

BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-93-M2

**“ Net-time and Conflation:
Improving Classification Models with Ablated Input”**

by
Ron Papka

Net-time and Conflation:
Improving Classification Models with Ablated Input

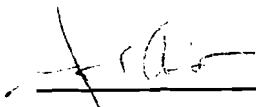
by
Ron Papka
B.S. Columbia University, 1987

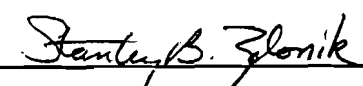
Thesis

Submitted in partial fulfillment of the requirements for the
Degree of Master of Science in the Department of
Computer Science at Brown University

May 1993

This Thesis by Ron Papka
is accepted in its present form by the
Department of Computer Science
as satisfying the thesis requirement for the degree of
Master of Science

Date Jan 8, 1993 
James A. Anderson


Date JAN 8, 1993 
Stanley B. Zdonik

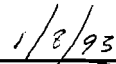
Approved by the Graduate Council

Date _____

Authorization to lend and reproduce Thesis


As the sole author of this thesis, I authorize Brown University to lend it to other institutions or individuals for the purpose of scholarly research.

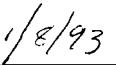


Ron Papka


Date

I further authorize Brown University to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.



Ron Papka


Date

1.0 INTRODUCTION

Clearly, there are methods superior to neural networks for enabling an artificial system to tell time from the image of a wall clock. However, the purpose of this project was to set up the problem for a neural network as a springboard for gaining better insight into connectionist learning methods. It was initially believed that the problem would be trivial to learn, but thousands of processing hours later it seems that this is not the case.

The software program that attempts to tell time from the image of a wall clock has been named **Net-time**. In an effort to improve Net-time's performance, several experiments utilizing ablation patterns on vectorized input instances were conducted, and it was determined that an ablated input instance could give rise to a correct output response where the unablated input instance did not. Net-time was ported to a parallel machine so that several ablation patterns on an input instance could be tested simultaneously. This software port led to the development of **Conflation**, a model that produces output by processing in parallel several vectors of an input instance, each having a different ablation pattern imposed on it.

Conflation was developed as an enhancement to the Brain State in a Box (BSB) [2] neural network model, and proved successful on Net-time data. In an effort to determine if this enhancement was significant, it was tested on other domains in addition to Net-time, with similar success.

The remainder of this paper is organized into two main sections. The first explains the Net-time program, the dataset and information representations it utilizes to tell time, and experimental results. The second section explains a controlled benchmark study which explores the significant effects of ablation patterns on several connectionist learning models.

1.1 LEARNING TO TELL TIME

Machine learning involves a training phase, where a subset of a domain is used to *train* the network, and a testing phase where a portion of the remainder of the domain is used to *test* a learning model's ability to produce correct output responses from input instances that were not used during the training phase. For many domains, input instances can be mapped to classes, and the learning model is used as a classification process to realize this mapping. In *real world* systems, when it is possible to enumerate and store the entire domain, the classification process is most accurately realized by lookup table, and no "*learning*" need take place.

In order to perform non-trivial classification on domains whose instances can not be stored because they are infinite or exponential in the number of attributes used to represent instances, the domain must be learned. The task of telling time involves responding with the time-class that the configuration of the clock's arms represents. Although the number of classes for this task is limited to the different times of day, for some prespecified level

of granularity of time, the number of wall-clock instances in the universe representing a particular time of day is essentially infinite, so a method for telling time must involve an ability to generalize. A human becomes proficient at telling time by learning the task on a limited number of clock faces, but s/he is still capable of telling time on a clock face whose image was never previously encountered. The attributes of a clock's image vary between times of day and between different clocks. Different clocks have different tick markings, numeric symbol representations, arm types, arm lengths, etc., and we expect these attributes will continue to vary as clock makers develop new aesthetics, and introduce new clock instances into the universe. Furthermore, we expect that these new clock instances will not present a problem to those who know how to tell time.

Unfortunately, there is not a great deal of research concerned with the way humans tell time, so at this point we can not expect to create a system that mimics this ability. But in order to gain better insight into this process in humans, I conducted some informal experiments. One experiment involved drawing the configuration of the arms of a clock (represented by two pencil lines) on a sheet of paper. Most people had no problem responding with an approximation of the time that was intended. Since these drawings were void of any clock feature other than arms, it was apparent that the angle between the large hand and the small hand was sufficient for telling approximate time. However, when the same people were asked questions of the type, "What time is it when the little hand is 6 degrees south of 7 and the large hand is 66 degrees north of 7," responses were relatively slow, and mostly "I give up." I concluded from these informal experiments that people do not utilize the absolute angles found in mathematics, but develop an intuition for fuzzy angles between arms in a configuration with no numerical significance other than larger or smaller. With this in mind, I proceeded to train a neural network on a vectorized representation of the image of the configurations of the arms of a clock.

1.2 NET-TIME

The training task of Net-time involves teaching a neural network vectors representing every 5 minutes of wall-clock time from the image of a 12-hour analog clock. Testing is conducted by presenting a BSB neural network with a vector representing a non-5-minute time. A successful test is one where a non-5-minute time input gives rise to an output representing the closest 5-minute time. A complete testing run processes vectors representing every minute around the clock (including training instances), and the percentage of correct responses represents the network's ability to tell approximate time. Usually, when we speak of *the performance of a learning system*, two measurements are implied: the percentage of correctly learned training instances, and the percentage of correctly classified testing instances. However, in the context of Net-time, the overall ability to tell time is most important, and represents the combination of testing and training performance.

The actual output of Net-time is a human (pre-recorded) voice telling the approximate time. 144 voice files, one for every 5-minute time around the clock, were recorded by Claudia Papka, Jennifer and John Reubens, Paul Reilly, Andrew MacKeith, and Noela Cabrera. In User-Mode, a person can enter a time at the command line, and a few seconds

later, the program will reply “out loud” with what it believes to be the approximate time. If the time is “unknown,” a reply to that effect is heard.

The program was developed on a Sun Sparc 2, and later ported to a parallel implementation on Thinking Machine’s Connection Machine (CM-2 w/4k processors). The code was written in C and TM’s C/Paris using GNU’s C compiler. The clock images were generated using the Brown Workstation Environment BWE/ASH, an X11 graphics toolkit written by Steve Reiss et al.

1.3 A VECTOR REPRESENTATION FOR TIME

One of the most important factors contributing to successful learning using neural networks is an appropriate vector representation for input and output information. Problems will exist for networks containing one layer if the resulting vector representation imposed on a domain is not linearly separable, or if instances are in general position [9]. In Net-time, an ideal representation will not only be learnable, but will also be flexible enough to allow generalization. A $-1/1$ binary representation was used for the experiments that follow, and the input representation (image of the clock) remained constant, while the output representation (time) was varied. The training phase of the BSB model utilizes Widrow-Hoff [13] (delta rule) to auto-associate vectors. Widrow-Hoff is a supervised learning algorithm, where a weights matrix is adjusted during an iteration to compensate for the error each input instance’s output vector contains. This error is the element-wise difference between input vector and output vector. A pseudo-code program for the Widrow-Hoff algorithm is listed in the Appendix.

During training, input and output are contained in the same vector. During testing, input is present, but the output bits are ablated (zeroed). The testing process of the BSB model is responsible for reconstructing the output bits. It does this by means of a recurrent process. The output vector is fed back through the weights matrix for several matrix-vector inner product iterations. After each iteration, the ablated output elements begin to take on small real values in the interval $[-1, 1]$. If a vector element grows beyond the $-1/1$ binary limit, the element is clipped to the limit. There are 3 control constants used during this process, and they are listed in the Appendix along with pseudo-code for the BSB algorithm.

The training phase of BSB attempts to define points of attraction on the hypercube that represent training instances. If this process is successful, an input vector will *move* to the intended attractor during the testing process. This paradigm lends itself to retrieving approximate time. The arm configuration for every 5-minute time, and its associated symbol time (e.g. 12:55) are represented in a binary vector that is learned by a BSB network. Since these are binary vectors, they map to points on a hypercube. A good representation would allow an input instance of a non-5-minute time to move to the attractor representing the closest 5-minute time.

The image for a simulated wall-clock was created using the line and circle drawing functionality of BWE. A sample is contained in figure 1. The experiments were concerned only with learning the configurations of the arms of the clock, so the clock face (i.e. outer circle and tick marks) are drawn only for aesthetic purposes, and do not become part of vector information.

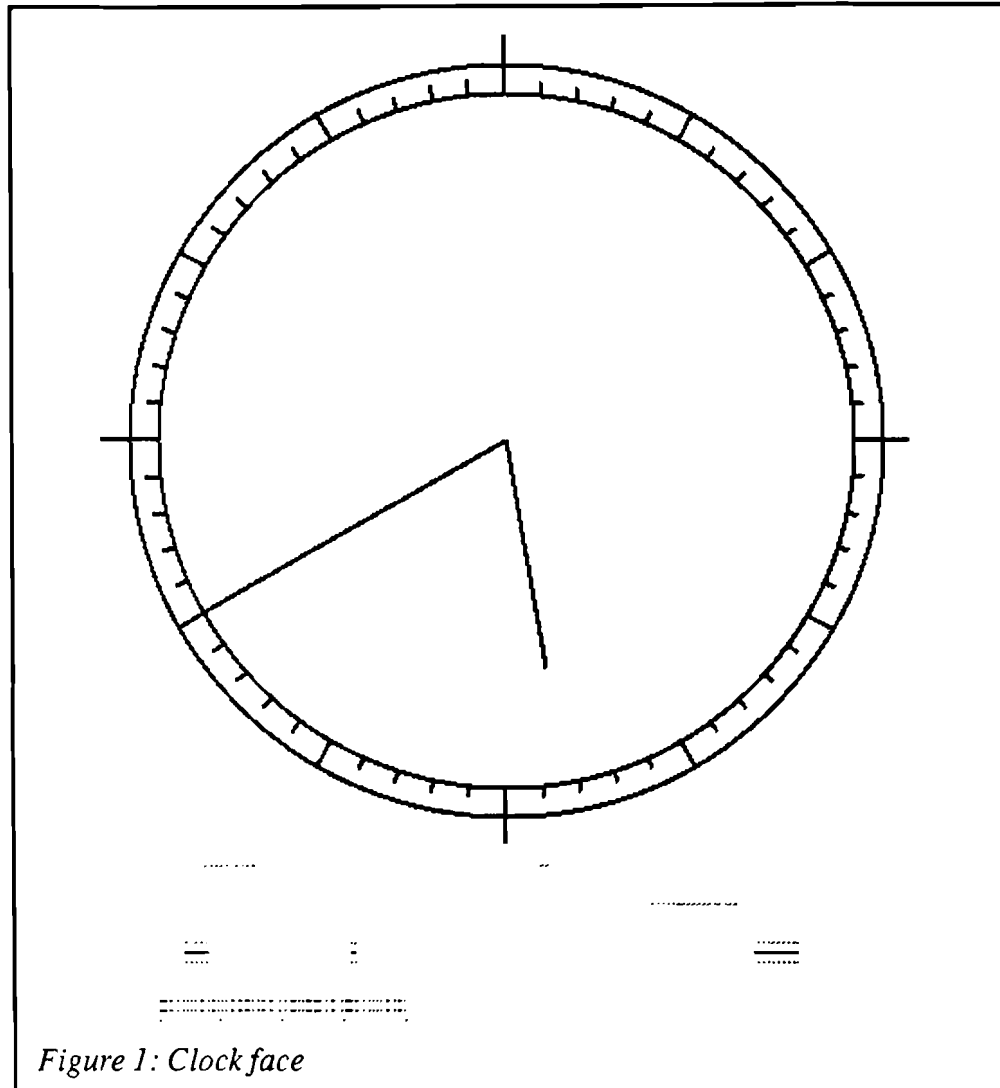


Figure 1: Clock face

The procedure for vectorizing the image for a particular time begins by drawing the arms within a circle having a 360 pixel diameter, and then taking a 360 x 360 bit-map snapshot of the arms, using the xwd windows function. Flattening the matrix to form a vector yields vectors with 129,600 elements, and since vector elements must be floating point numbers, each vector requires over .5 Megabytes of storage. To avoid this, the solution was first to create two 360-element vectors, one containing the column-wise reduction of pixel values, and the other the row-wise reduction of pixel values. Next, the two vectors were concatenated and shrunk into one vector containing 360 elements. The shrinking was accomplished by first taking every other element from the column-wise reduction, and then every other element from the row-wise reduction. Finally, a -1/1 binary representa-

tion was achieved by replacing all elements containing values greater than 1 with 1, and replacing elements containing a value of 0 with -1. The image vectors containing integer values are unique for each time around the clock; however, when binarized, adjacent times have vectors that are too similar. To circumvent this problem, only the tips of the clock's arms (30 pixels worth) were used to construct the image vector and thus produce unique vectors with less overlap for each 5-minute time.

The 360 elements of the image vector serve as the input portion of the vectors used by the BSB model. The output portion of this vector contains the encoding for the digits representing the time. For example, the time 12:05 can be represented with the digits 1,2,0, and 5, so the time encoding problem was reduced to finding a representation for the digits 0 through 9. Several representation, requiring different numbers of bits, were tested:

| # of Bits | Representation |
|-----------|---|
| 4 | 8421-BCD and Excess-3 BCD |
| 5 | 2-of-5 BCD |
| 10 | Grandmother Cell (one bit for each digit) |
| 32 | ASCII word (ONE, TWO, THREE, etc.) |
| 32 | Barcode (Thermometer: 1= * 2= ** 3=***) |
| 32 | Orthogonal Vectors (From Walsh functions) |
| 32 | Orthogonal Vectors (From Random Search) |
| 32 | Maximized Hamming Distance |

The Maximized Hamming Distance encoding was created by generating a set of 10 random binary vectors, and determining the Hamming distance a vector had with respect to every other one in the set. It was later found that an excellent process for generating approximately orthogonal or orthogonal vectors utilizes a Genetic Algorithm [4a, 6] instead of random search or Walsh functions. Walsh functions generate a set of orthogonal vectors containing fairly *regular* bit patterns. By utilizing the Genetic Algorithm, vector sets with irregular patterns can be created.

Figure 2 depicts the set of 144 training vectors. The 488 pixels on one line represent one vector. The dots on the extreme left of the figure delineate the vectors for each hour starting at 12:00. The two sinusoidal curves running down the vectors represent the signal that the tip of the hour hand emits. Figure 3 depicts all 720 vectors, and the sinusoidals emitted by the minute hand can also be seen. The "barcode" pattern on the right third of the image is the string of bits for the output representation. This pattern is 128 bits long, where four 32 bit strings decode to the digits of the corresponding time.

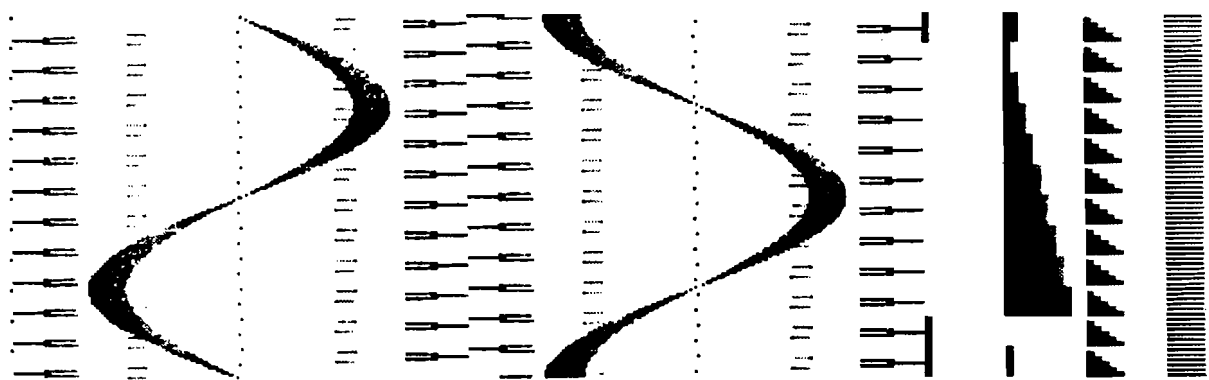


Figure 2: 5-minute vectors (Barcode output representation)

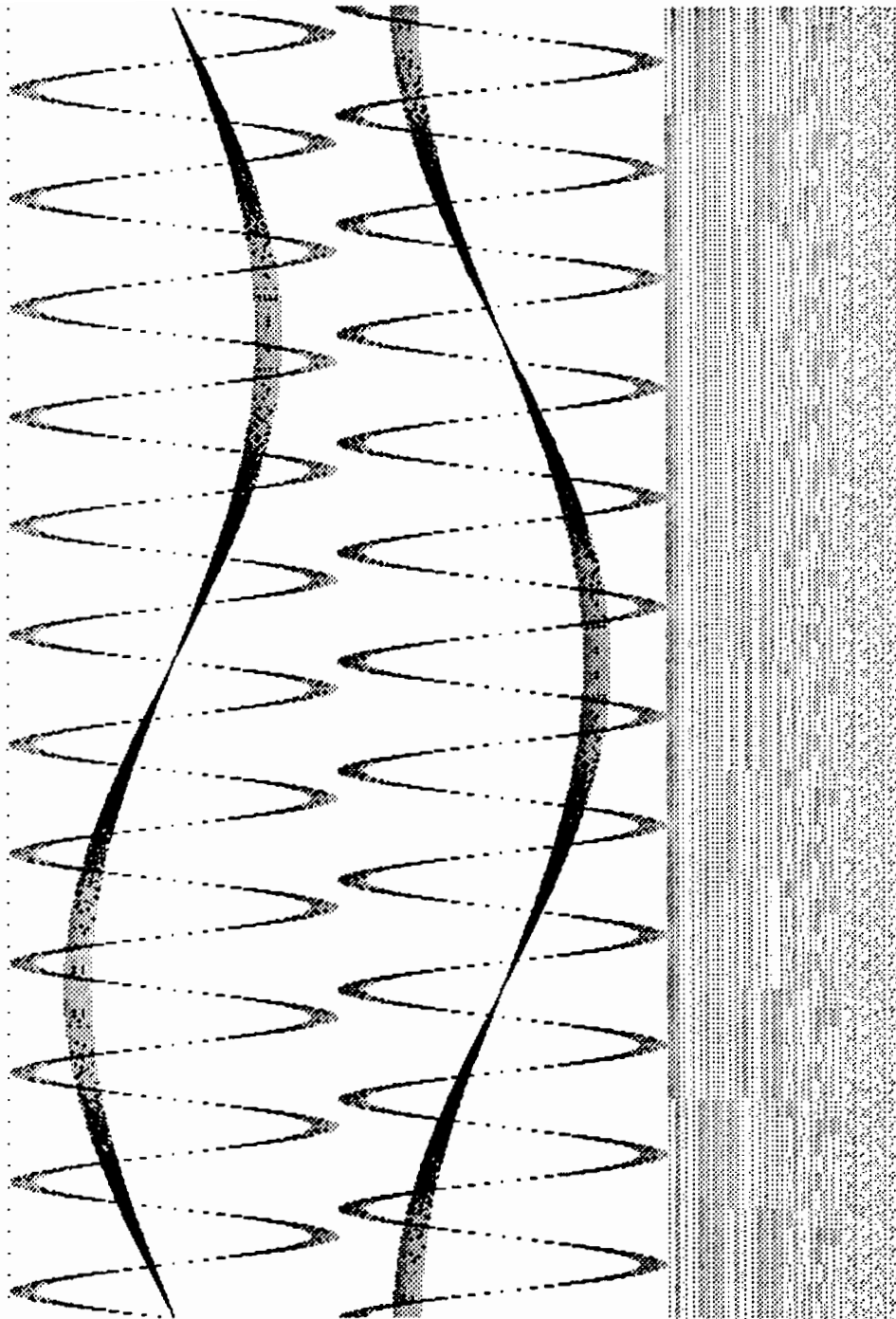


Figure 3: All 720 vectors (*Maximized Hamming Distance representation*)

In order to test a particular time, the image of the time is created and then vectorized in the manner described above. The output bits are ablated, and the vector is run through the testing phase of the BSB model. After the vector limits, or the maximum number of iterations is executed, the resulting output bits of the vector are decoded. In several cases, the decoding of the bits does not yield an exact match to a digit, hence the encoding nearest in Hamming distance (its nearest neighbor) is used. In User-Mode, Net-time will open an audio file whose filename is the decoded time, and write the file to the speaker device. This provides the system with its speech capabilities.

Early experiments were conducted on a Sun Sparc 2. The 144 5-minute vectors were trained using 6,000-10,000 random presentations, and the Widrow-Hoff learning rate was held to the inverse of the vector length. Training required approximately 12 hours for 10,000 presentations. The BSB testing phase was conducted on all 720 vectors utilizing 180 feedback iterations. The feedback constants alpha, delta, and lambda were held at 0.2, 1.0, and 0.9 respectively. The following table lists the results of the two best-performing representations (the Grandmother cell representation was not tested at this time.) Each test took 3 days on the Sparc 2. The 4 and 5 bit representations allowed approximately 65% of the training vectors to be learned, and the orthogonal vectors could facilitate the learning of only 39% of the training set. In the following table, results are presented with two values, the first being the number of correctly classified times, and the second being the percentage of correctly classified times.

| Representation | Performance | Training |
|----------------|-------------|-------------|
| ASCII word | 390 - .5416 | 134 - .9305 |
| Max. Hamming | 397 - .5513 | 141 - .9791 |

1.4 MORE ABOUT NET-TIME

In addition to generating time images and telling the times they represent, Net-time has a command line driven User Interface with the following commands:

'a' = set ablation pattern.
'b' = save window as bit map.
'c' = set color constant.
'd' = display matrix.
'h' = help.
'i' = set maximum iterations.
'l' = load matrix.
'p' = set feedback parameters (Alpha, Delta, Lambda).
'q' = quit.
's' = save matrix.
't' = toggle CM-2.
'v' = display vector map.
'w' = create vector map.
'XX:XXL9' = Relearn time XX:XX with 9 learning iterations.
'XX:XXU9' = Unlearn time XX:XX with 9 unlearning iterations.
'XX:XX' = Tell time XX:XX.

1.4.1

'a' = set ablation pattern.

A -1/1 binary vector is ablated by zeroing some of its elements. In addition to ablating the output portion of a vector, it is also possible to ablate some of the input elements. The motivation for ablating the input elements is that a vector representing full information may not reach its intended attractor, while some ablated instance of the vector does. There were several instances of this phenomenon found in some of the early experiments. The use of ablation patterns¹ on input instances was extensively studied, and the results are discussed in the second section of the paper.

If the user selects 'a' at the command line, s/he has the option of selecting one of eight ablation patterns.

1. An ablation pattern is a mask over an input instance that indicates the elements to be ablated.

1.4.2

'b' = save window as bit map.

'c' = set color constant.

'd' = display matrix.

'v' = display vector set.

'w' = create vector set.

When working with neural networks and information embedded in high dimensional space, visualization techniques are almost essential for ensuring that a program is manipulating data as intended. The images from figures 2 and 3 were created by first creating a vector set, then displaying the set, and finally saving the window it appeared in to a bit-map file in xwd format. Displaying a vector set or a matrix can provide a visualized insurance that a persistent structure is read correctly from disk. In addition, it provides a better intuition of the data one is working with. Figure 4 is the image of a neural network or weights matrix. Each pixel of the image is the corresponding value in a matrix multiplied by a constant (in order to produce a pixel value within the bounds of the display's color map.) By changing the color constant, the magnitude of the values of different regions in weight space is illuminated. The matrix in the figure indicates that auto-associating vectors produces a relatively symmetric matrix with regions of weights containing the same value. A closer examination reveals that the subregions of the matrix are symmetric about the diagonal, but that the values in these symmetric regions are not necessarily the same.

The speed improvements afforded by the Connection Machine and its Display Library allowed the visualization of the formation of the network in real-time. As training proceeds, regions within the matrix become well defined and converge to an image similar to that of figure 4.

Another technique is to display the current state of a vector being processed. In figure 1, there are three groups of vectors displayed just below the clock face. The first group contains the column-wise and row-wise reduction vectors of the arm configuration image. The top vector in the second group contains the input portion of the vector being tested, and the bottom one is the input portion of the vector representing the expected approximate time. The middle vector is a display of the current state of the feedback vector. Its image is updated every 10 or 20 iterations. The top line of the third group is the expected output portion of the feedback vector, and just below it is the output portion's current state. The five dots on the bottom line delineate the groups of 32 bits representing the four digits of a time. By displaying the feedback vector, one can see how quickly a vector moves to an attractor, and also intuit the problems caused by attractors created from poorly performing information representations.

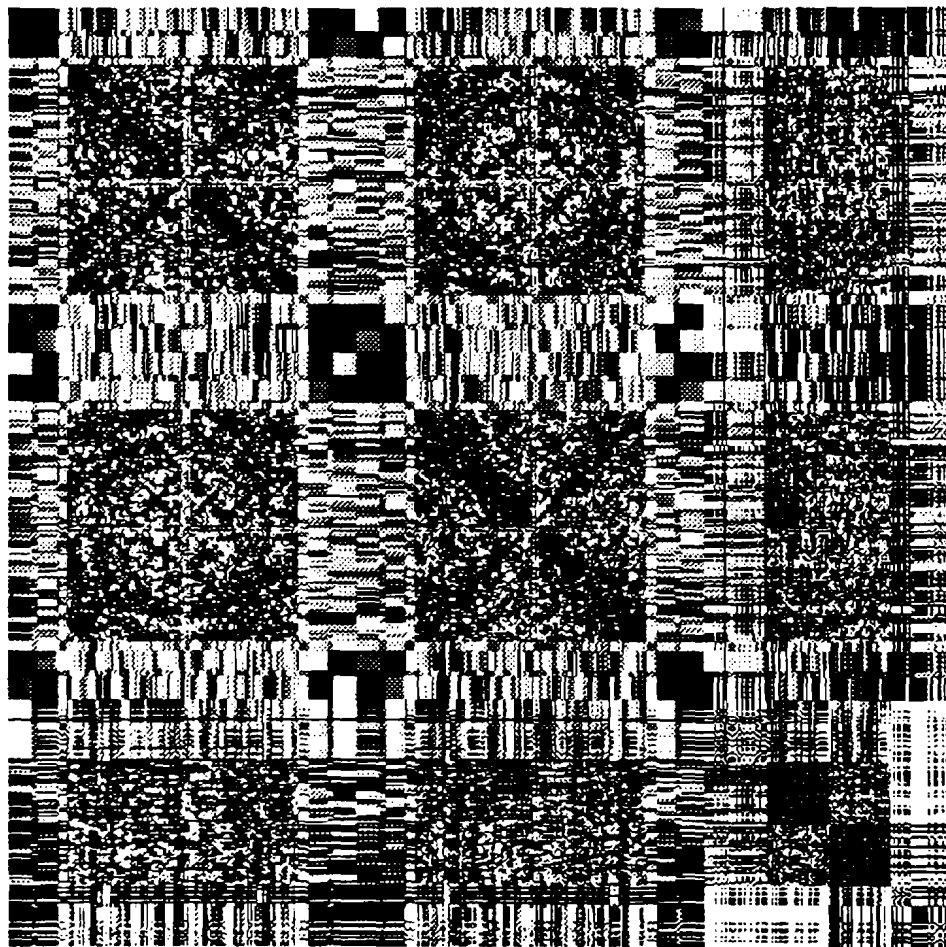


Figure 4: Weights Matrix / Neural Network (488 x 488)

1.4.3

'i' = set maximum iterations.

'p' = set feedback parameters (Alpha, Delta, Lambda).

The number of iterations and the feedback constants are the four main learning “knobs” with which the user can experiment. It was determined that 600 iterations were necessary to allow an input vector to reach a stable state. Setting alpha to 0.2, delta to 1.0, and lambda to 0.9 provided the best results.

1.4.4

'l' = load matrix.

's' = save matrix.

'XX:XXL9' = Relearn time XX:XX with 9 learning iterations.

'XX:XXU9' = Unlearn time XX:XX with 9 unlearning iterations.

Since networks were trained by randomly presenting instances, certain orderings provided for better performing networks than others. Furthermore, it was found that the performance of a network could be improved by “reteaching” 5-minute times that were not originally learned. Several experiments involving relearning were successful, and one session allowed a network utilizing the Maximum Hamming Distance representation to learn all 144 training vectors. In some cases a particular time was “overlearned” and caused many vectors to move to its attractor. “Unlearning” experiments were conducted with marginal success. It was found that unlearning would alleviate the problem caused by a particular time, but would create problems for other times.

1.4.5

't' = toggle CM-2.

After conducting initial experiments, it was determined that it would be necessary to port the application to a parallel machine to gain running-time improvements. Thinking Machine's Connection Machine (CM-2 w/4k processors) proved very suitable for implementing the BSB model. In addition, CM's CMSSL library provided a Matrix-Vector multiplication routine that would allow several vectors to be processed through a network in parallel. This capability was used to test the effects of ablation patterns imposed on vectors. Brown's CM-2 utilizes a Sparc 1 as the front-end. Running times comparing the Sparc 1 to the CM-2 are listed below. The Sparc 1 tests involved running one time vector through the BSB model for the specified number of iterations. The CM-2 tests were simi-

lar, but 8 vectors (7 ablated) were processed in parallel. Times are in minute-second (mm:ss) format.

| Iterations | CM-2 | Sparc 1 |
|------------|-------|---------|
| 60 | 00:05 | 01:06 |
| 180 | 00:08 | 03:07 |
| 360 | 00:12 | 06:09 |
| 600 | 00:18 | 10:14 |

1.4.6

'h' = help.

'q' = quit.

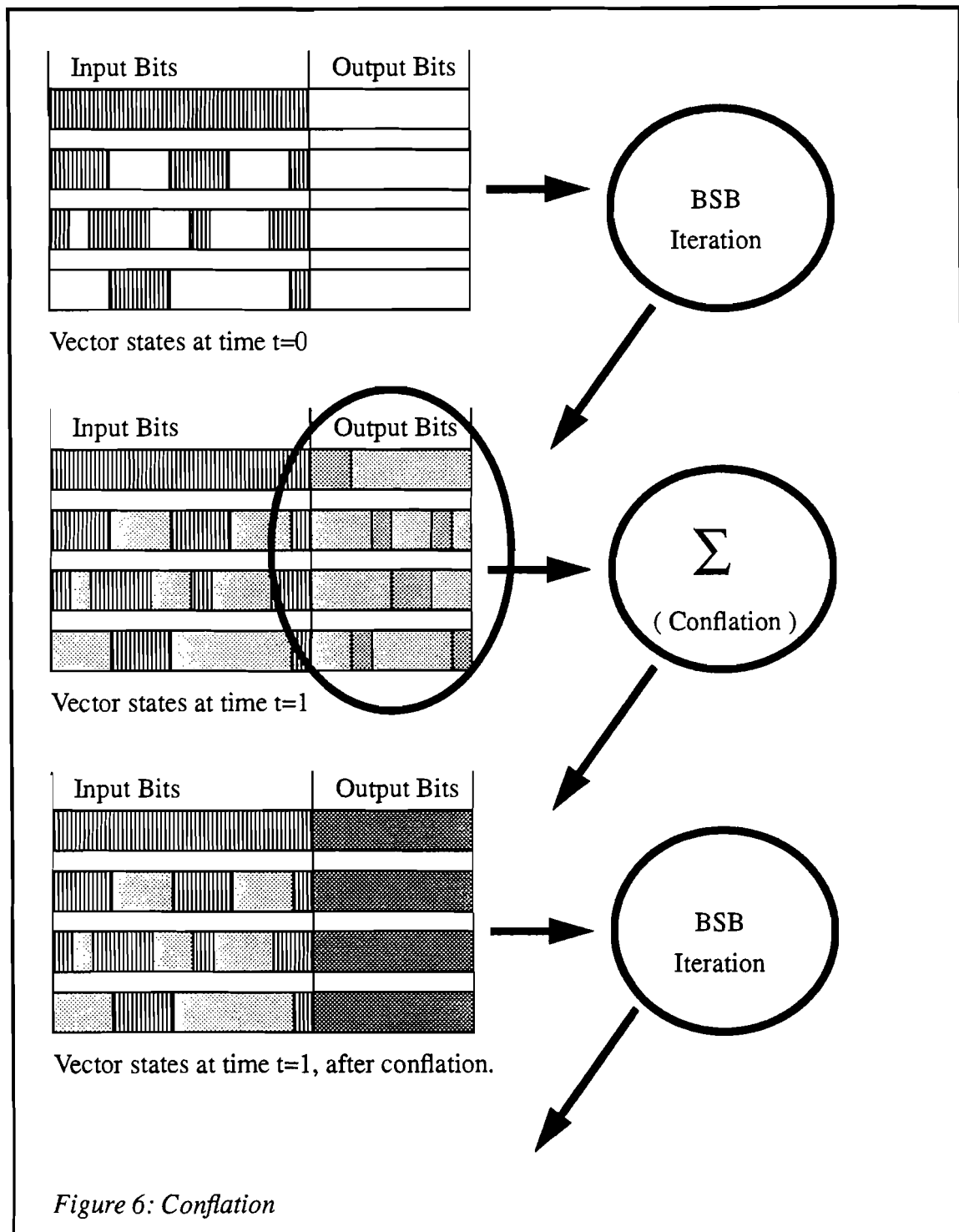
'XX:XX' = Tell time XX:XX.

If a time is typed at the command line, Net-time will process the vector representing its image, and attempt to tell approximate time. For example, if 12:43 is typed at the command line, the clock face representing this time is drawn, and the processing of the vector commences. A successful response for this example would be 12:45. If the user forgets a command, s/he can type h for help, or, if s/he wishes to terminate the session, q for quit.

1.5 CONFLATION AND THE USE OF ABLATION PATTERNS

A variation of BSB, named Conflation, was developed to improve the performance of Net-time. Conflation attempts to utilize the output portion of parallel-processed ablated vectors of an input instance to increase the possibility for a correct classification. Ablated vectors are created by duplicating an input instance and zeroing some of its attributes; they are used in Conflation by combining the output portions of the various ablated vectors and the unablated vector at time t in processing, and distributing this output to all vectors for processing at time $t+1$. It was found that Conflation works best if the output portions are combined after the first iteration ($t=1$).

Figure 6 illustrates Conflation using four vectors. At time $t = 0$, ablation patterns are applied to the vectors. The first vector in a group represents full information of an input instance, and the whited out sections of the following three vectors represent ablated elements on the same instance. These vectors are simultaneously processed using an iteration of the BSB testing algorithm. At time $t=1$, Conflation is applied before further BSB processing. Conflation essentially takes output bits of a group of vectors and sums them element-wise to create new output bits which are subsequently distributed to all vectors in the group. Hence, after Conflation, and before the next iteration of BSB processing, the output portion of all vectors will be identical.



It may seem that Conflation is simply perturbing the output bits, and that the perturbation is the reason for performance improvements. However, experiments were conducted to test the effects of random perturbation of output bits. The results indicate that random perturbation using values similar in magnitude to the changes caused by Conflation did not improve performance, and on average caused a performance degradation.

Ablated vectors often give rise to correct output responses in cases where the unablated vector does not. In general, simultaneously processing x vectors with various ablation patterns produces at most x different outputs. Ideally, one would like to have an oracle choose the correct output as the classification for an input instance. In the following experiments, an oracle that knew the correct time was created to determine the best possible performance a network could attain using Net-time data.

Table 1 lists the results from 3 experiments which utilized the Maximum Hamming Distance output representation. The results indicate that Conflation improved Net-time performance. The final output for Conflation was taken to be that belonging to ablation pattern 2, which consistently provided the best performance. The ablation patterns used for these experiments are explained in the Appendix.

| RUN | TRAINING | NO ABLATION | CONFLATION | PERTURBATION | ORACLE |
|-----|-------------|-------------|-------------|--------------|-------------|
| 1 | 140 - .9722 | 474 - .6583 | 500 - .6944 | 456 - .6333 | 616 - .8556 |
| 2 | 138 - .9583 | 473 - .6569 | 485 - .6736 | 474 - .6583 | 613 - .8553 |
| 3 | 144 - 1.000 | 457 - .6347 | 474 - .6583 | 456 - .6333 | 627 - .8708 |
| AVG | 140 - .9768 | 468 - .6500 | 486 - .6754 | 462 - .6416 | 619 - .8597 |

Table 1: Performances of 3 runs using Maximum Hamming Distance

It was later found that the Grandmother Cell output representation provided improved performance over Maximum Hamming Distance. The line labelled G in Table 2 lists a sample run using this representation. The line labelled P lists the results using a simple Perceptron model, and the line labelled N is the results using Nearest Neighbor. The Near-

| RUN | TRAINING | NO ABLATION | CONFLATION | PERTURBATION | ORACLE |
|-----|-------------|-------------|-------------|--------------|-------------|
| G | 144 - 1.000 | 492 - .6833 | 521 - .7236 | 491 - .6819 | 640 - .8889 |
| P | 142 - .9861 | 470 - .6528 | * | * | 544 - .7456 |
| N | 144 - 1.000 | 690 - .9630 | * | * | * |

Table 2: Grandmother Cell, Perceptron, and Nearest Neighbor

est Neighbor run provided the best overall performance, and confirmed that vectors representing adjacent times were closest in Hamming Distances for all but a few times.

Though Conflation improved the performance of Net-time, it remained to be seen if it was applicable to other domains. The following section describes controlled experiments on other domains. The experiments also explore the reasons ablation patterns help to improve the performance of connectionist classifiers in general.

2.0 INTRODUCTION

In the previous section, it was shown that Conflation could be used to improve the number of correct classifications for untrained input instances using the Brain State in a Box (BSB)[2] neural network model on Net-time data. This improvement is caused by a simple linear mechanism applied to several ablated vectors representing the same input instance. To understand the effects of ablation patterns on data vectors, a benchmark study across basic connectionist learning models was conducted. The study utilizes domains that have been previously examined by researchers to test the capabilities of statistical, probabilistic, and connectionist learning models, and examines them once again to test the capabilities of ablation patterns.

This benchmark study compares the classification performance of three machine learning models --BSB, Perceptron [11], and Nearest Neighbor. The models were tested using a (-1 / 1) binary vector representation on records from 2 databases: Voting-84 and LED-7, which were obtained from the Repository of Machine Learning Databases at U.C.I. The comparison of BSB to Perceptron is interesting because both utilize Widrow-Hoff [13] (delta rule) for supervised learning; however, BSB requires the auto-association of vectors, and Perceptron requires the hetero-association of the input elements to the output elements of a vector. BSB is a recurrent model, so during testing the output is fed back through the network for several matrix-vector inner product iterations. Perceptron requires just one inner product. Nearest Neighbor was used to confirm the published performance measures on the Voting-84 and LED-7 databases, and to provide an intuition for the locations of vectors in attribute-space. Furthermore, its contribution to the experiments engendered a variation of the model which utilizes a Genetic Algorithm and input ablation patterns to create a classifier superior for certain domains.

The utility of ablated vectors is not surprising. Booker, Goldberg, and Holland note:

The interaction among computing elements in a connectionist system make 'best-fit' searches a primitive operation. Activity in a partial pattern of elements is tantamount to an incomplete specification of a concept. Such patterns are automatically extended into a complete pattern of activity representing the concept most consistent with the given specification. [4b]

During BSB processing, then, it may be the case that the ablated instance ignores some attributes that would prevent correct classification from the unablated vector, and that instead, by means of recurrence, the ablated attributes are filled with values that allow correct classification. In nearly every experiment it was found that some ablated vectors give rise to correct classification where the unablated vector did not.

On certain domains, it will be shown that Conflation improves the classification ability of *some* vector with *some* ablation pattern, but the algorithm does not have a method for determining which one, so an *a priori* decision must be made as to which ablation pat-

tern's output will serve as the final classification. A general method for this decision is discussed in a following subsection.

Conflation is an intra-processing mechanism for improving classification. A post-processing method for choosing the correct classification is **Consensus**. It works as follows: Various ablated input instances and the unablated instance are run through a model. The classification that is represented most as output is the final classification. In the event of a tie, the classification belonging to the vector containing the least amount of ablation is the final classification.

With Nearest Neighbor, selecting a final classification in the event of a "tie" can make the difference between this model being the best candidate for a classification task, and the worst. In what follows, the Nearest Neighbor algorithm uses the points that training instances represent as neighborhoods, and classifies a testing instance as the classification of the training point whose neighborhood it falls into. In other words, the testing point is considered to be in the same class as the training point to which it is closest in Euclidean distance. For domains with few attributes, a testing instance could very well be equally distant to several training points. If these points all are in the same class, the classification of the testing instance is trivial; however, if they are in different classes, a decision must be made.

Kohonen et al. have developed LVQ algorithms [8], which are modifications to the Nearest Neighbor model used in these experiments, and which obviate the need for an additional decision step. Implementing LVQ (Linear Vector Quantizer) in this context would utilize the class probability distributions of a domain to move a training point in a direction to increase or decrease the likelihood of a testing point falling into its neighborhood and acquiring its classification. LVQ has been shown to provide significant performance improvements for Nearest Neighbor.

What is presented here is a different modification to Nearest Neighbor which utilizes an ablation pattern on domain instances. The ablation pattern represents a template or mask that defines which attributes will be used to represent instance points. A "good" ablation pattern ignores attributes which detract from, or do not contribute to, correct classification. We can view such attributes as being ambivalent, or as contributing insignificant information; when removed, their undesirable contribution to Euclidean distance is also removed. A method for finding good ablation patterns is discussed below.

The following subsection describes the results of the experiments using the Voting-84 domain. Section 3 discusses the LED-7 experiments. The Appendix contains the domain specifications and the ablation patterns used for the various experiments.

2.1 VOTING-84 DATABASE RESULTS

The Voting-84 database consists of the 1984 voting patterns of 435 members of the House of Representatives voting on 16 issues. The classification task is to conduct training

on a subset of database records, and attempt to determine the correct Party affiliation for the untrained subset. The Voting-84 database has been extensively tested using other learning models. Previous results on this domain state that: IB? has a ~91.5% success rate, C4 ~95.5%, STAGGER 90%-95% [1,10,12], and Backprop ~93.5%.²

In what follows, the records of the database were partitioned into 3 sets, and cross-validation experiments were conducted on both 145- and 290-record training sets, and their respective 290 and 145 testing sets. Hence, a total of 6 testing runs was conducted for each model. Final performance figures are those representing the average performance of 3 homogeneous runs (i.e. cross-validation runs using the same number of training set instances and testing set instances).

In order to limit the effects of independent testing variables, the following were held constant: All learning was done using Widrow-Hoff (delta rule) training, where the learn constant remained at .01111. To minimize the effects of training order, records were presented sequentially. A log containing the square-summed error of the error-vector, and its constituent elements, was maintained to determine if a perceptron reached a stable state during the training phase. In general, it was found that perceptrons required far less than 30,000 sequential presentations for training sets of 145 records, and 40,000 presentations for training sets of 290 records.

The error-vector log described above was also used to determine if a BSB overtrained. Apparently, when a BSB is overtrained the resulting network is very similar to the Identity Matrix, which is one obvious weights matrix solution when auto-associating vectors. While using this domain, if a BSB is trained until its error vector reached a stable state, the result is a matrix where diagonal elements approached 1 and non-diagonal elements approached 0. This type of matrix is undesirable, and it is therefore necessary to "under-train" a BSB. Empirically, it was found that 8,000 sequential presentations for training sets of 145 records, and 16,000 presentations for training sets of 290 records, were sufficient for a well performing BSB. This undertraining produced matrices whose diagonal elements still approached 1; however, the absolute values of the non-diagonal elements were large enough to counter the effects of the diagonal.

It was found that the use of a Bias or Clamp bit set to -1.0 for the training vector sets created improved performance for the BSB and Perceptron models, and Biases were therefore used for all tests. Two different bit-representations of Party membership were tested: The 3-bit representation encodes Republican as (-1, 1, -1) and Democrat as (1, -1, 1); and the 1-Bit representation encodes Republican as (-1) and Democrat as (1). In general, the 1-bit representation provided a better performing system. This result is consistent with Occam's Razor, which suggests: "given two explanations of the data, all other things being equal, the simpler explanation is preferable." [3]

In fact, Occam's Razor prevails for models used on the Voting database, because the Simple Perceptron consisting of one unit classifies untrained instances with a 95.86% success rate. Hence, for this domain, building probability trees and unnecessarily large matri-

2. Dr. Steve Romaniuk from the National University of Singapore conducted these experiments.

ces does not make a better classifier than the 17 correctly set weights used by a simple linear model.

A comparison of the models' average performances, using the parameters as described above, is consolidated in Table 3. The table is organized into two sections, one containing the results of unablated input, and the other containing the results of the best-performing ablated input pattern. A **type** of 0 in the last column specifies that the unablated input performed better than all ablated input patterns; otherwise, **type** represents one of the seven ablation patterns, which were handpicked for their geometric appeal and are further discussed in the Appendix.

| Test/ Training set size | Model | No Ablation | | Best Performing Ablation Patterns | | Type |
|-------------------------------|-------------------|-------------|--------|--------------------------------------|--------|-------|
| | | 3-bit | 1-bit | 3-bit | 1-bit | |
| 145/290 | Perceptron | 0.9494 | 0.9586 | 0.9494 | 0.9586 | 0 |
| | Perceptron- Cons. | 0.9494 | 0.9563 | * | * | * |
| | BSB | 0.8747 | 0.9023 | 0.9471 | 0.9482 | 7 |
| | BSB- Conflation | 0.8747 | 0.9000 | 0.9368 | 0.9425 | 7 / 1 |
| | BSB- Consensus | 0.8690 | 0.8850 | * | * | * |
| | Nearest Neighbor | 0.9103 | 0.9103 | 0.9333 | 0.9333 | 4 |
| 290/145 | Perceptron | 0.9563 | 0.9586 | 0.9563 | 0.9586 | 0 |
| | Perceptron- Cons. | 0.9563 | 0.9586 | * | * | * |
| | BSB | 0.9540 | 0.9448 | 0.9563 | 0.9563 | 7 |
| | BSB- Conflation | 0.9517 | 0.9471 | 0.9563 | 0.9540 | 7 / 1 |
| | BSB- Consensus | 0.9540 | 0.9471 | * | * | * |
| | Nearest Neighbor | 0.9126 | 0.9126 | 0.9402 | 0.9402 | 4 |

Table 3: Voting-84 testing results

Figures represent the percentage of correctly classified testing vectors, averaged over 3 runs.

Averaging the performances of a model using different training set sizes shows that the Perceptron models performed best, with the BSB models second, and the laggard being Nearest Neighbor. However, if we focus on runs created with 145 training vectors, we see that the BSB models are performing much worse than their counterparts using 290 training vectors. The reason for this poor result can be traced to one abysmally performing instance of the BSB while training with 145 vectors. For the 3-bit Party representation, this network correctly classified 77% unablated input instances, and for the 1-bit about 81%. For both representations it significantly lowered the average performance; however, it shed some light onto other aspects of training and testing networks.

The dataset used to train the abysmal BSB (BSB-a) was the 2nd third of the database, and it was tested on the 1st and 3rd thirds of the database. BSB-a's counterpart was trained on the 2nd and 3rd thirds of the database, and it was tested on the 1st third. The counter-

part network performed as well or better than its homogeneous runs, implying that the addition of the 3rd third of the database to training allowed the BSB network to learn the 2nd third. From Table 3 we see that training on more data improved performance for all models, and that for BSB-a's counterpart, the training on additional records also changed the order of presentation (i.e., after sequentially presenting the records of the 2nd third, records of the 3rd third were presented).

In order to determine if the 2nd third could be better learned, a few BSBs were created utilizing random presentation of the training vectors using the 3-bit Party representation. When tested, their performances averaged around 84%, revealing that there existed presentation orderings that could facilitate learning. In general, however, the 2nd third proved difficult for a BSB to learn.

Except for ablation types 7 and 1, the other ablated input vectors performed poorly (below 50%) while being processed using BSB-a. When an ablated vector performs poorly it is misclassifying input most of the time. If enough ablated vectors perform poorly, a consensus is reached for the wrong classification most of the time. This explains a comparatively low BSB-Consensus performance using 145-record training sets. For all other homogeneous runs, BSB-Consensus performs the same as BSB using unablated input, and Perceptron-Consensus performs as well as Perceptron.

These experiments also show that BSB-Conflation³ is slightly more resilient than BSB-Consensus to poorly performing ablated vectors; however, BSB-Conflation can not provide a performance improvement unless ablated vectors can give rise to correct classification most of the time. In other words, if the majority of ablated vectors is not working, neither is Conflation.

2.1.1 OPTIMAL ABLATION PATTERNS AND GENETIC ALGORITHMS

At this point in the experimentation, a search for optimal ablation patterns was pursued.

The best method for finding optimal ablation patterns is to use exhaustive search. One exhaustive search on the Voting-84 database resulted in a 3-day run which tested all 2^{16} ablation patterns for performance, using 290 neighborhood records, 145 test records, and NN classification performance as the metric for determining superior ablation patterns. The top 100 were saved. There was 1 ablation pattern that could correctly classify 142 test records (~98%) and 44 patterns capable of classifying 141; the remainder of the top 100 classified 140 records correctly.

The problem with a search of this type is that we simply do not have fast enough hardware for running times which are exponential in the number of attributes. The exhaustive search for ablation patterns for 16 attributes took 3 days of Sparc 1 computer time; adding one more attribute would require 6 days; adding 10 more attributes would require about

3. Conflation and BSB-Conflation henceforth will be used interchangeably.

2.82 years. Of course, NN is an ideal model for parallelization, and I would estimate that ablation patterns for 26 attributes would take a few days on a Thinking Machines CM-2; however, at some relatively small number of attributes, exhaustive search becomes unfeasible on any machine. The alternative is to use a Genetic Algorithm (GA) [4a] for near optimal ablation pattern search.

A GA operating on a population of 40 ablation patterns was run 3 times to generate a pool of 120 ablation patterns (duplicates included), from which 7 unique patterns were handpicked for further performance testing on all learning models. Each run took 1 hour of Sparc 1 wall clock time.

The GA worked as follows: An initial population of 40 binary vectors was generated using a uniform random number generator for the interval $[0,1)$. An ablation pattern element took 1 as its value if the generator produced a number $\geq .5$ otherwise it became 0. This resulted in a 50% probability for any element of any vector being a 1 for the vectors of the initial population. A 0/1 binary vector conveniently serves as a mask for an ablation pattern: if the element in the mask is 0, the corresponding attribute is not used.

A fitness value was calculated for each member (vector) in the population. This value was the percent of correctly classified testing records using NN on the same training and testing sets as the exhaustive search. To create *slots* for the roulette wheel [6], the percentage of a member's fitness to total population fitness was used, and a uniform random number $[0,1)$ was generated to serve as the croupier. For each iteration, 10 sets of 2 spins of the roulette wheel were conducted for choosing members for crossover. In other words, for each iteration of the GA two unique members of the population were chosen 10 times for crossover in order to generate 20 new members of the population. The 20 members lowest in fitness were decimated, leaving the new population with the 20 new members from crossover, and the 20 most fit from the previous instance of the population. The algorithm used no mutation and essentially consisted of a fitness function and a reproduction function that included decimation functionality.

Each of the 3 runs was conducted using a unique initial population and 20 iterations of the fitness and reproduction functions. The final populations always contained duplicates. In fact, one run resulted in a population where members were identical. However, the majority of the final population for each run was in the top 100 from the exhaustive search! The first run contained 14 unique ablation patterns in the top 100, two of which were in the top 44; the second run produced 40 duplicates of a pattern in the top 44; and the third produced 13 unique ablation patterns in the top 100, five of which were in the top 44.

The seven patterns used for subsequent testing were selected based on their membership in the top 100, and on variety (e.g., it did not make sense to pick two ablation patterns

that differed by one attribute). These seven handpicked ablation patterns were tested for all models using the 1-bit Party representation, and the results are consolidated in Table 4.

| Train/ Testing set size | Model | No Ablation 1-bit | Best Performing Ablation Patterns 1-bit | Type |
|-------------------------------|-------------------|-------------------------|---|-------|
| 145/2901 | Perceptron | 0.9586 | 0.9609 | 1 |
| | Perceptron- Cons. | 0.9574 | * | * |
| | BSB | 0.9022 | 0.9482 | 1 |
| | BSB- Conflation | 0.9195 | 0.9448 | 1 |
| | BSB- Consensus | 0.9321 | * | * |
| | Nearest Neighbor | 0.9103 | 0.9597 | 4 |
| 290/145 | Perceptron | 0.9586 | 0.9586 | 0 & 1 |
| | Perceptron- Cons. | 0.9517 | * | * |
| | BSB | 0.9448 | 0.9540 | 1 |
| | BSB- Conflation | 0.9494 | 0.9563 | 1 |
| | BSB- Consensus | 0.9517 | * | * |
| | Nearest Neighbor | 0.9126 | 0.9609 | 5 |

Table 4: Ablation types from Genetic Algorithm results

Figures represent the percentage of correctly classified testing vectors, averaged over 3 runs.

Ablation type 1 clearly performed well for the neural network models, and ablation types 4 and 5 served well for NN. The results of these experiments show that performance of an ablation pattern for NN is not correlated one-to-one with its performance for Perceptron and BSB. The ablation patterns detected while using the combination of GA and NN, however, were good enough to allow BSB-Conflation and BSB-Consensus to outperform the basic BSB model. In addition, this method produced an ablation pattern sufficient for a 1-unit Perceptron to set a performance record of 96% for correct Party classification of untrained instances using the Voting-84 database.

The seven ablation patterns all performed better than the unablated input when using NN. The average was 95% correct classification, and picking any of the ablation patterns for final output would have realized a substantial performance improvement over unablated input.

Based on reported performances using the same domain, it seems that probability-based algorithms, such as Quinlan's ID3 and its improvement, C4, have (inherently) the advantage that ablation gives to NN. These algorithms make training decisions based on the amount of information a particular attribute can contribute to a classification. Basic NN has only Euclidean Distance to make its decisions, and it is therefore unable to transcend the effects of attributes whose ambivalence across classes (or lack of information) makes classification less successful.

On the other hand, neural network models incorporate this lack of information in their weights during training; unfortunately, final network weights do not reveal ambivalent attributes. By using ablation in conjunction with Perceptrons, weights that are aligned with ambivalent attributes are ignored, allowing attributes with more information to wage their war across a hyperplane. With BSB, ablated elements are ultimately filled in with values that facilitate the feedback vector's reaching an attractor state while utilizing only unablated attributes. If the values for ablated attributes are those responsible for what would otherwise lead to an incorrect classification for many testing instances, clearly their removal can only improve total classification performance. The problem is finding a subset of attributes that can be removed so that a performance improvement is realized for the testing set as a whole, and from the results utilizing the Voting-84 domain, it is clear that the set of attributes that can be removed is learning-model-specific.

2.2 GENERALIZED TECHNIQUE

One result of these experiments is the development of a technique for improving the classification abilities of Nearest Neighbor and neural network models on some domains. The potential improvement is achieved by introducing ablation patterns and Genetic Algorithms into the models.

The technique is the following:

- 1) Partition the domain into 3 subsets of which two will be used for determining good ablation patterns, and one will be used for testing the model's classification capabilities. Each of the subsets should in turn be divided into training and testing sets.
- 2) Take one of the domain subsets and use a GA to determine ablation patterns which maximize classification performance for the testing records of the subset. The GA's fitness function should utilize the model which this technique attempts to improve, and the value returned will be the performance percentage of the ablation pattern on the testing input. It may be desirable to use another model for convenience. For example, using Perceptron in place of BSB would save a considerable amount of time for many domains.
- 3) Using the second subset of data, take several well-performing ablation patterns from 2, and test for ablation patterns that classify this subset best. The purpose of this step is to find ablation patterns that perform well on data that differ from the data that were used to create them.
- 4) Use the third subset to determine final classification performance.

During steps 2,3, or 4, unablated input should be tested to ascertain whether unablated input will provide for better performance. This will be important as the domain is explored.

2.3 LED-7 DATABASE RESULTS

The second domain used to test the effects of ablation patterns was LED-7. The records of the LED-7 database are generated by a C-code program, and represent the formation of lights turned on for a digit-LED display. An attribute is valued 1 or -1 according to whether the corresponding light is on or not. Noise is incorporated into a record by probabilistically complementing the value of an attribute. In the following experiments, this probability was held at 10%. Based on the classification abilities of other models, it seems that this database presents a more difficult learning task than Voting-84. Some published results include: IB? 70.0%, C4 68.3% - 72.6%, Optimal Bayes 74%, CART decision tree 71%, and Nearest Neighbor 71% [1, 10, 5]. Most of these results are based on different sized training and testing sets, but the models seem to have similar performance.

In order to train a neural network for this task, a binary encoding for the classification values [0..9] must be decided upon. A grandmother cell representation was chosen, where 10 elements of a training vector were used to account for each of the ten values (classes). This representation is discussed further in the Appendix. A 2-of-5 binary encoding using 5 elements to represent classes was also tested, but produced inferior results. 40,000 sequential presentations were used to train the Perceptron, and 16,000 sequential presentations were used for the BSBs. All training was executed with a learning constant of .01111. In addition, both models utilized a Bias element, but during BSB testing it was necessary to ablate the Bias element in order to produce a classification.

Experiments were conducted to test the effects of training with and without noise. The training set without noise contained one instance for each of the 10 possible digits, and the training set with noise contained 400 instances. Three testing sets, each containing 400 records, were generated with noise. The results for these experiments are listed in Table 5, where figures represent the average performance of a model on the three testing sets. Ablation patterns were hand-picked, and are specified in the Appendix. As expected, all models performed marginally better on the training set without noise.

Based on a 10% probability that any attribute's value will be complemented, the probability that a given record is noisy, thus having at least one complemented bit, is around 52%. The noisy training and testing sets for these experiments were examined and found to contain 51.5% noisy instances, of which most contained 1, 2, or 3 complemented bits. Since noisy and clean records are randomly ordered within a training or testing set, the neural network training on the noisy set was unsuccessful. It would appear that some of the adjustments made for a clean instance would be undone within a few presentations by a noisy instance.

Both BSB and Perceptron learned perfectly the training set without noise, to the extent that each training instance was capable of firing only one cell. However, in testing, the network models performed poorly. This can be attributed to the inability of a trained network to give rise to the "firing" of only one element of the grandmother cell representation when a noisy test instance is presented, and resulted in the use of a decision step for determining which of the fired cells to use as the final classification. For both BSB and Perceptron models, the cell with the strongest positive signal was taken to be the classification. In

the event of a tie, the cell representing the lowest digit value was taken as the classification. If no cell fired, the final class was considered unknown. The consensus models did not include unknown responses as part of their calculation.

| Train / Testing set size | Model | No Ablation 10-bit | Best Performing Ablation 10-bit | Patterns Type |
|---------------------------|-------------------|--------------------|---------------------------------|---------------|
| 10-Clean / 400-.1 Noise | Perceptron | 0.6556 | 0.6556 | 0 |
| | Perceptron- Cons. | 0.5962 | * | * |
| | BSB | 0.6237 | 0.6237 | 0 |
| | BSB- Conflation | 0.6312 | 0.6837 | 1 |
| | BSB- Consensus | 0.5962 | * | * |
| | Nearest Neighbor | 0.7312 | 0.7312 | 0 |
| 400-.1 Noise/400-.1 Noise | Perceptron | 0.3858 | 0.3858 | 0 |
| | Perceptron- Cons. | 0.4341 | * | * |
| | BSB | 0.3100 | 0.3235 | 7 |
| | BSB- Conflation | 0.2853 | 0.2853 | 0 |
| | BSB- Consensus | 0.2100 | * | * |
| | Nearest Neighbor | 0.6900 | 0.6900 | 0 |

Table 5: LED-7 testing results

Figures represent the percentage of correctly classified testing vectors, averaged over 3 runs.

2.3.1 Nearest Neighbor and LED-7

Nearest Neighbor proved to be the best performer among the models tested. Since these experiments utilized binary vectors, the Euclidean distance between any two points is equivalent to the Hamming distance of their binary representations. If the records in a noisy training set are considered the points that define neighborhoods, it is very likely that the point representing a testing instance has the same Hamming distance to several training set points with different classes. Consider the following example:

Training points:

N1 = 1, 1, 1, 1, 1, 1, 1, 0

N2 = 1, 1, 1, 1, 1, 1, 1, 8

N3 = 1, 1, 1, 1, 1, 1, 1, 6

N4 = 1, 1, 1, 1, 1, 1, 1, 9

Testing point:

TP= -1, 1, 1, 1, 1, 1, 1, 8

The Hamming distance between the testing point TP and all four training points is 1, but a decision must be made for a final classification. For the NN experiments, the method for deciding among classes was to use the class which was represented most in the training set of points with the minimum Hamming distance from the test point. In the event of a tie, the class whose value was lowest was considered the final classification. For the example above, 0 would be the final classification.

2.3.2 OPTIMAL ABLATION PATTERN FOR LED-7

Table 5 indicates that ablation patterns did not (except in one strong case) increase the performance of a model. It appeared that LED-7 required unablated input. This seems reasonable considering that a value of -1 for a few attributes could make the difference between lights representing a 0,6,8, or 9, implying that most attributes contain very high information to contribute to making a classification. However, through exhaustive search, it was found that one ablation pattern performed as well as the unablated input using NN.

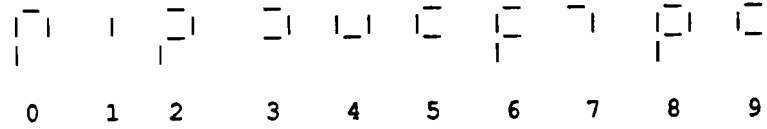
The exhaustive search experiment utilized 1 training set and 3 testing sets each containing 400 records with noise. One of the testing sets was used along with the training set for an exhaustive search for optimal ablation patterns. The unablated and best-performing ablated pattern classified the testing set 69.00% correctly. The two remaining testing sets had an average performance of 68.50% on unablated input, and 71.00% on ablated input, temporarily confirming that an ablation pattern could be used to improve NN on the LED-7 domain.

It should be noted that this optimal ablation pattern did not outperform the unablated pattern on other testing sets of various sizes. Three of the ablation patterns were within 1%, asymptotically, of the performance of the unablated input.

In addition, it was found that these three ablation patterns could be used successfully to classify 100% of an unnoisy testing set on a noisy training set. We would expect this to be the case for the unablated pattern, because 48% of the training set contains unnoisy instances, several of which will be of the same class. The testing instance will have an exact match to many unnoisy training instances, thus breaking any Hamming distance ties contested by noisy instances. The perfect performance using these ablation patterns would suggest that the attributes being ablated provide less information than the others.

Ablation type 1 (specified in the Appendix) implies that attributes 6 and 7 provide the least information for a classification decision. In fact, with no noise, these attributes pro-

vide no information. For example, if lights 6 and 7 are removed from the LED digit, there is still a unique configuration for each number.



2.3.3 TESTING WITH OPTIMAL ABLATION PATTERNS.

A final experiment utilizing the best-performing ablation patterns was conducted using the training set without noise, and 4 noisy testing sets containing 400 instances. The training parameters were the same used in the experiment described at the beginning of this section. The results are consolidated in Table 6.

| Train / Testing set size | Model | No Ablation 10-bit | Best Performing Ablation Patterns 10-bit | Type |
|--------------------------|-------------------|--------------------|--|------|
| 10-Clean/400-.1 Noise | Perceptron | 0.6556 | 0.6556 | 0 |
| | Perceptron- Cons. | 0.6812 | * | * |
| | BSB | 0.6237 | 0.6781 | 1 |
| | BSB- Conflation | 0.6356 | 0.6868 | 5 |
| | BSB- Consensus | 0.6550 | * | * |
| | Nearest Neighbor | 0.7312 | 0.7312 | 0 |

Table 6: LED-7 Testing results with optimal ablation patterns

Figures represent the percentage of correctly classified testing vectors, averaged over 4 runs.

The use of these ablation types helped improve Perceptron-Consensus over Perceptron, and BSB-Consensus over BSB. This result indicates that the optimal ablation patterns give rise to correct classification on many testing instances where unablated input does not.

The results of the Voting-84 experiments reveal that BSB-Conflation can provide a performance improvement over BSB, and in the unnoisy LED-7 experiments, it proves to be the best-performing network model. However, these statements are only true if we consider the final performance of BSB-Conflation as the classification performance of *some* ablation pattern. In general, taking the results of an ablation pattern as final classification

must be accompanied by an *a priori* decision regarding which of the ablation patterns to use for final classification. The results listed in the tables are averages; however, the individual experiments conducted with both domains indicate that an ablation pattern that performed well, consistently performed well for all testing sets on a neural network.

3.0 CONCLUSION

The use of ablation patterns to improve the classification performance of machine learning models was a by-product of the attempts to improve Net-time's ability to tell time. After various implementation experiments, it was found that for several vectorized input instances of the image of a clock, the ablated instance could give rise to the correct time where the unablated instance representing full image information could not. Because of this and the ensuing need to test several ablation patterns in a system requiring high dimensional vector space, a software port to a SIMD massively parallel hardware architecture was pursued, and led to the concept of Conflation.

The use of ablation is not, by any means, pervasive in learning literature, although Kibler and Langley [7] suggest lesion studies as an experimental tool for discovering the effects that a given component or operator has on a learning model, and cognitive simulations use lesion or ablation on the neural network itself as a method for simulating the effects of the loss of neurons. However, the use of ablation while working with the BSB model is quite natural. For example, one way to create an input / output mapping is to partition the elements of a vector into input and output portions. After Widrow-Hoff training, the mapping is tested by zeroing or ablating the output portion of a training vector. The model "fills in" the output portion as the vector is processed. Various tests indicate that BSB's mapping ability is relatively superfluous to a few zeroed elements on the input portion, and experimenting with Net-time showed that certain patterns of zeroed elements facilitated the mapping of several clock images to their approximate times.

The benchmark study in this paper indicates that there is potential for ablation patterns to be used by two neural network models for improving classification on real-world domains. If neural networks are to be used as classifiers, a method for improving their performance is to process (in parallel) several ablation patterns on an input instance, and to reach a classification decision based on an intra- and/or post-process method. The study also indicates that the Nearest Neighbor classifier is improved with ablation patterns if a domain contains attributes that complicate its classification process. For all three models, the Genetic Algorithm provides a method for determining well-performing patterns when exhaustive search is not feasible.

The results of the experiments presented in this paper also show that there exist some domains which are classified better by connectionist models, and some classified better by statistical and probabilistic models. Ablation patterns, Consensus, and Conflation have a promising future for domains benefitting from the connectionist models.

4.0 APPENDIX

The Appendix contains the specifications for the ablation patterns used in the various experiments, the specifications for the Voting-84 and LED-7 domains, and pseudo-code for Widrow-Hoff training and BSB testing. The Voting-84 and LED-7 databases are courtesy of the University of California at Irvine's Repository Of Machine Learning Databases. They are available via ftp from U.C.I.

4.1 NET-TIME ABLATION PATTERNS

| Ablation Type | Description |
|---------------|------------------------------------|
| 0 | Full information. |
| 1 | Ablate every other element. |
| 2 | Leave every 4th element unablated. |
| 3 | Randomly ablate 60 elements. |
| 4 | Ablate left half. |
| 5 | Ablate right half. |
| 6 | Ablate outside quarters. |
| 7 | Ablate middle half. |

4.2 VOTING-84 DATABASE

Vector set specifications

Binary -1/1 vectors of dimensionality 18 or 20.

Elements 1-16: y/n voting pattern.

Element 17: Bias set to -1.

Elements 18: Classification representation, $D=(1)$, $R=(-1)$, or

Elements 18-20: Classification representation, $D=(1,-1,1)$, $R=(-1,1,-1)$.

This data set includes votes for each of the U.S. House of Representatives Congressmen on the following 16 key votes: Handicapped infants, Water project cost sharing, Adoption of the budget resolution, Physician fee freeze, El Salvador aid, Religious groups in schools, Anti-satellite test ban, Aid to Nicaraguan Contras, MX missile, Immigration,

4.3 LED-7 DATABASE

The records of this database are generated by a C-code program and represent the formation of lights turned on for a digit-LED display. An attribute is valued -1 or 1 according to whether the corresponding light is on or not. For noise experiments, each attribute, excluding the class attribute(s), has a 10% percent chance of being inverted.

Vector set specification

Binary -1/1 vectors of dimensionality 18.

Elements 1-7: -1/1 segment pattern generated by U.C.I. C-code.

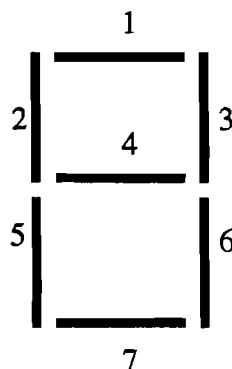
Element 8: Bias set to -1.

Elements 9-18: Classification representation of grandmother cell. (Shown in 0/1 binary below):

| Digit | Classification |
|-------|----------------|
| 0 | 1000000000 |
| 1 | 0100000000 |
| 2 | 0010000000 |
| 3 | 0001000000 |
| 4 | 0000100000 |
| 5 | 0000010000 |
| 6 | 0000001000 |
| 7 | 0000000100 |
| 8 | 0000000010 |
| 9 | 0000000001 |

The records contain 8 attributes and 10 classes, and the number of instances is determined by the user.

Attribute configuration in LED display



Sample records for the digit 8

(NO noise) 1, 1, 1, 1, 1, 1, 1, 8 or

(noise) -1, 1, 1, 1, 1, 1, -1, 8

The following ablation patterns were handpicked based on their aesthetic appeal:

| Attribute Number | | | | | | | |
|------------------|-------------------|---|---|---|---|---|---|
| Type | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | y | y | y | y | y | y | y |
| 1 | y | * | y | * | y | * | y |
| 2 | y | * | * | y | * | * | y |
| 3 | Randomly ablate 3 | | | | | | |
| 4 | y | y | y | y | * | * | * |
| 5 | * | * | * | y | y | y | y |
| 6 | * | * | y | y | y | * | * |
| 7 | y | y | * | * | * | y | y |

y = present * = not present

The following are the 8 top performing ablation patterns from an exhaustive search:

| Attribute Number | | | | | | | |
|------------------|---|---|---|---|---|---|---|
| Type | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | y | y | y | y | y | y | y |
| 1 | y | y | y | y | y | * | * |
| 2 | y | y | y | y | y | y | * |
| 3 | y | y | y | y | y | * | y |
| 4 | y | * | y | y | y | y | y |
| 5 | * | y | y | y | y | * | * |
| 6 | y | * | y | y | y | y | * |
| 7 | y | y | * | y | y | * | * |

y = present * = not present

Note: The order of overall performance is Type 0,1,2,3,4,5,6, and 7.

4.4 PSEUDO-CODE

Widrow-Hoff training algorithm

```
Vector f, g, wv, Diff; /* Binary vectors (-1,1) */
Matrix A, Ai;
float learn_constant;

for(i=1; i <= NUM_OF_ITERATIONS; i++){

    f = g = Next_vector();
    learn_constant = 1 / Inner_product(f,f);

    Matrix_times_vector( A, f, tempg); /*get expected association*/
    Subtract_vectors(g, tempg, Diff); /*get diff vector*/
    Scalar_times_vector(learn_constant, Diff, wv); /*create wv which incorporates learn constant*/
    Outer_product(f,wv, Ai ); /*create delta matrix*/
    Add_matrices(Ai, A, A); /*add delta matrix to associator*/

}
```

BSB testing algorithm

```
#define ALPHA 0.2 /*feedback constant*/
#define LAMBDA 0.9 /*gain*/
#define DELTA 1 /*original input, 1= present*/

/* Next state of vector */
/*  $x(t+1) = LAMBDA * x(t) + ALPHA * Ax(t) + DELTA * x(0)$ ; */

/* Limit_vector function */
for(i=0; i< Dimensionality; i++){
    if( x[i] > 1.0) x[i] = 1.0;
    if( x[i] < -1.0) x[i] = -1.0;
    else no_change;
}

for(i=0; i< NUM_OF_TEST_VECTORS; i++){ /*LOOP for number of pairs*/

    f = Next_Vector(i);
    for(k=451; k< 501; k++) f[k] = 0; /*ablate as you like*/
    for(k=1; k< Dimensionality; k++)
        next_vec[k] = f[k]; /*create first input vector*/

    for(j = 0; j< MAX_NUM_ITERATIONS; j++){
        Matrix_times_vector( A, next_vec, tempf);
        for(k=1; k< Dimensionality; k++){
            z = (next_vec[k] * LAMBDA) +
                (tempf[k] * ALPHA) +
                (f[k] * DELTA);
            next_vec[k] = z; /*create vector for time t+1 */
        }
        Limit_vector( next_vec );
    } /*inner for loop*/
    print_vector( next_vec );
} /*outer for loop*/
```

5.0 REFERENCES

- [1] - D.W. Aha, D. Kibler and M.K. Albert, "Instance-based Learning Algorithms," *Machine Learning*, 6: 37-66, 1991.
- [2] - J.A. Anderson, J.W. Silverstein, S.A. Ritz, and R.S. Jones, "Distinctive features, categorical perception, and probability learning: some applications of a neural model," *Psychological Review*, 84: 413-451, 1977.
- [3] - A. Blumer, A. Ehrenfeucht, D. Haussler and M.K. Warmuth, "Occam's Razor," *Information Processing Letters*, 24: 377-380, 1987.
- [4a]- J.H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd Edition, The MIT Press, Cambridge, MA, 1975 and 1992.
- [4b]- L.B. Booker, D.E. Goldberg and J.H.Holland, "Classifier Systems and Genetic Algorithms," *Artificial Intelligence*, 40: 235-282, 1989.
- [5] - L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees*, Wadsworth International Group: Belmont, CA, 1984.
- [6] - D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company: New York, N.Y., 1991.
- [7] - D. Kibler and P. Langley, "Machine Learning as an Experimental Science," *Proceedings of the Third European Working Session on Learning*, 1988.
- [8] - T. Kohonen, G. Barna, and R. Chrisley, "Statistical pattern recognition with neural networks: benchmarking studies," *Proceedings of the IEEE International Conference On Neural Networks*, San Diego, I-61 - I-68, 1988.
- [9] - J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company: New York, N.Y., 1991.
- [10] - J.R. Quinlan, "Induction of Decision Trees," *Machine Learning*, 1: 81-106, 1986.
- [11]- F. Rosenblatt, "The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, 65: 386-408, 1958.
- [12]- J. C. Schlimmer, *Concept acquisition through representational adjustment*, Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine, CA, 1987.
- [13]- B. Widrow and M.E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, New York: IRE, 96-104, 1960.