CSC 310: Information Theory

University of Toronto, Fall 2011

Instructor: Radford M. Neal

Week 11

More on Hamming Distance

Recall that the Hamming distance, $d(\mathbf{u}, \mathbf{v})$, of two codewords $\mathbf{u}$ and $\mathbf{v}$ is the number of positions where $\mathbf{u}$ and $\mathbf{v}$ have different symbols.

This is a proper distance, which satisfies the *triangle inequality*:

$$d(\mathbf{u}, \mathbf{w}) \ \leq \ d(\mathbf{u}, \mathbf{v}) + d(\mathbf{v}, \mathbf{w})$$

Here's a picture showing why:

```
u :  0 1 1 0 0 1 1 0 1 1 1 0
                - - - - - -
v :  0 1 1 0 0 1 0 1 0 0 0 1
              - - -         - -
w :  0 1 1 1 1 0 0 1 0 0 1 0
```

Here, $d(\mathbf{u}, \mathbf{v}) = 6$, $d(\mathbf{u}, \mathbf{v} = 5)$, and $d(\mathbf{u}, \mathbf{w}) = 7$.

Minimum Distance and Decoding

A code's *minimum distance* is the minimum of $d(\mathbf{u}, \mathbf{v})$ over all distinct codewords $\mathbf{u}$ and $\mathbf{v}$.

If the minimum distance is at least $2t + 1$, a nearest neighbor decoder will always decode correctly when there are t or fewer errors.

Here's why: Suppose the code has distance $d \geq 2t + 1$. If $\mathbf{u}$ is sent and $\mathbf{v}$ is received, having no more than t errors, then

- $d(\mathbf{u}, \mathbf{v}) \leq t$.
- $d(\mathbf{u}, \mathbf{u}') \geq d$ for any codeword $\mathbf{u}' \neq \mathbf{u}$.

From the triangle inequality:

$$d(\mathbf{u}, \mathbf{u}') \leq d(\mathbf{u}, \mathbf{v}) + d(\mathbf{v}, \mathbf{u}')$$

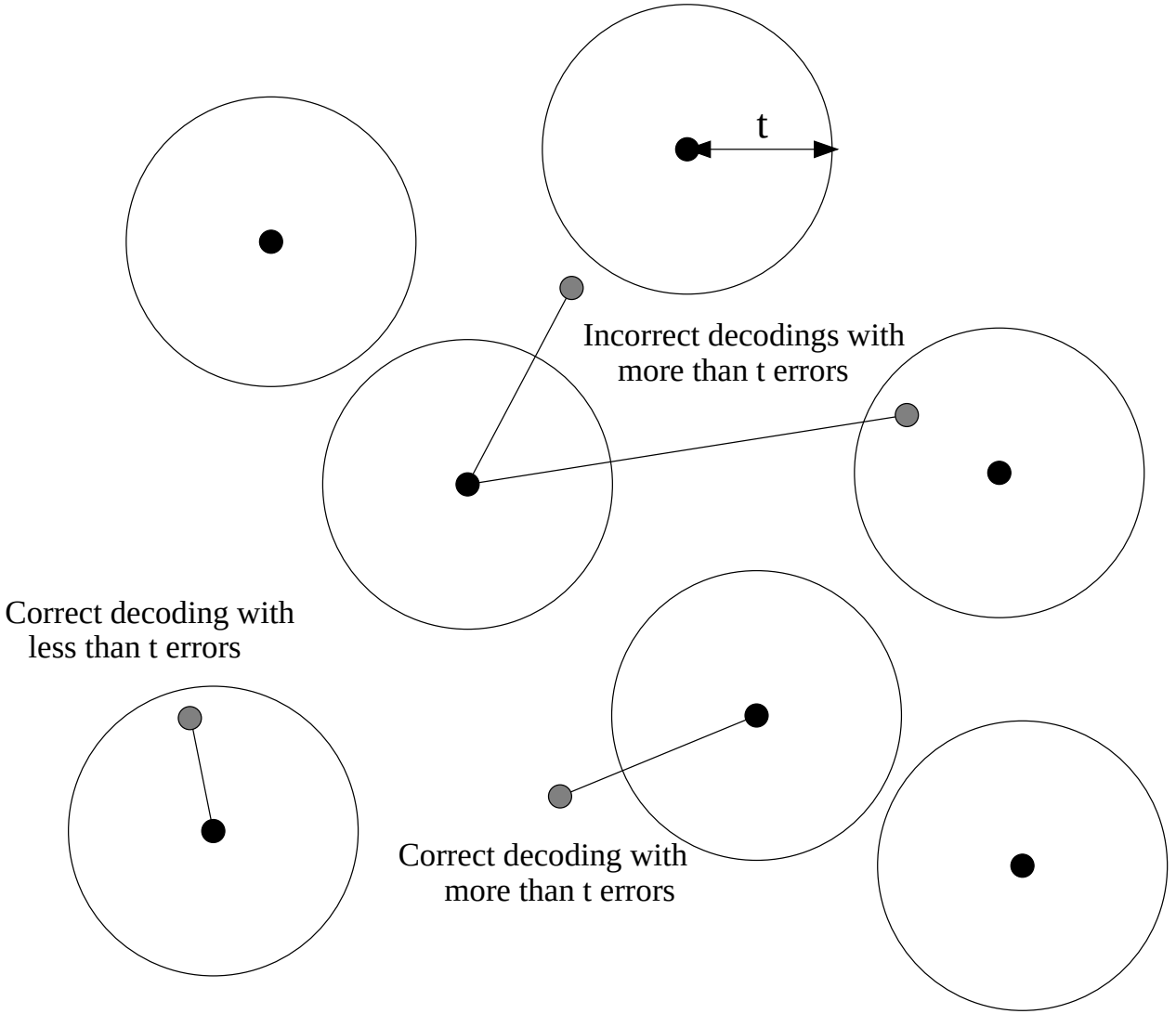
It follows that

$$d(\mathbf{v}, \mathbf{u}') \geq d(\mathbf{u}, \mathbf{u}') - d(\mathbf{u}, \mathbf{v}) \geq d - t \geq (2t + 1) - t \geq t + 1$$

The decoder will therefore decode correctly to $\mathbf{u}$, at distance t, rather than to some other $\mathbf{u}'$, at distance at least $t + 1$.

A Picture of Distance and Decoding

Here's a picture of codewords (black dots) for a code with minimum distance $2t + 1$, showing how some transmissions are decoded:



Minimum Distance for Linear Codes

To find the minimum distance for a code with 2^K codewords, we will in general have to look at all $2^K(2^K-1)/2$ pairs of codewords.

But there's a short-cut for linear codes...

Suppose two distinct codewords $\mathbf{u}$ and $\mathbf{v}$ are a distance d apart. Then the codeword $\mathbf{u}-\mathbf{v}$ will have d non-zero elements. The number of non-zero elements in a codeword is called its *weight*.

Conversely, if a non-zero codeword $\mathbf{u}$ has weight d, then the minimum distance for the code is at most d, since $\mathbf{0}$ is a codeword, and $d(\mathbf{u},\mathbf{0})$ is equal to the weight of $\mathbf{u}$.

So the minimum distance of a linear code is equal to the minimum weight of the 2^K-1 non-zero codewords. (This is useful for small codes, but when K is large, finding the minimum distance is difficult in general.)

Examples of Minimum Distance and Error Correction for Linear Codes

Recall the linear $[5, 2]$ code with the following codewords:

$$00000 \quad 00111 \quad 11001 \quad 11110$$

The three non-zero codewords have weights of 3, 3, and 4. This code therefore has minimum distance 3, and can correct any single error.

The single-parity-check code with $N = 4$ has the following codewords:

$$0000 \quad 0011 \quad 0101 \quad 0110$$
$$1001 \quad 1010 \quad 1100 \quad 1111$$

The smallest weight of a non-zero codeword above is 2, so this is the minimum distance of this code. This is too small to guarantee correction of even one error. (Though the presence of a single error can be detected.)

Finding Minimum Distance From a Parity-Check Matrix

We can find the minimum distance of a linear code from a parity-check matrix for it, H. The minimum distance is equal to the smallest number of linearly-dependent columns of H.

Why? A vector $\mathbf{u}$ is a codeword iff $\mathbf{u}H^T = \mathbf{0}$. If d columns of H are linearly dependent, let $\mathbf{u}$ have 1s in those positions, and 0s elsewhere. This $\mathbf{u}$ is a codeword of weight d. And if there were any codeword of weight less than d, the 1s in that codeword would identify a set of less than d linearly-dependent columns of H.

Special cases:

- If H has a column of all zeros, then $d = 1$.
- If H has two identical columns, then $d \leq 2$.
- For binary codes, if all columns are distinct and non-zero, then $d \geq 3$.

Example: The $[7, 4]$ Hamming Code

We can define the $[7, 4]$ Hamming code by the following parity-check matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Clearly, all the columns of H are non-zero, and they are all distinct. So $d \geq 3$. We can see that $d = 3$ by noting that the first three columns are linearly dependent, since

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

This produces 1110000 as an example of a codeword of weight three.

Since it has minimum distance 3, this code can correct any single error.

Hamming Codes

We can see that a binary $[N, K]$ code will correct any single error if all the columns in its parity-check matrix are non-zero and distinct.

One way to achieve this: Make the $N - K$ bits in successive columns be the binary representations of the integers 1, 2, 3, etc.

This is one way to get a parity-check matrix for a $[7, 4]$ Hamming code:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

When N is a power of two minus one, the columns of H contain binary representations of all non-zero integers up to $2^{N-K} - 1$.

These are called the *Hamming codes*.

Encoding Hamming Codes

By rearranging columns, we can put the parity-check matrix for a Hamming code in systematic form. For the $[7, 4]$ code, we get

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Recall that a systematic parity check matrix of the form $[P^T \mid I_{N-K}]$ goes with a systematic generator matrix of the form $[I_K \mid P]$. We get

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

We encode a message block, $\mathbf{s}$, of four bits, by computing $\mathbf{t} = \mathbf{s}G$. The first four bits of $\mathbf{t}$ are the same as $\mathbf{s}$; the remaining three are "check bits". Note: The order of bits may vary depending on how we construct the code.

Decoding Hamming Codes

Consider the non-systematic $[7, 4]$ Hamming code parity-check matrix:

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Suppose $\mathbf{t}$ is sent, but $\mathbf{r} = \mathbf{t} + \mathbf{n}$ is received. The receiver can compute the *syndrome*, $\mathbf{z} = \mathbf{r}H^T$. Using the fact that $\mathbf{t}H^T = \mathbf{0}$ for a codeword $\mathbf{t}$,

$$\mathbf{z} = \mathbf{r}H^T = (\mathbf{t} + \mathbf{n})H^T = \mathbf{t}H^T + \mathbf{n}H^T = \mathbf{n}H^T$$

If there were no errors, $\mathbf{n} = \mathbf{0}$, so $\mathbf{z} = \mathbf{0}$.

If there is one error, in position i, then $\mathbf{n}H^T$ will be the ith column of H — which contains the binary representation of the number i!

So, to decode, compute the syndrome, $\mathbf{z}$, and if it is non-zero, flip the bit it identifies. If we rearranged H to systematic form, we modify this procedure in corresponding fashion.

Syndrome Decoding in General

For any linear code with parity-check matrix H, we can find the nearest-neighbor decoding of a received block, $\mathbf{r}$, using the syndrome, $\mathbf{z} = \mathbf{r}H^T$.

We write the received data as $\mathbf{r} = \mathbf{t} + \mathbf{n}$, where $\mathbf{t}$ is the transmitted codeword, and $\mathbf{n}$ is the *error pattern*, so that $\mathbf{z} = \mathbf{n}H^T$.

A nearest-neighbor decoding can be found by finding an error pattern, $\mathbf{n}$, that produces the observed syndrome, and which has the smallest possible weight. Then we decode $\mathbf{r}$ as $\mathbf{r} - \mathbf{n}$.

Building a Syndrome Decoding Table

We can build a table indexed by the syndrome that gives the error pattern of minimum weight for each syndrome.

We initialize all entries in the table to be empty.

We then consider the non-zero error patterns, $\mathbf{n}$, in some order of non-decreasing weight. For each $\mathbf{n}$, we compute the syndrome, $\mathbf{z} = \mathbf{n}H^T$, and store $\mathbf{n}$ in the entry indexed by $\mathbf{z}$, *provided* this entry is currently empty. We stop when the table has no empty entries.

Problem: The size of the table is exponential in the number of check bits — it has $2^{N-K} - 1$ entries for an $[N, K]$ code.

Example: The $[5, 2]$ Code

Recall the $[5, 2]$ code with this parity-check matrix:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Here is a syndrome decoding table for this code:

z	**n**
001	00001
010	00010
011	00100
100	01000
101	10000
110	10100
111	01100

The last two entries are not unique.

Hamming's Sphere-Packing Bound

We'd like to make the minimum distance as large as possible, or alternatively, have as many codewords as possible for a given distance. There's a limit, however.

Consider a binary code with $d = 3$, which can correct any single error. The "spheres" of radius one around each codeword must be disjoint — so that any single error leaves us closest to the correct decoding.

For codewords of length N, each such sphere contains $1+N$ points. If we have m codewords, the total number of points in all spheres will be $m\,(1+N)$, which can't be greater than the total number of points, 2^N.

So for binary codes that can correct any single error, the number of codewords is limited by

$$m \;\leq\; 2^N/(1+N)$$

A More General Version of the Bound

A binary code of length N that is guaranteed to correct any pattern of up to t errors can't have more than this number of codewords:

$$2^N \left(1 + \binom{N}{1} + \binom{N}{2} + \cdots + \binom{N}{t} \right)^{-1}$$

The kth term in the brackets is the number of possible patterns of k errors in N bits:

$$\binom{N}{k} = \frac{N!}{k!\,(N-k)!}$$

If the above bound is actually reached, the code is said to be *perfect*. For a perfect code, the disjoint spheres of radius t around codewords cover all points.

Very few perfect codes are known. Usually, we can't find a code with as many codewords as would be allowed by this bound.

Hamming Codes are Perfect

For each positive integer c, there is a binary Hamming code of length $N = 2^c - 1$ and dimension $K = N - c$. These codes all have minimum distance 3, and hence can correct any single error.

They are also perfect, since

$$2^N/(1+N) \;=\; 2^{2^c-1}/(1+2^c-1) \;=\; 2^{2^c-1-c} \;=\; 2^K$$

which is the number of codewords.

One consequence: A Hamming code can correct any single error, but if there is more than one error, it will not be able to give any indication of a problem — instead, it will "correct" the wrong bit, making things worse.

The *extended Hamming codes* add one more check bit (ie, they add one more row of all 1s to the parity-check matrix). This allows them to detect when two errors have occurred.

The Gilbert-Varshamov Bound

The sphere-packing bound is an *upper* limit on how many codewords we can have. There's also a *lower* limit, showing there *is* a code with at least a certain number of codewords.

There is a binary code of length N with minimum distance d that has at least the following number of codewords:

$$2^N \left(1 + \binom{N}{1} + \binom{N}{2} + \cdots + \binom{N}{d-1} \right)^{-1}$$

Why? Imagine spheres of radius $d-1$ around codewords in a code with fewer codewords than this. The number of points in each sphere is the sum above in brackets, so the total number of points in these spheres is less than 2^N. So there's a point outside these spheres where we could add a codeword that is at least d away from any other codeword.

Product Codes

A *product code* is formed from two other codes $\mathcal{C}_1$, of length N_1, and $\mathcal{C}_2$, of length N_2. The product code has length $N_1 N_2$.

We can visualize the $N_1 N_2$ symbols of the product code as a 2D array with N_1 columns and N_2 rows.

Definition of a product code: An array is a codeword of the product code if and only if

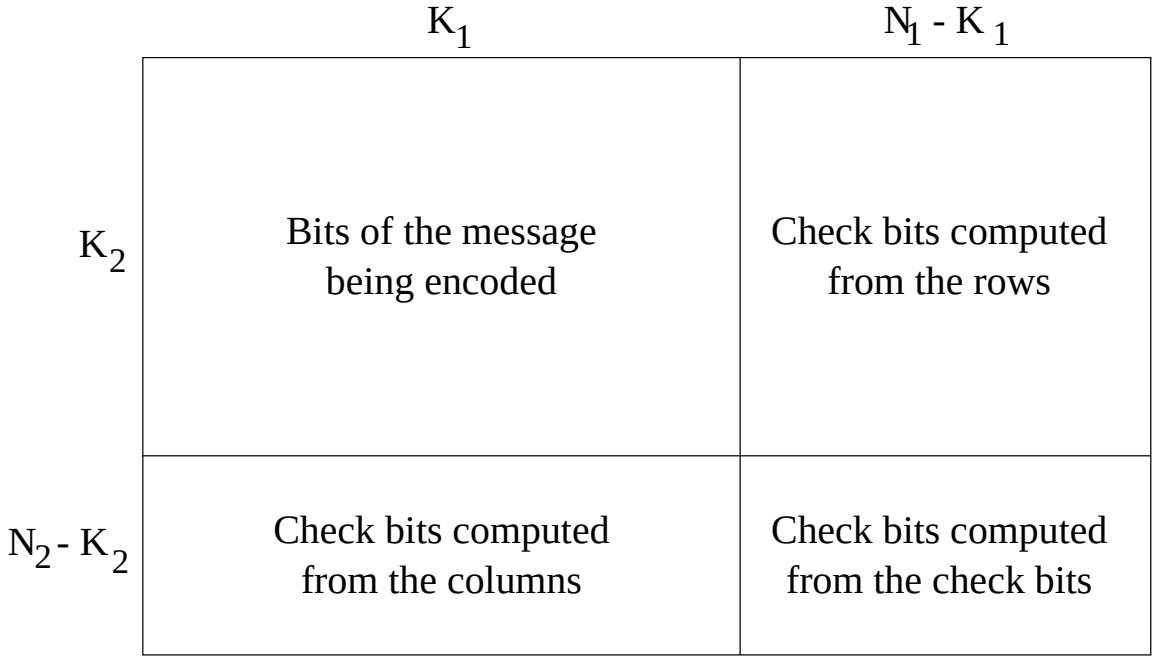
- all its rows are codewords of $\mathcal{C}_1$
- all its columns are codewords of $\mathcal{C}_2$

We will assume here that $\mathcal{C}_1$ and $\mathcal{C}_2$ are linear codes, in which case the product code is also linear. (Why?)

Dimensionality of Product Codes

Suppose $\mathcal{C}_1$ is an $[N_1, K_1]$ code and $\mathcal{C}_2$ is an $[N_2, K_2]$ code. Then their product will be an $[N_1 N_2, K_1 K_2]$ code.

Suppose $\mathcal{C}_1$ and $\mathcal{C}_2$ are in systematic form. Here's a picture a codeword of the product code:



The dimensionality of the product code is not more than $K_1 K_2$, since the message bits in the upper-left determine the check bits. We'll see that the dimensionality equals $K_1 K_2$ by showing how to find correct check bits for any message.

Encoding Product Codes

Here's a procedure for encoding messages with a product code:

1. Put $K_1 K_2$ message bits into the upper-left K_2 by K_1 corner of the N_2 by N_1 array.
2. Compute the check bits for each of the first K_2 rows, according to $\mathcal{C}_1$.
3. Compute the check bits for each of the N_1 columns, according to $\mathcal{C}_2$.

After this, all the columns will be codewords of $\mathcal{C}_2$, since they were given the right check bits in step (3). The first K_2 rows will be codewords of $\mathcal{C}_1$, since they were given the right check bits in step (2). But are the *last* $N_2 - K_2$ rows codewords of $\mathcal{C}_1$?

Yes! Check bits are linear combinations of message bits. So the last $N_2 - K_2$ rows are linear combinations of earlier rows. Since these rows are in $\mathcal{C}_1$, their combinations are too.

Minimum Distance of Product Codes

If $\mathcal{C}_1$ has minimum distance d_1 and $\mathcal{C}_2$ has minimum distance d_2, then the minimum distance of their product is $d_1 d_2$.

Proof:

Let $\mathbf{u}_1$ be a codeword of $\mathcal{C}_1$ of weight d_1 and $\mathbf{u}_2$ be a codeword of $\mathcal{C}_2$ of weight d_2. Build a codeword of the product code by putting $\mathbf{u}_1$ in row i of the array if $\mathbf{u}_2$ has a 1 in position i. Put zeros elsewhere. This codeword has weight $d_1 d_2$.

Furthermore, any non-zero codeword must have at least this weight. It must have at least d_2 rows that aren't all zero, and each such row must have at least d_1 ones in it.

Decoding Product Codes

Products of even small codes (eg, $[7, 4]$ Hamming codes) have lots of check bits, so decoding by building a syndrome table may be infeasible.

But if $\mathcal{C}_1$ and $\mathcal{C}_2$ can easily be decoded, we can decode the product code by first decoding the rows (replacing them with the decoding), then decoding the columns.

This will usually **not** be a nearest-neighbor decoder (and hence will be sub-optimal, assuming a BSC and equally-likely messages).

One advantage of product codes: They can correct some *burst errors* — errors that come together, rather than independently.

How Good Are Simple Codes?

Shannon's noisy coding theorem says we can get the probability of error in decoding a block, p_B, arbitrarily close to zero when transmitting at any rate, R, below the capacity, C — if we use good codes of large enough length, N.

For repetition codes, as N increases, $p_B \to 0$, but $R \to 0$ as well.

For Hamming codes, as $N = 2^c - 1$ increases, $R \to 1$, but $p_B \to 1$ as well, since there's bound to be more than one error in a really big block.

How Good are Products of Codes?

Let $\mathcal{C}$ be an $[N, K]$ code of minimum distance d (guaranteed to correct $t = \lfloor (d-1)/2 \rfloor$ errors).

What is the code we get by taking the product of $\mathcal{C}$ with itself p times like?

Length: $N_p = N^p$
Rate: $R_p = K^p/N^p = (K/N)^p \to 0$
Distance: $d_p = d^p$
Relative distance: $\rho_p = d_p/N_p = (d/N)^p \to 0$

The code can correct up to about $d_p/2$ errors, corresponding to the fraction of erroroneous bits being $\rho_p/2$.

For a BSC with error probability f, we expect that for large N, the fraction of erroneous bits in a block will be very close to f (ie, the number of erronenous bits will be close to Nf), from the "Law of Large Numbers".

So for large N, the codes have a low rate, and it's likely there are more errors than we can guarantee to correct (though possibly we might correct some error patterns beyond this guarantee).

Good Codes Aren't Easy to Find

In the 56 years since Shannon's noisy coding theorem, many schemes for creating codes have been found, but most of them *don't* allow one to reach the performance promised by theorem.

They can still be useful. For example, error correction in computer memory necessarily works on fairly small blocks (eg, 64 bits). Performance on bigger blocks is irrelevant.

But in other applications — computer networks, communication with spacecraft, digital television — we could use quite big blocks if it would help with error correction.

How can we do this in practice?