Scaling

- Scaling:
  - By scaling we can keep L and R in a reasonable range of values so that \( W = R - L \) does not underflow.
  - The code can be produced progressively, not at the end.
  - Complicates decoding some.

Scaling during Encoding

- Lower half:
  - If \( [L, R) \) is contained in \([0, .5)\) then
    - \( L = 2L \), \( R = 2R \)
    - Output 0, followed by \( C \)'s
    - \( C = 0 \).

- Upper half:
  - If \( [L, R) \) is contained in \([.5, 1)\) then
    - \( L = 2L - 1 \), \( R = 2R - 1 \)
    - Output 1, followed by \( C \)'s
    - \( C = 0 \).

- Middle half:
  - If \( [L, R) \) is contained in \([.25, .75)\) then
    - \( L = 2L - .5 \), \( R = 2R - .5 \)
    - \( C = C + 1 \).

Example

- baa

\[ L = \frac{1}{3} \quad R = \frac{3}{3} \]
\[ C = 0 \]

Scale middle half

- baa

\[ L = \frac{3}{9} \quad R = \frac{5}{9} \]
\[ L = \frac{3}{18} \quad R = \frac{11}{18} \]
**Example**

- baa

\[
\begin{align*}
C &= 1 \\
L &= 3/18 R &= 11/18 \\
L &= 9/54 R &= 17/54 \\
1/3 &= &2/3 \\
\end{align*}
\]

Scale lower half

- baa 01

\[
\begin{align*}
C &= 0 \\
L &= 9/54 R &= 17/54 \\
L &= 18/54 R &= 34/54 \\
1/3 &= &2/3 \\
\end{align*}
\]

**Example**

- baa

\[
\begin{align*}
C &= 1 \\
L &= 18/54 R &= 34/54 \\
1/3 &= &2/3 \\
\end{align*}
\]

In end \(L < 1/2 < R\), choose tag to be \(1/2\)

**Exercise**

Model: a: 1/4; b: 3/4

Encode: bba

**Decoding**

- The decoder behaves just like the encoder except that \(C\) does not need to be maintained.
- Instead, the input stream is consumed during when scaling.

**Scaling during Decoding**

- **Lower half**
  - If \([L,R)\) is contained in \([0,.5)\) then
  - \(L := 2L\), \(R := 2R\)
  - consume 0 from the encoded stream

- **Upper half**
  - If \([L,R)\) is contained in \([.5, 1)\) then
  - \(L := 2L - 1\), \(R := 2R - 1\)
  - consume 1 from the encoded stream

- **Middle half**
  - If \([L,R)\) is contained in \([.25, .75)\) then
  - \(L := 2L - .5\), \(R := 2R -.5\)
  - Replace 01 with 0 on stream
  - Replace 10 with 1 on stream
Scaling Math for the Tag

• **Lower Half**
  \[-.0b_1b_2\ldots \times 10 = .b_1b_2\]

• **Upper Half**
  \[-.1b_1b_2\ldots \times 10 - 1 = .b_1b_2\]

• **Middle Half**
  \[-.01b_1b_2\ldots \times 10 - .1 = .0b_1b_2\]
  \[-.10b_1b_2\ldots \times 10 - .1 = .1b_1b_2\]

Exercise

Model: \(a: 1/4; b: 3/4\)
Decode: 001 to 3 symbols

Integer Implementation

• \(m\) bit integers
  - Represent 0 with 000...0 (\(m\) times)
  - Represent 1 with 111...1 (\(m\) times)

• Probabilities represented by frequencies
  - \(n_i\) is the number of times that symbol \(a\) occurs
  - \(C_i = n_1 + n_2 + \ldots + n_i\)
  - \(N = n_1 + n_2 + \ldots + n_m\)

\[
L' = \frac{W + C_i}{N} \\
L = \frac{W + C_{i-1}}{N}
\]

Coding the \(i\)-th symbol using integer calculations.
Must use scaling!

Example with Scaling

\[
\begin{array}{ccc}
\text{acc} & \text{prev} & \text{next} \\
0 & 0.4 & 1 \text{ or } 0.2 \text{ or } 0.25 \\
1 & 2/3 & 0.1 \text{ or } 0.8 \text{ or } 0.25 \\
0 & 2/15 & 17/30 \\
\end{array}
\]

Code = 0101

Arithmetic Coding with Context

• Maintain the probabilities for each context.
• For the first symbol use the equal probability model
• For each successive symbol use the model for the previous symbol.
Adaptation

• Simple solution – Equally Probable Model.
  – Initially all symbols have frequency 1.
  – After symbol $x$ is coded, increment its frequency
    by 1
  – Use the new model for coding the next symbol
• Example in alphabet $a, b, c, d$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
</tr>
</tbody>
</table>

After aabaac is encoded

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>5/10</td>
</tr>
<tr>
<td>$b$</td>
<td>2/10</td>
</tr>
<tr>
<td>$c$</td>
<td>2/10</td>
</tr>
<tr>
<td>$d$</td>
<td>1/10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0.5</td>
</tr>
<tr>
<td>$b$</td>
<td>0.2</td>
</tr>
<tr>
<td>$c$</td>
<td>0.2</td>
</tr>
<tr>
<td>$d$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Zero Frequency Problem

• How do we weight symbols that have not occurred yet.
  – Equal weights? Not so good with many symbols
  – Escape symbol, but what should its weight be?
  – When a new symbol is encountered send the `<esc>`, followed
    by the symbol in the equally probable model. (Both encoded
    arithmetically.)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
</tr>
<tr>
<td><code>&lt;esc&gt;</code></td>
<td>1</td>
</tr>
</tbody>
</table>

After aabaac is encoded

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>4/7</td>
</tr>
<tr>
<td>$b$</td>
<td>1/7</td>
</tr>
<tr>
<td>$c$</td>
<td>1/7</td>
</tr>
<tr>
<td>$d$</td>
<td>0</td>
</tr>
<tr>
<td><code>&lt;esc&gt;</code></td>
<td>1/7</td>
</tr>
</tbody>
</table>

PPM

• Prediction with Partial Matching
  – Cleary and Witten (1984)
• State of the art arithmetic coder
  – Arbitrary order context
  – The context chosen is one that does a good
    prediction given the past
  – Adaptive
• Example
  – Context “the” does not predict the next symbol “a”
    well. Move to the context “he” which does.

Arithmetic vs. Huffman

• Both compress very well. For $m$ symbol grouping.
  – Huffman is within $1/m$ of entropy.
  – Arithmetic is within $2/m$ of entropy.
• Context
  – Huffman needs a tree for every context.
  – Arithmetic needs a small table of frequencies for every
    context.
• Adaptation
  – Huffman has an elaborate adaptive algorithm
  – Arithmetic has a simple adaptive mechanism.
• Bottom Line – Arithmetic is more flexible than
  Huffman.