01 May 1990

A direct access method using a neural network model

John William Meyer

George Winston Zobrist
Missouri University of Science and Technology

A DIRECT ACCESS METHOD USING A NEURAL NETWORK MODEL

J. W. Meyer* and G. W. Zobrist

CSc-90-4

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314)341-4491

*This report is substantially the M.S. thesis of the first author, completed May 1990.

ABSTRACT

One of the concerns in computer science involves optimizing usage of machines to make them more efficient and cost effective. One item of particular concern is the use of secondary storage devices, devices that store data other than in the main memory of the computer to which it is attached. The times for searching for data on these devices consistently proves to be a contributing factor in inefficient computer usage.

One data access method that avoids searching when possible is the hashing method. A function is defined to return the record number of a record based on its key field. The record can then be read in directly. A problem exists when more than one key maps to the same record number, called a collision, and must be dealt with, usually adding search time in the process.

Training a neural network to do this avoids these collisions. The Hamming network, based on the Hamming distances of two binary patterns, is trained to map the key fields directly to the record number of the data. The key must be converted to a binary format. The program passes the key to the network that simultaneously calculates the form of the Hamming distance between that key and all keys known to be in the file. A MAXNET network takes these distances and reduces them until no more than one is positive. The record number is found from the results, and the data can be accessed directly. All disadvantages from the software version are virtually eliminated.

TABLE OF CONTENTS

I. INTRODUCTION

Computer central processing units continue to accelerate in instruction processing times. Memory speeds up as ways of designing integrated circuits evolve. And though access time to secondary storage devices improves, data storage and retrieval still set upper limits to computer efficiency and speed.

Many methods have been developed in software to alleviate this problem. But software implementations add their own limitations to the problem. It would be desirable to blend in neural network structures with contemporary computing hardware to increase the speed of the methods now in software form.

With advances in hardware technology, computing models that were originally possible only through simulation can now find realization in electronic circuits. In particular, the neural network technologies become more than mathematical formulae and software procedures. Commercial neural network systems now exist so that machines that learn are no longer a promise of the future but are a reality of today.

The current and future applications of neural networks include target identification and tracking, speech recognition and synthesis, image processing and decision making systems similar to expert systems [Caudill, 1987, 48]. Instead of being programmed (in conventional terms) with algorithms and functions, neural networks are "trained" to supply the correct response for a given input stimulus. This gears them more towards the type of problem humans solve easily than towards the type computers now solve [Caudill, 1987, 48].

In this particular case, the idea centers around training neural

networks to access data directly from files on secondary storage devices. Due to flexibility in neural net models and the inherent speed of a hardware implementation, this paper proposes that neural networks can and will be a solution to slow rates of data retrieval. This proposal does not involve increasing transfer times of the data nor times dependent on the individual storage device. It instead demonstrates a way to accelerate the process of locating the appropriate data on the storage device, which is the main drawback of software solutions.

Journal articles refer to neural networks designed to solve this problem. However, they are only mentioned in passing. No details are given, and at best, the type of model being used is mentioned. Therefore, it was neccessary to take these references and eliminate any work already considered. This was rather easy since most references were to database queries, and this particular form of data storage and retrieval is not considered. Other references did not discuss the particular access method chosen, nor did they discuss the neural network model chosen.

II. FILE ACCESS METHODS

When considering efficiency in computing, one element poses the problem of causing the central processing unit to be idle for sometimes intolerable amounts of time. That element is data retrieval from secondary storage devices. Over the years, improvements in technology have sped up the access time for these devices. However, they continue to contribute to CPU idle time and spent resources. Therefore, the manner in which the data is stored and retrieved has been of great interest in optimizing computer usage.

There exist several types of storage devices. One class of storage devices depends on storing data sequentially due to physical constraints of the device. An example is magnetic tapes which require data to be stored sequentially along their length. The tape must be advanced and rewound to read and write data, making it difficult to go to specific positions. Direct access storage devices will be considered here--that is, devices on which data can be randomly placed. Magnetic disks and drums fall within this category. For simplicity sake, when speaking of the storage device, it will be assumed that the device is a magnetic disk.

A. DEFINITIONS

It is essential to define some terms involved in data storage before continuing. These terms include ways the data is organized logically (from the programmers standpoint) and physically (on the storage device).

1. Data Representations. A collection of related data about an individual or item that is treated as a separate unit is called a record.

Each portion of a record which gives one specific attribute of the item or individual is called a field. A collection of logically related records on a medium separate from the internal storage of the computer is a file. A unique field that distinguishes a record from all others in the file is called a key [Tremblay, 1975, 143]. In some instances, more than one field must be used to generate a unique key. For example, in a file of student data, the student identification number is used as the key. However, in a file of addresses where individuals have not been assigned such a number, the three fields comprising the name of the individual (last name, first name, middle initial) may be used to distinguish the records.

2. Addresses. When discussing the physical organization of the records, the term address is used to indicate a given record's placement. If the address indicates the physical location of the record on the storage device, it is termed the absolute address of the record. If the address reflects the record's position with respect to the beginning of the file, the address is referred to as the relative address. Finally, the record number is the physical position of a given record in a file, counting from the beginning of the file. This means the first record has record number 0, and the nth record in a file has record number n-1.

3. File Organization. Once the file is created, there must exist some way to retrieve the appropriate data. The way the file is arranged may be a part of the retrieval method. If the records are placed in consecutive positions in the file, the file is said to be arranged sequentially. If the records of the file are ordered by the values of the keys, the file is said to be sorted. When a second file is created

containing the keys and the addresses of the records of a data file, the data file is said to be indexed, and the second file is called an index file. The following sections detail these arrangements and how data is retrieved from them.

B. SEQUENTIAL FILES

The simplest way to organize records is to write them to the file in contiguous areas on the storage device as the data is obtained [Tremblay, 1975, 213]. The resulting file is called a sequential file. Sequential files require little maintenance when records are added later since they are appended at the end of the file. If records are deleted, the file can be rewritten without those records, or records at the end of a file can be copied on top of the records to be deleted. The size of the file is changed to ignore the duplications at the end of the file.

1. Sequential Search. Records are retrieved in a straightforward manner. One starts at the beginning of the file and checks the key of each record in turn until the key for the desired record is found. Then the record is read in full from the file. This method of searching for the key is called a sequential search. The average amount of time to retrieve the record is proportional to n/2 where n is the total number of records in the file [Tremblay, 1975, 213]. Note that if the record is not in the file, the entire file will be searched before this can be discovered.

2. Blocking. One way to improve transfer rates in sequential files is to organize the records in physical blocks of data. Instead of reading in a single record at a time, a block of data is read into memory, and

records are accessed from this until a record from a different block is required. The problem with blocking is the assumption that consecutive records will be needed at the same time. If this is not true, then instead of transfer times based on individual records, transfer times are based on retrieving entire blocks and increase accordingly.

The blocking size is dependant on the size of the records, how the blocking size affects retrieval efficiency and limitations of the hardware. Only best guesses can be offered to set the blocking sizes without actually experimentation on the file and the program that accesses the file.

C. SORTED FILES

The second file organization method revolves around a sequential file that has been ordered by the value of the keys. Such a file is said to be sorted. The keys may be sorted in ascending order (they increase as the file is traversed from beginning to end) or in descending order (they decrease from beginning to end.)

1. Sorting Methods. Many sorting methods have been devised to order the records in a file. Insertion sorts involve stepping through the file by positions, determining which record belongs in each position, then copying the record to that position. Recursive sorts involve breaking up the file into segments that eventually result in single records or smaller sorted segments. Then the segments are fitted back together in larger, sorted segments as the order in which they are broken up is reversed. Merge sorts involve taking sorted files, or sections of files, and combining them to give a single sorted file. Though this list does not

exhaust the available sorting schemes, it does give a flavor of the variety that exists.

When evaluating the efficiency of sorting methods, it can be shown that the optimal method for a given file depends on how close the file is to being sorted. Some sorts work better when the file is not very sorted. Others work better when the file is almost sorted. In light of this, some sorting systems have been created to use different sorts at different times during the process of sorting a single file.

Some programming language implementations supply sorts as part of the library accompanying the compilers. Operating systems come with utility programs that often include programs designed to do nothing more than sort files. So sorting a data file is not much of a concern, especially to the user.

2. Maintaining Sorted Files. Maintenance is more complicated than for simple sequential files. Adding a record becomes more difficult. The record must be inserted in its proper place, requiring that following records must be moved by one position. If many records are to be added, it may be more effective to append the new records to the end of the file and resort the file. Deleting files can be done by marking the deleted records in some way, and rewriting the file in order, skipping those records which are marked.

3. Sequential Search. Retrieving records in a sorted file can be done with the help of a sequential search as described above. The average time to retrieve a record is, again, proportional to n/2 where n is the number of records in the file [Tremblay, 1975, 213]. This time, a record not in the file is determined sooner, when a key is found that exceeds the

value of the search key (assuming the file is assorted in ascending order.)

4. Blocking. Blocking may be more suited to a sorted file. In unsorted file, since records are randomly placed, there may be no correlation between the order in which they are stored and the order in which they are processed. In a sorted file, blocking helps reduce retrieval times since records are often processed by the order of the keys.

5. Binary Search. With a sorted file on a direct access device, another search method is possible: the binary search. In general terms, the file is divided into halves. The half determined to contain the record is then divided into halves, and the process is repeated until the record is found. This method takes advantage of random accessing and gives a tangible rationale for bothering to sort the data.

The binary search proceeds as follows. The middle record of the file is determined by taking the number of records in the file and dividing it by two. If the desired key matches the key of the middle record, the search is terminated, and the data is read. If the desired key falls below the middle key, the lower half of the file is divided in half and the process repeated till the desired record is found. If the desired key falls above the middle key, the upper half of the file is searched in the same fashion.

The average search time for the binary search method is proportional to $\log_2 n$, where n is the number of records in the file [Tremblay, 1975, 213]. To contrast average search times, for n = 8, the sequential search is proportional to 8 / 2, or 4. The binary search is proportional to the

$\log_2 8$, or 3. For n = 1024, the sequential search time is proportional to 512, and the binary search time is proportional to 10, an increase factor of 50. The amount of time to sort the file is well justified as the size of the file increases and as the records in the file are accessed more often.

D. INDEXED FILES

The final method to be discussed in this chapter is a progression from the previous method. The data file is still a sequential file that may or may not be sorted. The difference is that a second file is created that acts as an index into the data file.

1. Index Files. The records of the index file have two fields: the key from the data record and the address of the corresponding record. The index file is then searched for the key, and the address is used to retrieve the data from the data file. The index files are usually sorted, and a search method similar to the binary search is used to search them.

2. Advantages and Disadvantages. A major advantage of the index file manifests itself in files with large records. To sort such files, the data in each record has to be moved, and this increases sort time. Sorting the index file poses no such problem. Also, in some systems, all the data in the records must be read in to look at just the key, and this adds to the search time. This poses no problem when the search involves only the key and the address.

The two main disadvantages revolve around the extra disk space required by the index file, and the increased complexity of maintaining the file. The extra space is an obvious detraction from the problem, and

there is no way to eliminate this, though there are ways to minimize it. In some indexing systems, there are hierarchies of indexes, as will be discussed, and they add to the required disk space.

Maintaining the index files is the most notable drawback. When records are added or deleted, the data file can be handled in the straightforward manner of any sequential file. But each time the data file is altered, the index file must be rebuilt. In some implementations, changes to the data file are recorded in a special file, and they are made at the same time so the index file needs rebuilding a minimum number of times. The changes will not be noted by users until they are effected, which is not desirable. Management methods have been developed to make the changes apparent to the user without actually being made to the data file. These systems are very complex and considerations must be made for instances when two or more users issue conflicting commands on the same data (for example, when one person is modifying a field while another is deleting the record.)

3. Other Index Systems. Other, more complex systems have been developed to make indexing more flexible. Instead of an index file that is sequential, there may be a hierarchy of indexes which eventually lead to the actual addresses in the data file. In effect, the indexes are followed down the hierarchy as one would search through a sorted file with a binary search. Instead of determining which half of the remaining records to test, the tests result in selecting which index to continue to follow. If the key were lower than the key at the given point in the hierarchy, one would search one path while a second path would be searched if it were greater than the key. And if the key were matched, the path

would lead to the correct entry in the index file, or possibly directly into the data file.

Some systems use this idea, except they have a list of keys at each stage for comparison instead of a single key. Each key gives the highest key on the list at the next level. The list is searched until a key that matches or exceeds the desired key is found. That entry points to the next list of keys to search, or when the last level in the hierarchy is reached, the entry points to the desired entry in the data file. The number of levels in the hierarchy depends on the size of the data file and the number of keys in each list.

One system that uses this last idea stores the indexes within the data file. The records are stored in blocks of records. The indexes are stored in blocks of the same size, and the blocks are marked to differentiate them from the record blocks. Each block of records has a corresponding index block. Then there is a hierarchy of indexes built on these indexes. A management system for this index method simplifies maintenance by altering existing index blocks when the corresponding data blocks are altered. However, this system is expensive and may not be a feasible alternative for some applications.

The last method to be discussed will be detailed in the next chapter. The method works for sequential files that need not be sorted. It removes the need for a search algorithm by ideally going directly to the record desired.

III. HASH METHODS

After considering the above file storage and retrieval methods, which demonstrate a progression to increase speed and efficiency, we will now consider a method that does not increase the amount of disk space used. Some disadvantages do exist with this method, as will be discussed, but it will serve as the basis for an improved system. The method is called hashing.

A. DESCRIPTION

Generally speaking, the hash method works by passing the key of the desired record to some function, called a hash function. The return value of the hash function is the relative position of the desired record within the data file, or the record number. The hash function is used to both build the file and retrieve data from the file. The following is a more detailed description of the method. Note that references to addresses refer to the record number, from which the relative address is calculated by multiplying the record number by the length of a record.

To begin with, all the keys found in the file constitute a set. There exists a second set comprised of the addresses of the records. The hash function, denoted h(k) for key k, is a mapping of the set of keys to the set of addresses [Tremblay, 1975, 214]. As will be discussed, the addresses are determined after the function is established.

In practice, the mapping is seldom one-to-one. That is, more than one key can be mapped to the same address. Obtaining a proper function, one that is one-to-one, is a non-trivial task [Tremblay, 1975, 215].

Producing a proper function may mean producing a complex function which may take so much time to evaluate that the desired increase in speed of the method will be negated.

Once a function has been defined, the file can be built. First, a file is created with a specified number of blank records. There must be at least as many blank records as there are data records. The keys are passed to the hash function, and the data records are written to the appropriate position based on the return value.

B. CONSIDERATIONS IN DEFINING THE HASH FUNCTION

In informal terms, we wish to take the key, perform some operation(s) on it, and derive the address of the record in that matter. These operations serve as the hash function. To simplify matters, we will assume the records are fixed length.

One of the things considered in devising a hash function is that the set of keys seldom contains all possible keys. A way of operating on the keys is devised so that more than the actual keys can generate return values from the hash function. Ideally, this will distribute the return values over the actual keys so that there will be fewer instances of keys generating the same address. Such a function can be thought of as a "near" proper function.

The reason for a near proper function and unique hash values will be discussed shortly. Suffice it to say that when the hash function has been defined, it should evaluate quickly and will have a minimal number of redundant return values.

C. COLLISION RESOLUTIONS

What is the problem with the hash function mapping more than one key to the same address? As mentioned before, the function is determined before the addresses are. This is due to the way the file is created, by writing records to the address the hash function determines from its key. When the hash function maps a key to an address already taken, the first record will be overwritten by the second one making it impossible to retrieve the data of the first record. When the file is being accessed, it means a key evaluating to an address that has data from a different record. The occurrence of two keys mapping to the same address is called a collision.

How are collisions resolved? First, when the file is created, some method must be found to include all records in the file. The same method must be designed so that all records can be accessed once the file is constructed. The following paragraphs describe three methods used and will outline the method used in the example program.

1. Chaining. The first method is called chaining [Augustein, 1979, 539]. It requires that there be a blank record for each possible value returned by the hash function. When the hash function maps a key to a blank address, its data is written there. If the address is occupied, the data is appended to the file, and the new address is written to a special field in the record at the old address. When a third or successive key is mapped to the same address, the record is checked to see if another record has been added. If so, it gets the address and checks to see if another record has been added. It does this until it gets to the last record in the chain and adds the record as described above.

To extract a record from this file, the key is passed to the hash function. The record at that address is checked for a matching key. If the keys match, the data is read. Otherwise, the special field is read for the address of the next record in the chain. This process is repeated until the record is found or until the end of the chain is reached, at which time it is determined the record is not in the file.

2. Open Addressing. The second method is a general method called open addressing or rehashing [Augustein, 1979, 537]. When a collision occurs, the address is passed to a second function, called a rehash function, that returns a new address. If a collision occurs again, the process is repeated until a blank record is found (or until the desired record for retrieval is found). If the series of rehashes results in a blank record being addressed when attempting a retrieval, this signals that the desired record is not in the file.

Exactly what the rehash function does depends on the implementation. There is a concern with how long it takes to evaluate the rehash function and how many times it must be called to resolve collisions to maintain efficiency. The range of return values of the rehash function must match that of the hash function. Also, it is desirable that if there are blank records in the file, the rehash function will not get into a loop returning the same series of values repeatedly without encountering a blank record. If this is not the case, it is not guaranteed that all collisions will be resolved.

One difference in open addressing and chaining is the fact that all the file space must be allocated beforehand and is fixed. With chaining, since records are appended to the end of the file, there is no way of

controlling the size of the file. But chaining initially allocates enough disk space to handle the possible return values and not every record.

The question that arises is how much space must be allocated? If only enough space for all the records is allocated, collisions will increase as the file fills up, and the rehash function will end up being called excessively. If too much space is allocated, much of the space will be unused, and this defeats one of the aims of the solution. If the number of entries in the file is fixed, adding just 10% to the total size of the file will help [Augustein, 1979, 553]. For hash functions using a division method (discussed below), it has been found that if the total number of records the file can contain is a prime number, the distribution of the addresses over the set of keys will be good [Augustein, 1979, 553].

3. Linear Probing. The last collision resolution method is a specific case of open addressing, called linear probing [Augustein, 1979, 537]. The rehash function simply returns the next address following the initial address. In other words, when creating a file, when a collision occurs, the file is searched sequentially for the next occurring blank record. When retrieving records, the file is searched sequentially until the desired key is found. Should a blank record be encountered, as above, the search is terminated in failure.

A problem with linear probing occurs when the end of the file is reached. If this is not taken into account, the program will attempt to read beyond the end of the file. This will cause an error in some language implementations. In the rehash function, if the input value in the file is found to be the last record in the file, it will return the first address in the file. This has a wraparound effect, the file being

treated as if the last record were followed by the first record.

Another problem is that if too many collisions occur, the amount of time spent on the sequential search reduces the effectiveness of the method. Again, increasing the size of the file helps to avoid this problem.

D. EXAMPLE HASH FUNCTIONS

Now that the collision resolution methods have been described, we will consider some of the hash functions used. Some involve arithmetic operations on the key. Some operate on segments of the key. Others operate on bit patterns within the key.

1. The Division Method. The simplest method is the division method [Augustein, 1979, 540]. A numeric key is divided by the total number of records in the file. The remainder of this division is the return value. (This function is represented by h(k) = mod(k, n) and is described as taking k modulo n, where k is the key and n is the number of records in the file.) The range of the return values is 0 to n-1, which is the desired range. As mentioned above, if n is prime, the return values have a good distribution over the keys, improving effectiveness.

To give an example, assume there is a file with 100 records (n = 100.) The return values will be in the range 0 to 99, which will actually be just the last 2 digits of the key. If the key is 3432, h(3432) = mod(3432, 100) = 32. So the record associated with key 3432 will be 32 records into the file.

2. The Mid-Squares Method. A second method is called the mid-squares method [Tremblay, 1975, 219]. The numeric key is squared, and

the middle few digits are extracted. Since the return value is governed by the number of digits, the file size must be a power of 10. For example, if the middle 3 digits are used, the file size must be 1000 for the return values to fall into the appropriate range. If the middle bits are taken, and not digits, then the file size must be a power of 2, which gives more flexibility for file sizes. For example, if the middle 5 bits are used, the file size must be 32 for the return values to be valid (2 to the 5th power is 32).

To give an example, if the key is 3432 and three digits are required, the result is the middle 3 digits of 3432 * 3432, or 11,778,624, which would be 778. In a file of 1000 records, the record would be 778 records into the file.

3. The Folding Method. The folding method is another method of turning a key into a record number. The key is folded in on itself. That is, the key is broken into segments with a specific number of digits, and these segments are added together (or in the case of bits, exclusive-or'ed together) to give a result with the specific number of digits (or bits.) If the number of digits is exceeded due to a carry, it is folded again until the desired number of digits are left. As with the mid-squares method, since the return values depend on the number of digits or bits, the file size must be a power of 10 or 2.

An example of this method is as follows. Again, for the key 3432, break the key into two digit segments: h(3432) = 34 + 32 = 66. So in a file of 100 records, the record would be 66 records into the file.

4. Combining Methods. There are other functions that are available, but the above three should give an idea of the way record

numbers are calculated from keys. The other alternative is to combine functions to get a hybrid function. Because the keys used for the project programs were text, the folding method and division method were combined to give one of the hash functions used for comparison.

Since each character has an ASCII numeric representation in the C language, it is easy to access these numbers in character type variables. So, the folding took place by adding the ASCII codes of the characters together. Then to get the result in the appropriate range, this sum was divided by the size of the file, and the remainder was returned as the record number.

5. Weighted Sums. The second function used in the project programs was based on the influence of each character position in sorting the file. It was assumed that the keys would be distributed evenly enough that they could be distributed evenly within the file in some semblance of alphabetic ordering. Because the first letter of the key has the most influence on positioning words in alphabetic sequence, this position was given the strongest weight. Since the number of keys was small, the last characters had little or no influence on the position. The weights assigned to each position reflect this.

After tweaking the factors of the hash function and comparing the numbers of collisions, the final factors in sequence for the formula were as following: .75, .08, .06, .05, .03, .02, .009, .001

The size of the file was 50 records. Since the values of the ASCII code for the characters (lower case only) are 97 for 'a' through 122 for 'z', they were reduced by 97 to give their position in the alphabet. This position was scaled from a range of 25 (if 'a' = 0, 'z' = 25) to the

record numbers in the file (in the range of from 0 to 49) by multiplying the result by 49 / 25. For a key of only a's, the result is 0 while for a key of only z's, the result is 49. These are the extreme return values and the desired range for these values.

IV. NEURAL NETWORKS

What are neural networks? They are described as a group of computer models of how the brain might work [Bower, 1988, 344]. Labels include connectionist models, parallel distributed processing models, neuromorphic models [Lippmann, 1987, 4], and learning machines [Brown, 1987, 16].

A. HISTORY

The initial basis of neural computing theory arose from a paper in 1943 by McCulloch and Pitts discussing how neurons in the brain might function. Another paper by Donald O. Hebb described how neurons might learn. The actual concept of neural networks took form in a doctoral dissertation by Marvin Minsky called "Neural Networks and the Brain-Model Problem". The first learning machine was built by Minsky and Dean Edmonds out of tubes, clutches and a gyropilot. And the first landmark neural network model, the perceptron, was developed by Frank Rosenblatt in the 1950's [Klimansaukas, 1988, 347].

B. THE RETINA AND BASIC NEURAL NETWORK ELEMENTS

The model of the perceptron holds many of the key elements of the general organization of neural networks. Rosenblatt based the model on the retina of the eye. The retina contains several light sensors arranged in a matrix. These sensors are connected to processing elements, or demons, that serve the purpose of recognizing certain patterns [Klimansaukas, 1988, 348]. The output of the processing elementss goes through a type of threshold logic unit which fixes the value to a certain

level when a specific input occurs.

This description of the retina can be broken down into the distinct parts of neural networks. First of all, the sensors correspond to the inputs of the neural net. Based on the type of signal to process, the inputs can be digital (0 and 1, or -1 and 1 depending on the model) or continuous. Next comes the connections to the processing elements. In the case of the retina, they are established and trained by predetermined genetic patterns. In the case of neural networks, there are defined connection architectures and learning methods to set the connection weights. Then there is the processing element which is responsible for processing the input based on the connections. Some thresholding function can be found in most neural network models, forcing the inputs to be at certain levels to distinguish responses. Finally, there are the outputs which convert the inputs into values representing some interpretation of the inputs as seen by the processing element.

C. DEFINING NEURAL NETWORKS AND THEIR PARTS

The goal of designing neural network models is to establish some new means of solving problems that conventional computing methods cannot handle easily or well. Given a model based on biological processes and on processing that occurs in human thinking, it is hoped that computer science can move one step closer to achieving artificial intelligence in its truest sense. It is also hoped that other sciences can benefit by a better understanding of the human brain and thought processes.

1. Inputs. The first part of the neural network, the given factor, is the inputs. The neural network models are designed to interpret these

inputs intelligently and intelligibly. Inputs can take two forms. They can be continuous or digital in nature. Continuous data describe quantities where digital data usually describe qualities.

a. Continuous Data. The input data from the sensors in a retina, as described above, are continuous. The source is light striking the sensors causing certain chemical and electron activities. The activity varies as the intensity of the light varies and as the wavelength of the light varies.

In neural networks, there are many forms of continuous input. In target recognition, there are several types of sensors feeding input to the networks--radar signals, infrared readings, seismographic data and so on. In speech recognition, sound waves act as inputs after being converted to electric signals. In chemistry applications, there are different instruments for taking readings in chemical processes, such as pH, electric potentials and nuclear magnetic resonance patterns, which can serve as input data for neural networks.

b. Digital Data. Qualitative forms of data are presented to neural nets as digital data. For example, in vision processing, an image is divided into a large array of smaller images, each of which corresponds to an element in an array of bits. Bits may be set to 1 if most of the small image is covered and 0 if not. A series of inputs may represent qualifications for a loan, where each position represents a single quality and the state of the input determines whether the quality applies to the given individual. An input vector may be a binary representation of some code, such as the binary form of a number.

c. Operating on Inputs. As will be described later, there may be

multiple layers of processing elements within a given neural network. The result of these extra layers is the combination of the inputs in different ways. It has been demonstrated that if inputs undergo nonlinear transformations before being presented to the input layer of a neural network, the hidden layers may be removed as their functionality is still present [Pao, 1989, 60].

2. Processing Elements. The second element of the neural model is the processing element. Each has multiple inputs and a single output. Each input is assigned a connection weight that influences its contribution to the processing element. The dot product (or weighted sum) of the inputs and the weights (both of which can be thought of in vector form) gives the initial value of the processing element. One extra input, called a bias, is added to serve as a threshold for the processing element.

This threshold determines whether the processing element "fires" or not. That is, the bias value times the weight of its connection to the processing element is subtracted from the dot product of the inputs and their weights. This forces the weighted sum to be a certain value for the output to be positive (or in some cases, nonzero.) A bias' weights are sometimes different for different processing elements since their thresholds are usually different.

Finally, this output may have to go through a transfer function before going to the next layer. The transfer function sometimes serves to alter the data so it is in a format the next layer requires. Several functions are used for this purpose. They may be linear functions, that is, the sum is simply passed on or multiplied by a gain before being

passed on. The sigmoid, or s-shaped, function is a popular function. It is a continuous monotonic mapping of the input into a value between 0 and 1. It is based on the reciprocal of the constant e raised to the negative of the weighted sum [Klimansaukas, 1988, 161]. The hyperbolic tangent is similar to the sigmoid function, except that it maps into the range of -1 to 1. Each neural network model requires some form of these or other functions for transferring information between layers.

3. Connections. All the processing elements, once defined, are then interconnected is some manner to form a network. It is within these connections that the knowledge of the network is found. The way the processing elements are connected and their corresponding weights hold the knowledge.

a. Feedforward Designs. First of all, the different architectures or network designs determine the manner in which the networks converge or process the input. In a feed forward architecture, information passes from the inputs to the processing element layer(s) and finally on to the output layer, using the summation or dot products and transfer functions of the particular network models. The individual layers have no feedback connections from one layer to another or to itself [Klimansaukas, 1988, 8].

b. Feedback Connections. If there are feedback connections, the values in the layers oscillate or change states until such a time as the values stabilize or until some other convergence criterion is met. At this time, the information is passed to the output buffer[Klimansaukas, 1988, 8].

c. Resonating Networks. A third connection model involves two

layers which resonate as values change and interact until the network reaches a stable state [Klimansaukas, 1988, 481]. In some implementations, the network can accept input from either layer, and the corresponding values of the opposite layer will be produced.

d. Random Connections. Some models work best with connections made randomly between given layers. In the perceptron network model, the input to middle layers may be connected randomly while the middle to output layers are fully connected. The randomness focuses certain features to specific areas in the network as it is being trained so that those features can be identified and evaluated later [Klimansaukas, 1988, 351].

e. Middle Layers. When passing inputs directly to the output layer, the amount of knowledge represented by the connections may not be sufficient as was shown by Minski and Papert in their book Perceptrons. The manner certain processing element models function is by breaking the input patterns into different parts of a single space. Without a middle layer, the way the space is divided is not sufficient for some tasks. The middle layers are included so that inputs can be combined nonlinearly, in effect, increasing the knowledge within the connections. However, in some cases, if the middle layers are too large, the network will memorize the input patterns rather than learn the general features of the input. If the middle layers are too small, the network will require extra time for convergence. The size of the middle layer is up to the user [Caudill, June 1988, 54].

4. Weights. Once the connections are established, the network is trained as described in the next section. During this process, the connection weights are first set, and then as training progresses, the

weights are modified until the network displays the desired behavior. These weights constitute the second aspect of the neural networks' knowledge.

a. Magnitude. As one considers a single connection, two things become apparent. Weights have magnitude and they have an associated sign. The magnitude determines how much a processing element influences the processing element it is connected to [Klimansaukas, 1988, 6]. The closer the magnitude of the weight is to 0, the less influence it has. The closer it is to one, the more influence it has. A single processing element may have a strong influence on one element while virtually none on a different element.

b. Sign. The sign of a connection determines the manner the transmitting processing element influences the receiving processing element. If the sign is positive, the connection is said to be excitatory and contributes to the weighted sum of the receiving processing element. If the sign is negative, the connection is said to be inhibitory, and detracts from the weighted sum of the receiving processing element. This means that if an output requires a given input so it may be set on, the connection should be strong and the weight positive.

c. Connection and Weight Interaction. It is the interaction of the weights and the connections that store the knowledge the neural network has learned. This is not an easy point to enumerate as the interactions sometimes appear to perform by magic. If the effects of one input pattern are considered, the interaction might be seen by examining the states of the individual processing elements. Which connections influence which elements in what ways may be apparent under these circumstances. But when

trying to characterize the entire network in general, no method appears to be sufficient to the task. This becomes worse as the number of layers increases.

5. Learning. To separate neural networks from vector manipulation and to give meaning and significance to the connections and weights, learning methods must be considered. The main idea behind learning models is to let the network learn by example [Klimansaukas, 1988, 10]. Instead of setting down explicit rules to guide the network, the knowledge evolves as the learning proceeds. There are two ways learning is carried out. Unsupervised learning leaves most of the details to the network, and supervised learning requires some outside means of adjusting the weights [Klimansaukas, 1988, 10].

a. Unsupervised Learning. In unsupervised learning, only the input stimuli are presented to the network. The network organizes itself so that the connection weights allow each element to react strongly to a different set of stimuli or similar type of stimuli patterns. As this happens, the inputs are arranged in clusters based on these reactions.

b. Supervised Learning. In supervised learning the input stimuli are presented. The output of the network is then compared to the desired output, and the network is altered to move the output values closer to the desired values. There are basically three learning methods which are used in varying forms in different models. Hebbian learning consists of strengthening connection weights for a processing element if the input on that connection is high when the output is high. Delta rule learning is based on reducing the error between an input to a processing element and its desired output. Competitive learning involves modifying a connection

to a processing element only when it has a stronger response to a stimulus than other, competing processing elements [Klimansaukas, 1988, 11].

c. Learning Times. Due to the learning rules, some networks take longer than others to train. Sometimes it is due to the amount the weights are altered. If the amount is minute, it will take longer, though it should learn details more finely than networks that are trained with larger amounts. Also, since there are so many connections in some networks, the process takes time to fine tune the weights.

6. Association of Systems. Lastly, there are two types of systems, auto-associative and hetero-associative [Klimansaukas, 1988, 218]. The type of system desired determines what the network expects during learning.

a. Auto-Associative Networks. An auto-associative network is one in which the output should match the input. That is, one trains the network to reproduce the input at the output layer. The idea is to give noisy data and retrieve the original data from the network with the noise filtered out. However, the systems may have outputs different than the inputs to benefit from characteristics of this type of system. Only the input patterns are needed to train these networks.

b. Hetero-Associative Networks. In a hetero-associative network, the input and output are expected to be different. This type of model can be thought of as mapping one data set, the inputs, to another data set, the outputs. Training in this case requires both the input and the output patterns.

7. New Trends and Organizations. The field of neuro-computing is by no means exhausted. New models and modes of organization are being

devised and tested. As mentioned before, inputs are being altered to give more functionality to networks with fewer layers. These functional links, as they are described, allow higher order networks without the accompanying layers and connections [Pao, 1989, 60]. Neural nets are also being arranged in hierarchies with one level of networks feeding the inputs of succeeding levels of networks. The resulting hierarchies can be thought of as networks of networks [Caudill, June 1988, 53]. And learning rules are being modified, such as the recent development of an unsupervised form of Hebbian learning [Hinton, 1987, 1].

D. APPLICATIONS

A discussion of neural networks would not be complete without a discussion of their applications.

1. Image Processing. One area of interest is image processing. Neural network technologies are being developed in medical image processing, machine-vision, handwriting verification and other areas [Buffa, 1988, 48]. Supplying vision to computers allows automatic processing without the aid of human help and without the aid of intense programming and formula manipulation. Precision afforded by robotic systems and diagnostic systems now available will increase with this added source of data and verification.

2. Robotic Motion. Neural networks are being developed to guide robotic motion [Josin, 1988, 53]. The main problem with the robotic motion comes when redundant degrees of freedom in a robotic arm produce equations with no unique solution. Inverse transformation functions are required to handle this condition. They are made up of transcendental

functions which must be programmed, and they require extensive computation. Neural networks on the other hand are trained to perform these functions without the complexities normally involved.

3. Natural Language Processing. Natural language processing is being done with neural networks [Klimansaukas, 1988, 15]. A neural computing system has been designed to learn the past tense of English verbs. It begins at a child-like stage and gets to a point where it can synthesize new verb forms from incomplete data.

4. Combinatorial Problems. Neural networks have also shown promise in solving combinatorial problems, such as the traveling salesman problem [Klimansaukas, 1988, 19]. The goal is to find the shortest route around a circuit of cities a salesman is to travel. This type of problem also has application in routing phone calls. Neural networks have been designed to solve problems of this nature.

V. THE HAMMING NETWORK

A restatement of the problem is in order at this time. In the original terms, the problem centers around training neural networks to access data directly from files on secondary storage devices in a manner that makes the process more efficient with respect to time and disk usage. In terms of neural networks, the problem becomes one of presenting a pattern (the key) to a neural network which will behave like a hashing function by returning the record number for that key. Though not discussed in detail earlier, the type of neural network used will be a pattern classifying network.

A. PATTERN RECOGNITION

Pattern recognition involves taking some input and indicating what known pattern, or exemplar, the input most closely resembles [Klimansaukas, 1988, 13]. Part of the idea is to filter out noise that may be included with the input values. If the network used is auto-associative, the output will be the exemplar pattern the network has learned. If the network is hetero-associative, each output node represents one of the exemplars, and the value of each node will give an indication of how close the input pattern matches the represented exemplar.

For the purposes of the problem, the hetero-associative models offer the desired output. An auto-associative model will return data that will not be usable. The hetero-associative model will generate a vector that can be used to get the record number needed. This establishes a link

between hash functions and neural networks--a neural network, in the form of a pattern classifier, can be built to simulate the type of function required.

B. HASH FUNCTIONS AND PATTERN RECOGNITION

As we consider this link, one of the problems of the conventional hash functions involves addressing collisions. This results from obtaining the address after the function. If it were possible to create the file first, then construct the function so that the keys are mapped to the correct address, collisions would vanish. Since hashing methods work backwards, collisions do occur.

However, in a neural net implementation, we can begin with the address and derive the "function". Since neural networks learn from example, the addresses must be known, and therefore, the file exists without the problem of collisions. The exemplar patterns will be the keys. Each key will map to its own class, which will be its record number. Since the output nodes of the network correspond to the classes of the input patterns, they simply have to be numbered with the record number associated with that class.

C. CHARACTERISTICS OF PATTERN CLASSIFIERS

The many uses of pattern recognition as a concept manifest themselves in as many ways in the neural network implementations. Part of this stems from having a neural network model and fitting problems to its peculiarities. Some of the models will not meet the needs of the problem under consideration.

1. Counter-Propagation Networks. For example, in the counter-propagation network, the input consists of a vector whose elements are ordered digital or continuous data. The vector is ordered in that each piece of data, or each input node, corresponds to one of the characteristics of the objects to be classified [Klimansaukas, 1988, 491]. The values in the vector represent the quality or measure of the characteristic--say the color of the object or its velocity. This particular model allows more information in its inputs than is needed, and therefore, results in a complexity that is not necessary.

2. Hopfield Networks. In Lippmann's paper, he describes how a Hopfield network could be constructed for classifying the images of arabic numerals. The inputs represent the specific bits of the image's data [Lippmann, 1987, 9]. They can correspond to the binary representations in the ASCII code for some letter of the alphabet. The Hopfield network described classifies the image data by producing the bit pattern of the exemplar matched by the input. That is, it is designed as an auto-associative system. As described earlier, this outputs more data than needed and in an unusable form.

A particular problem with the Hopfield network was discovered from observing some sample programs. If the input patterns were noisy enough, two spurious output formats were generated. The first consisted of two parts of different exemplars being combined into a new pattern. The second output resulted in the failure of the network to converge. The network oscillated between two partial patterns. It is also important for convergence to be guaranteed for the network model to be acceptable.

3. Size Constraints. Finally, pattern recognition problems may

contain large numbers of exemplars and/or may require massive inputs. As the number of nodes increases, the interconnections increase drastically, and these increases are not small matters. The size of hidden layers can have similar effects. Depending on resources, the network may not be feasible or possible. A model that avoids this behavior and can still handle the problem at hand would be the most desirable model.

D. THE HAMMING NETWORK MODEL

It was with these considerations in mind that a Hamming network augmented with a MAXNET was chosen. The input layer accepts binary values with each node representing one bit of data. There are no hidden layers, reducing the hardware requirements of other model, as well as removing the need for a number of interconnections. Each output node represents one exemplar pattern. The exemplar patterns are loaded directly into the connection weights by a function described later. Finally, each input is connected to each output node with no feedback or interconnections within the Hamming network--that it, the Hamming network is a feedforward network.

The MAXNET, used to select the resulting class from the Hamming network's output, can be proven always to converge and to find the node with the maximum value. This node will indicate the class of the input.

1. Hamming Distances. The Hamming network is based on the measure of Hamming distances. Given two vectors of size n whose coordinates are binary, the Hamming distance is the number of coordinates where the two vectors differ [Tremblay, 1975, 365]. For example, the Hamming distance between (1, 0, 1, 1, 1) and (1, 1, 1, 0, 1) is 2. To match an input

pattern to a given set of exemplars, the Hamming distances are calculated, and the exemplar that gives the smallest value is the best match. The implementation of the network based on this idea will be detailed later.

2. Perceptrons. The basic building block of the Hamming network is the perceptron. Recall that the perceptron was devised in the mid-1950's by Frank Rosenblatt to model the retina of the eye.

The perceptron processing element bears more detailed description. Each element has a number of inputs, including a threshold input whose value is constant. The element computes a weighted sum of the inputs and subtracts the threshold value. This weighted sum is calculated, as described above, by taking the dot product of the input values and their associated weights. The value is passed through a function which limits the output to 1 or -1, by which the processing element splits the inputs into two classes [Klimansaukas, 1988, 351].

In the perceptron model, the initial weights and thresholds are random. The weights are modified by an error function as described above. But this is altered when perceptrons are used to construct a Hamming network.

When a network of perceptrons is formed such that the inputs of all the processing elements are common, this network can be used to solve the problem at hand. As will be shown, each perceptron will be calculating a function of the Hamming distance of the input vector from the exemplar stored by its connection weights. The individual processing elements constitute the output layer of the Hamming network.

3. The Hamming Network Organization. With this in mind, attention will now be turned to the Hamming network model. As described, the inputs

will be the key of a desired record. The connection weights represent the bit patterns of the keys for all existing data in the file. The weighted sum corresponds to a form of the Hamming distance. The sums are fed into a second network, the MAXNET, which sets it's nodes to 0 for all but the node with the highest value, which corresponds to the matched exemplar. The vector of the output nodes is converted to a record number within the data file. Finally, the record number is returned to the program as the return value of the hash function.

4. Connection Weights. In the perceptron network, the initial values of the connection weights are set randomly. The network then undergoes the learning in which the connection weights are altered so that the inputs will give the desired results. In the Hamming network, the correspondences between the inputs and output are known (each key, the input, will set a specific output node to a value higher than the others). Thus they can be set directly.

Since the perceptron model requires inputs of -1 or 1, the bits in the key are converted from 0, 1 to -1, 1. The weight of the connection from the ith input position to the jth processing element is set to the ith input value of the jth key. The weights will have the values -1 or 1.

Thus, if the bit pattern of a key is (0, 1, 0, 0, 1), it is changed to (-1, 1, -1, -1, 1). These become the weights from the inputs to the output node that corresponds to this key. Note that the Hamming distances give the number of positions where two patterns vary. The weights set in the manner described will give a value that is higher when more positions match. This is due to the form the input to the MAXNET requires, since it finds a maximum value.

5. Thresholds. The thresholds of all the processing elements are the same, n / 2, where n is the number of inputs. When the inputs are presented, the weighted sum will be the number of positions where the input matches the exemplar pattern of the given processing element minus the number of positions where the patterns vary. This threshold value serves to reduce the values of non-matching patterns so the MAXNET converges more quickly.

6. Example Weighted Sums. Assume the weights for a given processing element are set to (-1, 1, -1, -1, 1). The threshold value will be 5/2 or 2.5. Assume the input pattern (0, 1, 1, 0, 1) is presented to the network. The pattern is first converted to (-1, 1, 1, -1, 1). The weighted sum is:

$$(-1)(-1) + 1(1) + 1(-1) + (-1)(-1) + 1(1) = 3.$$

Subtracting the threshold leaves a value of 0.5. Assume that the original pattern is presented. The weighted sum is 5, and the final value is 2.5.

There will never be an output higher than the node corresponding to the input pattern if it is one of the exemplar patterns. In all cases, that value will be half the number of inputs.

7. Network Size. One of the reasons for choosing the Hamming network was the size. There is one processing element for each exemplar pattern. The number of connections is equal to the number of inputs times the number of processing elements. Some networks have connection schemes that increase the number of connections exponentially as nodes are added. This feature of the Hamming network makes it that much more attractive.

To compare the Hamming network with the Hopfield mentioned before, consider the systems with 100 inputs and 10 classes. The Hamming network

requires 1000 connections where the Hopfield network requires almost 10,000. With the Hamming network augmented by a MAXNET, only 100 more connections are required [Lippmann, 1987, 9].

8. The MAXNET. Once the Hamming network calculates its results, the outputs pass to a MAXNET network. The idea behind its function involves subtracting from a node's current value a fraction of the sum of the other nodes. The node with the highest value will consequently decrease at a slower rate than the other nodes. When it is the only nonzero node, the network has converged, and iterations cease.

The MAXNET has one node for each of the Hamming network outputs. Each node is connected to every node in the network. The weights of these connections are initialized as follows: if the connection links a node to itself, the weight is set to 0; if the connection links two distinct nodes, the weight is set to $-\epsilon$. ϵ is the fraction used in decrementing the values of the nodes. It can be proven that MAXNET will always converge and find the node with the maximum value when $\epsilon < 1 / M$, where M is the number of classes.

When the Hamming network passes its output to the MAXNET, the MAXNET begins an iterative process. The value of a node for iteration t+1 is:

$$\mu_j(t+1) = f_t\left(\mu_j(t) - \epsilon \sum_{k \neq j} \mu_k(t)\right)$$

$$0 \leq j,k \leq M-1$$

Examining the function, the current value of a processing element is its previous value minus a fraction of the sum of the other processing elements' values; then this value is passed through a threshold logic function. This is repeated until convergence, after which no more than

one node remains positive. This node represents the class of the input pattern.

9. Interpreting the Output Vector. There now exists a vector with (at most) a single non-zero value. The vector can be returned to the software, and it can be converted to a number. For example, if the first position is numbered zero, the record number is k-1 when the kth coordinate is non-zero. Or the vector can be passed to a decoding circuit that performs the same conversion. The relative address of the record is calculated by multiplying the record number by the record length so that the record can be read directly. It is also possible to build the circuit to take the output vector from the MAXNET and convert it directly to the relative address.

VI. THE IMPLEMENTATIONS

Three programs were written to compare the Hamming network implementation with software implementations. The following is a description of the programs. The conclusions drawn from the literature and the programs are given in the next chapter.

A. THE DATA

While doing research in another area, it was evident that textual keys presented an added disadvantage. Numeric keys can be compared in a straight forward manner. But text keys require comparisons of individual character positions which would increase the search time. And since the research demanded the fastest access method possible to effect the desired results, hash methods offered some promise.

The data is comprised of 25 PC-DOS commands and a brief description of their use and function. The command names are used as the keys. The maximum size of a description is 255 characters, so the record sizes were fixed at this value.

The programs serve as DOS help programs. The program prompts the user to enter a DOS command. The command is then passed through a hash function (or the simulated Hamming network), which returns the record number of the description in the data file. The arrangement of the data files is described with the particular implementations.

B. SOFTWARE HASH METHODS

The first two programs implemented two different funtions to get an

idea of the complexities and flexibility of hash methods. For simplicity's sake, linear probing was used to resolve collisions. After testing the programs, the blank files were 49 records in length, even though this is almost twice the number of actual records. The reasons will be discussed shortly.

1. The Folding and Division Functions. As described above, due to the nature of the keys, the most convenient method is a form of the folding method. In the C programming language, strings of character variables are formed by allocating consecutive spaces to hold the ASCII character codes of the letters. Each code can be accessed directly, so the folding method could be implemented by treating each character as a distinct segment of the key and then adding their ASCII codes together. This was the first step in the first hash function used.

The second step forced the result to be in the desired range. The division method was used on the sum to get a value from 0 to 48, as the records are numbered in the file. Recall that this would involve dividing the sum by 49 and returning the remainder. Also, recall that if the number of records in the file is prime, the distribution is better over the set of keys. Though a file of 50 would present slightly less opportunities for collisions, there were in fact more collisions (including those resulting from rehashing) with the data used.

In Appendix B, it can be seen a number of collisions occurred as the last records were added. Apparently, the keys clustered around an area from record number 37 to record number 45. There were a total of 10 collisions in this cluster, but only three of the records caused these collisions. And overall, there were only 14 collisions. This function

performed adequately.

2. A Weighted Sum. As mentioned before, a second method was devised based on ordering the keys alphabetically. The factors each character code were multiplied by were, in order, .75, .08, .06, .05, .03, .02, .009, .001. Note that the last letters had almost no influence on the result. These factors were arrived at by testing different values, and since few of the keys had 8 characters (the maximum lenght of a DOS command), the last characters did not matter much.

Though the factors were tested and altered several times, there were still more collisions with this method than with the previous one. This was mostly due to the fact that there were clusters of keys in the resulting file because the commands were alphabetically close. In Appendix C, it can be seen that there were three small clusters of records. In one case, there was a range from record number 5 to 10 that filled up causing later collisions. In the other two cases, the record numbers 25 and 31 were returned 4 and 3 times, respectively. Overall there were 21 collisions, 1.5 times more than for the previous method.

3. Creating the File. The original data was stored in a sequential file that was indexed for easier access. The programs would read in the key and address for one of the entries from the index file. The description was then located and read in from the data file. The key was passed to one of the two hash functions described above, and the record number returned was used to write the description to the new data file. As was mentioned, linear probing was used for collision resolution. In this manner, the data files to be used with both hash functions were built, and a program had to be written for each function to do this.

4. Reading Data from the File. Once the descriptions are stored in the new data file, a second program is used to access them. The program prompts the user for a DOS command. The entered command is passed to the hash function for the appropriate file, and the search is begun. If the key is not found at the record number returned, linear probing is again used to handle collisions. Once the key is found, the description is written to the screen, and the user is prompted for another command. If the key is not found, that is, if a blank key field is encountered in the search, the program reports this fact and again prompts the user for another command.

C. THE HAMMING NET IMPLEMENTATION

The Hamming network was implemented as described in the previous chapter. There were 8 characters in each key, with 8 bits per character, giving a total of 64 bits in each key and setting the number of inputs to the network to 64. There were 25 different commands in the data file, so there were 25 processing elements in the Hamming and MAXNET networks.

1. Initializing the Network. The index file mentioned above was used to load in the bit patterns of the keys (the ASCII codes of each letter) into the network. Since the bit patterns of the keys are represented internally by 0's and 1's, they were converted to the required -1's and 1's. These patterns, as described in the previous chapter, became the weights of the connections. Then the connections in the MAXNET were set as described above.

2. Running the Network. The user is prompted for a DOS command after the networks are initialized. The bit pattern of the command is

converted to -1's and 1's and presented to the network. The network processes the inputs as described. When the network converges, the outputs are checked so that the position of the nonzero output can be returned as the record number. The description is read from the file and displayed on the screen.

It was mentioned earlier that if a key was equally close to two of the exemplar patterns, the time for convergence increases. A feature was added to the program to force convergence when all the outputs are less than 0.00001, not just zero. Due to the way real numbers are represented in the C programming language, numbers become minute, and zero may require a long time to reach. After the network converges, the outputs are checked to see if there is a value over 0.00001. If so, there was a single exemplar matching the input closely, and it is displayed. If not, the outputs are checked for nonzero values (that will be less than 0.00001). There will be more than one closely matching exemplar and every associated description is read and displayed after a message stating the output was ambiguous.

VII. A COMPARISON

Looking at the final product, the Hamming network implementation offers distinct advantages over the software implementation of hash methods. These advantages will be discussed, as will be the disadvantages. Some of the following observations come from implementing and running both versions.

A. DISK USAGE

One advantage the Hamming network implementation has over the software implementations and the indexed file methods is that Hamming networks need no more disk space than that necessary to store the data. Because of the mechanism of linear probing, free space must be available for collision resolution. Chaining also requires a number of blank records to be allocated to start the file, but not all of them necessarily get used.

Disk space is a major concern due to costs of secondary storage devices. They have come down in cost over the past few years, but they still demand optimization to be cost effective. Access speeds are also improving as technology advances. However, improved access times increase the cost of the storage devices.

B. FILE FORMATS

In a similar vein, the Hamming network requires only a simple sequential file. Since no collisions will occur, no special considerations have to be made for extra records or for chaining. With

the conventional hash methods, as the file fills up, the gain in time is lost in collision resolution. So the need for free disk space in the data file must be met. The design of the file is greatly improved with the Hamming network.

C. SEARCH TIME

As the problem under consideration states, the time to access a record of data must be decreased to increase the efficiency of the computer's usage. The hash method improves access time by decreasing search time. There are two factors of the search time by which the Hamming network justifies its usage.

The first is the increase in speed due to a hardware implementation. With the hash function in hardware form, the Hamming network provides an immediate speed increase over the software implementation. It is not simply in going from machine code instructions to circuitry that this increase is found. The Hamming network is designed to perform many similar functions in parallel. Once the key reaches the inputs of the Hamming network, there is no number of instructions to count. The Hamming network is resolved in a single step because all the necessary operations occur simultaneously. And the MAXNET requires only a dozen or so iterations in many cases until it converges [Lippmann, 1987, 9].

The second factor is due to the way the Hamming network carries out the hashing. The file is defined beforehand, and the keys are made to map to the correct record number. There is no need for collision resolution as the network results in no collisions. Therefore, no time is spent in rehashing or in traversing chains of records to the correct position, and

one of the disadvantages of hashing methods is thus eliminated.

D. MAINTENANCE COSTS

There is no apparent advantage in either implementation with respect to maintenance costs. For the software implementation, the file needs to be reconstructed before the file fills to prevent collision resolutions from slowing down the system. More free space must be added, and the hash function has to be revised to return the new range of record numbers.

In the Hamming network implementation, records just need to be appended to the file. But the Hamming network requires altering. If it is initially established with extra output nodes and their connection weights are set to 0, all that needs to be done as records are added to the file is to set the weights of one of the extra processing elements according to the key of the new record. Otherwise the network must be rebuilt entirely. How long this would take depends on the actual hardware used to build the network and should be considered when designing the system.

E. FILTERING NOISE

One feature available with the Hamming network, due to the basis of its design, is noise filtering. The Hamming network uses Hamming distances to select its output. Since the output processing element with the maximum value is selected, an exact match does not need to be found. If a mistake is made in entering the key, the correct key may still be found. With text keys, different tenses or misspellings may still generate the correct data. With the conventional hash method, if the key

does not match exactly, there is no return data.

If the input matches a key exactly, the Hamming network is guaranteed to return the record number. However, if the input does not match a key exactly, it may have the same Hamming distance from two different keys. This will result in all the outputs being set to 0. Since the MAXNET reduces the values of each processing element by a fraction of the sums of the other elements, a point will be reached where the two are the only elements with non-zero values. The time it takes for them to reach zero will be slower than when this is not the case. Even when there are more than two, there are more values contributing to the amount each is decremented. This serves to speed up convergence.

VIII. CONCLUSION

Though the problem under consideration is an old one and many solutions can be devised, adding neural network technologies gives a new dimension to viewing this and other problems. The Hamming network implementation of a hash function offers a solution, decreases the access time, and at the same time gives a flexibility not otherwise considered in filtering noise.

There are numerous problems for which neural network applications are being designed. Many of the designs consider the neural networks alone as the solutions to the problems. But the number of problems where existing technologies are the basis seems to be lacking. Radical promises made in the 60's and 70's are failing to emerge due to the failure to find the right atoms to describe intelligence [Vaughan, 1988, 346]. Until then, the old methods should be allowed to serve this purpose as best they can.

Computer science can benefit from the results of this research, but one has to be satisfied with more modest advances than were promised. In this case, the combining of a simple neural net paradigm with a software oriented algorithm gives definite and observable gains. Other areas of computer science will benefit similarly if ways of blending the two concepts together are discovered. It is a matter of identifying which ones are most compatible.

Appendix A

Hamming Network Algorithm

APPENDIX A

Hamming Network Algorithm [Lippmann, 1987, 7]

Step 1. Assign Connection Weights and Offsets

In the Hamming network:

$$W_{ij} = X_i^j, \quad \theta_j = \frac{N}{2},$$
$$0 \le i \le N-1, \qquad 0 \le j \le M-1$$

In the MAXNET network:

$$t_{kl} = \begin{cases} 1, & k = l \\ -\epsilon, & k \ne l, \quad \epsilon < \frac{1}{M}, \end{cases}$$
$$0 \le k, l \le M-1$$

In these equations w_{ij} is the connection weight from input i to node j in the Hamming network, and θ is the threshold in that node. The connection weight from node k to node l in the MAXNET network is t_{kl}, and all thresholds in this subnet are zero. x^j_i is element i of exemplar j. Here and below N is the number of inputs and M is the number of exemplar patterns.

Step 2. Initialize with Unknown Input Pattern

$$\mu_j(0) = f_t\left(\sum_{i=0}^{N-1} w_{ij}x_i - \theta_j\right)$$
$$0 \le j \le M-1$$

In this equation $\mu_j(t)$ is the output of node j in the Hamming network at time t, x_i is element i of the input pattern, and f_t is the threshold logic nonlinearity. Here and below it is assumed that the maximum input to this

nonlinearity never causes the output to saturate.

Step 3. Iterate Until Convergence

$$\mu_j(t+1) = f_t\left(\mu_j(t) - \epsilon\sum_{k\neq j}\mu_k(t)\right)$$
$$0 \leq j,k \leq M-1$$

This process is repeated until convergence after which the output of at most one node remains positive.

Step 4. Repeat for New Input by Going to Step 2.

Appendix B

Collisions for Hash Method Using Hash Function 1

Appendix B

Collisions for Hash Method Using Hash Function 1

Key	Record	Collisions
assign	8	0
attrib	9	0
cd	3	0
chdir	32	0
cls	28	0
comp	39	0
copy	2	0
dir	25	0
erase	38	0
format	12	0
join	40	0
label	22	0
md	13	0
mkdir	45	0
more	43	0
path	37	0
print	18	0
prompt	37	4
rd	18	1
rmdir	3	1
rename	44	0
subst	22	1
type	9	1
ver	39	3
vol	43	3

Appendix C

Collisions for Hash Method Using Hash Function 2

Appendix C

Collisions for Hash Method Using Hash Function 2

Key	Record	Collisions
assign	4	0
attrib	5	0
cd	2	0
chdir	3	0
cls	5	1
comp	6	1
copy	7	1
dir	5	4
erase	8	2
format	10	1
join	15	0
label	15	1
md	17	0
mkdir	18	0
more	20	0
path	23	0
print	25	0
prompt	25	1
rd	24	0
rmdir	25	2
rename	25	3
subst	31	0
type	31	1
ver	31	2
vol	33	1

BIBLIOGRAPHY

Augustein, Moshe J. and Aaron M. Tenenbaum. data structures and pl/I programming. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.

Bower, Bruce. "The Brain in the Machine," Science News, vol 134 (November 26, 1988), 344-345.

Brown, Robert Jay. "AI: An Artificial Neural Network Experiment," Dr. Dobb's Journal of Software Tools, vol. 12, no.4 (April 1987), 16-27.

Buffa, Michael G. "Neural Network Technology Comes to Imaging," Advanced Imaging, November 1988, 47-51.

Caudill, Maureen. "Neural Networks Primer Part I," AI Expert, vol. 2, no. 12 (December 1987), 46-52.

Caudill, Maureen. "Neural Networks Primer Part III," AI Expert, vol. 3, no. 6 (June 1988), 53-59.

Caudill, Maureen. "Neural Networks Primer Part IV," AI Expert, vol. 3, no. 8 (August 1988), 61-67.

Caudill, Maureen. "Neural Networks Primer Part V," AI Expert, vol. 3, no. 11 (November 1988), 57-65.

Caudill, Maureen. "Neural Networks Primer Part VII," AI Expert, vol. 4, no. 5 (May 1989), 51-58.

Caudill, Maureen. "Neural Networks Primer Part VIII," AI Expert, vol. 4, no. 8 (August 1989), 61-67.

Caudill, Maureen. "Using Neural Nets: Part 1 Representing Knowledge," AI Expert, vol. 4, no. 12 (December 1989), 34-41.

Hinton, Geoffrey E. "Connectionis Learning Procedures," Technical Report CMU-CS-87-115 (version 2), December 1987, Carnegie-Mellon University, Pittsburg, PA.

Josin, Gary. "Integrating Neural Networks with Robots," AI Expert, vol. 3, no. 8 (August 1988), 50-58.

Klimansaukas, Casimir C. and John P. Guiver. NeuralWorks--An Introduction to Neural Computing, NeuralWorks User's Guide, Networks I, Networks II revision 2.00. Pitsburg, PA: NeuralWare, Inc., 1988.

Lippmann, Richard P. "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, April 1987, 4-21.

McClelland, J. L., et al. Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vols 1 and 2. Cambridge, MA: Bradford Books, 1986.

Minsky, M. and S. Papert. Perceptrons. Cambridge, MA: MIT Press, 1969.

Pao, Yoh-Han. "Function Link Nets: Removing Hidden Layers," AI Expert, vol. 4, no. 4 (April 1989), 60-68.

Schwenk, Ulrich. VSAM Primer and Reference. IBM World Trade Systems Centers, 1979.

Silber, Margaret L. "Computational Tool or Curiosity," MOSAIC, vol. 19, no. 2 (Summer 1988), 44-52.

Tremblay, J. P. and R. Manohar. Discrete Mathematical Structures with Applications to Computer Science. New York: McGraw-Hill Book Company, 1975.

Vaughan, Christopher. "Artificial Intelligence and Natural Confusion," Science News, vol. 134 (November 26, 1988), 346.