Run-Length Coding

- Lots of 0’s and not too many 1’s.
  - Fax of letters
  - Graphics
- Simple run-length code
  - Input: 000001000000010000000000100100010001001......
  - Symbols: 6 9 10 3 2 ...
  - Code the bits as a sequence of integers
  - Problem: How long should the integers be?

Golomb Code of Order m
Variable Length Code for Integers

- Let $n = qm + r$ where $0 \leq r < m$.
  - Divide $m$ into $n$ to get the quotient $q$ and remainder $r$.
- Code for $n$ has two parts:
  1. $q$ is coded in unary
  2. $r$ is coded as a fixed prefix code

Example: $m = 5$

$$
\begin{array}{c}
0 & 1 & 2 & 3 & 4 \\
00 & 01 & 10 & 11 & 01 \\
000 & 011 & 110 & 111 & 100 \\
000000 & 000001 & 000011 & 000110 & 001101 \\
\end{array}
$$

In this example we coded 17 bit in only 9 bits.

Choosing m

• Suppose that 0 has the probability $p$ and 1 has probability $1-p$.
• The probability of 0^n1 is $p^n(1-p)$. The Golomb code of order $m = \lceil -\frac{1}{\log_2 p} \rceil$ is optimal.
• Example: $p = 127/128$.

$$m = \lceil -\frac{1}{\log_2 (127/128)} \rceil = 89$$

Average Bit Rate for Golomb Code

Average Bit Rate = \frac{\text{Average output code length}}{\text{Average input code length}}

• $m = 4$ as an example. With $p$ as the probability of 0.

$$ABR = 5p + 3p(1-p) + 3p(1-p) + 3p(1-p) + 3p(1-p) + 2p(1-p) + 2p(1-p)$$

<table>
<thead>
<tr>
<th>output</th>
<th>input</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>011</td>
<td>p</td>
</tr>
<tr>
<td>000</td>
<td>0001</td>
<td>p^2</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td>p^2(1-p)</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>p(1-p)</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1-p</td>
</tr>
</tbody>
</table>

Comparison of GC with Entropy

Notes on Golomb codes

• Useful for binary compression when one symbol is much more likely than another.
  – binary images
  – fax documents
  – bit planes for wavelet image compression
• Need a parameter (the order)
  – training
  – adaptively learn the right parameter
• Variable-to-variable length code
• Last symbol needs to be a 1
  – coder always adds a 1
  – decoder always removes a 1

Tunstall Codes

• Variable-to-fixed length code
• Example

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>000</td>
</tr>
<tr>
<td>b</td>
<td>001</td>
</tr>
<tr>
<td>ca</td>
<td>010</td>
</tr>
<tr>
<td>cb</td>
<td>011</td>
</tr>
<tr>
<td>cca</td>
<td>100</td>
</tr>
<tr>
<td>cdb</td>
<td>101</td>
</tr>
<tr>
<td>cc</td>
<td>110</td>
</tr>
</tbody>
</table>

Tunstall code Properties

1. No input code is a prefix of another to assure unique encodability.
2. Minimize the number of bits per symbol.
Prefix Code Property

1. Consider string "cc", if it occurs at the end of data, it does not have a code.
2. Generally, if there are k internal nodes in the prefix tree then there is a need for k-1 fixed codes.

Use for unused code

Example

P(a) = 0.7, P(b) = 0.2, P(c) = 0.1
n = 3

Designing a Tunstall Code

Suppose there are m initial symbols.
Choose a target output length n where $2^n > m$.

1. Form a tree with a root and m children with edges labeled with the symbols.
2. If the number of leaves is $> 2^n - m$ then halt.*
3. Find the leaf with highest probability and expand it to have m children.** Go to 2.

Example

P(a) = 0.7, P(b) = 0.2, P(c) = 0.1
n = 3
Bit Rate of Tunstall

- The length of the output code divided by the average length of the input code.
- Let \( p_i \) be the probability of, and \( r_i \) the length of input code \( l \) \((1 \leq i \leq s)\) and let \( n \) be the length of the output code.

\[
\text{Average bit rate} = \frac{n}{\sum_{i=1}^{s} p_i r_i}
\]

Example

ABR = \(\frac{3}{3} (0.343 + 0.098 + 0.049) + 2 (0.14 + 0.07) + 0.2 + 0.1\] = 1.37 bits per symbol

Entropy = 1.16 bits per symbol

Notes on Tunstall Codes

- Variable-to-fixed length code
- Error resilient
  - A flipped bit will introduce just one error in the output
  - Huffman is not error resilient. A single bit flip can destroy the code.