Contents

1 Lecture 1

1.1 What are Error-Correcting Codes?

Error-correcting codes provide a judiciously designed, noise-resilient redundant form of data.

The idea of "coding data", is robustness against noise through redundancy and the possibility of error recovery (pinpointing mistakes). Our hopes are:

- It is information-theoretically possible to recover the data
- Minimize redundancy (i.e. overhead)
- The algorithm to encode/recover the original data is efficient

Error-correcting codes have plenty of applications:

- Communication
- Storage (can be thought of communication in time)
- Central tool in discrete math, theoretical CS, information theory
- Quantum computing (heat and noise is present in every computation!)

We have many noise models in thinking about what we want our codes to protect against. We can mix and match to make a problem.

- Erasures, errors, deletions
- Bit level vs symbol/packet level
- Worst-case noise vs probabilistic/stochastic noise (or some hybrid model)
- Many desiderata/decoding models: local decoding/list decoding

The flow of a coding model is as follows.

1. Encoding: info $\to$ codeword
2. Noise: codeword $\to$ noised encoding
3. Decoding (Error Correction): noised encoding $\to$ info

The encoding map is generally injective (you can go from "valid" codeword to info easily).

1.2 One-Bit Erasures

Definition 1.1
A (binary) code is some subset $C \subseteq \{0,1\}^n$. The elements of C are called “codewords”.

Suppose our noise is such that one bit is missing (and the location of the noise is known). So the transformation was:

$$c_1c_2\dots c_n \mapsto c_1c_2\dots c_{i-1}?c_{i+1}\dots c_n$$

we want to find what was in the question mark.

A simple code is repeating the same bit twice:

$$C_{rep} = \{(c_1, \ldots, c_n) \mid c_1 = c_2, c_3 = c_4, \ldots, c_{n-1} = c_n\}$$

Then the question mark is clearly recoverable from the neighbor bit.

A better code that works with this is one with only even parity (called the parity-check code).

$$C_{pc} = \{(c_1, \ldots, c_n) \in \{0,1\}^n \mid c_1 + c_2 + \cdots + c_n \equiv 0 \pmod 2\}$$

Then, the question mark is just the parity of the rest, i.e.

$$? = c_1 + c_2 + \cdots + c_{i-1} + c_{i+1} + \cdots + c_n \bmod 2$$

The second one is better since we can represent more messages, i.e.

$$|C_{pc}| = 2^{n-1} > |C_{rep}| = 2^{\frac{n}{2}}$$

The number of redundant bits in C_{pc} is 1, while there are $n/2$ redundant bits in C_{rep}. Here are the encoding maps:

$$(x_1, \ldots, x_{n-1}) \mapsto (x_1, \ldots, x_{n-1}, \bigoplus_i x_i)$$

$$(x_1, \ldots, x_{n/2}) \mapsto (x_1, x_1, x_2, x_2 \ldots, x_{n/2}, x_{n/2})$$

Fact: C_{pc} is the BEST code to correct 1-bit erasure (it has the least overhead). Proof shell: suppose you had a code with more codewords. This means that there are some codewords that differ by at most 1 bit, but then if you erase that bit the noisy data is ambiguous.

To generalize this setup there are two models. Consider $\alpha < 1$:

1. Hamming model: worst-case, erase any αn out of n bits (zero-error info theory)
2. Shannon model: stochastic, erase each bit independently with probability α

1.3 One-Bit Errors

In this setup, one bit is flipped but we don't know which one.

$$c_1 c_2 \ldots c_n \mapsto c_1 c_2 \ldots c_{i-1} \overline{c_i} c_{i+1} \ldots c_n$$

Now, we can replicate the message 3 times as a code; we can use majority vote to correct one bit. But $\frac{2}{3}$ of the bits are redundant! This means $|C| = 2^{n/3}$. Let's see if we can be smarter.

$$C_{VT} = \{(c_1, c_2, \ldots, c_n) \mid c_1 + 2c_2 + \cdots + nc_n \equiv a \pmod B\}$$

This is a "weighted" parity check. The claim is that if $B > 2n$, then you can correct 1-bit errors. Proof shell: suppose you flip a bit, it will go up or down by at most n and this uniquely identifies which bit got flipped. If B is larger, note that there will be less codewords, so we'd prefer B to be as low as possible.

Since there are some that are bigger than average, there exists an $a \in \{0, \ldots, B-1\}$ such that $|C_{VT}| \geq \frac{2^n}{B} = \frac{2^n}{2n+1}$.

Definition 1.2 (Rate)
An **alphabet** Σ are the characters in the codeword.
The number of redundant bits is $n - \log_{|\Sigma|} |C|$. The **rate** of C is $R(C) = \frac{\log_{|\Sigma|} |C|}{n}$, i.e. the fraction of

non-redundant bits in your code. Note $|C| = |\Sigma|^{Rn}$.

So our code has $\leq \log_2 n + O(1)$ redundant bits, so a rate of nearly 1 asymbtotically.

Theorem 1.1 (Hamming Bound)
If $C \subseteq \{0,1\}^n$ corrects 1-bit flips, then $|C| \leq \frac{2^n}{n+1}$.

Here is a simple proof: consider the hamming ball of distance 1 (i.e. everything produced by 0 or 1 bit flips). These balls cannot overlap (this would imply ambiguous decoding). Thus, by pigeonhole there can only be at most $\frac{2^n}{n+1}$ such balls.

1.4 More than 1-bit flips

To correct two bit errors we can build off our old code:

$$\alpha_1 c_1 + \cdots + \alpha_n c_n \equiv m \pmod{M}$$

where $\pm\alpha_p \pm \alpha_q \neq \pm\alpha_r \pm \alpha_s$ for any 4 tuple. This is sufficient to avoid ambiguity.

We can also add a second check:

$$1^2 c_1 + 2^2 c_2 + \cdots + n^2 c_n \equiv a' \pmod{A}$$

where $A = O(n^2)$. This is a sufficient α because $f(n) = n^2$ is strictly convex. Thus, with the old check together, this makes a code strong enough. Similar to before:

$$|C| \geq \frac{2^n}{A \cdot B} = \Omega\left(\frac{2^n}{n^3}\right)$$

Theorem 1.2 (Generalized Hamming Bound)
If $C \subseteq \{0,1\}^n$ corrects (up to) k-bit flips, then $|C| \leq \frac{2^n}{\sum_{i=0}^{k}\binom{n}{k}} = O\left(\frac{2^n}{n^k}\right)$.

This follows by a similar ball argument, except you consider up to k bit errors.

1.5 Hamming Code

Let's revisit the single-bit flip case. Our "VT Check" was:

$$1c_1 + 2c_2 + \cdots + nc_n \equiv 0 \pmod{B}$$

Let's use linear algebra instead and map ic_i to $\mathbf{v_i}c_i$, thus writing

$$\mathbf{v_1}c_1 + \mathbf{v_2}c_2 + \cdots + \mathbf{v_n}c_n = \mathbf{0}$$

where we take operations mod 2.

To take a (potentially corrupted) $\mathbf{c}$ we can check the following using the parity check matrix on the left.

$$\begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \cdots & \mathbf{v_n} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \mathbf{0}$$

where m, the height of the matrix, is the total amount of bits to represent all the codewords. $m = \lceil \log_2(n+1) \rceil$. This can still correct a one-bit flip!

One can think of any codeword as a set of column linearly dependencies.

Definition 1.3 (Code notation)
A linear $[n, k, d]_q$ code has block size n, dimension k, minimum distance d and alphabet size q. The dimension k is the dimension of C as a subspace.

If we choose $m = 3$, what we constructed above is the $[7, 4, 3]_2$ Hamming code. And the parity check matrix is:

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

To fix an error:

$$\begin{aligned} Hy &= H(c + e_i) \\ &= Hc + He_i \\ &= 0 + (\text{binary representation of } i) \end{aligned}$$

where the quantity Hy is defined as the "syndrome."

We argue this achieves the Hamming bound. Suppose $n = 2^m - 1$ exactly. Note that $\dim(C_{Ham}) = 2^m - 1 - m$ (because the rank of the parity-check matrix is m) so

$$|C_{Ham}| = 2^{2^m - 1 - m} = \frac{2^n}{2^m} = \frac{2^n}{n+1}$$

Let's revisit our two-bit error correction with our Hamming knowledge. Let's write the equations in matrix vector form.

$$\begin{pmatrix} 1 & 2 & \dots & n \\ 1^2 & 2^2 & \dots & n^2 \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

To match the setup we made in Hamming, we would want some check like

$$\begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \dots & \mathbf{v_n} \\ \mathbf{v_1}^2 & \mathbf{v_2}^2 & \dots & \mathbf{v_n}^2 \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

We want a nice canonical way to square vectors. We can identify the m-bit vectors with a field, meaning we can add and multiply them. Formally:

$$\mathbb{F}_2 m \simeq \mathbb{F}_2^m$$

In particular, this identification are polynomials $\mathbb{F}_2$ of degree up to $m - 1$.

Note 1.1
If you have $f, g \in \mathbb{F}_2[x]$, then $(f(x) + g(x))^2 = f(x)^2 + g(x)^2$. (The cross term has coefficient 2, which is 0 mod 2).

This means squaring the vectors doesn't actually get us anything new that we couldn't do with the linear terms. So, let's instead define C_{BCH_2}, with parity check matrix:

$$H = \begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \dots & \mathbf{v_n} \\ \mathbf{v_1}^3 & \mathbf{v_2}^3 & \dots & \mathbf{v_n}^3 \end{pmatrix}$$

Exercise: Show that C_{BCH_2} can correct 2-bit flips, i.e. show that if:

$$\alpha \neq \beta, \gamma \neq \delta$$
$$\alpha + \beta = \gamma + \delta$$
$$\alpha^3 + \beta^3 = \gamma^3 + \delta^3$$

then $\{\alpha, \beta\} = \{\gamma, \delta\}$

2 Lecture 2

2.1 The Basic Definitions

We spam definitions which will help us formalize all the crazy stuff we saw last time.

Definition 2.1 (Distance of a Code)
$C \subseteq \Sigma^n$. Then the **distance** of the code is:

$$d(C) = \min_{c \neq c' \in C} \delta(c, c')$$

This is related to the "packing radius," which is the largest radius τ such that Hamming balls of radius τ around codewords are disjoint. This is exactly $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

The reason distance is so important is that it tells you how robust a code is.

Theorem 2.1 (Distance Property)
Consider a code $C \subseteq \Sigma^n$. The following are equivalent:

- C has minimum distance at least $s + 1$
- C can detect up to s errors
- C can correct up to s erasures

In addition, you can correct up to e errors if and only if the minimum distance is at least $2e + 1$.

This is true pretty intuitive (Hamming balls should not touch) but the proofs are not hard to figure out. What about both erasures and errors together?

Theorem 2.2 (Mixed Distances)
$C \subseteq \Sigma^n$ can correct e errors and s erasures if and only if $2e + s < d(C)$.

However, this distance analysis is a worst-case analysis. However, for typical codewords, you will not go in the direction of exactly another codeword! For stochastic errors, you can correct way more (with high probability).

Even with worst-case analysis, you can "LIST DECODE" many more errors (in fact almost $\approx d(C)$). By "LIST DECODE", we give a list of possible codewords we could've gotten.

2.2 Linear Codes

The Hamming code we saw last time with parity check

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

is an example of a linear code.

Definition 2.2 (Linear Code)
A q-ary linear code is a set $C \subseteq \mathbb{F}_q^n$ is a subspace of $\mathbb{F}_q^n$.

The nice thing about subspaces is you can represent them with little space, despite the fact that the size of C is exponential in the rate, since $|C| = q^{Rn}$. We can represent C by a basis:

$$G = \begin{bmatrix} g_1 & g_2 & \dots & g_k \end{bmatrix} \in \mathbb{F}_q^{n \times k}$$

where $C = \text{range}(G)$ or $C = \{Gx \mid x \in \mathbb{F}_q^k\}$.

Definition 2.3 (Generator Matrix)
Suppose you have the above basis for C, G. Then G is called a generator matrix for C. In turn, $\dim(C) = k$ and the rate is $R = \frac{k}{n}$. Note that G is not unique.

Furthermore, you can find the normal vectors to C, i.e. describe it as the nullspace of some transformation.

Definition 2.4 (Parity Check Matrix)
For a code C, the parity check matrix $H \in \mathbb{F}_q^{(n-k) \times n}$ is a matrix such that for any codeword c, $Hc = 0$. Furthermore, if G is the generator matrix of C, this implies $HG = 0$.

Both of these matrices are defined as full rank.

Hamming distance is translation invariant, and subspace contains the difference between two codewords. So:

Theorem 2.3
For any linear code:

$$d(C) = \min_{c \neq 0} \text{wt}(c)$$

where

$$\text{wt}(c) = |\{i \mid c_i \neq 0\}|$$

We can actually formulate distance in terms of the parity check matrix.

$$\min\{\text{wt}(c) \mid Hc = 0, c \neq 0\}$$

i.e. this is the sparsest linear dependency of any columns of H.

Definition 2.5 (Relative Distance)
The relative distance of a code C of length n is:

$$\delta(C) = \frac{d(C)}{n}$$

In general, there are a few trade-offs:

- k vs d trade-off
- R vs δ trade-off

We are going to see this trade-off in a bound.

Theorem 2.4 (Pigeonhole/Singleton bound)
Consider $C \subseteq [q]^n$. Then:

$$|C| \leq q^{n - d(C) + 1}$$

Proof 2.1
Consider the map:

$$\text{proj} : C \to [q]^{n-d(C)+1}$$
$$(c_1, c_2, \ldots, c_n) \mapsto (c_1, c_2, c_{n-d(C)+1})$$

proj is a one-to-one function, because if two codewords c, c' differ in the last bits cut off, their distance would be less than $d(C)$. Since it is one-to-one, we have $|C| \leq q^{n-d+1}$.

For any linear code, this means: $q^k \leq q^{n-d+1}$, or $d \leq n - k + 1$.

With linear algebra comes duality, and linear codes are no different.

Definition 2.6 (Dual of a Linear Code)
The dual of C is:

$$C^\perp = \{y \in \mathbb{F}_q^n \mid y \cdot c, \forall c \in C\}$$

where the dot product is done mod q. Note that $C^\perp$ is also a linear code (i.e. a subspace).

Another way to think of $C^\perp$ is the space of all parity checks that codewords of C must satisfy.

Note 2.1
We have

- $C^\perp = \text{range}\, H^T$
- the parity check matrix of the dual is G^T
- $(C^\perp)^\perp = C$

In a real vector space, $C \cap C^\perp = \{0\}$. However, over finite fields, it's also possible that $C \subseteq C^\perp$ (a self-orthogonal code) or $C = C^\perp$. For example, $C_{Ham}^\perp \subseteq C_{Ham}$.

It turns out, the dual to C_{Ham} is important. It is called the simplex code $C_{Simplex}$. The generator matrix is nice:

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

which in general is a $(2^r - 1) \times r$ matrix. The code iswhich has rate:

$$\text{Rate}(C_{simplex}) = \frac{r}{2^r - 1} \to 0, \text{Rate}(C_{Ham}) = 1 - \frac{r}{2^r - 1} \to 1$$

But the distance is much better!

Fix an $x \in \mathbb{F}_2^n$.

$$\mathbb{P}\left[z \in \mathbb{F}_2^n\right][x \cdot z = 1] = \frac{1}{2}$$

So the hamming weight will be at least half n, i.e.

$$d(C_{simplex}) = 2^{r-1}$$

So, we now ask, can we have both R and δ bounded away from zero?

3 Lecture 3

3.1 Asymptotically Good Codes

As we have seen previously, the Hamming code is a $[2^r - 1, 2^r - r - 1, 3]_2$ code. To generalize the code to any length finite field, we want to do the same thing (having the columns of the parity check count up) and remove linear dependencies (we don't want any two columns to be linearly dependent, because that would imply the minimum distance is 2).

We also saw its dual is a $[2^r - 1, r, 2^{r-1}]_2$ code. The first two come from dual properties, the third we proved last time. These are two opposite ends of the spectrum. The Hamming code is optimal for distance 3 (by the Hamming bound). Simplex code is also optimally-sized for its distance.

Theorem 3.1
If $C \subseteq \{0,1\}^n$ is a code of distance $d > \frac{n}{2}$ then

$$|C| \leq \frac{d}{d - \frac{n}{2}}$$

Proof shell: Map a codeword $c = (c_1, \ldots, c_n) \mapsto v_c = \frac{1}{\sqrt{n}}((-1)^{c_1}, \ldots, (-1)^{c_n})$. Then for any two codewords c_1, c_2, we have:

$$v_{c_1} \cdot v_{c_2} = 1 - \frac{2\Delta(c_1, c_2)}{n}$$

If $d > \frac{n}{2}$, then, $v_{c_1} \cdot v_{c_2} < 0$.

Lemma 1 *If $v_1, \ldots, v_m \in \mathbb{R}^n$, $\|v_i\| = 1$, such that $v_i \cdot v_j \leq -\alpha \; \forall 1 \leq i \leq j \leq m$, then $m \leq \frac{1}{\alpha} + 1$.* □

From here you can apply this theorem with the correct α.
Furthermore, the bound for a q-ary code is of distance $d > \left(1 - \frac{1}{q}\right)n$,

$$|C| \leq \frac{d}{d - \left(\frac{q-1}{q}\right)n}$$

As an aside, if you have $v_i \cdot v_j \leq 0$, then $m \leq 2n$ and $v_i \cdot v_j \leq \epsilon$ is exponentially many.

For the simplex code, $|C_{simplex}| = 2^r$ and $d = 2^{r-1}$ and $n = 2^r - 1$, so

$$\frac{2d}{2d - n} = 2^r = |C_{simplex}|$$

So our goal is:

Definition 3.1 (Asymptotically Good Family)
An asymptotically good family of codes is an infinite family of codes over a fixed alphabet $[q]$

$$C_1, C_2, \ldots$$

Where length n_i of C_i goes to ∞ as $i \to \infty$, where

$$R(C_i) \geq R_0, \delta(C_i) \geq \delta_0$$

for some $R_0, \delta_0 > 0$ which are absolute constants.

3.2 Gilbert-Varshamov Bound

How do we know such asymptotically codes exist? By using bounds.

Theorem 3.2 (Gilbert-Varshamov Bound)
For all $q, n, d \leq n$ there exists a code $C \subseteq [q]^n$ with $d(C) \geq d$ and

$$|C| \geq \frac{q^n}{\sum_{i=1}^{d-1} \binom{n}{i}(q-1)^i}$$

Proof 3.1
The denominator here is $|B_q(0, d-1)| = \{x \in [q]^n \mid \text{wt}(x) \leq d-1\}$, i.e. the volume of the Hamming ball around 0 of size $d-1$. Greedily pick codewords such that no codeword is not within $d-1$ of any previously chosen ones. Stop when you can't, the balls must cover the space. The size union of the balls is no more than the sum of their sizes.

$$|C| \cdot |B_q(0, d-1)| \geq q^n$$

In fact you can even modify this argument so that it gives you a linear code (over $\mathbb{F}_q$). Fill in columns of a $m \times n$ parity check matrix "greedily" (such that no $d-1$ columns are linearly dependent). For the last column, the amount of linear combinations are $\sum_{i=0}^{d-2} \binom{n-1}{i}(q-1)^i$ (this basically proves the claim). If this is less than q^m, then we'd never run out vectors. If it's MUCH BIGGER, then a random parity check matrix (RLC - random linear code) works with high probability.

We can also define an RLC via a $n \times k$ generator matrix. This works if:

$$q^k \ll \frac{q^n}{\sum_{i=1}^{d-1} \binom{n}{i}(q-1)^i}$$

Let us think about the asymptotic form, as n is large. For binary, fix relative distance $\delta \in [0, \frac{1}{2}]$. Note that binomial coefficients grow according to:

$$\frac{2^{h(\delta)n}}{\text{poly}(n)} \leq \binom{n}{\delta n} \leq 2^{h(\delta)n}$$

where the binary entropy is:

$$h(x) = x \log \frac{1}{x} + (1-x) \log \frac{1}{1-x}$$

Furthermore:

$$\sum_{i=0}^{\delta n} \binom{n}{i} \leq 2^{h(\delta)n}$$

By the GV bound:

$$\begin{aligned} |C| &\geq \frac{2^n}{\sum_{i=0}^{d-1} \binom{n}{i}} \\ &\geq \frac{2^n}{\sum_{i=0}^{\delta n} 2^{h(\delta)n}} \\ |C| &\geq 2^{(1-h(\delta))n} \end{aligned}$$

Thus,

Theorem 3.3 (Asymptotic Form of GV Bound)
There exists a binary linear code C with $\delta(C) = \delta$ and rate $R(C) \geq R_{GV}(\delta) = 1 - h(\delta)$.

So the rate graph over delta is the upside-down entropy plot (convex instead of concave).

In the q-ary case, we need to change the interval to $\delta \in [0, 1 - \frac{1}{q}]$, and using entropy function

$$h_q(x) = x \log_q(q-1) + x \log_q \frac{1}{x} + x \log_q \frac{1}{1-x}$$

which is the entropy of a random variable which is 0 with probability x and any value 1 to q with uniform other probability. If q is large, we approach the singleton bound.

For binary, no one knows if the asymptotic GV bound can be beaten (even existentially). It is known that for $q \geq 49$ that you can beat the bound. Also note that any algorithm trying to achieve the GV bound is either:

- Exponential in runtime (Greedy)
- Efficient to sample (RLC), but no known way to certify in polynomial time and no known way to decode.

In fact the lack of decoding is a popular hardness assumption in cryptography. This is called "Learning parity with noise."

Let's look at the regimes at the ends of the GV bound. If $\delta \to 0$, then

$$R_{GV}(\delta) = 1 - O\left(\delta \log\left(\frac{1}{\delta}\right)\right)$$

and if $\delta = \frac{1}{2} - \epsilon$, then

$$R_{GV}\left(\frac{1}{2} - \epsilon\right) = \Theta(\epsilon^2)$$

However, constructively, you can get from explicity construction: If $\delta \to 0$, then

$$R_{GV}(\delta) = 1 - O\left(\delta \cdot \text{polylog}\left(\frac{1}{\delta}\right)\right)$$

and if $\delta = \frac{1}{2} - \epsilon$, then

$$R_{GV}\left(\frac{1}{2} - \epsilon\right) = \Omega\left(\epsilon^{2+o(1)}\right)$$

We will achieve the SOTA with expander codes, expander walks, and pseudorandomness.