COMSATS Institute of Information Technology

ECI750 Multimedia Data Compression

Lecture 7

Arithmetic Coding

Dr. Shadan Khattak

Department of Electrical Engineering

COMSATS Institute of Information Technology - Abbottabad

Arithmetic Coding (1)

- Recall that $H(s) \leq R_{Huffman} \leq H(s) + 1$ i.e.,
  - Huffman coding guarantees a code rate R which is within 1 bit of the entopy of the source.
- It has been shown[1] that Huffman algorithm has a code rate within $p_{max} + 0.086$ of the entropy.
- For large values of p_{max}, Huffman coding is inefficient.
- Extended Huffman code can solve this problem, but not always!

Arithmetic Coding (2)

- **Example**
  - Consider a source that puts out iid letters from the alphabet $A = \{a_1, a_2, a_3\}$ with the probability model: $P\{a_1\} = 0.95, P\{a_2\} = 0.02, P\{a_3\} = 0.03$
  - Entropy = 0.335 bits/symbol
  - Huffman Code:

Letter	Codeword
a_1	0
a_2	11
a_3	10

  - Average codeword length: 1.05 bits/symbol
  - Redundancy: 0.715 bits/symbol (213% of the entropy)

Arithmetic Coding (3)

- **Example (2)**
  - Extended Huffman Code:

Letter	Probability	Code
a_1a_1	0.9025	0
a_1a_2	0.0190	111
a_1a_3	0.0285	100
a_2a_1	0.0190	1101
a_2a_2	0.0004	110011
a_2a_3	0.0006	110001
a_3a_1	0.0285	101
a_3a_2	0.0006	110010
a_3a_3	0.0009	110000

  - Average codeword length: 0.611 bits/symbol (in terms of original alphabet)
  - Redundancy: 82% of the entropy

Arithmetic Coding (4)

- **Example (3)**
  - Redundancy drops to acceptable levels when we block eight symbols together.
  - The alphabet size for this level of blocking is 6561.
  - A code of this size is impractical for many applications.
  - In order to find the Huffman codeword for a particular sequence of length m, we need codewords for all possible sequences of length m.
  - The *arithmetic coding* technique allows to assign codewords to particular sequences without having to generate codes for all sequences.

Arithmetic Coding (5)

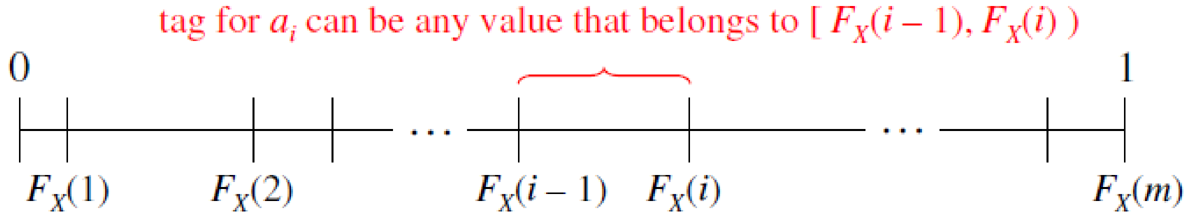
- **Arithmetic Coding**
  - Two step procedure:
    - $Step$ 1: A unique identifier or tag is generated for the sequence to be encoded.
    - $Step$ 2: A unique binary code is given to the tag generated in Step 1.

Arithmetic Coding (6)

- **Coding Sequence**
  - One possible set of tags for representing sequences of symbols are the numbers in the unit interval [0,1).
  - Shannon started (in 1948) the idea of using cumulative density function (cdf) for codeword design.
  - Peter Elias (Fano’s student and Huffman’s classmate) came up with a recursive implementation for this idea.
  - First practical approach published in 1976, by Rissanen (IBM).
  - Made well-known by a paper in Communication of the ACM, by Witten et al. in 1987.

Arithmetic Coding (7)

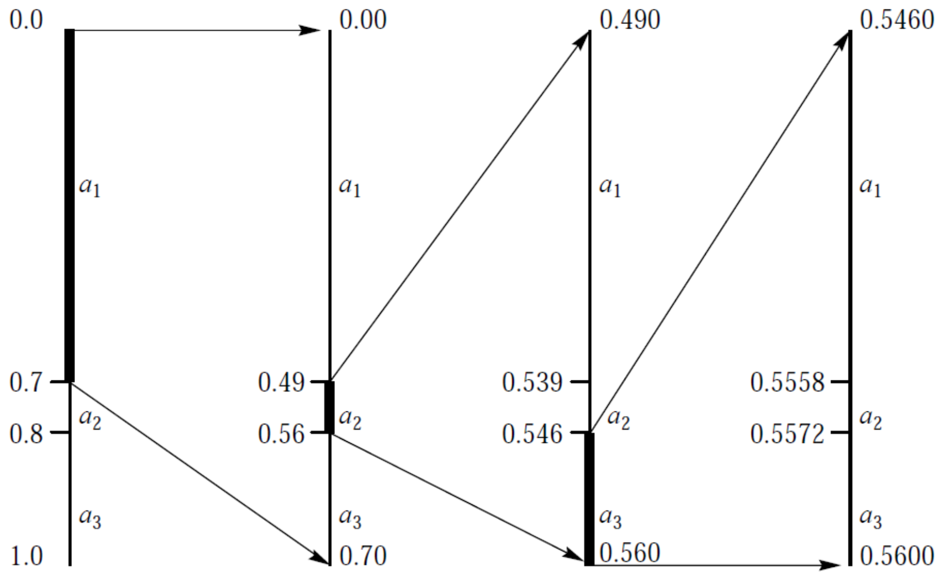
- **<u>Generating a Tag</u>**
  - The principle is to reduce the size of the interval in which the tag resides as more and more elements of the sequence are received.
  - We start by first dividing the unit interval into subintervals of the form $\left[F_x(i-1), F_x(i)\right), i = 1, \ldots, m$.
  - We associate the subinterval $\left[F_x(i-1), F_x(i)\right)$ with the symbol a_i.
  - Suppose the first symbol was a_k.
    - Then, the interval containing the tag value will be the subinterval $[F_x(k-1), F_x(k))$



Dr. Shadan Khattak
Department of Electrical Engineering
CIIT - Abbottabad

Arithmetic Coding (8)

- **Example:**
  - Consider a three-letter alphabet $A = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7, P(a_2) = 0.1, P(a_3) = 0.2$. Also, $F_x(1) = 0.7, F_x(2) = 0.8, F_x(3) = 1$

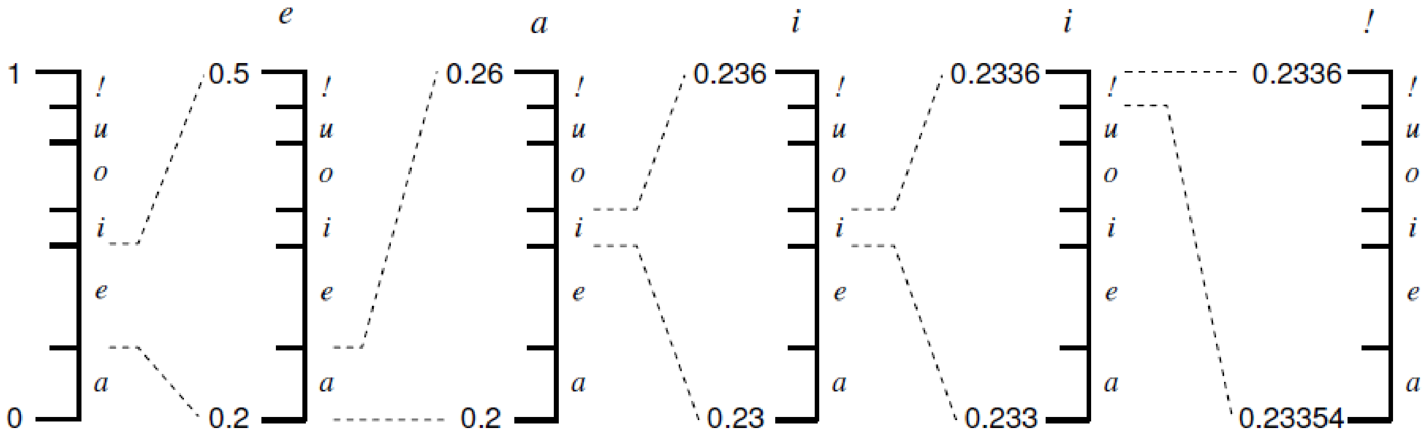


Dr. Shadan Khattak
Department of Electrical Engineering
CIIT - Abbottabad

Arithmetic Coding (9)

- **Example 2:**
  - **Message: “eaii!”**

Symbol	Probability	Interval
a	.2	[0, 0.2)
e	.3	[0.2, 0.5)
i	.1	[0.5, 0.6)
o	.2	[0.6, 0.8)
u	.1	[0.8, 0.9)
!	.1	[0.9, 1.0)



Arithmetic Coding (10)

- **Tag Selection for a message**
  - Since the intervals of messages are disjoint, we can pick any values from the interval as the tag
    - A popular choice is the lower limit of the interval.
  - Single symbol example: if the mid-point of the interval $[F_x(a_{i-1}), F_x(a_i))$ is used as the tag $T_x(a_i)$ of symbol a_i, then

$$T_x(a_i) = \sum_{k=1}^{i-1} P(X = k) + \frac{1}{2} P(X = i)$$

$$= F_x(i-1) + \frac{1}{2} P(X = i)$$

Arithmetic Coding (11)

- **Tag Selection for a message (2)**
  - To generate a unique tag for a long message, we need an ordering on all message sequences
    - A logical choice of such ordering rule is the lexicographic ordering of the message.
  - With lexicographical ordering, for all messages of length m, we have

$$T_x^{(m)}(x_i) = \sum_{y<x_i} P(y) + \frac{1}{2} P(x_i)$$

Where $y < x_i$ means y precedes x_i in the ordering of all messages.

  - But the problem is that we need $P(y)$ for all $y < x_i$ to compute $T_x(x_i)$.

Arithmetic Coding (12)

- **Recursive computation of Tags (1)**
  - Assume that we want to code the outcome of rolling a fair die for three times. Let's compute the upper and lower limits of the message "3-2-2"
    - For the first outcome "3", we have:

$$l^{(1)} = F_x(2), u^{(1)} = F_x(3)$$

    - For the second outcome "2", we have the upper limit

$$F_x^{(2)}(32) = [P(x_1 = 1) + P(x_1 = 2)] + P(\boldsymbol{x} = 31) + P(\boldsymbol{x} = 32)$$
$$= F_x(2) + P(x_1 = 3)P(x_1 = 1) + P(x_1 = 3)P(x_2 = 2)$$
$$= F_x(2) + P(x_1 = 3)F_x(2)$$
$$= F_x(2) + [F_x(3) - F_x(3)]F_x(2)$$

Thus, $u^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right)F_x(2)$

Similarly, the lower limit $F_x^{(2)}(31)$ is $l^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right)F_x(1)$

Dr. Shadan Khattak
Department of Electrical Engineering
CIIT - Abbottabad

Arithmetic Coding (13)

- <u>**Recursive computation of Tags (2)**</u>
  - For the third outcome “2”, we have

$$l^{(3)} = F_x^{(3)}(321), u^{(3)} = F_x^{(3)}(322)$$

  - Using the same approach above, we have

$$F_x^{(3)}(321) = F_x^{(2)}(31) + [F_x^{(2)}(32) - F_x^{(2)}(31)]F_x(1)$$
$$F_x^{(3)}(322) = F_x^{(2)}(31) + [F_x^{(2)}(32) - F_x^{(2)}(31)]F_x(2)$$

  - Therefore,

$$l^{(3)} = l^{(2)} + [u^{(2)} - l^{(2)}]F_x(1)$$
$$u^{(3)} = l^{(2)} + [u^{(2)} - l^{(2)}]F_x(2)$$

Arithmetic Coding (14)

- **Recursive computation of Tags (3)**
  - In genera, we can show that for any sequence

$$x = (x_1 x_2 \dots x_n),$$
$$l^{(n)} = l^{(n-1)} + [u^{(n-1)} - l^{(n-1)}]F_x(x_n - 1)$$
$$u^{(n)} = l^{(n-1)} + [u^{(n-1)} - l^{(n-1)}]F_x(x_n)$$

  - If the mid-point is used as the tag, then

$$T_x(x) = \frac{u^{(n)} + l^{(n)}}{2}$$

  - So, we only need the CDF of the source alphabet to compute the tag of any long messages.

Arithmetic Coding (15)

Example 4.3.5: Generating a tag

Consider the source in Example 3.2.4. Define the random variable $X(a_i) = i$. Suppose we wish to encode the sequence **1 3 2 1**. From the probability model we know that

$$F_X(k) = 0, \; k \leq 0, \quad F_X(1) = 0.8, \quad F_X(2) = 0.82, \quad F_X(3) = 1, \quad F_X(k) = 1, \; k > 3.$$

We can use Equations (4.9) and (4.10) sequentially to determine the lower and upper limits of the interval containing the tag. Initializing $u^{(0)}$ to 1, and $l^{(0)}$ to 0, the first element of the sequence **1** results in the following update:

$$l^{(1)} = 0 + (1-0)0 = 0$$
$$u^{(1)} = 0 + (1-0)(0.8) = 0.8.$$

That is, the tag is contained in the interval $[0, 0.8)$. The second element of the sequence is **3**. Using the update equations we get

$$l^{(2)} = 0 + (0.8-0)F_X(2) = 0.8 \times 0.82 = 0.656$$
$$u^{(2)} = 0 + (0.8-0)F_X(3) = 0.8 \times 1.0 = 0.8.$$

Arithmetic Coding (16)

Therefore, the interval containing the tag for the sequence **1 3** is $[0.656, 0.8)$. The third element, **2**, results in the following update equations:

$$l^{(3)} = 0.656 + (0.8 - 0.656)F_X(1) = 0.656 + 0.144 \times 0.8 = 0.7712$$

$$u^{(3)} = 0.656 + (0.8 - 0.656)F_X(2) = 0.656 + 0.144 \times 0.82 = 0.77408$$

and the interval for the tag is $[0.7712, 0.77408)$. Continuing with the last element, the upper and lower limits of the interval containing the tag are

$$l^{(4)} = 0.7712 + (0.77408 - 0.7712)F_X(0) = 0.7712 + 0.00288 \times 0.0 = 0.7712$$

$$u^{(4)} = 0.7712 + (0.77408 - 0.1152)F_X(1) = 0.7712 + 0.00288 \times 0.8 = 0.773504$$

and the tag for the sequence **1 3 2 1** can be generated as

$$\bar{T}_X(1321) = \frac{0.7712 + 0.773504}{2} = 0.772352.$$

♦

Arithmetic Coding (17)

- **Deciphering the Tag**
  - The algorithm to deciphering the tag is quite straightforward:
    1. Initialize $l^{(0)} = 0$, $u^{(0)} = 1$.
    2. For each k, find $t^* = (T_X(\mathbf{x}) - l^{(k-1)})/(u^{(k-1)} - l^{(k-1)})$.
    3. Find the value of x_k for which $F_X(x_k - 1) \leq t^* \leq F_X(x_k)$.
    4. Update $u^{(k)}$ and $l^{(k)}$.
    5. If there are more symbols, go to step 2.
  - In practice, a special "end-of-sequence" symbol is used to signal the end of a sequence.

Arithmetic Coding (18)

- **Deciphering the Tag**

❑ Given $\mathcal{A} = \{1, 2, 3\}$, $F_X(1) = 0.8$, $F_X(2) = 0.82$, $F_X(3) = 1$, $l^{(0)} = 0$, $u^{(0)} = 1$. If the tag is $T_X(\mathbf{x}) = 0.772352$, what is $\mathbf{x}$?

$t^* = (0.772352 – 0)/(1 – 0) = 0.772352$
$F_X(0) = 0 \leq t^* \leq 0.8 = F_X(1)$
$l^{(1)} = 0$, $u^{(1)} = 0.8$.
→ 1

$t^* = (0.772352 – 0)/(0.8 – 0) = 0.96544$
$F_X(2) = 0.82 \leq t^* \leq 1 = F_X(3)$
$l^{(2)} = 0.656$, $u^{(2)} = 0.8$.
→ 13

$t^* = (0.772352 – 0.656)/(0.8 – 0.656) = 0.808$
$F_X(1) = 0.8 \leq t^* \leq 0.82 = F_X(2)$
$l^{(3)} = 0.7712$, $u^{(3)} = 0.77408$.
→ 132

$t^* = (0.772352 – 0.7712)/(0.77408 – 0.7712) = 0.4$
$F_X(1) = 0 \leq t^* \leq 0.8 = F_X(1)$
→ 1321

Note:
$l^{(n)} = l^{(n-1)} + (u^{(n-1)} – l^{(n-1)})F_X(x_n–1)$
$u^{(n)} = l^{(n-1)} + (u^{(n-1)} – l^{(n-1)})F_X(x_n)$

Arithmetic Coding (19)

Binary Code for the Tag

- If the mid-point for an interval is used as the tag $T_x(x)$, a binary code for $T_x(x)$ is the binary representation of the number truncated to $l(x) = \left\lceil \log\left(\frac{1}{P(x)}\right) \right\rceil + 1$ bits.
- For example, $A = \{a_1, a_2, a_3, a_4\}$ with probabilities {0.5, 0.25, 0.125, 0.125}, a binary code for each symbol is as follows:

Symbol	F_X	$\bar{T}_X$	In Binary	$\lceil \log \frac{1}{P(x)} \rceil + 1$	Code
1	.5	.25	.010	2	01
2	.75	.625	.101	3	101
3	.875	.8125	.1101	4	1101
4	1.0	.9375	.1111	4	1111

Arithmetic Coding (20)

- **<u>Arithmetic vs Huffman Coding</u>**
  - Average codeword length of m symbols sequence:
    - Arithmetic Coding: $H(X) \leq I_A \leq H(X) + \frac{2}{m}$
    - Extended Huffman Coding: $H(X) \leq I_H \leq H(X) + \frac{1}{m}$
  - Extended Huffman coding requires a large codebook for m^n extended symbols while arithmetic coding does not.
  - Generally,
    - Small alphabet sets favour Huffman coding
    - Skewed distributions favour arithmetic coding
  - Arithmetic coding can adapt to input statistics easily.

Arithmetic Coding (21)

- **Arithmetic vs Huffman Coding (2)**
  - What is the entropy of the source in Slide 20?
  - What is the average codeword length using Huffman Coding?
  - What is the average codeword length using Arithmetic Coding?
  - What are the average codeword lengths if we increase the sequence size to 2 symbols?

Arithmetic Coding (22)

- **Applications of Arithmetic Coding**

TABLE 4.7 Compression using adaptive arithmetic coding of pixel values.

Image Name	Bits/Pixel	Total Size (bytes)	Compression Ratio (arithmetic)	Compression Ratio (Huffman)
Sena	6.52	53,431	1.23	1.16
Sensin	7.12	58,306	1.12	1.27
Earth	4.67	38,248	1.71	1.67
Omaha	6.84	56,061	1.17	1.14

TABLE 4.8 Compression using adaptive arithmetic coding of pixel differences.

Image Name	Bits/Pixel	Total Size (bytes)	Compression Ratio (arithmetic)	Compression Ratio (Huffman)
Sena	3.89	31,847	2.06	2.08
Sensin	4.56	37,387	1.75	1.73
Earth	3.92	32,137	2.04	2.04
Omaha	6.27	51,393	1.28	1.26