ELSEVIER

Neurocomputing 17 (1997) 195–216

NEUROCOMPUTING

# MLP iterative construction algorithm

Thomas F. Rathbun[a,*], Steven K. Rogers[b], Martin P. DeSimio[a], Mark E. Oxley[c]

[a] *WL\AACS, 2241 Avionics Circle, Suite 19, Wright-Patterson AFB, OH 45433-7321, USA*
[b] *Battelle Memorial Institute, 505 King Ave, Columbus, OH 43205-2693, USA*
[c] *Department of Mathematics and Statistics, Air Force Institute of Technology, Wright-Patterson AFB, OH 45433-7765, USA*

Received date 19 October 1996; accepted 6 May 1997

**Abstract**

This paper presents a novel multi-layer perceptron neural network architecture selection and weight training algorithm for classification problems. The MLP iterative construction algorithm (MICA) autonomously constructs an MLP neural network as it trains. Experimental results show the algorithm achieves 100% accuracy on the training data, the same or better generalization accuracies as Backprop on the test data, while using less FLOPS. Moreover, relaxation of the hidden layer nodes improves test set recognition accuracies to be greater than that of Backprop. Furthermore, seeding the Backprop algorithm with the hidden layer weights from MICA is demonstrated. The MICA seeding improves the effectiveness of Backprop and enables Backprop to solve a new class of problems, i.e., problems with areas of low mean-squared error.

*Keywords:* Neural network training; Construction; Ho–Kashyap method; Hidden node pruning; Multi-layered perceptron

## 1. Introduction

This paper details a novel algorithm for autonomous architecture selection and weight training of a single hidden layer multi-layer perceptron (MLP) neural network for

* Corresponding author. Tel: 513-255-3636, Ext. 1342; fax: 513-476-4055; e-mail: trathbun@afit.af.mil, mdesimio@afit.af.mil.

classification. The MLP iterative construction algorithm (MICA) realizes a Cybenko-like net, i.e., a single hidden layer MLP for classification problems. Cybenko [6] proved that given enough nodes a single hidden layer MLP can classify any training data such that the total measure of the incorrectly classified points can be made arbitrarily small. Cybenko, in his closing remarks states, "The important questions that remain to be answered deal with feasibility, namely, how many terms in the summation (or equivalently, how many neural nodes) are required to yield an approximation of a given quality." [6] Many researchers have tried to answer Cybenko's question in abstraction [1–3, 5, 16–19], but much of this research is impractical. For example, Baum's Theorem 1 states, "A one-hidden-layer net with $\lceil N/d \rceil$ internal units can compute an arbitrary dichotomy of $N$ $d$-dimensional vectors in general position" [1]. The problem is that one usually wants to compute a specific dichotomy not an arbitrary one, and thus $\lceil N/d \rceil$ hidden nodes may be too many and adversely effect generalization. Suppose we wish to solve the XOR problem with a 1000 data points. Baum's theorem suggests that we need 500 hidden nodes, where, in reality, four will suffice. Another well known work is Vapnik's VC dimension or capacity [19], which defines the maximum number of data points for which a classifier can implement on all possible dichotomies. Baum and Haussler have derived bounds on the VC dimension in term of the number of hidden nodes, total nodes, and total number of weights [3]. However, these bounds are very conservative and led Jain and Mao [9] to state, "The practical applicability of these bounds is questionable." It is apparent that the number of hidden nodes needed is data dependent and can not be gleaned except by processing the data. Therefore, the most intelligent algorithms will take the data into account when constructing the network. A theory on this subject is presented in a paper by Roy [13]. Roy talks about robust and efficient learning theory and states that an intelligent learning algorithm should have the following general characteristics:

1. Perform network design task;
2. Robustness in learning;
3. Quickness in learning;
4. Efficiency in learning;
5. Generalization in learning;

Roy's first characteristic is attacked by many constructionist algorithms [4, 20]. MICA, a construction algorithm, answers not only Cybenko's question of how many nodes, but also meets all of Roy's characteristics for an intelligent learning algorithm. MICA is unique in that it designs a standard MLP network, not a unique network. This allows for direct learning comparison against backprop.

This paper suggests adding a sixth characteristic to Roy's list for intelligent learning algorithms, namely *Feedback for characterization of the feature data.* MICA excels in this area. MICA reports the number of nodes it takes to separate each pair of classes. One can ascertain the relative separation of the classes by how many nodes needed to separate them. The more nodes required, the more "complex" the decision boundary. This information can help in feature selection [15, 14, 18, 11]. For example, if feature A requires many more hidden nodes to separate the data than feature B, one can theorize feature B is better for separation. MICA also gives the number of inter-class data pairs which are separated after adding each node. This information is useful in determining

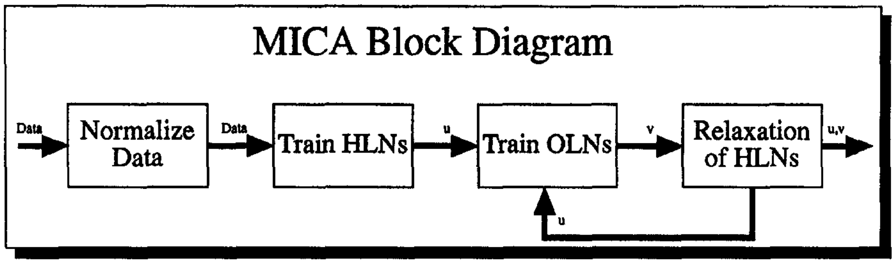


Fig. 1. This figure shows the major blocks of MICA. The hidden layer node (HLN) weights are designated by u and the Output Layer Node (OLN) weights are designated by v.

if part of the data sets overlap. MICA's capabilities also it to detect data which is not in general position. This is helpful in feature selection.

A high level block diagram is presented in Fig. 1. The normalization routine simply scales that data to a zero mean and standard deviation of one, while also augmenting the data with a column of ones. The Ho–Kashyap algorithm is central to training both the hidden layer nodes (TrainHLNs) and output layer nodes (TrainOLNs), and a brief overview is provided in Appendix A. The TrainHLNs and TrainOLNs blocks are discussed in Sections 2 and 3, respectively. The relaxation block is an attempt to increase the generalization ability of MICA and is discussed in Section 4. Test results are presented in Section 5 followed by concluding remarks in Section 6. Appendix B discusses the parallelization aspects of MICA.

## 2. Training hidden layer nodes

MICA trains hidden layer nodes (HLNs) in a four step process. First, MICA selects a pair of classes to separate. Second, MICA finds the inter-class data pair with the smallest Euclidean distance that has not yet been separated by a hyperplane. Third, MICA searches for the largest possible group of neighboring points around the inter-class data pair that can be separated by a single hyperplane. Fourth, the remaining unseparated inter-class data pairs are tested to see if the new hyperplane separates them. Two classes are separated when every inter-class data pair is separated. If there are unseparated data pairs the algorithm reverts to step 2, otherwise it returns to step 1. If all classes are separated the algorithm quits, see Algorithm 1.

### 2.1. Selecting class pairs

MICA separates classes on a pairwise basis. For example, in a three-class problem, data from classes one and two are separated, then the data from classes one and three,

**Algorithm 1.** TrainHiddenNodes

```
while more class pairs $i$ and $j$ do
  $Class_i \leftarrow ClasseData$ {Transfer all data pertaining to class $i$}
  $Class_j \leftarrow ClasseData$ {Transfer all data pertaining to class $j$}
  $\mathbf{D} \leftarrow distance(Class_i, Class_j)$ {Equation 2}
  while $\mathbf{D}_{l,m} \neq \infty \; \forall l,m$ do
    $[row, column] \leftarrow min(\mathbf{D})$
    $x_1 \leftarrow Class_i(row, :)$ {Transfer the data vector}
    $x_2 \leftarrow Class_j(column, :)$ {Transfer the data vector}
    $W_k \leftarrow AdaptNeighborhood(Class_i, Class_j, x_1, x_2)$
    $\mathbf{D} \leftarrow TestForSeparation(\mathbf{D}, W_k)$
  end while
end while
```

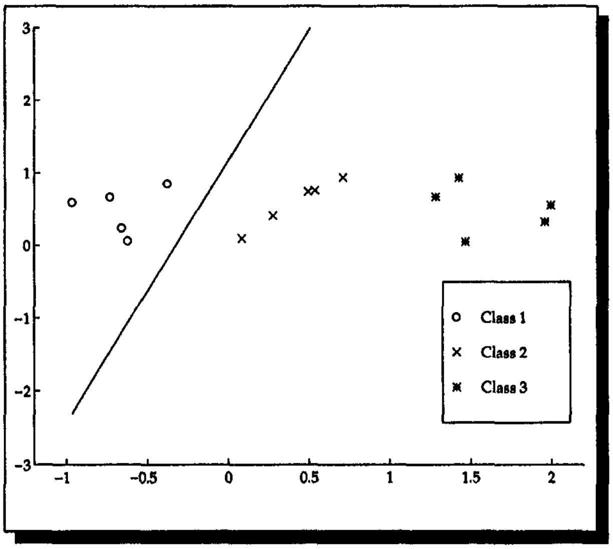


Fig. 2. This figure illustrates how a hyperplane can be reused when the base class remains the same.

and finally data from classes two and three. Separating all classes on a pair-wise basis is effective, but may be inefficient because of the redundant nodes. Fig. 2 illustrates how an HLN that separates classes one and two will also separate classes one and three. Thus, MICA has an option of reusing HLNs.

Reuse can be employed whenever the base class remains the same. The base class is the class with the smaller value and in the previous example would be class one.

In a three class problem, there will only be one chance of reusing HLNs, but in a four class problem that expands to three and in a five class problem it expands to six. Eq. (1) defines this value:

$$ReuseChances = \sum_{i=1}^{N-1} i - 1, \tag{1}$$

where $N$ is the number of classes. In larger class problems the number of reuse chances can become significant and lead to large reductions in the number of HLNs needed for shattering the data. Each time the base class stays the same, the number of HLNs involved in the reuse increases. If MICA is separating classes 1 and 3, it is only reusing HLNs that separated classes one and two. Whereas, if MICA is separating classes 1 and 4, it is reusing HLNs that separated classes 1 and 2 and HLNs that separated classes 1 and 3. However, when MICA separates classes 2 and 4, it is only reusing HLNs that separated classes 2 and 3. This is done for two reasons. First, it is illogical to assume that HLNs separating one pair of classes will be likely to separate two completely different classes. Secondly, HLNs are orientation specific and it will be confusing for the OLN training if an HLN separates class 1 low and class 2 high and separates class 3 high and class 4 low.

## 2.2. Selecting inter-class data pairs

MICA selects the inter-class data pair that are the closest together in Euclidean space. After the pair of classes are selected for separation, a distance matrix is calculated that contains a distance value between every pair of points in the two classes. This distance matrix is used to select the nearest neighbor inter-class data pairs and to keep track of which pairs have not yet been separated. Let **D** denote the distance matrix whose entries are the distances for each inter-class data pair:

$$\mathbf{D} = \begin{bmatrix} \|p_1^1 - p_1^2\| & \|p_1^1 - p_2^2\| & \cdots \|p_1^1 - p_m^2\| \\ \|p_2^1 - p_1^2\| & \|p_2^1 - p_2^2\| & \cdots \|p_2^1 - p_m^2\| \\ \vdots & \vdots & \vdots \\ \|p_n^1 - p_1^2\| & \|p_n^1 - p_2^2\| & \cdots \|p_n^1 - p_m^2\| \end{bmatrix}. \tag{2}$$

The indices of the smallest distance value in the matrix are used to select the nearest neighbor inter-class data pair. Let $(i,j)$ be the location of the smallest value in **D**. The corresponding points are derived from the class data as follows: $\mathbf{p}_i^1 = Class_i^1$ *and* $\mathbf{p}_j^2 = Class_j^2$. A distance value of zero indicates an inter-class data pair is not in general position. Each time a selection is made it is replaced with a large value. Thus, MICA adopts the convention that a symbol of $\infty$ in **D** indicates that an inter-class data pair is already separated. Thus, two classes are completely separated if $\mathbf{D}_{i,j} = \infty,\ \forall i,j$.

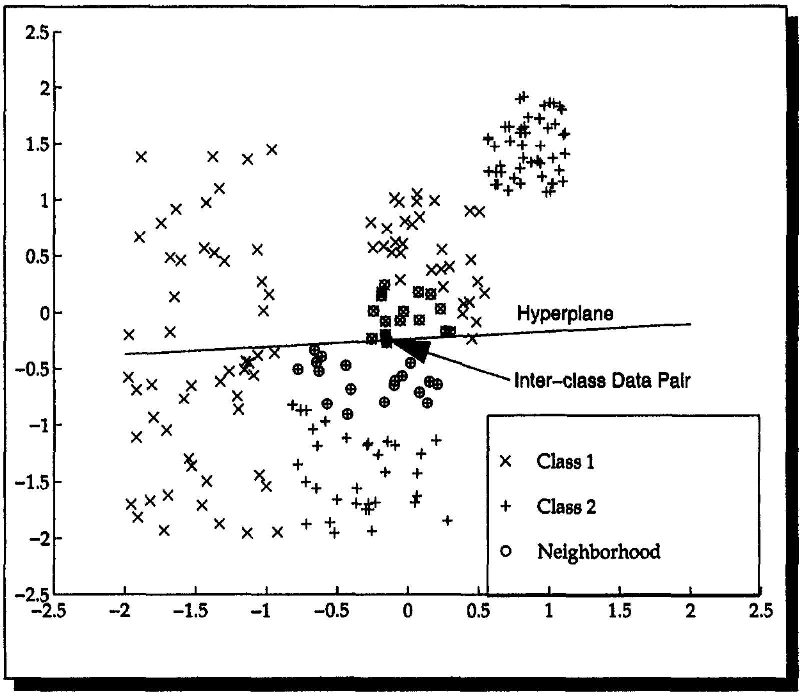


Fig. 3. This figure is a snapshot of MICA placing a hyperplane .

## 2.3. Selecting class neighborhoods

To achieve good generalization neighborhood selection is a crucial aspect of MICA. The larger the neighborhood, the better the generalization and the faster the algorithm converges. Fig. 3 is a snapshot of MICA training. This snapshot is taken as MICA is placing the first HLN. The hyperplane must separate the inter-class data pair pointed to by the arrow. The $\otimes$'s are the class 1 data in the neighborhood of the inter-class data pair member from class 1 and the $\oplus$'s are the class 2 data in the neighborhood of the inter-class data pair member from class 2. The hyperplane is trained using the Ho–Kashyap algorithm, see Algorithm 6, and uses the neighborhood data as input.

Neighbors are selected by first calculating the distances between the inter-class data pair members and the data from their respective classes. The data points whose distances are less than $x$ number of standard deviations away are designated in the neighborhood. Larger neighborhoods are better since, in general, they use fewer HLNs and result in less complex decision boundaries. However, if the neighborhood size is too large the data may not be linearly separable. Therefore, an adaptable neighborhood size algorithm is used, see Algorithm 2.

**Algorithm 2.** AdaptNeighborhood

```
$xstd \leftarrow x$
repeat
    $xstd \leftarrow xstd + \Delta std$
    $Neighborhood_1 \leftarrow GetNeighbors(Class_1, x_1, xstd)$
    $Neighborhood_2 \leftarrow GetNeighbors(Class_2, x_2, xstd)$
    $\mathbf{X} \leftarrow [Neighborhood_1; -Neighborhood_2]$ {This places the negative of the
        neighborhood 2 vectors under the neighborhood 1 vectors}
    $\mathbf{X}^\dagger \leftarrow (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$
    $weights \leftarrow HoKashyap(\mathbf{X}, \mathbf{X}^\dagger, \varepsilon)$ {Call to Ho–Kashyap Algorithm 6}
until $\neg separated(weights, Neighborhood_1, Neighborhood_2)$
repeat
    $xstd \leftarrow xstd - \Delta std$
    $Neighborhood_1 \leftarrow GetNeighbors(Class_1, x_1, xstd)$
    $Neighborhood_2 \leftarrow GetNeighbors(Class_2, x_2, xstd)$
    $\mathbf{X} \leftarrow [Neighborhood_1; -Neighborhood_2]$ {This places the negative of the
        neighborhood 2 vectors under the neighborhood 1 vectors}
    $\mathbf{X}^\dagger \leftarrow (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$
    $weights \leftarrow HoKashyap(\mathbf{X}, \mathbf{X}^\dagger, \varepsilon)$ {Call to Ho–Kashyap Algorithm 6}
until $separated\,(weight, Neighborhood_1, Neighborhood_2)$
```

This algorithm is a crude search process for finding the largest neighborhood sizes for a given inter-class data pair. The neighborhood size is initialized to some value and if the neighborhoods are separated by the resulting hyperplane, the neighborhood size grows until the neighborhoods are no longer separable. The neighborhood size is then reduced to the previous level where the neighborhoods were separable. If the initial neighborhood size was too large, the neighborhood size is stepped down until the neighborhoods are separable. As an efficiency note, one key aspect of MICA is its ability to glean that the neighborhoods are or are not separated by the hyperplane from inspection of the error from the Ho–Kashyap algorithm. Without this ability, considerably more processing time would be required to perform separation testing on all possible pair combinations from the two neighborhoods. The call to the Ho–Kashyap method, see Algorithm 6, is done with a large $\varepsilon$ value for efficiency reasons. The downside is that the neighborhood sizes may be smaller than would otherwise be the case, but the increase in efficiency is tremendous.

Fig. 4 show a series of snapshots as the neighborhood size adapts. In Fig. 4(a) the hyperplane separates the neighborhoods from the initial neighborhood size. In Fig. 4(b) the neighborhood size is bigger but the hyperplane still separates the neighborhoods. In Fig. 4(c) the size is too large and the hyperplane cannot separate the neighborhoods. In Fig. 4(d) the neighborhood reverts to the last correctly separated neighborhood size and thus the hyperplane separates the neighborhoods again.

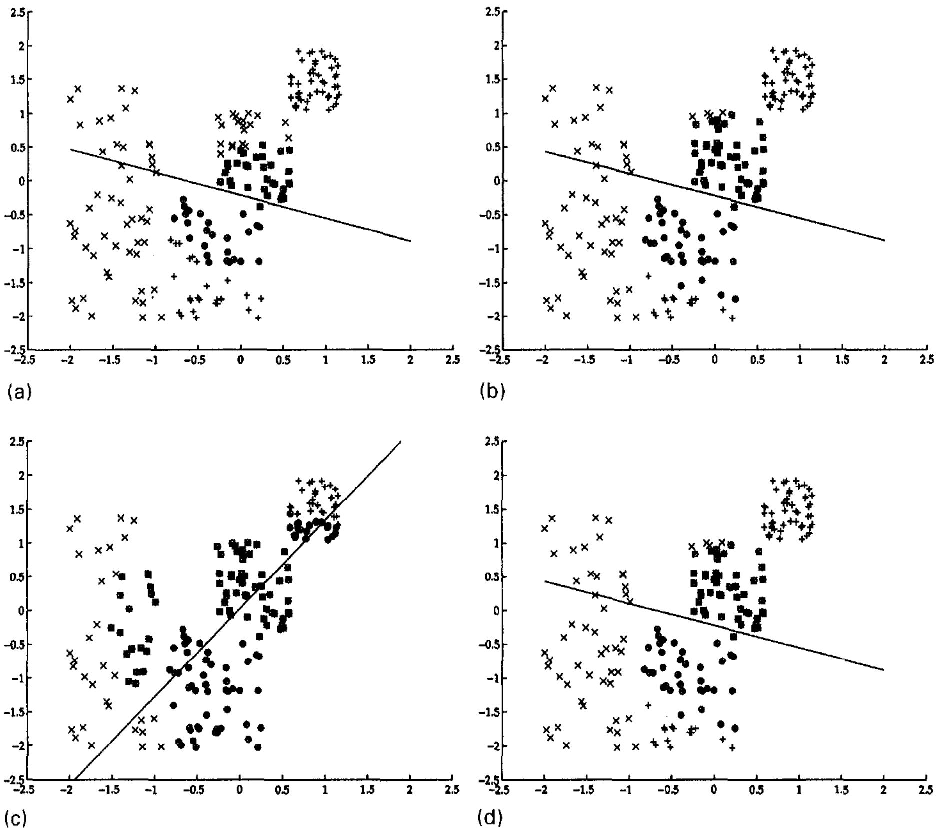


Fig. 4. This figure shows a series of snapshots as the adaptive neighborhood size algorithm trains. The hyperplane initially separated the neighborhoods in (a) and also after the first increase in (b), but not after the second increase in (c). Therefore the size reverts back to the last size where the hyperplane did separate in (d). (a) Initial Neighborhood size, hyperplane separates. (b) Neighborhood size increases, hyperplane separates. (c) Neighborhood size increases, hyperplane does not separate. (d) Neighborhood size decreases, hyperplane separates.

Another key to generalization is for the hyperplanes to be roughly halfway in between the two inter-class data pairs. If the above neighborhood selection approach does not accomplish this goal, then the neighborhood defaults to a radius around each inter-class data point that is slightly less than the distance between the two inter-class data points. If this approach does not result in a good hyperplane, then, as a fail safe, the vector normal to the line connecting the two inter-class data points is used to define the separating hyperplane, see Eq. (3).

$$(p_1 - p_2)\cdot(X - p_m) = 0, \tag{3}$$

where $p_m = \frac{1}{2}(p_1 + p_2)$ is the midpoint between the inter-class data points $p_1$ and $p_2$.

### 2.4. Testing for separability

A hyperbolic tangent activation function makes determining separability straightforward. MICA tests for opposite polarity of the function output

$$|y_j^1 + y_j^2| = |y_j^1| + |y_j^2|, \tag{4}$$

where $y_j^1$ and $y_j^2$ are the scalar outputs for the $j$th HLN for data points from two different classes. If the equality in Eq. (4) is true, then the points are separated by the hyperplane. Once a hyperplane has separated two points it becomes orientation specific. Consider a hyperplane that separates class 1 to the positive side and class 2 to the negative side, then that hyperplane can only separate other inter-class data pairs if they fall on the correct side of the hyperplane.

Using the **D** matrix to determine when two classes are separated is effective, yet inefficient, because the inter-class data pair that has the smallest distance may already be separated. Therefore, each time an HLN is produced, each inter-class data pair in the **D** matrix is tested to see if the new HLN separated them. Inter-class data pairs already marked "separated" are skipped each round and processing is expedited for each new HLN produced. There is one shortcut that is employed for bypassing some of the testing of the **D** matrix. When a new HLN is produced, it must separate the points of the inter-class data pair and their respective neighborhoods. All possible inter-class data pair combinations from the two neighborhoods can be marked separated without having to test them for verification. Thus, the **D** matrix can be changed by the process of finding the nearest neighbor inter-class data pairs, the process of finding the HLNs, and the process of testing for separability.

## 3. Training output layer weights

Since HLN weights are likely to project the training data into a linearly separate space, the output layer node (OLN) weights are determined using the Ho–Kashyap method, see Algorithm 6. MICA trains one OLN per class. The **X** matrix changes slightly to be class $i$ and NOT class $i$ as shown below. The $\boldsymbol{t}$ vector is of length $n+m$, where $m$ is the number of all vectors not in class $i$. The $\boldsymbol{t}$ vector is initially set to all ones. Any positive value will work, but the lower values tend to converge to a solution faster.

$$\mathbf{X} = \begin{bmatrix} x^{i}_{1,1} & x^{i}_{1,2} & \cdots & x^{i}_{1,k} & 1 \\ x^{i}_{2,1} & x^{i}_{2,2} & \cdots & x^{1}_{2,k} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x^{i}_{n,1} & x^{i}_{n,2} & \cdots & x^{i}_{n,k} & 1 \\ -x^{\neg i}_{1,1} & -x^{\neg i}_{1,2} & \cdots & -x^{\neg i}_{1,k} & -1 \\ -x^{\neg i}_{2,1} & -x^{\neg i}_{2,2} & \cdots & -x^{\neg i}_{2,k} & -1 \\ \vdots & \vdots & & \vdots & \vdots \\ -x^{\neg i}_{m,1} & -x^{\neg i}_{m,2} & \cdots & -x^{\neg i}_{m,k} & -1 \end{bmatrix}$$

The Ho–Kashyap algorithm iterates until the error equals zero. If a solution is found for each OLN the training is completed and the entire set of training data is guaranteed to be classified with 100% accuracy. The TrainOLNs Algorithm is shown below (Algorithm 3).

**Algorithm 3.** TrainOutputNodes

```
$\mathbf{X} \leftarrow -[Data, \mathbf{1}^t]$ {1 is a vector of ones to augment the data}
for $i = 1 : NumClasses$ do
    $\mathbf{X}_i \leftarrow -\mathbf{X}_i$ {This makes the data for class i positive and the rest negative}
    $\mathbf{X}^\dagger \leftarrow (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$
    $weights_i \leftarrow HoKashyap(\mathbf{X}, \mathbf{X}^\dagger, \varepsilon)$ {Call to Ho–Kashyap Algorithm 6}
end for
```

Algorithm 3 is guaranteed to produce OLNs that will correctly classify the data if the data are projected into a linearly separable space; however, the convergence of the Ho–Kashyap algorithm may take many iterations. Often a good solution can be found quickly with the error driven to within an epsilon of zero. Algorithm 4 takes advantage of this by checking classification results periodically for 100% accuracy. If the accuracy is less than 100% then the epsilon is lowered and the training resumes using the previous $\boldsymbol{w}$ and $\boldsymbol{t}$ vectors. With $\varepsilon$ initially set to $5 \times 10^{-3}$ this algorithm usually converges in one iteration for the data used in this article.

**Algorithm 4.** FastTrainOutputNodes

```
$\varepsilon \leftarrow 5 \times 10^{-3}$
$Accuracy \leftarrow 0$
while $Accuracy \neq 100$ do
    $weights \leftarrow TrainOutputNodes(Data, \varepsilon)$ {Call to TrainOutputNodes Algorithm 3}
    $Accuracy \leftarrow TestForAccuracy(Data, weights)$
    $\varepsilon \leftarrow \varepsilon/10$
end while
```

## 4. Relaxation

The need for pruning nodes in neural networks is prevalent because the number of HLNs needed is an unsolved problem. A number of pruning techniques are discussed in [12]. Typically, networks are trained with the number of HLNs over estimated high and then pruned for the best results. Alternatively, construction algorithms start with smaller networks and grow the network for best results. MICA is of the latter category, but it does not know when to stop because it is designed to add HLNs until the training data is completely separated. This may have the adverse effect of lowering the generalization of the test data. This is typically called memorizing in MLP networks trained with backprop when the network is trained too long or uses too many HLNs. MICA cannot drive the error to zero since the training error is not known until after the OLNs are calculated. Hence, the strategy is to first grow the network using the training data, then prune the network until the test set accuracies stops increasing, see Algorithm 5. The pruning is accomplished by removing one HLN at a time starting with the least important node and moving toward the most important node. The importance of the HLNs is measured by the combined size, and number of points of the neighborhoods of the inter-class data pairs used to produce the HLNs. This strategy is predicated by the theory that HLNs that separate the smallest numbers of points may be used to memorize overlapping areas of the data. The results show this pruning strategy is effective and keeps the added FLOPS to a minimum.

**Algorithm 5.** Relaxation

```
$NewAccuracy \leftarrow TestAccuracy$
$[NumPointsSorted, Index] \leftarrow Sort(NumPoints)$ {Index is a mapping back to the
  original order of the data}
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $ReducedWeights^1 \leftarrow Weights^1(:, Index(i+1 : NumHLNs))$ {Transfers HLN
    weight values for all HLN whose importance value ranks higher than $i$}
  $ReducedWeights^2 \leftarrow FastTrainOutputNodes(ReducedWeights^1, TrainData)$
    {Call to OLN training algorithm 4}
  $OldAccuracy \leftarrow NewAccuracy$
  $NewAccuracy \leftarrow TestForAccuracy(TestData, ReducedWeights)$
until $NewAccuracy < OldAccuracy$
$i \leftarrow i - 1$
$Weights^1 \leftarrow Weights^1(:, Index(i+1 : NumHLNs))$
$Weights^2 \leftarrow TrainOLN(ReducedWeights^1, Data)$
```

## 5. Test results

Experiments were run to test MICA's different learning schemes and MICA's ability to define an appropriate network architecture for Backprop. There are four data sets in the tests. The spiral data is two streams of data spiraling out from the center [8]. The mesh data is a three-class problem with blocks of data situated adjacent to each other with some class overlap. Iris data is a common three class problem that attempts to recognize flower types from features measured from the flower's petals. The NIST Optical Character Recognition (OCR) data set is a 10 class problem trying to recognize the digits 0–9. There are 27 spectral features used in the OCR problem. The seven experiments listed below were run on each data set.

Table 1
Spiral data set results

| Algorithm | Points | Nodes | Epochs | Training | Test | Overall | Flops |
|---|---|---|---|---|---|---|---|
| MICA | 146 | 22.88 | NA | 100% | 63.75% | 81.87% | $2.9136 \times 10^7$ |
| MICA/relax | 146 | 22.24 | NA | 98.57% | 65.42% | 81.99% | $5.9171 \times 10^7$ |
| MICA/reuse | 146 | 22.88 | NA | 100% | 63.75% | 81.87% | $2.9136 \times 10^7$ |
| Backprop | 146 | 22.88 | 300 | 63.39% | 52.17% | 57.78% | $5.0117 \times 10^7$ |
| Backprop/fixed | 146 | 25 | 300 | 64.08% | 49.83% | 56.96% | $5.4548 \times 10^7$ |
| Backprop/HLN | 146 | 22.88 | 100 | 99.55% | 60.50% | 80.03% | $3.1547 \times 10^7$ |
| Backprop/HLN/OLN | 146 | 22.88 | 50 | 96.61% | 62.50% | 79.56% | $3.6657 \times 10^7$ |

Table 2
Mesh data set results

| Algorithm | Points | Nodes | Epochs | Training | Test | Overall | Flops |
|---|---|---|---|---|---|---|---|
| MICA | 300 | 17.24 | NA | 100% | 94.05% | 97.02% | $1.0436 \times 10^8$ |
| MICA/relax | 300 | 15.28 | NA | 99.88% | 94.59% | 97.23% | $2.4684 \times 10^8$ |
| MICA/reuse | 300 | 16.04 | NA | 100% | 94.09% | 97.04% | $1.0525 \times 10^8$ |
| Backprop | 300 | 17.24 | 300 | 98.42% | 94.69% | 96.55% | $1.7258 \times 10^8$ |
| Backprop/fixed | 300 | 15 | 300 | 98.37% | 94.79% | 96.58% | $1.5101 \times 10^8$ |
| Backprop/HLN | 300 | 17.24 | 100 | 99.29% | 93.71% | 96.5% | $1.2104 \times 10^8$ |
| Backprop/HLN/OLN | 300 | 17.24 | 50 | 100% | 94.21% | 97.1% | $1.3194 \times 10^8$ |

- MICA – MICA without pruning relaxation or reuse;
- MICA/relax – MICA with pruning relaxation (to reduce number of HLNs);
- MICA/reuse – MICA without relaxation but with HLN reuse (to prevent unnecessary HLNs);
- Backprop – Backprop using architecture found from MICA (Number of HLNs);
- Backprop/fixed – Backprop using a fixed number of hidden nodes;
- Backprop/HLN – Backprop started with the HLN weights found in MICA;
- Backprop/HLN/OLN – Backprop started with the HLN and OLN weights found in MICA.

Tables 1–4 contain the results of all the experiments for each data set. The results for the Spiral, Mesh, and Iris data are the average results from 25 test runs. In each test run, a random training and test set were selected prior to the algorithms being run on the data. The OCR test results are the average of ten test runs.

## *5.1. Spiral*

Table 1 indicates the spiral test results. This data set is used to demonstrate a class of problems where MICA is far Superior to that of Backprop. Backprop has

Table 3
Iris data set results

| Algorithm | Points | Nodes | Epochs | Training | Test | Overall | Flops |
|---|---|---|---|---|---|---|---|
| MICA | 75 | 7.08 | NA | 100% | 95.37% | 97.69% | $1.5462 \times 10^7$ |
| MICA/relax | 75 | 3.52 | NA | 98.99% | 97.28% | 98.8% | $1.8767 \times 10^7$ |
| MICA/reuse | 75 | 6.08 | NA | 100% | 95.14% | 97.57% | $1.4979 \times 10^7$ |
| Backprop | 75 | 7.08 | 300 | 99.15% | 96.16% | 97.65% | $2.3116 \times 10^7$ |
| Backprop/fixed | 75 | 8 | 300 | 99.23% | 95.92% | 97.58% | $2.5895 \times 10^7$ |
| Backprop/HLN | 75 | 7.08 | 100 | 100% | 95.29% | 97.65% | $2.1093 \times 10^7$ |
| Backprop/HLN/OLN | 75 | 7.08 | 50 | 99.96% | 95.45% | 97.71% | $1.9293 \times 10^7$ |

Table 4
OCR data set results

| Algorithm | Points | Nodes | Epochs | Training | Test | Overall | Flops |
|---|---|---|---|---|---|---|---|
| MICA | 3471 | 62.3 | NA | 100% | 95.65% | 97.76% | $1.4516 \times 10^{10}$ |
| MICA/relax | 3471 | 61.2 | NA | 100% | 95.81% | 97.91% | $1.8569 \times 10^{10}$ |
| MICA/reuse | 3471 | 55.4 | NA | 100% | 95.52% | 97.76% | $1.7283 \times 10^{10}$ |
| Backprop | 3471 | 62.3 | 300 | 99.63% | 96.04% | 97.83% | $1.8038 \times 10^{10}$ |
| Backprop/fixed | 3471 | 60 | 300 | 99.70% | 95.85% | 97.77% | $1.7379 \times 10^{10}$ |
| Backprop/HLN | 3471 | 62.3 | 100 | 99.80% | 95.98% | 97.89% | $1.6359 \times 10^{10}$ |
| Backprop/HLN/OLN | 3471 | 62.3 | 50 | 100% | 96.05% | 98.02% | $1.7568 \times 10^{10}$ |

difficulty training weights for data sets with areas of low Mean Squared Error (MSE) [8]. The results show that MICA achieves 100% accuracies on the training data, while Backprop's accuracies are approximately 60%, after only 300 epochs. Other results indicate that 150 000–200 000 epochs were needed for Backprop to solve the spiral problem [8]. MICA does not perform as well on the test set for two reasons. The points in the data set are not very dense and the training and test sets are chosen at random. Therefore, it is not likely that every other point is split between the two data sets. This allows whole arcs to be in one set or the other. Thus, the hyperplanes in MICA may be drawn to cut off whole arcs in the test set. The Backprop/HLN produces an important result. Backprop, given a good set of separating hyperplanes, can successfully learn the output layer weights to separate the data. This is the first result of its kind. The Backprop/HLN/OLN algorithm starts Backprop in an extremely confined area of the weight search space. This algorithm is unlikely to escape its starting valley, but it does act as a local search should there be a smaller minimum to find. It is interesting to note that although Backprop lowers the MSE of the weights found in MICA the classification accuracies did not increase. Fig. 5 shows the results of four of these algorithms on the entire data set.

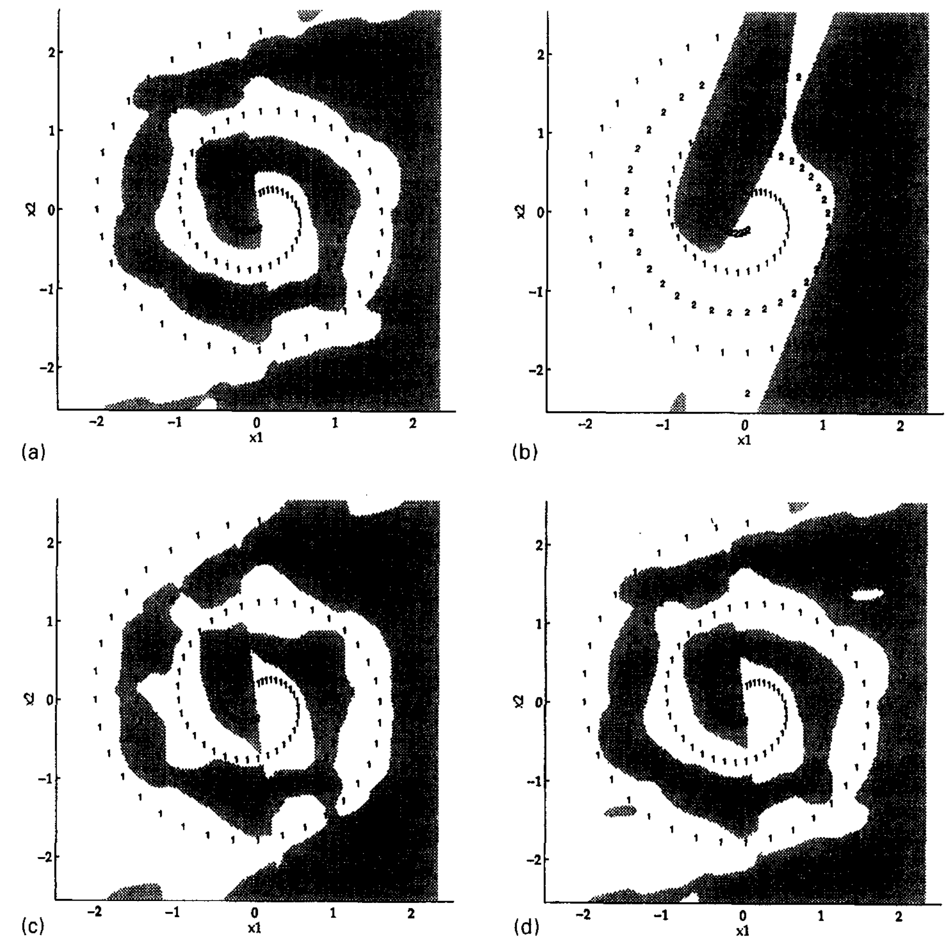


Fig. 5. The complete $x$–$y$ coordinate grid is fed into four different MLP networks trained using (a) MICA, (b) Backprop with momentum, (c) Backprop/HLN, which seeds Backprop with the HLN weights from MICA, and (d) Backprop/HLN/OLN which seeds Backprop with all the weights from MICA. (a) MICA on Spiral Data – 100% accuracy. (b) Backprop on Spiral Data – 61.2% accuracy. (c) Backprop/HLN on Spiral Data – 100% accuracy. (d) Backprop/HLN/OLN on Spiral Data – 100% accuracy.

Each algorithm is trained on the same data. MICA required 27 HLNs to correctly classify the data (Fig. 5(a)). Backprop was also given 27 HLNs (Fig. 5(b)), but after 300 epochs the MSE mostly oscillated and barely changes, see Fig. 6 top line. The Backprop/HLN algorithm (Fig. 5(c)) classifies the data correctly. In this case, Backprop quickly adjusts the output level weights while only slightly modifying the hidden level weights. As expected, this error is smooth and drops quickly, see Fig. 6 middle line. Notice that the random weights initially have a lower MSE than the partially seeded weights. The MSE for the fully seeded weights starts at a minimum level and decreases slightly, and the output has only minor changes from MICA's (Fig. 5(d)).

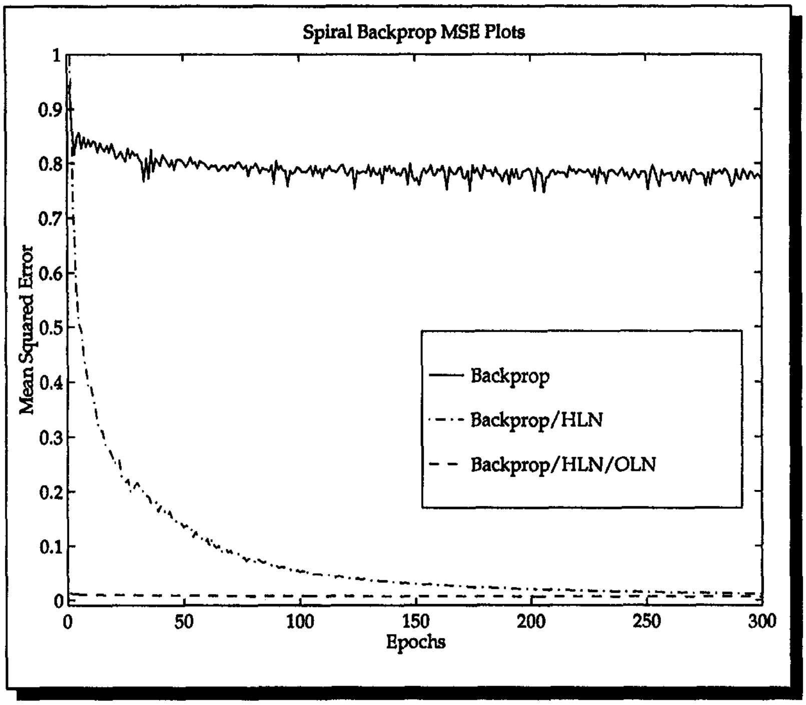


Fig. 6. Training error for the three Backprop algorithms.

## 5.2. Mesh

Table 2 reports the Mesh test results. The data set is used because it favors Backprop. Backprop's strength is defining hyperplanes for data that have all high areas of MSE. MICA achieves results slightly less accurate Backprop for the test set. MICA with relaxation achieves the same results on the test set without losing significant accuracy on the training set and achieves an overall accuracy higher than that of Backprop. Relaxation, on average, reduced the number of HLNs by two nodes. In one case it reduced the number of HLNs by 7, see Fig. 7 to see how the output space changes coloring. In contrast to its performance on the spiral data set, the Backprop/HLN/OLN algorithm improves upon MICA's performance.

Fig. 7 illustrates the decision boundaries before and after relaxation. Before relaxation the boundaries are sharper, see Fig. 7(a) and (b). After relaxation, see Fig. 7(c) and (d), the boundaries more closely resemble Backprop's, see Fig. 7(e) and (f). This suggests that the difference in MICA and Backprop is MICA's ability to define the HLNs in areas of low MSE. In addition, this figure shows that the test data are missed because data points were along the boundaries. Relaxation picks up some of the missed points because the new decision boundaries are not as tight to the training data.

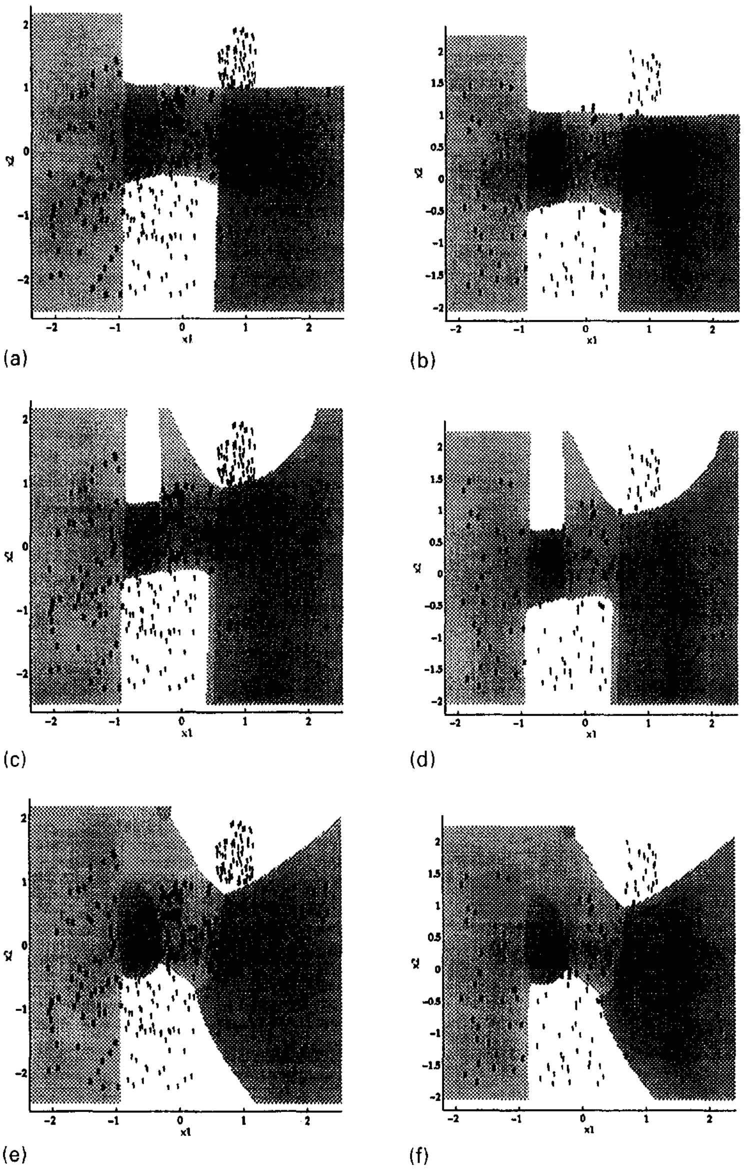


Fig. 7. The complete $x$–$y$ coordinate grid is fed into three different MLP networks trained using MICA, MICA, and Backprop, these results are overlayed with the training data in (a), (c) and (e) and with the test data in (b), (d), and (f). (a) MICA on train data – 100% accuracy. (b) MICA on test data – 91.04% accuracy. (c) MICA/Relax on train data – 99.25% accuracy. (d) MICA/Relax on test data – 92.54% accuracy. (e) Backprop on train data – 98.75% accuracy. (f) Backprop on test data – 89.55% accuracy.

## 5.3. *Iris*

Table 3 contains the Iris test results. The Iris data highlights the MICA/Relax algorithm's ability to generalize. In the Iris data, classes 1 and 2 are linearly separable as are classes 1 and 3; but classes 2 and 3 overlap. This information is derived for the MICA algorithm. Different choices of the train and test data set can cause the test results to vary between 80% to 100% accuracy. Again, MICA is slightly under Backprop for the test data, but MICA with Relaxation is two points better than MICA and one point higher than Backprop. The relaxation algorithm removes HLNs used to memorize the overlapping data and improves generalization. In trial 18, relaxation reduced the number of HLNs from 10 to 1 and increased test set accuracy from 96% to 100%. The Backprop/HLN/OLN improves upon MICA, but MICA appears to be so adept at learning weights near the bottom of a local extrema that there is little room for Backprop to maneuver.

## 5.4. *OCR*

Table 5 shows the OCR test results. This data set is used for two purposes. With its 10 classes and 3471 vectors, it is a suitable test for stressing the algorithm. It is also a good test of the reuse feature which affords 36 opportunities for reusing existing HLNs, see Eq. (1). This data test has been a barometer of the evolution of MICA. Originally, the algorithm used a genetic algorithm to find the weights for the HLNs. This approach took about 250 HLNs and resulted in test set accuracies in the 90% range. Then using the Ho–Kashyap method and a fixed neighborhood size, the HLNs reduced to around 160 and test accuracies of 92%. Finally with the variable size neighborhood the HLNs reduced further to approximately 60 and the test accuracies rose to the 95% range. The HLN reuse has further reduced the number of HLNs to an average of 55.4, with the test set accuracies just about equal with MICA. The relaxation technique is not as effective as reuse in reducing HLNs, but it does achieve a higher test set accuracy. The Backprop algorithm has similarly improved its accuracies as MICA has improved because Backprop is architecture sensitive. As MICA improved its ability to define the upper bound on the number of HLNs needed to shatter the data, Backprop improved its performance by training on a better network architecture. There appears to be a strong correlation at the macro level between the number of HLNs used and the test set accuracies, but only a small correlation at the micro level. That is, nets that have grossly too many or too few perform much worse than nets with the appropriate number. However, nets that vary by only a small amount of HLNs perform about the same. This is illustrated by the Backprop and Backprop/fixed tests.

## 5.5. *Overall*

MICA always achieved 100% recognition accuracies on the training data. When the data set is favorable to Backprop, MICA achieves similar accuracies as Backprop on the test data. When the data set is not favorable to Backprop, MICA achieves far better results than Backprop. The relaxation is effective in increasing the test set accuracies. The reuse algorithm is capable of reducing the number of HLNs needed on larger class data sets. The Backprop/HLN algorithms allows Backprop to solve a new class of problems. The Backprop/HLN/OLN can achieve modest increases over MICA as it performs a local search.

## 6. Conclusions

MICA is both effective and efficient as a learning algorithm for classification problems using MLP networks. MICA realizes the Cybenko network for classification and is particularly well-suited for data sets containing areas of low mean squared error. MICA is also a value-added tool for improving the use of Backprop. It can be used to determine the number of hidden nodes needed, seed the lower level weights or seed all the weights. In the case of low MSE problems like the spiral data set, MICA is an enabling technology for Backprop to be effective.

## Acknowledgements

The first author thanks all the coauthors for their insights and comments. In addition, thanks are due to the reviewers for helping to improve the presentation.

## Appendix A. Ho–Kashyap method

There are different methods for finding a hyperplane to separate linearly separable data sets; however, the Ho–Kashyap method has several desirable qualities. Ho–Kashyap's method is guaranteed to converge in a finite number of iterations if the data are linearly separable, via the convergence proof in [7, pp. 161–163]; and the method also has an indicator if the data are not linearly separable. The Ho–Kashyap method is a gradient descent algorithm, whereby it minimizes $\|\mathbf{X}\boldsymbol{w} - \boldsymbol{t}\|$, where $\mathbf{X}$ is a matrix containing the data and $\boldsymbol{w}$ and $\boldsymbol{t}$ are to be determined. If the data are linearly separable, then there exist a $\boldsymbol{w}_0$ and $\boldsymbol{t}_0$ such that

$$\mathbf{X}\boldsymbol{w}_0 = \boldsymbol{t}_0, \qquad \boldsymbol{t}_0 > 0, \tag{5}$$

where $\mathbf{X}$ is the augmented data matrix shown below for a two-class problem with $n$ data vectors in class 1 and $m$ data vectors in class 2. This is standard notation in Pattern Recognition books, see [7]. $\boldsymbol{w}$ is the discriminant vector, and $\boldsymbol{t}$ consist of all positive elements, but is otherwise arbitrarily set.

$$\mathbf{X}=\begin{bmatrix} x^1_{1,1} & x^1_{1,2} & \cdots & x^1_{1,k} & 1 \\ x^1_{2,1} & x^1_{2,2} & \cdots & x^1_{2,k} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x^1_{n,1} & x^1_{n,2} & \cdots & x^1_{n,k} & 1 \\ -x^2_{1,1} & -x^2_{1,2} & \cdots & -x^2_{1,k} & -1 \\ -x^2_{2,1} & -x^2_{2,2} & \cdots & -x^2_{2,k} & -1 \\ \vdots & \vdots & & \vdots & \vdots \\ -x^2_{m,1} & -x^2_{m,2} & \cdots & -x^2_{m,k} & -1 \end{bmatrix}$$

The superscript indicates class. In the Ho–Kashyap method, both $\boldsymbol{w}$ and $\boldsymbol{t}$ are allowed to vary with $\boldsymbol{t}$ required to stay positive. Therefore, $\boldsymbol{t}$ is constrained to vary only in a positive manner. See Algorithm 6 for an outline of the algorithm.

**Algorithm 6. Ho–Kashyap method**

```
$t_1 \leftarrow 1$ {This initial value is arbitrary but must be greater than 0}
$w_1 \leftarrow \mathbf{X}^\dagger t_1$ {X is training data, setup according to the problem, † defined below}
$e_1 \leftarrow \mathbf{X}w_1 - t_1$
$i \leftarrow 1$
while $e_i \neq \varepsilon$ do
    $t_{i+1} \leftarrow t_i + \rho(e_i + |e_i|)$ {$\rho$ is an update parameter, $0<\rho<1$}
    $w_{i+1} \leftarrow w_i + \rho\mathbf{X}^\dagger |e_i|$ {$\mathbf{X}^\dagger = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$ is the left pseudo-inverse}
    $e_i \leftarrow \mathbf{X}w_i - t_i$
    $i \leftarrow i + 1$
end while
```

The $\mathbf{X}$, $\mathbf{X}^\dagger$, *and* $\varepsilon$ values are provided as inputs. The Ho–Kashyap algorithm calls for $\varepsilon$ to be zero, but larger values can be used. For a complete derivation of the algorithm and proof of convergence if given linearly separable data see [7, pp. 159–166].

## Appendix B. Speedup

MICA is ideally suited for implementation in a parallel environment whereas Backprop is not. In a parallel environment, all pairs of classes can be separated simultaneously. Likewise, all OLNs can be trained simultaneously. There are well known algorithms for matrix operations [10] that can be employed if the matrix sizes are large enough to warrant their usage. The processing time is the time it takes to separate the hardest pair of classes (requires the most HLNs) plus the time it takes to train the hardest OLN (takes the most iterations) plus some overhead and communications time.

While this will provide a dramatic speedup, it is possible to do even better. Recall that the **D** matrix is used to keep track of separated inter-class data pairs and that MICA needs to test each non-separated inter-class data pair after each HLN is produced for efficiency reasons. In a parallel environment, MICA could forego the need to test the **D** matrix after each HLN is produced and overlap the HLN and OLN training. In this architecture, after each HLN is produced, it is given to the OLN training processors. Every time an OLN processor receives a new HLN it discards its previous training and begins again. In the beginning this will frequently be the case, but eventually new HLNs will stop being produced and the OLN training will be valid. It is likely that most of the HLNs will be produced from inter-class data pairs found from the smaller values in the **D** matrix which are tested earlier than the larger values. Thus, there can be a significant period during the overlap training that the OLN processors are producing valid solutions. For example, suppose you had 100 training samples from each of two classes. There will be 10 000 inter-class data pairs in the **D** matrix. Suppose, 10 HLNs are needed to separate the data. Furthermore, suppose the 10 inter-class data pairs that are used to find the ten HLNs are found in the first 500 inter-class data pairs tried. While MICA is busy iterating through the rest of the 9500 inter-class data pairs, the OLN processors would be busy finding a valid solution or be finished. This process will be faster if the **D** matrix is pre-sorted. The processing time may be as fast or faster than the time to separate the hardest pair of classes.

## References

[1] E.B. Baum, On the capabilities of multilayer preceptrons, J. Complexity 4 (1988) 193–215.
[2] E.B. Baum, What size net gives valid generalization, Neural Computation 1 (1989) 151–160.
[3] E.B. Baum, D. Haussler, What size net gives valid generalization, Neural Comput. 1 (1) (1989) 151–160.
[4] N. Burgess, A constructive algorithm that converges for real-valued input patterns, J. Neural Systems 5 (1) (1994) 59–66.
[5] Cover, M. Thomas, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, IEEE Trans. Electronic Comput. 22 (3) (1965) 326–334.
[6] G. Cybenko, Approximation by superposition of sigmoidal functions, Math. Control Signals Systems 2 (1989) 303–314.
[7] R.O. Duda, E.H. Peter, Pattern Classification and Scene Analysis, Wiley, New York, 1973.
[8] S.E. Fahlman, L. Christian, The cascade-correlation learning architecture, in: D.S. Touretzky (Eds.), Advances in Neural Information Processing Systems, vol. 2, San Mateo, Morgan Kaufmann, CA, 1990, pp. 524–532.
[9] A.K. Jain, M. Jianchang, Neural networks and pattern recognition, in: J.M. Zurada, et al. (Eds.), Computational Intelligence Imitating Life, Chap. 4, IEEE Press, New York, 1994, pp. 194–212.
[10] V. Kumar, et al., Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms, Benjamin/Cummings, Menlo Park, CA, 1994.
[11] K.L. Priddy, Feature extraction and classification of FLIR imagery using relative locations of non-homogeneous regions with feedforward neural networks, Ph.D. dissertation, Air Force Institute of Technology, WPAFB, OH, 1992.
[12] R. Reed, Pruning algorithms – a survey, Neural Networks 4 (5) (1993) 740–747.

[13] A. Roy, et al., An algorithm to generate radial basis functions (RBF)-like nets for classification problems, Neural Networks 8 (2) (1995) 179–201.

[14] D.W. Ruck, Characterization of multilayer perceptrons and their application to multisensor automatic target detection, Ph.D. dissertation, Air Force Institute of Technology, WPAFB, OH, 1990.

[15] D.W. Ruck, R. Steven, Feature selection using a multilayer perceptron, J. Neural Network Comput. (Fall 1990) 40–48.

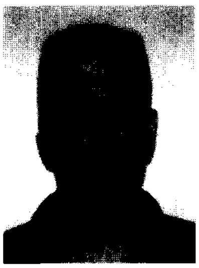
[16] E.D. Sontag, Sigmoids distinguish more efficiently than heaviside, Neural Comput. 1 (1989) 470–472.

[17] E.D. Sontag, Feedforward nets for interpolation and classification, J. Comput. Systems Sci. 45 (1992) 20–48.

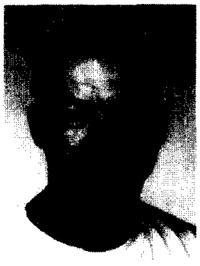
[18] G.L. Tarr, Multi-layered feedforward neural networks for image segmentation, Ph.D. dissertation, Air Force Institute of Technology, WPAFB, OH, 1991.

[19] V.N. Vapnik, Estimation of Dependences Based on Empirical Data, Springer, New York, 1982.

[20] J. Yang et al., M-TILING – A constructive neural network learning algorithm for multi-category pattern classification, In: Proc. World Congress on Neural Networks, vol. 961, INNS Press, San Diego, CA, 1996, pp. 182–187.



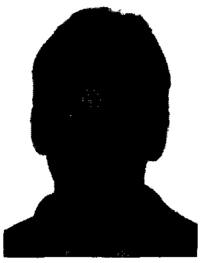
**Thomas F. Rathbun**, Captain, USAF, is currently pursuing a Ph.D. in the Department of Electrical and Computer Engineering at the Air Force Institute of Technology, Wright–Patterson Air Force Base, OH, USA. He received a B.S. in Computer Science Engineering from Norwich University in 1986, and an M.S. in Computer Engineering from AFIT in 1991.



**Steven K. Rogers** is a Professor of Electrical Engineering in the Department of Electrical and Computer Engineering at the Air Force Institute of Technology (AFIT), Wright–Patterson Air Force Base, OH, USA.

He is presently conducting an extensive research program on neural networks and pattern recognition. The research program addresses the problems inherent in making smart weapons.

He has published more than 200 papers in the areas of neural networks, pattern recognition, and optical information processing and several textbooks including *Introduction to Biological and Artificial Neural Networks for Pattern Recognition.* He is a SPIE Fellow.



**Martin P. Desimio** was born in Burbank, CA, USA in 1960. He received the B.S. degree with highest honors from Wright State University, Dayton, OH, USA, in 1983, the M.S.E.E. degree from the Air Force Institute of Technology, Wright–Patterson Air Force Base, OH, USA, in 1987, and the Ph.D. in electrical engineering from the University of Dayton, OH, USA, in 1993.

He joined the Department of Electrical and Computer Engineering at the Air Force Institute of Technology in 1989 and is currently an Assistant Professor. He teaches graduate courses in digital signal processing and pattern recognition. His research interests include speech recognition, speaker identification and verification, and target recognition.

Dr. DeSimio is a member of Eta Kappa Nu and Tau Beta Pi.

**Mark E. Oxley** received the B.S. degree in mathematics in 1978 from Cumberland College, Williamsburg, KY, USA, the M.S. degree in Applied Mathematics in 1980 from Purdue University, West Lafayette, IN, USA, and the Ph.D. degree in Mathematics in 1987 from North Carolina State University, Rayleigh, NC, USA.

Since 1987, he has been with the Graduate School of Engineering at the Air Force Institute of Technology, Wright–Patterson, OH, where he is an Associate Professor of Mathematics in the Department of Mathematics and Statistics. His current research interests include neural networks, wavelet analysis, functional analysis, and nonlinear partial differential equations.

Dr. Oxley is a member of Pi Mu Epsilon, the American Mathematical Society (AMS), the Society for Industrial and Applied Mathematics (SIAM), the American Geophysical Union (AGU), and SPIE the International Society for Optical Engineering.