Chapter 37
Algorithms for the Optimal Loading of Recursive Neural Nets

V. Chandru* A. Dattasharma† S. S. Keerthi‡ N. K. Sancheti§ V. Vinay¶

Abstract
We address the problem of choosing synaptic weights in a recursive (Hopfield) neural network so as to "optimize" the performance of the network on the recognition of binary strings. The problem has been called the net loading (or learning) problem in the literature [10]. The objective is to maximize the basins of attraction around the desired fixed points (binary strings) of the net. It is known that it is $\mathcal{NP}$-hard to evaluate even the two-step radius of attraction of a recursive neural net [3]. We focus on the radius of direct (one-step) attraction and will refer to this as the loading problem. We have both theoretical and computational results on this problem, that we summarize below.

- A proof that the net loading problem can be solved in polynomial time using linear programming techniques. This resolves a standing problem in the complexity of recursive neural networks [10].
- An alternate formulation of the net loading problem as a proximity problem in high-dimensional convex geometry.
- The design and implementation of a hybrid algorithm for the said proximity problem.
- Successful solution of large scale test problems including the optimal solution to a 900 × 900 Hopfield net with approximately 4 × 10^5 synaptic weights.

It may be noted that our experiments indicate that the radius of direct attraction is actually a very good proxy of the intractable (multi-step) radius of attraction. In all the test problems that we have solved, the synaptic weights obtained as a a solution to the maximum radius of direct attraction also maximize the radius of (multi-step) attraction. In this sense the message of this paper is that the effective design of large-scale associative memory based on recursive neural networks is possible.

1 Introduction.

A recursive neural network is a discrete time, discrete valued dynamical system with a bipolar ($\{+1,-1\}$) state vector. The evolution of these networks is governed by a matrix of weights W called the synaptic matrix. These networks are of interest because of their use as associative memory in classification and pattern recognition. Given a specified set of binary vectors $\{y^1, y^2, \cdots, y^p\}$ the network behaves as an associative memory if it eventually reproduces as its output one of the inputs, say y^i, when triggered at the input by a vector z which is sufficiently close to y^i. Here "sufficiently close" means that the Hamming distance between z and the fixed point y^i, i.e. the number of unequal components of the two bipolar strings, is no more than the radius of the basin of attraction around y^i.

The dynamical behaviour of recursive neural network makes it difficult to analyse the radius of attraction around a given pattern. It has been shown by Floreen and Orponen [3] that it is $\mathcal{NP}$-hard to evaluate even the two-step radius of attraction of a recursive neural net. We present exact algorithms to synthesize recursive neural networks that optimize the radius of direct attraction.

(Net Loading) *Given vectors $y^1, ..., y^p$, where each y^i is an $n-$ dimensional vector whose components are $\{+1,-1\}$, construct a weight matrix, W, such that the resulting network has maximum radius of direct attraction about the y^i.*

We shall see, from both theoretical and computational perspectives, that this problem is tractable.

2 The net loading problem is in $\mathcal{P}$.

A recursive neural network is a network composed of artificial neurons with interconnections. The interconnections have weights associated with them. We let $W = (w^i_j)$ represent the weight matrix, where w^i_j is the weight associated with the interconnection from neuron j to neuron i. It is convenient to think of W as a set of row vectors w^i.

At any instant, a neuron is in one of two bipolar states, {1, -1}. Consider the state of the i^{th} neuron ($1 \le i \le n$) at time t, s_i. Then the state of the neuron at the next time instant is governed by the equation,

$$s_i^{new} = sgn(\sum_{1\le j\le n} w^i_j s_j).$$

Let $\mathbf{s} = (s_1, \ldots, s_n)^t$ denote the state vector of the

*CS & Automation, Indian Institute of Science, Bangalore–560 012, INDIA email:chandru@csa.iisc.ernet.in

†CS & Automation, Indian Institute of Science, Bangalore–560 012, INDIA email:abhi@chanakya.csa.iisc.ernet.in

‡CS & Automation, Indian Institute of Science, Bangalore–560 012, INDIA email:ssk@chanakya.csa.iisc.ernet.in

§Motorola India, Bangalore–560001, INDIA email:nirmal@master.miel.mot.com

¶make.edu, 1087 Sapnavi, Girinagar II Phase, Bangalore–560 085, INDIA email:vinay@csa.iisc.ernet.in

neural network at some time instant. Then the updating rule may be defined as

$$\mathbf{s}^{new} = sgn(W\mathbf{s}).$$

We will use $y^1, ..., y^p$ to denote the n-dimensional input vectors which require to be stored as *fixed points* of the neural network. z is a fixed point of a neural network defined by weight matrix W if $z = sgn(Wz)$. We will also need the notion of a *potential of a vector* z, $\phi(z)$. It is a column vector defined by the equation,

$$\phi(z) = Wz.$$

We will concentrate on solving the following decision problem:

Given a set of input patterns, $\{y^1, \ldots, y^p\}$, is there a weight matrix W for which the radius of direct attraction is at least k? That is,

$$d(y^i, z) \leq k \Rightarrow y^i = sgn(Wz) \; i = 1, \ldots, p.$$

where $d(y^i, z)$ denotes the Hamming distance between y^i and z.

It is a simple matter to check that if a vector z converges to a pattern y in one step ($y = sgn(Wz)$), then so does every vector on a shortest path from z to y. Another useful observation is that the condition $y = sgn(Wz)$ may equivalently be expressed as $\sum_{1 \leq j \leq n} y_i w^i_j z_j > 0, \; 1 \leq i \leq n$. The latter observation follows from the fact that the product of two numbers with identical sign is always positive.

The above observations suggest that for each pattern y^i, we enumerate all vectors at Hamming distance k from y^i. For every such vector z, we may write n inequalities of the form

$$\sum_{1 \leq j \leq n} y_i w^i_j z_j \geq 1, \quad 1 \leq i \leq n.$$

As the weight matrix is scalable by any positive quantity, we replace the strict inequality as originally described by an inequality whose right-hand side is unity. If the above set of inequalities is feasible, then there exists a weight matrix. Otherwise the radius of direct attraction is strictly smaller than k. The total number of inequalities in the above formulation is $pn\begin{pmatrix} n \\ k \end{pmatrix}$. Unfortunately, this quantity is exponential in the number of variables (eg: let k be $n/4$).

Consider the following example of designing a recursive neural network as associative memory for character recognition. Let us consider the problem of recognizing uppercase characters A–Z of the English alphabet. Each character is printed on a 30×30 grid. If grid positions which are filled are assigned a value of 1 and blank grid positions are assigned a value of -1, we get a bipolar vector in $\{-1, 1\}^{900}$ representing a character.

In the instance of the example that we generated the Hamming distance between the closest pair of patterns was 60. Hence, it is easy to see that the upper bound on the radius of direct attraction is 29. Let us now estimate the size of the linear inequality system that expresses the net loading problem for this test problem.

$$l = \sum_{k=0}^{k=29} \begin{pmatrix} 900 \\ k \end{pmatrix} \geq 10^{50}.$$

Hence, the total number of constraints are more than $10^{50} \times 900 \times 26 \geq 10^{54}$.

However, there is at least one way to tackle such an enormous linear program. Let us consider the problem of testing if a polyhedron $\mathcal{Q} \subset \Re^d$, defined by linear inequalities, is non-empty. For technical reasons let us assume that $\mathcal{Q}$ is rational, i.e. all extreme points and rays of $\mathcal{Q}$ are rational vectors or equivalently that all inequalities in some description of $\mathcal{Q}$ involve only rational coefficients. The ellipsoid method (in contrast with the simplex method and Karmarkar's algorithm) does not require the linear inequalities describing $\mathcal{Q}$ to be explicitly specified. It suffices to have an oracle representation of $\mathcal{Q}$. Several different types of oracles can be used in conjunction with the ellipsoid method[7, 11, 5]. We will use the *strong separation oracle* described below.

Oracle: **Strong Separation**($\mathcal{Q}$,y)
`Given a vector` $y \in \Re^d$`, decide whether` $y \in \mathcal{Q}$`, and if not find a hyperplane that separates` y `from` $\mathcal{Q}$`; more precisely, find a vector` $c \in \Re^d$ `such that` $c^T y < \min\{c^T x \mid x \in \mathcal{Q}\}$.

It is well known [5, 7, 11] that the complexity of the linear inequality (feasibility) question is polynomially equivalent to the complexity of the oracle. Therefore, if we can construct a polynomial time separation oracle for the linear program formulated in the previous section, we would have a polynomial time algorithm to solve the decision version of (Net Loading). We are now ready to describe such a separation oracle.The oracle is based on a simple restatement of the potential of a bipolar vector z as given in the lemma below.

LEMMA 2.1. *Let z be any bipolar vector and y be one among the input vectors, $\{y^1, \cdots, y^p\}$. Further, let $A = \{j : z_j \neq y_j\}$. Then the i^{th} component of the vector $\phi(z)$ is,*

$$\phi_i(z) = w^i.z = \phi_i(y) - 2\sum_{j \in A} w^i_j * y_j.$$

Notes:

(i) The cardinality of the set A is precisely the Hamming distance between the vectors z and y.

(ii) We require $\phi_i(z)$ to have the same sign as y_i. For this to happen,
$\phi_i(z) * y_i = (\phi_i(y_i) - 2\sum_{j\in A} w^i_j * y_j) * y_i \geq 0$.
We introduce new variables $s_{ij} = 2w^i_j * y_j * y_i$. Clearly, z does not converge to y on component i if $\phi_i(y) * y_i < \sum_{j\in A} s_{ij}$.

These observations can be used to calculate the radius of direct attraction for a given weight matrix W. The obvious strategy is to arrange $s_{ij}, 1 \leq j \leq n$ in descending order and use greedy packing to determine the smallest cardinality set A that causes $\sum_{j\in A} s_{ij} > \phi_i(y) * y_i$.

```
Procedure: RDA(W,r)
for all patterns $y = y^1, \ldots, y^p$ and a component $i$, $1 \leq i \leq n$ do
form a set $S_i$ whose elements are $s_{ij} (1 \leq j \leq n)$ with weights $2 * w^i_j * y_j * y_i$.
calculate $t := \phi_i(y) * y_i$.
if $t < 0$ then $y$ is itself not a fixed point.
    set $A(y,i) = \emptyset$.
else
    do choose greedily the elements of $S_i$ with decreasing weight
    until their cumulative sum exceeds $t$.
Let $A(y,i)$ be the subset of elements chosen and $r(y,i)$ be the cardinality of the set.
Return $r - 1$, the radius of direct attraction of $W$, where $r = min_{y,i} r(y,i)$.
End
```

The strong separation oracle for the decision version of (Net Loading) uses the above procedure. Recall that the separation oracle needs to determine if a given weight matrix W results in a network with radius of direct attraction at least k. If the answer is negative, we need to identify a separating inequality that strictly separates W from the polyhedron described by the linear inequalities of the linear programming formulation.

```
Procedure: Separation(W,k)
call Procedure RDA(W,r);
if $(r \geq k)$ then output ``Feasible''
else $(r < k)$ and there is some $r(y,i) < k$. Then the inequality
```

$$\Big(\sum_{s_{ij}\notin A(y,i)} w^i_j y_j - \sum_{s_{ij}\in A(y,i)} w^i_j y_j\Big) * y_i \geq 1$$

```
is a separating inequality.
```

This completes the description of the oracle. It is easy to see that the procedure RDA(W,r) can in fact be implemented in $O(pn^2 \log n)$ time by using a sorting routine at the greedy step. Therefore, the oracle runs in polynomial time and as a consequence the decision version of the net loading problem is in $\mathcal{P}$. Using the usual layer of binary search on the decision problem, i.e. adding a $\log n$ factor, we obtain the tractability result for the optimization version of net loading.

THEOREM 2.1. ([2]) *For an arbitrary set of input bipolar patterns, the problem of designing a recursive neural network with maximum radius of direct attraction, about the input patterns, is soluble in polynomial time.*

The added restrictions that the weight matrix be symmetric with a non-negative diagonal can be encoded as additional (polynomially many) linear inequalities in our linear programming formulation. The separation oracle can explicitly check these inequalities. Using a small variant of the general theory of compact formulations of linear programs solvable by the ellipsoid method due to Kipp Martin [9], we have been able to show that the net loading problem can in fact be reformulated as a linear program with only $O(pn^2)$ constraints and variables, i.e. a polynomial size formulation [1][1]. These results imply that interior point methods of linear programming can also be used to realize polynomial time algorithms for net loading.

Notwithstanding the availability of these compact formulations, the results described above do not provide reasonable hope of efficient implementation of a solver for the net loading problem. The ellipsoid method is well known for its impracticality in solving linear programming of even medium scale (for well documented reasons). The use of simplex or interior point methods on the compact formulation also appears to be impractical. For example, consider the character recognition problem (Example 2). The quantity pn^2 (a very conservative lower bound on the number of variables and the number of constraints) in this case would be approximately 2×10^7. So, in order to realize an effective computational strategy for solving the net loading problem, we adopted an entirely different approach.

3 Net Loading as a Proximity Problem

The net loading problem may be viewed in the abstract as solving a system of linear, homogeneous (strict)

[1]This result has been independently reported by Prabhu and Shaw[12] using a different proof technique

inequalities of the form:

$$a \cdot x > 0 \ \forall a \in \mathcal{A}. \tag{3.1}$$

where $\mathcal{A}$ is a finite set of vectors in R^M. Note that scaling any of the vectors $a \in \mathcal{A}$ does not alter feasible solutions of (3.1). So, let us assume without loss of generality that $\|a\| = 1 \ \forall a \in \mathcal{A}$. The separation oracle that we described in the last section computes for any $x \in R^M$, a real number $h(x)$ and a vector $c(x) \in \mathcal{A}$ such that

$$\begin{aligned} h(x) &= \min\{a \cdot x : a \in \mathcal{A}\}, \\ c(x) &= \arg\min\{a \cdot x : a \in \mathcal{A}\}. \end{aligned}$$

Note that c is not a function, since for some x, $c(x)$ may not be unique. c is a many to many map. Let $CH(\mathcal{A})$ denote the convex hull of the rows of $\mathcal{A}$ interpreted as points in R^M. Then, $h(x)$ and $c(x)$ are known as *the support function* and *the contact function* of $CH(\mathcal{A})$, i.e. the hyperplane $\{y : x \cdot y = h(x)\}$ supports $CH(\mathcal{A})$ at $c(x)$. The support and the contact functions are of importance because the algorithms we will describe in the following sections will make use of these to solve the design problem. One more real number $g(x)$ associated with any $x \in R^M$ is of importance in the algorithms described later. This is defined as,

$$g(x) = -h(x) + \|x\|^2.$$

It should be easy to see that $x^* \in \mathrm{CH}(\mathcal{A})$ is the closest point to the origin from $\mathrm{CH}(\mathcal{A})$ if and only if $g(x^*) = 0$.

Note that if $\bar{x}$ is a solution of (3.1) then $\alpha\bar{x}$ is also a solution of (3.1) for every $\alpha > 0$. Thus, the solution set of (3.1) is either empty or infinite. We consider a tighter formulation of the design problem. Our aim till now was to find an x such that $h(x) > 0$. Instead, we will try to solve the following problem:

$$\max\ h(x) \ \text{ s.t. } \|x\| = 1. \tag{3.2}$$

If $\theta(a, x)$ denotes the angle between vectors a and x, i.e.,

$$\theta(a, x) = \cos^{-1} \frac{a^t x}{\|a\|\|x\|},$$

then (3.2) is equivalent to solving

$$\min_{x \in R^M, x \neq 0} \left(\max_{a \in \mathcal{A}} \theta(a, x) \right). \tag{3.3}$$

The motivation for solving (3.2) is robustness and fault-tolerance.

Consider the problem,

$$\min\ \|y\|^2 \ \text{ s.t. } y \in CH(\mathcal{A}). \tag{3.4}$$

Note that (3.4) has a unique solution. The relation between (3.2) and (3.4) is established in the following lemmas.

LEMMA 3.1. *Suppose (3.1) has a solution. Also, let $D_+(CH(\mathcal{A}), \{0\})$ be the Euclidean distance between $CH(\mathcal{A})$ and the origin. Then: $D_+(CH(\mathcal{A}), \{0\}) = \max\{h(x) : \|x\| = 1, h(x) > 0\}$.*

Proof. By definition,

$$D_+(CH(\mathcal{A}), \{0\}) = \inf\{\|x\| : x \in CH(\mathcal{A})\}.$$

Let $\{x_k\} \subset CH(\mathcal{A})$ be such that $\|x_k\| \to D_+(CH(\mathcal{A}), \{0\})$. Since, $CH(\mathcal{A})$ is compact the sequence $\{x_k\}$ will converge to a point $\bar{x} \in CH(\mathcal{A})$. Define $\bar{\eta} = \bar{x}/\|\bar{x}\|$. (Note that $\|\bar{x}\| = D_+(CH(\mathcal{A}), \{0\}) > 0$.) Optimality of $\bar{x}$ implies that the hyperplane, $\{x : \bar{\eta} \cdot x = D_+(CH(\mathcal{A}), \{0\})\}$ contains $\bar{x}$ and strictly separates $CH(\mathcal{A})$ from the origin. In other words,

$$\|\bar{x}\| = D_+(CH(\mathcal{A}), \{0\}) = h(\bar{\eta}).$$

Now choose any η satisfying $\|\eta\| = 1$. By Schwartz inequality and the above equation we get

$$\begin{aligned} h(\eta) &= \inf\{\eta \cdot w : w \in CH(\mathcal{A})\} \\ &\leq \eta \cdot \bar{x} \\ &\leq \|\bar{x}\| \\ &= D_+(CH(\mathcal{A}), \{0\}) \\ &= h(\bar{\eta}). \end{aligned}$$

Thus $D_+(CH(\mathcal{A}), \{0\}) = \max\{h(\eta) : \|\eta\| = 1\}$. Since $D_+(CH(\mathcal{A}), \{0\}) > 0$, the lemma follows.

LEMMA 3.2. *If (3.1) has solution then (3.2) and (3.4) are equivalent.*

Proof. Note that the optimum objective function value of (3.4) is the Euclidean distance between $\mathrm{CH}(\mathcal{A})$ and the origin, i.e. $D_+(\mathrm{CH}(\mathcal{A}), \{0\})$. Since (3.1) has a solution we know that $\exists \bar{x}$ such that

$$a \cdot \bar{x} > 0 \ \forall a \in \mathcal{A}.$$

This also implies that

$$a \cdot \bar{x} > 0 \ \forall a \in \mathrm{CH}(\mathcal{A}).$$

Hence $0 \notin \mathrm{CH}(\mathcal{A})$. This means that $\mathrm{CH}(\mathcal{A}) \cap \{0\} = \phi$ and $D_+(\mathrm{CH}(\mathcal{A}), \{0\}) > 0$. We can apply Lemma 3.1 to find $D_+(\mathrm{CH}(\mathcal{A}), \{0\})$. Note that by substituting $A = \mathrm{CH}(\mathcal{A})$ and $B = \{0\}$ the h defined above and the one used in Lemma 3.1 are same. Hence by part (i) of Lemma 3.1 we have,

$$D_+(\mathrm{CH}(\mathcal{A}), \{0\}) = \max\{h(x) : \|x\| = 1, h(x) > 0\}.$$

Since, 3.1 has solution the condition $h(x) > 0$ is superfluous and can be dropped. This proves the lemma.

The following lemma is also easy to establish.

LEMMA 3.3. *(3.1) has a solution if and only if the least objective function value of (3.4) is positive.*

Proof. As shown in the proof of Lemma 3.2 the existence of solution for (3.1) implies that $0 \notin \mathrm{CH}(\mathcal{A})$. Hence $D_+(\mathrm{CH}(\mathcal{A}), \{0\}) > 0$ and the optimum objective function value of (3.4) is positive. Let us assume that the optimum solution of (3.4) is $\bar{y}$. By using part (i) of Lemma 3.1 we have,

$$\max\{h(x) : \|x\| = 1\} = \|\bar{y}\|. \tag{3.5}$$

Let $\eta = \bar{y}/\|\bar{y}\|$. Clearly $\|\eta\| = 1$ and $\eta \cdot \bar{y} = \|\bar{y}\|$. Hence,

$$h(\eta) \leq \|\bar{y}\|. \tag{3.6}$$

Combining (3.5) and (3.6) we have,

$$h(\eta) = \|\bar{y}\| > 0.$$

Since,

$$h(\eta) = \inf\{\eta \cdot x : x \in \mathrm{CH}(\mathcal{A})\},$$

we have

$$\eta \cdot a > 0 \; \forall a \in \mathcal{A}.$$

Thus η is a solution of (3.1). On the other hand if the optimum objective function value of (3.4) is zero then it means that $0 \in \mathrm{CH}(\mathcal{A})$. But we know that if there exists a solution of (3.1) then origin cannot belong to the convex hull of $\mathcal{A}$. This proves the lemma.

4 Algorithms for the proximity problem.

In this section we will assume that the design problem has been represented as (3.1) (or equivalently (3.4)). The oracle presented in the previous section is used to compute contact and support function.

4.1 Perceptron Algorithm The Perceptron algorithm [13] is one of the simplest ways to solve (3.1). The perceptron algorithm can briefly be described as follows:

0. Choose x_0 randomly and set $k = 0$.

1. Compute $c(x_k)$ and $h(x_k)$. If $h(x_k) > 0$ then stop with the conclusion that x_k is a solution to (3.1). Else go to 2.

2. Set
$$\begin{aligned} x_{k+1} &= x_k + c(x_k), \\ k &= k+1, \end{aligned}$$
and go to 1.

The perceptron algorithm described above, converges in a finite number of iterations when the system (3.1) has a solution[13]. However, this algorithm will oscillate indefinitely when (3.1) does not have a solution. Hence, in this approach it is difficult to choose a termination criterion.

4.2 Gilbert's Algorithm Gilbert's algorithm [4] is the simplest descent algorithm for solving (3.4). The algorithm can be described as follows.

0. Choose any $y_0 \in \mathcal{A}$ and set $k = 0$.

1. Compute $c(y_k)$, $h(y_k)$ and $g(y_k)$. If $g(y_k) = 0$ then stop with y_k as the solution to (3.4). Else go to 2.

2. Set y_{k+1} = the point on the line segment joining y_k and $c(y_k)$ with least norm. Also, set $k \overset{!}{=} k+1$ and go to 1.

In general Gilbert's algorithm requires an infinite number of iterations to converge to the solution of (3.4). However, if the feasibility of (3.1) suffices, then we can replace the termination condition, '$g(y_k) = 0$' in step 1 of the above algorithm by '$h(y_k) > 0$'. In that case Gilbert's algorithm converges in a finite number of iterations when a solution to (3.4) exists.

Gilbert's algorithm is only very slightly more expensive than the perceptron algorithm. This comparison is for one iteration of both the algorithms. These two algorithms perform very differently. Gilbert's algorithm has a better termination capability. In this algorithm lack of existence of solution to (3.1) will cause the sequence $\{\|y_k\|\}$ to tend to zero. One more significant difference between the two algorithms is that the perceptron algorithm is not a descent algorithm. Hence, in the perceptron algorithm we cannot really say that the successive iterates are 'nearing the solution' in any sense.

4.3 Wolfe's algorithm The third algorithm we consider is due to Wolfe [15]. This is a simplex-type method for solving (3.4). For the case where $CH(\mathcal{A})$ is replaced by a convex cone in (3.4), Lawson and Hanson [8] gave a similar algorithm slightly before Wolfe's publication. However, Wolfe's discovery was independent of Lawson and Hanson's work. We will refer to this algorithm as Wolfe's algorithm. It is a descent algorithm which does a clever search over the set of simplices formed from the points of $\mathcal{A}$. Let us define for a convex polyhedron X,

$$\begin{aligned} \rho(X) &= \min \; \{\|x\| : x \in X\}, \text{and}, \\ \gamma(X) &= \arg\min \; \{\|x\| : x \in X\}. \end{aligned}$$

Now we will briefly describe the algorithm. At each iteration the algorithm maintains an affinely independent set of points, $\mathcal{A}_k \subset \mathcal{A}$, and a point y_k such that

$$\begin{aligned} &y_k = \gamma(AH(\mathcal{A}_k)) \text{ and,} \\ &y_k \in \mathrm{rel\,int}(CH(\mathcal{A}_k)), \end{aligned}$$

where $\mathrm{AH}(\mathcal{A}_k)$ denotes the affine hull of $\mathcal{A}_k$. If $g(y_k) = 0$ then y_k solves (3.4). If not then it is easy to show that

$\rho(CH(\tilde{\mathcal{A}})) < \rho(CH(\mathcal{A}_k))$, where $\tilde{\mathcal{A}} = \mathcal{A}_k \cup \{c(y_k)\}$. The algorithm now determines a set $\mathcal{A}_{k+1}$ and y_{k+1} that satisfy: $\mathcal{A}_{k+1} \subset \tilde{\mathcal{A}}$; $y_{k+1} = \gamma(AH(\mathcal{A}_{k+1}))$; $y_{k+1} \in \text{rel int}(CH(\mathcal{A}_{k+1}))$; and $\rho(CH(\mathcal{A}_{k+1})) < \rho(CH(\mathcal{A}_k))$.

The following is a brief, but reasonably self-contained description of the actual algorithm.

0. Choose any $y_0 \in \mathcal{A}$, set $\mathcal{A}_0 = \{y_0\}$ and $k = 0$.

1. Compute $c(y_k)$, $h(y_k)$ and $g(y_k)$. If $g(y_k) = 0$ then stop with the conclusion that y_k is the solution of (3.4). If not then set $\tilde{y} = y_k$, $\tilde{\mathcal{A}} = \mathcal{A}_k \cup \{c(y_k)\}$ and go to 2.

2. Compute $\hat{y} = \gamma(AH(\tilde{\mathcal{A}}))$. If $\hat{y} \in CH(\tilde{\mathcal{A}})$ then trim $\tilde{\mathcal{A}}$ such that $\hat{y} \in \text{rel int}(CH(\tilde{\mathcal{A}}))$, set $y_{k+1} = \hat{y}$, $\mathcal{A}_{k+1} = \tilde{\mathcal{A}}$, $k = k + 1$, and go to 1. Else go to 3.

3. Let L be the line segment joining $\hat{y}$ and $\tilde{y}$. Find $z = \arg\min\{\|y - \hat{y}\| : y \in L \cap CH(\tilde{\mathcal{A}})\}$. The point z lies on F, a face of $CH(\tilde{\mathcal{A}})$ whose dimension is smaller than that of $CH(\tilde{\mathcal{A}})$. Trim $\tilde{\mathcal{A}}$ by keeping only those points corresponding to F, set $\tilde{y} = z$ and go to 2.

It turns out that $\|z\| < \|\tilde{y}\|$. It is this fact which leads to $\|y_{k+1}\| < \|y_k\|$. Because $y_k = \gamma(AH(\mathcal{A}_k))$ and $\|y_{k+1}\| < \|y_k\|$, we get $\mathcal{A}_j \neq \mathcal{A}_k \quad \forall j \neq k$. Since the number of subsets of $\mathcal{A}$ is finite the algorithm converges in a finite number of iterations. The computation of $\hat{y}$ in step 2 is achieved by solving the system of linear equations:

$$\hat{y} = \sum_{a \in \tilde{\mathcal{A}}} \lambda_a a, \quad \sum_{a \in \tilde{\mathcal{A}}} \lambda_a = 1, \quad \hat{y}^t a = \delta \quad \forall a \in \tilde{\mathcal{A}}.$$

The λ_a's are referred to as the barycentric co-ordinates of $\hat{y}$ with respect to $\tilde{\mathcal{A}}$. In the algorithm, the points y_k, $\tilde{y}$, $\hat{y}$ and z are all represented using the barycentric co-ordinates with respect to appropriate subsets of $\mathcal{A}$, and, these co-ordinates play a crucial role in the trimming operations in steps 2 and 3 and the determination of z in step 3. The main computational cost consists of $c(y_k)$ and $h(y_k)$ in step 1 and the computation of $\hat{y}$ in step 2. The computation of $c(y_k)$ and $h(y_k)$ can be done using the oracle described in the previous section.

5 Computational Experiments

In this section we will describe our experiences with solving the net loading problem on specific examples. In all the problems the positivity of the diagonals of the weight matrix W was included in (3.1). When symmetry is enforced the vector x in (3.1) consists of the elements of upper triangular and diagonal parts of the weight matrix together with the thresholds. When symmetry is not enforced, then the neural network problem is a set of decoupled problems of the form (3.1), one for each row of the weight matrix and the corresponding threshold. All the algorithms were terminated as soon as a feasible solution of (3.1) was found.

The initialization of x_0 or y_0 was done by setting them equal to the $a \in \mathcal{A}$ corresponding to diagonal positivity when symmetry is not enforced. In the cases where symmetry was enforced it was set to the centroid of the vectors in $\mathcal{A}$ corresponding to the positivity of all the diagonals. Note that in the former case for every decoupled problem only one of the vectors in $\mathcal{A}$ correspond to the diagonal positivity condition. The only motivation for using this initialization was its simplicity. All the algorithms were executed on an Intel I-860 workstation[2] with an optimized Fortran compiler.

Now we will describe the problems which were tried using the 3 algorithms.

N_1: This problem consists of designing a recursive neural network for recognizing the patterns $+, -, \times, /, |$. These patterns were taken on a 5×5 grid. The dimension of the neural network is 25. Symmetry was enforced. Hence, the number of variables in (3.1), $M = 25 \times 13 + 25 = 350$. The pairwise minimum Hamming distance between patterns, D was found to be 4. Hence, the maximum radius of direct attraction for which the neural network can be designed is 1.

N_2: This problem is for recognizing the 7 patterns which are digits 0–6 drawn on a 9×7 grid. This problem was taken from Tou and Gonzalez [14]. (In [14] there are 14 patterns. We have chosen the first 7 of them.) The network dimension in this case is 63. Symmetry was enforced and hence, $M = 63 \times 32 + 63 = 2079$. D was found to be 10 and hence the maximum radius of direct attraction possible is 4. This problem is also known as Banker's character set.

N_3: This problem is same as N_2 except that all 14 patterns in [14] were included. In this case also D was found to be 10 and hence the maximum radius of direct attraction possible is 4.

Table 1 gives the performances of the algorithms on the 3 problems. The 2 numbers in the table are *the number of calls to the support-contact function oracle/CPU time* in minutes. This CPU time also includes time for some I/O operations. In the table an entry a means that the algorithm was terminated because the number of calls to the oracle exceeded 5000, and b means that the algorithm terminated because the

[2]The speed of the Intel I-860 workstation is roughly the same as that of a SPARC/RS6000.

Pb	R	P	G	W	S
N_1	1	73/0.05	59/0.05	50/0.05	47
N_2	1	191/1.1	185/1.1	153/1.2	148
	2	32/0.2	48/0.3	31/0.2	30
	3	143/1.0	414/3.0	249/4.0	242
	4	833/2.8	a	b	500
N_3	1	431/2.5	700/3.9	b	500
	2	241/1.2	a	b	500
	3	969/5.1	a	b	500
	4	a	a	b	500

Table 1: The performance of three algorithms

Problem	Radius	Hybrid algorithm
N_2	1	18/0.1
	2	5/0.03
	3	17/0.1
	4	88/0.5
N_3	1	18/0.15
	2	14/0.13
	3	27/0.21
	4	156/1.2

Table 2: Performance of the hybrid algorithm

size of the linear equation system to be solved exceeded 500.

It is interesting to note that the upper bound on the radius of direct attraction (computed using the Hamming distance between the nearest pair of patterns) is achieved. On problem N_3, none of the three algorithms succeeded. The fact that the upper bound of the radius is achievable was confirmed by using the Hybrid algorithm to be discussed later. In fact this upper bound was achieved with some other problems also, which we solved using the Hybrid algorithm. These problems will also be discussed later.

In the computations, the initialization mentioned earlier was used only for radius = 1. Then the solution was obtained for radius = 1. Then this solution was used to initialize the algorithm for radius = 2 and so on. Thus, the computational cost indicated in the tables are the incremental costs, i.e., the extra effort needed to solve the problem for that radius.

In table 1, column Pb gives the problem number; column R represents Radius and columns P, G and W show the results obtained using Perceptron Algorithm, Gilbert's Algorithm and Wolfe's Algorithm respectively. The column S shows the Simplex Dimension at last iteration.

6 The Hybrid Algorithm

Wolfe's algorithm is not suitable for our problem because the dimension of the simplex $CH(\mathcal{A}_k)$ which contains y_k generally grows as the iterations proceed. The dimensions become unmanageable above 500, with the algorithm spending a very large computational time in linear equation solving. Also, it was found that, the improvement in cost at each iteration, i.e., $\|y_k\| - \|y_{k+1}\|$ given by Wolfe's algorithm is not very much bigger than that given by Gilbert's algorithm. The hybrid algorithm has been designed to be somewhere in between the two algorithms, by using, at each iteration, a carefully chosen simplex of dimension much greater than two, but bounded above by a reasonably small number. A quick look at the oracle for computing the contact and support function gives a good way for choosing this simplex. The set $\mathcal{A}$ is actually grouped as:

$$\mathcal{A} = \cup\{\mathcal{B}_j : j = 0, 1, \ldots, p\},$$

where p is the number of patterns, $\mathcal{B}_0$ corresponds to the diagonal positivity, and, for $j \geq 1$, $\mathcal{B}_j$ corresponds to the set of constraint vectors for the j-th pattern.

The oracle which we have described computes, for each $j = 0, 1, \ldots, p$,

$$c_j(y) = \arg\ \min\{a^t y : a \in \mathcal{B}_j\},$$

and sets,

$$C(y) = \{c_j(y) : j = 0, 1, \ldots, p\},$$

and, computes $c(y)$ as,

$$c(y) = \arg\ \min\{a^t y : a \in C(y)\}.$$

The set $C(y_k)$ is used to form the simplex in the hybrid algorithm. The steps are as follows:

0. Choose any $y_0 \in \mathcal{A}$ and set $k = 0$.

1. Compute $c(y_k)$, $h(y_k)$ and $g(y_k)$. The set $C(y_k)$ comes as a byproduct. If $g(y_k) = 0$ then stop with the conclusion that y_k is the solution of (3.4). Else go to 2.

2. Use Wolfe's algorithm to compute y_{k+1}, the point in $CH(C(y_k))$ with least norm, set $k = k + 1$ and go to 1.

The hybrid algorithm has convergence properties very similar to that of Gilbert's algorithm. This can be proved using similar lines of proof to those used by Gilbert's algorithm [4]. The hybrid algorithm gave a vastly improved performance. Results on N_2 and N_3 are given in Table 2.

We then considered 2 large problems and applied the hybrid algorithm on them. These 2 problems are as follows.

N_4: This problem is that of uppercase English alphabet character recognition. This is the same as discussed in Example 2. The dimension of the neural network is 900. Symmetry of the weight matrix was enforced and hence $M = 900 \times 901/2 + 900 = 406350$. D was found to be 60. Thus, 29 is an upper bound on the maximum radius of direct attraction.

N_5: This problem is the same as N_4, except that the symmetry of the weight matrix was not enforced. So, this problem is equivalent to 900 decoupled problems, each with $M = 901$.

We found that in such large problems, the cost of each call to the oracle that computes the support and contact functions is rather large. Originally, in solving problems N_1–N_3 we did only a crude implementation of the oracle. To improve it two important factors were taken into consideration.

- The binary and sparse structure in the elements of $\mathcal{A}$.
- Efficient arrangement of sorting steps.

The improved implementation gave an excellent decrease in the oracle cost. For problem N_4, each call to the improved oracle implementation needed about 20 seconds of the CPU time, whereas the original implementation took more than a minute. All these CPU times are on the Intel I-860 machine. On the same problem, the cost of computing the point of minimum norm in the convex hull of 28 points (Step 2 of hybrid algorithm) takes only about 3 seconds. For problem N_4, the hybrid algorithm took about 71,000 iterations and radius of direct attraction of 29 was possible. This took CPU time of about 16 days. A radius of direct attraction of 20 was found to be feasible at the end of the first day. Radius of 25 was feasible at the end of the second day and 28 at the end of the fourth day.

For problem N_5, the hybrid algorithm took about 45 minutes of CPU time for solving all the 900 decoupled problems. The average iteration per problem was found to be 125. This problem was solved directly for a radius of 29, unlike the previous problems, where the radius was incremented by 1 starting with an initial value of 1.

7 Neural Implementation.

If the algorithms presented in this paper are to be implemented in neural hardware, then it is important to avoid global computations that are done by a central processor which communicates with all neurons. A careful look at the computations, described in section 2, shows that the calculation of support and contact functions required by the algorithms of sections 4 and 6 can be done locally by the neurons. If weight symmetry is not to be enforced then the remaining computational steps of these algorithms can also be done locally. But, enforcing symmetry requires a central processor and global computations. A problem for future work is to modify the hybrid algorithm so as to enforce symmetry via local computations.

References

[1] V. Chandru, M.R. Rao and V. Vinay, *"The Net Loading Problem is Tractable"*, Technical Report, Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India (October, 1993).

[2] V. Chandru and V. Vinay, *"Constructing highly attractive recursive neural networks"*, Proc. of the Intl. Symp. on Intelligent Robotics, Bangalore, India, 1993.

[3] P. Floreen and P. Orponen, *"Attraction radii in binary Hopfield nets are hard to compute"*, Technical Report, Department of Computer Science, University of Helsinki, Helsinki SF-00510, FINLAND (Available via ftp from archive.cis.ohio_state.edu in /pub/neuroprose directory).

[4] E. G. Gilbert, *"An Iterative procedure for computing the minimum of a quadratic form on a convex set"*, SIAM Jl. of control, vol. 4, No. 1, pp 61–80, 1966.

[5] Grötschel, M., Lovász, L., and Schrijver, A., *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, 1988.

[6] Kamp, Y., Hasler, M., *Recursive Neural Networks as Associative Memory*, John Wiley and Sons, 1991.

[7] Karp, R.M. and Papadimitriou, C.H. "On Linear Characterizations of Combinatorial Optimization Problems", *SIAM Journal on Computing*,**11** (1982), 620-632.

[8] C. L. Lawson and R. J. Hanson, *Solving least squares problems*, Prentice Hall Inc., New Jersey, 1974.

[9] R. Kipp Martin, *"Using Separation Algorithms to Generate Mixed Integer Reformulations"*, in Operations Research Letters 10, 119-128, 1991.

[10] P. Orponen, *"Neural Networks and Complexity Theory"*, in Proceedings of the 17th International Symposium on the Mathematical Foundations of Computer Science, LNCS 629, 50-61, 1992.

[11] Padberg, M.W. and Rao, M.R. "The Russian Method for Linear Programming III: Bounded Integer Programming", *Research Report 81-39*, Graduate School of Business Administration, New York University, New York (1981).

[12] N. Prabhu and D.X. Shaw, *"Linear Programming for Designing Recursive Neural Networks"*, Preprint, School of Industrial Engineering, Purdue University, West Lafayette, IN 47907, February 1994.

[13] F. Rosenblatt, *Principles of neurodynamics*, New York: Spartan, 1962.

[14] J. Tou and R. Gonzalez, *Pattern recognition principles*, Addison-Wesley, 1974.

[15] P. Wolfe, *"Finding the nearest point in a polytope"*, Mathematical Programming, vol. 11, pp 128–149, 1976.