EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Faculty of Science · Department of Computer Science · Group of Prof. Robert Bamler

Lecture 1, Part 1:

Course Overview: Data Compression With and Without Deep Probabilistic Models

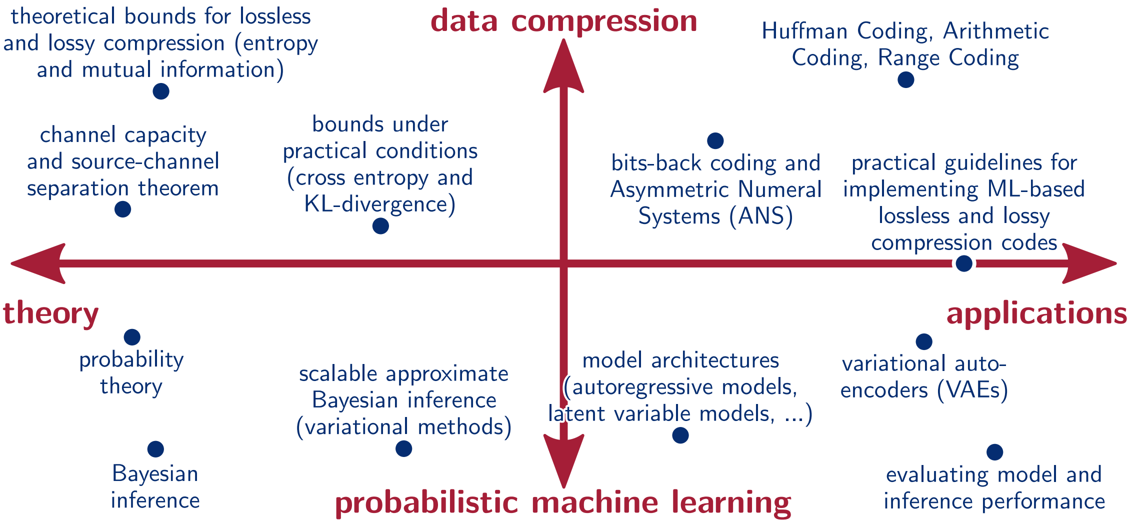
Robert Bamler · Summer Term of 2023

These slides are part of the course *"Data Compression With and Without Deep Probabilistic Models"* taught at University of Tübingen. More course materials—including video recordings, lecture notes, and problem sets with solutions—are publicly available at `https://robamler.github.io/teaching/compress23/`.

Recommended Literature

- **Written words:**
  - Yang, Mandt, Theis. "An Introduction to Neural Data Compression." *arXiv:2202.06533* (2022)
  - MacKay. "Information Theory, Inference, and Learning Algorithms." *Cambridge U. Press* (2003) → free PDF by the author: `http://www.inference.org.uk/mackay/itprnn/book.html`
  - Murphy. "Machine Learning: a Probabilistic Perspective." *MIT Press* (2012)
- **Moving pictures:**
  - **information theory course** by David MacKay: `https://youtube.com/playlist?list=PLruBu5BI5n4aFpG32iMbdWoRVAA-Vcso6`
  - **probabilistic machine learning course** by Philipp Hennig: `https://youtube.com/playlist?list=PL05umP7R6ij1tHaOFY96m5uX3J21a6yNd`
  - **information theory videos** by mathematicalmonk: `https://youtube.com/playlist?list=PLE125425EC837021F`

Spectrum of Topics Covered in This Course



data compression

theory

applications

probabilistic machine learning

theoretical bounds for lossless and lossy compression (entropy and mutual information)

channel capacity and source-channel separation theorem

bounds under practical conditions (cross entropy and KL-divergence)

Huffman Coding, Arithmetic Coding, Range Coding

bits-back coding and Asymmetric Numeral Systems (ANS)

practical guidelines for implementing ML-based lossless and lossy compression codes

probability theory

Bayesian inference

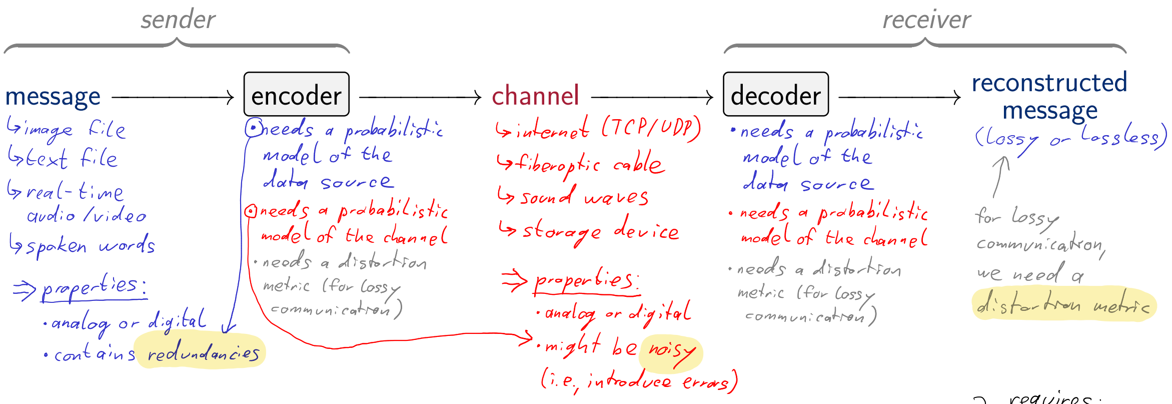
scalable approximate Bayesian inference (variational methods)

model architectures (autoregressive models, latent variable models, ...)

variational auto-encoders (VAEs)

evaluating model and inference performance

Problem Setting: Communication Over a Channel



sender

receiver

message → encoder → channel → decoder → reconstructed message

↳ image file
↳ text file
↳ real-time audio / video
↳ spoken words

⇒ properties:
- analog or digital
- contains redundancies

- needs a probabilistic model of the data source
- needs a probabilistic model of the channel
- needs a distortion metric (for lossy communication)

↳ internet (TCP/UDP)
↳ fiberoptic cable
↳ sound waves
↳ storage device

⇒ properties:
- analog or digital
- might be noisy (i.e, introduce errors)

- needs a probabilistic model of the data source
- needs a probabilistic model of the channel
- needs a distortion metric (for lossy communication)

(lossy or lossless)

for lossy communication, we need a distortion metric

Goal: transmit a message from *sender* to *receiver*

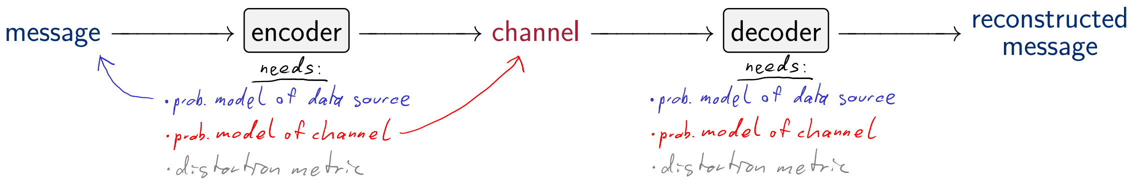
- **efficiently**, i.e., using the channel as little as possible
- **reliably**, i.e., without errors or with as little (relevant) distortion as possible

requires:
- prob. model of data source
- prob. model of channel
- distortion metric

Admin Stuff

- time and place
- resources (problem sets + solutions, lecture notes, videos, ...)
  - website: `https://robamler.github.io/teaching/compress23/`
  - Outdated Ilias link
  - new videos every Friday (covering lecture from preceding Wednesday)
- exam: 26 July during lecture hours (tentatively)

Preview: Source-Channel Separation Theorem

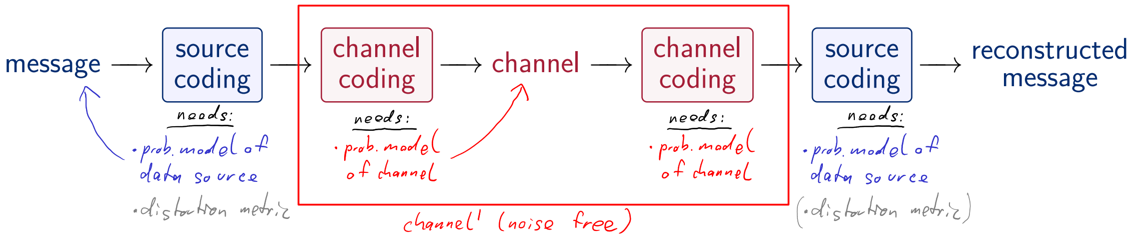


message → encoder → channel → decoder → reconstructed message

needs:
- prob. model of data source
- prob. model of channel
- distortion metric

needs:
- prob. model of data source
- prob. model of channel
- distortion metric

- *We'll prove in Lecture 10 that one can (in principle) always use a more modular setup:* (in principle)



message → source coding → channel coding → channel → channel coding → source coding → reconstructed message

needs:
- prob. model of data source
- distortion metric

needs:
- prob. model of channel

needs:
- prob. model of channel

needs:
- prob. model of data source

(· distortion metric)

channel' (noise free)

Test Your Understanding

- **Question 1:** Consider a data source that generates messages which are strings of N bits. Further, consider a channel that transmits one bit each time it is invoked, but it sometimes flips the transmitted bit due to noise. You want to communicate one message, and you want to be certain beyond reasonable doubt that the receiver can decode the message without errors. How many times do you have to invoke the channel?

  (a) N bits (b) more than N bits (c) fewer than N bits [✓ (d) it depends]

- **Question 2:** Consider a noise-free channel and a message with some redundancies (e.g., English text). What should an encoder and a decoder do with these redundancies?

  encoder: remove redundancies
  decoder: re-introduce redundancies } source coding (compression)

- **Question 3:** Same message as in Question 2, but now with a noisy channel. What *additional* task do encoder and decoder have to do now? Think again about redundancies.

  encoder: introduce redundancies that are taylored for the channel
  decoder: use these redundancies to detect & correct errors } channel coding

Outlook

- **Problem 0.1 on Problem Set 0:** simple examples of source coding vs. channel coding
- **Coming up:** "Lossless Compression I: Symbol Codes"
  - unique decodability & prefix codes
  - Huffman coding (used in zip, gzip, png, ...)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Faculty of Science · Department of Computer Science · Group of Prof. Robert Bamler

Lecture 1, Part 2:

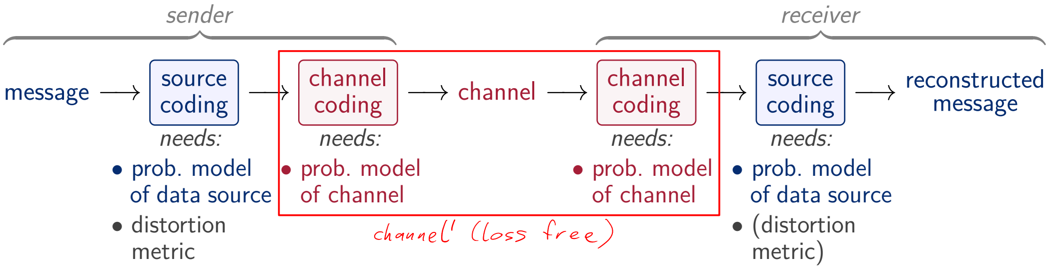
Lossless Compression I: Symbol Codes

Robert Bamler · Summer Term of 2023

These slides are part of the course *"Data Compression With and Without Deep Probabilistic Models"* taught at University of Tübingen. More course materials—including video recordings, lecture notes, and problem sets with solutions—are publicly available at `https://robamler.github.io/teaching/compress23/`.

Overview

- **Recap from last video:** Source Coding & Channel Coding



- **This course:** focus on $\boxed{\text{source coding}}$, i.e., data compression
  - We'll begin with *lossless* compression.
- **Two approaches** to lossless compression:
  (a) **symbol codes:** conceptionally simple but suboptimal
  (b) **stream codes:** more involved but close to optimal

Definitions and Notation

- **Example:**
  - a data source generates ASCII-encoded text of variable length;
  - we want to send the message $\mathbf{x}$ = "Rosebud" over a noise-free binary channel.
- **More generally:**
  - The message $\mathbf{x}$ is a *variable-length sequence of symbols from a discrete alphabet:*

$\boxed{\mathbf{x} = (x_1, x_2, \ldots, x_{k(\mathbf{x})}) \in \mathfrak{X}^*}$ where

$\mathfrak{X}$ is called the "alphabet" (discrete set, known to sender and receiver);

$k(\mathbf{x})$ is the length of a given message $\mathbf{x}$;

$x_i \in \mathfrak{X}$ is called a "symbol" $\forall i \in \{1, \ldots, k\}$;

$\mathfrak{X}^* := \bigcup_{k \geq 0} \mathfrak{X}^k$ is called the "Kleene closure" of $\mathfrak{X}$.

  - **The channel** transmits B-ary bits, i.e., elements of $\{0, \ldots, B-1\}$ for some $B \geq 2$.

Definition of Symbol Codes

i.e., $\mathfrak{X}$ is either finite or countably infinite

- **Have:** discrete alphabet $\mathfrak{X}$, message $\mathbf{x} = (x_1, x_2, \ldots, x_{k(\mathbf{x})}) \in \mathfrak{X}^*$, lossless B-ary channel.
- **Goal (lossless compression):** find a mapping $\mathfrak{X}^* \to \{0, \ldots, B-1\}^*$ such that
  - the mapping is *injective* (i.e., invertible); and → so that decoding is possible without ambiguities
  - the resulting bit strings are short. → more details later
- **B-ary symbol code:** map each x_i to a bit string $C(x_i)$, then simply concatenate them.

$C : \mathfrak{X} \to \{0, \ldots, B-1\}^*$ is called the "code book";

$C(x_i)$ is called the "code word" for symbol $x_i \in \mathfrak{X}$;

$|C(x_i)|$ denotes the length of the code word (i.e., the number of B-ary bits);

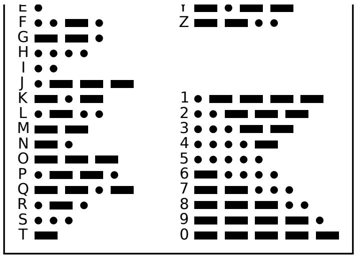
$C^*(\mathbf{x}) := C(x_1) \parallel C(x_2) \parallel \ldots \overset{\text{concatenation}}{\parallel} C(x_{k(\mathbf{x})})$ is the resulting encoding of the message $\mathbf{x}$;

$|C^*(\mathbf{x})| = \sum_{i=1}^{k(\mathbf{x})} |C(x_i)|$ denotes the length of the encoding (= "bit rate")

Examples of Symbol Codes

- **Morse code**

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



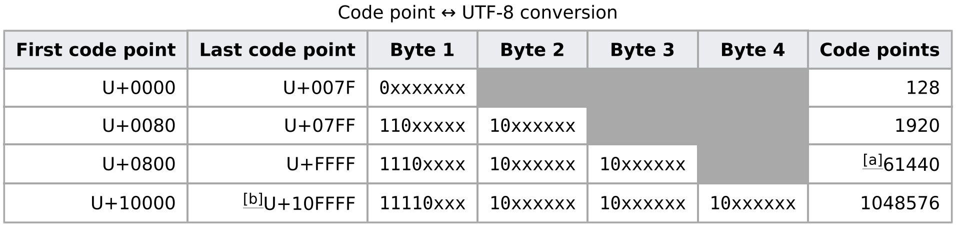
(figure source: Wikipedia; CC-0)

B = 3 or 4

(dot, dash, and either a single space character or a short space and a long space character, depending on how closely we want to model the convention)

Examples of Symbol Codes

- Morse code
- **UTF-8**



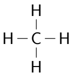
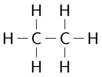
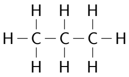
Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	Code points
U+0000	U+007F	0xxxxxxx				128
U+0080	U+07FF	110xxxxx	10xxxxxx			1920
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx		[a]61440
U+10000	[b]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	1048576

(figure source: Wikipedia)

B = 256 (each symbol is encoded into a sequence of bytes)

Examples of Symbol Codes

- Morse code
- UTF-8
- **Counterexample (not a symbol code): molecular formulae in chemistry**
  - methane:  $= CH_4$
  - ethane:  $= C_2H_6$
  - propane:  $= C_3H_8$
  - **Note:** The DEFLATE algorithm (used in zip, gzip, png, ...) works somewhat similarly.

Note: no one-to-one mapping between constituents of condensed formula (right) and constituents of structural formula (left), e.g. the first "C" in structural formula of ethane is encoded as "C" in the condensed formula, but the second "C" is encoded as subscript "2".

⇒ Condensed formulae in chemistry are <u>not</u> symbol codes.

Examples of Symbol Codes

- Morse code
- UTF-8
- Counterexample: molecular formulae in chemistry
- **Toy Example: Simplified Game of Monopoly (SGoM)**
  - message $\mathbf{x} \in \mathfrak{X}^*$ is sequence of symbols;
  - for each symbol: throw a pair of fair dice and record their sum;
  - for simplicity, let's use 3-sided dice $\Rightarrow \mathfrak{X} = \{2, 3, 4, 5, 6\}$.

x	$C^{(1)}(x)$	$C^{(2)}(x)$	$C^{(3)}(x)$	$C^{(4)}(x)$	$C^{(5)}(x)$
2	10	010	010	10	010
3	11	011	10	011	01
4	100	100	00	11	00
5	101	101	11	00	11
6	110	110	011	010	110

→ you'll analyze these code books on Problem Set 0.

Properties of Symbol Codes

A symbol code with code book $C : \mathfrak{X} \to \{0, \ldots, B-1\}^*$ is called

- **"uniquely decodable"** if the resulting code $C^* : \mathfrak{X}^* \to \{0, \ldots, B-1\}^*$ is injective
  - necessary property for lossless compression
  - difficult to prove in general since it requires reasoning over $\mathfrak{X}^*$ (unique decodability is a property of C^*)
- **"prefix free"** (aka, C is a "prefix code") if no code word is a prefix of another code word
  - more formally:
  - easier to prove than unique decodability since it requires only reasoning over $\mathfrak{X}$. (prefix freeness is a property of C)
  - easier to decode than non-prefix-free codes (using a *greedy* algorithm).

We will show ...

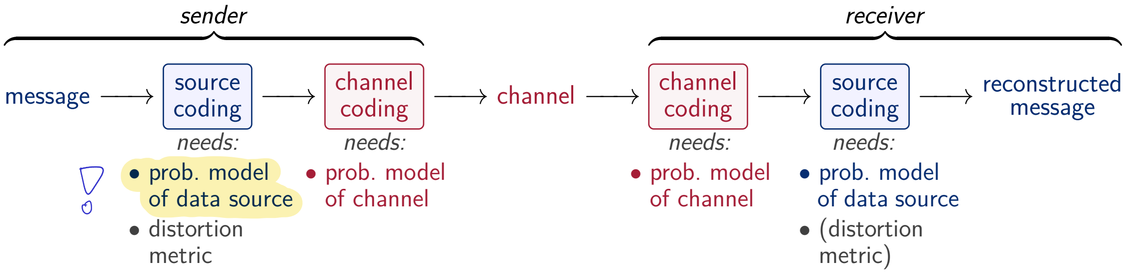
- ... on Problem Set 1 that "prefix free" $\Rightarrow$ "uniquely decodable" but not the reverse;
- ... in Lecture 3 that a slightly weaker generalization of the reverse does hold, however. (This will allow us to simplify future discussions.)

Back to Our SGoM

x	$C^{(1)}(x)$	$C^{(2)}(x)$	$C^{(3)}(x)$	$C^{(4)}(x)$	$C^{(5)}(x)$
2	10	010	010	10	010
3	11	011	10	011	01
4	100	100	00	11	00
5	101	101	11	00	11
6	110	110	011	010	110
uniquely decodable?	✗	✓	✓	✓	✓
prefix free?	✗	✓	✓	✓	✗
best choice?					

?

Reminder from Last Video



Probabilistic Model of the Data Source

x	possible throws	probability $p(x)$	$C^{(1)}(x)$	$C^{(2)}(x)$	$C^{(3)}(x)$	$C^{(4)}(x)$	$C^{(5)}(x)$
2	⚀⚀		10	010	010	00	010
3	⚀⚁, ⚁⚀		11	011	10	111	01
4	⚀⚂, ⚁⚁, ⚂⚀		100	100	00	01	00
5	⚁⚂, ⚂⚁		101	101	11	10	11
6	⚂⚂		110	110	011	110	110
		uniquely decodable?	✗	✓	✓	✓	✓
		prefix free?	✗	✓	✓	✓	✗
		expected code word length $L_C := \sum_{x\in\mathfrak{X}} p(x)\,\lvert C(x)\rvert$	$\frac{8}{3}\approx 2.67$	3	$\frac{22}{9}\approx 2.44$	$\frac{23}{9}\approx 2.56$	$\frac{22}{9}\approx 2.44$
		best choice?	✗	✗	✓	✗	✗

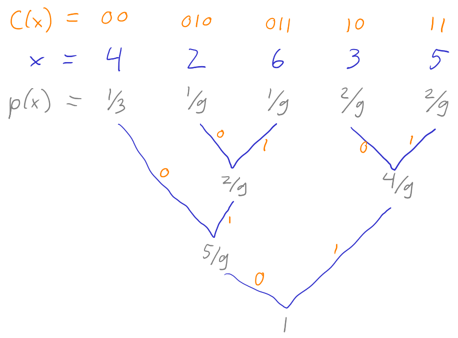
"Huffman Code for p"

Optimal Symbol Codes: Huffman Codes [Huffman, 1952]

- Our first example of an *entropy coder:*

  probabilistic model p of a data source $\longmapsto$ lossless compression code C

- **Example** for SGoM:



Note:

- symbols are sorted here such that the resulting tree has no crossing edges; this is purely for aesthetics and not technically necessary.
- there's a tie in the 2nd step which we could have chosen to break differently

  → see Problems 1.1 (c) and 1.2 on Problem Set 1

Huffman Coding for $B = 2$ (and finite $\mathfrak{X}$)

Informal algorithm: create a binary tree whose leaves are the symbols $x \in \mathfrak{X}$ as follows:

- start from the leaves; each leaf $x \in \mathfrak{X}$ is represented as a node with weight $p(x)$;
- while the graph is not fully connected:
  - identify two nodes with lowest weights w and w' among all nodes that don't yet have a parent;
  - combine these two nodes by introducing a parent node with weight $w + w'$;
  - label the edges from the new parent node to its two children with "0" and "1" in arbitrary order;
- interpret the resulting tree as a *trie* for a prefix code on $\mathfrak{X}$.

More formal algorithm: Problem Set 1 → also: implementation in python

Claim: Huffman codes are *optimal* uniquely decodable symbol codes (i.e., they minimize L_C).

- **Proof:** Lecture 3

Outlook

- **Problem Set 0:** simple warm-up exercises (mostly SGoM)
- **Problem Set 1:** Huffman coding
  - breaking ties
  - implementation in Python
- **Next 2 videos:** Source Coding Theorem
  - fundamental theoretical bound for lossless compression
  - We'll prove both that a certain bound holds and that it is meaningful in practice.
- **Later videos:**
  - use machine learning to model the data source
  - even better lossless codes than Huffman codes (stream codes)
  - lossy compression