Source Coding

Youlong Wu

ShanghaiTech University

wuyl1@shanghaitech.edu.cn

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —		
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —

Layering of Source Coding

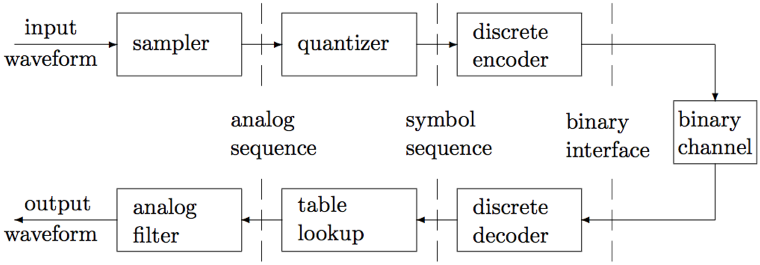


Figure: Split the source coding/decoding layer into three layers [Gallagar'Book]

- Discrete sources requires only the inner layer above
- Analog sequences use the inner two layers
- Waveform sources use all three layers

Discrete Memoryless Sources (DMS)

- The source output is an unending sequence $X_1, X_2, \ldots$ of randomly selected letter from a finite set $\mathcal{X}$, called the source alphabet.
- Each source output $X_1, X_2, \ldots$ is selected from $\mathcal{X}$ using the same P_X
- Each source output X_k is statistically independent of any other outputs X_j, for all $j \neq k$.

$$\boxed{X_1, X_2, \ldots \text{ are i.i.d according to } P_X}$$

Discrete Source Coding

Definition: Given a RV $X \sim P(x)$, map each $x \in \mathcal{X}$ to a finite-length sequence of symbols from a D-ary alphabet. (Normally, $D = 2$)

Let $c(x)$ denote the codeword for x

Example: Alphabet $\mathcal{X} = \{a, b, c, d, e, f\}$, $D = 3$

$$
\begin{aligned}
a &\longrightarrow c(a) = 000 \\
b &\longrightarrow c(b) = 111 \\
c &\longrightarrow c(c) = 222 \\
e &\longrightarrow c(e) = 012 \\
f &\longrightarrow c(f) = 210
\end{aligned}
$$

Code Length for DMS

- Let $l(x)$ be the length of the codeword $c(x)$.
- Then $L(x)$ is a RV where $L(x) = l(x)$ for $X = x$.
  (Note: *Capital letter for RV, and small letter for value*)
- The probability corresponding to $L(X) = l(x)$ is $p_X(x)$.
- The expected length of codeword
  (physical meaning: the number of encoder output bits per source symbol)

$$\boxed{E(L) = \bar{L} = \sum_x p_X(x) l(x)}$$

Discrete Source Coding

Simplest Approach: Map each source symbols into L-tuple of binary digits.

For an alphabet size of M, require $2^L \geq M$.

To avoid wasting bits, choose L as smallest integer satisfying $2^L \geq M$, i.e.,

$$L = \lceil \log_2 M \rceil \longrightarrow \log_2 M \leq L < \log_2 M + 1$$

(hint: Given a binary sequence of length L, there are 2^L different combinations of vlaues)

Fixed length code

Example: Alphabet $\mathcal{X} = \{a, b, c, d, e, f\}$.

$$
\begin{aligned}
a &\longrightarrow 000 \\
b &\longrightarrow 001 \\
c &\longrightarrow 010 \\
e &\longrightarrow 011 \\
f &\longrightarrow 100
\end{aligned}
$$

- $M = 6$, $L = \lceil \log_2 M \rceil = 3$
- Uniquely decoded
- Examples: acii code/GBK/GB2312. Maps letters, numbers, etc. into binary 8 bytes

ASCII code

- Maps letters, numbers, etc. into binary 8 bytes

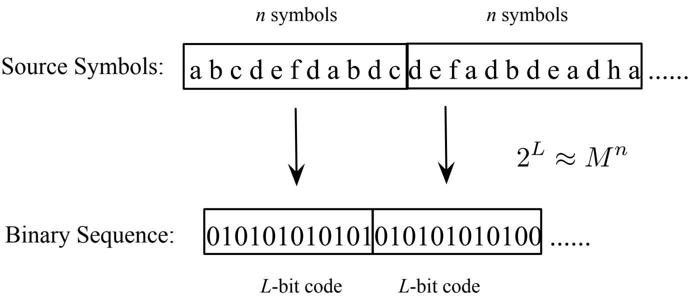
USASCII code chart

Bits				b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

More General Fixed Length Codes

Definition:

> Segment the sequence of source symbols into successive blocks of n source symbols at a time.



- Each source n-tuple is encoded into $L = \lceil \log_2 M^n \rceil$ bits.
- For each source symbol, $\bar{L} = L/n$: $\log_2 M \leq \bar{L} < \log_2 M + 1/n$

Variable Length Codes

Motivation: Fix length codes takes no account of whether some symbols occurs more frequently than others.

A variable-length source code $\mathcal{C}$ encodes each symbol $x \in \mathcal{X}$ to a binary codeword $\mathcal{C}(x)$ of length $l(x)$.

Example: Given $\mathcal{X} = \{a, b, c\}$

$$a \longrightarrow 0$$
$$b \longrightarrow 10$$
$$c \longrightarrow 11$$

Here, $l(a) = 1, l(b) = 2$ and $l(c) = 2$.

Problem:

- No commas, how to parse the received sequence: 01011... (A joke)
- Need bigger buffer

Variable Length Codes

Uniquely decodable:
Example: Given $\mathcal{X} = \{a, b, c\}$

$$a \longrightarrow 0$$
$$b \longrightarrow 10$$
$$c \longrightarrow 11$$

non-uniquely decodable:
Example: Given $\mathcal{X} = \{a, b, c\}$

$$a \longrightarrow 0$$
$$b \longrightarrow 1$$
$$c \longrightarrow 10$$

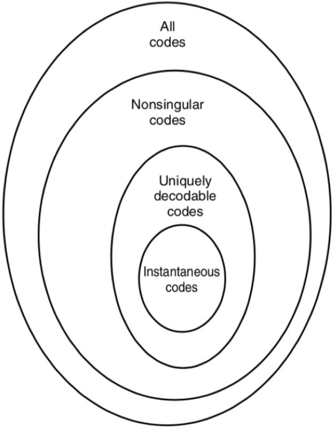
Received sequence 010110, how to decode?

Classes of Codes

X	Singular	Nonsingular, But Not Uniquely Decodable	Uniquely Decodable, But Not Instantaneous	Instantaneous
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

- Nonsingular: $x \neq x' \Rightarrow c(x) \neq c(x')$
- instantaneous code or prefix-free code

Classes of Codes (continue)



Prefix-free codes

- A code is prefix-free if no codeword is a prefix of any other codeword.
- A prefix-free code can be presented as binary code tree which grow from a root on the left to leaves on the right representing the codewords.

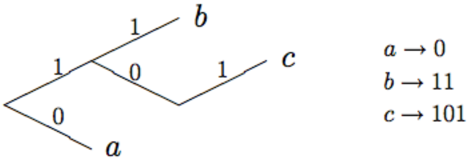
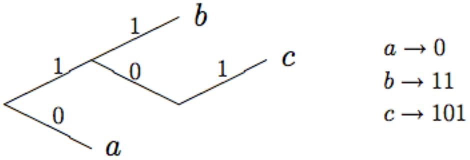


Figure: Binary code tree [Gallagar'Book]

- Every codeword is at a leaf, but not all leaves are codewords. Empty leaves can be shortened.
- A prefix-free code is **full** if no new codeword can be shorten.

Prefix-free codes

- Prefix-free codes are uniquely decodable.
- To decode, start at the left and parse whenever a leaf in the three is reached.



Received sequence is 110101, how to decode?

- Start at left: 1, this is not a leaf point
- keep going: 11, this is a leaf point for b
- keep going: 0, this is a leaf point for a

How to find a prefix-free code?

- The kraft inequality is a condition determining on the existence of prefix-free codes with a given set of codeword lengths $\{l(x), x \in \mathcal{X}\}$.

> **Theorem (Kraft)**: Every prefix-free code for an alphabet $\mathcal{X}$ with codeword lengths $\{l(x), x \in \mathcal{X}\}$ satisfies
>
> $$\sum_{x \in \mathcal{X}} 2^{-l(x)} \leq 1$$

- Conversely, if the inequality holds, then a prefix-free code with lengths $\{l(x)\}$ exists.

- A prefix-free code is full iff the equality above holds.
- Proof. (hint: binary-expansion)

A Small Example

Consider a alphabet $\{a, b, c, d\}$ with prefix-free codewords $\{0, 10, 110, 111\}$.

- If the symbol probabilities are $\{1/4, 1/4, 1/4, 1/4\}$, then the expected length is

$$1/4 * 1 + 1/4 * 2 + 1/4 * 3 + 1/4 * 3 = 2.25$$

- If the symbol probabilities are $\{1/2, 1/4, 1/8, 1/8\}$, then the expected length is

$$1/2 * 1 + 1/4 * 2 + 1/8 * 3 + 1/8 * 3 = 1.75$$

Motivation: Choose code which has shortest expected code length.

Prefix-free Codes for DMS(skip)

Objective: Choose $\{l(x)\}$ to minimize $\bar{L}$

- Recall that for prefix-free code: $\sum_{x\in\mathcal{X}} 2^{-l(x)} \leq 1$.
- Use *Lagrange* algorithm to find the optimal $\{l(x)\}$.

Solution: Assume $\mathcal{X} = \{a_1, a_2, \ldots, a_M\}$ with pmf $p_1, \ldots, p_M$, and denote the corresponding lengths by $l_1, \ldots, l_M$.

$$\bar{L}_{\mathsf{min}} = \min_{l_1,\ldots,l_M} \sum_i p_i l_i$$

$$\text{subject to : } \sum_i 2^{-l_i} \leq 1$$

- Let $\frac{\partial(\sum_i p_i l_i + \lambda 2^{-l_i})}{\partial l_i} = 0 \Longrightarrow p_i - \lambda(\ln 2)2^{-l_i} = 0$, ($l_i$ is a function of λ).
- Choose λ such that $\sum_i 2^{-l_i} = 1 \Longrightarrow l_i = -\log_2 p_i$.
- $\bar{L}_{\mathsf{min}} = \sum_i p_i l_i = -\sum_i p_i \log_2 p_i$ ($\bar{L}_{\mathsf{min}}$ might not be integer)

Entropy bound on Prefix-free Codes for DMS

$$\boxed{\bar{L}_{\text{min}}(\text{non-int.}) = \sum_i p_i l_i = \sum_i p_i \log_2 p_i = H(X)}$$

- The proof using Lagrange algorithms doesn't consider the L as integer
- $\bar{L}_{\min} = H(X)$ iff each p_i is integer power of 2

Theorem: Given a DMS X, the minimum expected codeword length for all prefix-free code satisfies

$$H(X) \leq \bar{L}_{\min} < H(X) + 1$$

Proof: see page 29 in Gallager's book.

Proof of $H(X) \geq \bar{L}_{\text{min}}$

For any code with average code length $\bar{L}$

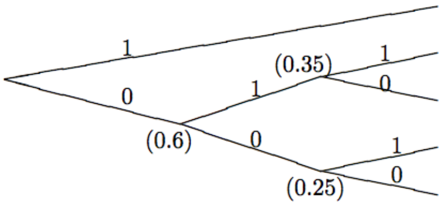
$$
\begin{aligned}
H(x) - \bar{L} &= \sum_{j=1}^{M} p_j \log \frac{1}{p_j} - \sum_{j=1}^{M} p_j l_j \\
&= \sum_{j=1}^{M} p_j \log \left(\frac{2^{-l_j}}{p_j} \right) \\
&\overset{(a)}{\leq} \log e \sum_{j=1}^{M} p_j \left(\frac{2^{-l_j}}{p_j} - 1 \right) \\
&= \log e \left(\sum_{j=1}^{M} 2^{-l_j} - \sum_{j=1}^{M} p_j \right) \leq 0
\end{aligned}
$$

where (a) holds by $\log u \leq (\log e)(u-1)$; the last equality holds by Kraft's inequality.

Huffman Coding

- Above theorem suggested that good codes have length $l_i \approx \log(1/p_i)$ (Note: Many researchers trying to find code using this way, but it turns out to work not well!!)

- To make $\bar{L}_{\min}$ small, if $p_i > p_j$, then $l_i \leq l_j$. Huffman used this simple idea to construct the code:



p_j	symbol	codeword
0.4	1	1
0.2	2	011
0.15	3	010
0.15	4	001
0.1	5	000

Figure: Huffman code [Gallagar'Book]

Problem 1: Huffman Code

Problem: Given a DMS $X \in \{a_1, a_2, \ldots, a_7\}$, with probability $\{0.35, 0.30, 0.20, 0.10, 0.04, 0.005, 0.005\}$.

- Design a Huffman code for this source
- Find $\bar{L}$, average codeword length
- Determine the efficiency of the code $\eta = \frac{H(X)}{\bar{L}}$

Solution:

Problem 1: Huffman Code

Problem: Given a DMS $X \in \{a_1, a_2, \ldots, a_7\}$, with probability $\{0.35, 0.30, 0.20, 0.10, 0.04, 0.005, 0.005\}$.

- Design a Huffman code for this source
- Find $\bar{L}$, average codeword length
- Determine the efficiency of the code $\eta = \frac{H(X)}{\bar{L}}$

Solution:

Letter	Probability	Self-information	Code
x_1	0.35	1.5146	00
x_2	0.30	1.7370	01
x_3	0.20	2.3219	10
x_4	0.10	3.3219	110
x_5	0.04	4.6439	1110
x_6	0.005	7.6439	11110
x_7	0.005	7.6439	11111
	$H(X) = 2.11$	$\bar{R} = 2.21$	

Game time

Optimality of A Source Code (skip)

Assume $\mathcal{X} = \{a_1, a_2, \ldots, a_M\}$ with pmf $p_1, \ldots, p_M$. Any optimal source code should satisfy

- If $p_i > p_j$, then $l_i \leq l_j$
- Optimal prefix-free code has a full tree
- For the longest code-words, its sibling is another longest codeword

Optimality of Huffman Code (skip)

Huffman is optimal for symbol-to-symbol coding with a known input probability distribution.

Proof:

- Huffman algorithm chooses an optimal code tree by starting with two least likely symbols, specifically M and $M-1$.
- Let X' be the reduced RV from X (Combining the two smallest probability symbols)
- Let $\bar{L}'$ be the expected length of X'. Then the optimal L satisfies

$$\bar{L} = \bar{L}' + p_{M-1} + p_M$$

  (Extending the codeword $\mathcal{C}'(M-1)$ in to two sibling for $M-1$ and M)
- $\bar{L}_{\min} = \bar{L}'_{\min} + p_{M-1} + p_M$
- Using Huffman algorithm, an optimal code for X' yields an optimal code for X. Prove X'' to X' and so forth, down to a binary symbol.

Huffman Code for Encoding a Block

Encode blocks of n symbols a a time instead of symbol-by-symbol. The average number of bits per n-symbol

$$nH(X) \leq \bar{L}_n < nH(X) + 1$$

Thus, $H(X) \leq \bar{L} < H(X) + \frac{1}{n}$, $\bar{L}$ goes to $H(X)$ as n goes into ∞.

Proof:

- $H(X^n) = H(X_1, \ldots, X_n) = nH(X)$, with X_i i.i.d $\sim P_x$
- Take $X^n \in \mathcal{X}^n$ as a big "source RV"
- By $H(X) \leq \bar{L}_{\text{min}} < H(X) + 1$, we have

$$nH(X) = H(X^n) \leq \bar{L}(X^n)_{\min} < H(X^n) + 1 = nH(X) + 1$$

- $nH(X) \leq \bar{L}_n < nH(X) + 1$

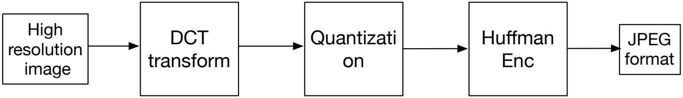
Huffman Code for Encoding a Block

Example: A DMS $\{x_1, x_2, x_3\}$ has probability {0.45,0.35,0.20}.

Letter pair	Probability	Self-information	Code
x_1x_1	0.2025	2.312	10
x_1x_2	0.1575	2.676	001
x_2x_1	0.1575	2.676	010
x_2x_2	0.1225	3.039	011
x_1x_3	0.09	3.486	111
x_3x_1	0.09	3.486	0000
x_2x_3	0.07	3.850	0001
x_3x_2	0.07	3.850	1100
x_3x_3	0.04	4.660	1101

- Using symbol-to-symbol Huffman code: $\bar{L} = 1.55$, $\eta = 97.6\%$
- Using Block Huffman code:
  - $\bar{L}_2 = 3.067$, $\bar{L} = \bar{L}_2/2 = 1.534 < 1.55$
  - $\eta = 98.6\% > 97.6\%$

Application of Huffman Code



- Discrete cosine transform (widely used in video and audio compression)

$$y = Cx$$

where x is the input image, C is an $n \times n$ transformation matrix

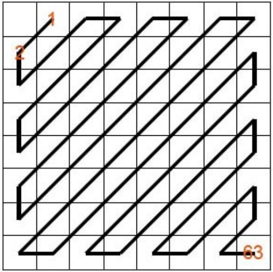
$$C = \sqrt{\frac{2}{n}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{2n} & \cos\frac{3\pi}{2n} & \cdots & \cos\frac{(2n-1)\pi}{2n} \\ \cos\frac{2\pi}{2n} & \cos\frac{6\pi}{2n} & \cdots & \cos\frac{2(2n-1)\pi}{2n} \\ \vdots & \vdots & & \vdots \\ \cos\frac{(n-1)\pi}{2n} & \cos\frac{(n-1)3\pi}{2n} & \cdots & \cos\frac{(n-1)(2n-1)\pi}{2n} \end{bmatrix}$$

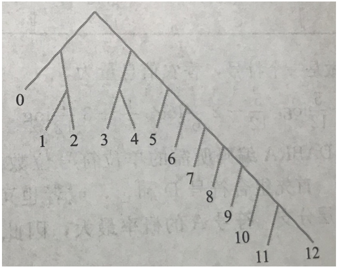
DCT: keep low frequency values and non-zero values gather in upper left

- Quantization: $z = \text{round}(\frac{y}{q})$, inverse Quantization: $\bar{y} = qz$

- Huffman-based encode

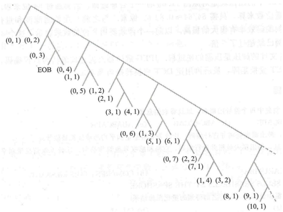
Huffman Code for JPEG





- In each block, the DC part y_{00} uses a DPCM Huffman tree
- Encode the difference between blocks
- the rest 63 AC terms uses run length encoding (RLC) with another Huffman tree and integer tabel

Huffman Code for JPEG



L	项	二进制编码
0	0	– –
1	−1, 1	0, 1
2	−3, −2, 2, 3	00, 01, 10, 11
3	−7, −6, −5, −4, 4, 5, 6, 7	000, 001, 010, 011, 100, 101, 110, 111
4	−15, −14, …, −8, 8, …, 14, 15	0000, 0001, …, 0111, 1000, …, 1110, 1111
5	−31, −30, …, −16, 16, …, 30, 31	00000, 00001, …, 01111, 10000, …, 11110, 11111
6	−63, −62, …, −32, 32, …, 62, 63	000000, 000001, …, 011111, 100000, …, 111110, 11111
⋮	⋮	⋮

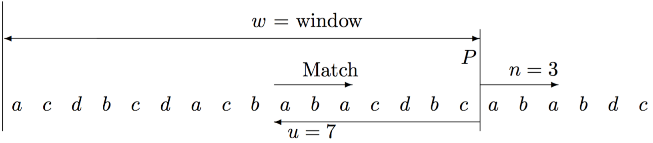
AC sequence -5,0,0,0,2

- present by $(0,3)-5(3,2)EOB$, where (n,L) means after n number of 0, there is a nonzero value with size $L=\lfloor \log_2|y|+1\rfloor$
- (0,3) after 0 number of 0, there is a value of size 3, its value is -5
- (0,3) in Huffman tree is 100, -5 in the integer table is 010,
- EOB (end of block) in tree is 1010 item -5,0,0,0,2 is encoded as (100)(010)(11111011)(1010)

Lempel-Ziv Data Compression

- Don't require prior knowledge of the source statistics
- Adapt to minimize average codelength $\bar{L}$
- Effectively optimal
- Widely used in practice (gif format)

Lempel-Ziv Data Compression



Set window size w

1. Encode the first w symbols in a fixed length code, without compression

2. Set pointer $P = w$

3. Find the largest $n \geq 2$ such that $x_{P+1}^{P+n} = x_{P+1-u}^{P+n-u}$ for some $u \in [1, w]$. x_{P+1}^{P+n} is encoded by encoding n and u (p. 53)
   - Encode n into a codeword from the unary-binary code
   - Encode $u \leq w$ using fixed-length code of length $\log w$

4. Set the pointer P to $P + n$ and go to step (3). Iterate forever

Lempel-Ziv Data Compression

Unary-binary code (prefix-free)

n	prefix	base 2 expansion	codeword
1		1	1
2	0	10	010
3	0	11	011
4	00	100	00100
5	00	101	00101
6	00	110	00110
7	00	111	00111
8	000	1000	0001000

Lossless Source Coding Theorem

Shannon's First Theorem: Let X denote a DMS with entropy $H(X)$, there exists a lossless source code for this source at any rate $R > H(X)$. There exists no lossless code for this source at rates less than $H(X)$.

Achievability:
Recall prefix-free code: Given a DMS X, the minimum expected codeword length for all prefix-free code satisfies

$$H(X) \leq \bar{L}_{\min} < H(X) + 1$$

Converse: If $R < H(X)$, then the error probability approaches 1 for large n. See p. 44 [Gallager'44].

Understanding by AEP: "Typical" sequence $\rightarrow$ "Typical" set.

Summary: Discrete Source Coding(skip)

Lossless Source Coding (Shannon's **First** Theorem)

- Simple fix-length code (ignore the source distribution)
- Variable-length code (parse problem$\rightarrow$ prefix-free code $\rightarrow$ Kraft-inequality$\rightarrow$Huffman-code)
- Lemp-Ziv code
- Lossless Source Coding Theorem: $R > H(X)$ is sufficient to decode X
- Source coding for Markov source (unimportant)

Lossy Source Coding (Shannon's **Third** Theorem)

- Rate Distortion Function (Definition and how to calculate)
- Lossy Source Coding: $R > R(D)$ is sufficient to decode X