# IMPROVED HOPFIELD NETWORKS BY TRAINING WITH NOISY DATA

Fred Clift and Tony R. Martinez
Computer Science Department, Brigham Young University, Provo Utah 84602
*Email*: fred@clift.org, martinez@cs.byu.edu

**Abstract**

*A new approach to training a generalized Hopfield network is developed and evaluated in this work. Both the weight symmetricity constraint and the zero self-connection constraint are removed from standard Hopfield networks. Training is accomplished with Back-Propagation Through Time, using noisy versions of the memorized patterns. Training in this way is referred to as Noisy Associative Training (NAT). Performance of NAT is evaluated on both random and correlated data. NAT has been tested on several data sets, with a large number of training runs for each experiment. The data sets used include uniformly distributed random data and several data sets adapted from the U.C. Irvine Machine Learning Repository. Results show that for random patterns, Hopfield networks trained with NAT have an average overall recall accuracy 6.1 times greater than networks produced with either Hebbian or Pseudo-Inverse training. Additionally, these networks have 13% fewer spurious memories on average than networks trained with Pseudo-Inverse or Hebbian training. Typically, networks memorizing over 2N (where N is the number of nodes in the network) patterns are produced. Performance on correlated data shows an even greater improvement over networks produced with either Hebbian or Pseudo-Inverse training - An average of 27.8 times greater recall accuracy, with 14% fewer spurious memories.*

## 1 Introduction

Hopfield [1] proposed using a fully-connected neural network as an associative memory. The intent is to store a set of bit-patterns in such a way that when a new pattern is given, the network produces the stored pattern that is closest to it. Hopfield suggested using the Hebbian [2] learning algorithm for training the weights of the network and showed that capacity of such networks is approximately $p = 0.15N$ patterns where $N$ is the number of nodes in the network.

Perhaps the most common and best performing general training algorithm currently used for Hopfield-type networks is the Pseudo-Inverse algorithm. It was developed by Personnaz et al., [3][4] and later refined by Kanter and Sompolinsky [5], and Gorodnichy, [6]. This method allows up to $p < N$ linearly independent patterns to be memorized.

While Hopfield associative memories are popular, they have two well-known weaknesses that can be improved upon. These weaknesses are:

1) The number of memorized patterns, $p$, of Hopfield associative memories is limited to approximately $p = N$ in current learning algorithms, where $N$ is the number of nodes in the network.

2) Current learning algorithms tend to produce poorly performing networks when learning linearly dependent patterns.

The work in this paper focuses on improving overall recall accuracy (basin size) and on increasing the number of memories it is possible to store in a Hopfield network (memory density). A new training algorithm is developed, using Back-Propagation Through Time (BPTT) [7] as its basis. This new algorithm is called *Noisy Associative Training* (NAT). NAT differs from existing work in a number of ways. First, the symmetricity constraint on the weights is removed. This effectively allows the number of weights in the network to be doubled, increasing the expressional capability and storage capacity. Second, unlike most current work, self-connection is allowed for each node. Again, this gives the network more expressional capability, and aids in performance with correlated memories.

Hopfield networks trained with BPTT using the memorized patterns as both input and target tend to have very high positive self-connection weights for each node compared to the other weights in the network. Hebbian and Pseudo-Inverse training would also produce high self-connection weights if not explicitly reduced or removed by those algorithms. In each of these cases, networks that merely latch any input pattern are produced. Pseudo-Inverse networks trained beyond capacity (N or more patterns for an N node network) also tend to produce networks that behave in this pathological manner. NAT trains with noisy versions of the memorized patterns to avoid this type of pathological network.

## 2 An Overview of NAT

Noisy Associative Training (NAT) works as follows. Back-Propagation Through Time (BPTT) is used to train the network, using a training set generated from the memorized patterns. To generate the input set for BPTT training, noisy versions of each of the memorized patterns are generated. All $k$-bit error versions (those within a hamming distance of $k$) of one of the desired memorized patterns is added to the

input set for training. This is done for each desired memorized pattern and then duplicate entries are removed from the input set. The target output for each member of the input set is then calculated to be the closest of the intended memorized patterns. If there is a tie for the closest memorized pattern, one of the closest is randomly chosen before training begins, and that patterns is used consistently as the target during training. An example set of memorized patterns for a 3-Node network is {[1,1,1], [1,-1,1]}. For $k$ = 1 bit of error, this would produce an input set of {[-1,-1,1], [-1,1,1], [1,-1,1], [1,1,-1], [1,1,1]} with respective targets of {[1,-1,1], [1,1,1], [1,-1,1], [1,1,1], [1,1,1]}. BPTT for a chosen number of time-steps, $t$, is then performed using this input and output set.

## 3 Performance Metrics

Before discussing comparative performance of NAT verses Hebbian and Pseudo-Inverse training, methods for comparing performance must be considered. Performance of Hopfield networks can be examined in three basic ways. The first is the number of memorized patterns which can be stored in a network. The second is some measure of basin-size or error-correction. The third is the number of spurious memories a network contains. All three of these measures are important to understanding network performance.

For this work the following metric is used. First, any network that memorized more memorized patterns from the training set (including those noisy versions of the memorized patterns relaxing to their appropriate memorized patterns) is superior to a network that memorizes less. This somewhat combines both the first and second methods discussed above. Next, if both networks store the same number of memorized and noisy patterns, then the network that has fewer spurious-memories is considered superior. This last measure can be expensive to calculate. For very small networks the entire pattern space can be enumerated and tested relatively quickly. However, in this work, a different technique is used. All bit patterns within a hamming distance of 5 of any of the memorized patterns are considered. This gives an idea of basin size and also shows spurious memories that are likely to interfere with performance of the network in a real application.

## 4 Test Data

To compare the performance of the three training techniques examined, six data sets were used. The first is uniformly distributed random data. The other five come from the Machine Learning Repository (MLR) [8] provided by the University of California at Irvine. Set one consists of an arbitrary number of 10-bit patterns uniformly distributed across the pattern space. Typically, research with Hopfield memories is done only with random data similar to that of the first data set. Rarely has any work been done showing performance on non-random data. Performance on non-random data is useful in determining how a Hopfield memory would perform in real applications.

The other data sets used are ones commonly used to evaluate classification algorithms. Classification is not the intent of this work. Rather, the output class for each of these data sets is discarded, and the input data is used as described below. This data is known to have interesting correlations, (hence its use in classification problems) and should provide sets of patterns more similar to data stored in a Hopfield memory in real applications.

The second set used was the LED data set. It is comprised of 10 examples of 7 bits in length. These represent the elements of a 7-segment LED display. The patterns indicate which segments should be turned on to represent the digits 0 through 9. Software provided with the LED data set can introduce artificial noise into patterns. This facility was not used. The software used for this research was used to add noise to the test and training data. The second and third data sets from the MLR were the Congressional Voting Records Database and the Zoo Database. The voting records data set represents votes either for or against a set of specific bills under consideration in the US House of Representatives in 1984. 16 attributes (16 actions voted upon) and 151 unique patterns were used from this set. The 151 patterns were selected by first discarding any entries with missing attributes. The classification class provided with this database was discarded, and then all duplicate entries were discarded. What remained was 151 unique pattern vectors of 16 bits in length. The Zoo data set contains many binary attributes representing various features of animals found in a typical zoo, comprising 15 binary attributes and 43 unique patterns. The Zoo data set was constructed using the same method as for the voting records data set. Non-binary attributes were discarded. Instances with missing values were discarded. Finally, duplicate patterns were discarded, yielding 43 unique 15 bit vectors. The Horse Colic and Hepatitis data sets were also used, with the training sets being produced in the same fashion.

## 5 Test Methodology

To compare one weight-setting algorithm to another in terms of overall accuracy and basin-size, a sequence of runs of each algorithm was performed. Separate runs were performed for each data set. For each run, a number of patterns were randomly selected from the available data. Each data set was tested with three quantities of memorized patterns, with 0.5$N$ patterns, $N$ patterns, and 1.5$N$ patterns. Several hundred runs were made, each time selecting patterns from the available data set randomly. Final

evaluation was done by looking at network performance for all the selected patterns and noisy versions of those patterns with up to 5 bits of noise. Each training algorithm was used on each random set of patterns chosen. Results for each experiment (each set of runs) were averaged.

To evaluate absolute storage density, rather than basin-size or overall accuracy, only the random data set was used. Several hundred runs were made for each tested number of patterns while selecting new random data each run. Runs for $p$ = 5, 6, 7, … 29 patterns were performed, all with a network of 10 nodes.

## 6 Results

This section presents simulation results for this work. First, NAT's overall recall accuracy and spurious cycle state performance is examined on both random and correlated data. Then NAT's capacity relative to Hebbian and Pseudo-Inverse training is examined. Finally, a few other observations made during the course of this work are discussed.

Except where noted, all the simulations discussed in this section are done with $t$ = 2 folds (input, hidden-layer, output) for BPTT and $k$ = 1 bit of error. Standard back-propagation was used in BPTT, with a learning rate of 0.005. During training a 'best network so far' was kept and updated after each epoch. If the network failed to converge after a large number of epochs, training was stopped and the best seen network was used.

Note that the Pseudo-Inverse implementation used in this simulation is a modified, improved pseudo-inverse rule with 'reduced self-connection'. A complete description of this algorithm can be found in [6]. The decoupling parameter used for this work is 0.15.

Another important note is that the simulator used in this work used synchronous updating, which is known in some cases to lead to pathological generation of cycle-states or oscillations. Information on the quantity of cycle-states is provided for reference. It is likely that if a different node-updating scheme were used, that all three algorithms would produce fewer cycles. Interestingly, few cycle states were produced even with synchronous updating.

### 6.1 Basin Size With Random Data

Figure 1 is a graph of the average retrieval rates over 800 runs for NAT, Pseudo-Inverse and Hebbian Training. The data set used was the random set described in section 4. The network was trained with five 10-bit vectors, with a new set of vectors chosen for each run. The horizontal axis represents the error rate for all test patterns of a specific bit-error - e.g. '2 bit-error' represents all those patterns that differ from any of the memorized patterns by 2 bits.

Figure 1 shows that, for this experiment, NAT yields higher retrieval rates for the memorized patterns (bit-error = 0) and has a larger basin of attraction when considering all patterns up to a hamming distance of 5 away from the memorized patterns. Note that for the networks examined in this work, on average, Hebbian training will not be able to make all the memorized patterns stable states.

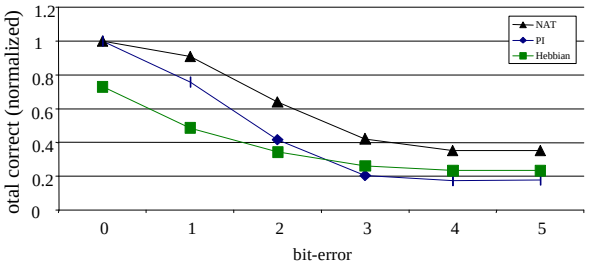


Figure 1 - Random Data, 10 nodes, 5 patterns (avg over 800 runs)

Figures 2 and 3 show similar data for 10 node networks trained with random data, with 10 and 15 memorized patterns, respectively.

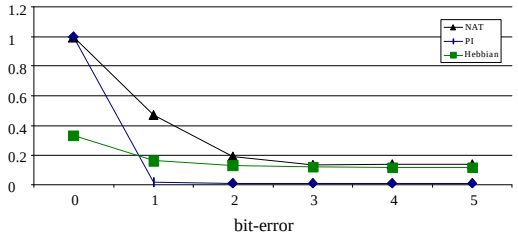


Figure 2 - Random Data, 10 nodes, 10 patterns (avg over 800 runs)

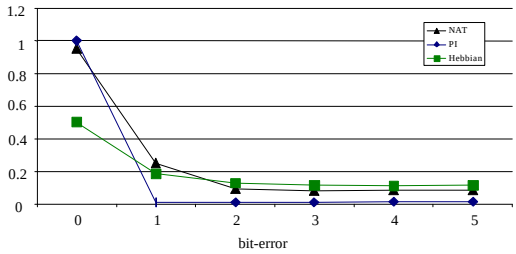


Figure 3 - Random Data, 10 nodes, 15 patterns (avg over 725 runs)

Summarized accuracy results for the random data set is found in Table 1. Each row provides results for one of the experiments. Three experiments were performed, examining basin size (overall accuracy gives an estimate of basin size) for 5 memorized patterns, 10 memorized patterns, and 15 memorized patterns. For example, ‘random 5’ refers to an experiment consisting of a certain number of runs, using random data and 5 memorized patterns. Both the average

number of memorized patterns stored, and the overall recall accuracy are provided for each of the three training algorithms. The numbers are averaged over all the runs for that experiment. Overall recall accuracy is the percentage of tested memories that correctly settle to the closest memorized pattern. The memories tested are those that are within 5 bits of error of any of the memorized patterns. The last two columns are the ratio of the overall performance of NAT compared to the other two algorithms. So, for instance, the right-most number in the first row of data shows that the NAT trained networks for experiment 'random 5' correctly recalled 1.860 times more of the test patterns than the Pseudo-Inverse trained networks on the same data. The last line is an average of each column to give an overall idea of relative performance of the three algorithms. More detailed results and a more complete description of this research is available in [9]

| | NAT | | Pseudo-Inverse | | Hebbian | | NAT acc / | **NAT acc /** |
|---|---|---|---|---|---|---|---|---|
| | # of mem | overall acc | # of mem | overall acc | # of mem | overall acc | Heb acc ratio | **PI acc ratio** |
| random 5 | 5.000 | 0.408 | 5.000 | 0.219 | 3.653 | 0.258 | 1.585 | **1.860** |
| random 10 | 9.961 | 0.158 | 10.000 | 0.014 | 3.319 | 0.120 | 1.311 | **11.334** |
| random 15 | 14.301 | 0.096 | 15.000 | 0.018 | 7.548 | 0.122 | 0.786 | **5.190** |
| average | 9.754 | 0.221 | 10.000 | 0.084 | 4.840 | 0.167 | 1.227 | **6.128** |

Table 1 - Accuracy Summary Per Algorithm (random data set)

On random data (uniformly distributed 10-bit vectors) NAT trained networks correctly recalled 1.227 times more of the test vectors than Hebbian trained networks – 22.7 % better. Note however that the Hebbian trained networks were not able to store more than a few of the memorized patterns as stable states, while NAT networks typically stored all of the desired memorized patterns. The overall accuracy of a network that does not have at least most of the intended patterns as stable states is less interesting. On the same data, NAT was 6.128 times better than Pseudo-Inverse trained networks – 613% better. This extreme difference is due to the poor behavior of Pseudo-Inverse trained networks at or above saturation ($p > N$). As discussed previously, these networks tend to latch any bit-pattern placed in them.

In general, in this work, Hebbian training was not able to effectively compete with either NAT or Pseudo-Inverse training. In any application of a Hopfield associative memory, having 50% or less of the desired memorized patterns as stable states would not be acceptable behavior. In this work a network was considered faulty if it could not memorize most of the desired patterns and in those cases was not considered when comparing network performance.

In Table 1 there is an interesting anomaly. The overall accuracy of Hebbian training for 'random 15' appears to be better than for the other algorithms. The overall recall accuracy *is* higher for this network but the average number of memories that were memorized from this set is 7. It appears that Hebbian training produced networks that were good at recalling only a subset of the memorized patterns. For that subset however, recall accuracy was quite good leading to good overall accuracy. However, with only 7 of the memorized patterns as stable states, the Hebbian-trained networks can not compete.

### 6.2 Spurious Memories with Random Data

Another important factor in examining the performance of NAT is the number of spurious states that are produced by it, compared to the other algorithms. Table 2 shows the average percentage of spurious memories generated by the three training algorithms in the same experiments reported in Table 1 – those experiments done on the random data set.

| | NAT % of spurious | PI % of spurious | Hebb % of spurious | NAT/Heb ratio | **NAT/PI ratio** |
|---|---|---|---|---|---|
| random 5 | 53.952 | 58.428 | 65.803 | 0.820 | **0.923** |
| random 10 | 77.405 | 93.443 | 73.515 | 1.053 | **0.828** |
| random 15 | 84.562 | 98.074 | 80.822 | 1.046 | **0.862** |
| average | 71.973 | 83.315 | 73.380 | 0.973 | **0.871** |

Table 2 - Percentage of Spurious Memories Summary (random data set)

On the random data tests, NAT had 5% less spurious memories on one set, and about 4% more on the other two data sets than Hebbian Trained networks. On all three sets, NAT had at least 5% less spurious memories than Pseudo-Inverse trained networks. The performance of NAT trained networks on the other data sets tended to be a few percent worse than Hebbian trained networks and 10% to 20% better than Pseudo-Inverse trained networks. The results show that NAT outperforms Pseudo-Inverse for this data.

While Hebbian trained networks sometimes had less spurious memories, it is important to remember that these networks often stored only a small percentage of the desired memorized patterns. NAT trained networks had only slightly more spurious memories while memorizing a much larger number of memorized patterns.

### 6.3 Performance on Correlated Data

Performance of Hopfield networks on non-correlated data is what is typically found in studies of this type. However, for Hopfield networks to be useful as associative memories, it is more important to know the typical performance found on data more representative of what the system will be using. Generally, data that is interesting has correlations whether or not they are known in advance. Hence, performance of NAT was evaluated on several data sets that are not random.

In Table 3, the results for average performance on the LED, the Zoo, the House Voter, the Hepatitis, and the Horse Colic data sets are shown. This Table is read in the same way that Table 1 is read. Each line represents a single experiment of many runs selecting a random number of patterns equal to the digit in the first column. `led 3' represents the average performance of the three training algorithms over numerous runs with 3 patterns

| | NAT | | Pseudo-Inverse | | Hebbian | | NAT/Heb | **NAT/PI** |
|---|---|---|---|---|---|---|---|---|
| | # mem | overall | # mem | overall | # mem | overall | ratio | **ratio** |
| led 3 | 3.000 | 0.505 | 3.000 | 0.288 | 2.310 | 0.342 | 1.476 | **1.756** |
| led 7 | 6.989 | 0.379 | 7.000 | 0.074 | 2.872 | 0.189 | 2.007 | **5.157** |
| led 10 | 9.947 | 0.336 | 10.000 | 0.108 | 5.507 | 0.243 | 1.380 | **3.096** |
| zoo 7 | 6.99 | 0.515 | 7.000 | 0.175 | 2.750 | 0.196 | 2.622 | **2.940** |
| zoo 15 | 14.927 | 0.263 | 15.000 | 0.004 | 3.007 | 0.097 | 2.706 | **61.229** |
| zoo 22 | 21.545 | 0.193 | 22.000 | 0.002 | 4.478 | 0.087 | 2.212 | **79.671** |
| voter 8 | 7.891 | 0.395 | 8.000 | 0.114 | 2.101 | 0.099 | 4.001 | **3.459** |
| voter 16 | 15.571 | 0.119 | 16.000 | 0.001 | 1.021 | 0.033 | 3.661 | **113.204** |
| voter 24 | 22.156 | 0.069 | 24.000 | 0.001 | 1.690 | 0.029 | 2.357 | **59.528** |
| hepatitis 10 | 9.995 | 0.231 | 10.000 | 0.015 | 2.379 | 0.086 | 2.681 | **15.717** |
| hepatitis 13 | 12.951 | 0.153 | 13.000 | 0.004 | 2.194 | 0.065 | 2.357 | **38.698** |
| hepatitus 16 | 15.916 | 0.113 | 16.000 | 0.004 | 2.971 | 0.061 | 1.864 | **26.433** |
| colic 3 | 3.000 | 0.513 | 3.000 | 0.271 | 2.176 | 0.376 | 1.366 | **1.894** |
| colic 5 | 4.870 | 0.486 | 5.000 | 0.189 | 2.371 | 0.327 | 1.484 | **2.569** |
| colic 8 | 7.483 | 0.488 | 8.000 | 0.285 | 5.874 | 0.439 | 1.111 | **1.715** |
| average | 10.882 | 0.317 | 11.133 | 0.102 | 2.913 | 0.178 | 2.219 | **27.804** |

Table 3 - Accuracy Summary Per Algorithm (correlated data sets)

randomly chosen from the data set each time. The first several columns represent the overall recall accuracy for each algorithm on the test set. The patterns tested are all those patterns up to 5 bits distant (hamming distance) from any of the memorized patterns. The results on these patterns give an estimate of the sizes of the basins of attraction in each network. The overall accuracy reported is the percentage of patterns in the test set that lie within the proper basin of attraction.

As expected, Hebbian trained networks behave very poorly on this data. The correlations in the data reduce performance and the size of the patterns restricts the networks to a very low number of nodes compared to what would be needed to store all the patterns correctly using Hebbian training.

Pseudo-Inverse trained networks have reasonable performance for those tests in which the number of stored memories is less than the number of nodes in the network. For the other tests, those where the number of memorized patterns was greater than or equal to the number of nodes in the network, Pseudo-Inverse training produced the pathological networks discussed previously. The network simply latches any value put into it. All the memorized patterns are stable states, but, every other bit pattern is also a stable state of the network. These networks had as much as 96% of the test patterns as spurious memories.

The performance of NAT was very good on these data sets. Typical performance shows NAT yielding basins of attraction several times larger than those produces by either Pseudo-Inverse or Hebbian training. The overall accuracy of NAT was an average of 27.8 times larger than Pseudo-Inverse, and 2.2 times larger than Hebb training. The worst that NAT did on correlated data still produced basins at least 71% larger than those produced by Pseudo-Inverse (colic 8).

While Hebbian trained networks appear to be performing better than Pseudo-Inverse trained networks for this data, the average number of memorized patterns stored in the Hebbian networks was a fraction of what the other algorithms could store.

Table 4 shows data on the average number of spurious memories produced during each experiment.

| | Percentage of Spurious Memories | | | Ratio | |
|---|---|---|---|---|---|
| | NAT | PI | Heb | NAT/Heb | **NAT/PI** |
| led 3 | 47.445 | 37.867 | 51.496 | 0.921 | **1.253** |
| led 7 | 57.730 | 71.829 | 49.083 | 1.176 | **0.804** |
| led 10 | 61.820 | 89.164 | 48.203 | 1.282 | **0.693** |
| zoo 7 | 39.979 | 62.777 | 50.125 | 0.798 | **0.637** |
| zoo 15 | 65.831 | 76.698 | 54.497 | 1.208 | **0.858** |
| zoo 22 | 73.879 | 85.325 | 55.163 | 1.339 | **0.866** |
| voter 8 | 51.445 | 72.790 | 75.926 | 0.678 | **0.707** |
| voter 16 | 82.518 | 84.788 | 82.322 | 1.002 | **0.973** |
| voter 24 | 89.642 | 96.482 | 80.743 | 1.110 | **0.929** |
| hepatitis 10 | 69.149 | 85.993 | 73.447 | 0.941 | **0.804** |
| hepatitis 13 | 78.298 | 89.943 | 73.530 | 1.065 | **0.871** |
| hepatitus 16 | 83.488 | 96.008 | 75.456 | 1.106 | **0.870** |
| colic 3 | 47.381 | 41.734 | 46.923 | 1.010 | **1.135** |
| colic 5 | 49.103 | 61.825 | 41.215 | 1.191 | **0.794** |
| colic 8 | 48.044 | 70.201 | 39.278 | 1.223 | **0.684** |
| average | 63.050 | 74.895 | 59.827 | 1.070 | **0.859** |

Table 4 – Percentage of Spurious Memories Summary (correlated data sets)

On average, NAT produced about 14% fewer spurious memories than Pseudo-Inverse, and about 7% more spurious memories than Hebbian. Again, keep in mind that the Hebbian-trained networks had only a small fraction of the desired memorized patterns as stable states.

### 6.4 Other Experiments

The overall capacity of NAT trained networks was estimated by training a 10 node network with a varying number of random patterns. 400 runs were made with each number of memorized patterns and the results were averaged. Figure 4 shows the results. Accuracy is over 90% for those networks memorizing under 20 patterns. At 20 patterns, the accuracy

drops to just below 90%. Given this data the capacity of NAT trained networks is approximately $p = 2N$. Even for $p > 2N$ there were a high percentage of networks that had perfect recall. With $p = 3N$, overall accuracy is over 75%.

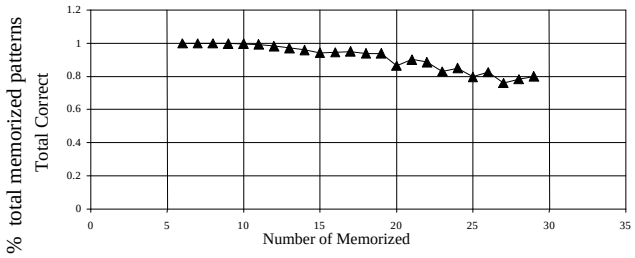


Figure 4 - Capacity Test Results (400 runs each)

Another experiment was performed to determine if training with even noisier data would have a beneficial effect. Training with noisier data appears to have no positive effect on network performance. Training time was increased significantly with no performance gain.

Finally, another experiment was performed to determine if increasing the number of time steps with BPTT would increase performance. Performance for $t = 3$, $t = 5$ for the same data in table 1 was examined. It was noticed that there was some small benefit for training with increased $t$. At best, improvement of overall accuracy by a few percent was achieved. The learning rate for BPTT had to be lowered to 0.0005 to encourage convergence. Even with this adjustment, training was much less likely to converge. Increasing $t$ seems to produce slightly larger basins of attraction, but convergence may not be as likely and training time is increased.

## 7 Conclusions

A new approach to training a generalized Hopfield network was developed and evaluated in this work.

In essence, the algorithm is training Hopfield networks using Back-Propagation Through Time, using noisy examples. This is referred to as Noisy Associative Training, or NAT.

Results show that Hopfield networks storing random patterns trained in this way typically have many times the capacity of networks produced with either Hebbian or Pseudo-Inverse training (average 6.1 times greater recall accuracy on random data, worst case 1.8 times greater recall accuracy on 5-pattern experiment). Additionally, these networks have an average of 13% fewer spurious memories compared to Hebbian or Pseudo-Inverse trained networks. Typically, networks memorizing over $2N$ (where $N$ is the number of nodes in the network) patterns are produced. Performance on correlated data shows an even greater improvement over networks produced with either Hebbian or Pseudo-Inverse training – An average of 27.8 times greater recall accuracy, with no worse than 1.715 times greater recall accuracy, and with an average of 14% fewer (25% greater, worst case) spurious memories.

## 8 References


[1] Hopfield, J.J. (1982) Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. of the National Academy of Sciences USA*, **79**, 2554-2558.

[2] Hebb, D.O. (1949) *The Organization of Behavior*. New York: Wiley.

[3] Personnaz, L. Guyon, I. And Dreyfus, G. (1985) Information Storage and Retrieval in Spin-Glass-like neural networks. *Journal de Physique Lettres*, 46:359-365.

[4] Personnaz, L. Guyon, I. And Dreyfus, G. (1986) Collective computational properties of neural networks: New learning algorithms. *Physical Review A*, 34:4217-4228.

[5] Kanter, I. & Sompolinsky, H. (1987) Associative Recall of Memory Without Errors. *Physical Review A* 35, p. 380-392.

[6] Gorodnichy, D. O. The optimal Value of Self-connection or How to Attain the Best Performance with Limited Size Memory" In *Proceedings of IJCNN '99*, Washington DC, 1999.

[7] Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986) *Learning Internal Representations by Error Propagation* in Rumelhart & McClelland, editors, *Parallel Distributed Processing: Explorations in the microstructure of cognition; Vol. 1 Foundations*, The MIT Press, Cambridge, MA.

[8] Murphy, P. M. & Aha, D. W. UCI Repository of machine learning databases *http://www.ics.uci.edu/~mlearn/MLRepository.html* Irvine, CA: University of California, Department of Information and Computer Science.

[9] Clift, F. Improving the Performance of Hopfield Associative Memories. *Masters Thesis*, Brigham Young University – Computer Science Department, Provo UT, April 2001