4. Arithmetic coding / decoding

4.1. Segment coding
4.2. Binary representation of segments
4.3. Arithmetic coding
4.4. Arithmetic decoding
4.5. A note on the bit encoding of a segment
4.6. Arithmetic coding with conditional probabilities
4.7. Incremental arithmetic coding based on segment rescaling
4.1. Segment coding

Consider a source alphabet $\mathcal{A} = \{a_1, \ldots, a_n\}$, with probabilities $\{p_1, \ldots, p_n\}$. Let $M = a_{j_1} \cdots a_{j_N}$ be a source message of length $N$. In *segment coding* (the first step in what will be *arithmetic coding*) a segment (or interval)

$$S_M = [l_M, h_M) \subset [0,1)$$

is assigned to any such $M$. Moreover, this assignment has the following properties:

1. $h_M - l_M = P(M)$
2. $S_M \cap S_{M'} = \emptyset$ for any pair two distinct messages $M, M'$ of length $N$.
3. $\bigcup M S_M = [0,1)$, the union is with respect to all messages of length $N$.
4. If $M$ is a prefix of $M'$, then $S_{M'} \subset S_M$.

So the length of $S_M$ is $P(M)$; the different $S_M$, for fixed $N$, form a *partition* of $[0,1)$, and the partition for $N' > N$ is a refinement of that for $N$. 
The case $N = 1$

For messages of length 1 (any one of the symbols $a_j$ of $\mathcal{A}$), we set

$$S_{a_j} = [p_1 + \cdots + p_{j-1}, p_1 + \cdots + p_{j-1} + p_j) = [\sigma_{j-1}, \sigma_j),$$

where we define $\sigma_j = p_1 + \cdots + p_{j-1} + p_j, \ j = 1, \ldots, n$ (we say that $\sigma_1, \ldots, \sigma_n$) is the cumulative probability distribution. Thus we have

$$l_{a_j} = \sigma_{j-1}, \ h_{a_j} = \sigma_j, \ h_{a_j} - l_{a_j} = p_j$$

From the definitions it follows that $0 < p_1 = \sigma_1 < \cdots < \sigma_n = 1$ and hence that the segments $S_{a_j}$ cover $[0,1)$ and are pairwise disjoint. See the left column of the illustration on next page for an example in which $n = 3$.

Example. Before considering the general construction, we first describe it in a simple example. We will see how to obtain the segment $S_M$ for the message $babc$ produced by the source $\{ 'a': 0.2, 'b': 0.5, 'c': 0.3 \}$. 
By our stipulation of the case $N = 1$, we know that

$$S_b = [0.2, 0.7).$$

Let us proceed now to assign an interval to $ba$ (second column of the illustration), then to $bab$ (third column) and finally to $babc$ (fourth column). The interval $S_{ba}$ is obtained by subdividing

$$S_b = [l_b = 0.2, h_b = 0.7)$$

into segments of relative length $p(a) = 0.2$, $p(b) = 0.5$ and $p(c) = 0.3$ and choosing the $a$-segment as $S_{ba}$. So the division points are

$$l_{ba} = 0.2, h_{ba} = l_{bb} = 0.2 + 0.5 \times \sigma(a) = 0.30,$$

$$h_{bb} = l_{bc} = 0.2 + 0.5 \times \sigma(b) = 0.55, h_{bc} = 0.2 + 0.5 \times \sigma(c) = 0.70,$$

so

$$S_{ba} = [0.2, 0.3), S_{bb} = [0.30, 0.55), S_{bc} = [0.55, 0.70).$$
Now the interval $S_{bab}$ is obtained in a similar way: subdivide $S_{ba}$ into intervals that are proportional to $p(a), p(b)$ and $p(c)$ and choose as $S_{bab}$ the segment corresponding to $b$. Actually we have

$$h_{baa} = l_{bab} = 0.2 + 0.1 \times \sigma(a) = 0.22,$$
$$h_{bab} = l_{bac} = 0.2 + 0.1 \times \sigma(b) = 0.27,$$
$$h_{bac} = 0.2 + 0.1 \times \sigma(c) = 0.30,$$

and hence

$$S_{bab} = [l_{bab}, h_{bab}) = [0.22, 0.27).$$

Finally we get, following the same procedure with $S_{bab}$,

$$S_{babc} = [0.22 + 0.05 \times \sigma(b), 0.22 + 0.05 \times \sigma(c))] = [0.255, 0.270).$$
The general case

Suppose that we already know the interval \( S_M = [l_M, h_M] \) of a message of length \( N \) and that \( h_M - l_M = P(M) \). Then the interval of \( M' = Ma_j \) is defined as follows:

\[
S_{M'} = [l_{M'}, h_{M'}) = [l_M + P(M) \times \sigma_{j-1}, l_M + P(M) \times \sigma_j].
\]

We note that

\[
h_{M'} - l_{M'} = P(M) \times (\sigma_j - \sigma_{j-1}) = P(M)p_j = P(M').
\]

Remark. The conditions 1-4 at the beginning are a direct consequence of the definitions.
4.2. Binary representation of segments

**Binary representation of numbers in the unit segment**

```
# Binary representation

<table>
<thead>
<tr>
<th>Number</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.270</td>
<td>1010001</td>
</tr>
<tr>
<td>0.255</td>
<td>1000101</td>
</tr>
<tr>
<td>0.25</td>
<td>0.01001</td>
</tr>
<tr>
<td>0.75</td>
<td>0.11011</td>
</tr>
<tr>
<td>0.76</td>
<td>0.11100</td>
</tr>
<tr>
<td>0.7601</td>
<td>0.111001</td>
</tr>
<tr>
<td>0.76011</td>
<td>0.111001</td>
</tr>
<tr>
<td>0.760111</td>
<td>0.111001</td>
</tr>
<tr>
<td>0.7601111</td>
<td>0.111001</td>
</tr>
</tbody>
</table>
```

The binary unit segment

**Examples.**

- 0.5 → 0.1, 0.25 → 0.01, 0.75 → 0.11, 0.125 → 0.001;
- 0.255 → 0.01000, 0.270 → 0.01001;
- 0.011001 → $1/4 + 1/8 + 1/64 = 25/64 = 0.390625$. 

0.255 → 0.01000, 0.270 → 0.01001;
4.3. Arithmetic coding

The basic idea of *arithmetic coding* is to select an element in the segment $S_M = [l_M, h_M)$ that requires the minimal number of bits. Then we encode $M$ using the binary word formed with those bits.

This can be accomplished as follows. Suppose that the first bit that is different in the binary representations of $l_M$ and $h_M$ is the $r$-th, so that we will have

$$l_M = 0.b_1b_2 \cdots b_{r-1}0 \cdots, \quad h_M = 0.b_1b_2 \cdots b_{r-1}1 \cdots$$

If $0.b_1b_2 \cdots b_{r-1}1 < h_M$, then the number $0.b_1b_2 \cdots b_{r-1}1$, or the word $b_1b_2 \cdots b_{r-1}1$, satisfies the requirements, for $0.b_1b_2 \cdots b_{r-1}1 \in S_M$ and any other number in the interval $S_M$ will require more bits. Thus we encode $M$ as $b_1b_2 \cdots b_{r-1}1$. If $0.b_1b_2 \cdots b_{r-1}1 = h_M$, then the shortest binary word will be $b_1b_2 \cdots b_{r-1}$ if $l_M = 0.b_1b_2 \cdots b_{r-1}0$. Otherwise $l_M = 0.b_1b_2 \cdots b_{r-1}0x$, with $x \neq 0$. If $x = 0 \cdots$, we can take $0.b_1b_2 \cdots b_{r-1}01$, else if $x = 1 \cdots 10$ ($s$ ones), we may take $0.b_1b_2 \cdots b_{r-1}01 \cdots 11$ ($s + 1$ ones).
Remark. For the decoding, it will be convenient to include in the encoding the length of the original message. Thus it can be represented as a pair \((L, c)\), where \(L\) is the length of the message and \(c\) is the binary string given by the arithmetic encoding.

For example, if \(S = [‘a’ : 0.2, ‘b’ : 0.5: ‘c’ : 0.3]\) and \(M= ”babc”\), then the arithmetic encoding is \([4,”010001”]\), for the binimals of 0.255 and 0.270 are, respectively:

\[
01000001010 \ldots
\]

\[
01000101000 \ldots
\]
4.4. Arithmetic decoding and effective computations

Suppose we have the arithmetically encoded message
\[ C = [N, x] \] (\( N \) the number of symbols, \( x \) the binary string encoding).

Let \( P \) be a representation of the source as a list of pairs (symbol, probability). Let \( A \) denote the list of pairs (symbol, cumulative-probability). The decoding algorithm is implemented in the function \( AD(C,P) \) which will be studied, together with the encoding function \( AE(M,P) \).

**Example.** If \( P=[('a', 0.25), ('b', 0.4), ('c', 0.15), ('d', 0.1), ('e', 0.1)] \) and \( M= "badbbdcbabea" \), then the arithmetic encoding \( AE(M,P) \) is
\[ C=[12,"01010101101101101101101101"
\] and \( AD(C,P) \) is “badbbdcbabea”.

**Remark.** This description of arithmetic encoding and decoding would work if floats had unlimited precision, but in pyzo it only works for short messages (of the order of 54 encoded bits, due to the fact of that floats have 16 significant digits, which amounts to \( 16 \times \log(10) / \log(2) \approx 53.15 \) bits.
4.5. A note on the bit encoding of a segment

The question is how to select a binary interval $\left[\frac{n}{2^k}, \frac{n+1}{2^k}\right)$ of highest length $1/2^k$ (least $k$) contained in a segment $[a, b) \subseteq [0,1)$.

In particular, $\frac{1}{2^k} \leq b - a$, hence $-k \leq \log_2(b - a)$, or $k \geq -\log_2(b - a)$. It follows that the least possible $k$ is $k = \lceil -\log_2(b - a) \rceil$.

Now we must have $a \leq n/2^k$, or $2^k a \leq n$, and the least possible integer $n$ is $n_k = \left\lfloor 2^k a \right\rfloor$.

With these choices, if the binary segment $I_k = \left[\frac{n_k}{2^k}, \frac{n_k+1}{2^k}\right)$ is contained in $[a, b)$, which is equivalent to say that $\frac{n_k+1}{2^k} \leq b$, then we are done: we can take the binary representation of $n_k$ as the bit encoding of $[a, b)$.

If $\frac{n_k+1}{2^k} > b$, then the interval $I_{k+1}$ is contained in $[a, b)$, for

$$n_{k+1} - 1 < 2^{k+1}a, \text{ or } \frac{n_{k+1}}{2^{k+1}} < a + \frac{1}{2^{k+1}}$$
and therefore
\[ \frac{n_{k+1} + 1}{2^{k+1}} < a + \frac{1}{2^{k+1}} + \frac{1}{2^{k+1}} = a + \frac{1}{2^k} \leq a + (b - a) = b. \]

So in this case we can take the binary representation of \( n_{k+1} \) as the bit encoding of \([a, b)\).

This can be easily programmed into an alternative bit encoder of intervals.

**Note.** If \( x \) is the binary string representing \( n / 2^k \), the function \( \text{segment}(x) \) in \texttt{cdi.py} returns the endpoints of \( I_k \).

From the analysis above, it is easy to show that arithmetic coding becomes practically optimal for long messages. Indeed, for messages \( M \) of length \( N \), the average length is 
\[ \bar{\ell}_N = \sum_M P(M) \ell(M), \]
where \( \ell(M) \) is the number of bits of the arithmetic encoding of \( M \). Since we just have seen that
\[ \ell(M) \leq \left\lceil \log_2 \frac{1}{P(M)} \right\rceil + 1 \leq \log_2 \frac{1}{P(M)} + 2 \]
we get
\[ H_N \leq \bar{\ell}_N \leq H_N + 2 \]
where $H_N$ stands for the entropy of messages of length $N$ (the first inequality by Shannon’s theorem). Now $H_N = NH$, where $H$ is the entropy of the symbol source (an easy calculation using that $P(a_{j1} \cdots a_{jN}) = \prod_k p_{j_k}$). Therefore

$$H \leq \bar{\ell} \leq H + \frac{2}{N},$$

where $\bar{\ell}$ is the average length per symbol, and this means that $\bar{\ell} \approx H$ for large $N$.

Note that if sending the encoding of messages $M$ of length $N$ includes a fixed overhead (like sending the number $N$ as part of the encoding), then the conclusion still holds (the 2 above is replaced by some constant $K_N$ such that $\frac{K_N}{N} \to 0$ as $N$ grows).
4.6. Arithmetic coding with conditional probabilities

When the probabilities are conditional (so they depend on the current symbol and of the preceding symbols in the message), the (high) cumulative probabilities for each symbol $a_j \in \mathcal{A}$ when we are scanning symbol $x_k$ in the message $x_1 \cdots x_k \cdots x_N$, are given by

$$\sigma(a_k | x_1, \ldots, x_{k-1}) = \sum_{i=1}^{k} P(a_i | x_1, \ldots, x_{k-1}).$$

For examples, see L1016-1.
4.7. Incremental arithmetic coding based on segment rescaling

If the current segment \([l, h)\) in AE is contained in \([0, 1/2)\), then we know that next bit is 0. So we can send 0 to the decoder and rescale \([l, h)\) to \([2l, 2h)\). This rescaling, which in binary is a shift to the left, forgets the information about that bit, but this is not a loss because it is already available to the decoder. Similarly, we can send a 1 to the decoder if \([l, h)\) is contained in \([1/2, 1)\) and rescale \([l, h)\) to \([2l - 1, 2h - 1)\).

The decoding has to perform the same rescaling each time a bit is received. The implementation of these rules is illustrated by the functions SAE and SAD in cdi_SACD.py.
4.8. Arithmetic coding with integers

We will work with blocks of $m$ bits (we may take $m = 16$, giving a total of $2^{16} = 65536$ numbers). This can be done as follows:

Let $N$ be the length of the message, $M$.
If $f[x]$ is the frequency of symbol $x$, let $F[x] = \sum_{y \leq x} f(y)$.

**Encoding algorithm** (of a message $M$):

$l = 0; h = N$

for $x$ in $M$:

\[ u = h - l + 1 \]

\[ h = l + \left\lfloor \frac{u \cdot F[x]}{N} \right\rfloor \]

\[ l = l + \left\lfloor \frac{u \cdot f[x]}{N} \right\rfloor - 1 \]

\[ u = h - l + 1 \]
\[
\begin{align*}
    h &= l + (u \cdot \sigma'[x]) / \sigma' - 1 \\
    l &= h - (u \cdot F'[x]) / \sigma'
\end{align*}
\]

**Decoding algorithm**

\[
\begin{align*}
    x &= \text{encoding value (string)} \\
    l &= 0; h = \sigma'
\end{align*}
\]

\[
\begin{align*}
    u &= h - l + 1 \\
    h &= l + (u \cdot \sigma'[x]) / \sigma' - 1 \\
    l &= h - (u \cdot F'[x]) / \sigma'
\end{align*}
\]