Arithmetic Coding

Weifeng Sun
Contents

- Huffman coding revisited
- History of arithmetic coding
- Ideal arithmetic coding
- Properties of arithmetic coding
- Binary arithmetic coding
Huffman Coding Revisited
-- How to Create Huffman Code

- Construct a Binary Tree of Sets of Source Symbols.
  - Sort the set of symbols with non-decreasing probabilities.
  - Form a set including two symbols of smallest probabilities.
  - Replace these by a single set containing both the symbols whose probability is the sum of the two component sets.
  - Repeat the above steps until the set contains all the symbols.
  - Construct a binary tree whose nodes represent the sets. The leaf nodes representing the source symbols.

<table>
<thead>
<tr>
<th>symbol</th>
<th>probability</th>
<th>code</th>
<th>binary fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>0.5</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>X</td>
<td>0.25</td>
<td>01</td>
<td>0.01</td>
</tr>
<tr>
<td>Y</td>
<td>0.125</td>
<td>001</td>
<td>0.001</td>
</tr>
<tr>
<td>Z</td>
<td>0.125</td>
<td>000</td>
<td>0.000</td>
</tr>
</tbody>
</table>
Traverse each path of the tree from root to a symbol, assigning a code 0 to a left branch and 1 to a right branch. The sequence of 0’s and 1’s thus generated is the code for the symbol.

<table>
<thead>
<tr>
<th>symbol</th>
<th>probability</th>
<th>code</th>
<th>binary fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>0.5</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>X</td>
<td>0.25</td>
<td>01</td>
<td>0.01</td>
</tr>
<tr>
<td>Y</td>
<td>0.125</td>
<td>001</td>
<td>0.001</td>
</tr>
<tr>
<td>Z</td>
<td>0.125</td>
<td>000</td>
<td>0.000</td>
</tr>
</tbody>
</table>
Prosperities of Huffman Coding

- Huffman codes are minimum redundancy codes for a given probability distribution of the message.
- Huffman coding guarantees a coding rate $l_H$ within one bit of the entropy $H$.
  - Average code length $l_H$ of the Huffman coder on the source $S$ is bounded by $H(S) \leq l_H \leq H(S) + 1$.
- Studies showed that a tighter bound on the Huffman coding exists.
  - Average code length $l_H < H(S) + p_{max} + 0.086$, where $p_{max}$ is the probability of the most frequently occurring symbol.
  - So, if the $p_{max}$ is quite big (in case that the alphabet is small and the probability of occurrence of the different symbols is skewed), Huffman coding will be quite inefficient.
Prosperities of Huffman Coding (continued)

- Huffman code does not achieve ‘minimum redundancy’ because it does not allow fractional bits.

- Huffman needs at least one bit per symbol.
  - For example, given alphabet containing two symbols with probability: $p_1 = 0.99, p_2 = 0.01$
  - The optimal length for the first symbol is: $-\log(0.99) = 0.0145$
  - The Huffman coding, however, will assign 1 bit to this symbol.

- If the alphabet is large and probabilities are not skewed, Huffman rate is pretty close to entropy.
Prosperities of Huffman Coding (continued)

- If we block m symbols together, the average code length $l_H$ of the Huffman coder on the source S is bounded by $H(S) \leq l_H \leq H(S) + \frac{1}{m}$

- However, the problem here is that we need a big codebook. If the size of the original alphabet is K, then the size of the new code book is $K^m$.

- Thus, Huffman’s performance becomes better at the expense of exponential codebook size.
Another View of Huffman Coding

- Huffman code re-interpreted here by mapping the symbols to subintervals of \([0,1)\) at the base value of the subintervals.

- The code words, if regarded as \textbf{binary fractions}, are pointers to the particular \textit{interval} in the binary code.

- An extension to this idea is to encode the symbol sequence as a subinterval leads to arithmetic coding.

<table>
<thead>
<tr>
<th>symbol</th>
<th>probability</th>
<th>code</th>
<th>binary fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>0.5</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>X</td>
<td>0.25</td>
<td>01</td>
<td>0.01</td>
</tr>
<tr>
<td>Y</td>
<td>0.125</td>
<td>001</td>
<td>0.001</td>
</tr>
<tr>
<td>Z</td>
<td>0.125</td>
<td>000</td>
<td>0.000</td>
</tr>
</tbody>
</table>
Here Comes Arithmetic Coding

- The idea is to code string as a binary fraction pointing to the subinterval for a particular symbol sequence.

- Arithmetic coding is especially suitable for small alphabet (binary sources) with highly skewed probabilities.

- Arithmetic coding is very popular in the image and video compression applications.
The idea that code string can be a binary fraction pointing to the subinterval for a particular symbol sequence is due to Shannon [1948]; and was used by Elias [1963] to successive subdivision of the intervals.

Shannon observed that if the probabilities were treated as high precision binary numbers, then it may be possible to decode messages unambiguously.

David Huffman invented his code around the same time and the observation was left unexplored until it resurfaced in 1975.
The idea of arithmetic coding was suggested by Rissanen [1975] from the theory of enumerative coding by Pasco [1976].

The material of this notes is based on the most popular implementation of arithmetic coding by Witten, etc., published in Communications of the Association for Computing Machinery (1987).

Moffat, etc (1998) also proposed some improvements upon the 1987 paper; however, the basic idea remains same.
The basic idea of the arithmetic coding is to use a high-precision fractional number to encode the probability of the message.

- Message $M = [a, b, a, a, a, e, a, a, b, a]$, alphabet $\sum = \{a, b, c, d, e, f\}$
- Probability distribution $P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]$.
- Probability of the message is
  \[ P[a] \cdot P[b] \cdot P[a] \cdot P[a] \cdot P[a] \cdot P[e] \cdot P[a] \cdot P[a] \cdot P[b] \cdot P[a] \]
  \[= 0.67 \cdot 0.11 \cdot 0.67 \cdot 0.67 \cdot 0.67 \cdot 0.05 \cdot 0.67 \cdot 0.67 \cdot 0.11 \cdot 0.67 \]
  \[= 0.00003666730521220415. \]

However, we cannot simply use this probability to encode the message, because we know there exist many messages which have exactly the same probability, such as $M_1 = [b, a, a, a, e, a, a, b, a]$, or $M_2 = [a, a, b, a, a, e, a, a, b, a]$, etc.

In fact, all combinations of the message $M$ have the same probability as the message $M$. So, to encode this message, we need to enforce the order of the letters occurred in $M$.

As we know that $P$ is the probability density function (pdf) of the random variables, maybe we can try the cumulative density function (cdf).
Cumulative Density Function (cdf)

For random variable X, the probability \( P(X \leq x) \) is the \textit{cumulative density function (cdf)}, denoted as \( F_X(x) \).

Here are some of the properties of the cdf:
- \( 0 \leq F_X(x) \leq 1 \). This follows from the definition of the cdf.
- The cdf is a monotonically nondecreasing function. That is, \( x_1 \geq x_2 \Rightarrow F_X(x_1) \geq F_X(x_2) \)
Example of Cumulative Density Function (cdf)

- Alphabet $\sum = \{a, b, c, d, e, f\}\$
- \(P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04] \)

$$F_x(x) = \begin{cases} 
0.00 & x \notin \sum \\
0.67 & x \in \{a\} \\
0.78 & x \in \{a, b\} \\
0.85 & x \in \{a, b, c\} \\
0.91 & x \in \{a, b, c, d\} \\
0.96 & x \in \{a, b, c, d, e\} \\
1.00 & x \in \{a, b, c, d, e, f\}
\end{cases}$$

For simplicity, we denote it as $PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]$
Use CDF to Encode the Message
--- Basic Idea

- Given the cdf, we can say that the probability of one symbol $x_i \in \Sigma$ is given by $F_x(x_i) - F_x(x_{i-1})$, which is exactly the probability density function.

- However, the probability of the symbol $x_i$ can also be interpreted by the interval $[F_x(x_{i-1}), F_x(x_i))$. This interval has the same length as the value $F_x(x_i) - F_x(x_{i-1})$.

- If we go ahead further, this interval can also be represented by a tuple $[F_x(x_{i-1}), F_x(x_i) - F_x(x_{i-1}))$.

- For simplicity, we use $[\text{LOW}, \text{RANGE})$ to represent the interval.

- Now, we can use $[\text{LOW}, \text{RANGE})$ to represent one symbol. Multiple symbols (message) can be encoded by $\prod_i [\text{LOW}, \text{RANGE})$. 
Use CDF to Encode the Message
--- Example

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( M = [a, b, a, a, a, e, a, a, b, a] \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04] \).
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

- \( M[1] = a \),
  - LOW = 0.0
  - RANGE = \( P[a] = 0.67 \)
Use CDF to Encode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( M = [a, b, a, a, a, e, a, a, b, a] \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]. \)
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

- \( M[1] = a, \)
  - \( LOW = 0.0 \)
  - \( RANGE = P[a] = 0.67 \)

- \( M[2] = b, \)
  - \( LOW = LOW + PC[b] \times RANGE = 0.0 + 0.67 \times 0.67 \)
  - \( = 0.44890000000000 \)
  - \( RANGE = RANGE \times P[b] = 0.67 \times 0.11 \)
  - \( = 0.07370000000000 \)
Use CDF to Encode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( M = [a, b, a, a, a, e, a, a, b, a] \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04] \).
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

- \( M[2] = b \),
  - \( LOW = LOW + PC[b] \times RANGE = 0.0 + 0.67 \times 0.67 \)
    - \( = 0.44890000000000 \)
    - \( RANGE = RANGE \times P[b] = 0.67 \times 0.11 \)
      - \( = 0.07370000000000 \)

- \( M[3] = a \),
  - \( LOW = LOW + PC[a] \times RANGE \)
    - \( = 0.44890000000000 + 0.0 \times 0.07370000000000 \)
      - \( = 0.44890000000000 \)
    - \( RANGE = RANGE \times P[a] \)
      - \( = 0.07370000000000 \times 0.67 \)
        - \( = 0.04937900000000 \)
Use CDF to Encode the Message
--- Example (continued)

- Alphabet: \[\sum = \{a, b, c, d, e, f\}\]
- \(M = [a, b, a, a, e, a, a, b, a]\)
- \(P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]\).
- \(P_{CDF} = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]\)

- \(M[3] = a,\)
  - LOW = LOW + PC[a] \* RANGE
    - LOW = 0.44890000000000 + 0.0 \* 0.07370000000000
    - LOW = 0.44890000000000
  - RANGE = RANGE \* P[a]
    - RANGE = 0.07370000000000 \* 0.67
    - RANGE = 0.04937900000000

- \(M[4] = a,\)
  - LOW = LOW + PC[a] \* RANGE
    - LOW = 0.44890000000000 + 0.0 \* 0.04937900000000
    - LOW = 0.44890000000000
  - RANGE = RANGE \* P[a]
    - RANGE = 0.04937900000000 \* 0.67
    - RANGE = 0.03308393000000
Use CDF to Encode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( M = [a, b, a, a, e, a, a, b, a] \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04] \).
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

- \( M[4] = a, \)
  - LOW = LOW + PC[a] * RANGE
    = 0.4489 * 0.049379 + 0.0 * 0.049379
    = 0.049379
  - RANGE = RANGE * P[a]
    = 0.049379 * 0.67
    = 0.0330839

- \( M[5] = a, \)
  - LOW = LOW + PC[a] * RANGE
    = 0.4489 * 0.049379 + 0.0 * 0.049379
    = 0.049379
  - RANGE = RANGE * P[a]
    = 0.049379 * 0.67
    = 0.0330839
Use CDF to Encode the Message
--- Example (continued)

- **Alphabet**  \[ \sum = \{a, b, c, d, e, f\} \]
- **M** = [a, b, a, a, e, a, a, b, a]
- **P** = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04].
- **PC** = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]

- **M[5] = a**,
  - LOW = LOW + PC[a] * RANGE
    = 0.44890000000000 + 0.0 * 0.04937900000000
    = 0.44890000000000
  - RANGE = RANGE * P[a]
    = 0.03308393000000 * 0.67
    = 0.02216623310000

- **M[6] = e**,
  - LOW = LOW + PC[e] * RANGE
    = 0.44890000000000 + 0.91 * 0.02216623310000
    = 0.46907127212100
  - RANGE = RANGE * P[e]
    = 0.02216623310000 * 0.05
    = 0.00110831165500
Use CDF to Encode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( M = [a, b, a, a, e, a, a, b, a] \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]. \)
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

- \( M[6] = e, \)
  - LOW = LOW + PC[e] * RANGE
    - LOW = 0.44890000000000 + 0.91 * 0.02216623310000
    - LOW = 0.46907127212100
  - RANGE = RANGE * P[e]
    - RANGE = 0.02216623310000 * 0.05
    - RANGE = 0.00110831165500

- \( M[7] = a, \)
  - LOW = LOW + PC[a] * RANGE
    - LOW = 0.46907127212100 + 0.0 * 0.00110831165500
    - LOW = 0.46907127212100
  - RANGE = RANGE * P[a]
    - RANGE = 0.00110831165500 * 0.67
    - RANGE = 0.00074256880885
Use CDF to Encode the Message --- Example (continued)

- Alphabet $\sum = \{a, b, c, d, e, f\}$
- $M = [a, b, a, a, e, a, a, b, a]$
- $P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]$.
- $PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]$

- $M[7] = a$,
  - $LOW = LOW + PC[a] * RANGE$
    - $= 0.46907127212100 + 0.0 * 0.00110831165500$
    - $= 0.46907127212100$
  - $RANGE = RANGE * P[a]$
    - $= 0.00110831165500 * 0.67$
    - $= 0.00074256880885$

- $M[8] = a$,
  - $LOW = LOW + PC[a] * RANGE$
    - $= 0.46907127212100 + 0.0 * 0.00074256880885$
    - $= 0.46907127212100$
  - $RANGE = RANGE * P[a]$
    - $= 0.00074256880885 * 0.67$
    - $= 0.0004975211019295$
Use CDF to Encode the Message
--- Example (continued)

- Alphabet $\sum = \{a, b, c, d, e, f\}$
- $M = [a, b, a, a, a, e, a, a, b, a]$
- $P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]$. 
- $PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]$

- $M[8] = a$,
  - $LOW = LOW + PC[a] \times RANGE$
    - $= 0.46907127212100 + 0.0 \times 0.00074256880885$
    - $= 0.46907127212100$
  - $RANGE = RANGE \times P[a]$
    - $= 0.00074256880885 \times 0.67$
    - $= 0.0004975211019295$

- $M[9] = b$,
  - $LOW = LOW + PC[b] \times RANGE$
    - $= 0.46907127212100 + 0.67 \times 0.0004975211019295$
    - $= 0.469404611259293$
  - $RANGE = RANGE \times P[b]$
    - $= 0.0004975211019295 \times 0.11$
    - $= 0.00005472732121245$
Use CDF to Encode the Message
--- Example (continued)

- Alphabet \[ \sum = \{a, b, c, d, e, f\} \]
- \( M = [a, b, a, a, a, e, a, a, b, a] \)
- \( P=[0.67, 0.11, 0.07, 0.06, 0.05, 0.04]. \)
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

- \( M[9] = b, \)
  - \( LOW = LOW + PC[b] \times RANGE \)
  - \( = 0.46907127212100 + 0.67 \times 0.0004975211019295 \)
  - \( = 0.469404611259293 \)
  - \( RANGE = RANGE \times P[b] \)
  - \( = 0.0004975211019295 \times 0.11 \)
  - \( = 0.000054727321212245 \)

- \( M[10] = a, \)
  - \( LOW = LOW + PC[a] \times RANGE \)
  - \( = 0.469404611259293 + 0.0 \times 0.0737 \)
  - \( = 0.469404611259293 \)
  - \( RANGE = RANGE \times P[a] \)
  - \( = 0.000054727321212245 \times 0.67 \)
  - \( = 0.00003666730521220415 \)
Use CDF to Encode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( M = [a, b, a, a, a, e, a, a, b, a] \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04] \)
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)
- \( \text{LOW} = 0.469404611259293 \)
  \( \text{RANGE} = 0.00003666730521220415 \)
- OUTPUT 0.46942

![Figure 1. Illustration of arithmetic coding for message [abaaeaaba]](image-url)
Update Frequency in Arithmetic Encoding

- A *static zero-order model* in the example.
- Dynamic update is more accurate.
  - Initially we have a frequency distribution.
  - Every time we process a new symbol, update its frequency immediately.
Two main parts in the arithmetic coding

- Update frequency
- Update subinterval

Initially we have the interval \([\text{Low}, \text{range})\] as \([0, 1)\)

for symbol \(s\), the interval is updated by

- **Low**: \(L = L + R \times \sum_{j=1}^{s-1} P[j] \) This summation can be pre-calculated.
- **Range**: \(R = R \times P[s] \)
Decode the Message  
--- Example

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \(|M| = 10\)
- \(P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]\).
- \(PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]\)
- \(M = 0.46942\)

- **Recover symbol #1.**
  - LOW = 0.0
  - RANGE = 1.0
  - \(M = 0.46942\), lies in the interval [0.0, 0.67)
  - Output symbol \(a\)
  - we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)\)
  - Update the \(M\): \(M = (M - \text{newLOW}) / \text{newRANGE}\).
  - We have \(M = 0.46942/0.67 = 0.70062686567164\)
Decode the Message
--- Example (continued)

- Alphabet \[ \sum = \{a, b, c, d, e, f\} \]
- \(|M| = 10\)
- \(P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]\).
- \(PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]\)

**Recover symbol #1.**
- \(LOW = 0.0\)
- \(RANGE = 1.0\)
- \(M = 0.46942\), lies in the interval \([0.0, 0.67)\). Output symbol \(a\)
- we have the interval \([\text{newLOW, newRANGE}] = [0.0, 0.67]\)
- Update the \(M\): \(M = (M - \text{newLOW}) / \text{newRANGE}\).
- We have \(M = 0.46942/0.67 = 0.700626867164\)

**Recover symbol #2.**
- \(LOW = 0.0\)
- \(RANGE = 1.0\)
- \(M = 0.7006268667164\), lies in the interval \([0.67, 0.78)\). Output symbol \(b\)
- we have the interval \([\text{newLOW, newRANGE}] = [0.67, 0.11]\)
- Update the \(M\): \(M = (M - \text{newLOW}) / \text{newRANGE}\).
- We have \(M = (0.7006268667164 - 0.67)/ 0.11 = 0.27842605156036\)
Decode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \(|M| = 10\)
- \(P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]\).
- \(PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]\)

**Recover symbol #2.**
- LOW = 0.0
- RANGE = 1.0
- \(M = 0.70062686567164\), lies in the interval \([0.67, 0.78)\). Output symbol \(b\)
- we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.67, 0.11)\)
- Update the \(M\): \(M = (M-\text{newLOW}) / \text{newRANGE}\).
- We have \(M = (0.70062686567164 -0.67)/ 0.11 = 0.27842605156036\)

**Recover symbol #3.**
- \(M=0.27842605156036\), check the CDF, lies in the interval \([0.0, 0.67)\), so target symbol is \(a\).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)\).
- We have \(M = (0.27842605156036 -0.0)/ 0.67 =0.41556127098561\)
Decode the Message
--- Example (continued)

- Alphabet $\sum = \{a, b, c, d, e, f\}$
- $|M| = 10$
- $P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]$.
- $PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]$

**Recover symbol #3.**
- $M = 0.27842605156036$, check the CDF, lies in the interval $[0.0, 0.67)$, so target symbol is $a$.
- Now, we have the interval $[\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)$.
- We have $M = (0.27842605156036 - 0.0)/0.67 = 0.41556127098561$

**Recover symbol #4.**
- $M = 0.41556127098561$, check the CDF, lies in the interval $[0.0, 0.67)$, so target symbol is $a$.
- Now, we have the interval $[\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)$.
- We have $M = (0.41556127098561 - 0.0)/0.67 = 0.62024070296360$
Decode the Message
--- Example (continued)

- Alphabet $\sum = \{a, b, c, d, e, f\}$
- $|M| = 10$
- $P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]$
- $PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]$
- **Recover symbol #4.**
  - $M = 0.41556127098561$, check the CDF, lies in the interval $[0.0, 0.67)$, so target symbol is $a$.
  - Now, we have the interval $[\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)$.
  - We have $M = (0.41556127098561-0.0)/0.67 = 0.62024070296360$
- **Recover symbol #5.**
  - $M = 0.62024070296360$, check the CDF, lies in the interval $[0.0, 0.67)$, so target symbol is $a$.
  - Now, we have the interval $[\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)$.
  - We have $M = (0.62024070296360-0.0)/0.67 = 0.92573239248299$
Decode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( |M| = 10 \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04] \)
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

**Recover symbol #5.**
- \( M = 0.62024070296360 \), check the CDF, lies in the interval \([0.0, 0.67)\), so target symbol is \( a \).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)\).
- We have \( M = (0.62024070296360-0.0)/0.67 = 0.92573239248299 \)

**Recover symbol #6.**
- \( M = 0.92573239248299 \), check the CDF, lies in the interval \([0.91, 0.96)\), so target symbol is \( e \).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.91, 0.05)\).
- We have \( M = (0.92573239248299-0.91)/0.05 = 0.31464784965980 \)
Decode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( |M| = 10 \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04] \).
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

**Recover symbol #6.**
- \( M = 0.92573239248299 \), check the CDF, lies in the interval \([0.91, 0.96)\), so target symbol is \( e \).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.91, 0.05)\).
- We have \( M = (0.92573239248299-0.91)/0.05 = 0.31464784965980 \)

**Recover symbol #7.**
- \( M = 0.31464784965980 \), check the CDF, lies in the interval \([0.0, 0.67)\), so target symbol is \( a \).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)\).
- We have \( M = (0.31464784965980-0.0)/0.67 = 0.46962365620866 \)
Decode the Message
--- Example (continued)

- **Alphabet** \( \sum = \{a, b, c, d, e, f\} \)
- \(|M| = 10\)
- \(P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]\).
- \(PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]\)

**Recover symbol #7.**
- \(M = 0.31464784965980\), check the CDF, lies in the interval \([0.0, 0.67)\), so target symbol is \(a\).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)\).
- We have \(M = (0.31464784965980 - 0.0)/ 0.67 = 0.46962365620866\)

**Recover symbol #8.**
- \(M = 0.46962365620866\), check the CDF, lies in the interval \([0.0, 0.67)\), so target symbol is \(a\).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.0, 0.67)\).
- We have \(M = (0.46962365620866 - 0.0)/ 0.67 = 0.70093083016218\)
Decode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \(|M| = 10\)
- \(P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04]\).
- \(PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00]\)

**Recover symbol #8.**
- \(M = 0.46962365620866\), check the CDF, lies in the interval \([0.0, 0.67)\), so target symbol is \(a\).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}] = [0.0, 0.67)\).
- We have \(M = (0.46962365620866 - 0.0) / 0.67 = 0.70093083016218\)

**Recover symbol #9.**
- \(M = 0.70093083016218\), check the CDF, lies in the interval \([0.67, 0.78)\), so target symbol is \(b\).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}] = [0.67, 0.11)\).
- We have \(M = (0.70093083016218 - 0.67) / 0.11 = 0.28118936511073\)
Decode the Message
--- Example (continued)

- Alphabet \( \sum = \{a, b, c, d, e, f\} \)
- \( |M| = 10 \)
- \( P = [0.67, 0.11, 0.07, 0.06, 0.05, 0.04] \).
- \( PC = [0.0, 0.67, 0.78, 0.85, 0.91, 0.96, 1.00] \)

**Recover symbol #9.**
- \( M = 0.70093083016218 \), check the CDF, lies in the interval \([0.67, 0.78)\),
  so target symbol is \( b \).
- Now, we have the interval \([\text{newLOW}, \text{newRANGE}) = [0.67, 0.11)\).
- We have \( M = (0.70093083016218 - 0.67)/0.11 = 0.28118936511073 \)

**Recover symbol #10.**
- \( M = 0.28118936511073 \), check the CDF, lies in the interval \([0.0, 0.67)\), so
  target symbol is \( a \).
- Now, we have recovered all the symbols. STOP decoding here.
Ideal Arithmetic Encoding
---- Transmit the Message

/*Use the high-precision fractional number to transmit the message*/

ALGORITHM ARITHMETIC_SEND_V0 (message M)
1. INIT the transmission channel.
2. transmit the alphabet $\sum$, preceded by the size of the alphabet $|\sum|$
3. transmit the probability distribution $P$
4. Encode the message with [LOW, RANGE]
5. transmit any value between [LOW, RANGE) using a high-precision fractional number
6. END transmission.

END ALGORITHM
/*Receive and decode the message*/

ALGORITHM ARITHMETIC_RECEIVE_V0 (received stream)
   1. INIT the receiving channel.
   2. Receive the alphabet $\sum$, preceded by the size of the alphabet $|\sum|$
   3. Receive the probability distribution $P$
   4. Receive the high-precision fractional value
   5. END receiving.
   6. Decode the message from the fractional value.
END ALGORITHM
Ideal Arithmetic Encoding

---- Encode the Message

/* Use an idealized arithmetic coder to represent the m-symbol message \( M \), where \( 1 \leq M[i] \leq n \) for \( 1 \leq i \leq m = |M| \). Normalized symbol probabilities are assumed to be given by static vector \( P \), with \( \sum_{i=1}^{n} P[i] = 1 \). */

ALGORITHM ARITHMETIC_IDEAL_ENCODE(M)
1. set \( L = 0 \) and \( R = 1 \)
2. FOR \( i = 1 \) to \( |M| \) DO
3. \hspace{.5cm} set \( s = M[i] \)
4. \hspace{.5cm} set \( L = L + R \times \sum_{j=1}^{i-1} P[j] \)
5. \hspace{.5cm} set \( R = R \times P[s] \)
6. END FOR
7. /*If the algorithm has to be adaptive, code has to be inserted before the above ‘end’ statement to recompute probabilities \( p(a_i)^s \).*/
8. transmit \( V \), where \( V \) is the shortest (fewest bits) binary fractional number that satisfies \( L \leq V < L + R \)

END ALGORITHM

September 23, 2004
Weifeng Sun
/* Decode and return an m-symbol message assuming an idealized arithmetic coder. */

ALGORITHM ARITHMETIC_IDEAL_DECODE(m)
1. Let V be the fractional value transmitted by the encoder
2. FOR i = 1 to m DO
3. Determine s such that \( \sum_{j=1}^{s-1} P[j] \leq V < \sum_{j=1}^{s} P[j] \)
4. Recover L and R from s.
5. Update V = (V-L)/R
6. Output M[i]=s
7. END FOR
8. RETRUN M
END ALGORITHM
Efficiency of the Ideal Arithmetic Coding

- The average length per symbol using the arithmetic coding is $H(X) \leq l_A \leq H(X) + 2/m$,
  - where $m$ is the length of the message.
  - Proved in the text book (Sayood, page 91).
- So, it is guaranteed that the encoding rate is close to the entropy given a long enough message.
Compare Huffman Coding and Arithmetic Coding

- Huffman coding: Create binary (Huffman) tree such that path lengths correspond to symbol probabilities. Use path labels as encodings.

- Arithmetic coding: Combine probabilities of subsequent symbols into a single fixed-point of high precision. Encode that number in binary.

- Arithmetic coding is slower than Huffman coding.
Compare Huffman Coding and Arithmetic Coding (continued)

- **Arithmetic coding efficiency:**
  - $H(X) \leq l_A \leq H(X) + \frac{2}{m}$
  - $m$ is the length of the message

- **Huffman coding efficiency:**
  - $H(S) \leq l_H \leq H(S) + \frac{1}{m}$
  - $m$ is size of the block

- Is Huffman coding more efficient than arithmetic coding?
Compare Huffman Coding and Arithmetic Coding (continued)

- It seems that Huffman coding is more efficient than the arithmetic coding.
  - However, in this case, the size of the codebook will be exponentially big, making Huffman encoding not applicable.

- If the probabilities of the symbols are powers of two, Huffman coding can achieve the entropy. In this case, we cannot do any better with arithmetic coding, no matter how long a sequence we pick.
Also, Average code length for Huffman coding

\[ l_H < H(S) + p_{\text{max}} + 0.086 \]

- \( p_{\text{max}} \) is the probability of the most frequently occurring symbol.

- If the alphabet size is relatively large and the probabilities are not too skewed, \( p_{\text{max}} \) will be generally small. In this case, the Huffman coding is better than the arithmetic coding in favor of the speed.

- However, if the alphabet size is small, and the probabilities are highly unbalanced, arithmetic coding is generally worth the added complexity.
Compare Huffman Coding and Arithmetic Coding (continued)

- Arithmetic coding can handle adaptive coding without much increase in algorithm complexity. It calculates the probabilities on the fly and less primary memory is required for adaptation.
- Canonical Huffman is also fast but use only static or semi-static models.
Compare Huffman Coding and Arithmetic Coding (continued)

- It is not possible to start decoding in the middle of a compressed string which is possible in Huffman by indexing “starting points”.
- So, from random access point of view and from the point of view of compressed domain pattern matching, arithmetic coding is not suitable.
Compare Huffman Coding and Arithmetic Coding (continued)

- For text using static model, Huffman is almost as good as Arithmetic.
- Arithmetic is better suited for image and video compression.
- Once again, Huffman is faster than Arithmetic.
- Moffat’s implementation (1998) is slightly better than Witten’s (1987).
Ideal Arithmetic Coding

Remarks

- High probability events do not decrease the interval $Range$ very much, but low probability events result in a much smaller next interval requiring large number of digits.

- A large interval needs only a few digits. The number of digits required is $-\log(size\ of\ interval)$.

- The size of the final interval is the product of the probabilities of the symbols encoded. Thus a symbol $s$ with probability $p(s)$ contributes $-\log p(s)$ bits to the output which is the symbol’s self-information.
Ideal Arithmetic Coding
---- Remarks

- Theoretically, therefore, arithmetic code can achieve compression identical to the entropy bound. But, finite precision of computer limits the maximum compression achievable.

- Note, the algorithm does not output anything until encoding is completed.

- In practice, it is possible to output most significant digits sequentially during the execution while at the same time utilize the finite precision of the machine effectively.
Ideal Arithmetic Coding
---- Remarks

- We need an arbitrary precision floating point arithmetic operator.
- Decoding cannot start until the data communication finishes.
  - Both problems will be fixed in the binary implementation.
  - As it turns out, arithmetic coding is best accomplished using standard 32 bit and 64 bit integer math. No floating point math is required, nor would it help to use it.
  - What is used instead is an incremental transmission scheme, where fixed size integer state variables receive new bits in at the low end and shift them out the high end, forming a single number that can be as many bits long as are available on the computer's storage medium.
Ideal Arithmetic Coding
---- Underflow

- During the encoding, every time we read the next symbol, we scale the \([\text{LOW}, \text{RANGE})\) to the new (smaller) value according to the probability of symbol.
- Suppose \(\text{RANGE}\) is very small, such that different symbols will be mapped to the same interval \([\text{LOW}, \text{RANGE})\). In this case, it is impossible for the decoder to recover the symbol correctly.
- To solve this problem, we need to ensure that the \(\text{RANGE}\) is as large as possible.
- In the following implementation, we will enforce that \(\text{RANGE}\) is always no less than 0.25 (after Witter’s 1987 paper). That is, if \(\text{RANGE}\) is less than 0.25, we will perform scaling to double it.
Binary Arithmetic Coding

---- Basic Idea

- Shift the L and R left whenever L and L+R have the same prefix.
- When we shift out (and output) the prefix, the range should be re-normalized by shifting the LOW and double the RANGE.
When the interval \([L, L+R)\) lies in the lower half \([0.0, 0.5)\), we can expand this lower half to make it occupy the full interval \([0.0, 1.0)\), and adjust the \(LOW = 2 \times LOW\) and \(RANGE = 2 \times RANGE\).
Binary Arithmetic Coding

---- E2 Mapping

When the interval \([L, L+R)\) lies in the upper half, we can expand this upper half to make it occupy the full interval \([0.0, 1.0)\), and adjust the \(LOW = 2 \times (LOW - 0.5)\) and \(RANGE = 2 \times RANGE\).
Binary Arithmetic Coding

--- E3 Mapping

- When the interval \([L, L+R)\) straddles the middle point 0.5, \(L\) is translated by 0.25 before \(L\) and \(R\) are doubled. So, we adjust \(\text{LOW} = 2 \times (\text{LOW} - 0.25)\) and \(\text{RANGE} = 2 \times \text{RANGE}\).

- But, what to output?

![Diagrams](attachment://diagrams.png)
Binary Arithmetic Coding

---- E3 Mapping

- Lots of possibilities
  - Output 01, interval [0.25, 0.50)
  - Output 10, interval [0.5, 0.75)
  - But, how about new scaled interval straddles 0.5 again?

![Diagram](image-url)
Binary Arithmetic Coding

---- E3 Mapping, new scaled interval straddles 0.5 again

- One bit is output for each scaling.
- Suppose the current subinterval has been expanded a total of three times.
- Suppose the next bit turns out to be zero, which means it lies in [0.0, 0.5). Then the next three bits will be ones, since the arrow is not only in the top half of the bottom half of the original range --- [0.25, 0.5), with binary encoding starting with 01, more specifically ---, but in the top quarter--- [0.375, 0.5), with binary encoding starting with 011, more specifically ---, and moreover the top eighth--- [0.4375, 0.5), with binary encoding starting with 0111, more specifically ---, of that half-this is why the expansion can occur three times.
Binary Arithmetic Coding

---- E3 Mapping, new scaled interval straddles 0.5 again

- Suppose the current subinterval has been expanded a total of three times
- Similarly, if the next bit turns out to be a one, it will be followed by three zeros. Consequently, we need only count the number of expansions and follow the next bit by that number of opposites.
Binary Arithmetic Coding

--- Pseudo Code

- Assume \( b = 32 \).
- The corresponding values for arithmetic coding, real-number interpretation and scaled integer interpretations.

<table>
<thead>
<tr>
<th>Decimal Encoding</th>
<th>1.00</th>
<th>0.50</th>
<th>0.25</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL Binary Encoding</td>
<td>(\overline{32.1s})</td>
<td>(\overline{31.0s})</td>
<td>(\overline{30.0s})</td>
<td>(\overline{32.0s})</td>
</tr>
<tr>
<td>0.1111...111</td>
<td>0.1000...000</td>
<td>0.010000...000</td>
<td>0.000000...000</td>
<td></td>
</tr>
<tr>
<td>SHORT-FORM Binary Encoding</td>
<td>(\overline{32.1s})</td>
<td>(\overline{31.0s})</td>
<td>(\overline{30.0s})</td>
<td>(\overline{32.0s})</td>
</tr>
<tr>
<td>1111...111</td>
<td>1000...000</td>
<td>010000...000</td>
<td>0000...000</td>
<td></td>
</tr>
</tbody>
</table>

- Rather than probabilities, the algorithm deals with frequencies.
Binary Arithmetic Coding

----Encoding One Symbol

/*Arithmetically encode the range [l/t, h/t) using fixed-precision integer arithmetic. The state variables L and R are modified to reflect the new range, and then renormalized to store the initial and final invariants \(2^{b-2} \leq R < 2^{b-1}, 0 \leq L < 2^{b-2-2}, L+R < 2^b\) (or, 0.25 < realR < 0.50, 0 < realL < 0.75, realL+realR < 1.00) */

ALGORITHM ARITHMETIC_ENCODE_ONE_SYMBOL(l, h, t)

1. set \( L = L + R \times l / t \)
2. set \( R = R \times h / t - R \times l / t \)
3. WHILE \( R \leq 2^{b-2} \) DO
4. /* renormalize \( R \), adjust \( L \), and output one bit */
5. IF \( L + R \leq 2^{b-1} \) THEN
6. \( \text{bit_plus_follow}(0) \)
7. ELSE IF \( 2^{b-1} \leq L \) THEN
8. \( \text{bit_plus_follow}(1) \)
9. \( \text{set } L = L - 2^{b-1} \) /* clear the leftmost one */
10. ELSE
11. \( \text{set } \text{bits_outstanding} = \text{bits_outstanding} + 1 \)
12. \( \text{set } L = L - 2^{b-2} \)
13. END IF
14. \( \text{set } L = 2 \times L \) and \( R = 2 \times R \) /* shift \( L \) and \( R \) left by one bit */
15. END WHILE

END ALGORITHM
Binary Arithmetic Coding

----E3, Output Outstanding Bits

/*Write the bit x (value 0 or 1) to the output bit stream, plus any outstanding following bits, which are known to be of opposite polarity. */

ALGORITHM bit_plus_follow(x)
    1. put_one_bit(x)
    2. WHILE bits_outstanding > 0 DO
    3.     put_one_bit(1-x)
    4.     set bits_outstanding = bits_outstanding - 1
    5. END WHILE
END ALGORITHM
Binary Arithmetic Coding
----Decoding One Symbol

/************************************************************************* 
 Adjust the decoder’s state variable L and R to reflect the changes made in the 
 encoder during the corresponding call to ARITHMETIC_ENCODE_SEQUENCE() */ 
**************************************************************************/

ALGORITHM ARITHMETIC_DECODE_ONE_SYMBOL(l, h, t)
1. set L = L + R * I / t 
2. set R = R * h / t - R * I / t 
3. WHILE R <= 2^{b-2} DO 
4.   /* Renormalize R, adjust L and V, and shift the next input bit into V */ 
5.       IF L + R <= 2^{b-1} THEN 
6.           /* do nothing */ 
7.       ELSE IF 2^{b-1} <= L THEN 
8.           set L = L - 2^{b-1} /* clear the leftmost one */ 
9.           set V= V - 2^{b-1} /* clear the leftmost one */ 
10.      ELSE 
11.        set L = L - 2^{b-2} 
12.        set V = V - 2^{b-2} 
13.      END IF 
14.      set L = 2 * L, R = 2 * R and V = 2 * V + get_one_bit() 
15. END WHILE 
END ALGORITHM
Binary Arithmetic Coding

--- Encoding the Message

/* Use an arithmetic code to represent the n-symbol message M, where 1 <= M[i] <= n for 1 <= i <= |M| */

ALGORITHM ARITHMETIC_ENCODE_SEQUENCE(M)
1. /* 1-14: calculate the cumulative frequency */
2.  FOR s = 0 to n DO
3.      set PC[s] = 0;
4.  END FOR
5.  FOR i = 1 to |M| DO
6.      set s = M[i]
7.      PC[s] +=
8.  END FOR
9.  encode and transmit |M| and n
10.  FOR s = 1 to n DO
11.     encode and transmit (1+PC[s]);
12.     PC[s] = PC[s-1] + PC[s]
13.  END FOR
14.
15. /* 16: init for encoding */
16.  set L = 0, R = 2^(s-1) (corresponding to 0.5), bits_outstanding = 0
17.
18. /* 19-23: encode the message */
19.  FOR i = 1 to |M| DO
20.     set s = M[i]
21.     l = PC[s-1], h = PC[s], t = |M|
22.     ARITHMETIC_ENCODE_ONE_SYMBOL(l, h, t)
23.  END FOR
24.
25.    encode and transmit L as one integer (b bits long) using bit_plus_follow

END ALGORITHM
Binary Arithmetic Coding

----Decoding the Message

/*Decode and return an m-symbol message M using an arithmetic code.*/

ALGORITHM ARITHMETIC_DECODE_SEQUENCE()
    1. /* 1-8: receive and decode the cumulative frequency */
    2. receive and decode m=[M] and n
    3. set PC[0] = 0
    4. FOR s = 1 to n DO
    5.       receive and decode value pc
    6.       set PC[s] = pc - 1;
    7.       set PC[s] = PC[s-1] + PC[S]
    8.     END FOR
    9.
    10. /* init for decoding */
    11. set R = 2^{b-1}
    12. set L = 0
    13. set V = get_one_integer(b)
    14.
    15. /* receive and decode the message */
    16. FOR i = 1 to |M| DO
    17.        set target = decode_target(m);
    18.        determine s such that PC[s-1] <= target < PC[s]
    19.          l = PC[s-1], h = PC[s], t = m
    20.        ARITHMETIC_DECODE_ONE_SYMBOL(l, h, t)
    21.        set M[i] = s
    22.     END FOR
    23. RETURN M
END ALGORITHM

ALGORITHM decode_target(t)
  1. RETURN (((V - L + 1) * t) - 1) / R
END ALGORITHM
Binary Arithmetic Coding

----Example

- Check the notes.
Application Arithmetic Coding

- Image compression
- Video compression
- Lossless/lossy

Why?
- The size of the alphabet is small, and the probabilities are highly unbalanced.