Lecture 9: Practical Arithmetic Coding

Thinh Nguyen
Oregon State University
Issues with Arithmetic Coding

- The intervals are getting smaller as the sequence of symbols is getting longer.
- Arithmetics (computations) on very small numbers results in underflow!
- Need to rescale at every step!
Representation of Real Number in Binary

- Always scale the interval to unit size, but $x$ must be changed as part of the scaling.
Binary Conversion with Scaling

\[
y := x; \ i := 0\\
\text{while } y > 0 * \\
\quad \ i := i + 1; \\
\quad \text{if } y < 1/2 \text{ then } b_i := 0; \ y := 2y; \\
\quad \text{if } y \geq 1/2 \text{ then } b_i := 1; \ y := 2y - 1; \\
\text{end\{while\}} \\
\text{b}_j := 0 \text{ for all } j \geq i + 1
\]

* Invariant: \( x = .b_1 b_2 \ldots b_i + y/2^i \)
Proof of Invariant

• Initially $x = 0 + y/2^0$

• Assume $x = .b_1b_2 \ldots b_i + y/2^i$
  
  – Case 1. $y < 1/2$. $b_{i+1} = 0$ and $y' = 2y$
    
    $\begin{align*}
    .b_1b_2 \ldots b_i b_{i+1} + y'/2^{i+1} &= .b_1b_2 \ldots b_i 0 + 2y/2^{i+1} \\
    &= .b_1b_2 \ldots b_i + y/2^i \\
    &= x
    \end{align*}$

  – Case 2. $y \geq 1/2$. $b_{i+1} = 1$ and $y' = 2y - 1$
    
    $\begin{align*}
    .b_1b_2 \ldots b_i b_{i+1} + y'/2^{i+1} &= .b_1b_2 \ldots b_i 1 + (2y-1)/2^{i+1} \\
    &= .b_1b_2 \ldots b_i + 1/2^{i+1} + 2y/2^{i+1} - 1/2^{i+1} \\
    &= .b_1b_2 \ldots b_i + y/2^i \\
    &= x
    \end{align*}$
Exercise

\[ x = \frac{1}{3} \]

<table>
<thead>
<tr>
<th>y</th>
<th>i</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2/3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1/3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2/3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ x = \frac{17}{27} \]

<table>
<thead>
<tr>
<th>y</th>
<th>i</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>17/27</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

... ... ... ...
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 Algorithm for Arithmetic Coding

<table>
<thead>
<tr>
<th>Lower half</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>While</strong> ([L, R)) is contained in ([0, .5)) then</td>
</tr>
<tr>
<td>(L := 2L; R := 2R)</td>
</tr>
<tr>
<td>output 0, followed by C 1’s</td>
</tr>
<tr>
<td>(C := 0).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Upper half</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>While</strong> ([L, R)) is contained in ([.5, 1)) then</td>
</tr>
<tr>
<td>(L := 2L - 1, R := 2R - 1)</td>
</tr>
<tr>
<td>output 1, followed by C 0’s</td>
</tr>
<tr>
<td>(C := 0).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Middle Half</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>While</strong> ([L, R)) is contained in ([.25, .75)) then</td>
</tr>
<tr>
<td>(L := 2L -.5, R := 2R -.5)</td>
</tr>
<tr>
<td>(C := C + 1).</td>
</tr>
</tbody>
</table>
Example

- $\text{baa}$

$C = 0$

$L = 1/3 \quad R = 3/3$

$0 \quad a \quad 1/3$

$2/3 \quad b \quad 1$
Example

- $b_{aa}$

\[ C = 0 \]

\[ L = \frac{1}{3} \quad R = \frac{3}{3} \]
\[ L = \frac{3}{9} \quad R = \frac{5}{9} \]

Scale middle half
Example

- $baa$

$$C = 1$$

$L = \frac{3}{9}$ $R = \frac{5}{9}$
$L = \frac{3}{18}$ $R = \frac{11}{18}$
Example

- baa

C = 1

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

Scale lower half
Example

- baa 01

\[
\begin{align*}
C &= 0 \\
L &= \frac{9}{54} \quad R = \frac{17}{54} \\
L &= \frac{18}{54} \quad R = \frac{34}{54}
\end{align*}
\]
Example

- baa 011

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

C = 0

\[ L = \frac{9}{54} \quad R = \frac{17}{54} \]
\[ L = \frac{18}{54} \quad R = \frac{34}{54} \]
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 $a_i$ occurs
  - $C_i = n_1 + n_2 + ... + n_{i-1}$
  - $N = n_1 + n_2 + ... + n_m$

\[
W := R - L + 1
\]
\[
L' := L + \left\lfloor \frac{W \cdot C_i}{N} \right\rfloor
\]

Coding the i-th symbol using integer calculations.
Must use scaling!
\[
R := L + \left\lfloor \frac{W \cdot C_{i+1}}{N} \right\rfloor - 1
\]
\[
L := L'
\]
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.
Arithmetic Coding with Context

- 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 $a,b,c,d$.

```
  a  a  b  a  a  c
a  1  2  3  3  4  5  5
b  1  1  1  2  2  2  2
  1  1  1  1  1  1  2
  1  1  1  1  1  1  1
  
After aabaac is encoded
The probability model is
a 5/10   b 2/10
  c 2/10   d 1/10
```
Arithmetic Coding with Context

- 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.