# Compression: Arithmetic Coding

Thursday, November 20, 2025 12:19 AM

Recall:

- for any uniquely decodable code $C$, $\ell(C) \geq H(S)$
- there is a prefix code $C$ with $\ell(C) \leq H(S)+1$

When $S$ = English text, $H(S) \approx 4.22$. So $\ell(C) \leq 5.22$, $\approx 20\%$ worse.

The $+1$ is due to assigning lengths $\lceil \log \frac{1}{p(s)} \rceil$. How to avoid them?

Idea: encode multiple messages at a time.

Suppose encode 3 characters at a time: "and" etc.

Then more possible messages, but $p(s)$ much smaller per message, so $\log \frac{1}{p(s)}$ much larger, and ceiling has less effect.

<u>Arithmetic Coding</u>: way to encode entire sequence of messages

Idea: assign each sequence $m_1 m_2 m_3 \cdots m_n$ an interval in $[0,1)$ of length $p(m_1) \cdot p(m_2) \cdot \cdots \cdot p(m_n)$. Each of the $|S|^n$ possible messages gets a disjoint interval.

Example: $S = \{\text{"0"}, \text{"1"}\}$, each probability $1/2$.

00...00 gets interval $[0, \frac{1}{2^n})$

00...01 " " $[\frac{1}{2^n}, \frac{2}{2^n})$

00...10 " " $[\frac{2}{2^n}, \frac{3}{2^n})$

$\vdots$

11...11 " " $[\frac{2^n-1}{2^n}, \frac{2^n}{2^n})$

Example: $S = \{\text{"a"}, \text{"b"}, \text{"c"}\}$, probabilities 0.2, 0.5, 0.3

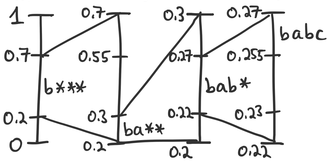
How to encode "babc"?

Start with dividing $[0,1)$ into $\underbrace{[0, 0.2)}_{\text{all a***}}$, $\underbrace{[0.2, 0.7)}_{\text{all b***}}$, $\underbrace{[0.7, 1)}_{\text{all c***}}$

Zoom in on b***. Again, divide into 0.2 : 0.5 : 0.3 sub-intervals:

$\underbrace{[0.2, 0.3)}_{\text{all ba**}}$, $\underbrace{[0.3, 0.55)}_{\text{all bb**}}$, $\underbrace{[0.55, 0.7)}_{\text{all bc**}}$

Zoom in on ba**, etc. Overall, looks like




Given probabilities $p(1), \ldots, p(m)$, define accumulated probability $f(j) = p(1) + \cdots + p(j-1)$.

In example above, $f(1) = 0$, $f(2) = 0.2$, $f(3) = 0.7$.

<u>Algorithm</u>

- Initialize $\ell_0 = 0$, $s_0 = 1$ // interval is $[\ell_i, \ell_i + s_i)$
- For $i = 1, \ldots, n$:
  - $\ell_i = \ell_{i-1} + f(m_i) \cdot s_{i-1}$
  - $s_i = s_{i-1} \cdot p(m_i)$
- Final interval is $[\ell_n, \ell_n + s_n)$

How to encode the message?

Attempt 1: Consider any binary number inside target interval $[\ell_n, \ell_n + s_n)$ send binary representation $(0.\underbrace{0110101})$ as codeword.

Issue: not prefix-free. Maybe $\underbrace{0.01}$ is in a different interval.

Solution: each codeword $0.\underbrace{0110}$ now represents the interval $[0.0110\overline{0}, 0.0110\overline{1})$. Assign a codeword whose interval is contained inside target interval $[\ell_n, \ell_n + s_n)$

This is prefix free because it's not possible for prefix $0.\underbrace{011}$ with interval $[0.011\overline{0}, 0.011\overline{1})$ to be in a different target interval $[\ell_n', \ell_n' + s_n')$, since

$$[0.0110\overline{0}, 0.0110\overline{1}) \subseteq [\ell_n, \ell_n + s_n)$$

$$[0.0110\overline{0}, 0.0110\overline{1}) \subseteq [0.011\overline{0}, 0.011\overline{1}) \subseteq [\ell_n', \ell_n' + s_n')$$

implies the target intervals $[\ell_n, \ell_n + s_n)$ and $[\ell_n', \ell_n' + s_n')$ overlap, which is not allowed.

How long is the codeword?

Lemma: the interval represented by $\ell_n + s_n/2$ truncated to $\lceil -\log_2 s_n \rceil + 1$ bits is contained in $[\ell_n, \ell_n + s_n)$.

Proof: Truncating a number to $\lceil -\log_2 s_n \rceil + 1$ bits lowers the number by at most $1/2^{\lceil -\log_2 s_n \rceil + 1} \leq 1/2^{-\log_2 s_n + 1} = 2^{\log_2 s_n - 1} = s_n/2$.

So the interval starts at $\geq (\ell_n + s_n/2) - s_n/2 = \ell_n$.

A number with $\lceil -\log_2 s_n \rceil + 1$ bits represents an interval of length $1/2^{\lceil -\log_2 s_n \rceil + 1} \leq s_n/2$, so the interval ends at $\leq (\ell_n + s_n/2) + s_n/2$
$\leq \ell_n + s_n$.

Recall that $s_n = p(m_1) \cdot p(m_2) \cdot \cdots \cdot p(m_n)$. So the codeword has length

$$\lceil -\log_2 s_n \rceil + 1 = \lceil -\log_2 (p(m_1) \cdot \cdots \cdot p(m_n)) \rceil + 1$$

$$= \lceil -\log_2 p(m_1) + \cdots + -\log_2 p(m_n) \rceil + 1$$

$$= \underbrace{\lceil \log_2 \tfrac{1}{p(m_1)} + \cdots + \log_2 \tfrac{1}{p(m_n)} \rceil}_{\text{compare to } \lceil \log \frac{1}{p(m_1)} \rceil + \cdots + \lceil \log_2 \frac{1}{p(m_n)} \rceil \text{ previously}} + 1$$

So, it's within $+2$ of how many bits we "must" use to encode $m_1 \cdots m_n$.

On average, uses $\leq n \cdot \mathbb{E} \log_2 \frac{1}{p(s)} + 2 = n \cdot H(S) + 2$ bits to send $n$ messages, which is close to the $n \cdot H(S)$ lower bound.