Multimedia Software Systems CS4551

Lossless Compression – Part II

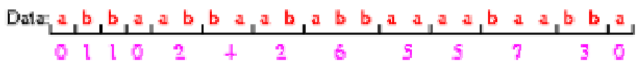
Examples of Entropy Encoding

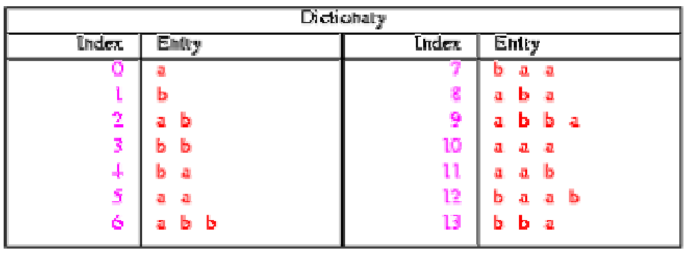
- Types of techniques vary depending on how much you statistically analyze the signal
- Run-length Encoding
- Repetition Suppression
- Variable Length Coding (Entropy Coding)
  - Shannon-Fano Algorithm
  - Huffman coding
  - Adaptive Huffman coding
- Pattern Substitution : A pattern (not a single symbol), which occurs frequently, is replaced by a specific codeword
  - Dictionary based Coding **LZW**
- Arithmetic Coding

LZW – Pattern Substitution Encoding

- Invented by **L**empel and **Z**iv in 1978(LZ78) and improvements given by **W**elch in 1984.
- <u>**Encoding Process**</u>:
  1. Initialize the dictionary to contain all single/initial symbols.
     - The vocabulary (collection of symbols) forms the initial dictionary entries.
     - Every entry in the dictionary is given an index.
  2. While scanning the message to compress, search for the longest sequence of symbols that has appeared as an entry in the dictionary. Call this entry E.
  3. Encode E in the message by its index in the dictionary.
  4. Add a new entry to the dictionary, which is E followed by the next symbol in the scan.
  5. Repeat the process (step 2- step 4) until you reach the end of the message.

Pattern Substitution – Example 1





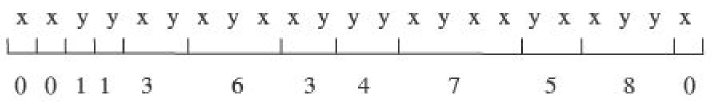
Dictionary			
Index	Entry	Index	Entry
0	a	7	b a a
1	b	8	a b a
2	a b	9	a b b a
3	b b	10	a a a
4	b a	11	a a b
5	a a	12	b a a b
6	a b b	13	b b a

Number of bits to represent indices of dictionary with 14 entries = 4 bits

Number of indices to encode the input = 13

Data Size after Compression = 13 x 4 bits = 52 bits (Ignore the dictionary size with single-length entries)

Pattern Substitution –Example 2



Index	Entry	Index	Entry
0	x	7	x y x x
1	y	8	x y y
2	x x	9	y y x
3	x y	10	x y x x y
4	y y	11	y x x
5	y x	12	x y y x
6	x y x		

Pattern Substitution –Example 3

- Data : `abababab`
- Encode the data using LZW. Show the dictionary.

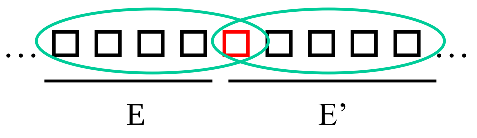
LZW – Pattern Substitution Decoding

- **<u>Decoding Process</u>** (with the exception handling)
  1. Start with the dictionary containing blocks of length one
  2. while not (the end of encoded data)
     - K <- the next index
     - Output Entry E at the index K from the dictionary
     - Update the Dictionary
       – If the entry E' at the next index exists from the dictionary, then add to the dictionary E + First_Symbol(E').
       – *Otherwise, add E + First_Symbol(E) to the dictionary.*

LZW – Pattern Substitution Decoding

- Handling exceptions (cases when an entry for the index does not exist in the dictionary for decoding)

  Otherwise, add E + First_Symbol(E) to the dictionary.
- Analysis:
  – An exceptional case is produced in the encoding side when the entry `E+Next_Sysmbol` that has just entered into the dictionary was used immediately for encoding the next entry `E'`.
  – The next entry `E'` must be the form of `E+First_Symbol(E)`. That means `E+Next_Sysmbol` is same as `E+First_Symbol(E)`.



Pattern Substitution –Example 1

- Given the compressed data below, decode it using LZW. Show the dictionary.

```
0 1 1 0 2 4 2 6 5 5 7 3 0
```

Index	Entry
0	a
1	b

Pattern Substitution –Example 2

- Given the compressed data below, decode it using LZW. Show the dictionary.

```
0 0 1 1 3 6 3 4 7 5 8 0
```

Index	Entry
0	x
1	y

Pattern Substitution –Example 3

- Consider the previous encoding example with `abababab.` Decode the data using LZW. Show the dictionary.

LZW – Pattern Substitution

- Variable length pattern to fixed length index
- Size of the dictionary
  - Theoretically, it can grow infinitely
  - Practically, recommended size is 4096 (12 bits). Once the limit is reached, no more entries are added.
- Works well when the input data is sufficiently large and there are sufficient redundancy in data.
- Eg. – UNIX compress utility, gzip, gunzip, Windows WinZip, GIF and TIFF (optional) compression

Arithmetic Coding

- Arithmetic coding is a more modern coding method that usually outperforms Huffman coding in practice.
- Initial idea was introduced by Shannon in 1948.
- It was fully developed in the late 1970s and 1980s.
- Arithmetic coding treats the whole message as one unit. A message is represented by a half-open interval `[a,b)` where `a` and `b` are real numbers between 0 and 1 and encoded as a number that falls with the range `[a,b)`.
- In practice, the input data is usually broken up into chunks. In this class, we take a short and simple message and encode it.

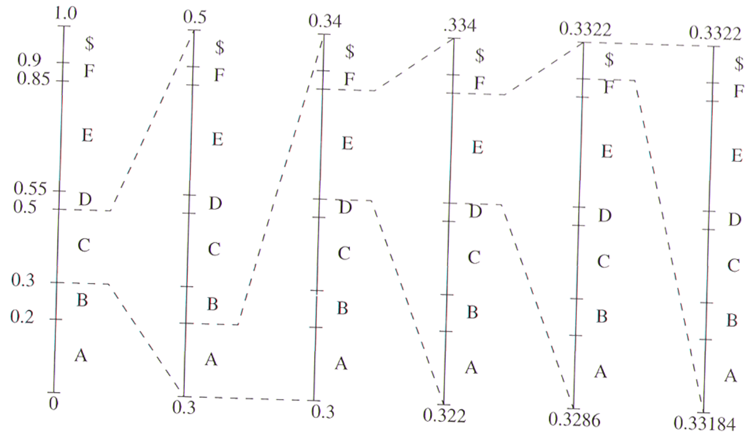
Arithmetic Coding

- **Suppose that we have 7 symbols {A,B,C,D,E,F,$} and their probabilities are 0.2, 0.1, 0.2, 0.05, 0.3, 0.05, 0.1.**

Symbol	Probability	Symbol Range
A	0.2	[0, 0.2)
B	0.1	[0.2, 0.3)
C	0.2	[0.3,0.5)
D	0.05	[0.5,0.55)
E	0.3	[0.55,0.85)
F	0.05	[0.85,0.9)
$	0.1	[0.9,1)

Arithmetic Coding

- **Suppose that we have a text string "CAEE$" to encode. Arithmetic coding finds a unique range [a,b) to represent this string.**



Arithmetic Coding

- **Suppose that we have a text string "CAEE$" to encode. Arithmetic coding finds a unique range [a,b) to represent this string.**

Symbol	Low	High	Range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
$	0.33184	0.33220	0.00036

- **Then, generate a number (a binary fraction) that falls in within a range.**

Arithmetic Coding

- **<u>Algorithm: Arithmetic Coding Encoder</u>**

```
– BEGIN
    • low = 0.0; high = 1.0; range = 1.0
    • while (not end of the message)
        – get (symbol);
        – Update low and high by
            tmp = low
            low = tmp + range * range_low(symbol);
            high = tmp + range * range_high(symbol);
        – range = high-low;
    • output a binary code that falls
      within [low, high)--> algorithm in
      the next slide
– END
```

CSULA CS451 Multimedia Software Systems by Eun-Young Kang

Binary Fractions

$2^0 = 1$

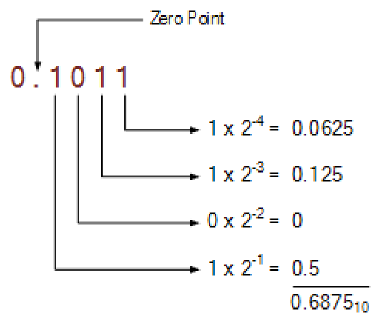
$2^{-1} = \frac{1}{2^1} = \frac{1}{2} = 0.5$

$2^{-2} = \frac{1}{2^2} = \frac{1}{4} = 0.25$

$2^{-3} = \frac{1}{2^3} = \frac{1}{8} = 0.125$

$2^{-4} = \frac{1}{2^4} = \frac{1}{16} = 0.0625$

$2^{-5} = \frac{1}{2^5} = \frac{1}{32} = 0.03125$



https://www.electronics-tutorials.ws/binary/binary-fractions.html

CSULA CS451 Multimedia Software Systems by Eun-Young Kang

Arithmetic Coding

- **Algorithm: Generating Codeword for Encoder**

```
– BEGIN
    • code = 0;
    • k=1;
    • while (value(code) < low)
        – assign 1 to the k-th binary fraction bit
        – if (value(code) > high)
            » replace the k-th bit by 0
        – k = k+1;
– END
```

- **For CAEE$, we have [0.33184, 0.3322). Using the algorithm, we acquire a binary fraction number 0.01010101, which is 0.33203125 in decimal fraction. Therefore, to represent the sequence of symbol "CAEE$", we need 8 bits.**

Arithmetic Coding

- **Decoding 0.01010101(=0.33203125) . We need symbols and probability information for decoding.**
- **Algorithm: Arithmetic Coding Decoder**

```
– BEGIN
    • value <- Get binary code and covert to decimal
      value
    • Do
        – Find a symbol s so that value is within
          [Range_low(s), Range_high(s))
        – Output s
        – low = Range_low(s)
        – high = Range_high(s)
        – range = high - low
        – value = [value – low]/range; // rescaling
    • Until end of the message
– END
```

Arithmetic Coding

- Decoding 0.01010101(=0.33203125) .

Value	Output symbol	Low	high	Range
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	$	0.9	1.0	0.1

Arithmetic Coding

- The presented Arithmetic coding algorithm requires to use very high-precision numbers to do encoding when the interval shrinks. It is possible to rescale the intervals and use only integer arithmetic for a practical implementation.
- Arithmetic coding is best accomplished using standard 16 bit or 32 bit integer math.
- Known to be better than Huffman coding.
  - Eg. Let's assume we have symbols {A,B,C} and their probabilities p(A)=0.5, p(B) = 0.4, p(C) =0.1 (assign half-open intervals in this order for the Arithmetic coding.) For "BBB", Huffman coding will require 6 bits, whereas Arithmetic coding will require only 4 bits.

Lossless Image Compression – Differential Coding of Image

- Differential coding of image
  - Use the idea that neighboring image pixels are similar.
- Given the original image, I(x,y) means a pixel value in (x,y) location and we can define a difference image using the following differential operation d(x,y) = I(x,y) – I(x-1, y), which is a simple approximation of a partial differential operator ∂/ ∂x applied to an image.
- An example

The original image

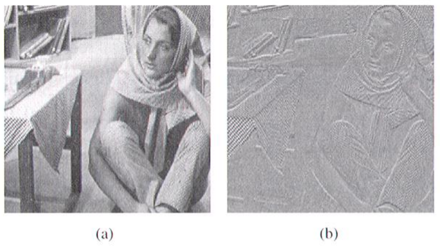
10	11	12
13	11	12
14	16	16

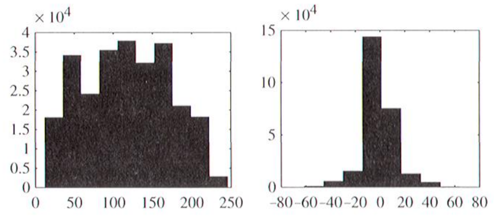
Differential image

10	1	1
13	-2	1
14	2	0

Lossless Image Compression – Differential Coding of Image

- **Usually the entropy of the original image of the original image is larger than the entropy of the differential image. So we can apply VLC (such as Huffman coding) to the differential image and achieve better compression ratio.**


(a) (b)



Lossless Image Compression – Differential Coding of Image

- For decoding, use I(x,y) = I(x-1,y) + d(x,y)

10	1	1
13	-2	1
14	2	0

$\longrightarrow$

Lossless Image Compression – Prediction Based

- Predictors that use only 1 pixels are called first-order. Differential Coding method presented in the previous slides use a first-order predictor.
- Predictors that use only 2, 3 or more pixels are called second-order, third order or high-order. This prediction based image compression is used for JPEG lossless mode compression.
- Let's assume that we want to predict X pixel

and compute the difference. There are many ways to predict.

P = 1/2A + 1/2C

or P = A – B + C

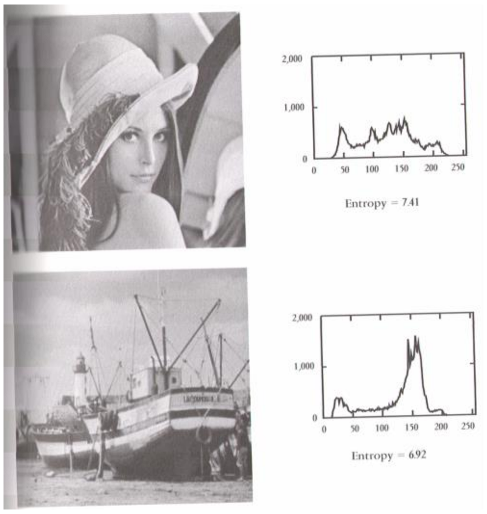
…

Then d(x,y) = X - P = I(x,y) – P

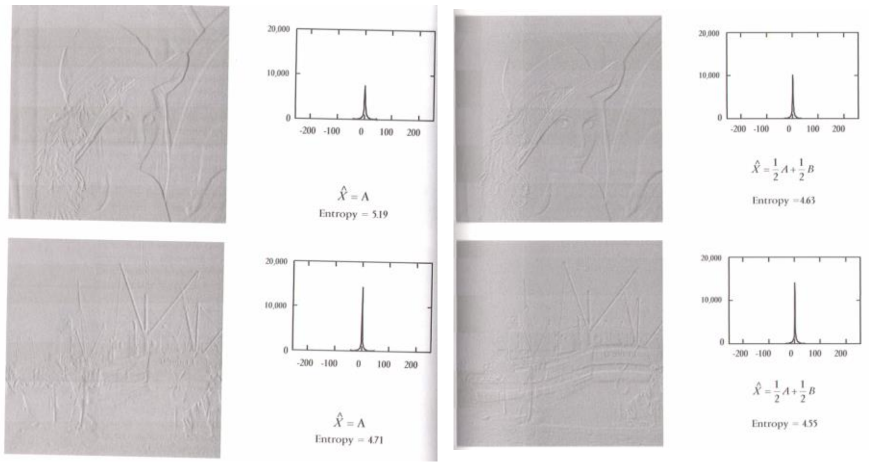
eg. d(x,y) = I(x,y) – (1/2 *I(x-1,y) +1/2* I(x, y-1))

B	C	D
A	X	

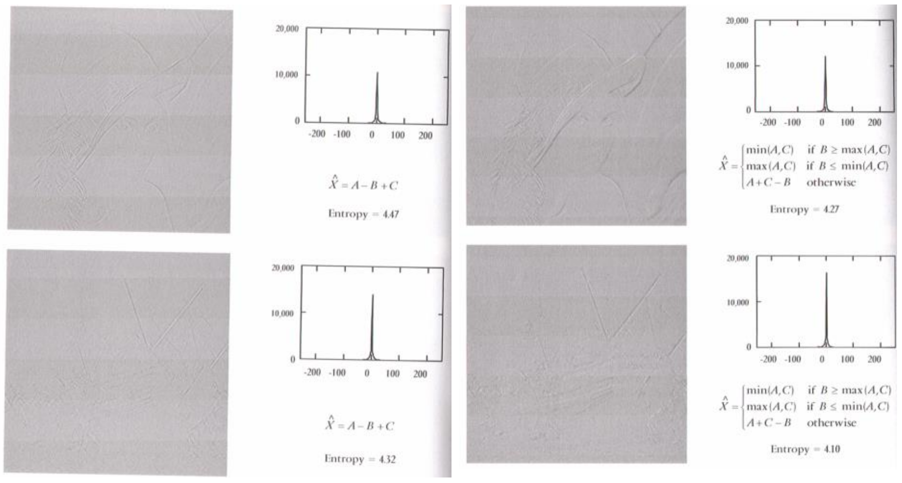
Lossless Image Compression – Prediction Based



Lossless Image Compression – Prediction Based



Lossless Image Compression – Prediction Based



$\hat{X} = A - B + C$

Entropy = 4.47

$\hat{X} = \begin{cases} \min(A,C) & \text{if } B \geq \max(A,C) \\ \max(A,C) & \text{if } B \leq \min(A,C) \\ A+C-B & \text{otherwise} \end{cases}$

Entropy = 4.27

$\hat{X} = A - B + C$

Entropy = 4.32

$\hat{X} = \begin{cases} \min(A,C) & \text{if } B \geq \max(A,C) \\ \max(A,C) & \text{if } B \leq \min(A,C) \\ A+C-B & \text{otherwise} \end{cases}$

Entropy = 4.10