Assignment 14

Yehonatan Keypur

September 10, 2024

1 Question 1

1.1 Question 8.17

Given a four-symbol source $[a, b, c, d]$ with source probabilities $[0.1, 0.4, 0.3, 0.2]$, we will arithmetically encode the sequence $bbadc$.

Cumulative Probabilities First, we define the cumulative probabilities:

$$
\begin{aligned}
a &: [0.0, 0.1) \\
b &: [0.1, 0.5) \\
c &: [0.5, 0.8) \\
d &: [0.8, 1.0)
\end{aligned}
$$

Encoding Process We encode the sequence $bbadc$ one symbol at a time:

- Start with interval $[0.0, 1.0)$
- Encode 'b': New interval $[0.1, 0.5)$
- Encode 'b' again: New interval $[0.1 + (0.5 - 0.1) \cdot 0.1, 0.1 + (0.5 - 0.1) \cdot 0.5) = [0.14, 0.30)$
- Encode 'a': New interval $[0.14 + (0.30 - 0.14) \cdot 0.0, 0.14 + (0.30 - 0.14) \cdot 0.1) = [0.14, 0.156)$
- Encode 'd': New interval $[0.1528, 0.156)$
- Encode 'c': Final interval $[0.1528 + (0.156 - 0.1528) \cdot 0.5, 0.1528 + (0.156 - 0.1528) \cdot 0.8) = [0.1544, 0.15536)$

Final Representation Any number within the final interval $[0.1544, 0.15536)$ can represent the encoded sequence. We choose the lower bound: 0.1544.

Symbol	Probability
a	0.2
e	0.3
i	0.1
o	0.2
u	0.1
!	0.1

Table 1: Coding Model

1.2 Question 8.18

Given the encoded message 0.23355 and the following coding model:

We will decode the message using the arithmetic decoding process.

Cumulative Probability Ranges First, we establish the cumulative probability ranges for each symbol:

$$\begin{aligned} \text{a} &: [0.0, 0.2) \\ \text{e} &: [0.2, 0.5) \\ \text{i} &: [0.5, 0.6) \\ \text{o} &: [0.6, 0.8) \\ \text{u} &: [0.8, 0.9) \\ \text{!} &: [0.9, 1.0) \end{aligned}$$

Decoding Process We will decode the message step by step:

1. Start with the encoded value: 0.23355
2. First symbol: $0.23355 \in [0.2, 0.5)$, so it's 'e'
   - New interval: [0.2, 0.5)
   - Rescaled value: $\frac{0.23355-0.2}{0.5-0.2} = 0.11183$
3. Second symbol: $0.11183 \in [0.0, 0.2)$, so it's 'a'
   - New interval: [0.2, 0.26)
   - Rescaled value: $\frac{0.11183-0.0}{0.2-0.0} = 0.55915$
4. Third symbol: $0.55915 \in [0.5, 0.6)$, so it's 'i'
   - New interval: [0.25, 0.26)
   - Rescaled value: $\frac{0.55915-0.5}{0.6-0.5} = 0.5915$
5. Fourth symbol: $0.5915 \in [0.5, 0.6)$, so it's 'i'
   - New interval: [0.255, 0.256)
   - Rescaled value: $\frac{0.5915-0.5}{0.6-0.5} = 0.915$
6. Fifth symbol: $0.915 \in [0.9, 1.0)$, so it's '!'

Conclusion The decoding process stops here as we've reached the termination symbol '!'. Therefore, the decoded message is: "eaii!"

2 Question 2

2.1 Question 8.7

Theorem. *For a zero-memory source with q symbols, the maximum value of the entropy is $\log q$, which is achieved if and only if all source symbols are equiprobable.*

Proof. Let p_i be the probability of the i-th symbol, where $i = 1, 2, ..., q$.

The entropy of the source is given by:

$$H(z) = -\sum_{i=1}^{q} p_i \log p_i$$

We want to prove that $H(z) \leq \log q$, with equality if and only if all $p_i = \frac{1}{q}$.

Consider the quantity $\log q - H(z)$:

$$\begin{aligned}
\log q - H(z) &= \log q + \sum_{i=1}^{q} p_i \log p_i \\
&= \sum_{i=1}^{q} p_i \log q + \sum_{i=1}^{q} p_i \log p_i \\
&= \sum_{i=1}^{q} p_i (\log q + \log p_i) \\
&= \sum_{i=1}^{q} p_i \log \frac{qp_i}{1}
\end{aligned}$$

Now, we use the inequality $\ln x \leq x - 1$ (with equality if and only if $x = 1$). Applying this to $\frac{1}{qp_i}$, we get:

$$\ln \frac{1}{qp_i} \leq \frac{1}{qp_i} - 1$$

Multiplying both sides by $-qp_i$ (which is negative), we get:

$$-qp_i \ln \frac{1}{qp_i} \geq qp_i - qp_i = 0$$

Dividing by $\ln 2$ (to convert to $\log_2$) and rearranging:

$$p_i \log \frac{qp_i}{1} \geq 0$$

Summing over all i:

$$\sum_{i=1}^{q} p_i \log \frac{qp_i}{1} \geq 0$$

But this sum is exactly $\log q - H(z)$, which we derived earlier. Therefore:

$$\log q - H(z) \geq 0$$
$$H(z) \leq \log q$$

Equality holds if and only if $\frac{qp_i}{1} = 1$ for all i, which means $p_i = \frac{1}{q}$ for all i.

Thus, we have proved that the maximum value of the entropy is $\log q$, and this is achieved if and only if all source symbols are equiprobable with $p_i = \frac{1}{q}$ for all i.

$\square$

2.2 Question 8.8

2.2.1 Part (a) - Huffman Codes for a Three-Symbol Source

To determine the number of unique Huffman codes for a three-symbol source:

For a three-symbol source, there is only one possible structure for a Huffman tree: two symbols as leaves at one level, and the third symbol as a leaf at the next level.

The number of unique codes depends on how we can arrange these three symbols in this structure:

- 3 choices for the symbol at the deeper level
- 2 choices for which side (left or right) to place this deeper symbol

Therefore, the total number of unique Huffman codes is:

$$3 \times 2 = 6$$

2.2.2 Part (b)

Let's construct all these unique Huffman codes. We'll denote our three symbols as A, B, and C. We'll use 0 for left branches and 1 for right branches.

1. A deeper, on left:

```
A: 00
B: 01
C: 1
```

2. A deeper, on right:

```
A: 10
B: 0
C: 11
```

3. B deeper, on left:

```
A: 01
B: 00
C: 1
```

4. B deeper, on right:

```
A: 0
B: 10
C: 11
```

5. C deeper, on left:

```
A: 01
B: 1
C: 00
```

6. C deeper, on right:

```
A: 0
B: 1
C: 10
```

These six codes represent all possible unique Huffman codes for a three-symbol source. Each code ensures that no codeword is a prefix of another, which is a key property of Huffman codes. The codes are uniquely decodable and satisfy the requirements for Huffman coding of a three-symbol source.

Conclusion We have shown that there are 6 unique Huffman codes for a three-symbol source and have constructed all of them. These codes cover all possibilities of which symbol is at the deeper level (A, B, or C) and whether the deeper symbol is on the left (0) or right (1) branch.

3 Question 8.19

Consider the 7-bit ASCII string "aaaaaaaaaaa". We will encode this string using the LZW (Lempel-Ziv-Welch) compression algorithm.

Encoding Process Initialize the dictionary with 128 entries (0-127) for all possible 7-bit ASCII characters. The ASCII code for 'a' is 97.

Step	Input	Output	Dictionary Addition
1	a	97	128: aa
2	aa	128	129: aaa
3	aaa	129	130: aaaa
4	aaaa	130	131: aaaaa
5	a	97	(end of string)

Table 2: LZW Encoding Steps

Result The final LZW-encoded output for the string "aaaaaaaaaaa" is:

$$[97, 128, 129, 130, 97]$$

4 Question 8.5

Given an 8-bit image of dimensions 1024×1024 with an entropy of 5.3 bits per pixel, we analyze its compression potential using Huffman coding.

Eq. (8-7) $\tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k) \log_2 p_r(r_k)$

4.1 Part (a) - Maximum Expected Compression

The maximum theoretical compression is determined by the ratio of the original bits per pixel to the entropy of the image.

$$\begin{aligned}
\text{Original bits per pixel} &= 8 \text{ (for an 8-bit image)} \\
\text{Entropy} &= 5.3 \text{ bits/pixel} \\
\text{Maximum compression ratio} &= \frac{\text{Original bits}}{\text{Entropy bits}} \\
&= \frac{8}{5.3} \\
&\approx 1.509
\end{aligned}$$

This means the image can theoretically be compressed to about 66.2% (1/1.509) of its original size, or a compression ratio of approximately 1.509:1.

4.2 Part (b) - Will It Be Obtained?

The maximum theoretical compression is unlikely to be achieved in practice with Huffman coding for several reasons:

1. Huffman coding produces integer-length codes, while entropy calculations allow for fractional bits, introducing inefficiency.
2. Huffman coding is optimal for symbol probabilities that are negative powers of 2 (1/2, 1/4, 1/8, etc.), which is rarely the case in real images.
3. There's overhead in storing the Huffman table itself, not accounted for in the entropy calculation.
4. The actual pixel value distribution may not be ideal for Huffman coding efficiency.
5. Practical implementations often use techniques like byte-alignment, which can slightly reduce compression efficiency.

4.3 Part (c) - Achieving Greater Lossless Compression

To achieve greater lossless compression, several advanced techniques can be employed:

- **Advanced Entropy Coding:**
  - Arithmetic coding: Can approach the entropy limit more closely than Huffman coding.

  - Context-adaptive binary arithmetic coding (CABAC): Adapts to changing statistics in the data.

- **Exploiting Spatial Redundancy:**
  - Predictive coding: e.g., Differential Pulse Code Modulation (DPCM).
  - Transform coding: e.g., Discrete Cosine Transform (DCT) or wavelet transforms.
- **Preprocessing:**
  - Pixel reordering to increase local correlation.
  - Run-length encoding for areas with low variation.
- **Context Modeling:** Adapt coding based on surrounding pixels or previously encoded data.
- **Combining Multiple Techniques:** Modern algorithms often use a combination of methods.

These techniques can potentially achieve compression ratios beyond what simple Huffman coding provides, while maintaining lossless compression. The effectiveness of each method depends on the specific characteristics of the image being compressed.

—

Thank you for reviewing this assignment.