Overview of Compression

Dr. Zein Al Abidin IBRAHIM

ibrahim.zein@gmail.com

IN433
Multimedia Processing

Motivation (1)

- Characteristics of digital multimedia information produced today:
  - text, video, audio, 3D graphics, and combinations of these media types.
  - extraordinarily large, and
  - the rate of creation increases every day.
- The result → a growing mass of data needs to be:
  - stored,
  - accessed, and
  - delivered to a multitude of clients over digital **networks**, which have varying bandwidths.
    - fiber-optic networks can provide a bandwidth upwards of 1G bits/second,
    - cell phone networks, are limited to a range of 20–300 kbps.
    - 3G cell phone standard provides transmission bandwidths up to 5 Mbps.

Motivation (2)

- For thus, people are expecting to receive multimedia data in the form of video, audio, and graphics on a variety of devices,
  - televisions, set-top boxes, desktop computers, handheld portable devices such as PDAs and cell phones, ….
- The existence of voluminous data, from creation to storage and delivery, motivates the need for compression.

Need of compression

storage and transmission requirements

Need for compression

- In the early days of information age:
  - disk space was limited and expensive,
  - bandwidth was limited.
  - → Compression was a must
- Today:
  - Inexpensive storage in a variety of forms
  - high-bandwidth connections (DSL, fiber optics and T-1).
  - Compression is a must ??????
  - Absolutely yes and there is more need because:
    - amount and type of information we consume has also increased many fold, with the use of images, audio, video, and graphical animations.
    - modern applications, such as high-definition video, require even larger bandwidths.

- Example of HDTV:
  - The interlaced HDTV (1080i) format has a frame size of 1920 x 1080.
  - With YUV 4:2:0 subsampling → pixel = 12 bits.
  - Bandwidth = 1080 x 1920 x 12 x 30 = 745 Mb/s.
    - Home network connections do not support this type of bandwidth and
    - the digital signal will need to be compressed for delivery.
  - HDTV 1080i is considered to be acceptable when compressed down to 18 Mb/s.

Need for compression

- Example of most popular digital cameras:
  - Produce up to 14-megapixel images and increasing fastly.
  - We take lot of pictures than ever.
  - The storage needs are very demanding, and
  - the information contained in one such uncompressed image takes a long time to write to disk.
  - Therefore, captured images are compressed prior to writing them to the memory devices.

Need for compression

- We look at ***data*** as ***containing information***.
- ***Compression → a science that reduces the amount of data used to convey the same information.***
  - relies on the fact that information, such as an image or video, is not a random collection of pixels but exhibits order and patterns.
  - If this coherence can be understood or even modeled, the information can often be represented and stored/transmitted with a lesser amount of data.

Need for compression

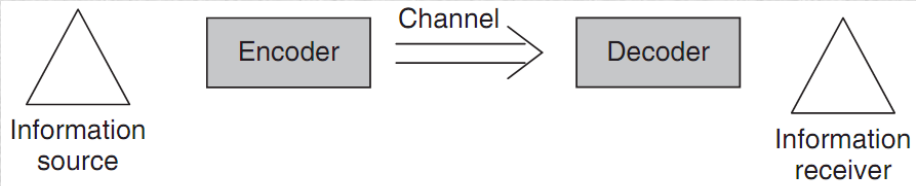
Basic of information theory

- In the information theory → efficiency and reliability of the transmission.
  - **Efficient transmission**—How efficiently can a source transmit information to a receiver over a given channel and, thereby, communicate the information in the least amount of bits?
    - Related to the compression.
    - Shannon proposed his model of encoding/decoding of transmitted information
    - Named **source coding**.

- In the information theory → efficiency and reliability of the transmission.
  - **Reliable transmission**—How reliably can a source transmit information to a receiver over a given channel, knowing that the channel is noisy and can corrupt the information during transmission.
    - ensuring reliable delivery of data across a noisy channel.
    - → add redundant bits to the coded symbols to be able to detect transmission errors and even correct them.
    - Named **channel coding**.

Basic of information theory

- To compress information, we need:
  - to formalize what information means, and
  - have a metric to measure the amount of information a sequence of bits and bytes contains.
  - Then, refer to the *information* theory to describe the process to compress data without losing its *information content*.
- Data may be of different types, such as simple text, documents, images, graphics, and even binary executables.
- We treat here all generic data represented digitally in a binary form by a sequence of 1s and 0s.



Shannon's communication model. Information produced by a source is encoded in binary form and decoded at the receiver.

Basic of information theory

- To understand compression, → necessary to understand the information content of a message.
  - data is organized as a sequence of symbols.
  - If the sequence changes, so does the information.
  - Ex: a binary data stream that consists of 80,000 bits.
  - bits might not need to be treated individually, but might instead be grouped into symbols.
  - If we know that bits represent a gray-intensity image with each pixel represented by 8 bits, → so we have 10,000 pixels or symbols.
  - if width and height of the image are both 100
    - the bit stream might be interpreted as a gray image of size 100 x100 and
    - each symbol is represented by 8 bits, and there can be $2^8 = 256$ different possible gray levels or symbols.
    - Also, the arrangement of the symbols is important.
- image representing a scene carries more information than a black image, where all the pixel or symbols are zero.

Information theory definitions

Alphabet and Symbols

- *Information* → organization of individual elements called *symbols*.
- *alphabet* → a distinct and nonempty set of *symbols*.
- number or length of *symbols* in the set is known as the *vocabulary*.
- We can define an *alphabet* of *symbols* as S = $\{s_1, s_2, s_3, s_4 \ldots s_n\}$.
  - n can be very large,
  - in practice, an *alphabet* is limited and finite and, hence, has a well-defined *vocabulary*.

Information theory definitions

Information theory definitions

- Ex1: A grayscale image, pixel coded on 8 bits.
  - vocabulary consists of 256 symbols ({0,….., 255}).
  - Each symbol is represented or coded by 8 bits.
- Ex2: English alphabet has 26 unique letters
  - vocabulary of 26 symbols {a, ….}.
  - each symbol will need 5 bits ($\log_2 26 = 4.7$ or simply 2^5 is the nearest greater power 2 to 26).
- Ex3: An RGB color image, pixel coded on 3x8=24 bits per symbol.
  - vocabulary consists of $2^8 x 2^8 x 2^8$=16,777,216 symbols (0 to 16,777,215)
  - or nearly 17 million colors.

- *Sequence* = series of symbols of a given alphabet.
  - Exp: For the alphabet $\{s_1, s_2, s_3, s_4\}$, a sample sequence is $s_1s_2s_1s_2s_2s_2s_1s_2s_3s_4s_1s_2s_3s_3$.
- *Message* = A sequence of symbols produced by the source using the alphabet, and represents information.
  - A change in the sequence corresponds to
    - a change in the message and,
    - thereby, a change in the information.
  - Each symbol has a binary representation and, hence, a message translates digitally to a bit stream.
  - An image, a video, and text are all examples of binary messages.

Sequence

- In a sequence → some symbols might occur more commonly than other symbols.
  - symbol s_1 occurs more frequently than the others in the previous example.
  - the letter e occurs more frequently than the other letters in the English language.
- frequency of occurrence of a symbol is an important factor when coding information represented by the symbols.
  - known as **probability**.

Information theory definitions

Symbol probability

- **Probability or likelihood** of a symbol = average number of times a symbol occurs in the entire message.
- → Ratio of the number of occurrences of that symbol over the length of the entire message
  - $P_i = \frac{m_i}{N}$

m_i = number of times the symbol i is present in the message

N = length of the message

- Used to obtain optimal codes for compression.

Information theory definitions

Symbol probability

- Ex: Let the alphabet be $\{s_1, s_2, s_3, s_4\}$
  - a message of 100 symbols, s_1 occurs 70 times, s_2 occurs 5 times, s_3 occurs 20 times, and s_4 occurs 5 times.
  - Each symbol is coded on 2 bits → message of 200 bits
  - Why not to give s_1 short code as it occurs more than the others?
  - Idea: codes depends on the probability of occurrences of symbols.

Information theory definitions

Symbol	Code	Number of occurrences (probability)	Bits used by each symbol	
s_1	00	70 (0.7)	$70 \times 2 = 140$	
s_2	01	5 (0.05)	$5 \times 2 = 10$	200
s_3	10	20 (0.2)	$20 \times 2 = 40$	
s_4	11	5 (0.05)	$5 \times 2 = 10$	

Symbol	Code	Number of occurrences (probability)	Bits used by each symbol	
s_1	1	70 (0.7)	$70 \times 1 = 70$	
s_2	001	5 (0.05)	$5 \times 3 = 15$	140
s_3	01	20 (0.2)	$20 \times 2 = 40$	
s_4	000	5 (0.05)	$5 \times 3 = 15$	

Symbol	Code	Number of occurrences (probability)	Bits used by each symbol	
s_1	0	70 (0.7)	$70 \times 1 = 70$	
s_2	01	5 (0.05)	$5 \times 2 = 10$	110
s_3	1	20 (0.2)	$20 \times 1 = 20$	
s_4	10	5 (0.05)	$5 \times 2 = 10$	

The left column of tables shows different code assignments that can be used for symbols s_1, s_2, s_3, and s_4 in the same message. The top table has equal code length for all symbols, whereas the bottom two tables have variable-length codes. The boxes on the right show the total number of bits used for the 100 symbol message. The code representation affects the overall bits used. The first two tables have uniquely decodable codes, but the third does not.

Information theory definitions

- Decodable codes → resulting bit stream can be uniquely decoded by the decoder to reconstruct the original symbol sequence.
- Problem: In the third table of the previous slide:
  - the message "$s_1\ s_1\ s_2\ s_3$" is mapped to a binary sequence "00011".
  - can be decoded as "$s_1\ s_1\ s_2\ s_3$" or as "$s_1\ s_1\ s_1\ s_3\ s_3$".
  - table represents a code that is not uniquely decodable.
- Solution: Prefix codes
  - A prefix code is a special kind of uniquely decodable code
  - no one code is a prefix of another code.
  - Ex: C={(s_1, 1), (s_2, 001), (s_3, 01), (s_4, 000)} is a prefix code.

Symbol Probability

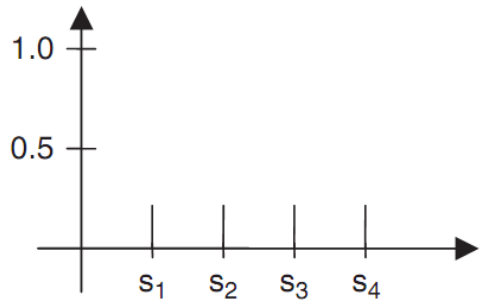
- Let a source has a vocabulary of N symbols $\{s_1, s_2, \ldots, s_N\}$.
  - Produces a message consisting of M symbols,
  - Symbols are emitted at frequency of 1/T Hz, which is one symbol every T seconds.
  - Let each symbol s_i be emitted m_i times.
  - The frequency of s_i in the message of M symbols is m_i.
  - Let l_i = the length of each symbol s_i.
- $M = \sum_{i=1}^{N} m_i$ is the length of the message.
- $L = \sum_{i=1}^{N} m_i l_i$ is the total number of bits of the message.
- $\lambda = \frac{L}{M}$ is average length by symbol.
- $\lambda = \frac{\sum_{i=1}^{N} m_i l_i}{M} = \sum_{i=1}^{N} (\frac{m_i}{M}) l_i = \sum_{i=1}^{N} P_i l_i$: P_i is probability of occurrences of s_i

Aim of compression in the information processing domain

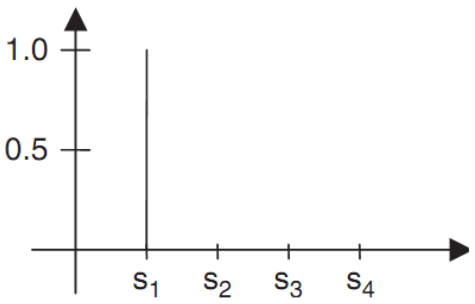
- Let S = $\{s_i\}$ a set of symbols having probabilities P_i for each i in an alphabet.
- Goal → finding a binary prefixed code C=$\{(s_1, c_1), (s_2, c_2) \ldots (s_n, c_n)\}$ where each code word c_i has code length l_i such that the average code length λ is minimized.
- It's attempting to find a C, such that the average number of bits required per symbol is the minimum.
  - When P_i increases, l_i should decrease and vice versa.
- Logical question: How low can the value of λ be?
- This was proved by Shannon to be measured by the ***entropy***.

Entropy

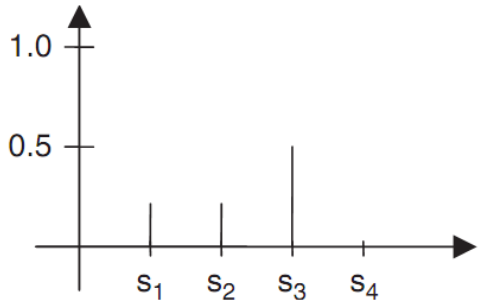
- **Entropy** is to quantify the amount of information contained in a message of symbols given their probabilities of occurrence.
- $H = \sum P_i \log_2 \frac{1}{P_i} = -\sum P_i \log_2 P_i$ where $\log_2 \frac{1}{P_i}$ is called self-information.
- Self-information = # of bits of information contained in the symbol
  - It means the number of bits used to send that message.
  - When probability of s_i is high, the self-similarity is low.
- Entropy, then, becomes the weighted average of the information carried by each symbol and, hence, the average symbol length.
- H has highest values when the source produces symbols that are equally probable.



$$P_i = \{0.25,\ 0.25,\ 0.25,\ 0.25\}$$
$$H = -(4 \times 0.25 \times \log_2 0.25)$$
$$H = 2$$

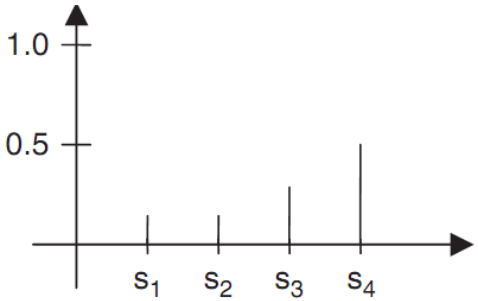


$$P_i = \{1.0,\ 0.0,\ 0.0,\ 0.0\}$$
$$H = -(1 \times \log_2 1)$$
$$H = 0$$



$$P_i = \{0.25\ \ 0.25\ \ 0.5\ \ 0.0\}$$
$$H = -(2 \times 0.25 \times \log_2 0.25 + 0.5 \times \log_2 0.5)$$
$$H = 1.5$$

$$P_i = \{0.125\ \ 0.125\ \ 0.25\ \ 0.5\}$$
$$H = -(2 \times 0.125 \times \log_2 0.125 + 0.25 \times \log_2 0.25 + 0.5 \times \log_2 0.5)$$
$$H = 1.75$$



Entropy. Four examples of probability distribution for the symbols {s_1, s_2, s_3, s_4} and the corresponding entropies are shown. The first distribution with equal probability assignments for all symbols has the highest entropy.

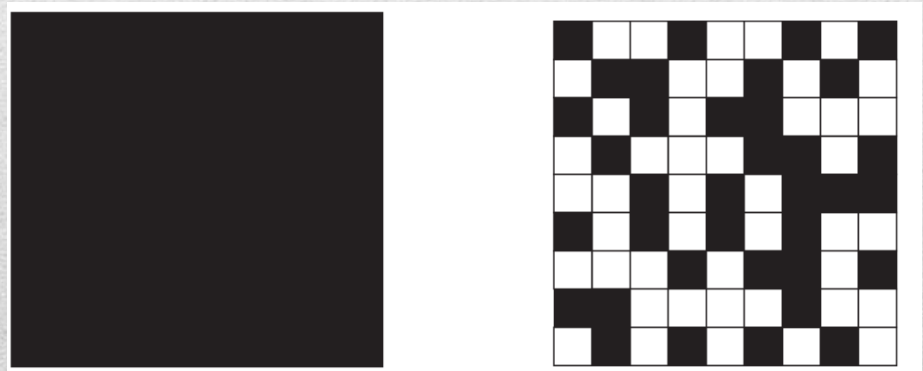
Information theory definitions

Entropy: Discussion

- When the *entropy* is higher (or the information content is higher), more bits per symbol are needed to encode the message.
- when symbols are not random but seem more predictable
  - Means some symbols have a higher probability than the others,
  - → results in a lower entropy, hence, in lower information content.
  - In this case, fewer bits per symbol are needed to encode the message.

- Two messages each message is a black-and-white image made up of symbols (s_1, s_2).
- The first message is a completely black image.
  - P1 is 1 and P2 is 0. H in this case evaluates to zero, suggesting that there is no information in the image.
- The second message has equal distribution of black and white pixels.
  - P1 and P2 are both roughly 0.5. H in this case evaluates to 1, which is exactly the minimum number of bits needed to represent the two symbols (s1 = 0, s2=1)

Example



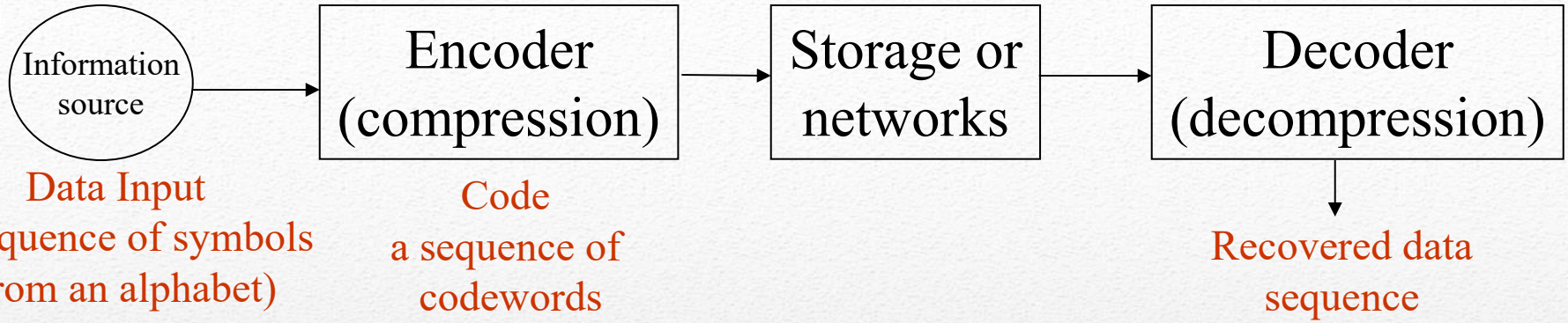
Two images are shown. The left image contains all black pixels and the information content is lower resulting in zero entropy. The right image contains roughly equal distribution of black and white pixels. The information content is higher in this case and the entropy evaluates to 1.

- Given an alphabet of symbols:
  - *Goal of any coding system*: transmit the information contained in a string composed of these symbols in the most efficient manner, that is, in the least number of bits.
  - *Way*: Evaluation of frequency of occurrences of each symbol + decide how many bits to code each symbol = the overall number of bits used for the message is the least.
- The optimal average symbol length is given by the entropy.
- Thus, any coding system's goal is to assign binary codes to each symbol in a manner so as to minimize the average symbol length.

$$Efficiency = \frac{Entropy}{Average\ Symbol\ Length}$$

- Means → the optimal average symbol length over the used one.
  - Example: if the optimal one is 4.5 and the used is 5 → 4.5/5 =0.9

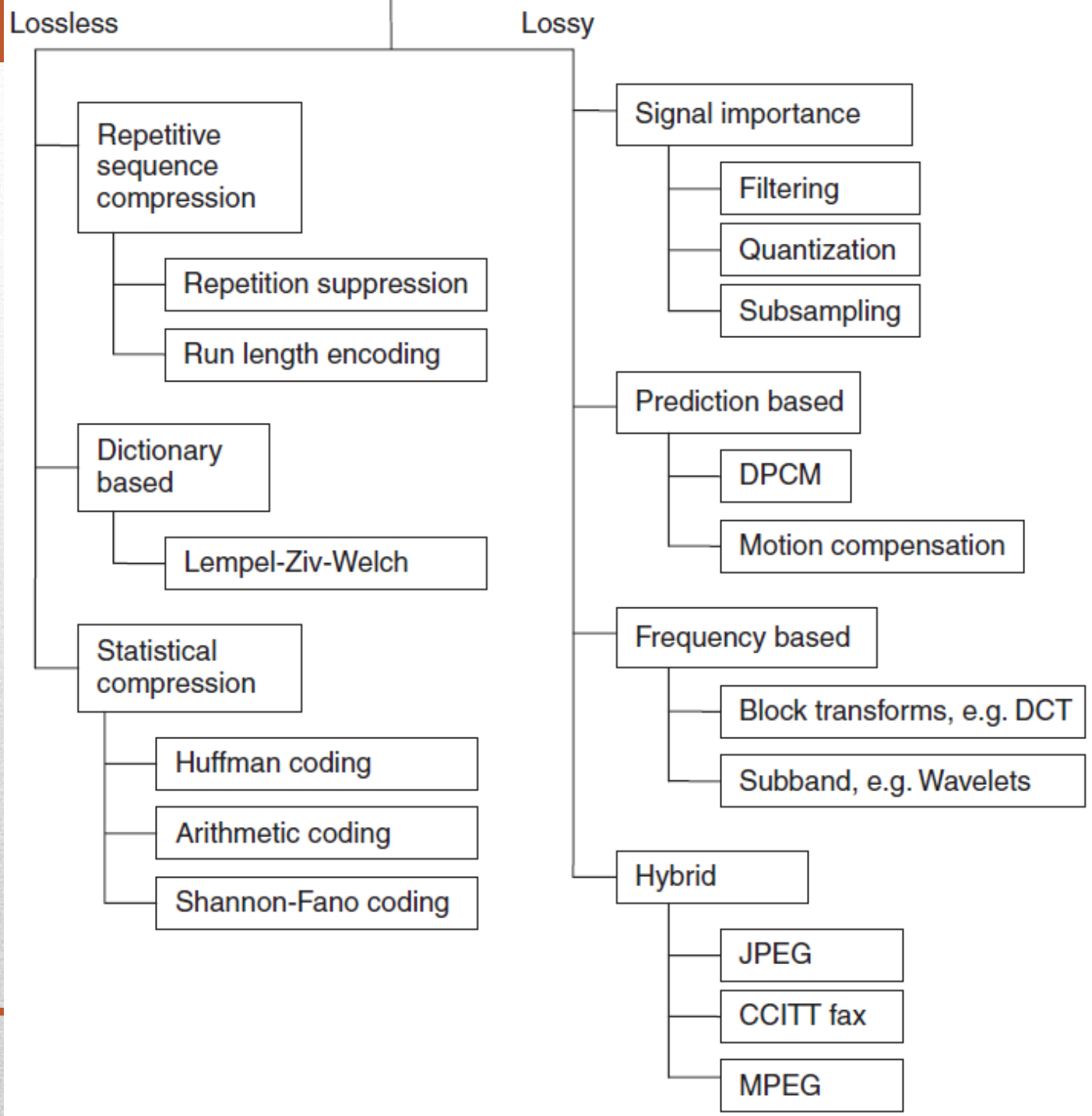
Efficiency



- Lossless compression: The recovered data is exactly the same as the input.
- Lossy compression: The recovered data approximates the input data.
- Compression ratio = (bits used to represent the input data) / (bits of the code after compression)
- We desire an encoder with compression ratio > 1.0

Some Terms

Overview of compression techniques



Compression Metrics

- **Compression rate**: related to the transmission of data
  - = rate of compressed data
  - Expressed Bits/symbol such as bits/sample, bits/pixel, bits/second
- **Compression ratio**: related to the storage of data
  - Rate of non-compressed to the compressed
  - Ex: audio signal: original = 16 bits/sample and compressed = 4 bits/sample
  - So 4-to-1 compression ratio.

- **Distortion rate:**
  - in lossy compression
  - Distortion criteria should be defined
  - Ex of simple criteria = difference between original and compressed one
  - On image → sum(difference pixel by pixel)
- Not all the time is really related to the distortion
- Ex: shift the image by one pixel → distortion is high while perceived image is the same
- Should use other types of criteria → may be subjective one

- **Distortion rate**:
  - Let y = original signal and $\hat{y}$ be the constructed one
  - $error = f(y - \hat{y})$ where f measures the difference
  - the most popular f is mean square error MSE
  - others → SNR (signal-to-noise-ratio) and PSNR (peak SNR)
    - More informative than mean square error
- lossy compression = trade-off between distortion and rate

Rate Distortion

- Lossless compression techniques → removing the redundancy in the signal.
  - Achieved by assigning new codes to the symbols based on the frequency of occurrence of the symbols in the message.
  - More frequent symbols are assigned shorter codes and vice versa.
- Sometimes the entire message to be coded is not readily available and frequency of symbols cannot be computed a priori.
  - happens when a source is "live," for example, in real-time communication systems.
  - In such cases, source symbols codes bases on a *probabilistic model*.

Lossless compression

- **Problem of probabilistic model**:
- In the English language e is the most frequently occurring letter—13% on average—and z is the least frequently occurring letter—0.07% on average.
- Sentence: “Zoos display zebras from Zambia, Zimbabwe, and Zaire,” contains more z than e.
- Lead to → not efficient compression, and sometimes might even do the opposite.

Lossless compression

- Rely on the repeated sequences or “runs” of the same symbol.
- Runs of the same symbol are encoded using two entities:
  - count suggesting the number of repeated symbols.
  - the symbol itself.
- Ex: a run of five symbols aaaaa will be replaced by 5a.
  - BBBBEEEEEEEECCCCDAAAAA → 4B8E4C1D5A.
  - 22-byte message reduced to 10 bytes, giving a compression ratio of 2.2.
- Problems:
  - will it work if the string contains numbers ?
    - Add a special character between two runs or specify a fixed number of bytes
  - Is it practical to encode runs of length 1 ? → no counts

Run Length Encoding RLE

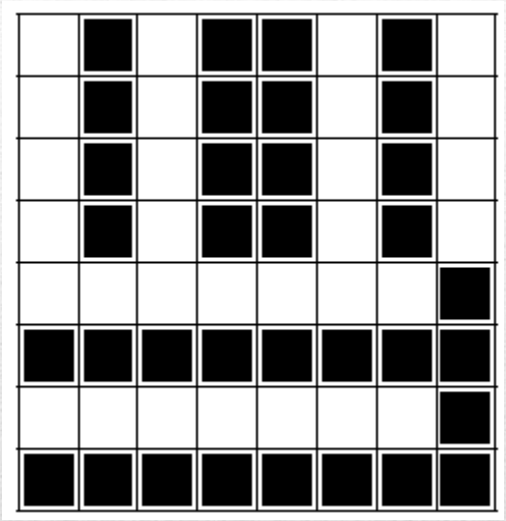
- Run length encoding performs best in the presence of repetitions or redundancy of symbols in the information.
- if no repetition or the seeming randomness in the symbols is high,
  - run length encoding is known not to give any compression and,
  - depending on implementation flavors, it could very well result in increasing the size of the message.
- used in a variety of tools involving text, audio, images, and video.

Run Length Encoding (RLE)

- For example, two people are having a telephone conversation:
  - Lot of gaps represented by zero values when nobody is speaking.
- For example, in images and videos:
  - have areas, which have the same pixel distribution, such as in the background.
- Run length encoding has also become a part of compression standards.
- it is supported by most bitmap file formats such as TIFF, BMP, and PCX and a variant of it is even adopted in the JPEG compression pipeline.

Run Length Encoding (RLE)

- Given the following binary image, what would be the RLE of it ?



Run Length Encoding (RLE)

- Works by reserving a specific symbol for a commonly occurring pattern in a message.
- For example, a specific series of successive occurrences of the letter a in the following example is replaced by the flag ψ:

Abdcbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

is replaced by

Abdcbψ

- Considered as an instance of run length encoding, where successive runs of specific characters are replaced.

Repetition Suppression

LZW CODING

Lossless Compression

- Based on the dynamic creation of a dictionary of strings.
  - Each string is a subsequence of symbols, as they appear in the message.
  - strings are given a code, or an index, which is used every time the string appears in the input message.
  - no pre-analysis of the message is required to create a dictionary of strings, but rather the strings are created as the message is scanned.

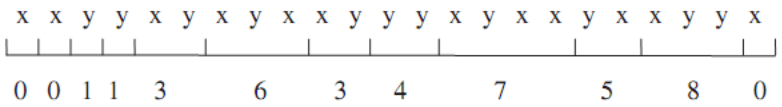
Pattern Substitution (LZW=Lempel, Ziv, Welch)

- Initialize the dictionary to contain all initial symbols. The vocabulary forms the initial dictionary entries. Every entry in the dictionary is given an index.
- 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.
- Encode E in the message by its index in the dictionary.
- Add a new entry to the dictionary, which is E followed by the next symbol in the scan.
- Repeat the process of using the dictionary indexes and adding until you reach the end of the message.

Pattern Substitution (LZW=Lempel, Ziv, Welch)

```
InputSymbol = Read Next input symbol;
String = InputSymbol;
WHILE InputSymbol exists DO
        InputSymbol = Read Next input symbol;
        IF  (String + InputSymbol) exists in DICTIONARY then
             String = String + InputSymbol
        ELSE
             Output dictionary index of String
             Add (String + InputSymbol) to DICTIONARY
             String = InputSymbol
       END // end of if statement
END // end of while loop
Output dictionary index of String
```

Pattern Substitution (LZW=Lempel, Ziv, Welch)



Iteration number	Current length symbol(s)	Dictionary code used	Next symbol in string	New dictionary code generated	New dictionary index generated
				x	0
				y	1
1	x	0	x	x x	2
2	x	0	y	x y	3
3	y	1	y	y y	4
4	y	1	x	y x	5
5	x y	3	x	x y x	6
6	x y x	6	x	x y x x	7
7	x y	3	y	x y y	8
8	y y	4	x	y y x	9
9	x y x x	7	y	x y x x y	10
10	y x	5	x	y x x	11
11	x y y	8	x	x y y x	12
12	x	0	END		

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 v x		

Dictionary-based encoding. The top illustration shows the input sequence and the dictionary indexes while encoding. The middle table depicts the step-by-step iteration while dictionary codes and indexes are generated. The bottom table shows the final set of indexes and dictionary code words.

LZW

- a b a b a b a b a
- b a c a c b a a c b
- a b a b c b a b a b a a a a a a a

Try

- Try the following sequences with the associated initial dictionary.

wabbaϕwabbaϕwabbaϕwabbaϕwooϕwooϕwoo

Initial LZW dictionary.

Index	Entry
1	*ϕ*
2	*a*
3	*b*
4	*o*
5	*w*

Try

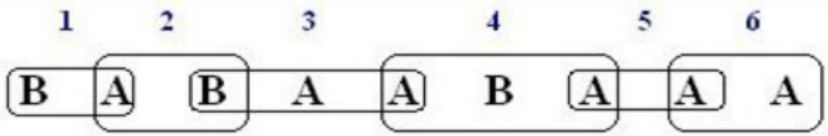
Index	Entry	Index	Entry
01	*ϕ*	14	*aϕ w*
02	*a*	15	*wabb*
03	*b*	16	*baϕ*
04	*o*	17	*ϕ wa*
05	*w*	18	*abb*
06	*wa*	19	*baϕ w*
07	*ab*	20	*wo*
08	*bb*	21	*oo*
09	*ba*	22	*oϕ*
10	*aϕ*	23	*ϕ wo*
11	*ϕ w*	24	*ooϕ*
12	*wab*	25	*ϕ woo*
13	*bba*		

The encoder output sequence is 5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4

Solution

Example 1: Compression using LZW

Encode the string **BABAABAAA** by the **LZW** encoding algorithm.



1. **BA** is not in the Dictionary; insert **BA**, output the code for its prefix: **code(B)**
2. **AB** is not in the Dictionary; insert **AB**, output the code for its prefix: **code(A)**
3. **BA** is in the Dictionary.
   BAA is not in Dictionary; insert **BAA**, output the code for its prefix: **code(BA)**
4. **AB** is in the Dictionary.
   ABA is not in the Dictionary; insert **ABA**, output the code for its prefix: **code(AB)**
5. **AA** is not in the Dictionary; insert **AA**, output the code for its prefix: **code(A)**
6. **AA** is in the Dictionary and it is the last pattern; output its code: **code(AA)**

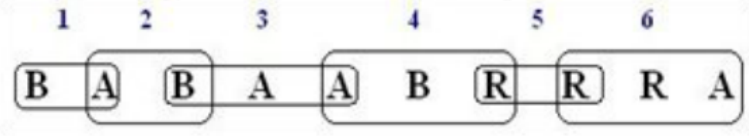
output	Dictionary	
	Code Word	String
66	256	BA
65	257	AB
256	258	BAA
257	259	ABA
65	260	AA
260		

The compressed message is: **<66><65><256><257><65><260>**

LZW

Example 2: Compression using LZW

Encode the string **BABAABRRRA** by the LZW encoding algorithm.



1. **BA** is not in the Dictionary; insert **BA**, output the code for its prefix: **code(B)**
2. **AB** is not in the Dictionary; insert **AB**, output the code for its prefix: **code(A)**
3. **BA** is in the Dictionary.
   BAA is not in Dictionary; insert **BAA**, output the code for its prefix: **code(BA)**
4. **AB** is in the Dictionary.
   ABR is not in the Dictionary; insert **ABR**, output the code for its prefix: **code(AB)**
5. **RR** is not in the Dictionary; insert **RR**, output the code for its prefix: **code(R)**
6. RR is in the Dictionary.
   RRA is not in the Dictionary and it is the last pattern; insert **RRA**, output code for its prefix: **code(RR),** then output code for last character: **code(A)**

output	Dictionary	
	Code Word	String
66	256	BA
65	257	AB
256	258	BAA
257	259	ABR
82	260	RR
260	261	RRA
65		

The compressed message is: **<66><65><256><257><82><260> <65>**

LZW

LZW: Number of bits transmitted

Example: Uncompressed String: **aaabbbbbbaabaaba**

Number of bits = Total number of characters * 8

= 16 * 8

= 128 bits

Compressed string (codewords): **<97><256><98><258><259><257><261>**

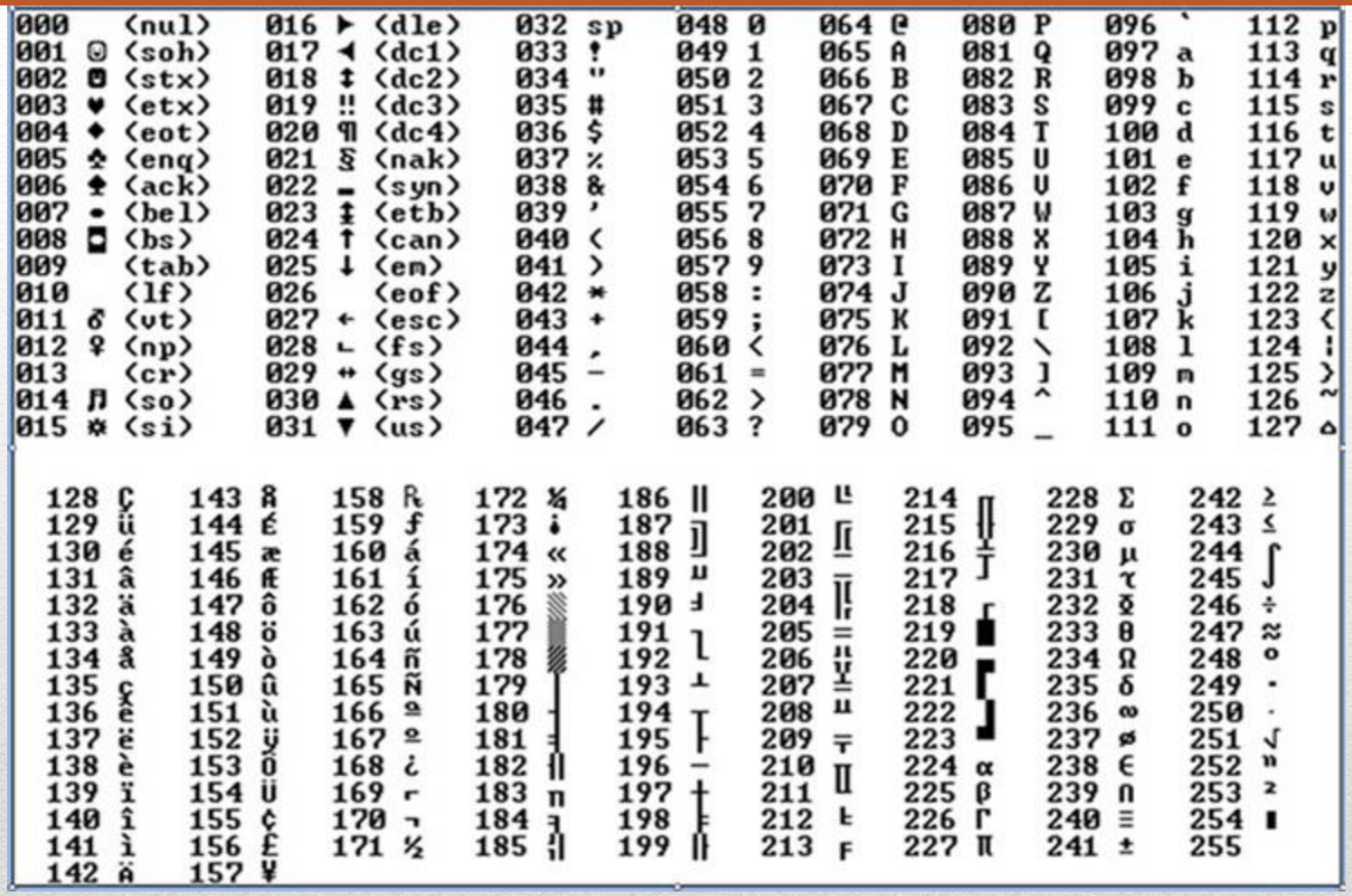
Number of bits = Total Number of codewords * 12

= 7 * 12

= 84 bits

Note: Each codeword is 12 bits because the minimum Dictionary size is taken as 4096, and

$2^{12} = 4096$



000		(nul)	016	▶	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◀	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	¦
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

128	Ç	143	Å	158	₧	172	¼	186	‖	200	╚	214	╓	228	Σ	242	≥
129	ü	144	É	159	ƒ	173	¡	187	╗	201	╔	215	╫	229	σ	243	≤
130	é	145	æ	160	á	174	«	188	╝	202	╩	216	╪	230	µ	244	⌠
131	â	146	Æ	161	í	175	»	189	╜	203	╦	217	┘	231	τ	245	⌡
132	ä	147	ô	162	ó	176	▒	190	╛	204	╠	218	┌	232	Φ	246	÷
133	à	148	ö	163	ú	177	▓	191	┐	205	═	219	█	233	Θ	247	≈
134	å	149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Ω	248	°
135	ç	150	û	165	Ñ	179	│	193	┴	207	╧	221	▌	235	δ	249	∙
136	ê	151	ù	166	ª	180	┤	194	┬	208	╨	222	▐	236	∞	250	·
137	ë	152	ÿ	167	º	181	╡	195	├	209	╤	223	▀	237	ø	251	√
138	è	153	Ö	168	¿	182	╢	196	─	210	╥	224	α	238	∈	252	ⁿ
139	ï	154	Ü	169	⌐	183	╖	197	┼	211	╙	225	β	239	∩	253	²
140	î	155	¢	170	¬	184	╕	198	╞	212	╘	226	Γ	240	≡	254	■
141	ì	156	£	171	½	185	╣	199	╟	213	╒	227	π	241	±	255	
142	Ä	157	¥												

- Encode: **TOBEORNOTTOBEORTOBEORNOT**

Ascii Code Table

T	O	TO	T	TO = <256>
O	B	OB	O	OB = <257>
B	E	BE	B	BE = <258>
E	O	EO	E	EO = <259>
O	R	OR	O	OR = <260>
R	N	RN	R	RN = <261>
N	O	NO	N	NO = <262>
O	T	OT	O	OT = <263>
T	T	TT	T	TT = <264>
T	O	TO		
TO	B	TOB	<256>	TOB = <265>
B	E	BE		
BE	O	BEO	<258>	BEO = <266>
O	R	OR		
OR	T	ORT	<260>	ORT = <267>
T	O	TO		
TO	B	TOB		
TOB	E	TOBE	<265>	TOBE = <268>
E	O	EO		
EO	R	EOR	<259>	EOR = <269>
R	N	RN		
RN	O	RNO	<261>	RNO = <270>
O	T	OT		
OT			<263>	

LZW

	CHAR	STRING + CHAR	In Table?	Output	Add to Table	New STRING	Comments
1	t	t				t	first character- no action
2	h	th	no	t	256 = th	h	
3	e	he	no	h	257 = he	e	
4	/	e/	no	e	258 = e/	/	
5	r	/r	no	/	259 = /r	r	
6	a	ra	no	r	260 = ra	a	
7	i	ai	no	a	261 = ai	i	
8	n	in	no	i	262 = in	n	
9	/	n/	no	n	263 = n/	/	
10	i	/i	no	/	264 = /i	i	
11	n	in	yes (262)			in	first match found
12	/	in/	no	262	265 = in/	/	
13	S	/S	no	/	266 = /S	S	
14	p	Sp	no	S	267 = Sp	p	
15	a	pa	no	p	268 = pa	a	
16	i	ai	yes (261)			ai	matches *ai*, *ain* not in table yet
17	n	ain	no	261	269 = ain	n	*ain* added to table
18	/	n/	yes (263)			n/	
19	f	n/f	no	263	270 = n/f	f	
20	a	fa	no	f	271 = fa	a	
21	l	al	no	a	272 = al	l	
22	l	ll	no	l	273 = ll	l	
23	s	ls	no	l	274 = ls	s	
24	/	s/	no	s	275 = s/	/	
25	m	/m	no	/	276 = /m	m	
26	a	ma	no	m	277 = ma	a	
27	i	ai	yes (261)			ai	matches *ai*
28	n	ain	yes (269)			ain	matches longer string, *ain*
29	l	ainl	no	269	278 = ainl	l	
30	y	ly	no	l	279 = ly	y	
31	/	y/	no	y	280 = y/	/	
32	o	/o	no	/	281 = /o	o	
33	n	on	no	o	282 = on	n	
34	/	n/	yes (263)			n/	
35	t	n/t	no	263	283 = n/t	t	
36	h	th	yes (256)			th	matches *th*, *the* not in table yet
37	e	the	no	256	284 = the	e	*the* added to table
38	/	e/	yes			e/	
39	p	e/p	no	258	285 = e/p	p	
40	l	pl	no	p	286 = pl	l	
41	a	la	no	l	287 = la	a	
42	i	ai	yes (261)			ai	matches *ai*
43	n	ain	yes (269)			ain	matches longer string *ain*
44	/	ain/	no	269	288 = ain/	/	
45	EOF	/		/			end of file, output *STRING*

TABLE 27-3
LZW example. This shows the compression of the phrase: *the/rain/in/Spain/falls/mainly/on/the/plain/*.

- Decompression works in the reverse manner.
- Given the encoded sequence and the dictionary, the original message can be obtained by a dictionary lookup.
- No need to transmit the entire dictionary to the decoder because it can be re-created.
- The last symbol of the most recent dictionary entry is the first symbol of the next parse block → used to resolve possible conflicts in the decoder.

LZW

- how long should the dictionary grow?
- Clearly, as the dictionary size increases, so does the size of the strings that form the dictionary and consequently the compression gets better.
- with the dictionary growth, the number of bits required to represent the index also grows (bits required $b=\log_2 N$, where N is the dictionary size.
- Impractical to have a large N, and this value is normally limited to 4096, which gives 12 bits per index or it might also be an input parameter to the algorithm.
- LZW algorithm works well only when the input data is sufficiently large and there is sufficient redundancy in the data.
- Many popular programs, such as the UNIX-based compress and uncompress, gzip and gunzip, and the Windows-based WinZip program, are based on the LZW algorithm. It is also used while compressing images using the GIF format.

HUFFMAN CODING

Lossless Compression

- data sampling → each sample is assigned the same number of bits.
- length is $\log_2 n$, where n is the number of distinct samples or quantization intervals.
- Exp: audio data (8 bits or 16 bits per sample), images (24 bits per pixel), text ASCII data, ….
- some symbols occur more commonly than others → goal = to have more frequent symbols that have smaller code lengths and less frequent symbols that have longer code lengths.
- Idea: assign new codes to each symbol, depending on the frequency of occurrence of the symbol.
  - process starts by computing or statistically determining the probability of occurrence of each symbol.
  - symbols are then organized to form the leaves of a tree, and a binary tree is built.
  - most frequent symbol is closer to the root and the least frequent symbol is farthest from the root.

Huffman Coding

Huffman Coding

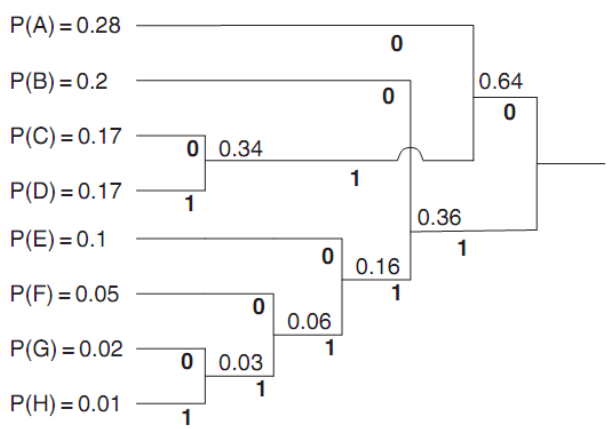
- Leafs of the tree → associated with the list of probabilities.
  - leaves sorted in increasing or decreasing order.
- Combination of the two nodes with the smallest probabilities.
  - The new node has a probability equal to the sum of the child node probabilities.
- The combination is repeated until the root node.
- Label the branches with a 0 and 1 starting from the root and going to all intermediary nodes.
- Traverse the Huffman tree from the root to a leaf node noting each branch label (1 or 0).
  - Huffman code of a symbol is the sequence of labels from root till leaf.

Symbol	Probability	Binary code	Code length
A	0.28	000	3
B	0.2	001	3
C	0.17	010	3
D	0.17	011	3
E	0.1	100	3
F	0.05	101	3
G	0.02	110	3
H	0.01	111	3

Average symbol length = 3

Entropy = 2.574

Efficiency = 0.858



Symbol	Probability	Huffman code	Code length
A	0.28	00	1
B	0.2	10	3
C	0.17	010	3
D	0.17	011	3
E	0.1	110	3
F	0.05	1110	4
G	0.02	11110	5
H	0.01	11111	5

Average symbol length = 2.63

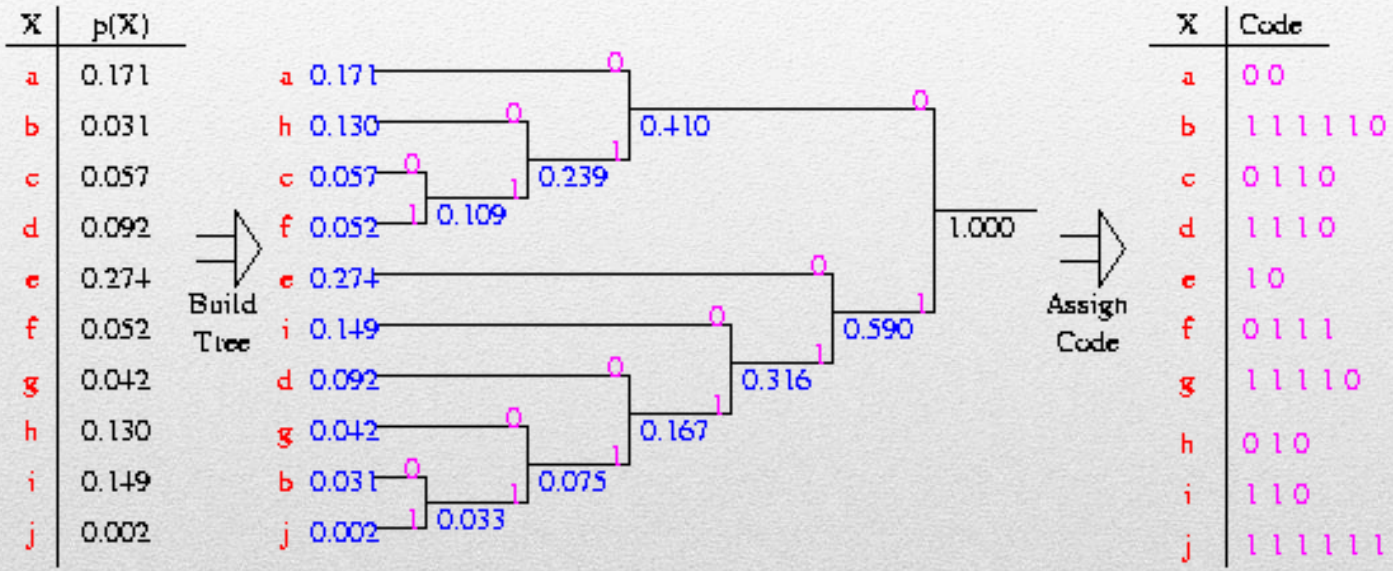
Entropy = 2.574

Efficiency = 0.9792

Huffman coding. The top table shows the symbols used with their probability distribution and original binary code length. Each symbol has the same code length 3. Also shown are the entropy and the efficiency of the code. The middle figure shows the Huffman tree with branches labeled in bold. The bottom table shows the symbol codes obtained from the Huffman tree and the corresponding code lengths. The average code lengths are decreased, thus increasing the coding efficiency.

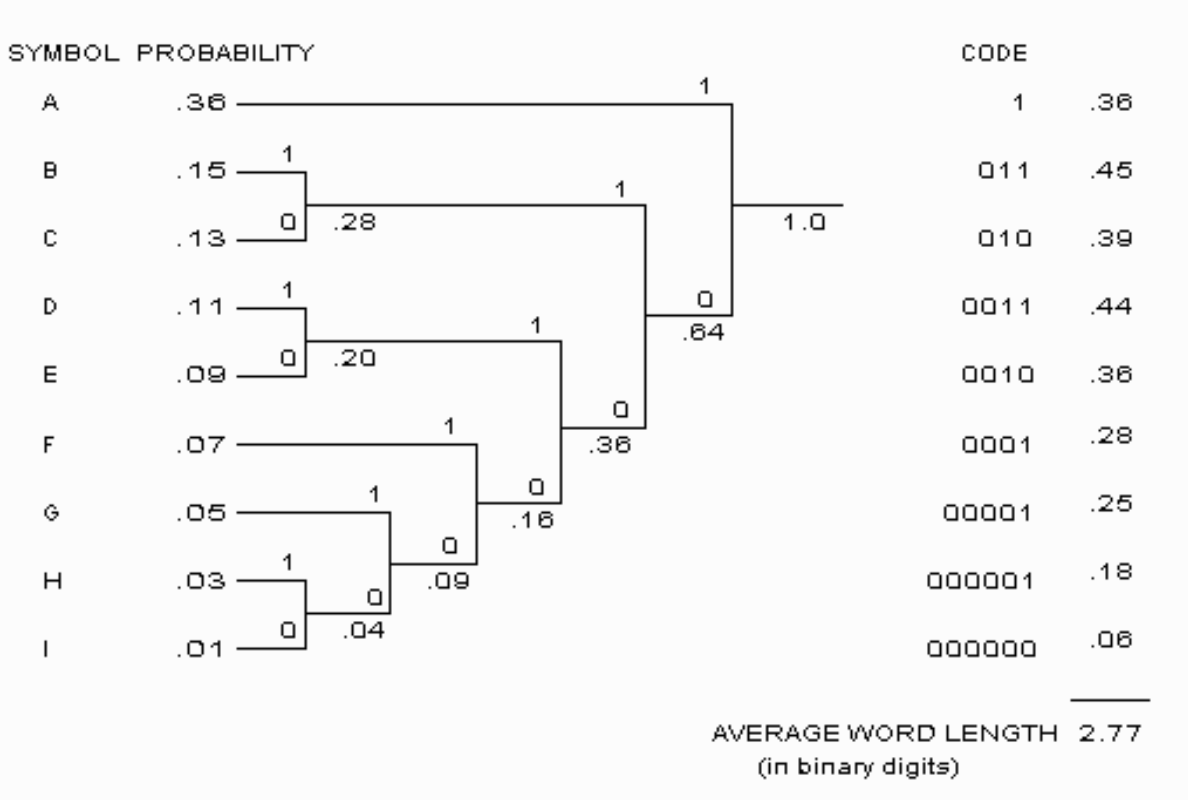
Huffman Coding

- Apply the Huffman coding to obtain the codes
- Compute the compression ratio



Try

- Apply the Huffman coding to obtain the codes
- Compute the compression ratio



Try

ARITHMETIC CODING

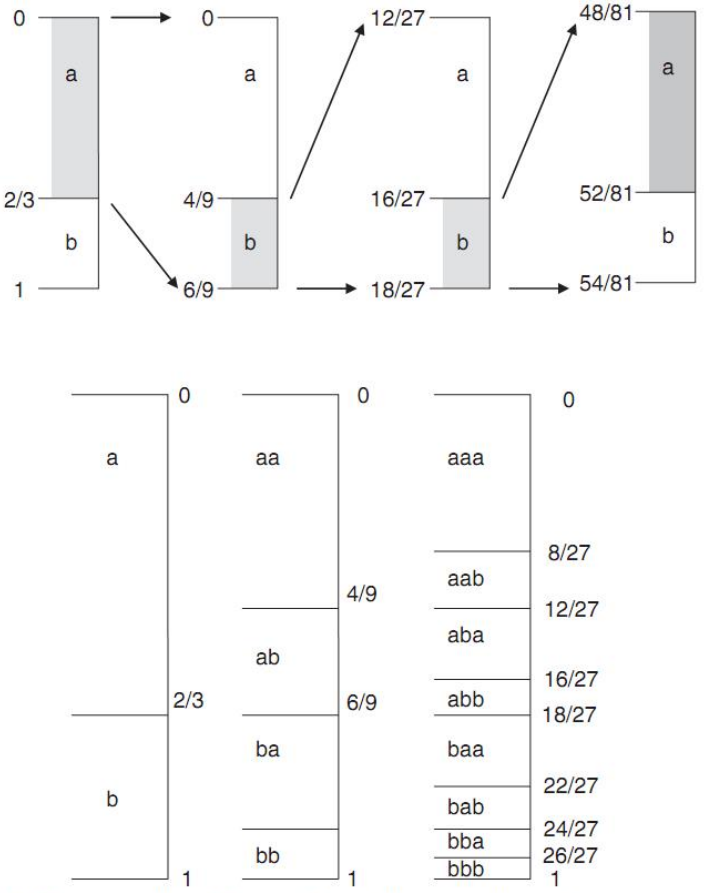
Lossless Compression

- Previous coding techniques → a whole number of bits were required to encode a symbol (or symbol group). This number is never less than 1.
- Arithmetic coding overcome the problem by mapping an entire message to a real number between zero and one.
- The algorithm can be outlined as follows. Suppose we have a vocabulary of n symbols:

1. Divide the interval [0,1] into n segments corresponding to the n symbols; the segment of each symbol has a length proportional to its probability. Each segment i has an upper bound U and lower bound L corresponding to the start of the segment and the end of the segment ($U - L = P_i$).
2. Choose the segment that corresponds to the first symbol in the message string. This is the new current interval with its computed new upper and lower bounds.
3. Divide the new current interval again into n new segments with length proportional to the symbols probabilities and compute the new intervals accordingly.
4. From these new segments, choose the one corresponding to the next symbol in the message.
5. Continue Steps 3 and 4 until the whole message is coded.
6. Represent the segment's value by a binary fraction in the final interval.

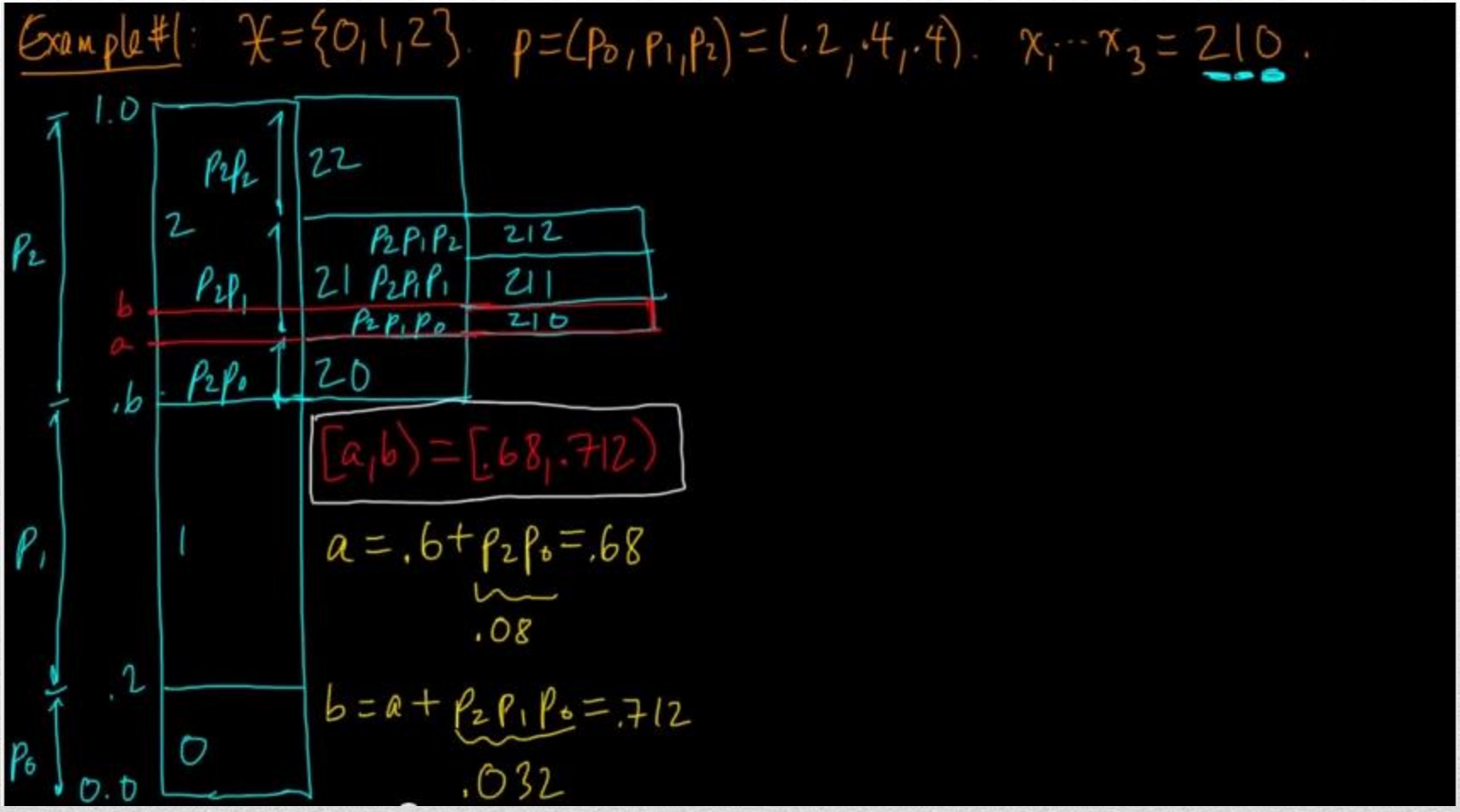
Arithmetic Coding

Two symbols a and b: P(a)=2/3, P(b)=1/3
« abba » fall into the interval [48/81, 52/81].
Its binary representation is
[0.10010111, 0.1010010].
This interval can be represented by 0.101, which corresponds to the decimal 0.625.
Thus, 101 should suffice as a code for "abba".

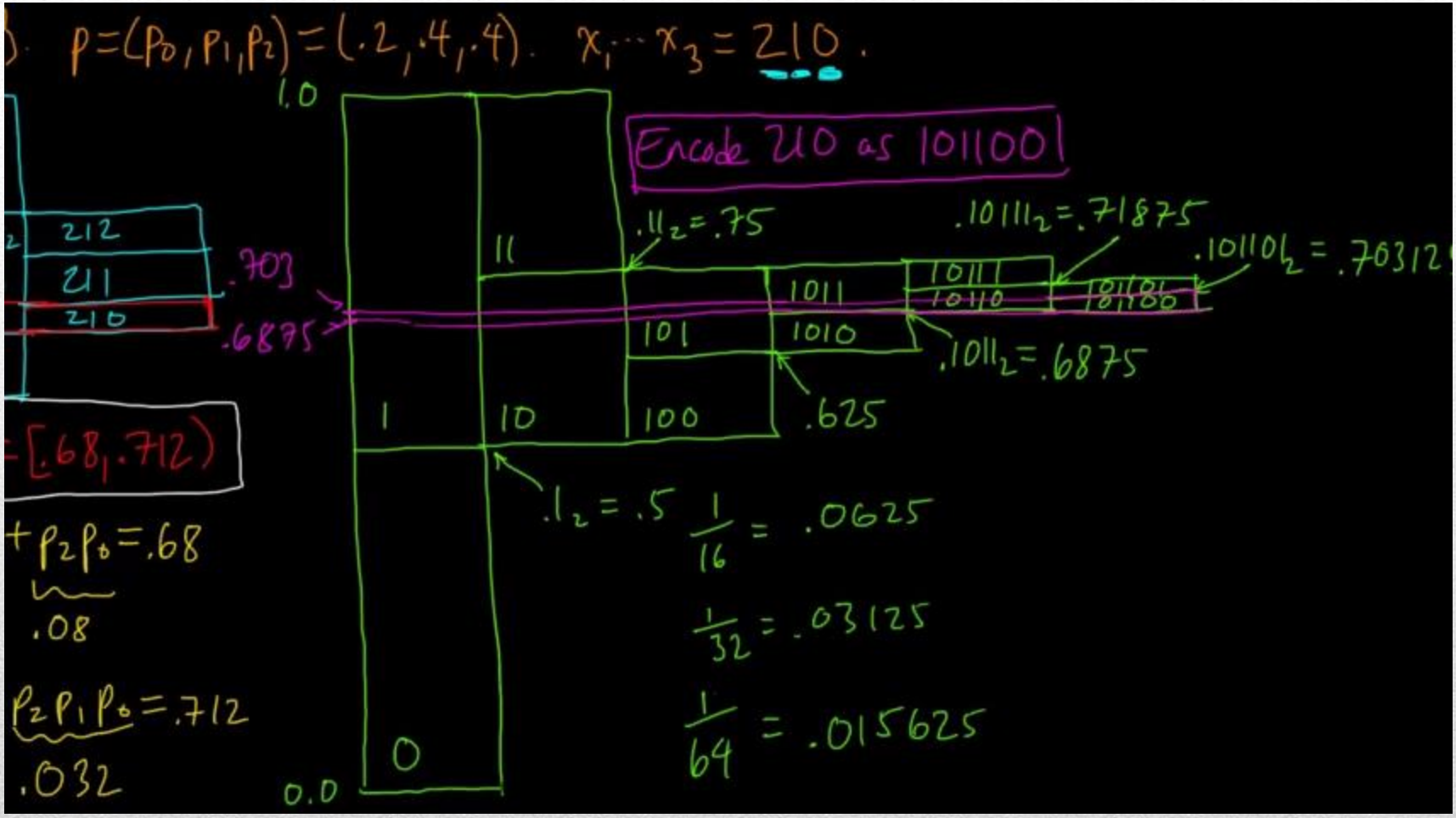


Arithmetic coding. The top picture illustrates the coding process to code the message "abba" given the original symbol probabilities. The bottom picture shows all possible intervals up to level 3 using the same probability distribution. At each stage, every subinterval is broken into two intervals in the ratio of the symbol probabilities.

Arithmetic Coding



Arithmetic Coding



Arithmetic Coding

- To decode → should know when to stop
  - Encode the length of the sequence with the sent one.
- Try the one below that has the length 5

- $M = a_1a_0a_3a_2a_0$

Information Source

$A = \{a_0, a_1, a_2, a_3, a_4\}$

$P = \{0.4, 0.2, 0.2, 0.1, 0.1\}$

What is going on...

The transmitted codeword from the arithmetic encoder is

01111000001

which corresponds to the following decimal number,

$b = 0.46923828125.$

Arithmetic Coding

SHANON-FANO CODING

Lossless Compression

Shanon-Fano Coding

- One of the first methods that constructs a variable-size codes
- Given n symbols with known probabilities:
  - Arrange them in descending order based on their probabilities
  - Divide the set into two subset with almost equal sum of probabilities
  - Label one subset with 0 and the other 1
  - Repeat the split and append to the already given label a 0 or a 1 until the subset contain one symbol

Shannon-Fano Coding

	Prob.	Steps					Final
1.	0.25	1	1				:11
2.	0.20	1	0				:10
3.	0.15	0		1	1		:011
4.	0.15	0		1	0		:010
5.	0.10	0		0		1	:001
6.	0.10	0		0		0 1	:0001
7.	0.05	0		0		0 0	:0000

Shannon-Fano Example

The average size of this code is $0.25 \times 2 + 0.20 \times 2 + 0.15 \times 3 + 0.15 \times 3 + 0.10 \times 3 + 0.10 \times 4 + 0.05 \times 4 = 2.7$ bits/symbol.

This is a good result because the entropy (the smallest number of bits needed, on average, to represent each symbol) is:
$0.25 \log_2 0.25 + 0.20 \log_2 0.20 + 0.15 \log_2 0.15 + 0.15 \log_2 0.15 + 0.10 \log_2 0.10 + 0.10 \log_2 0.10 + 0.05 \log_2 0.05 \approx 2.67$.

Shannon-Fano coding

Shanon-Fano Coding

- Exercise:
- Repeat the calculation on the previous example but place the first split between the third and fourth symbols.
- Calculate the average size of the code and show that it is greater than 2.67 bits/symbol.

LOSSY COMPRESSION

- Entropy-coding or lossless compression has theoretical limits on the amount of compression that can be achieved.
- Why not to sacrifice some amount of information in order to increase the ratio of compression.

Lossy Compression

- Lossless compression have limits on the amount of compression that can be achieved
- Visual Studies have shown that distortion introduced by image compression are still perceptually acceptable.
- Many lossy schemes introduce a distortion because of quantizing the symbol code representation.
- Quantization is inherently lossy.

Lossy compression

- Usually → successive samples in a signal are almost similar
- Instead of sending the individual samples, why not sending the difference or error between successive samples?
- Errors normally should be small and require less bits to represent.
- A prediction can be made of the next value using the current and past samples.

Differential PCM (1)

Differential PCM (2)

- Differential Pulse code modulation (DPCM) uses the simplest form of prediction.
  - Simplest prediction is the signal itself
- If the $(n-1)^{th}$ sample is y(n-1) then n^{th} sample is y(n) = y(n-1) + d(n)
- so d(n) = y(n) – y(n-1) is the prediction error or the difference between the predicted and actual samples.

Original	27	28	29	28	26	27	29	28	30	32	34	36	38
Encoded	27	1	1	-1	-2	1	2	-1	2	2	2	2	2

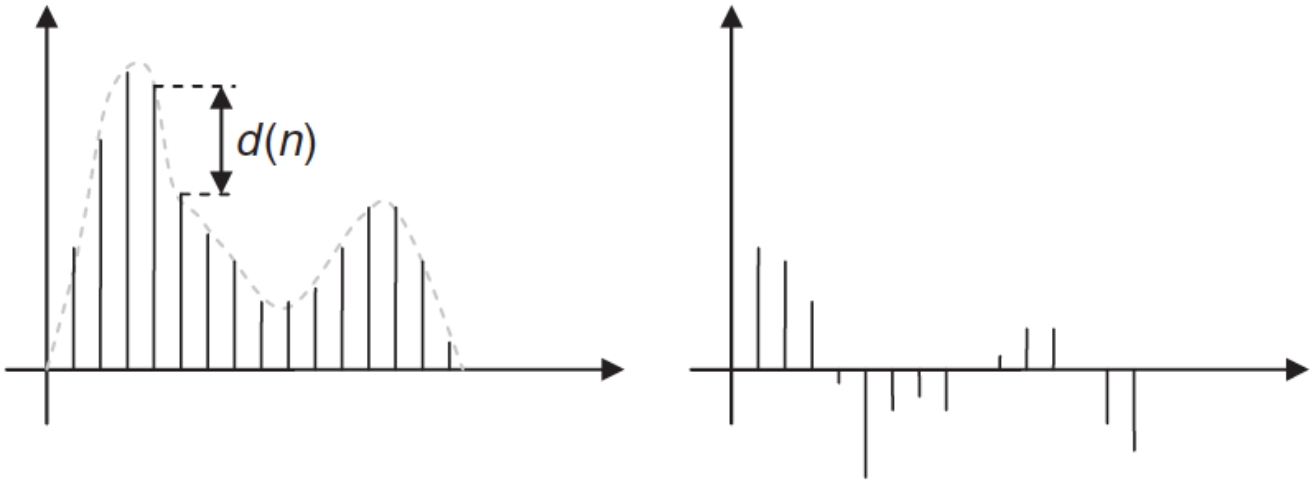


Figure 6-10 Differential pulse code modulation. The left signal shows the original signal. The right signal samples are obtained by computing the differences between successive samples. Note the first difference d(1) is the same as y(1); all the successive differences d(n) are computed as y(n) − y(n − 1).

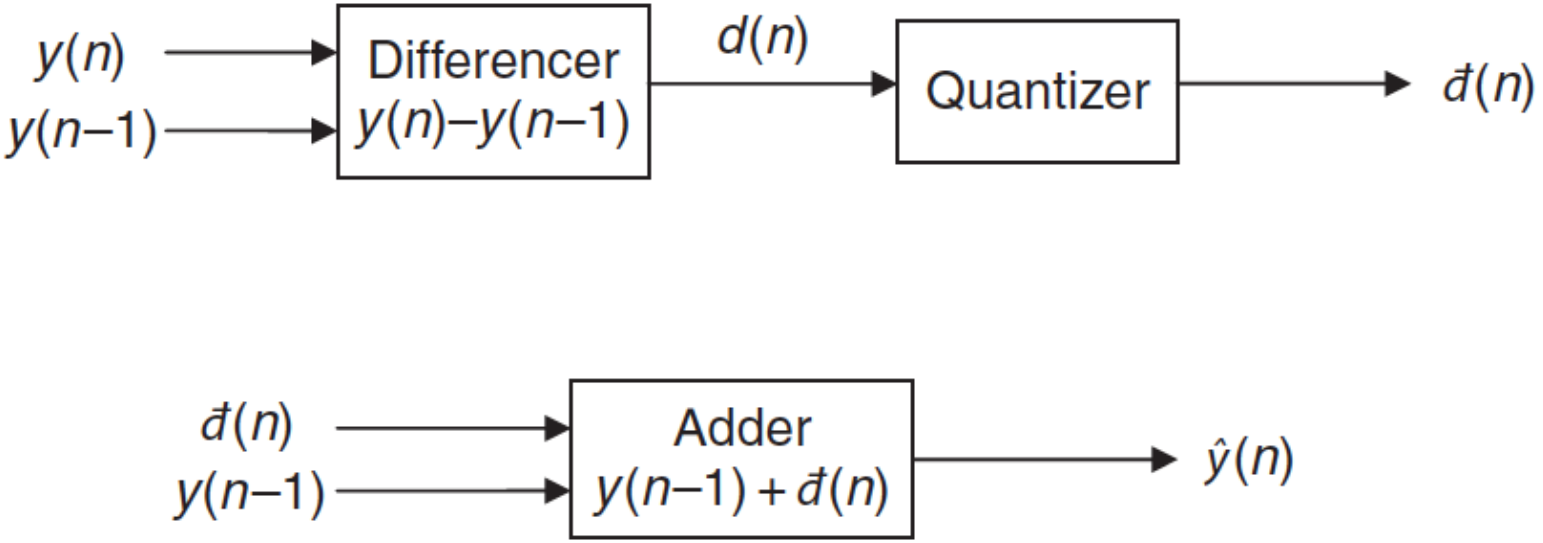
Note that after computing d(n), we need to quantize them into $\hat{d}(n)$

Differential PCM (3)

- At the encoder, DPCM sends the quantized values of d(n)
- Open loop case.
- Decoder works by recovering the lossy signal $\hat{Y}(n)$ as follows:

$$\begin{aligned} &\text{If } n = 1, && \hat{y}(n) = \hat{d}(n) \\ &\text{else} && \hat{y}(n) = y(n-1) + \hat{d}(n). \end{aligned}$$

Differential PCM (4)



Can you spot any problem?

$\hat{y}(n) = y(n-1) + \text{đ}(n)$

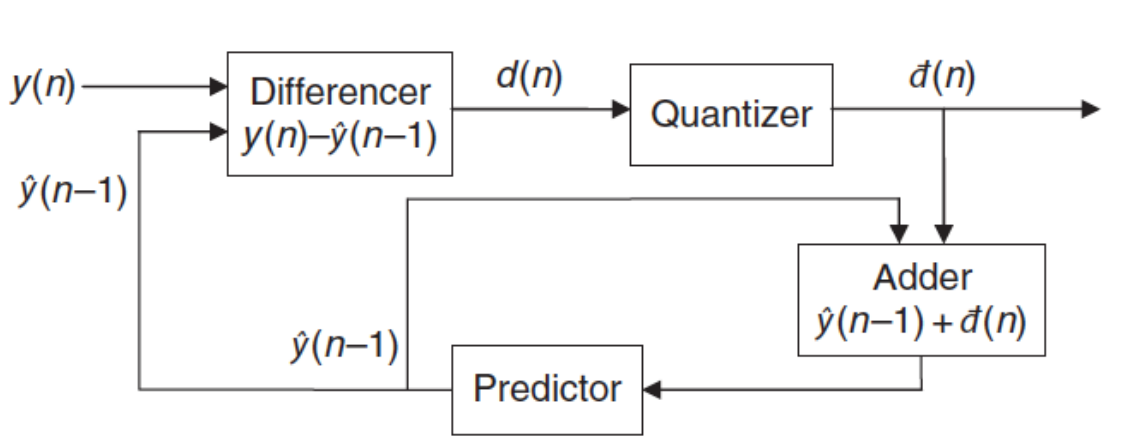
But y(n-1) is not available → we have $\hat{y}(n-1)$

Open loop differential pulse code modulation problem

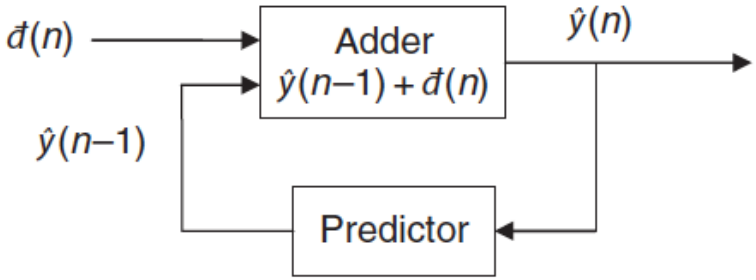
Differential PCM (5)

- To solve the open loop case:
- The difference d(n) at the encoder is not computed as y(n)-y(n-1) anymore but y(n)-ŷ(n-1)

Encoder



Decoder



Differential PCM (5)

Exercise

- Given the following piece of image, answer the following:

a. Apply on the image the lossless version of the DPCM after reading it in zig-zag order.

b. Apply on the image the lossy version of the DPCM by quantizing the values to 4 levels except the first one.

244	242	244	241	252
238	240	240	239	236
234	232	236	242	234
228	232	236	243	232
230	228	226	231	223

- Quantization → normally performed on individual samples
- Causes a lossy compression
- Most compression techniques operate by quantizing each symbol taken individually.
- known as scalar quantization.
- Ex: instead of using 3 bytes per channel, use 1

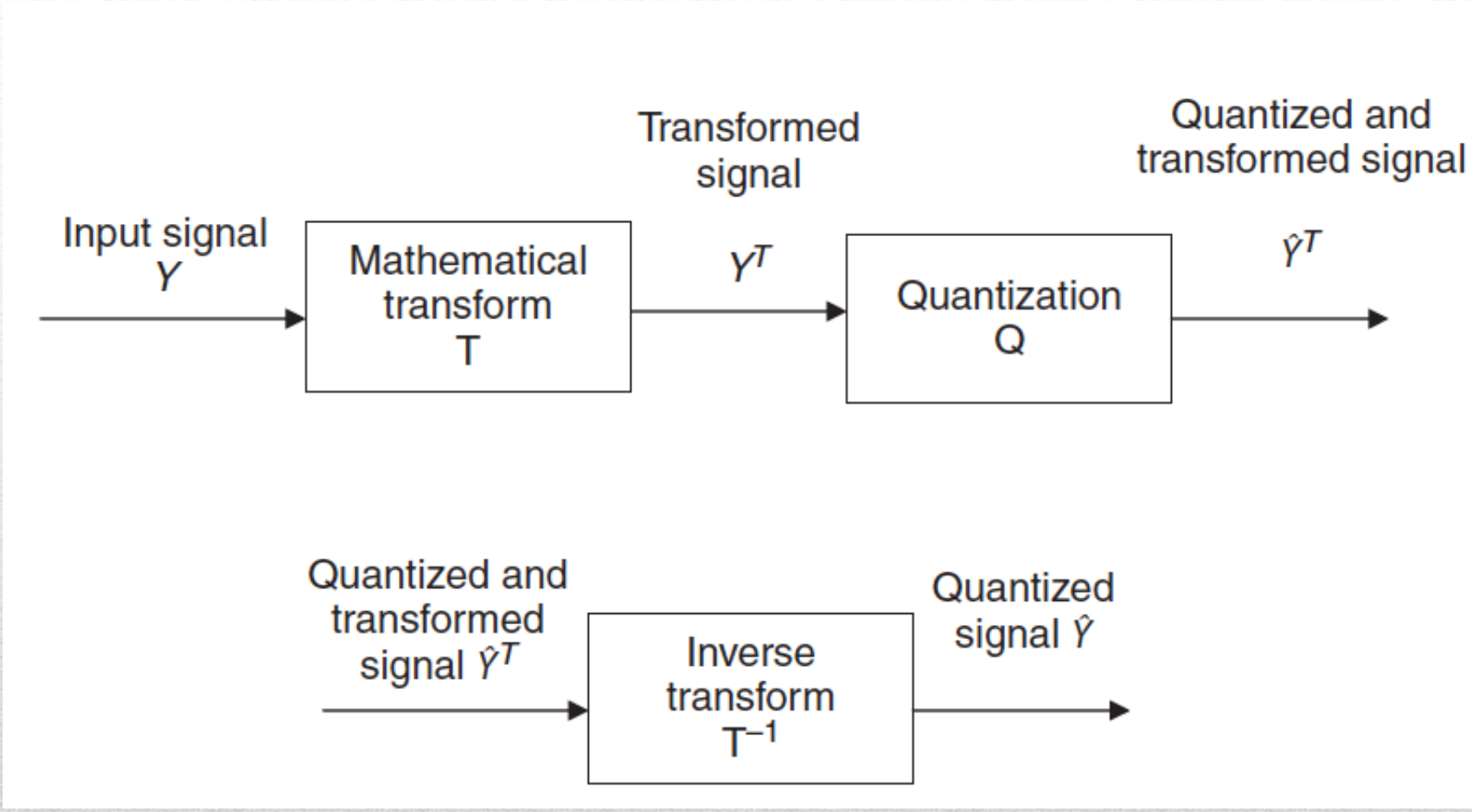
Vector Quantization

- Advantages
  - Simple to implement,
  - fast to quantize/dequantize
  - lead to a good symbol redundancy
- Drawbacks
  - insensitive to intersample correlations → symbol are independent
- In reality → digital signals are not random but exhibit a high degree of correlation in the local neighborhood
- One of solutions proposed → Quantize group of samples

Vector Quantization

- Mathematical transformation on the input signal
- Results in a different signal
- Transformation must be invertible
- In the new domain, the signal may have lower entropy and hence lower bits to represent
- By themselves, transform coding is not lossy but they usually employ quantization which is lossy

Transform coding techniques



Transform coding

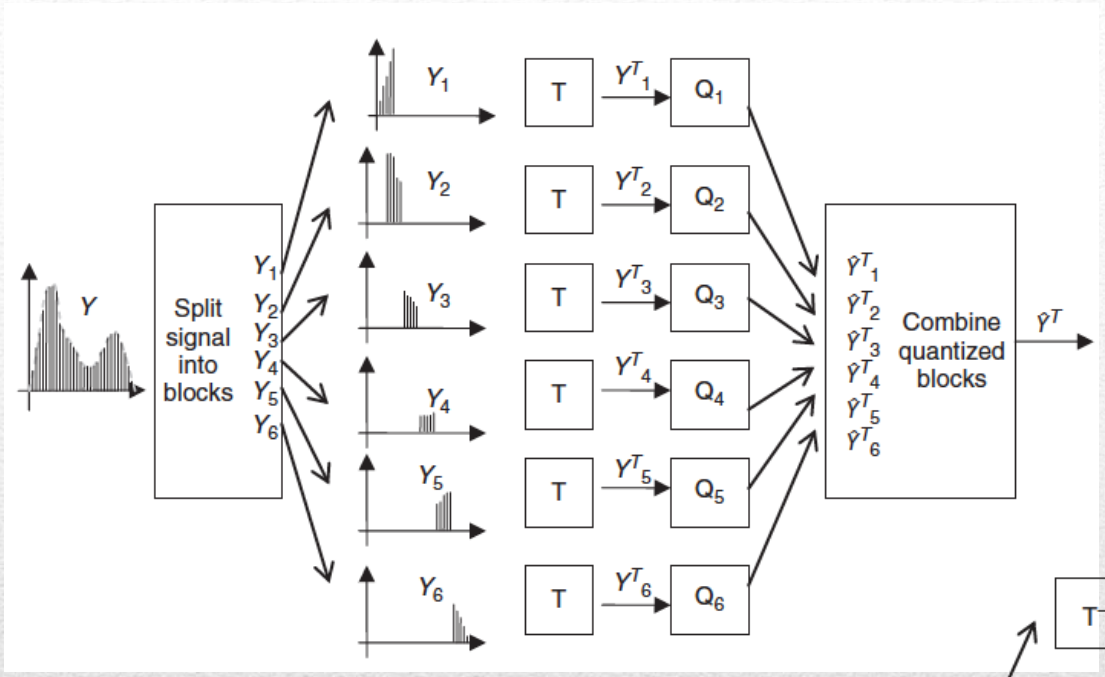
Transform coding

- Very well suited for 1-D (sound) and 2-D (images) signals.
- Three categories of approaches:
  - Frequency Transforms: DFT, DCT, Hadamard transforms, Lapped Orthogonal transforms, …
  - Statistical Transforms: PCA (changes the number of dimensions of the signal using correlation and eigenvectors/values), Karhunen-Loeve transforms, …
  - Wavelet Transforms: multi-resolution representation of the signal.

- To keep computations manageable, break images in *n* 8x8 blocks)
- On each block, apply DCT
- Each block may be compressed differently
- Important when controlling bit rates
- If we have B bits per frame, we give higher entropy blocks higher bits.

$$B = \sum_{i=1}^{i=n} b_i$$

Transform coding (DCT/JPEG example)

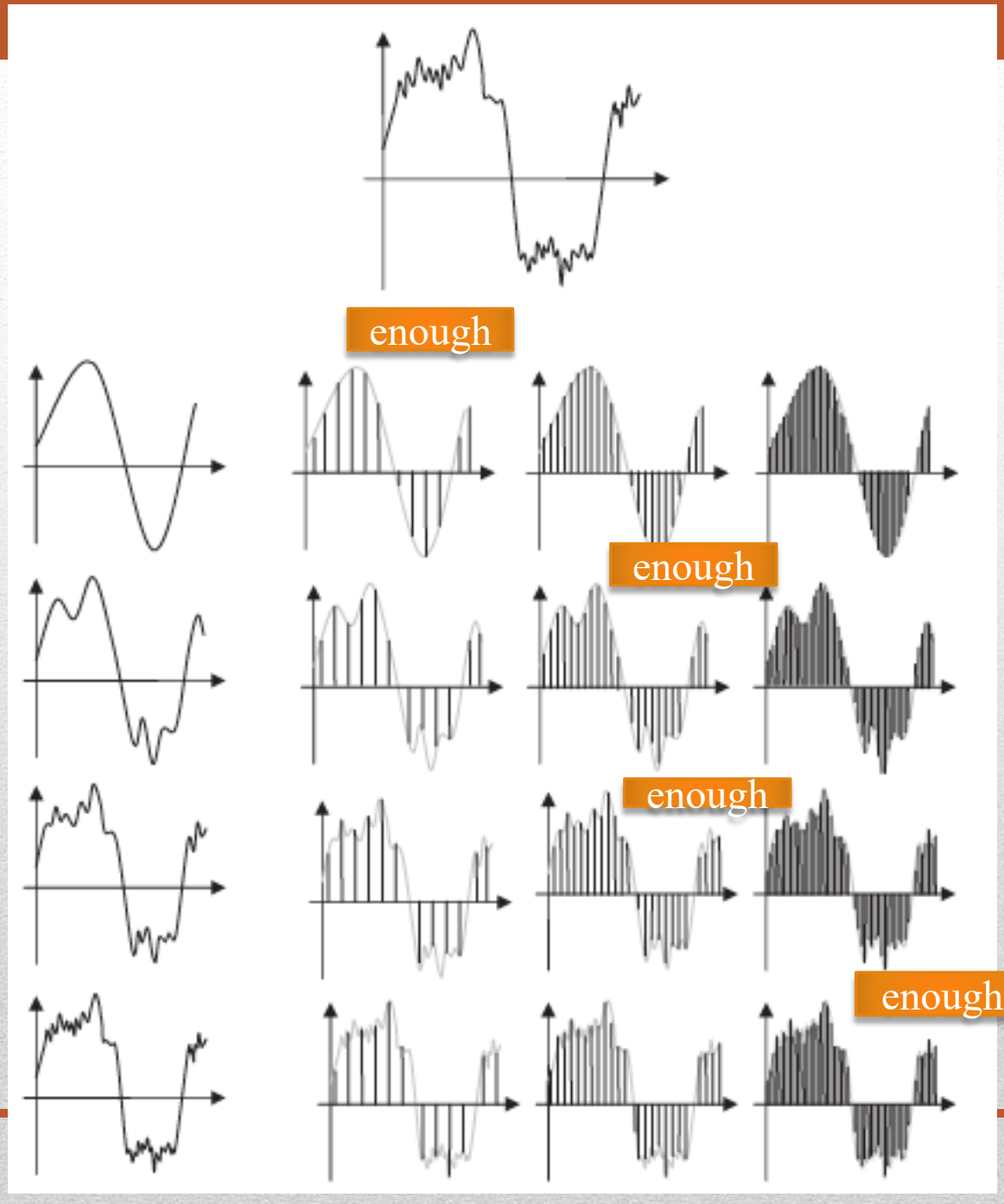


Transform coding (JPEG/DCT)

- Successive approximation or multi-resolution
- Signal is decomposed into a coarse (low frequency) and added details (high frequency) at various resolutions
- Depending on the band importance, the more important band is given more quantization levels (bits) than less important ones

Subband Coding

Subband Coding

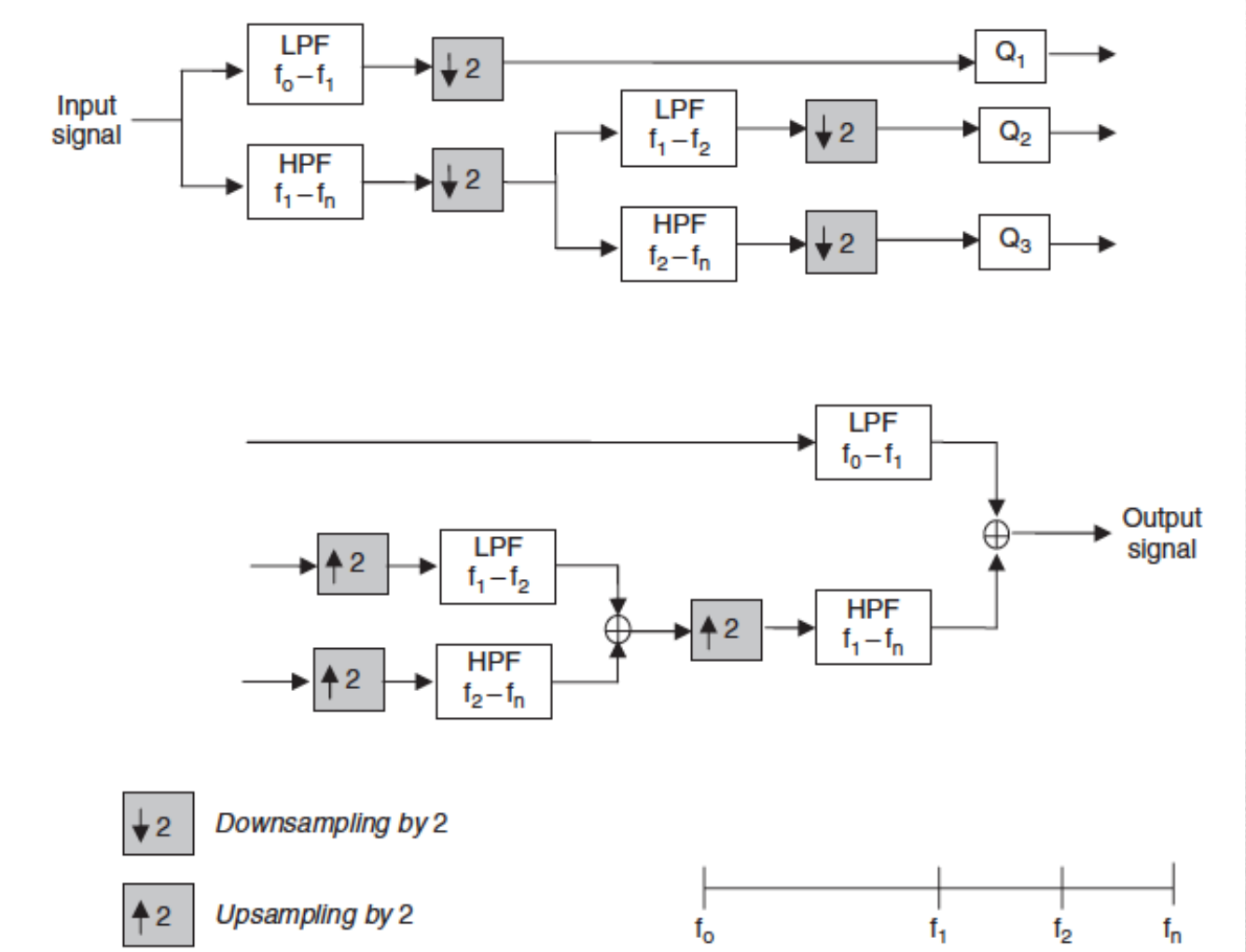


Low freq

High freq

- Signal is passed through low pass and high pass filters
- At each frequency resolution, the low pass signal is downsampled by 2 to reduce the time domain resolution

Subband Coding (Wavelet)



Subband Coding (Wavelet)

COMPRESSION ISSUES

- Performance requirement: real-time, offline
- Data available for analysis

Encoder speed and complexity

- Symmetric: encoder and decoder work in the exact but opposite way, requiring same time
- Asymmetric: one of them may be more complex than the other and requires more time. (DVD, complex to encode but easy to decode and view)
- Adaptive versus non-adaptive compression

Symmetric and asymmetric compression