USING NEURAL NETWORKS TO SOLVE CODING THEORY PROBLEMS

Gregory Francis and Keith Fuller

Whenever binary information is transmitted over a "noisy" channel, part of the message can be lost or misinterpreted. By adding redundancy to the transmissions, it is possible to reduce the effect of noise. Redundancy is introduced by converting information into an established code, sending the encoded information, and then decoding the information at the receiving end. The parity check used by computer communication systems is an example of an error detecting code. Some communication environments, however, such as the distances between planetary spacecraft and receiving stations on Earth or critical military transmissions, require codes that can detect more than one error and in many cases correct the errors.

Find a code which can account for a given amount of noise and is as efficient as possible is a difficult optimization problem. This paper uses the recently discovered optimization abilities of artificial neural networks (ANNs) to find good codes.

I. CODING THEORY. Coding theory is a broad field of which only a small part is discussed in this paper. (For an in depth look at coding theory, see van Lint 1982). For notational convenience, let $B = \{0, 1\}$ and let B^n be the set of all possible sequences of zeros and ones of length n. Each element of B^n may also be called an n-bit word. We define a metric on the set B^n.

For x, y elements of B^n, we define their *Hamming distance* as

$$H(x, y) := |\{i : x_i \neq y_i,\ 1 \leq i \leq n\}|,$$

where x_i represents the ith member of the sequence x and $|*|$ represents cardinality. In other words, the Hamming distance (or just distance) between x and y is the number of places where their values differ in the sequences. For example, $H(010, 111) = 2$ because the words differ in two places.

Hamming distance is used to precisely define a code. For some fixed n and d we define $C \subset B^n$ to be a *code of length n, distance d* if for every pair of distinct elements x, y of C,

$$H(x, y) \geq d.$$

The elements of C are called *codewords*. We refer to C as an (n, M, d) code, where M denotes the cardinality of C.

Codes are extremely important in information theory. As digital information is passed between computers through various media, noise can affect a signal. When this happens, a 0 is sometimes interpreted as a 1 and vice versa. Consequently, a word can be misread by the receiving computer. However, if both sender and receiver agree to use only the codewords in a code of distance $d > 1$, a single error in the transmission of a word can be detected and the computers can recognize that the word needs to be sent again. For $d \geq 3$, not only can errors be detected but if fewer than $d - 1$ errors occur, the codeword closest to the faulty received word can be safely recognized as the intended transmission.

The most familiar code is the parity check. This is an $(n + 1, 2^n, 2)$ code. It can detect the presence of an odd number of errors (namely one error) in the transmission of a word. To encode a given n-bit word x, we create an $n + 1$-bit word z by the inclusion of a designated parity bit p, given by

$$p := \left(f + \sum_{i=1}^{n} x_i\right) \bmod 2,$$

where f is zero or one, depending on convention. If $f = 0$, we say z has even parity, and if $f = 1$, z has odd parity. Both conventions are in frequent use.

For some telecommunication applications we need a more robust code. Consider the following $(7, 16, 3)$ code:

$$R := \left\{ \begin{array}{l} 0000011,\ 0001100,\ 0010110,\ 0011001, \\ 0100101,\ 0101010,\ 0110000,\ 0111111, \\ 1000000,\ 1001111,\ 1010101,\ 1011010, \\ 1100110,\ 1101001,\ 1110011,\ 1111100 \end{array} \right\}.$$

Because $|R| = 16$ the sender and receiver can agree on a unique word in R for each word in B^4. This code also has the nice property of symmetry; every word in R also has its complement in R. Not all codes are symmetric, and finding non-symmetric codes is usually more difficult than finding symmetric codes.

A fundamental problem in coding theory is to maximize the size of C for given length n and distance d. For fixed n and d, a code C is said to be *optimal* if $|C| \geq |D|$ where D is any other code. The above-mentioned code of length seven with sixteen words is one of several optimal codes. For codes of small n, it is fairly easy to find optimal codes, but for codes involving words of greater length it becomes increasingly difficult to pick optimal codes out of the large number of possible codes. As the length of words increases, so does the number of words because $|B^n| = 2^n$. Due to this exponential increase in the number of words, it becomes impractical to attempt to construct all possible codes and find the codes with the largest number of codewords.

Due to the computational difficulty of the problem, most good codes are found through methods other than direct construction of codes; group theory, for example, has provided many insights into the formation of good codes. However, many cases have not yet yielded to study. Although the amount of computation required for a brute force attack on the problem is usually too great for a computer, it is possible to develop methods requiring much less computation that can find very good, although not necessarily optimal codes. The following section shows how a type of ANN can be used to find good codes for given length and distance.

II. HOPFIELD NETWORKS. Artificial neural networks consist of many connected simple processors or "neurons." Each neuron produces an output signal which it sends to many other neurons. One of the simplest networks consists of neurons which multiply the incoming signals by a factor and then sum the products. This summation is then tested against a threshold and based upon that test the neuron produces a single binary digit as output. If V_i represents the output of neuron i and T_{ij} represents the multiplying term for the signal from neurons j to neuron i, each neuron obeys the threshold function

$$V_i := \begin{cases} 1 & \text{if } \sum_{j \neq i} T_{ij} V_j + I_i > U_i \\ 0 & \text{if } \sum_{j \neq i} T_{ij} V_j + I_i \leq U_i, \end{cases} \tag{1}$$

where I_i is some input to neuron i from outside the network and U_i is the threshold for neuron i.

Hopfield (1982) showed that a network of this type will always converge to a stable state whenever $T_{ij} = T_{ji}$ and $T_{ii} = 0$. To prove stable convergence, consider the following function:

$$E := -\frac{1}{2} \sum \sum_{i \neq j} T_{ij} V_j V_i - \sum_i I_i V_i + \sum_i U_i V_i. \tag{2}$$

Assuming a finite number of neurons, E can have only a finite number of possible values. If neurons are randomly chosen to be updated one at a time, it can be shown that any change in the output of a neuron will decrease E. In fact, the change of E due to the change in the output of any neuron V_i is given by

$$\Delta E = -\left[\sum_{j \neq i} T_{ij} V_j + I_i - U_i \right] \Delta V_i. \tag{3}$$

By the definition of a neuron's output in equation (1), ΔV_i is either 0, +1, or −1. If $\Delta V_i = -1$, by (1) the terms in brackets in equation (3) must also be negative and hence ΔE is negative. Likewise, if $\Delta V_i = +1$, the terms in brackets are positive and again ΔE is negative. Because E has a finite number of possible

values, it is bounded and eventually it cannot decrease any further. If after some time ΔE is always zero, equation (3) implies that $\Delta V_i = 0$ for all neurons, so the network is in a stable state.

To apply these ideas to coding theory we create 2^n neurons, one for each word in B^n. We set the weights between neurons by the following rule:

$$T_{ij} := \begin{cases} +1 & \text{if } H(i,j) \geq d \\ -J & \text{if } H(i,j) < d \\ 0 & \text{if } i = j, \end{cases} \tag{4}$$

where $-J$ is a negative number and d is the desired minimum distance between codewords. Clearly, $T_{ij} = T_{ji}$ and $T_{ii} = 0$, so the proof of convergence applies. I_i and U_i both equal zero for every neuron in equation (1). If the proper value of J is chosen, we can force the network to converge to a stable state where only neurons that represent codewords have outputs equal to one. If J is chosen to be greater than the cardinality of an optimal code, there cannot exist a stable state containing two words, not both part of the same code, which both have output one. Words which are both part of the same code support each other by adding positive values to the summation in (1). A word that is not part of the same code (i.e., is closer than d to an active word) receives a large negative addition in (1) and is less likely to have a total summation above threshold. Of course this subtraction goes both ways because $T_{ij} = T_{ji}$. The neurons can be thought of as competing for a position in a code by trying to keep active other neurons which are part of a common code, and trying to make inactive neurons which are not part of a common code. Networks of this type are called *cooperative-competitive* networks.

Although the cardinality of an optimal code is not always known, an upper limit is frequently available. Setting J equal to an upper limit of the code under investigation will still cause the network to converge to a code. However, there is never a guarantee that the network will converge to an optimal code. Although E in equation (2) is minimized when the network converges to an optimal code, there are often other stable states (i.e., non-optimal codes). It is quite possible that the network will decrease E to a *local* minimum rather than a global minimum. In that case, the network will find a code that is not optimal.

To aid the network's convergence to a global minimum we used ***simulated annealing*** as suggested by Hinton and Sejnowski (1986). This approach replaces the threshold in equation (1) with a probability. The probability that neuron i will have an output of one is

$$P(V_i = 1) := \frac{1}{1 + \exp[-(1/Q)\sum_{j \neq i} T_{ij} V_j]}, \tag{5}$$

where $Q > 0$ is called the *temperature* of the network. By starting with a large value of Q and gradually reducing the value, the network was much more successful at finding optimal codes than the original network.

A network using simulated annealing was modeled on a PC and later on a VAX 11/780. We were able to find optimal codes for various distances up to word length ten. (The cardinalities of optimal codes are known for these lengths.) For example, to find the $(8,20,3)$ code, which is optimal, we started with a temperature of six. Every time 256 neurons were checked, we decreased the temperature by using the function

$$Q(t+1) = \begin{cases} Q(t) - \frac{1}{n}\sqrt{1/Q(t)} & \text{if } Q(t) \geq 1.0 \\ .95\,Q(t) & \text{if } Q(t) < 1.0 \end{cases} \tag{6}$$

There is a lot of guesswork when using ANNs, and this function is a mix of several ideas. As our experience with the network and its convergence rates developed, we continually modified this function into its present form. It is very likely that there are other methods of decreasing the temperature which would provide faster and better convergence. Using this function, the network found an $(8,20,3)$ code after 35 iterations on some trials. It is important to note that during some trials the network found codes containing only 17, 18 or 19 words. Such codes are not optimal, but are stable states for the network. Networks designed to find codes of length greater than ten have not been well explored because of the substantial time required for the computer to finish its task. Faster computers could find good codes for longer lengths and the technique is well suited to parallel processing; however, we have yet to explore these other possibilities.

III. CONCLUSION. The use of ANNs for solving optimization problems is a small but important part of neural network research. For an introduction to neural networks literature, the reader should investigate Grossberg (1988) and Rumelhart and McClelland (1986).

Other problems can also be solved with neural networks. Blayne Carroll, a fellow student, was recently investigating intersecting hypergraphs. Given N people, he was interested in finding the largest set of teams consisting of $k < N$ people subject to the following constraints:

1) Every two teams must have at least $M \leq k$ people in common.
2) No person may be on every team.

Blayne created an algorithm that worked well for $M = 1$, but he had difficulties extending his algorithm for some values of M. However, one of us (GF) demonstrated that an ANN could be created that finds teams for general M. This ANN is very similar to the coding theory network described in this paper.

ANNs provide a flexible approach that can be advantageously applied to some problems. If standard methods are too slow, too complicated to implement, or if appropriate parallel hardware is available, ANNs may be worth considering.

REFERENCES

S. Grossberg, *Neural Networks and Natural Intelligence,* MIT Press, Cambridge, MA (1988).

G. Hinton & T. Sejnowski, Learning in Boltzman machines, *Parallel Distributed Processing,* MIT Press (1986).

J.J. Hopfield, Neuronal networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences,* **79**, 2554–2558 (1982).

J.J. Hopfield & D.W. Tank, "Neural" computation of decisions in optimization problems, *Biological Cybernetics,* **52**, 141–152 (1985).

D.E. Rumelhart & J.L. McClelland, (Eds.) *Parallel Distributed Processing* (Vol. I), MIT Press, Cambridge, MA (1986).

J. van Lint, *Introduction to Coding Theory,* New York: Springer-Verlag (1982).

Gregory Francis and Keith Fuller
Butler University

This paper was written under the direction of Professors Prem Sharma *and* James Fink.