Proving that Binary Huffman Codes are Optimal

We can prove that the binary Huffman code procedure produces optimal codes by induction on the number of symbols, q.

For $q = 2$, the code produced is obviously optimal — you can't do better than using one bit to code each symbol.

For $q > 2$, we assume that the procedure produces optimal codes for any alphabet of size $q - 1$ (with any symbol probabilities), and then prove that it does so for alphabets of size q as well.

The Induction Step

Suppose the Huffman procedure produces optimal codes for alphabets of size $q - 1$.

Let L be the average length of the code produced by the procedure when it is applied to the source $\mathcal{S}$, with q symbols, $s_1, \ldots, s_q$, having probabilities $p_1, \ldots, p_q$. Without loss of generality, let's assume that $p_i \geq p_{q-1} \geq p_q$ for all $i \in \{1, \ldots, q-2\}$.

The recursive call in the procedure will have produced a code for the source $\mathcal{S}'$, with $q - 1$ symbols, $s_1, \ldots, s_{q-2}, s'$, having probabilities $p_1, \ldots, p_{q-2}, p'$, with $p' = p_{q-1} + p_q$. By the induction hypothesis, this code is optimal. Let its average length be L'.

The Induction Step (Continued)

Suppose some other instantaneous code for $\mathcal{S}$ had average length less than L. We can modify this code so that the codewords for s_{q-1} and s_q are "siblings" — ie, they have the form $x0$ and $x1$, for some $x \in T^+$, while keeping its average length the same, or smaller.

Let the average length of this modified code be $\widehat{L}$, which must also be less than L.

From this modified code, we can produce a code for $\mathcal{S}'$. We keep the codewords for $s_1, \ldots, s_{q-2}$ the same, and encode s' as x. Let the average length of this code be $\widehat{L}'$.

The Induction Step (Conclusion)

We now have two codes for $\mathcal{S}$ and two for $\mathcal{S}'$. The average lengths of these codes satisfy the following equations:

$$\begin{aligned} L &= L' + p_{q-1} + p_q \\ \widehat{L} &= \widehat{L}' + p_{q-1} + p_q \end{aligned}$$

Why? The codes for $\mathcal{S}$ are like the codes for $\mathcal{S}'$, except that one symbol is replaced by two, whose codewords are one bit longer. So the average length of the code for $\mathcal{S}$ is the same as that for $\mathcal{S}'$, except for the increase due to this one bit, which is added with probability $p' = p_{q-1} + p_q$.

Since L' is the optimal average length, $L' \leq \widehat{L}'$. From these equations, we then see that $L \leq \widehat{L}$, which contradicts the supposition that $\widehat{L} < L$.

The Huffman procedure therefore produces optimal codes for alphabets of size q. By induction, this is true for all q.

What Have We Accomplished?

We seem to have solved the main problem: We now know how to construct an optimal code for any source.

But: This code is optimal **only** if the assumptions we made in formalizing the problem match the real situation.

Often they don't:

- Symbol probabilities may vary over time.
- Symbols may not be independent.
- There is usually no reason to require that X_1, X_2, X_3, ... be encoded one symbol at a time, as $\mathcal{C}(X_1)\mathcal{C}(X_2)\mathcal{C}(X_3)\cdots$.

We would require the last if we really needed instantaneous decoding, but usually we don't.

Example: Black-and-White Images

Recall the example from the first lecture, of black-and-white images. There are only two symbols — "white" and "black". The Huffman code is white $\mapsto$ 0, black $\mapsto$ 1.

This is just the obvious code. But we saw that various schemes such as run length coding can do better than this.

Partly, this is because the pixels are not independent. Even if they were independent, however, we would expect to be able to compress the image if black pixels are much less common than white pixels.

Solution: Coding Blocks of Symbols

We can do better by using Huffman codes to encode *blocks* of symbols.

Suppose our source probabilities are 0.7 for white and 0.3 for black. Assuming pixels are independent, the probabilities for blocks of two pixels will be

white white	$0.7 \times 0.7 = 0.49$
white black	$0.7 \times 0.3 = 0.21$
black white	$0.3 \times 0.7 = 0.21$
black black	$0.3 \times 0.3 = 0.09$

Here's a Huffman code for these blocks:

$WW \mapsto 0,\ WB \mapsto 10,\ BW \mapsto 110,\ BB \mapsto 111$

The average length for this code is 1.81, which is less than the two bits needed to encode a block the obvious way.

How Well Can We Do Using Blocks?

If we use blocks of more than two pixels, can we do even better?

If so, what is the limit to how well we can do, using bigger and bigger blocks?

These questions will lead us to the concept of *entropy*, which measures how well we can possibly compress data.