INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Incremental Class Learning approach and its application to Handwritten Digit Recognition

Jacek Mańdziuk* and Lokendra Shastri

TR-98-015

June 1998

Abstract

Incremental Class Learning (ICL) provides a feasible framework for the development of scalable learning systems. Instead of learning a complex problem at once, ICL focuses on learning subproblems incrementally, one at a time — using the results of prior learning for subsequent learning — and then combining the solutions in an appropriate manner. With respect to multi-class classification problems, the ICL approach presented in this paper can be summarized as follows. Initially the system focuses on one category. After it learns this category, it tries to identify a compact subset of features (nodes) in the hidden layers, that are crucial for the recognition of this category. The system then *freezes* these crucial nodes (features) by fixing their incoming weights. As a result, these features cannot be obliterated in subsequent learning. These frozen features are available during subsequent learning and can serve as parts of weight structures build to recognize other categories. As more categories are learned, the set of features gradually stabilizes and learning a new category requires less effort. Eventually, learning a new category may only involve combining existing features in an appropriate manner. The approach promotes the *sharing* of learned features among a number of categories and also alleviates the well-known *catastrophic interference* problem. We present results of applying the ICL approach to the Handwritten Digit Recognition problem, based on a spatio-temporal representation of patterns.

*Senior Fulbright Scholar visiting ICSI and EECS Dept. University of California at Berkeley. On leave from the Institute of Mathematics, Warsaw University of Technology, Plac Politechniki 1, 00-661 Warsaw, Poland. e-mail: mandziuk@alpha.im.pw.edu.pl

1 Introduction

The *catastrophic interference problem* [6, 5, 7] is one of the greatest impediments in building large, scalable learning systems based on neural networks. In its simplest form, the problem may be stated as follows: when a network trained to solve task A is subsequently trained to solve task B, it "forgets" the solution to task A. In other words, the network is unable to acquire new knowledge without destroying previously built knowledge structures. A seemingly simple solution to this problem is to retrain the network on a cumulative training set containing examples from all previously learned categories. However, for real, large-scale problems this approach is not practical.

The Incremental Class Learning (ICL) approach ([9]) attempts to address the catastrophic interference problem and at the same time offers a learning framework that promotes the sharing of previously learned knowledge structures. With respect to object recognition and classification problems, the approach may be summarized as follows: The system starts off with all the nodes and links it will ever have, but initially, it focuses on only a small number of categories. After it learns to recognize these categories, it tries to identify which features formed in the "hidden layers" play a critical role in the recognition of these categories. The system "freezes" these critical features by fixing their input weights. As a result, they cannot be obliterated by subsequent learning. These frozen features, however, can participate in structures that are learned subsequently to recognize other categories. As the system learns to recognize more and more categories, it is hoped that the set of features will gradually stabilize and eventually, learning a new category will primarily consist of combining existing features in novel ways.

The paper is organized as follows. In Section 2, the proposed ICL approach is described and its relation to some of the existing incremental learning methods is discussed. Section 3 presents computer simulation results of the ICL for the handwritten digit recognition problem. Conclusions and directions for future work appear in Section 4.

2 Incremental Class Learning

The ICL approach is a supervised learning procedure for neural networks that can be described as follows:

- Subproblems are learned incrementally
- Structures playing a critical role in solving a subproblem are frozen
- The above structures are available for subsequent learning
- Solutions to subproblems are combined in an appropriate manner to solve the complete problem.

The success of the approach depends on four key factors. First, it should be possible to decompose the problem into subproblems in an effective manner. Second, the learning algorithm must develop relatively sparse (compact) structures to solve a subproblem. Third, it should be possible to identify features that play a critical role in solving a particular subproblem so that such features may be frozen. Fourth, it should be possible to combine solutions to subproblems whereby frozen features are shared among various solutions.

The proposed method can be viewed as a member of the general class of *constructive algorithms* since the functional structure of the network is constructed incrementally during learning by freezing appropriate nodes and links. There is, however, a significant difference between most constructive approaches and ICL. Typically, constructive methods (e.g. Cascade-Correlation [1] or Upstart Algorithm [2]) start with a minimal network and expand its structure by adding new nodes and links in order to minimize the overall network's error. In our approach, the network starts off with all the nodes and links it will ever have. Consequently, the representational capacity of the whole network is available right from the very start of the learning process (as is the case of backpropagation nets).

The ICL method also shares some common features with *modular approaches* (e.g. [12, 10]) in that the problem to be learned is divided into subproblems. In modular approaches, however, the subproblems are learned independently by separate modules, and then the solutions for the subproblems are combined to yield the solution for the initial problem. In the ICL method, subproblems are *not learned independently* and the structures learned for solving one subproblem are available for solving subsequent subproblems. Thus, unlike the modular approach, ICL allows considerable *sharing of structure* across subnetworks.

The ICL approach resonates with the notion of *competitive learning* (cf. [8, 4]) and also *longlife learning* (cf. [11]) wherein learning new tasks becomes relatively easier when the number of tasks that have already been learned increases.

The current system implementation does not involve any relearning. However, it is possible to include some form of interleaved learning [5] within the ICL paradigm.

3 ICL application to Handwritten Digit Recognition problem

In this section we present the application of the proposed ICL method to the handwritten digit recognition (HDR) problem. The main objective of the work reported here is to show the efficacy of the proposed learning scheme, rather than to develop a state of the art HDR system.

First we briefly describe the spatio-temporal representation of patterns that was used in computer simulations. Then we present the architecture of the neural networks

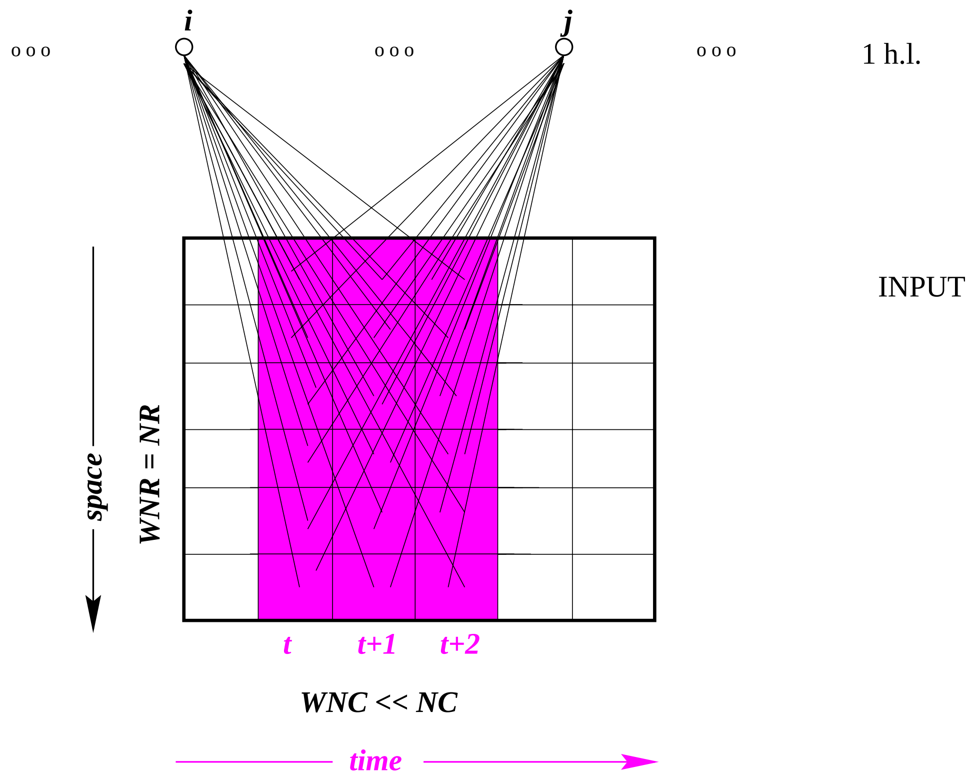


Figure 1: Spatio-temporal pattern representation. A two dimensional input pattern is converted into a sequence of signals generated by a window sliding over the pattern along the *temporal* axis. At each window position (at each time t) all nodes in the first hidden layer (1hl) receive the normalized signal generated by the part of the pattern which is currently covered by the window. NR and NC denote the number of rows and columns, respectively, in the input pattern, WNR and WNC denote the number of rows and columns, respectively, in the window being moved along the pattern.

system used to implement the ICL method, and discuss experimental results that provide evidence for the effectiveness of the ICL approach in solving classification problems such as HDR.

3.1 Spatio-temporal representation of patterns

In the spatio-temporal representation used in this work (cf. [10]), one of the spatial dimensions of the pattern is replaced by the *temporal* dimension, and the two dimensional static pattern is converted into a sequence of signals produced by sliding a window over the pattern along a direction of scan (see *Figure* 1). We will refer to the direction of scan as the *temporal axis*. At time $t = 0$, the window is positioned at the beginning of the pattern, and for each subsequent t it moves by one pixel column along the temporal axis. At each window position all nodes in the first hidden layer receive the input signal generated exactly by the part of the pattern covered by the window. Note that since the width of the window along the temporal axis is greater than 1, the areas covered by neighboring window positions have non-empty intersections.

Advantages of the spatio-temporal representation: The proposed representation has some important advantages over the two dimensional spatial representation. First, it ensures *shift invariance along the temporal axis*. Since the window generates input signals only when the actual content of the pattern is reached, the generated input sequence is independent of the location of the pattern along the temporal axis. Second, the spatio-temporal representation provides a natural framework for dealing with patterns of arbitrary extent along the temporal axis. Finally, the spatio-temporal representation *simplifies a model's architecture*. Usually, the extent of local features along the temporal direction is much smaller than the extent of the pattern itself. Therefore, the sliding window mechanism allows detection of local features with a much smaller number of links between the input layer and the first (hidden) layer of a network. For a detailed discussion of the advantages of the spatio-temporal approach refer to [10].

3.2 System architecture

The system is composed of two modules operating simultaneously and independently on the input data. One module performs scanning of the input pattern along columns and the other one along rows - *Figure* 2. Each module is a feed-forward neural network with two hidden layers and an output layer. In the testing phase, an additional output layer is used to combine evidence from the two modules, and to generate the final output of the system. The only difference between the modules is the direction of scan. Therefore, for the sake of brevity, the detailed description of the system will, henceforth, be based only on the Column Scan Module (CSM).

The CSM is composed of the input layer, two hidden layers and the module-output layer.

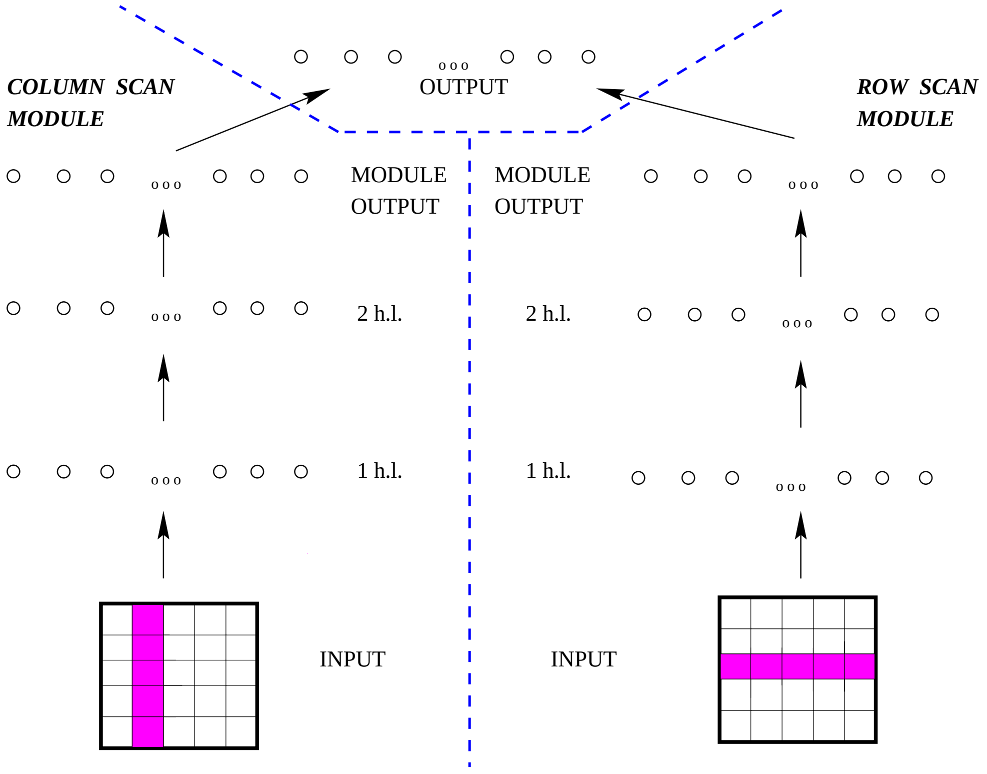


Figure 2: System overview. In the training phase the system is composed of two independently operating modules - Column Scan Module (CSM) and Row Scan Module (RSM). Each module is a feed-forward neural network composed of the input layer, two hidden layers and the module-output layer (denoted by "module-output" in the figure). In the testing phase the additional output layer is placed above the modules to combine evidence from both of them to produce the final output of the system.

Input layer: In the input layer of the CSM, a sliding window of size $WNR \times WNC$, ($WNR = NR, WNC \ll NC$, where NR and NC denote numbers of rows and columns in the input patterns, respectively) moves from left to right along the temporal axis - one pixel-column at a time.
First hidden layer (1hl): Each node in 1hl is fully connected to the sliding window in the input layer with random initial weights. The role of the 1hl nodes is to find the representation of local features pertaining to the training patterns. These nodes are not tied to a particular class.
Second hidden layer (2hl): Higher level features composed of lower level features formed in 1hl are represented in this layer. Each node in 1hl is fully connected to nodes in 2hl with random initial weights. The role of a 2hl node is to find a complex feature pertaining to a *specific class*.
Module-output layer (m-ol): The module-output layer is composed of K nodes - one per class — where K denotes the number of classes in the training set. The final response of the module is expressed in this layer. At the beginning of the training process the connections between 2hl and the m-ol are extremely weak. The appropriate connections are strengthened during the learning process.

3.2.1 Terminology and notation

The training set is composed of K classes C_k, $k = 0, \dots, K-1$, of binary $\{0,1\}$ patterns. Each class C_k is composed of M exemplars each of which is a matrix of NR rows and NC columns. The items in the training set will be referred to as $X_m^k[p][q], m = 1, \dots, M;\ p = 0, \dots, NR-1;\ q = 0, \dots, NC-1$.

Let $w_1[r][c][i]$ denote the weight between the $[r][c]$ element in the input window and the i-th node in the 1hl, let $w_2[i][j]$ denote the weight between the i-th 1hl node and the j-th 2hl node, and let $w_3[i][j]$ denote the weight between the i-th 2hl node and the j-th output node.

Moreover, let $FR_1[k]$ and $FR_2[k]$ denote sets of frozen nodes for class C_k in the 1hl and the 2hl, respectively, and let $I_1^r[n], I_2^r[n]$, and $I_3^r[n]$, for $r = tr$ and $r = ts$ denote input activations to the n-th node in the 1hl, the 2hl and the m-ol, in the $\underline{\text{tr}}$aining and $\underline{\text{t}}$e$\underline{\text{s}}$ting phases, respectively. Similarly, let $O_1^r[n], O_2^r[n]$, and $O_3^r[n]$ stand for the respective output activations of the n-th node in the 1hl, the 2hl and the m-ol, respectively in the training ($r = tr$) and testing ($r = ts$) phases.

Finally, let $H1SIZE$ and $H2SIZE$ denote the numbers of nodes in the 1hl and the 2hl, respectively.

3.2.2 Training phase

Learning overview: First, all weights between input and 1hl nodes and between 1hl and 2hl nodes are initialized with small random values and then normalized so that the sum of squares of incoming weights of all 1hl and 2hl nodes equals 1. Initially the weights on connections between the 2hl and the m-ol nodes are also set close to zero. The following pseudo-code presents the learning overview:

```
initialize_1hl_weights();
initialize_2hl_weights();
for ($k = 0; k < K; k++$)
{
learn_1hl_representation(k);
learn_2hl_representation(k);
connect_module_output(k);
}
```

3.2.3 Learning the 1hl feature representation of the class

The following discussion assumes that the system is currently learning the k-th class, $0 \leq k < K$.

The learning of 1hl features for class k occurs as a result of the repeated presentation of all patterns of class k. The presentation of a pattern X_m^k during a given epoch involves the following steps:

- the input window is positioned over a part of the image
- activation within the window is normalized
- a node in the 1hl with the highest input activation - called the winner - is found
- its incoming weights are updated and normalized
- the window is moved one step in the direction of scan and the above steps repeated until the image is fully scanned.

Window positioning: A window of WNR rows and WNC columns is initially positioned at the extreme left of the pattern (in the experiments reported here, WNR was set equal to NR. Thereafter, the window shifts right, one step at a time, until the image is fully scanned.

Normalizing window activation: Y_m^k, the activity resulting from the image at a given step of the scan is obtained by normalizing the image pixel values under the window in the following way:

$$Y_m^k[r][t+c] := \frac{X_m^k[r][t+c]}{\sqrt{\sum_{p=0}^{p=WNR-1} \sum_{q=0}^{q=WNC-1} X_m^k[p][t+q]}}, \tag{1}$$

$$r = 0, \ldots, WNR-1, \qquad t = \text{scan step}, \qquad c = 0, \ldots, WNC-1$$

Finding the winner: Each node i in the 1hl computes its input activation $I_1^{tr}[i](t)$:

$$I_1^{tr}[i](t) = \sum_{r=0}^{WNR-1} \sum_{c=0}^{WNC-1} Y_m^k[r][t+c] \cdot w_1[r][c][i] \tag{2}$$

and the node whose weight vector is closest to Y_m^k is found, i.e.

$$winner = argmax_{i=0,\ldots,H1SIZE-1} \; I_1^{tr}[i](t) \tag{3}$$

Updating the winner's weights: The change in the weights of a 1hl node depends on the degree of match between the node and window's content. The weight change procedure consists of the following three cases:

α): The winning node has a *high degree of match* with the window's content, but the node has already been frozen during the learning of some prior class j. That is,

$$\sum_{r=0}^{WNR-1} \sum_{c=0}^{WNC-1} Y_m^k[r][t+c] \cdot w_1[r][c][winner] > \theta_{shar} \tag{4}$$

and $\exists \; j < k : \; winner \in FR_1[j]$. In this case, the weights of the winner are *not changed.* Note that, the winner has a high match with the current pattern, and hence, will automatically be shared between classes k and j (and any other classes with which the winner has a high degree of match). θ_{shar} is a predefined (high) threshold value for sharing nodes between classes.

β): The winning node has a *low degree of match* with the window's content, and the node has already been frozen during the learning of some prior class j. That is,

$$\sum_{r=0}^{WNR-1} \sum_{c=0}^{WNC-1} Y_m^k[r][t+c] \cdot w_1[r][c][winner] \leq \theta_{shar} \tag{5}$$

and $\exists \; j < k : \; winner \in FR_1[j]$. In this case, the *unfrozen* node that *best matches* the window's content is found and its weights are changed as in case γ below.

γ): The winning node has not been frozen during the learning of prior classes. That is, $\forall j < k : \; winner \not\in FR_1[j]$.
In this case the weights of the *winner* are updated as follows:

$$\begin{cases} \Delta w_1[r][c][winner] := \frac{\eta(Y_m^k[r][t+c] - w_1[r][c][winner])}{log(mass[winner])} \\ \\ mass[winner] := mass[winner] + 1 \end{cases} \tag{6}$$

where $mass[i], i = 0, \ldots, H1SIZE - 1$ is the "inertia" of node i and η is a predefined learning rate [1]. A node's inertia increases each time it is a winner, and hence, nodes tend to develop into stable feature detectors. A node's inertia is reset (set to 3) at the beginning of each training epoch.

[1] In the experiments presented in the paper η was set to 0.05.

Normalizing winner's incoming weights: The winning node's incoming weights are normalized in the following way:

$$w_1[r][c][winner] := \frac{w_1[r][c][winner]}{\sqrt{\sum_{p=0}^{p=WNR-1}\sum_{q=0}^{q=WNC-1}(w_1[p][q][winner])^2}}, \tag{7}$$
$$r = 0, \ldots, WNR-1, \qquad c = 0, \ldots, WNC-1$$

Freezing relevant nodes: All the 1hl nodes, that became winners in the last epoch form the set of features, $FR_1[k]$, associated with class C_k in the 1hl. All of these nodes are *frozen*.

3.2.4 Learning the class representation in the 2hl

The learning of complex features for class k in the 2hl also occurs as a result of the repeated presentation of patterns of class k over several epochs. In each epoch, for each pattern X_m^k, the following steps are performed:

- 1hl representation of a pattern is found
- some number of winners are found in the 2hl
- winners' weights are updated
- winners' weights are normalized

Finding 1hl representation of a pattern: For each window position t, each 1hl node i calculates its incoming activation $I_1^{tr}[i](t)$. These activations are stored in the auxiliary memories $MEM_i, i = 0, \ldots, H1SIZE-1$ (see *Figure* 3).

After the scanning of the pattern is completed, the 1hl representation of the pattern is defined by the set of nodes with the highest input activations across the layer, for each instance t. In other words, WIN_m^k, the representation of pattern X_m^k in 2hl is defined as:

$$WIN_m^k = \bigcup_t \; argmax_{i=0,\ldots,H1SIZE-1} \;\; MEM_i(t) \tag{8}$$

Output activations of the 1hl nodes are defined as:

$$O_1^{tr}[i] = \begin{cases} \frac{1}{\sqrt{|WIN_m^k|}}, & \text{if} \quad i \in WIN_m^k \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

Finding winners in the 2hl: Having the 1hl representation of the pattern activated, the input activations to the 2hl nodes are calculated:

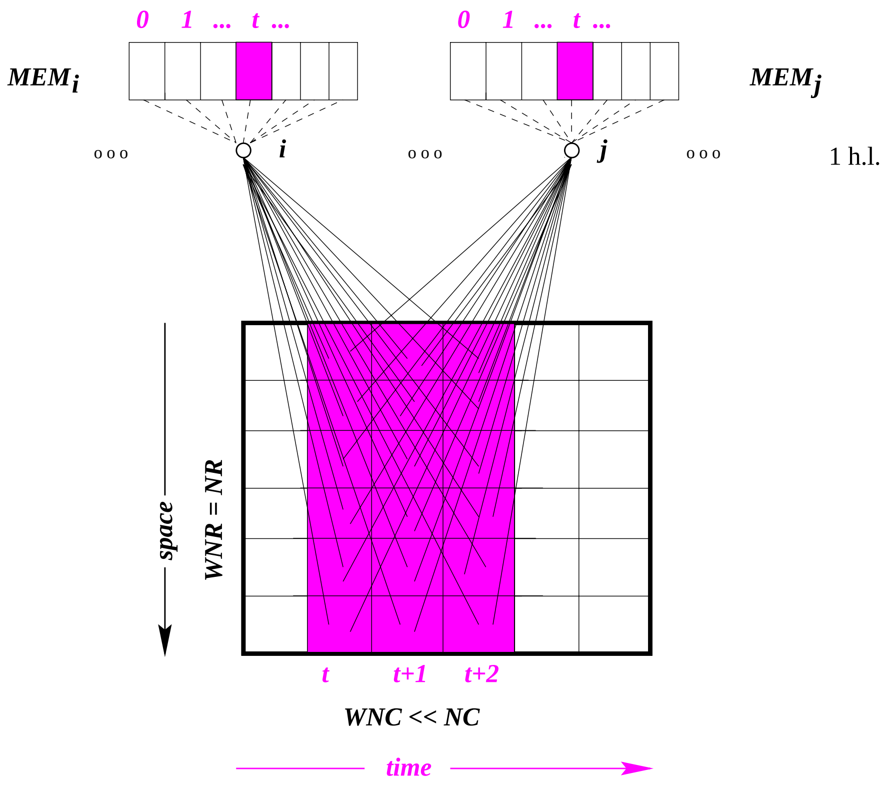


Figure 3: Training of the 2hl. First the 1hl representation of the pattern is found. Each 1hl node i, at each window position, stores its incoming activation in the auxiliary memory MEM_i. The 1hl pattern representation is defined as the set of nodes with the highest activations for all instances t - one "winning" node per one window position.

$$I_2^{tr}[p] = \sum_{i=0}^{H1SIZE-1} O_1^{tr}[i] \cdot w_2[i][p], \qquad p = 0, \ldots, H2SIZE-1 \tag{10}$$

and the first ρ nodes with the highest input activations are chosen - denoted by $win_1, \ldots, win_\rho$ in the eqs. below[2]. Namely,

$$win_s = argmax_{p=0,\ldots,H2SIZE-1, p \neq win_1, \ldots, p \neq win_{s-1}, p \notin FR_2} \; I_2^{tr}[p], \qquad s = 1, \ldots, 5 \tag{11}$$

Note, that the winning nodes are chosen only among not yet frozen nodes. Therefore, there is no sharing of nodes between classes in the 2hl.

Updating winners' weights: Weights between the 1hl nodes and the winning nodes in the 2hl are updated in the following way:

$$\Delta w_2[i][win_s] := \frac{\psi(O_1^{tr}[i] - w_2[i][win_s])}{scale[s]}, \tag{12}$$

$$i = 0, \ldots, H1SIZE-1, \qquad s = 1, \ldots, 5, \qquad scale = [1, 2, 2, 3, 3],$$

where ψ is a predefined learning rate [3]. The degree of weights change is scaled by coefficient $scale$ based on the "winning position" of the winning node. Scaling of learning coefficients adds some flexibility in assigning 2hl nodes to particular 1hl representations in the learning process. Some nodes develop into precise detectors of 1hl representations, while others detect "average" 1hl representations of patterns from class k.

Normalizing winners' weights: The incoming weights of the winning nodes are normalized in a manner similar to that used for normalizing the incoming weights of the 1hl nodes.

$$w_2[i][win_s] := \frac{w_2[i][win_s]}{\sqrt{\sum_{j=0,\ldots,H1SIZE-1}(w_2[j][win_s])^2}}, \tag{13}$$

$$i = 0, \ldots, H1SIZE-1, \qquad s = 1, \ldots, 5$$

Freezing relevant nodes: All 2hl nodes, which became winners in the last epoch form a representation $FR_2[k]$ of class C_k in the 2hl. All of them are *frozen*.

3.2.5 Building connections between the 2hl and the m-ol:

Once the 2hl training phase for class k is completed the connections between all 2hl nodes which were frozen for that class and a node representing class k in the m-ol are strenghten to be equal 1 ($Figure$ 4).

[2]In the current implementation, ρ is set to 5.

[3]In the experiments presented in the paper ψ was set to 0.05.

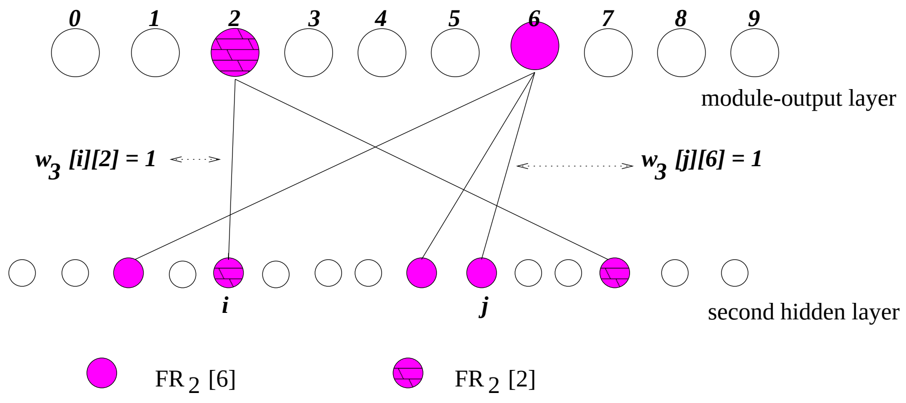


Figure 4: Building connections between the 2hl and the m-ol layer. For the currently learned class k, connection weights between nodes in the 2hl that represent class k (nodes from the set $FR_2[k]$) and the node representing class k in the mo-l are strenghten to be equal 1.

3.2.6 Pruning the network

Once the training process is completed the network structure is minimized by pruning irrelevant nodes and links. All nodes (along with their respective links) in the 1hl and the 2hl, which were not frozen by any class are deleted.

Pruning operation has no impact on system's efficiency. The main reason for pruning is to make the system more compact. However, *prunning should not be done* if the system is going to be exposed to other learning tasks in the future since unfrozen nodes may be required for capturing new features created during subsequent learning.

3.2.7 Testing phase

In the testing phase (*Figure* 5), the unknown pattern is presented to the input layer, and scanned left-to-right by the CSM, and top-to-bottom by the RSM, using a scanning window. While a pattern is being scanned, each node in 1hl maintains a cumulative "winning activation". In other words, at each scan step t, the node i in the 1hl that receives the highest input activation in that step increases its cumulative input activation $I_1^{ts}[i]$ by the current activation value $MEM_i(t)$. More formally,

$$I_1^{ts}[i] = \sum_{t \in T(i)} MEM_i(t), \qquad i = 0, \ldots, H1SIZE - 1 \tag{14}$$

where

$$T(i) = \{t : argmax_{m=0,\ldots,H1SIZE-1} \;\; MEM_m(t) = i\} \tag{15}$$

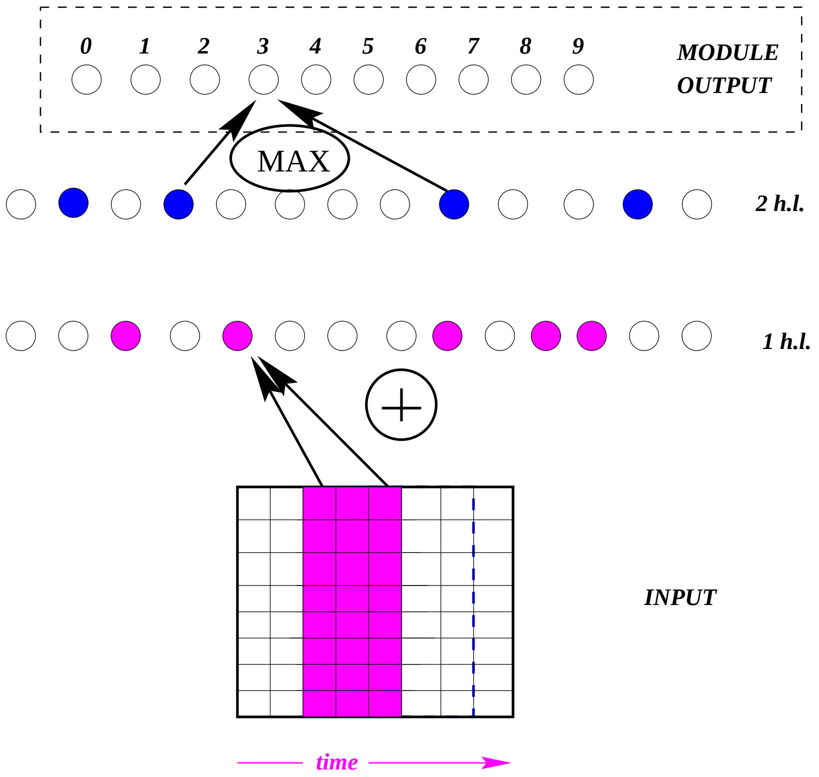


Figure 5: Overview of the testing procedure - Column Scan Module. See text for description.

Output activations from the 1hl nodes are normalized:

$$O_1^{ts}[i] = \frac{I_1^{ts}[i]}{\sqrt{\sum_{j=0}^{H1SIZE-1}(I_1^{ts}[j])^2}}, \qquad i = 0, \ldots, H1SIZE - 1 \tag{16}$$

The input activations to all 2hl nodes are calculated:

$$I_2^{ts}[i] = \sum_{j=0}^{H1SIZE-1} w_2[j][i] \cdot O_1^{ts}[j], \qquad i = 0, \ldots, H2SIZE - 1 \tag{17}$$

and the set WIN_2 composed of p nodes with the p highest input activations is defined (p is a system parameter). Then the output activations from 2hl nodes are calculated in the following way:

$$O_2^{ts}[i] = \begin{cases} I_2^{ts}[i], & \text{if} \quad i \in WIN_2 \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

Input to each of the m-ol nodes is calculated as the *maximum* input among the 2hl nodes contributing to this m-ol node. Namely,

$$I_3^{ts}[k] = \max_{i=0,\ldots,H2SIZE-1}(w_3[i][k] \cdot O_2^{ts}[i]), \qquad k = 0, \ldots, K - 1 \tag{19}$$

Finally,

$$O_3^{ts}[k] = I_3^{ts}[k], \qquad k = 0, \ldots, K - 1 \tag{20}$$

The evidence from the CSM and the RSM in the testing phase is combined in the final output layer ($Figure$ 6) in the following way:

$$OUT[k] = \alpha \cdot OUT^c[k] + \beta \cdot OUT^r[k] \tag{21}$$

where $OUT^c[k], OUT^r[k]$ denote outputs from the k-th nodes in the module-output layers in the CSM and RSM, respectively, and $OUT[k]$ is the input (and output) activation of the k-th node in the final output layer.

3.3 Experimental results

The efficacy of proposed ICL approach was tested on the HDR problem by computer simulations of the neural network architecture and learning scheme described above. The set of binary $\{0, 1\}$ patterns representing handwritten digits was extracted from the USPS CEDAR database. The data was composed of ten classes $C_k, k = 0, \ldots, 9$ (one class per digit) each of which contained 600 training and 600 test patterns $X_m^k[\,][\,], m = 0, \ldots, 599, 600, \ldots, 1199$, and additional 100 patterns were used for cross-validation.

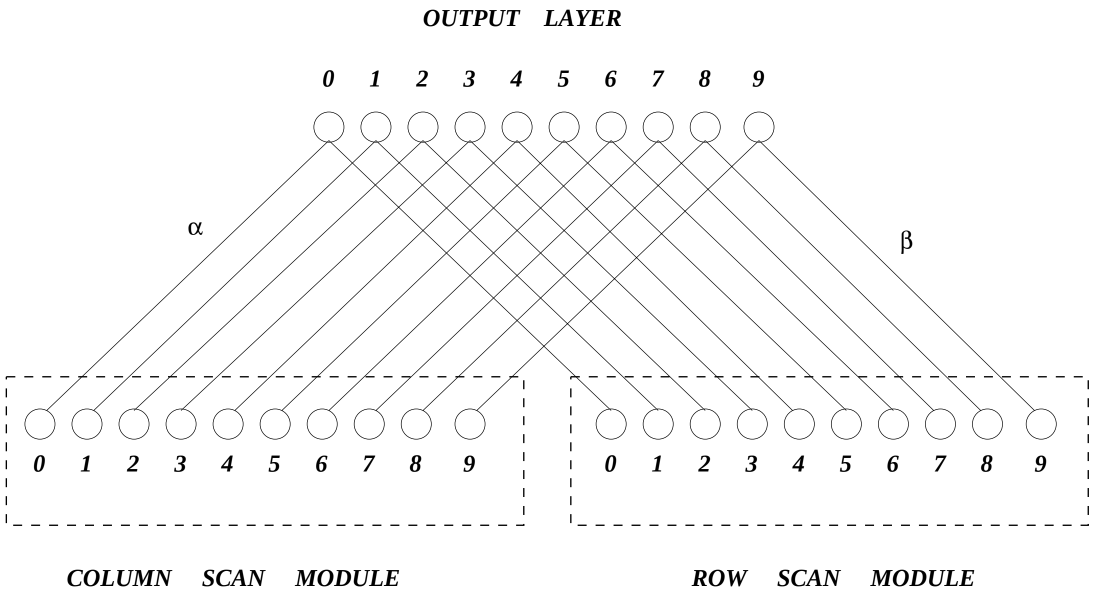


Figure 6: Testing phase. Each output layer node's value is a linear combination of respective evidence from CSM and RSM. All links between the m-ol nodes in the CSM and in the RSM and the output layer nodes are equal to α and β, respectively.

3.3.1 Preprocessing

Size normalization: Patterns, originally of various sizes, were normalized [3] by resizing from rectangle to square, sampling with a regular interval, comparing the normalized sum of activations of pixels in the surrounding square with the threshold value ($= 0.5$) and, if greater or equal than the threshold, setting the corresponding pixel in the resulting image to 1, or setting it to 0, otherwise. The size of patterns after normalization was equal to 17×17. Some exemplar digits before and after size normalization are presented in $Figure$ 7.

Other preprocessing techniques: No other preprocessing techniques (low pass filtering, skeletonization, skew normalization, etc.) were applied to the data. Consequently, preprocessed digits preserved high variations in thickness and skewness. Additional examples of size-normalized input patterns are presented in $Figure$ 8.

3.3.2 Simulation results

One of the observations from preliminary simulations was that the order of classes in the learning phase has some influence on the quality of results. Therefore, in the final system setup four modules - two CSMs and two RSMs were used. Combining evidence from four (instead of two) independent modules resulted in greater flexibility and reliability in the system's behavior. One CSM and one RSM used the ascending

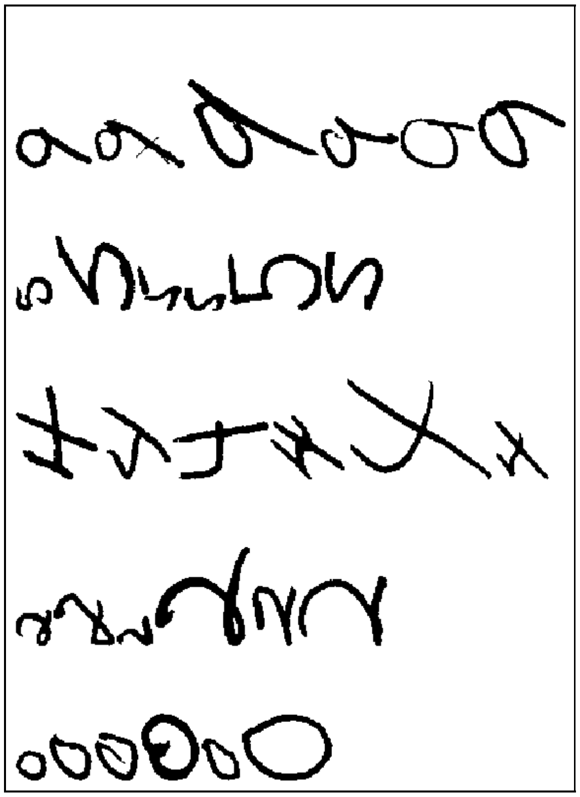
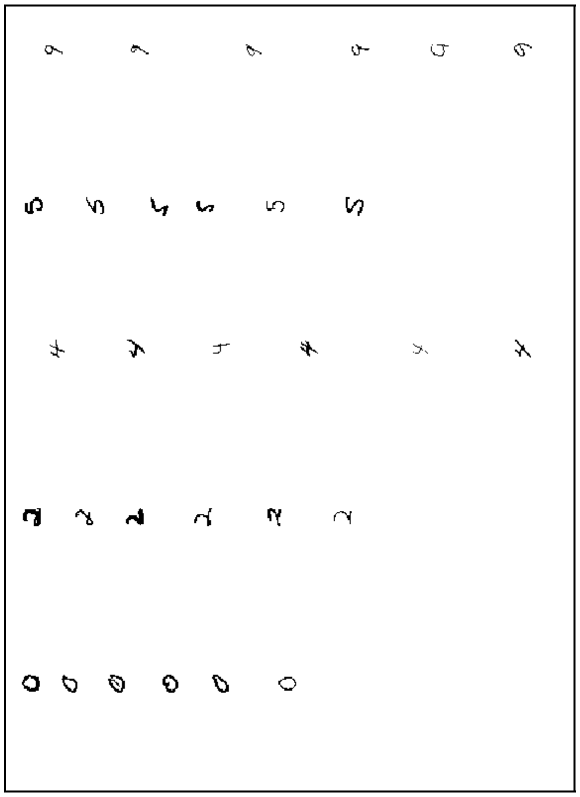
(a)   (b)

Figure 7: Examples of training patterns before (*a*) and after (*b*) size normalization. Figures have the same scale.

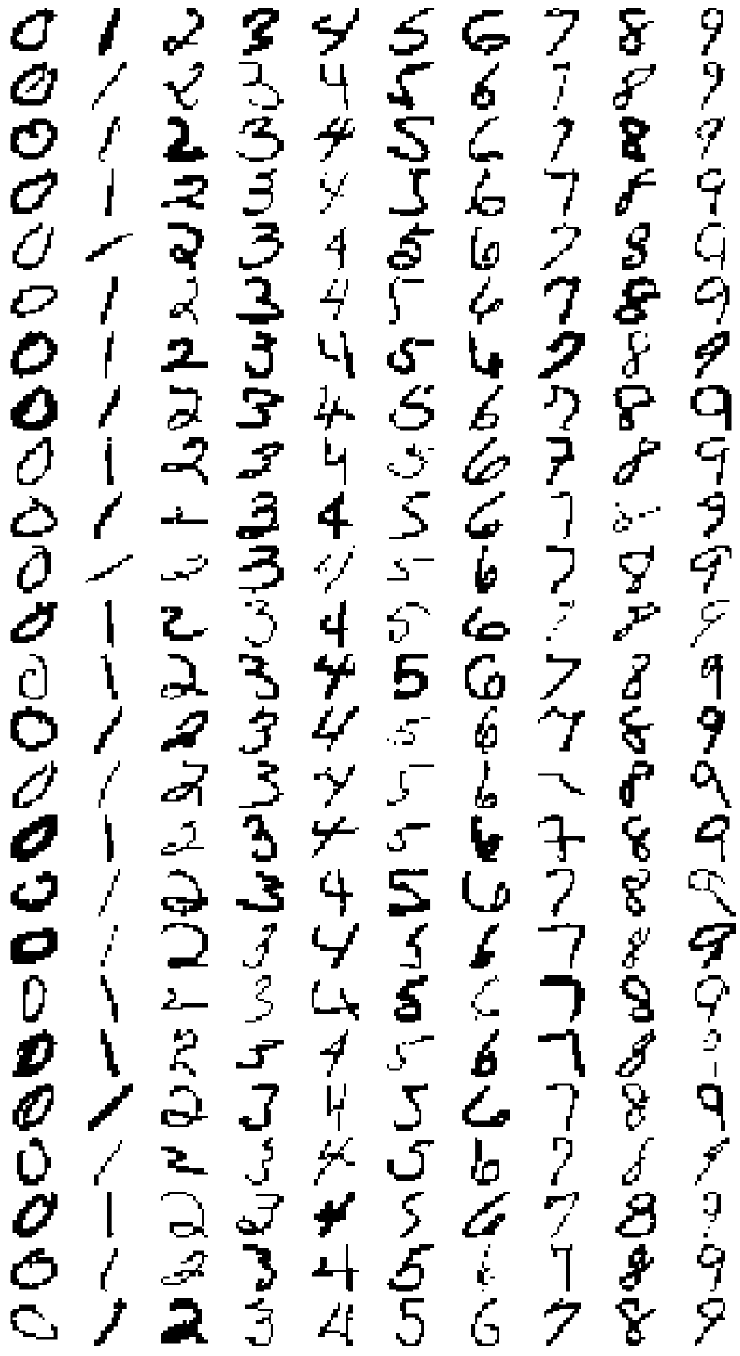


Figure 8: Examples of training patterns from the CEDAR database, after size normalization, used in the simulations.

order of classes, that is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The order of learning was chosen independently for the remaining two modules ($\{9, 4, 1, 6, 5, 7, 3, 8, 2, 0\}$ for the other CSM, and $\{9, 7, 6, 5, 0, 4, 3, 8, 2, 1\}$ for the other RSM). In the final output layer both CSMs were treated equally (with the same weight). Similarly, the respective links between each of the RSMs and the final output layer were pairwise equal. In general, links between CSMs and the final output layer were different from those between RSMs and the final output layer. Hence, the input to node k in the final output layer, denoted by $OUT[k]$ in (22) was equal to

$$OUT[k] = \alpha \cdot OUT^{c1}[k] + \beta \cdot OUT^{r1}[k] + \alpha \cdot OUT^{c2}[k] + \beta \cdot OUT^{r2}[k] \qquad (22)$$

where $OUT^{c1}[k], OUT^{c2}[k], OUT^{r1}[k]$ and $OUT^{r2}[k]$ denote outputs from the respective module-output layers of the four modules. The final output of the system was the class number corresponding to the maximum value $OUT[k], k = 0, \ldots, 9$.

The number of nodes in each layer was established based on some preliminary simulations, and for each of the four modules was equal to 5000, 1200 and 10 for the 1hl, the 2hl and the m-ol, respectively.

Among the 20,000 and 4,800 available nodes in the 1hl and the 2hl of the four modules, respectively, the system froze 10,442 and 1,840 nodes, respectively, during the training phase. The system effectively used about $2.1 * 10^6$ out of about $2.5 * 10^7$ available links.

As described above, learning in the 1hl and in the 2hl was based on multiple presentation of all exemplars from the currently learned class. The numbers of training epochs per class were equal to 50 and 10 for the 1hl and the 2hl, respectively. Based on preliminary simulations we observed that increasing 1hl training epochs to 75 or 100 had no particular impact on the system efficiency. On the other hand, reducing the number of epochs below 30 resulted in degraded system performance.

Several alternative schemes for computing the network's output were considered. Two of these are described below.

Raw performance: The output was computed under the following condition:

- $\alpha = \beta = 1$,
- $p = 1$ (i.e. *one* winner was selected in the 2hl in the testing phase in each of the four modules).

The system's performance was 97.87% on the training set and 92.55% on the test set.

Enhanced performance: We experimented with a small set of alternate values for α, β and p and observed that the system's performance could be improved by differentially weighting the responses of the row and column networks. For example, the choice of

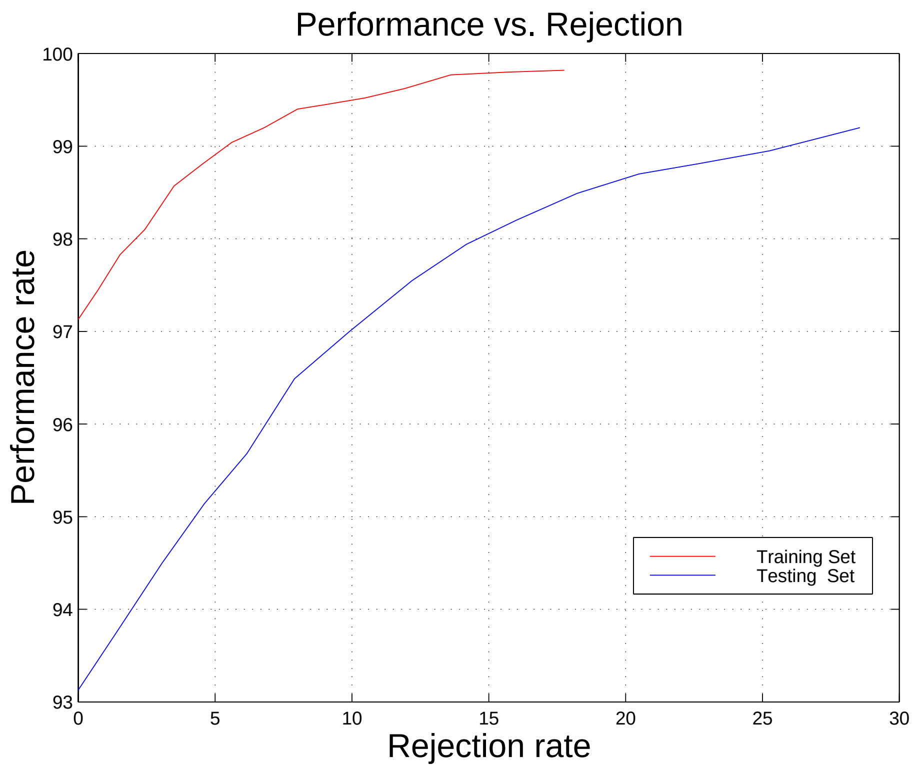


Figure 9: A plot of system's performance on the training and test sets with rejection.

- $\alpha = 0.9, \beta = 1.3$,
- $p = 8$ for CSMs and $p = 3$ for RSMs

improved the system's performance on the test set to 93.13%.

Results with rejection: In the experiments reported thus far, a pattern was rejected if the ratio between the second and the first best choices in the final output layer was relatively high, that is if

$$k_1 := \max_k \{OUT[k]\} \qquad \text{and} \qquad k_2 := \max_{k \neq k_1} \{OUT[k]\}, \tag{23}$$

then

$$\frac{OUT[k_2]}{OUT[k_1]} > \text{threshold} \qquad \Longrightarrow \qquad \text{reject pattern} \tag{24}$$

The plot of system's performance with various levels of rejection is presented in *Figure* 9.

The performance on the test set exceeded 94%, 95%, 96% and 97% with 1.9%, 4.4%, 6.9% and 9.9% rejection, respectively. The performance on the training set exceeded 98% and 99% with 2.1% and 5.4% rejection, respectively.

Misclassified patterns: In the experiment without rejection, the performance on the test set was equal to 93.13, that is 412 out of 6000 patterns were incorrectly classified. Some of these misclassified patterns were highly ambiguous, even for humans. Examples of misclassified patterns are presented in *Figure* 10. The confusion matrix for the test set is presented in *Table* 1.

—	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	$\sum$
0	—	2	1	1	0	1	6	0	15	0	26
1	0	—	0	0	10	0	0	1	3	2	16
2	7	4	—	11	3	4	1	14	9	4	**57**
3	0	0	6	—	0	13	0	10	15	4	48
4	0	22	5	0	—	1	0	1	3	13	45
5	0	2	4	27	0	—	0	1	7	3	44
6	4	9	9	1	5	8	—	1	7	0	44
7	1	10	2	0	11	8	0	—	2	12	46
8	7	9	1	16	3	9	0	0	—	3	48
9	0	7	1	1	17	0	0	10	2	—	38
$\sum$	19	**65**	29	57	49	44	7	38	**63**	41	*412*

Table 1. Misclassification table for the test set in the experiment without rejection. In each row of the table, the number of misclassifications of the respective digit (the first value in a row) with all other classes is presented. The rightmost column presents a sum of all misclassifications for this digit. For example, digit 3 was misclassified with digit 5 in 13 cases, and digit 5 with digit 3 in 27 cases.

Significance of experimental results:

The simulation results obtained for the CEDAR USPS database suggest that the ICL approach outlined in this paper is an effective and plausible method of solving classification problems involving a large number of classes.

In evaluating the results it may be appropriate to note that our goal was simply to evaluate the efficacy of the ICL learning scheme rather than to develop a state of the art HDR system. It seems reasonable to assume that the performance of the system could be improved further by using larger training sets and performing more sophisticated preprocessing (e.g. skew normalization and skeletonization).

The performance of the ICL based HDR system was compared with that of Nearest Neighbor Classifier (NNC) using the training patterns used in the ICL experiment. The recognition rate of NNC on the test set was 86.36%.

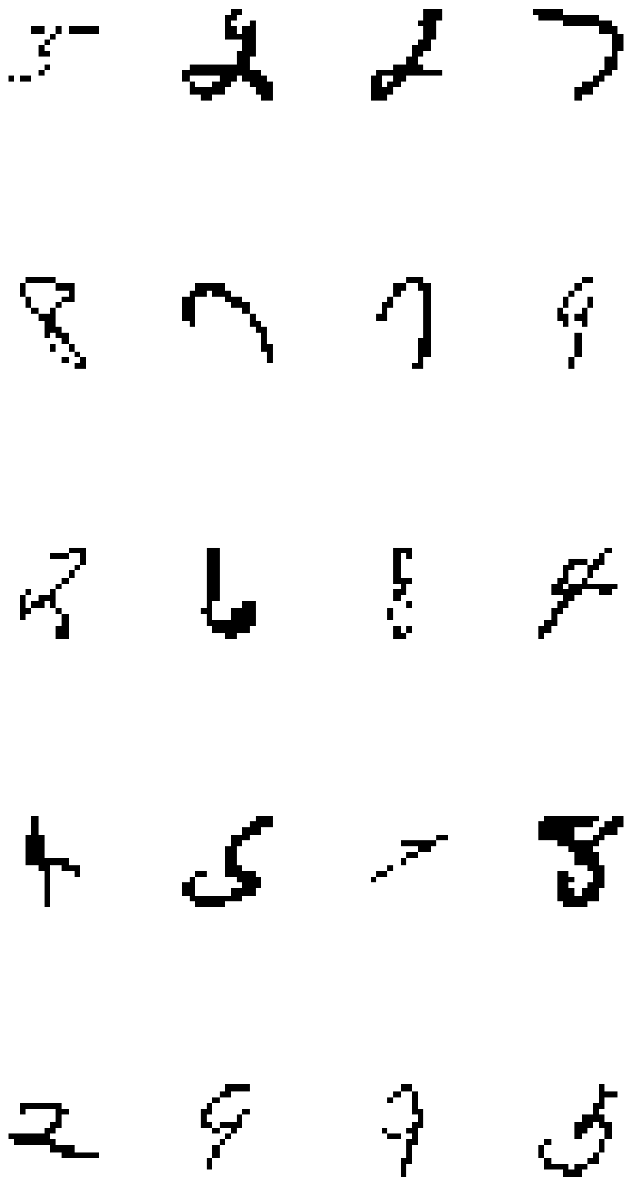


Figure 10: Some examples of misclassified patterns from the test set.

Possible directions for improvement: The system's performance may be improved by refining the ICL approach and we are pursuing several possibilities. These include:

- selection of multiple winning nodes in 2hl,
- propagation of negative evidence from the 1hl nodes to the 2hl nodes, and
- the fine tuning of shared (frozen) nodes by allowing such nodes to modify their input weights — albeit with very high inertia.

Based on pilot simulations we believe that the fine tuning of shared nodes can improve the system's performance.

3.3.3 Stabilization of feature set in the 1hl

One of the expected properties of the system is the asymptotic stabilization of the feature set in the 1hl. One would expect that after the network has learned a *sufficient number* of classes, learning would become relatively easy and primarily involve the use of features from previously learned tasks.

CLASS	NEW NODES	ALL NODES		CLASS	NEW NODES	ALL NODES
0	176	176		**9**	352	352
1	60	111		**4**	292	497
2	416	564		**1**	72	238
3	390	648		**6**	243	464
4	312	612		**5**	348	704
5	346	856		**7**	125	511
6	255	778		**3**	419	919
7	107	527		**8**	563	1254
8	546	1265		**2**	332	1077
9	291	951		**0**	174	790
		2899				2920

Table 2. Numbers of nodes recruited by subsequent classes in the CSM based training in two different orders of classes. For each CSM learning order, columns from left to right denote: the class number, number of new nodes for this class, number of all (new and shared with previous classes) nodes for the class.

The numbers of recruited nodes per class for the two orders used in the training of CSM are presented in *Table* 2. These results suggest that such asymptotic behavior was not observed in the current set of experiments and subsequent classes still recruit many new nodes (e.g. digit 8). It appears that the number of classes will have to be increased beyond ten in order to observe the expected convergence of the feature set. In order to verify the hypothesis that there will be a gradual decrease in the number of nodes frozen for a new class we plan to extend the experiment to the set of all alphanumeric characters. The other possibility is to continue training the existing "trained system" on *another* handwritten digits database and see if there is significant overlap between the nodes frozen in response to the new set of exemplars of a given class and those frozen in response to the old set of exemplars of that class.

An interesting phenomenon observed in *Table* 2 is that the number of nodes recruited by a class is higher when the class is learned later in the training sequence compared to the number of nodes recruited by the same class when it is learned earlier in the training sequence. This can be observed, for example, in the case of digit 9. This digit recruited 951 nodes when it was the last one in the learning order, but only 352 nodes when it was the first class in the learning order. This observation is true for all classes, that is, for each class *the further the class in the learning order, the relatively bigger its representation.*

The other observation from *Table* 2 is that the overall number of nodes frozen by all classes is independent of the class order (the same observation holds for RSMs).

4 Conclusions

In this work we have proposed the Incremental Class Learning approach based on freezing relevant features and sharing common (similar) features among multiple classes. The sharing of relevant features among classes makes subsequent learning more effective (easier and faster), especially when new tasks exhibit a degree of similarity to any of the previously learned tasks. The ICL approach not only takes advantage of existing knowledge when learning a new problem, it also offers immunity from the catastrophic interference problem. Promising results obtained for the unconstrained Handwritten Digit Recognition problem suggest that the approach may be a suitable framework for building large, scalable learning systems.

References

[1] S. E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture", *Technical Report: CMU-CS-90-100*, 1990

[2] M. Frean, "The upstart algorithm: a method for constructing and training feed-forward neural networks", *Neural Computation*, 2, pp. 198-209, 1990

[3] Hou, *Digital Document Processing*, J. Willey & Sons, 1983

[4] T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps", *Biological Cybernetics*, 43, pp. 59-69, 1982

[5] J. L. McClelland, B. L. McNaughton and R. C. O'Reilly, "Why there are Complementary Learning Systems in the Hippocampus and Neocortex: Insights from the Successes and Failures of Connectionist Models of Learning and Memory", *Technical Report: PDP.CNS.94.1*, 1994

[6] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem", In G. Bower (Ed.) *The psychology of learning and motivation*, 24, pp. 109-165, 1989, Academic Press

[7] E. Pessa and M. P. Pennaand, "Catastrophic Interference in Learning Process by Neural Networks", *Proc. of the ICANN'94, Sorrento, Italy*, pp. 589-592, May 1994

[8] D. Rumelhart and D. Zipser, "Feature Discovery by Competitive Learning", *Cognitive Science*, 9, pp. 75-112, 1985

[9] L. Shastri, "Attribution Learning as a solution to the Catastrophic Interference Problem in Learning with Neural Nets". *Working Paper, International Computer Science Institute.* December 1994

[10] L. Shastri and T. Fontaine, "Recognizing Handwritten Digit Strings Using Modular Spatio-temporal Connectionist Networks", *Connection Science*, 7(3), 1995, pp. 211-235.

[11] S. Thrun and T.M. Mitchell, "Learning One More Thing", *Technical Report: CMU-CS-94-184*, 1994

[12] A. Waibel, "Consonant Recognition by Modular Construction of Large Phonemic Time-Delay Neural Networks", in D.S. Touretzky (ed.), *Advances in Neural Information Processing Systems (NIPS) 1*, Morgan Kaufmann, pp. 215-223, 1989