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 $i$th member of the sequence $x$ and $|\cdot|$ 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 $C \subseteq B^n$ 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 $d \geq 1$, 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

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

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:
the terms in brackets are positive and again terms in brackets in equation (3) must also be negative and hence by the definition of a neuron's output in equation (1), and $T_{ij} = T_{ij}$.

The following section shows how a type of ANN can be used to find good codes for given length and distance. 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 neuron $j$ to neuron $i$, each neuron obeys the threshold function

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

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:

$$E = -\frac{1}{2} \sum_{i\neq j} T_{ij} V_i V_j - \sum_i V_i U_i V_i. \quad (2)$$

Assuming a finite number of neurons, $E$ can have only a finite number of possible values. If neurons are randomly chosen to be updated at one 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

$$\Delta E = - \sum_{j\neq i} T_{ij} V_j + I_i - U_i \Delta V_i. \quad (3)$$

By the definition of a neuron's output in equation (1), $\Delta V_i$ is either $0$, $+1$, or $-1$. If $\Delta V_i = +1$, by (1) the terms in brackets in equation (2) must also be negative and hence $\Delta E$ is negative. Likewise, if $\Delta V_i = +1$, the terms in brackets are positive and again $\Delta E$ 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 $\Delta E$ 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:

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

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 (1988). This approach replaces the threshold in equation (1) with a probability. The probability that neuron $i$ will have an output of one is

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

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 100. At each temperature, we set the weights $T_{ij}$ using equation (4). The network was run for 35 iterations. We set the weights 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. There 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).
FRANCIS AND FULLER

USING NEURAL NETWORKS TO SOLVE CODING THEORY PROBLEMS

values, it is bounded and eventually it cannot decrease any further. If after some time $\Delta E$ is always zero, equation (2) 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 two neurons, one for each word in $B^n$. We set the weights between neurons by the following rule:

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

(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

$$P(V_i = 1) := \frac{1}{1 + \exp(-JV_i)}$$

(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

$$Q(t+1) = \begin{cases} Q(t) - \frac{1}{10} \sqrt{Q(t)} & \text{if } Q(t) \geq 1.0 \\ 0.95 Q(t) & \text{if } Q(t) < 1.0 \end{cases}$$

(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 decrease the time to solution 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


Gregory Francis and Keith Fuller
Butler University

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

### WHEN POLYNOMIAL RINGS ARE PRINCIPAL IDEAL RINGS

Lillian J. Kinkade and Joyce K. Wagner

An important theorem in the study of polynomial rings is: If \( F \) is a field, then \( F[x] \) is a principal ideal ring \(^1\). We propose to determine the structure of \( R \) if \( R[x] \) is a principal ideal ring; in particular, we show that \( R[x] \) is a principal ideal ring if and only if \( R \cong R_1 \times R_2 \times \cdots \times R_n \) where each \( R_i \) is a field.

Much research has been devoted to determining when certain rings are principal ideal rings, and a nice treatment of this subject can be found in \(^2\); in particular, \(^3\), Theorem 18.10 generalizes our main theorem. We now give definitions and notation that will be used throughout the paper. A ring is said to be Noetherian if every ideal is finitely generated. A primary ring is a ring which, for any ideal \( I \), if \( ab \in I \), then \( a \in I \) or \( b \in I \) for some \( n \). A principal ideal ring is a ring in which all ideals can be generated by a single element. The radical of an ideal \( I \) is \( rad(I) = \{ b \in R : b^n \in I \text{ for some } n \} \). Note that \( rad(I) \) is an ideal of the ring. Finally, all rings in this paper are assumed to be commutative Noetherian rings with unity.

The following proposition gives us an idea of the structure of \( R \) if \( R[x] \) is a principal ideal ring.

**Proposition 1.** If \( R \) is a principal ring and \( R[x] \) is a principal ideal ring, then \( R \) is a field.

**Proof:** Assume \( R \) is not a field. Then there must be some nonzero element \( r \) of \( R \) that is not a unit. Now consider the ideal \( (r, x) \) in \( R[x] \). Since \( R[x] \) is a principal ideal ring, \( (r, x) = (p(x)) \) for some \( p(x) \) in \( R[x] \). From this we have that \( r \in (p(x)) \) implies \( r = q(x)p(x) \) for some \( q(x) \in R[x] \). Similarly, \( s(x)p(x) \) for some \( s(x) \in R[x] \). Performing the multiplication and setting the coefficients equal, we see that

\[
\begin{align*}
(1) & \quad r = q_0p_0 \\
(2) & \quad 0 = s_0p_0 \text{ and} \\
(3) & \quad 1 = s_0p_1 + p_0t
\end{align*}
\]

where the subscript \( i \) denotes the coefficient of \( x^i \). From (1), since \( r \neq 0 \), then \( p_0 \neq 0 \). Since \( R \) is a principal ring and from (2), we see that

\[
(4) \quad s_0^2 \neq 0
\]

for some \( n \in Z \). Now using (3), we have \( 1^m = (s_0p_1 + p_0t)\) \( m \). Expanding the right side by the binomial theorem and ignoring coefficients, we have that the 4th term in \( (s_0p_1 + p_0t)^m \). Since \( R \) is commutative, the 4th term will cancel as long as \( k \neq 0 \) and \( k \neq \) by (2). So we have \( 1 = (s_0p_1)^m + p_0^mt \). But the first term cancels by (4). So

\[
(5) \quad 1 = (p_0r_1)^m
\]

Also, \( p(x) \in (r, x) \), so \( p(x) = f(x)p_0 + q(x)2 \). Therefore \( p_0 \neq 0 \). Substituting this into (5) gives \( 1 = (f_0r_1)^m \) which contradicts the fact that \( r \) is not a unit. Thus \( R \) must be a field.

The next lemma shows that the homomorphic image of a principal ideal ring is a principal ideal ring.

**Lemma 2.** If \( R \) is a principal ideal ring and \( \varphi : R \to S \) is an onto ring homomorphism, then \( S \) is a principal ideal ring.

**Proof:** Let \( R \) be a principal ideal ring and \( \varphi : R \to S \) an onto ring homomorphism. If \( I \) is an ideal of \( S \), then \( \varphi^{-1}(I) \) is an ideal of \( R \). To see this, recall from group theory that \( \varphi^{-1}(I) \) is a subgroup of the additive group of \( R \). So let \( x \in \varphi^{-1}(I) \) and \( a \in R \). Now \( \varphi(ax) = \varphi(a)\varphi(x) \). Since \( \varphi(x) \in I, \varphi(\varphi(x)) \in I \). Thus \( ax \in \varphi^{-1}(I) \) is an ideal of \( R \). Since \( R \) is a principal ideal ring, \( \varphi^{-1}(I) = \langle r \rangle \) for some \( r \in R \). If we can show that \( I = \langle \varphi(r) \rangle \), then we shall have the desired result. \( \langle \varphi(r) \rangle \subseteq I \) is obvious since \( \varphi(r) \in I \). For \( I \subseteq \langle \varphi(r) \rangle \), let \( x \in I \). Since \( \varphi \) is onto, there is a \( y \in \varphi^{-1}(I) \) such that \( \varphi(y) = x \). Since \( y \in \varphi^{-1}(I), y = br \) for some \( b \in R \). Thus \( x = \varphi(y) = \varphi(br) = \varphi(b)\varphi(r) \in \langle \varphi(r) \rangle \). So \( I \subseteq \langle \varphi(r) \rangle \), and \( S \) is a principal ideal ring.

Our next major proposition will show how \( R \) can be broken into a product of primary rings. To do that, we need some preliminary results.

---

\(^1\) Theorem 47.3

\(^3\) [L. Kinkade and J. Wagner, "When polynomial rings are principal ideal rings," Proceedings of the National Academy of Sciences, 79, 2554–2558 (1982).]