Lecture notes on Information Theory
ENS Lyon
(Under construction)

Omar Fawzi

September 13, 2017

Chapter 1

Introduction

The modern theory of information comes from communications theory. What is called information theory nowadays was pioneered by Shannon in 1948 in his landmark paper [2]. The fundamental problem that started the area is the problem of reliable communication over an unreliable channel. In Shannon's words, "The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point." This is a very broad class of problems. A point can be in space or in time.

1. Point 1: Computer memory at time t_1, Point 2: Computer memory at time t_2
2. Point 1: DNA of a parent cell, Point 2: DNA of a daughter cell
3. Point 1: Your computer, Point 2: Another computer in the network
4. Point 1: IT notebook in 2016, Point 2: IT notebook in 2026

Between these two points: there is a communication channel which might add some noise to the signal. In the examples shown above, this channel might be modelling the passage of time or the errors introduced by the wires in the network. Our objective would be that the message selected at Point 1 is the same as the message obtained at Point 2. From an engineering point of view, we may solve this problem in two ways.

1. Improve the communication channel, to reduce the errors
2. Accept the errors as given, and build a system on top of the channel that allows one to correct for errors.

We will take this second approach here, which might be illustrated as follows

Our objective is to find a "good" encoding and decoding strategy. In particular, one property we would like to have is $\mathbf{P}\{s \neq \hat{s}\}$ is small.

1.1 An introductory example

Suppose I know how to build a memory cell storing a bit with the following properties.After a year, the bit stays the same with probability $1 - f$ and gets flipped with probability f. Moreover, the different cells are assumed to behave independently. One cell is modeled as a channel which is nothing but a collection of distribution over the bitstrings. We use conditional probability notation $W(y|x)$ for the probability of seeing output y when the input is x. For our case, we have for $b \in \{0, 1\}$, $W(b|b) = 1 - f$ and $W(1 - b|b) = f$. Using these cells, I want to be able to store $s \in \{0, 1\}^n$.

Think of $n = 10^6$ bits and $f = 0.1$.

1.1.1 Encoding 1: Trivial encoding

In this case, the encoding function is trivial with just $t_i = s_i$, and the decoder sets $\hat{s}_i = r_i$. Note that r_i is random because the noise applied is random, and thus $\hat{s}_i$ is also random. Let us look at some properties of this code.

Bit error probability. This is the average error in a bit, $\frac{1}{n}\sum_{i=1}^{n} \mathbf{P}\{s_i \neq \hat{s}_i\}$. In this case, we have for any $i \in \{1, \ldots, n\}$, $\mathbf{P}\{s_i \neq \hat{s}_i\} = f$.

We note here that the number of bit flips is the Hamming distance $\Delta(s, \hat{s})$ between s and $\hat{s}$. This has a binomial distribution with parameters (n, f). So $\mathbf{E}\{\Delta(s, \hat{s})\} = nf$ and $\mathbf{Var}\{\Delta(s, \hat{s})\} = nf(1 - f)$. So with high probability, the number of flips is going to be close to nf.

Block error probability. Ideally, we would like to have no error at all in $\hat{s}$ not in even one bit. We would like the block error probability $\mathbf{P}\{s \neq \hat{s}\}$ to be small. For this trivial encoding, we have $\mathbf{P}\{s \neq \hat{s}\} = 1 - \mathbf{P}\{s = \hat{s}\} = 1 - (1 - f)^n$. When $fn \gg 1$, which is the case for the numbers we are thinking of, this probability is extremely close to 1.

Rate. The rate is the number of bits that are stored divided by the number of cells that are used. For the trivial code, the rate is $\frac{n}{n} = 1$.

1.1.2 Encoding 2: Repetition code

Here the idea is to take each one of the n bits and repeat it 3 times. So for $j \in \{1, 2, 3\}$ and $i \in \{1, \ldots, n\}$ $t_{3(i-1)+j} = s_i$.

Rate. We now use 3 cells to encode each bit so the rate is $\frac{1}{3}$.

To determine the error probabilities, we need to choose a way of decoding. It is natural to decode r by looking at blocks of 3 bits each, and for each triple, we decode it to the bit which appears the most among the 3. This means that $\hat{s}_i = \text{maj}\{r_{3(i-1)+1}, r_{3(i-1)+2}, r_{3(i-1)+3}\}$.

Bit error probability.

$$\begin{aligned}\mathbf{P}\{s_i \neq \hat{s}_i\} &= \mathbf{P}\{\text{2 or more flips}\} \\ &= \mathbf{P}\left\{|\{j \in \{1,2,3\} : t_{3(i-1)+j} \neq r_{3(i-1)+j}\}| \geq 2\right\} \\ &= 3f^2(1-f) + f^3 = 3f^2 - 2f^3\,.\end{aligned}$$

Note that this is smaller than f for $f < \frac{1}{2}$. In particular, for $f = 0.1$, $3f^2 - 2f^3 = 0.028$. So at the cost of decreasing the rate, we managed to decrease the error probability.

Block error probability.

$$\begin{aligned}\mathbf{P}\{s \neq \hat{s}\} &= 1 - \mathbf{P}\{\forall i \in \{1,\dots,n\}, s_i = \hat{s}_i\} \\ &= 1 - (1 - 3f^2 + 2f^3)^n \\ &\approx 1 - 0.972^n\end{aligned}$$

1.1.3 Encoding 3: Taking larger blocks

Now we block the message $s_1 \dots s_n$ into blocks of 4 bits and encode each bit separately. For each one of these blocks, we take the $(7,4)$-Hamming code, which is defined as follows. We take $t_1 = s_1, t_2 = s_2, t_3 = s_3, t_4 = s_4$ and we add parity bits $t_5 = s_1 \oplus s_2 \oplus s_3$, $t_6 = s_2 \oplus s_3 \oplus s_4$ and $t_7 = s_1 \oplus s_3 \oplus s_4$.

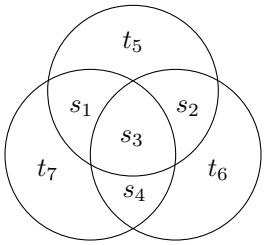

Figure 1.1: Illustration of the Hamming code. Note that the parity of each circle is 0.

Rate. The rate is given by $\frac{n}{7 \cdot \frac{n}{4}} = \frac{4}{7}$.

We now have to determine a method of decoding. And here again, it is natural to decode our message by blocks of 7 bits. So given $r_1, \dots, r_7$, we would like to compute a candidate $\hat{s}_1, \dots, \hat{s}_4$. Suppose that there is at most one flip that occurred when going from $t_1 \dots t_7$ to $r_1 \dots r_7$. Then it is possible to determine which bit was flipped. In fact, among the three circles, some of them have a parity of 0 and some others a parity of 1. We consider the bit that is in all the circles of parity 1 and not in the circles of parity 0, this defines a unique bit and flipping it gives back string that satisfies all the parity constraints of the code. Then taking the first 4 bits of this string gives our decoding string $\hat{s}$. And of course for each block of 7 bits, we do the same procedure.

As such, if there is at most one error that occurs in a block of 7 bits, then we can correct it. So we can compute the block error probability (note that block here refers to the full block of n bits).

Block error probability.

$$\begin{aligned}
\mathbf{P}\{s \neq \hat{s}\} &= 1 - \mathbf{P}\left\{\forall i \in \left\{0, \ldots, \frac{n}{4} - 1\right\}, j \in \{1,2,3,4\} : s_{4i+j} = \hat{s}_{4i+j}\right\} \\
&= 1 - \prod_{i=0}^{n/4-1} \mathbf{P}\{\forall j \in \{1,2,3,4\} : s_{4i+j} = \hat{s}_{4i+j}\} \\
&\leq 1 - (\mathbf{P}\{\text{at most 1 error in a block of 7 bits}\})^{n/4} \\
&= 1 - \left((1-f)^7 + 7f(1-f)^6\right)^{n/4} .
\end{aligned}$$

For $f = 0.1$, we get that $(1-f)^7 + 7f(1-f)^6 \approx 0.85$. So the error probability is roughly $1 - 0.85^{n/4} \approx 1 - 0.96^n$, so this is a slightly worse than the repetition code, which was $1 - 0.972^n$ but better than the trivial code.

Bit error probability. One easy bound is to just say that

$$\begin{aligned}
\mathbf{P}\{s_1 \neq \hat{s}_1\} \leq \mathbf{P}\{s_1 s_2 s_3 s_4 \neq \hat{s}_1 \hat{s}_2 \hat{s}_3 \hat{s}_4\} &\leq 1 - \mathbf{P}\{\text{at most 1 error in the block of 7 bits}\} \\
&= 1 - (1-f)^7 - 7f(1-f)^6 \\
&\approx 0.15 \quad \text{for } f = 0.1 .
\end{aligned}$$

Note that for a bit error probability, this is not great as it is worse than 0.1 which is what one gets for the trivial code. One can obtain better bounds by studying error patterns with two or more flips that do not cause the bit 1 to change. If at least one circle in which s_1 is included does not change parity and s_1 itself is not flipped, then after applying the decoder, $\hat{s}_1 = s_1$. Examples of error patterns of weight two that do not cause s_1 to flip include: $\{s_3, s_4\}, \{s_3, t_7\}, \{s_4, t_7\}, \{s_2, s_3\}, \{s_2, t_5\}, \{s_3, t_5\}, \{t_5, t_6\}, \{s_2, t_6\}, \{t_6, t_7\}, \{s_4, t_6\}$. These are 10 error patters. So we have the bound

$$\begin{aligned}
\mathbf{P}\{s_1 \neq \hat{s}_1\} &\leq 1 - (1-f)^7 - 7f(1-f)^6 - 10f^2(1-f)^5 \\
&\approx 0.09 \quad \text{for } f = 0.1 .
\end{aligned}$$

One can still improve the bound but we will stop here as this is becoming tedious. We have at least shown that the bit error probability is less than 0.1! Note that the calculation is similar of $\mathbf{P}\{s_i \neq \hat{s}_i\}$ is similar for $i \in \{2,3,4\}$.

We can summarize the various encoding strategies we have seen so far in the the following plot. Our objective is to determine the boundary between what is achievable and what is not.

The conventional wisdom would say that in order to decrease the error probability to zero, one needs the rate to go to zero. **However, what Shannon remarkably showed is that one can make the error probability arbitrarily close to 0 while the rate stays strictly positive.** Even more, it is the block error that can be made arbitrarily small, not only the bit error probability.

For our example, with a memory cell with flip probability $f = 0.1$, it turns out that the optimal rate is 0.53. This means that one can use only $2n$ cells in order to store a file of n bits, and this with a very small block error probability (this error probability can be made arbitrarily small as $n \to \infty$).

`blue`[Course structure. The first part is Information theory where we discuss the two main typical problems: Data compression and then channel coding. In the second part we discussion efficient error correcting codes.]

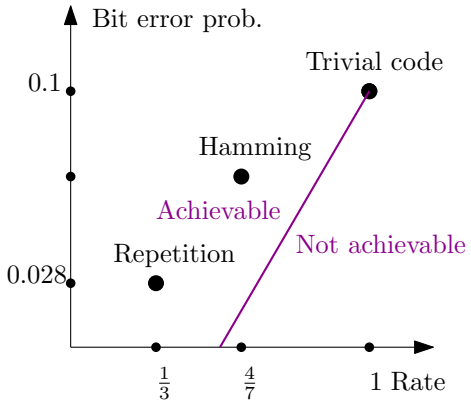

Figure 1.2: Bit error versus rate. We have discussed 3 examples of codes, and they seem to suggest that if we ask the error to go to zero, then the rate needs to go to zero as well. Shannon's noisy coding theorem remarkably shows that it is possible to have an arbitrarily small bit error probability and a nonzero rate.

Chapter 2

Information measures

There are multiple ways of defining how much information there is in some data X. A natural starting point is to say that it is the size of the minimum description of X. For a computer scientist a natural description of the data X is a computer program that generates output X. In this case, one might define the information in a given X as the size of the smallest program that outputs X on some fixed input. This is referred to as the Kolmogorov complexity of X or the algorithmic entropy of X. Even though such a definition is very elegant, this quantity is not computable, which limits its usefulness.

Rather the more successful approach which is taken in "standard" information theory is to define a probabilistic model for the data X and define entropy in terms of the minimum description length (this could be in expectation over the probabilistic model or by allowing a small probability of error).

2.1 Probability notation

All the systems in this course are going to be finite. We consider a probability triple $(\Omega, \mathcal{E}, \mathbf{P})$ with a finite sample space Ω, a set of events $\mathcal{E}$ and a probability measure $\mathbf{P}$. Let $X : \Omega \to \mathcal{X}$ be a random variable taking values in a finite set $\mathcal{X}$. We define the function $P_X : \mathcal{X} \to \mathbb{R}$ by $P_X(x) = \mathbf{P}\{X = x\}$. We will also use for an event E with nonzero probability, $P_{X|E}(x) = \mathbf{P}\{X = x|E\} = \frac{\mathbf{P}\{X=x \cap E\}}{\mathbf{P}\{E\}}$. We also use $X \sim P_X$ to say that X has a distribution P_X.

2.2 Entropy of an event or a random variable

Suppose we have the following simple model for the weather: it is sunny with probability $\frac{1}{2}$ and cloudy with probability $\frac{1}{2}$. Then it is natural to say that learning about whether a day is sunny or cloudy reveals one bit of information. In fact, if you wanted to describe in terms of a bitstring the weather, one bit is the best one can do. As such, we might assign to both events "sunny" and cloudy an entropy of 1.

Suppose now on the other hand that it is sunny with probability 0.999 and rainy with probability 0.001. Then discovering that it is sunny does not give us much information as this is an event that almost always happens. What value of entropy shall we assign to such events? For this let us state

some natural properties we would like the entropy $h(E)$ of an event E to have. First, we ask that $h(E)$ only depends on $\mathbf{P}\{E\}$. This is to ensure that the entropy is not a quantity that depends on the way we build up the probability space. In addition, we require something stronger that for a given probability $p \in [0,1]$, all events E in all probability spaces that have $\mathbf{P}\{E\} = p$ have the same value of the entropy. This means that $h(E) = f(\mathbf{P}\{E\})$ for some function $f : [0,1] \to \mathbb{R}_+$.

An additional natural property we would want is the following additivity property. Suppose I see two events E and E' that are independent, we would like to say that the information in the event $E \cap E'$ is the sum of the entropies of E and E'. In particular, if we take two independent events, each with probability $\frac{1}{2}$, then we require that $f(\frac{1}{4}) = h(E \cap E') = h(E) + h(E') = 2$. More generally, we require that $f(\left(\frac{1}{2}\right)^m) = m$. In addition, we have $f(p) = f((p^{1/n})^n) = nf(p^{1/n})$. Applying this to $p = \left(\frac{1}{2}\right)^m$, we obtain $f(\left(\frac{1}{2}\right)^{\frac{m}{n}}) = \frac{m}{n}$. Assuming in addition that f is a continuous function leads to $f(\left(\frac{1}{2}\right)^z) = z$ for any $z \in [0,\infty)$, which is the same as $f(p) = -\log_2 p$. This leads us to the definition of the entropy of an event.

Definition 2.2.1 (Entropy of event)**.** *The entropy of events is defined as*

$$\begin{aligned} h : \mathcal{E} &\to \mathbb{R}_+ \cup \{\infty\} \\ E &\mapsto -\log_2 \mathbf{P}\{E\} \ . \end{aligned}$$

In particular, a random variable X naturally defines the events $\{X = x\}$, which leads to a restriction of the function h defined as

$$\begin{aligned} h_X : \mathcal{X} &\to \mathbb{R}_+ \cup \{\infty\} \\ x &\mapsto -\log_2 P_X(x) \ . \end{aligned}$$

We call the random variable $h_X(X)$ the surprisal *of the random variable X.*

Thus, if X is a random variable that represents the data, it is natural to define the $h_X(X)$ as its entropy. Note that $h_X(X)$ is a *random variable*, i.e., it takes different values depending on the realisation of X. To get a single number that represents this random variable, it is natural to take its expectation.

Definition 2.2.2. *The (Shannon) entropy of a random variable $X \in \mathcal{X}$ is given by*

$$H(X) = \mathbf{E}\{h_X(X)\} = -\sum_{x \in \mathcal{X}} P_X(x) \log_2 P_X(x) \ .$$

Remark.

- Note that the entropy of X only depends on the probabilities P_X and not the values taken by X. For example, the random variable $2X$ has exactly the same entropy as X.
- The unit of this quantity is bits (it would be nats if we take the $\log$ base e)
- We take the convention that $0 \log_2 0 = 0$, so an $x \in \mathcal{X}$ for which $P_X(x) = 0$ does not contribute to the entropy.
- A comment on the definition of $P_X(X)$. Note that this is *not* $\mathbf{P}\{X = X\}$ but rather it is just the function $P_X : \mathcal{X} \to \mathbb{R}$ applied to the random variable.

- Wa also note that the average is not the only interesting property of $h_X(X)$, the minimum also has a name $H_{\min}(X) = \min_x h_X(x)$ and is often used in cryptography.

□

Proposition 2.2.3. *For any random variable $X \in \mathcal{X}$, we have*

$$0 \leq H(X) \leq \log |\mathcal{X}| ,$$

with equality $H(X) = 0$ if and only if X is a constant random variable and $H(X) = \log |X|$ if and only if X is uniformly distributed.

Proof The positivity is clear. For the upper bound, we use the concavity of the $\log_2$ function:

$$H(X) = \mathbf{E}\left\{\log_2 \frac{1}{P_X(X)}\right\} \leq \log_2 \mathbf{E}\left\{\frac{1}{P_X(X)}\right\} = \log_2 \sum_{x\in\mathcal{X}} P_X(x)\frac{1}{P_X(x)} = \log_2 |\mathcal{X}| .$$

The equality condition comes from the strict concavity of the logarithm. □

Example. If $X \in \{0, 1\}$ with $P_X(1) = p$, then $H(X) = -p \log_2 p - (1 - p) \log_2(1 - p)$. This is a function that often comes up so we give it a name $h_2 : [0, 1] \to [0, 1]$.

TODO: graph here.

2.2.1 Joint entropy and conditional entropy

If we consider two random variables $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, then we can define the joint entropy $H(X, Y) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} P_{XY}(x, y) \log_2 P_{XY}(x, y)$. We also use drop the comma, writing $H(XY)$ for $H(X, Y)$, even though XY does not in any way refer to a product.

Definition 2.2.4. *The conditional entropy $H(X|Y)$ is defined by*

$$H(X|Y) = \sum_{y\in\mathcal{Y}} P_Y(y)H(X|Y = y) ,$$

where $H(X|Y = y)$ is the entropy of the conditional distribution $P_{X|Y=y}$. Note that lements $y \in \mathcal{Y}$ with $P_Y(y) = 0$ do not participate to the sum.

Example. If Y is a copy of X, i.e., $Y = X$, then $H(P_{X|Y=y}) = 0$ for all $y \in \mathcal{Y}$ and thus $H(X|Y) = 0$. On the other hand, if X and Y are independent random variables, then $H(X|Y) = \sum_{y\in\mathcal{Y}} P_Y(y)H(P_{X|Y=y}) = \sum_{y\in\mathcal{Y}} P_Y(y)H(P_X) = H(X)$.

Proposition 2.2.5. *We have $H(X|Y) = H(XY) - H(Y)$.*

Proof This is just writing $P_{X|Y=y}(x) = \frac{P_{XY}(x,y)}{P_Y(y)}$ in terms of entropies. We have

$$\begin{aligned} H(XY) &= -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} P_{XY}(x,y)\log_2(P_{XY}(x,y)) \\ &= -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} P_{XY}(x,y)\log_2(P_Y(y)P_{X|Y=y}(x)) \\ &= -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} P_{XY}(x,y)\log_2 P_Y(y) - \sum_{x\in\mathcal{X},y\in\mathcal{Y}} P_{XY}(x,y)\log_2 P_{X|Y=y}(x) \\ &= -\sum_{y\in\mathcal{Y}} P_Y(y)\log_2 P_Y(y) - \sum_{y\in\mathcal{Y}} P_Y(y)\sum_{x\in\mathcal{X}} P_{X|Y=y}(x)\log_2 P_{X|Y=y}(x) \\ &= H(Y) + H(X|Y)\ . \end{aligned}$$

□

One can build a measure of correlations between sources X and Y out of the entropy, by asking how much the entropy of X decreases when we learn Y.

Definition 2.2.6. *The mutual information is defined by*

$$\begin{aligned} I(X:Y) &= H(X) - H(X|Y) \\ &= H(X) + H(Y) - H(XY)\ . \end{aligned}$$

Writing out the definitions, we get $I(X:Y) = \sum_{x\in\mathcal{X},y\in\mathcal{Y}} P_{XY}(x,y)\log_2 \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)}$.

Example. If $X = Y$, then $I(X:Y) = H(X)$. Also if X and Y are independent, $I(X:Y) = 0$.

To summarize the various entropic measures we have introduced, one usually represents them pictorially in a Venn diagram (Figure 2.2.1)

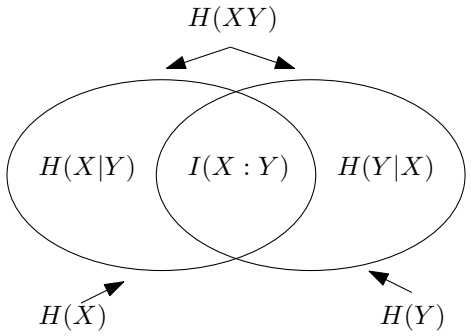

Figure 2.1: Relation between the entropic measures we have introduced

The last definition we introduce here is for the relative entropy (also called Kullback-Leibler divergence). It is a measure of closeness between distributions that is used in many settings. And as we will see shortly, the various entropies we have introduced can be interpreted in terms of this measure of closeness.

Definition 2.2.7. *The relative entropy between two distributions P and Q on $\mathcal{X}$ is defined as*

$$D(P\|Q) = \begin{cases} \infty & \text{if } P(x) > 0 \text{ and } Q(x) = 0 \text{ for some } x \in \mathcal{X} \\ \sum_{x\in\mathcal{X}} P(x) \log_2 \frac{P(x)}{Q(x)} & \text{otherwise.} \end{cases}$$

Remark.

- If for some x, $P(x) = 0$, then we set $P(x) \log_2 P(x)/Q(x) = 0$. However, if there is an x such that $P(x) > 0$ but $Q(x) = 0$, then we set the relative entropy to ∞. This also shows that there is no bound on the relative entropy that depends on $|\mathcal{X}|$ for example, it may be as large as we want.
- Note that $D(P\|P) = 0$ and the relative entropy is not symmetric in its arguments, i.e., $D(P\|Q) \neq D(Q\|P)$ in general.
- The relative entropy usually has this interpretation where if you take as a model distribution Q but the distribution is actually P, then $D(P\|Q)$ measures the difference to optimality because of the wrong choice of distribution.
- We can write $I(X : Y) = D(P_{XY}\|P_X \times P_Y)$, $H(X) = \log|\mathcal{X}| - D(P_X\|U_X)$, where U_X is the uniform distribution on X and $H(X|Y) = \log|\mathcal{X}| - D(P_{XY}\|U_X \times P_Y)$.

□

Proposition 2.2.8. *For any distributions P and Q, we have*

$$D(P\|Q) \geq 0 \,,$$

with equality if and only if $P = Q$.

Proof Let $S = \{x \in \mathcal{X} : P(x) > 0\}$. Then

$$\begin{aligned} D(P\|Q) &= -\sum_{x\in S} P(x) \log_2 \frac{Q(x)}{P(x)} \\ &\geq -\log_2 \sum_{x\in S} P(x) \frac{Q(x)}{P(x)} \\ &= -\log_2 \sum_{x\in S} Q(x) \\ &\geq 0 \,, \end{aligned}$$

where we use the concavity of $\log_2$. To get the equality condition, both inequalities should be saturated. From the saturation of the first inequality together with the strict concavity of $\log_2$, we get that there is a c such that for all $x \in S$, $\frac{P(x)}{Q(x)} = c$. Then, the saturation of the second inequality says that $\sum_{x\in S} Q(x) = 1$. This implies that $c = 1$ and that $P = Q$. □

By writing $I(X : Y) = D(P_{XY}\|P_X \times P_Y)$, a direct corollary of Proposition 2.2.8 is the nonnegativity of the mutual information which is also known as the subadditivity of the entropy.

Corollary 2.2.9. *For any pair of random variables $I(X : Y) \geq 0$. This inequality can also be written as $H(X) \geq H(X|Y)$ or as $H(X) + H(Y) \geq H(XY)$.*

Note that the property $H(X) \geq H(X|Y)$ is a very desirable one: having access to more information (namely Y here) cannot increase the uncertainty of a given system (here X).

Chapter 3

Data compression

The kind of data we are usually interested in, e.g., a text written in some human or computer language, or an image has a lot of structure. Indeed, in the natural representation of the data, we do not expect all the possible values to occur. For example, the natural representation for an English text is as a sequence of letters and spaces. In such a representation, we do not expect to see the sequence of letters `qwt` to appear for example. Similarly, we do not expect all letters to appear as many times. As such the set of possible English texts with n characters is much smaller than the set of all possible sequences of n characters. Data compression is about finding a representation of the data that tries to use as few bits as possible.

Our approach in this chapter will be to model the source we are interested as a random variable X and determine the smallest number of bits needed to store X as a function of the distribution of P_X. Note that it is not easy to describe the distribution P_X over English texts for example, but it is often possible to consider simple distributions that are good enough approximation in that they allow us to exploit at least some of the structure in an English text, even if not all. The easiest model one could think of in this setting is to consider the empirical frequency of letters Q_A and then consider $P_X = Q_A^{\times n}$. This is a very unrealistic model but one can obtain better and better accuracy by considering blocks of b letters, or assuming that letters are generated according to a Markov chain.

This chapter is heavily inspired by the lecture notes [1].

3.1 Setting

Given the data represented by the random variable $X \in \mathcal{X}$ we want to compress it into a bitstring $\{0,1\}^*$ with minimal length. There are multiple natural variants we consider.

1. In variable-length compression, different values of X might be compressed into bitstrings of different lengths. Our aim is then to minimize the *expected* length of the compressed. Note that it might be interesting to minimize other properties such as the median length or the length of the shortest 99% possible values of the data. In general in this model, we require the compression/decompression to be always correct.

2. In fixed-length compression, we fix a small allowable error probability ϵ and the objective is to design a compressor $C : \mathcal{X} \to \{0,1\}^k$, where k is to be minimized.

3.2 Variable-length lossless compression

3.2.1 General compressors

Definition 3.2.1. *A variable-length lossless compressor C for a source $X \in \mathcal{X}$ is a function $C : \mathcal{X} \to \{0,1\}^*$ such that there exists a decompressor $D : \{0,1\}^* \to \mathcal{X}$ with $D \circ C = \mathrm{id}_{\mathcal{X}}$.*

- Such a compressor is *lossless* as because it is possible to decompress perfectly.
- Note that the existence of D such that $D \circ C = \mathrm{id}_{\mathcal{X}}$ is equivalent to C being injective.
- For $x \in \mathcal{X}$, we say that $C(x)$ is a codeword. The set $\{C(x) : x \in \mathcal{X}\}$ is called a code, so we also use the words encoder/decoder for compressor/decompressor.

The length of the compressed data is given by the length of the bitstring $|C(X)|$, which is a random variable. The expected length $\mathbf{E}\{|C(X)|\}$ is thus a natural quantity to minimize. In this case, it is quite simple to determine an optimal compressor. For this, we order all the possible bitstrings $\{0,1\}^*$ in the shortlex order, i.e., if $w \leq_{slex} w'$ if $|w| < |w'|$ or $|w| = |w'|$ and $w \leq_{lex} w'$ and let w_i the i-th bitstring in this order. For example, $w_1 = \emptyset$ is the empty string, $w_2 = 0$, $w_3 = 1$, $w_4 = 00$, etc...

Theorem 3.2.2. *Let P_X be a distribution and $x_1, \dots, x_{|\mathcal{X}|}$ be such that $P_X(x_1) \geq \cdots \geq P_X(x_{|\mathcal{X}|})$. Then C^* defined by $C^*(x_i) = w_i$ is an optimal compressor, i.e., $\mathbf{E}\{|C^*(X)|\} \leq \mathbf{E}\{|C(X)|\}$ for any lossless compressor C. Moreover, we have*

$$H(X) - \log_2(1 + \lfloor \log_2 |\mathcal{X}| \rfloor) \leq \mathbf{E}\{|C^*(X)|\} \leq H(X) \,. \tag{3.1}$$

Proof Let C be a lossless compressor, then C is injective and so for $k \geq 0$, we have

$$|\{x \in \mathcal{X} : |C(x)| \leq k\}| \leq \sum_{\ell=0}^{k} 2^\ell = 2^{k+1} - 1 \,.$$

As a result, we can upper bound the sum of the probabilities of the elements in $\{x \in \mathcal{X} : |C(x)| \leq k\}$ by the sum of the largest $2^{k+1} - 1$ probabilities:

$$\sum_{x \in \mathcal{X} : |C(x)| \leq k} P_X(x) \leq \sum_{i=1}^{\min(2^{k+1}-1, |\mathcal{X}|)} P_X(x_i) \,.$$

But now for the choice of compressor C^*, this inequality becomes an equality: the elements $x_1 \dots x_{2^{k+1}-1}$ are all assigned to bitstrings of length at most k. Note that the case when $|\mathcal{X}| < 2^{k+1} - 1$, both sides of the equality evaluate to 1 we have that $\sum_{x \in \mathcal{X} : |C^*(x)| \leq k} P_X(x) = \sum_{i=1}^{\min(2^{k+1}-1, |\mathcal{X}|)} P_X(x_i) = 1$. As a result,

$$\sum_{x \in \mathcal{X} : |C(x)| \leq k} P_X(x) \leq \sum_{x \in \mathcal{X} : |C^*(x)| \leq k} P_X(x) \,.$$

This implies that

$$\begin{aligned}\mathbf{E}\left\{|C^*(X)|\right\} &= \sum_{k=0}^{\infty} \mathbf{P}\left\{|C^*(X)| > k\right\} \\ &= \sum_{k=0}^{\infty} \sum_{x\in\mathcal{X}:|C^*(x)|>k} P_X(x) \\ &= \sum_{k=0}^{\infty} \left(1 - \sum_{x\in\mathcal{X}:|C^*(x)|\leq k} P_X(x)\right) \\ &\leq \mathbf{E}\left\{|C(X)|\right\} \ .\end{aligned}$$

To prove (3.1), we observe that $|C^*(x_i)| = \lfloor \log_2(i) \rfloor$. Note also that $P_X(x_i) \leq 1 - \sum_{j=1}^{i-1} P_X(x_j) \leq 1 - (i-1)P_X(x_i)$ which implies that $P_X(x_i) \leq \frac{1}{i}$.

$$\begin{aligned}\mathbf{E}\left\{|C^*(X)|\right\} = \sum_{i=1}^{|\mathcal{X}|} P_X(x_i)|C^*(x_i)| &\leq \sum_{i=1}^{|\mathcal{X}|} P_X(x_i)\log_2(i) \\ &= -\sum_{i=1}^{|\mathcal{X}|} P_X(x_i)\log_2\left(\frac{1}{i}\right) \\ &\leq -\sum_{i=1}^{|\mathcal{X}|} P_X(x_i)\log_2(P_X(x_i)) = H(X) \ .\end{aligned}$$

To prove the lower bound in (3.1), let us write for short $L = |C^*(X)|$. Note that $L \in \{0, 1, \ldots, \lfloor \log|\mathcal{X}| \rfloor\}$. Also note that L is a deterministic function of X and as such $H(X, L) = H(X) + H(L|X) = H(X)$. As a result,

$$\begin{aligned}H(X) = H(X, L) &= H(X|L) + H(L) \\ &\leq \sum_{k=0}^{\infty} P_L(k)H(X|L=k) + \log(1 + \lfloor \log|\mathcal{X}| \rfloor) \\ &\leq \sum_{k=0}^{\infty} P_L(k)k + \log(1 + \lfloor \log|\mathcal{X}| \rfloor) \\ &= \mathbf{E}\left\{L\right\} + \log(1 + \lfloor \log|\mathcal{X}| \rfloor) \ ,\end{aligned}$$

where we used in the second inequality the fact that conditioned on $L = k$, $C^*(X) \in \{0,1\}^k$ and as C^* is injective, X takes at most 2^k possible values when we condition on $L = k$ which implies $H(X|L=k) \leq k$. □

3.2.2 Uniquely decodable and prefix-free compressors

We determined the optimal variable-length general compressor C^*. This was very simple to construct, just order the values $x \in \mathcal{X}$ in decreasing order and use up the short bitstrings in shortlex order. We now look at a specific class of compressor that can have a fast encoding and decoding properties. Usually, we want to encode data which is a concatenation of multiple symbols (could be letters, words, pixel colors, etc...) so it makes sense to look look at compressors which compress the data symbol by symbol.

In fact, for a compressor C on the alphabet $\mathcal{A}$, we can naturally define its extension C^+ on $\mathcal{A}^+ = \cup_{n\geq 1}\mathcal{A}^n$ by $C^+(a_1 \cdots a_n) = C(a_1) \cdot C(a_2) \cdots C(a_n)$, where $\cdot$ denotes the concatenation. What condition do we need on C so that C^+ is a lossless compressor? C needs to be at least injective, but this might not be enough to guarantee that C^+ is injective. For example, if $C(a) = 0$, $C(b) = 010$ and $C(c) = 01$, then $C^+(b) = C^+(ca) = 010$. Note that we chose the letter $\mathcal{A}$ for "alphabet" here instead of $\mathcal{X}$ to suggest that we want to encode a concatenation of multiple symbols from $\mathcal{A}$.

Definition 3.2.3 (Uniquely decodable compressor)**.** *A compressor $C : \mathcal{A} \to \{0,1\}^*$ is said to be* uniquely decodable *if its extension C^+ is injective.*

It might not be easy to check that a compressor C is uniquely decodable. But a simple sufficient condition for a compressor to be uniquely decodable is for it to be prefix-free.

Definition 3.2.4 (Prefix-free compressor)**.** *C is a prefix-free compressor if no codeword is a prefix of any other.*

Observe that we often say prefix-free code as this property depends only on the code $\{C(a) : a \in \mathcal{A}\}$. Note that if C is prefix-free then it is uniquely decodable, in fact one can construct a simple decompressor as follows. To decompress $C(a_1) \cdots C(a_n)$, we note that $C(a_1)$ is a unique prefix of $C(a_1) \cdots C(a_n)$ which is a valid codeword, which means one can find it and then keep decoding each symbol a_i sequentially. Note that this decompression algorithm is done in a streaming way, i.e., there is no need to read the whole compressed bitstring to get a_1 for example. This is a nice property of prefix-free codes.

An example of a prefix-free code for $\mathcal{A} = \{a, b, c\}$ is given by $C(a) = 0$, $C(b) = 10$, $C(c) = 110$. But not every uniquely decodable codes needs to be prefix-free. In fact, if $C(a) = 10$, $C(b) = 11$ and $C(c) = 110$. This code can be decoded as follows: if we see a 10 then we decode to a, if we see 11 we look at the next bit: if it is a 1, then we decode the 11 to b, else if it is a 0, then we decode the 110 to c. So uniquely decodable codes are strictly speaking more general but, as we will see in Proposition 3.2.7, not in a very useful way: it is possible to transform any uniquely decodable compressor into a prefix-free one while keeping the length of the encodings of each symbol the same.

The class of uniquely decodable are a natural subclasses of lossless codes that have the advantage that they can be used to encode an arbitrary sequence of symbols from $\mathcal{A}$ without adding any delimiter. This is really not the case for general compressors, for example the empty string $\emptyset$ would be an encoding of some $x \in \mathcal{X}$ but also of any number of copies of x gets compressed to $\emptyset$.

The question we address now is given a distribution P_A on $\mathcal{A}$, if we restrict ourselves to prefix-free codes how to find a code with minimum expected length? And can we relate this minimum length to the entropy of the source? The following theorem only considers prefix-free codes and

not general uniquely decodable codes, but Proposition 3.2.7, which comes later, will show that the optimal uniquely decodable code can be assumed to be prefix-free.

Theorem 3.2.5. *Let $A \in \mathcal{A}$ be a random variable. The Huffman algorithm computes in $O(|\mathcal{A}| \log |\mathcal{A}|)$ a prefix-free compressor $C_H : \mathcal{A} \to \{0,1\}^*$ that has an optimal expected length.*

Moreover, this optimal expected length satisfies

$$H(A) \leq \mathbf{E}\{|C_H(A)|\} < H(A) + 1 \,. \tag{3.2}$$

In order to prove this result, a key observation is a correspondence between prefix-free codes and binary trees. In fact, a binary tree naturally defines a prefix-free code in the following way. First, we label the outgoing edges of a node by $\{0,1\}$, for example the left edge is labeled 0 and the right edge labeled 1. This allows us to assign a bitstring $s(u)$ to every node u by considering the path from the root to u. The set of bitstrings $S = \{s(u) : u \text{ is a leaf}\}$ assigned the *leaves* of such a tree is a prefix-free code. In fact, if $c = s(u)$ and $c' = s(u')$ are codewords such that c is a prefix of c', then by construction of the tree u' should be a descendent of u. As u and u' are supposed to be leaves, this implies that $u = u'$ and thus $c = c'$. Conversely, given a prefix-free code $S \subset \{0,1\}^*$, we can construct a binary tree T in the same way such that $S = \{s(u) : u \text{ is a leaf of } T\}$. See Figure 3.2.2 for an illustration. Note that the length of a given codeword corresponds to the *depth* of the corresponding leaf in the tree. As such, the problem of finding an optimal prefix-free compressor can be formulated as follows. Given a distribution P_A on the set $\mathcal{A}$, find a binary tree T and a function C from $\mathcal{A}$ to the leaves of T such that $\sum_{a \in \mathcal{A}} P_A(a) \cdot \text{depth}(C(a))$ is minimized. There is a beautiful and very efficient algorithm due to Huffman to find such a binary tree that will be seen in the tutorial and homework.

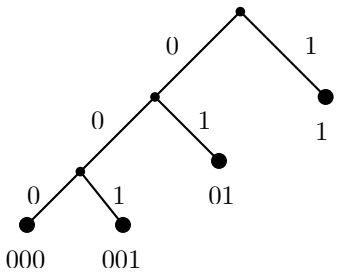

Figure 3.1: A binary tree corresponding to the code $S = \{000, 001, 01, 1\}$.

We now focus on proving the second statement in Theorem 3.2.5, which is an estimate of the expected minimum length in terms of the entropy of the source $H(P_A)$. In order to do this, a very useful tool is the following property which characterizes the set of length of a prefix code.

Lemma 3.2.6 (Kraft inequality)**.** *For any prefix-free compressor C with codewords of lengths $\ell_a = |C(a)|$ for $a \in \mathcal{A}$, we have*

$$\sum_{a \in \mathcal{A}} 2^{-\ell_a} \leq 1 \,. \tag{3.3}$$

Conversely, given a set of lengths $\{\ell_a\}_{a\in\mathcal{A}}$ satisfying Kraft's inequality, we can construct a prefix-free compressor $C : \mathcal{A} \to \{0,1\}^*$ with $|C(a)| = \ell_a$.

Proof Let us start with a prefix-free compressor C and construct the corresponding binary tree. Then ℓ_a is exactly the depth of the leaf that is assigned the bitstring $C(a)$. But it is well-known that for a binary tree T, we have $\sum_{u\in V(T)} 2^{-\text{depth}(u)} \leq 1$. One way of seeing this is by completing the tree into a complete binary tree of height $\ell_{\max} = \max_a \ell_a$ and considering the nodes at depth $\ell_{\max}$. Then for the codeword $C(a)$, we have $2^{\ell_{\max}-\ell_a}$ descendants at depth $\ell_{\max}$. Taking all such nodes for every $a \in \mathcal{A}$, we get a total of $\sum_{a\in\mathcal{A}} 2^{\ell_{\max}-\ell_a}$ nodes at depth $\ell_{\max}$. But on the other hand, there are at most $2^{\ell_{\max}}$ at depth $\ell_{\max}$. This gives inequality (3.3).

To prove the converse, we order the elements of $\mathcal{A}$ as $a_1, \ldots, a_{|\mathcal{A}|}$ so that $\ell_{a_1} \leq \cdots \leq \ell_{a_{|\mathcal{A}|}}$. We define $C(a_i)$ to be the first ℓ_{a_i} bits in the binary expansion of $b_i = \sum_{j=1}^{i-1} 2^{-\ell_{a_j}}$. These numbers b_i lie in the interval $[0,1)$ using the condition (3.3). For example, $C(a_1) = 0^{\ell_{a_1}}$ where 0^ℓ is a concatenation of ℓ zeros, and $C(a_2) = 0^{\ell_{a_1}-1}10^{\ell_{a_2}-\ell_{a_1}}$ etc... Let us show that C is prefix-free. Take two codewords $C(a_i)$ and $C(a_k)$ with $i < k$. Note that $b_k - b_i = \sum_{j=i}^{k-1} 2^{-\ell_{a_j}} \geq 2^{-\ell_i}$. Note that any codeword that has $C(a_i)$ as a prefix is a binary expansion of a number that is at most $b_i + \sum_{p=\ell_{a_i}+1}^{\max_a \ell_a} 2^{-p} < b_i + 2^{-\ell_i}$. As a result, $C(a_i)$ cannot be a prefix of $C(a_k)$ and this concludes the proof that C is prefix-free.

□

Using this lemma, the problem of finding an optimal compressor is equivalent to the following optimization program with variables $\ell_a \in \mathbb{N}_+$ for $a \in \mathcal{A}$:

$$\text{minimize} \quad \sum_{a\in\mathcal{A}} P_A(a)\ell_a \tag{3.4}$$

$$\text{subject to} \quad \ell_a \in \mathbb{N}_+ \quad \text{for } a \in \mathcal{A} \tag{3.5}$$

$$\sum_{a\in\mathcal{A}} 2^{-\ell_a} \leq 1\,. \tag{3.6}$$

As you probably know, optimization programs with integrality constraints are often difficult algorithmically. However, this one turns out to have an efficient algorithm: Huffman's algorithm (to be seen in the tutorials). Using this formulation, it is relatively easy to prove the inequality (3.2).

Proof [of Theorem 3.2.5] Note that using Lemma 3.2.6, we have $\mathbf{E}\{|C_{\text{H}}(X)|\}$ is equal to the value of the maximization in (3.4). As such, it suffices to find an upper and lower bound on the value of (3.4).

Let us start with the lower bound. For that we relax the constraint that $\ell_a \in \mathbb{N}_+$ into just $\ell_a \in \mathbb{R}_+$ (or just $\mathbb{R}$ even as the fact that $\ell_a \geq 0$ is already implied by the other constraint). We now simply rename the variables to $Q(a) = 2^{-\ell_a}$. The program becomes

$$\text{minimize} \quad -\sum_{a\in\mathcal{A}} P_A(a)\log_2 Q(a) \tag{3.7}$$

$$\text{subject to} \quad \sum_{a\in\mathcal{A}} Q(a) \leq 1\,. \tag{3.8}$$

We can interpret Q as a subnormalized distribution. The objective function is supposed to remind

us of the relative entropy between P_A and the subnormalized distribution Q. In fact,

$$\begin{aligned}-\sum_{a\in\mathcal{A}} P_A(a)\log_2 Q(a) &= -\sum_{a\in\mathcal{A}} P_A(a)\log_2 P_A(a) + \sum_{a\in\mathcal{A}} P_A(a)\log_2 P_A(a) - \sum_{a\in\mathcal{A}} P_A(a)\log_2 Q(a)\\ &= H(P_A) + D(P_A\|Q)\,,\end{aligned}$$

where we slightly generalized the definition of the relative entropy to subnormalized distributions. Remember that our objective was to show that $H(P_A)$ is an upper bound on the minimum expected length. For that it only remains to show that for a subnormalized distribution Q, $D(P_A\|Q) \geq 0$. For that, let $Q'(a) = \frac{Q(a)}{\sum_a Q(a)}$. Then we have

$$\begin{aligned}D(P_A\|Q) &= \sum_a P_A(a)\log_2 P_A(a) - \sum_a P_A(a)\log_2(Q'(a)\cdot\sum_a Q(a))\\ &= \sum_a P_A(a)\log_2 P_A(a) - \sum_a P_A(a)\log_2 Q'(a) - \sum_a P_A(a)\log_2(\sum_a Q(a))\\ &= D(P_A\|Q') - \log_2(\sum_a Q(a))\\ &\geq 0\,,\end{aligned}$$

using the fact that the relative entropy between distributions is always nonnegative and that Q is subnormalized. Observe that we actually showed that the value of the relaxation (3.7) is exactly equal to $H(P_A)$: it suffices to take $Q = P$.

The proof of this lower bound gave us a good hint for which values of ℓ_a we should be choosing: $\ell_a = -\log P_A(a)$. And in fact, we showed that if these values of ℓ_a turn out to be integers, then we get an minimum expected length of $H(P_A)$ exactly. However, in general, $-\log P_A(a)$ are not integers, so we cannot always make this choice to find a good compressor.

To prove the upper bound, the idea should now be clear. We fix $\ell_a = \lceil -\log_2 P_A(a)\rceil$. Then we have $\sum_a 2^{-\ell_a} \leq \sum_a 2^{-\log_2 P_A(a)} = 1$ so the constraint is satisfied. In addition, the objective function take the value $\sum_a P_A(a)\lceil -\log_2 P_A(a)\rceil < \sum_a P_A(a)(-\log_2(P_A(a))+1) \leq 1 + H(P_A)$. □

To conclude our study of uniquely decodable codes, we show next that cannot achieve a smaller expected length than the best prefix code. Not that in order to show this, it is sufficient to show that any uniquely decodable code still satisfies the inequality (3.3).

Proposition 3.2.7. *For any uniquely decodable compressor C with codewords of lengths $\ell_a = |C(a)|$ for $a \in \mathcal{A}$, we have*

$$\sum_{a\in\mathcal{A}} 2^{-\ell_a} \leq 1\,.$$

Proof Let us define for a string $a^n = a_1\dots a_n \in \mathcal{A}^n$, $C(a^n) = C(a_1)\cdots C(a_n)$ and thus the length $\ell_{a_1\dots a_n} = \ell_{a_1} + \cdots + \ell_{a_n}$. Then we have

$$\begin{aligned}\left(\sum_{a\in\mathcal{A}} 2^{-\ell_a}\right)^n &= \sum_{a^n\in\mathcal{A}^n} 2^{-\ell_{a^n}}\\ &= \sum_{m=1}^{n\ell_{\max}} N_m 2^{-m}\,,\end{aligned}$$

where $N_m = |\{a^n \in \mathcal{A}^n : \ell_{a^n} = m\}|$. As C is uniquely decodable $C(a^n)$ should give different bitstrings for different values of a^n. As a result, $N_m \leq 2^m$. This implies that

$$\left(\sum_{a\in\mathcal{A}} 2^{-\ell_a}\right)^n \leq n\ell_{\max} .$$

So for any $n \geq 1$, we have $\sum_{a\in\mathcal{A}} 2^{-\ell_a} \leq (n\ell_{\max})^{\frac{1}{n}} \to 1$ as $n \to \infty$. □

So we have now seen that if we restrict ourselves to the natural class of uniquely decodable codes, we can find the one which minimizes the expected length efficiently and moreover, this minimum expected length lower bounded by $H(P_A)$, which can be achieved if $-\log_2 P_A(a)$ are all integers, and is upper bounded by $H(P_A) + 1$. If we compare this to the completely general compressor we studied in Section 3.2.1, we see that we are loosing at most $1+\log_2(1+\lceil\log_2|\mathcal{A}|\rceil)$ in order to be able to encode multiple letters without needing a delimiter.

3.3 Fixed-length almost lossless compression

We now again take our source $X \in \mathcal{X}$ but we would like to compress it into just k bits, i.e., $C : \mathcal{X} \to \{0,1\}^k$. In fact, it might not be desirable for some applications to have very different encoding lengths for different files. In this case, if we require the compression to be perfectly lossless, then we have to take $k \geq \log|\mathcal{X}|$. However, we might make significant gains in terms of space by allowing a small error probability that we'll call δ.

Definition 3.3.1. *A fixed-length compressor for a source $X \in \mathcal{X}$ of length ℓ is a function $C : \mathcal{X} \to \{0,1\}^\ell$.*

It has error probability at most δ if there exists a function $D : \{0,1\}^\ell \to \mathcal{X}$ such that

$$\mathbf{P}\{D \circ C(X) \neq X\} \leq \delta .$$

Note that our objective now is to understand the minimum ℓ for some small δ. We introduce a notation for the best compressor that achieves an error probability $\leq \delta$:

$$\ell^{\text{opt}}(X,\delta) = \min\{\ell : \text{ there exists a length } \ell \text{ compressor for } X \text{ with error probability } \delta\} .$$

It should be quite intuitive how to find an optimal encoder. Construct the set $S^*_\delta \subseteq \mathcal{X}$ in the following way. Sort the elements by their probabilities $\mathcal{X} = \{x_1, \ldots, x_{|\mathcal{X}|}\}$ so that $P_X(x_1) \geq \cdots \geq P_X(x_{|\mathcal{X}|})$. Then for $i = 1$ to $|\mathcal{X}|$ keep adding x_i to S_δ until $\sum_{x\in S^*_\delta} P_X(x) \geq 1-\delta$. Once this is satisfied, we stop.

Theorem 3.3.2. *The size of the set S_δ determines ℓ^{opt}:*

$$\ell^{opt}(X,\delta) = \lceil\log_2|S^*_\delta|\rceil .$$

Proof We start by proving the inequality "$\leq$". Using the notation above, let $S^*_\delta = \{x_1, \ldots, x_k\}$ and $\ell = \lceil\log_2|S^*_\delta|\rceil$. Note that $k \leq 2^\ell$. We define C as follows.

$$C(x_i) = \begin{cases} \text{bin}^\ell(i-1) & \text{if } i \leq k \\ 0^\ell & \text{if } i > k , \end{cases}$$

where $\text{bin}^\ell(i)$ is the binary string of length ℓ representing the integer $i \in \{0, \dots, 2^\ell - 1\}$. Then defining $D(b) = x_i$ with $i - 1$ the number in $\{0, \dots, 2^\ell - 1\}$ whose binary representation is b.

$$\mathbf{P}\left\{D \circ C(X) = X\right\} \geq \sum_{i=1}^{k} P_X(x_i) \geq 1 - \delta \, .$$

For the converse, let C be a compressor with length ℓ and error probability $\leq \delta$. Then consider the set $S_\delta = D(\{0,1\}^\ell)$. Then

$$\begin{aligned} \sum_{x \in S_\delta} P_X(x) &= \mathbf{P}\left\{X \in D(\{0,1\}^\ell)\right\} \\ &\geq \mathbf{P}\left\{X = D \circ C(X)\right\} \\ &\geq 1 - \delta \, . \end{aligned}$$

But among the sets with total probability at least $1 - \delta$, S_δ^* is the smallest one, so $|S_\delta^*| \leq |S_\delta| \leq 2^\ell$. This implies that $\ell \geq \log_2 |S_\delta^*|$ and as ℓ is an integer $\ell \geq \lceil \log_2 |S_\delta^*| \rceil$. □

Remark. Unlike the previous sections where we could always relate the minimum length (approximately) to the Shannon entropy of the source $H(X)$, here $\ell^{\text{opt}}(X, \delta)$ is characterized by different quantity. In fact the term $\log_2 |S_\delta^*|$ can be quite different from $H(X)$. For example, for $\delta = 0$, we can take the distribution on $\{0, \dots, m\}$ defined by $P_X(0) = 1 - \epsilon$ and $P_X(i) = \epsilon/m$ for $i \in \{1, \dots, m\}$. Then $\log_2 |S_0^*| = \log_2(m+1)$, but $H(X) = -(1-\epsilon)\log(1-\epsilon) - m \cdot \frac{\epsilon}{m} \log \frac{\epsilon}{m} = h_2(\epsilon) + \epsilon \log m$, which can be much smaller than $\log_2(m+1)$.

Even if it is not the Shannon entropy, the quantity $\log_2 |S_\delta^*|$ can still be called an entropic measure and indeed it has a name: it is called the Hartley entropy, or more precisely the smoothed version. The Hartley entropy can be defined by

$$H_0(X) = \log_2 |\text{supp}(P_X)| \, .$$

Note that this shares some properties with the Shannon entropy $H(X)$. For example, $H(X) = 0$ if and only if X takes only one value and $H(X) = \log_2 |\mathcal{X}|$ when X is uniformly distributed. Note that using the concavity of the logarithm, we always have

$$H(X) = \sum_{x \in \text{supp}(P_X)} P_X(x) \log_2 \frac{1}{P_X(x)} \leq \log_2 |\text{supp}(P_X)| = H_0(X) \, .$$

But as shown above there can be a very large gap between the two. $H_0(X)$ naturally corresponds to the number of bits needed to store X with zero probability of error. However, if we allow an error probability of δ, we can change the distribution P_X by δ in order to reduce it's support: we get the *smoothed* version:

$$H_0^\delta(X) = \min_{S_\delta : \sum_{x \in S_\delta} P_X(x) \geq 1 - \delta} \log_2 |S_\delta| \, .$$

Note that the smoothed Hartley entropy can be very different from the Hartley entropy itself. For an extreme example, we can have for $X \in \{0, 1, \dots, m\}$ with $X = 0$ with probability $1 - \delta$ and $X = i$ with probability δ/m, we have $H_0(X) = \log_2(1+m)$ but $H_0^\delta(X) = 0$. We have also seen an

example in the first homework. If $X = X_1, \ldots, X_n$ is a bitstring of length n with X_i independent and with distribution Bernoulli p, then $H_0(X) = n$ but $H_0^\delta(X) \approx h_2(p)n$. □

Recall that the Shannon entropy $H(X)$ is the expectation of the surprisal of X, which is the random variable $h_X(X) = -\log_2 P_X(X)$. It turns out that one can relate $\ell^{\text{opt}}(X, \delta)$ not to the expectation of the random variable $h_X(X)$ but rather another property of it, given in the next proposition. Before getting into the statement, let us try to familiarize with the surprisal random variable $h_X(X)$ with some examples.

- If X is uniformly distributed on $\mathcal{X}$, then $-\log_2 P_X(x) = \log_2 |\mathcal{X}|$ for all $x \in \mathcal{X}$. So $h_X(X)$ is the constant random variable equal to $\log |\mathcal{X}|$ (notice also that in this case, the entropy is $\log |\mathcal{X}|$).
- If $\mathcal{X} = \{1, \ldots, 2t\}$ with $P_X(x) = \frac{3}{4t}$ for $x \in \{1, \ldots, t\}$ and $P_X(x) = \frac{1}{4t}$ for $t \in \{t+1, \ldots, 2t\}$. Then $h_X(X)$ will take the value $\log \frac{4t}{3}$ with probability $\frac{3}{4}$ and $\log \frac{t}{4}$ with probability $\frac{1}{4}$.
- Another example is for $\mathcal{X} = \{1, \ldots, t+1\}$ and $P_X(i) = 2^{-i}$ for $i \in \{1, \ldots, t-1\}$ and $P_X(t) = P_X(t+1) = 2^{-t}$. In this case, the random variable $h_X(X)$ takes value i with probability 2^{-i} and value t with probability 2^{-t+1}.

The next proposition shows that we can relate $\ell^{\text{opt}}(X, \delta)$ to the smallest ℓ such that the total probability above ℓ is at most δ. So this property is more about the tails of $h_X(X)$ than about its expectation.

Proposition 3.3.3.

$$\ell^{opt}(X, \delta) \leq \min\{\ell \in \mathbb{N}_+ : \mathbf{P}\{h_X(X) > \ell\} \leq \delta\}\,. \tag{3.9}$$

Moreover, for any $\tau > 0$,

$$\ell^{opt}(X, \delta) \geq \min\{\ell \in \mathbb{N}_+ : \mathbf{P}\{h_X(X) > \ell + \tau\} - 2^{-\tau} \leq \delta\}\,. \tag{3.10}$$

Note that the statement in (3.10) is referred to as an achievability result as it gives a way of constructing a compressor. On the other hand an inequality of the type (3.10) is known as a converse bound as it gives a limitation on the length that can be achieved by compressors.

Proof For the upper bound on ℓ^{opt}, let ℓ satisfy $\mathbf{P}\{h_X(X) \leq \ell\} \leq \delta$. Then we construct $S = \{x \in \mathcal{X} : P_X(x) \geq 2^{-\ell}\}$. Then we have $|S| \leq 2^\ell$. In addition, $\mathbf{P}\{X \in S\} = \mathbf{P}\left\{P_X(X) \geq 2^{-\ell}\right\} = \mathbf{P}\{-\log_2 P_X(X) \leq \ell\} \geq 1-\delta$ using the property of ℓ. So we can compress the elements of the set S with no error and so $\ell^{\text{opt}}(X, \delta) \leq \ell$.

For the lower bound, let C be a compressor with error probability $\leq \delta$ and length $\ell = \ell^{\text{opt}}(X, \delta)$. Define $S = \{x \in \mathcal{X} : D(C(x)) = x\}$. Then as $S \subseteq D(\{0,1\}^\ell)$, $|S| \leq 2^\ell$. By the assumption that the error probability is at most δ, we have $\mathbf{P}\{X \in S\} \geq 1 - \delta$. The idea now is that because S is a set with total probability $1 - \delta$ and having at most 2^ℓ elements, the elements $x \in S$ cannot have

a probability much smaller than $2^{-\ell}$. More precisely,

$$\begin{aligned}1-\delta &\leq \sum_{x\in S} P_X(x) \\ &\leq \sum_{x\in\mathcal{X}:P_X(x)\geq 2^{-\ell-\tau}} P_X(x) + \sum_{x\in S:P_X(x)<2^{-\ell-\tau}} P_X(x) \\ &\leq \mathbf{P}\left\{h_X(X)\leq \ell+\tau\right\} + 2^{\ell}2^{-\ell-\tau} \\ &= 1-\mathbf{P}\left\{h_X(X)>\ell+\tau\right\}+2^{-\tau}\ .\end{aligned}$$

This implies that ℓ satisfies the condition in the right hand side of (**??**). □

A sufficient condition for a random variable to be with high probability close to its expectation? Being a sum of independent random variables. For $h_X(X)$, this is the case if the source is a concatenation of independent symbols, also called a memoryless source. We assume $X^n = X_1 \dots X_n$ with X_i independent and identically distributed random variables in $\mathcal{A}$.

Theorem 3.3.4 (Shannon's source coding theorem)**.** *Let $X^n = X_1 \dots X_n$ be an sequence of independent and distributed as $X \in \mathcal{A}$. Then for any $\delta \in (0,1)$,*

$$\lim_{n\to\infty} \frac{\ell^{opt}(X^n,\delta)}{n} = H(X)\ .$$

Proof We use Proposition 3.3.3 applied to a memoryless source. For that we need to get a handle on $\mathbf{P}\left\{h_{X^n}(X^n) > \ell\right\}$. Recall that $h_{X^n}(X^n) = -\log_2 P_{X^n}(X^n)$ and using the iid assumption this gives

$$h_{X^n}(X^n) = \sum_{i=1}^{n} -\log_2 P_X(X_i)\ .$$

As such we have a sum of iid random variables each one having an expectation that is equal to $H(X)$. We thus know from the weak law of large numbers that such a random variable is very close to its expectation. The expectation $\mathbf{E}\left\{h_{X^n}(X^n)\right\} = n\mathbf{E}\left\{h_X(X_1)\right\} = nH(X)$. For the sake of completeness, we give a complete proof not relying on the weak law of large numbers. We use Chebychev's inequality

$$\mathbf{P}\left\{|h_{X^n}(X^n) - \mathbf{E}\left\{h_{X^n}(X^n)\right\}| \geq t\right\} \leq \frac{\mathbf{Var}\left\{h_{X^n}(X^n)\right\}}{t^2} \tag{3.11}$$

$$= \frac{n\cdot\mathbf{Var}\left\{h_X(X_1)\right\}}{t^2}\ . \tag{3.12}$$

Note that in our setting X takes finitely many values so $\mathbf{Var}\left\{h_X(X)\right\} < \infty$. As a result, applying this inequality with $t = \sqrt{\frac{n\mathbf{Var}\{h_X(X)\}}{\delta}}$ we get

$$\mathbf{P}\left\{h_{X^n}(X^n) > nH(X) + \sqrt{\frac{n\mathbf{Var}\left\{h_X(X)\right\}}{\delta}}\right\} \leq \delta\ .$$

So $\ell^{\text{opt}}(X^n, \delta) \leq nH(X) + \sqrt{\frac{n\mathbf{Var}\{h_X(X)\}}{\delta}}$. By taking the limit as $n \to \infty$, this proves that the entropy is an upper bound on the asymptotic rate of compression.

For the lower bound, let $\alpha > 0$ parameter we choose later. Then take $t = \sqrt{\frac{n\mathbf{Var}\{h_X(X)\}}{\alpha}}$ in (3.11) and get

$$\mathbf{P}\left\{h_{X^n}(X^n) \leq nH(X) - \sqrt{\frac{n\mathbf{Var}\left\{h_X(X)\right\}}{\alpha}}\right\} \leq \alpha\,.$$

As a result, taking $\ell = nH(X) - 2\sqrt{\frac{n\mathbf{Var}\{h_X(X)\}}{\alpha}}$ and $\tau = \sqrt{\frac{n\mathbf{Var}\{h_X(X)\}}{\alpha}}$. We now choose $\alpha > 0$ small enough so that $1 - \delta > \alpha + 2^{-\tau}$. This gives us $\ell^{\text{opt}}(X^n, \delta) \geq nH(X) - 2\sqrt{\frac{n\mathbf{Var}\{h_X(X)\}}{\alpha}}$, which leads to the desired result by taking the limit $n \to \infty$. □

Random coding: done in TD

In order to prepare for more complex tasks, it is useful to give another proof of the achievability result in (3.10). In fact, it is even going to give us a result that is weaker than the (3.10) but it is nonetheless a good warm-up for the technique used in Shannon's noisy coding theorem. This technique is the probabilistic method: the compressor C will be chosen *at random* and remarkably we will show that it can have length roughly the same as what we obtained in (3.10) by choosing the set of symbols we compress as the ones with probability $\geq 2^{-\ell}$.

Proposition 3.3.5. *For any $\tau > 0$.*

$$\ell^{opt}(X, \delta) \leq \min\{\ell \in \mathbb{N}_+ : \mathbf{P}\left\{h_X(X) > \ell - \tau\right\} + 2^{-\tau} \leq \delta\}\,. \tag{3.13}$$

Proof Let $\tau > 0$ and let ℓ be such that $\mathbf{P}\left\{h_X(X) > \ell - \tau\right\} + 2^{-\tau} \leq \delta$. Our objective is to construct a compressor with length ℓ and error probability at most δ. Let us start by fixing a simple decompressor. We define $D(y) = x$ if there is a unique x such that $C(x) = y$ and $P_X(x) \geq 2^{-\ell+\tau}$. If no such x exists or if it is not unique then $D(y)$ is set to an arbitrary fixed $x_0 \in \mathcal{X}$. Let us analyze the error probability of this compressor-decompressor pair. For a given x, there are two possible reasons it can get decompressed incorrectly. First, if $P_X(x) < 2^{-\ell+\tau}$ and second if $P_X(x) \geq 2^{-\ell+\tau}$ but there is another $x' \neq x$ such that $C(x) = C(x')$ and $P_X(x') \geq 2^{-\ell+\tau}$. More precisely, we can write

$$\begin{aligned}\mathbf{P}\left\{D(C(X)) \neq X\right\} &= \sum_{x\in\mathcal{X}} P_X(x)\mathbf{P}\left\{D(C(x)) \neq x | X = x\right\} \\ &\leq \sum_{x\in\mathcal{X}} P_X(x)\left(\mathbf{1}_{P_X(x)<2^{-\ell+\tau}} + \sum_{x'\neq x}\mathbf{1}_{C(x)=C(x'),P_X(x')\geq 2^{-\ell+\tau}}\right)\,.\end{aligned}$$

The first term does not depend on the choice of C and it corresponds to $\mathbf{P}\left\{h_X(X) > \ell - \tau\right\}$. Now the idea will be to choose the codewords $C(x)$ uniformly at random from $\{0,1\}^\ell$ so that in expectation the second term is small. In fact if $x \neq x'$, then $\mathbf{E}\left\{\mathbf{1}_{C(x)=C(x')}\right\} = \frac{1}{2^\ell}$. As such the second term is in expectation over the choice of codewords $\{C(x)\}_{x\in\mathcal{X}}$ given by

$$\sum_{x\in\mathcal{X}} P_X(x) \sum_{x'\neq x: P_X(x')\geq 2^{-\ell+\tau}} \frac{1}{2^\ell} \leq \sum_{x\in\mathcal{X}} P_X(x)\cdot\frac{2^{\ell-\tau}}{\tau} = 2^{-\tau}\,.$$

□

So far, we have studied the fundamental limit of compression in quite generic scenarios, that do not necessarily correspond to the compression tasks we would like to perform in practice. In fact,

1. The sources we are interested in X are composed of a long stream of symbols (for example in a text is composed of letters) but the symbols are *not* independent. For example, given a text that starts with `informatio`, it is extremely likely that the next letter is an `n`. The context can thus be used in order to improve the compression. To take the context into account, one could cut the stream into sufficiently large blocks of b symbols each, and then say that the blocks are independent and identically distributed. This will certainly work better than assuming that the symbols are independent. In fact, an even better model would be a Markov model where the probability of the n-th symbol only depends on the $b-1$ symbols coming before it. However, note that a general distribution over a block of b letters is given by 26^b parameters, which very quickly becomes prohibitive.

2. Another assumption we made is that we have full knowledge of the distribution of the source X. This is unrealistic for several reasons. As mentioned above, even the describing the distribution might be very costly. In addition, we would like to build compressors that work for a wide variety of inputs and it should be able to "learn" what are the specifics of the stream it is currently compressing.

3.4 Universal compression

In this section, we consider a stream $X^n = X_1 \dots X_n$ of n symbols with $X_i \in \mathcal{X}$. We do not have access to the distribution P_{X^n}.

3.4.1 Arithmetic codes

The idea here is to build a probabilistic model for the symbols while we are reading the stream. For example, we could start with the model $P_1(a) = \frac{1}{|\mathcal{X}|}$ for all $a \in \mathcal{X}$ which gives the same probability for every symbol. Then assuming we have seen the symbols $x_1 \dots x_{i-1}$, we could define the model for step i by

$$P_i(a|x_1 \dots x_{i-1}) = \frac{|\{j \in \{1, \dots, i-1\} : x_j = a\}| + 1}{i - 1 + |\mathcal{X}|} \,. \tag{3.14}$$

Observe that this is a valid distribution over elements $a \in \mathcal{X}$ and the idea is to give more weight to the symbols a that have occurred more often. Then, naturally, the larger the weight given to a symbol, the fewer bits we would like to use for that symbol.

To describe the encoding strategy, it is very useful to think of it in two steps. In the first step we are going to assign our stream $x^n \in \mathcal{X}^n$ to an interval $I(x^n) \subseteq [0, 1)$. The second step would be to map the interval $I(x^n)$ into a bitstring $C(x^n)$.

From x^n to $I(x^n)$ The idea is that the empty stream corresponds to the full interval $I(\emptyset) = [0, 1)$ and then when we read the symbol x_1, we take a subinterval $I(x_1)$ of the current interval $I(\emptyset)$ with length proportional to $P_1(x_1)$. More generally, when reading x_i at step i, we define $I(x_1 \dots x_i)$ to be a subinterval of $I(x_1 \dots x_{i-1})$ with length proportional to $P_i(x_i|x_1 \dots x_{i-1})$.

More precisely, let us index the symbols of $\mathcal{X} = \{a^1, \dots, a^{|\mathcal{X}|}\}$ in an arbitrary way. We define $I(\emptyset) = [0, 1)$. Then given $I(x_1 \dots x_{i-1}) = [u_{i-1}, v_{i-1})$, we compute $I(x_1 \dots x_i)$ as follows. Suppose $x_i = a^k$ for some $k \in \{1, \dots, |\mathcal{X}|\}$. The length of the subinterval of $I(x_1 \dots x_{i-1})$ that we should take when adding x_i should be proportional to $P_i(x_i|x_1 \dots x_{i-1})$. The interval will be defined in terms of $\alpha = \sum_{p=1}^{k-1} P_i(a^p|x_1 \dots x_{i-1})$ and $\beta = \sum_{p=1}^{k} P_i(a^p|x_1 \dots x_{i-1})$ by

$$I(x_1 \dots x_i) = [u_i, v_i) \quad \text{with } u_i = u_{i-1} + (v_{i-1} - u_{i-1}) \cdot \alpha \text{ and } v_i = u_{i-1} + (v_{i-1} - u_{i-1}) \cdot \beta \,. \tag{3.15}$$

From $I(x^n)$ to $E(x^n)$ In this step, we take the interval $I(x^n)$ and map it to a bitstring. Note first that for $x^n \neq \tilde{x}^n$ two streams in $\mathcal{X}^n$, the intervals $I(x^n)$ and $I(\tilde{x}^n)$ are disjoint. In other words, given a real number $y \in [0, 1)$, it is in at most one interval $I(x^n)$ (actually exactly one interval) for some $x^n \in \mathcal{X}^n$. So in order to encode an interval $I(x^n)$ it suffices to take a number of the form $0.y_1 \dots y_\ell \in I(x^n)$ for bits $y_j \in \{0, 1\}$ trying to have ℓ as small as possible. Writing again the interval $I(x^n) = [u_n, v_n)$, one possible choice is

$$C(x^n) = \frac{\lceil 2^\ell u_n \rceil}{2^\ell} \quad \text{where } \ell \in \mathbb{N} \text{ is the smallest satisfying } \frac{\lceil 2^\ell u_n \rceil}{2^\ell} < v_n \,. \tag{3.16}$$

Note that we are naturally identifying rational numbers of the form $\frac{k}{2^\ell}$ for $k \in \{0, \dots, 2^\ell - 1\}$ with bitstrings of length ℓ.

Decompressor Now that we have defined the compression function, we need to check that it is possible to decompress with no error. Given $y \in \{0, 1\}^m$, we first see it as rational number in $[0, 1)$ with binary representation $0.y_1 \dots y_m$. We construct the intervals $I(a)$ for every possible $a \in \mathcal{X}$. Let x_1 be the unique symbol such that $y \in I(x_1)$. Then, at step i, as we have determined $x_1 \dots x_{i-1}$, we can compute $P_i(a|x_1 \dots x_{i-1})$ and thus can determine the intervals $I(x_1 \dots x_{i-1}a)$ for any $a \in \mathcal{X}$. We set x_i to be the unique symbol such that $y \in I(x_1 \dots x_i)$. Note that this decompression algorithm can actually output an infinite stream $x_1 x_2 \dots$. The compressed stream is a prefix of this infinite stream. To get the compressed stream exactly we should either know its length n in advance (or encode it separately) or have an end-of-file symbol to tell us when to stop.

Computational aspects We briefly comment on the fact that the compression and decompression computations can be performed quite efficiently. For example, to compute the probabilities given by the model in (3.14), it suffices to keep a counter for the number of times we have seen each symbol. Then the interval can be computed very efficiently on the fly via (3.15). Then to determine the bitstring $C(x^n)$, a simple rounding procedure is sufficient as shown in (3.16).

3.4.2 Lempel-Ziv coding (TODO)

The Lempel-Ziv algorithm does not build a probabilistic model for the stream but rather, it is based on a dictionary of words that already appeared. Then, we replace the occurrence of that word with

its index in the dictionary. There are multiple variants, we describe a simple one here.

See `https://www.cs.cmu.edu/~guyb/realworld/compression.pdf` for a nice introduction to data compression with discussion of practical aspects.

3.5 Lossy compression (TODO)

The compression algorithms we looked at were lossless or almost lossless in the sense that it is possible to decompress perfectly (or with probability close to 1). In lossy compression, the original data cannot be recovered. But this does not mean that it is completely useless. In fact, for structured data, such as images, it is often enough to be able to recover an image that looks the same to a human eye. In particular, one might obtain very significant space savings by eliminating imperceptible noise that has large entropy.

Chapter 4

Noisy channel coding

4.1 Basic setting

In this chapter, we consider a channel with input alphabet $\mathcal{X}$ and output alphabet $\mathcal{Y}$. By default, the sets $\mathcal{X}$ and $\mathcal{Y}$ are going to be finite sets. The channel is described by a conditional probability distribution that we write W. For $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, $W(y|x)$ is the probability that the output of the channel is y when the input is x. When using random variables, we write $W_{Y|X}$ for W to indicate that X is the input random variable and Y is the corresponding output. The simplest example of channel is the identity channel for which $\mathcal{X} = \mathcal{Y}$ and $W(y|x) = \mathbf{1}_{x=y}$.

Our task is to communicate reliably over this noisy channel. Note that we assume we have full knowledge of the channel W. We can model this question by designing an encoding function E and a decoding function D that allows us to send M messages.

TODO: Figure

We introduce a useful shorthand notation: $[M] \stackrel{\text{def}}{=} \{1, \dots M\}$.

Definition 4.1.1. *An M-code for a channel W is a pair of functions $E : [M] \to \mathcal{X}$ and $D : \mathcal{Y} \to [M]$. The set $\{E(1), E(2), \dots, E(M)\}$ is called a codebook and an $E(s)$ for $s \in [M]$ is called a codeword.*

A property of interest of a given M-code is its error probability. The main notion of error probability we consider is the average error probability for the M-code (E, D) is given by

$$P_{err}(W, E, D) = 1 - \frac{1}{M} \sum_{s=1}^{M} \sum_{y \in \mathcal{Y}} W(y|E(s)) \mathbf{1}_{D(y)=s} \, .$$

It is often more intuitive to reason in terms of random variables so we introduce the probability space $(\Omega, \mathcal{P}(\Omega), \mathbf{P})$, with $\Omega = [M] \times \mathcal{X} \times \mathcal{Y} \times [M]$ and $\mathbf{P}$ defined by

$$\mathbf{P}\left\{(s, x, y, \hat{s})\right\} = \frac{1}{M} \mathbf{1}_{x=E(s)} W(y|x) \mathbf{1}_{\hat{s}=D(y)} \, , \tag{4.1}$$

for any $(s, x, y, \hat{s}) \in \Omega$. As Ω is discrete, this completely specifies $\mathbf{P}$. We then naturally define the random variables $S, X, Y, \hat{S}$. For example, $S : \Omega \to [M]$ with $S(s, x, y, \hat{s}) = s$ and similarly for $X, Y, \hat{S}$. In words, this corresponds to taking a message $S \in [M]$ uniformly at random, and

then defining $X = E(S)$, the Y is the output of the channel for input X and $\hat{S} = D(Y)$. In this notation, we have

$$P_{err}(W, E, D) = \mathbf{P}\left\{S \neq \hat{S}\right\} .$$

There are some other error probabilities one might be interested in such as

- We can ask for a worst case criterion in the sense that we look at the message s for which the error probability is the largest $P_{err,max}(W, E, D) = \max_{s\in[M]} \mathbf{P}\left\{\hat{S} \neq s | S = s\right\}$
- When the message set is the set of bitstrings of lengths k, i.e., $S \in \{0,1\}^k$, the bit error probability may be defined as $P_{err,bit}(W, E, D) = \frac{1}{k}\sum_{i=1}^{k} \mathbf{P}\left\{\hat{S}_i \neq S_i\right\}$.

Naturally, we would want M to be as large as possible and P_{err} to be as small as possible but there is a tension between them. Our objective is to understand this tradeoff between M and P_{err}. In particular, if we fix an acceptable error probability δ, how large can we make M? For that define

$$M^{opt}(W, \delta) = \max\{M : \text{there is an } M\text{-code } (E, D) \text{ with } P_{err}(W, E, D) \leq \delta\} .$$

Note that M is a number of messages, so $\lceil \log_2 M^{opt}(W, \delta) \rceil$ is the number of bits that can be sent with error probability at most δ.

Even though we are going to deal with the question in the case of a general channel, there is a very important special case to keep in mind. This corresponds to taking a channel W with input $\mathcal{X}$ and output $\mathcal{Y}$, such as the bit channel that flips a bit with probability f, and taking n independent copies of W. This gives a channel which we call W^n with input $\mathcal{X}^n$ and output $\mathcal{Y}^n$ defined by

$$W^n(y_1 \dots y_n | x_1 \dots x_n) = W(y_1|x_1)W(y_2|x_2)\cdots W(y_n|x_n) .$$

Now we are interested in the number of bits we can send per use of the channel, when n becomes large. In particular we are interested in $\lim_{n\to\infty} \frac{\log_2 M^{opt}(W,\delta)}{n}$ for small values of δ. We will see it is possible to characterize precisely this limit.

Examples

1. We start with the identity channel, with input $\mathcal{X} = [N]$ and output $\mathcal{Y} = [N]$, $W(y|x) = \mathbf{1}_{x=y}$. In this case, it is easy to exactly determine the tradeoff between M and P_{err}. In fact, for $M \leq N$, we can consider an M-code given by $E(s) = s$ for all $s \in [M]$ and $D(y) = y$ for all $y \in [M]$ and $D(y) = 1$ if $y > M$. In this case, $P_{err}(W, E, D) = 0$.

   If $M > N$, intuitively, we want to say that $P_{err} > 0$. For that, let us write

$$P_{err} = 1 - \frac{1}{M}\sum_{s=1}^{M}\sum_{y\in\mathcal{Y}} W(y|E(s))\mathbf{1}_{D(y)=s} \tag{4.2}$$

$$= 1 - \frac{1}{M}\sum_{y\in\mathcal{Y}}\sum_{s=1}^{M} W(y|E(s))\mathbf{1}_{D(y)=s} \tag{4.3}$$

$$= 1 - \frac{1}{M}\sum_{y\in\mathcal{Y}} W(y|E(D(y))) \tag{4.4}$$

$$\geq 1 - \frac{|\mathcal{Y}|}{M} . \tag{4.5}$$

So $P_{err} \geq 1 - \frac{N}{M}$. Note that in this calculation, we did not use any specifics of the identity channel: the error probability of an M-code is always lower bounded by $1 - \frac{|\mathcal{Y}|}{M}$. But for the identity channel, this bound can be achieved by some M-code. In fact, let $E(s) = s$ if $s \in [N]$ and otherwise set $E(s) = 1$. And set $D(y) = y$ for $y \in [N]$. Then $P_{err} = \frac{M-N}{M} = 1 - \frac{N}{M}$. So in other words, $M^{opt}(W, \delta) = \lfloor N/(1-\delta) \rfloor$.

Note that if $N = 2^n$, we can see the identity channel as n independent copies of the perfect bit channel. Then we have $\frac{\log_2 M^{opt}(W,\delta)}{n} \leq 1 + \frac{\log(\frac{1}{1-\delta})}{n}$ (and the lower bound is very close, the correction term is exponentially small in n).

2. Consider the channel with $\mathcal{X} = \{a, b, c\}$ and $\mathcal{Y} = \{0, 1\}$. We have $W(0|a) = 1, W(1|c) = 1$ and $W(0|b) = W(1|c) = \frac{1}{2}$. This was done in the HW.

3. The binary symmetric channel. We define the channel BSC_f as follows: for $x, y \in \{0, 1\}$, $\text{BSC}_f(y|x) = 1 - f$ if $x = y$ and $\text{BSC}_f(y|x) = f$ if $x \neq y$. As usual we consider n independent copies of this channel BSC_f^n. In the first lecture, we studied some codes for this channel. In particular, we had defined the repetition code which is a $2^{n/3}$-code. Recall we had $E(s_1 \dots s_{n/3}) = s_1 s_1 s_1 s_2 s_2 s_2 \dots s_{n/3} s_{n/3} s_{n/3}$ and $D(y_1 \dots y_n) = \text{maj}(y_1 y_2 y_3)\text{maj}(y_4 y_5 y_6) \dots$. We had analyzed the bit error probability and found that $P_{bit} = 3f^2 - 2f^3$ and thus $P_{err} = 1 - (1 - 3f^2 + 2f^3)^{n/3}$. In the regime f constant and n growing, P_{err} goes to 1 so this is not a very good code. Understanding the tradeoff between M and P_{err} for this channel is more complicated.

4.2 The information capacity

We now define an entropic quantity capture how good a channel is.

Definition 4.2.1 (Information capacity)**.** *The information capacity of a channel $W_{Y|X}$ is given by*

$$\text{C}(W) = \max_{P_X} I(X : Y) \,,$$

where the joint distribution of X, Y is defined by $P_{XY}(x, y) = P_X(x)W_{Y|X}(y|x)$. Notice that the maximization is over all distribution P_X over the set $\mathcal{X}$.

Let us compute this quantity for some channels.

1. The identity channel with $\mathcal{X} = \mathcal{Y} = [N]$. For any P_X, we have $I(X : Y) = H(X) - H(X|Y)$. But for the identity channel, we have $X = Y$ with probability 1. As a result, $H(X|Y) = 0$. So it suffices to take P_X to maximize $H(X)$. This gives $\text{C}(W) = \log N$.

2. Consider the channel with input alphabet $\mathcal{X} = \{a, b, c\}$ and output alphabet $\mathcal{Y} = \{0, 1\}$. Then if we try to take the distribution on the input to be uniform $P_X(x) = \frac{1}{3}$ for $x \in \mathcal{X}$.

Then using the fact that $P_{X|Y=0} = (\frac{2}{3}, \frac{1}{3}, 0)$, we have

$$\begin{aligned} I(X:Y) &= H(X) - H(X|Y) \\ &= \log_2 3 - h_2(\frac{1}{3}) \\ &= \log_2 3 - \frac{1}{3}\log_2 3 - \frac{2}{3}\log_2 \frac{3}{2} \\ &= \frac{2}{3}\,. \end{aligned}$$

But is this the largest we can obtain? If we instead take $P_X = (\frac{1}{2}, 0, \frac{1}{2})$, then we obtain

$$I(X:Y) = H(X) - H(X|Y) = 1\,.$$

And it fact, we cannot do better as $I(X:Y) = H(Y) - H(Y|X) \leq H(Y) \leq \log|\mathcal{Y}| = 1$.

3. Now consider the binary symmetric channel BSC_f. Let us write $P_X = (1-p, p)$ for $p \in [0,1]$. Then, $P_Y(0) = (1-p)(1-f) + pf$ and $P_Y(1) = p(1-f) + (1-p)f$. Moreover, $P_{Y|X=0} = (1-f, f)$. As a result,

$$\begin{aligned} I(X:Y) &= H(Y) - H(Y|X) \\ &= h_2((1-p)(1-f) + pf) - h_2(f)\,. \end{aligned}$$

Now observe that $h_2((1-p)(1-f)+pf) \leq 1$ with equality if $p = \frac{1}{2}$. We have shown that

$$\mathrm{C}(\text{BSC}_f) = 1 - h_2(f)\,.$$

4.3 Converse bounds

A converse bound is an upper bound on $M^{opt}(W, \delta)$.

Theorem 4.3.1. *Any M-code for W with error probability P_{err} satisfies*

$$\log_2 M \leq \frac{\mathrm{C}(W) + h_2(P_{err})}{1 - P_{err}}\,.$$

Proof Let us start with the proof for an M-code with $P_{err} = 0$. Given this code, we can define the probability space as in (4.1). We then write $\log_2 M$ in terms of an entropic quantity.

$$\begin{aligned} \log_2 M = H(S) &= I(S:\hat{S}) + H(S|\hat{S}) && (4.6) \\ &= I(S:\hat{S})\,, && (4.7) \end{aligned}$$

where we use the fact that $P_{err} = \mathbf{P}\left\{S \neq \hat{S}\right\} = 0$, and thus $H(S|\hat{S}) = 0$. Now we want to relate $I(S:\hat{S})$. For that we use the data processing inequality, saying that if $A \to B \to C$ forms a short Markov chain, then $I(A:B) \geq I(A:C)$. By construction, the random variables $S, X, Y, \hat{S}$ form

a short Markov chain $S \to X \to Y \to \hat{S}$. We thus have $I(S:\hat{S}) \leq I(S:Y) \leq I(X:Y)$. As a result, we obtain

$$\log_2 M \leq I(X:Y) \leq \mathrm{C}(W) \; .$$

We now need to handle the general case with P_{err} larger than 0. The intuition is simple, if P_{err} is small, then $H(S|\hat{S})$ is also small.

Lemma 4.3.2 (Fano's inequality)**.** *If $S \in [M]$ and $\hat{S}$ is such that $\mathbf{P}\left\{S \neq \hat{S}\right\} = \epsilon$. Then*

$$H(S|\hat{S}) \leq h_2(\epsilon) + \epsilon \log M \; .$$

Proof [of Fano's inequality] We introduce the random variable $E = 1$ if $S = \hat{S}$ and $E = 0$ if $S \neq \hat{S}$.

$$\begin{aligned} H(S|\hat{S}) &= H(ES|\hat{S}) - H(E|S\hat{S}) \\ &= H(ES|\hat{S}) \\ &= H(E|\hat{S}) + H(S|\hat{S}E) \; . \end{aligned}$$

Now $H(E|S) \leq H(E) = h_2(\epsilon)$. Moreover,

$$H(S|\hat{S}E) = P_E(0) H(S|\hat{S})_{P_{S\hat{S}|E=0}} + P_E(1) H(S|\hat{S})_{P_{S\hat{S}|E=1}} \; .$$

Note that $P_{S\hat{S}|E=1}(s,\hat{s}) = 0$ if $s \neq \hat{s}$, which implies that $H(S|\hat{S})_{P_{S\hat{S}|E=1}} = 0$. In addition, $P_E(0) = \epsilon$ and $H(S|\hat{S})_{P_{S\hat{S}|E=0}} \leq \log_2 M$, which gives the desired bound. □

To conclude the proof of Theorem 4.3.1, we simply use Fano's inequality in (4.6) and get

$$\log_2 M \leq \mathrm{C}(W) + h_2(P_{err}) + P_{err} \log_2 M \; ,$$

which leads to the desired bound. □

Let us now apply this converse bound to some examples of channels.

1. For the identity channel on $[N]$. This gives

$$\log_2 M \leq \frac{\log_2 N + h_2(P_{err})}{1 - P_{err}} \; . \tag{4.8}$$

   Let us compare this to the simple bound we obtained in (4.5). There we showed that for any M-code for the identity channel on $[N]$, we have $\frac{M}{N} \leq \frac{1}{1-P_{err}}$. Taking the logarithm, we obtain

$$\log_2 M \leq \log_2 N + \log_2 \left(\frac{1}{1 - P_{err}} \right) \; . \tag{4.9}$$

   Note that the bound (4.9) can be much better than (4.8), in particular if P_{err} is not close to 0. For example, if we are thinking of $P_{err} = \frac{1}{2}$, the bound (4.8) gives $\log_2 M \leq 2 \log_2 N + 2$,

whereas (4.9) gives $\log_2 M \leq \log_2 N + 1$. Note that the latter bound says that even if we allow an error probability of $\frac{1}{2}$, the number of bits that can be sent through the channel as at most 1 more than can be sent with very small error probability. For this reason it is called a strong converse. On the other hand the weak converse bound of (4.8) only says that if we allow an error probability of $\frac{1}{2}$, we cannot send more than twice the number of bits that could be sent with very small error.

For this very simple channel, proving a strong converse was easy but in general it can be much more difficult that the weak converse.

2. The binary symmetric channel $\text{BSC}_f^{\times n}$, we get that

$$\log M \leq \frac{\mathrm{C}(\text{BSC}_f^{\times n}) + h_2(P_{err})}{1 - P_{err}} .$$

We have computed $\mathrm{C}(\text{BSC}_f)$ for $n = 1$ but not for the n independent copies. For larger n, it seems harder to compute $\mathrm{C}(\text{BSC}_f^{\times n})$ as this is an optimization problem over distribution P_{X^n} on bitstrings of length n. However, as we will see next, C has a very interesting additivity property which makes it easy to compute for channels of the form $W^{\times n}$. It turns out that $\mathrm{C}(\text{BSC}_f^{\times n}) = n \cdot (1 - h_2(f))$. Thus, we get

$$\log M \leq \frac{n(1 - h_2(f)) + h_2(P_{err})}{1 - P_{err}} .$$

Theorem 4.3.3. *Given two channels W^i with inputs in $\mathcal{X}_i$ and outputs in $\mathcal{Y}_i$, for $i \in \{1, 2\}$. Define the channel W^{12} with inputs in $\mathcal{X}_1 \times \mathcal{X}_2$ and outputs in $\mathcal{Y}_1 \times \mathcal{Y}_2$ as $W^{12}(y_1y_2|x_1x_2) = W^1(y_1|x_1) \cdot W^2(y_2|x_2)$. Then*

$$\mathrm{C}(W^{12}) = \mathrm{C}(W^1) + \mathrm{C}(W^2) . \tag{4.10}$$

An immediate corollary of this theorem is that for any channel W, the information capacity of n copies of W is simply n times the information capacity of W:

$$\mathrm{C}(W^{\times n}) = n\mathrm{C}(W)$$

In particular for the binary symmetric channel, $\mathrm{C}(\text{BSC}_f^{\times n}) = n(1 - h_2(f))$ as advertized above.

Proof [of Theorem 4.3.3] We prove the equality (4.10) by establishing the two inequalities $\geq$ and $\leq$ separately.

Let us start with the easy direction, $\geq$. For any distributions P_{X_i} on $\mathcal{X}_i$ for $i \in \{1, 2\}$, we can define the joint distribution $P_{X_1X_2} = P_{X_1} \times P_{X_2}$ to be the product distribution. Applying the channel W^{12} to the random variable X_1X_2, we obtain random variables Y_1Y_2 as outputs. Note that using the fact that X_1 and X_2 are independent and by definition of the product channel W^{12}, we have $P_{X_1Y_1X_2Y_2} = P_{X_1Y_1} \times P_{X_2Y_2}$. As a result,

$$I(X_1X_2 : Y_1Y_2) = I(X_1 : Y_1) + I(X_2 : Y_2) . \tag{4.11}$$

Now observe that $I(X_1 : Y_1)$ is simply the mutual information between X_1 and the output of channel W^1 applied to X_1. As such, we have $\sup_{P_{X_1}} I(X_1 : Y_1) = \mathrm{C}(W^1)$ and $\sup_{P_{X_2}} I(X_2 :$

$Y_2) = \mathrm{C}(W^2)$. Now, as (4.11) is valid for any distributions P_{X_1} and P_{X_2}, we can take the supremum over than and get

$$\sup_{P_{X_1}, P_{X_2}} I(X_1X_2 : Y_1Y_2) = \mathrm{C}(W^1) + \mathrm{C}(W^2) \; . \tag{4.12}$$

Note that the difference between the left-hand side and the definition of $\mathrm{C}(W^{12})$ is that in $\mathrm{C}(W^{12})$ we optimize over all distributions $P_{X_1X_2}$ whereas in (4.12) we are only looking at product distributions of the form $P_{X_1} \times P_{X_2}$. So clearly

$$\mathrm{C}(W^{12}) \geq \mathrm{C}(W^1) + \mathrm{C}(W^2) \; .$$

Let us now move to the non-trivial direction, $\leq$. We now take a *general* distribution $P_{X_1X_2}$ which might not have product form. Then let Y_1Y_2 be the output of channel W^{12} on input X_1X_2. Note that Y_1 and Y_2 are in general not independent. Then,

$$\begin{aligned} I(X_1X_2 : Y_1Y_2) &= H(Y_1Y_2) - H(Y_1Y_2|X_1X_2) \\ &\leq H(Y_1) + H(Y_2) - H(Y_1Y_2|X_1X_2) \; , \end{aligned}$$

where we used the subadditivity of the entropy. Now to analyze the second term $H(Y_1Y_2|X_1X_2)$, we want to use the structure of the channel W^{12} as a product of the two channels $W^1 \times W^2$. Note that for any fixed x_1, x_2 conditioned on $X_1 = x_1$ and $X_2 = x_2$, the random variables Y_1 and Y_2 are independent, as

$$P_{Y_1Y_2|X_1X_2=x_1x_2}(y_1y_2) = W^1(y_1|x_1)W^2(y_2|x_2) \; .$$

As a result,

$$H(Y_1Y_2)_{P_{Y_1Y_2|X_1X_2=x_1x_2}} = H(Y_1)_{P_{Y_1|X_1=x_1}} + H(Y_2)_{P_{Y_2|X_2=x_2}} \; .$$

As a result,

$$\begin{aligned} I(X_1X_2 : Y_1Y_2) &\leq H(Y_1) + H(Y_2) - \sum_{x_1 \in \mathcal{X}_1} P_{X_1}(x_1) H(Y_1)_{P_{Y_1|X_1=x_1}} - \sum_{x_2 \in \mathcal{X}_2} P_{X_2}(x_2) H(Y_2)_{P_{Y_2|X_2=x_2}} \\ &= I(X_1 : Y_1) + I(X_2 : Y_2) \\ &\leq \mathrm{C}(W^1) + \mathrm{C}(W^2) \; . \end{aligned}$$

We conclude by taking the supremum over all distributions $P_{X_1X_2}$. □

In other words, this theorem is saying that for product channels $W^1 \times W^2$, an optimal choice of input distribution to maximize the mutual information is to take the product of an optimal distribution for W^1 and an optimal distribution for W^2.

4.4 Achievability bounds

Remember the surprisal random variable $h_X(X) = -\log_2 P_X(X)$. We say this was the relevant random variable for data compression and its expectation is the Shannon entropy $\mathbf{E}\{h_X(X)\} = H(X)$. For channel coding, the relevant random variable should be related to the correlation between the input and output. We introduce a random variable whose expectation will be the mutual information.

Definition 4.4.1 (Mutual information density)**.** *For random variables $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, we define the mutual information density as*

$$\begin{aligned} i_{XY}(x:y) &= \log_2 \frac{P_{Y|X}(y|x)}{P_Y(y)} \\ &= \log_2 \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} , \end{aligned}$$

provided $P_{XY}(x,y) \neq 0$. If $P_{XY}(x,y) = 0$, we set $i_{XY}(x:y) = -\infty$ if $P_X(x)P_Y(y) > 0$ and if $P_X(x)$ or $P_Y(y) = 0$ then $i_{XY}(x:y) = +\infty$ (this choice is arbitrary but this case is not interesting anyway because x or y have zero probability).

Observe that

$$\begin{aligned} \mathbf{E}\{i_{XY}(X:Y)\} &= \sum_{x,y} P_{XY}(x,y) i_{XY}(x:y) \\ &= I(X:Y) . \end{aligned}$$

Let us consider an example. Assume $X_1 \dots X_n \in \{0,1\}^n$ are n uniform and independent bits. Then assume we apply a binary symmetric channel to each bit independently, i.e., we flip each bit with probability f, getting the random variables $Y_1 \dots Y_n$.

Let us compute $i_{X^nY^n}(x^n : y^n)$ for these random variables.

$$\begin{aligned} i_{X^nY^n}(x^n : y^n) &= \log_2 \frac{P_{Y^n|X^n}(y^n|x^n)}{P_{Y^n}(y^n)} \\ &= \log_2 \frac{\prod_{i=1}^n (1-f)^{x_i \oplus y_i} f^{x_i \oplus y_i \oplus 1}}{2^{-n}} \\ &= \log_2 \frac{(1-f)^{n-d_H(x^n,y^n)} f^{d_H(x^n,y^n)}}{2^{-n}} \\ &= n + (n - d_H(x^n,y^n)) \log_2(1-f) + d_H(x^n,y^n) \log_2 f , \end{aligned}$$

where d_H is the Hamming distance between strings. If we fix $x^n \in \{0,1\}^n$, this quantity is the largest when $y^n = x^n$ and decreases as the Hamming distance between them increases.

Theorem 4.4.2. *Let W be a channel with input alphabet $\mathcal{X}$ and output alphabet $\mathcal{Y}$. For any distribution P_X on $\mathcal{X}$, define the distribution P_{XY} on $\mathcal{X} \times \mathcal{Y}$ by $P_{XY}(x,y) = P_X(x)W(y|x)$. Then for any $\tau > 0$, there exists an M-code for W with*

$$P_{err} \leq \mathbf{P}\{i_{XY}(X:Y) \leq \log M + \tau\} + 2^{-\tau} .$$

This is roughly saying that if we look at the random variable $i_{XY}(X:Y)$, we can roughly send $\log M$ bits of information with error probability δ whenever $\mathbf{P}\{i_{XY}(X:Y) \leq \log M\} \leq \delta$. If we have a channel such that $i_{XY}(X:Y)$ is close to its expectation with high probability, then we can take $\log M \approx I(X:Y)$. We will discuss such a case in more detail after proving the theorem.

Proof This is an achievability proof so we need to construct an M-code, by describing E and D. Let us for now keep E arbitrary and we will choose it later. We try to determine first how to choose D if E is fixed. We can write the error probability as

$$\begin{aligned} P_{err} &= 1 - \frac{1}{M}\sum_{s=1}^{M}\sum_{y\in\mathcal{Y}} W(y|E(s))\mathbf{1}_{D(y)=s} \\ &= 1 - \frac{1}{M}\sum_{y\in\mathcal{Y}} W(y|E(D(y)))\ . \end{aligned}$$

To minimize this quantity, we want to choose $D(y)$ to maximize $W(y|E(D(y)))$. So we may define $D^*(y) = argmax_{s\in[M]}W(y|E(s))$. If we choose the decoder D^*, then the error probability becomes

$$P_{err} = 1 - \frac{1}{M}\sum_{y\in\mathcal{Y}} \max_{s\in[M]} W(y|E(s))\ .$$

This expression is not so easy to analyze. So we will consider another decoder that is easier and not much worse. The idea is to replace the maximum by a cutoff value: if there is an s such that $W(y|E(s)) \geq t$, then we return such an s and otherwise we simply abort. This threshold t will basically be $M \cdot P_Y(y)$ where M is the number of messages and $P_Y(y) = \sum_x P_X(x)W(y|x)$. Recall that P_X is the distribution that we have in the statement of the theorem. By taking the logarithm, this is equivalent to the condition that $i_{XY}(E(s) : y) \geq \log M + \tau$, where $P_{XY}(x,y) = P_X(x)W(y|x)$. More precisely, let us define

$$D(y) = \begin{cases} s & \text{if there is a unique } s : i_{XY}(E(s):y) \geq \log M + \tau \\ x_0 & \text{otherwise.} \end{cases}$$

Let us now analyze the error probability for this D and try to see which properties of E we would like to have. We have $P_{err} = \frac{1}{M}\sum_s P_{err,s}$ where $P_{err,s}$ is the error probability if the message to be sent is s. For a given s, the probability that it is decoded incorrectly can be written as

$$P_{err,s} = \sum_{y\in\mathcal{Y}} W(y|E(s))\mathbf{1}_{D(y)\neq s} \tag{4.13}$$

$$\leq \sum_{y\in\mathcal{Y}} W(y|E(s))\mathbf{1}_{i_{XY}(E(s):y)<\log M+\tau \text{ OR } \exists s'\neq s: i_{XY}(E(s'):y)\geq \log M+\tau} \tag{4.14}$$

$$\leq \sum_{y\in\mathcal{Y}} W(y|E(s))\mathbf{1}_{i_{XY}(E(s):y)<\log M+\tau} + \sum_{y\in\mathcal{Y}} W(y|E(s)) \sum_{s'\neq s} \mathbf{1}_{i_{XY}(E(s'):y)\geq \log M+\tau}\ . \tag{4.15}$$

To have more intuition, it might be helpful to just write the above calculation in terms of random variables. We consider the random variables $SY\hat{S}$ defined by $P_{SY\hat{S}}(s,y,\hat{s}) =$

$\frac{1}{M}W(y|E(s))\mathbf{1}_{D(y)=\hat{s}}$.

$$\begin{aligned}
P_{err,s} &= \mathbf{P}\left\{\hat{S} \neq S | S = s\right\} \\
&\leq \mathbf{P}\left\{i_{XY}(E(s):Y) < \log M + \tau \text{ OR } \exists s' \neq s : i_{XY}(E(s'):Y) \geq \log M + \tau | S = s\right\} \\
&\leq \mathbf{P}\left\{i_{XY}(E(s):Y) < \log M + \tau | S = s\right\} + \sum_{s' \neq s} \mathbf{P}\left\{i_{XY}(E(s'):Y) \geq \log M + \tau | S = s\right\} \ ,
\end{aligned}$$

where we used a union bound in the last step. Note that these probabilities are only over Y which is drawn from the distribution given by $W(.|E(s))$. Let us try to understand what these terms are for the channel W being n copies of the binary symmetric channel with flip probability f. Let us also think of X as uniformly distributed on bitstrings on length n. Recall that in this case

$$i_{XY}(E(s):Y) = n + (n - d_H(E(s), Y)) \log_2(1-f) + d_H(E(s), Y) \log_2 f \ .$$

As Y corresponds to $E(s)$ with bits independently flipped with probability f, we have with high probability $d_H(E(s), Y) \approx fn$. So

$$i_{XY}(E(s):Y) \approx n + n\left((1-f)\log_2(1-f) + f\log_2 f\right) = n(1 - h_2(f)) \ .$$

Note that doesn't even depend on the choice of $E(s)$ we make.

Now what if we look at the second term. Y is still obtained by flipping bits of $E(s)$ at random and we look at $i_{XY}(E(s'):Y)$. It is now

$$i_{XY}(E(s'):Y) = n + (n - d_H(E(s'), Y)) \log_2(1-f) + d_H(E(s'), Y) \log_2 f \ .$$

For this quantity to be small, we would like $E(s')$ to be far from $E(s)$ so that $d_H(E(s'), Y) \gg d_H(E(s), Y)$ and so $i_{XY}(E(s'):Y) \ll i_{XY}(E(s):Y)$. For example, if we choose $E(s) = 0^n$ and $E(s') = 1^n$, then we expect $d_H(E(s), Y) \approx fn$ and $d_H(E(s'), Y) \approx (1-f)n$. For two codewords, it is relatively easy to choose them far apart, but this shows what we want to do in general, choose $E(s)$ so that for any pair $E(s)$ and $E(s')$ are far apart from each other.

A good way of achieving this is by choosing $E(1), \ldots E(M)$ at random and independent from the distribution P_X. Then it becomes easy to analyze the expectation. Let us now analyze the error probability terms in (4.15). The first term is

$$\begin{aligned}
\mathop{\mathbf{E}}_{E(s)\sim P_X}\left\{\sum_{y\in\mathcal{Y}} W(y|E(s))\mathbf{1}_{i_{XY}(E(s):y)<\log M+\tau}\right\} &= \sum_{x\in\mathcal{X}, y\in\mathcal{Y}} P_X(x)W(y|x)\mathbf{1}_{i_{XY}(x:y)<\log M+\tau} \\
&= \mathbf{P}\left\{i_{XY}(X:Y) < \log M + \tau\right\} \ .
\end{aligned}$$

Now let us look at the second term for $s \neq s'$,

$$\begin{aligned}
&\mathop{\mathbf{E}}_{E(s)\sim P_X, E(s')\sim P_X} \left\{ \sum_{y\in\mathcal{Y}} W(y|E(s)) \mathbf{1}_{i_{XY}(E(s'):y)\geq \log M+\tau} \right\} \\
&= \sum_{x,x'\in\mathcal{X}, y\in\mathcal{Y}} P_X(x)P_X(x')W(y|x)\mathbf{1}_{i_{XY}(E(s'):y)\geq \log M+\tau} \\
&= \sum_{x,x'\in\mathcal{X}, y\in\mathcal{Y}} P_X(x)P_X(x')W(y|x)\mathbf{1}_{W(y|x')\geq P_Y(y)(M2^{\tau})} \\
&\leq \sum_{x'\in\mathcal{X}, y\in\mathcal{Y}} P_X(x')\frac{2^{-\tau}}{M}W(y|x')\mathbf{1}_{W(y|x')\geq P_Y(y)(M2^{\tau})} \\
&\leq \frac{2^{-\tau}}{M} .
\end{aligned}$$

As a result, getting back to (4.15), we get back for all s, in expectation over the choice of the codebook

$$\begin{aligned}
\mathbf{E}\left\{P_{err,s}\right\} &\leq \mathbf{P}\left\{i_{XY}(X:Y) < \log M + \tau\right\} + (1 - \frac{1}{M})2^{-\tau} \\
&\leq \mathbf{P}\left\{i_{XY}(X:Y) < \log M + \tau\right\} + 2^{-\tau} ,
\end{aligned}$$

as advertised. □

Important special case: memoryless channels As we said, an important special class of channels are memoryless channels that have the form $W^{\times n}$. In this case, an M-code is said to have rate $\frac{\log_2 M}{n}$, which we interpret as the number of bits of information we can send per channel use. In this case, the encoder takes a message in $[M]$ and maps it to a codeword in $\mathcal{X}^n$ and the decoder takes an element in $\mathcal{Y}^n$ and maps it to a message in $[M]$. The examples of codes we saw in the first lecture looked at blocks of size 1 or 3 and encoded each block independently. This gave use the impression that if we want the error probability to go to zero, then we need to take the rate $\frac{\log_2 M}{n}$ to 0. Shannon's theorem below says that we can do much better, in fact for any nontrivial channel, we can code at a nonzero rate with an error probability going to 0 as $n \to \infty$.

Theorem 4.4.3 (Shannon's noisy coding theorem)**.** *Let W be a channel. Recall that the information capacity* $\mathrm{C}(W) = \max_{P_X} I(X:Y)$*. Then for any $\delta > 0$,*

$$\mathrm{C}(W) \leq \lim_{n\to\infty} \frac{\log_2 M^{opt}(W^{\times n}, \delta)}{n} \leq \frac{\mathrm{C}(W)}{1-\delta} .$$

Proof Let P_X be a distribution achieving the maximum defining $\mathrm{C}(W)$. Let $X_1, \ldots, X_n$ be independent copies of X and $Y_1, \ldots, Y_n$ be the corresponding outputs of the channels. Then the pairs (X_i, Y_i) are mutually independent, so

$$i_{X^nY^n}(X^n : Y^n) = \sum_{i=1}^{n} i_{XY}(X_i : Y_i) .$$

This is a sum of iid random variables and by the law of large numbers, for any $\epsilon > 0$,

$$\lim_{n\to\infty} \mathbf{P}\left\{i_{X^nY^n}(X^n : Y^n) \leq n(I(X : Y) - \epsilon)\right\} = 0 \,.$$

Take $M = \left\lfloor 2^{n(I(X:Y)-2\epsilon)} \right\rfloor$ and $\tau = n\epsilon$ so that

$$\mathbf{P}\left\{i_{X^nY^n}(X : Y) \leq \log M + \tau\right\} + 2^{-\tau} \leq \mathbf{P}\left\{i_{X^nY^n}(X^n : Y^n) \leq n(I(X : Y) - \epsilon)\right\} + 2^{-\epsilon n} \,,$$

which is smaller than δ for sufficiently large n. As a result, for any $\epsilon > 0$, we have for large enough n,

$$\frac{\log_2 M^{opt}(W^{\times n}, \delta)}{n} \geq I(X : Y) - 2\epsilon \,.$$

□

Some comments on the theorem:

- The $1 - \delta$ in the upper bound is only an artifact of the proof. It turns out one can prove that for any $\delta > 0$, $\lim_{n\to\infty} \frac{\log_2 M^{opt}(W^{\times n},\delta)}{n} = \mathrm{C}(W)$. This means that there is a sharp threshold that happens at a rate given by $\mathrm{C}(W)$. Below it, the probability can be made arbitrarily small and above it the error probability gets close to 1.
- One can obtain quite good finite n bounds. It turns out that $\frac{\log_2 M^{opt}(W^{\times n},\delta)}{n} = \mathrm{C}(W) + \frac{Q_\delta}{\sqrt{n}} + O(\frac{\log n}{n})$.
- Proof is not explicit, does not give you a good code. Even to specify the codebook, need exponential space in n. Also, in order to use this in practice, we would like the encoding and decoding function to be efficiently computed as a function of n. Since Shannon's theorem, an important focus in information theory theory is to find explicit codes with efficient encoding and decoding that come close to capacity. This is the study of error correcting codes. That will be the topic of the second part of the course.

Before concluding this chapter, an important point to comment on is that we always allowed a probability of error in the transmission. What happens if I ask for a zero error probability. Take for example our favorite example, n copies of a binary symmetric channel. For such a channel, there is no way to send even a single bit without error. For any n and $f \in (0, 1)$,

$$M^{opt}(\mathrm{BSC}_f^{\times n}, 0) = 1 \,.$$

In fact, for any choice of $E(1)$ and $E(2)$, both codewords can be mapped to 0^n with nonzero probability which shows there has to be some error. Another example would be the channel with inputs and output $1, 2, 3$ and $W(i|i) = W(i + 1|i) = \frac{1}{2}$ for $i \in \{1, 2\}$ and $W(3|3) = 1$. Note that in this case, 1 and 2 cannot be both codewords and similarly 2 and 3 cannot be codewords at the same time. However 1 and 3 can be codewords together. For the zero-error setting, the relevant parameter of a channel to consider is the so-called confusability graph $G(W)$ of W. The set of vertices is indexed by the inputs of the channel and we have an edge (x, x') if there exists a $y \in \mathcal{Y}$ such that $W(y|x) > 0$ and $W(y|x') > 0$. This means that if both x and x' are used as

codewords there is a chance of confusing them so they cannot both be codewords if we are aiming for zero error probability. Then M^{opt} is given by the maximum independent set of the graph $G(W)$,$M^{opt}(W,0) = |MIS(G(W))|$. As for the case with nonzero error, we can take the setting of a memoryless channel and compute the asymptotic rate: $\frac{\log_2 M^{opt}(W^{\times n},0)}{n}$. Can we find a simple characterization of the limit as $n \to \infty$ as in the case with nonzero error? Note that this corresponds to computing the maximum independent set in the graph product $G(W^{\times n}) = G(W)^{\otimes n}$, defined by $(x_1, \ldots, x_n) \sim (x'_1, \ldots, x'_n)$ iff $x_1 \sim x'_1, \ldots, x_n \sim x'_n$. This turns out to be quite complicated. Even for C_5, the cycle of length 5, it took about 30 years to prove it. It is clear that $M^{opt}(C_5, 0) = 2$. It is clear that $M^{opt}(C^{\times n}, 0) \geq 2^n$, by simply choosing the product of the independent set. But it turns out one can find a larger one: $M^{opt}(C_5^{\times 2}, 0) = 5$. The hard part is then to show that as one takes $2n$ copies of the graph, there are no independent sets of size $\geq 5^n$. This is shown via semidefinite programming. Then one has

$$\lim_{n\to\infty} \frac{\log_2 M^{opt}(C_5^{\times n}, 0)}{n} = \frac{1}{2} \log 5 \, .$$

Note that this limit for C_7 is not known.

Chapter 5

Information theory and combinatorics

For counting objects, the properties of the Shannon entropy is sometimes useful. We give two simple examples. Both use a simple inequality about the entropy of a collection of random variables.

Lemma 5.0.1. *If $S_1, \ldots, S_m \subseteq [n]$ and for every $i \in [n]$ appears at least k times, then*

$$k \cdot H(X_1 \ldots X_n) \leq \sum_{i=1}^{m} H(X_{S_i})$$

Proof We write $H(X_1 \ldots X_n) = H(X_1) + H(X_2|X_1) + \cdots + H(X_n|X_1 \ldots X_{n-1})$. Then for each $S_j = \{e_j(1), \ldots, e_j(|S_j|)\}$. We have

$$\begin{aligned} H(X_{S_j}) &= H(X_{e_j(1)}) + H(X_{e_j(2)}|X_{e_j(1)}) + \ldots \\ &\geq H(X_{e_j(1)}|X_1 \ldots X_{e_j(1)-1}) + H(X_{e_j(2)}|X_1 \ldots X_{e_j(2)-1}) + \ldots \ . \end{aligned}$$

As for each $i \in [n]$, i appears in at least k sets, then the term $H(X_i|X_1 \ldots X_{i-1})$ appears at least k times and we get the desired result. □

5.1 Projection of point sets

Suppose we have m points $S = \{a(1), \ldots, a(m)\} \in \mathbb{R}^3$. We write the coordinates of point $a(i) = (a_1(i), a_2(i), a_3(i))$. Suppose we only know the sizes of the projection sets on XY, XZ and YZ planes, i.e.,

$$\begin{aligned} \Pi_{XY} &= \{(a_1(i), a_2(i)), i \in [m]\} \\ \Pi_{YZ} &= \{(a_2(i), a_3(i)), i \in [m]\} \\ \Pi_{XZ} &= \{(a_2(i), a_3(i)), i \in [m]\} \ . \end{aligned}$$

Assume that $|\Pi_{XY}|, |\Pi_{YZ}|, |\Pi_{XZ}| \leq n$. How large can m be? Let us consider an example, if the points form a $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ grid, then the conditions are satisfied and $m = n^{3/2}$. We will show that this is the largest m can be.

Proposition 5.1.1. *If a S of m points in $\mathbb{R}^3$ has all projections of size at most n. Then $m \leq n^{3/2}$.*

Proof Define random variables $A_1A_2A_3$ obtained by choosing the coordinates of a random point in S, i.e.,

$$P_{A_1A_2A_3}(a_1a_2a_3) = \begin{cases} \frac{1}{m} & \text{if } a_1a_2a_3 = a_1(i)a_2(i)a_3(i) \\ 0 & \text{otherwise} \end{cases} .$$

We have $H(A_1A_2A_3) = \log m$. The condition $|\Pi_{XY}| \leq n$ says that $H(A_1A_2) \leq \log n$. Similarly $H(A_2A_3), H(A_1A_3) \leq n$. Then using Shearer's lemma, we get

$$\log m = H(A_1A_2A_3) \leq \frac{1}{2}(H(A_1A_2) + H(A_1A_3) + H(A_2A_3)) \leq \frac{1}{2}3\log n ,$$

and so $m \leq n^{3/2}$. □

5.2 Number of independent sets in a graph

How many independent sets can there be in a graph on n vertices? Let us restrict to d-regular graphs. If $d = 1$, the graph is just a perfect matching so for each pair you have 3 choices: either nothing, vertex 1 or vertex 2, so the number of independent sets is $3^{n/2}$. If $d = 2$, then there are more possibilities for the graph. If we take a graph that is a union of cycles with 4 vertices, then we get 7 independent sets for each cycle and there are $n/4$ cycles so we get $7^{n/4}$ independent sets.

Theorem 5.2.1. *If G is a bipartite with n vertices, then the number of independent sets of G is at most $(2^{d+1} - 1)^{\frac{n}{2d}}$.*

For simplicity of the proof, we restricted our attention to bipartite graphs, the statement is also true in general. Also note that this bound is achieved by some graph. If you take $\frac{n}{2d}$ copies of the complete bipartite graph with d vertices in each side, you obtain exactly this many independent sets.

Proof We assume the vertices of G are labelled by $[n] := \{1, \ldots, n\}$ and let A and B two subsets of $[n]$ with edges only between A and B. We assume moreover that $|A| \geq |B|$. We write $\mathcal{I}(G)$ as the vertex set of G. Let I be a uniformly random independent set in $\mathcal{I}(G)$. We can specify I by specifying whether for each vertex v, $v \in I$ or not. For $v \in [n]$, let $X_v = \mathbf{1}_{v \in I}$. As a result,

$$H(X_1 \ldots X_n) = H(I) = \log |\mathcal{I}(G)| .$$

We now decompose the entropy into parts:

$$H(X_1 \ldots X_n) = H(X_A) + H(X_B|X_A) .$$

Let us start with the second term. We have

$$\begin{aligned} H(X_B|X_A) &\leq \sum_{b \in B} H(X_b|X_A) \\ &\leq \sum_{b \in B} H(X_b|X_{N(b)}) , \end{aligned}$$

where $N(b) = \{a \in A : (a,b) \in E(G)\}$ is the neighbourhood of b. We want to still simplify this again by just conditioning on whether $N(b)$ has any vertex in the independent set. For that, introduce $Q_b = 1$ if $|I \cap N(b)| = \emptyset$ and $Q_b = 0$ otherwise.

$$\begin{aligned} H(X_B|X_A) &\leq \sum_{b\in B} H(X_b|Q_b) \\ &\leq \sum_{b\in B} P_{Q_b}(0)\cdot H(P_{X_b|Q_b=0}) + P_{Q_b}(1)\cdot H(P_{X_b|Q_b=1}) \ . \end{aligned}$$

Now if $Q_b = 0$, $b \notin I$ and so $X_b = 0$ and $H(P_{X_b|Q_b=0}) = 0$. We also have $H(P_{X_b|Q_b=1}) \leq 1$. As a result, write $q_b = P_{Q_b}(1)$, we get

$$H(X_B|X_A) \leq \sum_{b\in B} q_B \ .$$

We do not know the value of q_b and in fact it depends on the structure of the graph on which we do not want to make assumptions. Our objective now is to find an upper bound on $H(X_A)$ depending on q_b. For that we use Shearer's lemma: considering the sets $N(b)$ for $b \in B$, each element $a \in A$ appears in exactly d such sets using the fact that the graph is regular. As a result,

$$H(X_A) \leq \frac{1}{d}\sum_{b\in B} H(X_{N(b)}) \ .$$

Now we want to relate $H(X_{N(b)})$ to q_b which is the probability that $|I \cap N(b)| = \emptyset$. For that observe that $H(X_{N(b)}) = H(X_{N(b)}Q_b)$ as Q_b can be determined from $X_{N(b)}$. Then we can write

$$\begin{aligned} H(X_{N(b)}Q_b) &= H(Q_b) + H(X_{N(b)}|Q_b) \\ &= h_2(q_b) + q_b H(P_{X_{N(b)}|Q_b=1}) + (1-q_b)H(P_{X_{N(b)}|Q_b=0}) \quad \leq h_2(q_b) + (1-q_b)\log(2^d-1) \ , \end{aligned}$$

where we used the fact that if $Q_b = 1$, then $X_{N(b)} = 0\ldots0$ and if $Q_b = 0$, then $X_{N(b)}$ can take all possible values except $0\ldots0$ and there are $2^d - 1$ such possibilities. As a result, we get

$$\begin{aligned} \log|\mathcal{I}(G)| = H(X_1\ldots X_n) &\leq \frac{1}{d}\sum_{b\in B} h_2(q_b) + (1-q_b)\log(2^d-1) + \sum_{b\in B} q_b \\ &= \frac{|B|}{d}\log(2^d-1) + \frac{1}{d}\sum_{b\in B} h_2(q_b) + q_b \log\frac{2^d}{2^d-1} \ . \end{aligned}$$

If we differentiate this with respect to q_b, the maximum is for $q_b = \frac{2^d}{2^{d+1}-1}$ which leads to

$$\begin{aligned} \log|\mathcal{I}(G)| &\leq \frac{|B|}{d}\log(2^d-1) + \frac{1}{d}\sum_{b\in B} h_2\left(\frac{2^d}{2^{d+1}-1}\right) + \frac{2^d}{2^{d+1}-1}\log\frac{2^d}{2^d-1} \\ &= \frac{n}{2d}\log(2^{d+1}-1) \ , \end{aligned}$$

which concludes the proof of the desired result. □

Chapter 6

Error-correcting codes

Shannon's theorem says that for any nontrivial channels there are M-codes with $M \approx 2^{n\mathrm{C}(W)}$ codewords that can be decoded with very small error probability given the output of the channel W. It even said that provided we pick the codewords at random with a good distribution, then *most* codes are good. Our objective now is to explicitly construct good codes.

The notion of a good code depends on the channel being studied and involves both the construction of an encoder and a decoder. To simplify the study it is useful to consider a different error model than the one we considered so far and in this model the existence of a decoder is directly related to a simple property of the codebook. Recall that in the Shannon model, an encoder is good if there exists a decoder that can decode with a small error probability. In the Hamming model, a good encoder is one for which there is a decoder that can correct any error of weight at most t. The models are not exactly the same but they are related and we will see that it is possible to construct good codes in the Shannon sense using good codes in the Hamming sense.

6.1 General error-correcting codes

Definition 6.1.1. *A code C of blocklength n over an alphabet Σ is a subset of Σ^n. We usually write $q = |\Sigma|$.*

The dimension of a code is defined as $k = \log_q |C|$.

Remark. Note that a way to specify a code is as an injective encoding function $C : \Sigma^k \to \Sigma^n$ and the code corresponds to the image of the encoding function C. Even though they are not the same objects, we will be using the word "code" for both of these.

As mentioned before, we consider the Hamming error model where our objective is to be able to correct all errors of weight at most t. Note that if you want to think it terms of channels, you should see $\mathcal{X} = \mathcal{Y} = \Sigma$ and then taking n copies of the channel for example.

Definition 6.1.2. *C is t-error correcting if there exists a decoding map $D : \Sigma^n \to C$ such that for any $c \in C$ and any error pattern e with at most t errors $D(c + e) = c$.*

Let us look at simple examples

1. The repetition code $C_{rep} = \{000, 111\}$. This code has $q = 2, n = 3, k = 1$. It is 1-error correcting. In fact, my decoding function can map to 000 inputs of weight at most 1 and map to 111 inputs of weight ≥ 2.

2. The binary code defined by $C_{\oplus}(x_1x_2) = x_1x_2(x_1 \oplus x_2)$ has $q = 2, n = 3, k = 2$. It is not 1-error correcting. In fact $C_{\oplus}(00) = 000$ and $C_{\oplus}(01) = 011$. If I apply a weight 1 error to the first codewords I can get 010, but I can also get to 010 by applying a weight 1 error to the second codeword. So I can detect that there is an error but I cannot correct for it.

From this example, one sees that the relevant parameter that governs how many errors a code can correct is the Hamming distance between the codewords.

Definition 6.1.3 (Minimum distance of a code)**.** *The Hamming distance between $u, v \in \Sigma^n$ is defined by $\Delta(u, v) = |\{i \in [n] : u_i \neq v_i\}|$.*

The minimum distance (or just distance) of a code C is defined as

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

Note that in the Hamming distance, we do not have a notion of distance between two symbols in Σ, they are either the same or different. For example, if we think of $\Sigma = \{0, 1\}$ and consider the bitstrings $u = 0010$ and $v = 1110$, their Hamming distance is 2. However, if we consider $\Sigma = \{0, 1\}^2$ and consider $u, v \in \Sigma^2$, then their Hamming distance is 1.

Let us look at the examples we considered before

1. The repetition code C_{rep} has a minimum distance of 3

2. The code $C_{\oplus}$ has a minimum distance of 2. In fact, take two different codewords $c = C_{\oplus}(x_1x_2)$ and $c' = C_{\oplus}(y_1y_2)$. Then if $\Delta(x_1x_2, y_1y_2) = 2$, then $\Delta(c, c') \geq 2$. Otherwise, if $\Delta(x_1x_2, y_1y_2) = 1$, then $\Delta(c, c') = 2$.

We now see that minimum distance is directly related to the number of errors that can be corrected. We only do here the special case of d odd, the even case will be done in the tutorial.

Proposition 6.1.4. *Assume $d \geq 3$ is odd. Then the following are equivalent.*

- *C has minimum distance d*
- *C can correct $\frac{d-1}{2}$ errors*

Proof Suppose C has minimum distance d. Then define the function $D : \Sigma^n \to C$ by $D(y) = argmin_{c \in C} \Delta(c, y)$. Then suppose c_1 is transmitted and $\Delta(c_1, y) \leq t$. Then let $D(y) = c$. We have $\Delta(c_1, c) \leq \Delta(c_1, y) + \Delta(y, c) \leq t + t$. This is equal to $2d$ provided $t = \frac{d-1}{2}$. As such $c = c_1$.

Now suppose C has distance $\leq d - 1$. Then there exists $c_1, c_2 \in C$ with $\Delta(c_1, c_2) \leq d - 1$. Consider y such that $\Delta(y, c_1), \Delta(y, c_2) \leq \frac{d-1}{2}$. This y could be received for either c_1 or c_2 so C cannot correct $\frac{d-1}{2}$ errors. □

Notation: We use the notation $(n, k, d)_q$-code when blocklength n, dimension k, minimum distance d and the alphabet Σ has size q.

Let us see another less trivial code that we have already encountered in the first lecture. This is the Hamming code. It is also a binary code with $q = 2$. We may define it by

$$C_H(x_1x_2x_3x_4) = (x_1, x_2, x_3, x_4, x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_2 \oplus x_3 \oplus x_4) .$$

This is a $(7, 4, d)_2$ code where we still have to determine d. I claim that the minimum distance is 3. First $0000000 \in C_H$ and $1000110 \in C_H$ and they are at distance 3. Moreover, for two different codewords $C_H(x)$ and $C_H(y)$, we can write

$$\begin{aligned}\Delta(C_H(x), C_H(y)) &= |\{i \in [7] : C_H(x)_i \neq C_H(y)_i\}| \\ &= |\{i \in [7] : C_H(x)_i + C_H(y)_i \neq 0\}| \\ &= |C_H(x) + C_H(y)| \\ &= |C_H(x + y)| \ ,\end{aligned}$$

as the mapping C_H is a *linear* map. So it suffices to determine $\min_{x \neq 0} |C_H(x)|$. We do this by considering the different cases for the Hamming weight of x. If $|x| = 1$, then two or three of the following bits evaluate to 1: $x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_2 \oplus x_3 \oplus x_4$. If $|x| = 2$, then at least one of these bits evaluates to 1 and if $|x| = 3$, we already have $|C_H(x)| \geq 3$. We conclude that C_H is a $(7, 4, 3)_2$ code.

Note that this code has a very nice property that we will be exploiting further. The encoding function is a linear function. In fact, we can see messages as elements of $\mathbb{F}_2^4$ and codewords as elements of $\mathbb{F}_2^7$ and the transformation is given by a matrix

$$G_H = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

and $C_H(x) = xG_H$ where we see x as a row vector in $\mathbb{F}_2^4$. One can in general define linear codes whenever Σ has a field structure so that Σ^n is a vector space over the field Σ and $C \subseteq \Sigma^n$ is a subspace. Before getting into the detailed study of linear codes, let us determine some simple bounds on the best parameters one can achieve for codes.

6.1.1 General bounds on the best codes

For a fixed n and q, we would like k and d to be as large as possible. For example, the Hamming code is a $(7, 4, 3)_2$ code, is it possible to improve it to a $(7, 5, 3)_2$ code for example? The answer is no by the following simple packing bound. Again, we only state here a simplified for with $q = 2$ and $d = 3$ but it is easy to generalize (see tutorial).

Theorem 6.1.5 (Hamming bound (special case))**.** *Every binary code with blocklength n, dimension k and distance $d = 3$ satisfies*

$$k \leq n - \log_2(n + 1) \ .$$

For $n = 7$ and $d = 3$, this gives $k \leq 4$, which means the Hamming code is optimal in this sense.

Proof Let C be such a code and c_1, c_2 be two codewords. For $u \in \{0, 1\}^n$, let $B(u, 1) = \{v \in \{0, 1\}^n : \Delta(u, v) \leq 1\}$. We have $B(c_1, 1) \cap B(c_2, 1) = \emptyset$. In addition $|B(u, 1)| = 1 + n$. As a result,

$$|\cup_{c \in C} B(c, 1)| = (n + 1)2^k \ .$$

But clearly this number is at most the size of the whole space which is 2^n. So

$$k \leq n - \log_2(n+1) \ .$$

□

Note that having equality in this bound means that we have perfect packing, i.e., $\cup_{c \in C} B(c,1) = \{0,1\}^n$. Such codes are called perfect codes. In general, the Hamming bound can be written as: for any $(n,k,d)_q$ code, we have $k \leq n - \log_q \mathrm{Vol}_q(\lfloor \frac{d-1}{2} \rfloor, n)$, where $\mathrm{Vol}_q(t,n) = \sum_{i=0}^{t} \binom{n}{i}(q-1)^i$ is the number of points in Σ^n that are at distance at most t from a given point. Recall that we have $\mathrm{Vol}_q(t,n) \approx q^{nh_q(\frac{t}{n})}$ with $h_q(x) = -x\log_q(x) - (1-x)\log_q(1-x)$.

We can also give a simple construction of codes that achieve a good distance by choosing the codewords in a greedy way.

Theorem 6.1.6 (Gilbert-Varshamov bound)**.** *Let $q \geq 2$ and $1 \leq d \leq n$. There exists an $(n,k,d)_q$ code for some k with*

$$k \geq n - \log_q \mathrm{Vol}_q(d-1,n) \ .$$

Proof We analyze a greedy algorithm to construct C. Start with $C = \emptyset$. Then while there is an $x \in \Sigma^n$ such that $\Delta(x,c) \geq d$ for all $c \in C$, then add x to C. We do so as long as we can, i.e., as long as such an x exists. At any time of the algorithm, the code C has minimum distance at least d. Now at the end of the algorithm for all $x \in \Sigma^n$, there exists a $c \in C$ such that $\Delta(x,c) \leq d-1$. In other words, we have

$$\Sigma^n \subseteq \cup_{c \in C} B(c, d-1) \ ,$$

where $B(c,d-1) = \{c' \in \Sigma^n : \Delta(c,c') \leq d-1\}$. Looking at the cardinality of both sets we obtain

$$q^n \leq \sum_{c \in C} |B(c,d-1)| = q^k \mathrm{Vol}_q(d-1,n) \ .$$

Taking the logarithm, we get the desired inequality. □

So in terms of parameters we see that we can have codes with

$$n - \log_q \mathrm{Vol}_q(d-1,n) \leq k \leq n - \log_q \mathrm{Vol}_q(\left\lfloor \frac{d-1}{2} \right\rfloor, n) \ .$$

There is still a gap, but also the codes obtained using this Gilbert-Vashamov bound do not have a very nice structure. In particular, only to describe the code, I should give q^k codewords in Σ^n which is exponentially large in k. And remember our goal was also to get encoding and decoding function that are *efficient* which is difficult if we don't even have an efficient description of the code.

6.2 Linear error-correcting codes

We now consider a very important family of error-correcting codes: linear codes. As we will see a primary advantage of such a family is that a linear code can be represented in an efficient way. A linear code is simply a code that has the structure of a vector space. For this to be meaningful, we need to have a vector space structure on Σ^n. For that we need a field, in particular a finite field.

Theorem 6.2.1. *The size of any finite field is $q = p^s$ for some prime p and some integer $s \geq 1$. Moreover, there is a unique field of size q denoted $\mathbb{F}_q$.*

For primes $q = p$, $\mathbb{F}_p$ can be seen as the integers $\{0, 1, \ldots, p-1\}$ with the usual addition and multiplication taken mod p. For prime powers $q = p^s$, the construction is a bit more complicated. An element in $\mathbb{F}_q$ can be seen as a polynomial over $\mathbb{F}_p$ taken modulo an irreducible polynomial of degree s.

Definition 6.2.2. *Let q be a prime power. $C \subseteq \mathbb{F}_q^n$ is a linear code if it is a linear subspace of $\mathbb{F}_q^n$, i.e., if $x, y \in C$ then $x + y \in C$ and $a \cdot x \in C$ for $a \in \mathbb{F}_q$. If C has dimension k and distance d, we use the notation $[n, k, d]_q$.*

Remark. Note that here we are talking about the dimension as defined for general codes: $\log_q |C|$. Note that this corresponds here with the notion of dimension of a subspace of a vector space (and this is where the name comes from).

Observe that the repetition code $C = \{000, 111\}$ is a linear code with parameters $[3, 1, 3]_2$.

As mentioned earlier, the big advantage of linear codes is that they can be concisely represented using a basis. The following theorem gives two important representations we will be using.

Proposition 6.2.3. *Let S be a linear subspace of $\mathbb{F}_q^n$. Then the following properties hold.*

1. *$|S| = q^k$ for some $k \in \{0, \ldots, n\}$, and k is called the dimension of S*
2. *There exists a basis $v_1, \ldots, v_k \in S$ such that for any $x \in S$ there is a unique tuple $(a_1, \ldots, a_k) \in \mathbb{F}_q^k$ such that $x = \sum_{i=1}^k a_i v_i$. Then the $k \times n$ matrix with rows given by the vectors v_i is called a generator matrix*

$$G = \begin{pmatrix} \leftarrow & v_1 & \rightarrow \\ \leftarrow & v_2 & \rightarrow \\ & \vdots & \\ \leftarrow & v_k & \rightarrow \end{pmatrix}.$$

   With this notation we have $x = (a_1 \ldots a_k) \cdot G$, where x is seen as a row vector. Note that the rows of this matrix are linearly independent so G has full rank k.

3. *There exists a full rank $(n-k) \times n$ matrix H called parity check matrix such that for all $x \in S$, $Hx^T = 0$.*

Remark. There are in general many generator matrices and parity check matrices for the same subspace S. Also note that the name parity check matrix comes for the setting of $\mathbb{F}_2$ where $H_i x^T$ can be seen as a parity check condition on x.

Before we sketch a proof, let us give a generator and parity check matrix for the repetition code:

$$G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \qquad \text{and} \qquad H = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} .$$

Proof [Sketch for Proposition 6.2.3] To construct a basis, one can do so in a greedy way. This will also show that $|S| = q^k$. If $S = \{0\}$, then there is nothing to show. Otherwise, take any nonzero element in S and call it v_1. Then at step t, we define $v_t \in S$ with $v_t \notin \{\sum_{i=1}^{t-1} a_i v_i, a_i \in \mathbb{F}_q\}$. Then we stop whenever such a v_t cannot be found. It is then clear that one has then obtained a set of vectors that generate S. And it is also simple to show by induction that at any step t, the set $\{\sum_{i=1}^{t} a_i v_i, a_i \in \mathbb{F}_q\}$ contains exactly q^t elements in S.

To construct a parity check matrix, consider the set $N = \{y \in \mathbb{F}_q^n : \sum_{i=1}^n x_i y_i = 0 \text{ for all } x \in S\}$. Then N is a linear subspace of $\mathbb{F}_q^n$. Note that N is also the kernel of the map $y \mapsto Gy^T$, so by the rank nullity theorem, the dimension of N is $n - k$. To obtain a parity check matrix for S simply take a basis of N. □

What is the relation between these two representations, can we go from one representation to the other? Yes there are some simple relations, to be seen in the tutorial.

6.2.1 Minimum distance of a linear code

Proposition 6.2.4. *The minimum distance of a linear code $C \subseteq \mathbb{F}_q^n$ is given by $d = \min_{c \in C, c \neq 0} |c|$, where we write $|c| = \{i \in [n] : c_i \neq 0\}$ for the weight of c.*

Proof $0 \in C$ and $\Delta(0, c) = |c|$ so the minimum distance is at least $\min_{c \in C, c \neq 0} |c|$. Moreover, for $c_1, c_2 \in C$, we have $\Delta(c_1, c_2) = |c_1 - c_2|$. But $c_1 - c_2 \in C$ and is nonzero so $\Delta(c_1, c_2) \geq \min_{c \in C, c \neq 0} |c|$. □

The minimum distance also has a nice characterization in terms of the parity check matrix.

Proposition 6.2.5. *Let C be an $[n, k, d]_q$ code C with parity check matrix*

$$H = \begin{pmatrix} H^1 & H^2 & \cdots & H^n \end{pmatrix} ,$$

with columns denoted H^i. Let t denote the minimum number of linearly dependent columns of H, i.e., $t = \min\{|T| : T \subseteq [n], \sum_{i \in T} a_i H^i = 0 \text{ for some } a_i \in \mathbb{F}_q\}$. Then

$$d = t .$$

Proof Let us start with $t \leq d$. Let c be a codeword with $|c| = d$. Then $Hc^T = 0$, which maybe written as $\sum_{i=1}^n c_i H^i = 0$. The support of c gives d columns of H that are linearly dependent.

For $d \leq t$, Let $H^{i_1}, \ldots, H^{i_t}$ be linearly dependent columns. Then there exists nonzero elements $c_{i_1}, \ldots c_{i_t}$ such that $\sum_{j=1}^t c_{i_j} H^{i_j} = 0$. But this means that $x \in \mathbb{F}_q^n$ defined by $x_{i_j} = c_{i_j}$ for all j and $x_i = 0$ otherwise belongs to the code C. As x has weight t, we obtain $d \leq t$. □

Let us now look at an example of a family of codes, namely the **generalized Hamming codes**. These are binary codes, i.e., $q = 2$ and we define them with their parity check matrix.

$$H_r = \begin{pmatrix} H_r^1 & H_r^2 & \ldots & H_r^{2^r - 1} \end{pmatrix} ,$$

where H_r^i is the column vector of length r representing i in binary. For example, for $r = 3$, we have

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

First, note that this is a valid parity check matrix as it has full rank, i.e., rank r, as for example for $r = 3$, the columns $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ are linearly independent. As a result, the dimension of the Hamming code of order r has blocklength $2^r - 1$ and dimension $2^r - 1 - r$. In particular, for $r = 3$, the dimension is 4. This actually corresponds to the Hamming code we have seen earlier as

$$C_H(x_1x_2x_3x_4) = (x_1, x_2, x_3, x_4, x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_2 \oplus x_3 \oplus x_4) .$$

One can check that all these codewords satisfy the parity conditions, and as the dimensions match, this shows the codes are the same. Now we would like to determine the minimum distance of these codes. For $r = 3$, we know the distance is 3.

Claim. The Hamming code with parity check matrix H_r is a $[2^r - 1, 2^r - 1 - r, 3]_2$ code for all $r \geq 3$.

First, let us show that the distance is at least 3. In fact, a distance of 2 would mean that there are two columns that are linearly dependent, i.e., $H_r^i + H_r^j = 0$ for some $i \neq j$. But this implies that $H_r^i = H_r^j$ which implies $i = j$.

Now, observing that $H_r^1 + H_r^2 + H_r^3 = 0$, the distance is exactly 3.

Note that this family of codes matches exactly the Hamming bound which says $k \leq n - \log_2(n + 1)$. These are perfect codes (i.e., if we take a ball of radius 1 around every codeword we cover the space of all bitstrings.

This family of codes have a very good rate of $\frac{2^r - r - 1}{2^r - 1}$ (very close to 1), but a poor minimum distance that is constant equal to 3.

6.2.2 Dual code of a linear code

Definition 6.2.6. *Let C be a linear code with parity check matrix H. The code with generator matrix H is denoted $C^\perp$ and called the dual code.*

Exercise: The dual code $C^\perp$ does not depend on the choice of the parity check matrix H. Also note that this is very different from a complement, in fact we can even have $C^\perp = C$.

In terms of parameters, if C is an $[n, k]_q$ code, then $C^\perp$ is an $[n, n - k]_q$ code.

Consider the Hamming code $C_{Ham,r}$ with parameters $[2^r - 1, 2^r - r - 1, 3]_2$, the dual code is called the simplex code $C_{Sim,r}$ is a $[2^r - 1, r, ?]_2$ code. Note that this code has a very small rate of $\frac{r}{2^r - 1}$, but it is still sometimes useful. As it has H_r as generator matrix, one possible encoding function for the code to be $C_{Sim,r}(x) = xH_r$. So the i-th bit of $C_{Sim,r}(x)$ is the inner product between x and H_r^i. In other words, to encode x, we consider the inner product with all the bitstrings of length r, except the 0 bitstring. It is convenient to actually add also the 0 bitstring of length r as the first column of the generator matrix, this defines the Hadamard code $C_{Had,r}$, which only differs from $C_{Sim,r}$ by adding a 0 at the beginning of every codeword.

Proposition 6.2.7. *The minimum distance of the codes $C_{Sim,r}$ and $C_{Had,r}$ is 2^{r-1}.*

Proof It is sufficient to prove the claim for $C_{Had,r}$ as it only has a leading zero compared to $C_{Sim,r}$.

Our objective is to determine a codeword of minimum weight. Note that it is not sufficient to look only at the rows of the generator matrix H_r and take the minimum weight as a linear combination of these rows might have a smaller weight. But for this special code, we will show a very strong property: for any $c \in C_{Had,r}$, $c \neq 0$, we have $|c| = 2^{r-1}$. In fact, for any row vector $x \in \mathbb{F}_q^r$, we have

$$\begin{aligned} c &= x \begin{pmatrix} H_r^0 & H_r^1 & \dots & H_r^{2^r-1} \end{pmatrix} \\ &= (xH_r^0, xH_r^1, \dots, xH_r^{2^r-1}) \, . \end{aligned}$$

Another way of writing c is by labelling the components by bitstrings of length r directly so $c = (\langle x|u\rangle)_{u\in\{0,1\}^r}$. As $x \neq 0$, there exists an $i \in \{1, \dots r\}$ such that $x_i \neq 0$. Note that for any bitstring $u \in \{0,1\}^r$ and letting e_i be the bitstring with 1 at coordinate i and zero elsewhere, then

$$\langle x|v\rangle = \langle x|u\rangle + \langle x|e_i\rangle = \langle x|u\rangle + x_i = \langle x|u\rangle + 1$$

This means that the component u and v of the codeword c are distinct, so exactly one of them takes the value 1. Thus, we can find a perfect matching between the components u for which $\langle x|u\rangle = 1$ and the ones for which $\langle x|u\rangle = 0$ and thus $|c| = 2^{r-1}$. □

Algorithm 1 Generic decoding of linear code

input: $y \in \Sigma^n$ with the promise that $\min_{x \in C} \Delta(y, x) \leq t$
output: $x \in C$
for $i = 0$ to t **do**
    for $e \in \mathbb{F}_q^n$ with $|e| = i$ **do**
        if $Hy^T = He^T$ return $y - e$ **then**
        end if
    end for
end for

Let us now summarize the codes we have seen so for. We focus on binary codes, i.e., $q = 2$ and we let the blocklength $n \to \infty$. In this case the general bounds tell us $1 - h_2(\frac{d}{n}) \leq \frac{k}{n} \leq 1 - h_2(\frac{d}{2n})$.

TODO: add graph here

6.2.3 Encoding and decoding of a linear code

Recall that an important goal we had is to construct codes with efficient encoding and decoding algorithms. We saw that a linear code has a efficient representation, we now discuss the encoding and decoding function.

Encoding. The encoding function maps the message we would like to send to corresponding codeword. For an $[n, k, d]_q$ linear code over $\mathbb{F}_q$, we can associate a natural encoding function: if we can think of messages as vectors in $a \in \mathbb{F}_q^k$, then the encoding of a is $aG \in \mathbb{F}_q^n$. This encoding function is quite efficient, it takes $k \cdot n$ operations in $\mathbb{F}_q$ to encode a message. If the generator matrix is more structured, for example is sparse, it is possible to improve this to almost linear in n.

Decoding. Let us start with something simpler than decoding: detecting whether an error occurred or not. Suppose we have a parity check matrix H for the code. If we receive y, we simply need to compute Hy^T. If this is the zero vector, we say we have no error and otherwise, there is an error. Of course, this only works if the error has weight at most $d - 1$. This cost in general $n(n - k)$ operations in $\mathbb{F}_q$.

But in general, we not only want to detect whether there has been an error in y but also be able to find $x \in C$ in the code that is the closest to y (more precisely, we might want to determine the message m such that $C(m) = x$ but we will not worry about this distinction here, once you obtain x you can get m by solving a linear system). Here is a simple generic algorithm to corrects t errors in a code (for this to be possible, the minimum distance of C has to be at least $d \geq 2t + 1$). We assume that the code is represented in terms of its parity check matrix. Suppose we send the codeword x and an error e occurs, i.e., $y = x + e$. Note that finding x or finding e is the same thing. So let us try to find e. For that, we can compute the *syndrome* of the error, which is defined as the vector $Hy^T = Hx^T + He^T = He^T$. As such, we want to find the vector $e \in \mathbb{F}_q^n$ with the smallest weight such that He^T is equal to the observed syndrome.

Unfortunately, the running time of this generic decoding algorithm is large. In fact, we have to go over all error vectors of weight at most t which is given by $\sum_{i=0}^{t} \binom{n}{i}(q-1)^i$ which is exponentially large if t is linear in n for example. However, if we want to correct a constant number

of errors, then we obtain a polynomial time decoding algorithm. Note that for each candidate error e, we should multiply it with H, which uses in general $n(n-k)$ operations in $\mathbb{F}_q$.

Looking at the example of the Hamming codes $[2^r, 2^r - 1 - r, 3]_2$. The generic algorithm corrects $t = 1$ error by going through all the $n + 1$ errors of weight at most 1. For each one of these we multiply it with the parity check matrix. Generically, this gives a runtime of $O(n^3)$. However, for the Hamming code, the decoding can even be done in linear time. For that we simply compute the syndrome $s = H_r y^T$ and our objective is to find the e with minimum weight such that $H_r e^T = s$. So either, $s = 0$, in which case we know that $e = 0$, otherwise, we search for e of weight 1. If e_i has a one in position i, then $H_r e_i^T = H_r^i$ but recall that this is nothing but the binary representation of i. So the only thing we need to do is look at the syndrome and interpret it as a number in $\{0, \ldots, 2^r - 1\}$ and the error we are looking for is exactly at this position.

In general, the problem of decoding a linear code is the same as finding the solution e of a linear system $He^T = s$ that has the smallest weight. This is a computationally hard problem (should have it as a homework).

6.3 Reed-Solomon codes

We now introduce one of the most important families of codes. They have many applications even outside of coding theory. It is defined in terms of univariate polynomials.

Recall that a polynomial over $\mathbb{F}_q$ can be seen as a function of the form $F(X) = \sum_{i=0}^{d} f_i X^i$. If $f_d \neq 0$, then $d = \deg(F)$. The set of polynomials is denoted $\mathbb{F}_q[X]$.

The idea of Reed-Solomon codes is very simple, a message is a tuple $(m_0, m_1, \ldots, m_{k-1}) \in \mathbb{F}_q^k$ that we can interpret as a polynomial $f_m(X) = \sum_{i=0}^{k-1} m_i X^i$. Then the encoding of this message is simply the evaluation of the polynomial f_m in n distinct points.

Definition 6.3.1. *Let $\alpha_1, \ldots, \alpha_n$ be distinct elements from $\mathbb{F}_q$ and choose n, k with $k \leq n \leq q$. Then we define the Reed-Solomon codes $RS : \mathbb{F}_q^k \to \mathbb{F}_q^n$ by $RS(m) = (f_m(\alpha_1), \ldots, f_m(\alpha_n))$, where $f_m(X) = \sum_{i=0}^{k-1} m_i X^i$.*

A common special case is to take $n = q - 1$ and $\alpha_1, \ldots, \alpha_n$ all the nonzero elements in $\mathbb{F}_q$.

Example. Let us consider an example with $n = 3, k = 2$ and $q = 3$. Then a message $(m_0, m_1) \in \mathbb{F}_3^2$ or $m_0 + m_1 X \in \mathbb{F}_3[X]$. We then choose $\alpha_1 = 0$, $\alpha_2 = 1$, $\alpha_3 = 2$. We then have $RS((m_0, m_1)) = (m_0, m_0 + m_1, m_0 + 2m_1)$. For example, this gives $(0, 0) \mapsto (0, 0, 0)$ and $(1, 2) \mapsto (1, 0, 2)$.

Observe that Reed-Solomon codes are linear codes. In fact, $RS(m^0 + m^1) = RS(m^0) + RS(m^1)$ for $m^0, m^1 \in \mathbb{F}_q^k$ and $RS(am) = aRS(m)$ for $m \in \mathbb{F}_q^k$ and $a \in \mathbb{F}_q$.

Proposition 6.3.2. *The minimum distance of a Reed-Solomon code with parameters n, k is $n - k + 1$.*

Proof For a linear code, the minimum distance is given by the minimum weight of a nonzero codeword. Let $RS(m) = (f_m(\alpha_1), \ldots, f_m(\alpha_n))$ be a codeword, then the weight of this codeword is simple $|\{i \in [n] : f_m(\alpha_i) \neq 0\}|$.

Important fact: A nonzero polynomial $P \in \mathbb{F}_q[X]$ of degree t has at most t roots.

But $\deg(f_m) \leq k-1$ and $f_m \neq 0$ as $m \neq 0$. As a result, $|\{i \in [n] : f_m(\alpha_i) = 0\}| \leq k-1$ and as a result, $|RS(m)| \geq n-k+1$. And clearly we can define a polynomial with exactly $k-1$ roots. □

Note that this distance is optimal as it meets exactly the Singleton bound. For any code, we have $d \leq n-k+1$. But note that we have a strong constraint that $q \geq n$, in particular this does not work for binary codes. This is only for large alphabets. Let us see an example of a generator matrix for Reed-Solomon codes. If we choose the natural basis $1, X, \ldots, X^{k-1}$ for the messages, then one can write

$$G = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \ldots & \alpha_n^2 \\ \vdots & & & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \ldots & \alpha_n^{k-1} \end{pmatrix}$$

Such a matrix is often called a Vandermonde matrix.

6.3.1 Efficient decoding of Reed-Solomon codes

Using the generic algorithm for decoding linear codes, we know we can correct up to $\frac{d-1}{2}$ errors. But this algorithm is inefficient. We will now see how to determine the error efficiently. The task is the following: given $y \in \mathbb{F}_q^n$, we would like to find a polynomial P such that $\Delta((P(\alpha_1), \ldots, P(\alpha_n)), y) < \frac{d}{2}$, where $d = n-k+1$. Note that as we assume that y arises from applying an error of weight $< \frac{d}{2}$ to some valid codeword, such a P exists and is unique. Note that in the case where there are no errors, this is an interpolation problem, which can be solve efficiently. But we cannot afford testing all the error patterns and applying interpolation in each case.

The strategy will be to look for P together with a polynomial determining that characterizes the positions where an error occurred. Let us introduce the error locator polynomial whose zeros are exactly the positions that contain an error

$$E(X) = \prod_{\substack{i=1 \\ y_i \neq P(\alpha_i)}}^{n} (X - \alpha_i) .$$

Note that the degree of E is exactly the number of errors and thus it is $< \frac{d}{2}$. Of course, we do not have access to E but we would like to find it. Note that the polynomials P and E we would like to find satisfy the following identities for all $i \in \{1, \ldots, n\}$:

$$y_i E(\alpha_i) = P(\alpha_i) E(\alpha_i) . \tag{6.1}$$

In fact, either $E(\alpha_i) = 0$ in which case equality clearly holds, or otherwise, $E(\alpha_i) \neq 0$ in which case there is no error in position i and $P(\alpha_i) = y_i$. Recall that P has degree at most $k-1$ and so is specified by k elements in $\mathbb{F}_q$ and E has a leading coefficient of 1 and so is specified by $\deg(E)$ coefficients in $\mathbb{F}_q$. So overall we are looking for $k + \deg(E) < k + \frac{n-k+1}{2} = \frac{n+k+1}{2}$

Algorithm 2 Decoding algorithm for a Reed-Solomon code RS

1: **input:** $(y_1, \ldots, y_n) \in \mathbb{F}_q^n$ with the promise that $\min_{m \in \mathbb{F}_q^k} \Delta(y, RS(m)) \leq t$
2: **output:** A polynomial $P(X)$ of degree $\leq k-1$
3: let $e_0, \ldots, e_{t-1}$ and $n_0, \ldots, n_{t+k-1}$ be variables in $\mathbb{F}_q$
4: and $E(X) = e_0 + e_1 X + \ldots e_{t-1} X^{t-1} + X^t$ and $N(X) = n_0 + \ldots n_{t+k-1} X^{t+k-1}$.
5: solve the system of n equations in the $2t+k$ variables defined above:
6: $y_i E(\alpha_i) = N(\alpha_i), \quad 1 \leq i \leq n$
7: **if** no solution exists or the obtained $E(X)$ does not divide $N(X)$ **then**
8: return fail
9: **end if**
10: $P(X) \leftarrow \frac{N(X)}{E(X)}$
11: return $P(X)$

elements in $\mathbb{F}_q$. So we have $< \frac{n+k+1}{2}$ variables and n equations in (6.1) so we can hope to be able to recover all the coefficients of P and E. The however is that the equations (6.1) are quadratic in these coefficients. So what we will do instead is solve the relaxation of equations (6.1) given by weakening the condition that the right hand side is a product of P and E and treat it as a general polynomial N of degree at most $\deg(P) + \deg(E)$. We obtain the equations:

$$y_i E(\alpha_i) = N(\alpha_i) \,, \tag{6.2}$$

where $\deg(E) < \frac{d}{2} = \frac{n-k+1}{2}$ and $\deg(N) < k - 1 + \frac{n-k+1}{2} = \frac{n+k-1}{2}$. Considering the coefficients of E and N as variables, we have $\deg(N)+1+\deg(E) < \frac{n-k+1+n+k-1}{2} + 1 = n+1$ variables, i.e., at most n variables Note that the set of equations (6.2) definitely have a solution: simply take E to be the actual error locator polynomial and take $N = P \cdot E$. But as we will see by considerations on the degree, we will see that this is indeed the unique solution. Let us now state the decoding algorithm, called Welsh-Berlekamp. The parameters used are n, k and we use e to denote the number of errors to be corrected: this is $t = \frac{n-k}{2}$ if $n-k$ is even or $t = \frac{n-k-1}{2}$ if $n-k$ is odd.

Running time. A linear system of n equations and at most n variables can be solved using Gaussian elimination using $O(n^3)$ operations in $\mathbb{F}_q$. Also note that the division $\frac{N(X)}{E(X)}$ can be done just like primary school division: this is certainly bounded by $O(n^2)$ operations in $\mathbb{F}_q$.

Correctness. First we show that provided $\min_{m \in \mathbb{F}_q^k} \Delta(y, RS(m)) \leq t$, the set of equations has a valid solution. Recall that $RS(m) = (f_m(\alpha_1), \ldots, f_m(\alpha_n))$ with $m \in \mathbb{F}_q^k$. Let us define $E^*(X) = X^{t-\Delta(y,RS(m))} \prod_{i:y_i \neq f_m(\alpha_i)} (X - \alpha_i)$ and $N^*(X) = f_m(X) E^*(X)$. Then $E^*(X)$ has degree exactly t with a leading coefficient of 1 and $N^*(X)$ has degree at most $t+k-1$ as required. In addition, $E^*(X)$ and $N^*(X)$ satisfies the linear equations line 6. For these solutions, $E^*(X)$ divides $N^*(X)$ and the returned polynomial is $P = f_m$, which is what we wanted.

But it remains to show that this valid solution is the unique one satisfying the linear equations line 6. In fact, let E_1, N_1 and E_2, N_2 be two solutions of line 6. Then define

$$R(X) = N_1(X) E_2(X) - N_2(X) E_1(X) \,.$$

By assumption $\deg(N_1 E_2) \leq 2t + k - 1$, so $\deg(R) \leq 2t + k - 1$. But as $N_1(\alpha_i) = y_i E_1(\alpha_i)$ and $N_2(\alpha_i) = y_i E_2(\alpha_i)$, we have $N_1(\alpha_i)E_2(\alpha_i) - N_2(\alpha_i)E_1(\alpha_i) = 0$. As such R has n distinct roots and degree $\leq 2t + k - 1 < n$. This implies that $R(X) = 0$ and thus that $\frac{N_1(X)}{E_1(X)} = \frac{N_2(X)}{E_1(X)}$. Thus any solution E, N leads to the same returned polynomial P. This concludes the correctness of the algorithm.

So the family of Reed Solomon codes have very nice properties: it has optimal minimum distance $d = n - k + 1$ which matches the Singleton bound and in addition we can encode and decode in time that is polynomial in the blocklength n. For example, for $k = n/2$ we get a minimum distance $d = \frac{n}{2} + 1$ so we have both $k = \Omega(n)$ and $d = \Omega(n)$. This is what we called a good code. But the downside of these codes is that it only works if the alphabet size is large enough, in particular $q \geq n$. What if we want to construct good binary codes? In the next section, we study a simple operation on codes that allows us to reach this.

6.4 Concatenation of codes

The idea of concatenation is simple. Let us start with a code C over $\mathbb{F}_q$ with blocklength n. Then each codeword is composed of n symbols in $\mathbb{F}_q$. Assume that $q = 2^t$ for some integer t. Then we can associate to each symbol in $\mathbb{F}_q$ a bitstring of length t. So any codeword $(x_1, \ldots, x_n) \in C$ can be seen as a bitstring of length tn of the form $(x_{1,1} \ldots x_{1,t} x_{2,1} \ldots x_{2,t} \ldots x_{n,1} \ldots x_{n,t}) \in \{0,1\}^{nt}$. This procedure gives a *binary* code with blocklength $t \cdot n$ and dimension $t \cdot k$ (the number of codewords stays the same: it is $q^k = 2^{tk}$ if k is the dimension of C).

For example, if we take a $[n, \frac{n}{2}, \frac{n}{2} + 1]_n$ RS code. By doing this construction, we obtain an $[n \log_2 n, \frac{n}{2} \log_2 n, ?]_2$ error-correcting code. So the rate is good as it is linear in the blocklength. Let us now analyze the minimum distance

$$\begin{aligned} \Delta(x_{1,1} \ldots x_{n,t}, y_{1,1} \ldots, y_{n,t}) &= |\{(i,j) \in [n] \times [t] : x_{i,j} \neq y_{i,j}\}| \\ &\geq |\{i \in [n] : x_i \neq y_i\}| \\ &\geq \frac{n}{2} + 1 \,, \end{aligned}$$

where we used the fact that the minimum distance of the RS code is $\frac{n}{2} + 1$. But this is not quite linear in the blocklength, which was $n \log_2 n$. Note that the first inequality is tight if for every i with $x_i \neq y_i$, there is a unique j such that $x_{i,j} \neq y_{i,j}$. And such codewords will in general exists. For example take the zero codeword for x and the codeword with $y_1 = \cdots = y_{k-1} =$ any nonzero element of $\mathbb{F}_q$ whose binary representation has a unique 1, and $y_k = \cdots = y_n = 0$. But this suggests a idea to improve the minimum distance: instead of using a "trivial" representation for the symbols x_i, a better representation would be to encode these x_i's into an error correcting code for each block.

Definition 6.4.1 (Code concatenation)**.** *Let $C_{out} : [Q]^K \to [Q]^N$ a $(N, K, D)_Q$ code and $C_{in} : [q]^k \to [q]^n$ be a $(n, k, d)_q$ code with $q^k = Q$. Then the concatenation $C_{out} \circ C_{in}$ is a code with alphabet $[q]$, blocklength nN and dimension kK defined by the encoding function $C : [Q]^K \to [q]^{nN}$ as: for $m \in [Q]^K$,*

$$C(m) = (C_{in}(C_{out}(m)_1), C_{in}(C_{out}(m)_2), \ldots, C_{in}(C_{out}(m)_n)) \,,$$

where $C_{out}(m)_i$ is the i-th symbol of the codeword $C_{out}(m)$.

Remark. In this construction, we have identified the set $[Q]$ with the set $[q]^k$. For this identification, we can take an arbitrary bijection σ between the two sets (for efficiency considerations, we also want this bijection to be efficiently computable both ways). Also when the codes C_{out} and C_{in} are linear, we can take the bijection to be a linear map so that $C_{out} \circ C_{in}$ is a linear code. To construct such a linear map, it suffices to see that $\mathbb{F}_{q^k}$ is a vector space over $\mathbb{F}_q$ (in fact it is the set of polynomials in $\mathbb{F}_q[X]$ modulo an irreducible polynomial E), then $\sigma : \mathbb{F}_q^k \to \mathbb{F}_{q^k}$ mapping $(a_0, \ldots, a_{k-1}) \mapsto \sum_{j=0}^{k-1} a_j X^j \mod E$ is a valid bijection. We can also write a generator matrix $G_{out \circ in} \in \mathbb{F}_q^{kK \times nN}$ for $C_{out} \circ C_{in}$ as a function of generator matrices G_{out} and G_{in} for C_{out} and C_{in}:

$$G_{out \circ in} = \begin{pmatrix} \sigma^{-1} & 0 & \ldots & 0 \\ 0 & \sigma^{-1} & 0 & \ldots \\ \vdots & & & \vdots \\ 0 & \ldots & 0 & \sigma^{-1} \end{pmatrix} \cdot G_{out} \cdot \begin{pmatrix} \sigma & 0 & \ldots & 0 \\ 0 & \sigma & 0 & \ldots \\ \vdots & & & \vdots \\ 0 & \ldots & 0 & \sigma \end{pmatrix} \cdot \begin{pmatrix} G_{in} & 0 & \ldots & 0 \\ 0 & G_{in} & 0 & \ldots \\ \vdots & & & \vdots \\ 0 & \ldots & 0 & G_{in} \end{pmatrix}$$

From now we identify between the sets without explicitly mentioning that we are applying σ or σ^{-1}.

In our example, C_{out} is a Reed-Solomon code with parameters $[N, \frac{N}{2}, \frac{N}{2} + 1]_N$ and C_{in} is the "identity" code with $(n = \log N, k = \log N, d = 1)_2$ which maps a bitstring of length $\log N$ to the same bitstring of length $\log N$. The natural question is now about the distance of the concatenated code $C_{out} \circ C_{in}$.

Proposition 6.4.2 (Distance of a concatenated code)**.** *If C_{out} is a $(N, K, D)_{q^k}$ code and C_{in} is a $(n, k, d)_q$ code, then*

$$C_{out} \circ C_{in} \quad \textit{is a} \quad (nN, kK, dD)_q \textit{ code.}$$

Proof Let $m \neq m' \in [q^k]^K$ then $\Delta(C_{out}(m), C_{out}(m')) \geq D$. Note that here these are words of length N with symbols in $[q^k]$. Now for any $i \in [N]$ such that $C_{out}(m)_i \neq C_{out}(m')_i$ we can view it as an element in $[q]^k$ and apply the encoding C_{in} to get $\Delta(C_{in}(C_{out}(m)_i), C_{in}(C_{out}(m')_i)) \geq d$. Note that here the Hamming distance is between words of length n on the alphabet $[q]$.

$$\Delta(C_{out} \circ C_{in}(m), C_{out} \circ C_{in}(m')) = \sum_{i : C_{out}(m) \neq C_{out}(m')} \Delta(C_{in}(C_{out}(m)_i), C_{in}(C_{out}(m')_i)) \geq dD \; .$$

Note that we decomposed here a Hamming distance between words of length nN on the alphabet $[q]$. □

So now if we would like to find a good code, we can take C_{out} to be a Reed-Solomon code and it remains to find a good inner code C_{in}. But what have we gained in reducing the problem to finding a good inner code. The idea is that the inner code is a code over a small block length so we can more easily find a good one. In particular, if $n = \log N$ as was the case for our example, we can afford to search for a good code and have an exponential time decoder in the blocklength of C_{in}.

6.4.1 Explicit construction of a good binary code

Now we are in a position to construct an explicit good binary code, i.e., one with dimension $k = \Omega(n)$ and $d = \Omega(n)$. What we mean by explicit here is that we can construct a representation of this code in time that is polynomial in the blocklength n. Note that if we take a random binary linear code, it will be a good code with high probability, but then there is not much hope in getting an efficient decoder.

This construction will use concatenation of two linear codes. We construct generator matrices G_{out} and G_{in} of the outer and inner code in time that should be polynomial in the overall blocklength of $C_{out} \circ C_{in}$.

Naturally, we take C_{out} to be a Reed-Solomon code with parameters $[N, \frac{N}{2}, \frac{N}{2} + 1]_N$ with $N = 2^k$. Note that we can take the alphabet size to be N as N is a prime power. G_{out} is simply an $\frac{N}{2} \times N$ Vandermonde matrix which can be constructed in time $O(N^2)$ and N will be smaller than the overall blocklength.

Now for the inner code, it should have a dimension k satisfying $N = 2^k$. We construct a parity check matrix of a code achieving the Gilbert-Varshamov bound. For example, recall that GV bound for linear codes in the HW. It said that provided $2^{n-k} > 1 + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2}$, then there is a $[n, k, d]_2$ code. This code was obtained by adding columns to the parity check matrix one by one in a way that each new column is not in the span of $d-1$ previous columns. If we take $n = 2k$ and $d = 0.1n$, then the condition is satisfied:

$$2^{n-k} = 2^k$$
$$1 + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} \leq 2^{(n-1)h_2(\frac{d-2}{n-1})} < 2^{2kh_2(0.1)} < 2^k \, .$$

So there exists a parity check matrix $H \in \mathbb{F}_2^{k \times 2k}$. How long does constructing such a parity check matrix take? For each new column, we should try all the possible linear combinations of $d-1$ columns: there are at most $\binom{2k}{d} \leq 2^{2k}$. Each step costs $(d-1)k = O(k^2)$. So overall $O(k^3 2^{2k})$ steps to build the parity check matrix H. It is then simple to transform this parity check matrix into a generator matrix for the same code. Recalling that $k = \log N$, this number of steps is polynomial in N.

In summary, we have constructed the generator matrix of a binary code with blocklength $N \cdot (2 \log N)$, dimension $\frac{N}{2} \cdot \log N$ and minimum distance $(\frac{N}{2} + 1) \cdot 0.2 \log N$. Observe that both the dimension and the minimum distance are linear in the blocklength. In addition, generating this matrix can be done in time polynomial in the blocklength.

6.4.2 Decoding a concatenated code

We would like to decode the code define in the previous section efficiently. For this, there is a natural decoding algorithm for a concatenated code. Decode each block first using the inner code. And then decode the outer code. Assume we have a decoding functions $D_{C_{in}} : [q]^n \to [q]^k$ and $D_{C_{out}} : [q^k]^N \to [q^k]^K$. Then, for $(y_1 \dots y_n) \in (\mathbb{F}_q^n)^N$, define $D_{C_{out} \circ in}(y_1 \dots y_n) = D_{C_{out}}(D_{C_{in}}(y_1), \dots, D_{C_{in}}(y_n))$. Note that as usual, we have identified $[q]^k$ and $[q^k]$ here.

The running time of the function $D_{C_{out} \circ in}$ is $N \cdot T(D_{C_{in}}) + T(D_{C_{out}})$, where $T(D)$ is the running time of the function D. For the example introduced in the last section, we have $T(D_{C_{out}}) = O(N^3)$

as shown in the last lecture. For the code C_{in}, we can just use the generic decoding for linear codes which runs in time $O(2^{2k}k^2) = O(N^2 \log^2 N)$. Overall, we obtain a running time that is polynomial in N.

But now it remains to show that this algorithm $D_{C_{out} \circ in}$ actually does correct errors.

Proposition 6.4.3. *The algorithm $D_{C_{out} \circ in}$ can correct $< \frac{Dd}{4}$ errors.*

Remark. This is not optimal. The distance of the concatenated code id Dd so in principle can correct $< Dd/2$ errors. A smarted choice of decoder can do that.

Proof Let m be such that $\Delta(C_{out} \circ C_{in}, y) < \frac{dD}{4}$. Want to show that the algorithm outputs m. Define the bad set

$$B = \{i \in [N] : D_{C_{in}}(y_i) \neq C_{out}(m)_i\} .$$

If $|B| < \frac{D}{2}$, then the decoder $D_{C_{out}}$ will be able to correct all the errors and the output of the algorithm will be m.

Otherwise, if $|B| \geq \frac{D}{2}$. Then for each $i \in B$, $D_{C_{in}}(y_i) \neq C_{out}(m)_i$, then $\Delta(y_i, C_{in}(C_{out}(m)_i)) \geq \frac{d}{2}$. So

$$\Delta\left((y_1 \dots y_n), (C_{in}(C_{out}(m)_1), \dots, C_{in}(C_{out}(m)_N))\right) \geq \frac{D}{2} \cdot \frac{d}{2} ,$$

which contradicts our assumption. □

As a conclusion, there exists binary codes with dimension $k = \Omega(n)$ and distance $d = \Omega(n)$ with encoding and decoding that can be computed in polynomial time in n.

6.5 An application of error-correcting codes

Error-correcting codes have many applications in various areas of computer science. We will see a simple application to the problem of testing the equality between two bitstrings. This is also the opportunity to introduce a very useful model: communication complexity.

6.5.1 Communication complexity

Very simple model that quantifies the amount of communication needed to compute a distributed function.

More precisely, suppose Alice has some bitstring $x \in \{0,1\}^n$ (we call it his input) and Bob has a bitstring $y \in \{0,1\}^n$. Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a function known to both Alice and Bob.

Objective: Compute $f(x,y)$ using a protocol

Resource: Minimize communication between Alice and Bob (we don't care about computation)

Let us consider some examples of functions.

- $\text{PAR}(x,y) = \sum_{i=1}^n (x_i + y_i) \mod 2$. To compute this function, Alice can send the parity of her bits. Bob can then compute the parity as $(\sum_{i=1}^n x_i \mod 2) + \sum_{i=1}^n y_i \mod 2$, and send the result to Alice. The communication cost of this protocol is 2 bits.

- $\text{EQ}(x,y) = \begin{cases} 0 & \text{if } x \neq y \\ 1 & \text{if } x = y \end{cases}$ Alice could send x to Bob and Bob computes $\text{EQ}(x,y)$ and sends back the result to Alice. The communication cost of this protocol is $n+1$. This trivial protocol actually works for any function.

Description of a protocol. In general a protocol is a sequence of deterministic algorithms modelling Alice and Bob's behavior. Alice computes $a_1 = A_1(x)$ and sends it to Bob, then Bob computes $B_1(y, a_1)$ and sends it to Alice etc... and the last message is the output of the function. The cost of the protocol on inputs (x,y) is the total number of bits communicated: $|a_1| + |b_1| + \cdots + |a_k| + |b_k|$. The cost of a protocol is the maximum over all possible inputs (x,y) of the cost on input (x,y).

We can then define for any function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, the deterministic communication complexity of f as

$$D(f) = \min_{\mathcal{P} \text{ protocol computing } f} \text{cost}(\mathcal{P}) \ .$$

We saw for example that $D(\text{PAR}) \leq 2$ and $D(\text{EQ}) \leq n+1$. Can we do better for EQ?

Proposition 6.5.1. $D(\text{EQ}) \geq n+1$.

Proof We will only prove $D(\text{EQ}) \geq n$. Suppose we have a protocol $\mathcal{P}$ of cost $\leq n-1$. How many possible transcripts of length $\leq n-1$ are there? At most 2^{n-1}. So there must exist two inputs (x,x) and (x',x') for $x \neq x'$ for which the communication transcript is the same. Let us call the transcript $(a_1, b_1, \ldots, a_k, b_k)$, note that we have $a_1 = A_1(x) = A_1(x')$, $b_1 = B_1(x, a_1) = B(x', a_1)$ etc... So in particular, the output of the protocol is the same for these two inputs $\mathcal{P}(x,x) = \mathcal{P}(x',x') = 1$. This is very good because the value of EQ is the same on these two inputs. But now let us analyze the protocol for the input (x,x'). The first message that is sent by Alice is $A_1(x)$ which happens to be equal to a_1. Then the message sent by Bob is $B_1(x', a_1)$ which in turn happens to be b_1. So this means that the transcript of the protocol is exactly the same for the input (x,x'), which implies that $\mathcal{P}(x,x') = 1$, which contradicts the fact that $\mathcal{P}$ computes f. □

Randomized communication complexity

Now assume that Alice and Bob have some random coins and they are allowed to make some small error probability ϵ. Now $\mathcal{P}$ is a randomized protocol (randomness comes from the coins of Alice and Bob). We say that $\mathcal{P}$ computes f with error ϵ if for all inputs x, y, $\mathbf{P}\{\mathcal{P}(x,y) \neq f(x,y)\} \leq \epsilon$. Note that the probability is over the coins of Alice and Bob and not over the input. We also have to define the cost a protocol as it might depend on the randomness. We define it also in the worst case over the coins of Alice and Bob.

$$R_\epsilon(f) \min_{\text{randomized } \mathcal{P} \text{ with } \mathbf{P}\{\mathcal{P}(x,y) \neq f(x,y)\} \leq \epsilon} \text{cost}(\mathcal{P}) \ .$$

We can focus on say $\epsilon = 1/3$. By repeating, we can reduce the error.

Proposition 6.5.2. $R_{1/3}(\text{EQ}) = O(\log n)$

Proof As a first try, we can try the following. Alice chooses $i \in [n]$ at random and sends (i, x_i) and Bob says 1 if $x_i = y_i$ and 0 if $x_i \neq y_i$. Then if $x = y$, then the protocol always outputs 1, there is no error. But if $x \neq y$, we have $\mathbf{P}\{\mathcal{P}(x, y) = 1\} = 1 - \frac{\Delta(x,y)}{n}$. Note that this could be $\frac{n-1}{n}$ if x and y are at distance 1 (recall that we want the probability bound for every inputs).

Now the natural idea here is to use an error correcting code. Take a $[3n, n, 2n + 1]_q$ Reed-Solomon code with $q = O(n)$. The protocol is then as follows

- Alice chooses an index $i \in [3n]$ at random and sends $(i, C(x)_i)$
- Bob sends 1 to Alice if $C(x)_i = C(y)_i$ and 0 otherwise

Let us start with the correctness of the protocol. If $x = y$, the output is always 1. If $x \neq y$, then

$$\begin{aligned}\mathbf{P}\{\mathcal{P}(x, y) = 1\} &= \frac{|\{i : C(x)_i = C(y)_i\}|}{3n} \\ &\leq \frac{3n - (2n + 1)}{3n} \leq \frac{1}{3}\,.\end{aligned}$$

Now the cost of the protocol is $\log(3n) + \log q = O(\log n)$. □

6.6 Introduction to quantum information theory

In all our discussion in this course, we never talked about the physical way of storing information as this was not relevant. The only important property is that the information carrying system should have a certain number of *distinguishible* states that we can label by a finite set Σ. This can be physically realized in many ways: presence of electric current, pixel lighting, color of a paper, etc... For concreteness, let us take a switch that can have two distinguishable positions: "on" also labeled 1 or "off" also labeled 0. When we only care about the information stored in such a system, the state of the system is given by an element $s \in \Sigma$, in our example $\Sigma = \{0, 1\}$. For example, a channel with inputs in $\mathcal{X}$ and outputs in $\mathcal{Y}$ can be implemented by a physical system that can be prepared in one of $|\mathcal{X}|$ distinguishable states and evolves into a system that has $|\mathcal{Y}|$ distinguishable states.

In such a discussion, we made a hidden assumption: that the complete way of describing the state of the switch is by an element in $\Sigma = \{0, 1\}$. Quantum theory tells us that these are not the only valid states of the system: namely any "superposition" of the two valid distinguishable states is also a valid quantum state and can in principle be realized. Quantum information theory is about exploiting this whole set of possibilities that is given to us by such superpositions.

6.7 Representation of a quantum system

To get to the representation of a quantum system, it is useful to start with a probabilistic model. Continuing with our switch example, now the way of describing the state of the switch is by a probability vector $v = \begin{pmatrix} 1/4 \\ 3/4 \end{pmatrix}$. We see the first entry $(1/4)$ as the probability of being in the state 0 and the second entry $(3/4)$ as the probability of being in the state 1. An important point to understand is that the vector v is a representation of our knowledge about the system and looking at the switch, we don't see v but we see that it is on or off. The entries of v are only giving probabilities for each setting.

Note here that one important feature that will be important in quantum theory is that the action of looking at the switch, I changed my description of the state. I started with v and after looking, my description became $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ if I saw that it was off and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ if I saw it was on.

Let us now model to a quantum model for the same switch system. We still have the distinguishable states "off" and "on", which we can write as in the probabilistic model as $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Now the difference with a probabilistic model is that the state of knowledge is represented by vector $v = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. The numbers α and β are called amplitudes. Some examples include:

$$\begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 3/5 \\ 4i/5 \end{pmatrix}$$

Such a system holds one *qubit* of information.

There are two kinds of operations you can do:

1. Look at the switch that we call perform a measurement. We do not see v, we see either 0 or 1 with probability $|\alpha|^2$ and $|\beta|^2$ respectively. Again, after the measurement, the state becomes $\begin{pmatrix}1\\0\end{pmatrix}$ or $\begin{pmatrix}0\\1\end{pmatrix}$ depending on the observed outcome. Another way of writing this is that for a vector $v \in \mathbb{C}^2$, the probability of observing outcome 0 is $|\langle e_0|v\rangle|^2$ and the probability of observing $|\langle e_1|v\rangle|^2$, where $e_0 = \begin{pmatrix}1\\0\end{pmatrix}$.

2. Transform without looking that we call perform a unitary. Note that a valid transformation has to transform vectors of unit ℓ_2 norm into vectors of unit ℓ_2 norm. These are exactly unitary transformations, i.e., matrices U such that $U^*U = \mathrm{id}$, with U^* is the conjugate transpose of U. Examples of unitaries include

$$I = \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}, NOT = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}, H = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{pmatrix}.$$

So what happens if we start with a state that if "off", then apply Hadamard and then measure? $\begin{pmatrix}1\\0\end{pmatrix}$, then $H\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix}$ which gives "off" with probability $1/2$ and "on" with probability $1/2$.

Let us do a small exercise: suppose we have a system and we are guaranteed in is in on of the two states: $v_0 = \begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix}$ or $v_1 = \begin{pmatrix}\frac{1}{\sqrt{2}}\\ -\frac{1}{\sqrt{2}}\end{pmatrix}$. Can I determine whether the system is in v_0 or v_1?

Yes just apply Hadamard and then measure. If I see 0 then the system was in v_0 and if I see 1 the system was in v_1. More generally, if v_0 and v_1 are *orthogonal* then I can perfectly distinguish them.

How many bits can we store in a qubit? Well suppose you want to store a trit (i.e., 3 possible values $0, 1, 2$) using such a qubit system. Then for each value you would have a corresponding state: v_0, v_1, v_2. Storing reliably means that by performing some operations on the system you should be able to retrieve which value is stored. It is simple to store a bit in a qubit system: just let $v_0 = \begin{pmatrix}1\\0\end{pmatrix}$ and $v_1 = \begin{pmatrix}0\\1\end{pmatrix}$ and to retrieve the value we simply measure. However, intuitively, if we try to have 3 states then they cannot be orthogonal so I won't have a way of distinguishing them perfectly.

What if we allow some error? It turns out that in some settings having a qubit can have an advantage for better storage.

Bibliography

[1] Y Polyanskiy and Y Wu. Lecture notes on information theory, 2014.

[2] C. Shannon. A mathematical theory of communications. *Bell System Technical Journal*, 27:379–423, 1948.