

1 The mathematics of communication

People have used signals to communicate with other, distant parties for a long time.

- examples:**
- smoke signals
 - light signals
 - semaphores (do you know what those are?)
 - check out the historical note at the end!
 - telegraph
 - ⋮



In most of these setups, some “coding” of the alphabet or the messages was necessary. For the telegraph, for instance, Samuel Morse invented Morse code. He first invented it for English only; later it was extended to include special letters from other “roman” alphabets that did not exist in English (such as Å).

The tables on the right show this “enriched” Morse code.

Relative lengths of dots and dashes: if the duration of a dot is taken to be one unit then that of a dash is three units. The space between the components of one character is one unit, between characters is three units and between words seven units. To indicate that a mistake has been made and for the receiver to delete the last word, send $\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot$ (eight dots).

Source:
www.scphillips.com/morse.html

If we represent

Letter	Morse	Letter	Morse	Letter	Morse
A	.-	N	-.	0	-----
B	-. . .	O	---	1	.-----
C	-. -.	P	.-.-.	2	..-----
D	-. .	Q	---.-	3	...----
E	.	R	.-.	4-
F	..-.	S	5
G	--.	T	-	6	-....
H	U	..-	7	--...
I	..	V	...-	8	----..
J	.---	W	.-.-	9	-----.
K	-.-	X	-.-.-		
L	.-. .	Y	-.--		
M	--	Z	--..		

Letter	Morse	Punctuation Mark	Morse
Ä	.-.-	Full-stop (period)	.-.-.-
Á	.-.-.-	Comma	--..--
Å	.-.-.-	Colon	---...
Ch	----	Question mark (query)	..--..
É	..-..	Apostrophe	.-----
Ñ	---.-	Hyphen	-....-
Ö	---.	Fraction bar	-...-
Ü	..--	Brackets (parentheses)	-.-.-.-
		Quotation marks	.-...-
		At sign	.-.-.-.
		Equals sign	-....-

International Morse Code

a dot by	1	
a dash by	111	(since it takes 3 units)
and spaces by	0	(for each unit)

then the Morse code table shows that

A is presented by	10111 <u>000</u>
	the space that indicates the letter ends and another letter will start
B	111010101000
⋮	
E (shortest representation)	1000
⋮	
Q (longest representation, tied with <i>J</i> and <i>Y</i>)	1110111010111000

i.e. the length, in bits, goes from 4 bits (for E) to 16 bits (for Q), among the “standard” letters.

It makes sense that a longer representation would be used for Q than for E, since E is a much more frequent letter in English than Q, and one therefore saves time (= bits in the telegraph application) by using few bits for E and many for Q rather than, e.g., the converse.

Tables 1 and 2 (next page) give two ways in which the frequency of letters can be counted (experimentally) in English.

Morse himself did not have access (in the 19th century!) to computers that can easily scan texts and count how many Es and Qs (and other letters) occur. He used a clever trick instead: he went to a printer’s office. At that time text was still truly “type-set”, with lead type, one for each letter; printer’s offices had typesetters who would assemble the lines of text from bins that held the individual letters, and then assemble lines into pages. Once a page was printed (or a template made of the whole typeset page that could be used for the printing), the page was taken apart again, and the letters reassigned to their bins. At any moment quite a large number of pages might be set, waiting to be printed or molded or taken apart, so many copies were needed of each letter. From experience, printers knew that some letters were needed much more than others. Morse simply found out what the standard quantities were that each letter bin would hold in a printer’s shop, and inferred from this the frequency ordering of letters.

How well did he do when he designed the Morse code? Could he have made it more efficient? If yet, how much more efficient? That is a question we shall be able to answer when we have learned some information theory. Information theory is a field that was created in 1948 by Claude Shannon, in a remarkable paper that contained many results already, and that, in a certain sense, set a program for several subsequent generations of applied mathematicians.

E	11.1607%	56.88	M	3.0129%	15.36
A	8.4966%	43.31	H	3.0034%	15.31
R	7.5809%	38.64	G	2.4705%	12.59
I	7.5448%	38.45	B	2.0720%	10.56
O	7.1635%	36.51	F	1.8121%	9.24
T	6.9509%	35.43	Y	1.7779%	9.06
N	6.6544%	33.92	W	1.2899%	6.57
S	5.7351%	29.23	K	1.1016%	5.61
L	5.4893%	27.98	V	1.0074%	5.13
C	4.5388%	23.13	X	0.2902%	1.48
U	3.6308%	18.51	Z	0.2722%	1.39
D	3.3844%	17.25	J	0.1965%	1.00
P	3.1671%	16.14	Q	0.1962%	(1)

Table 1: Frequency of letters in the main entries of the *Concise Oxford English Dictionary*, 9th ed. Here, ‘e’ is about 57 times as frequent as ‘q.’

Source:

www.askoxford.com/asktheexperts/faq/aboutwords.

Letter	Standard frequency	Letter	Standard frequency
a	.0761	n	.0711
b	.0154	o	.0765
c	.0311	p	.0203
d	.0395	q	.0010
e	.1262	r	.0615
f	.0234	s	.0650
g	.0195	t	.0933
h	.0551	u	.0272
i	.0734	v	.0099
j	.0015	w	.0189
k	.0065	x	.0019
l	.0411	y	.0172
m	.0254	z	.0009

Table 2: Frequency of letters in English texts

Source: *Silicon Dreams*, by R. Lucky.

Here ‘e’ is about 125 times as frequent as ‘q.’

Historical interlude: The Semaphore¹

The semaphore line was a signalling system invented by the Chappe brothers in France. It is different from the naval semaphore system that uses hand-held flags, which was invented later.

Claude Chappe began development when he and his four brothers lost their livelihoods because of the French Revolution. They determined by experiment that it was easier to see the angle of a rod than determine the presence of a panel. Their system was composed of black movable wooden arms, the position of which indicated alphabetic letters. The Chappe system was controlled by only two handles, and was mechanically simple, and reasonably rugged. Night operation with lamps on the arms was unsuccessful.

Each of the two arms showed seven positions, and the cross bar connecting the two arms had four different angles, for a total of $7 \times 7 \times 4 = 196$ symbols (see Fig. 1)

A crucial innovation was the use of a group of trained, dedicated men to pass the signals.

¹Source: Wikipedia, the free encyclopedia.

See http://en.wikipedia.org/wiki/Semaphore_%28communication%29.

The pictures are taken from <http://people.deas.harvard.edu/~jones/cscie129/images/history/chappe.html> and <http://www.cs.dartmouth.edu/~rockmore/semaphore.html>.

The first Chappe semaphore line was established between Paris and Lille in 1792. It was used to carry dispatches for the war between France and Austria that followed the French revolution (in which, after all, the French had executed Queen Antoinette, sister to the Austrian emperor Joseph II). In 1794, it brought news of a French capture of Condé-sur-l'Escaut from the Austrians less than an hour after it occurred. Other lines were built, including a line from Paris to Toulon.

The first symbol of a message to Lille would pass 193 km (120 miles) through 15 stations in only nine minutes. The speed of the line varied with the weather, but the line to Lille typically transferred 36 symbols, a complete message, in about 32 minutes.

By 1824, the Chappe brothers were promoting the semaphore lines for commercial use, especially to transmit the costs of commodities.

The system was widely copied by other European states, especially after it was used by Napoleon to coordinate his empire and army. In most states, the postal union ran the semaphores.

Many national services adopted signaling systems different from the Chappe system. For example, Britain and Sweden adopted systems of flapping panels (in contradiction to the Chappe brothers' discovery that angled rods are more visible).

Britain developed a series of semaphore towers which allowed rapid communications between London and the naval dockyards at Portsmouth.

This was the period in which the naval semaphore system was invented. This system uses hand-held flags. It is still accepted for emergency communication in daylight (see Fig. 2).

Semaphore lines had several crucial advantages over post roads (roads with stations to change horses). First, a semaphore message could easily travel at several thousand miles per hour. Second, with large signals and a telescope, the distance between stations could be as much as 30 km (20 miles), over mountain ranges and bad terrain, reducing investment and the number of stations over other forms of communication. Finally, techniques were developed to permit a semaphore relay line to serve a region, not just a single town, permitting a service to amortize the line's expense over several towns, and reach the headquarters of a bivouacked army.

The semaphores' crucial disadvantages were that they were affected by weather, especially fog and rain, and they could be read by anyone with training.

The first code book was developed for use with semaphore lines. The directors of the Chappes' corporation used a secret code that took 92 of the basic symbols two at a time to yield 8,464 coded words and phrases.

Napoleon Bonaparte saw the military advantage in being able to transmit information between locations, and carried a portable semaphore with his headquarters. This allowed him to coordinate forces and logistics over longer distances than any other army of his time.

Semaphores were adopted and widely used (with hand-held flags replacing the mechanical arms) in the maritime world in the early 1800s. Semaphore signals were used, for example, at the Battle of Trafalgar.

The semaphores were successful enough that Samuel Morse failed to sell the electrical telegraph to the French government. However, France finally committed to replace semaphores with electric

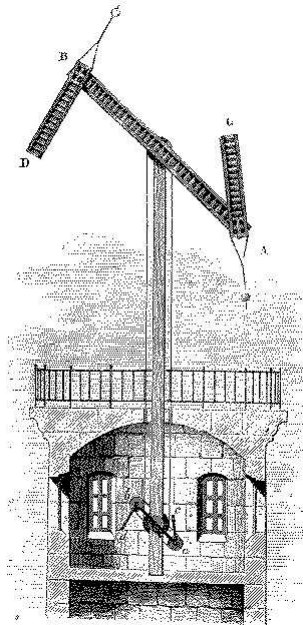


Figure 1: Cross section of a Chappe semaphore tower.

THE SEMAPHORE ALPHABET.

CHAR- ACTERS	HAND FLAGS	CHAR- ACTERS	HAND FLAGS	CHAR- ACTERS	HAND FLAGS	CHAR- ACTERS	HAND FLAGS
A		H		O		V	
B		I		P		W	
C		J		Q		X	
D		K		R		Y	
E		L		S		Z	
F		M		T		ATTENTION	
G		N		U		BREAK	

Figure 2: The alphabet for the naval semaphore.

telegraphs in 1846. Note that electric telegraphs are both more private and unaffected by weather. Many contemporaries predicted the failure of electric telegraphs because "they are so easy to cut."

The last stationary semaphore link in regular service was in Sweden, connecting an island with a mainland telegraph line. It went out of service in 1880.

Relative Costs

The semaphore system was cleverly designed, and provided a strategic advantage for France in a difficult time. However, it was about 25 times more expensive per message than the electric telegraph, which explains why it became obsolete fairly quickly when telegraphic communication became operational in the 1840s. Here's a brief breakdown comparing costs using present-day \$US (but 19th century working conditions: 10 to 12 hour days, seven days per week...):

Semaphore line, 120 miles (Paris to Lille)

- 15 towers (\$1,500,000)
- At least 15 full-time operators (one for each tower) (\$450,000/year).
- Operates at most ten hours a day.
- Sends roughly 2 words per minute (1 symbol per minute, at 2 symbols per phrase, using the efficient directors' codebook).
- Cost to send one word one mile, at 10% interest: \$0.0114

Electric Telegraph line, 120 miles

- At least eight full-time operators (counting that the message needed to be repeated every 40 miles, which required 4 stations, each with one day and one night operator) (\$240,000/year)
- Poles, right-of-way, wires, installation: \$15,000/mile (\$1,800,000)
- Operates 24 hours a day.
- Sends 15 words per minute (includes breaks for the operators).
- Cost to send one word one mile, at 10% interest: \$0.000444

2 The basic formula in information theory

Suppose we intercept regularly typed texts which we suspect of containing secret and sensitive material, in some coded form. Here are 2 such pages:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
dintracerascurofittoamanodrit
taperositravidivaunabaraccava
doaoagliarelatorciadisfefazio
quannotornochiunouvamenteilca
ncelloconlasbarraeaddrumolato
rciasavvicinaroquateლოსამთე
allaportadellabaraccaesadduna
rocheeramezzapertaevidentemen
tefilibertoperilcavudononcela
facivaastariconlaportainserra
taoralosisintivarunfuliaridel
labellanondobbiamodarglitempo
diragionarimumuriontalbanoa
```

It is fairly obvious that the page on the left cannot possibly convey much information: it is entirely predictable; the page on the right may or may not be of interest, but it has at least some potential, since it does not look as boring, and uses many more symbols.

Can we try to capture this “potential” in some formula? Could we quantify it so that it clearly shows that the page on the right has potentially more information than that on the left? Clearly we will have to use the fact that more letters occur than on the left.

In a first attempt, let us just look at the frequency of the letters in both cases. Both pages contain 377 letters; if we pick a letter at random on the left, we always have ‘a’, i.e. the probability we pick ‘a’ is 1; the probability of getting another letter is 0. On the right we have a much more interesting picture: if we pick a letter at random here, then the following table gives the probabilities for the different letters (based on their frequency; e.g. 65 of the 377 letters are ‘a’ so the probability or frequency for ‘a’ is $65/377 = .172$)

Number of occurrences for each letter

a	65	f	5	k	0	p	6	u	13
b	8	g	3	l	24	q	2	v	10
c	18	h	2	m	11	r	33	w	0
d	16	i	33	n	26	s	12	x	0
e	29	j	0	o	35	t	23	y	0
								z	3

but also

$$\begin{aligned}
377 \wp &= \text{Total information in the original text} \\
&= \text{Total information in the w-text} + \text{Total information in the a-e-sequence} \\
&= 377 \underbrace{\left[\sum_{\substack{i=2 \\ i \neq 10,11,23,24,25}}^{26} \varphi(p_i) + \varphi\left(\frac{65+29}{377}\right) \right]}_{\text{total information potential for text with w's}} + \underbrace{(65+29)}_{\text{number of letters in the a-e sequence}} \underbrace{\left[\varphi\left(\frac{65}{65+29}\right) + \varphi\left(\frac{29}{65+29}\right) \right]}_{\text{inf. potential per letter in a-e sequence}} \\
&= 377 \left[\sum_{\substack{i=2 \\ i \neq 10,11,23,24,25}}^{26} \varphi(p_i) + \varphi(p_1 + p_5) \right] + 377(p_1 + p_5) \left[\varphi\left(\frac{p_1}{p_1 + p_5}\right) + \varphi\left(\frac{p_5}{p_1 + p_5}\right) \right],
\end{aligned}$$

which simplifies to

$$\varphi(p_1) + \varphi(p_5) = \varphi(p_1 + p_5) + (p_1 + p_5) \left[\varphi\left(\frac{p_1}{p_1 + p_5}\right) + \varphi\left(\frac{p_5}{p_1 + p_5}\right) \right].$$

It clearly didn't matter what the exact values were of p_1, p_5 in this argument, so that we conclude that φ must satisfy the equation

$$\varphi(x) + \varphi(y) = \varphi(x + y) + (x + y) \left[\varphi\left(\frac{x}{x + y}\right) + \varphi\left(\frac{y}{x + y}\right) \right] \quad (1)$$

for arbitrary x, y in $[0, 1]$, subject to the condition $0 < x + y \leq 1$. Note that if $x + y = 1$, this equation implies

$$\varphi(1) = 0.$$

This is consistent with what we observed for the text that had only 'a'-s: for that text

$$\wp = \varphi(1) = 0$$

and it had indeed no information potential whatever.

On the other hand, if we set $x = 0$, we find

$$\varphi(0) + \varphi(y) = \varphi(y) + y [\varphi(0) + \varphi(1)]$$

or $(1 - y)\varphi(0) = 0$, which implies $\varphi(0) = 0$. (This is also consistent with our earlier sum for \wp , in which we didn't write a term for any of the letters (j, k, w, x, y) that didn't occur in the text, for which the corresponding p_i were therefore 0. With the convention $\varphi(0) = 0$, we could also have written \wp more simply as $\wp = \sum_{i=1}^{25} \varphi(p_i)$.)

Surprisingly, these constraints *completely* determine the function φ (up to normalization).

Let's assume that φ is continuous on $[0, 1]$, and twice differentiable on $(0, 1)$. (Later, we'll weaken these restrictions.)

We can rewrite the equation for φ as follows. Define

$$F_z(\alpha) = \varphi(\alpha z) + \varphi((1 - \alpha)z) - z[\varphi(\alpha) + \varphi(1 - \alpha)].$$

Because $F_z(\alpha) = 0$ for all $z, \alpha \in [0, 1]$, F_z does not depend on α . This implies, for all $\alpha \in (0, 1)$,

$$\frac{d}{d\alpha} F_z = 0, \text{ or further}$$

$$z\varphi'(\alpha z) - z\varphi'((1 - \alpha)z) - z[\varphi'(\alpha) - \varphi'(1 - \alpha)] = 0,$$

$$\text{or } \varphi'(\alpha z) - \varphi'((1 - \alpha)z) = \varphi'(\alpha) - \varphi'(1 - \alpha).$$

This in turn implies that $G_\alpha(z) := \varphi'(\alpha z) - \varphi'((1 - \alpha)z)$ does not depend on z . Then, for all $z \in (0, 1)$,

$$\begin{aligned} \frac{d}{dz} G_\alpha(z) &= 0 \\ \Rightarrow \alpha\varphi''(\alpha z) - (1 - \alpha)\varphi''((1 - \alpha)z) &= 0 \\ \Rightarrow \alpha z\varphi''(\alpha z) &= (1 - \alpha)z\varphi''((1 - \alpha)z). \end{aligned} \tag{2}$$

This holds for all $\alpha, z \in (0, 1)$. Setting $x = \alpha z$ and $y = (1 - \alpha)z$ again, we get thus, for all $x, y \in [0, 1]$, with $x + y \leq 1$,

$$x\varphi''(x) = y\varphi''(y).$$

If $x \in (0, \frac{1}{2}]$, then we can choose $y = 1/2$ (since $x + 1/2 \leq 1$), so that we obtain $x\varphi''(x) = (1/2)\varphi''(1/2)$ for all $x \in (0, \frac{1}{2}]$. On the other hand, if $x > 1/2$, then we can pick $y = 1 - x$; clearly $y < 1/2$, so that we already know (from what we just derived) that $y\varphi''(y) = (1/2)\varphi''(1/2)$. Since we also have $x + y \leq 1$, we have $x\varphi''(x) = y\varphi''(y)$. It follows that $x\varphi''(x) = (1/2)\varphi''(1/2)$ for $x > 1/2$ as well. In summary,

$$x\varphi''(x) = \frac{1}{2}\varphi''\left(\frac{1}{2}\right) = \text{constant for all } x \in (0, 1].$$

Now we can integrate:

$$\begin{aligned} \varphi''(x) &= \frac{c_1}{x} \\ \Rightarrow \varphi'(x) &= c_1 \ln x + c_2 \\ \Rightarrow \varphi(x) &= c_1 x(\ln x - 1) + c_2 x + c_3. \end{aligned}$$

We have:

$$\begin{aligned}\varphi(0) = 0 &\Rightarrow c_3 = 0; \\ \varphi(1) = 0 &\Rightarrow c_2 - c_1 = 0 \\ \Rightarrow \varphi(x) &= c_1 x \ln(x).\end{aligned}$$

We thus proved (constructively!) that every solution of the initial equation (1) must be a multiple of $x \ln x$. In fact, we have

Lemma 1 Consider the class \mathcal{C} of functions $\varphi : [0, 1] \rightarrow \mathbb{R}_+$ that are continuous on $[0, 1]$, twice differentiable on $(0, 1)$, not identically zero, and that satisfy that for all $\alpha, z \in [0, 1]$,

$$\varphi(\alpha z) + \varphi((1 - \alpha)z) = \varphi(z) + z \left[\varphi(\alpha) + \varphi(1 - \alpha) \right].$$

Then

$$\mathcal{C} = \{ \varphi_c : [0, 1] \rightarrow \mathbb{R}_+ \mid c \in \mathbb{R}, c < 0, \varphi_c(0) = \varphi_c(1) = 0, \varphi_c(x) = cx \ln x \} .$$

Proof:

Just copy the text above, starting from “We can write the equation . . . ,” until the conclusion $\varphi(x) = c_1 x \ln(x)$. This proves that all elements of \mathcal{C} have the prescribed form. Because the range of φ is in \mathbb{R}_+ , and φ cannot be identically zero, we must have $c_1 < 0$. It remains to check that φ of this form do satisfy the equation, as well as all the conditions of the lemma. This is an easy exercise left to the reader. ■

This result is only a “Lemma” and not a “Theorem” because it establishes an auxiliary result only, a stepping stone on the way to the proof of the following theorem (which may look identical at first sight – until you notice that we only require that φ be *continuous* here, not twice continuously differentiable!):

Theorem 2 Consider the class \mathcal{C} of functions $\varphi : [0, 1] \rightarrow \mathbb{R}_+$ that are continuous on $[0, 1]$, not identically zero, and that satisfy that for all $\alpha, z \in [0, 1]$,

$$\varphi(\alpha z) + \varphi((1 - \alpha)z) = \varphi(z) + z \left[\varphi(\alpha) + \varphi(1 - \alpha) \right].$$

Then

$$\mathcal{C} = \{ \varphi_c : [0, 1] \rightarrow \mathbb{R}_+ \mid c \in \mathbb{R}, c < 0, \varphi_c(0) = \varphi_c(1) = 0, \varphi_c(x) = cx \ln x \} .$$

Proof:

Exercise! (Don’t worry, you’ll get hints.) ■

It is customary to choose $c_1 = -(\ln 2)^{-1}$, so that $\varphi(x) = -x \log_2 x$; substituting this back into \wp we obtain

$$\wp = - \sum_i p_i \log_2 p_i.$$

This is the expression for the “information potential per symbol” of a source that produces different symbols s_i , each with its corresponding probability p_i .

One can also view this expression as making explicit the “uncertainty” about what will happen next (no uncertainty for the text of all a’s...!), or a “lack of predictability.” Later we shall see that it is related to compressibility. None of these “ ”-names are the official one: although Shannon did consider calling \wp the “uncertainty,” he finally settled on the name **entropy**, because this was already the name for a very similar sum in statistical mechanics. It is traditionally denoted by H :

$$H = - \sum_i p_i \log_2 p_i$$

Remarks:

1. For the text example we saw, the “true” entropy is more complex: patterns exist not only in the different frequencies of the letters, but also in combinations of letters. (For instance, vowels and consonants nicely “take turns” – very rarely does one find more than 3 consecutive ones.)

Similarly, the entropy of English does not simply result from applying the formula to the table of letter frequencies we saw earlier; patterns in combinations of two, three, and more letters, as well as often recurring words, ... need to be taken into account as well. This amounts to defining a richer “alphabet” (with combinations of several letters as the building blocks), and then applying Shannon’s entropy formula.

2. We can view the formula of H as a sum, over all the possibilities i , of $p_i \times b_i$, where p_i is the probability for i , and $b_i = -\log_2 p_i$. What does this mean?

Let’s take an example:

Suppose we have 4 possibilities:

- $A \quad p_A=1/2$
- $B \quad p_B=1/4$
- $C \quad p_C=1/8$
- $D \quad p_D=1/8$

and suppose we want to encode strings of these 4 letters (in which they occur with these frequencies) into binary. For 4 possibilities, we can use 2 bits, e.g.

- $A \leftrightarrow 00$
- $B \leftrightarrow 01$
- $C \leftrightarrow 10$
- $D \leftrightarrow 11$

We can, however, gain by using a different correspondence, in which each A uses up only 1 bit, at the price of using 3 bits for C and D :

$$\begin{aligned} A &\leftrightarrow 0 \\ B &\leftrightarrow 10 \\ C &\leftrightarrow 110 \\ D &\leftrightarrow 111 \end{aligned}$$

On average, we use then, per letter,

$$\underbrace{\frac{1}{2} \times 1 \text{ bit}}_{\substack{\text{if the letter is } A \\ \text{which happens half} \\ \text{the time}}} + \underbrace{\frac{1}{4} \times 2 \text{ bits}}_{\substack{1/4 \text{ of the time} \\ \text{we have } B \\ \leftrightarrow 2 \text{ bits}}} + \underbrace{\frac{1}{8} \times 3 \text{ bits}}_{\substack{\text{the cases where} \\ \text{we have } C}} + \underbrace{\frac{1}{8} \times 3 \text{ bits}}_{\substack{\text{cases for } D}} = \frac{1}{2} + \frac{1}{2} + \frac{3}{8} + \frac{3}{8} = 1.75 \text{ bits.}$$

This new correspondence thus uses, on average, 1.75 bits per letter instead of 2: it is more efficient!

Note that

$$\begin{aligned} -\log_2 p_A &= -\log_2 \frac{1}{2} = 1; \\ -\log_2 p_B &= -\log_2 \frac{1}{4} = 2; \\ -\log_2 p_C &= -\log_2 p_D = -\log_2 \frac{1}{8} = 2, \end{aligned}$$

i.e. this last correspondence assigns to each symbol a number of bits equal to $-\log_2$ (its probability). In other words, we assigned to the symbol i the number of bits $b_i = -\log_2 p_i$; the average number of bits used per letter is then $\sum_{i \in \{A,B,C,D\}} p_i \times b_i = -\sum_{i \in \{A,B,C,D\}} p_i \log_2 p_i = H$.

We shall see later that this is the most efficient coding that we could have carried out: no scheme, however clever its bit-assignment, can beat the “entropy-coding.” In many applications, considerable effort is spent on designing practical coding schemes that get close to the bound indicated by the entropy.

- Note that entropy gives a value for only the “potential for information,” or the “innovation potential” as we go from one letter in the text to the next. A high value for H does not mean that there is necessarily a great deal of information in the observed text or sequence. For instance, if we toss a fair coin and denote the sequence of ‘heads’ and ‘tails’ that we obtain by H ’s and T ’s, then $p_H = p_T = 1/2$, so that the entropy is

$$2 \frac{1}{2} \left(-\log_2 \frac{1}{2} \right) = 1,$$

which is the maximum value that H could achieve, yet that random sequence conveys no information whatsoever...

- It is clear that in order to do more interesting mathematics in this framework, we need a good, solid grounding of

- probability;
- random variables;
- stochastic processes.

The next few lectures will provide such a grounding, and then we will be equipped to tackle information theory.

[Here came the lectures about probability theory; those theoretical results will not be part of the exam, but you should of course be able to handle applications!]

3 Source coding

We'll see that the “entropy” plays a basic role in several aspects of information theory, and especially in communications. The first one is *source coding*, the means by which a text in English (or Sanskrit, or in mathematical symbols, or whatever) is translated into sequences that use only symbols that can be stored in a (computer) memory. (Typically, these are sequences of 0s and 1s.)

More formally:

If \mathcal{A} is our alphabet for the *source*, and \mathcal{B} is the alphabet for coding, then a *code* \mathcal{C} is a map

$$\mathcal{C} : \mathcal{A} \longrightarrow \mathcal{B}^* = \cup_{n \in \mathbb{N}} \mathcal{B}^n$$

where $\mathcal{B}^n = \{(b_1, b_2, \dots, b_n); b_i \in B \text{ for } i = 1, \dots, n\}$

For each $a \in \mathcal{A}$, we have that $\mathcal{C}(a)$ lies in one well-determined \mathcal{B}^n ; this n is called *the length of a* (with respect to the code \mathcal{C}), or *the length of the codeword $\mathcal{C}(a)$* , and denoted by $L(a)$:

$$\mathcal{C}(a) \in \mathcal{B}^{L(a)} \quad \text{or} \quad \mathcal{C}(a) = (b_1, \dots, b_{L(a)})$$

for appropriately chosen $b_1, \dots, b_i, \dots, b_{L(a)} \in \mathcal{B}$

If $a, \tilde{a} \in \mathcal{A}$ are such that $L(a) < L(\tilde{a})$ and

$$\begin{aligned} \mathcal{C}(a) &= (b_1 \dots b_{L(a)}) \\ \mathcal{C}(\tilde{a}) &= (b_1 \dots b_{L(a)} b_{L(a)+1} \dots b_{L(\tilde{a})}) \end{aligned}$$

(i.e. their first $L(a)$ elements coincide), then $\mathcal{C}(a)$ is called a *prefix* of $\mathcal{C}(\tilde{a})$; notation: $\mathcal{C}(a) \prec \mathcal{C}(\tilde{a})$.

Codes that are *prefix-free*, i.e. for which one *never* has $\mathcal{C}(a) \prec \mathcal{C}(\tilde{a})$ for $a \neq \tilde{a}, a, \tilde{a} \in \mathcal{A}$, are especially nice: whenever elements (b_1, \dots, b_L) present themselves, at a place where we know a codesord must start, and such that there exists $a \in \mathcal{A}$ for which $\mathcal{C}(a) = (b_1, \dots, b_L)$, we can be confident that this $\mathcal{C}(a)$ is the codeword that comes next, and that the following codeword must start just after b_L .

Is Morse code prefix-free? Not if you just look at the dots and dashes! For instance, $\cdot \cdot \cdot$ could just as well mean *EEE* as *S* (this is possible because the codeword for *E* is a prefix for the codeword for *S*). If you also take into account the 3-unit silence at the end of each letter, compared with the 1-unit silence between dots and/or dashes within the codeword for a single letter, than the situation is different: since no letter has a Morse codeword that contains a 3-unit silence, the Morse-code-with-silences (which really has *three* symbols: dots, dashes, and silences) is prefix-free.

The first prefix-free code for English (or any other language system in general) was invented by David Huffman, then a graduate student at MIT, and is now called *Huffman coding*.

In the Huffman code that corresponds to only English lower case letters, no punctuation, we have, for instance,

A=011	B=00001	...
N=1110	...	T=110
...	Y=00000	

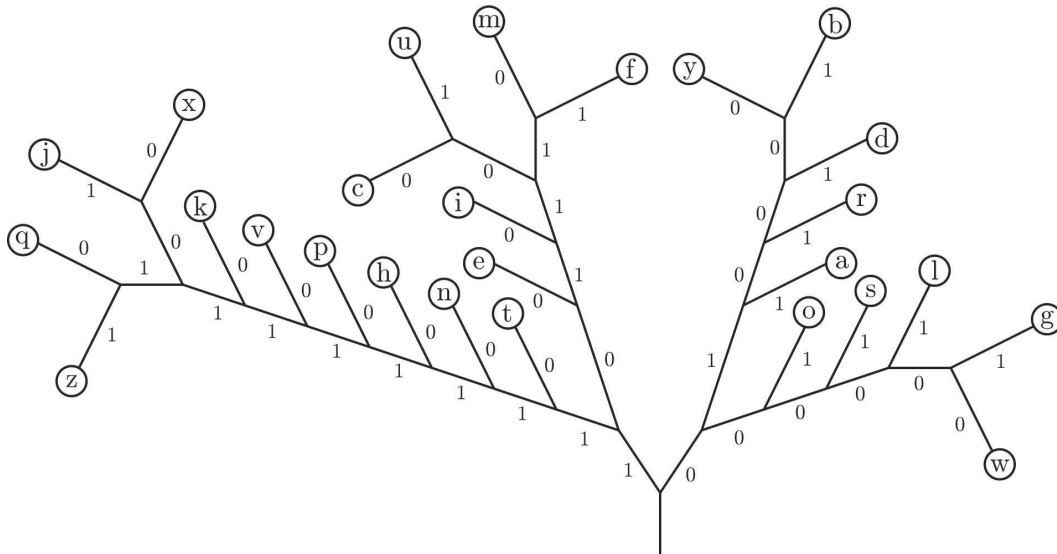
This is prefix free (at least for these 5 letters, we can see this by simple inspection).

It allows us to immediately decode:

Exercise: try it out on 110011111000000011 .

How does one find the Huffman code for an alphabet?

The code is completely determined by a binary tree, as follows. Let's look at the binary tree for the Huffman code, below. We start from the root. It splits into two branches; we label the left branch with a 1 and the right branch with a 0. Each of these branches splits again; to label these "next generation" branches we take the label of the mother branch and attach a 1 for the left "child" branch and a 0 for the right. We keep doing this, following the tree. When a branch stops without splitting further, we say that we have arrived at a leaf. To each leaf a letter is assigned. (See Figure.) This letter is encoded by the path taken at the different branchings that led to this leaf. For instance, to get to the leaf with letter A we first take the right branch up (0), then left (1) then left again (1) and then we have already reached the leaf; so A is encoded as 011.



Tree for Huffman coding of the English alphabet

Take now the two letters F=101111 and C=101100. What is the largest string that is a prefix for both? We can see that it is 1011 — indicating the common path followed on the way from the root to the leaves F and C until the point where the two paths separate. This tree structure explains why in this code no codeword is a prefix to another codeword: codewords

represent leaves, and from a leaf we cannot go to any other leaf, since the branching process stopped. So this code encodes uniquely; decoding will be unambiguous.

This argument would work for any binary tree, as long as we attach the letters to leaves only. Clearly, there is more to the Huffman tree than this: for instance, the letter e is encoded by 100, while q (a very infrequent letter) has a much longer codeword, namely 111111110: ten bits versus the three used for e: the Huffman code seems to adapt the length of the codewords to the frequency with which letters are used in English.

In fact, to *build* the binary tree in the figure above, one uses the table of frequencies of the letters in English. We saw such tables earlier; here is another one (the one used for this figure).

LETTER	STANDARD FREQUENCY	LETTER	STANDARD FREQUENCY
a	.0761	n	.0711
b	.0154	o	.0765
c	.0311	p	.0203
d	.0395	q	.0010
e	.1262	r	.0615
f	.0234	s	.0650
g	.0195	t	.0933
h	.0551	u	.0272
i	.0734	v	.0099
j	.0015	w	.0189
k	.0065	x	.0019
l	.0411	y	.0172
m	.0254	z	.0009

Frequencies of Occurrence of Individual Letters in English Text

The two least frequently encountered letters are Z (about 9 per 10,000), or with a frequency of .0009) and Q (about 1 per 1000 – frequency .001). We start drawing our tree from these two letters down: they will be neighboring leaves, coming from the same branching. (It won't matter which one we label by 0 or 1; both leaves will have labels of the same length.) Now forget about the distinction between Z or Q – think of all the texts as written on a typewriter that had only 25 possible keys, with the same symbol V_1 (standing here for “Z or Q”) replacing all Zs and all Qs. Then this new letter V_1 occurs with frequency $.0019 = .0009 + .001$ (the sum of the frequencies of Z and Q). For this new alphabet of 25 letters, we can proceed as before: take the least frequent letters J (.0015) and X (.0019) (you can take V_1 instead of X since they have the same frequency .0019 – it does not matter – we would get another tree that is as optimal as the tree shown above), draw the branches

down from them to the common vertex V_2 , which can again be thought of as standing for a new letter “J or X” that would be assigned the frequency .0034 ($= .0015 + .0019$), and replace the two letters J and X by V_2 , reducing our alphabet to 24 letters. Go on: in the reduced table of 24 letters we can again pick the least frequent letters, now V_1 and V_2 , and proceed as before. In this way we draw the tree from the leaves down; the most frequently occurring letters will only start coming into play after many reductions, and they will end up with much shorter labels than Z or X.

The end result is that we use fewer bits for frequent letters than for less frequent letters, resulting in compression. Taking into account the frequencies of the letters in standard English, this means that in a long text, we will use about $3.9 \times n$ bits for n symbols (lower case letters and spaces), which is a compression over the $5 \times n$ bits we would have needed without this scheme! Huffman coding was used in most compression schemes until recently, and is still very widely used. You can of course use it for other things than text as well—the idea is always that you determine the statistics of the different symbols, and then encode the most frequent symbols with fewer bits than the less frequent ones.

Using the frequencies of only letters is, in fact, suboptimal. The letters t and e are very frequent in English, but if we looked at frequencies for 3-letter groups, then we could fine-grain this, and find the group “the” is *very* frequent. Of course, this means that our alphabet \mathcal{A} has $26^3 = 17,576$ entries instead of only 26... But the result would give a much better impression of the structure of English sentences. We’ll come back to this later, in block coding. One can also, for each letter, look at its probability, *conditioned* on the preceding letters. To see how this adapts to the structure of language, let’s look at the results of the following simple experiment.

First, we generate a random text, given simply the probabilities for individual letters (including capitalization, punctuation, spacing, ...). This would be a “zero-th” order: no influence of preceding letters. Next, we look at the probabilities for the different letters, given the preceding letter, and we generate a random text where each letter is drawn according to this first-order approximation: only the one letter that just precedes has an influence.

By incorporating 2, 3, 4, ... letters we can likewise get second, third, fourth... order examples.

The following are taken from Shannon’s original article “A Mathematical Theory of Communication,” which is very readable despite its age (it appeared in 1948!), and which can easily be found on the web (e.g. at www.essrl.wustl.edu/~jao/itrg/shannon.pdf)

1. Zero-order approximation (symbols independent and equi-probable).
XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGXYD
QPAAMKBZAACIBZLHJQD
2. First-order approximation (symbols independent but with frequencies of English text).

OCR0 HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI
ALHENHTTPA OOBTTVA NAH BRL

3. Second-order approximation (digram structure as in English).
ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY
ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO
TIZIN ANDY TOBE SEACE CTISBE
4. Third-order approximation (trigram structure as in English),
IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID
PONDENOME OF DEMONSTURES OF THE REPTAGIN IS
REGOACTIONA OF CRE
5. First-Order Word Approximation. Rather than continue with tetragram,
 \dots , n -gram structure it is easier and better to jump at this point to word
units. Here words are chosen independently but with their appropriate
frequencies.
REPRESENTING AND SPEEDILY IS AN GOOD APT OR
COME CAN DIFFERENT NATURAL HERE HE THE A
IN CAME THE TO OF TO EXPERT GRAY COME TO
FURNISHES THE LINE MESSAGE HAD BE THESE.
6. Second-Order Word Approximation. The word transition probabilities are
correct but no further structure is included.
THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH
WRITER THAT THE CHARACTER OF THIS POINT IS
THEREFORE ANOTHER METHOD FOR THE LETTERS THAT
THE TIME OF WHO EVER TOLD THE PROBLEM FOR
AN UNEXPECTED

Note that even when only 3-grams are used, the text (although meaningless) already looks a lot like real language!

In the next section we return to simple prefix-free codes (without conditioning).

4 Shannon's noiseless coding theorem

Each prefix-free code has an “expected length” per codeword (i.e. the average length per letter in \mathcal{A}), defined by

$$E(L) = \sum_{a \in \mathcal{A}} p(a)L(a) .$$

For prefix-free codes we can prove the following nice theorem:

Theorem 3

$$E(L) = \sum_{a \in \mathcal{A}} p(a)L(a) \geq H(\text{prob. distr. in } \mathcal{A}) = - \sum_{a \in \mathcal{A}} p(a) \log_2 p(a)$$

Moreover, there is a prefix-free code with length function $L(a) = \lceil -\log_2 P(a) \rceil$; its length function satisfies

$$E(L) < H(P) + 1.$$

Notes:

- we have implicitly assumed here that $\mathcal{B} = \{0, 1\}$; for another \mathcal{B} we would take logarithms in base $B = \#\mathcal{B}$. We shall assume $\mathcal{B} = \{0, 1\}$ in all that follows, unless it is stated explicitly that we choose another \mathcal{B} .
- $\lceil x \rceil = \min\{n \in \mathbb{Z}; n \geq x\}$
↙
“ceiling of x ”

Proof the proof uses a lemma, the *log-sum-inequality*, that we prove first:

Lemma 4 if p_1, \dots, p_M and q_1, \dots, q_M are all in \mathbb{R}_+ , then

$$\sum_{i=1}^M p_i \log \frac{p_i}{q_i} \geq \left(\sum_{i=1}^M p_i \right) \log \frac{\sum_{i=1}^M p_i}{\sum_{i=1}^M q_i}$$

Proof left as exercise! ■

The proof also uses another lemma, the *Kraft inequality*.

Lemma 5 a function $L : \mathcal{A} \rightarrow \mathbb{N}$ is the length function of some prefix-free code if and only if it satisfies the inequality

$$\underbrace{\sum_{a \in \mathcal{A}} 2^{-L(a)} \leq 1}_{\substack{\uparrow \\ \text{this is the Kraft inequality}}}$$

Proof:

- If $C : \mathcal{A} \rightarrow \mathcal{B}^*$ is a prefix-free code, define then for each $a \in \mathcal{A}$ the number $t(a) \in [0, 1]$ as follows: if $C(a) = (b_1, \dots, b_{L(a)})$, then $t(a) = \sum_{n=1}^{L(a)} b_n 2^{-n} = 0.b_1 b_2 \dots b_{L(a)}$ in binary notation.

For $a \neq \tilde{a}$, we have $C(a) \not\prec C(\tilde{a})$, so that $t(\tilde{a}) \notin [t(a), t(a) + 2^{-L(a)}] \Rightarrow$ the intervals $[t(a), t(a) + 2^{-L(a)})$, for $a \in \mathcal{A}$, must all be disjoint \Rightarrow their total length is ≤ 1 .

- Suppose L satisfies the Kraft inequality. Number the elements in \mathcal{A} so that L is increasing, i.e.

$$L(a_{n+1}) \geq L(a_n).$$

Then $T(n) := \sum_{i < n} 2^{-L(a_i)}$ is a number in $[0, 1]$ with a dyadic expansion that we write out using $L(a_n)$ bits: $T(n) =: 0.b_1(n) \dots b_{L(a_n)}(n) = \sum_{i=1}^{L(a_n)} b_i(n) 2^{-i}$. (Note: it is possible that $T(n)$ could also be written using fewer than $L(a_n)$ bits; then we just put the “unnecessary” bits to zero, because we *must* have $L(a_n)$ bits for a_n .) Set now $\mathcal{C}(a_n) = b_1(n) \dots b_{L(a_n)}(n) \in \mathcal{B}^N$. This defines a code $\mathcal{C} : \mathcal{A} \rightarrow \mathcal{B}^*$. An easy argument shows that different elements in \mathcal{A} are mapped to different codewords: if $\mathcal{C}(a_n) = \mathcal{C}(a_{n+j})$, with $j \geq 1$, then it would follow that $T(n) = T(n+j)$, or $\sum_{i=0}^{j-1} 2^{-L(a_{n+i})} = 0$, which is impossible. It is also easy to check that the code must be prefix-free: if $\mathcal{C}(a_n)$ is a prefix for $\mathcal{C}(a_{n+j})$, then it follows that the binary expansion of $T(n+j) - T(n) = \sum_{i=0}^{j-1} 2^{-L(a_{n+i})}$, written out using $L(a_{n+j})$ bits, has its first $L(a_n)$ bits equal to zero, which is impossible, since $T(n+j) - T(n) \geq 2^{-L(a_n)}$. ■

We are now ready for the

Proof of Shannon’s noiseless theorem:

- Apply the first lemma to $p(a)$, $a \in \mathcal{A}$, and to $2^{-L(a)}$, $a \in \mathcal{A}$.

$$\begin{aligned} \Rightarrow \sum_{a \in \mathcal{A}} p(a) \log \frac{p(a)}{2^{-L(a)}} &\geq 1 \log \frac{1}{\sum_{a \in \mathcal{A}} 2^{-L(a)}} \\ \Rightarrow \sum_{a \in \mathcal{A}} p(a) \log p(a) + \sum_{a \in \mathcal{A}} p(a) L(a) &\geq 0 \\ \Rightarrow E(L) \geq - \sum_{a \in \mathcal{A}} p(a) \log p(a) &= H \end{aligned}$$

- Since $L(a) = \lceil -\log P(a) \rceil$ satisfies the Kraft inequality, we need to take only that prefix-free code. ■

Earlier, we saw Huffman’s strategy for building a prefix-free code. It turns out that this is an optimal strategy, as shown by the following lemma:

Lemma 6 *Suppose $\mathcal{C}' : \mathcal{A} \rightarrow \mathcal{B}^*$ is a prefix-free code with optimal expected length per codeword. Then, if a, b are the elements of \mathcal{A} with lowest probability, and if $p(c) > \max[p(a), p(b)]$ for all other c in \mathcal{A} , the lengths of $\mathcal{C}'(a)$ and $\mathcal{C}'(b)$ must be equal. In addition, we also have $\ell(\mathcal{C}'(a)) = \ell(\mathcal{C}'(b)) = \max_{c \in \mathcal{A}} \ell(\mathcal{C}'(c))$.*

Proof:

- Suppose $\ell'(a) := \ell(\mathcal{C}'(a)) < \ell(\mathcal{C}'(b)) := \ell'(b)$. (We want to show that this leads to a contradiction.)
- Then there must exist an element c in \mathcal{A} such that $\ell(\mathcal{C}'(c)) \geq \ell(\mathcal{C}'(b))$. (Because, if all other codewords $\mathcal{C}'(c)$ were strictly shorter than $\ell(\mathcal{C}'(b))$, then we could shorten the codeword $\mathcal{C}'(b)$ of b and still have a valid prefix-free code, contradicting the length-optimality of \mathcal{C}' .)
- Then we could exchange the codewords for a and c ; since $p(c) > p(a)$ this would lead to a code \mathcal{C}'' with strictly shorter expected codeword length than \mathcal{C}' , contradicting the length-optimality of \mathcal{C}' .
- We have thus established that $\ell(\mathcal{C}'(a)) = \ell(\mathcal{C}'(b))$.
- If now there were a $c \in \mathcal{A}$ with larger $\ell(\mathcal{C}'(a))$, then we could lower the expected length by exchanging a and c . Since this contradicts the optimality of \mathcal{C}' , it is impossible. ■

Exercise: Why does this Lemma imply that the Huffman tree-building strategy is optimal?

What if one allows codes that are not prefix-free? How much would expected length improve?

Theorem 7 *For arbitrary (not necessarily prefix-free) codes, one has*

$$E(L) = \sum_{a \in \mathcal{A}} p(a)L(a) \geq H - \log \log(\#\mathcal{A}).$$

Note: This shows that the entropy of \mathcal{A} (with its probability distribution) tells us a lot about how “compressible” a text with alphabet \mathcal{A} is: H plays a role both for prefix-free codes and codes that are more general. We’ll have to come back to the role of the extra terms in the lower bounds.

Proof

- Clearly, we make the left hand side as small as possible by using code-words that are as short as possible. Since $|\mathcal{B}^N| (= \#\mathcal{B}^N) = 2^N$, we shall thus pick codewords from $\mathcal{B}^1, \mathcal{B}^2, \mathcal{B}^3, \dots$ consecutively until we have a sufficient number, i.e. until we have $|\mathcal{A}| (= \#\mathcal{A})$ of them.
- So suppose $|\mathcal{A}| = 2 + 2^2 + \dots + 2^m + 4 = 2^{m+1} - 2 + r$ where $0 \leq r < 2^{m+1}$. Then we will have 2 codewords of length 1, 4 codewords of length 2, *ldots*, 2^ℓ codewords of length ℓ , *ldots*, 2^m codewords of length m and r codewords of length $m + 1$.

Let us estimate $\sum_{i=1}^n 2^{-L(a_i)}$:

$$\begin{aligned} \sum_{i=1}^n 2^{-L(a_i)} &= \sum_{\ell=1}^m 2^\ell 2^{-\ell} + r 2^{-(m+1)} \\ &= m + r 2^{-(m+1)} \end{aligned}$$

- On the other hand,

$$\begin{aligned} \log_2 |\mathcal{A}| &= \log_2 [2^{m+1} - 2 + r] \\ &= m + \log_2 \left[2 + \frac{r-2}{2^m} \right] \end{aligned}$$

- In this step, we'll show that

$$\sum_{i=1}^n 2^{-L(a_i)} \leq \log_2 |\mathcal{A}|.$$

From what precedes, it suffices to show that

$$r 2^{-(m+1)} \leq \log_2 \left[2 + \frac{r-2}{2^m} \right]$$

For $r = 0$, the left hand side is 0, and, since $m \geq 1$, the right hand side is $\geq 0 \Rightarrow$ OK.

For $r = 1$, we have to prove that $2^{-(m+1)} \leq 1 + \log_2(1 - 2^{-(m+1)})$ for $m \geq 1$. If we set $f(x) = 1 - x + \log_2(1 - x)$, then $f(0) = 1$, $f(1/4) = 3/4 + \log_2(3/4) > 0$, and $f'(x) = -1 - [(1-x) \ln 2]^{-1} < 0$ on $[0, 1/4]$. This shows that $f(x) > 0$ for all x in $[0, 1/4]$, establishing the desired inequality for $r = 1$.

For $r \geq 2$, $\log_2 \left[2 + \frac{r-2}{2^m} \right] \geq 1$, whereas $r 2^{-(m+1)}$ is strictly smaller than 1, since $r < 2^{(m+1)}$.

- We now apply the log-sum inequality:

$$\begin{aligned} \sum_{i=1}^n p_i [\log_2 p_i + L(a_i)] &\geq \left[\sum_{i=1}^n p_i \right] \log_2 \frac{\left[\sum_{i=1}^n p_i \right]}{\left[\sum_{i=1}^n 2^{-L(a_i)} \right]} \\ \Rightarrow -H + \sum_{i=1}^n p_i L(a_i) &\geq 1 \log_2 \frac{1}{\log_2 |\mathcal{A}|} = -\log \log |\mathcal{A}| \\ \Rightarrow \sum_{i=1}^n p_i L(a_i) &\geq H - \log \log |\mathcal{A}| \end{aligned}$$

■

How bad is this extra term $-\log \log |\mathcal{A}|$?

Let's think again of texts in English, in the simple case where we drop capitalization BUT use also some punctuation (the 6 symbols , . ? ! “ ”), so that we have 32 symbols in total. One can then very easily represent every symbol by 5 bits (because $2^5 = 32$), in a code that would clearly not be prefix-free. (If we have capitalization and allow few more punctuation marks such as (and) and : and ; , we can make do with 6 bits; special letters with accents such as á, ê or diacritic marks ü, Å, Ø and standard symbols used in texts (% , . . .) can all be accommodated with 8 bits.) Even for $|\mathcal{A}| = 32$, we already have $\log_2 \log_2 |\mathcal{A}| = \log_2 5 > 2$, which is not negligible at all compared with $H \dots$ For instance, for the frequency of letters given at the start of the course, we get

$$H \simeq - \sum_{i=1}^{26} p_i \log_2 p_i = 4.1666.$$

It follows that there can be an appreciable difference between H and $H - \log_2 \log_2 |\mathcal{A}|$. However, a general (non-prefix-free) code is impossible to decode, because it is not clear where one codeword starts and another finishes. Maybe we could get around this by reserving one particular string to mark the end of a codeword, and the start of another one ...? In that case we'd better use very long codewords, so that these extra pieces of bitstring wouldn't matter much. This would mean working with *blocks* of N consecutive “letters”; this amounts to using the alphabet $\mathcal{A}^N := \mathcal{A} \times \mathcal{A} \times \dots \times \mathcal{A}$, in which the elements are N -tuples of letters, $\mathbf{a} = (a_1, a_2, \dots, a_N)$, with probabilities $\mathbf{p}(\mathbf{a}) = p(a_1) \cdot p(a_2) \cdot \dots \cdot p(a_N)$. The corresponding entropy is

$$\begin{aligned} \mathbf{H} &= \sum_{\mathbf{a} \in \mathcal{A}^N} \mathbf{p}(\mathbf{a}) \log_2 (\mathbf{p}(\mathbf{a})) = \sum_{a_1, a_2, \dots, a_N \in \mathcal{A}} p(a_1) p(a_2) \dots p(a_N) \left[\sum_{i=1}^N \log_2 (p(a_i)) \right] \\ &= N \left[\sum_{a \in \mathcal{A}} p(a) \log_2 (p(a)) \right] + N H \quad . \end{aligned}$$

On the other hand, $\#(\mathcal{A}^N) = (\#\mathcal{A})^N$, and thus

$$\log_2 [\log_2 (\#(\mathcal{A}^N))] = \log_2 [N \log_2 (\#\mathcal{A})] = \log_2(N) + \log_2 \log_2(\#\mathcal{A}) .$$

The lower bound on the expected bit length (per letter in the *original* alphabet \mathcal{A}) is thus

$$\frac{1}{N} \mathbf{E}(\mathbf{L}) = \frac{1}{N} \{ \mathbf{H} - \log_2 [\log_2 (\#(\mathcal{A}^N))] \} = H - \frac{1}{N} \log_2 \log_2(\#\mathcal{A}) - \frac{\log_2(N)}{N} .$$

This means that for large N (which must be used in this approach, because only the use of very long codewords can offset the loss incurred by the use of a “STOP” string), we essentially are limited again to the lower bound of H bits per (source) alphabet letter.

5 Shannon's noisy coding theorem

All the above was for “lossless” coding: we always assumed that different elements \mathbf{a} of \mathcal{A}^N are encoded by different codewords. Could we gain by allowing some errors? After all, the string !!!!!!!!!!!!!!! (16 exclamation marks) is not only extremely unlikely, as will be reflected by its very low probability, but also not really essentially different from !!!!!!!!!!!!!!! (15 exclamation marks), and we will not really care if the encoding is done slightly sloppily, so that these two cannot be distinguished from each other. Frankly, we wouldn't even care if it could not be distinguished from other very low probability strings such as ?????????????????? or ?????????????????? . By cutting unimportant corners of this type, we may hope that, by being clever, we could *really* bring down the average number of bits per letter.

Let's try to phrase this more mathematically.

Given a tolerance for “errors” or “failures” $\delta > 0$, we denote by $\mathcal{A}_{N,\delta}$ the smallest subset of \mathcal{A}^N such that

$$\mathbf{p}(\mathcal{A}^N \setminus \mathcal{A}_{N,\delta}) < \delta ,$$

where the probability \mathbf{p} on \mathcal{A}^N is defined by setting, for each $\mathbf{a} = (a_1, a_2, \dots, a_N) \in \mathcal{A}^N$, $\mathbf{p}(\mathbf{a}) = p(a_1) \cdot p(a_2) \dots p(a_{N-1}) \cdot p(a_N)$.

Equivalently, $\mathcal{A}_{N,\delta}$ is the smallest subset of \mathcal{A}^N so that

$$\mathbf{p}(\mathcal{A}_{N,\delta}) \geq 1 - \delta .$$

Our hope is thus that we might be able to manage, by trying to encode only $\mathcal{A}_{N,\delta}$ (and giving up on the strings of N letters in $\mathcal{A}^N \setminus \mathcal{A}_{N,\delta}$), to make our code *much* more efficient, at the cost of not being able to encode a set of probability δ .

Shannon's lossy source coding theorem shows that this is a vain hope ...

More precisely, using the definitions above,

Theorem 8 *Suppose we have \mathcal{A} , and a probability distribution p on \mathcal{A} with entropy H . Then, for arbitrary $\epsilon > 0$, $\delta \in]0, 1[$, there exists N_0 such that, for all $N \geq N_0$,*

$$\left| \frac{\log_2(\#\mathcal{A}_{N,\delta})}{N} - H \right| \leq \epsilon .$$

Before we proceed with the proof, let us discuss the meaning of this result. The quantity $\log_2(\#\mathcal{A}_{N,\delta})$ is the number of bits we would need to encode the elements of $\mathcal{A}_{N,\delta}$ if we just numbered them in the binary system, and allocated to each element the same number of bits. The noisy source coding theorem thus says that with this (fairly stupid) way of coding $\mathcal{A}_{N,\delta}$ we would not, for large N , “beat the entropy”, since the number of bits per letter in \mathcal{A} , or $N^{-1} \log_2(\#\mathcal{A}_{N,\delta})$ would be close to H .

What if we were not as stupid in encoding $\mathcal{A}_{N,\delta}$? On $\mathcal{A}_{N,\delta}$ we inherit a probability distribution $\tilde{\mathbf{p}}$ from the distribution \mathbf{p} on \mathcal{A}^N , by setting

$$\forall \mathbf{a} \in \mathcal{A}_{N,\delta}, \quad \tilde{\mathbf{p}}(\mathbf{a}) = \frac{\mathbf{p}(\mathbf{a})}{\mathbf{p}(\mathcal{A}_{N,\delta})},$$

with corresponding entropy

$$\begin{aligned} \tilde{\mathbf{H}} &= \sum_{\mathbf{a} \in \mathcal{A}_{N,\delta}} \tilde{\mathbf{p}}(\mathbf{a}) \log_2(\tilde{\mathbf{p}}(\mathbf{a})) \\ &= [\mathbf{p}(\mathcal{A}_{N,\delta})]^{-1} \sum_{\mathbf{a} \in \mathcal{A}_{N,\delta}} \mathbf{p}(\mathbf{a}) \log_2(\mathbf{p}(\mathbf{a})) - \log_2[\mathbf{p}(\mathcal{A}_{N,\delta})] \\ &\leq (1 - \delta)^{-1} \sum_{\mathbf{a} \in \mathcal{A}_{N,\delta}} \mathbf{p}(\mathbf{a}) \log_2(\mathbf{p}(\mathbf{a})) - \log_2[\mathbf{p}(\mathcal{A}_{N,\delta})]. \end{aligned}$$