Communicated by Stephen Gallant

A "Thermal" Perceptron Learning Rule

Marcus Frean
Physiological Laboratory, Downing Street,
Cambridge CB2 3EG, England

The thermal perceptron is a simple extension to Rosenblatt's perceptron learning rule for training individual linear threshold units. It finds stable weights for nonseparable problems as well as separable ones. Experiments indicate that if a good initial setting for a temperature parameter, T_0, has been found, then the thermal perceptron outperforms the Pocket algorithm and methods based on gradient descent. The learning rule stabilizes the weights (learns) over a fixed training period. For separable problems it finds separating weights much more quickly than the usual rules.

1 Introduction

This paper is about a learning rule for a linear threshold unit, often called a perceptron. This is a unit connected by variable weights to a set of inputs across which patterns occur. In the following the ith element of an input pattern (a binary or real number) is denoted by ξ_i, and its associated weight by W_i. In response to a given pattern, a perceptron goes into one of two output states given by

$$o = \begin{cases} 1 \quad (\text{ON}) & \text{if } \phi > 0 \\ 0 \quad (\text{OFF}) & \text{otherwise} \end{cases} \tag{1.1}$$

where

$$\phi = \sum_{i=0}^{N} W_i \xi_i$$

The extra input ξ_0, taken to be 1 for every pattern, is commonly included as it enables the zeroth weight to act as a bias. The learning task for such units is to find weights such that the perceptron's output o matches a desired output t for each input pattern $\boldsymbol{\xi}$.

On presentation of the μth input pattern the perceptron learning rule (Rosenblatt 1962), henceforth abbreviated to PLR, alters the weights in the following way:

$$\textbf{PLR}: \quad \Delta W_i^\mu = \alpha\,(t^\mu - o^\mu)\xi_i^\mu$$

where α is a positive constant.[1] For instance, if the output o elicited by a given input pattern is 0 whereas t is 1, this rule alters weights to increase the value of ϕ. If repeated this would eventually result in the output becoming 1. The perceptron convergence theorem (Block *et al.* 1962, Minsky and Papert 1969) states that *if* a set of weights exists for which the perceptron makes no errors, the PLR will converge on a (possibly different) set making no errors after only a finite number of pattern presentations. Hence perceptrons can learn any classification for which appropriate weights exist: such training sets are termed "linearly separable."

A far more common, and in many ways more interesting, task is to find a set of weights that gives a minimal number of errors in the case where perfect weights do not exist. Clearly if the PLR is used in such cases, the weights are never stable since they change every time an error is made. Neither are they "good on average," as will be seen in Section 4. The task of finding stable weights that engender a small number of errors in this "nonseparable" case is addressed in this paper.

2 The Thermal Perceptron Rule

One rationale goes as follows: the trouble is that the PLR does the same thing for *every* error made. Instead, the benefit from improving the value of ϕ elicited by a given input pattern should be tempered by the possibility that the new weights now misclassify patterns they previously got right. Since the change in ϕ due to the effect of the PLR is independent of the value itself (other than its sign), an error with a large associated ϕ is less likely to be corrected in this step than an error where ϕ is small. Conversely the weight changes that would be necessary to correct a large error are themselves large, and hence much more likely to corrupt the existing correct responses of the unit.

Ideally the weight changes made should be biased toward correcting errors for which ϕ is close to zero. A simple way to do this is to make the weights changes given by Rosenblatt's PLR tail off exponentially with $|\phi|$:

$$\textbf{Thermal PLR:} \quad \Delta^{\mu} W_i = \alpha \, (t^{\mu} - o^{\mu}) \, \xi_i^{\mu} \, e^{-|\phi|/T}$$

The parameter T controls how strongly the changes are attenuated for large $|\phi|$. T is somewhat analogous to a temperature: at high T the PLR is recovered since the exponential becomes almost unity for any input, whereas at low T there are no appreciable changes unless ϕ is close to zero. If T is very small the weights are frozen at their current values.

A natural extension is to anneal this effect by gradually reducing the temperature from high T, where the usual PLR behavior is seen, toward

[1]Note that if weights are initially all zero, then the magnitude of α is irrelevant because of the threshold function.

zero. This gradual freezing is particularly desirable because it stabilizes the weights in a natural way over a finite learning period.

If T is held constant, convergence of the thermal PLR can be deduced (Frean 1990b) from the perceptron convergence theorem. That is, Minsky and Papert's proof can be modified to hold where weight changes have the correct sign and use a positive step size, α_t, chosen arbitrarily between bounds $0 < a \leq \alpha_t \leq b$.

One picture of the way this rule works is given by considering the patterns as points in a space in which each dimension corresponds to the activity of a particular input line. Points in this space at which the perceptron's output changes define a *decision surface* that (from equation 1.1) is a hyperplane perpendicular to the weights vector in this space. In the usual PLR, this hyperplane moves by approximately the same amount whenever there is an error, whereas in the thermal PLR it moves by an amount that is appreciable only if the pattern causing the error is close to the hyperplane (i.e., where $|\phi|$ is small). As an approximation one can imagine a zone immediately to either side of the hyperplane within which an error will cause it to move. The perceptron will be stable if there are no errors occurring in this zone. The annealing of the temperature is then the gradual reduction of the extent of the "sensitive" zone. In the limit of $T \to 0$ the zone disappears altogether and the perceptron is stable.

As will be seen, best results are obtained if α is reduced from 1 to 0 at the same time as T is. In terms of the simplified picture above, this means that we reduce the amount the hyperplane moves in reaction to a pattern within the zone near the hyperplane in proportion to the size of the zone.

3 Other Methods

The two established ways of learning the weights for a threshold perceptron are reviewed here. In the next section the thermal rule is compared with these methods.

3.1 The Pocket Algorithm. A simple extension of perceptron learning for nonseparable problems called the *Pocket algorithm* (Gallant 1986a) suffices to make the PLR well behaved, in the sense that weights that minimize the number of errors can be found. This method is currently the preferred one for learning weights in threshold perceptrons, and is used in a number of applications. The Pocket algorithm consists of applying the PLR with a randomly ordered presentation of patterns, but also keeping a copy of a second set of weights "in your pocket." If the patterns are presented in random order, good sets of weights have a lower probability of being altered at each presentation, and therefore tend to remain unchanged for a longer time than sets of weights that engender

more errors. Whenever a set of perceptron weights lasts longer than the current longest run, they are copied into the pocket. As the training time increases, the pocketed weights give the minimum possible number of errors with a probability approaching unity. That is, if a solution giving say p or fewer errors exists then the Pocket algorithm can, in theory, be used to find it. Note that if the patterns are linearly separable, the algorithm reduces to the usual PLR. Gallant (1986a) notes that the Pocket algorithm outperforms a standard technique [Wilks' method, in (SPSS-X 1984)] by about 20% for a set of 15 learning problems.

In this form the algorithm improves the pocketed weights in an entirely stochastic fashion—there is nothing to prevent a good set of weights being overwritten by an occasional long run of successes involving bad weights. The "ratchet" version of the Pocket algorithm (Gallant 1990) ensures that every time the pocket weights change, they actually improve the number of errors made. Whenever a set of weights is about to be copied into the pocket, the actual number of errors these weights engender if every pattern in the training set were to be applied to the input is compared with the same number for the existing pocket weights. The weights in the pocket are replaced only if the actual number of errors is lower.

The main strength of the Pocket algorithm is the fact that optimal weights are found with probability approaching unity, given sufficient training time. Unfortunately there is no bound known for the training time actually required to achieve a given level of performance. Moreover, in practice the weights do not improve much beyond the first few cycles through the training set. Although the weights so obtained are better than many other methods, they still tend to make many more errors than an optimal set would make. To get better weights from the same procedure the "ratchet" is required. A potential disadvantage of this is that the computational cost is greatly increased in cases where there are a lot of weight vectors that are almost as good as those in the pocket (i.e., where the ratchet is likely to be invoked frequently).

3.2 Methods Based on Gradient Descent. Another way to generate weights for a threshold perceptron is to learn them by a gradient descent procedure. To do this requires adopting some differentiable function such as the "sigmoid" or "logistic" function, which approximates the perceptron's output function:

$$y = 1/(1 + e^{-\phi})$$

Gradient descent attempts to minimize the total error attributable to the unit:

$$E = \sum_{\mu} E^{\mu}$$

where E^μ is some measure of how well y^μ matches the desired output t^μ for the μth input pattern. This error is reduced by moving down an estimate of the gradient of the error (namely that of E^μ rather than E) with respect to $\mathbf{W}$:

$$\Delta^\mu W_i = -\alpha \frac{\partial E^\mu}{\partial W_i}$$

This approximates true gradient descent of E provided α is very small (typically ~ 0.001). One common form used for E^μ is the *squared* error

$$E^\mu = (t^\mu - y^\mu)^2$$

from classical statistics, giving rise to delta rule or "least-mean-squared" (*LMS*) methods (Widrow and Hoff 1960). This gives the following rule:

LMS: $\Delta^\mu W_i = \alpha\ (t^\mu - y^\mu) y^\mu (1 - y^\mu)\ \xi_i^\mu$

Another form is the *cross-entropy* error measure

$$E^\mu = t^\mu \log_2 y^\mu\ +\ (1 - t^\mu) \log_2 (1 - y^\mu)$$

from information theory. Gradient descent of this measure gives rise to a different rule (Hinton 1989):

Cross-entropy: $\Delta^\mu W_i = \alpha\ (t^\mu - y^\mu)\ \xi_i^\mu$

which is just the PLR except that the sigmoid y^μ replaces the threshold o^μ.

4 Simulations

Simulations were performed comparing the thermal PLR, Pocket algorithm, LMS, and cross-entropy on four classification tasks. The training sets in these tasks vary from being highly nonseparable to linearly separable.

For the thermal rule both T and α are reduced linearly from their starting values (T_0 and 1, respectively) to zero over the stated training time, measured in epochs. One epoch consists of the presentation of P patterns chosen at random (with replacement) from a training set of size P. The simulations also show the other three combinations of annealing T and α. For the purposes of comparison, the optimal value of α for LMS and cross-entropy was found by exhaustive search for each task. This optimal value is used to compare these methods with the thermal rule.

A notable advantage of the Pocket algorithm is that it requires no such tuning of parameters. For a given type of problem, finding a good value of T_0 for the thermal rule can take considerable time. The same is true of α in the LMS and cross-entropy rules, although the exact choice is typically less crucial for these rules. It is important to note that improvements in performance are obtained at the cost of this preliminary parameter search.

Regarding actual computational speeds, all the algorithms discussed here have similar run times except for the ratchet version of the Pocket algorithm. The Pocket algorithm has the advantage that it requires only integer operations whereas the other methods must calculate exponentials in order to alter the weights. On the other hand it must occasionally copy weights to the pocket as well as change those actually in use. In these simulations, run times[2] of the thermal rule and the Pocket algorithm (without ratchet) never differ by more than 10%. Also note that the Pocket algorithm and thermal PLR calculate and make weight changes only if there is an error, whereas the rules based on gradient descent do this for every pattern presentation. This effect shows up particularly for "easy" problems, where the Pocket algorithm and thermal rule alter their weights relatively infrequently. At most this resulted in a 50% slowing of the gradient descent rules relative to the others, for the problems reported here.

4.1 Randomly Targeted Binary Patterns. In this task all 1024 of the binary patterns across 10 inputs are used, and each is assigned a target 0 or 1 with equal probability. Clearly this problem is highly nonseparable. Figures 1 and 2 show the performance of the various rules on this problem versus the initial temperature, T_0, used in the thermal PLR. Each trial consists of generating a random training set and, starting with zero weights, training a perceptron for 100 epochs. The performance is simply the proportion of patterns classified correctly to the total number of patterns. For the gradient descent methods the optimal value of α for this problem and this training period is 10^{-3}. The ratchet version of the Pocket algorithm takes on average 6 times longer for this problem. Standard deviations are of the order of 0.02 for all points and all algorithms. Note that the PLR corresponds to the "no annealing" curve at high T_0, which shows poor performance.

4.2 Discriminating Two Overlapping Gaussian Clusters. In this experiment the training sets consist of 100 real-valued patterns across 10 inputs. To generate a training set, two points inside the unit hypercube were chosen at random, and rescaled to be a unit distance apart. These were taken to be the centers of two gaussian probability distributions of variance 2. 50 patterns whose target output is 1 were generated at random from the first such distribution, and 50 patterns with target 0 from the second. For LMS and cross-entropy the optimal value of α was found to be 0.5 and 0.001 respectively. Figure 3 shows the performance of each method on this task. Both LMS and the thermal PLR do much better than the Pocket algorithm. The ratchet slows down the Pocket algorithm by three times for this problem but significantly improves performance.

[2]Note however that some implementations may exploit the advantages of integer arithmetic available to the Pocket algorithm to a greater extent than that used here.

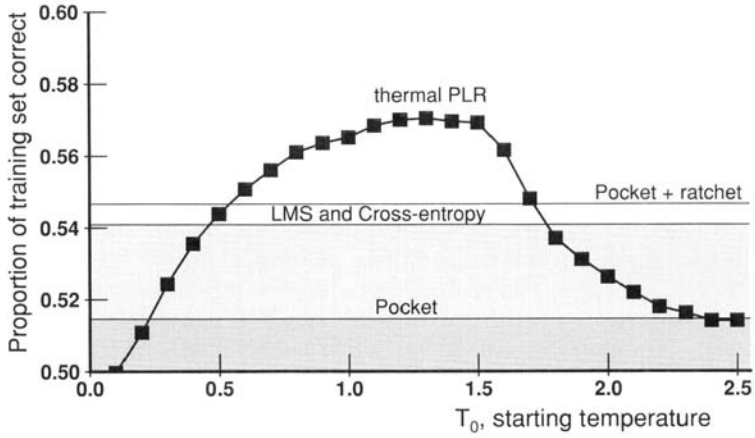

Figure 1: Performance on the random targets problem versus starting temperature. Half the patterns are target 1, so a unit with zero valued weights gets 50% of the patterns correct by default. Each point is the mean of 1000 independent trials, each on a different training set. The levels attained by gradient descent methods and the Pocket algorithm are indicated by the shaded regions. The two gradient descent rules achieve virtually the same performance.

4.3 Convergence Time for Separable Training Sets. Where the training set is linearly separable the question is how quickly a set of separating weights can be found. Again all 1024 binary patterns across 10 inputs were included. The targets are then defined by inventing a perceptron with random weights, and adjusting its bias so that exactly half the total number of patterns are ON. Figure 4 shows how many trials out of 100 converged, where each algorithm was run for the indicated number of iterations. T_0 for the thermal rule is set at 1.5 here, but values very much higher (e.g., $T_0 = 100$) are equally good. The value of α used for LMS and cross-entropy is 0.1 which is the optimal value for LMS. If α is very much larger than this, the cross-entropy rule effectively approaches the PLR. This is because the weights are correspondingly large, so the sigmoid is almost always in its "saturated" range, mimicking a threshold unit.

If there is no annealing, the thermal and Rosenblatt's PLR (and hence the Pocket algorithm) have very similar performance. Annealing results in a striking improvement (an order of magnitude) in the speed of convergence over the other methods.

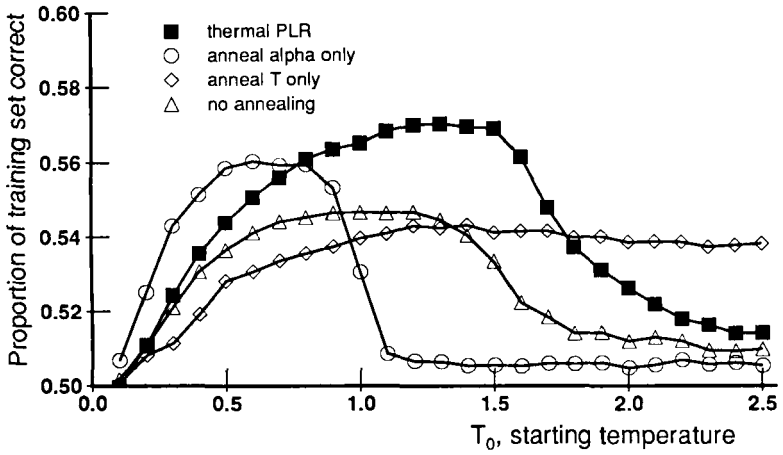

Figure 2: Performance on the random targets problem, for the various combinations of annealing, versus starting temperature.

4.4 Nearly Separable Training Sets: Dealing with Outliers. In many problems of interest, the patterns would be linearly separable were it not for a small number of outliers that prevent the PLR from converging. In such cases we want to ignore the outliers and find the weights that would correctly classify the linearly separable patterns.

Training sets for this example were generated in the following way: first, a linearly separable training set is produced as above. Then some small number n of the patterns are chosen at random and their targets flipped from 1 to 0 or vice versa. Hence the problem would be linearly separable were it not for these patterns,[3] and a good learning rule should produce units giving at most n errors. Parameter values are the same as in the separable case above. Results are shown in Figure 5. Inclusion of the ratchet slows the Pocket algorithm by an average 300 times (for $n = 1$), indicating that a great many long runs with suboptimal weights are occurring. Note that the thermal rule performs slightly *better* than the level attainable by ignoring the flipped patterns, indicating that it occasionally finds a better set of weights than those originally used to generate the patterns. Although T_0 is 1.5 here, virtually the same performance is obtained for starting temperatures up to $T_0 = 30$.

[3]If n is small the problem may still be linearly separable: such cases were rejected.

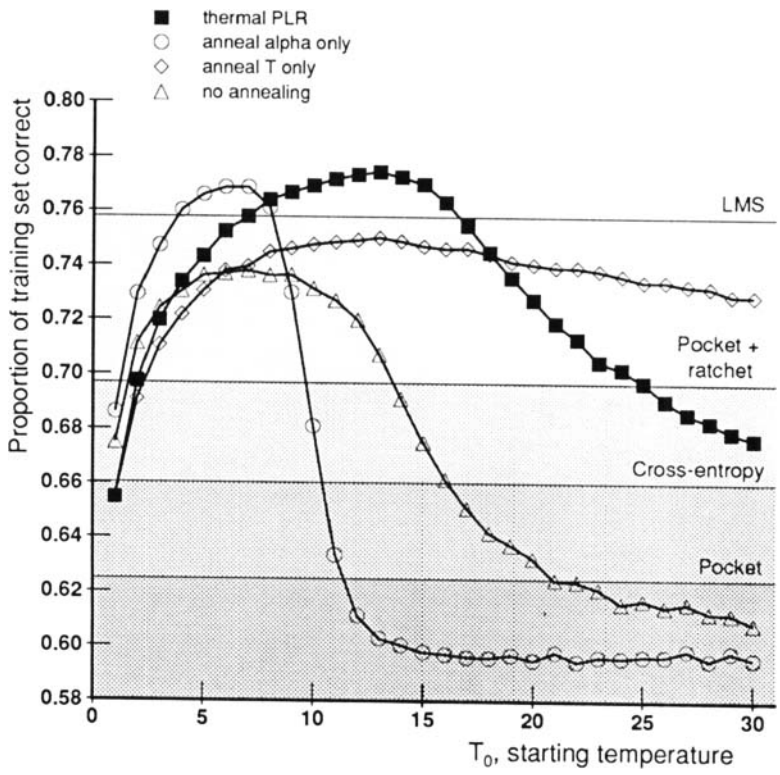

Figure 3: Performance on the overlapping gaussians problem versus starting temperature. Performance is the proportion of the training set correctly classified after 100 epochs. Each point is an average over 1000 trials. Standard deviations are of the order of 0.04 for all points. The mean performance of the PLR on this problem is 0.588.

5 Discussion

The thermal PLR, which is closely related to the classical PLR, outperforms both the Pocket algorithm and methods based on gradient descent in terms of efficiently generating weights that give a small number of errors in a threshold perceptron. If the patterns are linearly separable, perfect weights are located quickly. In addition it produces stable weights in a given training time.

It is interesting to compare the rationale behind the thermal rule with that of the LMS procedures. In the case of the thermal rule, it is argued

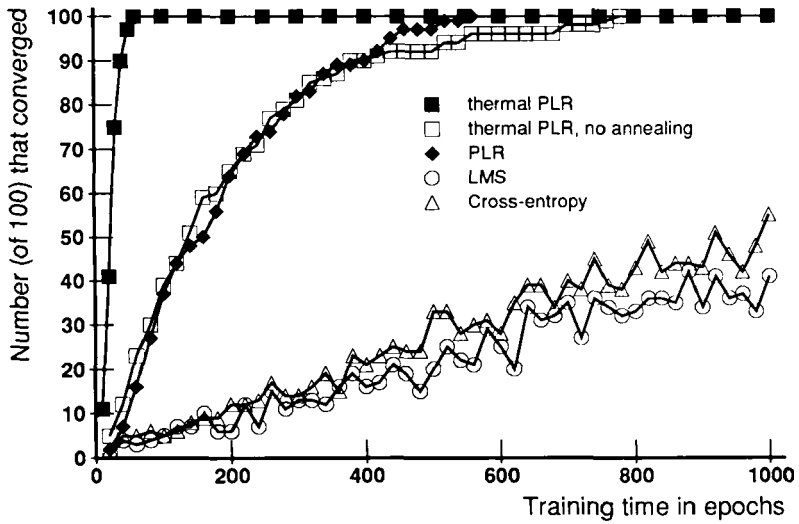

Figure 4: The graph shows how many out of 100 independent trials (each on a different training set) converged to zero errors after the training time denoted by the abscissa. For example, after 200 epochs 60% of runs using the PLR had converged.

that large errors (in ϕ) should be penalized lightly, since endeavoring to correct these errors means corrupting existing weights to a large degree. In LMS, large errors are penalized more heavily than small ones, since these large errors contribute proportionally more to the quantity being minimized (the sum of squared errors). Hence these two approaches have opposite motivations.

The twin constraints of stability and optimality of weights are of particular significance for *constructive algorithms*, in which perceptrons are added one at a time, enabling eventual convergence of the whole network to zero errors. At present the Pocket algorithm is the procedure of choice for most methods of this type (e.g., Gallant 1986b; Mèzard and Nadal 1989; Nadal 1989; Golea and Marchand 1990). The thermal rule has been applied to this type of algorithm (Frean 1990a; Burgess *et al.* 1991; Martinez and Estève 1992), and can dramatically reduce the size of the networks so produced (by between 2 and 5 times for a range of problems), resulting in better generalization and computational efficiency (Frean 1990b). In addition more difficult problems may now be successfully tackled with the same constructive algorithms.

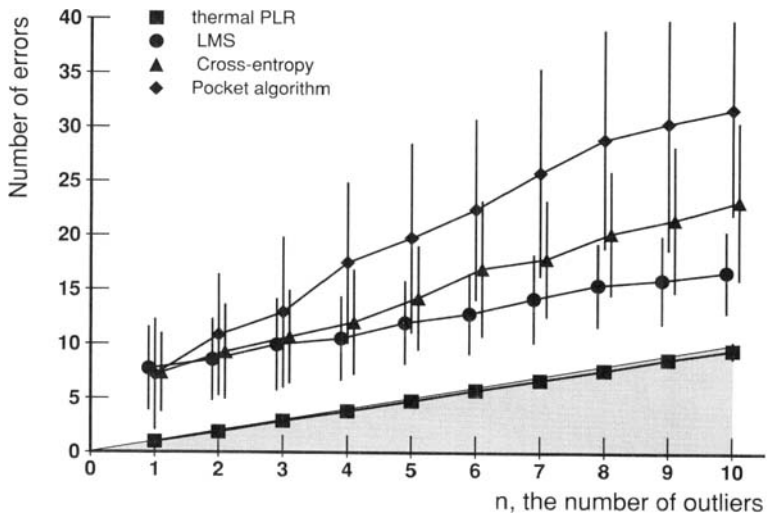

Figure 5: The number of errors made after 100 epochs is plotted against the number of patterns whose targets were flipped from an initially separable training set. For instance, if a single target is flipped the thermal PLR always makes one error whereas the other methods make ~ 8 errors. In the shaded region the number of errors is less than the number of outliers. Each point is the average of 200 trials, each on a different training set. Where not shown the error bars are smaller than the points.

Acknowledgments

The author would like to thank Peter Dayan, David Willshaw, and Jay Buckingham for their helpful advice in the preparation of this paper.

References

Block, H. D., Knight, B. W. Jr, and Rosenblatt, F. 1962. Analysis of a four-layer series-coupled perceptron (II). *Rev. Modern Phys.* **34**(1), 135–142.

Burgess, N., Granieri, M. N., and Patarnello, S. 1991. 3-D object classification: Application of a constructor algorithm. *Int. J. Neural Syst.* **2**(4), 275–282.

Frean, M. R. 1990a. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Comp.* **2**(2), 198–209.

Frean, M. R. 1990b. Small nets and short paths: Optimising neural computation. Ph.D. thesis, University of Edinburgh.

Gallant, S. I. 1986a. Optimal linear discriminants. *IEEE Proc. 8th Conf. Pattern Recognition, Paris.*

Gallant, S. I. 1986b. Three constructive algorithms for network learning. *Proc. 8th Annual Conf. Cognitive Sci. Soc.*

Gallant, S. I. 1990. Perceptron-based learning algorithms. *IEEE Transact. Neural Networks* **1**(2), 179–192.

Golea, M., and Marchand, M. 1990. A growth algorithm for neural network decision trees. *Europhys. Lett.* **12**(3), 205–210.

Hinton, G. E. 1989. Connectionist learning procedures. *Artificial Intelligence* **40**, 185–234.

Martinez, D., and Estève, D. 1992. The offset algorithm: Building and learning method for multilayer neural networks. *Europhysics Lett.* **18**(2), 95–100.

Mèzard, M., and Nadal, J-P. 1989. Learning in feedforward layered networks: The tiling algorithm. *J. Phys. A*, **22**(12), 2191–2203.

Minsky, M., and Papert, S. 1969. *Perceptrons*. The MIT Press, Cambridge, MA.

Nadal, J-P. 1989. Study of a growth algorithm for neural networks. *Int. J. Neural Syst.* **1**(1), 55–59.

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

Widrow, B., and Hoff, M. E. 1960. Adaptive switching circuits. IRE WESCON Convention Record, New York: IRE, 96–104.

Received 12 August 1991; accepted 24 April 1992.