Lecture 4: Huffman Coding, Block Coding, and Entropy Rate

ENGR 76 lecture notes — April 9, 2026

David Tse, Stanford University

Overview

In the previous lecture we proved that for a discrete source X,

$$H(X) \le \bar{\ell}_{\min} \le H(X) + 1,$$

and introduced the Huffman algorithm for constructing an optimal prefix-free code. In this lecture we first review the recursive structure of Huffman coding, and then ask whether one can do even better by coding many source symbols together rather than one symbol at a time. This leads to the notions of block coding, higher-order entropies, and entropy rate.

1 Recall: optimal prefix-free coding

Let

$$X \in \{0, 1, \ldots, M-1\}$$

be a discrete random variable with

$$P(X = i) = p_i, \qquad i = 0, 1, \ldots, M-1.$$

A prefix-free code assigns to letter i a binary codeword of length ℓ_i, and the expected codeword length is

$$\bar{\ell} = \sum_{i=0}^{M-1} p_i \ell_i.$$

The optimization problem is

$$\min_{\{\ell_i\}} \sum_{i=0}^{M-1} p_i \ell_i \qquad \text{subject to the code being prefix-free.}$$

We denote by $\bar{\ell}_{\min}$ the minimum value over all prefix-free codes.

The source coding theorem says

$$H(X) \le \bar{\ell}_{\min} \le H(X) + 1.$$

2 Base case of the Huffman algorithm

When the alphabet has size 2, say

$$\mathcal{X} = \{a, b\},$$

the optimal prefix-free code is immediate:

$$a \mapsto 0, \qquad b \mapsto 1.$$

This is the recursion anchor for Huffman coding.

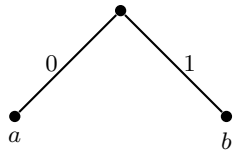

Figure 1: Base case of the Huffman algorithm for an alphabet of size 2.

3 Recursive structure of Huffman coding

Suppose now the alphabet has size M, and the symbol probabilities are ordered as

$$p_a \geq p_b \geq p_c \geq \cdots .$$

A key structural fact is that in an optimal prefix-free code tree, the two letters with the smallest probabilities must appear as siblings at the lowest level.

For the three-symbol case, if

$$p_a \geq p_b \geq p_c,$$

then b and c should be placed deeper than a, and they should be siblings.

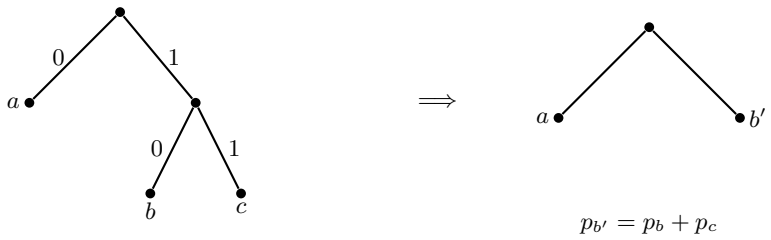

Figure 2: In an optimal prefix-free code, the two least likely letters are siblings at the lowest level. They can be merged into a single superletter b'.

This leads directly to the Huffman recursion:

1. Start with the source alphabet $\mathcal{X}$.
2. Find the two letters with the smallest probabilities.
3. Merge them into a single superletter whose probability is the sum of the two probabilities.
4. Repeat until only two letters remain.
5. Assign bits 0 and 1 at the final split, then work backward through the recursion to assign codewords to all letters.

Thus Huffman coding is a recursive procedure for constructing the optimal prefix-free code.

4 Example of the Huffman recursion

Consider the source

$$\mathcal{X} = \{a, g, c, t\}$$

with probabilities

$$p_a = \frac{1}{2}, \qquad p_g = \frac{1}{4}, \qquad p_c = \frac{1}{8}, \qquad p_t = \frac{1}{8}.$$

The two least likely letters are c and t, so we merge them into a superletter x with probability

$$p_x = p_c + p_t = \frac{1}{4}.$$

Now the reduced alphabet is

$$\{a, g, x\}$$

with probabilities

$$\frac{1}{2}, \qquad \frac{1}{4}, \qquad \frac{1}{4}.$$

Then g and x merge into a new superletter y with probability $p_y = \frac{1}{2}$.

So the final two-letter alphabet is

$$\{a, y\}$$

with probabilities $\frac{1}{2}, \frac{1}{2}$. Hence we assign

$$a \mapsto 0, \qquad y \mapsto 1.$$

Tracing backward,

$$g \mapsto 10, \qquad x \mapsto 11,$$

and then

$$c \mapsto 110, \qquad t \mapsto 111.$$

So the Huffman code is

symbol	probability	codeword	length
a	$\frac{1}{2}$	0	1
g	$\frac{1}{4}$	10	2
c	$\frac{1}{8}$	110	3
t	$\frac{1}{8}$	111	3

Table 1: A Huffman code for the source $\{a, g, c, t\}$.

Its expected codeword length is

$$\bar{\ell} = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1.75 \text{ bits}.$$

5 Can block coding do better?

Even though Huffman coding is optimal among prefix-free codes for a given alphabet, we can still ask whether there is gain from coding *blocks* of symbols rather than one symbol at a time.

5.1 Symbol-by-symbol coding

In symbol-by-symbol coding, each source symbol is encoded separately. For example, the sequence

$$acaagt$$

is treated as

$$a,\ c,\ a,\ a,\ g,\ t.$$

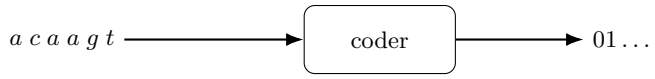

Figure 3: Symbol-by-symbol coding treats each source symbol separately.

5.2 Block coding

In block coding, several consecutive symbols are grouped together and treated as a *supersymbol*.

For example, grouping the same sequence into pairs gives

$$(ac)\ (aa)\ (gt).$$

The new two-symbol alphabet is

$$\mathcal{A}^{(2)} = \{aa, ag, ac, at, ga, gg, gc, gt, \ldots\}.$$

If we know the probabilities

$$p_{aa},\ p_{ag},\ p_{ac},\ p_{at}, \ldots,$$

then we can construct a Huffman code for this new alphabet.

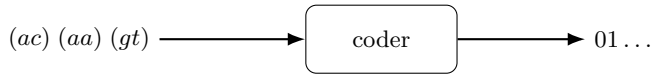

Figure 4: Block coding groups several source symbols into a supersymbol and then codes the supersymbols.

So block coding does not replace Huffman coding. Rather, it applies Huffman coding to a *larger alphabet of supersymbols*.

6 Two-symbol blocks

Let H_2 denote the entropy of the two-symbol supersource:

$$H_2 = \sum_{i,j} p_{ij} \log_2 \frac{1}{p_{ij}}.$$

If we apply prefix-free coding to this supersource, then the source coding theorem gives

$$H_2 \le \bar{\ell}_2^* \le H_2 + 1,$$

where $\bar{\ell}_2^*$ is the minimum expected codeword length *per two-symbol block*.

Dividing by 2, we obtain the average number of bits per original symbol:

$$\frac{H_2}{2} \le \frac{\bar{\ell}_2^*}{2} \le \frac{H_2}{2} + \frac{1}{2}.$$

Thus the overhead above the entropy is now only $\frac{1}{2}$ bit per original symbol, rather than 1 bit.

Similarly, for three-symbol blocks,

$$H_3 \le \bar{\ell}_3^* \le H_3 + 1$$

per block, so

$$\frac{H_3}{3} \le \frac{\bar{\ell}_3^*}{3} \le \frac{H_3}{3} + \frac{1}{3}$$

per original symbol.

In general, for block length n,

$$H_n \leq \bar{\ell}_n^* \leq H_n + 1$$

per block, hence

$$\frac{H_n}{n} \leq \frac{\bar{\ell}_n^*}{n} \leq \frac{H_n}{n} + \frac{1}{n}$$

per original symbol.

So the coding overhead per symbol decreases as n increases.

7 Example: English

To illustrate this idea, consider English text. Let the alphabet be

$$\mathcal{A} = \{a, b, c, \ldots, z, \text{space}\},$$

of size 27.

A natural question is:

What is the entropy of the English language?

If one looks only at single symbol, one gets a relatively large entropy. If one looks at pairs, triples, and longer blocks, the effective uncertainty per symbol decreases because letters are highly dependent.

$$H_0 \approx 4.75 \text{ bits/symbol},$$
$$H_1 \approx 4.03 \text{ bits/symbol},$$
$$H_2 \approx 6.64 \text{ bits/2 symbols}, \bar{H}_2 = \frac{H_2}{2} = 3.32 \text{ bits/symbol},$$
$$\bar{H}_3 \approx 3.10 \text{ bits/symbol},$$
$$\bar{H}_4 \approx 2.80 \text{ bits/symbol}.$$

H_0 is the entropy of each symbol assuming that all letters in the alphabet are equallu likely; $H_0 = \log_2 27 \approx 4.75$; one can regard this as the 0th order approximation of the entropy of English. H_1 is the entropy of each individual symbol, taking into consideration that the letters are not equally likely ("e" is much more likely than "z".) H_1 can be viewed as the first-order approximation of the entropy of English. H_2 is the entropy of blocks of two symbols, also called 2-grams. $\bar{H}_2$ is obtained by normalizing H_2 to the entropy per symbol.

For much longer blocks, the effective entropy per symbol appears to keep decreasing. Shannon estimated that around block length 100,

$$\bar{H}_{100} \sim 0.6 \text{ to } 1.3 \text{ bits/symbol}.$$

8 How did Shannon estimate these higher-order entropies?

The key idea is *prediction from context*.

For 2-grams, the probabilities can be decomposed as:

$$p_{ij} = p_i p_{j|i}.$$

Here:

- p_i is the probability that an individual symbol takes on letter i ,
- p_{ij} is the probability that the 2-gram (or 2-supersymbol) takes on letter ij;
- $p_{j|i}$is the *conditional probability* that the second symbol is j given the first symbol is i.

Lets take a pause here. Quantities p_i and p_{ij} are concepts defined previously; they are all probabilities of certain events. Moreover, they are related as follows:

$$p_i = \sum_j p_{ij}.$$

For example

$$p_a = p_{aa} + p_{ag} + p_{ac} + p_{at} :$$

in the event that the first symbol is an "a", the second symbol can be any letter, so we can calculate the probability that the first symbol is an "a" by summing up the probabilities of "aa", "ag", "ac", "at".

The concept of *conditional* probability, however, is a new one; in fact, it is *defined* to be:

$$p_{j|i} := \frac{p_{ij}}{p_i};$$

so the decomposition above is simply a restatement of this definition rather than from some deep mathematical theorem. However. the decomposition gives a natural interpretation of conditional probability; it is a measure of how likely it is to see that the next symbol is j when the first symbol is observed to be i.

Similarly, for 3-grams, one can decompose the probabilities as:

$$p_{ijk} = p_i p_{j|i} p_{k|ij}$$

is the probability of seeing letter k next after seeing ij.

Shannon estimated these conditional probabilities via a *guessing game*. He asked humans to predict the next symbol after observing the previous symbols. Shannon can be viewed not only as the founding father of information theory but also of modern day prediction markets! Indeed, the best way to estimate the conditional probabilities of an event is often to look at the payoffs in prediction markets like Polymarket.

In AI jargon, a window of observation like ij used for predicting the next token is known as the *context*. While humans can only store context of size say 100 (thus Shannon got estimates of H_{100}), modern day large language models (LLMs) like Claude or Gemini or ChatGPT can have contexts of size up to 1 million. Indeed, experiments using LLMs to predict next symbol in English text yields estimates of very high order entropy close to 1 bits/symbol.

9 Entropy rate

We already know that H_1 must be less than H_0, because the uniform distribution maximizes the entropy. This is a mathematical fact, not dependent on the values of the probabilities from which the entropies are computed (of course, the actual values of the entropies depend on the probabilities.) Similarly, it can also be shown mathematically that it must be true that $\bar{H}_n \geq \bar{H}_{n+1}$ for all n. This is consistent with the estimates for English text.

For each block length n, let H_n be the entropy of the n-grams. Then

$$\bar{H}_n := \frac{H_n}{n}$$

is the entropy per original symbol for block length n.

We have the following theorem.

Theorem

The sequence

$$\{\bar{H}_n\}$$

is monotonically decreasing and converges to a limit:

$$\bar{H}_n \downarrow \bar{H}_\infty.$$

The limit H_∞ is called the **entropy rate**.

Thus the entropy rate is the ultimate average uncertainty per source symbol when arbitrarily long context is taken into account.

10 Why block coding helps

For block length n, prefix-free coding of n-grams yields

$$H_n \leq \bar{\ell}^{(n)}_{\min} \leq H_n + 1$$

per block, hence

$$\frac{H_n}{n} \leq \frac{\bar{\ell}^*_n}{n} \leq \frac{H_n}{n} + \frac{1}{n}$$

in bits per original symbol.

As $n \to \infty$,

$$\frac{H_n}{n} \to H_\infty \qquad \text{and} \qquad \frac{1}{n} \to 0.$$

So by applying Huffman coding to sufficiently large blocks, the average number of bits per original symbol can be made arbitrarily close to the entropy rate.

So the main gain from block coding is : 1) getting to the entropy rate rather than the individual symbol entropy H_1, which can be larger; 2) amortizing the 1-bit gap over many symbols.

11 Summary

1. Huffman coding constructs an optimal prefix-free code recursively by repeatedly merging the two least likely symbols.
2. For a source alphabet $\mathcal{X}$, Huffman coding minimizes the expected codeword length $\bar{\ell}$ over all prefix-free codes.
3. In block coding, n consecutive source symbols are grouped into a supersymbol and coded together.
4. Huffman coding can then be applied to the supersource alphabet of blocks.
5. If H_n is the entropy of the n-symbol supersource, then

   $$H_n \leq \bar{\ell}^*_n \leq H_n + 1$$

   per block, or

   $$\frac{H_n}{n} \leq \frac{\bar{\ell}^*_n}{n} \leq \frac{H_n}{n} + \frac{1}{n}$$

   per original symbol.

6. For English, the effective entropy per symbol decreases as the block size increases.

7. Higher-order block probabilities are described using conditional probabilities such as

$$p_{ij} = P(j \mid i)P(i), \qquad p_{ijk} = P(k \mid ij)P(j \mid i)P(i).$$

8. The sequence H_n/n decreases to the entropy rate H_∞.

Takeaway

Huffman coding gives the optimal prefix-free code for a given alphabet. By applying Huffman coding to larger and larger blocks of source symbols, one exploits longer-range statistical structure in the source, and the achievable number of bits per symbol approaches the entropy rate.