# Stream Codes: Arithmetic Coding, Range Coding, and Asymmetric Numeral Systems (ANS)

Robert Bamler · 19 May 2022

This lecture is a part of the Course “Data Compression With and Without Deep Probabilistic Models” at University of Tübingen.

More course materials (lecture notes, problem sets, solutions, and videos) are available at:
`https://robamler.github.io/teaching/compress22/`



## Stream Codes vs. Symbol Codes

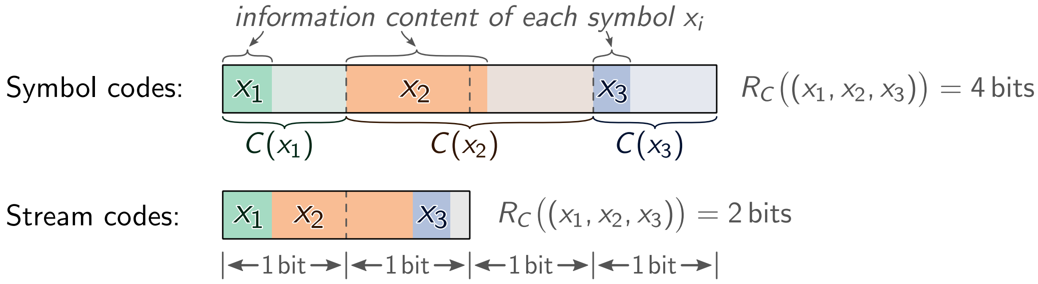
- **Reminder:** Huffman coding [Huffman, 1952] creates an optimal *symbol code*; but:
  - Symbol codes are restrictive: each symbol contributes an *integer* number of bits.
  - Modern machine-learning based (lossy) compression methods typically use models with very low entropy *per symbol* (e.g., $H_P[X_i] \approx 0.3$ bits).

    $\Rightarrow$ Any symbol code has $>$ 200 % overhead (since it needs at least 1 bit per symbol).
- **Naive idea:** Block codes (Problem 2.4)
  - apply Huffman coding to large blocks of symbols rather than to individual symbols
  - problem: cost scales exponentially in the block size
- **Better idea:** stream codes — amortize efficiently over multiple symbols
  - Arithmetic Coding and Range Coding [Rissanen and Langdon, 1979; Pasco, 1976]
  - Asymmetric Numeral Systems (ANS) [Duda et al., 2015]





## Amortizing Compressed Bits Over Symbols




- Intuitively: “pack” information content as closely as possible
- We can no longer associate each bit in the compressed representation with any specific symbol

# Arithmetic Coding and Range Coding

[Pasco, 1976; Rissanen and Langdon, 1979]

Idea: Similar to Shannon coding, but applied to the entire message of k symbols rather than to each symbol individually

→ challenge: making it computationally efficient

Arithmetic Coding and Range Coding are two very similar algorithms. They are both
- conceptionally simple
- but a bit tricky to fully implement due to a number of edge cases

Consider a probability distribution P($\underline{X}$) over messages $\underline{X} = (X_1, X_2, \ldots, X_k)$ with $X_1, X_2, \ldots, X_k \in \mathfrak{X}$

Define some total ordering on the message space, i.e., for $\underline{X}, \underline{X}' \in \mathfrak{X}^k$, you have exactly one of

$\underline{X} = \underline{X}'$ or $\underline{X} < \underline{X}'$ or $\underline{X} > \underline{X}'$

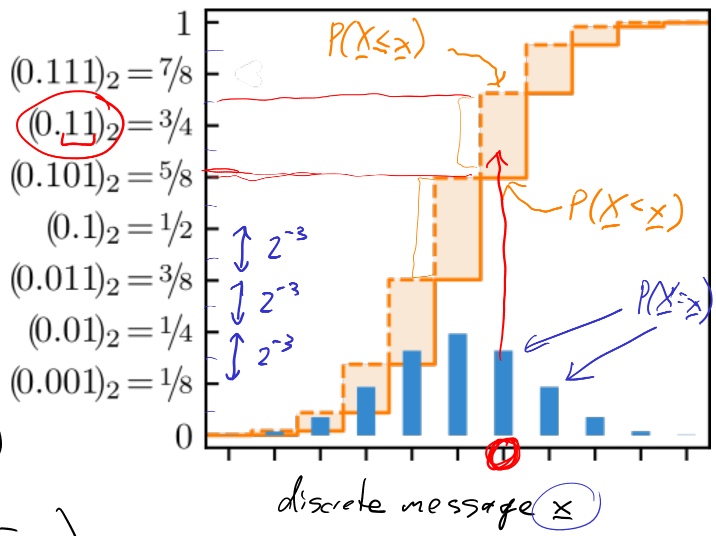
Now consider the left- and right-sided cumulative distribution functions:

- $P(\underline{X} < \underline{x}) = \sum_{\underline{x}' < \underline{x}} P(\underline{X} = \underline{x}')$
- $P(\underline{X} \leq \underline{x}) = \sum_{\underline{x}' \leq \underline{x}} P(\underline{X} = \underline{x}')$

→ partition the interval $[0, 1)$ into pairwise disjoint subintervals

$$I_{\underline{x}} := [P(\underline{X} < \underline{x}), P(\underline{X} \leq \underline{x})) \subset [0, 1)$$

of size $P(\underline{X} \leq \underline{x}) - P(\underline{X} < \underline{x}) = P(\underline{X} = \underline{x})$




Idea: identify each $I_{\underline{x}}$ by some (arbitrary) number $\xi_{\underline{x}} \in I_{\underline{x}}$ with a finite-length binary representation

$$\xi_{\underline{x}} = (0.\underbrace{b_1 b_2 \ldots b_{R(\underline{x})}}_{\text{compressed representation } C(\underline{x})})_2, \quad b_1, b_2 \in \{0,1\}$$

Question: what is the rate R(x), i.e., how long does the binary representation of $\xi_{\underline{x}}$ have to be if we want to have $\xi_{\underline{x}} \in I_{\underline{x}}$?

→ Consider the set of all numbers $\xi = (0.\underbrace{??? \ldots ?}_{r \text{ bits}})_2$

↳ equidistant grid with spacing of $2^{-r}$

⇒ if $\underbrace{\text{the size of } I_{\underline{x}}}$ $\geq 2^{-r}$ then $I_{\underline{x}}$ must contain a grid point

$$P(\underline{X}=\underline{x}) \geq 2^{-r} \Leftrightarrow r \geq -\log_2 P(\underline{X}=\underline{x})$$

⇒ set $R(\underline{x})$ to smallest $r$ that satisfies

$$\boxed{R(\underline{x}) = \lceil -\log_2 P(\underline{X}=\underline{x}) \rceil}$$

e.g. if $r=3$

$(0.000)_2$
$(0.001)_2$
$(0.010)_2$
$(0.011)_2$
$\vdots$
$(0.111)_2$

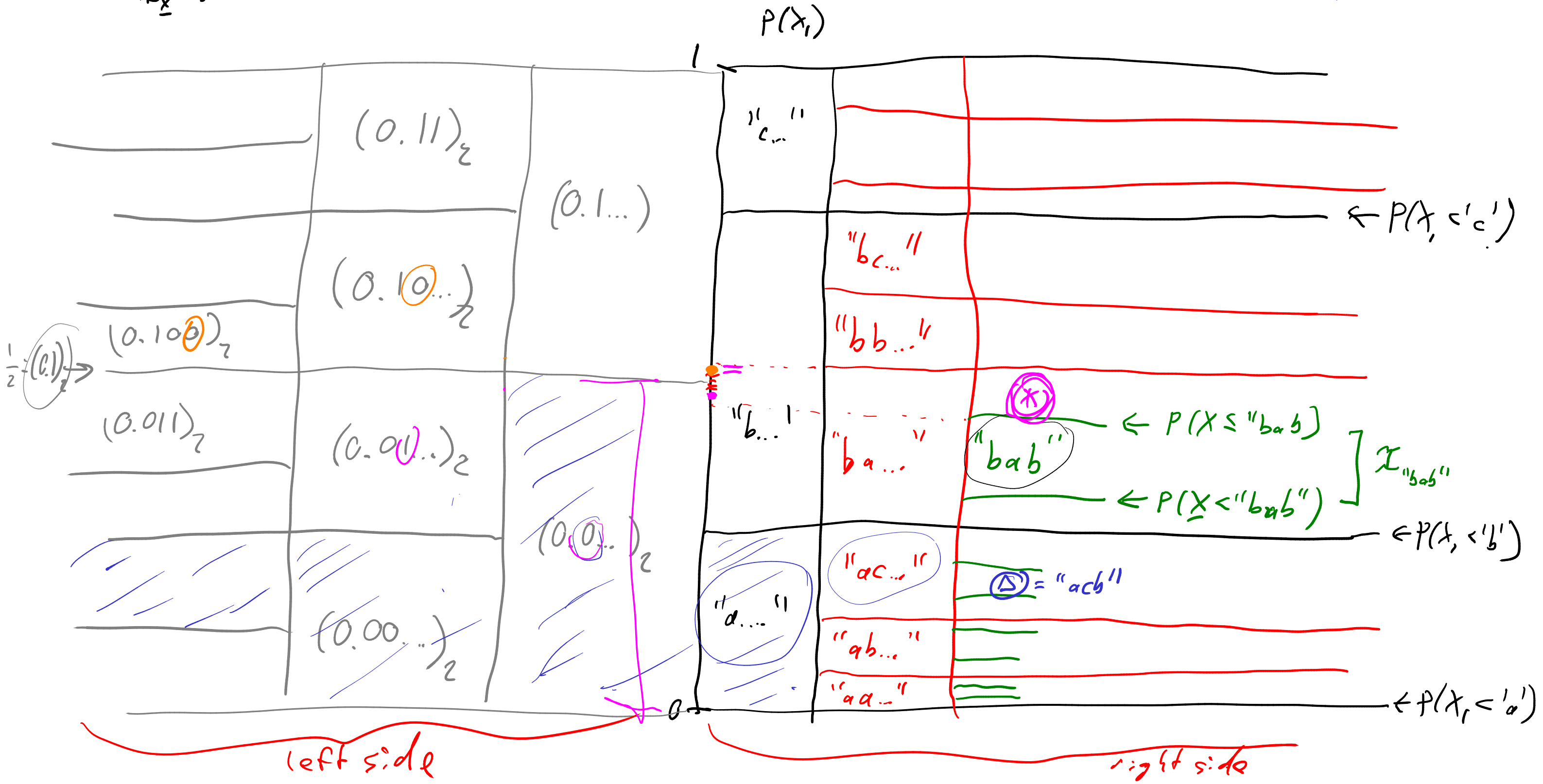
So far, we've more or less reinvented Shannon coding, except that
- we apply it to the whole message rather than a single symbol; and
- we don't care about unique decodability here since we don't expect users to concatenate the compressed representations of entire messages without some form of container format or protocol

But: how can we find a suitable $\xi_{\underline{x}} \in I_{\underline{x}}$ without iterating over all possible messages?

$$I_{\underline{x}} = [P(\underline{X} < \underline{x}), P(\underline{X} \leq \underline{x}))$$

$O(|\mathfrak{X}|^k)$ runtime ☹

Strategy ("Arithmetic Coding"):
- use chain rule of probability theory → $P(\underline{X}) = P(X_1)\, P(X_2|X_1)\, P(X_3|X_1, X_2) \cdots$ (e.g. autoregressive model)
- use lexicographic order of messages → e.g., for $k=3$, $\mathfrak{X} = \{\text{'a'}, \text{'b'}, \text{'c'}\}$: "aaa", "aab", "aac", "aba", "abb", ..., "ccc"
- find $I_{\underline{x}}$ by iterative refinement



Idea: · iteratively refine intervals on right side

· as soon as interval on the right lies entirely within some interval on left side: → emit the corresponding bits that now can't change anymore

Problem: Cases like ⊛ where lots of trailing bits are not yet resolved

→ can only happen if all but the first unresolved bit are equal

→ just keep a counter of the number of unresolved bits and emit them all at once as they get resolved

Remarks:

- in practice, Arithmetic coding becomes more complicated because the intervals quickly become too small for typical numerical precisions. Thus, every time one emits a bit, one should rescale all intervals on both the left side and the right side by a factor of 2. This also works in situations like ⊛, but it is a bit tedious to work out the details.
- Range coding is similar, but it works with larger bases than 2 (e.g., 2^32 or 2^64) to improve practical computational efficiency on real hardware (→ emits compressed data in blocks of, e.g., 32 or 64 bits).
- on next week's problem set, you will use a range coder provided by a library ("constriction") to improve our machine-learning based compression method for natural language from Problem Set 3.