PARSIMONIOUS SIDE PROPAGATION

P. S. Bradley & O. L. Mangasarian

Computer Sciences Department, University of Wisconsin
1210 West Dayton Street, Madison, WI 53706
paulb@cs.wisc.edu, olvi@cs.wisc.edu

ABSTRACT

A fast parsimonious linear-programming-based algorithm for training neural networks is proposed that suppresses redundant features while using a minimal number of hidden units. This is achieved by propagating sideways to newly added hidden units the task of separating successive groups of unclassified points. Computational results show an improvement of 26.53% and 19.76% in tenfold cross-validation test correctness over a parsimonious perceptron on two publicly available datasets.

We consider the problem of determining the weights and thresholds of a neural network to discriminate between the elements of two disjoint sets $\mathcal{A}$ and $\mathcal{B}$ in n-dimensional real space R^n. We present a parsimonious, linear-programming-based approach to determine a minimal number of hidden units and weight vectors with a minimal number of nonzero components so that the neural network achieves a prescribed degree of accuracy on the training data. Using tenfold cross-validation we compare neural networks trained by this method with a parsimonious perceptron (i.e. trained with a minimal number of nonzero weights) to discriminate between points in $\mathcal{A}$ and $\mathcal{B}$.

A word about our notation. All vectors will be column vectors unless transposed to a row vector by a superscript T. The scalar product of two vectors x and y in R^n will be denoted by x^Ty. The notation $A \in R^{m\times n}$ will represent an $m \times n$ real matrix A, A_i will denote the ith row of A and A_{ij} will denote the element in row i and column j. A vector of ones of arbitrary dimension will be denoted by e. The base of the natural logarithm will be denoted by ε, and for $y \in R^m$, ε^{-y} will denote a vector in R^m with components ε^{-y_i}, $i = 1, \ldots, m$. The notation $\arg\min\limits_{x\in S} f(x)$ will denote the set of minimizers of $f(x)$ on the set S. For a vector $x \in R^n$, $x_* \in R^n$ is the step function applied to each component of x with $(x_*)_i = 1$ when $x_i > 0$ and $(x_*)_i = 0$ when $x_i \leq 0$, $i = 0, \ldots, n$. By a separating plane, with respect to two given point sets $\mathcal{A}$ and $\mathcal{B}$ in R^n, we shall mean a plane that attempts to separate R^n into two half spaces such that each open halfspace contains points mostly of $\mathcal{A}$ or $\mathcal{B}$. Alternatively, such a plane can also be interpreted as a classical perceptron [3, 4]. Tenfold cross-validation refers to the re-sampling method which successively removes 10% of the available data for testing a separator generated with the remaining 90%. For a point set $\mathcal{A} \in R^n$, card($\mathcal{A}$) will denote the cardinality of $\mathcal{A}$, the number of points of $\mathcal{A}$.

1. PARSIMONIOUS SIDE PROPAGATION

The Parsimonious Side Propagation algorithm trains a neural network with a minimal number of hidden units, each unit utilizing a minimal number of problem features, to classify an n-dimensional input vector as belonging to either of two finite point sets $\mathcal{A}$ or $\mathcal{B}$ in R^n. The m training points of $\mathcal{A}$ are represented by the matrix $A \in R^{m\times n}$ and the k training points of $\mathcal{B}$ are represented by the matrix $B \in R^{k\times n}$. We initially consider the set $\mathcal{A} \cup \mathcal{B}$ "unclassified". In the notation of our algorithm, we set $\mathcal{A}^1 := \mathcal{A}$ and $\mathcal{B}^1 := \mathcal{B}$.

In training the hidden units, we begin with the entire training data $\mathcal{A} \cup \mathcal{B}$ and compute a plane $\{x|w^Tx = \gamma\}$ utilizing a minimum number of the n problem features (i.e. setting as many elements of $w \in R^n$ to zero), while attempting to separate $\mathcal{A}$ from $\mathcal{B}$ to the extent possible. The values of w and $\gamma \in R$ defining this separating plane are obtained by solving the following optimization problem by a linear-programming-based Successive Linearization Algorithm [2, Algorithm 3.1]:

$$(w,\gamma,y,z,v) \in \arg\min_{T}(1-\lambda)\left(\frac{e^Ty}{m}+\frac{e^Tz}{k}\right)+\lambda(n-e^T\varepsilon^{-\alpha v}),$$
$$T := \left\{ (w,\gamma,y,z,v) \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \leq y, \\ Bw - e\gamma + e \leq z, \\ y \geq 0, z \geq 0, \\ -v \leq w \leq v \end{array} \right\}. \qquad (1)$$

The number of features utilized by the separating plane (i.e. the number of nonzero elements of w) is determined by the feature suppression parameter $\lambda \in [0, 1]$. When $\lambda = 0$, no features are suppressed while attempting to separate $\mathcal{A}$ and $\mathcal{B}$. When $\lambda = 1$, w is totally suppressed to a useless zero solution. Thus λ is taken in the open interval $(0, 1)$, while the smoothing parameter α is typically set to 5, so that the smooth exponential $1 - \varepsilon^{-\alpha x}$ approximates fairly accurately the step function $x_* \in R$ on the nonnegative real line. For $\lambda = 0$, the error in separating the training data is minimized, but our goal is not to produce a classifier with minimum error on the the training data, but we wish to determine the classifier (neural network) to perform with low error on future unseen data. Classification error on unseen data, or generalization error, is minimized for solutions of (1) with $1 > \lambda > 0$ [2].

The separating plane computed by (1) attempts to put $\mathcal{A}$ in the open halfspace $\{x|w^Tx > \gamma\}$ and put $\mathcal{B}$ in the open halfspace $\{x|w^Tx < \gamma\}$ and thus can be used to generate a parsimonious perceptron classification function $(w^Tx - \gamma)_*$ which is 1 if $x \in \mathcal{A}$ and 0 otherwise.

This work was supported by National Science Foundation Grant CCR-9322479 and Air Force Office of Scientific Research Grant F49620-97-1-0326 as Mathematical Programming Technical Report 97-11, October 1997.

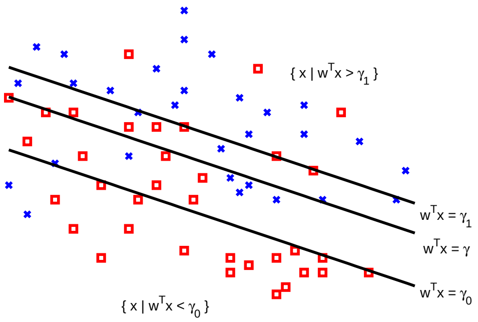

Figure 1: Separating Planes: Since the separating plane $\{x \mid w^T x = \gamma\}$ does not satisfactorily separatate $\mathcal{A}$ (**x**) from $\mathcal{B}$ (□), we generate two planes parallel to it so that $\{x \mid w^T x > \gamma_1\}$ contains mostly points of $\mathcal{A}$ or $\mathcal{B}$ (here $\mathcal{A}$) and $\{x \mid w^T x < \gamma_0\}$ contains mostly points of $\mathcal{A}$ or $\mathcal{B}$ (here $\mathcal{B}$).

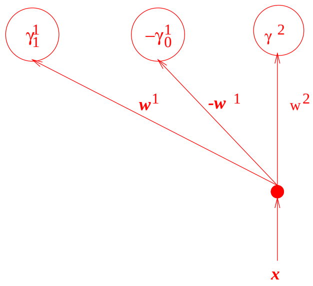

Figure 2: Hidden Units from Separating Planes: The first two hidden units of the neural network, corresponding to the planes $\{x \mid w^T x = \gamma^1\}$ and $\{x \mid w^T x = \gamma^0\}$ respectively, have respective weight vectors $w^1 = w$, $-w^1 = -w$, and thresholds $\gamma_1^1 = \gamma_1$ & $-\gamma_0^1 = -\gamma_0$. The third hidden unit with weight vector w^2 and threshold γ_1^2 corresponds to the plane $\{x \mid (w^2)^T x = \gamma^2\}$ depicted in Figure 3.

If the single plane generated by (1) is able to discriminate between $\mathcal{A}^1 = \mathcal{A}$ and $\mathcal{B}^1 = \mathcal{B}$ above a pre-defined acceptable accuracy, we add one hidden unit to the neural network with weight vector $w^1 = w$ and threshold $\gamma^1 = \gamma$ and proceed to training the output unit.

From here on, to follow the indexing in the algorithm, we set $w^1 = w$ and $\gamma^1 = \gamma$. If the accuracy tolerance is not achieved, we use the solution (w^1, γ^1) as a starting point to determine two open halfspaces, $\mathcal{H}_1^1 := \{x | (w^1)^T x > \gamma_1^1\}$ and $\mathcal{H}_0^1 := \{x | (w^1)^T x < \gamma_0^1\}$, with $\gamma_1^1 > \gamma_0^1$ by moving the plane $w^T x = \gamma$ parallel to itself. The value of γ_1^1 is determined so that the *purity* of the halfspace $\mathcal{H}_1^1$ is above the accuracy tolerance and the number of points of $\mathcal{A}^1 \cup \mathcal{B}^1$ in $\mathcal{H}_1^1$ is maximal (purity is defined below in Definition 1.1). Similarly, the value of γ_0^1 is determined so that the purity of $\mathcal{H}_0^1$ is above the accuracy tolerance and the number of points of $\mathcal{A}^1 \cup \mathcal{B}^1$ is maximal. See Figure 1. (Such a construction was first proposed in [1] using a different separation criterion that is determined by the worst error, without feature suppression and letting the sets $\mathcal{A}$ and $\mathcal{B}$ fall only in prescribed halfspaces.)

Definition 1.1 Purity. *Let $\mathcal{H}$ be a halfspace. The purity of $\mathcal{H}$ with respect to $\mathcal{A}$ and $\mathcal{B}$ is the ratio of the maximum number of points of $\mathcal{A}$ in $\mathcal{H}$ and the number of points of $\mathcal{B}$ in $\mathcal{H}$ to the total number of data points in $\mathcal{H}$. Specifically:*

$$purity(\mathcal{H}, \mathcal{A}, \mathcal{B}) := \frac{\max(card(\mathcal{A} \cap \mathcal{H}), card(\mathcal{B} \cap \mathcal{H}))}{card(\mathcal{H} \cap (\mathcal{A} \cup \mathcal{B}))}. \quad (2)$$

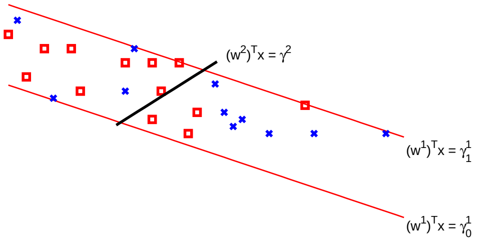

Figure 3: Three Separating Planes: With the points "correctly" classified by the first 2 parallel planes discarded, we attempt to correctly classify the remaining points with the separating plane $\{x \mid (w^2)^T x = \gamma^2\}$.

The crux of the algorithm is this. Move the plane $w^T x = \gamma$ parallel to itself to create two halfspaces $\mathcal{H}_1^1$ and $\mathcal{H}_0^1$ each containing mostly points of $\mathcal{A}$ or $\mathcal{B}$ to a certain desired purity tolerance. Note that *both* halfspaces are allowed to contain mostly points of the *same* set, that is $\mathcal{A}$ and $\mathcal{A}$, or $\mathcal{B}$ and $\mathcal{B}$, or *different* sets $\mathcal{A}$ and $\mathcal{B}$. Once this is achieved we throw away the points contained in these two halfspaces as being classified. The points that we still wish to consider further (i.e. the training data remaining "unclassified") lie between the two halfspaces and are:

$$\begin{aligned} \mathcal{A}^2 &:= \mathcal{A}^1 \cap \{x | (w^1)^T x \geq \gamma_0^1, (w^1)^T x \leq \gamma_1^1\}, \\ \mathcal{B}^2 &:= \mathcal{B}^1 \cap \{x | (w^1)^T x \geq \gamma_0^1, (w^1)^T x \leq \gamma_1^1\}. \end{aligned} \quad (3)$$

We represent these "unclassified" points by the matrices A^2 and B^2. The set $\mathcal{A}^2 \cup \mathcal{B}^2$ are those points of $\mathcal{A} \cup \mathcal{B}$ on or between the parallel planes $\{x | (w^1)^T x = \gamma_1^1\}$ and $\{x | (w^1)^T x = \gamma_0^1\}$ that have remained after the removal of the points of $\mathcal{A} \cup \mathcal{B}$ on the outside of these two planes. See Figures 1 and 3.

At this point we attempt to separate "unclassified" data by a plane $\{x | (w^2)^T x = \gamma^2\}$ obtained by solving (1) with A replaced by A^2, B replaced by B^2, m replaced by the number of points in $\mathcal{A}^2$ and k replaced by the number of points in $\mathcal{B}^2$. See Figure 3.

We now add three hidden units to the neural network. See Figure 2. The first hidden unit has weight vector w^1 and threshold γ_1^1. If this hidden unit "fires" for some input vector, this implies that $((w^1)^T x - \gamma_1^1)_* = 1$ which implies that $(w^1)^T x > \gamma_1^1$, which indicates that $x \in \mathcal{H}_1^1$. Since $\mathcal{H}_1^1$ was constructed to be a "pure" halfspace, we can correctly classify x. The second hidden unit has weight vector $-w^1$ and threshold $-\gamma_0^1$. Similarly, if this hidden unit "fires" for a given input vector, we conclude that this input vector is in $\mathcal{H}_0^1$ and can be correctly classified. The third hidden unit is added with weight vector w^2 and threshold γ^2. If neither of the first two hidden units "fire", then this hidden unit will attempt to correctly classify the given input vector.

At this stage we have either added one hidden unit, in which case the plane separating the "unclassified" training data performs with accuracy above our predefined tolerance. Otherwise, we have to determine two parallel planes defining two "pure" halfspaces, remove the points "classified" by these two parallel planes, and attempt to classify the remaining data by a single plane. In this case, three hidden units are added to the neural network.

We now need to train the output unit of the neural network constructed. If this resulting network performs with acceptable accuracy on the training data, the Parsimonious Side Propagation algorithm terminates. If not, the algorithm continues to iterate, adding more hidden units.

We change notation and let $\{\hat{w}^1, \ldots, \hat{w}^h\}$ and $\{\hat{\gamma}^1, \ldots, \hat{\gamma}^h\}$ be the h hidden unit weight vectors and thresholds computed up to this point. We define the output of hidden unit ℓ for a given input vector $x \in R^n$ as $((\hat{w}^\ell)^T x - \hat{\gamma}^\ell)_*$. Thus, the h hidden units map an input vector $x \in R^n$ to a vertex of the unit cube in R^h. The problem of training the output unit of the neural network to classify $\mathcal{A}$ and $\mathcal{B}$ reduces to separating the vertices of the cube to which points of $\mathcal{A}$ are mapped from those vertices to which $\mathcal{B}$ are mapped [4]. We define the $m \times h$ matrix of ones and and zeros $H_A \in \{0,1\}^{m \times h}$ where $(H_A)_{ij}$ is the output of hidden unit j on data point i of $\mathcal{A}$. Similarly, $H_B \in \{0,1\}^{k \times h}$ where $(H_B)_{ij}$ is the output of hidden unit j on data point i of $\mathcal{B}$.

We compute two candidate weight vectors v^1 and v^2 and thresholds τ^1 and τ^2 for the output unit and choose that which performs with maximal accuracy. For the first, we calculate a separating plane $\{u \in R^h | (v^1)^T u = \tilde{\tau}^1\}$ to separate the points of H_A from those of H_B. This plane is calculated by solving (1) with $\lambda = 0$ (emphasizing separation) and A replaced by H_A and B replaced by H_B. We then translate the plane (i.e. vary $\tilde{\tau}^1$) to minimize the number of points of H_A and H_B misclassified. This is a simple linesearch in one dimension. Call the optimal value of this linesearch procedure τ^1. The first candidate output unit weight vector is then $v^1 \in R^h$ and threshold is $\tau^1 \in R$.

The second candidate weight vector v^2 is preemptive [1] in the sense that the hidden unit outputs are weighted according to the order in which they were calculated. This will ensure separation of the training set by the successive planes in the order in which they were generated. Here v^2 is determined in the following way.

Algorithm 1.2 Preemptive Weighting for the Neural Network Output Unit *Given h hidden units (h odd), determine the h elements of the weight vector v^2 of the output unit as follows.*

- *0 Weight the last hidden unit which attempts to classify the remaining "unclassified" points by 1,* i.e. *set $v_h^2 = 1$.*
- *1 Weight the remaining hidden units by the reverse order in which they were calculated as follows:*
    - *(i) Set $j = h - 1, k = 1$ and $p = (h-1) - k$.*
      Remark: j is a backward hidden unit counter, k is a backward hidden unit **pair** *counter except for the last hidden unit which is counted as a singleton and p is the correct superscript on the datasets considered "unclassified" at a given iteration and on parallel planes computed from these datasets.*
    - *(ii) If the halfspace $\{x \mid (w^p)^T x < \gamma_0^p\}$ contains a majority of points of $\mathcal{A}^p$, then $v_j^2 = 2^k$. If the halfspace $\{x \mid (w^p)^T x < \gamma_0^p\}$ contains a majority of points of $\mathcal{B}^p$, then $v_j^2 = -2^k$. Set $j = j - 1$.*
    - *(iii) If the halfspace $\{x \mid (w^p)^T > \gamma_1^p\}$ contains a majority of points of $\mathcal{A}^p$, then $v_j^2 = 2^k$. If the halfspace $\{x \mid (w^p)^T x > \gamma_1^p\}$ contains a majority of points of $\mathcal{B}^p$, then $v_j^2 = -2^k$. Set $j = j - 1, k = k + 1$ and $p = (h-1) - k$.*
    - *(iv) If $j = 0$, the weight vector v has been defined. If $j \neq 0$, then go to step (ii).*

Given this preemptive output weight vector v^2, threshold τ^2 is determined so that the number of points of H_A and H_B misclassified by the plane $\{u \in R^h | (v^2)^T u = \tau^2\}$ is minimized. This is accomplished by the linesearch procedure mentioned earlier.

Now two classification functions based on the two candidate output units are defined. We choose the candidate corresponding to the classification function with best performance on the training data $\mathcal{A} \cup \mathcal{B}$ (i.e. the one misclassifying the fewest points of $\mathcal{A} \cup \mathcal{B}$).

$$f_1(x) := \left[\sum_{i=1}^{h} v_h^1[(\hat{w}^h)^T x - \hat{\gamma}^h]_* - \tau^1\right]_*, \qquad (4)$$

$$f_2(x) := \left[\sum_{i=1}^{h} v_h^2[(\hat{w}^h)^T x - \hat{\gamma}^h]_* - \tau^2\right]_*.$$

If the best performer performs with accuracy above our predefined tolerance, the algorithm terminates. If the tolerance is not surpassed, then we proceed to the following post-processing step prior to beginning another iteration.

The post-processing step consists of removing the output unit and the last hidden unit computed from the neural network. The next iteration begins with the plane corresponding to the last hidden unit computed to determine two parallel planes which define "pure" halfspaces over the points "unclassified". Then points are removed which fall into these "pure" halfspaces and a candidate separating plane is computed on the remaining the points. Candidate output units are then computed and performance of the classification functions (4) is determined. Iterations cease when one of the classification functions (4) performs acceptably. We summarize the algorithm now.

Algorithm 1.3 Parsimonious Side Propagation (PSP) Algorithm. *Choose $\lambda \in [0,1)$, $\alpha > 0$ and choose $\kappa \in [0,1]$ to be an accuracy tolerance.*

- *0.* **Initialization.** *Set $\mathcal{A}^1 = \mathcal{A}$, $A^1 = A$, $\mathcal{B}^1 = \mathcal{B}$, $B^1 = B$, $j = 1$ and the number of hidden units $h = 0$. Compute (w^1, γ^1) by solving (1).*
- *1.* **Determine either one or three hidden units.**
    - *(i) ("Final" Hidden Unit) If the plane $(w^j)^T x = \gamma^j$ achieves separation correctness greater than or equal to κ with respect to $\mathcal{A}^j$ and $\mathcal{B}^j$, add one hidden unit with weight vector w^j and threshold γ^j, set $h = h + 1$. Go to 2.*
    - *(ii) (Hidden Unit Pair + Candidate "Final" Unit)*
        - *(a) Determine $\mathcal{H}_1^j := \{x | (w^j)^T x > \gamma_1^j\}$ so that $purity(\mathcal{H}^j, \mathcal{A}^j, \mathcal{B}^j) \geq \kappa$ and the number of data points in $\mathcal{H}_1^j$ is maximal.*
        - *(b) Similarly determine $\mathcal{H}_0^j := \{x | (w^j)^T x < \gamma_0^1\}$.*
        - *(c) Set $\mathcal{A}^{j+1}$ and $\mathcal{B}^{j+1}$ to be the points considered "unclassified", as in (3), and construct the matrices A^{j+1} and B^{j+1}.*
        - *(d) Calculate the candidate last hidden unit by solving (1) with A replaced by A^{j+1}, B replaced by B^{j+1}, m replaced by the number of rows of A^{j+1} and k replaced by the number of rows of B^{j+1}.*

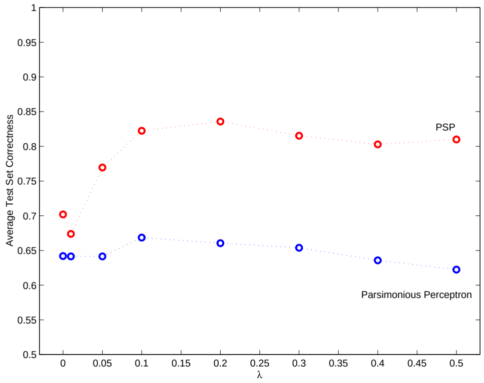

Figure 4: Tenfold cross-validation correctness versus parsimony parameter λ for the WPBC dataset.

Set $h = h + 3$. Go to 2.

2. **Compute weights and threshold of the output unit.** *Denote the neural network weight vectors and thresholds of the h hidden units as $(\hat{w}^1, \ldots, \hat{w}^h)$ and $(\hat{\gamma}^1, \ldots, \hat{\gamma}^h)$.*

   (i) *Determine the $m \times h$ matrix H_A of hidden unit outputs on $\mathcal{A}$ and the $k \times h$ matrix H_B of hidden unit outputs on $\mathcal{B}$.*

   (ii) *(Candidate Output Unit 1). Solve (1) with A replaced by H_A and B replaced by H_B to determine $v^1, \tilde{\tau}^1$. Determine τ^1 so that the number of points of H_A and H_B misclassified by the plane $\{u \in R^h | (v^1)^T u = \tau^1\}$ is minimized.*

   (iii) *(Candidate Output Unit 2). Determine preemptive output weight vector v^2 by Algorithm 1.2. Determine τ^2 so that the number of points of H_A and H_B misclassified by the plane $\{u \in R^h | (v^2)^T u = \tau^2\}$ is minimized.*

3. **Stopping Criterion.**

   (i) *Let $f_{Best}(x)$ be the classification function, from among those in (4), performing best over the entire training data $\mathcal{A} \cup \mathcal{B}$.*

   (ii) *If the correctness of $f_{Best}(x)$ is greater than κ, stop.*

   (iii) *If the correctness of $f_{Best}(x)$ does not surpass κ, set $j = j + 1$ and $h = h - 1$ and go to 1.*

2. COMPUTATIONAL RESULTS

The Wisconsin Prognostic Breast Cancer (WPBC) and the Ionosphere problems [5] were used as test problems. The 32-feature WPBC problem has 28 points in category one of breast cancer patients for which the cancer recurred within 24 months and category two of 119 patients for which the cancer did not recur within 24 months. The 34-feature Ionosphere problem has 225 points of radar returns from the ionosphere in one category and 126 points in the other. For the WPBC dataset normalized to have 0 mean and 1 standard deviation, the average tenfold cross-validation training set correctness for the Parsimonious Perceptron ($\lambda = 0.2$) was 68.71% and test set correctness was 66.05%. This perceptron utilized an average of 2 of the 32 WPBC features. The average tenfold cross-validation training set correctness for the Parsimonious Side Propagation neural networks was 92.44% ($\lambda = 0.2$) and test set correctness was 83.57%, an improvement of 26.53% over the Parsimonious Perceptron test set correctness. These neural networks utilized an average of 3.78 of the WPBC problem features and had, on average, 5 hidden units. These results are depicted in Figure 4. For the Ionosphere dataset, the average tenfold cross-validation training set correctness for the Parsimonious Perceptron ($\lambda = 0.3$) was 77.23% and test set correctness was 73.24%. This perceptron utilized an average of 2.1 of the 34 Ionosphere features. The average tenfold cross-validation training set correctness for the Parsimonious Side Propagation neural networks was 90.72% ($\lambda = 0.3$) and test set correctness was 87.71%, an improvement of 19.76% over the Parsimonious Perceptron test set correctness. These neural networks utilized an average of 9.1 of the Ionosphere features and had, on average, 6.5 hidden units. A graph similar to Figure 4 for this dataset was omitted for lack of space.

3. SUMMARY & CONCLUSION

We have proposed and implemented a fast linear-programming-based algorithm in which the task of separating successive groups of unclassified points is propagated sideways along hidden units until an overall separation accuracy on the training set is reached. Features deemed unnecessary by the mathematical program are removed by each hidden unit or unit-pair that is added. The economy in the use of available features and the minimal number of hidden units added to achieve an acceptable accuracy leads to a parsimonious neural network that appears to generalize well on unseen data.

4. REFERENCES

[1] K. P. Bennett and O. L. Mangasarian. Neural network training via linear programming. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 56–67, Amsterdam, 1992. North Holland.

[2] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Feature selection via mathematical programming. *INFORMS Journal on Computing*, 1998. To appear. Available at ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-21.ps.Z.

[3] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, California, 1991.

[4] O. L. Mangasarian. Mathematical programming in neural networks. *ORSA Journal on Computing*, 5(4):349–360, 1993.

[5] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, www.ics.uci.edu/AI/ML/MLDBRepository.html, 1992.