n$, then the binary solution vector, if one exists, is essentialy unique. We illustrate this in Fig. 4: for a fixed value of n we plot

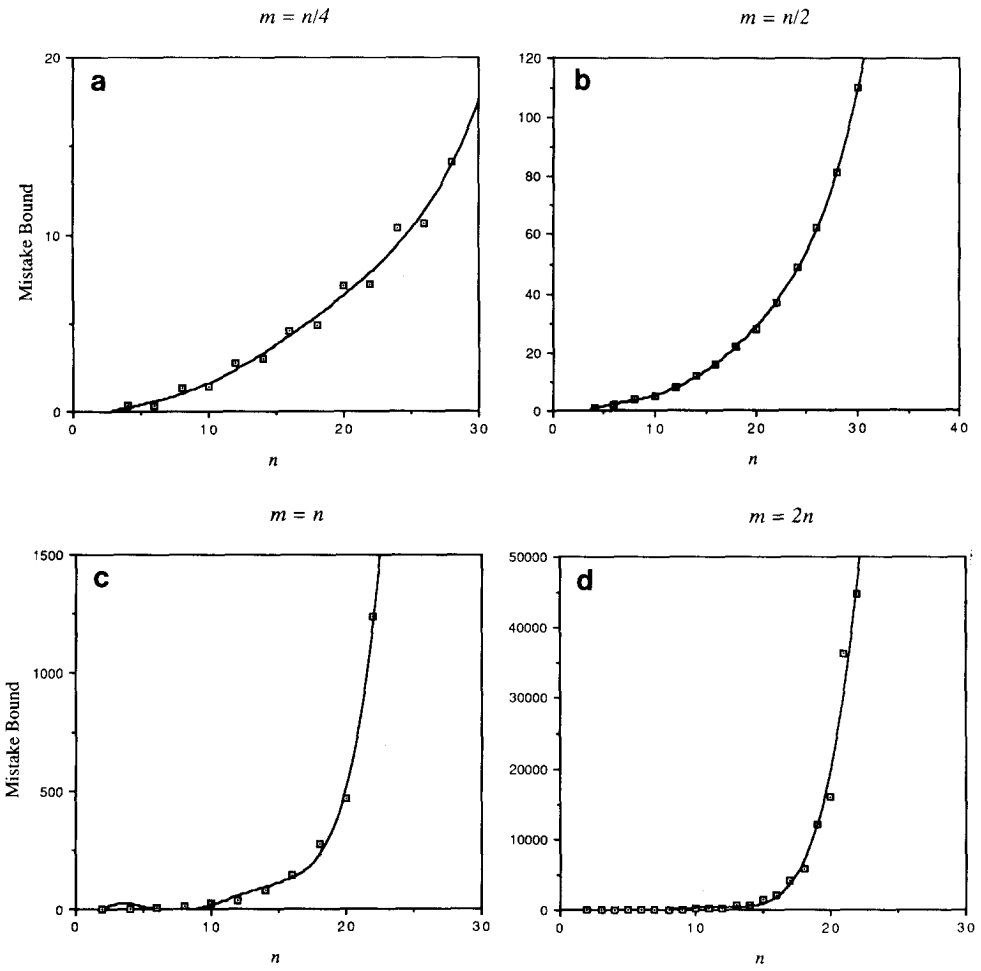

FIG. 2. Plots of the expected mistake bound (averaged over 1000 trials) of the directed drift algorithm as a function of n for four cases with the number of patterns, m, chosen equal to $n/4$, $n/2$, n, and $2n$. The first two cases are within capacity, and the curves reflect best fit quadratics. For $m = n$ and $m = 2n$ the number of patterns exceed capacity, and the curves fitted are polynomials of degree 4 and 5, respectively. The best fit exponential curve, $2^{\alpha n}$, above capacity has an exponent of roughly $0.75n$ in the range considered.

simultaneously the expected mistake bound and the relative frequency with which the algorithm terminates in an initially chosen binary solution vector. The saturation in mistake bound around $3n$ is again evident, as well as the threshold behaviour around capacity. Note that the probability that there are multiple solution vectors in the dual of the mistake bound curve: while for a small number of patterns there exist many binary solution vectors, a precipitous drop in the probability of multiple solutions is evidenced around the capacity and, finally, around the $3n$ there exists only one solution vector with high probability.

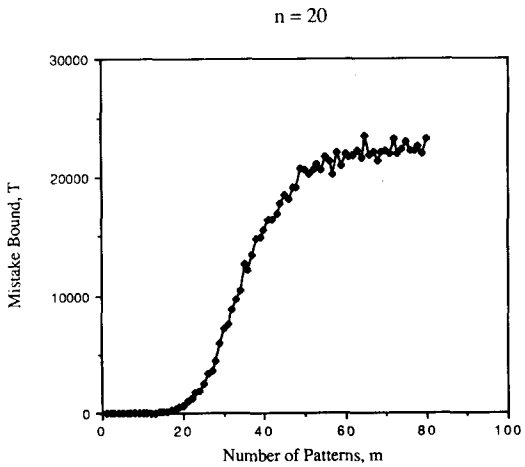

FIG. 3. The expected mistake bound (averaged over 1000 independent trials) of directed drift is plotted against the number of patterns for $n = 20$. A threshold phenomenon is observed around capacity when the mistake bound rises abruptly. The mistake bound saturates to a fixed value when the number of patterns exceeds approximately $3n$.

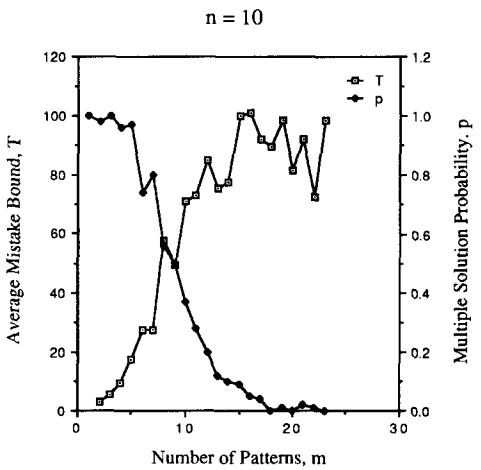

FIG. 4. The expected mistake bound and the probability that directed drift terminates in a different solution vector than the one specified (at random) initially are plotted as a function of the number of patterns for $n = 10$. (Results are averaged over 100 independent trials.) Note the same threshold behaviour around capacity and the saturation pheneomenon for the mistake bound as observed in Fig. 3. The probability that there is more than one binary solution is a dual of the curve for the mistake bound: the probability of many binary solution vectors is high when the number of patterns is small and plunges abruptly around capacity to essentially zero around $3n$. The saturation of the mistake bound around $3n$ patterns thus seems to be a consequence of the reduction in the binary solution space till there is only a unique binary solution vector around $3n$ patterns. Specifying more examplar patterns then does not yield any further information on the solution vector.

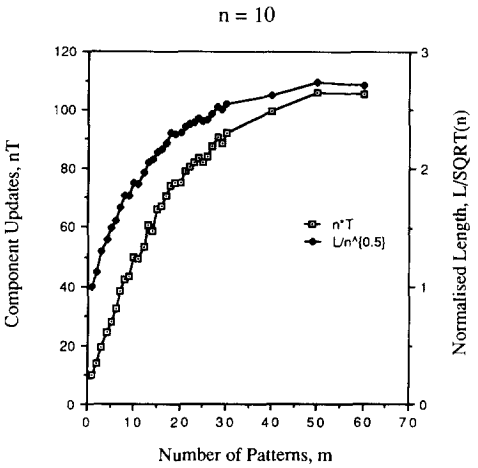

FIG. 5. The expected mistake bound and the length of the solution vector (normalised by $\sqrt{n} = \sqrt{10}$) at convergence is plotted against the number of patterns for $n = 10$ for fixed increment perceptron training. (Results were averaged over a 100 independent trials.) The averaged mistake bound saturates around $6n$ patterns, reflecting the larger capacity, while the normalised length of the solution vector returned by the algorithm saturates at a value somewhat larger (about a factor of 2.5) than the length of a binary solution vector.

This observation has an important consequence from the point of view of generalisation in learning. *Any linearly separable Boolean function for which there exists a binary solution vector can be learnt with no more than* $3n$ *examples (of the total of* 2^n *instances) of the functions drawn at random.* (See Baum and Lyuu [11].)

In Fig. 5 we plot the average number of component updates before convergence versus the number of patterns (with $n = 10$ fixed) for fixed increment perceptron training. (The expected mistake bound is an order of magnitude smaller: the average number of component updates is n times the mistake bound for fixed increment perceptron training. For single update directed drift, as noted before, the mistake bound coincides with the number of component updates.) The sharp threshold behaviour seen in directed drift is not so much in evidence here. Saturation again appears to be around twice the capacity ($2n$ for real weights). Note that the average mistake bound is an order of magnitude lower than the worst-case upper bound $O(n^2)$. The derived worst-case bound may, hence, be too conservative. On the same figure we also plot the normalised length, $L/\sqrt{n}$, of the solution vector returned by the algorithm. A similar saturation phenomenon is in evidence, with the length of the solution vector saturating at a value somewhat larger than the length, $\sqrt{n}$, of the binary solution vector.

6. CONCLUSIONS

While the general problem of learning binary weights is NP-complete, the rapid convergence of the directed drift algorithms indicates that the *typical* problem may

well be tractable even if there exist, perhaps pathological, intractable bad instances. The simplicity of these probabilistic (binary) learning algorithms allows of several possible extensions to networks of neurons—in particular, feedforward structures. This is clearly of some theoretical and practical import and is under investigation (Fang and Venkatesh [10]).

ACKNOWLEDGMENTS

My thanks to Stephen Judd for staying up working on the convergence rate, David Miller for his assistance in running the simulations of Figs. 2 and 3, Inchi Hu for bringing Siegmund's book to my attention, Ronny Meir and Ron Rivest for their comments, and the two referees who helped improve the presentation of the material.

REFERENCES

1. F. ROSENBLATT, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms," Spartan Books, Washington, DC, 1962.
2. E. BAUM, The perceptron algorithm is fast for non-malicious distributions, preprint.
3. M. GAREY AND D. JOHNSON, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco, 1979.
4. S. S. VENKATESH, Computation and learning in networks with binary synapses, presented at "Workshop on Neural Networks for Computing, Snowbird, Utah, April 1989;" also in, Learning binary weights for majority functions, *in* "Proceedings of the 4th Workshop on Compute Learning Theory" (L. G. Valiant and M. K. Warmuth, Eds.), San Mateo, CA, Morgan Kaufmann, 1991.
5. S. S. VENKATESH AND J. FRANKLIN, How much information can one bit of memory retain about a Bernoulli sequence? *IEEE Trans. Inform. Theory* **37** (1991), 1595–1604.
6. S. KIRKPATRICK, C. D. GELATT, JR., AND M. P. VECCHI, Optimisation by simulated annealing, *Science* **220** (1983), 671–680.
7. W. FELLER, "An Introduction to Probability Theory and Its Applications," Vol. I, 3rd ed., Wiley, New York, 1968.
8. D. SIEGMUND, Sequential Analysis: Test and Confidence Intervals. New York: Springer-Verlag, 1985.
9. H. CHERNOFF, A measure of asymptotic efficiency for test of a hypothesis based on a sum of observations, *Ann. Math. Statist.* **23** (1952), 493–507.
10. S. FANG AND S. S. VENKATESH, The outrageous effectiveness of batching in learning binary weights, in preparation.
11. E. B. BAUM AND Y. LYUU, The transition to perfect generalisation in perceptrons, *Neur. Comput.* **3** (1991), 386–401.