# Introduction to Coding Theory

Whenever data is transmitted we run into the following practical problem - when a 0 is sent, your device usually receives a 0, but occasionally, *noise* on the channel, perhaps in the form of atmospheric disturbances or hardware malfunctions, causes the 0 to be received as a 1. We would like to develop ways to combat these errors that can occur during data transmission.

*Error-control codes* are used to detect and correct errors that occur when data are transmitted across some noisy channel or stored on some medium. When photographs are transmitted to Earth from deep space, error-control codes are used to guard against the noise caused by lightning and other atmospheric interruptions. Compact discs (CDs) use error-control codes so that a CD player can read data from a CD even if it has been corrupted by noise in the form of imperfections on the CD.

Correcting errors is even more important when transmitting data that have been encrypted for security. In a secure cryptographic system, changing one bit in the ciphertext propagates many changes in the decrypted plaintext. Therefore, it is of utmost importance to detect and correct errors that occur when transmitting enciphered data.

The study of error-control codes is called *coding theory*. This area of discrete applied mathematics includes the study and discovery of various coding schemes that are used to increase the number of errors that can be corrected during data transmission. Coding theory emerged following the publication of Claude Shannon's seminal 1948 paper, “A mathematical theory of communication"

## 1. Introductory Examples

To get the ball rolling, we begin with some examples of error-control codes. Our first example is an error-detecting, as opposed to error-correcting, code. A code that can only *detect* up to *t* errors within a codeword can be used to determine whether *t* or fewer errors occurred during the transmission of the codeword, but it cannot tell us exactly what error(s) occurred nor can it fix the error(s). A code that can *correct* up to *t* errors can be used to actually correct up to *t* errors that occur during the transmission of a codeword. This means that the code detects that errors occurred, figures out what errors occurred in which positions, and then corrects the errors.

**Example 1.1.** (Parity Check) For a given binary string *m* of length *k*, a *parity check bit* is appended to obtain a codeword *c* of length *k + 1*. The value of the *parity check bit* is chosen so that the total count of occurrences of 1s in *c* is an even number. Therefore, if number of bits in *m* is odd, the *parity check bit* value is set to 1, and otherwise 0.

**Question 1.1.** How will we detect a single error in a codeword with the above scheme? What will happen if two errors occur?

**Example 1.2.** (*The Repetition Code*) This example is a simple error-correcting code. Suppose we have only two messages - 0 or 1. If we simply send one bit, then there is a chance that noise will corrupt that bit and an unintended message will be received. A simple alternative is to use a *repetition code*. Instead of sending a single bit, we send 11111 to represent 1 (or yes), and 00000 to represent 0 (or no). Since the codewords consist of 0s and 1s, this is called a *binary* code. We say that the code's alphabet is the set *{*0*,* 1*}*, with all arithmetic done modulo 2

Alternatively, we can say that this code alphabet is the *finite field* with two elements, *GF* (2). Finite fields will be formally defined in section 2. Under certain reasonable assumptions about the channel in use, the receiver decodes a received 5-tuple using a "majority vote": The received 5-tuple is decoded as the bit that occurs most frequently. This way, if zero, one, or two errors occur, we will still decode the received 5-tuple as the intended message. In other words, the receiver decodes a received 5-tuple as the "closest" codeword. This is an example of *nearest neighbor decoding*, formally defined in definition 6. This can be extended to sending n-bit message by repeating the entire message.

**Question 1.2.** How many errors can a binary repetition code of length 10 correct? How many errors can it detect?

**Question 1.3.** How many repetitions are needed to correct r errors?

**Example 1.3.** (*The 2-D Parity Check Code*) It is possible to extend the idea of a parity check bit which detects a single error to *correct* single errors by first arranging a message into a matrix, or two-dimensional array. For example, suppose we want to encode the 20-bit message 10011011001100101011. First, we arrange this message into a 4x5 matrix M

$$M = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \qquad C = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & \spadesuit \end{bmatrix}.$$

Then, we look at the *parity* of each row Then, we calculate and append a *parity check bit* to the end of each row. After appending a parity check bit to each row, we have four rows of length 6. We then repeat the process by calculating and appending parity check bits along along each of the original columns, giving C in the above diagram.

The entry in the lower right hand corner marked by ♠ is then calculated as the parity check bit of the new row (in other words, the ♠ entry is calculated as the parity of the parity bits that were calculated along the original columns). In this case it will be 1.

We then send this stream of 30 bits across our noisy channel. To see how this extended matrix can correct a single error, suppose that the receiver arranges the received 30 bits into a 5x6 matrix and obtains R.

$$R = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

The receiver then checks the parity of each row and column: If all parities check to be even, then we assume there are no errors. (Is this a fair assumption?) However, in this example, we see that the parities of the third row and fourth column are odd. Therefore, we conclude that there is an error in the (3*;* 4) position, correct the error, and proceed to read the message

## 2. Formalization of the Basics

**Definition 2.1** Let A = {$a_1$, . . . , $a_q$} be an alphabet; we call the $a_i$ values symbols. A block code C of length n over A is a subset of $A^n$. A vector c ∈ C is called a codeword. The number of elements in C, denoted |C|, is called the size of the code. A code of length n and size M is called an (n, M)-code.

A code over A = {0, 1} is called a binary code and a code over A = {0, 1, 2} is called a ternary code.

**Remark** We will almost exclusively talk about "sending a codeword c" and then finding the codeword c that was originally sent given a vector x obtained by introducing errors into c. This may seem strange at first since we ignore the problem of mapping a message m into a codeword c and then finding m again from c. As we will see later, this is typically not a problem (especially for linear codes) and thus the mapping of original messages to codewords and back is not a concern.

The rate of a code is a measure of its efficiency. Formally:

**Definition 2.2** Let C be an (n, M )-code over an alphabet of size q. Then, the rate of C is defined by

$$\text{rate}(C) = \frac{\log_q M}{n}.$$

Observe that it is possible to specify M messages using $\log_q$ M symbols when there is no redundancy. Thus, the longer that n is, the more wasteful the code (in principal, $\log_q$ M symbols suffice and the n − $\log_q$ M additional symbols are redundancy). Observe that the code C = $A^n$ has an optimal rate of 1, but cannot detect or correct any errors.

**Question 2.1:** Give an expression for the rate of repetition code. What does the rate tend to if number of errors to be corrected increase?

We make a few simple assumptions:

1. The errors occur independently for each symbol with a probability < 1/2 (thus it is more likely to have less errors than more errors)
2. We assume an upper bound on the number of errors that occur (if we are wrong, then an incorrect message may be received)

**Definition 2.3** Hamming distance Let x = $x_1, \ldots, x_n$ and y = $y_1, \ldots, y_n$. Then, for every i *define*

$$d(x_i, y_i) = \begin{cases} 1 & x_i \neq y_i \\ 0 & x_i = y_i \end{cases}$$

*and define*

$$d(x, y) = \sum_{i=1}^{n} d(x_i, y_i).$$

We stress that the Hamming distance is not dependent on the actual values of $x_i$ and $y_i$ but only if they are equal to each other or not equal.

**Theorem 1** The function d is a metric. That is, for every x, y, z ∈ An 1. 0 ≤ d(x,y) ≤ n

1. d(x, y) = 0 if and only if x = y
2. d(x, y) = d(y, x)
3. d(x, z) ≤ d(x, y) + d(y, z) (triangle inequality)

**Definition 2.4** Let C be a code of length n over an alphabet A. The nearest neighbor decoding rule states that every x ∈ A is decoded to $c_x$ ∈ C that is closest to x. That is, D(x) = $c_x$ where $c_x$ is such that {d(x, c)}. If there exist more than one c at this minimal distance, then D returns ⊥.

Code distance and error detection and correction. Intuitively, a code is better if all the codewords are far apart. We formalize this notion here.

**Definition 2.5** Let C be a code. The distance of the code, denoted d(C), is defined by:

d(C) = min { d($c_1$, $c_2$) | $c_1$, $c_2$ ∈ C, $c_1 \neq c_2$ }

An (n, M)-code of distance d is called an (n, M, d)-code. The values n, M, d are called the parameters of the code.

Restating what we have discussed above, the aim of coding theory is to construct a code with a short n, and large M and d; equivalently, the aim is to construct a code with a rate that is as close to 1 as possible and with d as large as possible. We now show a connection between the distance of a code and the possibility of detecting and correcting errors.

**Definition 2.6** Let C be a code of length n over alphabet A

- C detects u errors if for every codeword c $\in$ C and every x $\in$ $A^n$ with x $\neq$ c, it holds that if d(x, c) $\leq$ u then x $\notin$ C.
- C corrects v errors if for every codeword c $\in$ C and every x $\in$ $A^n$ it holds that if d(x, c) $\leq$ v then nearest neighbor decoding of x outputs c.

**Theorem 2**
- A code C detects u errors if and only if d(C) > u.
- A code C corrects v errors if and only if d(C) $\geq$ 2v + 1.

## 3. Linear Codes

A general code might have no structure and not admit any representation other than listing the entire codebook. We now focus on an important subclass of codes with additional structure called linear codes. Many of the important and widely used codes are linear.

Linear codes are defined over alphabets $\Sigma$ which are finite fields. We will denote by $F_q$ the finite field of q elements where q is a prime power.

Recall from linear algebra that:
1. A subspace has the 0 element and it is closed under scalar multiplication (with respect to the the given field) and addition.
2. A finite field is a set closed under multiplication and addition, having both additive and multiplicative identities and inverses, and where multiplication distributes over addition. All finite fields are isomorphic to the set of integers mod q, where q is a prime power.
3. A set of vectors (nx1 matrices) spans a space if for each element in the space there exists a linear combination of the vectors of the set which is equal to that element.

**Definition 3.1** (Linear code) If $\Sigma$ is a field and C $\subset$ $\Sigma^n$ is a subspace of $\Sigma^n$ then C is said to be a linear code

**Example 3.1** The binary repetition code of length 5 is a binary linear code. You can quickly confirm that it satisfies the requirement that the sum of any two codewords is another codeword (closure under addition): 00000 + 00000 = 00000 $\in$ C , 00000 + 11111 = 11111 $\in$ C , and 11111 + 11111 = 00000 $\in$ C

**Question 3.1** Is { 000 , 100 , 001 } a binary linear code? How about { 100 , 001 , 101 } ?

**Definition 3.2** (Generator matrix and encoding) Let C $\subseteq$ $(F_q)^n$ be a linear code of dimension k . A matrix G $\in$ $(F_q)^{n \times k}$ is said to be a generator matrix for C if its k columns span C . The generator matrix G provides a way to encode a message x $\in$ F k q (thought of as a column vector) as the codeword *Gx* $\in$ C $\subseteq$ $(F_q)^n$. Thus a linear code has an encoding map E :$(F_q)^k \longrightarrow (F_q)^n$ which is a linear transformation $x \longrightarrow Gx$ .

**Example 3.2** Suppose we need a [3,2] binary linear code C , that is, we need a 2-dimensional vector subspace of V (3 , 2), the vector space of binary 3-tuples. In order to define a 2-dimensional vector subspace of V (3 , 2), we need 2 linearly independent basis vectors. Arbitrarily, let's choose these vectors to be 011 and 110. Let C be spanned by these vectors. Then the generator matrix is $G^T$ where G is

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

Note: the generator matrix is the **transpose** of the above matrix.
Taking all the linear combinations of the rows in G , we generate the code C = { 110 , 011 , 101 , 000 }.

**Notation** A q-ary linear code of block length n and dimension k will be referred to as an $[n, k]_q$ code. Further, it the code has minimum distance d , it will be referred to as an $[n, k, d]_q$ code.

**Theorem 3** The minimum distance, d(C), of a linear code C is equal to w∗(C), the weight of the lowest-weight nonzero codeword.

**Proof.** There exist codewords x and y in C such that d(C) = d(x, y). By the definition of Hamming distance, we can rewrite this as d(C) = w(x − y). Note that x − y is a codeword in C by the linearity of C . Therefore, since w ∗(C) is the weight of the lowest weight codeword, we have w∗(C) ≤ w(x — y) = d(C) . 17 On the other hand, there exists some codeword c ∈ C such that w∗(C) = w(c). By the definition of the weight of a codeword, we can write w(c) = d(c, 0). Since c and 0 are both codewords, the distance between them must be greater than or equal to the minimum distance of the code: d(c, 0) ≥ d(C). Stringing together these (in)equalities, shows that w∗(C) ≥ d(C)

Since we have now shown that both d(C) ≥ w∗(C) and d(C) ≤ w ∗(C), we conclude that d(C) = w∗(C)

The above theorem greatly facilitates finding the minimum distance of a linear code. Instead of looking at the distances between all possible pairs of codewords, we need only look at the weight of each codeword. Notice that the proof does not restrict the linear codes to be binary.

## 4. And a Dash of Cyclic Codes

**Definition 4.1** An *[n, k, d]* linear code C is cyclic if whenever ($c_0$, $c_1$, …., $c_{n-1}$) is a codeword in C, then ($c_{n-1}$, c0, …., $c_{n-2}$) is also a codeword in C.

**Example 4.1** The binary code C = {000, 110, 011, 101} is a cyclic code.

Cyclic codes can be implemented efficiently via simple hardware devices called shift registers. This is of great interest in applications involving fiber optics, where high-speed data rates are possible.

**Definition 4.2** Code Polynomials. When working with cyclic codes, it is convenient to convert codeword vectors $c = (c_0, c_1, \ldots, c_{n-1})$ of length n into code polynomials $c(x) = c_0 + c_1x + \ldots + c_{n-1}x^{n-1}$ of degree less than n. Note that the left-most bit in a codeword is associated with the constant term in the code polynomial.

**Question 4.1** What is the corresponding code polynomial for the codeword (1, 0, 1, 1)?

**Question 4.2** For a codeword of length 5, what is the maximum degree of the associated code polynomial?

From now on, we will use the terms "codeword" and "code polynomial" interchangeably. This abuse reflects the fact that you should be thinking about codewords and code polynomials as representing the same thing.

The power of using code polynomials will be apparent when we develop some additional structure. We will parallel the development of $Z_p$, the integers modulo p, from the regular integers Z. Recall that with the integers modulo p, we constantly wrap around according to the modulus p. In other words, the largest number we can use is p - 1, since as soon as we hit p or larger, we are reduced modulo p. We think of p as being equivalent to 0, so all multiples of p are "ignored."

Now consider the code polynomials $\{0, 1 + x, x + x^2, 1 + x^2\}$ corresponding to the code C = {000, 110, 011, 101}. Notice that the highest power appearing among the code polynomials is $x^2$, since a higher power term would indicate a codeword with length greater than 3. Similarly to working with integers modulo a specific prime integer p, we can work with polynomials modulo a specific polynomial, for example $p(x) = x^3 - 1$. In this situation, we think of $x^3 - 1$ being equivalent to 0, or in other words, $x^3$ is equivalent to 1. Let's see what this means with an example.

**Example 4.2** Consider the code polynomial $c(x) = 1+x^2$ corresponding to the codeword c = 101 from C = {000, 110, 011, 101}. Let's multiply c(x) by x to obtain $c(x)x = c_0(x) = x + x^3$: However, if we are working modulo $p(x) = x^3 - 1$, then $x^3$ is equivalent to 1. So, $c_0(x)$ is equivalent to x + 1 modulo p(x). Notice that rearranging this new polynomial with terms from lowest to highest powers gives 1 + x, which is the code polynomial for the code word 110 $\in$ C. Notice furthermore that 110 is the right cyclic shift of 101. Coincidence? Let's try this again. This time, start with the code polynomial $d(x) = x + x^2$ corresponding to the codeword 011 in C. Multiplying d(x) by x gives $d_0(x) = x^2 + x^3$. Taking this result modulo $x^3 - 1$ gives $x^2 + 1$, which (upon rearranging terms from lowest to highest powers) corresponds to the codeword 101 $\in$ C and is the right cyclic shift of 011. Interesting!

The example above suggests a true fact about cyclic codes. In a cyclic code C of length n, the product *xc(x)* modulo $x^n - 1$ produces another code polynomial in C, namely the right cyclic shift of c(x). More precisely, when working with code polynomials of degree less than n corresponding to codewords of length n, by working modulo $x^n - 1$, we can achieve a right cyclic shift of a codeword by multiplying the associated code polynomial by x: Consider the code polynomial $c(x) = c_0 + c_1x + \ldots + c_{n-1}x^{n-1}$. Multiplying c(x) by x modulo $x^n$ - 1 gives $c_0(x) = c_0x + c_1x^2 + \ldots.+ c_{n-1}x^n \equiv c(x) = c_{n-1} + c_0x + \ldots.+ c_{n-2}x^{n-1}$ modulo $x^n$ - 1. The codeword associated with $c_0(x)$ is $(c_{n-1}, c_0, \ldots., c_{n-2})$, which is clearly the right cyclic shift of the codeword associated with c(x).

**Question 4.3** Show that multiplying $c(x) = 1+x^2$ by $x^2$ corresponds to doubly-shifting the associated codeword.

## References:

1. http://www.cs.cmu.edu/~venkatg/teaching/codingtheory/notes/notes1.pdf
2. http://www.math.niu.edu/~beachy/courses/523/coding_theory.pdf
3. http://u.cs.biu.ac.il/~lindell/89-662/coding_theory-lecture-notes.pdf

## Mechanical Questions: *(1 point each)*

Q 1. Given a binary code with length 10, if probability of a bit arriving correctly is 0.7, what is the probability that it will be received with two errors?
Q 2. Give the rate of 2-D parity check code
Q 3. The **transpose** of the following matrix is not a generator matrix for any linear code: why?

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

Q4. Transpose of the following matrix generates a code. Determine the code

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

## Some thinking required:

Q 5. *(2 points)* Show that the q-ary repetition code C is a linear code, where q is prime.

Q 6. *(2 points)* Given a p-length message, where p is prime, you insist on using the 2-D parity check code as illustrated in example 1.3 to encode your message. You choose to make your matrix with m row and n columns. Since *mn* cannot equal *p*, you can pad the message with *mn - p* zeroes so it fits nicely into your matrix. You choose m and n optimally to minimize the number of parity bits you will have to add. What is the minimum possible number of parity bits you could add?

Q 7. *(2 points)* In the 2-D parity check, how would we correct an error in one of the parity bits?

Q 8. (5 points) Show that A linear q-ary code of length n is cyclic if and only if *C* satisfies the following two conditions:

1. If *a(x)* and *b(x)* are code polynomials in *C*, then *a(x) - b(x)* ∈ *C*
2. If *a(x)* is a code polynomial in *C* and *r(x)* is any polynomial of degree less than n, then *r(x)a(x)* ∈ *C*

Q 9. (4 points) Prove that in a binary symmetric channel* with p < ½, maximum likelihood decoding* is equivalent to nearest neighbor decoding.

*A symmetric channel is one in which every symbol has same probability of error and if there's an error, the probability of it being changed to any other symbol is equally likely. All errors are independent of each other.

A *binary symmetric channel* has two probabilities:

$$\begin{aligned}\Pr[1 \text{ received} \mid 0 \text{ was sent}] &= \Pr[0 \text{ received} \mid 1 \text{ was sent}] = p\\ \Pr[1 \text{ received} \mid 1 \text{ was sent}] &= \Pr[0 \text{ received} \mid 0 \text{ was sent}] = 1 - p\end{aligned}$$

The probability $p$ is called the crossover probability.

*Maximum likelihood decoding is -

**Definition 1.12** *Let $C$ be a code of length $n$ over an alphabet $A$. The* maximum likelihood decoding *rule states that every $x \in A^n$ is decoded to $c_x \in C$ when*

$$\Pr[x \text{ received} \mid c_x \text{ was sent}] = \max_{c \in C} \Pr[x \text{ received} \mid c \text{ was sent}]$$

Q10. (6 points) Sometimes we can start with a known binary linear code and append an overall parity check digit to increase the minimum distance of a code. Suppose C is a linear (n, M, d) code. Then we can construct a code C′, called the extended code of C , by appending a parity check bit to each codeword x ∈ C to obtain a codeword x′ ∈ C′ This ensures that every codeword in the extended code has even Hamming weight.

Prove that C′ is a linear code, and that the minimum distance of C′ is d + 1 if d = d(C) is odd, and otherwise it is d.