A little formality about coding

An **alphabet** Σ is a finite set. Σ^* is all finite **strings** composed of characters from Σ. That is, Σ^* is all finite ordered lists of elements of Σ.

The alphabet might be $\Sigma = \{0, 1\}$, or $\Sigma = \{a, b, \ldots, y, z\}$. The nature of the characters does not matter, only the *number* of characters in the alphabet.

A **code** or **encoding** f of a (memoryless) source $S = X_1, X_2, \ldots$ (emitting sourcewords in a set W) into **codeword** strings over an alphabet Σ is a function

$$f : W \to \Sigma^*$$

Extend f with respect to concatenation of strings: define

$$f(\text{ concatenation } w_1 w_2 \ldots w_n)$$

$$= \text{ concatenation } f(w_1) f(w_2) \ldots f(w_n)$$

Example: Encoding English into dots and dashes for telegraphy, the alphabet is $\Sigma = \{\text{dot}, \text{dash}\}$. The 'words' W are the usual English alphabet. (This illustrates abstraction of the notion of 'word'.) An important feature is that commonly used letters such as 'e' and 't' have shorter expressions in Morse.

Example: Encoding of the English alphabet (with numerals, punctuation, ...) into ASCII is an encoding to numbers 0–255. Source words are single characters rather than 'words' in the ordinary sense. For code alphabet Σ we have choices. If numbers 0–255 are written in **binary**, then $\Sigma = \{0, 1\}$ and encoding of each 'word' takes 8 characters. If numbers 0–255 are written in **octal** then $\Sigma = \{0, 1, 2, \ldots, 7\}$, and encoding a source word takes 3 characters. If numbers 0–255 are written in **decimal** then $\Sigma = \{0, 1, \ldots, 8, 9\}$, and encoding of each source word may take 3 characters. If numbers 0–255 are written in **hexadecimal**, then $\Sigma = \{0, 1, \ldots, 8, 9, A, B, C, D, E, F\}$ and encoding of a source character takes only 2 characters.

Example: Braille encodes the 26-letter alphabet (with numerals, some punctuation, and a few common words) into a 3-by-2 pattern of raised-or-not dots. The 3-by-2 grid gives 6 choices of whether to raise a dot, so there are 2^6 code words.

Example: Abbreviations used in ordinary English are examples of coding. We may take as the set of 'words' W the set of genuine words in English, and Σ to be the usual alphabet with numerals and punctuation. We can define many different encoding maps $f : W \to \Sigma^*$. For example,

$$f(\text{word}) = \begin{cases} \text{'St.'} & \text{if word} = \text{'Street'} \\ \text{'Ave.'} & \text{if word} = \text{'Avenue'} \\ \text{'Blvd.'} & \text{if word} = \text{'Boulevard'} \\ \text{'Rd.'} & \text{if word} = \text{'Road'} \\ \text{'word'} & \textit{otherwise} \end{cases}$$

Depending upon the context 'St.' may be an abbreviation for either 'Street' or for 'Saint'. 'IP' may be either 'intellectual property' or 'internet protocol'.

Uniquely decipherable codes
Prefix/instantaneous codes

We only want **uniquely decipherable** codes, in which two different messages will never be encoded the same way. No information is lost. The encoding function $f : W \to \Sigma^*$ is **injective**.

Given two strings

$$s = s_1 s_2 \dots s_m \qquad t = t_1 t_2 \dots t_n$$

in Σ^*, s is a **prefix** of t if s is an initial segment of t, that is, if $m \leq n$ and

$$t_1 = s_1, t_2 = s_2, \dots, t_m = s_m$$

A code $f : W \to \Sigma^*$ is **instantaneous** or **prefix** if for all source words w, w'

$$f(w) \text{ is not a prefix of } f(w') \quad \text{for} \quad w \neq w'$$

An instantaneous code can be **decoded** without **lookahead**. That is, the decoding does not require waiting to see what comes later.

Natural languages do *not* have this property, since often word occurs as an initial fragment of a longer word. For example, 'red' is a prefix of 'reduction'.

Example: The code with words 11, 01, 110, and 001 is *not* a prefix code, because the first codeword 11 is the first part of the third codeword 110.

Example: If all codewords are of the same length, then we know when a codeword is completed. But this wastes the possibility of using short code words for common source words!

Example: We could make a very wasteful code by having the codewords be strings of 0's ended by a 1. But then we'd only have n codewords of length $\leq n$, far short of the total 2^{n+1} binary codewords of lengths $\leq n$.

Example: We could use a pattern of symbols (such as two 1s in a row) to signal the end of a codeword, and to guarantee the prefix property. But still instead of having 2^{n+1} binary codewords of length less than or equal n, we would have far fewer.

Example: Find all possible decodings of the message 11001 which was encoded by the scheme $a = 1$, $b = 10$, $c = 01$.

Start from the left and decode (in all possible ways) character by character, considering all possibilities, and backtracking both if failure is encountered and to find all other possible decodings after a successful complete decoding is found.

The partial decoding $a = 1$ matches the initial '1', and leaves string '1001' remaining to be decoded. In fact, there is no other possible beginning of a partial decoding, since neither the b nor c match the initial 11.

Then we have a choice: another a would match the next '1', and also a b would match the next '10'. First we continue with the a, so cumulatively we have partial decoding aa with remaining string '001'.

(*cont'd*: decoding 11001 with
$a = 1$, $b = 10$, $c = 01$ ...)

Ok, partial decoding aa with remaining string '001'.

But now there's a problem, that no one of a, b, c matches the next characters '00'. We must go back. Our system is to undo as little as possible and try the other choice(s). Thus, we discard the second a in the partial decoding aa, and go back to partial decoding a with string remaining 1001. And take the choice we did not take before, to decode the next '10' as b. Thus, the partial decoding is ab and remaining string is '01'.

That remaining string has no choice but to be decoded as c, so there is just one decoding, abc.

Remark: Even though there turned out to be a unique decoding, it should rightly be viewed as somewhat painful to have to backtrack and try again in decoding.

Example: Find all possible decodings of the message 11001 which was encoded by the scheme $a = 1$, $b = 10$, $c = 00$.

Again, start from the left and decode (in all possible ways) character by character, and backtracking *both* if failure is encountered and to find all other possible decodings after each successful complete decoding is found.

The partial decoding $a = 1$ matches the initial '1', and leaves string '1001' remaining to be decoded. In fact, there is no other possible beginning of a partial decoding, since neither the b nor c match the initial 11.

Then we have a choice: another a would match the next '1', and also a b would match the next '10'. First we continue with the a, so cumulatively we have partial decoding aa with remaining string '001'.

Only c matches the next characters '00'. Thus, partial decoding aac with remaining string 1. No choice but to decode that as another a, so *one* complete decoding is $aaca$.

(*cont'd*: decoding 11001 with
$a = 1$, $b = 10$, $c = 00$ ...)

But to find *all* possible decodings, we must go back to the last place where a choice was made, namely after the partial decoding a leaving string '1001', instead of decoding the next '1' as a we could have decoded '10' as b, giving partial decoding ab with remaining string '01'.

But that remaining string '01' has no decoding, so this partial decoding cannot be completed.

So there is just one decoding, $aaca$.

Example: Find all possible decodings of the message 101101 which was encoded by the scheme $a = 1$, $b = 0$, $c = 01$.

Again, start from the left and decode character by character, considering all possibilities, backtracking both for failure and to find all possible decodings.

The initial '1' can be decoded as a leaving '01101' remaining. We can decode the next '0' as b, giving partial decoding ab and leaving '1101'. Can only decode further as aba leaving '101'. Can continue only to $abaa$, leaving '01'. One choice of continuation is $abaab$, leaving '1', allowing only $abaaba$. One complete decoding.

(*cont'd:* 101101 ... $a = 1$, $b = 0$, $c = 01$)

The last choice made was at partial decoding $abaa$, where instead of continuing with b and then necessarily a, we could have continued with c, giving complete decoding $abaac$.

Going back further to a possible choice, after partial decoding a leaving '01101', we could decode the '01' as c, giving ac leaving '101'. The only continuation is to aca leaving '01'. One continuation is $acaba$, and another is $acac$.

Thus, altogether, there are decodings $abaaba$, $acaba$, and $acac$.

Example: Find all possible decodings of the message 10010110101011010, which was encoded by the scheme $a = 10$, $b = 100$, $c = 10101$, $d = 10110$.

We start from the left and decode (in all possible ways) character by character, considering all possibilities, and backtracking both if failure is encountered after partially successful decoding and to find all other possible decodings after a successful complete decoding is found.

[decode 1001011010101011010, with
$a = 10, b = 100, c = 10101, d = 10110$:]
Partial decoding: 'b' leaves '101101010101010'.
Partial decoding: 'bd' leaves '101011010'.
Partial decoding: 'bdc' leaves '1010'.
Partial decoding: 'bdca' leaves '10'.
One complete decoding 'bdcaa'. For more, backtrack to where there was a choice.
Partial decoding: 'bda' leaves '1011010'.
Partial decoding: 'bdad' leaves '10'.
Another complete decoding is 'bdada'. For more, backtrack to where there was a choice.
Partial decoding: 'bdaa' leaves '11010'.
Partial decoding 'bdaa' cannot be continued. For more, backtrack to where there was a choice.
Partial decoding: 'ba' leaves '110101011010'.
Partial decoding 'ba' cannot be continued. For more, backtrack to where there was a choice.
Partial decoding: 'a' leaves '0101101010101010'.
Partial decoding 'a' cannot be continued. For more, backtrack to where there was a choice. No more possibilities remain. Thus, the list of all possible decodings is 'bdcaa', 'bdada'.

Theorem: *(Kraft's inequality.)* Let there be m source words W. Let the encoding alphabet Σ have q characters. A necessary and sufficient condition that there exist an **prefix** code $f : W \to \Sigma^*$ with lengths $\ell_1, \ldots, \ell_m$ is

$$\sum_{i=1}^{m} \frac{1}{q^{\ell_i}} \leq 1$$

Theorem: *(McMillan's inequality.)* A necessary and sufficient condition that there exist a **uniquely decipherable** code $f : W \to \Sigma^*$ with lengths $\ell_1, \ldots, \ell_m$ is

$$\sum_{i=1}^{m} \frac{1}{q^{\ell_i}} \leq 1$$

Corollary: If there is a uniquely decipherable code with prescribed word lengths, then there is an *prefix* (uniquely decipherable) code with those word lengths.

Remark: The precise nature of the source words does not matter in Kraft-MacMillan, only the lengths of the code words.

Example: Is there a *binary* (meaning alphabet $\{0,1\}$) prefix code with lengths $3,3,4,5,6,7,7,7$? By the Kraft-MacMillan inequalities, there exists such a code if and only if

$$\sum_{\text{lengths } \ell_i} (\text{no. elts in alphabet})^{-\ell_i} \leq 1$$

For a binary code $q = 2$, and the proposed lengths are given by the list above. So we are asking whether or not

$$2^{-3}+2^{-3}+2^{-4}+2^{-5}+2^{-6}+2^{-7}+2^{-7}+2^{-7} \leq 1$$

The left-hand side simplifies: use repeatedly $2^{-n}+2^{-n}=2^{-(n-1)}$,

$$2^{-3}+2^{-3}+2^{-4}+2^{-5}+2^{-6}+2^{-7}+2^{-7}+2^{-7}$$

$$= 2^{-2}+2^{-4}+2^{-5}+2^{-6}+2^{-6}+2^{-7}$$

$$= 2^{-2}+2^{-4}+2^{-5}+2^{-5}+2^{-7}$$

$$= 2^{-2}+2^{-4}+2^{-4}+2^{-7} = 2^{-2}+2^{-3}+2^{-7} < 1$$

So, by Kraft-MacMillan, there *is* such a code.

Shannon noiseless coding theorem

Let $f : W \to \Sigma^*$ be a code with source words $w_1, \ldots, w_m$, codewords $f(w_i)$ of lengths ℓ_i, with p_i the probability w_i is emitted. The **average length** of codewords is

$$\text{average length } f = \sum_{i=1}^{m} p_i \, \ell_i$$

Theorem: For memoryless source X with entropy $H(X)$, a uniquely decipherable code $f : W \to \Sigma^*$ has average length

$$\text{average length } f \geq \frac{H(X)}{\log_2 |\Sigma|}$$

There *exists* a code f with

$$\text{average length } f < 1 + \frac{H(X)}{\log_2 |\Sigma|}$$

Remark: This theorem tells the best possible performance, in terms of average word length, of any encoding of a given set W of source words. *Noiseless* means we are ignoring errors.

Example: Source X: there are 5 words with probability 1/10, 5 with probability 1/12, and 5 with probability 1/60. *Estimate* the optimal average word length achievable by a uniquely-decodable binary encoding.

One approach to this would be to compute the Huffman encoding and (which we know to be optimal) and from that compute the optimal average word length. But this is too tedious.

Invoke the Noiseless Coding Theorem: the average word length of an optimal binary (uniquely decodable) code is between $H(X)$ and $H(X)+1$. The entropy of this source (using the definition of entropy) is

$$H(X) = -5 \cdot (1/10) \cdot \log_2(1/10)$$

$$-5 \cdot (1/12) \cdot \log_2(1/12) - 5 \cdot (1/60) \cdot \log_2(1/60)$$

$$\approx 3.64693930571154$$

Thus, we have bounds for average word length of optimal code

$3.65 \leq$ optimal code avg word length $\leq 3.65+1$

Huffman encoding

Huffman's 1953 encoding scheme achieves optimality for noiseless coding, with a fixed choice of sourcewords. Keep in mind that there are many other issues, too, such as optimal *decoding* algorithms.

binary Huffman encoding: Let $W = \{w_1, \ldots, w_n\}$ be the source words with probabilities

$$P(x = w_i) = p_i$$

Suppose (without loss of generality)

$$p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_{n-1} \geq p_n$$

Create the encoding $f : W \rightarrow \{0,1\}^*$ **recursively**: let

$$W' = \{w'_1, w'_2, \ldots, w'_{n-1}\}$$

with

$$w'_1 = w_1, w'_2 = w_2, \ldots, w'_{n-2} = w_{n-2}$$

and w'_{n-1} is a word *representing the case that* w_{n-1} *or* w_n *is emitted*. Make source X' with probabilities

$$P(X' = w'_1) = p_1, \ldots, P(X' = w'_{n-2}) = p_{n-2}$$

and

$$P(X' = w'_{n-1}) = p_{n-1} + p_n$$

Thus, emissions w_{n-1} and w_n by X are combined into a single emission of w'_{n-1} by the new source X'. By induction, let

$$f' : W' \to \{0,1\}^*$$

be a binary Huffman encoding for X'. Define the encoding for source X by

$$f(w_i) = f'(w_i) \quad \text{for } \; i = 1, 2, \ldots, n-2$$

and

$$\begin{aligned} f(w_{n-1}) &= f'(w'_{n-1}) + \text{`0'} \\ f(w_n) &= f'(w'_{n-1}) + \text{`1'} \end{aligned}$$

where the '+' denotes **concatenation of strings**. That is, $f(w_{n-1})$ is obtained by appending a '0' to the encoding $f'(w'_{n-1})$, while $f(w_n)$ is obtained by appending a '1' to the encoding $f'(w'_{n-1})$.

Remark: The source X' has one fewer word in its 'vocabulary' than did the original source X. Thus, if we continue in this manner, the problem of defining the encoding will become the question of defining an encoding on just two source words. It is easy to see that an optimal binary encoding of a two-word vocabulary should encode one of the words as '0' and the other as '1' (or *vice-versa*).

Remark: To estimate the average word length of an optimal code on a given list of probabilities of source words, one *could* do a Huffman encoding and compute directly the average word length. But this is much more work than just using the Shannon estimate, and probably will not give much better information.

Example: Let X be a source emitting 3 words w_1, w_2, w_3 with probabilities $\frac{2}{5}, \frac{3}{10}, \frac{3}{10}$, respectively. The source X' should combine w_2 and w_3 as a single emission of a word w_2' with probability

$$\frac{3}{5} = \frac{3}{10} + \frac{3}{10}$$

Thus, X' emits two words, $w_1' = w_1$ with probability $\frac{2}{5}$ and w_2' with probability $\frac{3}{5}$. Encode X' by

$$\begin{aligned} f'(w_1') &= \text{`0'} \\ f'(w_2') &= \text{`1'} \end{aligned}$$

Encoding f of X is defined in terms of the encoding f' of X' by

$$\begin{aligned} f(w_1) &= f'(w_1) &&= \text{`0'} \\ f(w_2) &= f'(w_2') + \text{`0'} &&= \text{`10'} \\ f(w_3) &= f'(w_2') + \text{`1'} &&= \text{`11'} \end{aligned}$$
