43054

Weather Prediction Using A Genetic Memory

David Rogers

P18

February 1990

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 90.6

NASA Cooperative Agreement Number NCC 2-408 and NCC 2-387

RIACS

Research Institute for Advanced Computer Science
An Institute of the Universities Space Research Association

Weather Prediction Using a Genetic Memory

David Rogers

Research Institute for Advanced Computer Science
MS 230-5, NASA Ames Research Center
Moffett Field, CA 94035

RIACS Technical Report 90.6

February 1990

Abstract. Kanaerva's sparse distributed memory (SDM) is an associative-memory model based on the mathematical properties of high-dimensional binary address spaces. Holland's genetic algorithms are a search technique for high-dimensional spaces inspired by evolutionary processes of DNA. "Genetic Memory" is a hybrid of the above two systems, in which the memory uses a genetic algorithm to dynamically reconfigure its physical storage locations to reflect correlations between the stored addresses and data. For example, when presented with raw weather station data the Genetic Memory discovers specific features in the weather data which correlate well with upcoming rain and reconfigures the memory to utilize this information effectively. This architecture is designed to maximize the ability of the system to scale-up to handle real-world problems.

This work was supported in part by Cooperative Agreements NCC 2-408 and NCC 2-387 from the National Aeronautics and Space Administration (NASA) to the Universities Space Research Association (USRA). Funding related to the Connection Machine was jointly provided by NASA and the Defense Advanced Research Projects Agency (DARPA).

WEATHER PREDICTION USING A GENETIC MEMORY

David Rogers
Research Institute for Advanced Computer Science
MS 230-5, NASA Ames Research Center
Moffett Field, CA 94035

1. INTRODUCTION

Nature eliminates surplus and compensates for deficiency.
- Lao-Tzu

The future success of neural networks depends on an ability to "scale-up" from small networks and low-dimensional toy problems to networks of thousands or millions of nodes and high-dimensional real-world problems. (The dimensionality of a problem refers to the number of variables needed to describe the problem domain.) Unless neural networks are shown to be scalable to real-world problems, they will likely remain restricted to a few specialized applications.

Scaling-up adds two types of computational demands to a system. First, there is a linear increase in computational demand proportional to the increased number of variables. Second, there is a greater, nonlinear increase in computational demand due to the number of interactions that can occur between the variables. This latter effect is primarily responsible for the difficulties encountered in scaling-up many systems. In general, it is difficult to scale-up a system unless it is specifically designed to function well in high-dimensional domains.

Two systems designed to function well in high-dimensional domains are Kanerva's sparse distributed memory (Kanerva, 1988) and Holland's genetic algorithms (Holland, 1986). I hypothesized that a hybrid of these two systems would preserve this ability to operate well in high-dimensional environments, and offer greater functionality than either individually. I call this hybrid Genetic Memory. To test its capabilities, I applied it to the problem of forecasting rain from local weather data.

Kanerva's sparse distributed memory (SDM) is an associative-memory model based on the mathematical properties of high-dimensional binary address spaces. It can be represented as a three-layer neural-network with an extremely large number of nodes (1,000,000+) in the middle layer. In its standard formulation, the connections between the input layer and the hidden layer (the input representation used by the system) are fixed, and learning is done by changing the values of the connections between the hidden layer and the output layer.

Holland's genetic algorithms are a search technique for high-dimensional spaces inspired by evolutionary processes of DNA. Members of a set of binary strings compete for the opportunity to recombine. Recombination is done by selecting two ''successful'' members of the population to be the parents. A new string is created by splicing together pieces of each parent. Finally, the new string is placed into the set, and some ''unsuccessful'' older string removed.

"Genetic Memory" is a hybrid of the above two systems. In this hybrid, a genetic algorithm is used to reconfigure the connections between the input layer and the hidden layer. The connections between the hidden layer and the output layer are changed using the standard method for a sparse distributed memory. The ''success'' of an input representation is determined by how well it reflects correlations between addresses and data, using my previously presented work on statistical prediction (Rogers, 1988). Thus, we have two separate learning algorithms in the two levels. The memory uses a genetic algorithm to dynamically reconfigure its input representation to better reflect correlations between collections of input variables and the stored data.

I applied this Genetic Memory architecture to the problem of predicting rain given only local weather features such as air pressure, cloud cover, month, temperature, etc. The weather data contained 15 features, sampled every 4-hours over a 20-year period on the Australian coast. I coded each state into a 256-bit address, and stored at that address a single bit which denoted whether it rained in the 4 hours following that weather state. I allowed the genetic algorithm to reconfigure the memory while it scanned the file of weather states.

The success of this procedure was measured in two ways. First, once the training was completed, the Genetic Memory was better at predicting rain than was the standard sparse distributed memory. Second, I had access to the input representations discovered by the Genetic Memory and could view the specific combinations of features that predicted rain. Thus, unlike many neural networks, the Genetic Memory allows the user to inspect the internal representations it discovers during training.

2. KANERVA'S SPARSE DISTRIBUTED MEMORY

Sparse distributed memory can be best illustrated as a variant of an algorithm commonly used to implement random-access memory. The structure of such a random-access memory is shown in figure 1. (The example given is for a RAM with 10-bit addresses and data.)

2.1 Structure of Random-Access Memory

The address at which reading or writing will be requested is called the *reference address*. The memory compares that address against the address of each of the memory locations. The location that matches the reference address is selected, which is denoted by a 1 in the select vector.

If writing to the memory, the *input data* is supplied. The input data is stored in the ten 1-bit data storage registers of the selected location.

If reading from the memory, the contents of the selected data registers are broadcast on the data bus and made available as the *output data*.

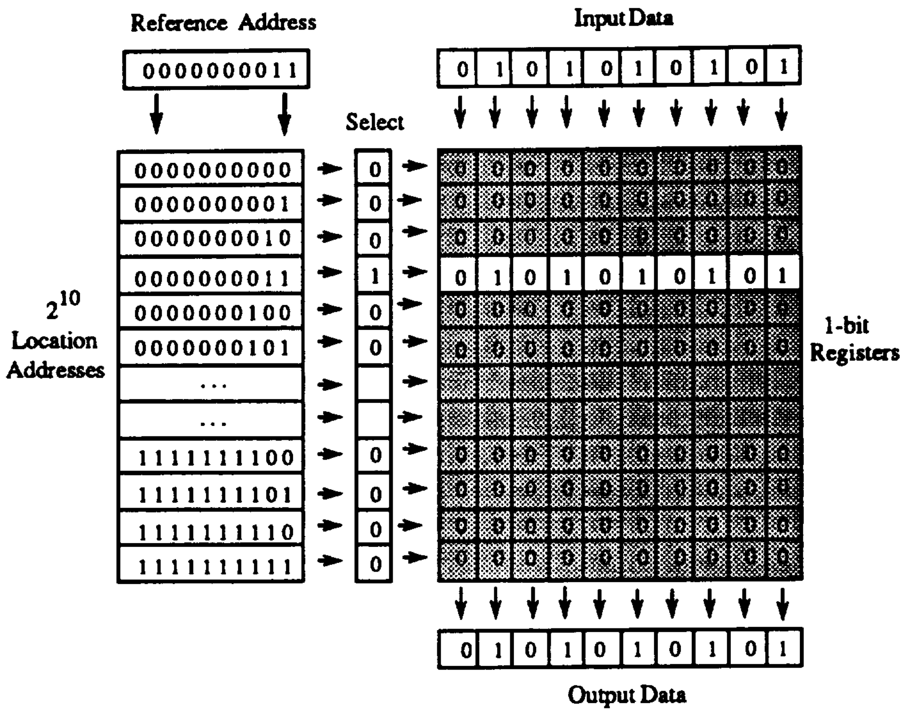


Figure 1: Structure of a simple random-access memory.

2.2 Structure of a Sparse Distributed Memory

Sparse distributed memory can be considered an extension of random-access memory. The structure of a sparse distributed memory is shown in figure 2. (The reader should note that a typical SDM often has 256-bits of address and data, and can have more than a thousand bits; the example shown uses only 10 bits for ease of illustration.)

In each of the three computations done by SDM (addressing, reading, and writing) there exists a major alteration to the RAM algorithm:

- Instead of looking for an exact match between the reference address and the location addresses, the memory calculates the Hamming distance between the reference address and each location address. Each distance is compared to a given radius; if it is less than or equal to that radius, then that location is selected. More than one location is usually selected in this process.

- The data registers are now counters instead of single-bit storage elements. These data counters are n-bits wide, including a sign bit. When writing to the selected locations, instead of overwriting, the memory increments the counter if the corresponding input data bit is a 1, and decrements the counter if the corresponding input data bit is a 0.

- When reading, the memory usually selects more than one location. The memory sums the contents of the selected locations columnwise, then thresholds each sum. Sums that are greater than or equal to zero correspond to output bits of 1, and sums that are less than zero correspond to output bits of 0.

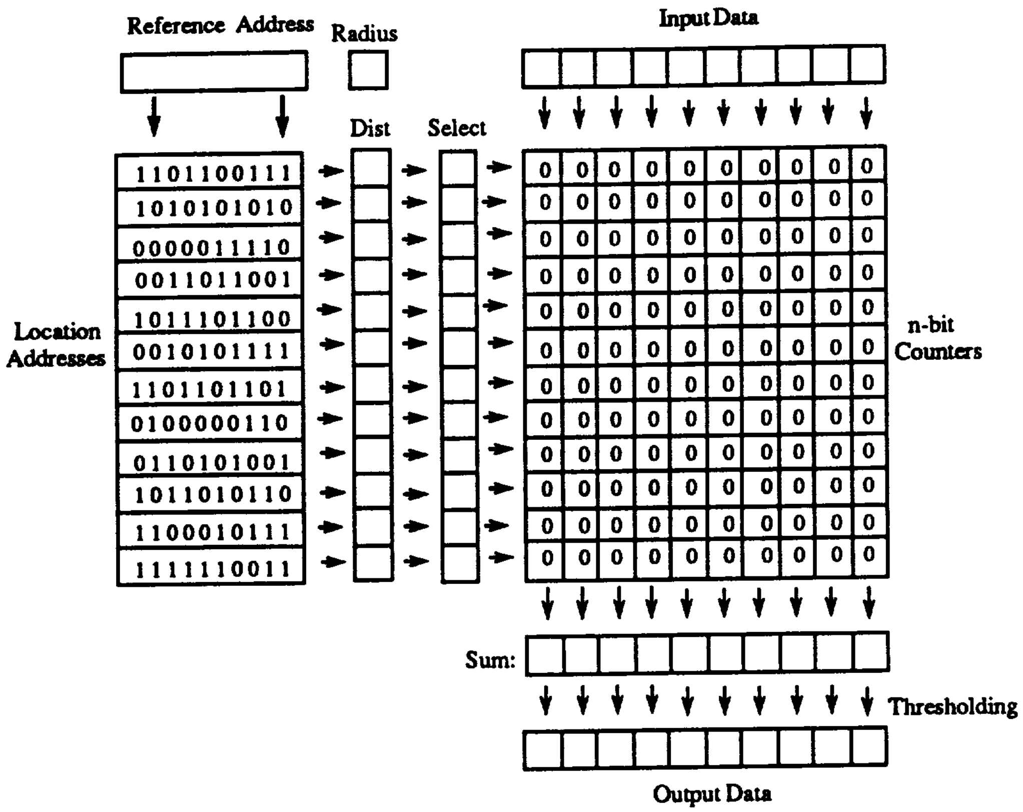


Figure 2: Structure of a sparse, distributed memory upon initialization. The location addresses have been assigned, and the data counters zeroed, but no reading or writing has been performed yet.

This example shows that a datum is *distributed* over the data counters of the selected locations when writing, and that the datum is reconstructed during reading by *averaging* the sums of these counters. However, depending on what additional data were written into some of the selected locations, and depending on how these data correlate with the original data, the reconstruction may contain noise.

2.3 Sparse Distributed Memory as a Neural Network

Though the RAM analogy is perhaps the clearest way to explain the structure of a sparse distributed memory, the SDM model can also be described as a fully-connected three-layer feed-forward neural network. A neural-network equivalent to sparse distributed memory is shown in figure 4.

The bottom layer is where the reference address is given; that is, there is one node in this layer for each bit of the reference address. These nodes are locked at either 1 or -1 depending on whether the corresponding bit of the reference address is 1 or 0.

The connections between the bottom layer and the nodes of the so-called *hidden* layer are either 1 or -1 in strength. These strengths are never changed, as they determine the address of the physical memory locations.

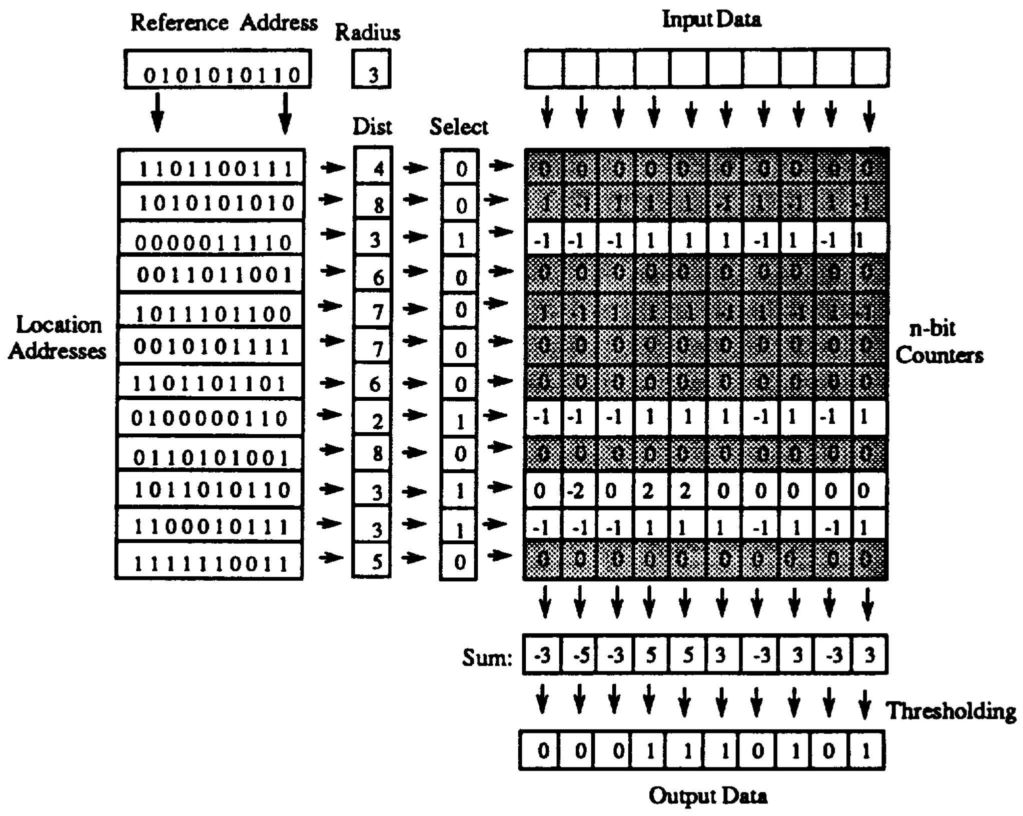


Figure 3: Reading from a sparse distributed memory after two write operations.

Each of the hidden-layer nodes corresponds to a memory location in the SDM model. A memory location is selected if the sum of its inputs (i.e., the dot product of the reference address and the location's weight vector) is greater than or equal to its threshold. This threshold corresponds to the radius in the SDM model, and the sum of the inputs is effectively taking the Hamming distance between the memory location's address and the reference address.

The top layer is where the output data appear. Each hidden-layer node is fully connected to the top-layer nodes. The data counters of a memory location are represented in the strengths of the connections between a hidden unit and the output nodes. This is the only part of the network that is plastic.

Reading the memory involves setting the values of the reference address and reading the output from the output nodes. Writing to the memory involves setting both the reference address *and* the data input nodes to the desired values; internal nodes that are active then add the value of each data input node (one or minus one) to its connection.

In this form, the SDM appears quite similar to other neural architectures. However, for an SDM the number of hidden-layer nodes is much larger than is commonly used for neural

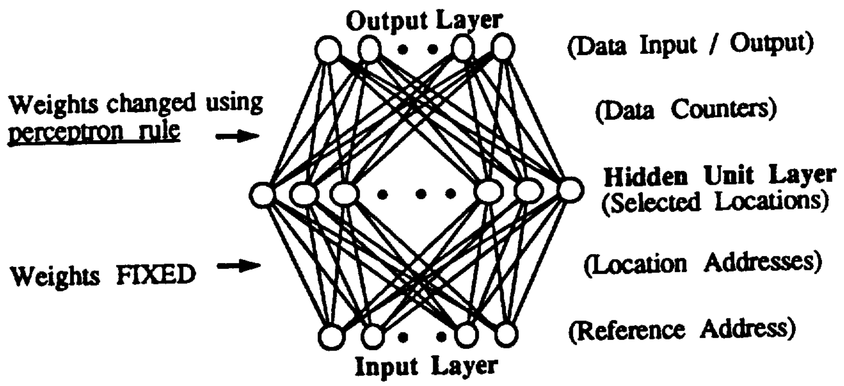


Figure 4: Neural-network representation of a sparse distributed memory. For a reasonable size memory, there might be 1,000 nodes in each the top and bottom layers and 1,000,000 nodes in the "hidden" layer.

networks. A reasonable size memory may have an address and data size of 1,000 bits, which would correspond to 1,000 nodes in each of the top and bottom layers. This is large, but not beyond the capabilities of current neural-network algorithms. However, if the memory has 1,000,000 memory locations, this would correspond to a network with 1,000,000 nodes in the hidden layer. It is unclear how standard algorithms, such as backpropagation, would perform with such a large number of units in the hidden layer.

3. HOLLAND'S GENETIC ALGORITHMS

The most extensive computation known has been conducted over the last billion years on a planet-wide scale: it is the evolution of life. The power of this computation is illustrated by the complexity and beauty of its crowning achievement, the human brain. What is the central computational technique that allowed the development of such an object, and of all nature, so quickly?

Darwin (1859) postulated that the process of evolution is based in part on the technique of random mutation. That view is reflected in the mildly disparaging modern belief that life is "just a product of chance". However, pure randomness can hardly account for the progress made in such a short period of time. A billion monkeys, typing one character a second for a billion years, would likely not stumble onto even the first line of "Hamlet". But if simple mutation is not sufficient to account for the grandeur of creation, what technique does evolution use which has the power to create all that we see? Or with the power to create us?

The answer to this question was anticipated by Mendel (1884), who observed that a offspring inherits a combination of distinct attributes from each parent, rather than a blend of both parent's attributes. The basis for this recombination of attributes was found later with the discovery of DNA, and is called genetic recombination or crossover. In crossover we find the key to the power of evolution. The crossover mechanism is the basis for a class of algorithms known as *genetic algorithms*. These algorithms constitute powerful tools for searching complex, high-dimensional domains.

```
1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 1 0 1 ...      (17)
1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 ...      (512)  ←—— Good selection
0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 ...      (222)
0 0 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 0 ...      (32)
1 0 1 1 1 0 1 1 0 0 0 0 1 0 1 1 0 1 ...      (87)
0 0 1 0 1 0 1 1 1 1 1 1 1 0 1 0 0 0 ...      (156)
1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 ...      (302)  ←—— Good selection
0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 ...      (3)
0 1 1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 ...      (1)    ←—— Bad selection
1 0 1 1 0 1 0 1 1 0 0 1 1 1 0 0 1 1 ...      (55)
1 1 0 0 0 1 0 1 1 1 0 1 0 1 0 1 0 1 ...      (216)
              ... etc ...                    (. . .)
Population of Binary strings           Fitness   scores
```

Figure 5: A genetic algorithm operates over a domain of binary strings. Each string has an assigned fitness score. New members are created by crossing-over two highly rated string and replacing a lowly-rated string.

The domain of a genetic algorithm is a *population of fixed-length binary strings* and a *fitness function*, which is a method for evaluating the fitness of each of the members. We use this fitness function to select two highly-ranked members for recombination, and one lowly-ranked member for replacement. (The selection may be done either *absolutely*, with the best and worst members always being selected, or *probabilisticly*, with the members being chosen proportional to their fitness scores.)

The member selected as *bad* is removed from the population. The two members selected as *good* are then recombined to create a new member to take its place in the population. In effect, the genetic algorithm is a search over a high-dimensional space for strings which are highly-rated by the fitness function.

The core of the algorithm is the crossover process. To crossover, we align the ends of the two good candidates and segment them at one or more crossover-points. We then create a new string by starting the transcription of bits at one of the parent strings, and switching the transcription to the other parent at the crossover-points. This new string is placed into the population, taking the place of the poorly-rated member.

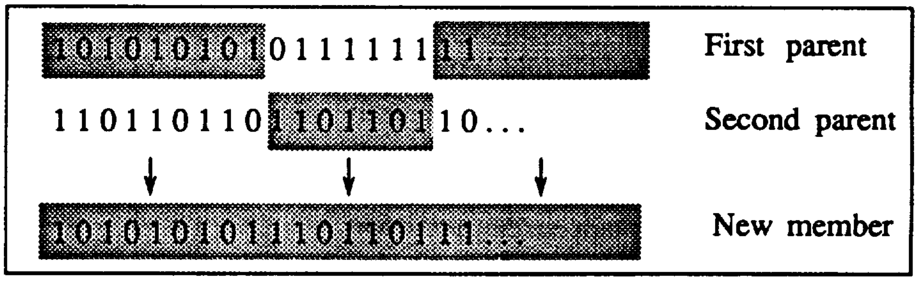


Figure 6: Crossover of two binary strings

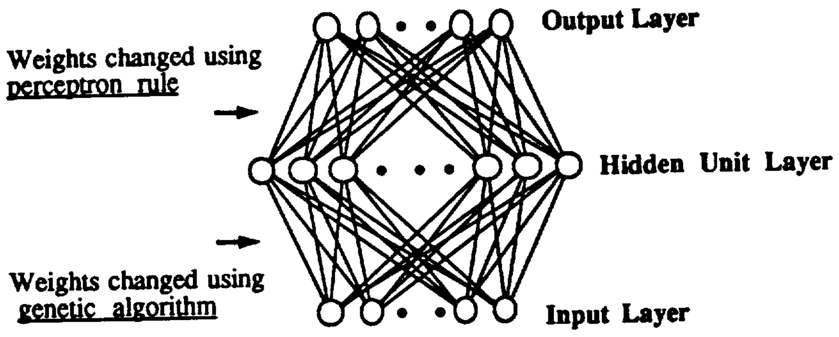

Output Layer
Weights changed using perceptron rule
Hidden Unit Layer
Weights changed using genetic algorithm
Input Layer

Figure 7: Structure of a Genetic Memory as a Neural Network

By running the genetic algorithm over the population many times, the population "evolves" towards members which are rated more fit by our fitness function.

Holland has a mathematical proof that genetic algorithms based on the crossover procedure are an extremely efficient method for searching a high-dimensional space.

4. GENETIC MEMORY

Genetic Memory is a hybrid of Kanerva's sparse distributed memory and Holland's genetic algorithms. In this hybrid, the location addresses of the SDM are not held constant; rather, a genetic algorithm is used to move them to more advantageous positions in the address space. In other words, location addresses are the population of binary strings that serve as the domain of the genetic algorithm.

If we view SDM as a neural net, the Genetic Memory uses a genetic algorithm to change the weights in the connections between the input layer and the hidden unit layer, while the connections between the hidden unit layer and the output layer are changed using the standard method for a SDM.

The role of the fitness function is filled by a value that is derived from the data counters; this value is a measure of the *statistical predictiveness* of the memory locations towards that bit in the data (Rogers 1989). The data counter value is a measure of the correlation between the selection of a location and the occurrence of a given bit value. Thus, we can use the data counters to judge the fitness, i.e., the predictiveness, of each memory location. Highly-predictive locations are recombined using crossover; the newly-created location address is given to a location which is relatively unpredictive.

The Genetic Memory is operated by presenting the memory with data samples; after a number of samples are seen (~10), it ranks the locations by their predictiveness. Two highly-ranked locations are selected as parents, and one lowly-ranked location is selected for replacement. Genetic crossover is performed on the parents, and the newly-created address replaces the address of the lowly-ranked location. Any data in the lowly-rated location are cleared. More data samples are then presented to the memory and the process is repeated.

Most other work which combined neural networks and genetic algorithms kept *multiple* networks (Davis 1987); the genetic algorithm was used to recombine the more successful of these networks to create new entire networks. In a Genetic Memory there is a *single* network with different algorithms changing the weights in different layers. Thus, a Genetic Memory incorporates the genetic algorithm directly into the operation of a single network.

Many of these systems require the networks to be presented with the full set of data samples, often a large number of times, before the genetic algorithm is executed. In a Genetic Memory the genetic algorithm operates as the data is being presented to the memory. Thus, the Genetic Memory can reconfigure itself without having access to the entire sample data set.

5. AUSTRALIAN WEATHER DATA

Weather data was collected at a single site on the Australian coast. Samples were taken every 4 hours for 25 years, resulting in a data file containing over 58,000 weather samples.

The data file contained 15 distinct features: station number, year, month, day of the month, time of day, pressure, dry bulb temperature, wet bulb temperature, dew point, wind speed, wind direction, cloud cover, present weather code, past weather code, and whether it rained in the past four hours.

For this work, I coded each weather sample into a 256-bit word. Each weather sample was coded into a 256-bit binary address, giving each feature a 16-bit field in that address. The feature values were coarse-coded into a simple thermometer-style code. For example, the following is the code used for the month field:

JAN:	1111111100000000	JUL:	1000000001111111
FEB:	0111111111000000	AUG:	1100000000111111
MAR:	0011111111100000	SEP:	1111000000011111
APR:	0000111111110000	OCT:	1111100000001111
MAY:	0000011111111000	NOV:	1111110000000011
JUN:	0000001111111110	DEC:	1111111000000001

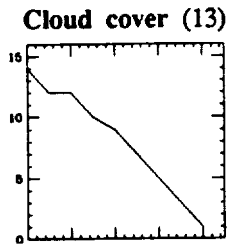
Each of the 15 fields is coded into its 16-bit representation, and the results concatenated into a 256-bit address that represents the current weather state. For example, here is the weather state for 1 January, 1961, at midnight:

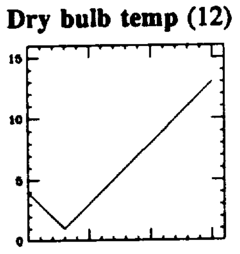
```
1111111100000000 1111111100000000 1111111100000000 1111111100000000
1111111100000000 0111111110000000 0001111111110000 0000011111111100
0000000111111110 1110000000011111 0111111111000000 0000001111111110
1111111110000000 0111111110000000 0000000000000000 0000000000000000
```

(Note that while the data samples must be of this form, the location addresses in the memory are not so restricted: 16-bit fields which do not represent any specific weather feature value are possible. Indeed, a Genetic Memory begins with completely random bit patterns in its location addresses. The analysis of location addresses after training will be discussed in depth in the experimental section.)

To train the memory, we present it with each weather state in turn. The memory is *not* shown the data a multiple number of times. For each state, the memory is addressed with the 256-bit address which represents it. "0" is written to the memory if it does not rain in the next four hours, and "1" if it does.

After the memory has seen a given number of weather samples, the genetic algorithm is performed to replace a poorly-predictive location with a new address created from two [illegible] The procedure is continued until the memory has seen 50,000 weather





6.1 Experiment 1: Features important in predicting rain

When the training is completed, we can analyze the structure of memory locations which performed well to discover *which features* they found most discriminatory and *which values* of those features were preferred. For example, here is a memory location which was rated highly-fit for predicting rain after training. (The 16-bit field corresponding to *pressure* is underlined.)

```
1101001100000011 1111011110101011 0111111100010000 1100000011011010
0100110011111011 1111110000000011 0111111011000000 0011101101100110
0000001011110110 0110000001000010 0001001110110100 0100000111111111
0000000111111110 0000000011111111 0011011111111111 0100110000001000
```

By measuring the distance between a given 16-bit field and all possible values for that field, we can discover *which values* of the feature are most desired. (Closer in hamming distance is better.) The absolute range of values is the *sensitivity* of the location to changes along that feature dimension. Figure 8 shows an analysis of the 16-bit field for month in the given memory location:

```
Location's 16-bit field
for month:0111111100010000

Values for months     Distance
JAN:  1111111100000000    2
FEB:  0111111110000000    2
MAR:  0011111111000000    4
APR:  0000111111110000    6
        ... etc ...
```

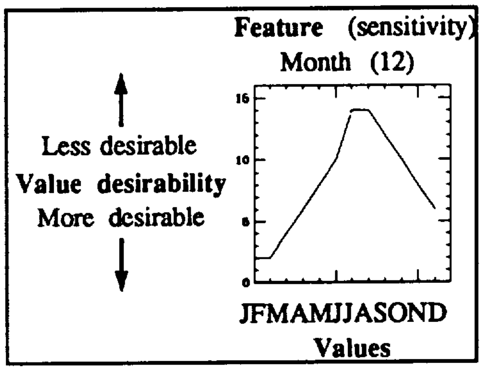


Figure 8: Analyzing a location field

In this case, the location finds January and February the most desirable months for rain, and July and August the least desirable months.

The relative sensitivity towards different features measures which features are most important in making the prediction of rain. In this case, we have a change of distance of 12 bits, which makes this location very sensitive to the value of the month.

We can estimate which features are the most important in predicting rain by looking at the relative sensitivity of the different fields in the location to changes in their feature. The following graphs show the most sensitive features of the previously shown memory location towards predicting rain; that is, the location is very sensitive to the combination of all these fields with the proper values.

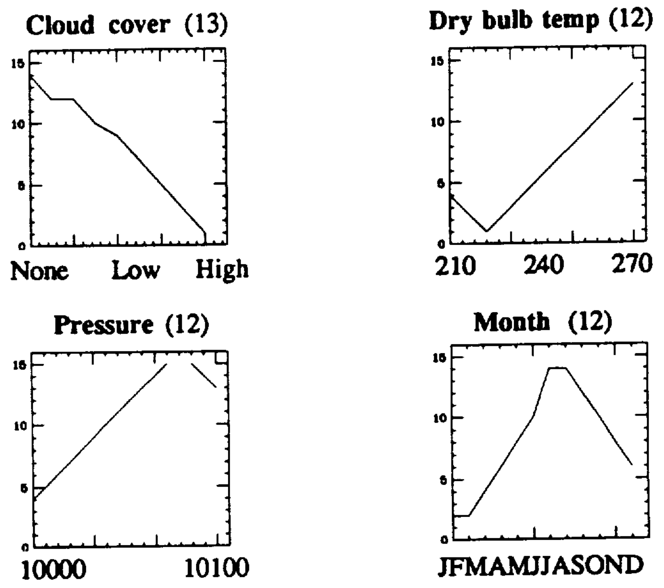


Figure 9: The four most sensitive features

The "most preferred values" of these fields are the *minima* of these graphs. For example, this location greatly prefers January and February over June and July. The other preferences of this location are for low pressure, high cloud cover, and low temperature. Surprisingly, whether it rained in the *last* four hours is not one of the most important features for this location.

We can also look some of the least sensitive features. The following graphs show the least sensitive features of the memory location towards predicting rain; that is, the location is relatively insensitive to the values of these features.

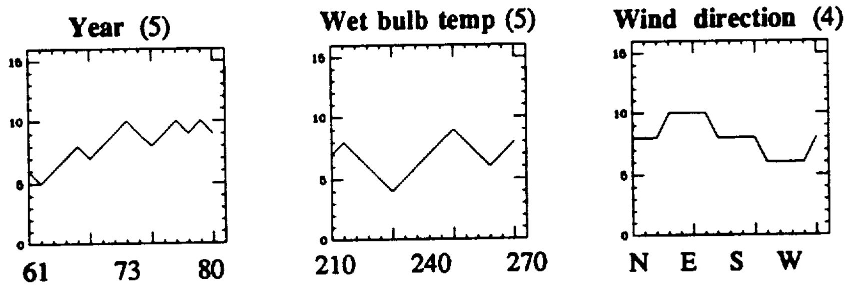


Figure 10: The three least sensitive features

This set contains some fields that one *would* expect to be relatively unimportant, such as year. Fields such as wind direction is unimportant to this location, but interestingly other highly-rated locations find it to be very useful in other regions of the weather space.

6.2 Experiment 2: Discovery of feature changes that predict rain

The previous experiment used the information in a single weather state to predict rain. However, for some fields, it may not be the current value that is important, but the change in that value since the last reading. I wanted to know whether the Genetic Memory would be able to discover such feature combinations.

In the previous experiment, I presented the memory with 256-bit addresses which represented the current weather state. In this experiment, I presented the memory with 512-bit addresses which were a concatenation of the current weather state and the weather state from 4 hours ago. The memory was shown 50,000 of these data samples. After training, a small number of locations were rated as much better than average at predicting rain.

Of special interest to me was the pressure feature, since I knew that falling pressure was a useful harbinger of upcoming rain. Each memory location address now contained two 16-bit fields which represented pressure; the first was for the preferred current pressure, and the second was for the pressure 4 hours ago. Figure 11 shows the value of these fields in one of the memory locations which was better at predicting rain. (Remember, the preferred values of the fields are the *minima* of the graphs.)

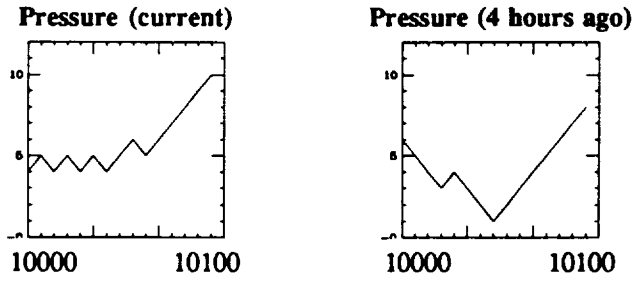


Figure 11: Pressure sensitivity in a predictive memory locations

This location prefers the current pressure to be relatively low, which is expected. However, it also prefers the pressure 4 hours ago to be ***higher than the current pressure,*** that is, for the pressure to be falling. Thus, this memory location has discovered that falling pressure is a useful predictor for rain.

This experiment shows that the Genetic Memory can be presented with values that are changing over time, and use information in those changes to add to the predictiveness of the memory. It should be noted that the feature combinations I noted were ones that confirmed my limited intuitions on what features predict rain. A more detailed analysis of successful memory locations by an expert in meteorology would likely show the locations to also contain more subtle but equally important combinations that I missed in my cursory extraction of "interesting" feature combinations.

7. COMPARISON WITH DAVIS' METHOD

Davis' algorithm has been shown to be a powerful new method for augmenting the power of a backpropagation-based system. The following is an attempt to contrast our approaches, without denigrating the importance his groundbreaking work. The reader is referred to his book for detailed information about his approach (Davis, 1987).

It is difficult to directly compare the performance of these techniques given the preliminary nature of the experiments done with Genetic Memory. However, it is possible to compare architectural features of the systems and estimate the relative strengths and weaknesses.

- **Backpropagation vs. Associative Memories:** Davis' approach relies on the performance of the backpropagation algorithm for the central learning cycle of the system. Associative memories have a far quicker learning cycle than backpropagation networks, and have been shown to have preferential characteristics after training in some domains. A system based on an associative memory may share these advantages over a system based on backpropagation.

- **Scalability:** Many issues concerning the scalability of backpropagation networks remain unresolved. It is not simple to build backpropagation networks of thousands or hundreds of thousands of units. In contrast, Kanerva's Sparse Distributed Memory is specifically designed for such massive construction; one implementation on the Connection Machine can contain 1,000,000 hidden units. The Genetic Memory shares this property.

- **Unity:** Davis' algorithm has two levels of processing. The first level consists of standard backpropagation networks, and the second a meta-level which manipulates these networks. The Genetic Memory has incorporated both algorithms into a single network; both algorithms are operating simultaneously.

My intuition is that *different algorithms may be best suited for the different layers of a neural network*. Layers with a large fan-out (such as the input layer to the layer of hidden units) may be best driven by an algorithm suited to high-dimensional searching, such as genetic algorithms or a Kohonen-style self-organizing system. Layers with a large fan-in (such as the hidden-unit layer to the output layer) may be best driven by a hill-climbing algorithms such a backpropagation.

8. CONCLUSIONS

- Real-world problems are often "high-dimensional", that is, are described by large numbers of dependent variables. Algorithms must be <u>specifically designed to function well in such high-dimensional spaces</u>. Genetic Memory is such an algorithm .

- Genetic Memory, while sharing some features with Davis' approach, has fundamental differences that may make it <u>more appropriate to some problems</u> and <u>easier to scale</u> to extremely-large (> 100,000 node) systems.

- The incorporation of the genetic algorithm <u>improves the recall performance</u> of a standard associative memory.

- The structure of the Genetic Memory <u>allows the user to access the parameters discovered</u> by the genetic algorithm and used to assist in making the associations stored in the memory.

Acknowledgments

This work was supported in part by Cooperative Agreements NCC 2-408 and NCC 2-387 from the National Aeronautics and Space Administration (NASA) to the Universities Space Research Association (USRA). Funding related to the Connection Machine was jointly provided by NASA and the Defense Advanced Research Projects Agency (DARPA). All agencies involved were very helpful in promoting this work, for which I am grateful.

The entire RIACS staff and the SDM group has been supportive of my work. Bruno Olshausen was a vital sounding-board. Richard Sincovec supported my work faithfully although it must have appeared very removed from his far more practical world of computational fluid dynamics. Pentti Kanerva trusted my intuitions even when the payoff wasn't yet clear. And finally, thanks to Doug Brockman, who decided to wait for me.

References

Darwin, Charles, *The Origin of the Species*, (1859), paperback version: New York: Washington Square Press, (1963).

Davis, L., *Genetic algorithms and simulated annealing*, London, England: Pitman Publishing (1987).

Holland, J. H., *Adaptation in natural and artificial systems*, Ann Arbor: University of Michigan Press (1975).

Holland, J. H., "Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Machine learning, an artificial intelligence approach, Volume II*, R. J. Michalski, J. G. Carbonell, and T. M. Mitchell, eds. Los Altos, California: Morgan Kaufmann (1986).

Kanerva, Pentti., "Self-propagating Search: A Unified Theory of Memory," Center for the Study of Language and Information Report No. CSLI-84-7 (1984).

Kanerva, Pentti., *Sparse distributed memory*, Cambridge, Mass: MIT Press, 1988.

Rogers, David, "Using data-tagging to improve the performance of Kanerva's sparse distributed memory," Research Institute for Advanced Computer Science Technical Report 88.1, NASA Ames Research Center (1988a).

Rogers, David, "Kanerva's Sparse Distributed Memory: an Associative Memory Algorithm Well-Suited to the Connection Machine," *Int. J. High-Speed Comput.*, **2**, pp. 349-365 (1989).

Rogers, David, "Statistical Prediction with Kanerva's Sparse Distributed Memory," *Advances in Neural Information Processing Systems I*, San Mateo: Morgan-Kaufmann (1989).