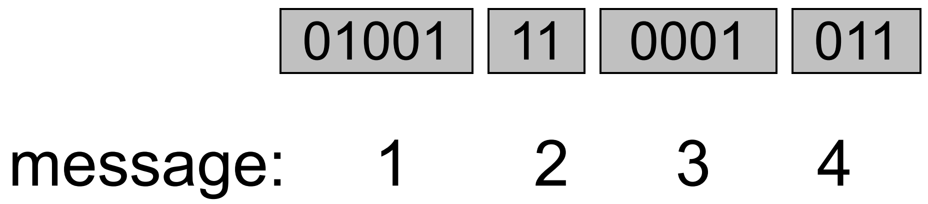
15-750:Algorithms in the Real World

Data Compression: Lecture 2

(Recap) Discrete or Blended

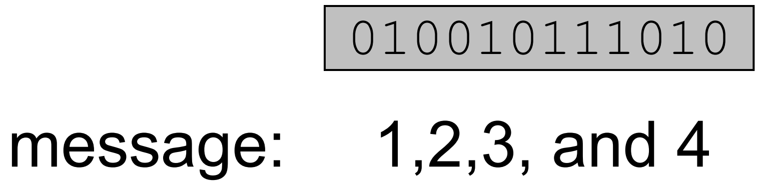
Discrete: each message is a fixed set of bits

– E.g., Huffman coding, Shannon-Fano coding



Blended: bits can be "shared" among messages

– E.g., Arithmetic coding



(Recap) Arithmetic Coding

Allows “blending” of bits in a message sequence.

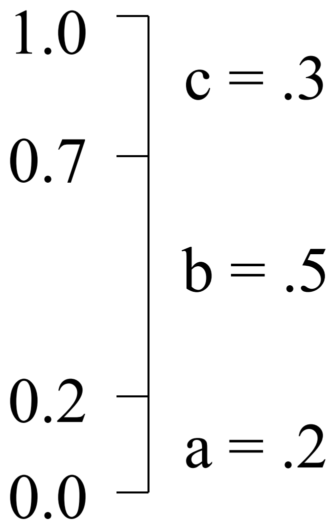
More expensive than Huffman coding

Used in many compression algorithms: E.g., JPEG/MPEG

Arithmetic Coding: message intervals

Assign each probability distribution to an interval range from 0 (inclusive) to 1 (exclusive).

e.g.

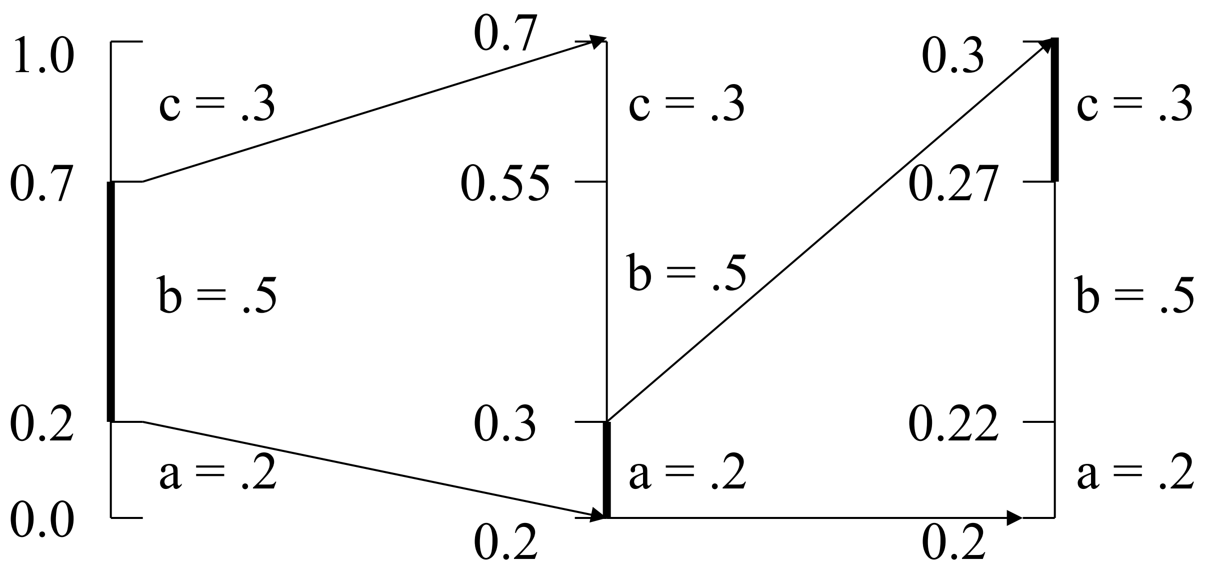


The interval for a particular message will be called the **message interval** (e.g for b the interval is [.2,.7))

Arithmetic Coding: Sequence intervals

Code a message sequence by composing intervals.

For example: **bac**



The final interval is **[.27,.3)**

Uniquely defining an interval

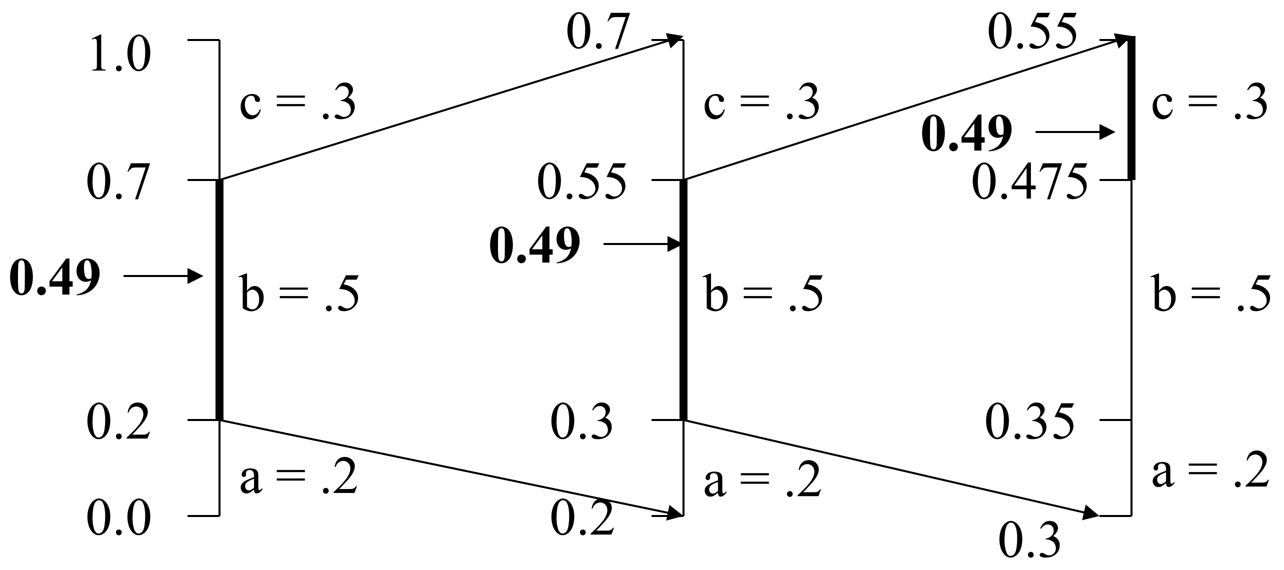
Important property: The sequence intervals for distinct message sequences of length *n* will never overlap

Therefore: specifying <u>any number in the final interval</u> uniquely determines the sequence.

Decoding is similar to encoding, but on each step need to determine what the message value is and then reduce interval

Arithmetic Coding: Decoding Example

Decoding the number .49, knowing the message is of length 3:



The message is **bbc.**

Transformation Techniques

1. Run length coding
2. Move-to-front coding
3. Residual coding
4. Burrows-Wheeler transform
5. Linear transform coding

Why transform?

Help skew the probabilities

In many algorithms message sequences are transformed into **integers with a skew towards small integers**

We will take a detour to study codes for integers ...

Integer codes

- There are several “fixed” codes for encoding natural numbers
- With non-decreasing codeword lengths

Integer codes: binary

n	Binary
1	..001
2	..010
3	..011
4	..100
5	..101
6	..110

“Minimal” binary representation: Drop leading zeros

Q: What is the problem with minimal binary representation?

Not a prefix code!

Integer codes: Unary

n	Binary	Unary
1	..001	0
2	..010	10
3	..011	110
4	..100	1110
5	..101	11110
6	..110	111110

n represented as n-1 ones and one 0
(0's and 1's can be interchanged)

Q: For what probability distribution unary codes are optimal prefix codes?

Integer codes: Gamma

n	Binary	Unary	Gamma
1	..001	0	0|
2	..010	10	10|0
3	..011	110	10|1
4	..100	1110	110|00
5	..101	11110	110|01
6	..110	111110	110|10

Many other fixed prefix codes:
  Golomb, phased-binary, subexponential, ...

Back to **transforming data** for encoding…

Transformation Techniques

1. Run length coding
2. Move-to-front coding
3. Residual coding
4. Burrows-Wheeler transform
5. Linear transform coding

1. Run Length Coding

Code by specifying message value followed by the number of repeated values:

e.g. **abbbaacccca => (a,1),(b,3),(a,2),(c,4),(a,1)**

The characters and counts can be coded based on frequency (i.e., probability coding).

Typically low counts such as 1 and 2 are more common => use small number of bits overhead for these.

Used as a sub-step in many compression algorithms.

2. Move to Front Coding

- Transforms message sequence into sequence of integers
- Then probability code

Start with values in a total order: e.g.: [a,b,c,d,…]

For each message

- output the position in the order
- move to the front of the order.

e.g.: **c a**

c => output: 3, new order: [c,a,b,d,e,…]
a => output: 2, new order: [a,c,b,d,e,…]

Probability code the output.

2. Move to Front Coding

The hope is that there is a bias for small numbers.

Q: Why?

Temporal locality

Takes advantage of **temporal locality**

Used as a sub-step in many compression algorithms.

3. Residual Coding

Typically used for message values that represent some sort of amplitude:
e.g. gray-level in an image, or amplitude in audio.

Basic Idea:

- Guess next value based on current context.
- Output difference between guess and actual value.
- Use probability code on the output.

E.g.: Consider compressing a stock value over time.

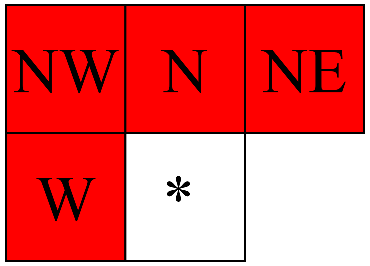
Residual coding is used in JPEG Lossless

Use of residual coding in JPEG-LS

JPEG Lossless

Codes in Raster Order.

Uses 4 pixels as context:



Tries to guess value of * based on W, NW, N and NE.

The residual between guessed and actual value is found and then coded using a Golomb-like code.

(Golomb codes are similar to Gamma codes)

4. Burrows –Wheeler Transform

Breaks file into fixed-size blocks and encodes each block separately.

For each block:

- Create full context for each character (wraps around)
- Reverse lexical sort each character by its full context. This is called the “block sorting transform”.

Burrows Wheeler: Example

To encode: **$d_1e_2c_3o_4d_5e_6$**

(Numbered the characters to distinguish them.)

Context "wraps" around. Last char is most significant.

Context	Char
$ecode_6$	d_1
$coded_1$	e_2
$odede_2$	c_3
$dedec_3$	o_4
$edeco_4$	d_5
$decod_5$	e_6

Sort Context $\longrightarrow$

Context	Output
$dedec_3$	o_4
$coded_1$	e_2
$decod_5$	e_6
$odede_2$	c_3
$ecode_6$	d_1 $\Leftarrow$
$edeco_4$	d_5

Q: Why is the output more easier to compress?

Burrows Wheeler: Example

Context	Char
ecode$_6$	d$_1$
coded$_1$	e$_2$
odede$_2$	c$_3$
dedec$_3$	o$_4$
edeco$_4$	d$_5$
decod$_5$	e$_6$

Sort Context $\longrightarrow$

Context	Output	
dedec$_3$	o$_4$	
coded$_1$	e$_2$	
decod$_5$	e$_6$	
odede$_2$	c$_3$	
ecode$_6$	d$_1$	$\Leftarrow$
edeco$_4$	d$_5$	

Gets similar characters together
(because we are ordering by context)

Why not just sort?

Can we invert BW Transform?

Output

o_4

e_2

e_6

c_3

d_1 $\Leftarrow$

d_5

Can we invert BW Transform?

Suppose we are given the context… then?

Context	Output
c_3	o_4
d_1	e_2
d_5	e_6
e_2	c_3
e_6	d_1 $\Leftarrow$
o_4	d_5

How can we get the last column of the context column from the output column?

Sort!

Any problem? Equal valued chars

Burrows-Wheeler (Continued)

Theorem: After sorting, equal valued characters appear in the same order in the output column as in the last column of the sorted context.

Proof sketch:

The chars with equal value in the most-significant-position (i.e., last column) of the context will be ordered by the rest of the context, i.e. the previous chars.

This is also the order of the output since it is sorted by the previous characters.

Context	Output
$\texttt{dedec}_3$	$\texttt{o}_4$
code$\texttt{d}_1$	$\texttt{e}_2$
deco$\texttt{d}_5$	$\texttt{e}_6$
$\texttt{odede}_2$	$\texttt{c}_3$
ecode$_6$	$\texttt{d}_1$
edeco$_4$	$\texttt{d}_5$

Burrows-Wheeler: Decoding

– What follows the underlined **a** ?

– What follows the underlined **b**?

– What is the whole string?

Context	**Output**	
a	c	
a	**b**	
a	b	
b	a	
b	**a**	⇐
c	a	

Answer: b, a, abacab

BZIP

Transform 1: (Burrows Wheeler)

– **input** : character string (block)
– **output** : reordered character string

Transform 2: (move to front)

– **input** : character string
– **output** : MTF numbering

Transform 3: (run length)

– **input** : MTF numbering
– **output** : sequence of run lengths

Probabilities: (on run lengths)

Dynamic based on counts for each block.

Coding: Originally arithmetic, but changed to Huffman in bzip2 due to patent concerns

5. Linear Transform Coding

Goal: Transform the data into a form that is easily compressible (through **lossless** or **lossy** compression)

Select a set of linear basis functions ϕ_i that span the space

- sin, cos, spherical harmonics, wavelets, …

Linear Transform Coding (continued)

Coefficients:

$$\Theta_i = \sum_j x_j \phi_i(j) = \sum_j x_j a_{ij}$$

Θ_i = i^{th} resulting coefficient

x_j = j^{th} input value

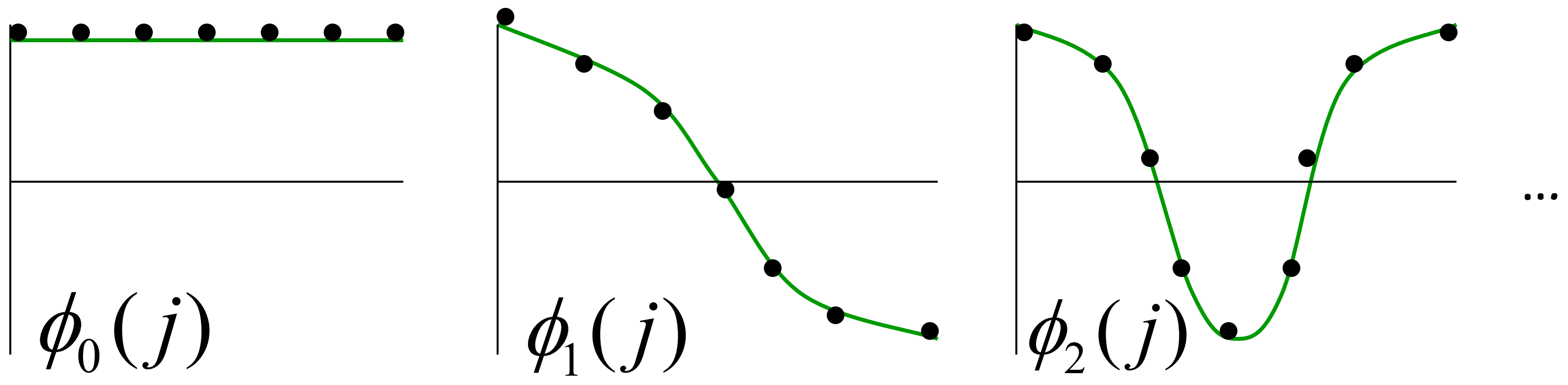
a_{ij} = ij^{th} transform coefficient = $\phi_i(j)$

In matrix notation: $\Theta = Ax$

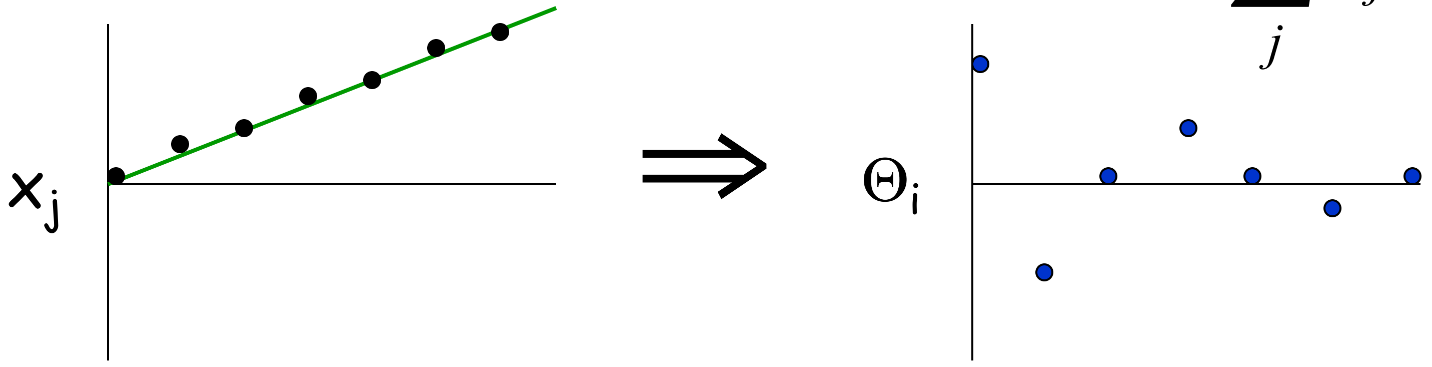
$$x = A^{-1}\Theta$$

Where A is an n x n "transform" matrix, and each row defines a basis function

Example: Cosine Transform

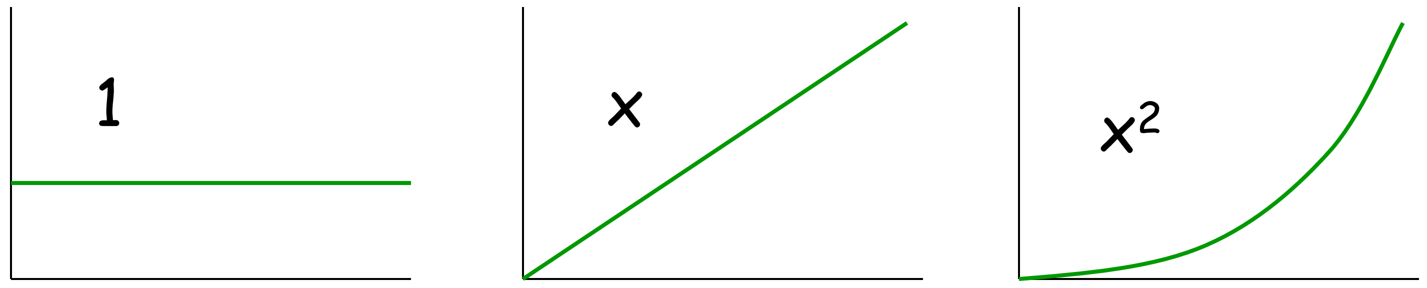


$$\Theta_i = \sum_j x_j \phi_i(j)$$

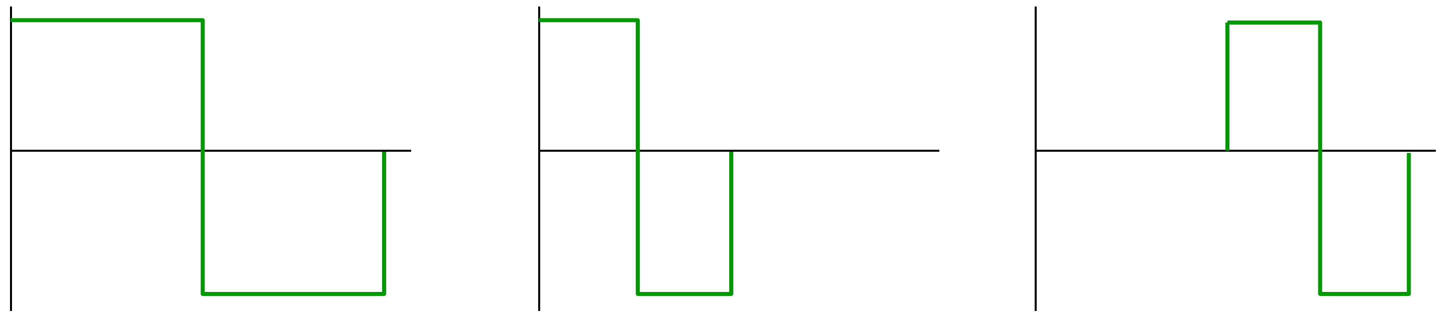


Other Transforms

Polynomial:



Wavelet (Haar):



How to Pick a Transform

Goals:

- Decorrelate the data
- Low coefficients for many terms
- Basis functions that can be ignored from the perception point-of-view

15-750:Algorithms in the Real World

Quantization (lossy)

Scalar Quantization

Quantize regions of values into a single value

E.g. Drop least significant bit

(Can be used to reduce # of bits for a pixel)

Q: Why is this lossy?

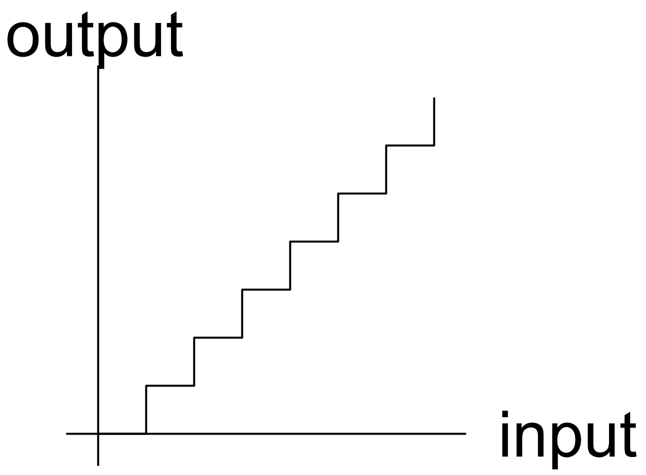
Many-to-one mapping

Two types

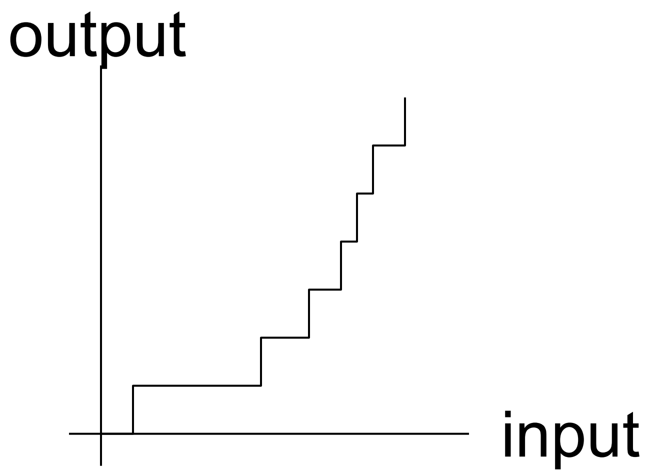
– Uniform: Mapping is linear

– Non-uniform: Mapping is non-linear

Scalar Quantization



uniform



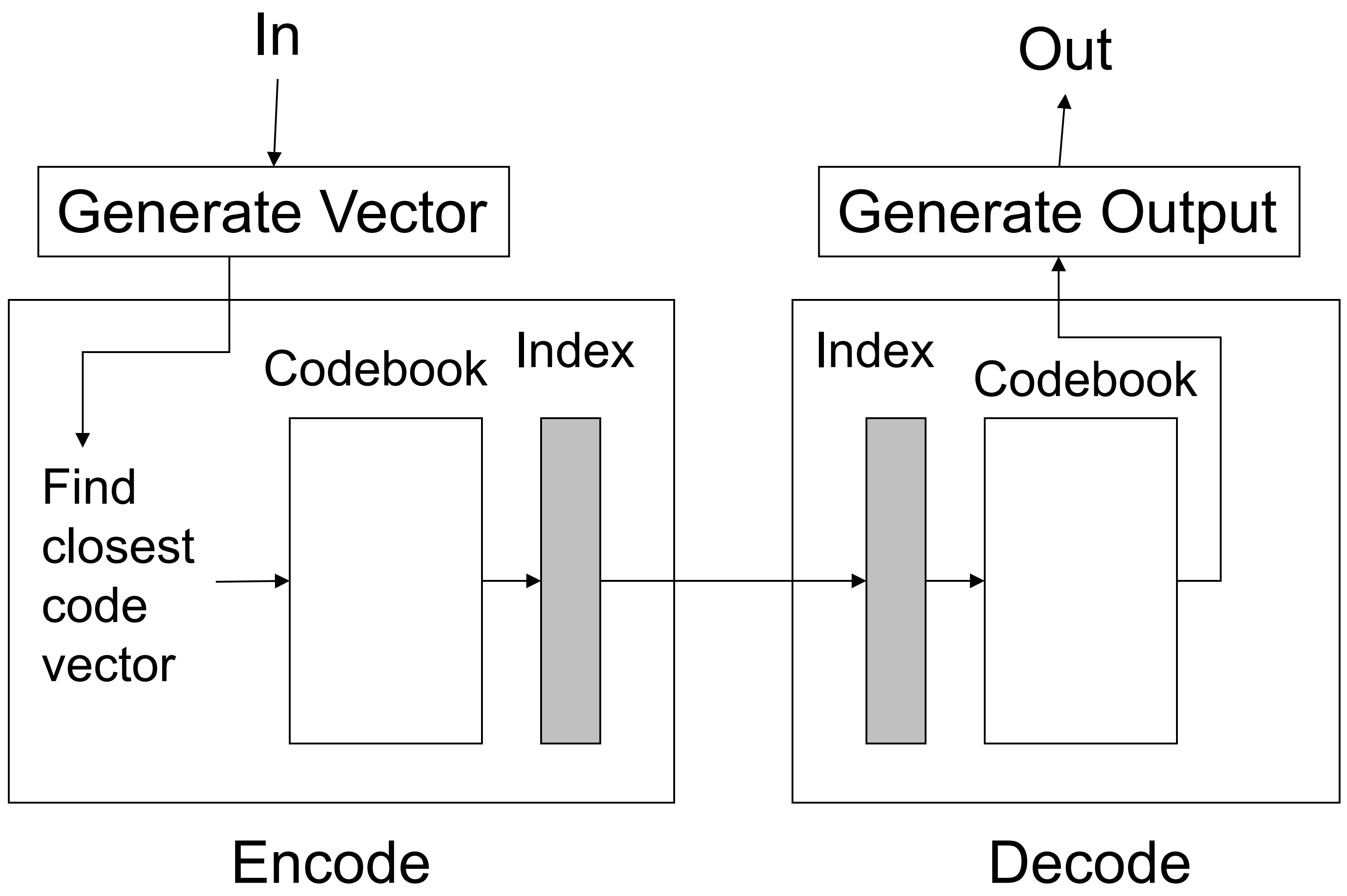
non uniform

Q: Why use non-uniform?
Error metric might be non-uniform.
E.g. Human eye sensitivity to specific color regions

Can formalize the mapping problem as an optimization problem

Vector Quantization

Mapping a multi-dimensional space into a smaller set of messages



Vector Quantization (VQ)

What do we use as vectors?

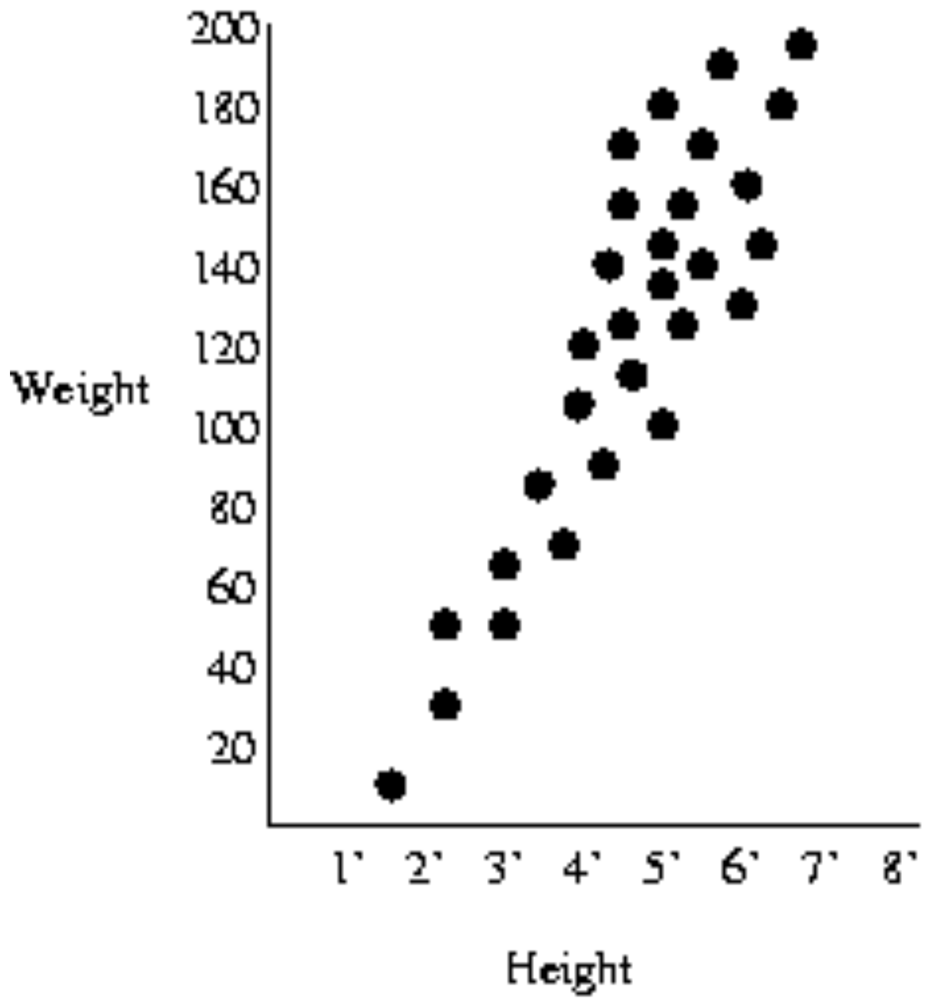
- Color (Red, Green, Blue)
  - Can be used, for example to reduce 24bits/pixel to 8bits/pixel
  - Used in some monitors to reduce data rate from the CPU (colormaps)
- K consecutive samples in audio
- Block of K pixels in an image

How do we decide on a codebook

- Typically done with **clustering**

VQ most effective when the variables along the dimensions of the space are correlated

Vector Quantization: Example



Observations:

1. Highly correlated:
   Concentration of representative points

2. Higher density is more common regions.

Case Study: JPEG

A nice example since it uses many techniques:

- Transform coding (Cosine transform)
- Scalar quantization
- Difference coding
- Run-length coding
- Huffman or arithmetic coding