Lecture 1: Error Correcting Codes

Anup Rao

September 25, 2019

Error correcting codes are mathematical objects that play a fundamental role in technology. In a nutshell, they allow for the reliable storage and transmission of information by giving us the capability to recover from physical corruptions. Beyond this obvious application, codes are important conceptual objects that have been used all over theoretical computer science. The study of error correcting codes has led to beautiful mathematics, and there are still many open questions that remain unanswered.

You are almost certainly carrying an error correcting code in your pocket. Your phone uses one to manage its storage, and another to handle communication over the airwaves.

The aim of this class is to give a brief introduction to the theory of error correcting codes — just enough so that we can make the leap to discussing more advanced research topics near the end of the term.

What I like about the theory of error correcting codes: their study lies in the intersection of combinatorics, probability, geometry and algebra. So, if you start thinking about codes, you end up thinking about all the things I like thinking about!

Defining codes

A code is a subset of strings that are far apart. Let Σ be a finite set. Given two strings $x, y \in \Sigma^n$, define the Hamming distance

$$\Delta(x,y) = |\{i \in [n] : x_i \neq y_i\}|.$$

As we shall see, it is easiest to construct codes over large alphabets, but codes over a small alphabet are the most useful.

The Hamming distance is the number of coordinates where x, y are not the same. A *code with distance d* is a subset $C \subseteq \Sigma^n$, such that for any two distinct elements $x, y \in C$, we have $\Delta(x,y) \geq d$. The parameter n is called the *block-length* of the code, and Σ is called the alphabet of the code. We usually write $q = |\Sigma|$ to denote the size of the alphabet. Of special interest is the case $\Sigma = \{0,1\}$. In this case, we call the code a *binary* code.

So far, you see that the definitions are very combinatorial. The plot will thicken when we begin to focus on algebraic definitions that guarantee good distance.

Why are codes useful? They are designed to tolerate corruptions. Suppose $C = \{c_1, \ldots, c_\ell\} \subseteq \{0,1\}^n$ is a set of binary strings of length n. Suppose you want to encode an integer in $[\ell]$. The most naive way to do this would be write down the integer in binary. But then if a single written bit is later corrupted, you would be unable to recover the integer that was stored. Instead, we should encode the integer i as c_i. Now, the point is that if at most $(d-1)/2$ symbols are corrupted, you can still recover c_i, and so i.

What if the symbols are erased rather than corrupted? Suppose less than d of the symbols of the codeword are replaced by a *. Then it is easy to see that we can recover the original codeword — so a code of distance d can tolerate upto $d/2$ corruptions, and d erasures.

Some more definitions are useful here. Given $x \in \Sigma^n$, write $B(x,d)$ to denote the ball of strings that are within distance d of x:

$$B(x,d) = \{y \in \Sigma^n : \Delta(x,y) \leq d\}.$$

We write $\mathsf{vol}(d)$ to denote the size $|B(x,d)|$ of this ball (note that the volume does not depend on x). We have

$$\mathsf{vol}(d) = \sum_{i=0}^{d} \binom{n}{i} (q-1)^i.$$

Then another way to assert that the code has distance d is to write that for all distinct $x, y \in C$, $B(x,(d-1)/2)$ and $B(y,(d-1)/2)$ are disjoint.

For a code to actually be useful, we need to have algorithms that allow for efficiently encoding messages as a codeword, and for efficiently decoding messages. So, error correcting codes are often defined with a more complicated definition, as follows. We would like to have functions

$$E : \Sigma^k \to \Sigma^n,$$

and

$$D : \Sigma^n \to \Sigma^k,$$

such that for all $u \in \Sigma^k$, $w \in \Sigma^n$ with $\Delta(E(u), w) \leq e$, we have $D(w) = u$. In words, if we encode u using E, and then at most e symbols of $E(u)$ are corrupted, the decoding is guaranteed to recover u.

For E, D to be practical, they should also be efficiently computable. This is an issue we deal with in later lectures.

For the above scheme to work, it is necessary that $E(\Sigma^k)$ should be a code of distance bigger than $2e$. Otherwise, if there are two strings at distance at most $2e$ in the image, we can find a string in between these two that cannot be properly decoded.

Usually, we think of the alphabet size $|\Sigma| = q$ as being fixed, and n as growing. We think of the code as a family of sets $C_n \subseteq \Sigma^n$, one for each n. Then $R = k/n = (\log_q |C|)/n$ is called the *rate* of the code. $\delta = d/n$ is called the *relative distance* of the code. We would like to find codes that simultaneously maximize the rate and the relative distance. The challenge is that these two parameters are inherently opposed to each other — the more codewords you pack into the space, the harder it is to ensure that they are far apart.

Of course there are many other notions of valid decoding that make sense here. We could ask for the decoding to work only when the errors are random, or to be able to recover some part of the message rather than the whole message, just to give examples. We shall issue some of these issues later.

Some (not so great) Codes

ENOUGH TALK, let us give some examples of codes.

Repetition code Consider the code that encodes a message by repeating each symbol d times. So, when $d = 2$, we encode 12112 by 1122111122. This code has distance d. The rate of the code is $1/d$. This is a terrible code.

Parity Check Code Consider the code obtained by encoding a message in $\mathbb{F}_2^k$ by encoding $(x_1, \ldots, x_k)$ with $(x_1, \ldots, x_k, x_1 + x_2 + \ldots x_k)$. Another way to describe the codewords is as the set of strings $y \in \mathbb{F}_2^n$ with $y_1 + \cdots + y_n = 0$. The distance is $d = 2$. So, we have $k + d = n + 1$. Unfortunately d is quite small. We would really like to have d be a constant fraction of n.

Singleton bound

Clearly, for large n, we cannot have a code rate 1 and relative distance 1 — if the rate is 1, all strings must be included in the code, but then the relative distance can be at most $1/n$ for the worst pairs.

Let us investigate the tradeoffs between the rate and relative distance. The pigeonhole principle already implies the following basic bound:

Theorem 1 (Singleton Bound)**.** *If $C \subseteq \Sigma^n$ be a code of size $|C| = |\Sigma|^k$, and distance d, then we have $k + d \leq n + 1$. In other words, the rate and relative distance must satisfy $R + \delta \leq 1 + 1/n$.*

Proof. Suppose this is not true, and that $k > n - d + 1$. Then by the pigeonhole principle, there are two codewords $x \neq y$ that are equal on the first $n - d + 1$ coordinates. These two codewords have distance at most $d - 1$, contradicting the distance of the code. □

There are q^k pigeons, namely the codewords, and $q^{n-d+1} < q^k$ holes, namely the values that the codewords take in the first $n - d + 1$ coordinates. Two pigeons must get mapped to the same hole.

The singleton bound gives us a basic limit about how much redundancy we need to add to our code to tolerate errors. In fact, it is tight — no better bound is possible if we do not take the alphabet size into account. This is because there is an explicit code, the Reed-Solomon code, which has alphabet size n, and $k + d = n + 1$. However, when the alphabet size is smaller, the singleton bound is not so tight.

Estimating Volume

We are working with Hamming space, so key to understanding the tradeoff between the parameters of a code is understanding the quantity $\mathsf{vol}(d)$. A basic estimate is given by using the following formula:

$$h_q(\epsilon) = (1 - \epsilon) \log_q \left(\frac{1}{1 - \epsilon} \right) + \epsilon \log_q \left(\frac{q - 1}{\epsilon} \right).$$

The quantity $h_q(\epsilon)$ is proportional to the maximum entropy of a random variable taking q values, subject to the constraint that the

random variable must be equal to the first value with probability at least $1-\epsilon$. It turns out that when $\epsilon \leq 1-1/q$, $h_q(\epsilon) \geq \epsilon$ and $\lim_{q\to\infty} h_q(\epsilon) = \epsilon$.

When $d \leq n(1-1/q)$, we shall prove the estimates

What happens when $d \geq n(1-1/q)$?

$$q^{n\cdot h_q(d/n)-O(\log_q n)} \leq \mathsf{vol}(d) \leq q^{n\cdot h_q(d/n)}. \tag{1}$$

We shall prove these bounds on the volume in a later lecture. For now, let us use it to give more bounds on codes.

Hamming bound

OUR ESTIMATE FOR the volume of balls in the Hamming metric immediately allows us to prove another lower bound:

Theorem 2 (Hamming Bound)**.** *If $C \subseteq \Sigma^n$ is a code of distance d, then we have $|C| \leq |\Sigma|^n/\mathsf{vol}((d-1)/2)$.*

Proof. If the distance of the code is d, then the balls of radius $(d-1)/2$ centered around the codewords must all be disjoint. This gives the bound. □

In terms of the rate R and relative distance δ, using (1), the above bound says that

$$R + h_q(\delta/2) \leq 1 + O(\log_q(n)/n).$$