Lab Session 4

May 18, 2009

Outline

- Review
- Manual Exercises
  - Comparing coding performance of different codes: Shannon code, Shannon-Fano code, Huffman code (and Tunstall code *)
- MATLAB Exercises
  - Working with Huffman codec @ MATLAB
  - Compressing an image with the Huffman encoder
  - Writing your own Huffman codec
  - Manipulating the colormap to add pseudo-color to UV channels
  - Testing Y4M video support

Image and video encoding: A big picture
The ingredients of entropy coding

- A random source \((X, P)\)
- A statistical model \((X, P')\) as an estimation of the random source
- An algorithm to optimize the coding performance (i.e., to minimize the average codeword length)
- At least one designer …

FLC, VLC and V2FLC

- FLC = Fixed-length coding/code(s)/codeword(s)
  - Each symbol \(x_i\) emitted from a random source \((X, P)\) is encoded as an \(n\)-bit codeword, where \(|X| \leq 2^n\).
- VLC = Variable-length coding/code(s)/codeword(s)
  - Each symbol \(x_i\) emitted from a random source \((X, P)\) is encoded as an \(n\)-bit codeword.
  - FLC can be considered as a special case of VLC, where \(n_1 = \cdots = n_{|X|}\).
- V2FLC = Variable-to-fixed length coding/code(s)/codeword(s)
  - A symbol or a string of symbols is encoded as an \(n\)-bit codeword.
  - V2FLC can also be considered as a special case of VLC.

Static coding vs. Dynamic/Adaptive coding

- Static coding = The statistical model \(P'\) is static, i.e., it does not change over time.
- Dynamic/Adaptive coding = The statistical model \(P'\) is dynamically updated, i.e., it adapts itself to the context (i.e., changes over time).
  - Dynamic/Adaptive coding \(\subset\) Context-based coding
- Hybrid coding = Static + Dynamic coding
  - A codebook is maintained at the encoder side, and the encoder dynamically chooses a code for a number of symbols and informs the decoder about the choice.

A coding ZOO

- Shannon coding
- Shannon-Fano coding
- Huffman coding
- Arithmetic coding (Range coding)
  - Shannon-Fano-Elisa coding
- Universal coding
  - Exp-Golomb coding (H.264/MPEG-4 AVC, Dirac)
  - Elias coding family
  - Levenshtein coding …
- Non-universal coding
  - Truncated binary coding, unary coding, …
  - Golomb coding \(\supset\) Rice coding
- Tunstall coding \(\subset\) V2FLC
- …

Shannon-Fano Code: An example

- $X=\{A,B,C,D,E\}$, $P=\{0.35,0.2,0.19,0.13,0.13\}$, $Y=\{0,1\}$

A Possible Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
</tr>
<tr>
<td>B</td>
<td>01</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>110</td>
</tr>
<tr>
<td>E</td>
<td>111</td>
</tr>
</tbody>
</table>

Tunstall coding: An example *

- $X=\{A,B,C\}$, $P=\{0.7,0.2,0.1\}$, $Y=\{0,1\}$, $n=3$

A Possible Code

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>000</td>
</tr>
<tr>
<td>AAB</td>
<td>001</td>
</tr>
<tr>
<td>AAC</td>
<td>010</td>
</tr>
<tr>
<td>AB</td>
<td>011</td>
</tr>
<tr>
<td>AC</td>
<td>100</td>
</tr>
<tr>
<td>B</td>
<td>101</td>
</tr>
<tr>
<td>C</td>
<td>110</td>
</tr>
</tbody>
</table>

Universal coding (code)

- A code is called **universal** if $L \leq C_1(H+C_2)$ for all possible values of $H$, where $C_1, C_2 \geq 1$.
  - You may see a different definition somewhere, but the basic idea remains the same – a universal code works like an optimal code, except there is a bound defined by a constant $C_1$.
- A universal code is called **asymptotically optimal** if $C_1 \to 1$ when $H \to \infty$.

Coding positive/non-negative integers

- **Naive binary coding**
  - $|X|=2^k$: $k$-bit binary representation of an integer.
- **Truncated binary coding**
  - $|X|=2^k+b$: $X=\{0,1,\ldots,2^k+b-1\}$
  - $f(x) = 0 \cdots 0 1 \cdots 1$ or $1 \cdots 0 \cdots 0$
- **Unary code (Stone-age binary coding)**
  - $|X|=\infty$: $X=\mathbb{Z}^+ = \{1,2,\ldots\}$
  - $f(x) = 0 \cdots 0 1 \cdots 1$ or $1 \cdots 0 \cdots 0$
Huffman’s rules of making optimal codes

- **Source statistics**: $P = P_0 = \{p_1, \ldots, p_m\}$, where $p_1 \geq \ldots \geq p_m > 0$.
- **Rule 1**: $L_1 \leq \ldots \leq L_m = L_m$.
- **Rule 2**: If $L_1 \leq \ldots \leq L_m$, $L_m$, and $L_m$ differ from each other only for the last bit, i.e., $f(x_m) = b0$ and $f(x_m) = b1$, where $b$ is a sequence of $L_m-1$ bits.
- **Rule 3**: Each possible bit sequence of length $L_m-1$ must be either a codeword or the prefix of some codewords.


Huffman code: An example

- $X = \{1, 2, 3, 4, 5\}$, $P = [0.4, 0.2, 0.2, 0.1, 0.1]$. A Possible Code

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

Huffman code: An optimal code

- Relation between Huffman code and Shannon code:
  $H \leq L_{\text{Huffman}} \leq L_{\text{Huffman-Fano}} \leq L_{\text{Shannon}} < H+1$
- A stronger result (Gallager, 1978)
  * When $p_{\text{max}} \geq 0.5$, $L_{\text{Huffman}} \leq H + p_{\text{max}} < H+1$
  * When $p_{\text{max}} < 0.5$, $L_{\text{Huffman}} \leq H + p_{\text{max}} - \log_2(2(\log_2(e)) = H + p_{\text{max}} + 0.086 < H+0.586$
- Huffman’s rules of optimal codes imply that Huffman code is optimal.
  * When each $p_i$ is a negative power of 2, Huffman code reaches the entropy.

Huffman code: Small $X$ problem

- **Problem**
  - When $|X|$ is small, the coding performance is less obvious.
  - As a special case, when $|X|=2$, Huffman coding cannot compress the data at all – no matter what the probability is, each symbol has to be encoded as a single bit.
- **Solutions**
  - Solution 1: Work on $X^*$ rather than $X$.
  - Solution 2: Dual tree coding = Huffman coding + Tunstall coding
Huffman code: Variance problem

- **Problem**
  - There are multiple choices of two smallest probabilities, if more than two nodes have the same probability during any step of the coding process.
  - Huffman codes with a larger variance may cause trouble for data transmissions via a CBR (constant bit rate) channel – a larger buffer is needed.

- **Solution**
  - Shorter subtrees first. (A single node’s height is 0.)

Modified Huffman code

- **Problem**
  - If $|X|$ is too large, the construction of the Huffman tree will be too long and the memory used for the tree will be too demanding.

- **Solution**
  - Divide $X$ into two set $X_1=\{s_i | p(s_i)>2^{-v}\}$, $X_2=\{s_i | p(s_i)\leq 2^{-v}\}$.
  - Perform Huffman coding for the new set $X' = X_1 \cup \{X_2\}$.
  - Append $f(X_2)$ as the prefix of naive binary representation of all symbols in $X_2$.

Comparing different entropy codes

- $X=\{A,B,C,D,E\}$, $P=[0.38 \ 0.18 \ 0.15 \ 0.15 \ 0.14]$, $Y=\{0,1\}$
- Construct the following codes for this source: Shannon code, Shannon-Fano code, Huffman code.
- Compare the coding performance of the above three codes in term of average codeword length.
- **Extra exercise (*)**: Construct a Tunstall code ($n=3$) and see how the performance is.
Huffman code for smaller X

- $X = \{A, B\}, \ p = [0.7, 0.3], \ Y = \{0, 1\}$
- Construct a Huffman code for the extended source $(X^2, p^2)$ and $(X^3, p^3)$.
- Compare the performance of them.

MATLAB Exercises

Working with Huffman codec

- MATLAB Communications Toolbox includes an implementation of Huffman codec: huffmandict, huffmanenco, huffmandeco

- Run the function huffmandict to verify the Huffman codes you got in the manual exercises.
- Given $X = \{A, B, C, D, E\}, \ p = [0.38 \ 0.18 \ 0.15 \ 0.15 \ 0.14]$, generate a random message of size 100 which satisfies the probability distribution.
- Encode the message with huffmanenco, calculate the compression rate (without counting the memory for storing the Huffman code) and compare it with the average codeword length of the Huffman code.
- Decode the encoded message with huffmandeco, and check if the decoding result is correct (using isequal function).

- Given $X = \{A, B, C, D, E\}, \ p = [0.38 \ 0.18 \ 0.15 \ 0.15 \ 0.14]$, generate a random message of size 100 which does NOT satisfy the probability distribution.
- Encode the message with huffmanenco, calculate the compression rate and compare it with the average codeword length of the Huffman code.
- Estimate the real probability distribution from the message itself and redo the above experiments.
Compressing images with Huffman code

- **Step 1**: read a RGB image from a file.
- **Step 2**: transform the RGB image to 4:2:0 YUV format.
- **Step 3**: estimate the statistics of YUV channels.
- **Step 4**: construct Huffman codes for YUV channels, respectively, and compress the whole YUV image (as a 1-D signal).
- **Step 5**: calculate the compression ratios of different channels and of the image as a whole (count the storage of the Huffman codes, too).

**Tip**: use `reshape` function to transform each image plane to a 1-D array for encoding and transform back for decoding.

My implementation of entropy codecs


- **Some auxiliary functions**: EntropyCodeShow, EntropyCodeTree, AverageCodewordLength, EntropyCodeTreeDepth, Z2Zplus, Zplus2Z.

- **Interesting features**
  - Compatibility with `huffmandict` function.
  - More entropy codes are supported.
  - Speed optimization – much faster than MATLAB’s implementation.
  - Three variance modes for Huffman code: ‘min’, ‘max’ and ‘rand’.
  - Saving/Loading compressed data into/from binary files.
  - Display of the binary tree representation and statistics of entropy code.

Exemplar code

```matlab
% x=imread('Images/lena_gray.bmp'); % Read a 512X512 grayscale image.
x=[0:255]; P=imhist(x); % Set statistics of the random source.
code=HuffmanCode(x,P); % Construct a Huffman code.
x_e=EntropyEncoder(x,code); % Encode x to get a 512X512 cell array.
x_d=EntropyDecoder_Cell(x_e,code); % Decode the encoded cell array.
imshow(x_d,[0 255]); % Show the decoded image.
```

```matlab
x_e2=[x_e{:}]; % Generate a 1-D bitstream from the encoded cell array.
x_d2=EntropyDecoder_BS(x_e2,code); % Decode the bitstream.
imshow(reshape(x_d2,size(x)), [0 255]); % Show the decoded image.
```

```matlab
% code=ShannonCode(x,P,1); % Generate a Shannon code
% code=ShannonFanoCode(x,P,1); % Generate a Shannon-Fano code
% code=PrefixFreeCode(x,L,1); % Generate a prefix code according to the codeword lengths (L)
% x_e=ExpGolombCodec(x,7); % Exp-Golomb encoder
% x_d=ExpGolombCodec(x_e,7,’Decoder’); % Exp-Golomb decoder
% x_e=GolombCodec(x,100); % Golomb encoder
% x_d=GolombCodec(x_e,100,’Decoder’); % Golomb decoder
```

...
Your implementation of Huffman codec...

- Do NOT refer to existing MATLAB implementations.
- You have to depend on cell array a lot to store elements of different sizes.
- You may assume $X = \{0, \ldots, m-1\}$ to simplify your implementation.
- You may not consider removing zero probabilities from the distribution.
- You may not consider the variance problem, which helps simplify your implementation.
- You may not consider optimizing your code to get a faster encoding/decoding process.

Some more tips

- Exemplar MATLAB codes about generating a Huffman table.
  - $P = [0.49, 0.21, 0.21, 0.09]$;
    - $HuffmanTable = cell(1, 4)$; % HuffmanTable = {[], [], [], []};
    - $HuffmanTree = [1, 2, 3, 4]$;
    - $HuffmanTable(3) = 0$; $HuffmanTable(3) = [1, HuffmanTable(3)]$;
    - $P = [0.49, 0.21, 0.3]$;
      - $HuffmanTree(3) = [3, 4]$;
      - $HuffmanTable(3) = [1, HuffmanTable(3)]$;
    - $P = [0.49, 0.51]$;
      - $HuffmanTree(2) = [2, 3, 4]$;
      - $HuffmanTable(2) = [1, HuffmanTable(2)]$;
    - $P = [1]$;
      - $HuffmanTree(1) = [1, 2, 3, 4]$.