Encoding and decoding "on the fly"

The encoding method I described thus far assumes that the encoder computes the l_k and u_k up to $k = n$, then computes the tag in binary, then truncate to a certian number of bits, and then sends these bits.

Is this the only possible approach, or can the encoder begin encoding for some $k < n$? At first glance, it might seem impossible to encode "on the fly", since the codeword $C(\vec{x})$ is defined by the tag which itself is defined by l_n and u_n.

However, notice that if l_k and u_k agree up to some number of bits, e.g.

$$l_k = .\underline{0010111001}0000111010101110100010101 0$$

$$u_k = .\underline{0010111001}0101011010101011010010101 0$$

then all subsequent l_k and u_k will agree on this bits as well. The reason is that l_k is increasing and u_k is decreasing, and $l_k < u_k$ for all k. In particular, note that

$$l_k > .\underline{0010111001}0000000000000000000000000$$

$$u_k < .\underline{0010111001}1111111111111111111111111$$

Since we are sure that these most significant agreed-upon bits will be part of the tag, these bits can be sent. (Note: to be sure that the encoder is not sending too many bits, then encoder could compute $\lceil \log \frac{2}{p(i_1,\ldots,i_k)} \rceil$ and only send this many bits.)

If l_k and u_k agree on the first λ bits, then

$$\sum_{i=1}^{\lambda} b_i 2^{-i} \;\le\; l_k \;<\; T(\vec{x}) \;<\; u_k \;\le\; \sum_{i=1}^{\lambda} b_i 2^{-i} + 2^{-\lambda}$$

The expression on the left is the sum of all (infinitely many) of the remaining bits of $T(\vec{x})$ were all zeros, and the sum on the right is if the remaining bits were all 1's.

The next question to ask is whether the decoder can use these λ bits to recover some of the original sequence.

Suppose the encoder sends the first λ bits of $C(\vec{x})$ where $\lambda < \lambda(\vec{x})$. Since the remaining bits of the tag could be anything, all the decoder can be sure of is that:

$$\sum_{i=1}^{\lambda} b_i 2^{-i} \;\le\; T(\vec{x}) \;\le\; \sum_{i=1}^{\lambda} b_i 2^{-i} + 2^{-\lambda} \tag{1}$$

Can the decoder use this information ? Recall how the decoder figured out $X_1, \ldots, X_n$, in the case discussed in previous lectures in which the decoder is given $C(\vec{x})$, namely it chooses a sequence l_k and u_k such that

$$l_k \;\le\; \sum_{i=1}^{\lambda(\vec{x})} b_i 2^{-i} \;\le\; u_k$$

(I have been assuming that the decoder has first been told what n is, although this could be avoided by including an EOF symbol in the alphabet.)

If the decoder is given the first λ bits only, then it only knows inequality (1) is true. What can the decoder do? *If* the decoder can choose a sequence $X_1, \ldots, X_j$ for some j, such that

$$l_j \;\leq\; \sum_{i=1}^{\lambda} b_i 2^{-i} \;\leq\; \sum_{i=1}^{\lambda} b_i 2^{-i} + 2^{-\lambda} \leq u_j$$

then the decoder is guarenteed that this sequence up to j symbols is correct. The reason is that, for any fixed j, any other sequence of the first j symbols would give an $[l_j, u_j)$ interval that does not overlap the interval between the two sums, and hence such an interval could not contain the tag.

Finite precision

In this lecture and the previous one, I made a number of comments about finite precision. That is, to implement arithmetic coding, the encoder and decoder need to represent the l_k and u_k sequences, and the conditional probabilities needed for the update, and in practice, they are limited to a certain number of bits each. *I will not include this discussion here, and hence you are not responsible for the details.*