Neural Net Term Projects
CS 477/577; 2000
W.L. Miranker

Yale/DCS/TR-1200
June 2000

Table of Contents

Associative Memory – A New Model

Haim Bar[1] and Alexander Ambash[2]
Department of Computer Science, Yale University,
New Haven, CT 06520

Abstract

A new type of neuron is introduced as a building block of an associative memory. We define the memory as a network of such neurons, using a state-space model to define the neurodynamics. We then explore some properties of the neuron and the network and demonstrate its favorable capacity and recall capabilities. Finally, the network is used in an application designed to find trademarks that sound alike.

Key words - associative memory, neural networks.

1. INTRODUCTION

The typical intelligent system that is capable of learning and improving its performance uses memory. The ability to learn and to use the acquired knowledge (experience) to solve problems is related to several qualities of the memory:

- Capacity – the larger the memory is, the more experience it can accommodate.
- Order relations – in order to evaluate success, compare, and measure input, it is essential to have the capability to determine whether an experience A is the same as B, "greater" than B, or "smaller" than B.
- Distribution and locality principles – any data storage device performs better when the knowledge is distributed in such a way that related pieces of information are physically close to each other.

[1] haim.bar@yale.edu
[2] alexander.ambash@yale.edu

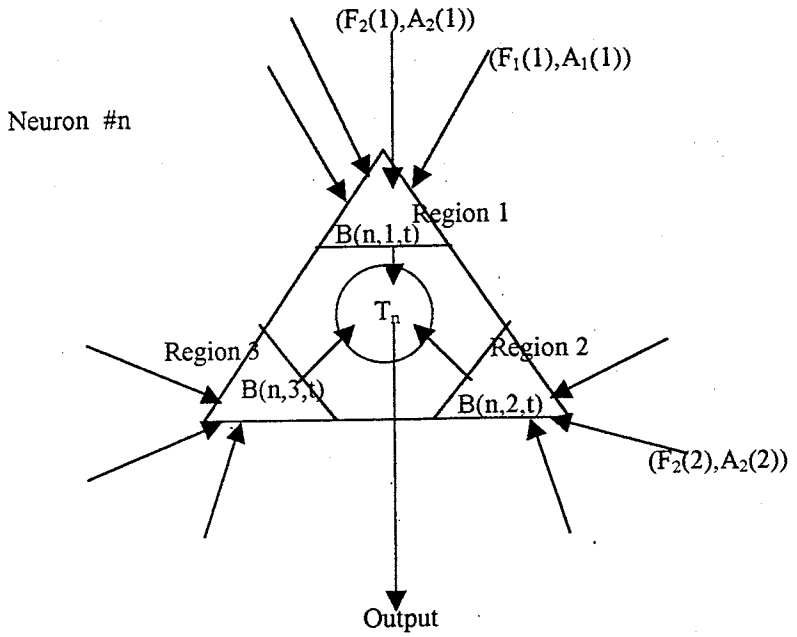
- High connectivity – the system performs better if different regions of the memory are connected. With the appropriate abstraction level, experience stored in one region can be used by another region to solve a new problem.
- Hebbian dynamics – learning by repetition, the Hebb principle of rewarding correlation.
- Parallelism.
- Randomness – two identical systems are expected to perform the same. Adding an element of randomness results in variations, some better and some worse. In nature, the improved variations tend to survive.
- Phantom recalls – when an input that has never been received before or no input at all causes an invalid (spurious) recall, it may result in a chain of other recalls that lead to a valid, innovative, idea.

In this paper we describe and implement an associative memory model that has some of these qualities. In Section 2 we describe a new kind of neuron. In Section 3 we describe a simple network of such neurons, and in Section 4 we introduce mathematical notation to formalize the description of the network in order to analyze its capabilities. In Section 5 we define the neurodynamics. In Section 6 we describe the training and recall processes and in Section 7 we explore some properties of the neuron and the network. In particular we investigate the capacity of the memory and explain how phantom memories can emerge. Finally, in Section 8 we describe a practical application and the results of our experiment in which we train the network to associate between short strings that may sound alike. This is useful for companies when applying to register a new trademark. For a trademark to be considered new, it must not sound like any existing one.

2. THE NEURON – GENERAL DESCRIPTION

We introduce a new model of a neuron. The neuron in this model (figure 1) resembles the pyramidal cell found in brain {Arbib, 1972}. We will use this analogy to explain its functionality, capacity and evolution. As we shall see, this neuron is the building block of an associative memory. Each neuron (cell) can associate between a small number of features, for instance – smell, taste and color.

Figure 1: The structure of a neuron



The neuron consists of a small number of input regions (receptor zones), and one output (axon). Each input region collects input signals from many sources (dendrites). The input signal from each source has two characteristics – the frequency (which encodes the information), and the amplitude (the strength of the signal). Each region is especially sensitive to a small range of frequencies (band). The center of this band[3] is selected randomly at first. After each attempt to learn a specific task (memory), a band that is sufficiently close to the selected input signal is preserved when the neuron fires. This resembles natural selection, where the fittest configuration is genetically preserved. Similarly, if a selected band did not contribute to the improvement in learning the task, a new random value is assigned.

The input to a receptor zone is built up by the superposition of the frequencies from the different sources (dendrites). The receptor zone of a neuron can decompose the input and detect only the frequencies that are within its band, but only if the amplitude of this frequency exceeds a certain threshold. These thresholds may change over time in the process of acquiring memories so that the performance of the network is improved.

Note that the same input (frequency) in different regions may represent different things. For example, if a neuron associates between foods and their taste, then the word

"pepper" may be encoded just like the taste "sweet". Since the inputs arrive from different (and in many cases completely segregated) regions, there should be no confusion. Receiving inputs from segregated zones allows re-use of frequencies, and therefore increases the capacity, as we shall discuss later.

Each input region propagates the deviation of the selected input signal from the center of the band. The neuron fires if each of the input regions receives a signal that is close enough to the center of the region's band. At first, the output is arbitrary. The neurons that receive this output may or may not detect this output. After a few generations, nature (or in the artificial network case, the designer of the network) may create cells that can detect this output signal or the firing neuron can change the output by means of Hebbian learning. In this paper, we will define the output to be a vector. The components of the output vector are the selected inputs to the neuron. This approach will simplify the recall process and the implementation of the network.

When the network is trained, neurons can decrease the tolerance level. The tolerance level is the maximum value of the difference between the band and the selected input frequency that still results in the neuron firing. We say that the neuron specializes. Given an accurate input, it will still be able to recall, but it will be less tolerant to noisy data. We claim that this feature results in an increased capacity of the network (Section 7).

When the neuron receives input in some, but not all the regions, it uses that input to recall the previous inputs to the idle regions. For example, if a neuron has three receptor zones and it fired when the input was the vector (Red, Sweet, Strawberry), then the next time it receives only "Red" and no input from the other regions, it will fire ("Red", "Sweet", "Strawberry").

In Section 4, we shall introduce mathematical notation to formalize the descriptive representation of our neuronal model.

3. THE NETWORK

The basic implementation uses a feed-forward network with one layer of neurons. The neurons are as described in Section 2. Figure 2 illustrates a simple one-layered feed-

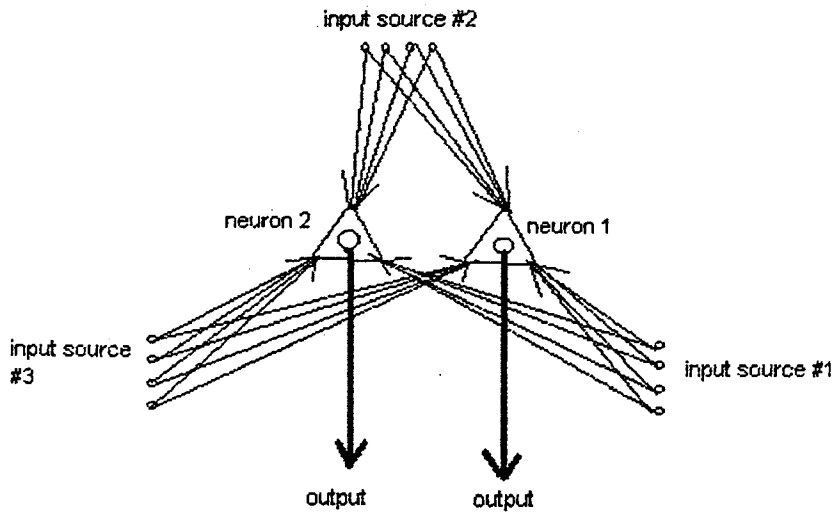
[3] From now on, when we use the term "band" we mean the small range of frequencies around the center of the band.

forward network with two neurons and three input sources. Each neuron has three receptor zones.

In future work, we expect to use a recurrent network in which the output from each of the neurons is fed back to the neuron's input lines. Each element in the output vector is fed back only to the corresponding input region. This allows us to recall higher-order memories. For example, if the input "Red" is associated with "Sweet" and "Strawberry", then feeding "Sweet" and "Strawberry" back to the input regions may result in recalling "Sour" and "Blueberry".

It is also possible to use a multi-layer feed-forward configuration but this involves a more complicated encoding/decoding scheme for the output from the neurons, which we defer for later study.

Figure 2 a one-layered feed-forward network with two neurons



4. THE MATHEMATICAL MODEL

Let N denote the number of neurons. We shall denote the number of receptor regions in neuron n by $R(n)$. For simplicity we assume that all neurons have the same number of receptor regions, i.e. $R(n) = R$ for all $n = 1,2,...,N$.

Input region r in *each* neuron receives a set of signals, $\mathbf{S}(r) = \{S_1(r), S_2(r), S_3(r), ...\}$ (some of which may be null). Note that $\mathbf{S}(r)$ is not a function of the neuron number, n, since in every neuron the r-th region receives the same

set of signals $\mathbb{S}(r) = \{S_1(r), S_2(r), S_3(r), \ldots\}$. $S_i(r)$, the i-th input to the r-th region of each neuron is a vector: $S_i(r) = (F_i(r), A_i(r))$ where $F_i(r)$ is the frequency and $A_i(r)$ is the amplitude of the input signal $S_i(r)$.

As already noted, R (the number of receptor regions) can take small values (typically 2-5), and $|\mathbb{S}(r)|$ (the number of input signals) is typically $O(10^3)$-$O(10^5)$ in the human brain.

Without loss of generality we shall normalize the range of input frequencies $F_i(r)$ so that $F_i(r) \in [0,1]$. The center of the band of input region r of neuron n at time t is denoted by $B(n,r,t)$. In Section 5 we describe how the center of the band, $B(n,r,t)$, changes over time (the neurodynamics). The tolerance level of neuron n is denoted by $T(n)$. For simplicity we assume that all the neurons have the same tolerance level – $T(n) = T$ for all $n = 1,2,\ldots,N$. T is a small positive number, typically $T \le 0.05$.

We also normalize the amplitude $A_i(r)$ so that $A_i(r) \in [0,1]$. Each region can reduce the effective amplitude of the input signal by a factor $0 \le \rho(n,r,t) \le 1$ where ρ is a function of n (the neuron number), r (the region), and the time (t). We shall call ρ the (amplitude) reduction factor. We assume that there is a threshold $0 < \tau < 1$ so that a neuron can detect an input signal $S_i(r)$ only if $\rho(n,r,t) \cdot A_i(r) \ge \tau > 0$. The product $\rho(n,r,t) \cdot A_i(r)$ will be termed the effective amplitude.

We say that a region r in neuron n is *active* at time t if for some input i, $1 \le i \le |\mathbb{S}(r)|$ $F_i(r) \in [B(n,r,t) - T, B(n,r,t) + T]$ and $\rho(n,r,t) \cdot A_i(r) \ge \tau$. In words, the region is active if there is an input signal with effective amplitude that exceeds the positive threshold and with a frequency that is sufficiently close to the center of the band of that region.

Let $I(n,r,t)$ denote those values of i such that $\rho(n,r,t) \cdot A_i(r) > \tau$ and $F_i(r) \in [B(n,r,t) - T, B(n,r,t) + T]$ at time t. $\{S_i(r)\}_{i \in I(n,r,t)}$ is the set of input signals which activate the r-th region in the n-th neuron at time t. Let $i^* = i^*(n,r,t) = \arg\max_{i \in I(n,r,t)} [\rho(n,r,t) \cdot A_i(r)]$ (i.e. we choose the input signal from the set

$\{S_i(r)\}_{i\in I(n,r,t)}$ with the highest effective amplitude). We say that $W(n,r,t)=F_i(r)$ is the *winning* input of region r of neuron n at time t. Note that the amplitude $A_i(r)$ is omitted since the information is frequency encoded. The amplitude is used only to select the winning input.

In the case that $I=\phi$, there is no winning input and we take $W(n,r,t)=-1$. Setting the input frequency to -1 will not allow phantom recalls, as we shall explain in Section 7. In the future we intend to study the network when it does allow for phantom recalls to emerge.

We assume that $B(n,r,t)$ and $W(n,r,t)$ are independent and identically distributed with a uniform distribution over $[0,1]$. Neuron n fires if all of its input regions are active at the same time. That is,

$$\text{neuron } n \text{ fires if } \max_{r=1,2,\ldots R}\left\{\left|W(n,r,t)-B(n,r,t)\right|\right\}<T. \tag{1}$$

5. NEURODYNAMICS

We use a state-space model as defined in {Haykin, 1999, p 666}: The states are $R\times N$ matrices, $\mathbf{B}(t)=\left[B(n,r,t)\right]_{n=1,2,\ldots N}^{r=1,2,\ldots R}$ where N is the number of neurons and R is the number of regions in each neuron. A matrix $\mathbf{B}(t)$ describes the band centers of all the neurons as they evolve over time.

Recall that region r receives a set of signals $\mathbf{S}(r)=\{S_1(r),S_2(r),S_3(r),\ldots\}$ such that $S_i(r)=(F_i(r),A_i(r))$ where $F_i(r)$ is the frequency and $A_i(r)$ is the amplitude of the i-th input signal $S_i(r)$. Band dynamics are defined as follows: for every element $B(n,r,t)$ in the matrix $\mathbf{B}(t)$ we define the change in the band over time by:

$$\frac{dB(n,r,t)}{dt}=\sum_{i=1}^{|\mathbf{S}(r)|}\frac{\operatorname{sgn}\left(F_i(r)-B(n,r,t)\right)\cdot\rho(n,r,t)\cdot A_i(r)}{\left|F_i(r)-B(n,r,t)\right|^2} \tag{2}$$

In words, band r in neuron n is "attracted" to every input signal received by that region, $(F_i(r),A_i(r))$, with a force that is proportional to the inverse of the squared distance between $F_i(r)$ (the frequency of i-th input to region r) and the band of that region,

$B(n,r,t)$. Also note that the force with which the i-th input signal "attracts" the band is proportional to the effective amplitude of that signal.

For reasons of stability in the case that region r in neuron n becomes active (namely, there exists $W(n,r,t)=F_i(r)\neq -1$ such that $|W(n,r,t)-B(n,r,t)|<T$), we define:

$$\frac{dB(n,r,t)}{dt}=0 \tag{3}$$

Equation (2) resembles the description of the motion of an object in a gravitational field. The inputs are analogous to the stars, and the bands of all the neurons to much smaller objects in space. The input signal $S_i(r)$ has a "gravitational constant" $\rho[n,r,t]\cdot A_i[r]$.

Equation (3) guarantees that if some band $B(n,r,t)$ is close enough to an input frequency $F_i(r)$ from the set $\mathbf{S}(r)$, that band will not change. In our celestial analogy this situation corresponds to a small object getting so close to a star that it cannot escape the gravitational force.

The reduction factor of the winning input signals also changes over time by means of Hebbian learning. Formally, the amplitude dynamics is defined as follows:

$$\rho[n,r,t+dt]=\begin{cases}\eta+(1-\eta)\cdot\rho[n,r,t] & \text{if neuron } n \text{ fires and region } r \text{ is active}\\ (1-\eta)\cdot\rho[n,r,t] & \text{if neuron } n \text{ does not fire and region } r \text{ is active}\end{cases} \tag{4}$$

where $0<\eta<1$ is a small learning rate parameter. When a region becomes active, the reduction factor of the winning input $W[n,r,t]$ is increased if the neuron fires and decreased if the neuron does not fire.

6. TRAINING AND RECALL

To train the network, we stimulate it with the inputs that we want it to memorize. A "*memory*" is an R-dimensional vector $M=(m[1],m[2],...,m[R])\in[0,1]^R$. Our goal is to create a network that associates each of the features $m[r]$, with the other features, $m[r']$, *for* $r'\neq r$. We denote the set of memories by $\mathbf{M}$. For every memory $M_j\in\mathbf{M}$ we define the inputs to each region as follows:

$$S_i[r]=\begin{Bmatrix}(m_j[r],1) & i=1\\ (0,0) & i\neq 1\end{Bmatrix}_{i=1,2,\ldots,|S[r]|}$$

In the training phase we eliminate noise by defining $S_i[r]=(0,0)$ for $i\neq 1$. The neurons in the network receive only the value $m_j[r]$ in region r since it is the only signal with positive amplitude. Therefore, there is only one signal that can activate region r in each neuron (that signal is $S_1[r]$ according to our definition). $S_1[r]$ is the only input that "attracts" the r-th region in all the neurons.

A training iteration consists of a single input $M_j \in \mathbf{M}$ that stimulates the neurons and the adjustment to the bands and the reduction factors that follow that input. We expect that after a number of training iterations some neurons will be "attracted" to some memories and fire when the memory is introduced to the network. That is to say, for some memory $M_j \in \mathbf{M}$, there will be a neuron n such that

$$\max_{r=1,2,\ldots,R}\left\{\left|B[n,r,t]-m_j[r]\right|\right\}<T \tag{5}$$

The training stops when $\max_{r=1,2,\ldots,R}\left\{\left|B[n,r,t]-m_j[r]\right|\right\}<T$ for all the memories $M_j \in \mathbf{M}$, or after a specified maximum number of iterations. The network can recall a memory $M_j \in \mathbf{M}$ if there exists a neuron n such that $\max_{r=1,2,\ldots,R}\left\{\left|B[n,r,t]-m_j[r]\right|\right\}<T$.

If after a number of training iterations $\max_{r=1,2,\ldots,R}\left\{\left|B[n,r,t]-m_j[r]\right|\right\}<T$ for all the memories $M_j \in \mathbf{M}$, each neuron in the network decreases the tolerance level to $T'<T$. In other words, when the network can recall all the memories, each region in each neuron will become less tolerant to deviations from the band of that region, $B[n,r,t]$. In Section 7 we will show that a smaller tolerance level in all the neurons results in increased memory capacity of the network.

In the recall phase, the activation of a single region is sufficient to make the neuron fire. Since we defined the output to be a vector of the winning inputs during the training phase, an input that activates a single region, results in R recalled features ($R-1$, if we exclude the input).

7. PROPERTIES OF THE MODEL

Recall our assumption that all neurons have the same number of input regions (R) and the same activation tolerance level (T). With the assumption made in Section 4 that the winning input $W[n,r,t]$ and the r-th band in neuron n $B[n,r,t]$, are independent and identically distributed with a uniform distribution over $[0,1]$, we get

$$prob\left[\left|W[n,r,t]-B[n,r,t]\right|<T\right]=2T-T^2 \tag{6}$$

See Appendix C for proof.

As a result, we get that

$$prob\left[\max_{r=1,2,\ldots,R}\left\{\left|W[n,r,t]-B[n,r,t]\right|\right\}<T\right]=\left(2T-T^2\right)^R \tag{7}$$

So $\left(2T-T^2\right)^R$ is the probability that a neuron will fire, given a specific band configuration and a specific input.

Therefore, we shall choose the total number of neurons to be $\frac{C}{\left(2T-T^2\right)^R}$ where $C\geq 1$. This is highly redundant, but it increases the probability that for every input vector (every memory) there will be a neuron that fires.

The memory capacity of the network depends on T and R (equation (7)). A necessary condition for perfect recall of all the input frequencies to region r is $\left|F_i[r]-F_j[r]\right|>2T \quad \forall i,j=1,2,\ldots,\left|\mathbf{S}[r]\right|$ (recall that $\left|\mathbf{S}[r]\right|$ is the number of all the distinct inputs to region r in all the neurons). To see this note that if there are i,j such that $\left|F_i[r]-F_j[r]\right|<2T$, then there may be a neuron n with a band $B[n,r,t]$ in region r such that $\left|B[n,r,t]-F_i[r]\right|<T$ and $\left|B[n,r,t]-F_j[r]\right|<T$. In this case two *different* input signals ($S_i[r], S_j[r]$) will activate the same region in the n-th neuron.

When the condition $\left|F_i[r]-F_j[r]\right|>2T \ \forall i,j=1,2,\ldots,\left|\mathbf{S}[r]\right|$ holds, $\frac{1}{2T}$ input signals (frequencies) can be detected in region r.

The frequency re-use mentioned in Section 2 increases the capacity of each cell. The number of distinct signals that each cell can receive is $\frac{R}{2T}$. Each input signal to a single region will result in $R-1$ recalled features from the other regions, which is more efficient than recalling only one feature from every input (compared with the correlation matrix memory {Haykin, 1999, p 82}).

When the network is trained, it will need only $\frac{1}{2T}$ neurons to recall any one of the $\frac{R}{2T}$ memories. The correlation matrix memory requires $\frac{R}{2T}$ neurons to recall $\frac{R}{2T}$ memories. This improved performance of our network memory is a result of the frequency re-use. However, to get perfect recall of $\frac{R}{2T}$ memories using only $\frac{1}{2T}$ neurons we have to start with more than $\frac{1}{2T}$ neurons.

When the neurons specialize (T is decreased), the capacity grows, allowing more inputs to be associated and then recalled. This follows directly from the fact just observed that the capacity is proportional to the inverse of T.

Decreasing the effective amplitude of a winning input in a region when a neuron doesn't fire may also increase the capacity. The band of that region, $B\{n,r,t\}$, is attracted to input signals with a force that is proportional to the effective amplitude. When the effective amplitude of the winning input frequency becomes small enough, a new input with a different frequency will be the winner. The neuron may then be able to store and recall a different memory.

Note: were we to encode the winning input as $W\{n,r,t\} \in [0,1]$ instead of $W\{n,r,t\} = -1$ when there is no input to region r may result in "phantoms" – recalled memories without any input.

8. AN APPLICATION

In our experiment we used the network to find homonyms, words that are spelled differently, but sound alike. The ability to detect such words is very important for companies seeking to register a new trademark. One criterion for a trademark to be

considered new is that it doesn't sound like any existing one. Using another company's trademark may result in great liability. The application that is required to handle a phonetic encoding of this sort is different from text-to-speech applications. The latter can benefit from using a database of dictionary words, which makes the pronunciation task somewhat easier. Trademarks however, are not subject to spelling rules. For example, "toysRus" sounds like "toys are us", but the former cannot be found in a dictionary. Building a trademark dictionary is a plausible idea, but since hundreds of new words are invented every week, the size of the database will increase, which means that maintaining it becomes more difficult.

During the training phase the network is presented with simple rules, which associate between pairs of one or two-letter strings. For example, "MN" is associated with "N", "GH" is associated with "F", and "PH" is also associated with "F". In most cases we treat all the vowels as if they were the same input. Appendix A contains the complete list of rules. We encode every possible input as described in Section 4. For example, the pair (GH,F) may be encoded as (0.313, 0.724) and the pair (PH, F) by (0.543, 0.724). We submit all the input pairs to the network, and repeat this step until every input pair has a neuron that fires when it receives this input pair. Then, as we described in Section 2, the neuron can associate between the two input values.

In the next step, a word is parsed into one or two letter sub-strings. For example, the word "ROCK" will be parsed in the following ways: R-O-C-K, R-O-CK, R-OC-K, RO-C-K and RO-CK (these are the valid partitions of the word ROCK). Then for every partition of the word, we submit its components to the network, one at a time. If the network associates the input with other strings, it outputs the other strings. Otherwise, it outputs the input string. For example, if the input is R-O-CK, the network will recall R-A-K (where A is the representation of any vowel).

At the end of the process we get several "tokens" that are associated with the input (the tokens are strings that are pronounced like the input). Word marks that sound alike should have a mutual token. For example, it is easy to see that ROCK, ROK, and ROCQ are all associated with the token ROK.

In our experiment we assigned a random value to each string that appears in appendix A. We started with a set of neurons, each having two random bands. All the

neurons have the same activation tolerance level T which is smaller than $\max_{i,j}|F_i - F_j|$ where F_i, F_j are all the possible input frequencies. We trained the network by feeding the input vectors one at a time. In Appendix B the distribution of the inputs is shown in figure 3. The points in $[0,1]^2$ that correspond to the inputs are selected randomly and never change. Figure 4 shows the initial distribution of the neurons. Figure 4 illustrates that in their initial state the neurons are distributed uniformly.

We repeated the training phase 800 times. Figure 5 shows the distribution of the neurons after 800 epochs. The first thing that we observe when looking at figure 5 is that most input vectors (memories) attracted several neurons. The attracted neurons can be used to recall one feature when they receive the other. For example, the sound "k" corresponds to the value 0.17 along the vertical axis. The strings "c" and "ck" correspond to the values 0.450 and 0.490 respectively, along the horizontal axis. There are neurons that fire when the input to the first region is 0.450 and there are neurons that fire when the input to the first region is 0.490. They all output 0.170 which represents the sound "k". As a result, if a person or a company is considering an application for a trademark using the string "hard roc café" the network will detect that it's similar to "hard rock café". These two strings have a mutual token in which "k" is used instead of "c" or "ck" in the words "roc" and "rock".

Figure 5 shows that the network organized itself so that many of the neurons lay within T from at least one component of a memory M. Essentially, it formed a grid of neurons that is fitted to accept not only the input vectors $(F_1\{1\}, F_1\{2\})$ and $(F_2\{1\}, F_2\{2\})$, but also $(F_1\{1\}, F_2\{2\})$ and $(F_2\{1\}, F_1\{2\})$. The neurons learn to expect the input values $F_i\{r\}$ $j=1,2$; $i=1,2,\ldots,|\mathbf{M}|$.

In our application, this result may be interpreted as the process of acquiring a vocabulary of homonyms. The network starts with no prior knowledge. First, the neurons learn to detect the atomic homonyms (this happens when the distance between one of the bands of the neuron and a component of one of the inputs is less than T). Some neurons learn to associate between pairs of homonyms. This happens when both regions in a neuron are activated when one of the memories (input vectors) is presented to the network. We expect that over-training the system will prevent the network from learning new associations. Every neuron that fires when the a memory is introduced to

the network cannot be changed (according to condition (3) in Section 4) and therefore, cannot learn to associate between other homonyms. We would like to stop the training process when every "memory" has at least one neuron that fires when the network receives this memory as input.

REFERENCES

1. Haykin, Simon (1999). *Neural Networks, a Comprehensive Foundation.* New Jersey: Prentice Hall
2. Hebb, D.O. (1949). *The Organization of Behavior. A Neuropsychological Theory.* New York: Wiley.
3. Singh, Jagjit (1966). *Great Ideas in Information Theory, Language and Cybernetics.* New York: Dover Publications, Inc.
4. Arbib, Michael A. (1972). *The Metaphorical Brain.* New York: Wiley-Interscience.
5. Anderson, John R. and Bower, Gordon H. (1973). *Human Associative Memory.* Washington D.C.: V.H. Winston & Sons
6. Kohonen, Teuvo (1977). *Associative Memory. A System-Theoretical Approach.* Berlin; New York: Springer-Verlag
7. Skapura, David M. (1996). *Building Neural Networks.* New York: ACM Press.

APPENDIX A

The following is a list of associated strings that sound alike. For example "mb" is associated with "m", as in the word "climb".

bb=b	ff=f	mn=m	ss=s
bt=t	g=j	mn=n	ss=sh
c=ch	gg=g	nn=n	ss=z
c=k	gg=j	pb=b	st=s
c=s	gh=f	ph=v	te=ch
c=sh	gh=g	ph=f	th=th
cc=k	gh=S	pn=n	th=t
cc=ks	gi=j	pp=p	ti=ch
ch=ch	gm=m	ps=s	ti=sh
ch=k	gn=n	pt=t	ts=s
ch=sh	gu=g	q=k	tt=t
ci=sh	h=S	qu=k	vv=v
ck=k	is=i	re=Ar	w=S
cq=k	j=h	rh=r	wh=w
cu=k	ju=w	rr=r	wh=h
cz=ch	kn=n	s=sh	x=z
cz=z	ld=d	s=z	x=ks
dd=d	lf=f	sc=s	y=A
dg=j	lk=k	sc=sh	z=s
di=j	ll=l	se=sh	zz=z
dj=j	lm=m	sh=sh	zz=ts
ed=d	mb=m	si=ch	
f=v	mm=m	si=sh	

APPENDIX B

The first graph shows the distribution of the input vectors. The second graph shows the initial distribution of the bands in all the neurons. The third graph shows the distribution of the bands after 800 epochs.

Figure 3 The distribution of the inputs.

The points in $[0,1]^2$ that correspond to the inputs are selected randomly and never change.

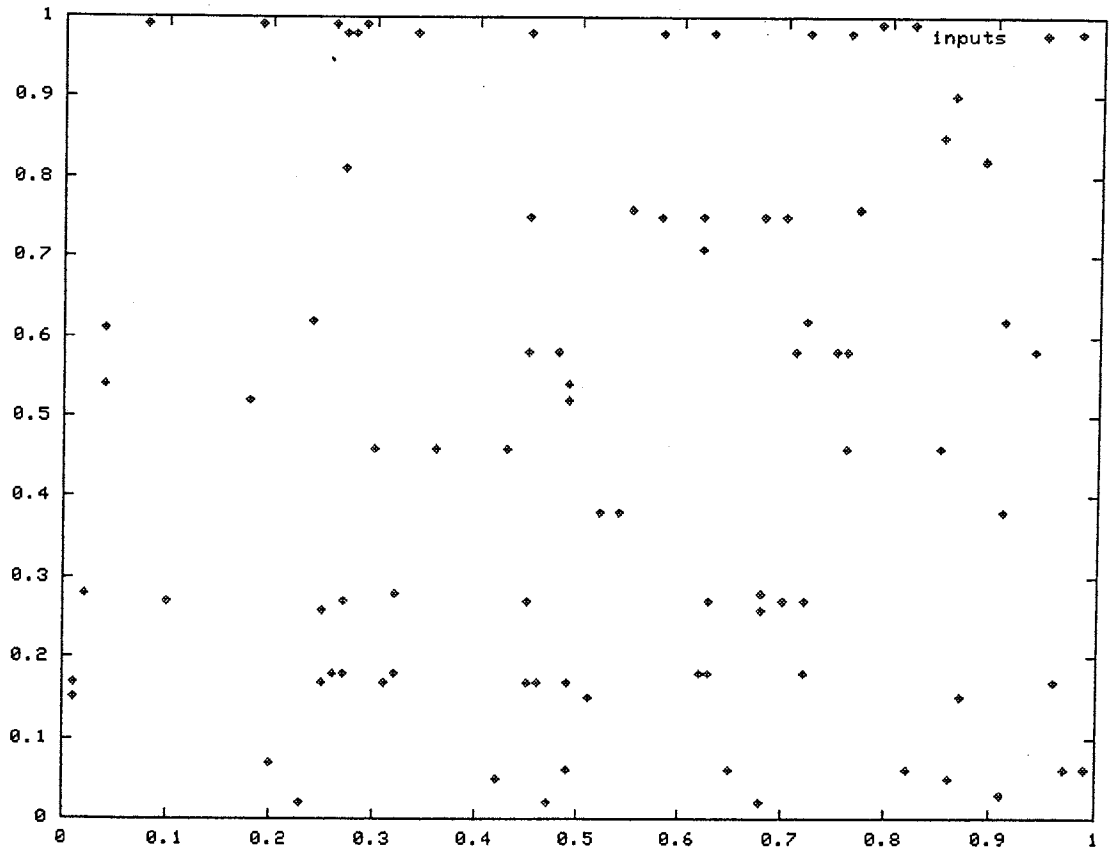


Figure 4 The initial distribution of the neurons.

In their initial state the neurons are distributed uniformly.

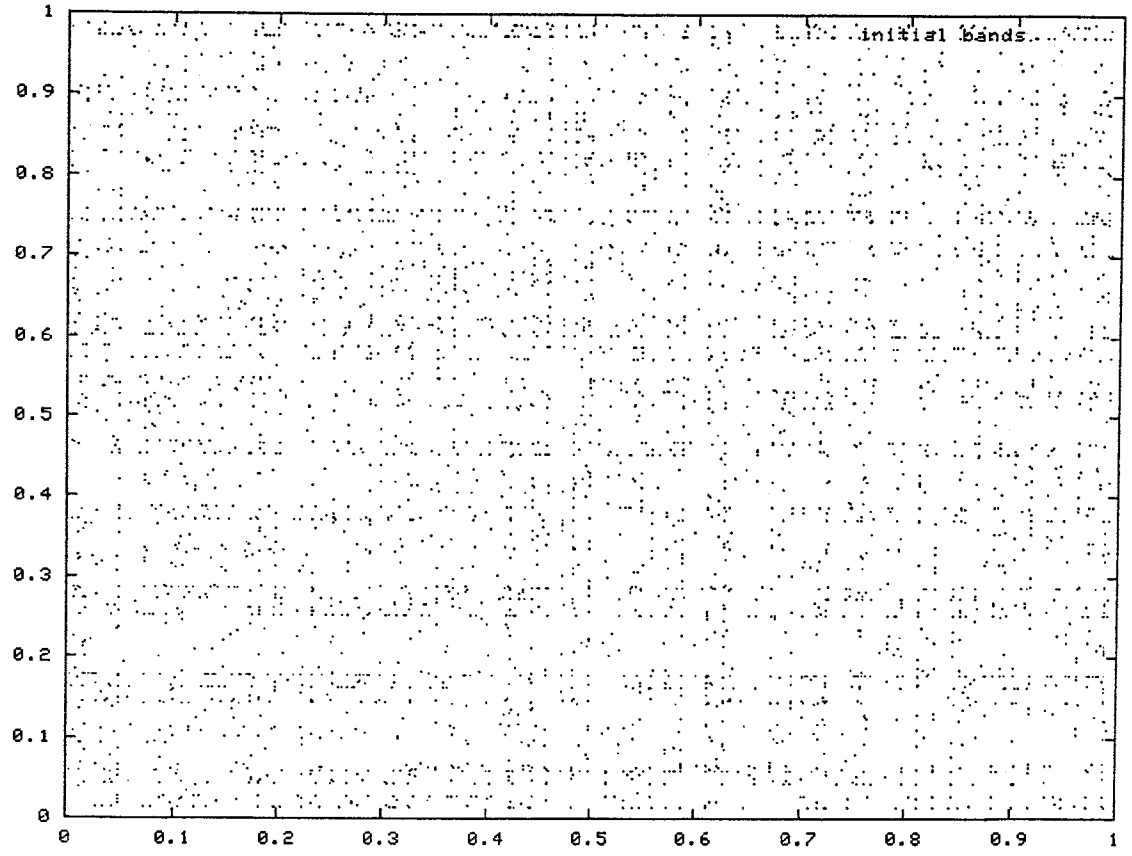
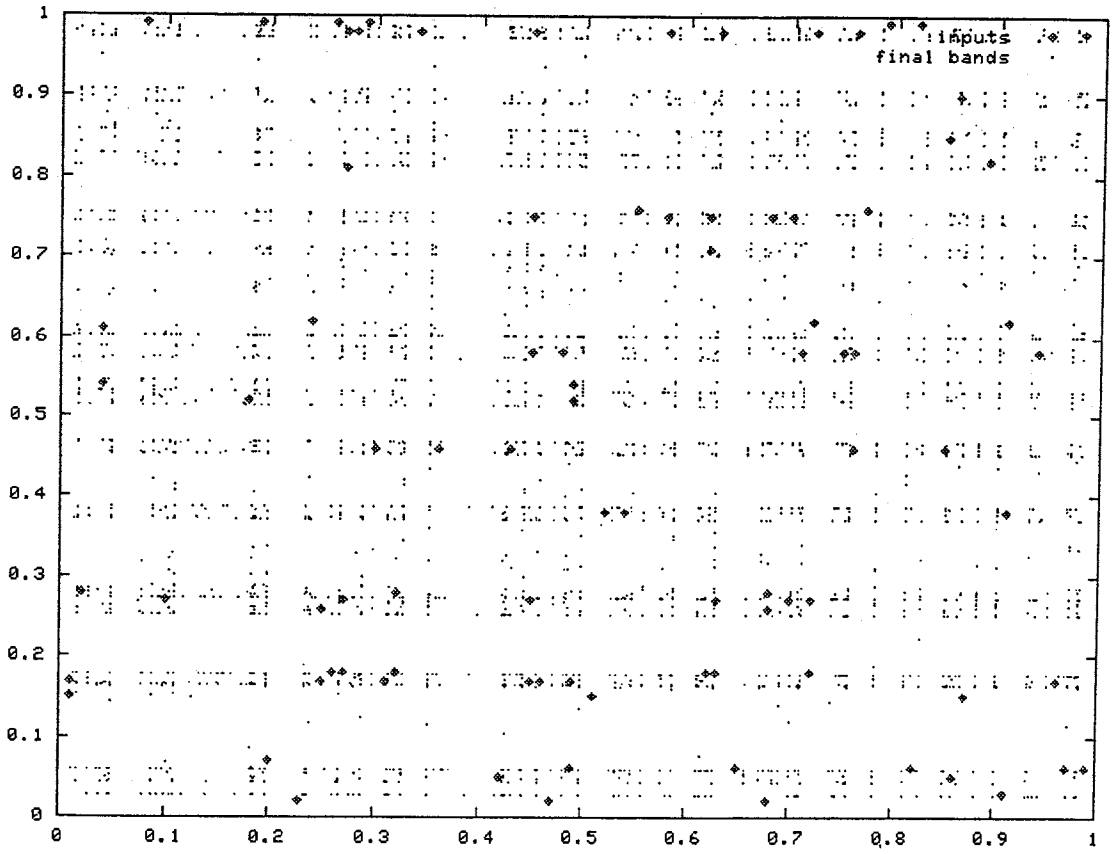


Figure 5: The distribution of the neurons after 800 iterations.

Most inputs attracted several neurons to their T-neighborhood. The network organized itself so that many of the neurons lay within T from at least one component of an input vector F. The neurons learn to expect the input values $\{F_{j,i}\}$ j=1,2; i=1,2,...,N.



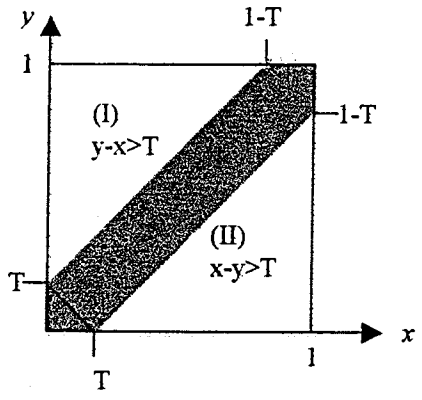
APPENDIX C

Claim: If x, y are independent and identically distributed random variables with a uniform distribution over $[0,1]$ then $Prob[|x-y|<T]=2T-T^2$

Proof:

$$Prob[|x-y|<T]=$$
$$1-Prob[|x-y|>T]=$$
$$1-(Prob[x-y>T]+Prob[y-x>T])=$$
$$1-(Prob[y<x-T]+Prob[x<y-T])=$$
$$1-\left(\int_0^1 \max(0,x-T)dx+\int_0^1 \max(0,y-T)dy\right)=$$
$$1-\left(\int_T^1 (x-T)dx+\int_T^1 (y-T)dy\right)=$$
$$1-\left(\left(\frac{x^2}{2}-xt\right)\Big|_T^1+\left(\frac{y^2}{2}-yt\right)\Big|_T^1\right)=$$
$$1-\left[\left(\frac{1}{2}-T-\left(\frac{T^2}{2}-T^2\right)\right)+\left(\frac{1}{2}-T-\left(\frac{T^2}{2}-T^2\right)\right)\right]=$$
$$1-(1-2T+T^2)=$$
$$2T-T^2$$

The shaded area in the following diagram represents the pairs (x,y) that satisfy $|x-y|<T$. The area of triangles I and II is $\frac{(1-T)(1-T)}{2}$ each. Therefore, the area of the shaded strip is $1-2\cdot\frac{(1-T)(1-T)}{2}=2T-T^2$



Facial Expression Recognition Performed by a Feed-forward Neural Net with Backpropagation

Brian Pasley[1] and Daniel Brambila[2]
Yale University, Computer Science Department
New Haven, CT 06520

Abstract

In this experiment we attempt to create a neural network, which can recognize the six basic emotional facial expressions. We implemented a feed forward backpropagation neural network, using training exemplars, in order to produce emotional classifications. The results were positive for a subset of these six emotions, but negative when all emotional categories were included for training. The possibility of improving performance was explored by preprocessing the images to reduce irrelevant information and implementing an additional hidden layer in the network to maximize feature detection. These supplementary techniques resulted in insignificant improvements for the entire emotional set. Failure of the net to learn all six emotional categories appeared to be due to two particular emotional categories, surprise and fear, that share very similar features. More effective image preprocessing may improve the ability of the hidden neurons to extract salient components and form a more accurate mapping of facial features of emotion.

1. INTRODUCTION

Recognizing emotional facial expressions is thought to be a biologically innate human ability (Matsumoto, 1998). Indeed, emotional facial expressions appear to be consistently recognized and expressed across culture and ethnicity (Matsumoto, 1998), suggesting a degree of universality. The observation of brain areas that appear to be specifically involved in face processing (Kuskowski & Pardo, 1999) along with the striking consistency and speed with which faces and emotions are recognized suggests

[1] bpasley@aya.yale.edu
[2] daniel.brambila@yale.edu

specialized brain processing for faces. Studies comparing object and face perception have suggested that these two classes of stimuli are processed differently by the brain. Objects appear to be perceived based on their isolated features, while faces appear to be processed not on isolated features, but rather holistically as "undecomposed wholes." (White, 1999). In other words, face perception appears to involve all major features of faces (mouth, nose, eye areas) viewed as a while rather than as separate components. This perceptual strategy has interesting parallels with a method of pattern classification used in many neural network applications. Feedforward neural networks often consist of an input and output layer, as well as one or more hidden layers of units (neurons). There are, in general, far fewer hidden neurons than input neurons which results in a nonlinear transformation of the input data that maps input into a "feature space" (Haykin, 1999, pp. 199) represented by the weights and interactions of these hidden neurons. Such a nonlinear mapping results in a holistic representation of the input data since features are not individually extracted for processing, but rather processed according to their configuration as a whole (Cottrell & Metcalfe, 1991). Using this biologically motivated strategy, we implement a neural net to classify six basic emotional expressions. In addition to exploring the validity of the holistic processing strategy, there are also practical applications for a successful automated expression classifier. Many lines of psychological research use facial expressions as stimuli, which initially require subjective classification into the emotional categories. An automated classifier such as a neural net trained on the pixel values of the image would be a useful assistant in supporting these subjective classifications.

2. EXPERIMENT ONE

2.1 Images as Input

A set of grayscale images of facial expressions was used as input. Images came from a database of face images used in previous psychology experiments (Pasley, 2000). These images were subjectively categorized as happy, sad, fearful, disgust, angry, and surprise by human observers. In order to minimize excessive and irrelevant information, each image was cropped to exclude the ears and to include only the region between the forehead and chin. To limit the number of inputs and make network training time reasonably short, each image was resized to 30 x 38 pixels, retaining the average height to width proportion of the face. Inputs were encoded by reading in the value of each pixel, x (ranging from 0 to 255 for grayscale intensity) and normalized by dividing by 255.

$$x_i = \frac{x}{255}$$

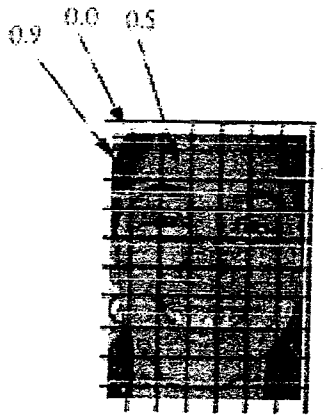


Figure 1: Grayscale values (after normalization) for respective pixels. Each value is then used as input into one input neuron.

2.2 General BPN structure

Based on its generalized success in pattern recognition and its parallels to biological holistic face perception, a standard three-layer fully connected feed forward network with backpropagation was implemented as a computer program using an adapted neural network library (CMU, 1994; Masters, 1993). In standard backpropagation learning, training patterns are collected and thought of as ordered vector pairs (a_i, b_i), where each a_i represents an input vector and b_i represents the associated output pattern vector. This procedure is repeated for each layer in the network. Once the training exemplars have propagated throughout the net (feed forward) to produce an output, the error between the actual and desired output is computed, and the network propagates this information in the reverse direction, using the error to adjust the relative weights in the hidden and output layers. (See Haykin, 1999, section 4.3 for additional detail.) The error was computed with the following expression:

$$\frac{1}{2}\sum_i (d_i - o_i)^2, \text{ where } d_i \text{ is the desired output for neuron } i \text{ and } o_i \text{ is the actual output value for that neuron.}$$

A backpropagation net is useful because it does not retain specific memory input patterns. Rather, it continually encodes the characteristics of a given input to produce the desired output. After many such training inputs, backpropagation nets typically develop the ability to generalize their classification decisions to entirely novel exemplars.

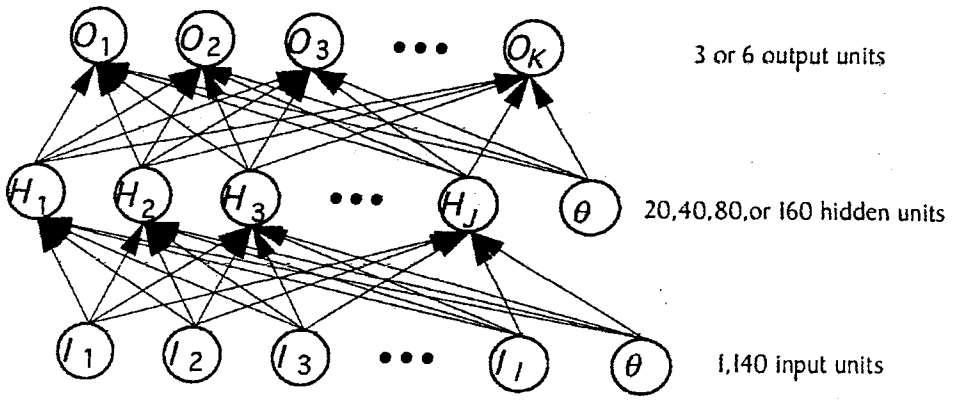


Figure 2: General BPN structure with experiment specific parameters (Skapura, 1995)

2.3 Network structure and Output

In this experiment, the BPN consisted of an input layer with 1,140 neurons (one for each pixel value, 30 x 38 pixels), a hidden layer whose number of neurons varied during different trial networks, and, during the initial experimentation phase, an output layer of 6 neurons. Output was encoded such that each individual output neuron corresponded to one of the six basic emotions. Thus, high activity (0.9) of one output neuron and low activity (0.1) of the remaining neurons signified the classification decision of the network.

Figure 3: In this 6 neuron output layer, the shaded circle indicates an active output neuron, while all other neurons are inactive. This output encoding then signifies which of the six emotional categories was classified.

In order to determine optimum network architecture, the number of neurons in the hidden layer was varied across 3 different trial networks summarized as follows:

6 outputs: 40, 80, and 160 hidden units

The following sigmoidal activation function was used on the hidden and output layers in order to allow differentiation of nonlinear input patterns.

$$Y = \frac{1}{1 + e^{-x}}$$

A set learning rate of 0.3 and a momentum of 0.3 were used in order to maximize the search for local minima in the error surface, so as to elude false absolute minima during training.

2.4 Initial Training

The net was trained using a set of exemplar images (36 images per emotion category). Using six categories, none of the network versions were able to learn the correct recognition of training exemplars.

2.5 Initial Testing and Results

The net was tested after each epoch using five novel test images. The weights were not changed during these test runs, so the next training epoch continued as if these novel images had never been seen. Because the network never sufficiently learned the training exemplars, however, testing at this point was uninformative. The failure of the net to learn the emotional categories left the goal of Experiment 1 unfulfilled, so, as a result, a second experiment was conducted to explore this failure.

3. EXPERIMENT TWO

3.1 Training Set Reduction

One explanation for the failure of the net to learn expression discrimination may have been the result of highly similar categories within the training set. For this reason, we reduced the set to subsets of three and four categories, respectively. These subsets were deemed the easiest to differentiate:

3 category net: disgust, happy, and surprise
4 category net: disgust, happy, anger, and fear

3.2 Image Preprocessing and Revised Network Structure

An attempt was also made to reduce the noise within each individual image. Self-organizing feature maps (SOMs) are a form of neural network known to have desirable properties for feature selection. The nonlinearity of the SOM as well as its sensitivity to the spatial distribution of inputs makes it an ideal tool to process data such as faces. Spatial sensitivity would presumably retain information crucial to a holistic perceptual

strategy. Unfortunately, preprocessing the images using an SOM tool in Matlab was prevented by memory limitations. The large number of inputs, 1140 (due to the 30 x 38 image size), demanded more memory than was obtainable on available machines. Rather than reduce the image size further and risk falling below the minimal resolution for human face identification (Samal, 1991), we decided on an alternative approach. Noise in each image was reduced using Adobe Photoshop filtering to adjust each image to a similar contrast and brightness level. It was hypothesized that this would normalize the distribution of the input data, a heuristic commonly used to improve backprop performance (Haykin, 1999, section 4.6)

In addition to image preprocessing, a second hidden layer was added in order to increase the ability of the hidden feature space to handle nonlinear input data.

3.3 Training and Testing

New nets were trained and tested in the following layer-hidden neuron-output combinations:

One hidden layer: 3 expression categories and 4 expression categories using 20, 40, or 80 hidden neurons

Two hidden layers: 4 and 6 expression categories using hidden layers of 20x10, 40x20, and 80x40.

Thus, the new reduced image sets (3 or 4 categories per set) were used to train and test the network architecture of Experiment One (1 hidden layer), while the new architecture (two hidden layers) was trained on the new 4 category set and the old 6 category training set.

3.4 Results

Networks trained on the reduced training sets showed better overall performance than those trained on the full set. Trained on the three expression image set, the single hidden layer network with a limited number of hidden neurons (20 or 40) converged relatively quickly (see figures 4, 5) and showed very good generalization to novel images (for the 40 hidden neuron net, 93% success on test images). To a lesser extent the single hidden layer network with 80 hidden neurons showed generalization success when trained on the four expression image set (see figures 6, 7). All other networks failed to display satisfactory training performance (>80% correct). (see figure 8 for example results of the unsuccessful 1140x80x40x6 network.)

4. CONCLUSION AND FUTURE DIRECTIONS

The failure of the trial networks to converge with the complete exemplar set (6 categories) is likely the result of a combination of close similarities between certain emotional expressions and insufficient noise reduction in the training images. Network convergence was observed with the reduced training sets that examined easily distinguished expressions according to subjective human opinions. Although fear and surprise were never isolated into their own training set during the current experiment, it is possible that network classifications confused these two expressions. Networks trained relatively well on image sets that included only one of these expressions (i.e., the 3 expression image sets included surprise but not fear, while the 4 expression set included fear but not surprise). This confusion of classification is not surprising considering the similarities between fearful and surprised facial expressions. For example, fear and surprise are both characterized by an open mouth with raised eyebrows and wide eyes. In differentiating the two expressions, a holistic perceptual strategy might recognize the subtle differences between the relative positions of the corners of the mouth (slightly drawn back as in a grimace during a fearful expression) and the eyes (slightly wider in a surprise expression). The recognition of these subtle differences, however, would require high resolution or well-defined input to make use of slight changes in form. Such resolution was greatly compromised in the present study due to the reduction of image size to 30 x 38 pixels. Larger images and successful implementation of an SOM preprocessor might achieve the necessary resolution to allow hidden layers to form appropriate feature mappings.

Based on the success of further attempts to build an automatic emotional expression classifier, it would be interesting to test the net on its ability to recognize multiple emotions in a single face. Often humans express combinations of emotions in a single face, such as a nervous smile, or an angry surprise. A test set of images could be created that combined smiles with fearful or sad eyes or that combined surprise faces with fearful or happy eyes in an attempt to gauge the ability of the net to recognize the existence of more than one emotional category within the same face. Applications such as this would be very beneficial to many lines of research that use such stimuli and that require standardization of subjective opinions about emotional valence.

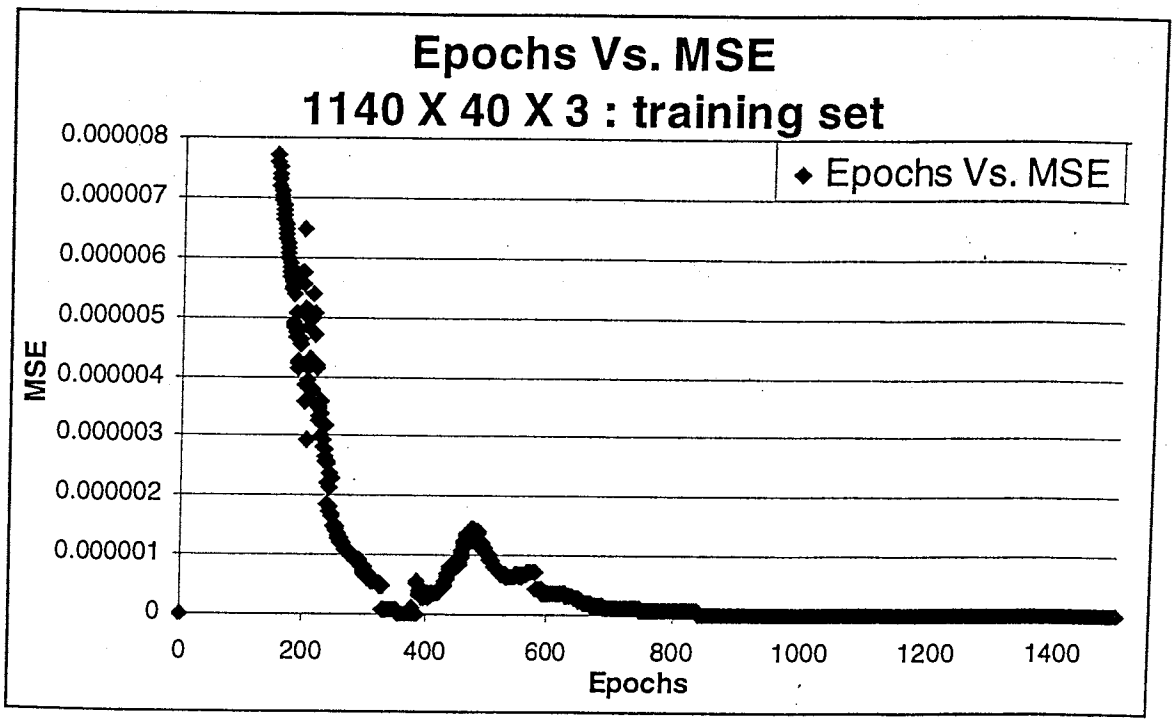


Figure 4 : Neural network classifying three emotional expressions (happy, disgust, surprise) clearly converges at 700 epochs.

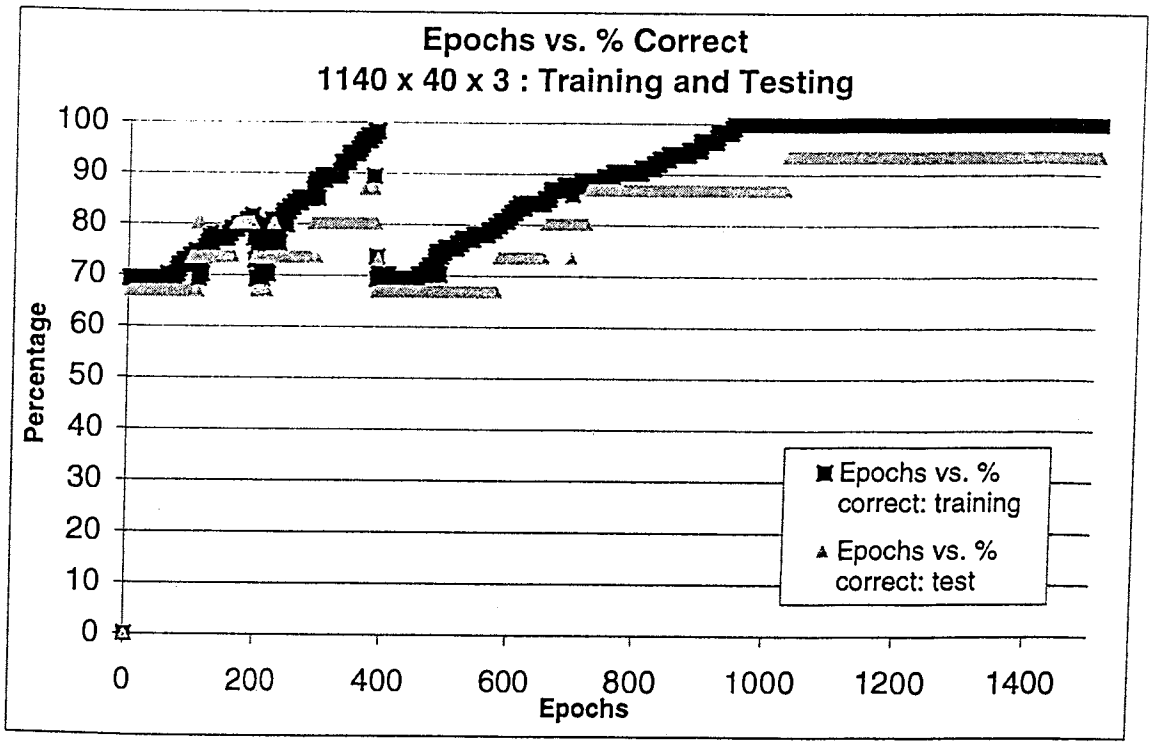


Figure 5: The neural network with 3 outputs learned the training exemplars within 1000 epochs. This network performed very well as it correctly classified over 93% of the test exemplars.

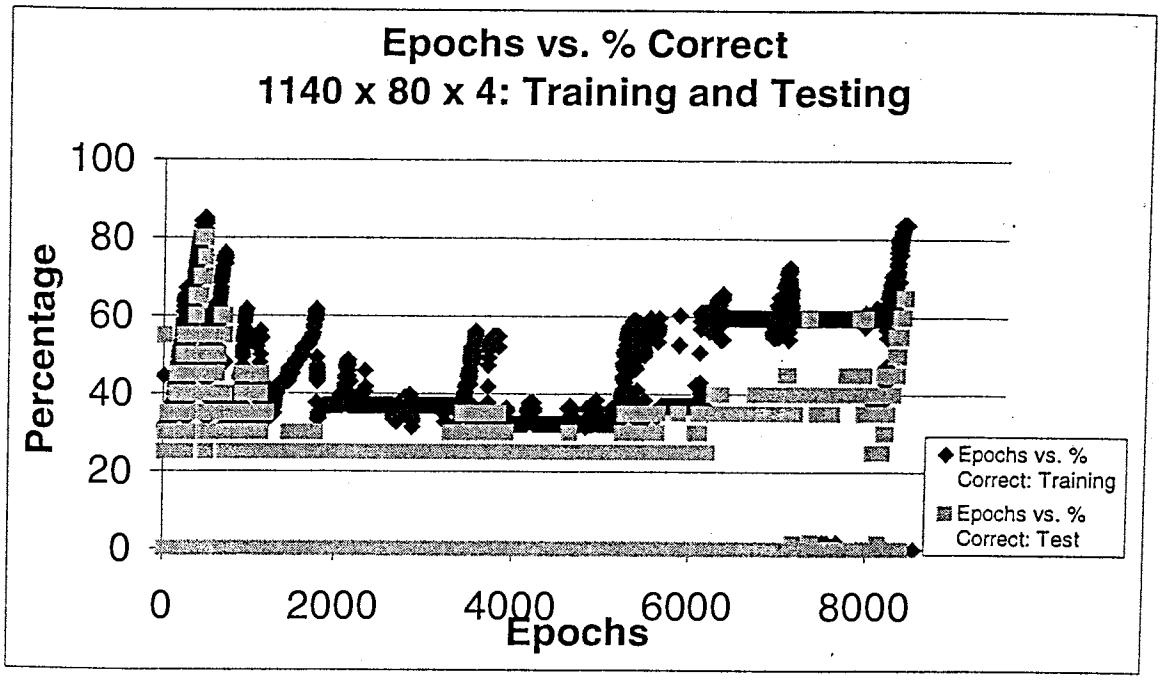


Figure 6: Compared to the network classifying 4 emotional expressions, this network performed slightly worse in correctly classifying both the training and testing exemplars.

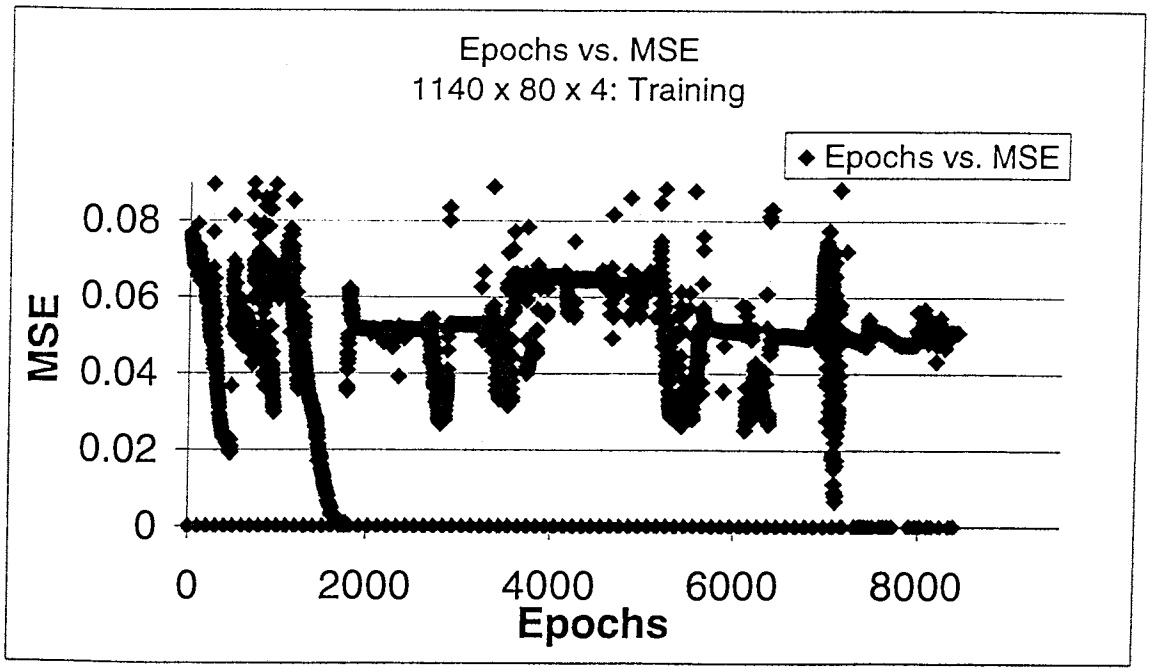


Figure 7: With 4 outputs, the nets failed to converge completely.

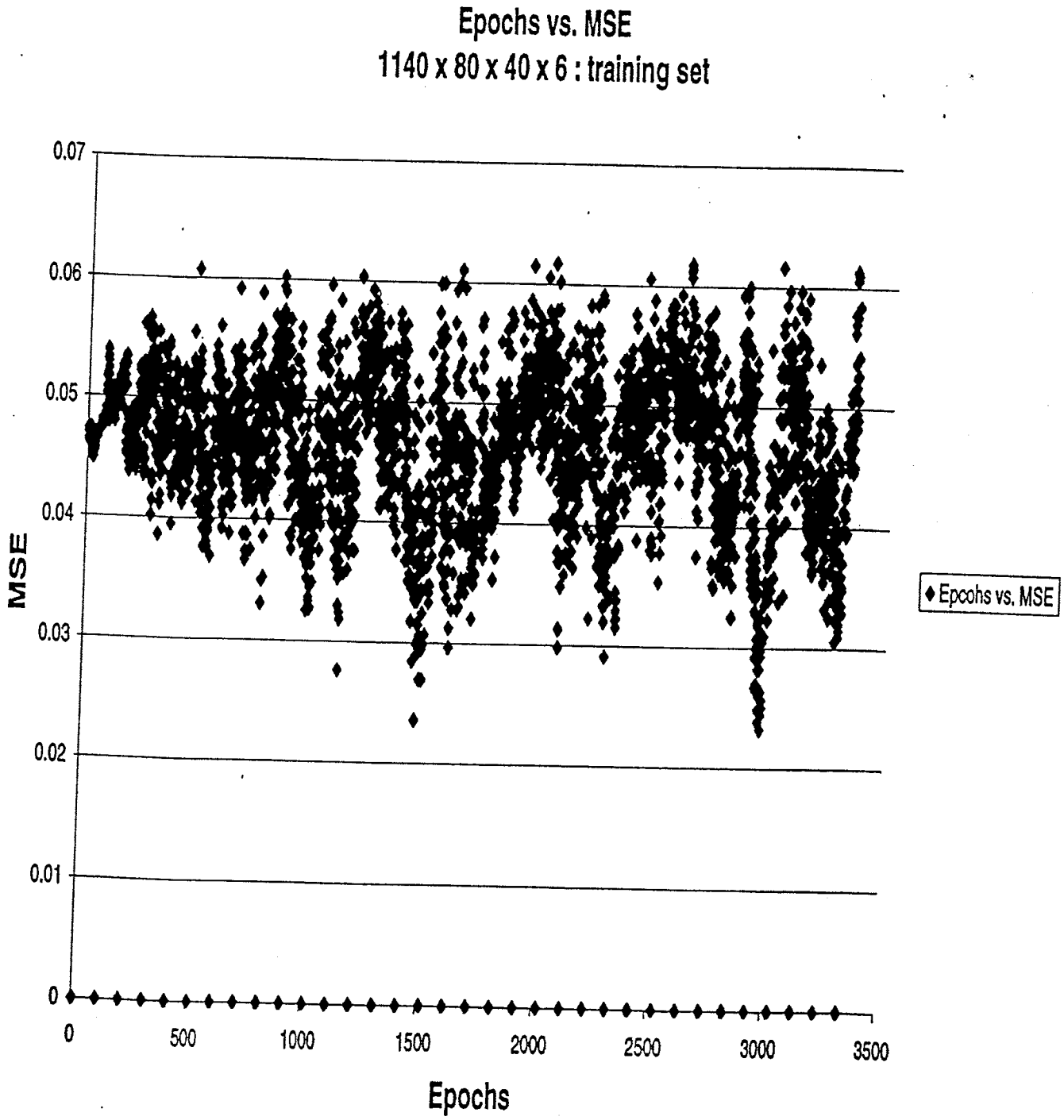


Figure 8: Failure of the a two hidden layer net to converge using the six expression training set.

REFERENCES

CMU. (1994). http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo8/ml-94faces/code

Cottrell, G.,W. and Metcalfe, J. (1991) EMPATH: Face, emotion, and gender recognition using holons. *Advances in Neural Information Processing Systems*, **3**, 564-571.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation.* New Jersey: Prentice Hall.

Kuskowski, M.A. and Pardo, J.V. (1999). The role of the fusiform gyrus in successful encoding of face stimuli. *Neuroimage*, **9**, 599-610.

Masters, T. (1993). Practical Neural Network Recipes in C++. New York: Academic Press, Inc.

Matsumoto, D. (1998). Culture and emotion. In Adlet, L.L. et. al. (Eds.) Cross-cultural topics in psychology. (pp. 115-124). Westport, CT, USA: Praeger Publishers / Greenwood Publishing Group, Inc.

Pasley, B.N. (2000). Automatic emotional processing: Evidence for a subcortical visual pathway. Senior Thesis, Yale University.

Samal, A. (1991). Minimum resolution for human face detection and identification. In *Proceedings of CVPR*, 123-129.

Skapura, D.M. (1995). Building Neural Networks. New York: ACM Press.

White, M. Representation of facial expressions of emotion. (1999). *American Journal of Psychology*, **112**, 371-81.

Musical Analysis and Prediction with Dynamically Driven Network Architectures

Brian Carp
Yale University , Department of Computer Science
New Haven, CT 06512

Abstract
We attempt to combine the benefits of dynamically driven recurrent networks and tapped delay line memory to achieve probabilistic prediction of musical lines with minimal training. Synaptic weights are modified using gradient descent and momentum to minimize the error in predicting time sequences. The trained network outputs its prediction of the next note in the sequence. Results indicate an accuracy of prediction within one to two notes on average, after at least twenty epochs of training with a small data set.

1. INTRODUCTION

1.1. Scope of the Project

The notion of taking a systematic approach to musical analysis and composition has been around for centuries. However it is generally assumed that there is a certain amount of creativity and unpredictability involved in music, and that the listening experience is in many ways characterized by the balance between anticipation and surprise. This project demonstrates, on a simple level, that patterns in music can in fact be generalized, and that a neural network can be trained to recognize a pattern and predict the movement of a simple melody.

Since the range of musical examples is so large, the scope of this project is limited to the tonal framework of sixteenth-century counterpoint. Species counterpoint, which is based primarily on the work of J. J. Fux, plays a central role in Western art music and introduces many of the fundamental concepts of the popular song. It is considered to be one of the most simplistic approaches to musical analysis and composition and by no means subsumes all tonal music. Rather, it is a scaled-down approach to analysis that represents a reduction of complex musical structure to simple melodies and harmonies. The study of counterpoint is above all the study of voice leading, and is more or less restricted to the domain of melodic motion.

This project uses of the simplified melodic lines of species counterpoint, called the *cantus firmus*, and trains a dynamically driven recurrent network to predict certain notes that fit basic patterns in an unfamiliar melody.

---

brian.carp@yale.edu

1.2. Design Concepts

Music to large extent is built on the relationships between notes, both on a local scale (notes in succession) and on a global one (the "shape" of the melodic line). Music theorists generally believe that what makes a note sound wrong or out of place to the listener is it's incompatibility with the notes that precede it or the notes that define global characteristics of a melody to which other notes are expected to conform. Abstracted away, these constraints can be used theoretically to predict which note should follow given a sequence of notes that precede it. Therefore, we need a network architecture with some form of memory that would allow it to represent its previous inputs and use those memories in a computation that predicts the next note in a sequence.

Recurrent networks that respond temporally to an externally applied input signal have feedback loops which enable the network to maintain representations of their state, which make them ideal devices for nonlinear prediction. The recurrent architecture provides the beneficial effects of global feedback which, when combined with the local feedback of the tapped input delay line, provides a suitable framework for learning patterns and relationships in a musical line.

2. NETWORK DESIGN

2.1. Architecture

A network devoted to the supervised learning of the rules of counterpoint must be able to generalize patterns on both a local and global scale, and to map similar inputs to a range of outputs with regard to these contextual influences. There are two standard tools which lend themselves to such a task, namely temporal processing units and recurrent network architectures.

2.1.1 Recurrent Network Architecture

The basic model therefore is a simple recurrent network, also known as an Elman network. An Elman network includes recurrent connections from a layer of hidden neurons back to the input layer via an array of unit delays (Fig. 2.1). The hidden neurons therefore define the state of the network and maintain some record of its activity, which allows the network to learn global patterns that extend over a long period of time (Haykin, 1999).

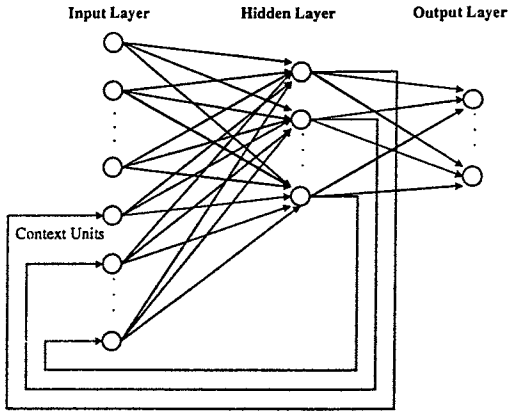


Figure 2.1 The structure of the Elman network

2.1.2 Temporal Processing Units

Alternatively, the use of time delays on the inputs to a feedforward network allows for the analysis of local temporal relationships. Tapped delay lines, the most common form of "short-term memory" in network architecture, consists of depth D unit delay operators which delay inputs for zero to D discrete time units (Fig. 2.2). This allows the network to make a computation based on current as well as relatively recent inputs and thus internally characterize external temporal relationships.

2.2. Structure of the Neural Network

The network is composed of 3 layers: an input layer (I), a hidden layer (H), and an output layer (O). There are recurrent connections from each of the neurons in the hidden layer to the input layer through context units. All other connections are feedforward. The output layer has one neuron which outputs the predicted value of the next note in the sequence. The input layer consists of the sequence input (single value), the recurrent connections from the hidden layer, and the delayed inputs coming off of a tapped delay line. The tapped input delays range from zero to three in number, which corresponds to a maximum short-term memory of three notes.

The network was tested separately with hidden layers consisting of 5 and 10 neurons. Neurons in the hidden layer were given hyperbolic tangent activation functions, while the output neuron uses a pure linear function. The network learns using the standard back-propagation method of gradient descent and using the generalized delta rule:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

where $\Delta w_{ji}(n)$ represents the change in the synaptic weight from neuron i to neuron j at time step n, η is the learning rate parameter, $\delta_j(n)$ is the local gradient, y_i is the input signal to neuron j, and α is the momentum constant.

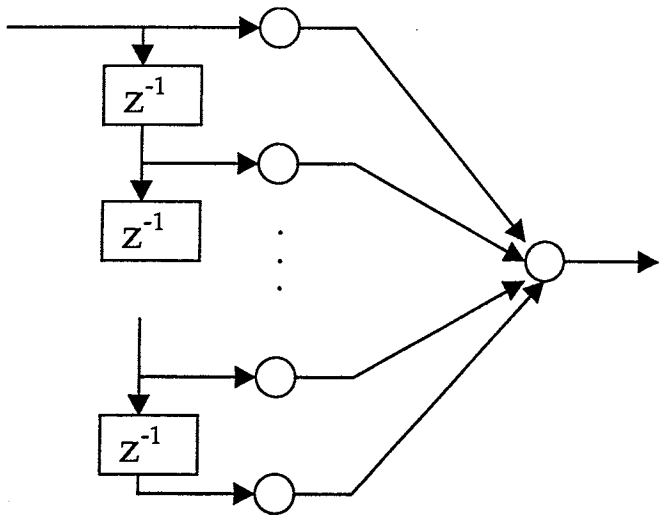


Figure 2.2 Ordinary tapped delay line memory

3. MELODIC FORM

3.1. Simple Melodies: Cantus Firmi

3.1.1 Explanation of Counterpoint Theory

The study of counterpoint is the study of voice leading, which involves the relationships between two or more melodic lines such that they create meaningful harmonic motion. All of species counterpoint is built around a single melodic line, which is called the *cantus firmus*. The principles governing the cantus firmi are fundamental to the study of counterpoint and to our ability to make any meaningful generalizations about tonal music.

Obviously some form of "music" can be generated by constructing a sequence of random notes; however we are concerned with meaningful note progressions and melodies. Of course "meaningful" is a subjective idea, but the theory of counterpoint explicitly defines which progressions and harmonies are to be considered. Specifically, notions of *consonance* and *dissonance* are based on what sounds "stable" or "pleasing" to the ear, and are partly based on the physics of sound and overtones. They are practical as well: singers have much more difficulty leaping up or down dissonant intervals, or singing a dissonant harmony, because they often have to internalize a note ("hear it in their head") before they may sing it aloud

Thus, melodies are limited by constraints that include not outlining dissonant intervals, compensating large leaps with step-wise motion in the opposite direction, etc., most of which are based on what sounds pleasing to the average listener as well as the preferences and limitations of singers. There are other principles that restrict the size, general shape, continuity, and balance of cantus firmi, along with rules about the use of leaps, use of repetitions, beginnings and endings, and unresolved melodic tension (Salzer and Schachter, 1989).

3.1.2 Normalization of Input Data

Here is an example of a cantus firmus, taken from Salzer and Schachter's *Counterpoint in Composition*, pg. 11, and transposed into the key of C:

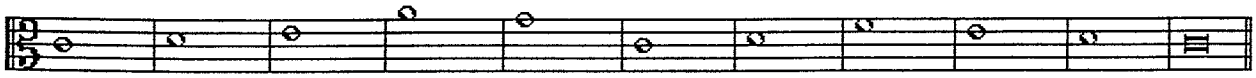


Figure 3.1 Example of a typical cantus firmus

In the interests of simplicity all of the examples are set in the same key (C Major). Without loss of generality we can transpose the all of the melodies into the same key since the relationships between the notes will remain the same. The goal is for the network to predict notes based on these relationships, not based on absolute pitch. Thus we simplify the problem by normalizing the input data. All cantus firmi begin and end on the tonic, which in this case is C.

3.2. Numeric Representation

40 cantus firmi were selected to make up the complete set of tonal melodies that the network would be expected to handle. Of these 10 were selected at random to form a training set for early experimentation.

The notes of the cantus are represented numerically as an offset from the tonic note, middle C. Therefore, 'C' translates to a value of 0, 'D' to a value of 1, and so on; an octave above middle C would have a value of 8. For the sake of computation it is preferred to have values between −1 and 1, so each note value is reduced by a factor of 10. Thus, the cantus firmus above would be represented by the following sequence:

```
0.0   0.1   0.2   0.5   0.4   0.0   0.1   0.3   0.2   0.1   0.0
```

The full set of coded sequence is included in Appendix A. The example above is listed under index eight in the Appendix.

4. TRAINING

4.1. Elman Network

An Elman network was created in MATLAB to simulate this network, initially with five hidden neurons. Weights and biases were initialized according to the Nguyen-Widrow initialization algorithm by default. The network was trained with a set of ten cantus firmi chosen from the data set. The mean-squared error was calculated after each epoch of the training sequence. Results indicated that on average, the network began to level off after about 25 epochs at a mean-squared error between 0.04 and 0.05, which indicates that the prediction was typically of by full tenths (i.e. not values).

Increasing the number of hidden neurons to 10 only improved performance slightly, if at all. Network performance (as revealed by averaged mean-squared error values) would typically level off after 25 to 50 epochs, but approach different values of the average mean-squared error ranging from 0.02 to 0.05. Since these values are squares, they represent differences in the range of 0.15 to 0.2, which according to our encoding indicates predictions that are off by 1.5 to 2 notes, on average.

However, the predictions of the Elman network were substantially better than random guessing. A process that randomly picked values between −0.5 and 0.5 (the range of the data set) had an average mean-squared error close to 0.14 (Figure 4.1).

4.2. Heuristic Function

A simple observation of the data set is that melodic progressions tend to start at the tonic (0.0) and approach a fifth above (0.5) before returning back to the tonic, mostly in step-wise fashion. We can therefore design a simple heuristic estimation that follows a perfect arch and compare it's predictability to that of the Elman network. The results indicate that the heuristic performs with a mean-squared error of 0.438. Experiments with the 10-neuron Elman network show that the network can outperform simple heuristics designed to predict the melodic line, as indicated by the graph in Figure 4.2.

Furthermore, the histogram of the error vectors in Figure 4.3 indicates that the network predicts notes with about the same error as the heuristic, though often does better. Random guessing produces a fairly well-distributed error, often in the range of 0.1 or higher. This means that random guessing on average will be off by three notes or more.

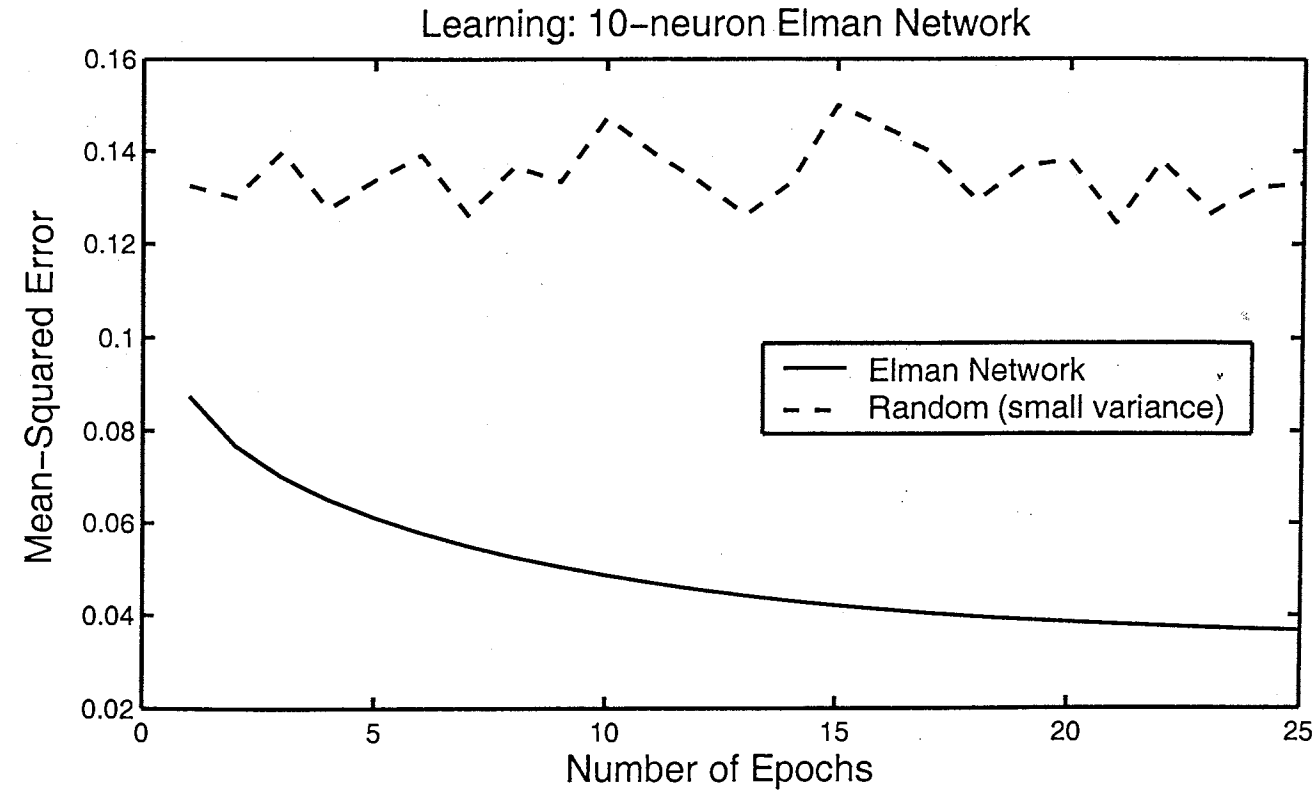


Figure 4.1 The neural network does much better than random guessing, and gradually improves its performance with increased training.

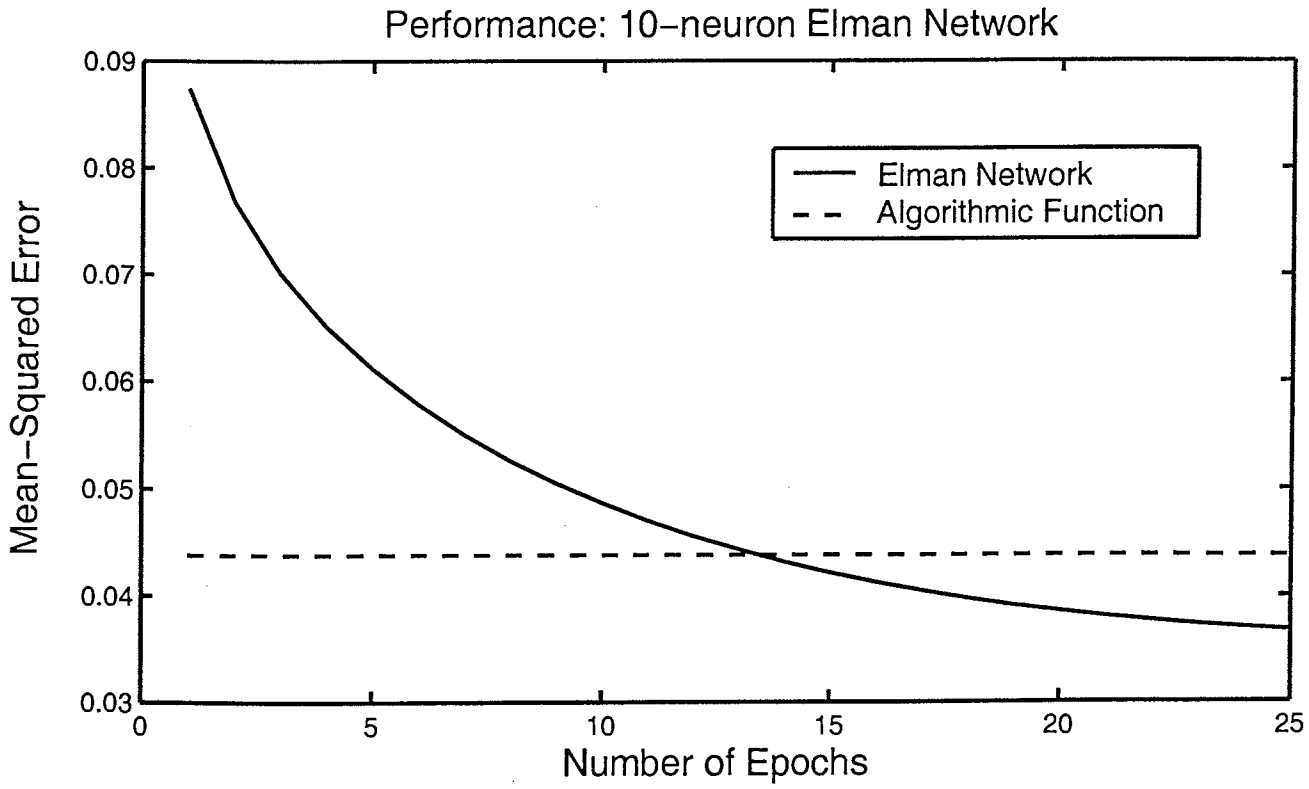


Figure 4.1 The Elman network's performance plotted against that of the heuristic function.

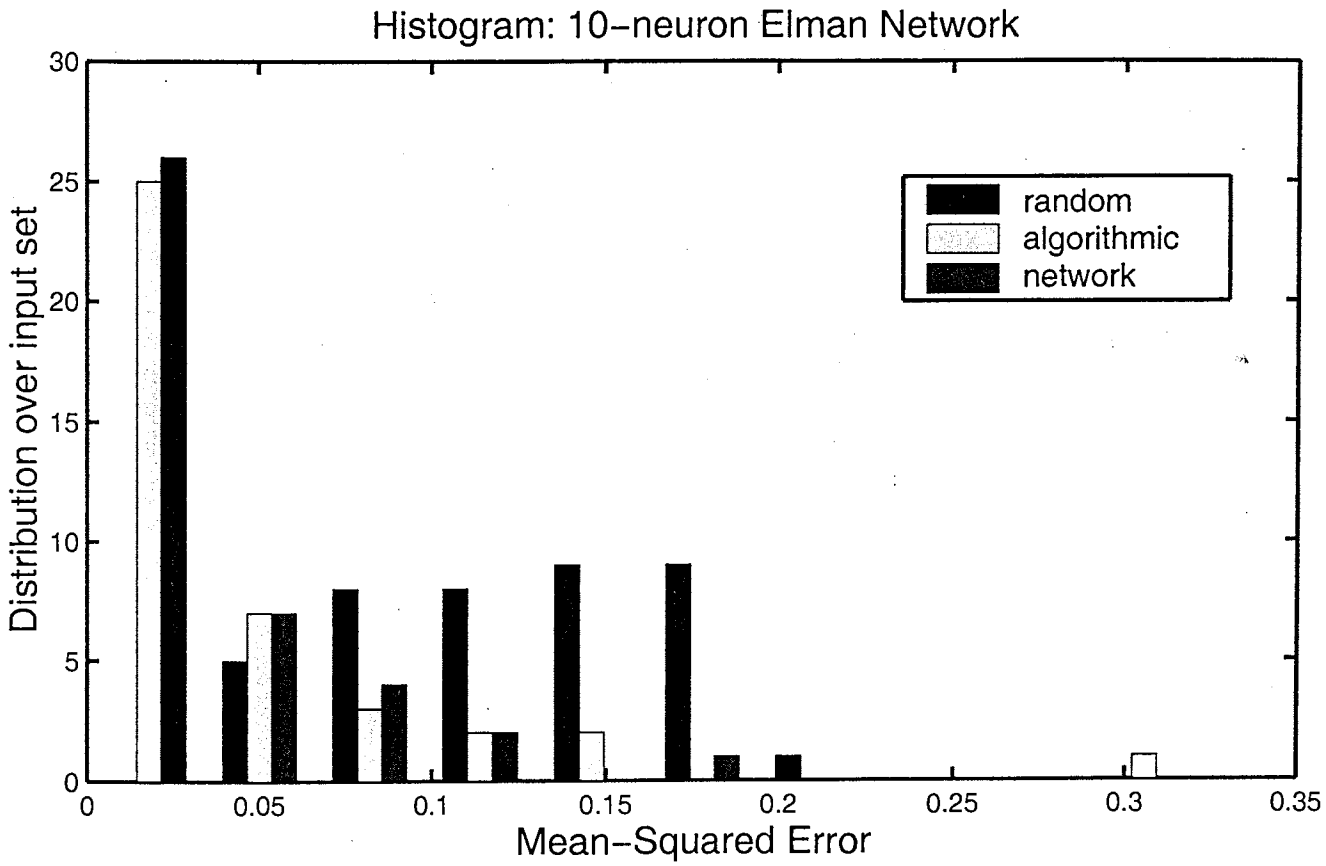


Figure 4.3 Histogram of mean-squared error values across the data set.

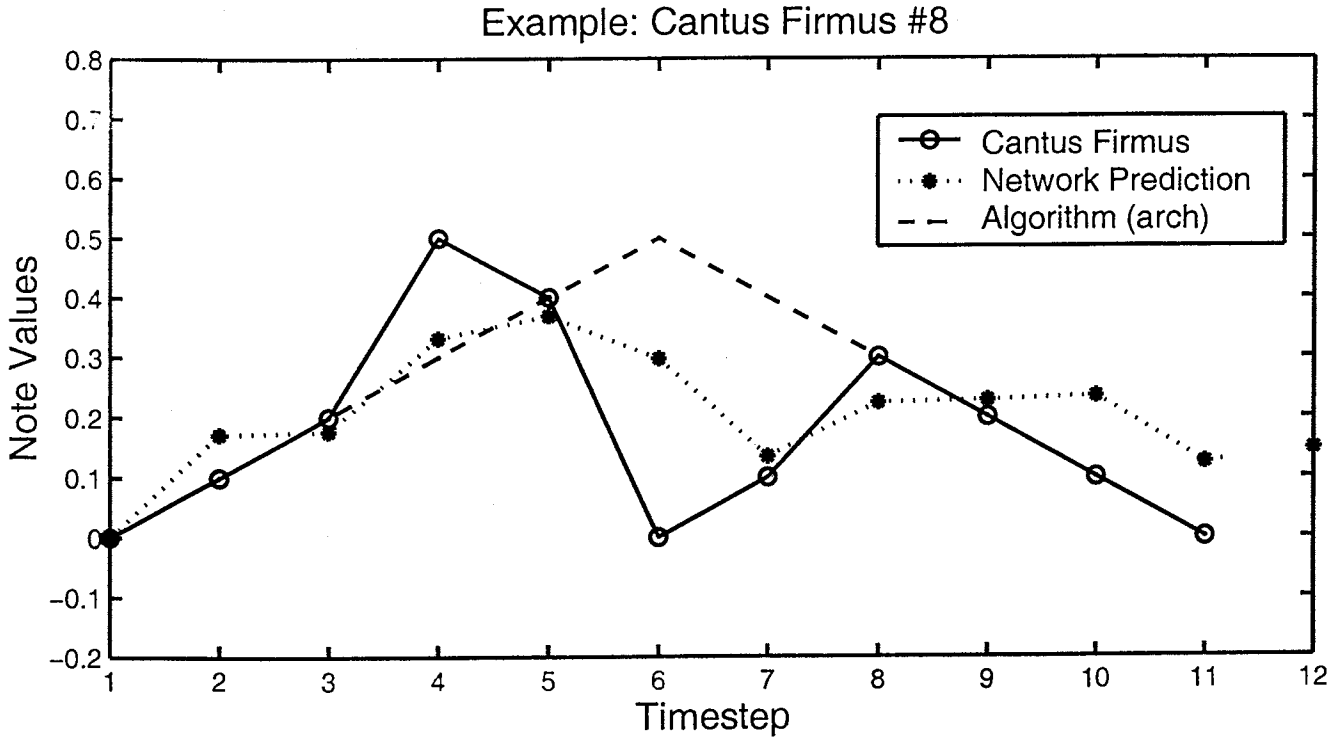


Figure 4.4 The output of the network is plotted against its input vector. While the network fails to predict the closing of the musical line, it appears to have learned to compensate for leap s in the melodic passage.

4.3. Analysis

The averaged error values are mostly useful for describing the overall performance of the network, but give no indication of what the network may or may not have been able to learn. If we plot the melodic line of cantus eight (the example from above) against the predictions of the network and the heuristic, we can deduce some information about what the network has been trained to do.

Figure 4.4 shows the predictions plotted against the cantus. It is apparent that the network has not learned about endings; in all of the examples the cantus returns to the original note (value 0.0) after about ten timesteps, but the network has failed to learn this trend. On the other hand, the network makes fairly accurate predictions each time the cantus leaps. In all of the cantus firmi a large leap in the melody will be compensated by step-wise motion in the opposite direction. This is a local relationship which the Elman network seems to have learned, as indicated by timesteps 4, 7 and 9. Timestep 7 in this example is particularly telling: it appears that the Elman network makes an "intelligent" prediction in direct contrast to that of the heuristic. (See Figure 4.4).

5. NETWORK MODIFICATIONS

The Elman networks were modified to include tapped delay lines on the inputs, which could possibly improve performance by giving the networks more direct information about their previous inputs.

Training of both the 5-neuron and 10-neuron networks indicate improved performance with a tapped input delay of depth one, and further improvements with tapped input delay of depth two. Depth-three tapped delay lines seemed to hinder the performance of the networks, and statistically did worse during simulations. Figure 5.1 and 5.2 show the performance curves of the networks plotted against the additions of tapped delay lines.

6. CONCLUSION

The results of the demonstration indicate that Elman networks can in fact be trained to recognize patterns in melodic sequences and often make accurate predictions about the next note in a simple melody. The general performance, measured by average mean-squared error, reaches a stable point at which the network is making predictions that are off by only one or two notes in the average case. The performance graphs show that the neural network will make better predictions than random guessing, and often better than simple heuristic predictions, after seeing only a fraction of the data space. Closer analysis reveals that an increased number of neurons in the hidden layer do not specifically improve network performance; if we compare Figures 5.1 and 5.2 we see that the 5-neuron network with tapped input delays achieves a much lower mean-squared error than the 10-neuron network with the same modifications. In general, the use of one or two input delays increased the accuracy of predictions on average, though the use of excessively long delay lines seemed to decrease it.

Further testing using larger data sets, more layers, and longer training times may reveal more precise limits on the ability of a neural network to predict musical lines. The project could also be extended to make predictions about sequences with the addition of harmonic complexity, although that is outside the scope of this particular investigation.

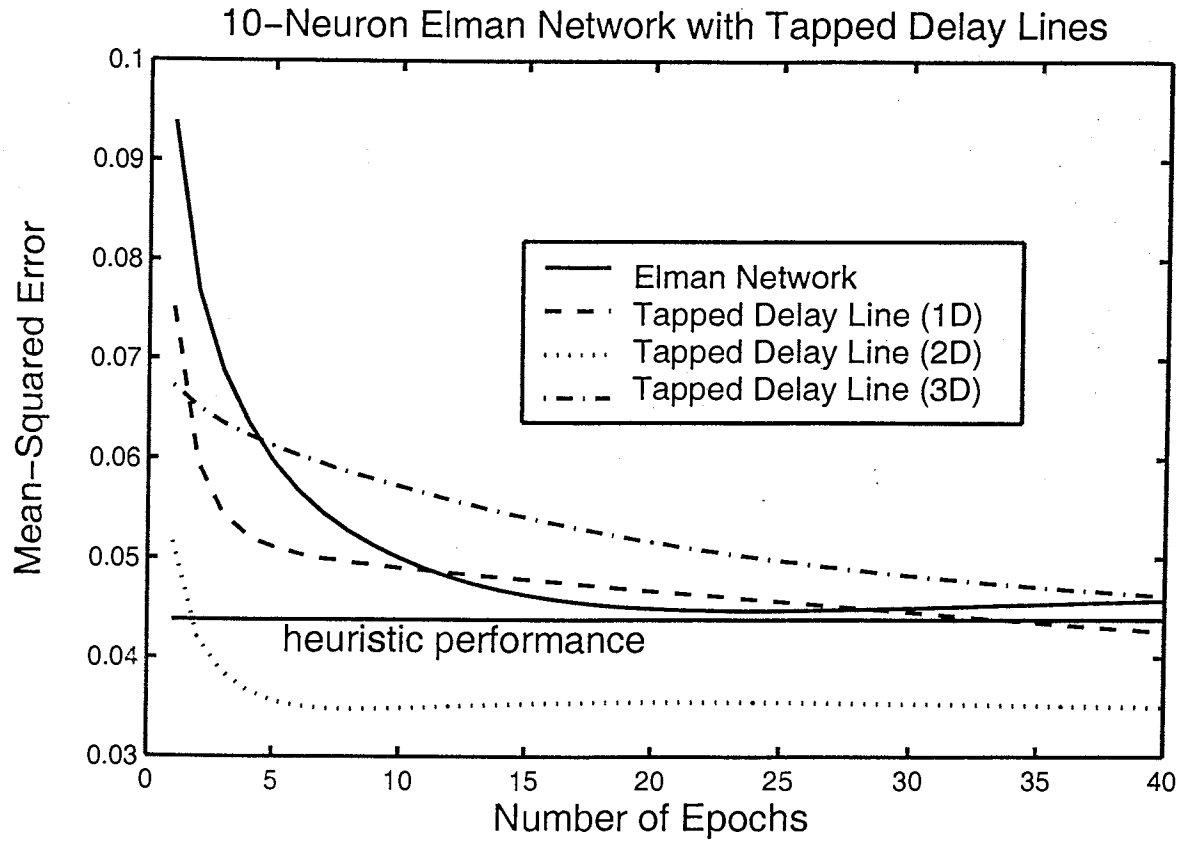


Figure 5.1 The effect of adding tapped delay lines to the 5-neuron network. **The horizontal line representing the heuristic is merely a constant for comparison.**

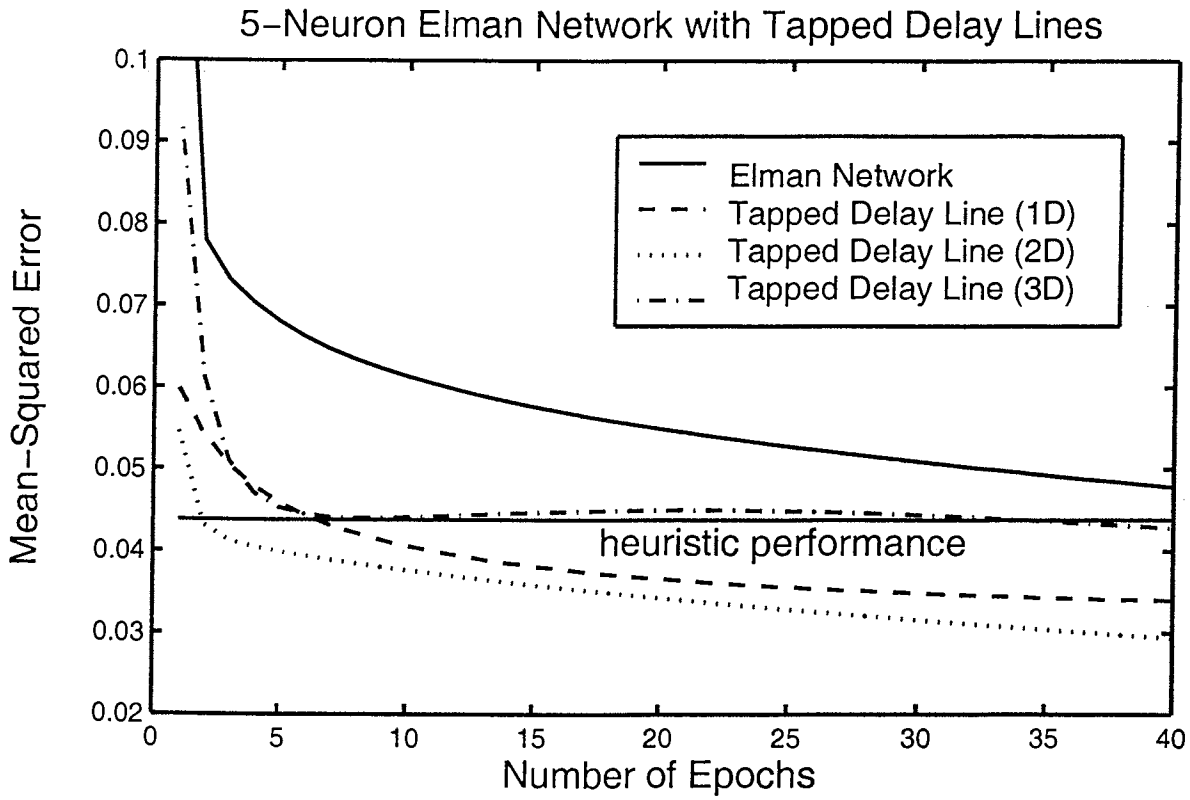


Figure 5.2 The effect of adding tapped delay lines to the 10-neuron network.

There are two formidable sources of error in this experiment. The first is the limited range of input presented to the network. The fact that the network on average learned to predict notes without being off by more than one or two is not necessarily impressive given a five to ten note range. The second is that is it difficult to determine the extent to which the network is simply memorizing sequences rather than finding real patterns. Increasing the size and range of the data set could reduce both of these sources of error.

Another subtle point is that the network may have just learned to average values rather than find real patterns. The predictions were often in the middle range, which statistically would reduce the error as much as possible if no real pattern-matching were performed. Indeed, when the network was given a starting note and then fed its own predictions, it produced a sequence of notes that fluctuated in the middle range, rather than produce any meaningful composition. Of course we would not expect a network of this scale to perform a high-level task such as musical composition, so this thought process merely reaffirms the fact that our expectations cannot be too high. Given these standards, we may conclude that the network does an impressive job at predicting melodic lines.

Moreover, in studying the subtleties of Figure 4.4, we may conclude that the neural network has learned an important rule of music strictly by example. Plots of other input vectors confirm this; on a given leap in the input the network almost always predicts compensatory step-wise motion. Thus, without being given any explicit information the network successfully discovered an important rule of melodic motion.

REFERENCES

1. Haykin, S. (1999). *Neural Networks: a Comprehensive Foundation*, 2nd ed. New Jersey: Prentice Hall, Inc.
2. Skapura, D. M. (1996). *Building Neural Networks*. New York: ACM Press.
3. Salzer, F. & Schachter, C. (1989). *Counterpoint in Composition*. Columbia University Press.
4. Jeppesen, K. (1992). *Counterpoint: the Polyphonic Vocal Style of the Sixteenth Century*. New York: Dover Publications, Inc.

APPENDIX A – ENCODED CANTUS FIRMI

* Cantus firmi transposed from examples in Salzer and Schachter, 1989; Jeppesen, 1992; and some original compositions.

(1) 0.0 0.1 0.3 0.2 0.3 0.4 0.5 0.4 0.2 0.1 0.0
(2) 0.0 0.1 0.2 0.3 0.4 0.1 0.3 0.2 0.1 0.0
(3) 0.0 0.1 0.3 0.2 0.5 0.4 0.3 0.1 0.2 0.1 0.0
(4) 0.0 0.3 0.2 0.5 0.4 0.2 0.3 0.2 0.1 0.0
(5) 0.0 0.2 0.1 0.0 0.3 0.2 0.4 0.3 0.2 0.1 0.0
(6) 0.0 0.1 0.2 0.0 -0.2 -0.1 0.0 0.4 0.2 0.0 0.1 0.0
(7) 0.0 0.4 0.3 0.5 0.4 0.2 0.3 0.2 0.1 0.0
(8) 0.0 0.1 0.2 0.5 0.4 0.0 0.1 0.3 0.2 0.1 0.0
(9) 0.0 0.2 0.1 0.2 0.3 0.4 0.2 0.1 0.0
(10) 0.0 0.2 0.1 0.5 0.4 0.2 0.3 0.2 0.1 0.0
(11) 0.0 -0.3 -0.1 -0.2 -0.3 0.2 0.1 0.3 0.2 0.1 0.0
(12) 0.0 0.4 0.3 0.1 0.2 0.3 0.2 0.1 0.0
(13) 0.0 0.1 0.2 0.3 0.4 0.2 0.3 0.1 0.0
(14) 0.0 0.2 0.3 0.1 0.2 0.5 0.3 0.4 0.2 0.1 0.0
(15) 0.0 0.4 0.3 0.2 0.1 0.0 0.2 0.1 0.0
(16) 0.0 -0.3 -0.2 -0.3 0.4 0.3 0.1 0.2 0.1 0.0
(17) 0.0 -0.1 -0.2 0.1 0.0 0.3 0.2 0.1 0.0
(18) 0.0 -0.3 -0.2 -0.1 0.0 0.1 -0.1 0.2 0.1 0.0
(19) 0.0 0.1 0.3 0.2 0.4 0.3 0.2 0.1 0.0
(20) 0.0 0.1 0.5 0.4 0.3 0.1 0.2 -0.1 0.0 0.1 0.0
(21) 0.0 0.3 0.2 0.1 0.2 0.4 0.3 0.0 0.2 0.1 0.0
(22) 0.0 0.1 0.4 0.3 0.2 0.3 0.5 0.4 0.1 0.2 0.1 0.0
(23) 0.0 0.1 0.0 0.4 0.3 0.1 0.2 0.1 0.0
(24) 0.0 -0.1 0.0 0.1 0.2 0.3 -0.1 0.1 0.0
(25) 0.0 -0.4 -0.3 -0.2 -0.1 0.0 0.1 -0.1 0.2 0.1 0.0
(26) 0.0 -0.2 -0.1 0.0 0.1 0.0 0.3 0.2 0.1 0.0
(27) 0.0 -0.4 -0.3 -0.2 -0.1 -0.2 0.2 0.1 0.0
(28) 0.0 0.4 0.2 0.5 0.4 0.3 0.2 0.1 0.0
(29) 0.0 0.5 0.4 0.2 0.3 0.4 0.2 0.3 0.2 0.1 0.0
(30) 0.0 0.1 -0.1 0.0 0.1 0.0 0.3 0.2 0.1 0.0
(31) 0.0 0.2 0.3 0.4 0.2 0.5 0.4 0.2 0.3 0.2 0.1 0.0
(32) 0.0 0.2 0.1 0.2 0.3 0.4 0.5 0.1 0.2 0.1 0.0
(33) 0.0 -0.3 -0.2 -0.1 0.2 0.1 0.2 0.3 0.1 0.0
(34) 0.0 -0.4 -0.3 -0.2 -0.1 0.2 0.1 -0.1 0.0
(35) 0.0 -0.2 -0.1 -0.2 0.4 0.3 0.1 0.2 0.1 0.0
(36) 0.0 -0.3 -0.2 -0.3 -0.5 -0.4 -0.3 -0.2 -0.3 -0.2 -0.1 0.0
(37) 0.0 0.2 0.1 0.5 0.4 0.2 0.3 0.4 0.1 0.2 0.1 0.0
(38) 0.0 0.3 0.2 0.1 0.4 0.3 0.4 0.5 0.3 0.2 0.1 0.0
(39) 0.0 0.3 0.2 0.1 0.4 0.3 0.1 0.2 0.1 0.0
(40) 0.0 0.2 0.1 0.0 0.5 0.4 0.2 0.3 0.2 0.1 0.0

Musical Key Identification by a Neural Network

Daniel Dormont
Department of Computer Science, Yale University
New Haven, CT 06520
May 5, 2000

Abstract

A musician with fairly simple training has no trouble identifying the key of most pieces of music. Even without a printed score with a key signature, there are reasonably easy ways to make this determination. However, this human intuition cannot easily be transferred to a deterministic computer algorithm. Our objective is to train a multi-layer feed-forward neural network with a Hebbian-style learning rule to learn to understand the subtleties that could not be found in a more straightforward algorithm and produce more accurate results. Unfortunately, the network did not perform as expected, and the end of this paper attempts to provide some explanation for this situation.

1. Introduction

The key of a piece of music is generally determined by the underlying sequence of chords that follows underneath the melody. In many cases, the key of the piece is best identified by examining its final chord. However, this approach is by no means complete. First, from the "piccardy third" technique of the Baroque period onwards, the composer would often intentionally modulate to a different key on or just before the final chord. Variations of this sort became increasingly common over the centuries. Second, short melodic fragments (for example, ten notes long such as the ones fed to the network in this project) are generally in a key of some sort, and it should be possible to dissect this key from such a short fragment, even if the chords are not actually provided. Although this problem is not one that requires a computer to solve, and therefore no attempts, to the best of my knowledge, have been made to apply a neural network to this particular problem, it is my objective that this could provide insight into the potential for applying neural networks to the study of music, which is one of the less well-understood areas of human cognition.

2. Architecture of the Network

2.1 Representation of the inputs and outputs:

For this network, I chose the most obvious method of representing notes: a vector of twelve values (one for each of the twelve pitches in Western music), with a value of one or zero specifying whether the pitch was sounding at that time. The output is a similar vector: all values should be approximately zero except for the one specifying the chosen key.

2.2 Construction of the network:

This network is fed vectors of input data one note (or chord) at a time. It is a feed-forward network with a number of layers equal to the number of notes it is fed, which for the moment is ten. Each of these layers contains twelve neurons. Each layer is fully-connected to the next, so all neurons receive all outputs from the preceding layer. All of these synapses are excitatory. In addition, each layer is fully laterally connected among its neurons. These synapses are inhibitory. The intention of this design is that at each layer, the neurons representing the notes that were playing at that time would produce stronger output, shutting out noise that might be coming in from other notes in the previous layer.

2.3 Activation:

Each neuron adds in all of its weights, both from the feed-forward and lateral synapses, and passes them through the standard sigmoid logistical function. This function was chosen because it limits the value of the output of the neuron, while being monotonically increasing so that results more consistent with expected progressions as established by the weights will produce stronger outputs.

3. Training

3.1 Learning rule:

We use a modified Hebbian learning rule. That is:

$w_{ij}(n+1) = w_{ij}(n) + \text{eta} * x_j(n) * y_l(n)$. This rule produces two very useful results:

1. The general theory behind this network is that the synapses between the layers should encode sequences of notes that fit more or less well together into patterns so that sequences that make sense are reinforced and those that do not are weakened. Then,

experimental data should fit into the patterns modeled after pieces of training data that were in the same key, so the network should produce the same result.

2. We also want to make sure that in the output for each layer, we reinforce only the correct pitches. In other words, the more musically distant a pitch is from a particular chord, the harder it should be for a layer which is transmitting that chord to produce that pitch. To this end, the application of the Hebbian rule will increase the strength of inhibitory synapses between more dissonant pitches within each layer, and decrease the strength of the synapses between more consonant pitches.

3.2 Training procedure:

The melodies are fed in to the first layer one note (or chord) at a time, and the outputs for its neurons are calculated. The synapses leading into the first layer, as well as the lateral synapses within it, are then adjusted according to the Hebbian rule as described above. This is the first epoch. As each subsequent set of inputs is presented, the outputs are calculated one layer ahead, and the weights are adjusted for all layers which have been reached so far. When the end of the melody is reached, we see the result from the output layer which should now be the correct key. The training set should contain examples from all twelve keys in different modes (major, minor, etc.) although experimentation may show that this network is, in fact, only suited for work in one particular mode.

4. Experiments

It was my intention that I my program be able to translate MIDI sequencer input into inputs to my network as described above. These sequences would of course be short, probably snippets of recordings of actual pieces. I would train the network using very straightforward examples such as Bach harmonized chorales. I would then input more complex examples, hoping that the nodes corresponding to the most prevalent key would show strong output, and that secondary possibilities would show weaker output. Unfortunately, though I attempted several approaches, I did not develop a good way of feeding in MIDI data. This required me to code input sequences by hand, which

prevented me from inputting non-trivial examples and hence obtaining an interesting result.

5. Conclusions

In my fairly extensive searches of the Internet, I have only discovered one published paper by a computer scientist using a neural network model for a musical application. Professor Michael Mozer of the University of Colorado has developed a neural network for automated music composition called CONCERT. He fed it a wide range of traditional pieces with the hope that it would learn to generate melodies and harmonies of a similar style. He used a recurrent network trained on a variation of the back-propagation network which, given a melodic sequence, was supposed to predict the next note in the sequence. His attempt was not successful. Although in very simple cases, Prof. Mozer was able to coax CONCERT into producing output that he considered reasonable, it failed when he tried to train it to understand broader compositional structure such as that of a Bach chorale. Mozer blames this result on the failure of the back-propagation algorithm to produce a network which can encompass both fine-grained and coarse-grained patterns, and proposes some adjustments that he imagines will improve the situation, though he confesses to not having pursued the issue. In his online description of the paper, he concludes: "overall results cast doubt on the promise of note-by-note prediction for composition."

Sadly, I suspect that a similar fate may be in store for many artificial intelligence-oriented approaches to processing music. A former CS 477 student attempted to create a neural network that could compose Jazz improvisations, using an approach somewhat similar to Mozer's. He met a similar fate. I have discerned from conversations with biologists that very little is understood about how the human brain processes music. It would be out of line to conclude that present algorithmic techniques in neural networks are insufficient for the understanding of music. Perhaps a complete redesign of the network using a new strategy would produce more successful results. However, this experience suggests that music is an area of greater complexity than computer scientists may appreciate.

Acknowledgements:

I would like to thank Matthew Croasmun for his advice on the musical aspects of this project, and for bringing the problem to my attention.

References:

Haykin, Simon S. Neural Networks: A Comprehensive Foundation. Upper Saddle River, NJ: Prentice Hall, 1999.

Mozer, Michael. "Neural Network Music Composition by Prediction: Exploring the Benefits of Psychophysical Constraints and Multiscale Processing." In *Connection Science* 1994, pp. 247-280.

Neural Network Modeling of Neural Disease

Matthew Dubeck[1] and Matthew Kerner[2]
Yale University, Department of Computer Science
New Haven, CT 06520

Abstract

A Hopfield network is used to simulate the effects of neural disease on biological memory. The net is trained with patterns and subsequent recall is tested. Both one–shot and continuous Hebbian learning are investigated and compared. The network is modified to simulate neural damage in various ways, including neuron and synapse deletion, and then reevaluated. A gradual decline in network function is observed with varying rates of decay, depending on disease mechanism. Structural and functional damages are differentiated.

Keywords – neural networks, neurodegenerative diseases, Alzheimer's, Hopfield network, biological computing, Hebbian learning, noise reduction.

1. Introduction

The operation of neural networks mirrors that of biological neural activity; thus it can be a valuable tool in the study of the human brain. This is especially true when it comes to experiments that would be unacceptably invasive or prohibitively expensive with a living subject. Neural networks have been used in various studies of neural disease with much success (see Ruppin et al., 1996, for a survey of studies).

Our study bears the most resemblance to those of the mid to late–80s, particularly those by Ralph Hoffman. These initial experiments used a small Hopfield network to simulate some neural diseases. Our implementation extends these early studies in some interesting ways. For one, we use many more neurons in our net than Hoffman did. This allows us to encode more interesting fundamental memories (at least to the human eye), and to explore a more complicated dynamical system. Furthermore, we have simulated more diseases than was attempted in early studies. Not only are some of these diseases different from those in the literature, but many are interesting from the perspective of a computer scientist as well as that of a physician. Finally, we explore two learning models, a one–shot learning calculation and a dynamic learning process.

One of the most debilitating effects of neural disease is memory degradation. Much current research involves the study of actual patients with memory–related neural diseases such as Alzheimer's disease. However, this type of research can often be costly and ethically questionable. By creating an artificial memory using a neural network, we

1 matthew.dubeck@yale.edu
2 matthew.kerner@yale.edu

simulate damage and repair without debilitating a patient or spending prohibitively to investigate our hypotheses. Once developed and explored in the model, the operation of the diseases can be confirmed through medical experiments with actual patients. This combination of experimental physiological studies with computational modeling has been done particularly well by Hasselmo et al. (see review of work Ruppin, 1996). We seek to extend this work by inducing memory loss or dysfunction and measuring the loss in overall memory performance of two different learning algorithms.

In line with previous studies, a Hopfield network was chosen as the subject for our experiments. A Hopfield net stores fundamental memories, or key patterns, and recalls those key patterns when presented with a noisy version as input.

After configuring the network in such a way that it correctly identifies all of the key patterns without noise, we measure the performance of this net on a variety of noisy inputs. Subsequently we damaged it in various ways, all simulating the effects of neural diseases such as Alzheimer's or Multiple Sclerosis on a portion of the human brain. We measured the performance of the damaged net, and compared the data with our original set. We explored the damage threshold at which the net cannot function reasonably well, and measured the performance decrease that corresponds to the progress of each disease.

For the one–shot training algorithm, the differences between global and local information in the network are highlighted by the simulation. In addition, the differences between diseases that alter network structure as compared to network function become evident.

For the continuous learning algorithm, we observe a nuanced result that indicates a critical mass of information present in the neural net. When destroyed, the network capacity decreases at a greater rate than before. The hill–climbing nature of the network is emphasized by slight increases in performance resulting from initial minor damage.

2. The Model

A Hopfield network models a neurodynamical system consisting of a number of fully connected neurons without direct feedback. The synapses each contain a synaptic weight, and the neuron operates with the signum activation function. That is

$$y_j(t) = \mathrm{sgn}\left(\sum_{i \neq j}^{N} x_i(t-1) w_{ji} \; - T \right) \qquad (1)$$

where $y_j(t)$ is the output of neuron j at time t, sgn is the signum function, N is the number of neurons in the network, x_i is the output of the ith neuron, and w_{ji} is the weight on the synapse from neuron i to neuron j (see paragraph below for a discussion of threshold T). If the weighted sum of the inputs to the neuron is exactly zero, then the neuron's output remains what it was prior to the calculation.

The work of Horn et al. (1993) suggested that all neurons have a uniform positive threshold T based upon the probability of a neuron firing in the aggregate fundamental memories. The number of active pixels, and hence neurons, in the fundamental memories were counted and then divided by the total number of pixels in all the fundamental memories to generate a probability p. This probability represents the likelihood that a pixel is active (or that a neuron is firing). Horn et al. (1993) indicated that the optimal value of the threshold T is determined by the following equation:

$$T = \frac{p(1-p)(1-2p)}{2} \tag{2}$$

Ideally, the individual neurons update and fire concurrently and continuously, but given our simulation environment, we choose to update them asynchronously. That is, we repeatedly select a single neuron whose output gets updated using the snapshot of the network state at that time as input. The network reaches a stable state when all of the neurons' outputs are aligned – that is, none of the neurons will change state if selected for update.

We have chosen to implement one efficiency heuristic, modifying the natural asynchronous updating paradigm. Instead of selecting a neuron at random from the entire set of neurons on each iteration, we choose a neuron at random from the subset of neurons not converged at that iteration. Each of our iterations produces some constructive change in the network state.

At time t=0, we set the neuron states according to an input vector, and allow the network dynamics to cause state change over time. Once the network has settled into a steady state, we extract the state vector and evaluate performance.

The learning model for a Hopfield network is based upon Hebbian dynamics. Hebb proposed that neurons modulate their synaptic strengths based on the degree of correlation between their firing times (Haykin, 1999). This theory explains the connection between correlation and memory – if one neuron is found to be consistently in agreement with another neuron that acts as an input to it, a memory develops that encourages this correlation in the future.

The baseline learning algorithm is a one–shot calculation of a static synaptic weight matrix for the network. The weights on a N–neuron network are represented as an N by N matrix **W**, calculated using the Hebbian outer product formula:

$$W = \frac{1}{N\alpha}\sum_{i=1}^{M} \xi_i \xi_i^T - M\,I \tag{3}$$

Here, **I** is the identity matrix, α is a weight scaling constant determined experimentally

at 38.0, M is the number of exemplars, and ξ_i is the ith exemplar in vector form. This yields a symmetric matrix **W** (i.e. $W=W^T$) that is installed on the network (each row is a vector of weights for one neuron). Instantiating the weights in this manner is a discretization of a continuous learning process into a single approximate calculation.

Our second learning algorithm is a dynamic implementation of Hebb's rule. Exemplars are presented to the network during successive epochs. At each iteration, the exemplar is presented to the network through an external input. This input is added to the summation calculated in equation (1) above, much like the input from an additional synapse. Specifically, the external input is:

$$F_i^e=e_l\xi_i,(e_l>0) \tag{4}$$

where F_i^e is the external input to a neuron i, ξ_i is the ith exemplar, and e_l is a constant scaling the strength of the exemplar during learning. This constant was determined to be optimal at a value of 1.5 by experimental simulation. During each presentation, the synaptic weights of the neurons are modified using the following formula:

$$W_{ij}(t)=W_{ij}(t-1)+\frac{\gamma}{N}(\bar{S}_i-p)(\bar{S}_j-p) \tag{5}$$

γ is an activity scaling factor, which was determined to be optimal at 10.0 by experimentation. $\bar{S}_k$ is set to 1 if and only if neuron k has been firing for the last five iterations, or is set to −1 if neuron k has been passive for five consecutive iterations. If either neuron i or j has not been stable for five iterations, no weight update is performed for this synapse on this iteration. Given a stable synapse for five iterations, the weight will vary directly with correlation, reflecting Hebb's hypothesis. See Ruppin et al., 1995 for further discussion of this algorithm.

Of the two learning algorithms, the continuous version more accurately resembles a biological network, where weights are continuously updated according to the local Hebbian dynamics. The appeal of the one–shot learning algorithm is its speed.

3. Input Data

We have chosen the arabic numerals, partly because its data elements are recognizable to the human eye (see Appendix A for data set enumeration). Ten patterns is a reasonable limit on the number of fundamental memories. This is convenient, as the network recall is perfect only when the data is orthogonal and uncorrelated. Without

these conditions, cross–talk between different exemplars begins to inhibit the functioning of the network.

We represent the decimal digits 0 – 9 in a grid of bits, with a 1 representing a black pixel, and a –1 representing a white pixel. The size of this grid was determined largely through trial and error. Our original choices of 3 x 5 and 4 x 6 grids were found to be too small. The inputs were too similar, and fundamental memories did not converge to themselves. We settled on a 10 x 10 grid, leaving us enough room to vary the size, position, thickness, and shape of each fundamental memory enough that the crosstalk is negligibly low.

The data set was originally designed for the one–shot learning algorithm. We found that this algorithm was able to successfully identify all ten data elements. Furthermore, it lent itself to fundamental memories with reasonably high activity levels. Using the p index as a guide, our data set for this algorithm has a firing probability of 31.2%.

On the other hand, when using the continuous learning algorithm, the network had difficulty learning this data set. We hypothesized that the network had a smaller memory capacity when learning with the continuous algorithm, so we redesigned the data set to have a smaller p, specifically 16.6%. After repeated tests, we concluded that a network of our scale that is trained with the continuous learning algorithm cannot accommodate more than five fundamental memories. The five that we chose were based on experimentation (see Appendix B for data set enumeration).

4. Diseases

Once the baseline network had been established, a series of different "diseases" were inflicted upon the network to investigate its performance and recall. The forms of damage were chosen from a variety of sources, resembling actual biological diseases in many cases.

The first disease, *neuron elimination*, deletes entire neurons from the network by eliminating all synapses in and out of a neuron. This effect can be seen in advanced stages of diseases such as Alzheimer's and Multiple Sclerosis when neuron cells die.

The second disease, *synaptic crossing*, while not biologically motivated, offers an interesting perspective on the dynamics of the network. Two incoming synapses of a neuron have their weights exchanged. Note that this disease may create a direct feedback loop in the network, a symptom that has been observed in Multiple Sclerosis patients.

The third disease, *asymmetric synaptic elimination*, simulates a common effect of Alzheimer's disease by deleting synaptic connections throughout the cortex. Simulations of this effect have been widespread and popular, and are generally accepted as an accurate simulation of neurological disease.

The fourth disease, *symmetric synaptic elimination*, deletes some fraction of the

incoming synapses equally for each neuron. While not biologically probable, this is an interesting variant on the previous disease.

The fifth disease, *neuron seizure*, simulates a symptom of Schizophrenia in which neurons fire spontaneously. This is postulated to be the cause of some schizophrenic conditions, such as hallucinations and phantom voices.

The final disease, *impulse jumping,* simulates this further. Synaptic connections are chosen at random and their weights are set to a constant positive value. This change short–circuits connections at random, producing an excitatory effect on the network.

5. Results

When running simulations on the network trained with a one–shot calculation, each atomic trial consisted of presenting the fundamental memories with some noise introduced. This noise was created by flipping the initial states of 10% of the neurons, on average. The network was then presented with this input. We measured the amount of noise remaining after convergence, and compared it with the initial figure, yielding the percentage of noise reduction.

For each stage of each disease, we ran 1000 atomic trials. Each of these trials used a different random seed, so we could gather statistically meaningful measurements from the outputs.

As the disease "progressed," the percentages of neurons that were damaged was increased, and we ran another 1000 atomic trials. By repeating this process ranging from 5% to 95% damage at 5% intervals, we traced the gradual decrease in the performance of the network.

The baseline undamaged network running the one–shot learning calculation correctly identified the fundamental memories when presented with them. Furthermore, when presented with "dirty" versions of the fundamental memories, randomly corrupted up to 10%, the network reduced the noise and converged to a state with only 3.5% error on average. Although this is not perfect, it shows considerable improvement from the noisy input. It is believed that the the interrelation of data created crosstalk in the stored memories, resulting in plateaus in the basins of attraction that could not be overcome.

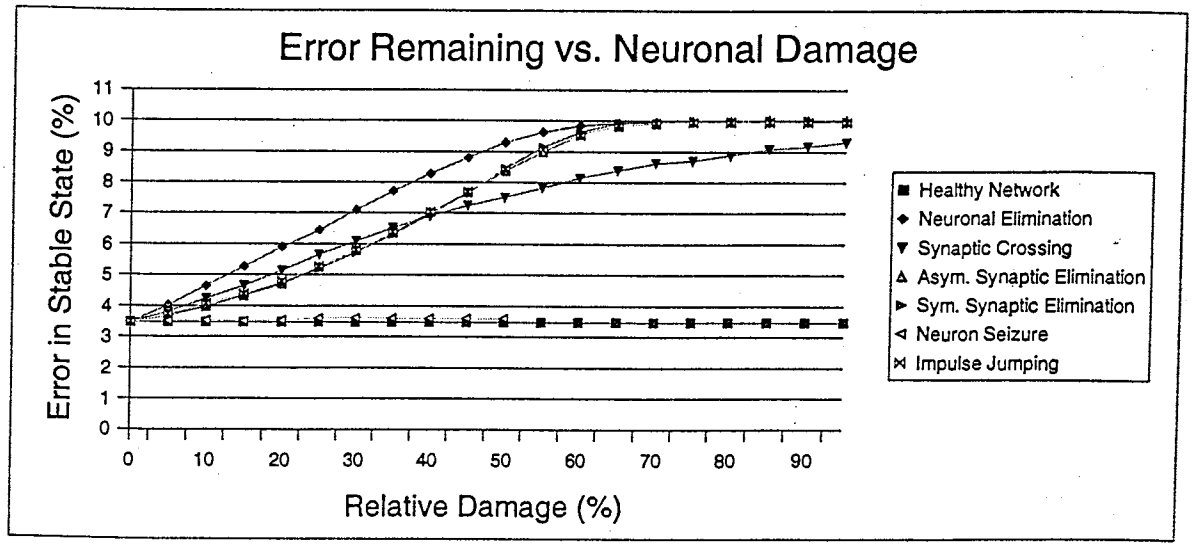


Fig 1 – Percentage of error in recall versus relative damage in a Hopfield network simulating biological memory recall with the one–shot training computation. Note that neuron seizure fails above the 50% mark. The noise internal to the simulation at this point prevents it from converging in any reasonable amount of time.

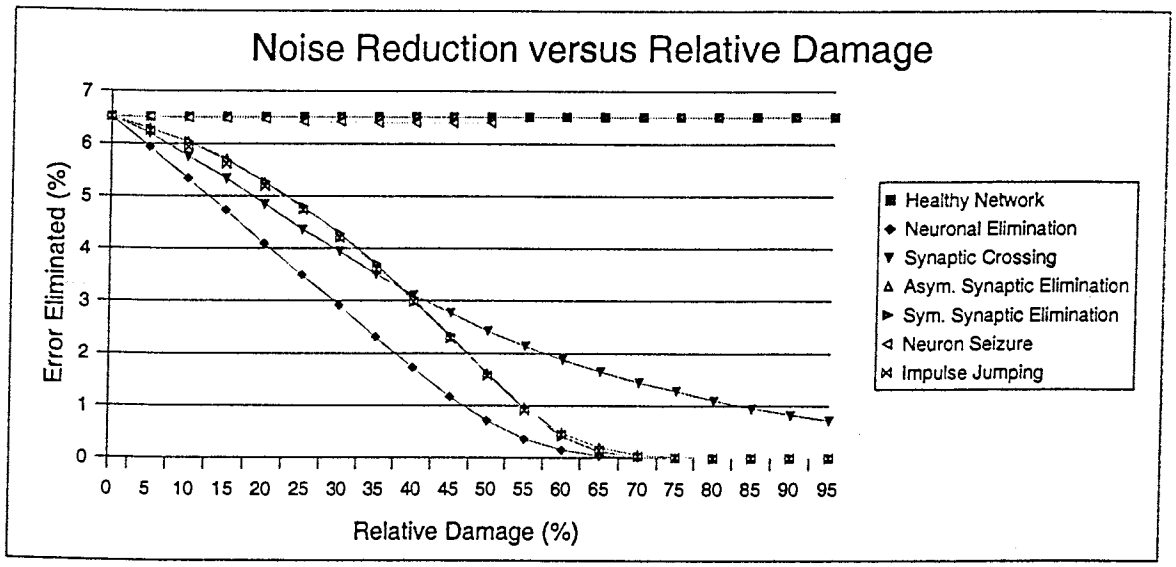


Fig 2 – Percentage of noise eliminated versus relative damage in a Hopfield network simulating biological memory recall with a one–shot learning computation.

As final data, we retained 20 data points for each disease, each of which is an average over 1000 atomic trials, each of which consists of 10 individual memory tasks. These data are shown in Figure 1. Figure 2 represents these same trials, but instead of showing the noise remaining, it shows the percentage of noise reduction on average.

It is interesting to note the various slopes of decline in the performance of the network infected with the different diseases. Most notable is the comparison between the effects of neuron elimination and synapse elimination. In these two diseases, the number of synapses that are eliminated are equal on average. Neuron elimination removes them locally, while synapse elimination removes them in a globally dispersed

manner. Figure 3 demonstrates the differences between these diseases at a greater resolution.

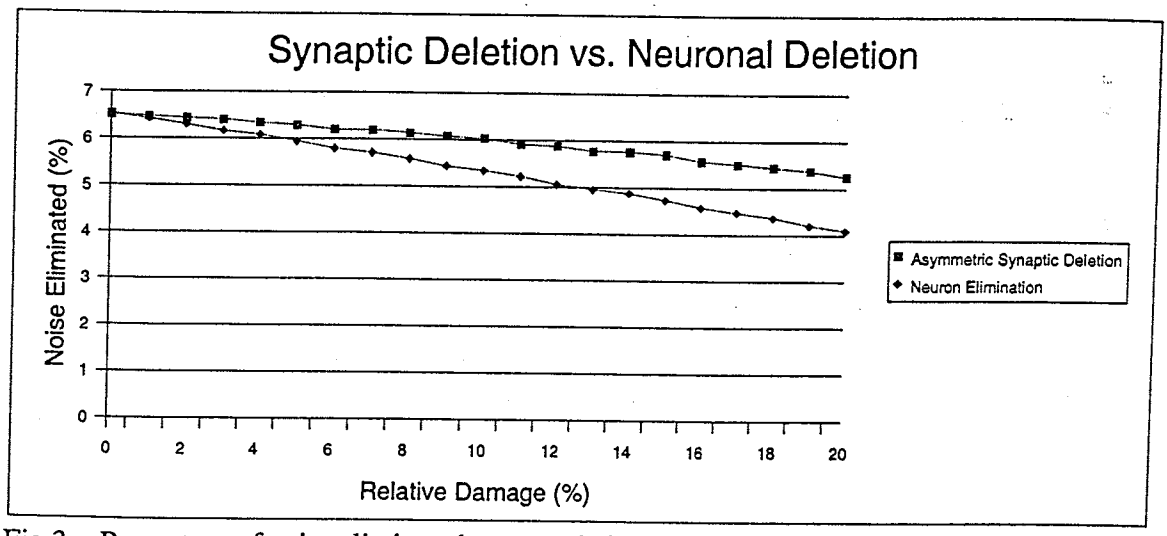


Fig 3 – Percentage of noise eliminated versus relative damage showing the greater performance degradation suffered from neuronal elimination than from asymmetric synaptic elimination.

Not only does the network suffering from neuron elimination perform less well than the network suffering from synapse elimination, but the difference grows as the disease advances.

Our metrics are in percentiles, and may at first seem dissimilar to the measurements reported in the literature on neural simulation. For example, Ruppin uses the metric of overlaps in his studies. Overlaps quantify how close the state of the network is to the desired output at any iteration, or upon convergence. This is analogous to Hamming distance, which is simply the number of bits in the binary vector representing the network state that differ from the desired state vector.

In fact, our metrics are nearly identical to these. Since we have 100 neurons in our network, a percentage of error is exactly the Hamming distance between the desired state and the stable state.

When running simulations using the continuous learning algorithm, the data was presented over the course of ten epochs. In each epoch individual memories were chosen in random order, and presented to the network repeatedly. At each presentation, the network ran until convergence while adjusting synaptic weights as in equation 4. It was found that the network performed better if the early epochs presented each data pattern only once or twice, while later epochs presented each pattern as many as ten times.

After the learning phase, the network was infected with the diseases listed above. As with the one–shot network, each disease was simulated at different levels of development, ranging from a healthy network to one with a relative damage metric of 40%. Furthermore, each simulation was run with a noise level ranging from pristine to

50% corruption. Each simulation was averaged over fifteen iterations to gather statistically valid data. Due to the computational requirements of the learning algorithm, it was infeasible to increase the sampling of this network.[3]

The continous–learning network performed as summarized in Figure 4 below. The ability of this network to reduce noise far surpassed that of the one–shot network. In fact, when presented with patters corrupted up to 10%, the continous–learning network returned exactly the memories that it had learned, and this performance degraded very slowly up until the 25% mark. However, the network had significant difficulty learning all five patterns exactly, and always contained some corruption in its stable states.

Interesting to note is the improvement in performance caused by relative damage of under 20%, particularly with asymmetric synapse deletion and neuron seizure. This improvement holds across the spectrum of noise. Other related studies have found similar results, particularly Ruppin et al., 1995.

Also worth noting is the stairway–like structure of the performance surface, particularly in synaptic crossing and symmetric synapse deletion.

3 24 hours of continuous simulation on six different machines was required to gather the data presented on the network with the continuous learning algorithm.

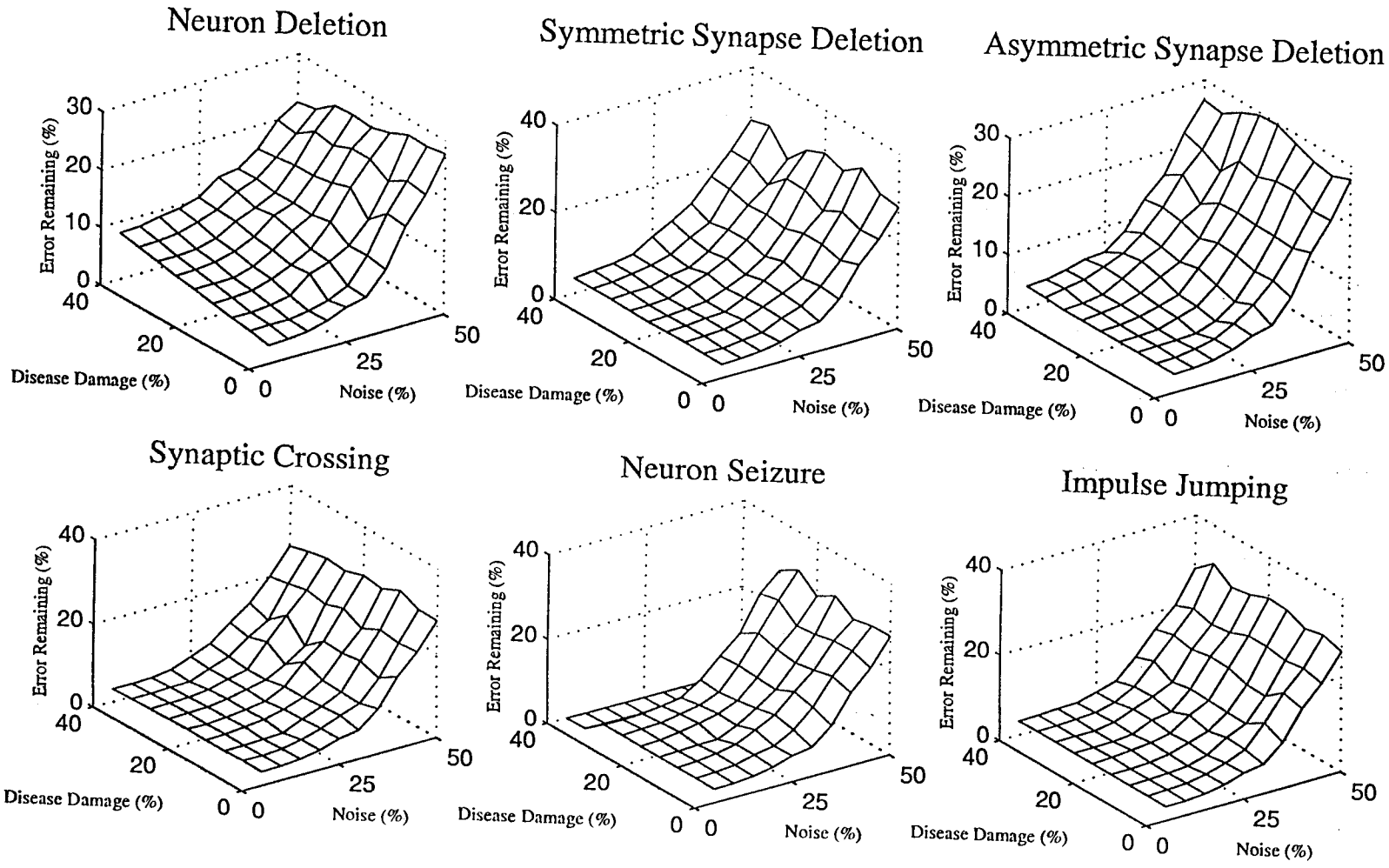


Fig 4 – Percentage error remaining in the pattern after the dynamically trained network converged plotted in the vertical axis. The x–y plane over which it is plotted contains the percentage of noise in the exemplar initially fed to network and the percentage of damage inflicted for the given disease. The general slope toward the far corner of the graph indicates the decreased performance of highly damaged networks with high noise levels. A low slope at the low end of the noise spectrum indicates that there was little noise to eliminate.

6. Conclusions

From these experiments, we draw several conclusions about the functioning of memory recall in the face of neurodegenerative disease. In the case of the one–shot network, it is surprising that three of the diseases operate in close approximation to each other: symmetric synapse elimination, asymmetric synapse elimination, and impulse jumping. The former two diseases have an inhibitory effect on the operation of the network, although the latter is excitatory. We would expect to see some difference in the performance of the network when damage is uniform, as in the symmetric disease, as opposed to the asymmetric variation. We believe that the network responds in a similar manner to any changes affecting the synapses alone. The initial rate of degradation is relatively small compared to other diseases, but accelerates as the disease progresses. We conclude that a critical mass of synaptic information is necessary to maintain normal network operation, and the loss of this information speeds the network's degradation. This conclusion agrees with much of Ruppin's work that indicates a 25% damage threshold. (see for example, Horn, Ruppin, et al, 1993)

Furthermore, the performance of the network with seized neurons is almost equal to that of the healthy network throughout the feasible range of the experiment. We conclude that the disruption caused by the firing of random neurons is functional in nature, not structural. As a result, the noise elimination capacity built into the network can compensate, although it becomes overloaded at a certain point. It is intuitive that this threshold would be roughly 50%: for every correction, there would be a corresponding introduction of noise, returning the network to the same distance from a stable state.

Synaptic crossing exhibits a slower and more gradual degradation than many of the other diseases we simulated. We conclude from this that the process at work is one of gradual confusion, rather than one of catastrophic structural damage. The weights themselves are preserved in this disease – it is just their location in the network that changes. Although this is biologically far fetched, it hints at the ability of a network to gradually modify its stable states as weights migrate throughout the network.

We see the diffuse quality of the information in the net through its graceful decline as it suffers from diseases. Prior to a damage threshold of 20%, where performance begins to slope downward at a greater rate for the majority of the diseases, this net is able to function in approximately healthy condition. A few trials showed an increase in noise reduction as some minor random degeneration occurred. This can be attributed to the hill–climbing nature of a neural network as it converged (hill–climbing algorithms often benefit from randomized influence). Overall, the network is surprisingly resilient at accomplishing its task when damaged.

While the network shows a capacity for dispersed information storage and retrieval, it also shows that some important information resides in the relation of weights

within each neuron. This is clearly shown by the decrease in network performance as neuron elimination sets in. This decrease is more pronounced than that of any other disease. We see that not only does the neuron itself fail to function, but the network as a whole loses a key part of its information. It is likely that this is especially true for a Hopfield network, since each node contributes to the inputs of every other node, globalizing any local damage.

The results of the continuous–learning model present more questions than those of the one–shot model. Particularly interesting is the presence of peaks and valleys in the performance surface as the disease and noise levels increase significantly. This suggests that various spurious memory elements are introduced and deleted as information in the network is corrupted. This would correspond with some clinical observations of biological neural disease, such as Schizophrenia, in which patients experience voices or hallucinations.

For the majority of the diseases, the threshold at which performance begins to decrease at a greater rate is just below the 20% damage mark. This is in line with our results from the one–shot network experiment, and suggests that a similar critical mass of information is stored with both approaches.

The unexpected relationship between symmetric synapse deletion and asymmetric synapse deletion occurs again, with asymmetric deletion demonstrating a gradually curved surface that contrasts with the bumpier surface produced by symmetric synapse deletion. We conclude that the locality of some of the synaptic information plays a large role in keeping the network memory consistent. When that locality is disrupted, as in symmetric synapse deletion, the network falls in and out of spurious configurations. As the locality is undisturbed in asymmetric synapse deletion, this behavior is less evident.

Despite the encouraging results from this study, it suffers from several shortcomings. We could test combinations of these diseases, or use additional mathematical tools to create higher fidelity models. For example, the literature suggests some enhancements over a basic Hopfield network that can illustrate some key aspects of neural disease. Instead of treating all of the neurons in the network as though they are in close proximity, many newer studies have simulated the physical distance between different neurons. (Reggia et. al., 1996) This gives researchers a clear criteria for designing diseases that are local in nature, but that affect a global network. Ruppin, in particular, has developed this idea and created simulations of a wide variety of interconnected structures.

One of the more intriguing ideas is that a network can dynamically compensate for damaged synapses. Various studies have looked at both global and local compensatory mechanisms that hide the damage done to synaptic connections. Unfortunately, as in Alzheimer's, the temporary improvement quickly leads to an even more drastic drop in network performance once the compensatory mechanism is saturated. (Horn et al., 1993) We considered simulations along these lines, only to

conclude that our continous–learning algorithm had to be refined before our simulation could support this feature.

All in all, neural networks offer a unique opportunity to researchers who wish to study brain function without participating in ethically questionable and financially draining clinical experiments. Many of the symptoms and effects of neural diseases have been successfully replicated in the computer lab, and offer a clear, detailed view of the mechanisms at work. Further research certainly remains to be done in this exciting field.

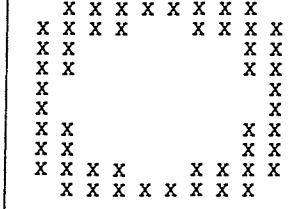
References

1. Haykin, Simon (1999). *Neural Networks: A Comprehensive Foundation*, 2nd ed. New Jersey: Prentice Hall.
2. Reggia, J. A., Ruppin, E., Berndt, R. S. ed. (1996). *Neural Modeling of Brain and Cognitive Disorders*. New Jersey: World Scientific.
3. Horn, D., Ruppin, E., Usher, M., Herrmann, M. (1993). Neural Network Modeling of Memory Deterioration in Alzheimer's Disease. *Neural Computation* v.5, 736–749.
4. Ruppin, E., Reggia, J. (1995). A Neural Model of Memory Impairment in Diffuse Cerebral Atrophy. *British Journal of Psychiatry* v.166, 19–28.
5. Hoffman, R. E., Dobscha, S. K. (1989). Cortical Pruning and the Development of Schizophrenia: A Computer Model. *Schizophrenia Bulletin* v.15, no. 3, 477–490.
6. Hoffman, R. E. (1987). Computer Simulations of Neural Information Processing and the Schizophrenia–Mania Dichotomy. *Archives of General Psychiatry* v.44, Feb 1987, 178–188.
7. Anderson, J. A. (1983). Cognitive and Psychological Computation with Neural Models. *IEEE Transactions on Systems, Man, and Cybernetics* v.SMC–13, no.5, September/October 1983, 799–815.
8. http://suhep.phy.syr.edu/courses/modules/MM/index.html

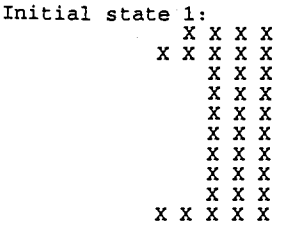
Appendix A – Data Set Enumeration

Our final data set for the one–shot calculations follows. The final data set includes enough differentiation in the sizes, thicknesses, positions, and shapes of the individual fundamental memories that convergence to those states is possible in a healthy net.

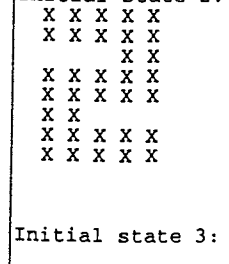
Initial state 0:



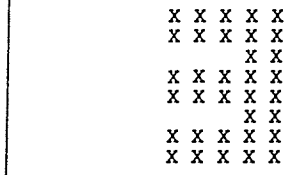
Initial state 1:



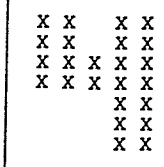
Initial state 2:



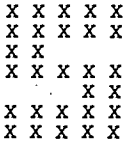
Initial state 3:



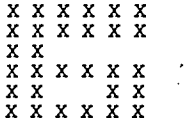
Initial state 4:



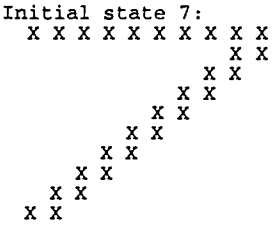
Initial state 5:



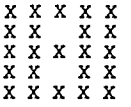
Initial state 6:



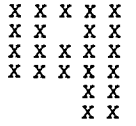
Initial state 7:



Initial state 8:



Initial state 9:



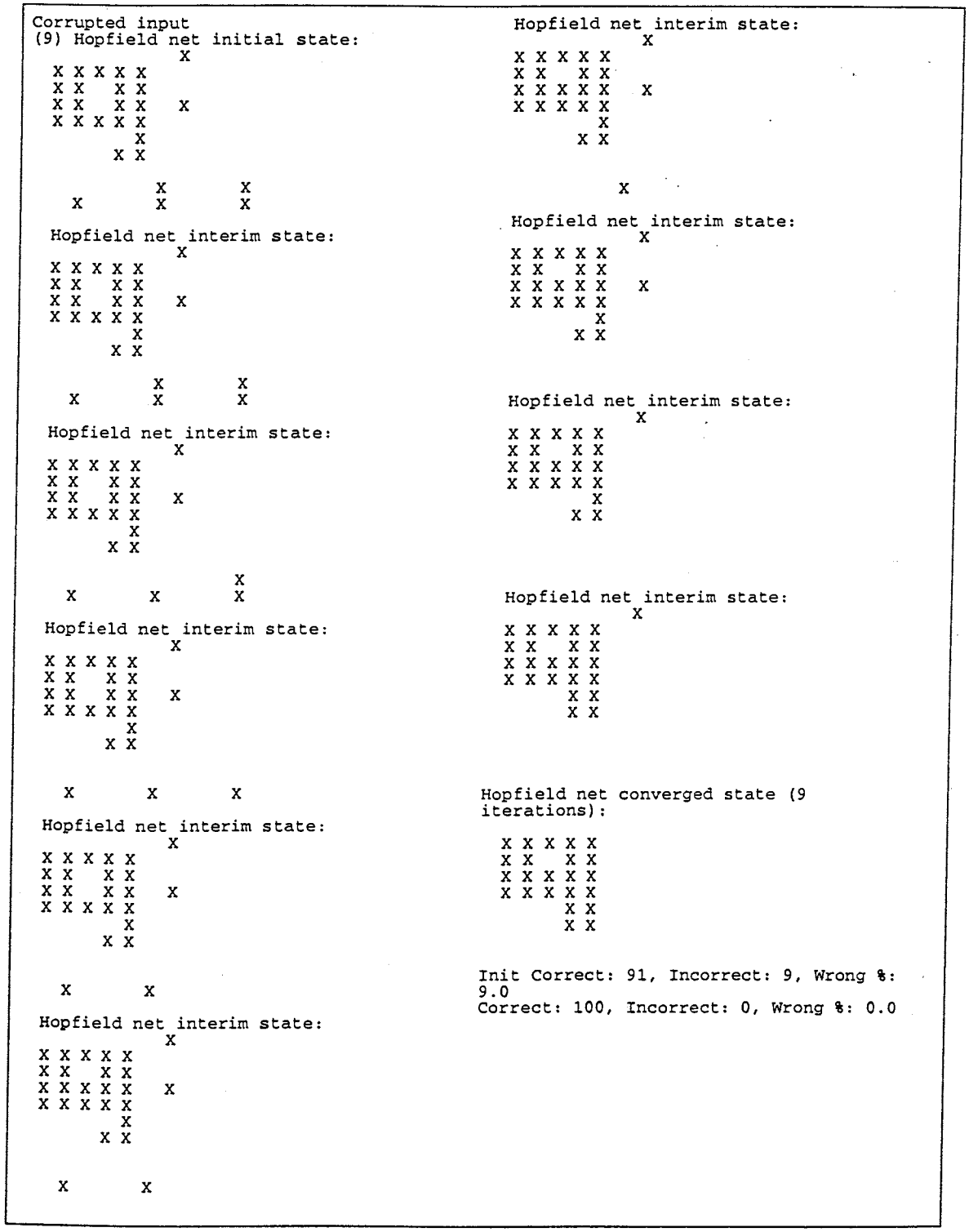
Appendix B – Continuous Learning Data Set

Our final data set for the continuous Hebbian learning algorithm follows. It was necessary to rescale the digits in order for the network to learn any of the patterns. Interestingly, if the data sets are swapped between one–shot and continuous learning, recall is significantly worse.

```
Initial State 0:

11111
1   1
1   1
1   1
1   1
11111

Initial State 1:

      1
      1
      1
      1
      1
      1
      1

Initial State 2:

     11111
         1
         1
     11111
     1
     1
     11111

Initial State 4:

  1   1
  1   1
  1   1
  11111
      1
      1
      1

Initial State 7

   11111
       1
       1
       1
       1
       1
       1
```

Appendix C – Convergence Trace

Below is a state–by–state diagram showing the convergence of a noisy input '9' to the output in a healthy network using a one–shot learning calculation.



Neural Network Models for Face Recognition

Robert Dugas[1] and Jesse Grauman[2]
Yale University, Department of Computer Science
New Haven, CT 06520

Abstract

We tested three neural network models for the purpose of recognizing faces. All three designs use variations on the backpropagation network (BPN) paradigm. The first design uses the actual images to identify the faces, while the other two models first preprocess and compress the inputs, one by using another BPN, and the other by using principal component analysis. We trained the networks to identify twenty images of faces and tested their effectiveness in identifying variations of the training images, as well as in correctly classifying other images as unfamiliar. We found that while preprocessing using principal component analysis provided a moderate improvement in identification rate, preprocessing using the BPN did not. We discuss the implications of these findings for the use of the BPN design as a feature detector.

Keywords – neural networks, face recognition, backpropagation network, neural modeling

1. INTRODUCTION

1.1 Background

Despite the ever-increasing capabilities of modern computers, computational modeling of seemingly basic neural tasks continues to confound researchers in artificial intelligence. Tasks which the human brain performs without conscious effort, such as navigating obstacles or understanding speech, prove difficult to recreate computationally. This project uses a computational neural network to explore one such task, that of face recognition.

The specific task we have chosen to approach is the identification of a finite set of faces from an exemplar set. In this problem, a network is first trained to identify each face in a set of unique exemplars. Once it can identify these faces, it is presented with slightly different images of the same faces, perhaps varying in pose or facial expression. The network is then tested on its ability to positively identify these images as being the same faces used for the exemplar set. In addition, it is presented with images of unfamiliar faces and tested on its ability to "reject," or fail to identify, these faces.

Successfully implemented, such a model has a variety of interesting applications. Police sketches, for example, could be mapped to possible suspects from a database of

[1] robert.dugas@yale.edu
[2] jesse.grauman@yale.edu

photographs. Job interviewers and test administrators could obtain reliable identification of subjects. Security systems could grant or deny access to individuals based on whether or not they could identify them as being in a company database. Beyond these applications, a successful model might be able to deepen our understanding and appreciation of the human brain's face recognition capabilities.

Part of the reason that face recognition—and vision in general—is so problematic for computers is that an image of a face is merely represented as a large matrix of intensity values, many of which are similar across many faces. Thus, it stands to reason that in order to accurately identify faces—particularly variations of the original images from an exemplar set—it is helpful to preprocess the image of a face in order to extract its salient features. Presented with a much smaller and more variant "feature vector" rather than the original intensity matrix, a network should find it easier to identify the variations of the training images. This is due to the fact that their features are more highly correlated with those from the images used for training.

1.2 The current project

In this study, we compare the performances of three different neural network designs that we created for face recognition. All three use the backpropagation network (BPN) paradigm. In the first design, the images are not preprocessed; the network is simply trained to identify each of the pixellated images in the training set and then tested accordingly. This network was used as a control in order to test the effects of the preprocessing in the other two. In the second design, the training images are compressed into feature vectors using a BPN in a scheme developed by Golomb et al. (1991, cited in Skapura, 1996). The preprocessed images are then used in order to train another BPN used to identify the faces. In the third design, we preprocessed the images using principal component analysis and then used the principal components to train a BPN to identify the faces.

We hypothesized that the two networks that used preprocessing would outperform the control (first) network in identifying the variations of the training images. We believed that in extracting the features of the training images, these two networks would create more generalized exemplars than simply by using the original pixellated images. While two images of the same face might differ somewhat in the actual pixel values, their feature vectors, we hypothesized, would remain relatively similar to each other.

After a series of experiments, our hypothesis was partially confirmed. The design using principal component analysis to extract features did show some improvement over the control in identifying variations of the training images. However, the design using backpropagation for preprocessing did not improve over the control, and was actually deficient to it in some respects. This caused us to question the validity of backpropagation as a tool for feature extraction from images.

2. EXPERIMENT

2.1 Data Set

Our set of training images consisted of 20 grayscale JPEG images of faces, taken from the face recognition database of AT&T Laboratories, Cambridge (http://www.cam-orl.co.uk/facedatabase.html). The original images were 92 x 112 pixels; in order to meet realistic speed and memory requirements, we used an image manipulation program to scale the images down to 30 x 36 pixels. The database contained 10 different images of each person, and we tried to choose the most generic one for our training set—one where the subject was facing forward, without any exaggerated facial expressions, and if possible, in medium lighting.

In order to test the effectiveness of our identification network, we chose three other sets of images. The first consisted of "easy variations" of the training set—20 images, one of each face, each of which was slightly different from its corresponding picture in the exemplar set, such as small tilts of the head and slight but noticeable changes in facial expression. The second, a set of "hard variations," contained images of the same faces that differed more substantially from those in the training set. These variations included pictures taken with or without glasses, more pronounced tilts of the head, and exaggerated facial expressions. These two sets were used to test whether our identification networks would be able to correctly identify these faces as being very similar to the faces used for training. The training images, as well as the easy and hard variations, are shown in Figure 1. Finally, the third set, which included images of 20 other individuals, was used to test whether our networks would correctly identify these unfamiliar faces as not corresponding to any of the exemplars.

Since the images were 30 x 36 matrices of intensity values, they were represented in our networks as vectors containing 1080 elements, and the elements were normalized to range from 0 to 1 before being fed through the network.

2.2 Network Parameters

All networks described in the following section were built using the neural networks toolbox in the MATLAB programming environment.

2.2.1 Control Network

Because our main goal in this project was to test the effects of preprocessing on face recognition, we first needed a control network that would attempt to identify faces without using any preprocessing. This network, a fully-connected BPN, contained 1080 input nodes, 20 output neurons, and no hidden neurons. The activation function used for the output neurons was the sigmoidal logistic function:

$$y_j = 1 / 1 + \exp(-v_j)$$

For training, each exemplar consisted of one of the face images as its inputs, and a 20-element bit vector as its output. For each of the 20 inputs x_i, the trained output was a bit

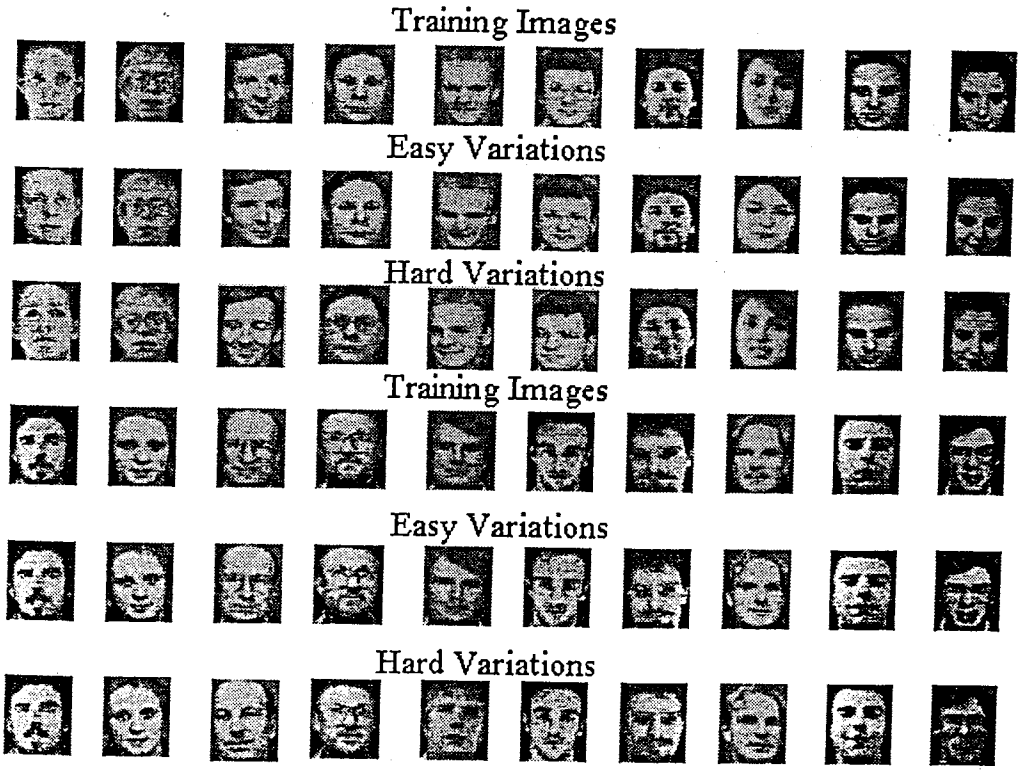


Figure 1. Set of training images and variations.

vector in which only bit i was set to 1, while the rest were set to 0. Thus, the network was trained to output a unique bit vector for each training image.

2.2.2 Preprocessing using Backpropagation

Our second network design used a BPN in order to preprocess the images. We modeled this network after one built by Golomb et al. (1991) in their design of a neural network to classify faces by gender. In this unique design, a BPN is used to map an image to itself. The input layer contains 1080 nodes, as does the output layer of neurons, which uses the linear activation function $y_j = v_j$. The hidden layer contains a much smaller number of neurons—40 in this case—and uses the logistic function. These neurons are designed to act as feature detectors. Each exemplar uses the basic 1080-element image vector as both its input and its output. Thus, in training, the purpose is to create an autoassociative network in which each of the 20 training images reproduces itself.

The outputs of interest in this network, however, are not the reproduced images but the 40 outputs of the hidden layer. These outputs represent a feature vector that can

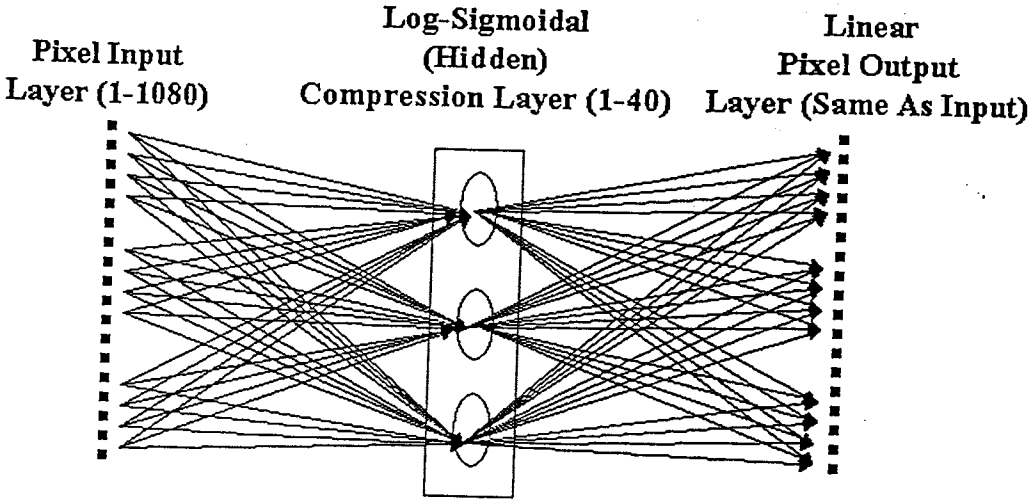


Figure 2. The Compression Network

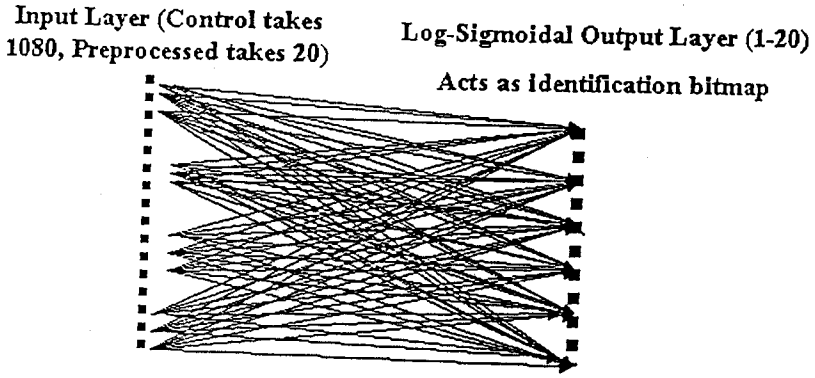


Figure 3. The Identification Network

be used to regenerate the original images. In order to extract these outputs for each training image, we created a feedforward network whose architecture, including the weights, was identical to that of the first two layers of the trained autoassociative network.

In order to identify the preprocessed inputs, we created a second network which consisted of 40 input nodes and 20 output neurons, which used the logistic function. This network was trained to map each feature vector created by the first network to a unique bit vector as in the control network.

2.2.3 Preprocessing using Principal Component Analysis

Our third network preprocessed the pixellated images using principal component analysis, another way of compressing a large input into a much smaller one. In this method of compression, the inputs are multiplied by a matrix whose rows are the eigenvectors of the inputs covariance matrix. The result is a set of input vectors whose

	No preprocessing (Control)	Preprocessing using BPN	Preprocessing using principal component analysis
Identification of easy variations	77%	69%	90%
Identification of hard variations	57%	49%	61%
Rejection of unfamiliar images	89%	77%	88%

Table 1. Comparison of Results (shows avg. % correct over 10 simulations)

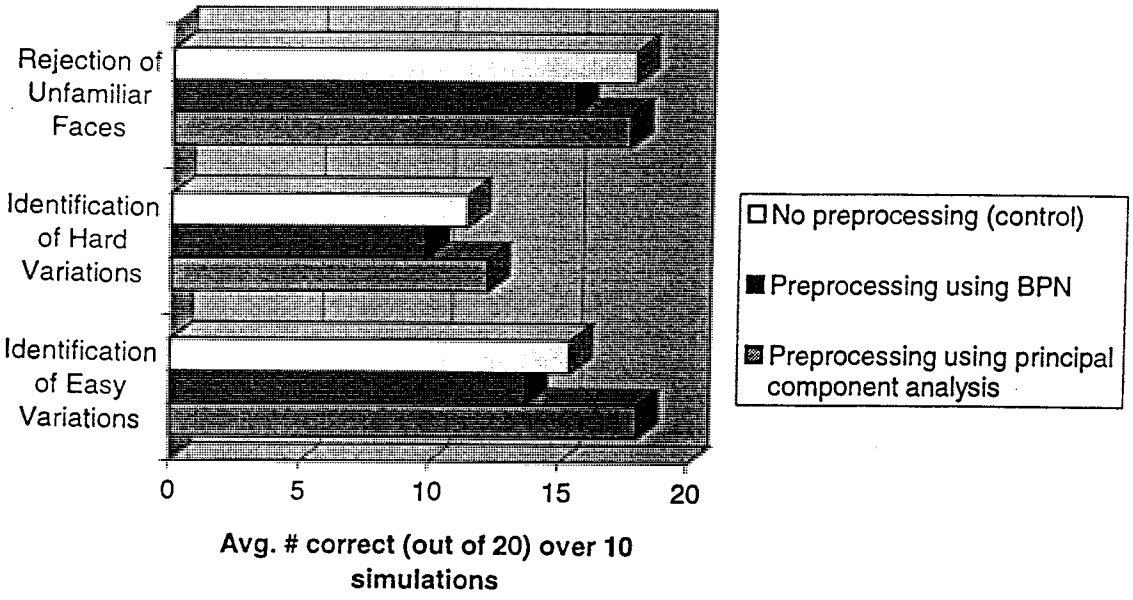


Figure 4. Comparison of Results

components are orthogonal and represent the components of the original inputs that contribute the greatest amount to the variance of the original inputs. In our implementation, all components that contributed less than 1% to the total variance were eliminated. The result were principal component vectors that consisted of 19 components, a significant reduction from the original data.

To identify these inputs, we created a second network similar in design to our other two identification networks. This network had 19 input nodes, one for each principal component, and 20 output neurons which used the logistic function. It was trained to map each training example's principal component vector to a unique bit vector as in the control network.

2.3 Method

In each of the three designs, we trained the networks to identify the 20 images in the training set. For the two designs in which preprocessing was used, we first preprocessed the training inputs, either using the BPN or principal component analysis. For the preprocessing, the BPN was trained for 2000 epochs. We then trained the identification networks using the feature vectors produced by the preprocessing. We tested the trained networks on both the "easy" and the "hard" variations of the training faces, as well as on the 20 additional unfamiliar faces. For the networks in which preprocessing was used, the same preprocessing techniques used for the original inputs were applied to the new inputs. We recorded the number of correct identifications of the easy and hard variations, as well as the number of correct "rejections"—that is, negative identifications—of the unfamiliar images.

3. RESULTS AND DISCUSSION

We defined a positive identification of an input with training image x_i as occurring if output neuron i had an activation of 0.5 or greater, while all other output neurons had activations of less than 0.5. Failure to meet both of these criteria was defined as a negative identification—a decision that an input did not represent a face used for the original training set.

All of the networks were able to accurately identify the images in the training set. Thus, the results we cite here concern the accuracy of the networks in identifying the variations of the inputs, as well as their accuracy in rejecting the unfamiliar inputs. The results showing the performances of all three models can be seen in Table 1 and in Figure 4. Percentages shown represent the average performance of the networks over ten simulations. Our first observation was that the control network performed surprisingly well, identifying 77% of the easy variations, 57% of the hard variations, and correctly failing to identify 89% of the unfamiliar images. However, preprocessing the data using the BPN did not provide the improvement in accuracy we had anticipated. Instead, the

performance of this network was actually worse than that of the control, as its accuracy rate was 8% less than the control's for the easy and hard variations, and 12% less for the unfamiliar images. Using principal component analysis to preprocess the data, however, did improve accuracy in identification of the easy variations. When the identification network was trained on the principal component vectors, it was able to identify 13% more of the easy variations than the control. Although its performance was comparable to the control's in the other two tests, the improvement in identification of easy variations was an encouraging one, as it indicated that there was a benefit to preprocessing the inputs.

In speculating as to why the preprocessing used by the BPN did not work as expected, we noted that over the course of a number of simulations, the accuracy rates of this model varied widely when compared to the accuracy rates of the other two models. One reason for this might be the random assignments used to initialize the variables for a BPN. This caused significant variations in the feature vectors produced by the hidden layer. Similarly, because the mechanism by which the BPN produces its feature vector has a "black box" kind of framework, it is virtually impossible to tell whether the network is actually extracting features, and if so, which ones. Conversely, using principle components analysis to extract features is a more rigorous mathematical method known to produce orthogonal components that are not correlated with each other, and thus are useful as a feature vector.

4. CONCLUSION

In order to further investigate the extent to which preprocessing images can help face recognition in neural networks, it would be useful to repeat the above study using larger data sets. Presumably, this would create a more difficult task for a network which did not use any preprocessing, and the improvements created by preprocessing would be more significant. In addition, it would be useful to investigate the performance of a network trained on a few separate exemplars for each face it desires to identify. This would likely increase the robustness of the network because it would learn to identify a more generalized version of an individual's face. Given the numerous applications for face recognition, it would be beneficial to continue to study the effectiveness of preprocessing and neural networks in this domain.

REFERENCES

1. Golomb, B.A., Lawrence, D.T., and Sejnowski, T.J. SEXNET; A neural network identifies sex from human faces. In Lippman, R.P., Moody, J.E., and Touretsky, S.,