Learning Finite Binary Sequences from Half-Space Data

Shao C. Fang and Santosh S. Venkatesh
Department of Electrical Engineering
University of Pennsylvania
Philadelphia, PA 19104
fang@pender.ee.upenn.edu venkatesh@ee.upenn.edu

ABSTRACT

The problem of inferring a finite binary sequence $w^* \in \{-1, 1\}^n$ is considered. It is supposed that at epochs $t = 1, 2, \ldots$, the learner is provided with random half-space data in the form of finite binary sequences $u^{(t)} \in \{-1, 1\}^n$ which have positive inner-product with w^*. The goal of the learner is to determine the underlying sequence w^* in an efficient, on-line fashion from the data $\{ u^{(t)}, t \geq 1 \}$.
In this context, it is shown that the randomised, on-line Directed Drift algorithm [5] produces a sequence of hypotheses $\{ w^{(t)} \in \{-1, 1\}^n, t \geq 1 \}$ which converges to w^* in finite time with probability one. It is shown that while the algorithm has a minimal space complexity of $2n$ bits of scratch memory, it has exponential time complexity with an expected mistake bound of order $\Omega\left(e^{0.139n}\right)$. Batch incarnations of the algorithm are introduced which allow of massive improvements in running time with relatively small cost in space (batch size). In particular, using a batch of $\mathcal{O}(n \log n)$ examples at each update epoch reduces the expected mistake bound of the (batch) algorithm to $\mathcal{O}(n)$ (in an asynchronous bit update mode) and $\mathcal{O}(1)$ (in a synchronous bit update mode). The problem considered here is related to BINARY INTEGER PROGRAMMING and to learning in a mathematical model of a neuron.

1. A Problem in Learning

Write $\mathbb{B} \triangleq \{-1, 1\}$ for simplicity and denote the vertices of the binary n-cube by $\mathbb{B}^n \triangleq \{-1, 1\}^n$. A *binary perceptron* (or McCulloch-Pitts neuron) is a computational device characterised by a vector of binary weights $w = (w_1, \ldots, w_n) \in \mathbb{B}^n$ which accepts literals $u = (u_1, \ldots, u_n) \in \mathbb{B}^n$ as input and produces as output a Boolean value $f_w(u) \in \mathbb{B}$ given by the sign of a linear form of the inputs:

$$f_w(u) = \operatorname{sgn}\langle w, u \rangle = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i u_i\right) = \begin{cases} -1 & \text{if } \sum_{i=1}^{n} w_i u_i < 0, \\ +1 & \text{if } \sum_{i=1}^{n} w_i u_i \geq 0. \end{cases}$$

Note that the Boolean functions f_w are simply majority functions of a set of literals. In the sequel we identify the finite binary sequence of weights $w = (w_1, \ldots, w_n)$ with the corresponding majority function f_w and refer to w simply as a perceptron.

In an equivalent formulation, each perceptron $w \in \mathbb{B}^n$ engenders a positive half-space of vertices

$$\mathbb{B}^n_+(w) \triangleq \{ u \in \mathbb{B}^n : \langle w, u \rangle \geq 0 \}.$$

The family of majority functions $\{f_w\}$ (the *concepts*) may hence be identified with the indicator functions for the class of positive half-spaces of vertices $\{\mathbb{B}^n_+(w)\}$.

Let $w^* \in \mathbb{B}^n$ be some fixed (but unknown) target perceptron. Suppose we are provided with a random sequence of positive examples $\{ u^{(t)}, t \geq 1 \}$ of w^* obtained by independent sampling from the uniform distribution on the positive half-space of vertices $\mathbb{B}^n_+(w^*)$.[1] Our goal is to infer the target perceptron w^* in an efficient manner from the sample $\{u^{(t)}\}$.[2]

For any $T \geq 1$, say that a *hypothesis* $w \in \mathbb{B}^n$ is *consistent* on the finite subsample $u^{(1)}, \ldots, u^{(T)}$ if

$$\langle w, u^{(t)} \rangle = \sum_{i=1}^{n} w_i u_i^{(t)} \geq 0 \qquad (1 \leq t \leq T). \quad (1)$$

[1] The statistical model of example generation espoused here is asymptotically equivalent to the other "natural" model where positive and negative examples are generated uniformly from the vertices $\mathbb{B}^n$ of the cube. In this case the learner is given a labelled sequence of examples $\{(u^{(t)}, v^{(t)}), t \geq 1\}$ where $v^{(t)} = f_{w^*}(u^{(t)}) = \operatorname{sgn}\langle w^*, u^{(t)} \rangle$ and the sequence $\{u^{(t)}, t \geq 1\}$ is generated by independent sampling from the uniform distribution on $\mathbb{B}^n$. The simple expedient of reflecting all negative examples about the origin, i.e., setting $u^{(t)} \leftarrow -u^{(t)}$ if $v^{(t)} = -1$, converts this into a situation wherein all examples are positive. It is easy to verify that for n odd this results in an induced distribution of positive examples which is uniform over the positive half-space of vertices $\mathbb{B}^n_+(w^*)$ while for n even the induced distribution over $\mathbb{B}^n_+(w^*)$ differs from the uniform only in an asymptotically negligible term of order $\mathcal{O}(n^{-1/2})$ corresponding to vertices orthogonal to w^*.

[2] The reader used to PAC models of learning may find the goal of *exact* learning, albeit for the uniform distribution, rather stringent. As we will see, however, exact learning can be accomplished in our setting with rather parsimonious demands on time and space. Trivial modifications of the analysis accommodate the more modest goal of finding a good approximation to the target binary perceptron w^*.

It is clear that if any hypothesis is consistent on the entire sample $\{\mathbf{u}^{(t)}, t \geq 1\}$ then it must necessarily coincide with the target perceptron with probability one as every vertex in the positive half-space $\mathbb{B}_+^n(\boldsymbol{w}^*)$ occurs (infinitely often) in the sample with probability one. Much fewer examples are actually needed on information theoretic grounds; indeed, Lyuu and Rivin [2] show, for instance, that, if $T \geq 1.45n$, any hypothesis which is consistent on a random finite subsample $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(T)}$ will have an exponentially small (in n) probability of differing from the target perceptron. Thus, if, for instance, an on-line procedure successfully generates a consistent hypothesis after a linear number of time steps ($\geq 1.45n$) then we can assert with high confidence that the algorithm has, in fact, identified the target perceptron.

It may be remarked that the decision problem associated with the system of inequalities (1) is an NP-complete variant of BINARY INTEGER PROGRAMMING[3] (cf. Pitt and Valiant [3] and Fang [1]) raising the spectre of an intractable worst-case instance of the problem even if one is provided with an off-line algorithm. We show nevertheless that, in a sense, typical instances of the problem are simple and are amenable to solution with a very modest expenditure of resources in time and space.

We consider on-line and batch learning scenarios where an algorithm generates a sequence of hypotheses $\{\boldsymbol{w}^{(t)} \in \mathbb{B}^n, t \geq 1\}$ which successively refines our estimate of the target binary perceptron $\boldsymbol{w}^*$. In both settings, learning proceeds at discrete epochs in time $t = 1, 2, \ldots$. In the on-line setting, at each update epoch t, the algorithm generates an updated hypothesis $\boldsymbol{w}^{(t+1)} \in \mathbb{B}^n$ as a (possibly randomised) function solely of the current hypothesis $\boldsymbol{w}^{(t)}$, the current (positive) example $\mathbf{u}^{(t)}$, and the update epoch t; more generally, in the batch learning setting the algorithm is allowed to draw upon a batch of m examples (where $m = m(n)$ may depend on n and is referred to as the *batch sample complexity*) at each update epoch so that the new hypothesis $\boldsymbol{w}^{(t+1)}$ is a (possibly randomised) function of the current hypothesis $\boldsymbol{w}^{(t)}$, examples $\mathbf{u}^{(m(t-1)+1)}, \ldots, \mathbf{u}^{(mt)}$, and the update epoch t.

We characterise the complexity of a given on-line (or batch) learning algorithm in terms of the resources it demands in space and time, i.e., its space and time complexity. The space complexity of an algorithm is the number of bits of buffer (or scratch) memory it requires. As a measure of an algorithm's time complexity we use its expected mistake bound, the expected number of epochs t at which $\langle \boldsymbol{w}^{(t)}, \mathbf{u}^{(t)} \rangle < 0$.

Our main algorithmic vehicle is the randomised, on-line Directed Drift learning algorithm of Venkatesh [4, 5]) and batch versions of the algorithm introduced here. Drawing from the theory of Markov chains we show that the sequence of hypotheses $\{\boldsymbol{w}^{(t)}, t \geq 1\}$ generated by Directed Drift (in on-line and batch modes) converges to the target binary perceptron $\boldsymbol{w}^*$ in finite time with probability one. The analysis throughout is predicated upon the availability of a random sample $\{\mathbf{u}^{(t)}, t \geq 1\}$ of positive examples obtained by independent sampling from the positive half-space of vertices $\mathbb{B}_+^n(\boldsymbol{w}^*)$. Table 1 encapsulates our main results.

2. On-Line Learning

In on-line learning, the adaptation of the hypothesis takes place in a sequence of trials at (machine) epochs $t = 1, 2, \ldots$. At each epoch t, the current hypothesis $\boldsymbol{w}^{(t)}$ is tested against the current (positive) example $\mathbf{u}^{(t)}$ and a new hypothesis $\boldsymbol{w}^{(t+1)}$ is generated based upon the outcome. In our setting the hypotheses are constrained to the vertices $\mathbb{B}^n$ of the n-cube (as are the examples); updating a hypothesis will hence necessitate a change in sign of at least one component which results in a macroscopic additive perturbation of magnitude 2 of that component. We consequently do not have the luxury of making incremental real modifications to the hypothesis *vide* the usual perceptron strategy by additively reinforcing an errant hypothesis $\boldsymbol{w}^{(t)}$ when it makes a mistake, i.e., when $\langle \boldsymbol{w}^{(t)}, \mathbf{u}^{(t)} \rangle < 0$. The Directed Drift algorithm (Venkatesh [4, 5]) is a randomised, on-line procedure which takes a different tack in this situation.

The main idea behind the Directed Drift algorithm is to update randomly chosen components of a hypothesis which makes a mistake, the auxiliary randomisation designed to ensure a positive probabilistic drift towards the desired solution, the concept $\boldsymbol{w}^*$. As we shall see, the algorithm shares similarities with asymmetric random walks with a preferred direction of drift. The following algorithm explains the single bit update incarnation of Directed Drift in more detail.

Algorithm D (*Directed Drift*). Given a sequence of positive examples $\{\mathbf{u}^{(t)}, t \geq 1\}$ generated by independent sampling from the positive half-space

[3] If the components of $\boldsymbol{w} = (w_1, \ldots, w_n)$ in (1) are allowed to range through real values, however, the problem (1) reduces to LINEAR PROGRAMMING which can be solved in polynomial time using standard techniques. Indeed, relatively simple approaches suffice when, as here, the examples $\mathbf{u}^{(t)}$ are drawn from the vertices of a cube—the classical perceptron training procedure for instance will converge in this situation in time $\mathcal{O}(n^2)$ (cf. Venkatesh [5]) to a *real* vector consistent on the sample.

	Algorithm D	Algorithm D′	Algorithm D″
Batch sample complexity	1	$\frac{1}{2}\pi n \log n$	$\pi n \log n$
Space complexity (in bits)	$2n$	$\frac{1}{2}\pi n^2 \log n + n$	$\pi n^2 \log n + n$
Time complexity (in epochs)	$\Omega(e^{0.139n})$ $\mathcal{O}(e^{n \log n})$	$\mathcal{O}(n)$	$\mathcal{O}(1)$

Table 1: Algorithm D is the plain vanilla Directed Drift algorithm operated on-line in a randomised single bit update mode. Algorithm D′ is the Directed Drift algorithm operated in batch mode with asynchronous bit updates. Algorithm D″ is the Directed Drift algorithm operated in batch mode with synchronous bit updates. The batch and space complexity estimates for Algorithms D′ and D″ are upper bounds.

of vertices $\mathbb{B}_+^n(\boldsymbol{w}^*)$, the algorithm produces a sequence of hypotheses $\{\boldsymbol{w}^{(t)} \in \mathbb{B}^n, t \geq 1\}$ on-line which, with probability one, converges in a finite number of steps to the concept $\boldsymbol{w}^*$.

D1. [Initialise.] Set $t \leftarrow 1$ and select an arbitrary initial hypothesis $\boldsymbol{w}^{(1)} \in \mathbb{B}^n$.

D2. [Is the hypothesis consistent on the example?] Set $Y \leftarrow \langle \boldsymbol{w}^{(t)}, \boldsymbol{u}^{(t)} \rangle$.

D3. [If it ain't broke, don't fix it.] If $Y \geq 0$, set $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)}$ and go to Step D5.

D4. [Update hypothesis.] Else (if $Y < 0$) denote by $J^{(t)} \triangleq \{j : w_j^{(t)} \neq u_j^{(t)}\}$ the subset of indices for which the corresponding components of $\boldsymbol{w}^{(t)}$ and $\boldsymbol{u}^{(t)}$ are opposite in sign, pick a random index $j^{(t)}$ from the uniform distribution on $J^{(t)}$, and form the new hypothesis $\boldsymbol{w}^{(t+1)}$ according to the following rule:

$$w_j^{(t+1)} \leftarrow \begin{cases} +w_j^{(t)} & \text{if } j \neq j^{(t)}, \\ -w_j^{(t)} & \text{if } j = j^{(t)}. \end{cases}$$

D5. [Increment time and iterate.] Set $t \leftarrow t + 1$ and go back to Step D2.

The intuition behind the algorithm is as follows. Let $\boldsymbol{u} \in \mathbb{B}_+^n(\boldsymbol{w}^*)$ be any positive example of $\boldsymbol{w}^*$. Then $\langle \boldsymbol{w}^*, \boldsymbol{u} \rangle = \sum_{j=1}^n w_j^* u_j \geq 0$. As there is a contribution of $+1$ to the sum if two corresponding components of $\boldsymbol{w}^*$ and $\boldsymbol{u}$ have the same sign, and -1 if the signs are mismatched, it follows that $\boldsymbol{w}^*$ has more component sign matches than mismatches with every example in $\mathcal{U}$.

Now the algorithm updates the current hypothesis $\boldsymbol{w}^{(t)}$ if, and only if, the hypothesis is not consistent on the current example $\boldsymbol{u}^{(t)}$ in which case a randomly chosen mismatched component $j^{(t)} \in J^{(t)}$ of the offending hypothesis is flipped (i.e., aligned with the sign of the corresponding component of the example). Since, by definition, the number $|J^{(t)}|$ of mismatched components exceeds $n/2$, by the pigeonhole principle there is at least one member of $J^{(t)}$ for which the corresponding components of $\boldsymbol{u}^{(t)}$ and $\boldsymbol{w}^*$ have the same sign. It follows that there is a positive probability that the hypothesis update is in the direction of the target binary perceptron $\boldsymbol{w}^*$.

Indeed, from a consideration of the equilibrium probability distribution of the underlying finite Markov chain representing the system we can show the following

Theorem 1 *The sequence of hypotheses $\{\boldsymbol{w}^{(t)}\}$ generated by Directed Drift converges to the target binary perceptron $\boldsymbol{w}^*$ in finite time with probability one. Moreover, for any initial hypothesis $\boldsymbol{w}^{(1)}$ not identically $\boldsymbol{w}^*$, the expected mistake bound of the algorithm is bounded above by $2^{-n}n^n(1 + \mathcal{O}(n^{-1}))$ and below by $\alpha e^{\beta n} + \mathcal{O}(n)$ where $\alpha = 1.771866547\ldots$ and $\beta = 0.139232271\ldots$ are absolute positive constants.*

In an actual implementation hypothesis updates would be done "in place." It is also simple to make provision for an explicit termination condition for the algorithm. Suppose that for some $\delta > 0$ it is required that the algorithm halt after generating a hypothesis $\boldsymbol{w}$ which, with confidence in excess of $1 - \delta$, coincides with the concept $\boldsymbol{w}^*$. Introduce a counter T (initially 0) which tracks the number of consecutive epochs for which the current hypothesis is consistent on the current example. (Reset T to 0 whenever the current hypothesis is not consistent on the current example.) Output the current hypothesis and terminate the algorithm when T exceeds $\sqrt{\frac{\pi n}{2}} \log \delta^{-1}$. It is simple to verify that the termination condition works as advertised as the probability that any given hypothesis $\boldsymbol{w} \neq \boldsymbol{w}^*$ correctly classifies T consecutive random positive examples of $\boldsymbol{w}^*$ is less than δ if $T > \sqrt{\frac{\pi n}{2}} \log \delta^{-1}$.

Note that at any epoch Directed Drift requires just the bare minimum of $2n$ bits in scratch (or

buffer) memory to store the current hypothesis and the current example. The algorithm hence has minimal space complexity at the price of exponential time complexity. Can the situation be improved? Yes, dramatically, by stirring in a modicum of batch information into the algorithmic pot.

3. Batch Learning

The pigeonhole principle shows that the probability that any invocation of Step D4 results in a hypothesis closer to the concept $\boldsymbol{w}^*$ is always positive. In any fixed neighbourhood of $\boldsymbol{w}^*$, however, this "probability of positive drift" is quite small (of order n^{-1} as can be readily verified by a pigeonholing argument) resulting in the exponential time complexity of Directed Drift. A correspondence with the classical gambler's ruin problem suggests that rapid convergence can be attained by improving the selection of component to be updated in Step D4 so as to obtain a minimum probability of positive drift bounded uniformly away from 0 (independent of n). Consider first a simple modification of Directed Drift which, when the current hypothesis misclassifies the current example, requests an additional batch of $m-1$ positive examples to improve the selection of component to be updated.

Algorithm D′ (*Directed Drift in batch mode with asynchronous bit updates*). Given a batch size m and a random sample of positive examples $\{\mathbf{u}^{(t)}, t \geq 1\}$, the algorithm proceeds exactly as Algorithm D with Step D4 replaced by

D4′ [Update hypothesis.] Else (if $Y < 0$) call an additional batch of $m-1$ examples $\mathbf{u}^{(t+1)}, \ldots, \mathbf{u}^{(t+m-1)}$. For integers $1 \leq k \leq n$ and $t \leq s \leq t+m-1$, define the indicator functions

$$I_k^{(s)} = \begin{cases} 1 & \text{if } w_k^{(t)} \neq u_k^{(s)}, \\ 0 & \text{if } w_k^{(t)} = u_k^{(s)}, \end{cases}$$

and select the index j garnering the most votes: $j \leftarrow \arg\max_k \sum_{s=t}^{t+m-1} I_k^{(s)}$. Set

$$w_k^{(t+1)} \leftarrow \begin{cases} -w_k^{(t)} & \text{if } k = j, \\ +w_k^{(t)} & \text{if } k \neq j, \end{cases}$$

and set $t \leftarrow t+m-1$.

By selecting the component to be updated based upon the majority vote of a random batch of positive examples, Algorithm D′ improves on the randomised approach of the plain vanilla Directed Drift algorithm. The approach bears fruit if the size of the batch is sufficiently large.

Theorem 2 *Algorithm D′ converges with an expected mistake bound of order $O(n)$ if the batch sample complexity satisfies $m \geq \frac{1}{2}\pi n \log n$.*

Convergence can be accelerated further at a slight additional expense in space complexity by using batch information to update several bits per epoch.

Algorithm D″ (*Directed Drift in batch mode with synchronous bit updates*). Given a batch size m and a random sample of positive examples $\{\mathbf{u}^{(t)}, t \geq 1\}$, the algorithm progresses exactly as Algorithm D with Step D4 replaced by

D4″ [Update hypothesis.] Else (if $Y < 0$) call an additional batch of $m-1$ examples $\mathbf{u}^{(t+1)}, \ldots, \mathbf{u}^{(t+m-1)}$. For integers $1 \leq k \leq n$ and $t \leq s \leq t+m-1$, define the indicator functions

$$I_k^{(s)} = \begin{cases} 1 & \text{if } w_k^{(t)} \neq u_k^{(s)}, \\ 0 & \text{if } w_k^{(t)} = u_k^{(s)}. \end{cases}$$

Tally the votes $b_k = \sum_{s=t}^{t+m-1} I_k^{(s)}$ for $k = 1, \ldots, n$. Set

$$w_k^{(t+1)} \leftarrow \begin{cases} -w_k^{(t)} & \text{if } b_k \geq m/2, \\ +w_k^{(t)} & \text{if } b_k < m/2, \end{cases}$$

and set $t \leftarrow t+m-1$.

Theorem 3 *Algorithm D″ converges with an expected mistake bound of order $O(1)$ if the batch sample complexity satisfies $m \geq \pi n \log n$.*

Note that we can attain convergence in constant expected time in a synchronous update mode with very small (log-linear) batch sizes. Thus, a rather modest space complexity of $O(n^2 \log n)$ bits of buffer memory will suffice to obtain rapid $O(1)$ convergence.[4]

The proofs of these theorems utilise large deviation central tendency in the tails of multivariate random walks coupled with Poisson clumping arguments. As a substitute for the formal proofs (which will appear elsewhere in the interests of brevity here) we offer the following computer simulation providing apodictic asymptotic support of the theorems.[5] In Figure 1, for each n the average mistake bound of Algorithm D′ (obtained from 1000 independent runs) is plotted against the batch size m. For the values of n employed, all the (normalised) mistake bounds saturate below 1.0 for sufficiently large batch sizes, bespeaking a high probability of positive drift. For small batch sizes, however, the exponential convergence behaviour of the

[4] "Qu'ils mangent de la brioche ... et du pain aussi," to misquote Marie-Antoinette, or, how to have the cake ... and the bread.

[5] "Merely corroborative detail, intended to give artistic verisimilitude to an otherwise bald and unconvincing narrative." — W. S. Gilbert, *The Mikado*.

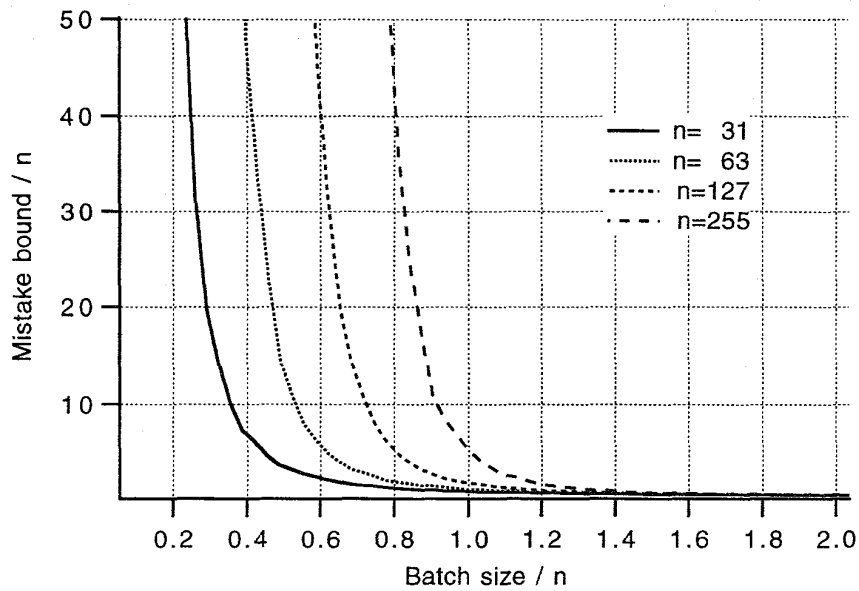

Figure 1: Computer simulations of the batch version of Directed Drift in the asynchronous bit update mode. Normalised mistake bounds are plotted versus normalised batch sizes with n, the dimensionality, as a parameter.

algorithm becomes apparent, a consequence of the diminishing probability of positive drift.

Acknowledgements

Support from the US Air Force Office of Scientific Research under grants F49620-93-1-0120 and F49620–92–J–0344 for the work reported here is gratefully acknowledged.

References

[1] S. C. Fang, *Capacity and Complexity in Learning Binary Perceptrons*, Ph.D. dissertation, University of Pennsylvania, 1995.

[2] Y.-D. Lyuu and I. Rivin, "Tight bounds on the transition to perfect generalisation in perceptrons," *Neural Computation*, vol. 4, pp. 854–862, 1992.

[3] L. Pitt and L. G. Valiant, "Computational limitations on learning from examples," *J. Assoc. Comput. Machinery*, vol. 35, no. 4, pp. 965–984, 1988.

[4] S. S. Venkatesh, "On learning binary weights for majority functions," in *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*. San Mateo, California: Morgan Kaufmann, 1991.

[5] S. S. Venkatesh, "Directed Drift: a new linear threshold algorithm for learning binary weights on-line," *J. Comp. Sys. Sciences*, vol. 46, pp. 198–217, 1993.