Soft-Decision Decoding of a Family of Nonlinear Codes Using a Neural Network

Thesis by

Ruth A. Erlanson

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California
1991
(Submitted December 5, 1990)

Acknowledgments

My experience at Caltech has been a very positive one. There are many people I wish to thank for their contribution to making my years here so enriching and wonderful. My advisor Yaser Abu-Mostafa created a a nurturing environment for my research; he was a source of interesting problems and he provided me with the encouragement and support to study these problems. Thank you Yaser!

My entire thesis committee was a source of inspiration for me. Through classes, seminars and outside interactions Professors Robert McEliece, Carver Mead, Ed Posner, Rick Wilson and Yaser introduced me to many interesting subjects and associated problem-solving methods. Each of them was a valuable resource for my research and this thesis. I also wish to thank Professor Murray Black at George Mason University for his encouragement and enthusiasm while I was pursuing my undergraduate degree. Without his support, I might never have had the opportunity to study at Caltech.

Numerous friends and student colleagues contributed to my time at Caltech. Foremost is Mass, who is responsible (with LaTeX) for the form of this thesis, and who contributed in more ways than I can count to my happiness while at Caltech. Thanks to Supper Club – Rick, Shelly, Ryan, Jim, Shenda, Tom, and Karen for wonderful dinners, and special times together. Thanks to Michael, Steve, Nan, and Andy for good times on and off the bicycles. Also, extra thanks to Nan for proofreading this thesis and for running an infinite number of errands on my behalf. I wish to thank Amir and Mark for our many discussions, both technical and non-technical. Thanks to Andrea and Kayo for their special friendships.

Above all, I thank my family, Dad, Mom, Beth, Edie and Simon for their love and unfailing support.

Abstract

We demonstrate the use of a continuous Hopfield neural network as a K-Winner-Take-All (KWTA) network. We prove that, given an input of N real numbers, such a network will converge to a vector of K positive one components and $(N - K)$ negative one components, with the positive positions indicating the K largest input components. In addition, we show that the $\binom{N}{K}$ such vectors are the only stable states of the system.

One application of the KWTA network is the analog decoding of error-correcting codes. We prove that the KWTA network performs optimal decoding.

We consider decoders that are networks with nodes in overlapping, randomly placed KWTA constraints and discuss characteristics of the resulting codes.

We present two families of decoders constructed by overlapping KWTA constraints in a structured fashion on the nodes of a neural network. We analyze the performance of these decoders in terms of error rate, and discuss code minimum distance and information rate. We observe that these decoders perform near-optimal, soft-decision decoding on a class of nonlinear codes. We present a gain schedule that results in improved decoder performance in terms of error rate.

We present a general algorithm for determining the minimum distance of codes defined by the stable states of neural networks with nodes in overlapping KWTA constraints.

We consider the feasibility of embedding these neural network decoders in VLSI technologies and show that decoders of reasonable size could be implemented on a single integrated circuit. We also analyze the scaling of such implementations with decoder size and complexity.

Finally, we present an algorithm, based on the random coding theorem, to communicate an array of bits over a distributed communication network of simple processors connected by a common noisy bus.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis we will discuss several systems of distributed processors. The system that we will consider first, and in most depth, is the neural network.

Many neural network models have been proposed. Overviews of the subject area include [AR88] and [RM86]. The structure common to these models is a system composed of many simple processors that are extensively interconnected. Beyond this basic form, these systems can differ in many ways. Some neural networks employ feedback, others do not. Networks can use analog or digital inputs, outputs, and connection strengths between processors. Also, processors can compute in a continuous or discrete manner. The specific neural network that we study here is that proposed in [Hop84]. This network, which we will discuss in more detail in Chapter 2, is a continuous system with feedback.

Neural networks are currently being employed on a variety of problems. Two classes of problems often solved by neural networks are associative memory recall [Hop82], and minimization problems [HT85]. In this thesis, the specific application we consider is the decoding of error-correcting codes. Here we utilize the ability of the network to receive partial (noisy) input and converge to a (close) stable state. Thus, the neural network is given a noisy codeword and decodes this input by settling to a stable state codeword.

Other work that discusses relationships between neural systems and coding theory includes: [PH86], [CG87], [Sou89], [BB87], and [MTD88].

The general decoding problem for linear codes has been shown to be NP-complete

[BMvT78]. Because nonlinear codes are in general more difficult to specify and manipulate than linear codes, there is no reason to believe that this problem is any easier for nonlinear codes. In addition, constructing a soft-decision decoder, where noisy analog inputs are mapped to binary codewords without discarding any of the information in the analog input, is an unsolved problem for many known linear codes. Thus, because soft-decision decoding is not an easy problem, a neural network solution is interesting.

We begin in Chapter 2 by presenting a general introduction to the encoding and decoding of information and to the Hopfield neural network. We then present a neural network that performs as a K–Winners–Take–All (KWTA) network. We prove that the only stable states of this network are those strings with K components that are $+1$ (the winners) and $(N-K)$ components that are -1. Furthermore, we prove that this network can be used as a decoder for a code consisting of the stable states of the network, and that this network performs optimal decoding. We say that the KWTA network places a KWTA constraint on the N nodes in the network. Chapter 2 follows [MEAM89] and [EAM88]. The KWTA code has a large rate, but a small minimum distance. The error-correcting capabilities of a code are directly related to its minimum distance. Therefore, we are interested in the existence of neural network codes with larger minimum distances. To this end, we have studied decoders constructed with multiple KWTA constraints.

In Chapter 3, we introduce the concept of overlapping KWTA constraints applied to the nodes of a neural network. A given node may now be involved in several constraints. We look at the characteristics of codes obtained from networks on which random sets of KWTA constraints are placed. These studies suggest decoder parameters, such as constraint size, and provide us a basis for comparison with other decoders constructed from overlapping KWTA constraints.

In Chapter 4, we present a family of decoders that are obtained by placing regular (nonrandom) KWTA constraints on the nodes in the network. The placement of these constraints is inspired by the structure of the hypercube. We also detail a neural network gain schedule that resulted in improved decoder performance, in terms of error rate. We present the performance of several example codes and illustrate the near-optimal decoding performance of the corresponding networks. The minimum distance of these codes can be made arbitrarily large, and therefore, these codes have greater potential than the KWTA codes.

Chapter 5 introduces another family of decoders constructed by the regular application of KWTA constraints on the nodes of a neural network. These constraints are specified by the points on the lines of a combinatorial net. The performance of several example codes is presented and the corresponding decoders are found to compare well with optimal decoders. Section 5.3 presents a graphical method for determining the minimum distance of codes defined by the stable states of networks with nodes in overlapping KWTA constraints. We find that the structured placement of KWTA constraints on the nodes of a neural network leads to decoders that are more powerful than the decoders specified by random constraints.

The nodes of the decoders of Chapters 4 and 5 exhibit a high degree of interconnectivity. Thus, we consider the complexity of such networks. We analyze this complexity in terms of the integrated circuit area necessary to implement such a network. Using Thompson's grid model, Chapter 6 considers the feasibility of embedding the decoder networks of Chapters 4 and 5 in Very Large Scale Integration (VLSI) technologies. We present several nontrivial systems that could be implemented on an integrated circuit and discuss the scaling of such circuits with decoder size and complexity. An overview of Chapters 4, 5 and 6 is presented in [EAM91].

The second distributed system we studied was the distributed communication network. Chapter 7 presents an algorithm, based on the random coding algorithm, to communicate reliably over a network of N simple processors connected by a common bus in $\Theta(N \log\log N)$ communications.

Chapter 2

The K–Winners–Take–All Code

2.1 Introduction

Information sent over a channel, or in certain storage conditions, is subject to the addition of noise. To prevent loss of information, error-correcting codes are utilized. Retrieval of the information from encoded bits is accomplished using a decoder.

In this chapter we present some general concepts relating to encoding and decoding information. We then present a general description of Hopfield's network of graded-response neurons and of the behavior of this network. These discussions lead to the development of a neural network that can be used to decode a specific code. An analysis of this network's performance is given.

2.2 Encoding and Decoding

We start by offering a motivation for encoding information and we give some explanation of important code parameters. For more information [McE77] discusses coding and decoding in depth.

Often, information that is sent from a source to a destination is transmitted over a noisy channel. That is, with some probability, the data which the source sends and the data which the destination receives are different. A solution to this problem of information loss is to add redundant digits to the information digits before transmission. For example, a simple form of such redundant digits is the repetition

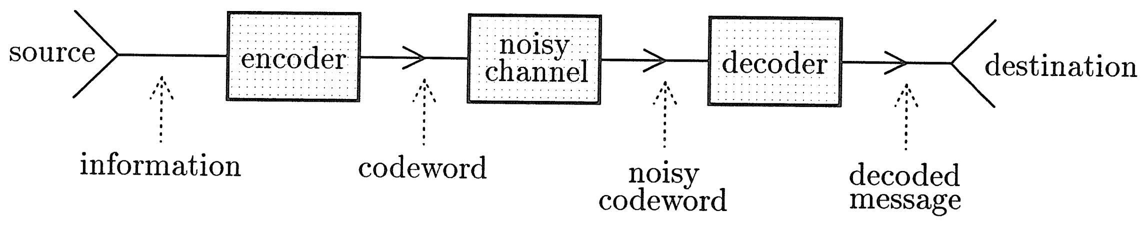


Figure 2.1: Communication channel.

codewords:			
0	0	0	0
1	0	1	0
0	1	0	1
1	1	1	1

Table 2.1: Example code of four codewords.

of the information digits (some number of times). Thus, the information x is sent over the channel as $xx...x$. As long as a majority of these digits arrive as x's, our best guess of the transmitted information (x) is correct. The original system of source–channel–destination has been expanded to source–encoder–channel–decoder–destination as illustrated in Figure 2.1. For the binary channel, the encoder performs a mapping from m information bits to n ($> m$ here) encoded bits, also called a codeword. A code is defined by a set of codewords. Table 2.1 lists one such set of codewords. Table 2.2 details an encoding algorithm for the code of Table 2.1. A string of information bits is divided into groups of two bits, each of which is encoded by the mapping in Table 2.2 into a codeword of four bits. The encoded bits are transmitted over the channel. The decoder performs the analogous mapping from n bits back to m information bits. Note that while the encoder will map one of 2^m

m information bits:		codeword of length n:
00	$\longrightarrow$	0000
10	$\longrightarrow$	1010
01	$\longrightarrow$	0101
11	$\longrightarrow$	1111

Table 2.2: Encoding the example code.

information strings into one of 2^m n-bit codewords, the decoder must be able to map one of 2^n n-bit strings back into m bits of information. To illustrate a simple decoder for our example code, we will assume that the channel introduces errors in the form of bit flips. Table 2.3 presents the mapping for a decoder that corresponds to the encoder in Table 2.2. Note that this decoder maps all $2^n = 16$ possible received strings into information bits. Note also that there are other decoders that correspond to the encoder of Table 2.2 (since the encoder does not specify how we should decode the received string 0010, for example).

We see that the decoder must effectively correct any errors in the transmitted codeword as well as perform the encoder's inverse mapping.

In most channels, the probability of one bit of the codeword being in error is less than that of two bits' being in error, which is less than three, etc. Thus, given that string y was received at the decoder, the most likely codeword z, which gave rise to it, is that which is closest to y in terms of Euclidean distance. A decoder that gives the codeword closest to the input, for this type of noise, is called a maximum likelihood decoder (MLD). An obvious MLD decodes an incoming string by comparing it to each codeword, finding the codeword closest to the string. For large codes however, this implementation of an MLD is computationally impractical.

In some channels, like the Gaussian channel, errors added by the channel are analog, i.e., continuous in value. The decoder can make use of the information in these analog errors by performing *soft-decision* decoding. A soft-decision decoder maps nonbinary strings directly into information bits. Soft-decision decoding differs from *hard-decision* decoding, in which incoming digits are quantized before any error-

received string:		decoder output:
0000	$\longrightarrow$	00
0001	$\longrightarrow$	00
0010	$\longrightarrow$	00
0011	$\longrightarrow$	00
0100	$\longrightarrow$	00
0101	$\longrightarrow$	01
0110	$\longrightarrow$	00
0111	$\longrightarrow$	01
1000	$\longrightarrow$	00
1001	$\longrightarrow$	00
1010	$\longrightarrow$	10
1011	$\longrightarrow$	10
1100	$\longrightarrow$	00
1101	$\longrightarrow$	01
1110	$\longrightarrow$	10
1111	$\longrightarrow$	11

Table 2.3: Decoder mapping for the example code.

correction or decoding occurs. This quantization in hard-decision decoding results in a significant degradation of the signal-to-noise ratio. From the code in Table 2.1, the codeword (1, 0, 1, 0) (using parentheses and commas for clarity), when sent over the Gaussian channel, could give rise to a string such as (0.5, 0.1, 0.95, −0.05). A hard-decision decoder might quantize by setting string components greater than 0.5 (half way between 0 and 1) to 1 and components less than or equal to 0.5 to 0. The above string would be quantized to (0, 0, 1, 0) and decoded, using the algorithm in Table 2.3, as 00, resulting in an error. A soft-decision decoder that picks the codeword closest to the string would perform an error-free decoding (of 10). From the above discussion, we see that the decoder's capabilities greatly affect the overall system performance.

In addition, several parameters of the code constrain the performance of the system. Here we discuss the code parameters *rate* and *minimum distance.* Note that by encoding our information, we send fewer information bits per time period. This fact is expressed as a *rate* of information transmission, given by m/n. Clearly, the rate of the code in Table 2.1 is 1/2. Thus, half of the bits sent over the channel convey information; the other half send redundant information. The rate of a code, used for error-correction, is bounded above by 1 (when $n = m$). To maximize the amount of information sent over a channel in any given time, we wish to use a code with a large rate. However, in general, a code with a large rate will have a small *minimum distance*. The minimum distance of a code is the Hamming distance between the two closest codewords in the code. The Hamming distance between two codewords is defined as the number of components in which the codewords differ (where the components may or may not be bits). The code of Table 2.1 has a minimum distance of two, since codewords 0000 and 0101 differ in two components and no two codewords differ in fewer than two components. The minimum distance of a code determines how many transmission errors the code can correct. For instance, our example code cannot correct many errors that arise when one bit of a codeword is flipped (from 0 to 1, or from 1 to 0). If noise changes the codeword 1111 to 0111, even an optimal MLD would at best decode 0111 correctly half the time (since 0101 could have given rise to the noisy string as well). Codes with a larger spacing between codewords (i.e., greater minimum distance) can correct more errors. In general, there is a tradeoff between rate and minimum distance; a code with a large rate (many codewords) will have a small minimum distance, and vice versa.

Most well-known codes are linear; that is, the codeword space of the code can be defined by a spanning *basis* set of the codewords, and all the other codewords can be obtained by combining words in the basis set. Linear codes are consequently easy to specify and manipulate. We will not place a restriction on the codes that they be linear. Also, we will be concerned with decoding algorithms that are maximum likelihood and utilize soft decisions.

2.3 The Hopfield Neural Network

Hopfield has presented a neural network model with continuous variables and responses [Hop84]. In this section, we give a short description of the model and some known results about it. (For more details see the reference.)

The network consists of N neurons with synapse strength, or *weight*, between the input of neuron i and the output of neuron j given by T_{ij}. The output or state of the ith neuron at time t, given by $V_i = V_i(t)$, is a continuous function of the input $u_i = u_i(t)$ according to the relationship $V_i = g_i(u_i)$. We use a monotonically increasing sigmoid g, such as $g(x) = \tanh(Gx)$, for a gain G, and set $g_i = g$ for all i. The inputs of the neurons have a rate of change given by

$$\frac{du_i}{dt} = \sum_j T_{ij}V_j - \frac{u_i}{R_i} + I_i + t_i \qquad (2.1)$$

for $i = 1$ to N. The I_i's are external inputs that we will henceforth ignore. The $\frac{1}{R_i}$'s are given by $\lambda = \sum_j |T_{ij}|$. t_i is a threshold term which we will set to θ for all i. Thus, to specify the system completely, we must give the T_{ij}'s, the threshold θ of the system and the sigmoid g. If we use Hopfield's outer product algorithm to find the T_{ij}'s,

$$T_{ij} = \sum_s (V_i{}^s)(V_j{}^s)$$

(and $T_{ii} = 0$) where the V^s's are the designed stable states for $s = 1$ to n, and $V_i^s \in \{-1, 1\}$, then the energy function at a state V is given by

$$E = -\frac{1}{2}\sum_s (V^s \cdot V)^2 + \sum_i 1/R_i \int_0^{V_i} g^{-1}(V)\, dV$$

Hopfield has shown that the dynamics of the network give stable states that are minima of the energy space. Given a state V, this energy function incorporates the distances from V to all the stable states. This energy, however, does not guarantee that the network will converge from V to the closest stable state V^s. One limitation, then, of this type of neural network is that we may not be able to design a network with the convergence behavior that we desire. Another limitation is that this network can have spurious (undesigned) stable states in addition to the desired stable states. Also, for a network of N neurons, the storage capacity, or the maximum number of arbitrary memories that can be stored and recovered exactly, has been shown to be bound above by N [AMJ85]; [MPRV87].

Using a neural network as a decoder could be potentially difficult. The difficulty in designing desired convergence behavior means that such a decoder might decode input strings as codewords far from the original string (not at all an MLD), or as an unchosen spurious state (not as a codeword). Also, if we utilize a system with a storage capacity that grows as the number of neurons, then the number of codewords, given by 2^m, is limited to be less than or equal to the number of stable states N. Thus, a neural network with N stable states can decode only codes of rate $= m/n = \log N/N$ or less. As N grows, this bound on the rate severely limits the number of codes that can be decoded by a system with a storage capacity of N. These properties seem to suggest that this type of network could have only limited use in a coding environment. In Sections 2.4 and 2.5, we show that for the storage of nonarbitrary memories we can obtain higher storage capacity and that, in fact, a neural network can decode codes of large rate.

2.4 The K–Winners–Take–All Code and Network

Here we present a family of neural networks that we have found to perform as maximum likelihood decoders for a given family of codes. Platt and Hopfield [PH86] presented a 1–in–N code that is decoded by a winner–take–all network. Given an input of N real numbers, the winner–take–all network will converge to a state with one positive number, in the position of the largest of the N input numbers, and $(N-1)$ negative numbers in the remaining positions. Here we present a generalization of the winner–take–all network that will decode a k–in–N code.

Our code consists of all words composed of 1's and -1's, of length N with the same number of 1's. (This code is often referred to as the constant weight code. To avoid confusion with the weight of the network's connections, we will not use this nomenclature). For a given k, each codeword has k components that are $+1$ and $(N-k)$ components that are -1. Clearly, the number of codewords in this code is $\binom{N}{k}$. Table 2.4 lists the code with $N=4$ and $k=2$.

To maximize the number of codewords and thus the rate, we take $k = \frac{N}{2}$ (or $k = \frac{(N-1)}{2}$ for N odd). Then, $m = \log_2 \binom{N}{\lfloor N/2 \rfloor} \geq N - \frac{1}{2}\log_2 2N$, and the rate is $m/N \geq 1 - \frac{1}{2N}\log_2 2N$. The rate asymptotically approaches one. The minimum

codewords:			
−1	−1	1	1
−1	1	−1	1
−1	1	1	−1
1	−1	−1	1
1	−1	1	−1
1	1	−1	−1

Table 2.4: Code of all words of length four with two 1's and two −1's.

distance of this code is 2, since changing a 1 component to a −1 and changing (a different) −1 component to a 1 in one codeword results in another codeword that differs in two positions from the first codeword (e.g., see the first two codewords in Table 2.4).

We will present a neural network with the property that the stable states of the network correspond exactly to the codewords of the k–in–N code. Codewords first sent over a noisy channel are then put through the neural network, where the network stabilizes to the codeword closest to the noisy input vector. We will first present the network and discuss its dynamic behavior and then illustrate its capability to perform maximum likelihood decoding.

The k–winners–take–all (KWTA) neural network that decodes the k–in–N code has the following parameters:

$$\begin{aligned} T_{ij} &= \delta_{ij} - 1 \\ g(u_i) &= \tanh(Gu_i) \\ \theta &= 2k - N \end{aligned}$$

where δ is the Kronecker delta and G is a given gain. Note that the weight matrix of all −1's (except for the diagonal) forces each node to inhibit all the other nodes. In this way, the nodes compete to be winners (i.e., to converge to 1).

From [Hop84] we know that a neural network is guaranteed to converge to a stable state if the connection matrix T is symmetric. Since our matrix is symmetric, the network will converge. Now let us examine these equilibrium states.

2.5 Stable States

At the stable states $\hat{\mathbf{u}}$ we have $\frac{d\hat{u}_i}{dt} = 0$ for all i. Or, substituting into Equation 2.1, we obtain for all i

$$0 = \sum_j T_{ij}\hat{V}_j - \hat{u}_i\lambda + (2k - N)$$

which gives

$$\hat{V}_i = \sum_j \hat{V}_j + \hat{u}_i\lambda - 2k + N \tag{2.2}$$

Theorem 2.1 For a given stable state $\hat{\mathbf{u}}$, every component $\hat{u}_i$ can have one of at most three distinct values.

Proof We can rewrite Equation 2.2 as

$$\hat{V}_i = \hat{u}_i\lambda + H(\hat{\mathbf{u}}) \tag{2.3}$$

where

$$H(\hat{\mathbf{u}}) = \sum_j \hat{V}_j - 2k + N$$

For a given stable state $\hat{\mathbf{u}}$, $H(\hat{\mathbf{u}})$ is fixed (i.e., independent of i). Thus, when we plot the two sides of Equation 2.3, we obtain, for all i, the diagram in Figure 2.2. We observe that the sigmoid function $\hat{V}_i$ and the linear sum $\{\hat{u}_i\lambda + H(\hat{\mathbf{u}})\}$ can intersect in up to three points. The corresponding three values of V_i are the only values that can be assumed by the stable state components $\hat{V}_i$. Note that if the slope of $\hat{u}_i\lambda + H(\hat{\mathbf{u}})$ is greater than the slope of the sigmoid at $\hat{u}_i = 0$ (or, $\lambda > G$), then the lines will intersect at only one point. These intersection points are the solutions to Equation 2.2, and hence the equilibrium points. ∎

In Theorem 2.1 we showed that components of stable states can take one of at most three values. We now distinguish between stable state components with $g'(\hat{u}_i) \geq \lambda$ and components with $g'(\hat{u}_i) < \lambda$. In terms of Figure 2.2, we are distinguishing between

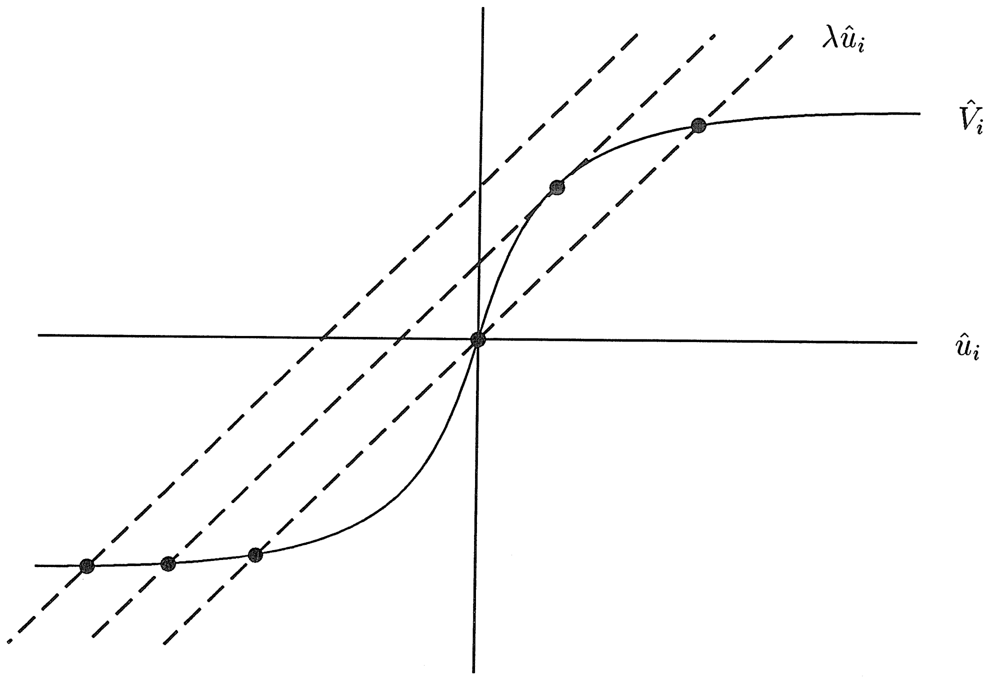


Figure 2.2: The intersection of $\hat{V}_i$ and $\hat{u}_i\lambda + H(\hat{\mathbf{u}})$.

equilibrium point components that are along the highly sloped center of the sigmoid, and those components that are along the flatter portions of the sigmoid, respectively. We categorize the equilibrium states as one of two types: Type I has one or more components with $g'(\hat{u}_i) \geq \lambda$; type II has no component with $g'(\hat{u}_i) \geq \lambda$. We will show that for a large enough gain G, no states of type I are stable. We begin by showing that type I stable states have exactly one component u_i such that $g'(\hat{u}_i) \geq \lambda$.

Theorem 2.2 Given any asymptotic stable state $\hat{\mathbf{u}}$, at most one of the components $\hat{u}_i$ of $\hat{\mathbf{u}}$ may satisfy

$$g'(\hat{u}_i) \geq \lambda \tag{2.4}$$

Proof We use the following claim to prove Theorem 2.2.

Claim Given any asymptotic stable state $\hat{\mathbf{u}}$, the following is true for any $i \neq j$:

$$\lambda > \sqrt{g'(\hat{u}_i)g'(\hat{u}_j)} \tag{2.5}$$

Proof Consider a small perturbation around a stable state $\hat{\mathbf{u}}$, say $\mathbf{u} = \hat{\mathbf{u}} + \Delta\mathbf{u}$ and $\mathbf{V} = \hat{\mathbf{V}} + \Delta\mathbf{V}$. We can linearize the sigmoid function as $\Delta V_j \approx \Delta u_j g'(\hat{u}_j)$, and Equation 2.1 as

$$\frac{d(\Delta u)}{dt} \approx L(\hat{\mathbf{u}})(\Delta u)$$

where

$$L(\hat{\mathbf{u}}) = T \cdot \operatorname{diag}(g'(\hat{u}_1), g'(\hat{u}_2), \ldots, g'(\hat{u}_N)) - \lambda I$$

A necessary and sufficient condition for the asymptotic stability of $\hat{\mathbf{u}}$ is for $L(\hat{\mathbf{u}})$ to be negative definite. A necessary condition for $L(\hat{\mathbf{u}})$ to be negative definite is for all 2×2 matrices $L_{ij}(\hat{\mathbf{u}})$, $i \neq j$, of the type

$$L_{ij}(\hat{\mathbf{u}}) = -\begin{pmatrix} \lambda & g'(\hat{u}_i) \\ g'(\hat{u}_j) & \lambda \end{pmatrix}$$

to be negative definite. Here we have made an infinitesimal perturbation about components i and j only. Any matrix $L_{ij}(\hat{\mathbf{u}})$ has two real eigenvalues. For both of the eigenvalues to be negative, we need

$$-\lambda + \sqrt{g'(\hat{u}_i)g'(\hat{u}_j)} < 0$$

Since we assumed $\hat{\mathbf{u}}$ to be asymptotically stable, this condition must hold, and Equation 2.5 follows. □

If we have an asymptotic stable state with two components $\hat{u}_i$ and $\hat{u}_j$ where $g'(\hat{u}_i) \geq \lambda$ and $g'(\hat{u}_j) \geq \lambda$, then $\sqrt{g'(\hat{u}_i)g'(\hat{u}_j)} \geq \lambda$, violating the claim. Therefore, to satisfy the claim, at most one component $\hat{u}_i$ of an asymptotic stable state $\hat{\mathbf{u}}$ may satisfy $g'(\hat{u}_i) \geq \lambda$. ∎

Theorem 2.2 established that stable states of type I have at most one component $\hat{u}_i$ with $g'(\hat{u}_i) \geq \lambda$. Next we will prove that, in fact, there are no stable states of type I.

Theorem 2.3 For sufficiently large gain G, there are no stable states of type I.

Proof We will prove this theorem by establishing a bound on $H(\hat{\mathbf{u}}) = \sum_j \hat{V}_j - 2k + N$. Then, by showing that there are no asymptotically stable states of type I that satisfy this bound, we will conclude that no stable states of this type exist.

Claim If $\hat{\mathbf{u}}$ is a stable state of the network described by Equation 2.2, then

$$\max_{\hat{u}_i < 0} \hat{V}_i \; < \; H(\hat{\mathbf{u}}) \; < \; \min_{\hat{u}_i > 0} \hat{V}_i \tag{2.6}$$

Proof In Figure 2.3 we present a graphical proof of this claim. The point a, on the line $\lambda \hat{u}_i + H(\hat{\mathbf{u}})$, is clearly $H(\hat{\mathbf{u}})$ above the horizontal axis. In addition, this point is greater than point b, the projection of the negative stable state on the vertical axis. And point a is less than point c, the projection of the smaller positive stable component value on the vertical axis. We also can prove this claim algebraically. Since $\hat{u}_i$ and $\hat{V}_i = g(\hat{u}_i)$ are of the same sign, $H(\hat{\mathbf{u}})$ can be neither too large ($\hat{u}_i > 0$) nor too small ($\hat{u}_i < 0$). We can rewrite Equation 2.2 as

$$\lambda \hat{u}_i \; = \; \hat{V}_i - \sum_j \hat{V}_j + 2k - N \tag{2.7}$$

For $\hat{u}_i > 0$,

$$\hat{V}_i - \sum_j \hat{V}_j + 2k - N \; > \; 0$$

or

$$\hat{V}_i \; > \; \sum_j \hat{V}_j - 2k + N$$

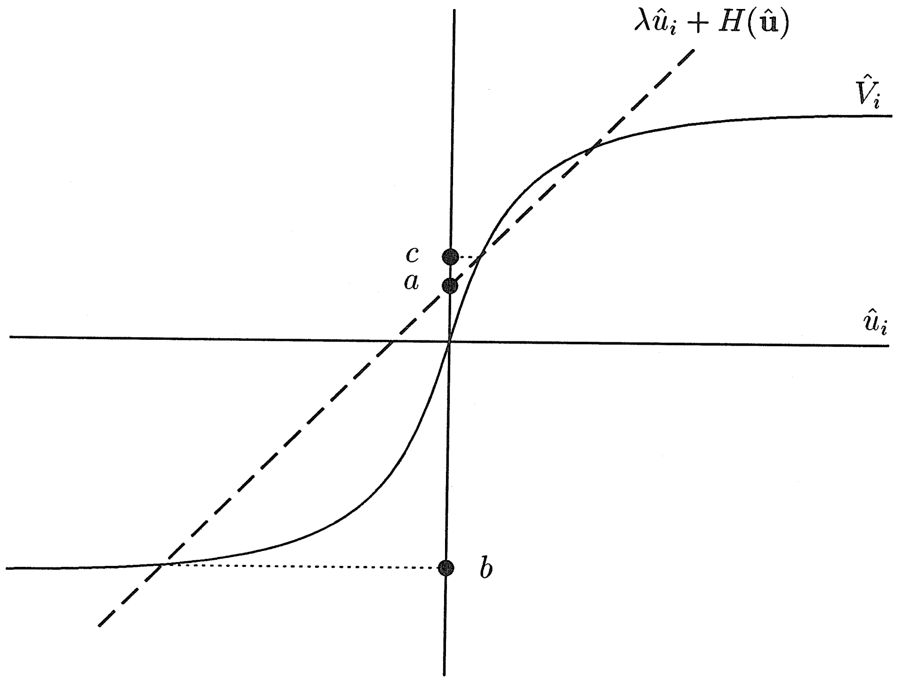


Figure 2.3: Bounding $H(\hat{\mathbf{u}})$ (point a) by the values of the stable state components (points b and c).

Likewise, for $\hat{u}_i < 0$,

$$\hat{V}_i - \sum_j \hat{V}_j + 2k - N \; < \; 0$$

or

$$\hat{V}_i \; < \; \sum_j \hat{V}_j - 2k + N$$

which results in Equation 2.6. □

From Theorem 2.2 a state of type I has exactly one component $\hat{u}_i$ with $g'(\hat{u}_i) \geq \lambda$. Let $\gamma = \hat{u}_i \geq 0$ be that component. Using Figure 2.3 we can make the following observation about $H(\hat{\mathbf{u}})$: If $c = g(\gamma) > 0$, then $a = H(\hat{\mathbf{u}}) > 0$ (because $\tanh(x)$ is concave for $x \geq 0$, and we assume that $G > \lambda$). (The cases where $\gamma < 0$ and $\gamma = 0$ are handled in a similar fashion.) In addition, such a stable state has N_+ components with $g'(\alpha) < \lambda$ and $\alpha > 0$, and N_- components with $g'(\beta) < \lambda$ and $\beta < 0$. Note also that $N_+ + N_- + 1 = N$. Now let us substitute these values into Equation 2.6:

$$0 \; < \; g(\alpha)N_+ + g(\beta)N_- + g(\gamma) - 2k + N \; < \; g(\gamma) \tag{2.8}$$

By substituting for N, and rearranging terms, we obtain

$$-1 - g(\gamma) \; < \; (g(\alpha) + 1)N_+ + (g(\beta) + 1)N_- - 2k \; < \; -1$$

For a large enough gain G, $g(\alpha)$ and $g(\beta)$ can be made arbitrarily close to $+1$ and -1, respectively. These assumptions yield

$$-1 - g(\gamma) \; < \; 2N_+ - 2k \; < \; -1 \tag{2.9}$$

Since $2N_+ - 2k$ is even, the only way to satisfy Equation 2.9 is for $-1 - g(\gamma) < -2$. Then $g(\gamma)$ must be greater than 1, which implies that $g'(\gamma) < \lambda$. This contradicts our assumption that $g'(\gamma) > \lambda$. Thus, there is no $\gamma > 0$, $g'(\gamma) > \lambda$ such that the condition in Equation 2.9 can be satisfied. This contradiction implies that there are no stable states of type I. ∎

The possible stable state equilibrium components that result from the curves of $\lambda\hat{u}_i + H(\hat{\mathbf{u}})$ and $\hat{V}_i$ intersecting in two locations result in states that can be shown, in a similar way, to be not stable (because Equation 2.6 can not be satisfied). Also, the only possible type I stable states resulting from a single crossing of $\lambda\hat{u}_i + H(\hat{\mathbf{u}})$ and $\hat{V}_i$ occur when the gain G is less than λ. We will preclude this state by choosing the gain G to be greater than λ. Hence, independent of the intersections of the lines $\lambda\hat{u}_i + H(\hat{\mathbf{u}})$ and $\hat{V}_i$, there are no stable states of type I. Now we consider stable states of type II.

Theorem 2.4 The KWTA network has $\binom{N}{k}$ stable states corresponding to the states with k positive components and $(N-k)$ negative components.

Proof We consider stable states that have N_+ components with $g'(\alpha) < \lambda$ and $\alpha > 0$, and N_- components with $g'(\beta) < \lambda$ and $\beta < 0$. (Other possible stable states of type I can be shown, using Equation 2.6, not to exist.) Applying Equation 2.6 to stable states of type II, we obtain

$$-1 \; < \; g(\alpha)N_+ + g(\beta)N_- - 2k + N \; < \; 1$$

Since $N_+ + N_- = N$, this results in

$$-1 \; < \; (g(\alpha)+1)N_+ + (g(\beta)+1)N_- - 2k \; < \; 1$$

Again, for a large enough gain G, $g(\alpha)$ and $g(\beta)$ can be made arbitrarily close to $+1$ and -1, respectively. These assumptions yield: $-1 < 2(N_+ - k) < 1$, or $N_+ = k$. Thus, stable states of type II have k components α ($\alpha > 0$) and $(N-k)$ components β ($\beta < 0$). By symmetry, all states with k components α and $(N-k)$ components β are stable. There are $\binom{N}{k}$ such states with k components α and $(N-k)$ components β. ∎

2.6 Maximum Likelihood Decoding

Finally, we need to show that our network performs maximum likelihood (or nearest neighbor) decoding. The network is a maximum likelihood decoder because it forces the largest k components of the input vector to α ($\alpha > 0$) and the remaining components to β ($\beta < 0$). This type of convergence occurs because the network upholds an invariance among the components of $\mathbf{u}$. In other words, if we order the components of $\mathbf{u}$ as ($u_1 \geq u_2 \geq \ldots \geq u_N$) at time t_0, then this order will remain true for all time.

Theorem 2.5 If $u_i(t_0) \geq u_j(t_0)$, then $u_i(t) \geq u_j(t)$ for all time $t \geq t_0$.

Proof From Equation 2.1,

$$\frac{d}{dt}\begin{bmatrix} u_i \\ u_j \end{bmatrix} = \begin{bmatrix} \sum_{m \neq i,j} V_m + V_j - \lambda u_i + 2k - N \\ \sum_{m \neq i,j} V_m + V_i - \lambda u_j + 2k - N \end{bmatrix}$$

With this equation and initial conditions $u_i(t_0)$ and $u_j(t_0)$, the functions u_i and u_j are completely specified. These functions have identical forms and thus, if $u_i(t_1) \geq u_j(t_1)$, then $u_i(t) \geq u_j(t)$ for $t > t_1$, and hence the invariance holds. ∎

Finally, Theorems 2.4 and 2.5 give us the following theorem.

Theorem 2.6 Given an initial state $\mathbf{u}(0)$ and a large enough gain G, the KWTA neural network will converge to a stable state with k components equal to a positive number ($\alpha > 0$) in the positions of the k largest initial components, and $(N - k)$ components equal to a negative number ($\beta < 0$) in the other $(N - k)$ positions.

Proof Assume that $u_i(0) > u_j(0)$. By Theorem 2.5 this ordering will be preserved by the network. Similarly, the ordering of all the components is preserved. By Theorem 2.4, the network will converge to a state with k positive components and $(N - k)$ negative components. To preserve the initial ordering of components, the k largest (at $t = 0$) will converge to α ($\alpha > 0$), and the remaining $(N - k)$ to β ($\beta < 0$). ∎

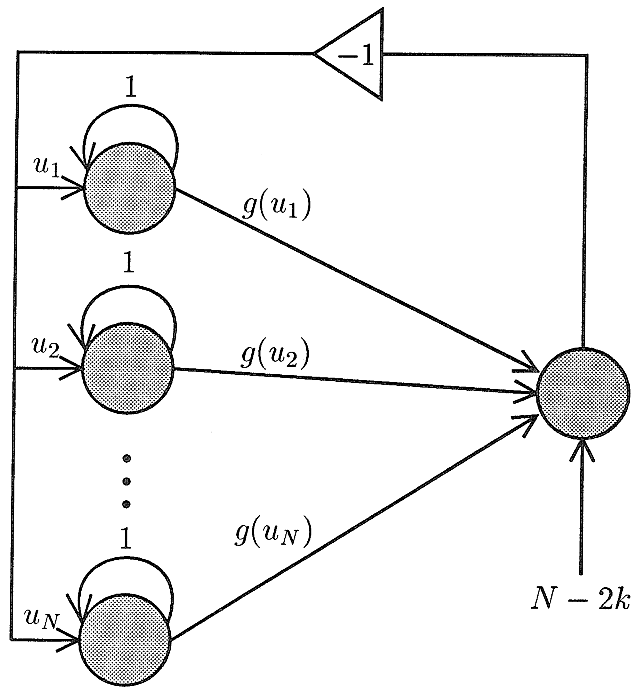


Figure 2.4: An implementation of the KWTA Network with $O(N)$ connections.

2.7 Discussion

The rate of the KWTA code is maximized by setting $N = k/2$. The rate of the $N/2$–winners–take–all code is bounded below by $1 - \frac{1}{2N}\log 2N$. This rate asymptotically approaches one.

Experimentally, we have found that a gain G greater than $\lambda = (N-1)$ was large enough to ensure that the network converged to the desired stable states.

The network has $O(N^2)$ connection weights. However, we can implement the network as shown in Figure 2.4 with only $O(N)$ connections. This network sums the output from all the neurons and compares it with the threshold. The result is then negated and fed back as input to all of the neurons. In addition, a self-connection is required at each node.

We have presented a neural network that behaves as a maximum likelihood decoder for a given nonlinear code. For $k = N/2$, we have seen that the number of stable states is exponential in the number of neurons. Because the network takes N

real numbers as inputs and converges to the closest codeword of k components that are $+1$, and $(N-k)$ components that are -1, it performs soft-decision decoding of KWTA codes. The minimum distance of this code is two. This code has a small minimum distance (the decoder can correct only small errors), and thus provides motivation to look for codes with larger minimum distances that can be decoded by a neural network. Chapters 3, 4, and 5 discuss such codes and develop networks to decode them.

Chapter 3

Interconnected KWTA Networks: Randomly Placed KWTA Constraints

3.1 Introduction

To construct codes with greater minimum distances than those achieved by using a single KWTA constraint, we have experimented with interconnecting KWTA constraints. Overlapping KWTA constraints of some size N are placed on M nodes. Thus, a given node may be in several constraints, each involving a different set of nodes. This interconnecting of constraints does not give rise to a neural network of different form. The inputs of the neurons still have a rate of change given by

$$\frac{du_i}{dt} = \sum_j T_{ij}V_j - \frac{u_i}{R_i} + I_i + t_i \qquad (3.1)$$

for $i = 1$ to N. Now, however, the weight matrix T will not be given by $T_{ij} = \delta_{ij} - 1$, as was the case for the KWTA code in Chapter 2. Instead, the weight matrix contains inhibitory (KWTA) connections for each set of nodes in a KWTA constraint. For example, let us consider the following two systems of four nodes each. The first system consists of a single KWTA constraint on the four nodes. The weight matrix is given by

$$T = \begin{pmatrix} 0 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{pmatrix}$$

The second system has three KWTA constraints of size two. The first constraint is composed of nodes 1 and 2, the second of nodes 3 and 4 and the third of nodes 1 and 3. The weight matrix for this network is given by

$$T = \begin{pmatrix} 0 & -1 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Note that the KWTA constraint composed of nodes 1 and 2 is evident in the submatrix given by T_{11}, T_{12}, T_{21} and T_{22}. We could rewrite Equation 3.1, for u_1, for example, to emphasize this constraint on nodes 1 and 2.

$$\frac{du_1}{dt} = T_{12}V_2 - \frac{u_1}{R_1} + I_1 + t_1 + (T_{13}V_3 + T_{14}V_4) \tag{3.2}$$

The last term in Equation 3.2, which is due to other constraints on node 1, could be thought of as a time-varying input.

In Chapter 2 we established a direct correspondence between the stable states of a neural network and the codewords of a code. We define the codewords in the code to be the stable states of the network. It is also convenient to consider those strings of $+1$'s and -1's that strictly satisfy the constraints (e.g., for the KWTA network, we consider strings of length N with K components that are 1 and $(N-k)$ components that are -1). For all of the analog decoders that we have observed, these strings correspond exactly to the stable states of the network, and therefore, we often discuss the stable states as strings that must satisfy a set of constraints on their components. We have not always seen this behavior with the digital decoders [Hop82].

We have looked at several families of codes where the KWTA constraints are placed in a regular fashion as well as codes with randomly placed constraints. In this chapter, we discuss decoders constructed with overlapping KWTA constraints that are placed randomly. In Chapters 4 and 5 we discuss decoders constructed with regularly placed KWTA constraints.

The neural network is simulated on a computer. We analyze the decoding performance of the network by adding Gaussian noise to each bit of a codeword, and then giving our neural network this noisy string as input. By repeating this many times and for different codewords, we can obtain an estimate of the decoding performance (in terms of error rate) for a given neural network when used with a given code. We also simulated an MLD by comparing each noisy input vector to all possible codewords and decoding the vector as the closest codeword. Since maximum likelihood decoding is optimal in this noise environment, the performance of the neural network can be compared to MLD performance to get a measure of the neural network's decoding ability.

3.2 Randomly Placed KWTA Constraints

To test random sets of KWTA constraints on M nodes as decoders, we first choose N, the size of the constraints. Our constraints were KWTA constraints with $N = 4$ and $k = 2$. By placing inhibitory connections between the nodes in a constraint, these constraints define a neural network. The stable states of the network define a code. We simulated networks defined by 3000 random sets of C constraints each, for various C ($7 \leq C \leq 14$). For each set of constraints, we recorded the minimum distance of, and the number of codewords in, the corresponding code. Many of the sets of constraints resulted in a network with no stable states, and thus in an empty code.

Table 3.1 shows an example set of nine constraints, randomly placed on 16 nodes. The first KWTA constraint is composed of nodes 1, 4, 11 and 14. The resulting code contains 166 codewords and has a minimum distance of two. To find the stable states of the network, an exhaustive test of the stability of all 2^M states was

										nodes:							
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	a	1			1							1			1		
	b	1						1							1		1
	c	1					1	1							1		
constraints:	d		1	1		1										1	
	e	1	1													1	1
	f			1				1	1					1			
	g		1	1		1										1	
	h			1			1				1		1				
	i	1						1		1							1

Table 3.1: Nine random KWTA constraints that yield a code of length 16 with 166 codewords and a minimum distance of two.

performed on the network. Thus, for practical considerations, we used a network of 16 nodes ($M = 16$) for most of these experiments with random constraints. (A 16 node system is large enough to be interesting but not too large to simulate.) Figures 3.1, 3.2, and 3.3 illustrate the maximum and average code size, and the number of codes found, at each minimum distance and number of constraints.

From these graphs, we can make the following three general observations: (1) The average random code size for a given minimum distance will be increased if we use fewer constraints. For example, at a minimum distance of two, the average code size decreases from 95.6 codewords for systems defined by seven constraints to 31.6 codewords for systems defined by fourteen constraints. (2) Codes with fewer constraints have smaller minimum distances, in general, than those with more constraints (e.g., there were no codes with a minimum distance greater than four for systems of seven or eight constraints). We explain some of this behavior by noting that adding a constraint to a system can eliminate only stable states, and thus, in general, more-constrained systems have fewer codewords than less-constrained systems. As a result of not having many codewords, the code's minimum distance will be large. (3) More codes were generated at each minimum distance for more-constrained systems than for less-constrained systems. We summarize these observations in Table 3.2. When we increased the constraint size to $N = 6$, we found that almost all the resulting codes

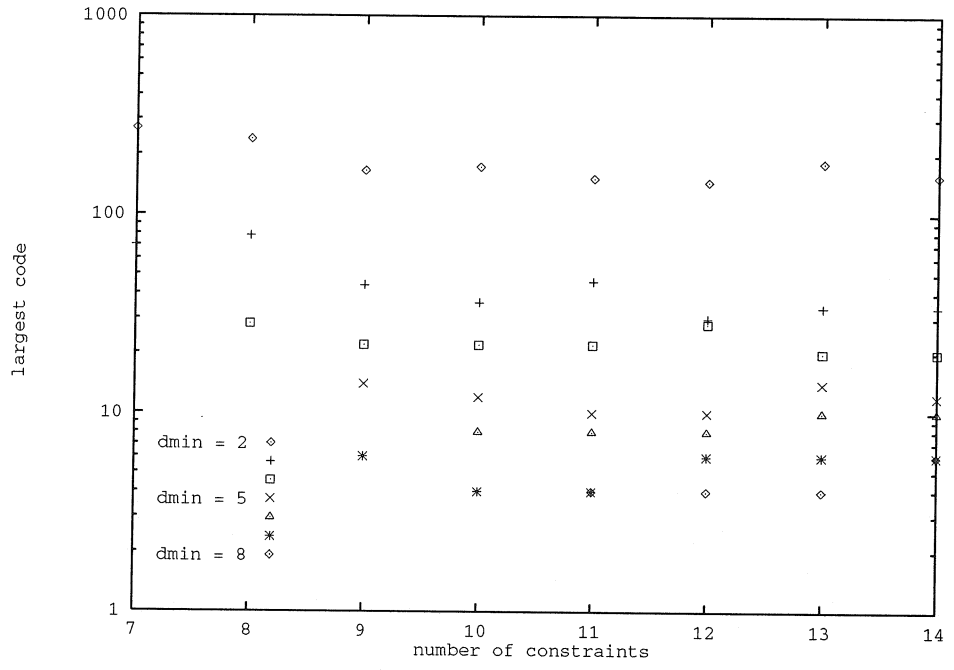


Figure 3.1: Maximum random code size by minimum distances (dmin) for various numbers of constraints. These experimental data were computed from 3000 random sets of constraints, for each set size from 7 to 14.

Number of Constraints:

	few	*many*
(1)	larger codes on average	smaller codes on average
(2)	codes have small dmin	codes have larger dmin
(3)	not many codes	many codes

Table 3.2: General characteristics of random codes.

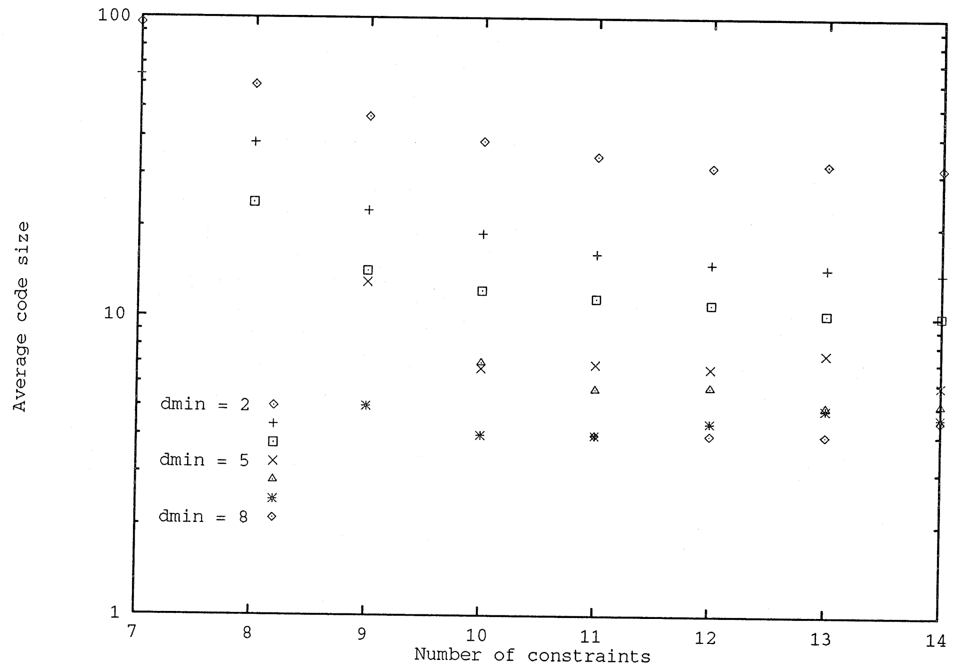


Figure 3.2: Average random code size by minimum distances (dmin) for various numbers of constraints. These experimental data were computed from 3000 random sets of constraints, for each set size from 7 to 14.

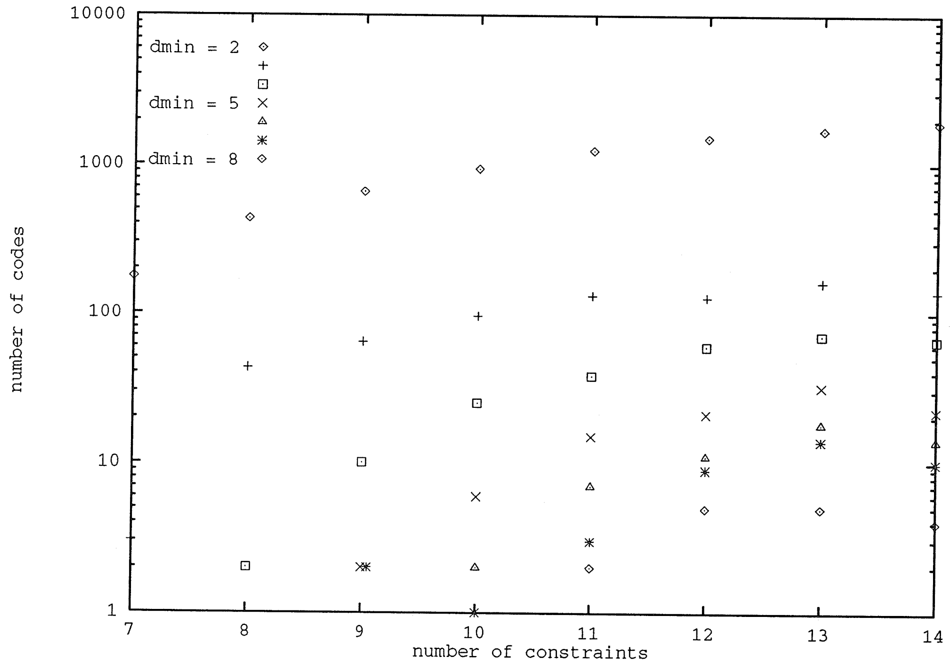


Figure 3.3: Number of random codes generated in 3000 trials at each value of C, the number of constraints, presented by code minimum distance (dmin).

had a minimum distance of two. As the constraint size grows, we are approaching the single KWTA system (i.e., $N = M$).

When the constraint size was decreased to 2, the system was overconstrained. Note that this system forces the two nodes in a constraint to have opposite values. Thus, every pair of nodes in a constraint can have one of at most two values in a codeword (i.e., (1,$-$1) or ($-$1,1)). In essence, the KWTA constraint of size 2 constructs codes where bits are repeated (actually, their complement is given), and because of the interconnection between constraints, this results in codes of large minimum distance but small code size.

To obtain insight into the properties of good codes, we looked at those codes with a large number of codewords at each minimum distance. We observed that some of these better random codes had repeated constraints. (A repeated constraint appears in the network's weight matrix T as larger inhibitory weights between the involved nodes.) Other codes had constraints that had three nodes in common (thus, they were almost-repeated constraints (arc)). For example, in Table 3.1 constraints d and g are identical, and constraints b and c both contain nodes 1, 7 and 14 (so they are arc). A general analysis of the probability of repeated and almost-repeated constraints as observed and as calculated follows.

First consider the probability of repeated constraints. For a system of M nodes and constraint size N, there are $\binom{M}{N}$ different constraints. There are $\binom{\binom{M}{N}}{C}$ sets with C unique constraints. The total number of sets of C constraints, if we permit repeated selection of individual constraints, is $\binom{\binom{M}{N}+C-1}{C}$. Assuming that we pick constraints independently, with equal probability, and with replacement, the probability of no repeated constraints is

$$p(\text{no repeats}) = \frac{\binom{\binom{M}{N}}{C}}{\binom{\binom{M}{N}+C-1}{C}} \tag{3.3}$$

For $C = 10$, $M = 16$ and $N = 4$, Equation 3.3 gives $p(\text{repeats}) = 1 - 0.9518 = .0482$. Over a group of 100 better codes with 7 to 13 constraints, the frequency of repeats was found experimentally to be 3%. Experimental results agree well enough with theoretical results for us to conclude that repeated rows are not an important characteristic of better codes.

Now let us consider the probability of almost-repeated constraints. Given a constraint, there are $\binom{(M-N)}{1}\binom{N}{1} = (M-N)N = 48$ other constraints that share three nodes with this constraint. For example, the first constraint in Table 3.1 is placed on nodes 1, 4, 11 and 14. One neighboring constraint is on nodes 1, 4, 11 and 16. Let us consider the number of sets of C constraints with no repeated or almost-repeated constraints. There are $\binom{M}{N}$ ways to pick the first constraint. The second constraint cannot be the same as the first constraint or any of its neighbors. Thus, there are $\binom{M}{N} - 49$ ways to choose the second constraint. There are *at least* $\binom{M}{N} - 49 - 49$ ways to choose the third constraint. If the first and second constraints overlap in $N-2$ positions, then they can share neighboring constraints. For example, in Table 3.1 constraints a and b share the four neighboring constraints given by (1, 4, 7, 14), (1, 4, 14, 16), (1, 7, 11, 14) and (1, 11, 14, 16). Thus, if we had picked constraints such as a and b for our first and second constraints, then four neighboring constraints would be subtracted twice from the number of constraints left from which to pick the third constraint. Therefore, there are *at most* $\binom{M}{N} - 49 - 49 + 4$ constraints from which to pick the third constraint. Similarly, there are at least $\binom{M}{N} - 49 - 49 - 49$ ways to pick the fourth constraint. And, since any pair of the first three constraints could share (four) neighboring constraints, there are at most $\binom{M}{N} - 49 - 49 - 49 + 4\binom{3}{2}$ ways to pick the fourth constraint.

Finally, letting $L = (M-N)N + 1$, we obtain

$$p(\text{no arc}) \geq \frac{\binom{M}{N}\left(\binom{M}{N} - L\right)\ldots\left(\binom{M}{N} - (C-1)L\right)\left(\frac{1}{C!}\right)}{\binom{\binom{M}{N}+C-1}{C}}$$

and

$$p(\text{no arc}) \leq \frac{\binom{M}{N}\left(\binom{M}{N} - L\right)\ldots\left(\binom{M}{N} - (C-1)L + 4\binom{(C-1)}{2}\right)\left(\frac{1}{C!}\right)}{\binom{\binom{M}{N}+C-1}{C}}$$

In Figure 3.4 we plot the upper and lower bounds for the probability of almost-repeated constraints as well as for the observed probability. These observations were made over a total of 100 good codes. For all but one value of C, the observed probability of almost-repeated constraints is below the upper bound, and in most

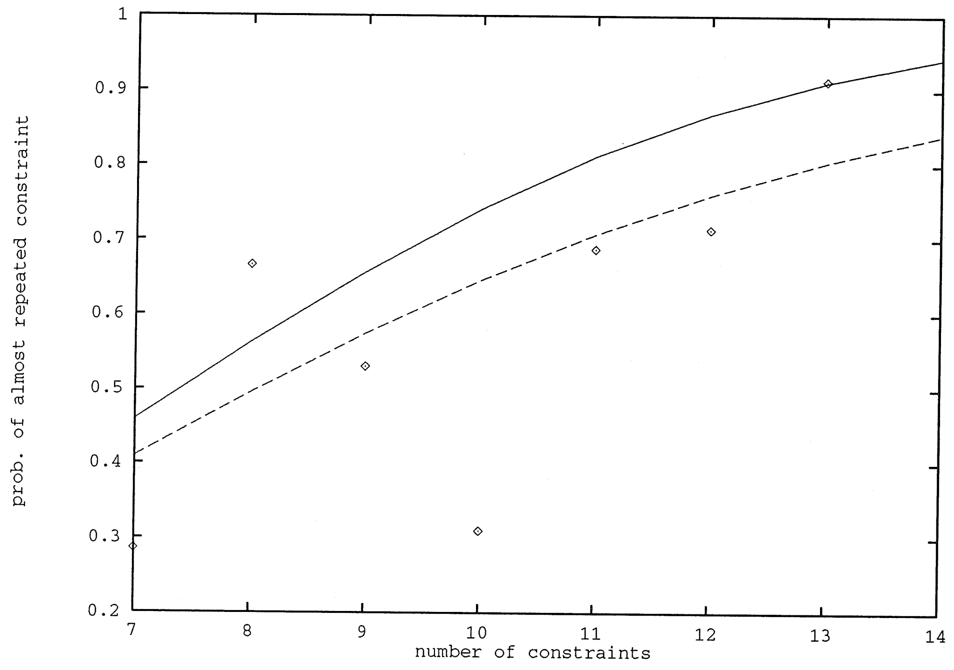


Figure 3.4: The observed probability of almost-repeated constraints, for a total of 100 codes, plotted against upper and lower bounds for the probability of almost-repeated constraints.

cases, below the lower bound. Thus, it appears that almost-repeated constraints are not an important characteristic of good codes.

In summary, by placing random KWTA constraints on the nodes in a neural network, we obtain stable states that define a code. By varying the number and size of the constraints we vary the size and minimum distance of the resulting code. We observed a tradeoff between less-constrained systems with fewer, larger codes displaying small minimum distances, and more-constrained systems with many codes of relatively large minimum distance but of smaller average size. In the next chapters, we consider systems with regularly placed constraints; the results of this section give us a basis for comparison.

Chapter 4

The Hypercube Family

4.1 Introduction

Here we present one class of codes that arise when we place a set of KWTA constraints on the nodes of a neural network in a regular fashion. We call these codes the hypercube family because the network with constraints of size two ($N = 2$) is interconnected like a hypercube, and networks with larger constraints resemble hypercubes with more than two nodes in each dimension (we discuss this structure in more detail later). We present example codes of this family and discuss their performance. We conclude by describing some general characteristics of the family.

A decoder for the hypercube family of codes has $M = N^i$ nodes that we can organize as an i-dimensional hypercube. We label each node $\{x_1, x_2, \ldots x_i\}$ with $x_j \in \{1, 2 \ldots N\}$. A KWTA constraint is placed on a set of N nodes that differ only in one index. For example, $\{1,1,1,\ldots,1\}$, $\{2,1,1,\ldots,1\}$, $\{3,1,1,\ldots,1\}$, $\ldots$, $\{N,1,1,\ldots,1\}$ are the nodes in one KWTA constraint. Each node is involved in i constraints, and the total number of constraints in the system is $C = \frac{M}{N} i = iN^{i-1}$. The minimum distance of an i-dimensional hypercube code is 2^i.

4.2 One-Dimensional Hypercube

The hypercube system with $i = 1$ reduces to the KWTA configuration with $M = N$. This system has been described in Chapter 2. With $k = N/2$, the rate of this code is

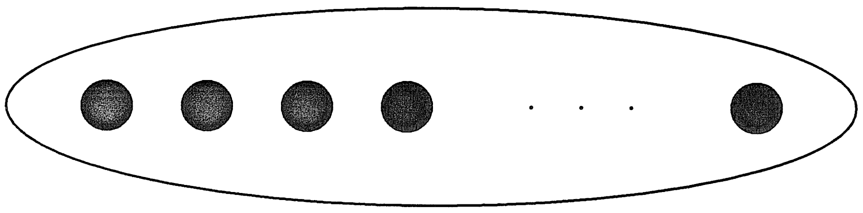


Figure 4.1: One-dimensional hypercube constraint placement. The oval represents the KWTA constraint.

greater than $1 - \frac{1}{2N} \log 2N$, and the minimum distance of this code is two. Figure 4.1 shows how the KWTA constraint (the oval) involves all the nodes (circles) in the network.

4.3 Two-Dimensional Hypercube

We will begin by discussing some general characteristics of two-dimensional hypercube codes, then we will present several example codes.

4.3.1 General Two-Dimensional Code

For a two-dimensional system ($i = 2$) the nodes can be arranged in an array where the KWTA constraints are along the rows and columns of the array. Each node is involved in two constraints (one involving the nodes in its row, the other involving the nodes in its column). Figure 4.2 illustrates the placement of constraints (ovals) on nodes (circles) on a system of N^2 nodes. From the figure, there are $iN = 2N$ total constraints.

We refer to the two-dimensional code of N^2 nodes as the $N \times N$ code. The general

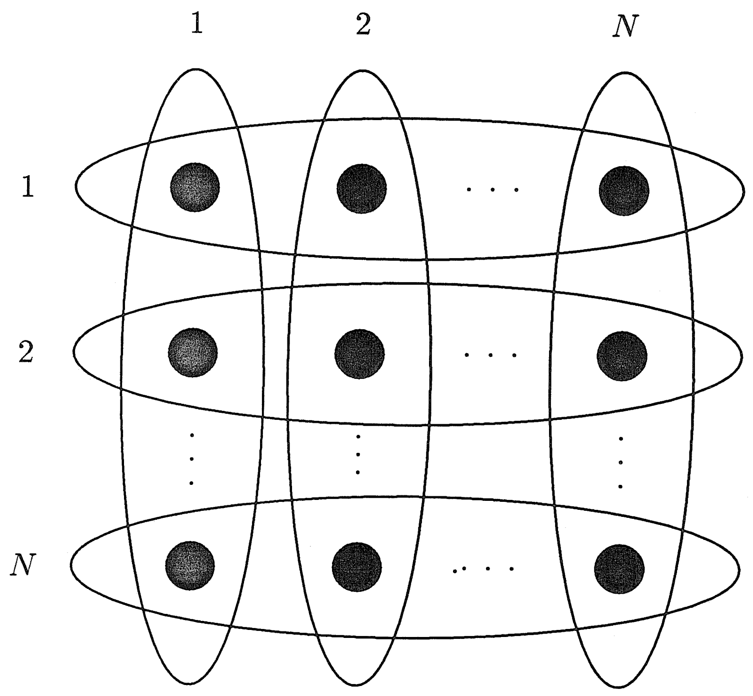


Figure 4.2: Two-dimensional hypercube constraint placement. The ovals represents the KWTA constraints of size N.

connection matrix for the corresponding $N \times N$ decoder is given by

$$T = -\begin{pmatrix} A & I & I & \dots & I \\ I & A & I & \dots & I \\ I & I & A & \dots & I \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I & I & I & \dots & A \end{pmatrix}$$

where $I = I_N$ and $A = J_N - I_N$, with J_N the matrix of size N by N with all 1 entries. Here, the A portions of T enforce the KWTA constraints along the rows, and the I portions enforce the KWTA constraints along the columns. A stable state $\mathbf{u}$ has the form $(u_1, u_2, \dots, u_{N^2})$. When discussing stable states, we will also refer to them in their matrix form (as an $N \times N$ codeword) as given by

$$\mathbf{u} = \begin{pmatrix} u_1 & u_2 & \dots & u_N \\ u_{N+1} & u_{N+2} & \dots & u_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ u_{N^2-N+1} & u_{N^2-N+2} & \dots & u_{N^2} \end{pmatrix}$$

The minimum distance of these codes is four. This is easily illustrated by the example 4×4 codeword

$$\mathbf{u} = \begin{pmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

If we negate the submatrix composed of u_1, u_2, u_5 and u_6, the result is another codeword (since the KWTA row and column constraints are still satisfied). These two codewords are at Hamming distance four apart. In fact, one can show that no closer codewords exist, and thus, the minimum distance of the two-dimensional hypercube code is four. In Section 5.3, the minimum distance of hypercube codes is discussed in more detail. In the next section, we discuss the rate of two-dimensional hypercube codes.

Counting the Number of Codewords

For $k = N/2$, we have found that the stable states of this network correspond to the (1,−1) matrices of size $N \times N$ with zero row and column sums (i.e., the matrices that have $(N/2)$ entries that are $+1$ and $(N/2)$ entries that are -1 per row and column). For general N, the problem of counting such matrices is nontrivial. A simple upper bound is obtained by counting the number of matrices constructed by using all possible rows (and ignoring the column constraints). Thus, the number of (1,−1) matrices with zero row and column sums is at most $\binom{N}{N/2}^N$. Knutson [Knu88] has shown that a lower bound on the number of codewords is

$$\text{number of codewords} \geq \frac{\binom{N}{N/2}^N}{(N+1)^N}$$

(The numerator gives the matrices with correct rows. Note that these matrices give rise to less than $(N+1)^N$ column sum vectors. We are interested only in one column sum vector, the one with all zero entries. Knutson [Knu88] shows that the all-zero column sum vector is the most common, and the bound results). These bounds on the number of (1,−1) matrices give us the following bounds on the rate of the two-dimensional code:

$$\text{rate} \leq 1 - \frac{\log\sqrt{\frac{1}{2}\pi N}}{N} < 1 - \frac{\log N}{2N}$$

$$\text{rate} \geq 1 - \frac{\log((N+1)\sqrt{2N})}{N} \approx 1 - \frac{3\log N}{2N}$$

In Chapter 2 we discussed setting the gain of the network to obtain desired network convergence. The gain affects the error-correcting performance of the decoder. Allowing the value of the gain to be time-dependent resulted in a system with less decoder error. In the next section, we detail a scheme for changing this gain that improved the decoder's performance.

4.3.2 Gaingain

Various decoder parameters can be adjusted to yield optimal decoder performance in terms of error rates. The most critical of these is the gain of the network. For example, in simulation of the 2×2 network (Section 4.3.3), a large gain resulted in a network that converged quickly, but that might make errors early that it would not correct. A small gain resulted in a network that was more accurate, but which took longer to converge. These results suggested that a better decoder network might be one in which the gain started small but became large towards the end of the processing. Thus, the network would have time to make a correct decision at the beginning of its computation, and once this decision was made, the network would converge quickly.

This general concept of a variable gain permits many possible formulations. The gain could simply have two values—one from step 1 to step j of the computation, another from step $(j+1)$ to the end of the computation. Or the gain could increase—linearly, exponentially, logarithmically, etc.— at every step, or at every kth step of the computation. Much experimentation showed that of these the best form of a variable gain is one where the gain increases exponentially during the computation. The amount by which the gain was increased (multiplicatively) at each step is the *gaingain.*

Other neural networks with parameters that are time-dependent have been proposed. Touretzky [Tou89] presents a Boltzmann machine that increases the threshold during computation to achieve better network performance. Platt and Barr [PB88] constructed a neural network to perform optimization in which the neurons, which estimate Lagrange multipliers, enforce constraints gradually over time. The gaingain concept is most reminiscent of simulated annealing [KGV83] and simulated mean field [BMM+89] techniques; these techniques involve decreasing the temperature of the system over time until the system "solidifies" at a stable state.

After establishing that we would use a gain that changed exponentially with time, we next needed to determine the best value of the gaingain. To test the neural network as a decoder, the input to the decoder was generated by adding random Gaussian noise of standard deviation σ and mean 0 to each codeword. This system was allowed to evolve until a minimum-change criterion was satisfied. Then the system was said

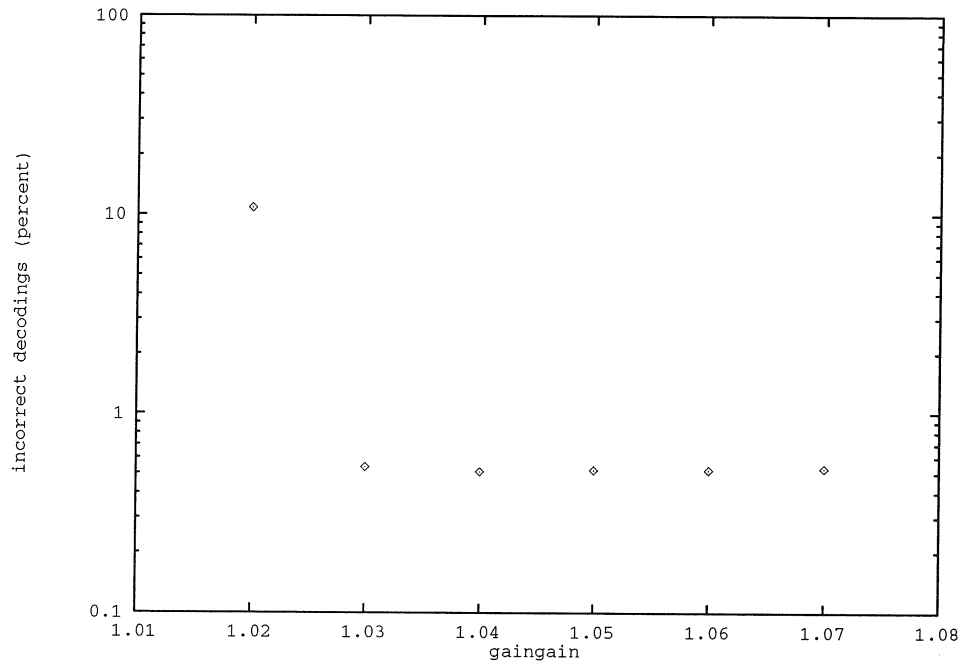


Figure 4.3: Error rate for 4×4 decoders with different values of gaingain, and noise of standard deviation 0.6.

to have converged to a codeword if it was within a small radius of the codeword. We found that a very small gain (and unity, or only slightly greater, gaingain) does not produce a system that behaves well because the minimum-change criterion would be satisfied before the computation had moved the decoder close enough to a codeword (and therefore incorrect decoding would occur). Figure 4.3 presents the error rate for the 4×4 decoder with different values of gaingain. From the figure, we see that there is a minimum value of gaingain above which the error rate is fairly constant, and below which the error rate increases a large amount. For this decoder and this level of noise, the minimum decoder error rate occurs at gaingain= 1.04. The value of the gaingain also affected how fast the simulation converged. Figure 4.4 illustrates the relative speed of convergence of the decoder for the 4×4 code, for different levels of gaingain. From the figures, we see that the fastest decoder (at gaingain= 1.02) was also the least accurate. The most accurate decoder was also one of the slowest decoders. This tradeoff was noticed for all our decoders. Since we wish to investigate

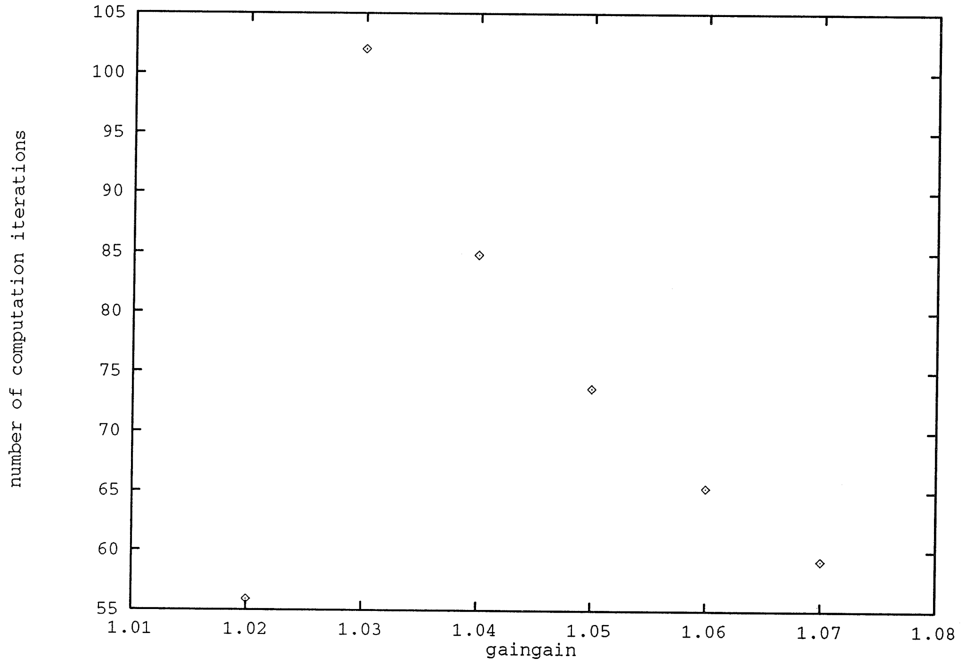


Figure 4.4: Relative convergence speed for the 4×4 decoder for different values of gaingain. The relative speeds are in average number of steps in a Euler simulation of the network with $\sigma = 0.6$.

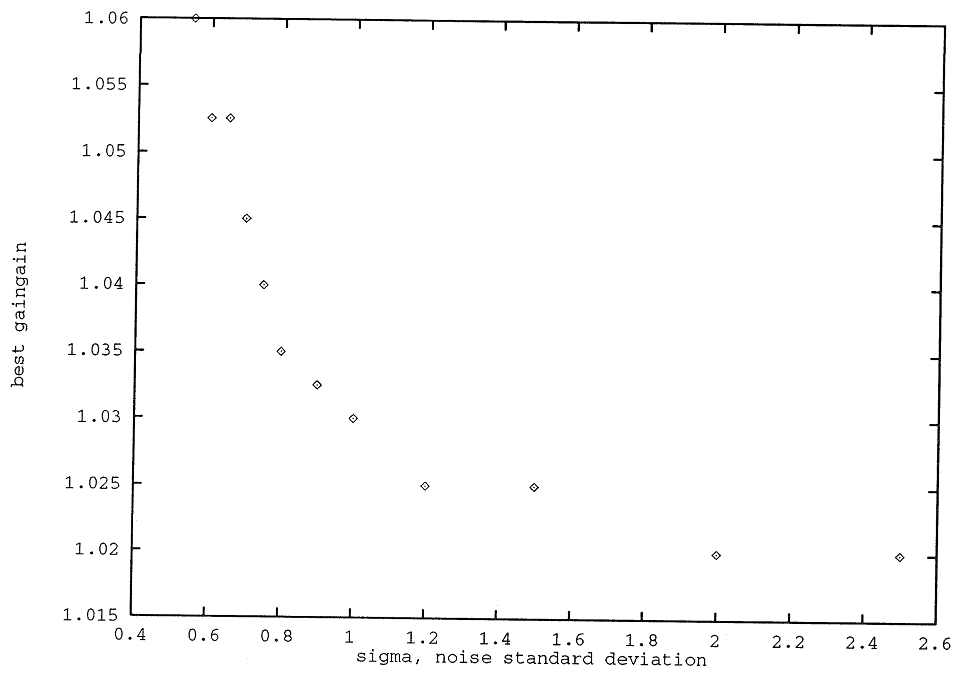


Figure 4.5: The best values of gaingain for the 2×2 decoder for different values of noise.

how accurate our decoder can be, we will always consider the best value of gaingain as that which minimizes the error rate. One could also consider the best value, given a certain time constraint.

Adaptive Gaingain

After much experimentation, it was also observed that the best value of gaingain was dependent on σ (again, the best being that value which minimized the decoder's error rate, for some large number of samples). Figure 4.5 presents the best experimental gaingain for different values of noise for the 2×2 decoder. In normal usage, input to a decoder may include a wide spread of noise variance. Consequently, a decoder that could adapt to the noise of its input would be quite useful. We will see in Section 4.3.3 that the 2×2 code has only two codewords. A measure of the noisiness of the system might be related to how far the input vectors are from the codewords—and how close they are to the hyperplane $\mathbf{h}$ that is equidistant from the two codewords. A point

$\mathbf{x} = x_1, x_2, x_3, x_4$ on the hyperplane satisfies $x_1 - x_2 - x_3 + x_4 = 0$. To calculate the distance from a vector $\mathbf{y}$ to the hyperplane, we project $\mathbf{y}$ on a vector normal to $\mathbf{y}$, such as $\mathbf{a} = \langle 1, -1, -1, 1 \rangle$. The distance is given by

$$d = \frac{\mathbf{a} \cdot \mathbf{y}}{\| \mathbf{a} \|} = \frac{1}{2}(y_1 - y_2 - y_3 + y_4).$$

We proposed several heuristics for setting the gaingain proportional to d, the input vector's distance from the hyperplane. Thus, a vector that began close to the hyperplane (far from the codewords) would be decoded by a network with a small gaingain (allowing the network to choose the correct codeword slowly). We also tried using other noisiness criteria to help determine the value of the gaingain. None of the methods gave systems that behaved consistently better than previous systems. Also, in more complex codes with many codewords, a useful noisiness criterion would be more difficult to construct. (Note that an obvious criterion involves comparing the input vector to each of the codewords. But then our decoding is accomplished! Thus, our criterion must be easy to compute.)

In conclusion, we have found that we can improve on the decoding performance of the constant-gain network by implementing a gain that grows exponentially with time. We consider one-value (vs. adaptive) gaingains, and our decoders utilize that value of gaingain which minimizes the error rate. (Since the best gaingain can vary over sigma, as in Figure 4.5, we will pick an average best value to use on the decoder.) The performance of decoders with gaingain will be illustrated as we discuss individual codes. In the next sections, we give several examples of two-dimensional codes and discuss their performance.

4.3.3 2×2 Code

The simplest two-dimensional code (and corresponding network) is the 2×2 code. Here, $N = 2$ and $i = 2$. Figure 4.6 shows how the nodes (dots in the figure) are placed in constraints (represented by ovals in the figure). The connection matrix for

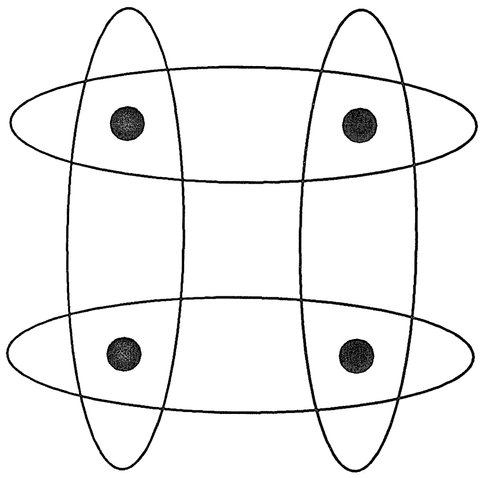


Figure 4.6: Arrangement of the constraints for the 2×2 code. Nodes encircled by an oval share a constraint.

this network is given by:

$$T = -\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Simulating this decoder, we find that the network has two stable states. As a result, the corresponding code has two codewords, $(-1, 1, 1, -1)$ and $(1, -1, -1, 1)$, or in matrix form (corresponding to a T of different form):

$$\begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Clearly, the minimum distance of this code is four.

Performance

Here we present the performance of the 2×2 decoder. As we have seen, this code has two codewords. In the best case, an input vector will be decoded by the system as the closer codeword. Let us consider the probability of error for this best case.

The input to the decoder is the sum of a codeword and a four-dimensional Gaussian noise vector contributed by the simulated noisy channel. Again, the hyperplane $\mathbf{h}$ is equidistant from the codewords. In this system we will make a decoding error if the noise vector moved the input vector across the hyperplane. Thus, the probability of making an error is the probability that the noise, when added to a codeword, will result in a vector that is on the opposite side of the hyperplane from the codeword. The shape of the input vector distribution at each of the two codewords is equivalent. Thus, we need to analyze the distribution around one of these codewords; the results will hold for the second as well. Let us consider the distribution peak around $(1, -1, -1, 1)$.

The probability of error of the system, as calculated in Appendix A, is

$$\begin{aligned} p_e &= 1 - G\left(\frac{2}{\sigma}\right) \\ &= \frac{1}{2} - \operatorname{erf}\left(\frac{2}{\sigma}\right) \end{aligned}$$

where G is the integral of the Gaussian function and erf is the error function [Pap84]. This probability of error indicates the theoretical performance of the 2×2 code. Now we can plot this function against our simulated MLD to calibrate against statistical deviation and simulation inaccuracies.

Figure 4.7 presents the performance of the neural network with constant gain and with an exponentially increasing gain. We see that the network with gaingain has improved performance except at very low noise rates. The low noise behavior is, at least in part, because we are using a gaingain of 1.03 which, as Figure 4.5 illustrates, is a good value for larger values of sigma, but perhaps not as good for small values of sigma. Figure 4.8 shows the performance of the decoder with gaingains of different forms. We see that the exponential form yields the best results. And Figure 4.9 shows the performance of the network with an exponentially increasing gain compared with the maximum likelihood decoder and the theoretical maximum likelihood decoder performance as calculated above. The simulated and theoretical error curves of the MLD are quite similar. From the figures we see that the decoder with gaingain has an error rate very close to that of the simulated MLD.

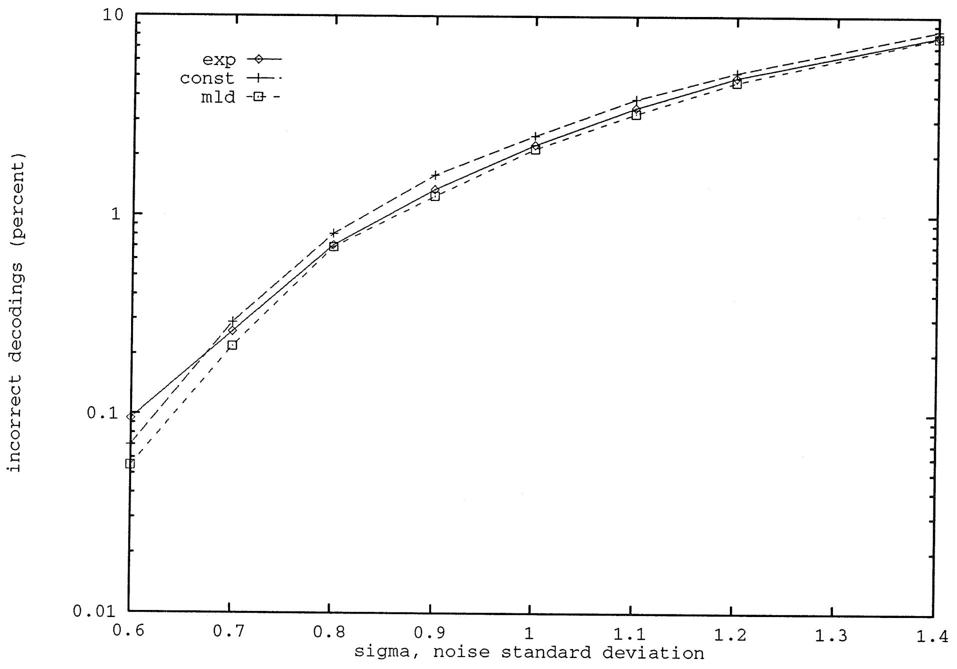


Figure 4.7: Decoder performance of a neural network over 20,000 inputs of the 2×2 code with constant gain (const), and gain that increases exponentially during the computation (exp). Also shown is the performance of a maximum likelihood decoder given the same data (mld).

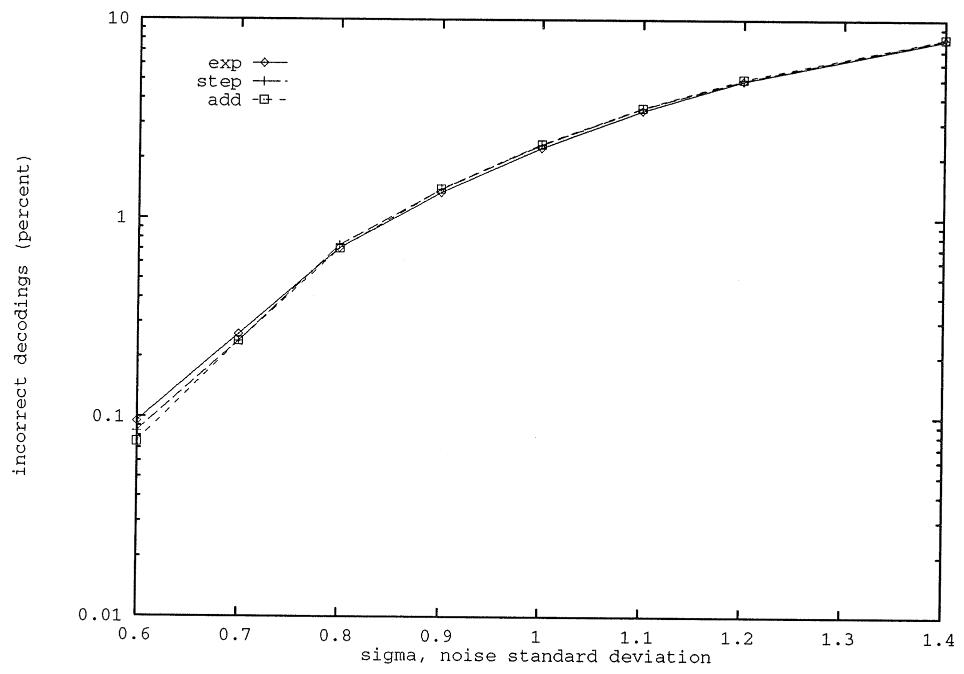


Figure 4.8: Performance of a neural network over 20,000 inputs of the 2×2 code with gain that increases at each step by an additive factor (add), or at only some steps by an additive factor (step), and gain that increases exponentially during the computation (exp).

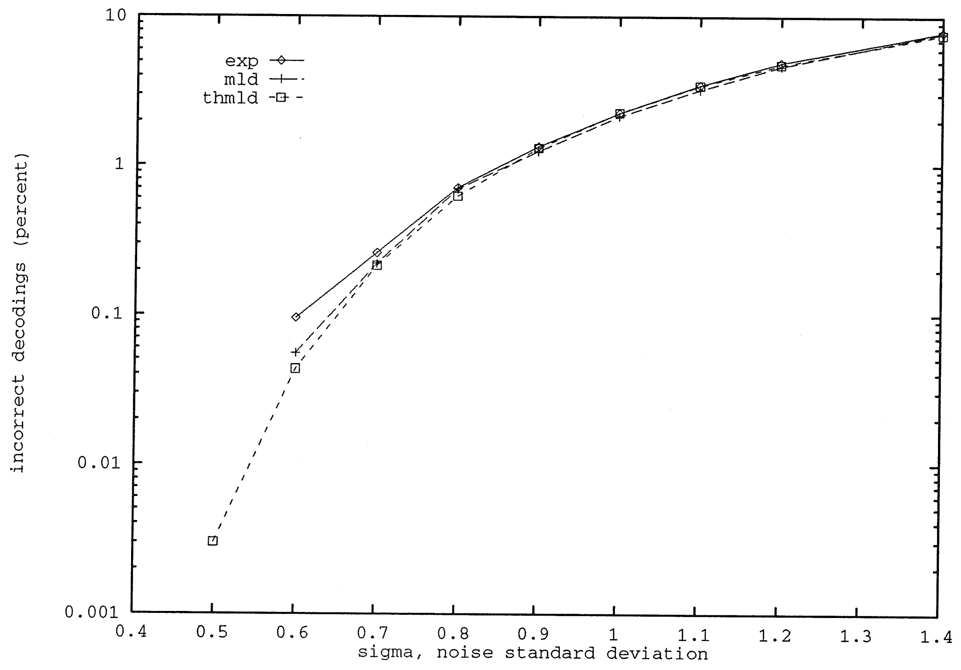


Figure 4.9: Performance of a neural network over 20,000 inputs of the 2×2 code with gain that increases exponentially during the computation (exp), and performance of a maximum likelihood decoder, given the same data (mld), and the theoretical performance as derived above (thmld).

4.3.4 4×4 Code

The decoder for this code has 16 nodes. If we place the nodes in a 4×4 matrix, the KWTA constraints are along the rows and columns of the matrix. We will consider $K = N/2 = 2$ for our analysis.

Codeword Classes

The stable states of this system are those matrices of size 4×4 with two entries that are 1 and two entries that are -1 in each row and column. This system has 90 stable states or codewords. We have found that the 90 codewords can be partitioned into two classes. Two codewords are in the same class if they satisfy these equivalent requirements: (1) We can obtain one from the other by permuting rows and/or columns of the codeword matrix, and (2) they are surrounded by identical codeword space. Two codewords have identical codeword space if the distribution of distances to other codewords is identical. These classes are of interest because they will give rise to different decoder error behavior. Let us first discuss the two classes, and then detail the effect of class on error behavior.

First note that each row in the codeword matrix will be one of $\binom{4}{2} = 6$ possible rows with two 1 entries and two -1 entries. (These rows are listed in Table 2.4.) Also, if a given row exists in the matrix, its negative will also appear as a row in the matrix. (This existence of pairs of matched rows results from a simple application of the pigeon-hole principle to the number of positions (three) left in a column after one row is chosen and since the number of 1's which must appear in each column is fixed.) Thus, we can pair up possible rows with their negative. Let us label the possible rows A, B, C and A', B', C' (where A' is the negative of A). The first codeword class has two sets of identical rows — such a codeword is $AAA'A'$. (The matrix $AAA'A'$ has row A for the first and second row, and row A' for the third and fourth rows.) There are $\binom{3}{1}\binom{4}{2} = 18$ (choose one of A, B, or C; then order the rows) codewords in this class. The second class has four different rows — $AA'BB'$ is such a codeword. There are $\binom{3}{2} \cdot 4! = 72$ (choose two out of A, B, C; then order the rows) codewords in this class. (The example 4×4 codeword in Section 4.3.1 is of the form $ABA'B'$, and thus, is a member of the second class.)

As we stated above, these classes are of interest because they will give rise to different decoder error behavior. The codeword space surrounding codewords of different classes is dissimilar. A codeword of class 1, such as $AAA'A'$, has 16 codewords at Hamming distance 4, and 56 codewords at distance 8 (and symmetrically 16 codewords at distance 12 and 1 codeword at distance 16). In contrast, a codeword of class 2, such as $AA'BB'$, has 12 codewords at distance 4, 16 codewords at distance 6, and 32 codewords at distance 8 (and symmetrically 16 codewords at distance 10, 12 codewords at distance 12, and 1 codeword at distance 16). To analyze the behavior of this code, we must consider the behavior of codewords of both types. In this system, where the code has only 90 codewords, the performance of the simulated decoder could be easily analyzed by repeatedly adding noise to each codeword and decoding this sum. But since the noise is Gaussian, two codewords of the same class will on average give rise to equivalent error rates. And so, alternatively, and of particular interest for larger codes, we could test the decoder with two noisy words — one of class 1 and the other of class 2.

Performance

This code has 90 codewords of length sixteen and has a minimum distance of four. Performance curves for this system are given below. First we present the performance of neural networks with different values of gaingain (Figure 4.10). Note that for values of the gaingain greater than or equal to 1.03, the performance of the system was fairly constant. For small values of the standard deviation of the noise, a larger gaingain gave slightly better results. In Figure 4.11 we see that the MLD and the neural network decoder have similar error rates. Also in Figure 4.11, we compare the neural network with gaingain to another with constant gain. *The network with gaingain outperforms the constant gain system at all levels of sigma.*

4.3.5 6×6 Code

Here we discuss a system of 12 constraints on 36 nodes. Again we will use KWTA constraints with $K = N/2$ ($K = 3$).

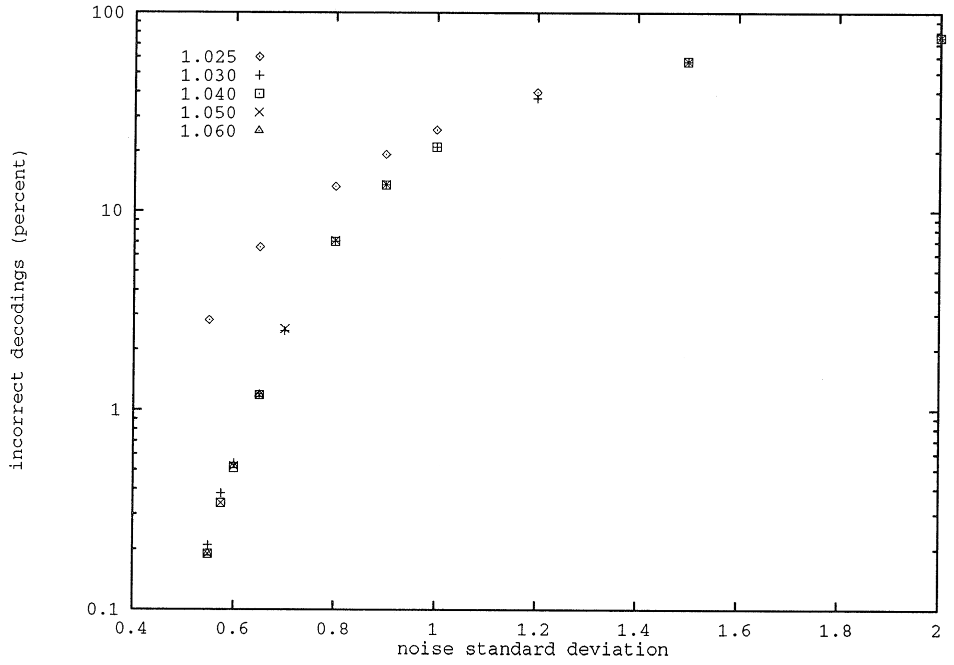


Figure 4.10: The performance of decoders with different gaingain levels. Each decoder was presented with 9000 inputs of the 4×4 code at each value of sigma.

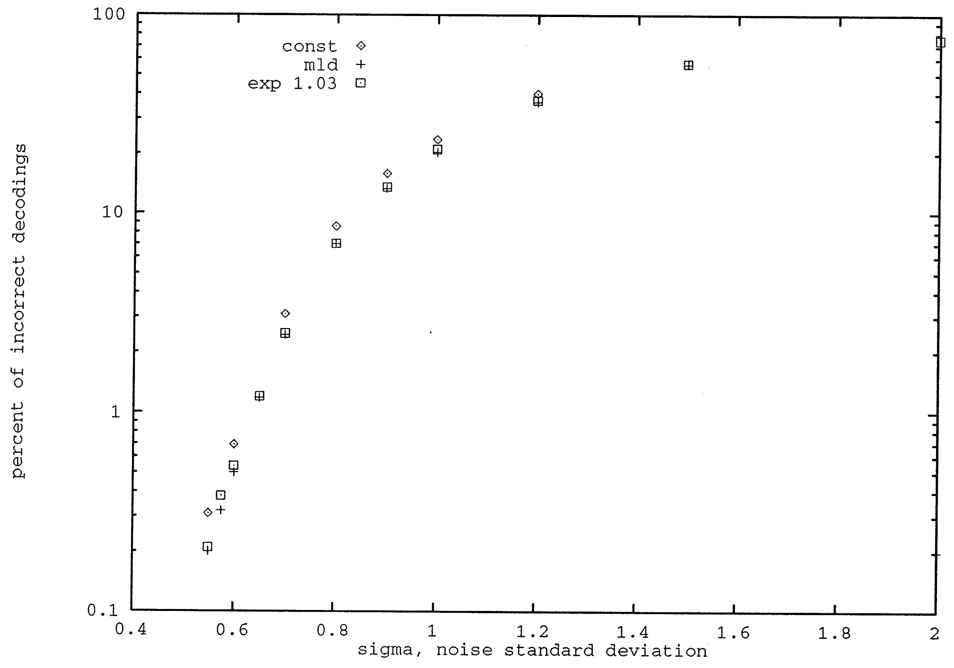


Figure 4.11: Performance of neural networks with constant gain (const), with an exponentially increasing gain (exp 1.03) and a maximum likelihood decoder (mld). Each was presented with identical data, 9000 inputs of the 4×4 code at each value of sigma.

A	1	1	1	-1	-1	-1	A'	-1	-1	-1	1	1	1
B	1	1	-1	1	-1	-1	B'	-1	-1	1	-1	1	1
C	1	1	-1	-1	1	-1	C'	-1	-1	1	1	-1	1
D	1	1	-1	-1	-1	1	D'	-1	-1	1	1	1	-1
E	1	-1	1	1	-1	-1	E'	-1	1	-1	-1	1	1
F	1	-1	1	-1	1	-1	F'	-1	1	-1	1	-1	1
G	1	-1	1	-1	-1	1	G'	-1	1	-1	1	1	-1
H	1	-1	-1	1	1	-1	H'	-1	1	1	-1	-1	1
I	1	-1	-1	1	-1	1	I'	-1	1	1	-1	1	-1
J	1	-1	-1	-1	1	1	J'	-1	1	1	1	-1	-1

Table 4.1: Legal rows for the 6×6 codewords.

Codeword Classes

This code has a total of 297,200 codewords. There are $\binom{6}{3} = 20$ possible rows for our matrix codewords. Let us label them $A, B, C, D, \ldots, J$ and $A', B', C', D', \ldots, J'$, where again A' is the negative of A. They are listed in Table 4.1.

Experimentally, we find that six codeword classes exist. Appendix B enumerates these six classes. Table 4.2 lists the six classes by representative codewords and gives the size of each class.

Also, Table 4.2 shows the number of codewords at given Hamming distances from a codeword of each class. There are no codewords at a distance of less than four from each other and there are no codewords that are an odd distance from each other. The number of codewords at distance i from a codeword is equivalent to the number at distance $36 - i$. We would expect a codeword of class 1 to behave quite differently than a codeword of class 6 in small noise environments, since a codeword of class 1 has 81 codewords at distance four and no codewords at distance six, while a codeword of class 6 has 51 codewords at distance four and 224 codewords at distance six. This difference will be illustrated in the next section.

In summary, the importance of the data given in the table is that the minimum distance of this code is four (since there are no codewords at a distance less than four from each other) and that there are six codeword classes. We will discuss in the next section how we can take advantage of the class structure of the codewords.

distance from codeword:								codeword:	
4	6	8	10	12	14	16	18	class	number
81	0	2754	0	32850	0	112914	0	$AAAA'A'A'$	200
65	144	1762	4544	18386	32592	60418	61376	$AAA'A'BB'$	16200
57	200	1506	5280	16850	35112	57162	64864	$AA'CC'FF'$	86400
57	196	1518	5280	16794	35268	56886	65200	$AA'AD'JF'$	43200
53	216	1446	5440	16546	35608	56474	65632	$CC'AHB'F'$	129600
51	224	1419	5496	16481	35664	56424	65680	$ABJC'D'E'$	21600

Table 4.2: Number of codewords at a given distance from 6×6 codewords of each class.

Performance

This code has 297,200 codewords of length 36 ($= 6 \times 6$) and has a minimum distance of four. The best neural network decoder for this code used a gaingain of 1.07. This value of gaingain produced the best decoder performance for all codeword classes and all values of sigma, the standard deviation of the noise. We found the performance of the decoder to be relatively constant for gaingain values greater than 1.04. The network with gaingain set to 1.07 reduced the error difference between the constant gain neural network and the *best-guess* decoder, described below, by more than half.

To test the performance of this decoder, we tested the performance of six codewords — one of each class. We present the performance of the neural network with gaingain set to 1.07, and a gain of 1.0, in Figure 4.12. Each data point represents 1000 decoded inputs. We see that codewords of different class do indeed give rise to different decoder performance. The curve represents the weighted average performance (according to the number of codewords in each class), over all six codewords, of the decoder.

The maximum likelihood decoder implemented in previous sections compared an input with each of the codewords to find the closest match. Because of the size of this code, such a decoder is not practical. Here we use a best-guess decoder, which compares each decoder input to the corresponding channel input's closest neighbors. Thus, if codeword R is sent over the channel, and string S is received by the decoder, the best-guess decoder will find the codeword at distance 0, 4, 6 or 8 from R that is

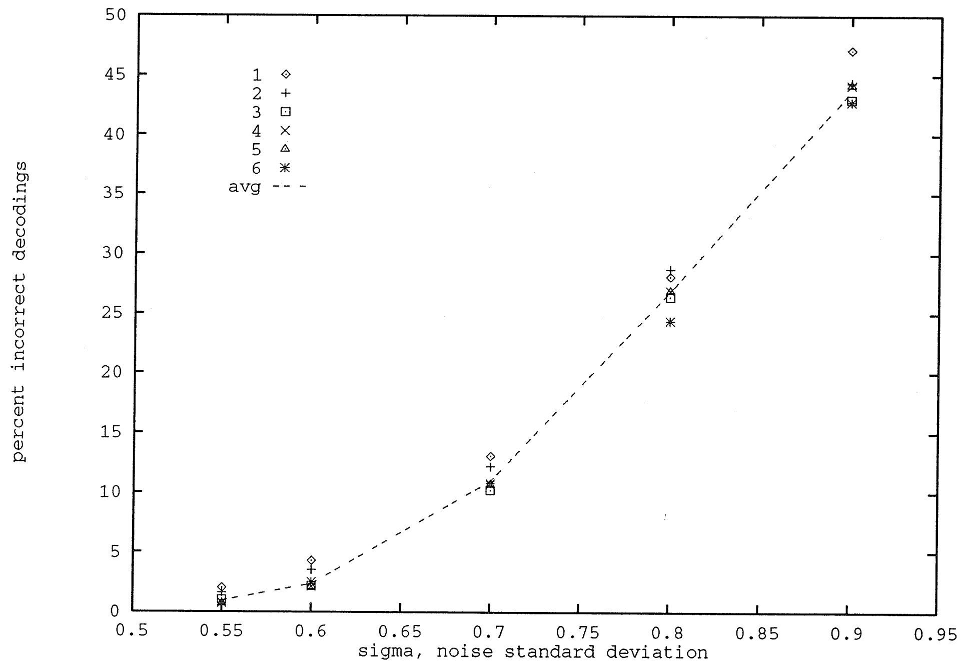


Figure 4.12: The performance of the neural network decoder with gain-gain set to 1.07 for 6×6 codewords of different classes and the weighted average (avg) performance.

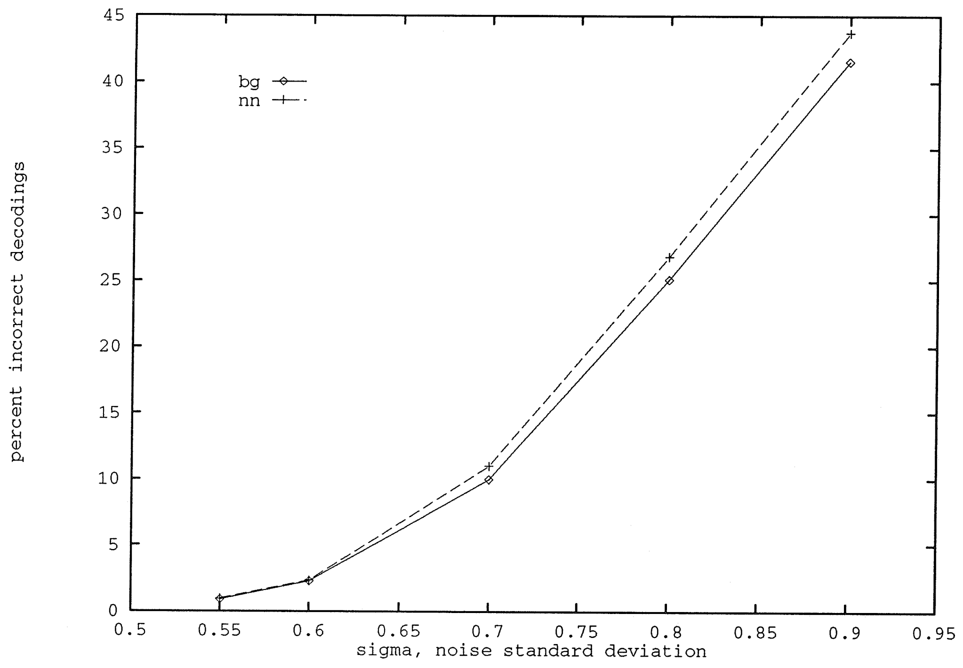


Figure 4.13: The weighted average performance of the neural network (nn) decoder and the best-guess (bg) decoder for the 6×6 code.

closest to the string S. (Note that this decoder uses information about the channel input, which is not usually available.) By using codewords at distance 0, 4, 6 or 8 from the input codeword, the best-guess decoder makes less than 3000 comparisons per decoding. Most of the noisy vectors from the channel will be closer to these codewords than to codewords that are at a greater (than 8) distance from the initial codeword, and thus, the best-guess decoder will have error-correcting behavior similar to the MLD. Figure 4.13 compares the performances of the weighted average (over all six codeword classes) neural network decoder and the best-guess decoder. The neural network's performance is close to that of the best-guess decoder.

4.3.6 Summary

Table 4.3 lists some two-dimensional codes. The codes we discussed in our examples utilized even values of N because codes with $K = N/2$ are symmetric (i.e., if A is a codeword, then so is $-A$), and the rate is maximum. Table 4.3 also gives a code

code	# codewords	rate	# codeword classes
2×2	2	0.25	1
4×4	90	0.406	2
6×6	297,200	0.505	6
7×7	68,938,800	0.5314	?
8×8	1.17×10^{11}	0.5745	> 13

Table 4.3: Some two-dimensional codes.

with $N = 7$ and $K = 3$, and statistics on the 8×8 code. By extending codeword class forms (such as $AAAA'A'A'$ for the 6×6 code), one can count at least 13 codeword classes for the 8×8 code.

We found that the two-dimensional hypercube codes could be decoded on neural networks at an error rate very close to maximum likelihood or best-guess decoders.

4.4 Three-Dimensional Hypercube

If we now arrange the $M = N^3$ nodes in a cube, the KWTA constraints are parallel to the horizontal, vertical and depthwise edges of the cube. Each node is involved in three constraints. There are a total of $iN^2 = 3N^2$ constraints. The minimum distance of a three-dimensional hypercube code is eight.

4.4.1 4×4×4 Code

As a case study, we now consider the 4×4×4 code, comprised of 51,678 codewords of length 64. Table 4.4 lists the ten codeword classes of the 4×4×4 codewords. Each letter of the codeword represents a row (take the binary representation of the hexidecimal letter and change 0's to −1's). 4×4×4 codewords consist of four matrices of size 4×4. The first matrix of the class two codeword $CC33CA3533CC35CA$, is given by the first four rows, $CC33$ (i.e., the first two rows are C =(1 1 −1 −1), and the second two rows are 3= (−1 −1 1 1)). The second matrix of the codeword is $CA35$, the third matrix is $33CC$, and the fourth matrix is $35CA$.

Table 4.5 gives the codewords at each distance from a codeword of a given class.

	codeword class	number of codewords
1	$CC33CC3333CC33CC$	54
2	$CC33CA3533CC35CA$	3456
3	$CC33C33C33CC3CC3$	648
4	$CC33C33C35CA3AC5$	10368
5	$CC33C35A35AC3AC5$	20736
6	$CC33A55A33CC5AA5$	1296
7	$CC33A55A3AC553AC$	5184
8	$CC33A56933CC5A96$	2592
9	$CA35A56939C6569A$	6912
10	$CA35A35C35CA5CA3$	432

Table 4.4: Codeword classes for the 4×4×4 code. Each hexadecimal letter of the codeword gives a row, where 0's are converted to −1's.

From the table, we see that there are no codewords at distance less than eight, or at any odd distance, or at distance 10, from each other. Thus, the minimum distance of this code is 8.

4.5 General-Dimensional Hypercube

The rate of these codes can be bounded only very loosely. One-dimensional hypercube codes have known rate $((\log\binom{N}{k})/N)$. We have shown that the rate of two-dimensional codes behaves like $1 - \frac{c\log N}{2N}$ for a constant c.

The minimum distance of these codes (also discussed in detail in Section 5.3) is 2^i.

The number of constraints for the one-dimensional code is 1. If we are given that the number of constraints for the $(i-1)$-dimensional code is $C_{(i-1)}$, then

$$C_i = C_{(i-1)}N + N^{i-1}$$

The first term results because we repeat the nodes (and their constraints) of the $(i-1)$-dimensional decoder N times in the i-dimensional decoder. The second term counts the constraints that connect these N repetitions. From this recursion, the

	distance from codeword:												
	0	8	12	14	16	18	20	22	24	26	28	30	32
1	1	64	0	0	1232	0	0	0	9792	0	0	0	29500
2	1	44	48	32	616	264	1068	864	3740	3264	5988	5432	8956
3	1	48	64	0	708	0	1472	0	6240	0	8320	0	17972
4	1	36	56	64	436	288	1176	1168	3468	3184	6480	5152	8660
5	1	28	54	104	312	364	1090	1508	3084	3740	5784	5996	7548
6	1	40	64	0	608	0	1568	0	6056	0	10080	0	14844
7	1	32	64	64	380	288	1208	1168	3592	3184	6536	5152	8340
8	1	28	48	96	380	256	1168	1408	3236	3520	6496	4576	9252
9	1	18	48	136	214	416	1148	1664	2662	4256	5588	6072	7232
10	1	32	96	0	448	0	1888	0	5728	0	10560	0	14172

Table 4.5: Number of codewords at a given distance from 4×4×4 codewords of each class.

number of constraints is $i\frac{M}{N} = iN^{i-1}$.

For the codes that we have studied, it appears that codes of higher dimensionality, in general, exhibit more codeword classes than codes of lower dimensionality.

4.5.1 Comparisons

Figure 4.14 compares the performance of the one-dimensional codes with $N = 4$, 6 and 8 (and $k = N/2$) and the two-dimensional codes with $N = 2$, 4 and 6. Figure 4.15 compares the performance of the two-dimensional 6×6 code with the Hamming code of length 15 and the constant weight of length 27 found in [Kos91]. (The second two codes are decoded by an MLD.) The vertical axis of these figures marks the decoded bit error probability. On the horizontal axis we measure the bit signal–to–noise ratio (SNR), plotted in dB, and given here as

$$\text{SNR} = 10\log_{10}(E_b/N_o)$$

where

$$\frac{E_b}{N_o} = \frac{1}{2\sigma^2 R}$$

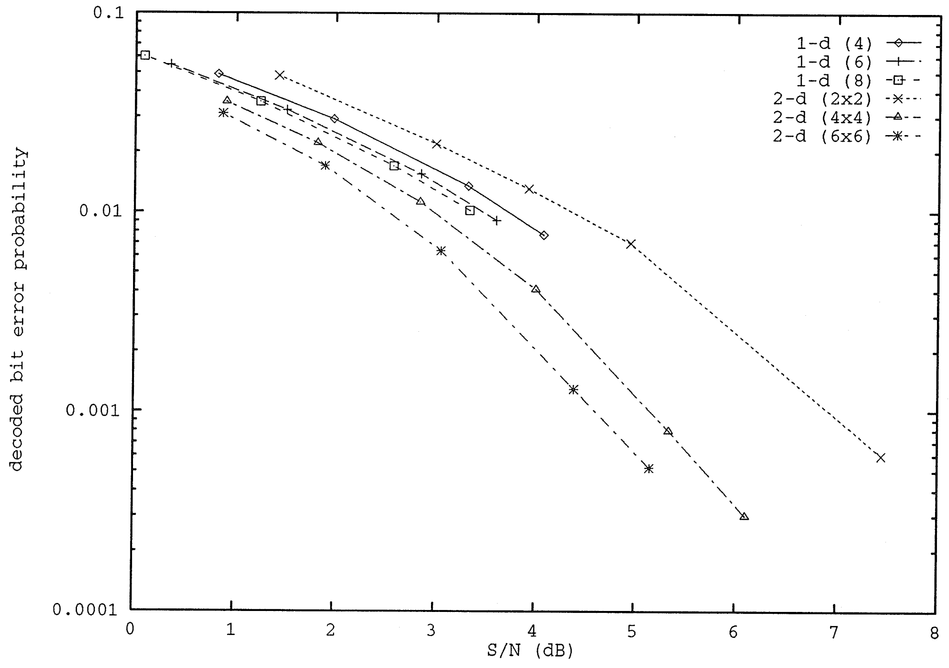


Figure 4.14: The performance of several one-dimensional (1-d) codes (of size $N = 4$, 6 and 8) and of several two-dimensional (2-d) codes (of size 2×2, 4×4 and 6×6) compared.

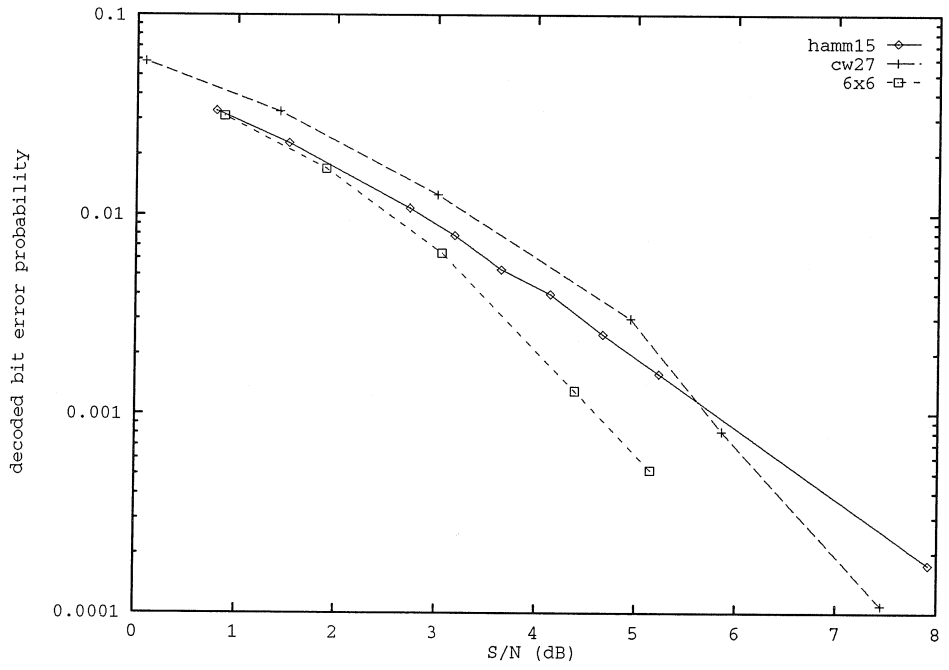


Figure 4.15: The performance of the 6×6 code, the Hamming code of length 15, and a constant weight code of length 27.

and E_b is the energy per bit of the signal, N_o is the noise density and R is the information rate. For more on signal–to–noise ratios see [McE77] and [PB88]. Ideally, one would like a code to have a small, decoded bit error probability (good performance) at a small SNR (not much power expended). Thus, the figure shows that for one-dimensional codes, the performance improves as N grows. Similarly, for two-dimensional codes, we see a marked improvement in code performance with increased constraint size (N).

Now let us compare some of the hypercube codes with the random codes from Chapter 3. For systems of sixteen nodes, we have the one-dimensional hypercube code of $\binom{16}{8} = 12,870$ codewords at a minimum distance of 2, and the two-dimensional code of 90 codewords at a minimum distance of 4. The largest random codes with minimum distances of 2 and 4 had 270 and 28 codewords, respectively. These comparisons lead us to conclude that the hypercube configuration of constraints is a good one.

In terms of general constant weight codes, Brower et al., [BSS90] list the best-known code of sixteen-bit codewords, weight eight (i.e., $k = 8$), and minimum distance of 4, with 1170 codewords. The two-dimensional 4×4 hypercube code has only 90 codewords, but it does have the advantage of a known almost–maximum likelihood decoder.

In conclusion, we have presented a family of decoders constructed by overlapping KWTA constraints on the nodes of the decoder. If we arrange the nodes of the decoder in an i-dimensional hypercube, the placement of the constraints is along the horizontal, vertical, depth-wise,...etc. edges of the hypercube. These decoders perform almost–maximum likelihood decoding on a family of constant weight codes. In the next chapter, we will consider another family of decoders constructed by placing KWTA constraints on the nodes of a neural network in a regular fashion.

Chapter 5

Net-generated codes

5.1 Introduction

We present another family of codes defined by the stable states of a neural network constructed by the regular placement of overlapping KWTA constraints on the nodes. Here we use combinatorial nets to specify the nodes in the KWTA constraints. We will first briefly discuss nets and then present an example net.

A *net* of order n consists of a set X of n^2 points and a family of parallel classes (of lines), each class being a partition of the n^2 points into n n-subsets (called lines) such that two lines from different classes meet in exactly one point. Dénes and Keedwell [DK74] discuss nets, projective planes, and affine planes, as well as the relationship of these geometric constructions to each other, and to many other interesting constructions. Next, we illustrate constructing a net from a projective plane.

If n is the integer power of a prime number, we can use a projective plane to generate a net with $(n+1)$ classes. For example, in Table 5.1 we have the projective plane of order $n = 2$. It consists of $n^2 + n + 1 = 7$ points and $n^2 + n + 1 = 7$ lines, where each line has $n + 1 = 3$ points and any 2 lines intersect in exactly one point.

We can generate a net of 3 (i.e., $n+1$) classes in the following way: Pick one line of the projective plane. Without loss of generality, we select the first line. Eliminate the line, and also eliminate the points in that line from all the lines in the projective plane, as shown in Table 5.2. Renumber the remaining $n^2 + n + 1 - (n + 1) = n^2$

	points:						
	1	2	3	4	5	6	7
	1	1		1			
		1	1		1		
			1	1		1	
lines:				1	1		1
	1				1	1	
		1				1	1
	1		1				1

Table 5.1: Projective plane of order 2. (Points are numbered for clarity.)

projective plane's points:	1	2	3	4	5	6	7	
	$\cancel{1}$	$\cancel{1}$		$\cancel{1}$				
		$\cancel{1}$	1		1			
			1	$\cancel{1}$		1		
lines:				$\cancel{1}$	1		1	
	$\cancel{1}$				1	1		L_1
		$\cancel{1}$				1	1	
	$\cancel{1}$		1				1	L_2
net's points:			1		2	3	4	

Table 5.2: Constructing a net from a projective plane.

classes:	lines:
1	(2,3) (1,4)
2	(1,2) (3,4)
3	(1,3) (2,4)

Table 5.3: Lines of a net. Each class has n $(= 2)$ lines that partition the n^2 points. Points on a line are enclosed by parentheses.

points. These are the points of the net. The first class of the net is composed of the reduced lines that previously contained the first point (old label 1) of the projective plane. In our example, this class contains two lines: L_1 consists of points 2 and 3, and L_2 consists of points 1 and 4. The remaining classes of the net are formed in a corresponding manner from the other points of the first line of the projective plane. Table 5.3 lists the lines of the three classes for this example.

Our neural network has n^2 nodes which correspond to the points in the net. If we impose a KWTA constraint on the nodes whose corresponding points are on a line, a net can be used to generate a family of decoders. If we use all $(n + 1)$ classes to specify KWTA constraints, we find that the nodes are over-constrained and the network has no stable states. We can obtain different decoders by using 1, 2,..., up to n classes to specify constraints on the nodes.

From the net construction process, we see that a network, of n^2 nodes, specified by a classes has a constraints per node, and an constraints overall. In the next section, we discuss some general characteristics of net-generated decoders, and consider some specific codes. Then, we will detail in Section 5.3 a general method for determining the minimum distance of codes generated by overlapping KWTA constraints. Finally, in Section 5.4 we compare these net-generated codes with other codes.

5.2 Net-Generated Codes

We first present general observations about net-generated codes. Then, we present several example nets, and discuss codes resulting from these nets. In Section 5.2.6 we list all the net-generated codes that we have considered.

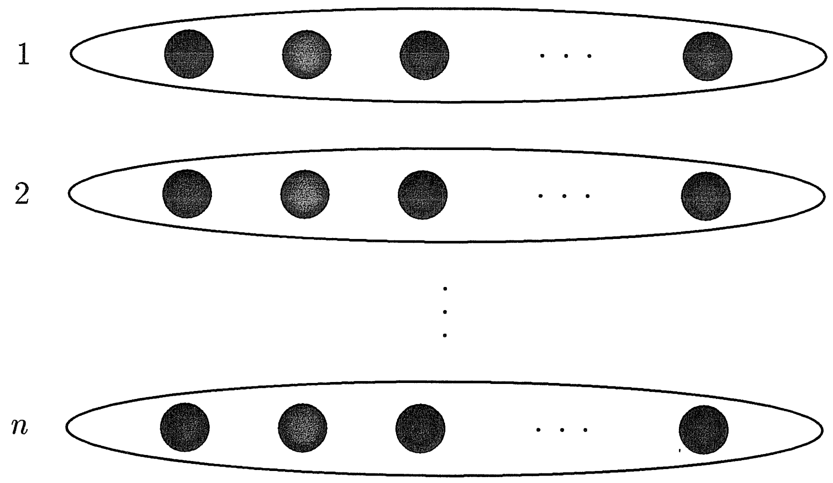


Figure 5.1: Placement of KWTA constraints on nodes for one-class code. Ovals around nodes represent constraints.

5.2.1 General Net Codes

Place the n^2 nodes of a net-generated decoder in an $n \times n$ matrix. A single class of a net partitions n^2 points into n disjoint lines of n points each. If we arrange the n^2 points in an $n \times n$ matrix, the constraints can be placed along the rows as in Figure 5.1. These n constraints are unconnected, yielding codes with a minimum distance of two, and $\binom{n}{n/2}^n$ codewords, each of length n^2. The decoding ability of this code compares with the (one-dimensional) KWTA code of Chapter 2. Because of their similar rate and minimum distance, the one-class codes have no advantage over the KWTA codes.

The decoder constructed with two classes of a net of size n^2 is identical to the two-dimensional decoder discussed in Section 4.3. The two-class codes have a minimum distance of four, and a rate that behaves like $1 - \frac{c \log n}{2n}$ for a constant c. The placement of constraints over nodes is illustrated in Figure 5.2.

The three-class decoder is a system of $3n$ constraints on n^2 nodes. If the nodes are arranged in an array, the constraints can be placed along the rows, columns, and diagonals of the array. The diagonal constraints are illustrated in Figure 5.3. Figure 5.4 illustrates the placement of all the constraints for a three-class decoder of 16 nodes. As we will see in Section 5.3, the minimum distance of a three-class code

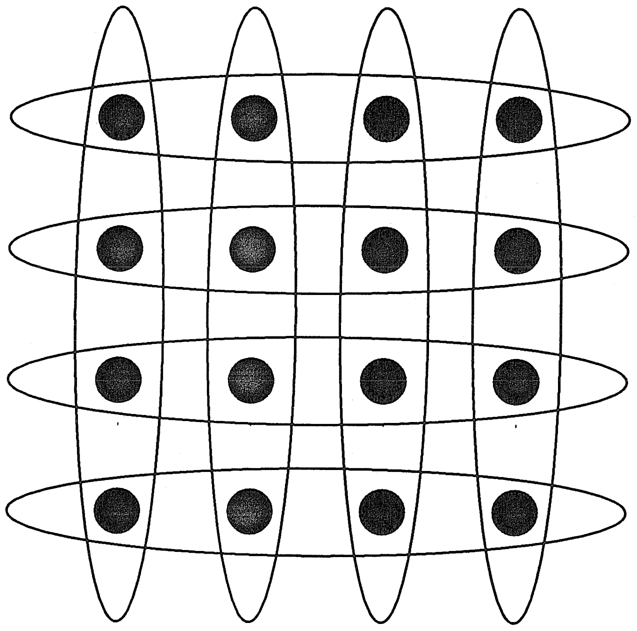


Figure 5.2: Placement of KWTA constraints on 16 nodes for the two-class code. Ovals around nodes represent constraints.

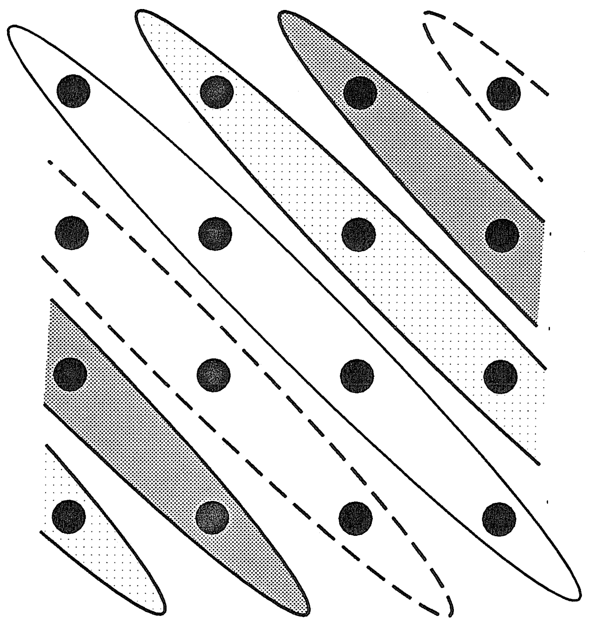


Figure 5.3: Placement of the third class of KWTA constraints for three-class codes. Ovals around nodes represent constraints for the third class.

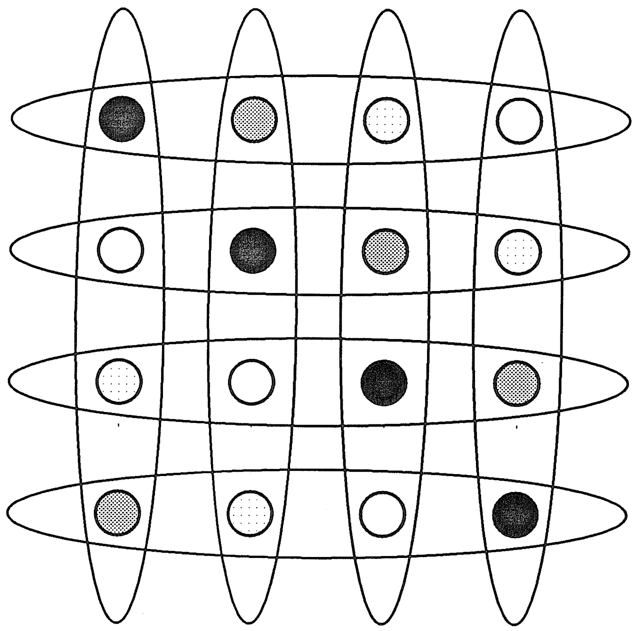


Figure 5.4: Placement of KWTA constraints on 16 nodes for three-class code. Ovals around nodes represent constraints for the first two classes. The third class constrains nodes that are the same shading.

is at least 6. The rate can be only bounded loosely.

For a general a-class code, the rate is hard to determine. However, for $a = n$ and $a = (n-1)$ we can find the number of codewords as follows. Let $N_n(a)$ be the number of codewords of a code generated by a net on n^2 points and a classes. Experimentally, we have found that $N_n(n) = \frac{1}{2}N_n(n-1)$. In addition, $N_n(n) = \binom{n}{k}$ where $k = n/2$ for n even, and $k = (n-1)/2$ for n odd. Thus, the rate of n-class codes is

$$\frac{\log \binom{n}{k}}{n^2} \approx \frac{1}{n}$$

and, similarly, the rate of $(n-1)$-class codes is approximately $(1+n)/n^2$.

Recall that the number of different rows that can occur in a codeword is $\binom{n}{k}$ (since the nodes in the row are in a KWTA constraint). We have observed that the number of codewords for an n-class code is also equal to $\binom{n}{k}$. Therefore, because nodes in a row can be reordered, one row of a codeword uniquely determines the codeword. In fact, in an n-class code, (since rows are not special) once any set of n nodes in a KWTA constraint is determined, the values of the remaining $(n^2 - n)$ nodes are fixed.

In Section 5.3.2 we find that the minimum distance of an a-class code is at least $2a$. Experimentally, we have found the minimum distances of a class codes to be $2a$ or $2(a+1)$.

Since the minimum distance of an a-class code on n^2 points is at least $2a$, and a can be great as n, this implies that we can construct an n-class code with a minimum distance of at least $2n$.

Next, we consider codes constructed from nets of size sixteen, twenty-five, thirty-six and forty-nine.

5.2.2 The Net of Sixteen Points

Here we will discuss several decoders constructed from the net on sixteen points. Recall that, in Section 4.3 we discussed the decoder constructed from two classes of constraints. The corresponding code (the 4×4 code) has ninety codewords of two codeword classes, and a minimum distance of four.

Figure 5.5 gives the performance of the net decoder of 16 nodes constructed with three classes of constraints. The neural network decodes at an error rate close to that of the MLD's error rate. The three-class code has twelve codewords of a single class and a minimum distance of eight. Table 5.4 lists the codewords.

Figure 5.6 gives the performance of the net decoder of 16 nodes constructed with four classes of constraints. The corresponding code has six codewords of a single class and a minimum distance of eight. These codewords are a subset of the codewords of the three-class code, and are indicated in Table 5.4 by an asterisk. As shown in Figure 5.6, the performance of the neural network decoder is almost identical to the MLD's performance.

In Table 5.5 we list the codes constructed from the net on sixteen points.

Codewords:

−1	−1	−1	−1	1	1	−1	1	1	−1	−1	1	1	1	1	−1	
−1	−1	−1	1	1	1	1	−1	1	−1	1	−1	−1	1	−1	1	*
−1	1	−1	−1	−1	1	−1	−1	1	1	1	−1	1	−1	1	1	
−1	1	−1	−1	1	−1	−1	1	−1	1	1	1	−1	1	−1	1	
−1	1	1	−1	−1	−1	1	−1	1	−1	−1	1	1	1	−1	1	*
−1	1	1	1	1	1	−1	−1	−1	−1	1	1	1	−1	−1	−1	*
1	−1	−1	−1	−1	−1	1	1	1	1	−1	−1	−1	1	1	1	*
1	−1	−1	1	1	1	−1	1	−1	1	1	−1	−1	−1	1	−1	*
1	−1	1	1	−1	1	1	−1	1	−1	−1	−1	1	−1	1	−1	
1	−1	1	1	1	−1	1	1	−1	−1	−1	1	−1	1	−1	−1	
1	1	1	−1	−1	−1	−1	1	−1	1	−1	1	1	−1	1	−1	*
1	1	1	1	−1	−1	1	−1	−1	1	1	−1	−1	−1	−1	1	

Table 5.4: The three-class code for the net on sixteen points. Codewords also in the four-class code are indicated by *.

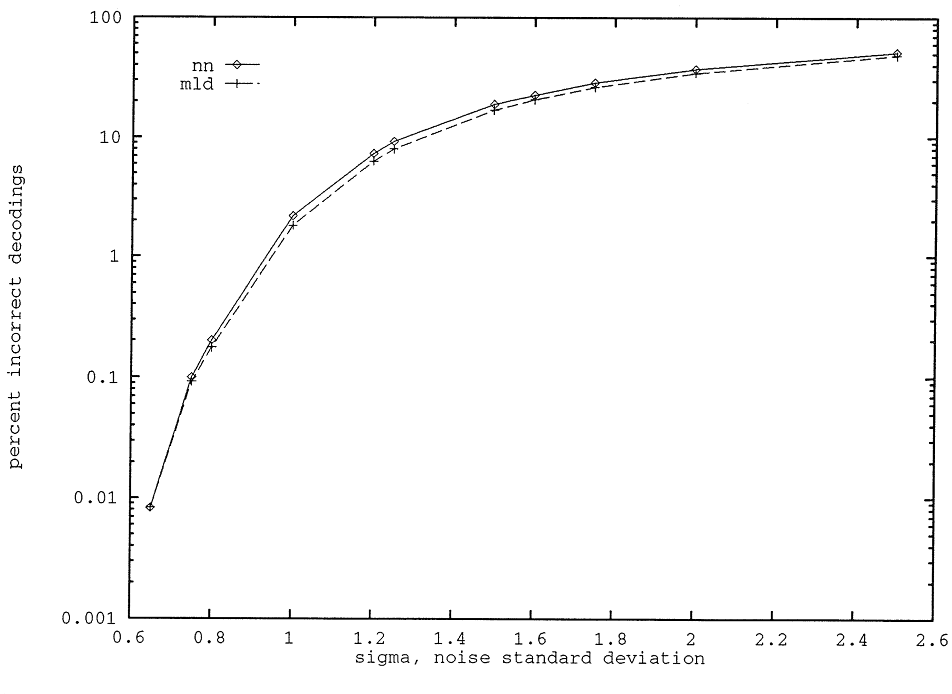


Figure 5.5: Neural network and maximum likelihood decoder performance, over 12,000 inputs per noise level, for the three-class net code on sixteen points. The neural network has a gaingain of 1.05.

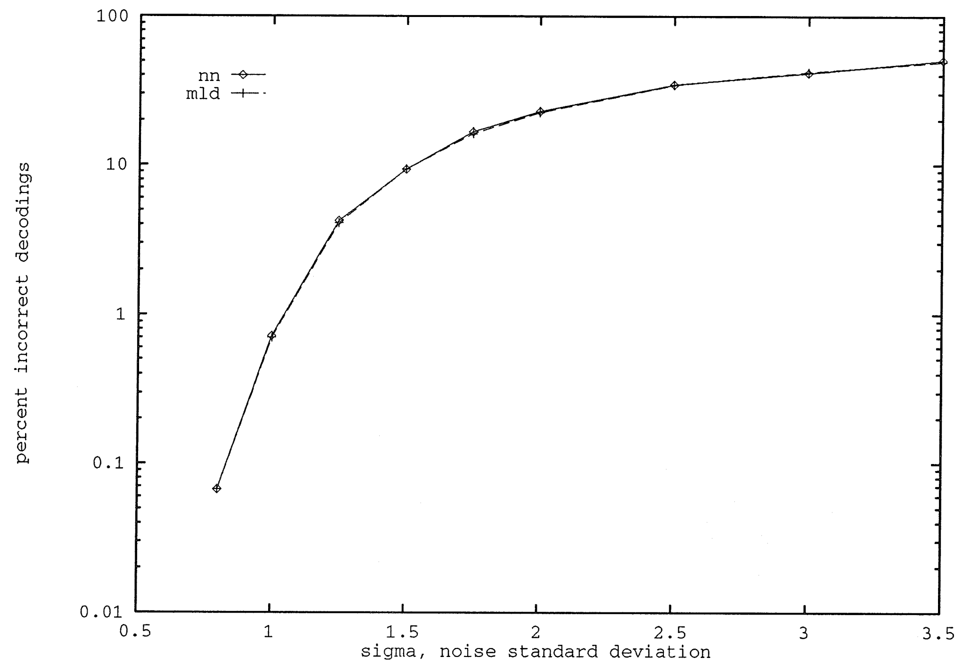


Figure 5.6: Neural network and maximum likelihood decoder performance, for 6000 inputs per noise level, for the four-class net code on sixteen points. The neural network has a gaingain of 1.05.

# of classes of net	number of codewords	rate	minimum distance	# codeword classes
1	1296	0.6462	2	1
2	90	0.4057	4	2
3	12	0.2241	8	1
4	6	0.1616	8	1

Table 5.5: Codes constructed from the net on sixteen points.

# of classes of net	number of codewords	rate	minimum distance	# codeword classes
1	10^5	0.6644	2	1
2	2040	0.4398	4	?
3	130	0.2809	6	2
4	20	0.1729	10	1
5	10	0.1329	10	1

Table 5.6: Codes constructed from the net on twenty-five points.

5.2.3 The Net of Twenty-five Points

The net on twenty-five points enforces KWTA constraints with $k = 2$. The three-class code has 130 codewords of two codeword classes. The four-class code has 20 codewords, each at distance 10 from six codewords, at distance 12 from ten codewords and at distance 20 from the remaining three codewords. The five-class code has 10 codewords, namely, all those from the four-class code that were not at distance 12 from each other. In Table 5.6 we list the codes constructed from the net on twenty-five points.

5.2.4 The Net of Thirty-six Points

Because six is not the power of a prime, we cannot construct a projective plane of order six. It has been shown that there exists a correspondence between k classes of a net of order n and $k-2$ *mutually orthogonal Latin squares* (MOLS) of size $n \times n$ [DK74]. Therefore, the nonexistence of two MOLS of size 6×6 implies that for a net on thirty-six points, less than four classes exist. We constructed three classes of this net, but found that imposing the constraints from all three classes on the nodes of a neural network resulted in a decoder with no stable states.

In Table 5.7 we list the codes constructed from the net on thirty-six points.

# of classes of net	number of codewords	rate	minimum distance	# codeword classes
1	20^6	0.7203	2	1
2	297,200	0.5050	4	6

Table 5.7: Codes constructed from the net on thirty-six points.

# of classes of net	number of codewords	rate	minimum distance	# codeword classes
1	35^7	0.7328	2	1
2	68,938,800	0.5314	4	?
3	310,198	0.3723	6*	?
4	5726	0.2548	8*	?
5	399	0.1763	12	3
6	70	0.1251	14	1
7	35	0.1047	14	1

Table 5.8: Codes constructed from the net on forty-nine points. Distances marked with $*$ are lower bounds.

5.2.5 The Net of Forty-nine Points

The code formed with 5 classes of the net on 49 points has 399 codewords in 3 classes. The code formed with 6 classes of the net has codewords of a single class. A codeword from this code has twelve codewords at distance 14, thirty-five codewords at distance 24, eighteen codewords at distance 28 and the remaining four codewords at distance 42. The code formed with 7 classes of the nets has thirty-five codewords; interestingly, these codewords are the thirty-five codewords from the 6-class code that were not at distance 24 from each other. (Thus, the difference code, formed from the codewords that are in the 6-class code but not in the 7-class code, is a code of thirty-five codewords with a minimum distance of 24.) In Table 5.8 we list the codes constructed from the net on forty-nine points.

5.2.6 Summary

Table 5.9 lists the net-generated codes that we have found.

	constraint size, n:					
n.c.:	2	3	4	5	6	7
1	(4,4,2)	(9,27,2)	(16,1296,2)	(25,10^5,2)	(36,20^6,2)	(49,35^7,2)
2	(4,2,4)	(9,6,4)	(16,90,4)	(25,2040,4)	(36,297200,4)	(49,68938800,4)
3	-	(9,3,6)	(16,12,8)	(25,130,6)	-	(49,310198,6*)
4	-	-	(16,6,8)	(25,20,10)	-	(49,5726,8*)
5	-	-	-	(25,10,10)	-	(49,399,12)
6	-	-	-	-	-	(49,70,14)
7	-	-	-	-	-	(49,35,14)

Table 5.9: Some net-generated codes arrayed by number of classes (n.c.). Codes are listed by (codeword size, number of codewords, code minimum distance). Distances marked with $*$ are lower bounds.

5.3 Determining Minimum Distance

Here we present a graphical method for determining the minimum distance of a code constructed by overlapping KWTA constraints on the nodes of a neural network. To illustrate the method, we will begin by finding the minimum distance of a known quantity: the hypercube codes of Chapter 4.

5.3.1 The Minimum Distance of the Hypercube Code

The basis of the algorithm is as follows: Choose one codeword. Change one bit of the codeword from a 1 to a -1. Now we will consider how many additional bits must be changed until all of the constraints are satisfied again. Since the minimum distance gives the distance between the two closest codewords, we will want to change as few nodes as possible. We will draw the changed node and its constraints in a *constraint graph* to facilitate the counting of other changed nodes.

Let us start by considering the one-dimensional hypercube codes. In these codes each node is present in exactly one constraint. Thus, if one node, x, is changed, one more node, call it y, must be changed to keep the KWTA constraint satisfied (see Figure 5.7). Since only two nodes must change their values to resatisfy the constraint, the minimum distance of the code is 2.

Next let us consider two-dimensional hypercube codes. A given node x is in two

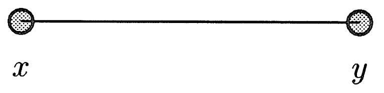


Figure 5.7: One-dimensional constraint graph. The line represents the KWTA constraint that x and y share.

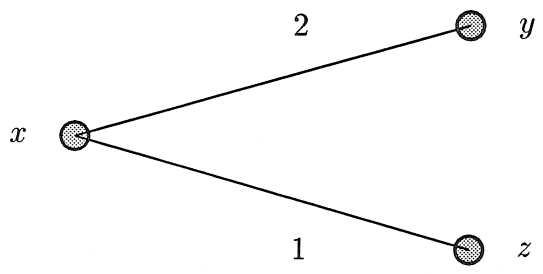


Figure 5.8: Two-dimensional constraint graph with two constraints, 1 and 2, on node x.

constraints, call them 1 and 2. (Here constraint 1 could be a horizontal constraint and constraint 2 a vertical constraint. Furthermore, all constraints of type 1 would be horizontal, and all constraints of type 2 would be vertical.) By Figure 5.8 we see that at least two more nodes (beyond x) must change so that the KWTA constraints on x are satisfied. Call these two nodes y and z. Now, y and z must also have their 2nd and 1st constraints, respectively, satisfied. Note that containing y and z in a common constraint will not satisfy the constraints (would such a constraint be type 1 or type 2?). Thus, we must add another node, obtaining the *complete* constraint graph in Figure 5.9. Note that if x were changed from a 1 to a -1, y and z would be changed from -1's to 1's, and w from 1 to -1. In this way, each constraint (or line segment in the graph) has one 1 node and one -1 node, and thus is satisfied. From this, we see that the minimum distance of two-dimensional hypercube codes is four.

Another interpretation of this counting argument involves traversing an $N \times N$ matrix. We begin at the changed node and move along the constraints we wish to resatisfy. Figure 5.10 shows the Figure 5.9 constraint graph reinterpreted in this way.

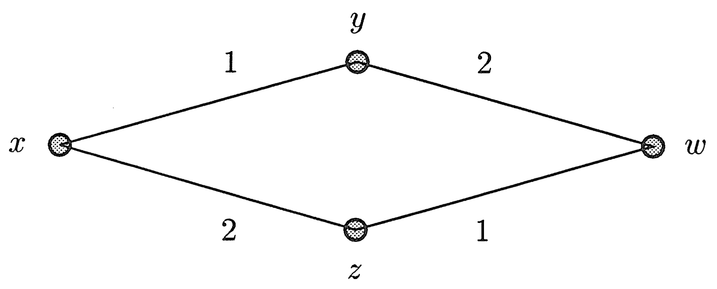


Figure 5.9: Complete two-dimensional constraint graph.

For simplicity we illustrate the example with a 4×4 matrix. In Figures 5.9 and 5.10, we traverse two paths from x to w. One path moves horizontally to y, then vertically to w; the other path moves vertically to z, then horizontally to w. (Actually, the two-dimensional constraint graph is the node matrix with the nonchanged nodes omitted.)

Now let us consider three-dimensional hypercube codes. Three KWTA constraints contain the node x. From Figure 5.11 we guess that w and y could share a type 3 constraint. However, if x changed from 1 to -1, then w, y and z must change from -1 to 1 to satisfy the 1, 2 and 3 constraints. Then, w and y cannot share a constraint (since changing two nodes in a KWTA constraint from -1 to 1 will not satisfy the constraint). With this set of KWTA constraints (which go horizontally, vertically, and depth-wise through an $N \times N \times N$ cube of nodes), a shortest path, along constraints, between nodes can be characterized by the types (or dimensions) of the constraints traveled. Thus, if a shortest path between nodes x and v is along constraints of type 1 and 2, then all shortest paths must be along paths of constraints of only type 1 and 2. For example, consider the $N \times N \times N$ cube of nodes. The shortest path between two nodes is characterized by the dimensions that must be traversed. Because of this dimensional characterization of the shortest path, the constraint graph in Figure 5.12 is invalid. This figure shows two paths of length two between x and v, where one traverses constraints of types 1 and 2, and the other traverses constraints of types 2 and 3. However, Figure 5.13 is a valid partial graph. Now the constraint graph shows two paths, both containing only type 1 and type 2 constraints, from node x to node v. Figure 5.13 leads to the complete constraint graph in Figure 5.14. The

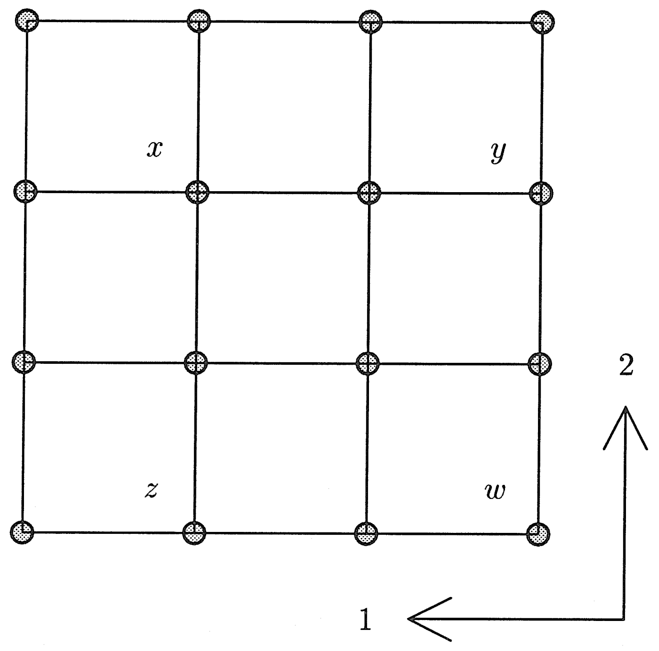


Figure 5.10: Two-dimensional constraint graph reinterpreted as matrix traversal. The directions of constraints of types 1 and 2 are indicated.

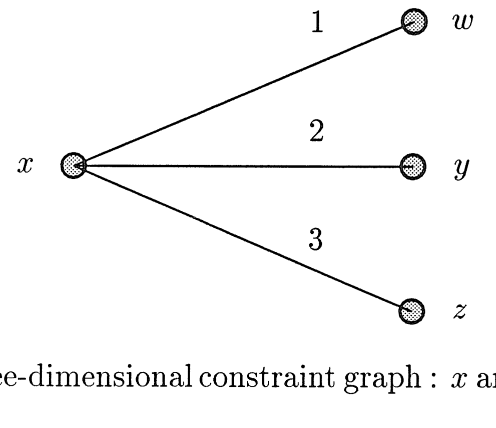


Figure 5.11: Three-dimensional constraint graph : x and its constraints.

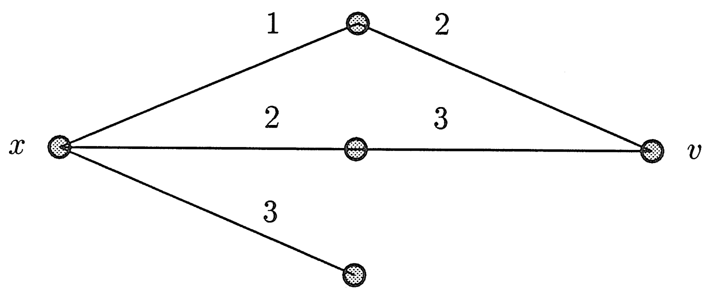


Figure 5.12: Invalid three-dimensional constraint graph.

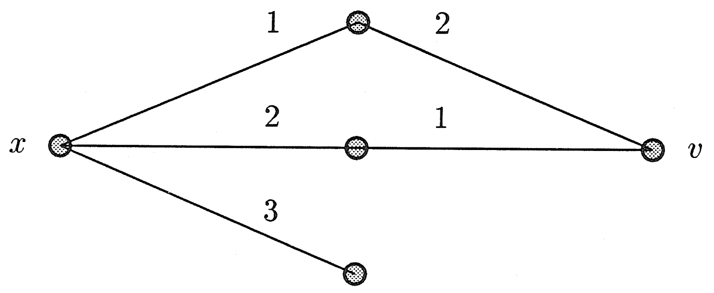


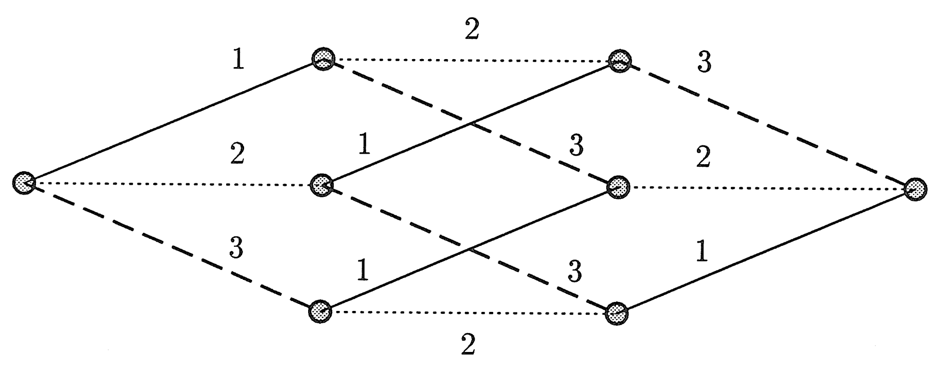
Figure 5.13: Valid constraint graph for the three-dimensional hypercube code.

closest codewords are at distance eight from each other, and therefore, the minimum distance of the three-dimensional hypercube code is 8.

This algorithm is easily extended to any dimension. In i dimensions, the constraint graph has one node at the first level (our node x, all the way to the left of the graph), i nodes at the next level (i paths involving just one constraint), $\binom{i}{2}$ nodes at the next level (representing nodes reached by paths of length two), ...,$\binom{i}{i-1}$ nodes at the next-to-last level, and $\binom{i}{i} = 1$ node at the last level. Thus, the closest codewords are at distance $1 + i + \binom{i}{2} + \ldots + \binom{i}{i-1} + \binom{i}{i} = 2^i$, and the minimum distance of this code is 2^i.

5.3.2 The Minimum Distance of Net-Generated Codes

Using the methodology introduced above, we now consider the minimum distance of a-class, net-generated codes. The one-class system produces the constraint graph of Figure 5.15. The minimum distance of the one-class code is two because two nodes



$1 \quad + \quad 3 \quad + \quad 3 \quad + \quad 1$

$=$ number of nodes changed $= 8$

Figure 5.14: Complete constraint graph for the three-dimensional hypercube code.

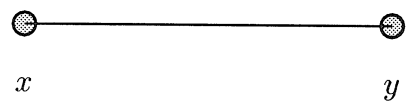


Figure 5.15: One-class constraint graph.

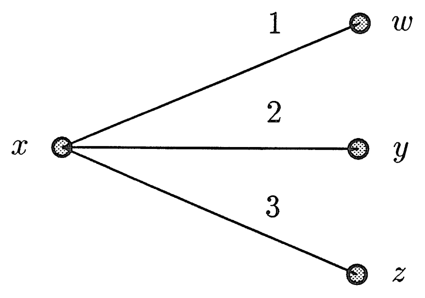


Figure 5.16: Three-class partial constraint graph.

must be changed to resatisfy the KWTA constraint.

The two-class code is identical to the two-dimensional hypercube code of Section 4.3 and therefore, has a minimum distance of four.

The three-class system has the partial constraint graph of Figure 5.16. As we explained for the three-dimensional hypercube code, the nodes w, y and z cannot share any constraints (because each constraint must have a 1 and a -1), and therefore more nodes must be added to the graph. Nodes w, y and z each have two additional constraints that must be satisfied. Any additional nodes will each have three constraints that need to be satisfied. Thus, in the best case, only $6/3 = 2$ nodes must be added. In fact, two nodes are sufficient. The resulting constraint graph is shown in Figure 5.17. Also, Figure 5.18 gives the location of one set of changed nodes in a matrix with n equal to four. The net-generated constraints do not force shortest paths between nodes to be of a certain dimensionality, as was the case for the hypercube codes. As we see in Figure 5.18, there are three paths of length two (along dimensions 1 and 2, 2 and 3, and 3 and 1, respectively) from x to u.

For a general a-class code, the partial constraint graph is illustrated in Figure 5.19. In the best case, the constraint graph in Figure 5.19 shows that the original $1 + a$ nodes change values, plus a minimum of $\frac{a(a-1)}{a} = a-1$ new nodes must change values. Thus, the minimum distance of the a-class code is at least $1 + a + (a - 1) = 2a$. In Table 5.9 most of the codes listed have a minimum distance of exactly $2a$ (and those that did not have a minimum distance of $2(a + 1)$).

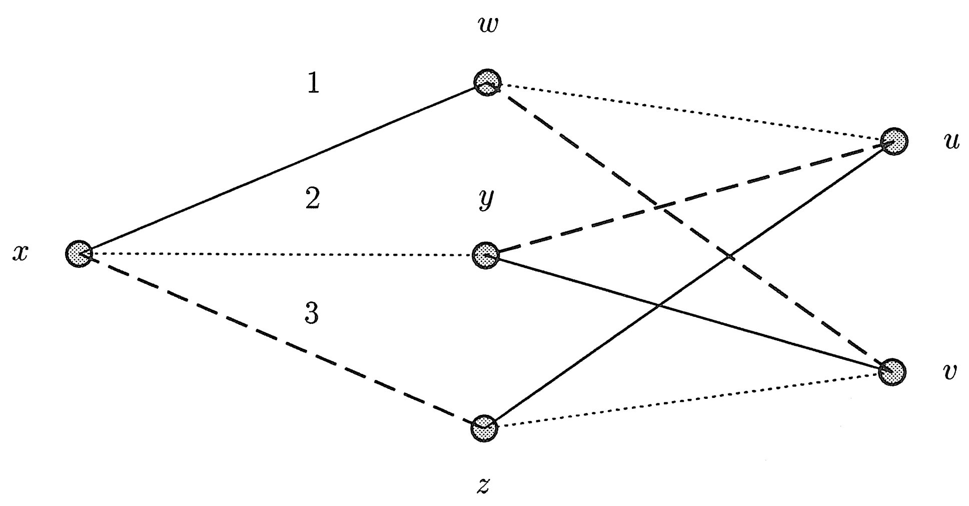


Figure 5.17: Complete three-class constraint graph.

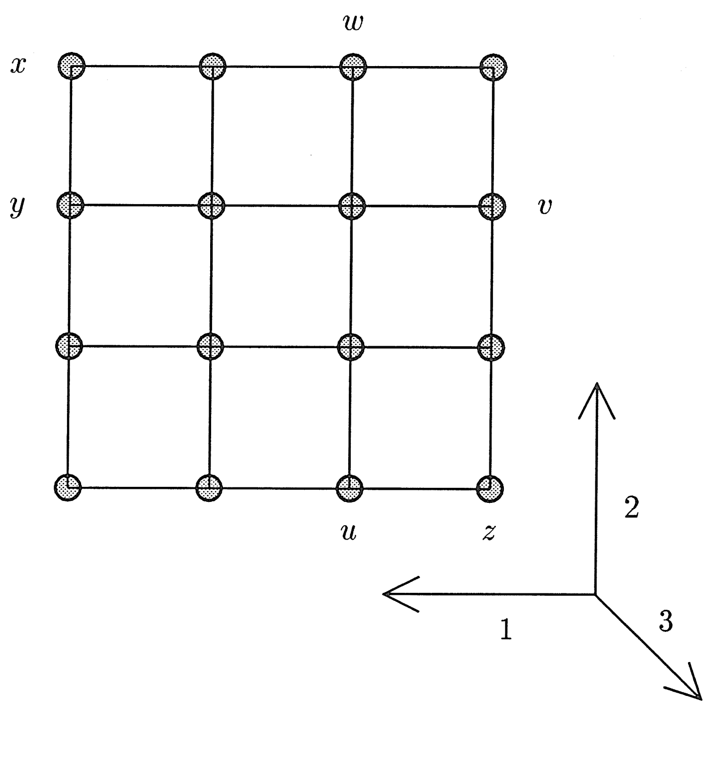
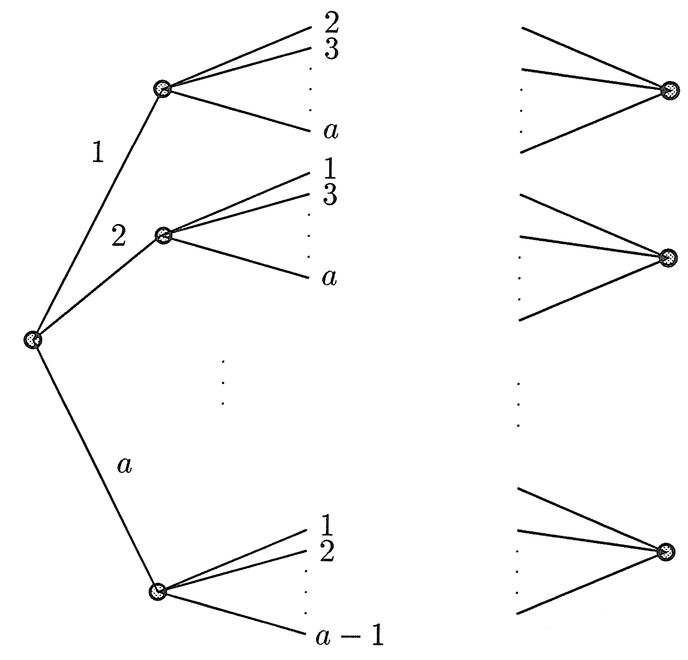


Figure 5.18: Matrix interpretation of the three-class constraint graph.



total of $a(a-1)$ constraints to be satisfied a constraints per node

Figure 5.19: Partial a-class constraint graph.

5.4 Comparisons

Experimentally, we have found that decoders generated using the methods presented in this chapter perform near-optimal decoding on their corresponding code.

Figure 5.20 compares the codes decoded by the two-, three- and four-class net-generated codes on sixteen points using the SNR discussed in Section 4.5.1.

At each minimum distance, the size of the net-generated codes surpassed the size of the largest randomly generated codes of Chapter 3.

We can also compare these codes to the best-known, constant weight codes from [BSS90]. Recall that the net of nine points gives rise to three codes. The three-class (9,3,6) code is also the largest constant weight code with codewords of size nine, weight three and a minimum distance of six. The two-class (9,6,4) code has half as many codewords as the largest code with a minimum distance of four.

The three-class (16,12,8) code has fewer codewords than the largest code with these parameters, which has 30 codewords. The largest constant weight codes with

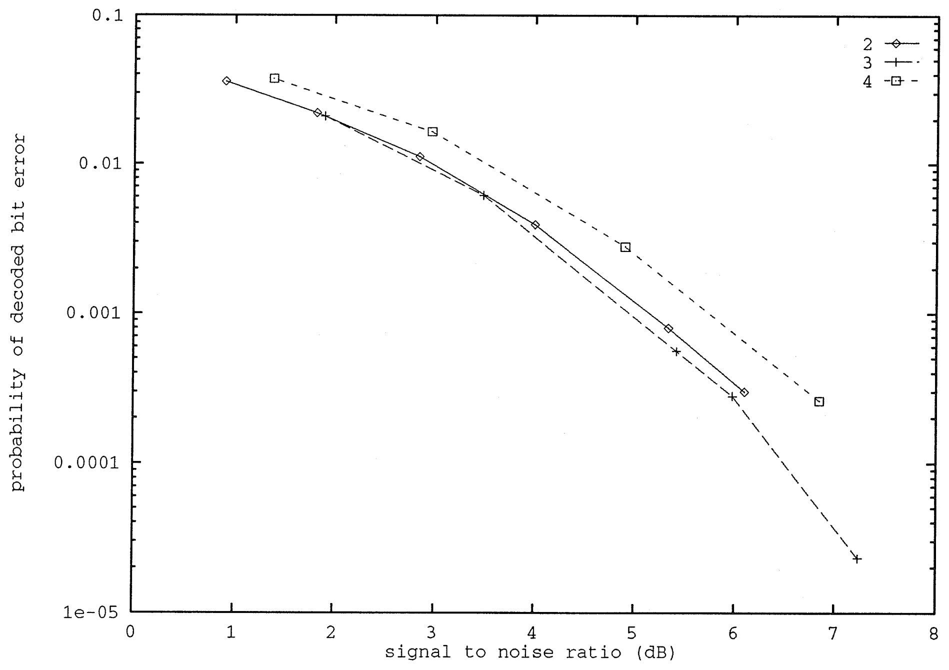


Figure 5.20: Performance of two- , three- and four-class (indicated by 2, 3 and 4, respectively) codes

codewords of length twenty-five and weight ten are (25,6036,6) and (25,100,10), as compared to the net-generated codes (25,130,6) and (25,20,10). These comparisons illustrate that the net-generated codes are smaller than the comparative largest constant weight code. The advantage of the net-generated code is the existence of a decoder which can quickly decode the net-generated code.

We can also make some general observations about hypercube and net codes. To see how these families of codes compare, we consider the ratio of minimum distance to codeword length. For an i-dimensional hypercube code, this ratio is $(2/N)^i$. As i grows, the ratio becomes small. Hence, to increase the minimum distance of a hypercube code, we must increase the codeword size. For an a-class, net-generated code, this ratio is $2a/n^2$. As a increases, the ratio increases. Thus, the codeword size does not need to be increased to increase the minimum distance. Of course, we are limited to a minimum distance of $2n$ for net-generated codes. Another interesting ratio is the number of constraints in the system to the number of nodes in the system. In Section 4.5 we found the number of constraints in an i-dimensional hypercube decoder to be iM/N. Thus, the ratio of constraints to nodes is i/N. The number of constraints in an a-class, net-generated decoder is an. Thus, the constraint to node ratio is a/n. Both the hypercube codes and the net-generated codes are more powerful than the random codes we discussed in Chapter 3. We hypothesize that a constraint to node ratio of c/N, for some constant c, is a useful property for decoders. In addition, in terms of decoder complexity, a constraint to node ratio that grows as c/N indicates that we are not trading a low processor complexity (here the number of processors is equal to the codeword size) for a high degree of processor interconnectivity (related to the number of constraints). In the next chapter, we will discuss, in detail, decoder complexity in terms of the space necessary to implement the decoder on an integrated circuit.

Chapter 6

Decoder Layout

6.1 Introduction

In this chapter, we consider the implementation of decoder circuits. Specifically, we will discuss decoders that can be constructed on a single, integrated circuit.

Components such as transistors and wires are constructed on a silicon chip by layering different materials. The limited precision of the fabrication *process* to pattern and align these layers, and the intrinsic electrical behavior of the devices, dictate a set of *design rules*, or minimum size and spacing rules, that chip designers must follow in order to obtain reliable chips. Because the minimum feature size changes with improvements in processes, these design rules are often given in terms of a dimensionless length unit λ. We will use a width allowance for one wire of 7λ; this wire *pitch* includes sufficient space around the wire such that it will not interfere with an adjacent wire. Currently, $\lambda \approx 1$ micron for typical processes. Mead and Conway [MC80] give an overview of the wafer fabrication process and discusses scalable design rules in depth.

Thompson's grid model [Tho80] is used to study the asymptotic complexity, in terms of relative area, of an integrated circuit. In this chapter, we apply Thompson's model in order to determine the scaling relationship of implementable circuits with decoder size and to estimate the maximum size of decoders that can be realized on a single, integrated circuit. Thompson's model assumes a chip of evenly spaced horizontal and vertical tracks. Wires on two layers run independently along these tracks. Wires go from layer to layer via contacts. Processors can be located at intersections of two tracks (one horizontal, the other vertical). For processors where more than four input/output lines are needed, a rectangular processor is used that will cover several intersections of horizontal and vertical tracks. Inputs and outputs for this type of processor run along the tracks that abut the processor (e.g., an $N \times M$ rectangle has $2(N+M)$ input/output lines).

Applying this model, we can construct layouts of circuits. Our system has simple processors, and each processor has a fanout that grows with the dimension of the system. To estimate the size of the chip needed for our system, we will make the processors large enough for the necessary fanout and assume that this size will be adequate for all the necessary processing. In this analysis, we will ignore wires not directly used in the computation, such as power and ground wires.

The size of the integrated circuit is limited by the defect density introduced by the fabrication process. A variety of fabrication errors can result in a chip that does not function properly. The yield of (good) chips is modeled as inversely exponential in the utilized area of the chip. Thus, beyond a certain size, fatal errors are likely to occur. Currently, the largest practical integrated circuits being fabricated are 1cm by 1cm.

If the size of a (two-dimensional) chip is limited to 1 cm on a side, and wires occupy 7μm each, then a chip containing only wires could have about 1430 wires in each dimension. Because we require processing elements in addition to wires on our chip, we will use a bound of 1000 wires in each dimension as our limit. From this analysis, we find that decoders of a reasonable size could be implemented on a single, integrated circuit.

Most of our analysis will focus on worst-case arguments for the chip size necessary to implement our decoders. After these arguments, we will present in Section 6.4 an aggressive layout suggested by Carver Mead that reduces greatly the circuit area necessary to implement an $N \times N$ decoder.

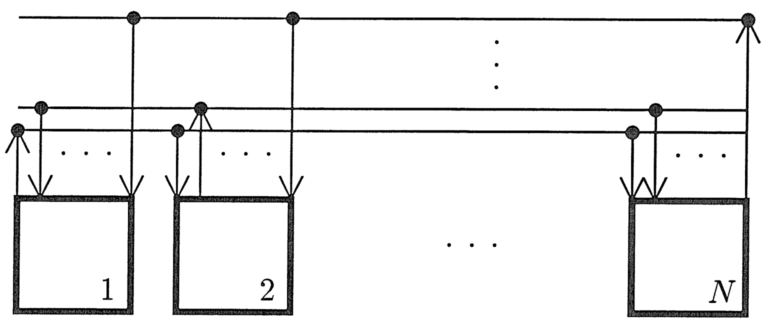


Figure 6.1: Layout of KWTA circuit with N nodes. Arrows indicate direction of information flow.

6.2 Layout of a Hypercube Decoder

In this section, we will present a bound on the physical size of a hypercube decoder circuit constructed on silicon. We will consider the size of the largest decoder that could be built on a typical 1cm by 1cm integrated circuit. We will use the Thompson grid model of VLSI chip layout and the assumptions discussed in Section 6.1.

The circuit we wish to construct has N^i nodes (processors). Each node is labeled as $\{x_1, x_2, \ldots x_i\}$ with $x_j \in \{1, 2 \ldots N\}$. A KWTA constraint is placed on a set of N nodes that differ only in one index (see Chapter 4). These groups of N processors are completely interconnected. That is, the output of each node in the group is an input to all the other nodes in the group. The layout of such groups is illustrated in Figure 6.1.

The processors are laid out on the silicon in the following manner. For $i = 2$, the usual matrix format is used—with node $(1, 1)$ in the upper left and node (N, N) in the bottom right. For $i = 4$, the $(i = 2)$-system is repeated N^2 times to form an $N \times N$ matrix of $(i = 2)$-systems. The third and fourth indices of a node give the identity of the particular $(i = 2)$-system in which it resides. Thus, to proceed from a size i to a size $(i + 2)$ system, we fill an $N \times N$ matrix with i-systems, where their labeling is done by the $(i + 1)$th and $(i + 2)$th indices. The result is a square matrix of $N^{i/2} \times N^{i/2}$ nodes.

Now let us consider the connections that exist between nodes. Each node will share constraints only with other nodes in the same row or column of the matrix.

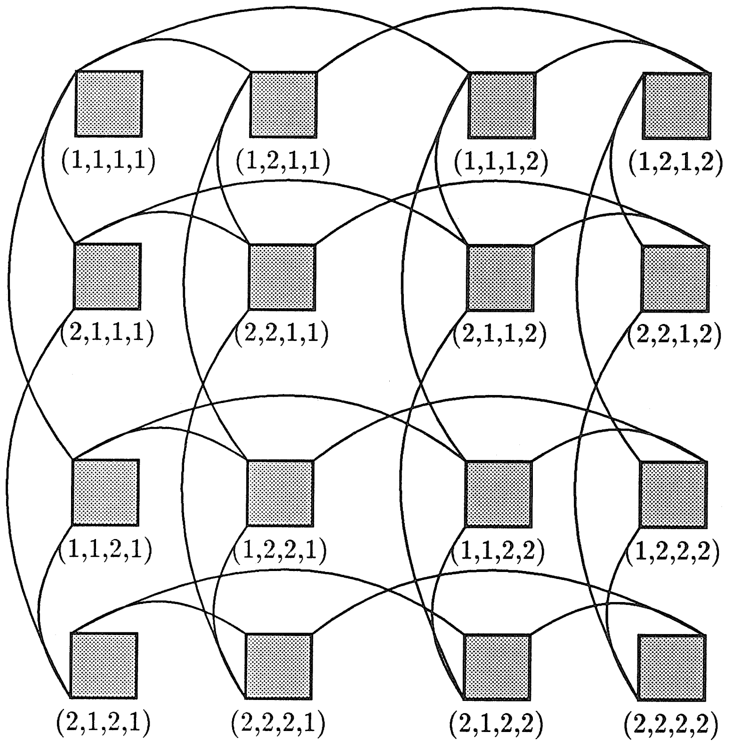


Figure 6.2: Node placement for the 2×2×2×2 decoder. Lines between nodes indicate that they share a KWTA constraint.

Figure 6.2 shows the layout of the nodes, their labels and the constraints for the $2{\times}2{\times}2{\times}2$ decoder. For a general decoder, the leftmost upper node is in i constraints, and thus, will be connected to $i(N-1)$ other nodes. For the first constraint, we have connections to the $(N-1)$ consecutive nodes as we move horizontally. Thus, connections exist to the second, third,..., Nth node in the top row of the matrix. So, node $(1,1,1,1,\ldots,1)$ is connected to nodes $(1,2,1,1,\ldots,1)$, $(1,3,1,1,\ldots,1)$,..., and $(1,N,1,1,\ldots,1)$. For the second constraint, there exists a connection to every Nth node in the row (for $N-1$ nodes) — ie., the $(N+1)$th node, the $(2N+1)$th,..., and the $[(N-1)N+1]$th node. Here node $(1,1,1,1,\ldots,1)$ is connected to nodes $(1,1,1,2,\ldots,1)$, $(1,1,1,3,\ldots,1)$,..., and $(1,1,1,N,\ldots,1)$. For the third constraint, we have connections to every N^2th node in the row (again for $N-1$ nodes) — or to the (N^2+1)th node, the $(2N^2+1)$th node,..., and, the $((N-1)N^2+1)$th node. Here, node $(1,1,1,1,1,1\ldots,1)$ shares a constraint with nodes $(1,1,1,1,1,2,\ldots,1)$, $(1,1,1,1,1,3,\ldots,1)$, ..., and $(1,1,1,1,1,N,\ldots,1)$. Finally, for the $i/2$th constraint, we have connections to every $N^{i/2-1}$th node in the row. This constraint connects node $(1,1,1,1,\ldots,1)$, to nodes $(1,1,1,1,\ldots,2)$, $(1,1,1,1,\ldots,3)$, ..., and $(1,1,1,1,\ldots,N)$. The other $i/2$ constraints of the upper left node are laid out vertically. Each of the above-mentioned horizontal connections has a corresponding vertical connection. Above, we specified horizontal connections between nodes that differed in their second, fourth, sixth and Nth indices. The vertical connections exist between nodes that differ in odd indices. The other nodes in the matrix (besides the upper left node) are involved in similar connections. Now that we have specified the layout, we are ready to measure the area it occupies.

The operation that we wish our processors to accomplish is a relatively simple one. However, the fanout of each processor grows rapidly with the dimension of the system. Thus, we will assume that our processor size is that necessary to accommodate its fanout.

The processors can have connections on all four sides. As shown in Figure 6.1, processors make vertical connections to horizontal buses. Similarly, processors make horizontal connections to vertical buses. Thus, a connection between processors in a (horizontal) row uses both horizontal and vertical space. Hence, if we wish to deter-

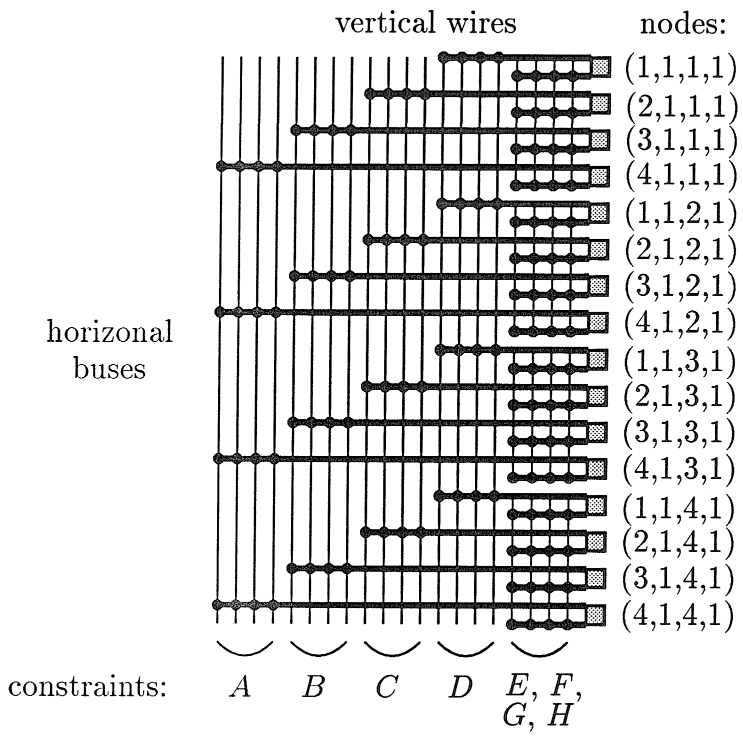


Figure 6.3: Vertical constraint layout for a column of 16 nodes of the 4×4×4×4 decoder. Thick horizontal lines represent buses (of $N = 4$ wires); vertical lines represent wires. Connections between wires and buses are shown as dots.

mine the width of our chip, we must include the space occupied by both horizontal and vertical buses in our computation.

We will determine the width of our system by measuring the width of one column of processors as well as its associated wires and then multiplying by the number of such columns. This is a legitimate calculation of the width because of the symmetry of the system; i.e., all columns have the same connections.

We have said that all connections from a processor use horizontal (and vertical) space. First let us look at the space used by the vertical buses associated with one column of processors. Figure 6.3 illustrates the layout of a column of nodes, and its associated vertical constraints, of the 4×4×4×4 decoder. Each constraint introduces a bus of N wires as in Figure 6.1. The first connections that we discussed enforce a constraint on N consecutive nodes. Thus, the $N^{i/2}$ nodes in the first column are divided into $N^{i/2-1}$ groups of N consecutive nodes. Each of the resulting buses will look like the bus in Figure 6.1 (rotated ninety degrees). These $N^{i/2-1}$ buses are nonoverlapping. Thus, the width allotted for the first group's bus can be reused by the second, third,..., and the $(i/2-1)$th group's bus. Hence, we need 1 bus width for this group of buses. These connections are labeled as constraints E, F, G, and H in Figure 6.3.

The next constraints that we discussed connected every Nth node. A group of N^2 nodes (say the first through the N^2th) has N of these overlapping buses. (That is, one bus will have a node between two nodes from another bus.) These buses are labeled A, B, C, and D in Figure 6.3. The next group of N^2 nodes, however, will have (again N) buses that do not overlap with the previous group's N buses. (Think of adding another copy of Figure 6.3 under the original. New constraints A', B', C', and D' would use the same width already allotted for constraints A, B, C, and D.) Hence, the space we allocate for the first N buses can also be used for the next N buses. Finally then, we need N bus widths for these connections. This pattern generalizes. Finally, for the $(i/2)$th mentioned buses, we need $N^{i/2-1}$ bus widths.

Each bus width is N wires. The number of columns of nodes in our chip is $N^{i/2}$. This gives a width of $(1+N+N^2+\ldots+N^{i/2-1})N(N^{i/2})$ wire widths to accommodate the vertical buses on the chip. We can rewrite this as

$$N^i + N^{i-1} + \ldots + N^{i/2+1} = \frac{N^i - N^{i/2}}{1-\frac{1}{N}} \text{ wire widths.}$$

Now we need to look at the width that is due to the horizontal buses. The buses themselves do not add to the width of the system; however, the connections from the processors to the buses do require space. There are $\frac{1}{2}iN$ of these connections per node (i.e., half of the total connections.) In the best case, we will have half of these connections to the top of the node and half to the bottom. This gives a width contribution of $\frac{1}{2}\cdot\frac{1}{2}iN = \frac{1}{4}iN$. In the worst case, all the connections would be on one side (top or bottom) of the node. This gives a width contribution of $\frac{1}{2}iN$. Horizontal buses constrain nodes in the same row. Thus, as in Figure 6.4, the other nodes in the column use this same horizontal space to make their vertical connections to horizontal buses. In the worst case, the horizontal buses contribute $\frac{1}{2}iN(N^{i/2}) = iN^{(i/2+1)}/2$ wire

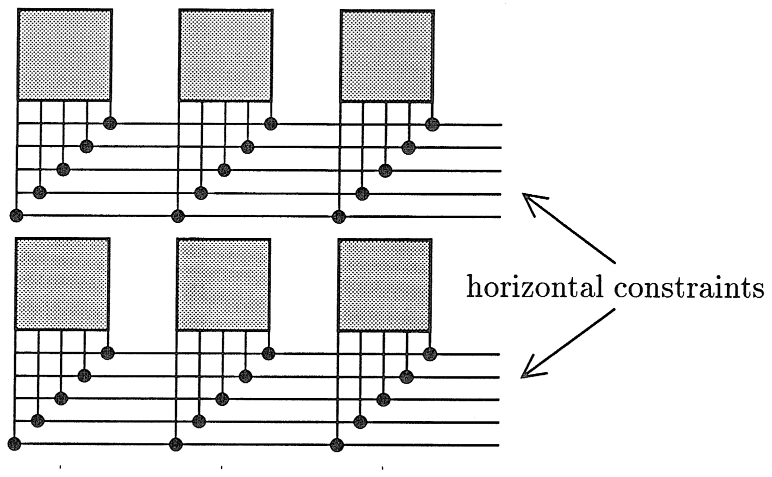


Figure 6.4: Vertical connections to horizontal constraints. The horizontal space used by one processor in a column of nodes to connect to constraints is sufficient for the other processors in the column.

widths to the chip ($iN^{(i/2+1)}/4$ in the best case).

Finally, we obtain a total (worst case) chip width of

$$\begin{aligned} \frac{N^i - N^{i/2}}{1-\frac{1}{N}} + \frac{i}{2}N^{(i/2+1)} &= \frac{N^{(i+1)}}{(N-1)} + N^{(i/2+1)}\Big(\frac{i}{2} - \frac{1}{(N-1)}\Big) \\ &\approx N^i + \frac{i}{2}N^{(i/2+1)} \text{ wire widths.} \end{aligned}$$

6.2.1 Winner-Take-All Circuits

For the particular case of the single-winner-take-all network ($K = 1$) Lazzaro et al. [LRMM89] have demonstrated an efficient circuit on silicon that uses a bus consisting of only two wires. If we restrict ourselves to networks of winner-take-all circuits, we can replace the bus widths of N in the above counting arguments with bus widths of 2.

$$\begin{aligned} \frac{2}{N}\Big[\frac{N^i - N^{i/2}}{1-\frac{1}{N}}\Big] + 2N^{i/2} &= \frac{2N^i}{(N-1)} + 2N^{i/2}\Big(1 - \frac{1}{(N-1)}\Big) \\ &\approx 2N^{i-1} + 2N^{i/2} \text{ wire widths.} \end{aligned}$$

6.2.2 Circuit Size

We can use the technique established above to count the width and height of non-square (i odd) systems. For a system of $N^{(i-1)/2} \times N^{(i+1)/2}$ nodes, we obtain a width of

$$\begin{aligned} \text{width} &= N^{(i+1)/2} N(1 + N + N^2 + \ldots + N^{(i-1)/2-1}) + \frac{(i+1)}{2} N N^{(i+1)/2} \\ &= N^i + \ldots + N^{(i+3)/2} + \frac{(i+1)}{2} N^{(i+3)/2} \text{ wire widths.} \end{aligned}$$

The height of the system is equal to

$$\begin{aligned} \text{height} &= N^{(i-1)/2} N(1 + N + N^2 + \ldots N^{(i+1)/2-1}) + \frac{(i-1)}{2} N N^{(i-1)/2} \\ &= N^i + \ldots + N^{(i+1)/2} + \frac{(i-1)}{2} N^{(i+1)/2} \text{ wire widths.} \end{aligned}$$

Assuming that we embed this nonsquare circuit on a square 1cm by 1cm chip, the width of the system will limit the size of our design since the width is greater than the height (for $N > 1$). Table 6.1 lists possible designs. Given a value of i, the chip width (in wires) was calculated. Then, the largest value of N that kept the chip width under 1000 wires was calculated. The number of total nodes M (for this value of N) is also given. These results suggest that there are hypercube decoders of reasonable size that could be implemented on a single, integrated circuit. Note that standard pad configurations can take up to 256 input/output lines to or from the chip. Thus, for systems with more than 256 nodes, signals to and from the nodes will need to be multiplexed.

In conclusion, the chip width of the decoder circuit grows linearly as M, the total number of nodes. The circuit area scales as the square of the network size. This analysis suggests that non-trivial hypercube decoder systems could be constructed with current VLSI technology.

i	chip width (in wires)	largest N	M
10	$N^{10} + N^9 + N^8 + N^7 + 6N^6$	< 2	-
9	$N^9 + N^8 + N^7 + 6N^6$	< 2	-
8	$N^8 + N^7 + N^6 + 5N^5$	2	256
7	$N^7 + N^6 + 5N^5$	2	128
6	$N^6 + N^5 + 4N^4$	2	64
5	$N^5 + 4N^4$	3	243
4	$N^4 + 3N^3$	5	625
3	$3N^3$	6	216
2	$2N^2$	22	484
1	N^2	31	31

Table 6.1: Chip width for i-dimensional system. Also given is largest N for which the chip width is less than 1000 wire widths, and the corresponding value of $M = N^i$.

6.3 Layout of a Net-Generated Decoder

In this section, we will present an argument for the size of a net-generated decoder (Chapter 5) circuit constructed on silicon. We will calculate the largest decoders that can be built on a 1cm by 1cm integrated circuit, using the Thompson grid model of VLSI chip layout and the assumptions discussed in Section 6.1.

We begin by placing the $m = n^2$ processors in an $n \times n$ array. The one-class decoder will have KWTA constraints enforced along the rows of the array. The width of the system is equal to the width of a KWTA constraint on n nodes. From the techniques in Section 6.2, we derive that the width of the KWTA constraint on n nodes is n^2 wire widths. Assuming processors of height one, the height of the one-class decoder is $n(n+1)$. (If we assumed square processors, the chip height would be $2n^2$.)

The two-class decoder is identical to the two-dimensional hypercube code and (from Section 6.2) has height and width equal to $2n^2$ wire widths. Each additional constraint for the three-class decoder will connect to one node per row and one node per column. Thus, there is no advantage to placing these constraints in a vertical direction versus placing them in a horizontal direction. Thus, if we are considering an a-class system, we will place $a/2$ classes vertically and $a/2$ classes horizontally. The 2-class layout accounts for one vertical and one horizontal class. The remaining

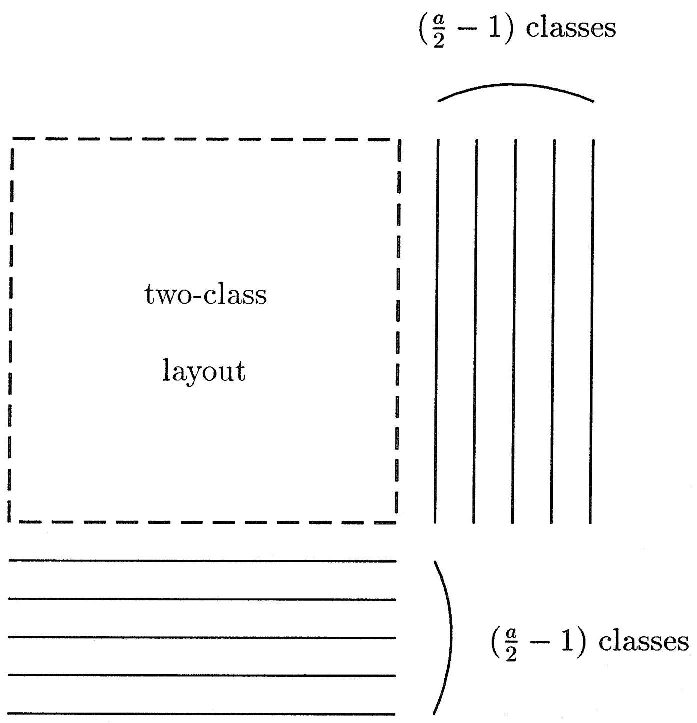


Figure 6.5: Placement of class constraints for the a-class net-generated decoder.

$(\frac{a}{2} - 1)$ vertical classes contribute $n(\frac{a}{2} - 1)$ constraints, or $n^2(\frac{a}{2} - 1)$ wire widths to the chip width. This arrangement is shown in Figure 6.5. Since we already know the width of the two-class layout $(2n^2)$, let us now consider the width of the constraints introduced by the additional $(a - 2)$ classes.

Figure 6.6 shows the connections from two processors in a column to the vertical constraints. In Section 6.2, the vertical constraints were connected to nodes by horizontal wires (as in a rotated Figure 6.1). Here, because each constraint is placed on exactly one node per row and one node per column, connections to constraints must span the array of nodes. Thus, we have vertical connections to horizontal wires that in turn connect to the vertical constraints (as in Figure 6.6).

Each class specifies one constraint for a given node. Thus, the $(\frac{a}{2} - 1)$ vertical classes specify $(\frac{a}{2} - 1)$ constraints for each processor. Since each processor connects to $(\frac{a}{2} - 1)$ vertical constraints, its vertical connections use $n(\frac{a}{2} - 1)$ wire widths. Two processors in the same column can use the same horizontal space for the connection

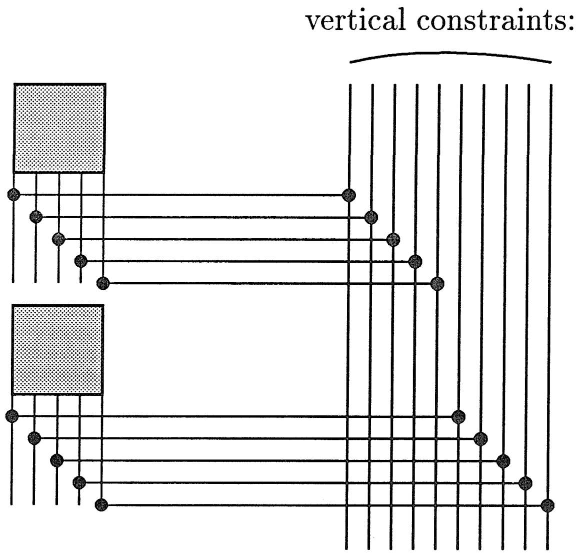


Figure 6.6: Connections to vertical constraints for the a-class net-generated decoder.

(as in Figure 6.6). Hence, over all n columns, $n^2(\frac{a}{2} - 1)$ wire widths are necessary.

Now we need to consider the width needed for the vertical connections to the $(\frac{a}{2} - 1)$ horizontal buses. Figure 6.7 shows these connections. Each processor in the column must satisfy $(\frac{a}{2} - 1)$ horizontal constraints. For each constraint that involves the processor, n wires are needed. Thus, the processors in a column need $n^2(\frac{a}{2} - 1)$ wire widths, for a total contribution to the chip area of $n^3(\frac{a}{2} - 1)$ wire widths.

We can now sum the width needed for the two-class decoder, plus the width of the $(\frac{a}{2} - 1)$ vertically laid classes and the width needed to connect to these classes, plus the width to connect to the $(\frac{a}{2} - 1)$ horizontally laid classes to obtain a total width of

$$\begin{aligned}\text{width} &= 2n^2 + n^2\left(\frac{a}{2} - 1\right) + n^2\left(\frac{a}{2} - 1\right) + n^3\left(\frac{a}{2} - 1\right) \\ &= n^3\left(\frac{a}{2} - 1\right) + an^2 \\ &\approx am^{3/2}/2 \text{ wire widths}\end{aligned}$$

To transform an a-class system to an $(a + 1)$-class system, where a is even, we add one horizontal class. By our previous arguments, this will add n^3 wire widths to the

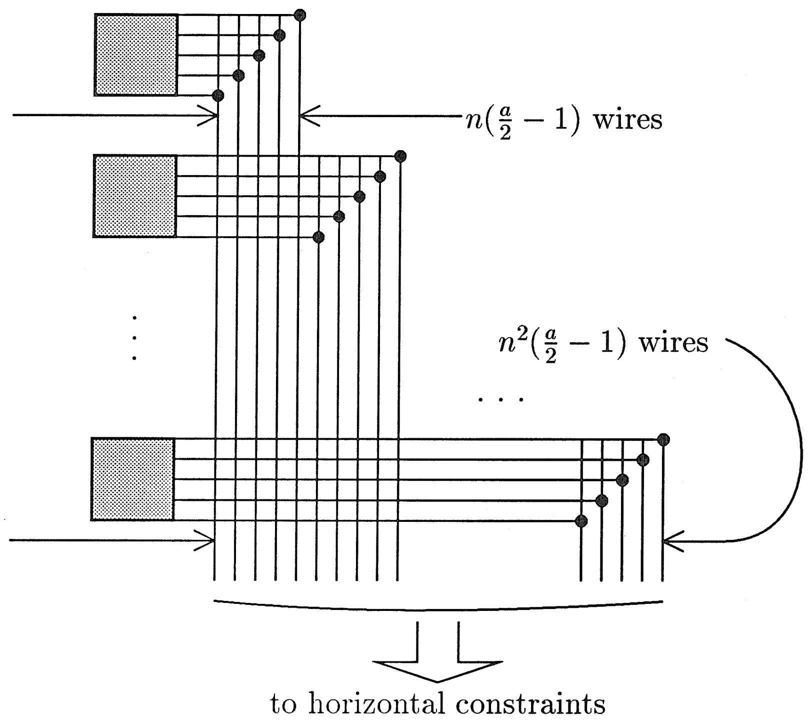


Figure 6.7: Connections to horizontal constraints for the a-class net-generated decoder.

a	chip width	largest n	m
8	$3n^3 + 8n^2$	-	-
7	$3n^3 + 6n^2$	-	-
6	$2n^3 + 6n^2$	7	49
5	$2n^3 + 4n^2$	7	49
4	$n^3 + 4n^2$	8	64
3	$n^3 + 2n^2$	9	81
2	$2n^2$	22	484
1	n^2	31	961

Table 6.2: Chip width for a-class decoder. Also given is largest n for which the chip width is less than 1000 wire widths, and the corresponding value of $m = n^2$.

width of the chip, and $2n^2$ wire widths to the height. From an $(a+1)$-class system to an $(a+2)$-class system (again with a even), the added vertical constraint contributes $2n^2$ wire widths to the chip width and n^3 wire widths to the chip height. For n greater than or equal to 2, the chip width is greater than or equal to the chip height. Thus, the chip width will limit the size of the decoder we can construct.

Table 6.2 lists possible designs for different numbers of classes. Recall from Chapter 5 that for the net-generated codes, the number of classes a must be less than or equal to n, the size of the constraints. For 7-class and 8-class systems, the largest n that results in a chip width less than 1000 is n equal to six. However, a net-generated decoder with constraints of size six can have six or fewer classes.

In conclusion, we have found that we can embed a net-generated decoder with a nets in an integrated circuit with width proportional to $\frac{1}{2}an^3$. The circuit area scales as the cube of the network size. This analysis suggests that, in a typical VLSI process, one could implement net-generated decoders of reasonable size.

6.4 An Improved $N \times N$ Decoder Layout

In this section we present a decoder layout suggested by Carver Mead. This layout utilizes aggressive circuitry implemented by students such as Misha Mahowold at

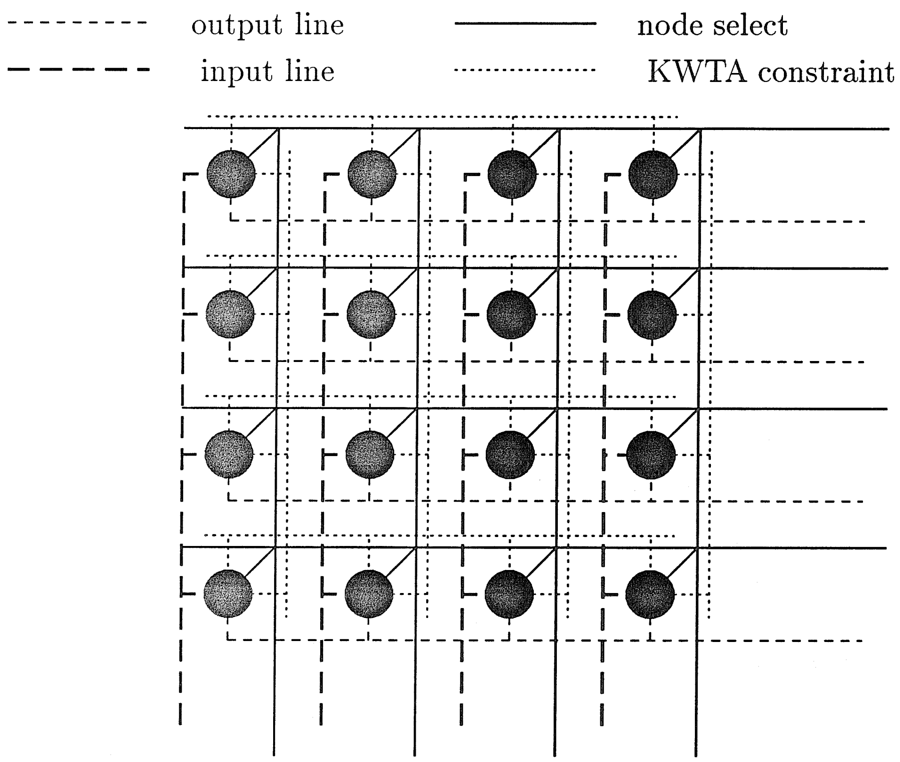


Figure 6.8: Connections to horizontal constraints for the a-class net-generated decoder.

Professor Mead's laboratory at Caltech.

We begin by placing our N^2 nodes into an $N \times N$ matrix. Nodes in a given row or column share a KWTA constraint. This constraint we now implement with a single wire. It is necessary for this implementation of a KWTA circuit that the transistors of the circuit are well matched. The nodes are addressed by horizontal and vertical select lines. The output of the circuit can then be read out one column at a time. Also, the input of the next computation can be read into the circuit as the output of the present computation is read out of the network. This leads to a two frame latency (thus, data is read in during one frame, used for computations during the next frame, and is read out in the following frame).

Figure 6.8 illustrates this layout. Note that the processor size now contributes to the size of the circuit. The width of this $N \times N$ decoder circuit is

$$3N \text{ processor widths}$$

An implementation of this circuit would present an interesting neural network application.

Chapter 7

Communication on a Broadcast Network

7.1 Introduction

El Gamal [El 87] presents a problem of communicating on a *distributed communication network* (also called broadcast network). The distributed communication network that we will be working with has N sender nodes $s_1, \dots s_N$ and one receiver node r. The senders may transmit or receive bits, while the receiver r can only receive bits. When s_i transmits a bit b, all of the other nodes will receive this transmission. The communication between any pair of nodes is modeled as over an independent, binary symmetric channel (BSC); i.e., each node has a probability ϵ of receiving an erroneous version, independently of the other nodes.

To begin, each s_i obtains an independent bit x_i (a Bernoulli trial $B(\frac{1}{2})$). The node can transmit this bit as well as any function of the bits it has heard and its own bit (i.e., s_1 can transmit $t_1 = f(s_1, \hat{t}_2, \hat{t}_3, \dots, \hat{t}_N)$, where $\hat{t}_i$ is s_1's best guess of $\hat{s}_i$'s transmission). El Gamal's problem is to determine how many transmissions are needed for r to have received the whole array of the bits $\mathbf{x}$ reliably. (A protocol gives a *reliable* estimate if, given an allowable probability of error, $\delta > 0$, there exists a T_0 such that for T transmissions where $T \geq T_0$, the probability that the estimate is correct is $\geq 1 - \delta$.)

Gallager [Gal85] (and [Gal87]) gave a bound of $O(N \ln \ln N)$ bits for the amount of communication needed to give the receiver a reliable estimate of the parity of N bits (one bit per node), as well as the array of N bits. It is clear (from Shannon's limit of $\Theta(N)$ for a binary symmetric channel [Sha48]) that $\Omega(N)$ bits are necessary for reliable communication. Here, using a random coding argument, we repeat Gallager's bound of $O(N \ln \ln N)$ bits to obtain a reliable estimate of the array.

Before proceeding, let us look at the network with $N = 6$. Clearly, if we wish to have r know the parity of the 6 bits, $p = x_1 \oplus x_2 \oplus \ldots \oplus x_6$, then each sender node must transmit its bit *at least* once (since if s_1 never transmits, then r will at best guess the parity with a 50% chance of error). If all the senders transmit once, then s_2 (for example) will have estimates of x_1, x_3, x_4, x_5, and x_6. Its next transmission could use this information; for example, it could send its estimate of the parity of the 6 bits. The next node to transmit can take advantage of this information, and so on. Let us leave this example now, and look at algorithms where N is assumed to be large.

We will give a protocol for obtaining a reliable estimate of the entire array at one node. Appendix C gives the proofs of the theorems presented in this chapter.

7.2 Array Protocol

We wish to obtain a reliable estimate of the array of N bits at r. The most straightforward algorithm to achieve this is to have each node repeatedly transmit its bit until r knows each bit well enough that it has a reliable estimate of the whole array. Theorem 7.1 tells us how many transmissions are necessary for this algorithm.

Theorem 7.1 For a distributed communication network of N nodes, let M be the number of times each node has to repeat its bit for r to have a reliable estimate of the array of N bits. Then $M = \Theta(\log N)$.

Proof See Appendix C.

One familiar with coding might guess that $\Theta(\log N)$ bits of communication per node are not necessary for all transmission schemes. (We can note that s_i never used the information it obtained about x_j $(i \neq j)$). That is, in fact, true, and next we present an algorithm that requires many fewer transmissions.

First, let us divide the senders $s_1, \ldots, s_N$ into (approximately) N/K groups with K members per group. Each sender transmits its bit M times, where M is enough times that each member of a group has a reliable estimate of *the group of K bits.* To satisfy this requirement, M should be $\Theta(\log K)$ (Theorem 7.1).

Now, each node in a given group of size K has a reliable estimate of that group – with error *independent* from node to node. If a sender node knew the group of K bits perfectly, that node could encode these K bits and send the encoding over the channel in about ρK transmissions. (Shannon [Sha48] has shown that ρK transmissions are necessary and sufficient to send K bits over a BSC reliably, where $\rho = (1/R) > (1/C)$, R is the rate, and C is the Shannon capacity.) Instead, each of the sender nodes has an *estimate* of the array of K, each with independent error. Thus, the system includes the channel error as well as the error in the node's estimates of the group.

Let us look at the encoding of $x_1, \ldots, x_K$, the first K bits. Again, if node s_1 knew this group perfectly, it could send about ρK bits, call them $g_1, g_2, \ldots, g_{\rho K}$, so that r would have a reliable estimate of $x_1, x_2, \ldots, x_K$ (i.e., with ρK bits sent, transmission errors could be corrected). But in our model, s_1 does not know the array of K perfectly. s_2's estimate of the array is also imperfect; however, the two estimates have independent errors. The same is true for any two s_i, s_j $(i \neq j)$. So if one node, say s_1, sends ρK bits, r could correct any errors made during the transmission – *but not errors in s_i's estimate.* Thus, r would obtain a reliable estimate of s_1's estimate of the array of K. But, as we have stated above, the errors in each node's estimate are independent from node to node. Thus, if each node transmits *one* bit of the encoding, i.e., s_i sends g_i, then

$$\begin{aligned} \Pr(g_i \text{ is in error at } r) &= \Pr(g_i \text{ is in error at } s_i) \cdot \Pr(\text{ correct transm. over BSC}) \\ &\quad + \Pr(g_i \text{ is correct at } s_i) \cdot \Pr(\text{ incorrect transm. over BSC}) \\ &\leq \epsilon(1-\epsilon) + 1 \cdot \epsilon \\ &< 2\epsilon \end{aligned}$$

where g_i's error is independent of g_j's error for all $i \neq j$. (Recall that the probability of error in s_i's estimate of the group of K (and therefore in g_i, an encoding of the group) is at most ϵ. Also, the channel error is ϵ.)

Now we have combined the channel error and the node's (estimate) error into one error that coding can correct. Our system now looks like perfect information being sent over a BSC with crossover probability 2ϵ. (We can reduce ϵ by a constant amount with a linear number of transmissions, so we do not have to be concerned with the case of $2\epsilon = 1/2$.)

There is one more detail of this encoding to be mentioned. Since $\rho > 1$ for $\epsilon > 0$, nodes $s_{K+1}, s_{K+2}, \ldots, s_{\rho K}$ will be sending $g_{K+1}, g_{K+2}, \ldots, g_{\rho K}$ (where the g_i's are functions of the K bits $x_1, \ldots, x_K$). Nodes $s_{K+1}, \ldots, s_{\rho K}$ received the initial transmissions of $x_1, \ldots, x_K$ (made by $s_1, \ldots, s_K$) and thus these nodes have a reliable estimate of $x_1, \ldots, x_K$, making their g_i bit reliable. So we see that having nodes outside the group of K nodes send bits of the their version of the K bit encoding does not compromise the reliability of the encoding. The encoding described above results in r's knowing the first K bits, $x_1, x_2, \ldots, x_K$, *super* reliably. (Before the encoded bits were sent, r had a reliable estimate of the group of K bits. Now, r's estimate of the group of K has a probability of error *exponential* in K – or super reliable.) In fact, if we repeat this encoding for each of the $N/K - 1$ remaining sets of K bits, r will know each group of K bits super reliably – so that the whole array of N will be reliable – if K is the correct size.

Thus, we have made $(\rho K \cdot \frac{N}{K}) = (\rho N) = \Theta(N)$ transmissions. These transmissions are detailed in Table 7.1. Next, Theorem 7.2 gives a restriction on K, the size of the group.

Theorem 7.2 If we communicate N bits of information over a BSC by sending K/R bits per group of K bits (for N/K groups), then the probability of error of the whole array of N bits goes asymptotically to zero when $K = W \log N$ for some constant W.

Proof see Appendix C

Thus, this scheme gives us the whole array of N reliably in

$$\#\text{transmissions} \quad = \quad \Theta(MN + N/R)$$

sender #	# of transmissions	transmission content
1	M	x_1
$\vdots$	$\vdots$	$\vdots$
N	M	x_N
$[1]_N$	1	g_1^1
$\vdots$	$\vdots$	$\vdots$
$[\rho K]_N$	1	$g_{\rho K}^1$
$[K+1]_N$	1	g_1^2
$\vdots$	$\vdots$	$\vdots$
$[K+1+\rho K]_N$	1	$g_{\rho K}^2$
$[2K+1]_N$	1	g_1^3
$\vdots$	$\vdots$	$\vdots$
$[2K+1+\rho K]_N$	1	$g_{\rho K}^3$
$\vdots$	$\vdots$	$\vdots$
$[N-K+1]_N$	1	$g_1^{N/K}$
$\vdots$	$\vdots$	$\vdots$
$[N-K+1+\rho K]_N$	1	$g_{\rho K}^{N/K}$

Table 7.1: Transmission scheme where $[x]_N = (x-1)_N + 1$, $(x)_N$ is x mod N, and g_i^j is the i^{th} bit of the encoding of the K bits with indices $((j-1)K+1, \ldots, jK)$ by the given sender. (So g_3^1 is the 3^{rd} bit of the encoding of $x_1, x_2, \ldots, x_K$).

$$= \Theta(N(\log\log N + (1/R)))$$

7.3 Summary

A reliable protocol for obtaining the entire array of N bits at one node on the distributed communication network of N nodes has been presented. By employing a random coding argument, this protocol gives reliable estimates of the array using $\Theta(N(\log\log N))$ transmissions.

Appendix A

Probability of Error for the 2×2 Code

Our first step in calculating the probability of error is to establish friendly coordinate axes. We will use the following orthonormal set of axes: $\frac{1}{2}\langle 1,-1,-1,1\rangle$, $\frac{1}{2}\langle 1,1,1,1\rangle$, $\frac{1}{2}\langle -1,-1,1,1\rangle$, $\frac{1}{2}\langle 1,-1,1,-1\rangle$. Note that the first axis goes from the origin towards the codeword $(1,-1,-1,1)$. The remaining three axes are in the hyperplane $\mathbf{h}$. Thus, they are equidistant from the two codewords.

In the original coordinate basis, the four dimensions of the noise vector n were independent. To see that this independence is preserved in the new basis system, we start by verifying that the new noise vectors $\hat{n}_1, \hat{n}_2, \hat{n}_3, \hat{n}_4$ are uncorrelated. Let

$$W = \frac{1}{2}\begin{pmatrix} 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & -1 \\ -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 \end{pmatrix}$$

be the transformation matrix. Thus, $\hat{n} = Wn$. The covariance matrix is given by

$$\begin{aligned} \hat{C} &= \mathbf{E}[(\hat{n}-\hat{u})(\hat{n}-\hat{u})^T] \\ &= \mathbf{E}[(Wn-Wu)(Wn-Wu)^T] \\ &= W\mathbf{E}[(n-u)(n-u)^T]W^T \end{aligned}$$

$$= WCW^T$$

Since $C = I_4$, $\hat{C} = I_4$ (where I_4 is the identity matrix of rank four). Thus, the noise vectors are uncorrelated. In addition, if two normal random variables are uncorrelated, then they are independent. Now that we have established the independence of the noise vectors $\hat{n}$, we can find the probability of error of the system. (Note that the point $(1, -1, -1, 1)$ is expressed as $2w_1$ in our new basis.)

$$\begin{aligned} p_e &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{0}\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^4 \exp\left[-\frac{(z_1-2)^2+{z_2}^2+{z_3}^2+{z_4}^2}{2\sigma^2}\right] dz_1\, dz_2\, dz_3\, dz_4 \\ &= \left[\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{-z^2}{2\sigma^2}\right) dz\right]^3 \int_{-\infty}^{0}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{-(z_1-2)^2}{2\sigma^2}\right) dz_1 \end{aligned}$$

Note that

$$\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{z^2}{2\sigma^2}\right) dz = 1$$

And hence,

$$\begin{aligned} p_e &= \int_{-\infty}^{0}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{-(z-2)^2}{2\sigma^2}\right) dz \\ &= G\left(-\frac{2}{\sigma}\right) - G(-\infty) \\ &= 1 - G\left(\frac{2}{\sigma}\right) \\ &= \frac{1}{2} - \text{erf}\left(\frac{2}{\sigma}\right) \end{aligned}$$

where G is the integral of the Gaussian function [Pap84]. This probability of error indicates the theoretical performance of the 2×2 code.

Appendix B

Enumerating the Classes of the 6×6 Code

The 6×6 code has ninety codewords of six classes. Table 4.1 lists the possible rows of the codewords. Here we give an example codeword, by rows, of each class and count the number of codewords of each class.

Class 1: $AAAA'A'A'$: First choose one of the ten pairs of rows. Then order the six rows of two classes.

$$\binom{10}{1}\binom{6}{3} = 200 \text{ codewords}$$

Class 2: $AAA'A'BB'$: Pick two pairs from the ten pairs of rows. Pick one of these pairs to appear twice. Pick two positions for the A's; pick two more positions for the A''s and one for the B.

$$\binom{10}{2}\binom{2}{1}\binom{6}{2}\binom{4}{2}\binom{2}{1} = 16,200 \text{ codewords}$$

Class 3 : $AA'BB'CC'$: Pick three pairs from ten. Order the six distinct rows.

$$\binom{10}{3}6! = 86,400 \text{ codewords}$$

Before continuing, we need to count the number of ways in which four nonpaired rows can sum to zero. (Classes 4 and 5 have two rows that cancel each other (A and A') and therefore, four rows that must sum to zero.) Four such rows are $AJD'F'$. The distance between any two of these rows is four. (Two rows differ in four positions.) The number of legal rows at distance four from A is $\binom{3}{2}\binom{3}{2} = 9$ (pick two -1's to change to $+1$'s, and pick two $+1$'s to change to -1's). Once we pick another row, J for example, we have only $\binom{2}{1}\binom{2}{1}/2$ ways left to complete the four-row matrix. (J and A agree in one $+1$ and one -1 position. The next rows, which are at distance four from A and J, must each agree in one of A's two remaining $+1$ positions and one of the remaining -1 positions. We divide by two because we take two rows at a time to complete the matrix.) So, we have found that there are two ways that AJ can be completed to yield a four-row matrix with zero row and column sums. In fact, these matrices are $AJD'F'$ and $AJC'G'$. In a similar way, pairing with A each of the nine rows at distance four from A will give rise to two legal third and fourth rows. So, how many legal (four-row, zero row and column sums) matrices have A in them?

$$\begin{aligned}\#\text{ matrices with } A &= \frac{(\#\text{ rows at dist. 4})(\#\text{ ways to finish each pair})}{(\text{rows per } A)}\\ &= \frac{9\cdot 2}{3}\\ &= 6\end{aligned}$$

These six matrices are $AJD'F'$, $AJC'G'$, $AHB'F'$, $AIB'G'$, $AHC'E'$, and $AID'E'$. Now, how many four-row sets are there that sum to zero?

$$\begin{aligned}&\#\text{ of four row sets that sum to zero}\\ &\qquad = \frac{(\#\text{ with } A)(\#\text{ legal rows})}{(\#\text{ rows per set})}\\ &\qquad = \frac{6\cdot 20}{4}\\ &\qquad = 30\end{aligned}$$

So, A appears in six of these sets, A' appears in six more, and neither A or A' appears in 18 ($= 30 - 6 - 6$) of the sets.

Class 4 : $AA'AJD'F'$: Pick one of the ten pairs. Pick A or A' to appear in the

pair 1:	1	1	1	−1	−1	−1
	1	1	−1	1	−1	−1
pair 2:			1	1		
			1	1		
pair 3:			−1	−1		
			−1	−1		

Table B.1: Partial 6×6 codeword of class six.

set of four. There are six ways to finish the set of four. Next order the rows. Pick two positions for the AA or $A'A'$. Order the remaining four distinct rows.

$$\binom{10}{1}\binom{2}{1}\binom{6}{1}\binom{6}{2}4! = 43,200 \text{ codewords}$$

Class 5 : $BB'AJD'F'$: Pick one of ten pairs. There are 18 sets of four rows that can complete the matrix. Then we order the six distinct rows.

$$\binom{10}{1}\binom{18}{1}6! = 129,600 \text{ codewords}$$

Codewords of class six have six distinct, nonpaired rows. We can break the six rows into three pairs of rows. Two rows in a pair differ in two positions. If we pick A as our first row, there are $\binom{3}{1}\binom{3}{1} = 9$ rows at distance two from A. Let us take B for the second row. A and B differ in the third and fourth positions. Since two rows in a pair disagree in only two positions, the rows in the remaining two pairs must agree in the third and fourth positions. (If they disagreed in the third and fourth, they would have to agree in the first, second, fifth and sixth; but then we would not have three $+1$ entries and three -1 entries in the first, second, fifth and sixth columns.) So, given A and B, we have the partial codeword as shown in Table B.1.

Pair 2 can either agree or disagree in positions five and six. If they agree, they must have $+1$ entries (because we need three $+1$ entries and three -1 entries in columns five and six). But the pair 2 rows have two $+1$ entries already. Therefore, pair 2 must disagree in positions five and six. Thus, the pairs 2 and 3 are fixed by our selection of pair 1 rows. The final six rows are given in Table B.2. How many

$$\begin{array}{lrrrrrr}
A & 1 & 1 & 1 & -1 & -1 & -1 \\
B & 1 & 1 & -1 & 1 & -1 & -1 \\
C' & -1 & -1 & 1 & 1 & -1 & 1 \\
D' & -1 & -1 & 1 & 1 & 1 & -1 \\
E' & -1 & 1 & -1 & -1 & 1 & 1 \\
J & 1 & -1 & -1 & -1 & 1 & 1
\end{array}$$

Table B.2: 6×6 codeword of class six.

different sets of six rows are there? A appears in nine such sets, there are twenty rows each of which appears nine times (with the nine rows that differ in two positions), and each codeword uses six rows.

Thus,

$$\text{number of different sets of six rows} = \frac{9 \cdot 20}{6} = 30$$

Class 6 : $ABJC'D'E'$. Pick one of the 30 sets of rows. Order the six rows.

$$\binom{30}{1} 6! = 21,600 \text{ codewords}$$

Appendix C

Proof of Theorem 7.1 and Theorem 7.2

C.1 Theorem 7.1

Theorem 7.1 For a distributed communication network of N nodes, let M be the number of times each node has to repeat its bit for r to have a reliable estimate of the array of N bits. Then $M = \Theta(\log N)$.

Proof

The following two claims prove the theorem.

Claim If $\Pr(x_i \neq y_i :\ 1 \text{ transmission}) \leq \epsilon$, and we would like $\Pr(x_i \neq y_i :\ M \text{ transmissions}) \leq (\epsilon/K)$, then $M = \Theta(\log K)$.

Proof Let x_i be the transmitted bit and y_i be the received bit.

$$\begin{aligned}\Pr(x_i \neq y_i :\ M \text{ transmissions}) &= \sum_{i=M/2}^{M} \binom{M}{i} \epsilon^i (1-\epsilon)^{M-i} \\ &\leq \lambda^{-\lambda M} \mu^{-\mu M} \epsilon^{\lambda M} (1-\epsilon)^{\mu M}\end{aligned}$$

where $\lambda M = M/2$ and $\mu M = M - \lambda M = M/2$ (see [PW72]). Thus,

$$\begin{aligned}\Pr(x_i \neq y_i :\ M \text{ transmissions}) &\leq \left(\frac{1}{2}\right)^{-M} (\epsilon(1-\epsilon))^{\frac{M}{2}} \\ &= \left(2\sqrt{\epsilon(1-\epsilon)}\,\right)^{M}\end{aligned}$$

For $0 \le \epsilon < 1/2$, $(2\sqrt{\epsilon(1-\epsilon)}\ ) < 1$; therefore,

$$\Pr(x_i \neq y_i : \ M \text{ transmissions}) \le (\alpha)^M$$

(where $\alpha < 1$, as determined by ϵ).

If one less bit had been sent, the desired probability of error ($\le \epsilon/K$) would not be met (since we would choose the minimum M). Therefore,

$$\Pr(x_i \neq y_i : \ (M-1) \text{ transmissions}) \ = \ \alpha^{M-1} > \frac{\epsilon}{K}$$

inverting this we obtain

$$\left(\frac{1}{\alpha}\right)^{M-1} < \frac{K}{\epsilon}$$

and thus,

$$M \ = \ O(\log K) \tag{C.1}$$

□

Also, to find a lower bound:

$$\begin{aligned}
\binom{M}{\frac{M}{2}}\epsilon^{\frac{M}{2}}(1-\epsilon)^{\frac{M}{2}} < \sum_{i=M/2}^{M}\binom{M}{i}\epsilon^{i}(1-\epsilon)^{M-i} \ &= \ \Pr(x_i \neq y_i : \ M \text{ transm.}) \\
\binom{M}{\frac{M}{2}}\epsilon^{\frac{M}{2}}(1-\epsilon)^{\frac{M}{2}} < \Pr(x_i \neq y_i : \ M \text{ transmissions}) \ &\le \ \frac{\epsilon}{K} \\
\left(\frac{1}{2}\right)\sqrt{\pi}\left(\frac{1}{\sqrt{2\pi M(\frac{1}{2})(\frac{1}{2})}}\right)\left(\frac{1}{2}\right)^{-M}(\epsilon(1-\epsilon))^{\frac{M}{2}} \ &< \ \frac{\epsilon}{K} \quad \text{[PW72]} \\
\left(\frac{1}{\sqrt{2M}}\right)(2\sqrt{\epsilon(1-\epsilon)}\ )^{M} \ &< \ \frac{\epsilon}{K}
\end{aligned}$$

and hence,

$$M \ = \ \Omega(\log K) \tag{C.2}$$

Therefore, by Equation C.1 and Equation C.2 :

$$M = \Theta(\log K).$$

Claim If $\Pr(x_i \neq y_i) \leq \delta/K$ and we would like $\Pr(\mathbf{x} \neq \mathbf{y}) \leq \delta$, then $K = \Theta(N)$.

Proof $\mathbf{x}$=$(x_1, x_2, \ldots x_N)$, $\mathbf{y}$=$(y_1, y_2, \ldots y_N)$, x_i is the transmitted bit, y_i is the received bit, and $0 \leq \delta < (1/2)$.

$$\begin{aligned} \Pr(\mathbf{x} \neq \mathbf{y}) &= \sum_{i=1}^{N} \binom{N}{i} \left(\frac{\delta}{K}\right)^{i} \left(1 - \frac{\delta}{K}\right)^{N-i} \\ &= 1 - \left(1 - \frac{\delta}{K}\right)^{N} \end{aligned}$$

We would like $\Pr(\mathbf{x} \neq \mathbf{y}) \leq \delta$, using the above equation we obtain

$$\begin{aligned} 1 - \left(1 - \frac{\delta}{K}\right)^{N} &\leq \delta \\ \text{or } K &\geq \frac{\delta}{1 - (1-\delta)^{1/N}} \end{aligned}$$

Call $(\frac{c}{1-(1-\delta)^{1/N}}) = A$ (where c is some constant). Now, let us assume that $K \leq A$. Therefore :

$$1 - \left(1 - \frac{\delta}{A}\right)^{N} \leq 1 - \left(1 - \frac{\delta}{K}\right)^{N} = \Pr(\mathbf{x} \neq \mathbf{y}) \leq \delta$$

or

$$1 - \left(1 - \frac{\delta}{A}\right)^{N} \leq \delta$$

which results in

$$\frac{\delta}{A} \leq 1 - (1-\delta)^{1/N}$$

or

$$\frac{\delta}{c}\left(1 - (1-\delta)^{1/N}\right) \leq 1 - (1-\delta)^{1/N}$$

Choose $c = 1$ so that $\delta/c \leq 1$ for all δ. Thus:

$$K \leq \frac{1}{1-(1-\delta)^{1/N}}$$

therefore,

$$K = \Theta\left(\frac{1}{1-(1-\delta)^{1/N}}\right)$$

Furthermore, this is equivalent to $\Theta(N)$, since

$$\lim_{N\to\infty} \frac{N}{\left(\frac{1}{1-(1-\delta)^{1/N}}\right)} = \text{constant}.$$

To see this, let

$$N\left(1-(1-\delta)^{1/N}\right) = b$$

Thus,

$$1-\frac{b}{N} = (1-\delta)^{1/N} = \sqrt[N]{\alpha} \qquad (\alpha = 1-\delta)$$

or

$$\alpha = \left(1-\frac{b}{N}\right)^{N}$$

and

$$\lim_{N\to\infty} \alpha = \lim_{N\to\infty}\left(1-\frac{b}{N}\right)^{N} = e^{-b}$$

Since α is a constant, therefore, e^{-b} is a constant; thus b *is* a constant. Therefore, $K = \Theta(N)$. □ Together, the claims prove the theorem. ∎

C.2 Theorem 7.2

Theorem 7.2 If we communicate N bits of information over a binary symmetric channel (BSC) by sending K/R bits per group of K bits (for N/K groups), then the probability of error of the whole array of N bits goes asymptotically to zero when $K = W\log N$ for some constant W. **Proof** Given a BSC with crossover error probability ϵ, if the sending end of the channel has K bits of information it wishes to have the receiving end know, it has been shown that $\Theta(K/R)$ bits suffice ($R < C = 1 - \mathcal{H}(\epsilon)$, where $\mathcal{H}$ is the binary entropy function) and give a decoder error probability for the K bits given approximately by $2^{-(KE(R)/R)}$ [McE77].

If we now look at all the groups of size K together, then the probability of error of the array of N bits is given by

$$\begin{aligned} p_e(\text{array of } N) &= 1 - \left(1 - 2^{(-KE(R)/R)}\right)^{N/K} \\ &= 1 - (1 - \beta K/N)^{N/K} \\ &\approx \beta \\ \text{where } \beta &= N/K2^{KE(R)/R} \end{aligned}$$

We would like β to approach zero asymptotically as N gets large; thus, we can take $K = W \log N$ and $E(R) \cdot W/R > 1$ ($E(R), W$ and R are $\Theta(1)$; therefore, K is $\Omega(\log N)$ as β goes to 0, for N approaching infinity. We wish to have K as small as possible to minimize the necessary number of transmissions.)

Thus, we have

$$p_e(\text{array of } N) \approx \beta = \frac{1}{(\log N)N^{\frac{E(R)W}{R}-1}}$$

which goes to zero as N goes to infinity (with $E(R)W/R > 1$).

[MO77] give bounds for the error exponent $E(R)$. In figure C.1 we plot two lower bounds for $E(R)$, $E_{\text{ex}}(R)$ and $E_{\text{r}}(R)$, for several values of ϵ. $E_{\text{r}}(R)$ is the random coding bound given here by

$$E_{\text{r}}(R) = \begin{cases} 1 - R - \log_2(1 + \sqrt{4\epsilon(1-\epsilon)}) & 0 \le R \le 1 - H_2(\frac{\sqrt{\epsilon}}{\sqrt{\epsilon}+\sqrt{1-\epsilon}}) \\ T_\epsilon(D) - H_2(D) & R \ge 1 - H_2(\frac{\sqrt{\epsilon}}{\sqrt{\epsilon}+\sqrt{1-\epsilon}}) \end{cases}$$

where $T_\epsilon(D) = -D \log_2 \epsilon - (1-D)\log_2(1-\epsilon)$ and D is defined by $R = 1 - H_2(D)$, for $H_2(D) = T_D(D)$, the binary entropy function. $E_{\text{ex}}(R)$ is the expurgated bound here given by

$$E_{\text{ex}}(R) = -D\left(1 + \frac{1}{2}\log_2 \epsilon(1-\epsilon)\right)$$

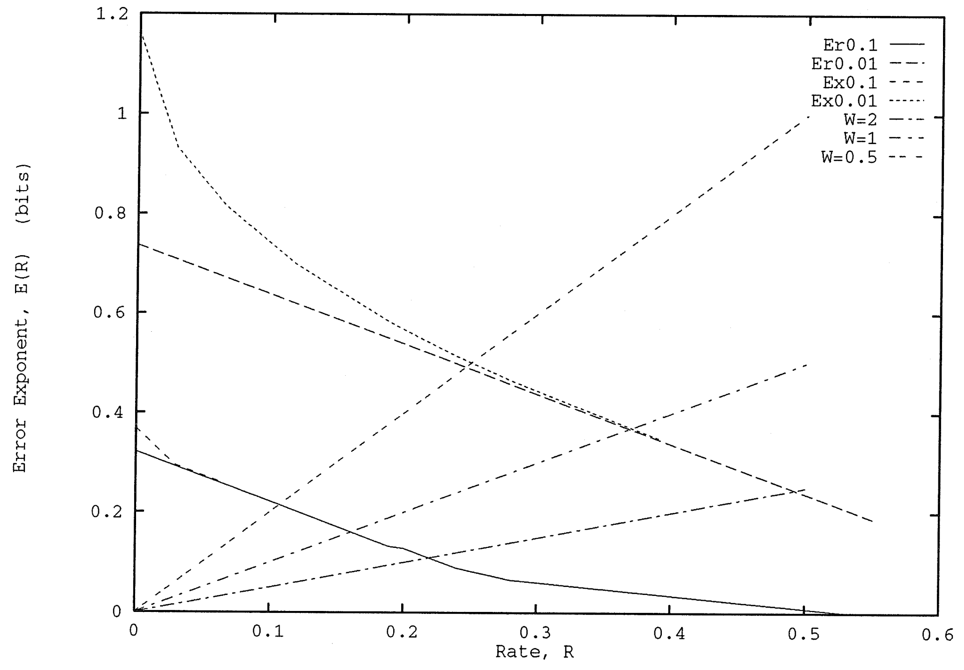


Figure C.1: E_{r} and E_{ex} lower bounds for the error exponent E(R) for $\epsilon = 0.1$ (Ex0.1 and Er0.1) and $\epsilon = 0.01$ (Ex0.01 and Er0.01) plotted against $W = 0.5$, 1.0 and 2.0.

$E_{\text{ex}}(R)$ is only valid from where it intersects $E_{\text{r}}(R)$ until $R = 0$. For more general forms of these bounds, see the reference. Also indicated on figure C.1 are lines for $W = 1$, $W = 1/2$ and $W = 2$. From the figure, we see that there exist rates R for which $E(R) > R/W$. For example, the space above and to the left of the $W = 1/2$ line, and above and to the right of the curves lower bounding $E(R)$ is the location of rates that satisfy $E(R)/2 > R$.

Thus, figure C.1 shows that there exist rates R, with $E(R) \cdot W > R$, at which we can transmit reliably. (Even if we take the tougher restriction $E(R) > 1$, we find that this implies that $\epsilon < 1/64$ (using $E_{\text{ex}}(R)$), a restriction we can always secure with a fixed number of repetitions.) $\blacksquare$

References

[AMJ85] Yaser Abu-Mostafa and Jeannine St. Jacques. Information capacity of the Hopfield network. *IEEE Transactions on Information Theory*, IT-31:461–464, 1985.

[AR88] James A. Anderson and Ed Rosenfeld, editors. *Neurocomputing: Foundations of Research.* MIT Press, Cambridge, 1988.

[BB87] Jehoshua Bruck and Mario Blaum. Neural network, error-correcting codes and polynomials over the binary n-cube. Technical Report RJ 6003 (59602), IBM Research Division, 1987.

[BMM$^+$89] Griff Bilbro, Reinhold Mann, Thomas K. Miller, Wesley E. Snyder, David E. Van den Bout, and Mark White. Optimization by mean field annealing. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 91–98. Morgan Kaufman, San Mateo, CA, 1989.

[BMvT78] Elwyn R. Berlekamp, Robert J. McEliece, and Hank C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, IT-24:384–386, 1978.

[BSS90] A. E. Brouwer, James B. Shearer, and N. J. A. Sloane. A new table of constant weight codes. *IEEE Transactions on Information Theory*, IT-36:1334–1380, 1990.

[CG87] Tzi-Dar Chiueh and Rodney Goodman. A neural network classifier based on coding theory. In Dana Z. Anderson, editor, *Neural Information Processing Systems*, pages 174–183, 1987.

[DK74] J. Dénes and A.D. Keedwell. *Latin Squares and Their Applications.* Academic Press, New York, 1974.

[EAM88] Ruth A. Erlanson and Yaser S. Abu-Mostafa. Using an analog neural network for decoding. In David S. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 186–190. Morgan Kaufman, San Mateo, CA, 1988.

[EAM91] Ruth A. Erlanson and Yaser S. Abu-Mostafa. Analog neural networks as decoders. In *Advances in Neural Information Processing Systems III.* Morgan Kaufman, San Mateo, CA, 1991. to appear.

[El 87] A. El Gamal. Reliable information of highly distributed information. In Thomas M. Cover and B. Gopinath, editors, *Open Problems in Communication and Computation*, pages 60–62. Springer-Verlag, New York, 1987.

[Gal85] Robert G. Gallager. Finding parity in a broadcast network. In *IEEE Symposium on Information Theory*, 1985.

[Gal87] Robert G. Gallager. Finding parity in a broadcast network. In Thomas M. Cover and B. Gopinath, editors, *Open Problems in Communication and Computation*, pages 208–209. Springer-Verlag, New York, 1987.

[Hop82] John J. Hopfield. Neuron networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science*, 79:2554–2558, 1982.

[Hop84] John J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science*, 81:3088–3092, 1984.

[HT85] John J. Hopfield and D. W. Tank. "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[KGV83] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Knu88] Allen Knutson, July 1988. personal communication.

[Kos91] Klaus-Uwe Koschnick. Some new constant weight codes. *IEEE Transactions on Information Theory*, IT-37:370–371, 1991.

[LRMM89] John Lazzaro, Sylvie Ryckebusch, Michelle A. Mahowald, and Carver A. Mead. Winner-take-all networks of $O(N)$ complexity. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 703–711. Morgan Kaufman, San Mateo, CA, 1989.

[MC80] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley, Reading, Mass., 1980.

[McE77] Robert J. McEliece. *The Theory of Information and Coding.* Addison-Wesley, Reading, Mass., 1977.

[MEAM89] Eric Majani, Ruth A. Erlanson, and Yaser S. Abu-Mostafa. On the k-winners-take-all feedback network. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 634–642. Morgan Kaufman, San Mateo, CA, 1989.

[MO77] Robert J. McEliece and J. K. Omura. An improved upper bound on the block coding error exponent for binary-input discrete memoryless channels. *IEEE Transactions on Information Theory*, IT-23(5):611–613, September 1977.

[MPRV87] Robert J. McEliece, Edward C. Posner, E. R. Rodemich, and S. S. Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, IT-33:461–482, 1987.

[MTD88] A. Moopen, A. P. Thakoor, and T. Duond. A neural network for Euclidean distance minimization. In *Proceedings of the Second International Conference on Neural Networks, IEEE ICNN*, volume 2, pages 349–356, 1988.

[Pap84] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*, page 47. McGraw-Hill, New York, second edition, 1984.

[PB88] John C. Platt and Alan H. Barr. Constrained differential optimization for neural networks. Technical Report CS-TR–88-17, California Institute of Technology, 1988.

[PH86] John C. Platt and John J. Hopfield. Analog decoding using neural networks. In John Denker, editor, *Neural Networks for Computing*, pages 364–369. American Institute of Physics, 1986.

[PW72] W. Peterson and E. Weldon. *Error Correcting Codes.* MIT Press, Cambridge, 1972. Appendix A.

[RM86] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing*, volume 1,2. MIT Press, Cambridge, 1986.

[Sha48] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[Sou89] Nicolas Sourlas. Spin-glass models as error-correcting codes. *Nature*, 339:693–662, 1989.

[Tho80] Clark D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Carnegie-Mellon University, 1980.

[Tou89] David S. Touretzky. Analyzing the energy landscapes of distributed winner-take-all networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 626–633. Morgan Kaufman, San Mateo, CA, 1989.