Digital Image Processing

Chapter 08
Image Compression

Dec. 30, 2010

Instructor: Lih-Jen Kau (高立人)
Department of Electronic Engineering
National Taipei University of Technology

Huffman Coding
Huffman Code

- Huffman Code: Variable Length Code
- Theorem:
  
  X, discrete random variable with pmf p(X).
  Let C be any prefix code for X and \( C_{Huff} \) be a Huffman code, then
  
  \[
  l(C_{Huff}) \leq l(C)
  \]
  
  and
  
  \[
  H(X) \leq l(C_{Huff}) \leq H(X) + 1
  \]

Entrophy

- The “first-order entropy” (熵) is defined by,

\[
H(S) = -\sum_x P(x) \log_2 P(x) = \sum_x P(x) \log_2 \frac{1}{P(x)}
\]

- Where \( x \) is the alphabet/symbol in the source \( S \).
- The unit of entropy : bits/sample
- A less likely outcome will bring more information
Example of Huffman Code

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_i$</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- **Step 1:**
  - 0.4 0.2 0.2 0.1 0.1

- **Step 2:**
  - 0.4 0.2 0.2 0.1 0.1

- **Step 3:**
  - 0.6 0.4 0.2 0.1 0.1

- **Step 4:**
  - 1.0 0.6 0.4 0.2 0.1 0.1
Example of Huffman Code

- Step 5:

\[ I(C) = 1 \times 0.4 + 2 \times 0.2 + 3 \times 0.2 + 4 \times 0.1 + 4 \times 0.1 \]
\[ = 2.2 \text{ bits/symbol} \]

\[ H(X) = 0.4 \log_2 \frac{1}{0.4} + 0.2 \log_2 \frac{1}{0.2} + 0.2 \log_2 \frac{1}{0.2} + \]
\[ 0.1 \log_2 \frac{1}{0.1} + 0.1 \log_2 \frac{1}{0.1} \]
\[ = 2.12 \text{ bits/symbol} \]

C: 0 10 110 1110 1111

Single-side Growing Huffman Tree

Example of Huffman Code

- Another solution of the same example.

- Step 1:

- Step 2:

Rearrange the order

\[ 0.2 \]
\[ 0.4 0.2 0.2 0.1 0.1 \]
\[ 0.4 0.2 0.2 0.2 \]
\[ 0.1 0.1 \]
Example of Huffman Code

- **Step 3:**

- **Step 4:**
  
  Rearrange the order

- **Step 5:**

  Rearrange the order again

- **Step 6:**
Example of Huffman Code

◆ Step 7: Rearrange the order again

◆ Step 8:

0.4   0.1   0.1 0.2   0.2
0.2
0.6
0.4   0.1   0.1 0.2   0.2
0.2
0.4
0.1
0.1
0.2
0.2
1.0

Lih-Jen Kau
Multimedia Communication Group
National Taipei Univ. of Technology
2010/12/23 11/35

Example of Huffman Code

◆ Step 9:

\[ I(C) = 2 \times 0.4 + 3 \times 0.1 + 3 \times 0.1 + 2 \times 0.2 + 2 \times 0.2 \]
\[ = 2.2 \text{ bits/symbol} \]

/(/C/) = 2*0.4 + 3*0.1 + 3*0.1 + 2*0.2 + 2*0.2
= 2.2 bits/symbol

Minimum Codeword Length Huffman Tree

C1: 00 010 011 10 11

Lih-Jen Kau
Multimedia Communication Group
National Taipei Univ. of Technology
2010/12/23 12/35
Example of Huffman Code

- **Note:**
  - Though the above codes have the same average bit rates, **C1** has smaller variance than **C**
  - Better to be stored in memory
  - Though the 4-bit codeword length seldom occurs, the storage should have the memory width enlarged in case this does take place. Therefore, **C** is not as good as **C1**.

A Short Summary

- **For minimum codeword length variance:**
  - Combine prob. with shorter length
  - Sort prob. in each step, large to small, from left to right
  - Move combined prob. as close to the left as possible, and combine prob. on the right. (已經combine過的必然造成較長的長度)
Remarks for Huffman Code

**Remark 1:** Percentage increasing

\[ H(X) \leq l(C_{Huff}) \leq H(X) + 1 \]

Percentage increasing \( = \frac{l(C_{Huff}) - H(X)}{H(X)} \)

**Eg.**

<table>
<thead>
<tr>
<th>( x )</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_i )</td>
<td>0.8</td>
<td>0.02</td>
<td>0.18</td>
</tr>
</tbody>
</table>

\[ H(X) = 0.816 \]

\[ l(C_{Huff}) = 1 * 0.8 + 2 * 0.18 + 2 * 0.02 = 1.2 \text{ bits/symbol} \]

\[ \text{P.I.} = \frac{l(C_{Huff}) - H(X)}{H(X)} = \frac{1.2 - 0.816}{0.816} = 47\% \]

Pretty high

Though 1.2 bit/symbol, still far apart from the entropy

---

Remarks for Huffman Code

**Remark 2:** Longest codeword length (Worst case)

- Alphabet size \( m \) => longest codeword length is \( m-1 \)
- Eg. Alphabet = 5 => longest codeword length = 4
Empirical PMF

- For Huffman trees
  - $X$: discrete random variable
  - Alphabet $H$: \{1, 2, ..., $m$\}
  - $N$: Length of the sequence to be coded
  - $N_k$: number of occurrence of symbol $k$

  Empirical pmf: $p_k = \frac{N_k}{N}$, $k = 1, 2, \ldots, m$

- Binary tree can be obtained from $N_k$ instead of $p_k$.

An example of using $N_k$

<table>
<thead>
<tr>
<th>$x$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_k$</td>
<td>40</td>
<td>20</td>
<td>20</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$p_k$</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

$N = 100$
Storage Requirement

Problem: How to construct the header such that the header size is optimized?

- The codeword length for the worst case is \( (m-1) \), where \( m \) is the number of symbols.
- Therefore, the space required for the storage of the code is \( mx(m-1) \).

Two-Pass Huffman Encoding

- Two-pass
  - First pass for \( p_k \)
  - Second pass for encoding
- Transmission of \( p_k \) or codewords are required
- \( p_k \) not updated for varying statistics.
Theorem: For a symbol source, S, with \( n \) symbols, the entropy of the source satisfies
\[
H(S) \leq \log n.
\]

Proof:
- Suppose the probabilities of the \( n \) symbols are \( p_1, p_2, \ldots, p_n \), respectively, and \( \sum_{i=1}^{n} p_i = 1 \).
- Define the entropy of the source to be \( H = -\sum_{i=1}^{n} p_i \log p_i \).

That is
\[
H = -\sum_{i=1}^{n} p_i \log p_i = -\sum_{i=1}^{n} p_i \log n = -\sum_{i=1}^{n} p_i \log p_i - \sum_{i=1}^{n} p_i \log n
\]
\[
\leq -\sum_{i=1}^{n} p_i \left( \frac{1}{p_i n} - 1 \right) = \sum_{i=1}^{n} \left( \frac{1}{i} - p_i \right) = \sum_{i=1}^{n} \frac{1}{i} - \sum_{i=1}^{n} p_i = 1 - \sum_{i=1}^{n} p_i = 0
\]

That is \( H - \log n \leq 0 \).
- Moreover, we see that a uniformly spread source results in a larger entropy.
Something About Log Function

\[ \log x \leq x - 1 \]

Kraft Inequality

- Given a symbol source \( s \) with alphabets \( a_1, a_2, \ldots, a_m \).
- Suppose \( s \) has a prefix code with codeword lengths \( l_1, l_2, \ldots, l_m \), respectively. Then the code satisfies the following Kraft Inequality:

\[
\sum_{i=1}^{m} 2^{-l_i} \leq 1
\]

- \textbf{Pf:} Let \( l_{\max} = \max_{i} l_i \)

\[
\sum_{i=1}^{m} 2^{l_{\max} - l_i} \leq 2^{l_{\max}}
\]

\[
2^{l_{\max}} \left[ \sum_{i=1}^{m} 2^{-l_i} \right] \leq 2^{l_{\max}} \cdot 1 \Rightarrow \sum_{i=1}^{m} 2^{-l_i} \leq 1
\]
Coding Length of a Prefix Code

Given a symbol source $S$ with alphabets $a_1, a_2, \ldots, a_m$. Suppose the prefix code is constructed for the source with codeword length $l_1, l_2, \ldots, l_m$, respectively. Then the average coding length $L$ satisfies the following inequality:

$$H(S) \leq L$$

**Pf:** Let

$$H(S) - L = \sum_{i=1}^{m} p_i \log \frac{1}{p_i} - \sum_{i=1}^{m} p_i \log 2^{l_i} = \sum_{i=1}^{m} p_i \log \frac{1}{p_i 2^{l_i}}$$

$$\leq \sum_{i=1}^{m} p_i \left( \frac{1}{p_i 2^{l_i}} - 1 \right) = \sum_{i=1}^{m} \frac{1}{2^{l_i}} - \sum_{i=1}^{m} p_i = \sum_{i=1}^{m} \frac{1}{2^{l_i}} - 1 \leq 0$$

Coding Length of a Prefix Code

For Shannon-Fano coding, the average coding length is bounded in the range:

$$H(S) \leq L < 1 + H(S)$$

For Shannon-Fano coding, the average coding length is bounded in the range $H(S) \leq L < 1 + H(S)$. 

\[ L = \sum_{i=1}^{m} p_i l_i < \sum_{i=1}^{m} p_i (1 + \log \frac{1}{p_i}) = \sum_{i=1}^{m} p_i + \sum_{i=1}^{m} p_i \log \frac{1}{p_i} = 1 + H(S) \]
Vector Quantization

\[ X = (x_1, x_2, \ldots, x_{16}) \]

\[ C_j = (C_{j1}, C_{j2}, \ldots, C_{j16}) \]

\[ d^2(X, C_j) = \min_j \sum_{a=1}^{16} (x_a - C_{ja})^2 \]

JPEG Still Image Coding Standard

- Please see Lecture-JPEG for details.
在MPEG中，先將視訊影像分成三類，分別為I、P和B影像。I影像用傳統的JPEG壓縮即可；P可利用前面的I影像，透過移動匹配(Block Matching)和補償(Compensation)來壓縮，夾在I和P之間的B影像之區塊就由I和P所匹配到的區塊內插而成。

- **I: Intra frame** (以真實的Frame直接進行壓縮編碼)
- **P: Predicted frame** (以先前編碼過的I frame或P frame進行移動估測編碼)
- **B: Bi-directional frame** (以前後之I frame或P frame進行編碼)

### Block Matching

在前一張參考影像中找到某一區塊使得這找到的區塊和目前區塊最匹配。通常是採用在前張影像中先訂出一個搜尋視窗，在這搜尋視窗內包含許多與目前區塊相同大小的正方形區塊。
目前區塊 $B_c$，算出利用初始移動向量所得的區塊位置 $B'_c$，然後 $B'_c$ 和 $B'_c$ 對應於參考影像中的區塊 $B'_c$ 計算兩者的 MAD:

$$MAD(B'_c, B'_c) = \sum_{x=1}^{N} \sum_{y=1}^{N} |B'_c(x, y) - B'_c(x, y)|$$

- Full-Search Method
- Logarithmic Search Method (每次搜尋都縮小尚未搜尋範圍的一半)
- Three-Step Search Method (Three-Step Search Algorithm)
**Fully Search**

- **Block size** = $M \times N$
- **Max search range** = $d_{max}$
- **Search area**
  $$= (M + 2d_{max})(N + 2d_{max})$$
- **Number of search points**
  $$= (2d_{max} + 1)^2$$

---

**Logarithmic Search**

- （對數搜尋）
Three-Step Search（三步搜尋）

\[ \sigma^n \text{ DENOTES A SEARCH POINT OF STEP } n \]