Chapter 3.
Lossless Compression

Entropy

Entropy: $H = \sum p_i \cdot \log_2 \frac{1}{p_i}$

$i = 2, p_1 = 1, \quad p_2 = 0 \quad \Rightarrow \quad H = 0$

$p_1 = 1/2, \quad p_2 = 1/2 \quad \Rightarrow \quad H = 1$ *bits / symbol*

$i = 4, p_1 = p_2 = p_3 = p_4 = 1/4 \quad \Rightarrow \quad H = 2$ *bits / symbol*

- From information theory, the average number of bits needed to encode the symbols in a source *S* (using zeroth-order entropy coding) is always bounded by the entropy of *S*.

zeroth-order每次coding都是一個一個來 v.s. vector quantization 綁一些人一起coding

The Kraft-Mcmillan Inequality (1/2)

Part 1:
Given a uniquely decodable variable-size code with *K* codes of size l_i, then

$$\sum_{i=1}^{K} 2^{-l_i} \leq 1, \qquad (1)$$

Part 2:
Given a set of *K* positive integers $\{l_1, l_2, \ldots, l_K\}$ that satisfies Eq. (1), there always exists a prefix code such that l_i are the sizes of its individual codes

The Kraft-Mcmillan Inequality (2/2)

For a source S with alphabet $A = \{a_1, a_2, \ldots, a_K\}$ the length l_i of code a_i can be written as

$$l_i = -\log_2 P(a_i) + e_i$$

where e_i is the extra length of a_i

$$2^{-l_i} = 2^{(\log_2 P(a_i) - e_i)} = \frac{2^{\log_2 P(a_i)}}{2^{e_i}} = \frac{P(a_i)}{2^{e_i}}$$

In the special case where all the extra lengths are all the same ($e_i = e$)
Then

$$\sum_{i=1}^{K} 2^{-l_i} \leq 1$$

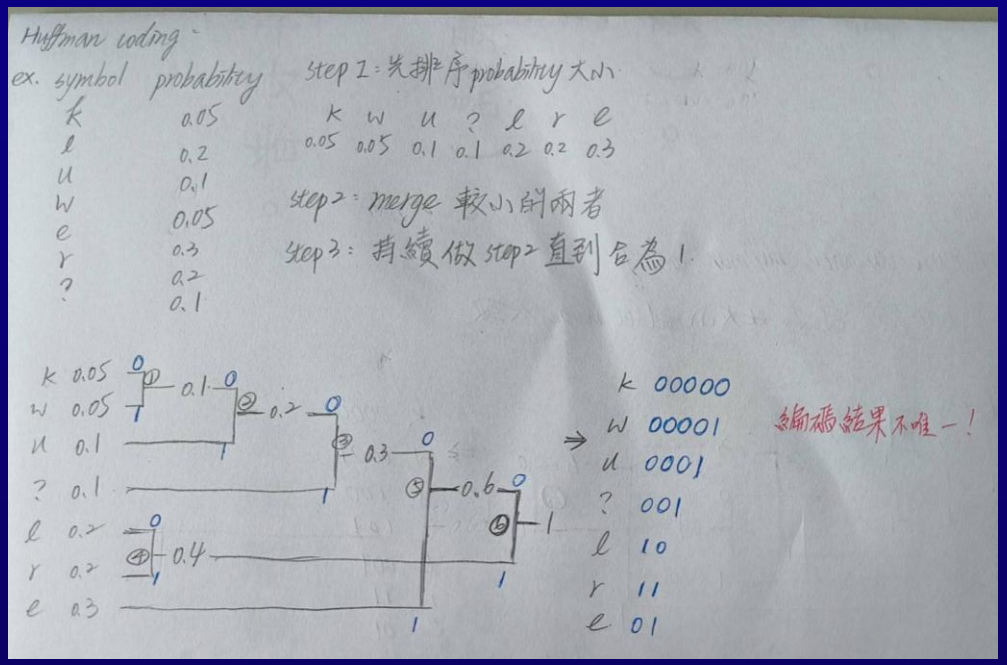
$$\Rightarrow \sum_{i=1}^{K} 2^{-l_i} = \sum_{i=1}^{K} \frac{P(a_i)}{2^e} = \frac{\sum_{i=1}^{K} P(a_i)}{2^e} = \frac{1}{2^e} \leq 1$$

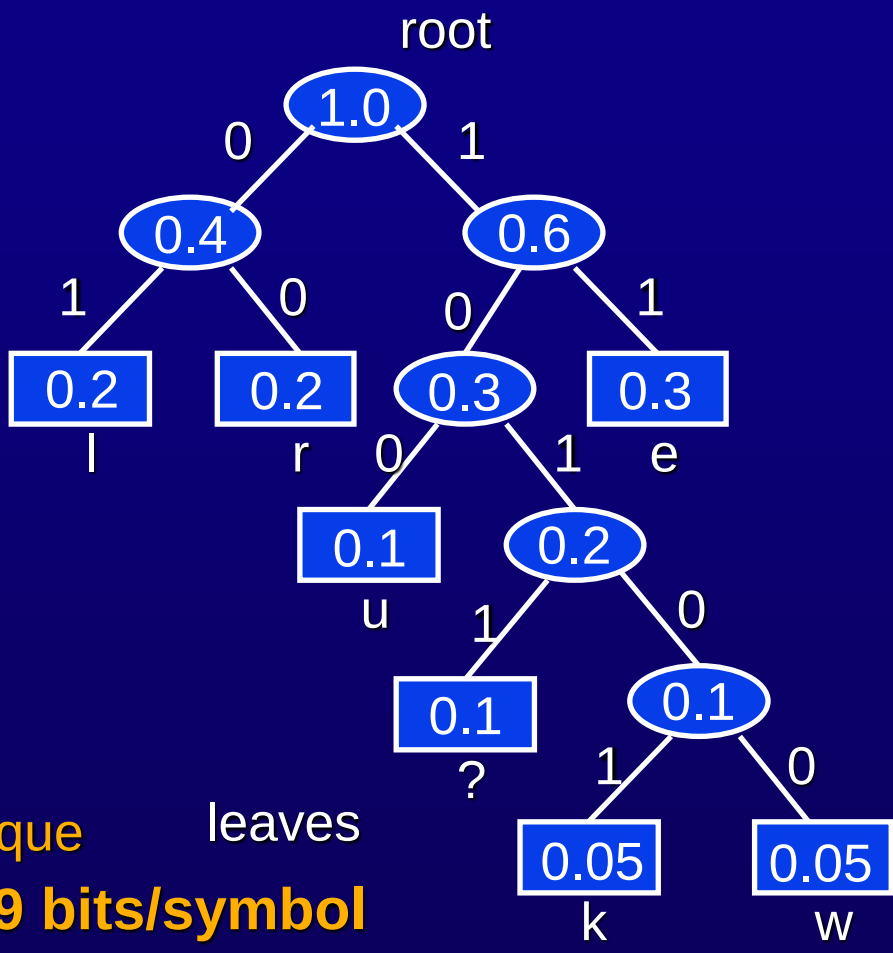
$$\Rightarrow 2^e \geq 1$$

$$\Rightarrow e \geq 0$$

The length of a uniquely decodable code is greater or equal to the length determined by its entropy

Huffman Coding (1/2)(必考)





- Not unique

leaves

Entropy: 2.546439 bits/symbol

Average coding rate: 2.6 bits/symbol

$$= \sum l_i p_i = 0.05 \times 5 + 0.2 \times 2 + 0.1 \times 3 + \cdots$$

Huffman Coding (2/2)

lluure?

01011001000011101l

- Compression ratio: 3x7/18 = 1.16
- Data expansion
- Prefix code
- Decoding
- Transmission error propagation
- Interface to fixed-length memory
- Probability modeling, code-book generation

Minimum Variance Huffman Coding

Example:

Letters	$P(a_i)$	Code 1	Code 2
a_0	0.4	1	00
a_1	0.2	01	10
a_2	0.2	000	11
a_3	0.1	0010	010
a_4	0.1	0011	011

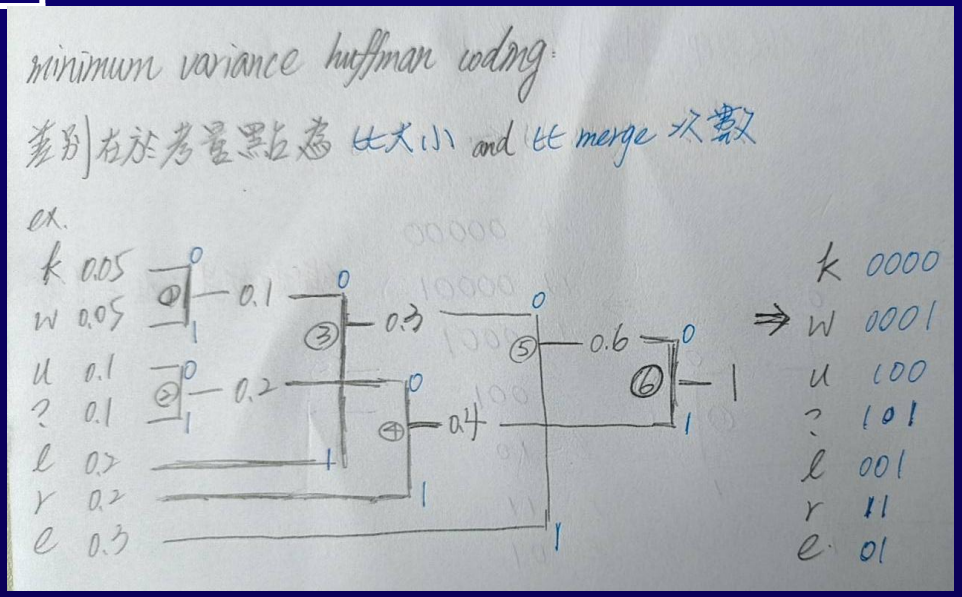
H = 2.122 bits/symbol

$l_1 = l_2$ = 2.2 bits/symbol

$\sigma_1^2 = 0.4(1-2.2)^2 + 0.2(2-2.2)^2 + 0.2(3-2.2)^2 + 0.1(4-2.2)^2 = 1.36$

$\sigma_2^2 = 0.4(2-2.2)^2 + 0.2(2-2.2)^2 + 0.1(3-2.2)^2 + 0.1(3-2.2)^2 = 0.16$

Minimum variance Huffman code



Unconstrained Length Huffman Coding

Symbol s_i	p_i	l_i	Codeword
0	0.28200	2	11
1	0.27860	2	10
2	0.14190	3	011
3	0.13890	3	010
4	0.05140	4	0011
5	0.05130	4	0010
6	0.01530	5	00011
7	0.01530	5	00010
8	0.00720	6	000011
9	0.00680	6	000010
10	0.00380	7	0000011
11	0.00320	7	0000010
12	0.00190	7	0000001
13	0.00130	8	00000001
14	0.00070	9	000000001
15	0.00040	9	000000000

Constrained Length Huffman Coding

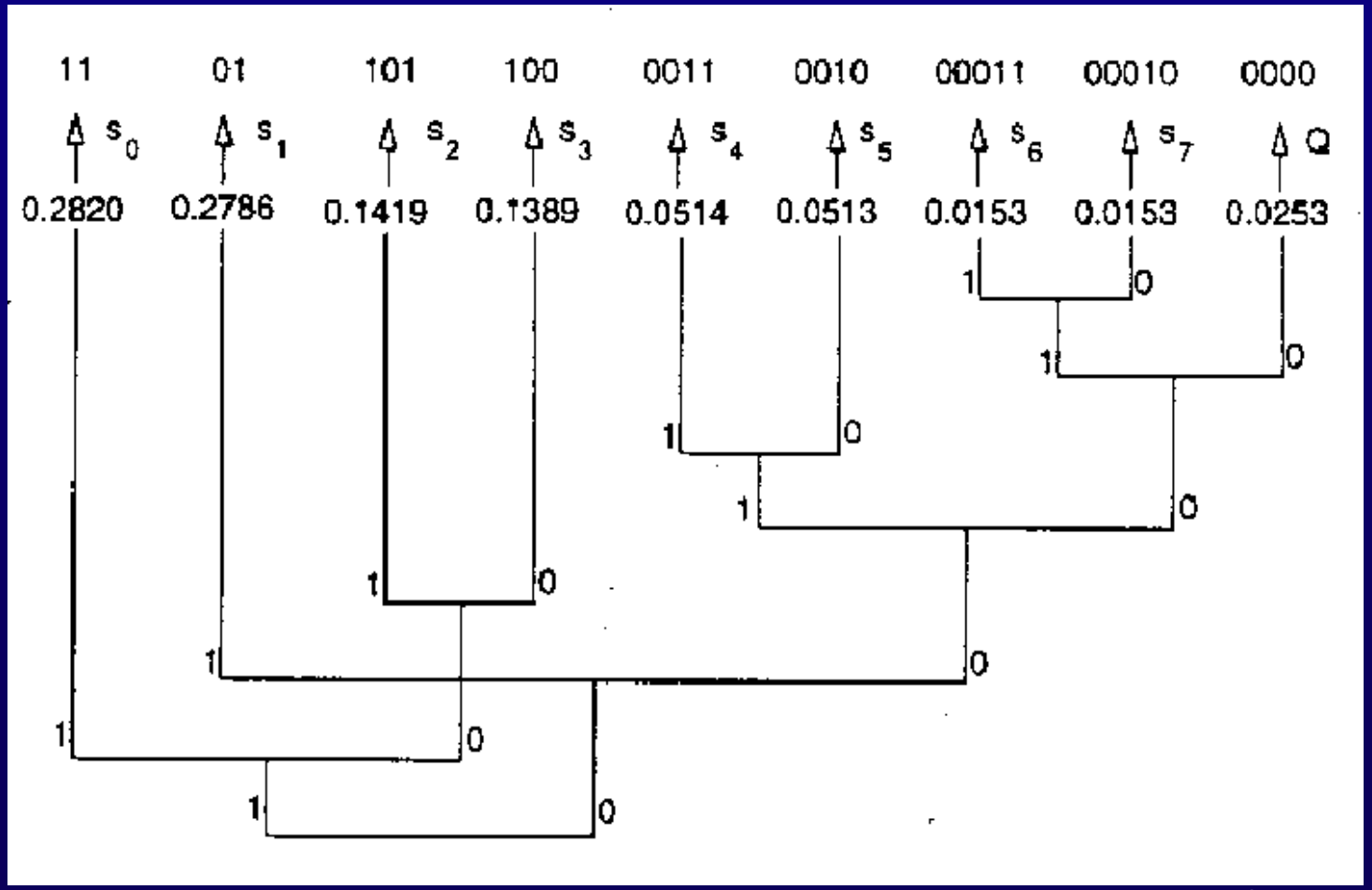
- Problem: when some of the symbol possibilities are extremely small, they will require a large number of bits
- Solution: shorten and hierarchy
- Algorithm:
  - ➔ Partition symbol set S into two sets S_1 and S_2 using $p_i=1/2^L$ as the boundary, where L is the max. codeword length expected.
  - ➔ Create a special symbol Q such that its frequency of occurrence is the sum of all p_i in S_2
  - ➔ Augment S_1 by Q to form a new set W. The new set has the occurrence frequencies corresponding to symbols in S_1 and the special symbol Q
  - ➔ Reconstruct the Huffman tree for W and S_2

Constrained Length Huffman Coding (1/5)

- Shortened Huffman code

  Combine symbols with probabilities <= $1/2^L$

e.g., L=7



Constrained Length Huffman Coding (2/5)

- Shortened Huffman code

Symbol i	p_i	l_i	Codeword	Additional
0	0.28200	2	11	
1	0.27860	2	01	
2	0.14190	3	101	
3	0.13890	3	100	
4	0.05140	4	0011	
5	0.05130	4	0010	
6	0.01530	5	00011	
7	0.01530	5	00010	
8	0.00720	7	0000	000
9	0.00680	7	0000	001
10	0.00380	7	0000	010
11	0.00320	7	0000	011
12	0.00190	7	0000	100
13	0.00130	7	0000	101
14	0.00070	7	0000	110
15	0.00040	7	0000	111

escape-code + fixed-length code

Constrained Length Huffman Coding (3/5)

- Ad-Hoc method

1. Sort symbols so that $p_1 \geq p_2 \geq \ldots \geq p_N$
2. For max. codeword length L, define $T=2^{-L}$
3. For $p_n<T$, set $p_n = T$
4. Design the codebook using the modified unconstrained Huffman coding method codewords according to the lengths

Constrained Length Huffman Coding (5/5)

Symbol i	p_i	Unconstrained	Ad-hoc	Voorhis
0	0.28200	11	11	11
1	0.27860	10	01	10
2	0.14190	011	101	011
3	0.13890	010	100	010
4	0.05140	0011	0010	0011
5	0.05130	0010	0001	0010
6	0.01530	00011	001100	00011
7	0.01530	00010	001101	00010
8	0.00720	000011	000010	0000111
9	0.00680	000010	000011	0000110
10	0.00380	0000011	000000	0000101
11	0.00320	0000010	000001	0000100
12	0.00190	0000001	0011110	0000011
13	0.00130	00000001	0011111	0000010
14	0.00070	000000001	0011100	0000001
15	0.00040	000000000	0011101	0000000
l_{avg}		2.6940	2.7141	2.7045

Run-Length Coding

Input sequence:

0,0,-3,5,1,0,-2,0,0,0,0,2,-4,3,-2,0,0,0,1,0,0,-2

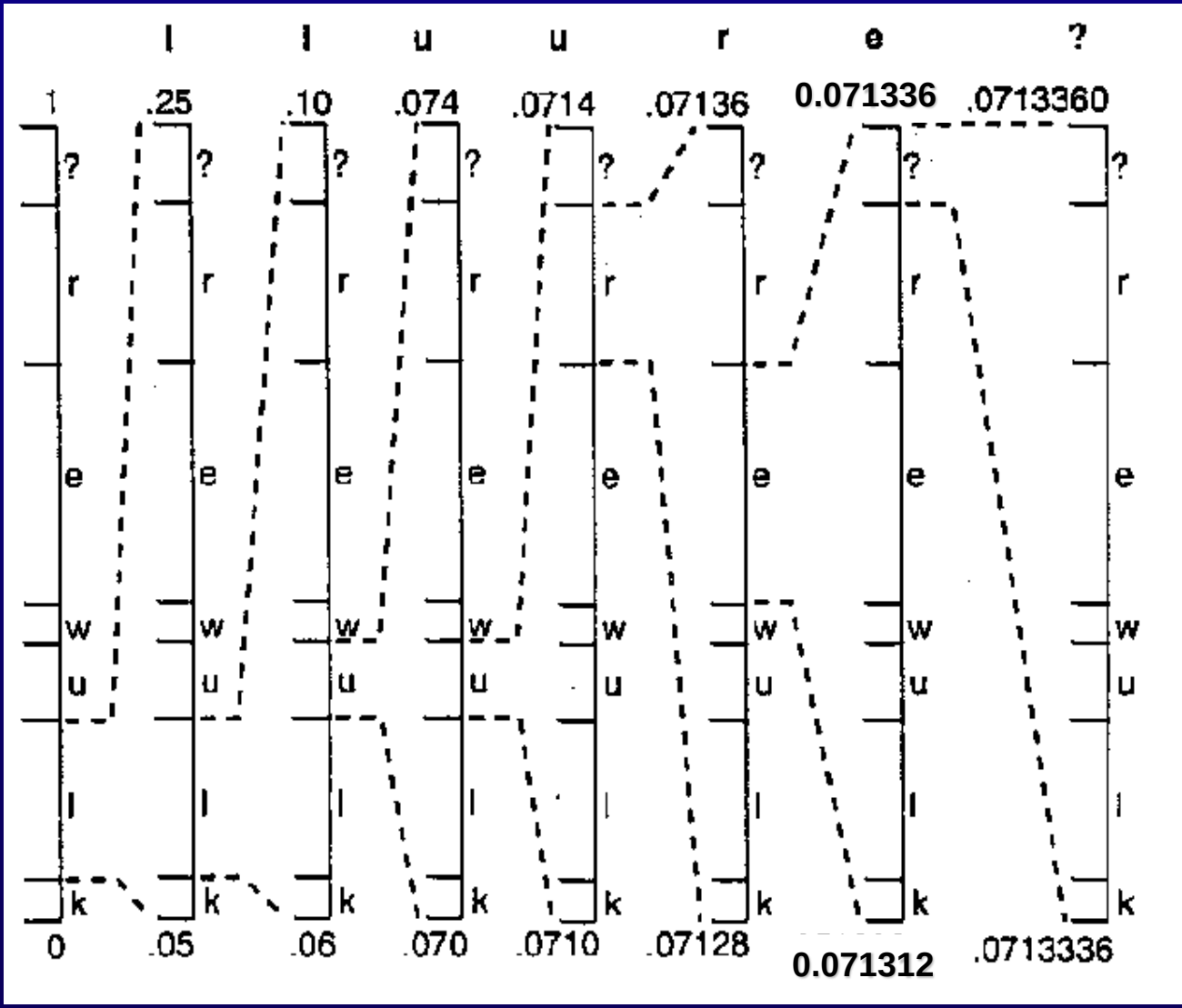
Run-length sequence:

(2,-3)(0,5)(0,1)(1,-2)(4,2)(0,-4)(0,3)(0,-2)(3,1)(2,-2)

- Reduce the number of samples to code
- Implementation is simple

Arithmetic Coding

S_i	p_i	sub-interval
k	0.05	[0.00, 0.05)
l	0.2	[0.05, 0.25)
u	0.1	[0.25, 0.35)
w	0.05	[0.35, 0.4)
e	0.3	[0.4, 0.7)
r	0.2	[0.7, 0.9)
?	0.1	[0.9, 1.0)



e.g., 0.0713348389 = 0.000100100100001

Why Arithmetic Coding? (1/2)

- For a source alphabet *S* with highly skewed probabilities

Assign one codeword for each symbol

Symbol	$P(a_i)$	Codeword
a_1	0.99	0
a_2	0.01	1

$$H = 0.08 \text{ bits/symbol}$$

$$\bar{l} = 1 \text{ bits/symbol}$$

Assign one codeword for every *two* symbols

Symbols	$P(a_i)$	Codeword
$a_1 a_1$	0.9801	0
$a_1 a_2$	0.0099	11
$a_2 a_1$	0.0099	100
$a_2 a_2$	0.0001	101

$$\bar{l}^{(2)} = 1.0299 \text{ bits/block}$$

$$\bar{l} = \frac{\bar{l}^{(2)}}{2} = 0.51495 \text{ bits/symbol}$$

Why Arithmetic Coding? (2/2)

Assign one codeword for every *three* symbols

Symbols	$P(a_i)$	Codeword
$a_1\,a_1\,a_1$	0.970299	0
$a_1\,a_1\,a_2$	0.009801	100
$a_1\,a_2\,a_1$	0.009801	101
$a_1\,a_2\,a_2$	0.000099	11100
$a_2\,a_1\,a_1$	0.009801	110
$a_2\,a_1\,a_2$	0.000099	11101
$a_2\,a_2\,a_1$	0.000099	11110
$a_2\,a_2\,a_2$	0.0001	11111

$$H = 0.08 \text{ bits/symbol}$$
$$\bar{l}^{(3)} = 1.05998 \text{ bits/block}$$
$$\bar{l} = \frac{\bar{l}^{(3)}}{3} = 0.3533 \text{ bits/symbol}$$

To "block" several symbols together at a time

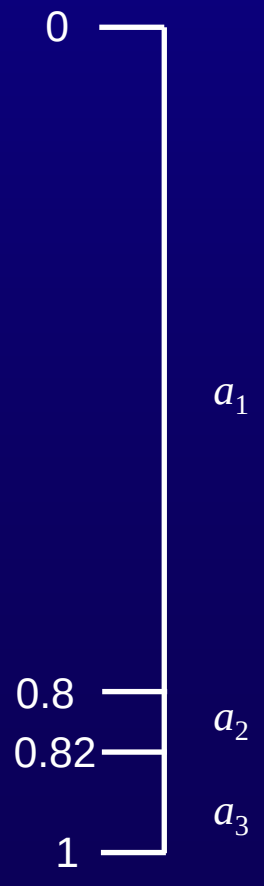
- per-symbol inefficiency is now spread over the whole block
- the size of the codebook increases exponentially
  - e.g., $A = \{a_1, a_2, \ldots, a_m\} \Rightarrow m^n$ codewords are needed to generate one codeword for every n symbols

Arithmetic Coding (1/4)

- Arithmetic Code
  - x assign codewords to individual symbols
  - - assign one (normally long) code to the entire input stream
  - $\Rightarrow$ A source message is represented by an interval of real number in [0,1)

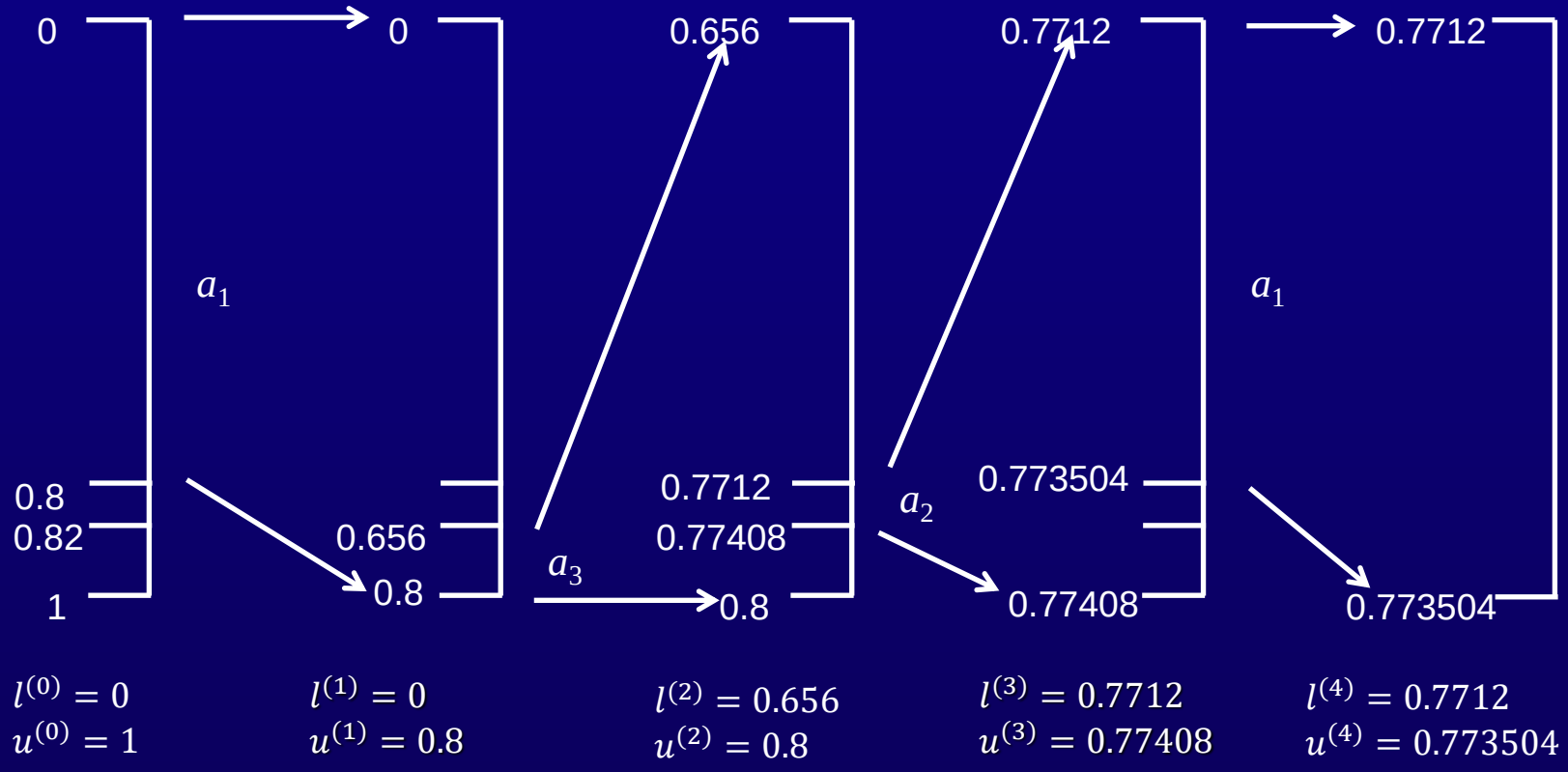
- Example:

Symbols	$P(a_i)$	cdf $F_x(i)$	range
a_1	0.8	0.8	[0,0.8)
a_2	0.02	0.82	[0.8,0.82)
a_3	0.18	1	[0.82,1)



Arithmetic Coding (2/4)

- Input sequence: $X = (a_1 a_3 a_2 a_1)$



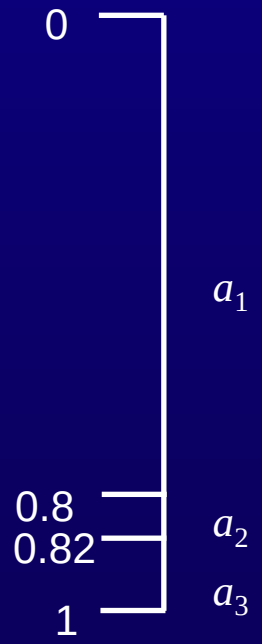
$$\begin{cases} l^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)}) F_x(x_{n-1}) \\ u^{(n)} = l^{(n-1)} + \left(u^{(n-1)} - l^{(n-1)}\right) F_x(x_n) \end{cases}$$

Arithmetic Coding (3/4)

Code = 0.772352

$$\begin{cases} l^{(0)} = 0 \\ u^{(0)} = 1 \end{cases}$$

$$x_1 = a_1$$
$$\begin{cases} l^{(1)} = 0 \\ u^{(1)} = 0.8 \end{cases}$$



$$code^* = \frac{0.772352 - 0.0}{0.8 - 0.0} = 0.96544$$
$$x_2 = a_3$$
$$\begin{cases} l^{(2)} = 0.656 \\ u^{(2)} = 0.8 \end{cases}$$

$$code^* = \frac{0.772352 - 0.656}{0.8 - 0.656} = 0.808$$
$$x_3 = a_2$$
$$\begin{cases} l^{(3)} = 0.7712 \\ u^{(3)} = 0.77408 \end{cases}$$

$$code^* = \frac{0.772352 - 0.7712}{0.77408 - 0.7712} = 0.4$$
$$x_4 = a_1$$

Decoding steps:

(1) Initialization

$$\begin{cases} l^{(0)} = 0 \\ u^{(0)} = 1 \end{cases}$$

(2) For each *k*

$$code^* = \frac{code - l^{(k-1)}}{u^{(k-1)} - l^{(k-1)}} = \frac{code - lower\_limit}{range}$$

(3) Find x_k s.t.

$$F_x(x_{k-1}) \leq code^* < F_x(x_k)$$

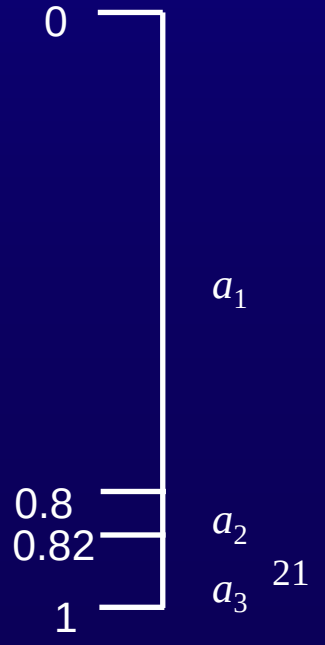
(4) Update $l^{(k)}$ & $u^{(k)}$

(5) Continue until the entire sequence has been decoded

Arithmetic Coding (4/4)

- How to know when the entire sequence has been decoded?
  - The length of the sequence is known in advance
    - the encoder can start by writing the unencoded size on the output stream
  - An additional symbol EOF is added
    - with a small probability
- A high-probability symbol narrows the interval less than a low-probability one does
  - high-probability symbols contribute fewer bits to the coded sequence

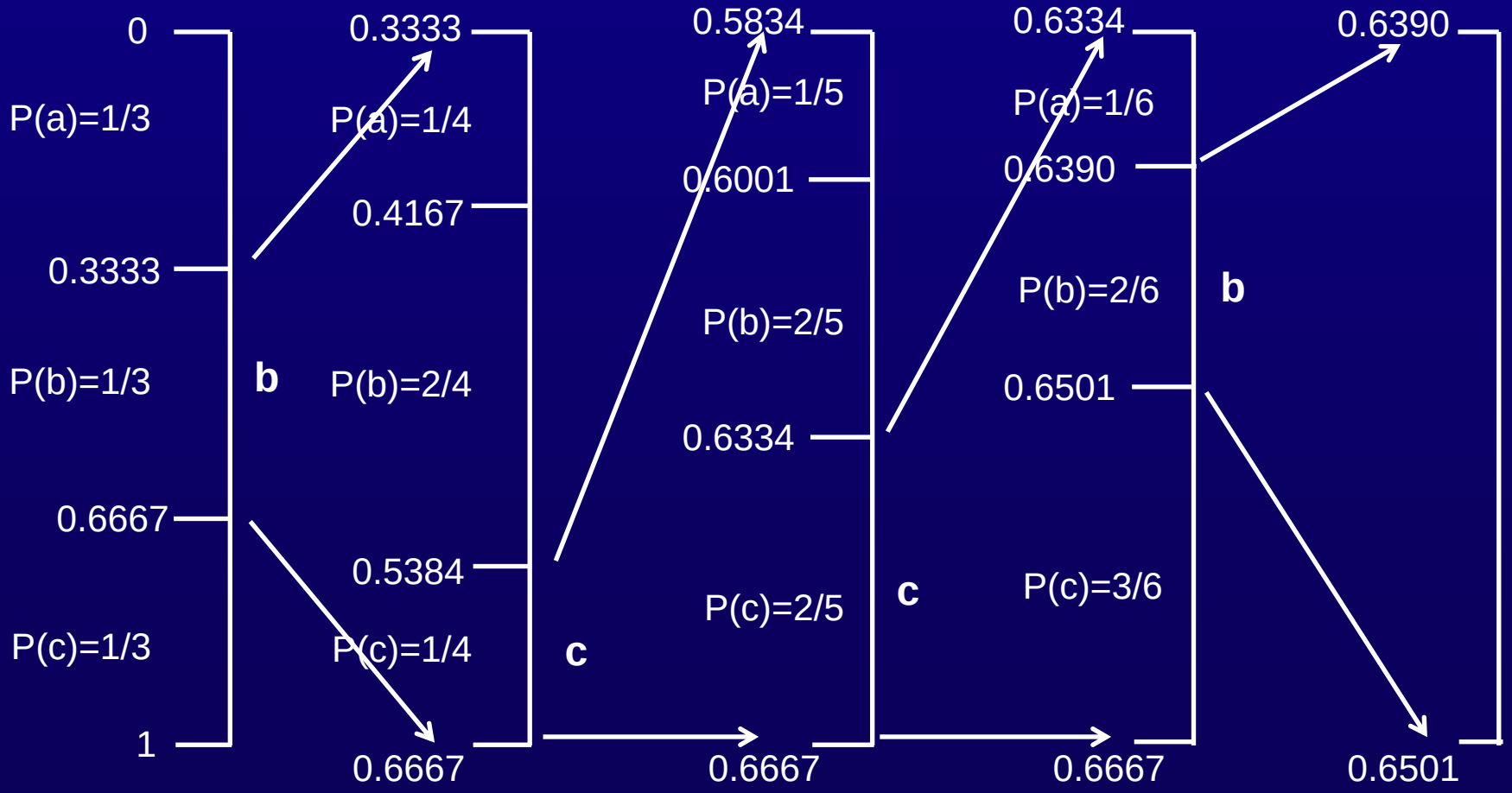
Symbols	$P(a_i)$	cdf $F_x(i)$	range
a_1	0.8	0.8	[0,0.8)
a_2	0.02	0.82	[0.8,0.82)
a_3	0.18	1	[0.82,1)



Adaptive Arithmetic Coding

Example

- for a source alphabet {a,b,c}
- input sequence = b c c b



Arithmetic Coding

- QM Coder - binary arithmetic coding
- Implementation issues
  - Incremental input/output
  - High-precision arithmetic
  - Re-normalization
  - Decoding termination
  - Probability modeling

Probability model 需預先知道資料性質（例如說誰誰誰出現機率高低），但是online（轉播）不行，因此可以拿之前類似的video 統計的結果先當作ref.，再依現在的資料進行fine-tune

- Why it works and why it works better than Huffman coding

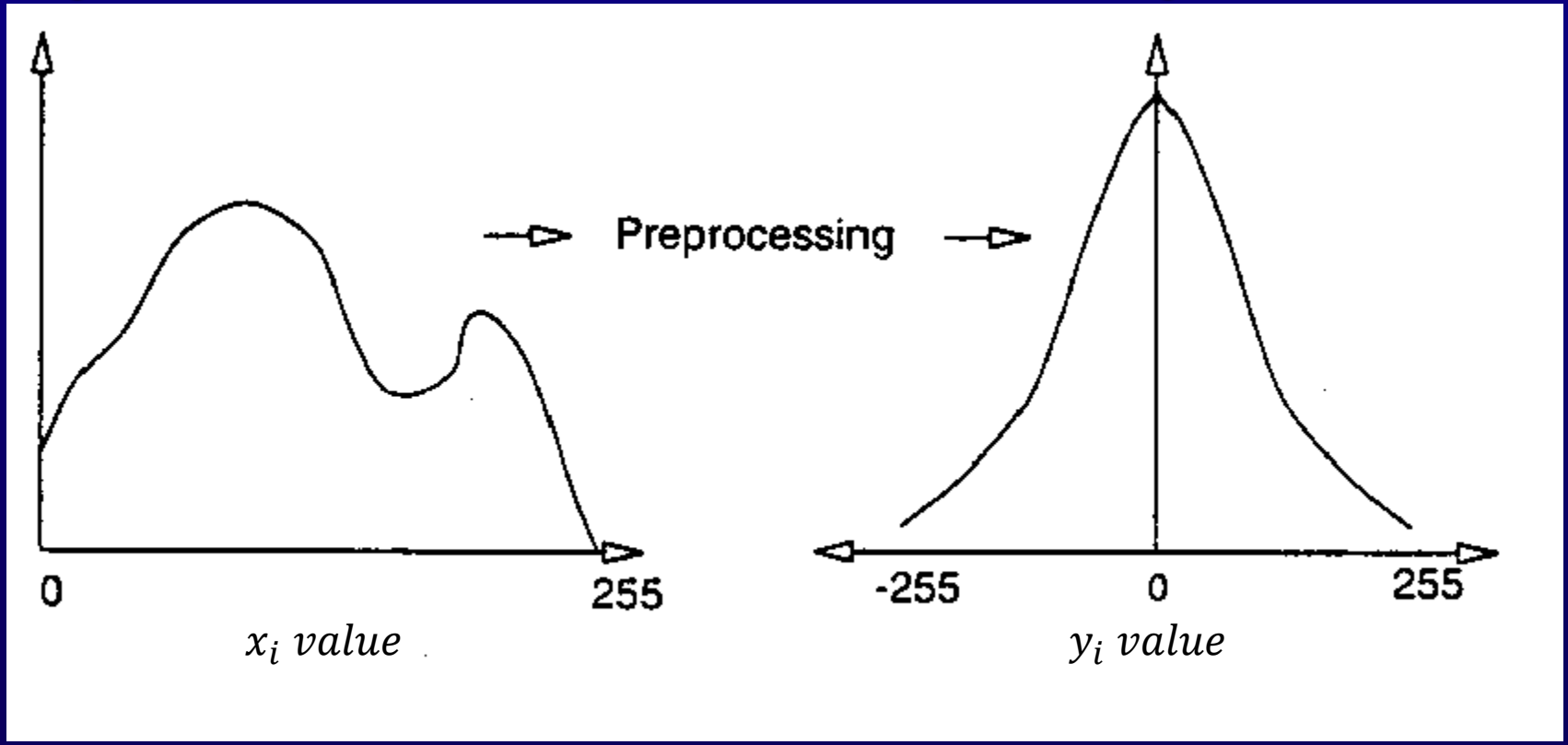
最接近最優entropy

Predictive Coding

進行鄰近（已知點）差值計算，得到preprocessing 後的樣子（相對均勻分布->相對集中分布，entropy 大->小，平均coding長度降低）

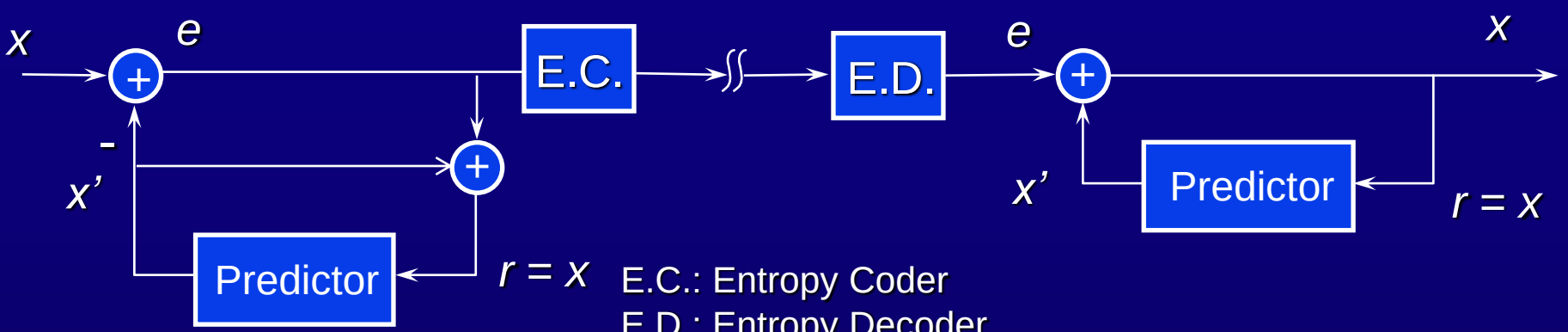
Histogram

Histogram



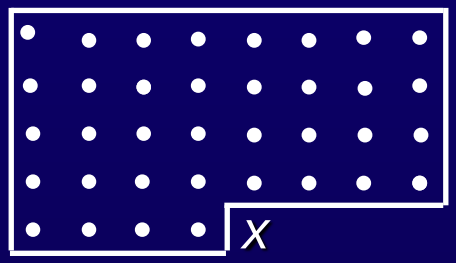
Lossless Predictive Coding

DPCM (differential pulse code modulation) System:



Encoder

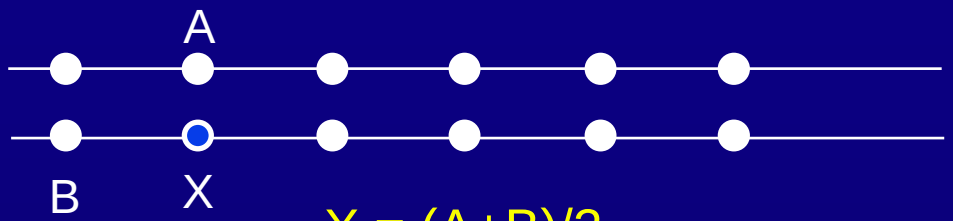
Decoder



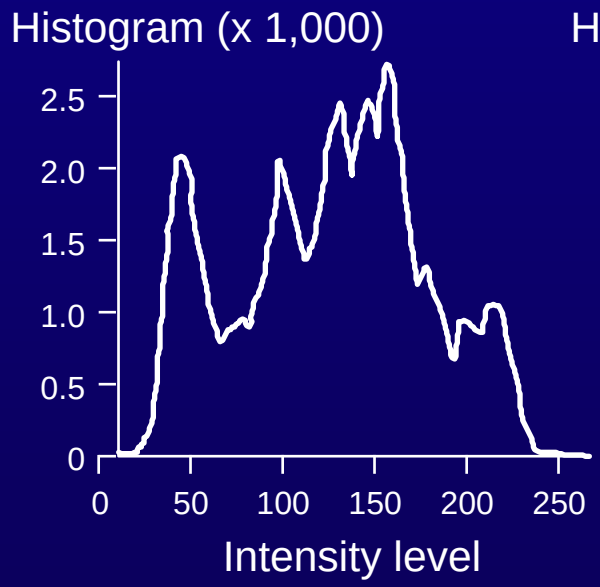
Region of causal predictor

- Dynamic range increased by 1-bit
- Synchronization between encoder and decoder
- Decoder inside the encoder
- Transmission error

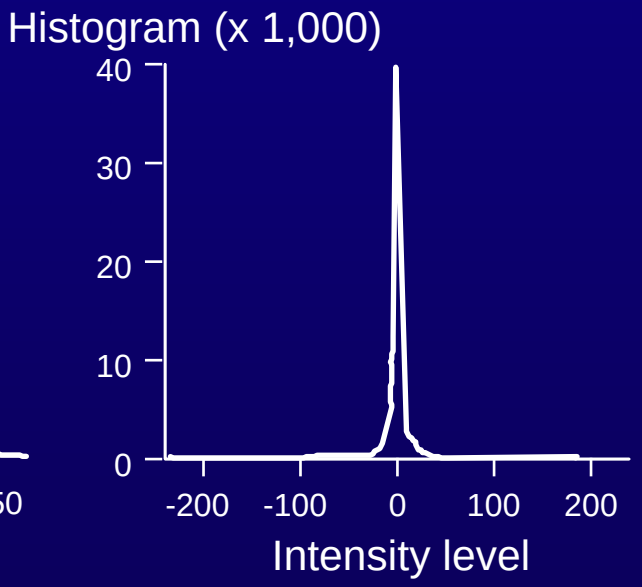
An Example of 2-D DPCM Coding



$X = (A+B)/2$

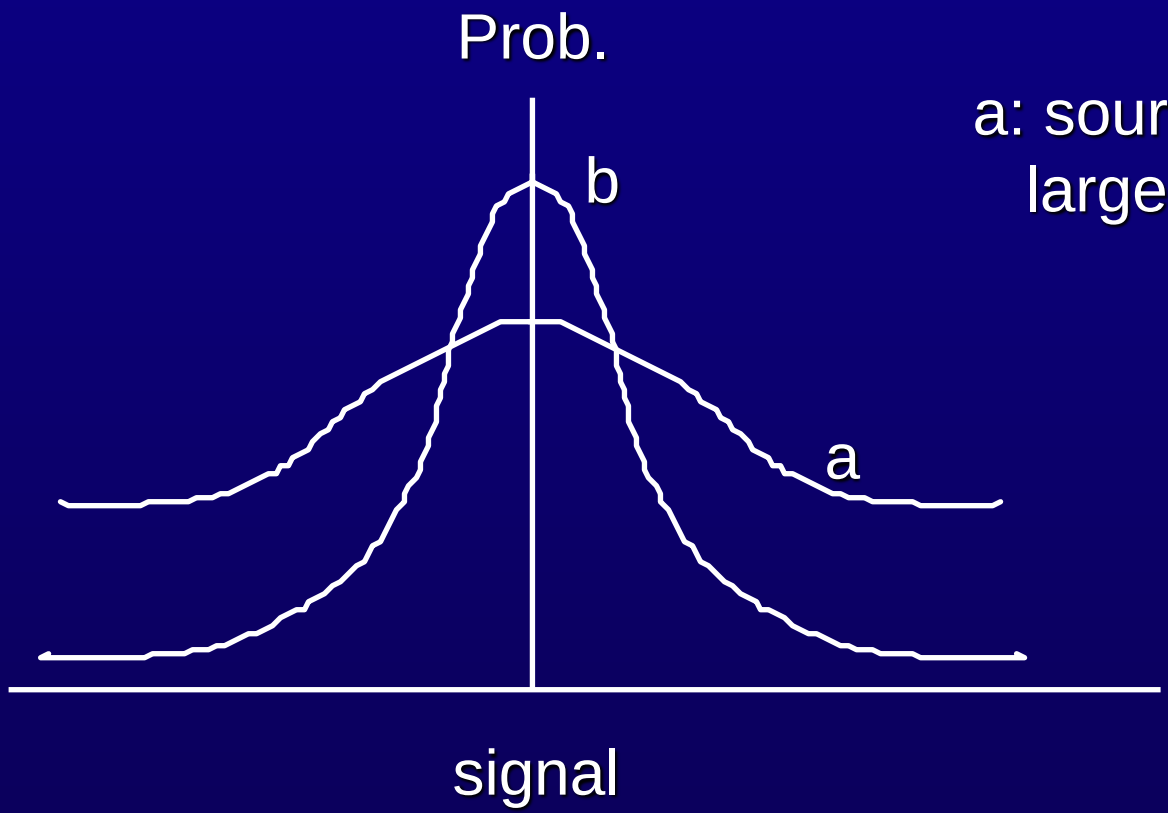


Histogram of the original "Lena" image



Histogram of the DPCM residual signal of "Lena"

Source-Model Mismatch



a: source, b: model with large degradation

Some Research Activities

- Re-synchronization after transmission errors
- Reversible Huffman code
- High-speed implementation
- Multiplication-free arithmetic coding
- Adaptive entropy coding
- Higher-order entropy coding