Convergence Properties and Stationary Points of a Perceptron Learning Algorithm

JOHN J. SHYNK, MEMBER, IEEE AND SUMIT ROY

The Perceptron is an adaptive linear combiner that has its output quantized to one of two possible discrete values, and it is the basic component of multilayer, feedforward neural networks. The least-mean-square (LMS) adaptive algorithm adjusts the internal weights to train the network to perform some desired function, such as pattern recognition. In this paper, we present an analysis of the stationary points of a single-layer Perceptron that is based on the momentum LMS algorithm, and we illustrate some of its convergence properties. When the input of the perceptron is a Gaussian random vector, the stationary points of the algorithm are not unique and the behavior of the algorithm near convergence depends on the step size μ and the momentum constant α.

I. Introduction

A single-layer Perceptron [1], or adaptive linear neuron (ADALINE) [2], consists of one summing node and N adaptive weights $\{w_k(n)\}$ as shown in Fig. 1. It is the simplest feedforward neural network structure, and it corresponds to a single "neuron" element. This basic component may be combined in parallel with many similar components to produce a multilayer Perceptron that has greater learning and computing capabilities [3]. The output $y(n)$ of the summer is filtered by a hard limiter[1] to produce a binary output $y_q(n)$ (denoted by $+1$ and -1). This output is compared to another binary signal, which corresponds to some desired response $d_q(n)$, and an error signal $e(n)$ is generated. An adaptive learning algorithm [4] then uses this error to make adjustments to the Perceptron weights so as to match $y_q(n)$ and $d_q(n)$ in a statistically meaningful way. The input signals $\{x_k(n)\}$ can be binary valued (± 1) or they can be drawn from a continuous distribution. In our analysis, we assume that they form a Gaussian random vector whose components may or may not be correlated.

The Perceptron in Fig. 1 can be used as a pattern classifier whereby the N-dimensional vector space represented by the input signals is partitioned into two subspaces, corresponding to the two classes (denoted by A and B). It is well known, however, that the partition in this case is a hyperplane [2]; more complex partitions can be achieved only by adding "hidden" layers to the Perceptron [1]. Figure 2 shows

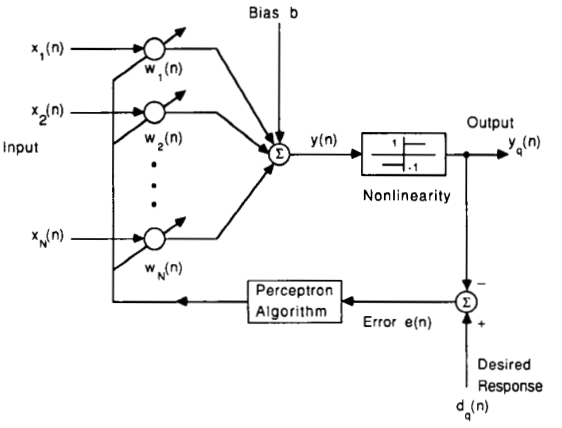

Fig. 1. Single-layer Perceptron with hard limiter.

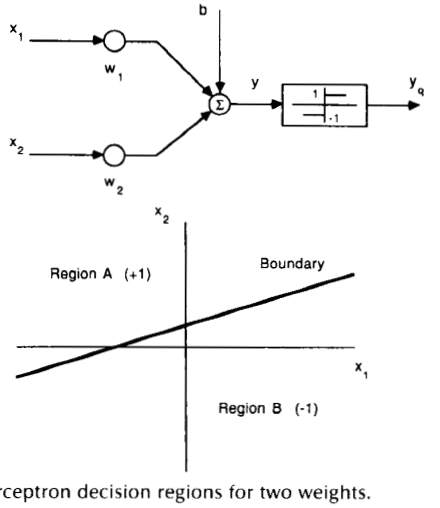

Fig. 2. Perceptron decision regions for two weights.

Manuscript received September 6, 1989; revised March 25, 1990. This work was supported by Rockwell, Inc., with matching support from the University of California MICRO Program, and by the University of Pennsylvania.

J. J. Shynk is with the Center for Information Processing Research, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA.

S. Roy is with the Department of Electrical Engineering, University of Pennsylvania, Philadelphia, PA 19104, USA.

[1]In some cases, the hard limiter is replaced by a smooth nonlinearity such as the sigmoid function [1].

IEEE Log Number 9039178.

0018-9219/90/1000-1599$01.00 © 1990 IEEE

an example of the partition for the case of two input signals ($N = 2$); observe that the decision boundary is defined by the line $x_2 = -(w_1/w_2)x_1 - (b/w_2)$, where b represents a bias factor. The perceptron can be trained to identify this decision boundary so that for a given input, it will correctly decide to which class the sample pair belongs.

In this paper, we present an analysis of the stationary (convergence) points of an adaptive algorithm that adjusts the perceptron weights [5]. This algorithm is identical in form to the least-mean-square (LMS) algorithm [4], except that a hard limiter is incorporated at the output of the summer as shown in Fig. 1. We include a momentum term in the weight update [3]; this modified algorithm is similar to the momentum LMS (MLMS) algorithm [6], [7], except again it contains the output nonlinearity. Section II describes the perceptron algorithm in detail, and it presents a simple two-input example that will be used for illustration purposes throughout the paper. The stationary points of the algorithm are then presented in Section III, and the properties of the adaptive weight vector near convergence are discussed. Computer simulations that verify the analysis are given in Section IV, and conclusions are outlined in Section V. A related analysis of this algorithm, which is also often called the delta learning rule, is presented in [8].

II. Perceptron Learning Algorithm

A. Algorithm Description

The perceptron algorithm considered here has the following weight updating mechanism:

$$W(n+1) = W(n) + 2\mu e(n)X(n) + \alpha[W(n) - W(n-1)] \tag{1}$$

where $W(n)$ and $X(n)$ are the N-dimensional weight and signal vectors, respectively:

$$W(n) = [w_1(n), \cdots, w_N(n)]^T \tag{2a}$$

and

$$X(n) = [x_1(n), \cdots, x_N(n)]^T. \tag{2b}$$

The superscript T is vector transpose, and n denotes the present sample time. Equation (1) is a second-order algorithm that requires storage of the present and previous weight vectors before the new weight vector can be computed. The step size μ and the momentum constant α, where $\mu > 0$ and $|\alpha| < 1$, determine the convergence rate and steady-state performance of the algorithm. The component given by $\alpha[W(n) - W(n-1)]$ is the so-called momentum term [3]. Intuitively, if the previous weight change is large, then adding a fraction of this amount to the current weight update will accelerate the descent process toward the algorithm convergence point. However, it can be shown that the misadjustment [4] of the MLMS algorithm is increased in direct proportion to α [6], [7].

The output error $e(n)$ is derived as the difference between the (binary-valued) desired response $d_q(n)$ and the quantized filter output $y_q(n)$:

$$e(n) = d_q(n) - y_q(n) = d_q(n) - \text{sgn}\,(y(n)) \tag{3}$$

where sgn (g) is the sign function defined as

$$\text{sgn}\,(g) = \begin{cases} +1, & g \geq 0 \\ -1, & g < 0. \end{cases} \tag{4}$$

Observe in Fig. 1 that there are N input signals which are weighted and then summed to produce the *unquantized* output $y(n)$; this part of the perceptron is simply a linear combiner. The intermediate output $y(n)$ is thus given by the following inner product:

$$y(n) = W^T(n)X(n) = X^T(n)W(n). \tag{5}$$

Note that $y(n)$ is always quantized to produce $y_q(n)$. The desired response $d_q(n)$ is also constrained to be ± 1, so we can view it as being a quantized version of some *underlying* process $d(n)$, that is

$$d_q(n) = \text{sgn}\,(d(n)). \tag{6}$$

In general, $d(n)$ will be correlated with $X(n)$, and it can often be represented as a function (possibly nonlinear) of the elements of $X(n)$. One interesting case that will be examined later is when $d(n)$ is a linear function of $X(n)$ according to

$$d(n) = F^T(n)X(n) = X^T(n)F(n) \tag{7}$$

where $F(n)$ is an unknown weight vector defined in a manner similar to $W(n)$. In our analysis, we assume that $F(n)$ is constant such that $F(n) = F$.

B. Two-Weight Example

For illustration purposes, we consider a simple example for $N = 2$, corresponding to a special case of the decision regions shown in Fig. 2. Assume that the bias term is zero ($b = 0$) so that the boundary passes through the origin, and let the data be generated such that the boundary lies at a 45° angle with respect to the signal axes. Region A (above and on the line) is denoted by $+1$, and region B (below the line) is denoted by -1.

The network is trained as follows. The input signals $x_1(n)$ and $x_2(n)$ are independently assigned values from a zero-mean, Gaussian distribution having a variance of 1. If the corresponding point on the plane lies in region A, [that is, $x_2(n) \geq x_1(n)$], $d_q(n)$ will be set equal to $+1$; otherwise, $d_q(n) = -1$. The input samples and the associated desired sample will be presented to the network, and the perceptron algorithm will adjust the weights according to the update in (1). Clearly, a stationary point should be of the form $w_1(n) = -w_2(n)$, since this is the way the data were generated.

For this simple example, it is straightforward to model the underlying desired process as the following linear combination of the input signals: $d(n) = x_2(n) - x_1(n)$. As such, the vector F in (7) is[2] $F = [-1 \quad 1]^T$. Since the input samples are assumed to form a Gaussian vector, then $d(n)$ is necessarily a Gaussian process. This property will lead to some convenient analytical expressions for the stationary points, as discussed in the next section.

III. Stationary Points of the Algorithm

A. Mean Weight Vector

In order to determine the stationary points of the Perceptron algorithm, we examine the expected value of (1):

$$E[W(n+1)] = E[W(n)] + 2\mu E[e(n)X(n)] + \alpha(E[W(n)] - E[W(n-1)]). \tag{8}$$

[2]As another example, if the boundary was chosen to lie along the x_2 axis, then $F = [-1 \quad 0]^T$.

At convergence, assuming that μ has been chosen "sufficiently small," we have $E[W(n+1)] = E[W(n)] = E[W(n-1)] = W_*$, where W_* represents a stationary point. Alternatively, we may view (1) as a gradient-descent algorithm, and we are interested in finding weight values such that the gradient is zero. As a result, (8) reduces to the following *orthogonality condition* [4]:

$$E[e_*(n)X(n)] = 0. \tag{9}$$

The subscript * indicates that the error is generated when the weights are at the stationary point W_*. Equation (9) states that the error and the input signal are statistically orthogonal at convergence. This result is identical to that of the standard (linear) LMS algorithm, and it can be used to find W_*. If we substitute (3), then (9) can be rearranged as

$$E[X(n)d_q(n)] = E[X(n)\ \text{sgn}\ (y_*(n))] \tag{10}$$

where $y_*(n)$ is the unquantized output at convergence. Define the cross-correlation between $X(n)$ and $d_q(n)$ as $P_q = E[X(n)d_q(n)]$. Note that the actual form of this vector depends on the statistics of the underlying process $d(n)$; later we consider the linear example described previously for $d(n)$, but for now we leave P_q in this more general form. Assuming that $X(n)$ is a zero-mean, Gaussian vector with correlation matrix $R = E[X(n)X^T(n)]$, then the right-hand side of (10) becomes [9]

$$E[X(n)\ \text{sgn}\ (y_*(n))] = \frac{1}{c\sigma_{y_\cdot}} E[X(n)y_*(n)] \tag{11}$$

where the constant $c = \sqrt{\pi/2}$ and $\sigma^2_{y_\cdot}$ is the variance of $y_*(n)$, given by

$$\sigma^2_{y_\cdot} = E[y^2_*(n)] = E[W^T_*X(n)X^T(n)W_*] = W^T_*RW_*. \tag{12}$$

Equation (5) was used to derive (12) and, because W_* is fixed, we have brought it outside the expectation. The expectation on the right-hand side of (11) can be expanded in a similar way, as follows:

$$E[X(n)y_*(n)] = RW_*. \tag{13}$$

Substituting (11) and (13) into (10), we have the following equivalent expression for (9):

$$W_* = c\sigma_{y_\cdot}R^{-1}P_q \tag{14}$$

which, after substituting (12), becomes

$$W_* = c\sqrt{W^T_*RW_*}\ R^{-1}P_q. \tag{15}$$

The square root is well defined because the variance in (12) is always nonnegative, and we assume that R is positive definite so that the inverse exists. This expression defines the stationary points of the perceptron algorithm. Observe that it is a *nonlinear* function of W_*; because of this form, there may be infinitely many solutions. However, if for fixed values of μ and α the convergence point $\sigma^2_{y_\cdot}$ of $\sigma^2_y(n) = E[y^2(n)]$ is unique, then the weight vector W_* will also be unique according to (14). Note that the output variance $\sigma^2_y(n)$ is nonstationary and dependent on n because the perceptron weights are time-varying.

It is interesting to consider when $d(n)$ is also a Gaussian process. In this case, we have from (6) that

$$P_q = \frac{1}{c\sigma_d} E[X(n)d(n)] \tag{16}$$

where σ^2_d is the variance of $d(n)$, which is independent of n when $d(n)$ is stationary. Substituting (16) into (14) and defining the cross-correlation between $X(n)$ and $d(n)$ as $P = E[X(n)d(n)]$, we have

$$W_* = \frac{\sigma_{y_\cdot}}{\sigma_d} R^{-1}P. \tag{17}$$

If $\sigma_{y_\cdot} = \sigma_d$, then the optimal solution here is identical in form to the Wiener solution of the standard (linear) MLMS algorithm [6], [7], that is

$$W_* = R^{-1}P. \tag{18}$$

In general, however, $\sigma_{y_\cdot}$ does not equal σ_d so that a *scaled* version of (18) would be obtained, as given in (17). Finally, if we assume that $d(n)$ is generated according to (7) for a fixed value of F, then $P = RF$, $\sigma^2_d = F^TRF$, and

$$W_* = \frac{\sigma_{y_\cdot}}{\sigma_d} F, \tag{19}$$

that is, the optimal weights are directly proportional to F, where the proportionality constant is a nonnegative scalar. This result is consistent with our intuition; if W_* and F are related as in (19), then $y(n)$ and $d(n)$ will have the same sign for any $X(n)$, and the error will necessarily be zero, corresponding to a stationary point. To continue further, we need to examine the convergence properties of the output variance $\sigma^2_y(n)$ of the linear combiner.

B. Steady-State Output Variance

The output variance of the linear combiner can be written as

$$\sigma^2_y(n, n) = E[W^T(n)X(n)X^T(n)W(n)] \tag{20}$$

where we have substituted (5). For notational clarity, we have added a second time argument; together these arguments correspond to those of the weight vector and the input signal vector, respectively, under the expectation. After the weights are updated, the *a posteriori* variance can be expressed as

$$\sigma^2_y(n+1, n) = E[W^T(n+1)X(n)X^T(n)W(n+1)]. \tag{21}$$

Substituting the weight recursion from (1), we have that

$$\begin{aligned}\sigma^2_y(n+1, n) = {} & (1+\alpha)^2E[W^T(n)X(n)X^T(n)W(n)] \\ & + 4\mu(1+\alpha)E[X^T(n)X(n)X^T(n)W(n)e(n)] \\ & + 4\mu^2E[X^T(n)X(n)X^T(n)X(n)e^2(n)] \\ & - 2\alpha(1+\alpha)E[W^T(n)X(n)X^T(n)W(n-1)] \\ & - 4\mu\alpha E[X^T(n)X(n)X^T(n)W(n-1)e(n)] \\ & + \alpha^2E[W^T(n-1)X(n)X^T(n)W(n-1)]\end{aligned} \tag{22}$$

which can be written more compactly as

$$\begin{aligned}\sigma^2_y(n+1, n) = {} & (1+\alpha)^2\sigma^2_y(n, n) + 4\mu(1+\alpha)a(n) + 4\mu^2b(n) \\ & -2\alpha(1+\alpha)\gamma(n, n-1) - 4\mu\alpha a(n-1) \\ & + \alpha^2\sigma^2_y(n-1, n)\end{aligned} \tag{23}$$

where, for convenience, we have defined the following scalar quantities:

$$\gamma(n, n-1) = E[W^T(n)X(n)X^T(n)W(n-1)] \quad (24a)$$

$$a(n) = E[X^T(n)X(n)X^T(n)W(n)e(n)] \quad (24b)$$

and

$$b(n) = E[X^T(n)X(n)X^T(n)X(n)e^2(n)]. \quad (24c)$$

In contrast to that in (20) and (21), the arguments of γ and a are defined only according to those of W under the expectation, and the argument of b is determined by that of e. To continue, we also need a recursion for $\gamma(n+1, n)$, as follows:

$$\begin{aligned}\gamma(n+1, n) &= E[W^T(n+1)X(n)X^T(n)W(n)] \\ &= (1+\alpha)E[W^T(n)X(n)X^T(n)W(n)] \\ &\quad + 2\mu E[X^T(n)X(n)X^T(n)W(n)e(n)] \\ &\quad - \alpha E[W^T(n)X(n)X^T(n)W(n-1)] \\ &= (1+\alpha)\sigma_y^2(n, n) + 2\mu a(n) - \alpha\gamma(n, n-1)\end{aligned} \quad (25)$$

where again (1) has been substituted.

Near convergence the weights approach W_*, a stationary point, and we have for "small"[3] μ that $\sigma_y^2(n+1, n) \approx \sigma_y^2(n, n) \approx \sigma_y^2(n-1, n) \to \sigma_{y,\infty}^2$, $\gamma(n+1, n) \approx \gamma(n, n-1) \to \gamma$, $a(n) \approx a(n-1) \to a$, and $b(n) \to b$, which are all independent of time. Therefore, we can replace (23) and (25) by the following coupled pair of *deterministic* equations:

$$\sigma_{y,\infty}^2 \approx (1 + 2\alpha + 2\alpha^2)\sigma_{y,\infty}^2 + 4\mu a - 2\alpha(1+\alpha)\gamma + 4\mu^2 b \quad (26a)$$

and

$$\gamma \approx (1+\alpha)\sigma_{y,\infty}^2 + 2\mu a - \alpha\gamma. \quad (26b)$$

By eliminating the common terms from these two expressions, we have the following condition that defines the output variance *near convergence:*

$$(1-\alpha)a + \mu b \approx 0. \quad (27)$$

Notice that this condition depends on the parameters μ and α. By examining a near convergence, we can approximate it as follows:

$$a \approx W_*^T E[X(n)X^T(n)X(n)e_*(n)] = W_*^T S \quad (28)$$

where W_* is the weight vector in (14), which has been factored from the expectation because we are assuming that the weight fluctuations near convergence are negligible. The vector S is given by $S = E[X(n)X^T(n)X(n)e_*(n)]$. By substituting (14) and (28) into (27) and solving for $\sigma_{y,\infty}$, we find that

$$\sigma_{y,\infty} \approx \frac{-b\mu}{c(1-\alpha)\,P_q^T R^{-1} S} = \frac{\mu}{c(1-\alpha)}\,k \quad (29)$$

where we have defined the positive scalar $k = -b/(P_q^T R^{-1} S)$. Substituting (29) into (14), the following expression is obtained, which represents the properties of the perceptron weight vector near convergence:

$$W_* \approx k\left(\frac{\mu}{1-\alpha}\right) R^{-1} P_q. \quad (30)$$

In general, it is difficult to determine closed-form expressions for P_q, S, b, and thus k. However, we are not so much interested in evaluating (30) as we are in the asymptotic relationship (i.e., near convergence) between the weights and the parameters μ and α. Notice that W_* is a *linear* function of μ; if μ is increased by a factor of 10, for example, then the weight values are also scaled by a factor of 10. On the other hand, W_* depends on α in a *nonlinear* way. Furthermore, it behaves differently for positive and negative values of α. If $\alpha > 0$, then the weights increase as $\alpha \to 1$, becoming extremely large as α approaches 1. However, for $\alpha < 0$, the weights decrease as $\alpha \to -1$, remaining relatively small.

Finally, if we assume that $d(n)$ is generated according to (6) and (7), then (30) simplifies to

$$W_* \approx k'\left(\frac{\mu}{1-\alpha}\right) F \quad (31)$$

where (19) has been used and $k' = -b/(F^T S)$ is a positive scalar. Thus, the weights near convergence are proportional to F.

IV. Computer Simulations

In the simulations presented here, a two-weight perceptron ($N = 2$) was examined with $b = 0$, $R = I$ (the identity matrix), and $F = [-1 \quad 1]^T$. As such, the boundary passes through the origin at an angle of 45° with respect to the signal axes, $P = [-1 \quad 1]^T$, and $\sigma_d^2 = 2$. The perceptron was trained to "learn" the location of this boundary starting from the zero weight vector. In all simulations, the weight trajectories were averaged over 100 independent computer runs to generate relatively smooth curves. We considered two cases: (a) one weight was fixed and the other was allowed to adapt, and (b) both weights were allowed to adapt.

Figure 3 shows the weight trajectories of $w_2(n)$ for four values of μ with $\alpha = 0$ and $w_1(n)$ fixed at -1. Since one weight

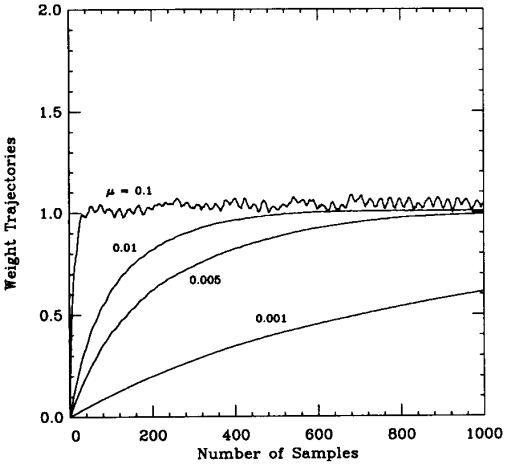

Fig. 3. Trajectories of $w_2(n)$ with $\alpha = 0$ and $w_1(n) = -1$.

[3]As a result, the weight fluctuations about W_* will be negligible and we can ignore them.

is fixed, the stationary points are unique in this case, corresponding to $w_2(n) \rightarrow 1$ [recall that we must have $w_2(n) = -w_1(n)$]. Observe that $w_2(n)$ converges to 1 as expected, and that the rate of convergence increases as μ is increased. The steady-state weight variance is greater for larger values of μ; this result is similar to that observed for the standard LMS algorithm, and it is a form of misadjustment [4]. Figure 4

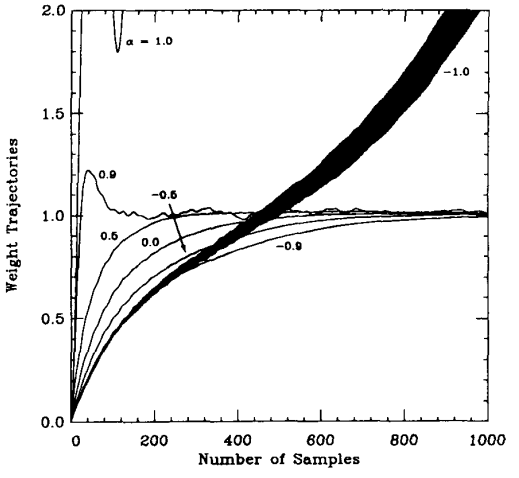

Fig. 4. Trajectories of $w_2(n)$ with $\mu = 0.01$ and $w_1(n) = -1$.

shows similar weight trajectories, except μ was kept fixed at 0.01 and α was varied for several positive and negative values. Observe that the rate of convergence increases as α is increased until it becomes unstable at $\alpha = 1$. On the other hand, the rate of convergence decreases as α becomes negative and it is again unstable when $\alpha = -1$. These results suggest that negative values of α would not be used even though the algorithm is stable.

Figures 5 and 6 show several weight trajectories of $w_2(n)$ for various values of μ and α, where $w_1(n)$ was also adapted. We show only the trajectories of $w_2(n)$ because we have found that $w_1(n) \approx -w_2(n)$ when we initialize them both to zero. In Fig. 5, observe that the weight trajectories are

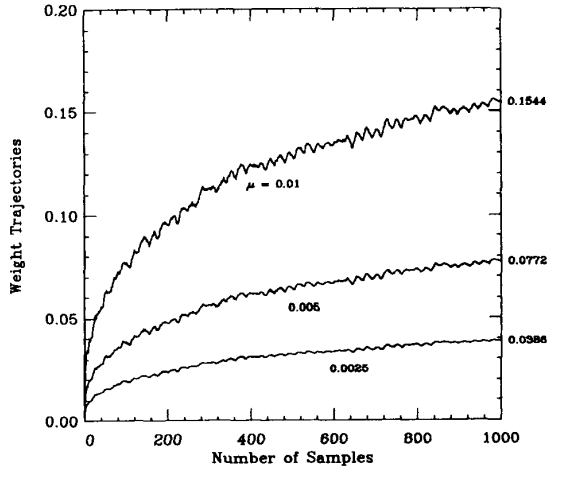

Fig. 5. Trajectories of $w_2(n)$ with $\alpha = 0$.

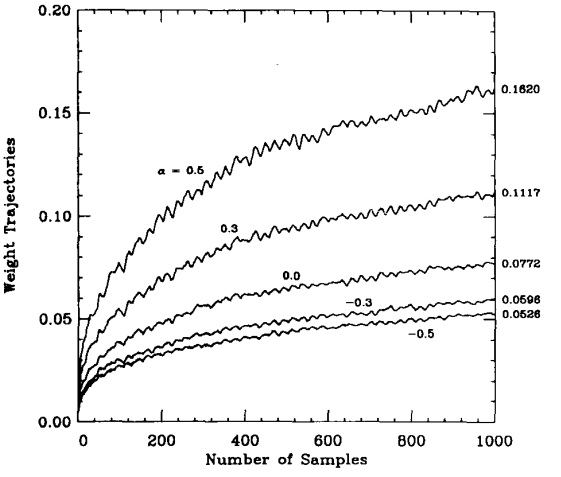

Fig. 6. Trajectories of $w_2(n)$ with $\mu = 0.005$.

directly proportional to changes in the step size μ, as predicted by the analysis in Section III and (31), and observe in Fig. 6 that they depend on α in a nonlinear way. (The weight value at iteration 1000 for each curve is shown to the right of the figures.) For a value of $\alpha = 0.5$, $w_2(n)$ should be scaled up by a factor of 2, and this result is verified by the simulation. On the other hand, the weights should be scaled down by a factor of 0.667 for $\alpha = -0.5$; this result is also verified in the simulation. A similar property is evident for $\alpha = \pm 0.3$, and we have observed the relationship predicted by (30) and (31) for other values of μ and α.

V. Conclusion

The stationary points and weight trajectories near convergence of a perceptron learning algorithm with momentum updating have been examined for a Gaussian input vector. It was demonstrated that the stationary points are not unique, and that the behavior of the algorithm near convergence depends on the step size μ and the momentum factor α, as well as the statistics of the underlying process $d(n)$. As μ is increased, the weight trajectories increase in direct proportion to changes in μ. On the other hand, the algorithm convergence properties depend on α in a nonlinear way, and it is unstable for $|\alpha| = 1$, as demonstrated by computer simulations.

Acknowledgment

The authors thank Neil Bershad for his helpful comments concerning the analysis of the steady-state output variance.

References

[1] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, pp. 4-22, Apr. 1987.

[2] B. Widrow, R. G. Winter, and R. A. Baxter, "Layered neural nets for pattern recognition," *IEEE Trans. Acoust., Speech, Sig. Proc.*, vol. 36, pp. 1109-1118, July 1988.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pp. 318-362, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: M.I.T. Press, 1986.

[4] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[5] J. J. Shynk and S. Roy, "Analysis of a perceptron learning algorithm with momentum updating," *Proc. IEEE Int. Conf. Acoust., Speech, Sig. Proc.*, Albuquerque, NM, Apr. 1990, pp. 1377–1380.
[6] J. J. Shynk and S. Roy, "The LMS algorithm with momentum updating," *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 2651–2654, Espoo, Finland, June 1988.
[7] S. Roy and J. J. Shynk, "Analysis of the momentum LMS algorithm," *IEEE Trans. Acoust., Speech, Sig. Proc.*, to be published.
[8] G. O. Stone, "An analysis of the delta rule and the learning of statistical associations," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pp. 444–459, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: M.I.T. Press, 1986.
[9] R. Price, "A useful theorem for nonlinear devices having Gaussian inputs," *IRE Trans. Inform. Theory*, vol. IT-4, pp. 69–72, June 1958.

John J. Shynk (Member, IEEE) was born in Lynn, MA, on June 20, 1956. He received the B.S. degree in systems engineering from Boston University, Boston, MA, in 1979. He received the M.S. degree in electrical engineering and in statistics and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1980, 1985, and 1987, respectively.

From 1979 to 1982, he was a Member of Technical Staff in the Data Communications Performance Group at AT&T Bell Laboratories, Holmdel, NJ, where he formulated performance models for voiceband data communications. He was a Research Assistant from 1982 to 1986 in the Electrical Engineering Department at Stanford University where he worked on frequency-domain implementations of adaptive IIR filter algorithms. From 1985 to 1986, he was also an Instructor at Stanford, teaching courses on digital signal processing and adaptive systems. Since 1987, he has been an Assistant Professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. He is also Associate Director of the Center for Information Processing Research. His current research interests include developing and analyzing efficient adaptive signal processing algorithms for applications in system identification, communications, adaptive array processing, and neural networks.

Dr. Shynk is an Associate Editor of the *IEEE Transactions on Acoustics, Speech, and Signal Processing*, and a member of Tau Beta Pi, Sigma Xi, and INNS.

Sumit Roy received the B. Tech. degree from the Indian Institute of Technology in 1983. He received the M.S. and Ph.D. degrees in electrical engineering from the University of California at Santa Barbara in 1985 and 1988, respectively, as well as the M.A. degree in applied probability and statistics in 1988.

He is an Assistant Professor at the Moore School of Electrical Engineering, University of Pennsylvania, engaged in research activities spanning the general area of statistical signal processing, with a particular emphasis on adaptive algorithms and their implementations for data communication systems, and on sensor array processing for radar, sonar, and underwater acoustics.