CM3106 Multimedia

Introduction to Compression

Dr Kirill Sidorov

SidorovK@cardiff.ac.uk
www.facebook.com/kirill.sidorov

Prof David Marshall

MarshallAD@cardiff.ac.uk

School of Computer Science and Informatics
Cardiff University, UK

Modelling and compression

- We are interested in modelling multimedia data.
- To model means to replace something complex with a simpler (= shorter) analog.
- Some models help understand the original phenomenon/data better:

Example: Laws of physics

Huge arrays of astronomical observations (*e.g.* Tycho Brahe's logbooks) summarised in a few characters (*e.g.* Kepler, Newton):

$$|\boldsymbol{F}| = G\frac{M_1 M_2}{r^2}.$$

- This model helps us understand gravity better.
- Is an example of tremendous compression of data.

- We will look at models whose purpose is primarily compression of multimedia data.

Raw video, image, and audio files can be very large.

Example: One minute of uncompressed audio

Audio type	*44.1 KHz*	*22.05 KHz*	*11.025 KHz*
16 bit stereo:	10.1 MB	5.05 MB	2.52 MB
16 bit mono:	5.05 MB	2.52 MB	1.26 MB
8 bit mono:	2.52 MB	1.26 MB	630 KB

Example: Uncompressed images

Image type	*File size*
640×480 (VGA) 8-bit gray scale	307 KB
1920×1080 (Full HD) 16-bit YUYV 4:2:2	4.15 MB
2560×1600 24-bit RGB colour	11.7 MB

The need for compression

Example: Videos (involves a stream of audio plus video imagery)

- Raw video — uncompressed image frames 512×512 True Colour at 25 FPS = 1125 MB/min.
- HDTV (1920×1080) — gigabytes per minute uncompressed, True Colour at 25 FPS = 8.7 GB/min.

- Relying on higher bandwidths is not a good option — M25 Syndrome: traffic will always increase to fill the current bandwidth limit whatever this is.
- Compression has to be part of the representation of audio, image, and video formats.

Suppose we have an information source (random variable) S which emits symbols $\{s_1, s_2, \ldots, s_n\}$ with probabilities $p_1, p_2, \ldots, p_n$. According to Shannon, the entropy of S is defined as:

$$H(S) = \sum_i p_i \log_2 \frac{1}{p_i},$$

where p_i is the probability that symbol s_i will occur.

- When a symbol with probability p_i is transmitted, it reduces the amount of uncertainty in the receiver by a factor of $\frac{1}{p_i}$.
- $\log_2 \frac{1}{p_i} = -\log_2 p_i$ indicates the amount of information conveyed by s_i, *i.e.*, the number of binary digits needed to code s_i (Shannon's coding theorem).

Entropy example

Example: Entropy of a fair coin

The coin emits symbols s_1 = heads and s_2 = tails with $p_1 = p_2 = 1/2$.
Therefore, the entropy if this source is:

$$H(\text{coin}) = -(1/2 \times \log_2 1/2 + 1/2 \times \log_2 1/2) =$$
$$-(1/2 \times -1 + 1/2 \times -1) = -(-1/2 - 1/2) = 1 \text{ bit.}$$

Example: Grayscale "image"

- In an image with uniform distribution of gray-level intensity (and all pixels independent), i.e. $p_i = 1/256$, then
  - The number of bits needed to code each gray level is 8 bits.
  - The entropy of this image is 8.
- We will shortly see that real images are not like that!

Example: Breakfast order #1.

Alice: "What do you want for breakfast: pancakes or eggs? I am unsure, because you like them equally ($p_1 = p_2 = 1/2$)..."
Bob: "I want pancakes."

Question:

How much information has Bob communicated to Alice?

Example: Breakfast order #1.

Alice: "What do you want for breakfast: pancakes or eggs? I am unsure, because you like them equally ($p_1 = p_2 = 1/2$)..."
Bob: "I want pancakes."

Question:

How much information has Bob communicated to Alice?

Answer:

He has reduced the uncertainty by a factor of 2, therefore 1 bit.

Example: Breakfast order #2.

Alice: "What do you want for breakfast: pancakes, eggs, or salad? I am unsure, because you like them equally ($p_1 = p_2 = p_3 = 1/3$)..."
Bob: "Eggs."
Question: What is Bob's entropy assuming he behaves like a random variable = how much information has Bob communicated to Alice?

Example: Breakfast order #2.

Alice: "What do you want for breakfast: pancakes, eggs, or salad? I am unsure, because you like them equally ($p_1 = p_2 = p_3 = 1/3$)..."

Bob: "Eggs."

Question: What is Bob's entropy assuming he behaves like a random variable = how much information has Bob communicated to Alice?

Answer:

$$H(\text{Bob}) = \sum_{i=1}^{3} \frac{1}{3} \log_2 3 = \log_2 3 \approx 1.585 \text{ bits.}$$

Example: Breakfast order #3.

Alice: “What do you want for breakfast: pancakes, eggs, or salad? I am unsure, because you like them equally ($p_1 = p_2 = p_3 = 1/3$)…”

Bob: “Hmm, I do not know. I definitely do not want salad.”

Question: How much information has Bob communicated to Alice?

Example: Breakfast order #3.

Alice: "What do you want for breakfast: pancakes, eggs, or salad? I am unsure, because you like them equally ($p_1 = p_2 = p_3 = 1/3$)…"
Bob: "Hmm, I do not know. I definitely do not want salad."
Question: How much information has Bob communicated to Alice?
Answer: He has reduced her uncertainty by a factor of $3/2$ (leaving 2 out of 3 equal options), therefore transmitted $\log_2 3/2 \approx 0.58$ bits.

English letter frequencies

i	a_i	p_i	$\log_2 \frac{1}{p_i}$	i	a_i	p_i	$\log_2 \frac{1}{p_i}$	i	a_i	p_i	$\log_2 \frac{1}{p_i}$
1	a	0.06	4.1	10	j	0.00	10.7	19	s	0.06	4.1
2	b	0.01	6.3	11	k	0.01	6.9	20	t	0.07	3.8
3	c	0.03	5.2	12	l	0.04	4.9	21	u	0.03	4.9
4	d	0.03	5.1	13	m	0.02	5.4	22	v	0.01	7.2
5	e	0.09	3.5	14	n	0.06	4.1	23	w	0.01	6.4
6	f	0.02	5.9	15	o	0.07	3.9	24	x	0.01	7.1
7	g	0.01	6.2	16	p	0.02	5.7	25	y	0.02	5.9
8	h	0.03	5.0	17	q	0.01	10.3	26	z	0.00	10.4
9	i	0.06	4.1	18	r	0.05	4.3	27	-	0.19	2.4
										$\sum_i p_i \log_2 \frac{1}{p_i}$	4.11

Figure 1.16. Probability distribution over the 27 outcomes for a randomly selected letter in an English language document (estimated from *The frequently asked questions manual for Linux*). The picture shows the probabilities by the sizes of white squares.

From D. MacKay "Information Theory, Inference, and Learning Algorithms", 2003.

Shannon's experiment (1951)

first line is the original text and the numbers in the second line indicate the guess at which the correct letter was obtained.

```
(1) T H E R E   I S   N O   R E V E R S E   O N   A   M O T O R C Y C L E   A
(2) 1 1 1 5 1 1 2 1 1 2 1 1 15 1 17 1 1 1 2 1 3 2 1 2 2 7 1 1 1 1 4 1 1 1 1 1 3 1
(1) F R I E N D   O F   M I N E   F O U N D   T H I S   O U T
(2) 8 6 1 3 1 1 1 1 1 1 1 1 1 1 1 6 2 1 1 1 1 1 1 2 1 1 1 1 1 1
(1) R A T H E R   D R A M A T I C A L L Y   T H E   O T H E R   D A Y
(2) 4 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1 1 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1    (9)
```

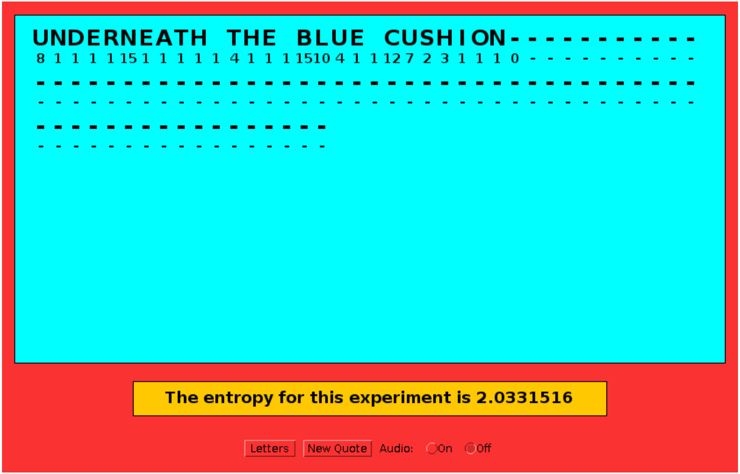
Out of 102 symbols the subject guessed right on the first guess 79 times, on the second guess 8 times, on the third guess 3 times, the fourth and fifth guesses 2 each and only eight times required more than five guesses. Results

Estimated entropy for English text: $H_{\text{English}} \approx 0.6 - 1.3$ bits/letter. (If all letters and space were equally probable, then it would be $H_0 = \log_2 27 \approx 4.755$ bits/letter.)

External link: Shannon's original 1951 paper.

External link: Java applet recreating Shannon's experiment.

Shannon’s experiment (1951): my attempt



Estimated entropy for my attempt: **2.03** bits/letter. Why?

Shannon’s coding theorem

Shannon 1948

> We will now justify our interpretation of H as the rate of generating information by proving that H determines the channel capacity required with most efficient coding.
>
> *Theorem 9*: Let a source have entropy H (bits per symbol) and a channel have a capacity C (bits per second). Then it is possible to encode the output of the source in such a way as to transmit at the average rate $\frac{C}{H} - \epsilon$ symbols per second over the channel where ϵ is arbitrarily small. It is not possible to transmit at an average rate greater than $\frac{C}{H}$.

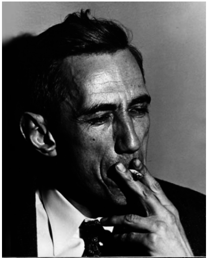
Basically:

The ideal code length for an event with probability p is $L(p) = -log_2 p$ ones and zeros (or generally, $-log_b p$ if instead we use b possible values for codes).

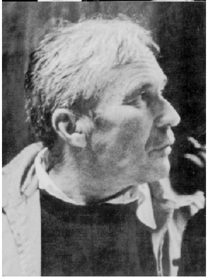
External link: Shannon’s original 1948 paper.

Shannon vs Kolmogorov

What if we have a finite string?



Shannon's entropy is a statistical measure of information. We can "cheat" and regard a string as infinitely long sequence of i.i.d. random variables. Shannon's theorem then approximately applies.



Kolmogorov Complexity: Basically, the length of the shortest program that ouputs a given string. Algorithmical measure of information.

- $K(S)$ is not computable!
- Practical algorithmic compression is hard.

Compression in multimedia data

Compression basically employs redundancy in the data:

Temporal in 1D data, 1D signals, audio, between video frames *etc*.

Spatial correlation between neighbouring pixels or data items.

Spectral *e.g.* correlation between colour or luminescence components. This uses the frequency domain to exploit relationships between frequency of change in data.

Psycho-visual exploit perceptual properties of the human visual system.

Lossless vs lossy compression

Compression methods can also be categorised in two broad ways:

Lossless compression: after decompression gives an exact copy of the original data.

Example: Entropy encoding schemes (Shannon-Fano, Huffman coding), arithmetic coding, LZ/LZW algorithm (used in GIF image file format).

Lossy compression: after decompression gives ideally a “close” approximation of the original data, ideally perceptually lossless.

Example: Transform coding — FFT/DCT based quantisation used in JPEG/MPEG differential encoding, vector quantisation.

Why lossy compression?

- Lossy methods are typically applied to high resoultion audio, image compression.
- Have to be employed in video compression (apart from special cases).

Basic reason:

- Compression ratio of lossless methods (*e.g*. Huffman coding, arithmetic coding, LZW) is not high enough for audio/video.
- By cleverly making a small sacrifice in terms of fidelity of data, we can often achieve very high compression ratios.
    - Cleverly = sacrifice information that is perceptually unimportant.

Lossless compression algorithms

- Entropy encoding:
  - Shannon-Fano algorithm.
  - Huffman coding.
  - Arithmetic coding.
- Repetitive sequence suppression.
- Run-Length Encoding (RLE).
- Pattern substitution.
- Lempel-Ziv-Welch (LZW) algorithm.

If a sequence a series on n successive tokens appears:

- Replace series with a token and a count number of occurrences.
- Usually need to have a special flag to denote when the repeated token appears.

Example:

`89400000000000000000000000000000000`

we can replace with:

`894f32`

where `f` is the flag for zero.

Simple repetition suppression

- Fairly straight forward to understand and implement.
- Simplicity is its downfall: poor compression ratios.

Compression savings depend on the content of the data.

Applications of this simple compression technique include:

- Suppression of zeros in a file (zero length suppression)
  - Silence in audio data, pauses in conversation etc.
  - Sparse matrices.
  - Component of JPEG.
  - Bitmaps, *e.g.* backgrounds in simple images.
  - Blanks in text or program source files.
- Other regular image or data tokens.

Run-length encoding (RLE)

This encoding method is frequently applied to graphics-type images (or pixels in a scan line) — simple compression algorithm in its own right. It is also a component used in JPEG compression pipeline.

Basic RLE Approach (*e.g.* for images):

- Sequences of image elements $X_1, X_2, \ldots, X_n$ (row by row).
- Mapped to pairs $(c_1, L_1), (c_2, L_2), \ldots, (c_n, L_n)$,
  where c_i represent image intensity or colour and L_i the length of the i-th run of pixels.
- (Not dissimilar to zero length suppression above.)

Run-length Encoding Example

Original sequence:
`111122233333311112222`

can be encoded as:
`(1,4),(2,3),(3,6),(1,4),(2,4)`

How much compression?

The savings are dependent on the data: In the worst case (random noise) encoding is more heavy than original file:
$2\times$integer rather than $1\times$integer if original data is integer vector/array.

MATLAB example code:
rle.m (run-length encode) , rld.m (run-length decode)

- This is a simple form of statistical encoding.
- Here we substitute a frequently repeating pattern(s) with a code.
- The code is shorter than the pattern giving us compression.

The simplest scheme could employ predefined codes:

Example: Basic pattern substitution

Replace all occurrences of pattern of characters ‘and’ with the predefined code ’&’. So:

```
and you and I
```

becomes:

```
& you & I
```

Reducing number of bits per symbol

For the sake of example, consider character sequences here. (Other token streams can be used — *e.g.* vectorised image blocks, binary streams.)

Example: Compression ASCII Characters EIEIO

$\overbrace{01000101}^{\texttt{E(69)}}\overbrace{01001001}^{\texttt{I(73)}}\overbrace{01000101}^{\texttt{E(69)}}\overbrace{01001001}^{\texttt{I(73)}}\overbrace{01001111}^{\texttt{O(79)}} = 5 \times 8 = 40$ bits.

To compress, we aim to find a way to describe the same information using fewer bits per symbol, *e.g.*:

$$\overbrace{\texttt{xx}}^{\texttt{E (2 bits)}} \quad \overbrace{\texttt{yy}}^{\texttt{I (2 bits)}} \quad \overbrace{\texttt{xx}}^{\texttt{E (2 bits)}} \quad \overbrace{\texttt{yy}}^{\texttt{I (2 bits)}} \quad \overbrace{\texttt{zzz}}^{\texttt{O (3 bits)}} \quad =$$

$\overbrace{(2 \times 2)}^{2\times\texttt{E}} + \overbrace{(2 \times 2)}^{2\times\texttt{I}} + \overbrace{3}^{O} = 11$ bits.

Code assignment

- A predefined codebook may be used, i.e. assign code c_i to symbol s_i. (*E.g.* some dictionary of common words/tokens).
- Better: dynamically determine best codes from data.
- The entropy encoding schemes (next topic) basically attempt to decide the optimum assignment of codes to achieve the best compression.

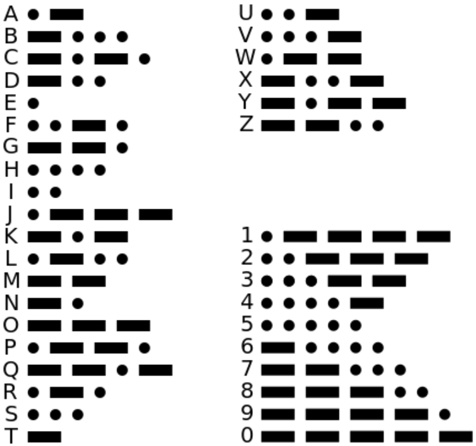
Example:

- Count occurrence of tokens (to estimate probabilities).
- Assign shorter codes to more probable symbols and vice versa.

Ideally we should aim to achieve Shannon's limit: $-log_b p$!

Morse code

Morse code makes an attempt to approach optimal code length: observe that frequent characters (E, T, . . .) are encoded with few dots/dashes and vice versa:



- This is a basic entropy coding algorithm.
- A simple example will be used to illustrate the algorithm:

Example:

Consider a finite string S over alphabet $\{$A, B, C, D, E$\}$:

$$S = \text{ACABADADEAABBAAAEDCACDEAAABCDBBEDCBACAE}$$

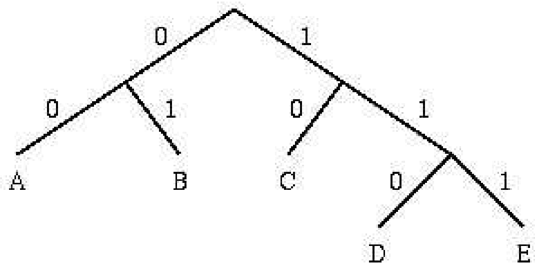
Count the symbols in the string:

Symbol	A	B	C	D	E
Count	15	7	6	6	5

Encoding with the Shannon-Fano algorithm

A top-down approach:

1. Sort symbols according to their frequencies/probabilities, *e.g.* ABCDE.
2. Recursively divide into two parts, each with approximately same number of counts, *i.e.* split in two so as to minimise difference in counts. Left group gets 0, right group gets 1.



Shannon-Fano algorithm

3. Assemble codebook by depth first traversal of the tree:

Symbol	Count	log(1/p)	Code	# of bits
A	15	1.38	00	30
B	7	2.48	01	14
C	6	2.70	10	12
D	6	2.70	110	18
E	5	2.96	111	15
			TOTAL (# of bits):	89

4. Transmit codes instead of tokens. In this case:

- Naïvely at 8 bits per char: $8 \times 39 = 312$ bits.
- Naïvely at $\lceil \log_2 5 \rceil = 3$ bits per char: $3 \times 39 = 117$ bits.
- SF-coded length = **89 bits**.

Shannon-Fano Algorithm: entropy

For the above example:

$$\begin{aligned}\text{Shannon entropy} &= (15 \times 1.38 + 7 \times 2.48 + 6 \times 2.7 \\ &\quad +6 \times 2.7 + 5 \times 2.96)/39 \\ &= 85.26/39 \\ &= \mathbf{2.19}.\end{aligned}$$

Number of bits needed for Shannon-Fano coding is: $89/39 = \mathbf{2.28}$.

Shannon-Fano Algorithm: discussion

Consider best case example:

- If we could always subdivide exactly in half, we would get ideal code:
  - Each 0/1 in the code would exactly reduce the uncertainty by a factor 2, so transmit 1 bit.
- Otherwise, when counts are only approximately equal, we get only good, but not ideal code.
- Compare with a fair vs biased coin.

Huffman algorithm

Can we do better than Shannon-Fano?

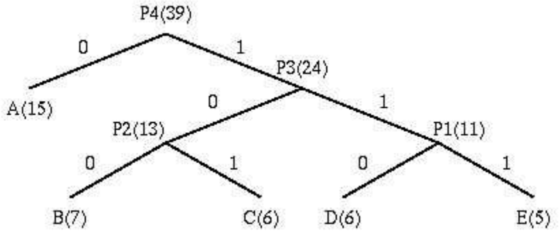
Huffman algorithm! Always produces best binary tree for given probabilities.

A bottom-up approach:

1. Initialization: put all nodes in a list L, keep it sorted at all times (e.g., ABCDE).
2. Repeat until the list L has more than one node left:
   - From L pick two nodes having the lowest frequencies/probabilities, create a parent node of them.
   - Assign the sum of the children's frequencies/probabilities to the parent node and insert it into L.
   - Assign code 0/1 to the two branches of the tree, and delete the children from L.
3. Coding of each node is a top-down label of branch labels.

Huffman encoding example

ACABADADEAABBAAAEDCACDEAAABCDBBEDCBACAE (same string as in Shannon-Fano example)



Symbol	Count	log(1/p)	Code	# of bits
A	15	1.38	0	15
B	7	2.48	100	21
C	6	2.70	101	18
D	6	2.70	110	18
E	5	2.96	111	15

TOTAL (# of bits): 87

Huffman encoding discussion

The following points are worth noting about the above algorithm:

- Decoding for the above two algorithms is trivial as long as the coding table/book is sent before the data.
  - There is a bit of an overhead for sending this.
  - But negligible if $|\text{string}| \gg |\text{alphabet}|$.
- Unique prefix property: no code is a prefix to any other code (all symbols are at the leaf nodes) $\rightarrow$ great for decoder, unambiguous.
- If prior statistics are available and accurate, then Huffman coding is very good.

For the above example:

$$\begin{aligned}\text{Shannon entropy} &= (15 \times 1.38 + 7 \times 2.48 + 6 \times 2.7 \\ &\quad +6 \times 2.7 + 5 \times 2.96)/39 \\ &= 85.26/39 \\ &= \mathbf{2.19}.\end{aligned}$$

Number of bits needed for Huffman Coding is: $87/39 = \mathbf{2.23}$.

Huffman coding of images

In order to encode images:

- Divide image up into (typically) 8×8 blocks.
- Each block is a symbol to be coded.
- Compute Huffman codes for set of blocks.
- Encode blocks accordingly.
- In JPEG: blocks are DCT coded first before Huffman may be applied (more soon).

Coding image in blocks is common to all image coding methods.

MATLAB Huffman coding example:
huffman.m (used with JPEG code later),
huffman.zip (alternative with tree plotting).

What is wrong with Huffman?

- Shannon-Fano or Huffman coding use an integer number (k) of binary digits for each symbol, hence k is never less than 1.
  - Ideal code according to Shannon may not be an integer number of binary digits!

Example: Huffman failure case

- Consider a biased coin with $p_{\text{heads}} = q = 0.999$ and $p_{\text{tails}} = 1 - q$.
- Suppose we use Huffman to generate codes for heads and tails and send 1000 heads.
- This would require 1000 ones and zeros with Huffman!
- Shannon tells us: ideally this should be $-\log_2 p_{\text{heads}} \approx 0.00144$ ones and zeros, so ≈ 1.44 for entire string.

Solution: arithmetic coding.

- A widely used entropy coder.
- Also used in JPEG — more soon.
- Only problem is its speed due possibly complex computations due to large symbol tables.
- Good compression ratio (better than Huffman coding), entropy around the Shannon ideal value.

Arithmetic coding: basic idea

The idea behind arithmetic coding is: encode the entire message into a single number, n, $(0.0 \leq n < 1.0)$.

- Consider a probability line segment, [0...1), and
- Assign to every symbol a range in this interval:
- Range proportional to probability with
- Position at cumulative probability.

Once we have defined the ranges and the probability line:

- Start to encode symbols.
- Every symbol defines where the output real number lands within the range.

Arithmetic coding example

Assume we have the following string: BACA
Therefore:

- A occurs with probability 0.5.
- B and C with probabilities 0.25.

Start by assigning each symbol to the probability range [0...1).

- Sort symbols highest probability first:

Symbol	Range
A	[0.0, 0.5)
B	[0.5, 0.75)
C	[0.75, 1.0)

- The first symbol in our example stream is B

We now know that the code will be in the range 0.5 to $0.74999\ldots$

Range is not yet unique.

- Need to narrow down the range to give us a unique code.

Arithmetic coding iteration:

- Subdivide the range for the first symbol given the probabilities of the second symbol then the symbol etc.

For all the symbols:

```
range = high - low;
high = low + range * high_range of the symbol being coded;
low = low + range * low_range of the symbol being coded;
```

Where:

- `range`, keeps track of where the next range should be.
- `high` and `low`, specify the output number.
- Initially `high = 1.0`, `low = 0.0`

Arithmetic coding example

For the second symbol we have:
(now `range` = 0.25, `low` = 0.5, `high` = 0.75):

Symbol	Range
B**A**	[0.5, 0.625)
BB	[0.625, 0.6875)
BC	[0.6875, 0.75)

We now reapply the subdivision of our scale again to get for our third symbol:
(`range` = 0.125, `low` = 0.5, `high` = 0.625):

Symbol	Range
BAA	[0.5, 0.5625)
BAB	[0.5625, 0.59375)
BA**C**	[0.59375, 0.625)

Arithmetic coding example

Subdivide again:
(`range` = 0.03125, `low` = 0.59375, `high` = 0.625):

Symbol	Range
BAC**A**	[0.59375, 0.60937)
BACB	[0.609375, 0.6171875)
BACC	[0.6171875, 0.625)

So the (unique) output code for BACA is any number in the range:

[0.59375, 0.60937).

Decoding

To **decode** is essentially the opposite:

- We compile the table for the sequence given probabilities.
- Find the range of number within which the code number lies and carry on.

Binary arithmetic coding

This is very similar to above:

- Except we use binary fractions.

Binary fractions are simply an extension of the binary systems into fractions much like decimal fractions. CM1101!

Fractions in **decimal**:
0.1 decimal = $\frac{1}{10^1} = 1/10$
0.01 decimal = $\frac{1}{10^2} = 1/100$
0.11 decimal = $\frac{1}{10^1} + \frac{1}{10^2} = 11/100$

So in **binary** we get:

0.1 binary = $\frac{1}{2^1} = 1/2$ decimal
0.01 binary = $\frac{1}{2^2} = 1/4$ decimal
0.11 binary = $\frac{1}{2^1} + \frac{1}{2^2}$ = 3/4 decimal

Binary arithmetic coding example

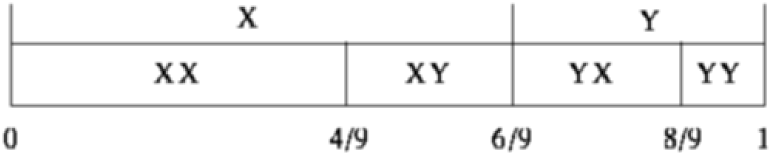
- Idea: Suppose alphabet was `X, Y` and consider stream:

  XXY

  Therefore:

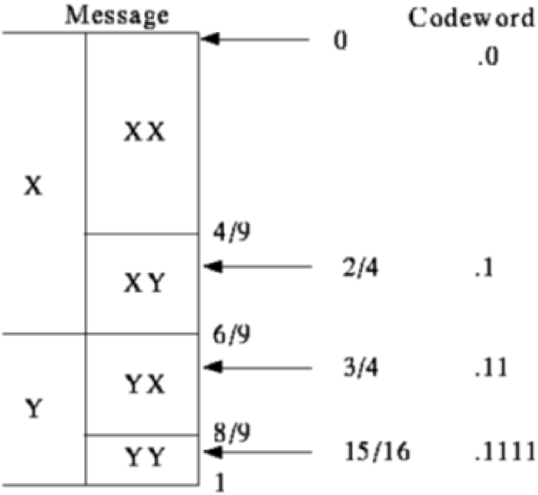
  ```
  prob(X) = 2/3
  prob(Y) = 1/3
  ```

- If we are only concerned with encoding length 2 messages, then we can map all possible messages to intervals in the range [0...1):

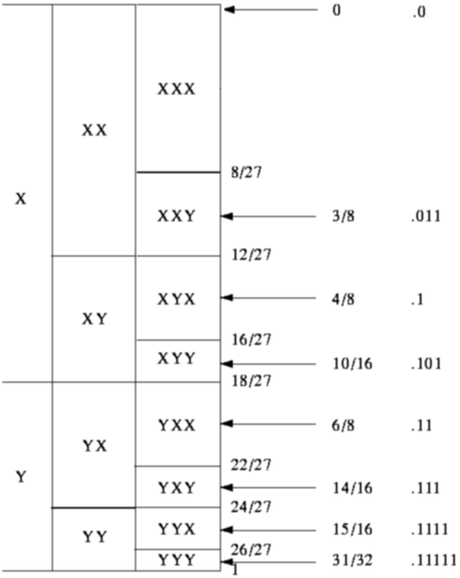


Binary arithmetic coding example

- To encode message, just send enough bits of a binary fraction that uniquely specifies the interval.



Binary arithmetic coding example



Similarly, we can map all possible length 3 messages to intervals in the range [0…1)

Code generation

- How to select a binary code for an interval?
- Let [L, H) be the final interval.
- Since they differ, the binary representation will be different starting from some digit (namely, 0 for L and 1 for H):

$$L = 0.d_1 d_2 d_3 \dots d_{t-1} 0 \dots$$

$$H = 0.d_1 d_2 d_3 \dots d_{t-1} 1 \dots$$

- We can select and transmit the t bits: $d_1 d_2 d_3 \dots d_{t-1} 1$.

- In general, number of bits is determined by the size of the interval. Asymptotically arithmetic code approaches ideal entropy: $-\log_2 p$ bits to represent interval of size p.
- Computation can be memory and CPU intensive.
- Resolution of the number we represent is limited by FPU precision.
  - So, write your own arbitrary precision arithmetic.
  - Or dynamically renormalise: when range is reduced so that all values in the range share certain beginning digits — send those. Then shift left and thus regain precision.

MATLAB Arithmetic coding examples:
Arith06.m (version 1), Arith07.m (version 2), `arithenco.m`

Lempel-Ziv-Welch (LZW) algorithm

- A very common compression technique.
- Used in GIF files (LZW), Adobe PDF file (LZW), UNIX `compress` (LZ Only)
- Patented — LZW not LZ. Patent expired in 2003/2004.

Basic idea/analogy:

Suppose we want to encode the Oxford Concise English dictionary which contains about 159,000 entries.

$$\lceil \log_2 159,000 \rceil = 18 \text{ bits.}$$

Why not just transmit each word as an 18 bit number?

LZW constructs its own dictionary

Problems:

- Too many bits per word,
- Everyone needs a dictionary to decode back to English.
- Only works for English text.

Solution:

- Find a way to build the dictionary adaptively.
- Original methods (LZ) due to Lempel and Ziv in 1977/8.
- Quite a few variations on LZ.
- Terry Welch improvement (1984), patented LZW algorithm
  - LZW introduced the idea that only the initial dictionary needs to be transmitted to enable decoding:
    The decoder is able to build the rest of the table from the encoded sequence.

LZW compression algorithm

The LZW Compression Algorithm can be summarised as follows:

```
w = NIL;
while ( read a character k ) {
    if wk exists in the dictionary
        w = wk;
    else {
        add wk to the dictionary;
        output the code for w;
        w = k;
    }
}
```

- Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.

LZW compression algorithm example:

Input string is "^WED^WE^WEE^WEB^WET".

w	k	output	index	symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

- A 19-symbol input has been reduced to 7-symbol plus 5-code output. Each code/symbol will need more than 8 bits, say 9 bits.
- Usually, compression doesn't start until a large number of bytes (*e.g.* > 100) are read in.

LZW decompression algorithm (simplified)

The LZW decompression algorithm is as follows:

```
read a character k;
output k;
w = k;
while ( read a character k )
/* k could be a character or a code. */
{
    entry = dictionary entry for k;
    output entry;
    add w + entry[0] to dictionary;
    w = entry;
}
```

Note: LZW decoder only needs the initial dictionary. The decoder is able to build the rest of the table from the encoded sequence.

LZW decompression algorithm example:

```
Input string is: "^WED<256>E<260><261><257>B<260>T"
     w      k    output    index    symbol
  ---------------------------------------
     ^      ^
     ^      W       W       256       ^W
     W      E       E       257       WE
     E      D       D       258       ED
     D    <256>    ^W       259       D^
   <256>    E       E       260      ^WE
     E    <260>   ^WE       261       E^
   <260>  <261>    E^       262     ^WEE
   <261>  <257>    WE       263      E^W
   <257>    B       B       264      WEB
     B    <260>   ^WE       265       B^
   <260>    T       T       266     ^WET
```

LZW decompression algorithm (proper)

```
read a character k;
output k;
w = k;
while ( read a character k )
/* k could be a character or a code. */
{
    entry = dictionary entry for k;
    /* Special case */
    if (entry == NIL) // Not found
        entry = w + w[0];

    output entry;
    if (w != NIL)
        add w + entry[0] to dictionary;

    w = entry;
}
```

MATLAB LZW code

norm2lzw.m: LZW Encoder

lzw2norm.m: LZW Decoder

lzw_demo1.m: Full MATLAB demo

Source coding techniques

Source coding is based on changing the content of the original signal.

Compression rates may be higher but at a price of loss of information. Good compression rates may be achieved with source encoding with (occasionally) lossless or (mostly) little perceivable loss of information.

Some broad methods that exist:

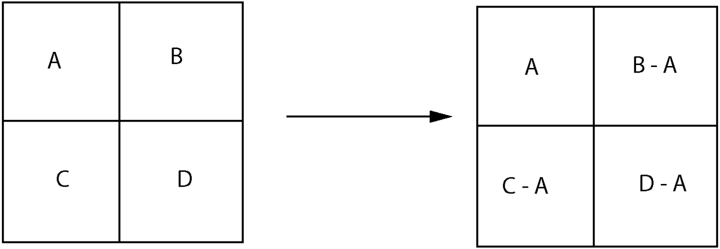
- Transform coding.
- Differential encoding.
- Vector quantisation.

Transform coding example

Consider a simple example transform:

A Simple Transform Encoding procedure maybe described by the following steps for a 2×2 block of gray scale pixels:

1. Take top left pixel as the base value for the block, pixel A.
2. Calculate three other transformed values by taking the difference between these (respective) pixels and pixel A, i.e. $B - A$, $C - A$, $D - A$.
3. Store the base pixel and the differences as the values of the transform.



Transform coding example

Given the above we can easily form the forward transform:

$$\begin{aligned} X_0 &= A \\ X_1 &= B - A \\ X_2 &= C - A \\ X_3 &= D - A \end{aligned}$$

and the inverse transform is:

$$\begin{aligned} A &= X_0 \\ B &= X_1 + X_0 \\ C &= X_2 + X_0 \\ D &= X_3 + X_0 \end{aligned}$$

Compressing data with this transform?

Exploit redundancy in the data:

- Redundancy transformed to values, X_i.
- Statistics of differences will hopefully be more amenable to entropy coding.
- Compress the data by using fewer bits to represent the differences — quantisation.
  - *E.g.* if we use 8 bits per pixel then the 2$\times$2 block uses 32 bits
  - If we keep 8 bits for the base pixel, X_0,
  - Assign 4 bits for each difference then we only use 20 bits.
  - Better with an average 5 bits/pixel.

Transform coding example

Consider the following 4×4 image block:

120	130
125	120

then we get:

$$
\begin{aligned}
X_0 &= 120 \\
X_1 &= 10 \\
X_2 &= 5 \\
X_3 &= 0
\end{aligned}
$$

We can then compress these values by taking fewer bits to represent the data.

Transform coding example: discussion

- It is too simple — not applicable to slightly more complex cases.
- Needs to operate on larger blocks (typically 8$\times$8 minimum).
- Simple encoding of differences for large values will result in loss of information.
  - Poor losses possible here with 4 bits per pixel = values $0 \ldots 15$ unsigned,
  - Signed value range: $-8 \ldots 7$ so either quantise in larger step value or massive overflow!

Practical approaches: use more complicated transforms *e.g.* DCT.

Differential transform coding schemes

- Differencing is used in some compression algorithms:
  - Later part of JPEG compression.
  - Exploit static parts (*e.g.* background) in MPEG video.
  - Some speech coding and other simple signals.
- Good on repetitive sequences.
- Poor on highly varying data sequences.
  - *E.g.* audio/video signals.

MATLAB simple vector differential example:

diffencodevec.m: Differential Encoder

diffdecodevec.m: Differential Decoder

diffencodevecTest.m: Differential Test Example

Differential encoding

Simple example of transform coding mentioned earlier is an instance of this approach.

- The difference between the actual value of a sample and a prediction of that values is encoded.
- Also known as predictive encoding.
- Example of technique include: differential pulse code modulation, delta modulation, and adaptive pulse code modulation — differ in prediction part.
- Suitable where successive signal samples do not differ much, but are not zero. *E.g.* video — difference between frames, some audio signals.

Differential pulse code modulation (DPCM)

Simple prediction (also used in JPEG):

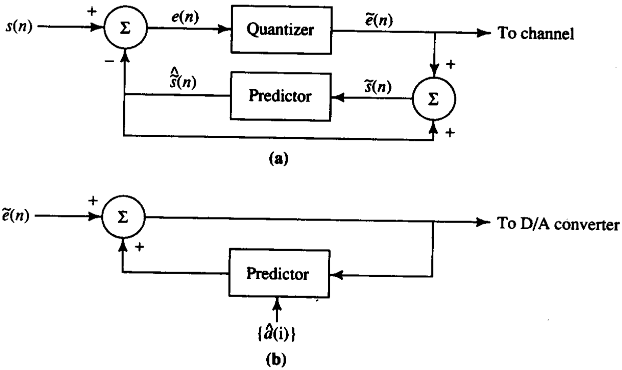
$$f_{\text{predict}}(t_i) = f_{\text{actual}}(t_{i-1})$$

I.e. a simple Markov model where current value is the predict next value. So we simply need to encode:

$$\Delta f(t_i) = f_{\text{actual}}(t_i) - f_{\text{actual}}(t_{i-1})$$

If successive sample are close to each other we only need to encode first sample with a large number of bits:

Simple DPCM



Actual data: 9 10 7 6

Predicted data: 0 9 10 7

$\Delta f(t)$: +9, +1, -3, -1.

MATLAB DPCM Example (with quantisation):

dpcm_demo.m, dpcm.zip.m:

- Delta modulation is a special case of DPCM:
  - Same predictor function.
  - Coding error is a single bit that indicates the current sample should be increased or decreased by a step.
  - Not suitable for rapidly changing signals.
- Adaptive pulse code modulation
  Better temporal/Markov model:
  - Data is extracted from a function of a series of previous values.
  - *E.g*. average of last n samples.
  - Characteristics of sample better preserved.

Frequency domain methods

another form of transform coding

Transformation from one domain — time (*e.g.* 1D audio, video: 2D imagery over time) or spatial (*e.g.* 2D imagery) domain to the frequency domain via

- Discrete Cosine Transform (DCT)— Heart of JPEG and MPEG Video.
- Fourier Transform (FT) — MPEG Audio.

Theory already studied earlier

Recap: compression in frequency space

How do we achieve compression?

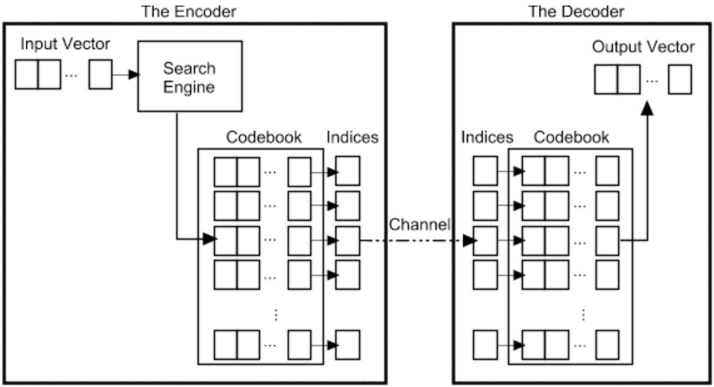
- Low pass filter — ignore high frequency noise components.
- Only store lower frequency components.
- High Pass Filter — spot gradual changes.
- If changes to low eye does not respond so ignore?

Vector quantisation

The basic outline of this approach is:

- Data stream divided into (1D or 2D square) blocks — regard them as vectors.
- A table or code book is used to find a pattern for each vector (block).
- Code book can be dynamically constructed or predefined.
- Each pattern for as a lookup value in table.
- Compression achieved as data is effectively subsampled and coded at this level.
- Used in MPEG4, Video Codecs (Cinepak, Sorenson), Speech coding, Ogg Vorbis.

Vector quantisation encoding/decoding

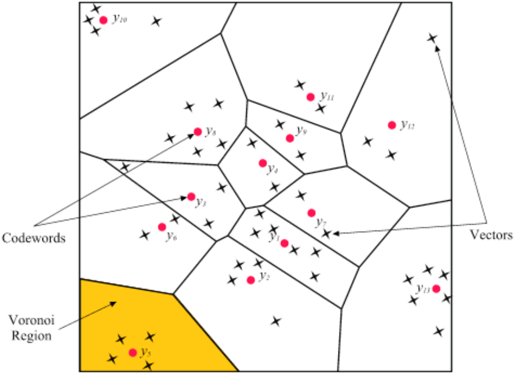


- Search Engine:
  - Group (cluster) data into vectors.
  - Find closest code vectors.
- When decoding, output needs to be unblocked (smoothed).

Vector quantisation code book construction

How to cluster data?

- Use some clustering technique,
  e.g. K-means, Voronoi decomposition
  Essentially cluster on some closeness measure, minimise inter-sample variance or distance.



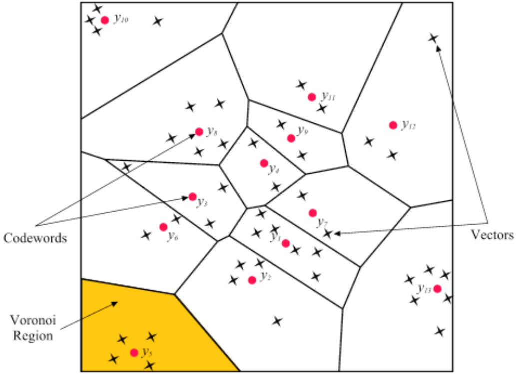
This is an iterative algorithm:

- Assign each point to the cluster whose centroid yields the least within-cluster squared distance. (This partitions according to **Voronoi** diagram with seeds = centroids.)
- Update: set new centroids to be the centroids of each cluster.

Vector Quantisation Code Book Construction

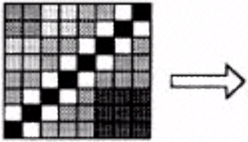
How to code?

- For each cluster choose a mean (median) point as representative code for all points in cluster.



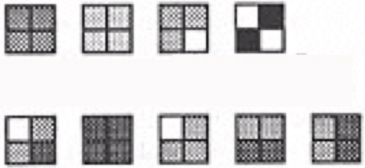
Vector quantisation image coding example

- A small block of images and intensity values



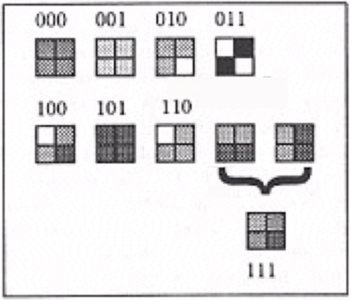
145	145	216	216	181	181	240	0
145	145	216	216	181	240	0	240
216	216	181	181	240	0	240	181
216	216	181	240	0	240	181	181
181	181	240	0	240	181	181	181
181	240	0	240	181	96	96	96
240	0	240	181	181	96	96	96
0	240	181	181	181	96	96	96

- Consider Vectors of 2x2 blocks, and only allow 8 codes in table.
- 9 vector blocks present in above:



Vector quantisation image coding example

- 9 vector blocks, so only one has to be vector quantised here.
- Resulting code book for above image



MATLAB example: vectorquantise.m