Unidade Curricular

“Informação e Codificação”

António José Ribeiro Neves

an@ua.pt

https://www.ua.pt/pt/uc/15264

Outline

Some guiding questions

- How Does Arithmetic Encoding Translate a Sequence of Symbols into a Single Number?
    - What are the steps involved in encoding a sequence of symbols using arithmetic encoding?
    - How does it divide a number range based on symbol probabilities to represent the entire sequence?
- How Can We Decode a Single Number Back Into the Original Sequence of Symbols in Arithmetic Encoding?
- How Does Arithmetic Encoding Depend on Accurate Probability Models for Efficient Compression?

Arithmetic encoding

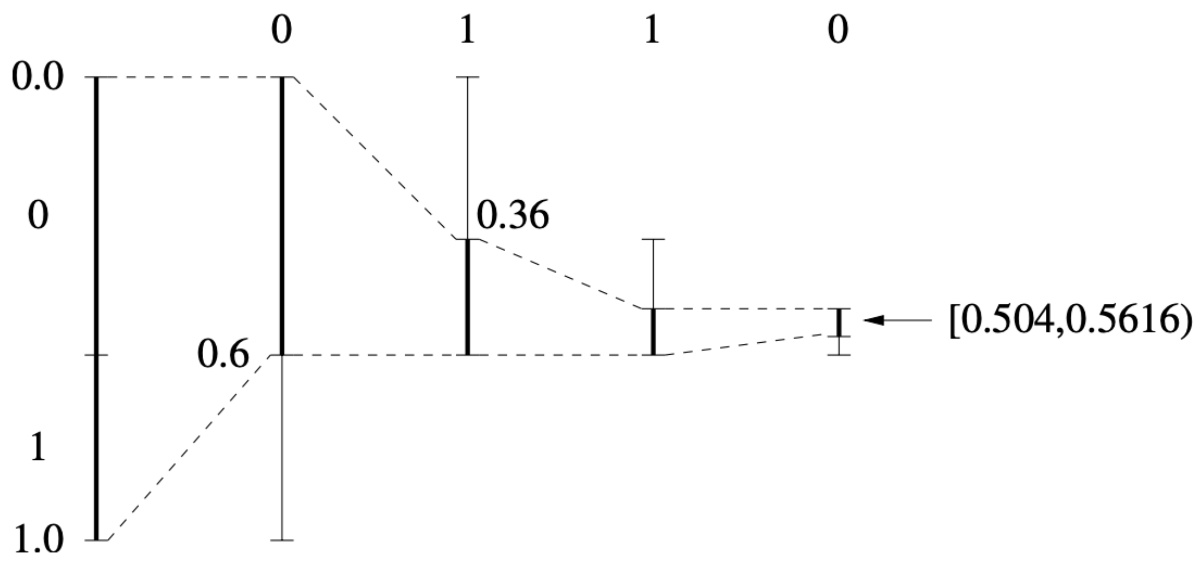
- In arithmetic coding, a unique identifier or tag is generated for the sequence to be encoded.
- This tag corresponds to a binary fraction, which becomes the binary code for the sequence.
- A unique arithmetic code can be generated for a sequence of length m without the need for generating codewords for all sequences of length m.
- In order to distinguish a sequence of symbols from another sequence of symbols we need to tag it with a unique identifier.
- One possible set of tags for representing sequences of symbols, are the numbers in the unit interval [0, 1).
- Because the number of numbers in the unit interval is infinite, it should be possible to assign a unique tag to each distinct sequence of symbols.

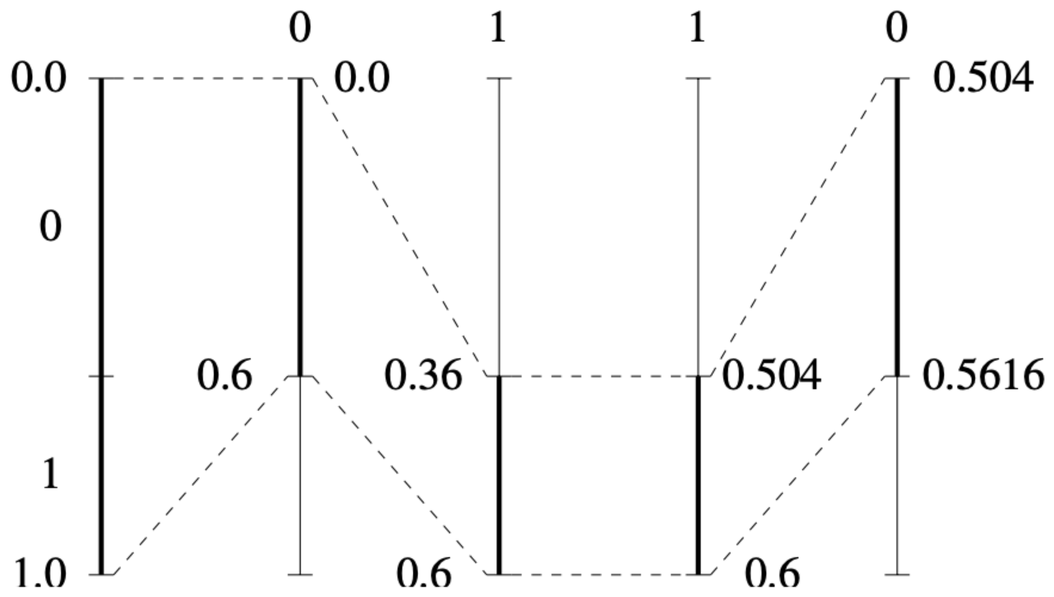
Arithmetic encoding

- In order to do this, we need a function that will map sequences of symbols into the unit interval.
- A function that maps random variables, and sequences of random variables, into the unit interval is the cumulative distribution function (cdf) of the random variable associated with the source.
- This is the function we will use in developing the arithmetic code.
- The tag of a sequence is any number in the interval bounded by the cdf of the sequence and the cdf of the sequence preceding it in some agreed upon ordering.
- The procedure for generating the tag works by reducing the size of the interval in which the tag resides as more and more elements of the sequence are received.

Example 5.17

Consider the encoding of the binary string `0110`*, using $P(0) = 0.6$ and $P(1) = 0.4$. The encoded message is a number in the interval $[0.504, 0.5616)$ (see Fig. 5.14). Another possible view of the encoding process can be seen in Fig. 5.15, in this case with renormalized intervals.*





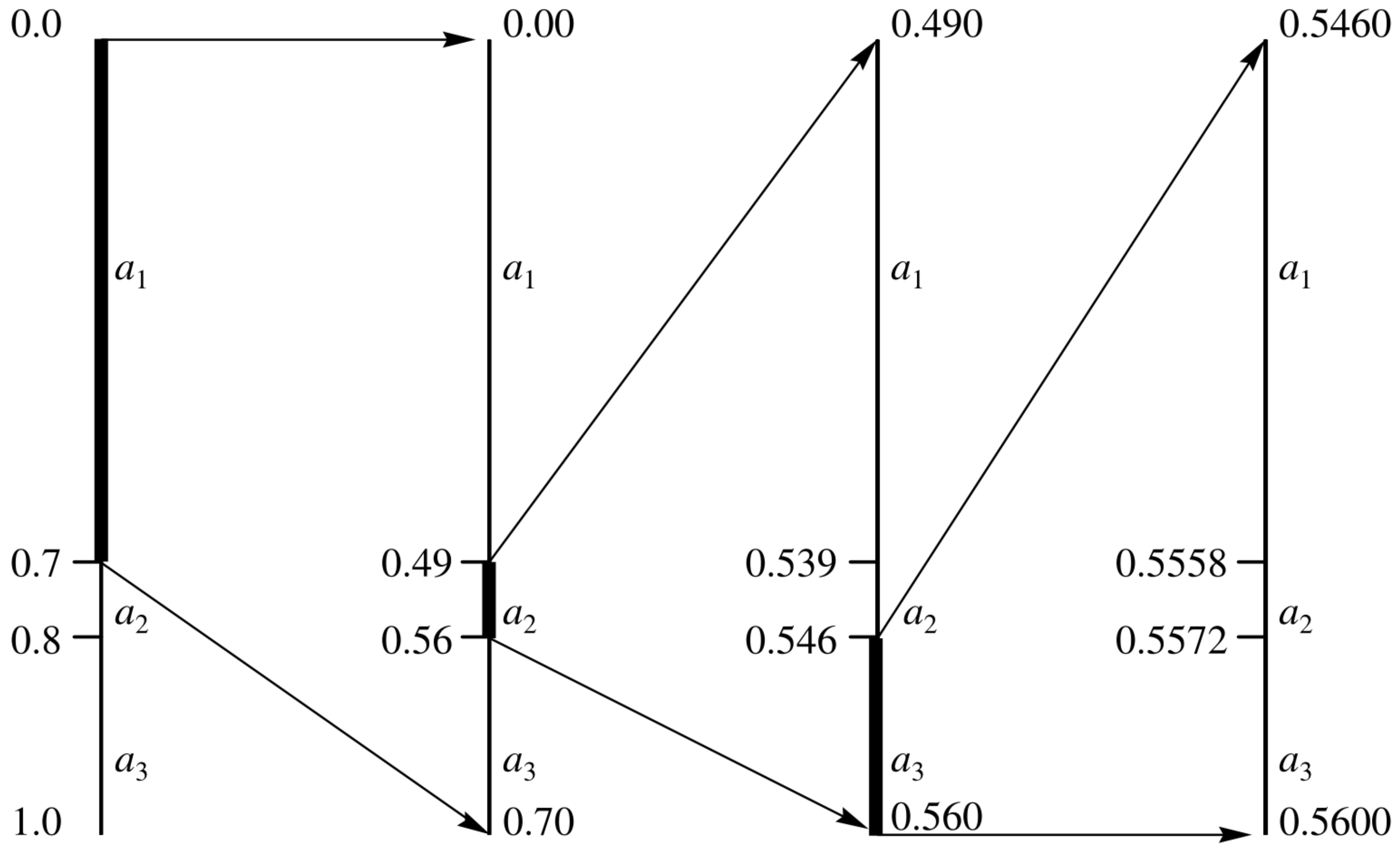
Therefore, if the current coding interval is $[L^n, H^n)$, and if the next symbol to encode is represented by the interval $[l, h)$, then

$$L^{n+1} = L^n + l(H^n - L^n)$$

and

$$H^{n+1} = L^n + h(H^n - L^n).$$

Example 4.3.1. Consider a three-letter alphabet $\mathcal{A} = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, and $P(a_3) = 0.2$. Using the mapping of Eq. (4.1), $F_X(1) = 0.7$, $F_X(2) = 0.8$, and $F_X(3) = 1$. This partitions the unit interval as shown in Fig. 4.1.



Challenge

Therefore, if the current coding interval is $[L^n, H^n)$, and if the next symbol to encode is represented by the interval $[l, h)$, then

$$L^{n+1} = L^n + l(H^n - L^n)$$

and

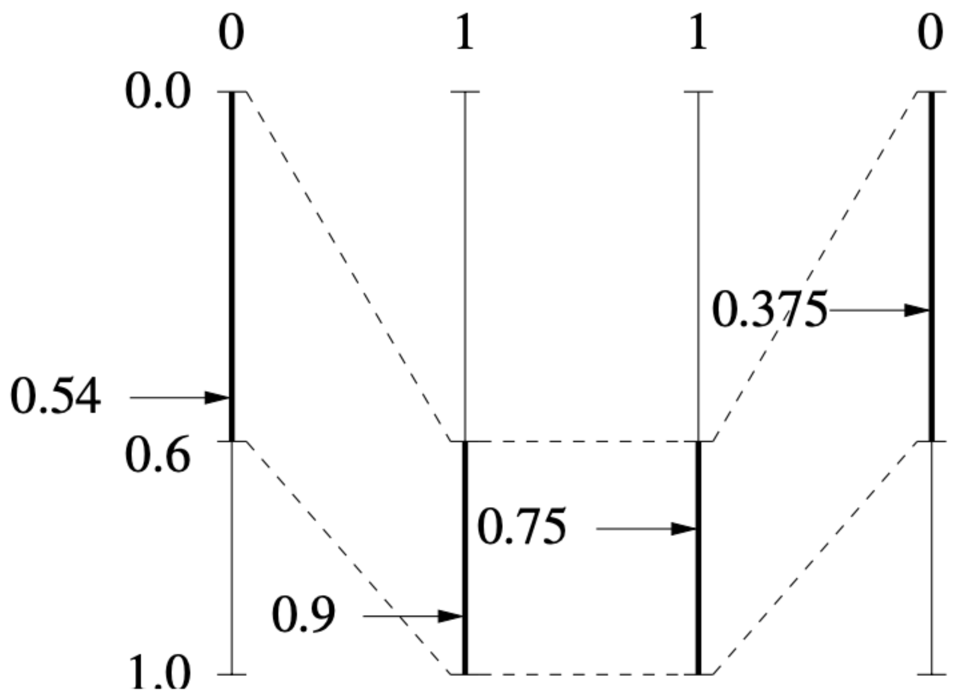
$$H^{n+1} = L^n + h(H^n - L^n).$$

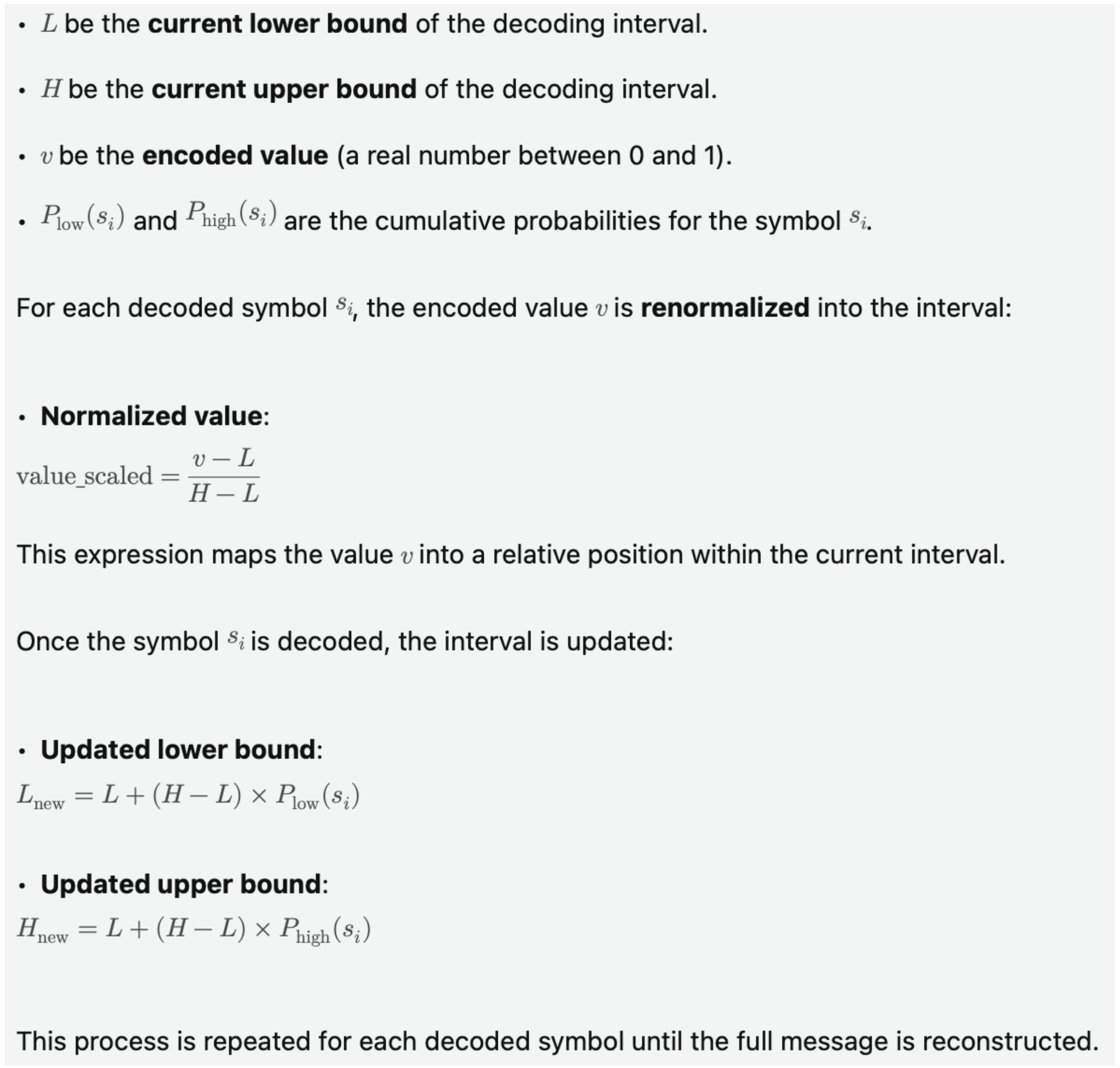
Consider the following alphabet and corresponding probability distribution:

Symbol	*Prob.*	*Interval*
d	0.1	$[0, 0.1)$
e	0.3	$[0.1, 0.4)$
f	0.1	$[0.4, 0.5)$
g	0.05	$[0.5, 0.55)$
n	0.1	$[0.55, 0.65)$
r	0.2	$[0.65, 0.85)$
u	0.05	$[0.85, 0.9)$
space	0.1	$[0.9, 1)$

Let us consider the encoding of the message “fred”. What is the interval representing this message?

Decoding process





- L be the **current lower bound** of the decoding interval.
- H be the **current upper bound** of the decoding interval.
- v be the **encoded value** (a real number between 0 and 1).
- $P_{\text{low}}(s_i)$ and $P_{\text{high}}(s_i)$ are the cumulative probabilities for the symbol s_i.

For each decoded symbol s_i, the encoded value v is **renormalized** into the interval:

- **Normalized value**:

$$\text{value\_scaled} = \frac{v - L}{H - L}$$

This expression maps the value v into a relative position within the current interval.

Once the symbol s_i is decoded, the interval is updated:

- **Updated lower bound**:

$$L_{\text{new}} = L + (H - L) \times P_{\text{low}}(s_i)$$

- **Updated upper bound**:

$$H_{\text{new}} = L + (H - L) \times P_{\text{high}}(s_i)$$

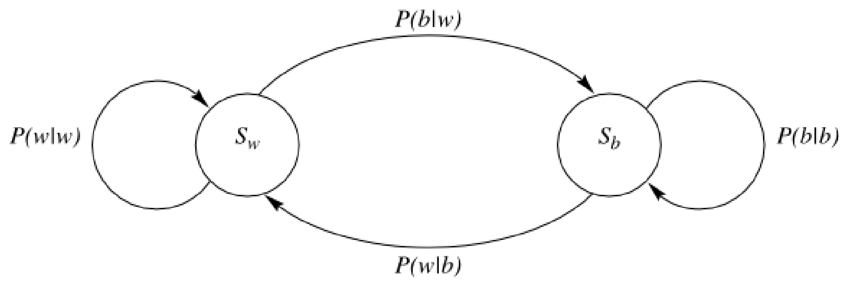
This process is repeated for each decoded symbol until the full message is reconstructed.

Practical issues

There are two ways to know when the entire sequence has been decoded. The decoder may know the length of the sequence, in which case the deciphering process is stopped when that many symbols have been obtained. The second way to know if the entire sequence has been decoded is that a particular symbol is denoted as an end-of-transmission symbol. The decoding of this symbol would bring the decoding process to a close.

In a real-world implementation, how does the finite precision of computer systems affect the arithmetic encoding process, especially when working with very small probability intervals? What strategies can be used to handle this issue?

MARKOV MODELS



For example, consider a binary image. The image has only two types of pixels, white pixels and black pixels.

We know that the appearance of a white pixel as the next observation depends, to some extent, on whether the current pixel is white or black.

Therefore, we can model the pixel process as a discrete time Markov chain.

The entropy of a finite state process with states S_i is simply the average value of the entropy at each state

$$H(S_w) = -P(b|w)\log P(b|w) - P(w|w)\log P(w|w)$$

$$H = \sum_{i=1}^{M} P(S_i)H(S_i).$$

Online learning

Often, the finite-context models are “learned” online, i.e., as the sequence of symbols is processed

In the case of online operation the probabilistic models are continuously adapting

A table (or some other more sophisti-cated data structure, such as an hash-table) is used to collect counts that represent the number of times that each symbol occurs in each context.

x_{i-3}	x_{i-2}	x_{i-1}	n_0	n_1
0	0	0	10	25
0	0	1	4	12
0	1	0	15	2
0	1	1	3	4
1	0	0	34	78
1	0	1	21	5
1	1	0	17	9
1	1	1	0	22

Binary source modeled by an order-3 finite-context model