Coding Theory

Jay Daigle

Occidental College

November 9, 2017

What is coding?

What is coding?

- *Not* a hidden message!

What is coding?

- *Not* a hidden message!
- Represent data in a regularized way.

What is coding?

- *Not* a hidden message!
- Represent data in a regularized way.
- Ideally gain extra features.

Formal Setup

Formal Setup

Definition

An alphabet Σ is a set of distinct symbols.

Formal Setup

Definition

An alphabet Σ is a set of distinct symbols.
Let X be the set of possible messages.

Formal Setup

Definition

An alphabet Σ is a set of distinct symbols.
Let X be the set of possible messages.
A code is a function $C : X \to \Sigma^*$ that converts a message into a finite string of sybmols.

Formal Setup

Definition

An alphabet Σ is a set of distinct symbols.
Let X be the set of possible messages.
A code is a function $C : X \to \Sigma^*$ that converts a message into a finite string of sybmols.

- Thoughts into strings of letters

Formal Setup

Definition

An alphabet Σ is a set of distinct symbols.
Let X be the set of possible messages.
A code is a function $C : X \to \Sigma^*$ that converts a message into a finite string of sybmols.

- Thoughts into strings of letters
- Letters into numbers

Formal Setup

Definition

An alphabet Σ is a set of distinct symbols.
Let X be the set of possible messages.
A code is a function $C : X \to \Sigma^*$ that converts a message into a finite string of sybmols.

- Thoughts into strings of letters
- Letters into numbers
- Pictures into binary strings

Types of Codes

- Numeric encoding

Types of Codes

- Numeric encoding
- Source coding

Types of Codes

- Numeric encoding
- Source coding
- Channel coding

Types of Codes

- Numeric encoding
- Source coding
- Channel coding

Why numeric encoding?

Why numeric encoding?

- Most cryptographic systems we've studied encrypt numbers.

Why numeric encoding?

- Most cryptographic systems we've studied encrypt numbers.
- Easy to turn binary strings into numbers; what about other messages?

Why numeric encoding?

- Most cryptographic systems we've studied encrypt numbers.
- Easy to turn binary strings into numbers; what about other messages?
- First developed for formal logic.

Why numeric encoding?

- Most cryptographic systems we've studied encrypt numbers.
- Easy to turn binary strings into numbers; what about other messages?
- First developed for formal logic. Gödel Incompleteness Theorem.

Gödel Numbers

- Relate logical sentences to number theory

Gödel Numbers

- Relate logical sentences to number theory
- Assign every message to a unique number

Gödel Numbers

- Relate logical sentences to number theory
- Assign every message to a unique number

Idea

Use prime factorization.

Gödel Numbers

- Relate logical sentences to number theory
- Assign every message to a unique number

Idea

Use prime factorization.

$$(1, 4, 3, 2, 7)$$

Gödel Numbers

- Relate logical sentences to number theory
- Assign every message to a unique number

Idea

Use prime factorization.

$$(1, 4, 3, 2, 7) \mapsto 2^1 \cdot 3^4 \cdot 5^3 \cdot 7^2 \cdot 11^7$$

Gödel Numbers

- Relate logical sentences to number theory
- Assign every message to a unique number

Idea

Use prime factorization.

$$(1, 4, 3, 2, 7) \mapsto 2^1 \cdot 3^4 \cdot 5^3 \cdot 7^2 \cdot 11^7 = 19,336,145,424,750$$

Gödel Numbers

- Relate logical sentences to number theory
- Assign every message to a unique number

Idea

Use prime factorization.

$$(1, 4, 3, 2, 7) \mapsto 2^1 \cdot 3^4 \cdot 5^3 \cdot 7^2 \cdot 11^7 = 19,336,145,424,750$$

Great for theoretical arguments. Not practical.

Gödel Numbers

- Relate logical sentences to number theory
- Assign every message to a unique number

Idea

Use prime factorization.

$$(1, 4, 3, 2, 7) \mapsto 2^1 \cdot 3^4 \cdot 5^3 \cdot 7^2 \cdot 11^7 = 19,336,145,424,750$$

Great for theoretical arguments. Not practical.

zzz

Gödel Numbers

- Relate logical sentences to number theory
- Assign every message to a unique number

Idea

Use prime factorization.

$$(1, 4, 3, 2, 7) \mapsto 2^1 \cdot 3^4 \cdot 5^3 \cdot 7^2 \cdot 11^7 = 19,336,145,424,750$$

Great for theoretical arguments. Not practical.

$$\text{zzz} \mapsto 2^2 53^2 55^2 5 = 847288609443000000000000000000000000$$

What is source coding?

What is source coding?

Goal

Data compression: output smaller than input.

What is source coding?

Goal

Data compression: output smaller than input.

- Lossless encoding: C is injective.

What is source coding?

Goal

Data compression: output smaller than input.

- Lossless encoding: C is injective.
- Lossy encoding: C is not injective.

Lossless Encoding

Lossless Encoding

Theoretical limits

Lossless Encoding

Theoretical limits

Entropy!

Lossless Encoding

Theoretical limits

Entropy! On average $C(x) \geq H(x)$.

Lossless Encoding

Theoretical limits

Entropy! On average $C(x) \geq H(x)$.
Possible because most data is redundant.

Lossless Encoding

Theoretical limits

Entropy! On average $C(x) \geq H(x)$.
Possible because most data is redundant. English is 70% redundant.

Lossless Encoding

Theoretical limits

Entropy! On average $C(x) \geq H(x)$.
Possible because most data is redundant. English is 70% redundant.
More common symbols get shorter codes.

Lossless Encoding

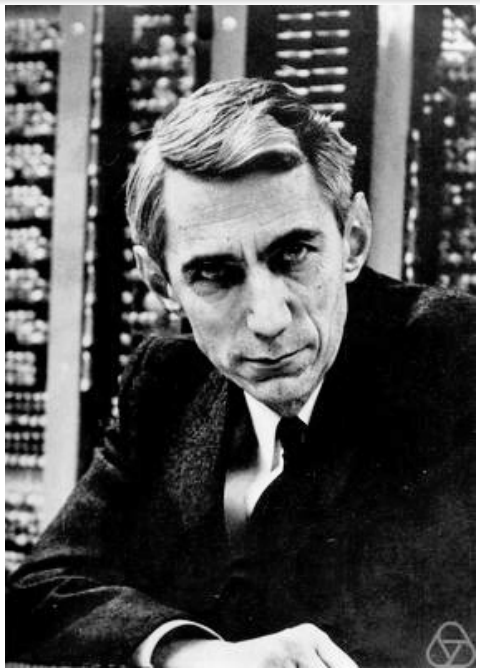
Theoretical limits

Entropy! On average $C(x) \geq H(x)$.
Possible because most data is redundant. English is 70% redundant.
More common symbols get shorter codes. Needs model of likely symbols.

Lossless Encoding

Theoretical limits

Entropy! On average $C(x) \geq H(x)$.
Possible because most data is redundant. English is 70% redundant.
More common symbols get shorter codes. Needs model of likely symbols.



Claude Shannon (1916 – 2001)

Shannon Coding

Shannon Coding

Shannon Coding (1948)

Shannon Coding

Shannon Coding (1948)

- First lossless encoding scheme

Shannon Coding

Shannon Coding (1948)

- First lossless encoding scheme
- Prefix-free code: no code word is the prefix to another code word

Shannon Coding

Shannon Coding (1948)

- First lossless encoding scheme
- Prefix-free code: no code word is the prefix to another code word
- Arrange messages by probability, assign string of $-\log_2 p_i$ binary digits based on probabilities of more likely messages.

Shannon Coding

Shannon Coding (1948)

- First lossless encoding scheme
- Prefix-free code: no code word is the prefix to another code word
- Arrange messages by probability, assign string of $-\log_2 p_i$ binary digits based on probabilities of more likely messages.

symbol i	p_i	$\lceil -\log_2 p_i \rceil$	$\sum_{n=0}^{i-1} p_n$	in binary	codeword
1	.36	2	0.0	.0000	00
2	.18	3	.36	.0101	010
3	.18	3	.54	.1000	100
4	.12	4	.72	.1011	1011
5	.09	4	.84	.1101	1101
6	.07	4	.93	.1110	1110

Other Lossless Encoding

Huffman Coding (1952)

- Lossless and prefix-free

Other Lossless Encoding

Huffman Coding (1952)

- Lossless and prefix-free
- Encodes using binary tree structure

Other Lossless Encoding

Huffman Coding (1952)

- Lossless and prefix-free
- Encodes using binary tree structure
- Provably optimal per-symbol: average length at least as good as any other coding scheme

Other Lossless Encoding

Huffman Coding (1952)

- Lossless and prefix-free
- Encodes using binary tree structure
- Provably optimal per-symbol: average length at least as good as any other coding scheme
- Not always optimal in real world

Other Lossless Encoding

Huffman Coding (1952)

- Lossless and prefix-free
- Encodes using binary tree structure
- Provably optimal per-symbol: average length at least as good as any other coding scheme
- Not always optimal in real world

Arithmetic Coding

- Lossless and prefix-free

Other Lossless Encoding

Huffman Coding (1952)

- Lossless and prefix-free
- Encodes using binary tree structure
- Provably optimal per-symbol: average length at least as good as any other coding scheme
- Not always optimal in real world

Arithmetic Coding

- Lossless and prefix-free
- Theoretically, encodes every symbol as an interval of the real line, with length equal to the probability of seeing that symbol

Other Lossless Encoding

Huffman Coding (1952)

- Lossless and prefix-free
- Encodes using binary tree structure
- Provably optimal per-symbol: average length at least as good as any other coding scheme
- Not always optimal in real world

Arithmetic Coding

- Lossless and prefix-free
- Theoretically, encodes every symbol as an interval of the real line, with length equal to the probability of seeing that symbol
- In practice, send one number in that interval

Other Lossless Encoding

Huffman Coding (1952)

- Lossless and prefix-free
- Encodes using binary tree structure
- Provably optimal per-symbol: average length at least as good as any other coding scheme
- Not always optimal in real world

Arithmetic Coding

- Lossless and prefix-free
- Theoretically, encodes every symbol as an interval of the real line, with length equal to the probability of seeing that symbol
- In practice, send one number in that interval
- Optimal under slightly different conditions

Lossy Encoding

Lossy Encoding

- Throws away "unimportant" information

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding
- Needs model of what counts as "important"

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding
- Needs model of what counts as "important"

- JPEG

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding
- Needs model of what counts as "important"

- JPEG
- GIF

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding
- Needs model of what counts as "important"

- JPEG
- GIF
- MP3

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding
- Needs model of what counts as "important"

- JPEG
- GIF
- MP3
- JPG

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding
- Needs model of what counts as "important"

- JPEG
- GIF
- MP3
- JPG
- AAC

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding
- Needs model of what counts as "important"

- JPEG
- GIF
- MP3
- JPG
- AAC
- MPEG-4

Lossy Encoding

- Throws away "unimportant" information
- Breaks theoretical bounds on lossless encoding
- Needs model of what counts as "important"

- JPEG
- GIF
- MP3
- JPG
- AAC
- MPEG-4
- The internet

Correcting Errors

- Source coding removes redundancy

Correcting Errors

- Source coding removes redundancy but sometimes redundancy is good.

Correcting Errors

- Source coding removes redundancy but sometimes redundancy is good.
- Noisy channels introduce errors

Correcting Errors

- Source coding removes redundancy but sometimes redundancy is good.
- Noisy channels introduce errors
- Error-correcting codes include check information.

Correcting Errors

- Source coding removes redundancy but sometimes redundancy is good.
- Noisy channels introduce errors
- Error-correcting codes include check information.

Parity Bits

Require every byte to sum to zero mod 2.

Correcting Errors

- Source coding removes redundancy but sometimes redundancy is good.
- Noisy channels introduce errors
- Error-correcting codes include check information.

Parity Bits

Require every byte to sum to zero mod 2.
Eighth bit is "parity bit".

Correcting Errors

- Source coding removes redundancy but sometimes redundancy is good.
- Noisy channels introduce errors
- Error-correcting codes include check information.

Parity Bits

Require every byte to sum to zero mod 2.
Eighth bit is "parity bit". Can detect single-bit error in each byte.

Hamming Code

Hamming Code

Richard Hamming 1915–1998

Hamming Code

Richard Hamming 1915–1998

Hamming $(7, 4)$ (1950)

Four bits of data get three check bits.

Hamming Code

Richard Hamming 1915–1998

Hamming $(7, 4)$ (1950)

Four bits of data get three check bits.
Can *fix* single-bit errors,

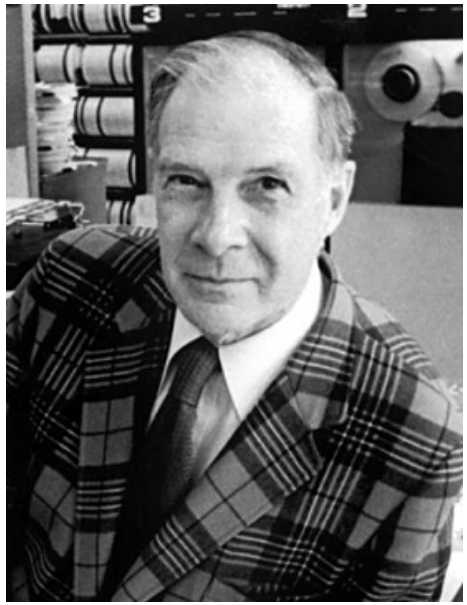
Hamming Code

Richard Hamming 1915–1998

Hamming $(7, 4)$ (1950)

Four bits of data get three check bits.
Can *fix* single-bit errors, and detect two-bit errors.

Hamming Code



Richard Hamming 1915–1998

Hamming $(7, 4)$ (1950)

Four bits of data get three check bits.
Can *fix* single-bit errors, and detect two-bit errors.
Invented to fix problems with a punched-card reader at Bell Labs.