Algorithms in the Real World

Data Compression 4
Compression Outline

Introduction: Lossy vs. Lossless, Benchmarks, ...
Information Theory: Entropy, etc.
Probability Coding: Huffman + Arithmetic Coding
Applications of Probability Coding: PPM + others
Lempel-Ziv Algorithms: LZ77, gzip, compress, ...
Other Lossless Algorithms: Burrows-Wheeler
Lossy algorithms for images: JPEG, MPEG, ...
  - Scalar and vector quantization
  - JPEG and MPEG
Compressing graphs and meshes: BBK
Scalar Quantization

Quantize regions of values into a single value:

Quantization is lossy
Can be used, e.g., to reduce # of bits for a pixel
Vector Quantization: Example

Input vectors are (Height, Weight) pairs. Map each input vector to a representative “codevector”. Codevectors are stored in a codebook.
Vector Quantization

generate input vector

find closest code-vector

codebook

vector

index of code-vector

compress index

encode

generate output

decompress index

index

decode

codebook

output

codevector
Vector Quantization

What do we use as vectors?

• Color (Red, Green, Blue)
  - Can be used, for example to reduce 24bits/pixel to 12bits/pixel
  - Used in some terminals to reduce data rate from the CPU (colormaps)

• \(k\) consecutive samples in audio

• Block of \(k \times k\) pixels in an image

How do we decide on a codebook

• Typically done with clustering
Linear Transform Coding

Want to encode values over a region of time or space
- typically used for images or audio
- represented as a vector \([x_1, x_2, \ldots]\)

Select a set of linear basis functions \(\varphi_i\) that span the space
- sin, cos, spherical harmonics, wavelets, ...
- defined at discrete points
Linear Transform Coding

Coefficients: \( \Theta_i = \sum_j x_j \phi_i(j) = \sum_j x_j a_{ij} \)

\( \Theta_i \) = \( i^{th} \) resulting coefficient

\( x_j \) = \( j^{th} \) input value

\( a_{ij} \) = \( ij^{th} \) transform coefficient = \( \phi_i(j) \)

\( \Theta = Ax \)

In matrix notation:

\( x = A^{-1} \Theta \)

Where \( A \) is an \( n \times n \) matrix, and each row defines a basis function
Example: Cosine Transform

\[ \phi_0(j), \phi_1(j), \phi_2(j), \ldots \]

\[ x_j \]

\[ j \]

\[ \Theta_i = \sum_j x_j \phi_i(j) \]

small values?
Other Transforms

Polynomial:

1

x

x^2

Wavelet (Haar):
How to Pick a Transform

**Goals:**
- Decorrelate (remove repeated patterns in data)
- Low coefficients for many terms
- Some terms affect perception more than others

Why is using a Cosine or Fourier transform across a whole image bad?
-- If there is no periodicity in the image, large coefficients for high-frequency terms

How might we fix this?
-- use basis functions that are not as smoothly periodic
Usefulness of Transform

Typically transforms $A$ are \textbf{orthonormal}: $A^{-1} = A^\top$

Properties of orthonormal transforms:

- $\sum x^2 = \sum \Theta^2$ (energy conservation)

Would like to compact energy into as few coefficients as possible

$$G_{TC} = \frac{1}{n} \sum \sigma_i^2 \left( \prod \sigma_i^2 \right)^{1/n}$$

(The \textbf{transform coding gain})

$\sigma_i = (\Theta_i - \Theta_{av})$

The higher the gain, the better the compression
Case Study: JPEG

A nice example since it uses many techniques:
- Transform coding (Discrete Cosine Transform)
- Scalar quantization
- Difference coding
- Run-length coding
- Huffman or arithmetic coding

**JPEG** (Joint Photographic Experts Group) was designed in 1991 for lossy and lossless compression of color or grayscale images. The lossless version is rarely used.

Can be adjusted for compression ratio (typically 10:1)
JPEG in a Nutshell

Typically down-sample I and Q planes by a factor of 2 in each dimension - lossy. Factor of 4 compression for I and Q, 2 overall.

break each plane into 8x8 blocks of pixels
### JPEG: Quantization Table

<table>
<thead>
<tr>
<th>16</th>
<th>11</th>
<th>10</th>
<th>16</th>
<th>24</th>
<th>40</th>
<th>51</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>14</td>
<td>19</td>
<td>26</td>
<td>58</td>
<td>60</td>
<td>55</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>16</td>
<td>24</td>
<td>40</td>
<td>57</td>
<td>69</td>
<td>56</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
<td>22</td>
<td>29</td>
<td>51</td>
<td>87</td>
<td>80</td>
<td>62</td>
</tr>
<tr>
<td>18</td>
<td>22</td>
<td>37</td>
<td>56</td>
<td>68</td>
<td>109</td>
<td>103</td>
<td>77</td>
</tr>
<tr>
<td>24</td>
<td>35</td>
<td>55</td>
<td>64</td>
<td>81</td>
<td>104</td>
<td>113</td>
<td>92</td>
</tr>
<tr>
<td>49</td>
<td>64</td>
<td>78</td>
<td>87</td>
<td>103</td>
<td>121</td>
<td>120</td>
<td>101</td>
</tr>
<tr>
<td>72</td>
<td>92</td>
<td>95</td>
<td>98</td>
<td>112</td>
<td>100</td>
<td>103</td>
<td>99</td>
</tr>
</tbody>
</table>

Divide each coefficient by factor shown. Also divided through uniformly by a quality factor which is under control.
JPEG: Block scanning order

- Scan block of coefficients in zig-zag order
- Use difference coding upper left (DC) coefficient between consecutive blocks
- Uses run-length coding for sequences of zeros for rest of block
JPEG: example

.125 bits/pixel (factor of 192)
Case Study: MPEG

Pretty much JPEG with interframe coding

Three types of frames

- **I** = intra frame (approx. JPEG) anchors
- **P** = predictive coded frames - based on previous I or P frame in output order
- **B** = bidirectionally predictive coded frames - based on next and/or previous I or P frames in output order

Example:

<table>
<thead>
<tr>
<th>Type:</th>
<th>I</th>
<th>B</th>
<th>B</th>
<th>P</th>
<th>B</th>
<th>B</th>
<th>P</th>
<th>B</th>
<th>B</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Order:</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

I frames are used for random access. In the sequence, each B frame appears after any frame on which it depends.
MPEG matching between frames
**MPEG: Compression Ratio**

356 x 240 image

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>18KB</td>
<td>7/1</td>
</tr>
<tr>
<td>P</td>
<td>6KB</td>
<td>20/1</td>
</tr>
<tr>
<td>B</td>
<td>2.5KB</td>
<td>50/1</td>
</tr>
<tr>
<td>Average</td>
<td>4.8KB</td>
<td>27/1</td>
</tr>
</tbody>
</table>

30 frames/sec x 4.8KB/frame x 8 bits/byte
= 1.2 Mbits/sec + .25 Mbits/sec (stereo audio)

HDTV has 15x more pixels
= 18 Mbits/sec
MPEG in the “real world”

- DVDs
  - Adds “encryption” and error correcting codes
- Direct broadcast satellite
  - Adds error correcting code on top
- Storage Tech “Media Vault”
  - Stores 25,000 movies

Encoding is much more expensive than decoding. Still requires special purpose hardware for high resolution and good compression.
Compression Summary

How do we figure out the probabilities
- Transformations that skew them
  - Guess value and code difference
  - Move to front for temporal locality
  - Run-length
  - Linear transforms (Cosine, Wavelet)
  - Renumber (graph compression)
- Conditional probabilities
  - Neighboring context

In practice one almost always uses a combination of techniques