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

Robert Bamler · 19 May 2022

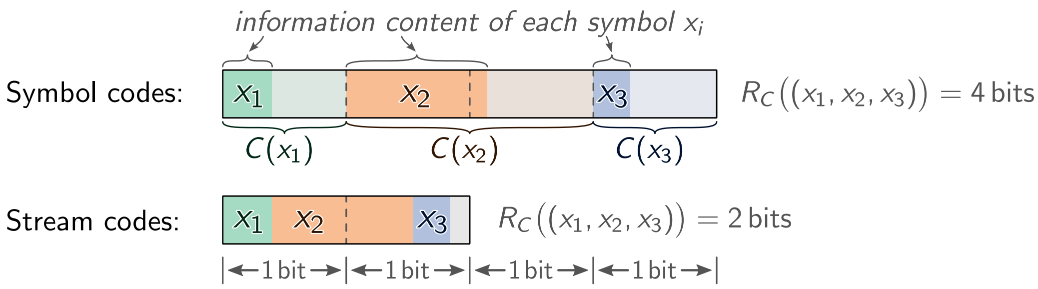
This lecture is a part of the Course “Data Compression With and Without Deep Probabilistic Models” at University of Tübingen.

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

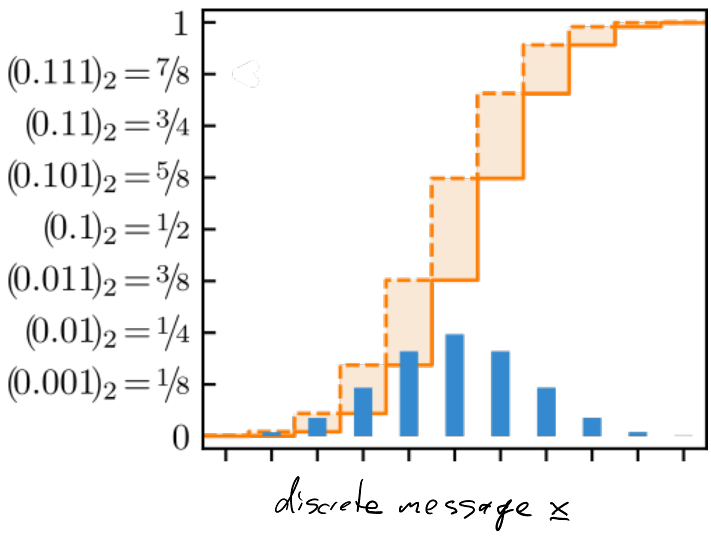
→ challenge: making it computationally efficient

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(X) over messages $\underline{X} = (X_1 \in \mathcal{X}, X_2, \ldots, X_k \in \mathcal{X})$

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

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



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

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

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 $\xi_{\underline{x}} \in \mathcal{I}_{\underline{x}}$ without iterating over all possible messages?

Strategy ("Arithmetic Coding"):
- use chain rule of probability theory

- use lexicographic order of messages

- find $\mathcal{I}_{\underline{x}}$ by iterative refinement

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.

Asymmetric Numeral Systems

[Duda et al., 2015]

Exercise

Consider a data source that generates a random message $\mathbf{X} \equiv (X_1, X_2, \ldots, X_k)$ of length k, where each symbol X_i, $i \in \{1, \ldots, k\}$ is drawn independently from all other symbols from a uniform probability distribution over the alphabet $\mathfrak{X} = \{0, 1, 2, \ldots, 9\}$.

(a) What is the entropy per symbol? $\frac{1}{k} H_P[\mathbf{X}] = H_P[X_i] =$

(b) What is the expected code word length of an *optimal symbol code* for this data source? $L := \mathbb{E}_P[\ell_{\text{Huff}}(X_i)] =$

(c) Can you do better than an optimal symbol code? Describe your approach first in words, then implement it in Python or in pseudo code. (about 4 lines of code for encoding and 4 lines of code for decoding; no library function calls necessary.)

(d) What is the expected bit rate per symbol of your method from part (c) in the limit of long messages? $\lim_{k \to \infty} \frac{1}{k} \mathbb{E}_P[R_C(\mathbf{X})] =$

Idea: generalize positional numeral systems from sequences of uniformly distributed symbols from the same alphabet to arbitrarily distributed symbols from symbol-dependent alphabets.