Lecture 5, Part 1:

Stream Codes: Encoding Into Fractional Bits

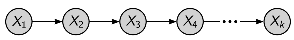
Robert Bamler · Summer Term of 2023

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/`.

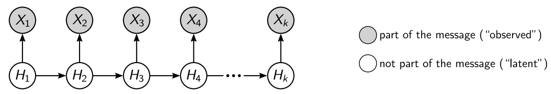
Recall: 4 Kinds of Scalable Probabilistic Models

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

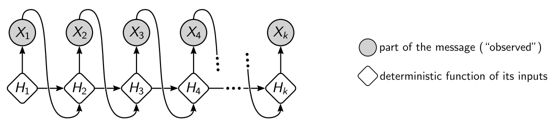
(1) Markov Process



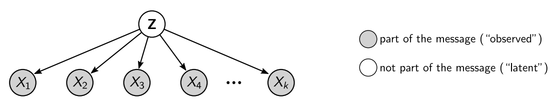
(2) Hidden Markov Model



(3) Autoregressive Model



(4) Latent Variable Model



Recall: Autoregressive Model + Huffman Coding

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- **Problem 3.2:** (link to solutions in video description)

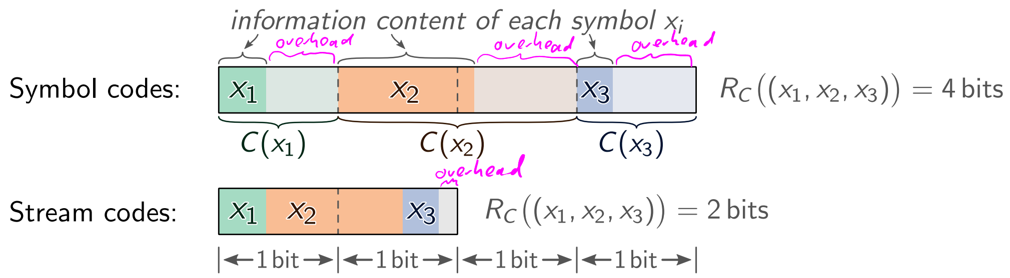
	msg. len (chars)	bits per character					
		Huffman	Shannon	inf. cont.	gzip	bzip2	bzip2'
validation set	106,864	**2.38**	2.72	2.12	3.43	2.82	2.40
test set	219,561	**2.38**	2.73	2.12	3.33	2.65	**2.38**
wikipedia-en	24,618	4.99	5.67	5.14	3.22	**2.92**	5.14
wikipedia-de	8,426	6.77	7.70	7.19	3.96	**3.76**	7.22

  - **Observation:** Huffman coding has overhead over information content of up to 1 bit *per symbol.*
  - Can be substantial in modern ML-based compression methods:

    $\left.\begin{array}{l}\text{e.g., information content} \approx 0.3 \text{ bits per symbol;}\\ \text{but Huffman coding needs} \geq 1 \text{ bit per symbol.}\end{array}\right\} \Longrightarrow$ about 200 % overhead. (→ useless)

- **Solution:** *amortize* compressed bits over symbols $\longrightarrow$ *"stream code"*

Stream Codes: Amortizing Bits Over Symbols



- **Intuitively:** stream codes "pack" information content as closely as possible.
  - We can no longer associate each bit in the compressed representation with any specific symbol.
- **2 important stream codes** with 2 different application domains:
  - **This lecture:** Arithmetic Coding & Range Coding [Pasco, 1976; Rissanen and Langdon, 1979]
  - **Next lecture:** Asymmetric Numeral Systems (ANS) [Duda et al., 2015]

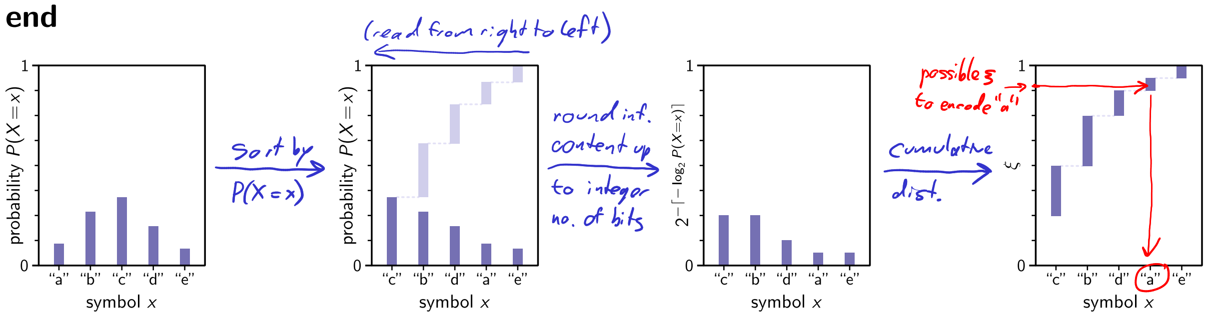
Arithmetic Coding: Overview [Pasco, 1976]

- **Idea:** similar to Shannon coding, but on entire *message space* $\mathfrak{X}^*$ instead of alphabet $\mathfrak{X}$.
  - **Thus:** overhead now only *per message*.
  - **Challenge:** computational efficiency

    assume message of length k → we'd have to iterate over $O(|\mathfrak{X}|^k)$ messages (astronomically expensive)
- **2 variants:** Arithmetic Coding & Range Coding
  - very similar to each other (Range Coding is faster on real hardware);
  - both conceptionally simple;
  - but a bit tricky in practice due to edge cases.

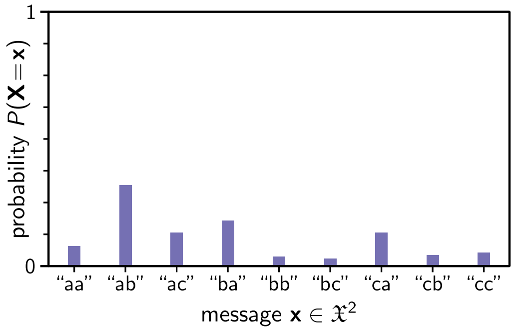
Reminder: Shannon Coding (for $B = 2$)

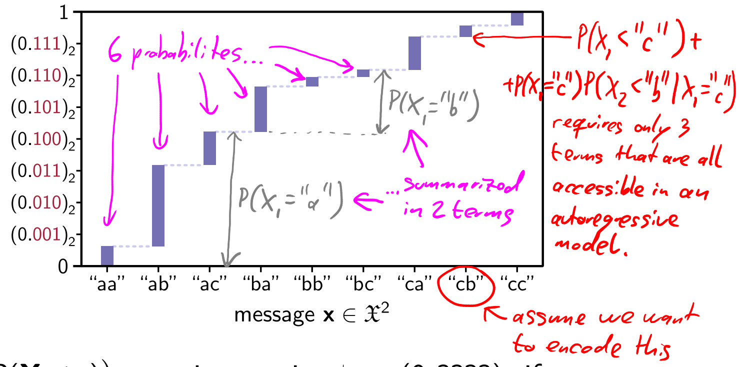
Input: alphabet $\mathfrak{X}$, probability measure P on $\mathfrak{X}$
output: prefix code $C_S : \mathfrak{X} \to \{0,1\}^*$.
Initialize $\xi \leftarrow 1$
for $x \in \mathfrak{X}$ *in order of increasing* $P(X{=}x)$ **do**
  Update $\xi \leftarrow \xi - 2^{-\lceil -\log_2 P(X=x) \rceil}$
  Write out $\xi \in [0,1)$ in binary: $\xi = (0.????\ldots)_2$
end



Arithmetic Coding: General Idea

- Consider probability measure P on *entire message space* $\mathfrak{X}^k$ (with fixed length k, for now).





- **Observation:** $\forall \mathbf{x} \in \mathfrak{X}^k$: interval $\big[P(\mathbf{X}<\mathbf{x}), P(\mathbf{X}\leq\mathbf{x})\big)$ contains a point $\xi_\mathbf{x} = (0.\underbrace{????}_{})_2$ if:
  size of interval $\leq$ spacing between numbers of form $(0.\underbrace{????}_{\mathcal{R}(\mathbf{x})\text{ bits}})_2$ $\quad \mathcal{R}(\mathbf{x})$ bits

$P(\underline{X}\leq\underline{x}) - P(\underline{X}<\underline{x}) = P(\underline{X}=\underline{x}) \leq (\underbrace{0.0001}_{\mathcal{R}(\underline{x})\text{ bits}})_2 = 2^{-\mathcal{R}(\underline{x})}$

$P(\underline{X}=\underline{x}) \leq 2^{-\mathcal{R}(\underline{x})}$
$\Leftrightarrow \mathcal{R}(\underline{x}) \geq -\log_2 P(\underline{X}=\underline{x})$
sufficient condition

Arithmetic Coding: Super Naive Algorithm

- **Encoder:**
  Initialize $c \leftarrow 0$ and $p \leftarrow 1$.
  for i **from** 1 **to** k **do**
  - Update $c \leftarrow c + p\,P(X_i < x_i \mid \mathbf{X}_{1:i-1}=\mathbf{x}_{1:i-1})$.
  - Update $p \leftarrow p\,P(X_i = x_i \mid \mathbf{X}_{1:i-1}=\mathbf{x}_{1:i-1})$.
  - ▷ ***Claim:*** *at this point, we have* $c = P(\mathbf{X}_{1:i}<\mathbf{x}_{1:i})$ *and* $p = P(\mathbf{X}_{1:i}=\mathbf{x}_{1:i})$
    $\Longrightarrow$ *invariant:* $[c, c+p) = \big[P(\mathbf{X}_{1:i}<\mathbf{x}_{1:i}), P(\mathbf{X}_{1:i}\leq\mathbf{x}_{1:i})\big) \subseteq [0,1)$

  end
  Encode some $\xi \in [c, c+p)$ in binary: $\xi = (0.\underbrace{?????}_{\lceil -\log_2 p\rceil \text{ bits}})_2$

extra step in decoder:
identify symbol $x_i \in \mathfrak{X}$ for which
$\xi \in \big[c + p\,P(X_i<x_i \mid \underline{X}_{1:i-1}=\underline{x}_{1:i-1}),\ c + p\,P(X_i\leq x_i \mid \underline{X}_{1:i-1}=\underline{x}_{1:i-1})\big)$

- **Decoder:** Analogous to encoder, but introduce an extra step

Caveats

- **Unique Decodability:**
  - not such a big deal as it was with symbol codes
    (it's unusual to concatenate entire compressed *messages* without deliminators);
  - can be solved with 1 extra bit: guarantee that $[\xi, \xi + 2^{-\mathcal{R}(\mathbf{x})}) \subseteq [c, c+p]$ (rather than just $\xi \in [c, c+p)$)
- **Variable message length:**
  - end of bit string $\not\Leftrightarrow$ end of message (since symbols can have information content < 1 bit)
  - for variable-length messages, the message length is fundamentally a *part of the message*
  - simple solution: introduce EOF symbol ($\rightarrow$ Problem Set 7)
- **Numerical precision:**
  - e.g, if bit rate $=$ 1 Mbit then c and p on last slide are 1-million-bit numbers
  - Run-time complexity for encoding k symbols: $\Theta(k^2)$

Arithmetic Coding: Naive Algorithm

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- **Encoder:**

  Initialize $c \leftarrow 0$ and $p \leftarrow 1$ (fixed point numbers $\in [0,1]$ with `precision` bits);

  for i **from** 1 **to** k **do**

  Update $c \leftarrow c + p\,P(X_i < x_i \,|\, \mathbf{X}_{1:i-1} = \mathbf{x}_{1:i-1})$; *(rounding to fixed point precision)*

  Update $p \leftarrow p\,P(X_i = x_i \,|\, \mathbf{X}_{1:i-1} = \mathbf{x}_{1:i-1})$; *(rounding to fixed point precision)*

  "rescaling"

  while $p < \frac{1}{2}$ **do**

  Emit first bit of $c = (0.????)_2$;

  Update $p \leftarrow 2p$;

  Update $c \leftarrow$ fractional part of $2c$;

  end

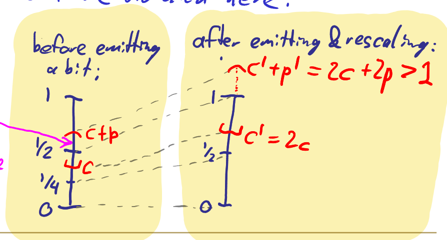
  end

  Emit first bit of $c = (0.????)_2$;

- **Decoder:** exercise

Problem: invariant $[c, c+p) \subseteq [0,1)$ can be violated here:

we might finally end up somewhere here → should instead have emitted a "1" bit



called "inverted" situation on next slide

Arithmetic Coding: Actual Algorithm

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Initialize $c \leftarrow 0$ and $p \leftarrow 1$ (fixed point numbers $\in [0,1]$ with `precision` bits);

Initialize `inverted` $\leftarrow 0$ (nonnegative integer);

for i **from** 1 **to** k **do**

Update $c \leftarrow c + p\,P(X_i < x_i \,|\, \mathbf{X}_{1:i-1} = \mathbf{x}_{1:i-1})$;

Update $p \leftarrow p\,P(X_i = x_i \,|\, \mathbf{X}_{1:i-1} = \mathbf{x}_{1:i-1})$;

"rescaling"

while $p < \frac{1}{2}$ **do**

~~Emit first bit of $c = (0.????)_2$;~~

Update $p \leftarrow 2p$;

Set $c' \leftarrow$ fractional part of $2c$;

Update $c \leftarrow c'$;

end

end

if inverted $\neq 0$: flush (exercise)

Emit first bit of $c = (0.????)_2$;

if inverted $\neq 0$ and $[c, c+p) \subseteq [0,1)$:
▷ transitioning from inverted to normal situation:
emit a single "0" bit followed by (inverted − 1) "1" bits
update inverted $\leftarrow 0$

if inverted $\neq 0$ and $[c, c+p) \subseteq [1,2)$
▷ transitioning from inverted to normal situation:
update $c \leftarrow c - 1$ $(\in [0,1))$
emit a single "1" bit followed by (inverted − 1) "0" bits
update inverted $\leftarrow 0$

if inverted $\neq 0$:
▷ rescaling from inverted to inverted situation:
update inverted $\leftarrow$ inverted + 1

if inverted $= 0$ and $[c', c'+p) \subseteq [0,1)$:
▷ rescaling from normal to normal situation:
emit first bit of $c = (0.????)_2$

if inverted $= 0$ and $[c', c'+p) \not\subseteq [0,1)$:
▷ rescaling from normal to inverted situation:
update inverted $\leftarrow 1$

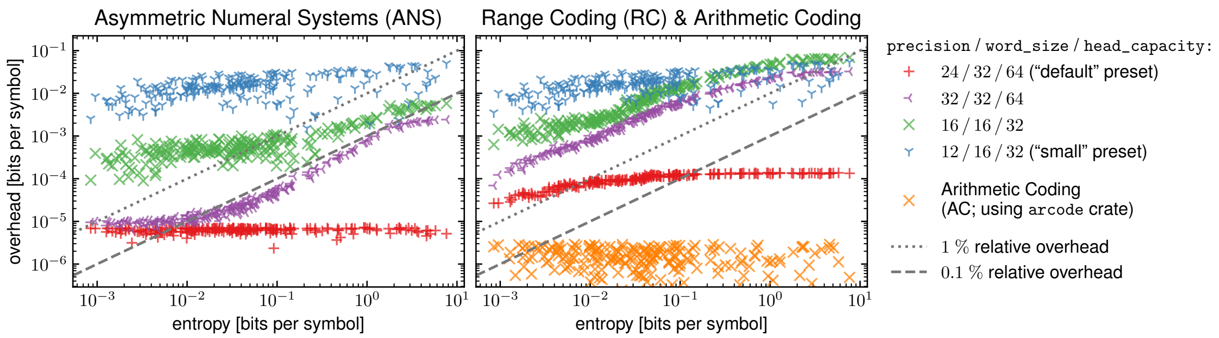
Real Hardware: Range Coding

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- CPUs are not optimized to operate on single bits
- *mechanical sympathy:* to best exploit the capabilities of a tool (e.g., a computer), one has to understand how the tool works.
- **Range Coding:** like arithmetic coding, but operating on `precision` bits at a time
  - accumulators c and p become numbers with $2\times$`precision` bits
  - individual symbol probabilities $P(X_i = x_i \,|\, \mathbf{X}_{1:i-1} = \mathbf{x}_{1:i-1}) \in (0,1)$ are `precision`-bit numbers
    $\Longrightarrow P(X_i = x_i \,|\, \mathbf{X}_{1:i-1} = \mathbf{x}_{1:i-1}) \geq 2^{-\texttt{precision}}$ (smallest representable nonzero number)
  - Emit `precision` bits at once when $p < 2^{-\texttt{precision}}$
    $\Longrightarrow$ always restores $p \geq 2^{-\texttt{precision}}$, thus at most 1 emission per symbol is necessary.
  - `inverted` keeps track of how many "$\underbrace{00000000}_{\texttt{precision}}$" or "$\underbrace{11111111}_{\texttt{precision}}$" blocks have accumulated.

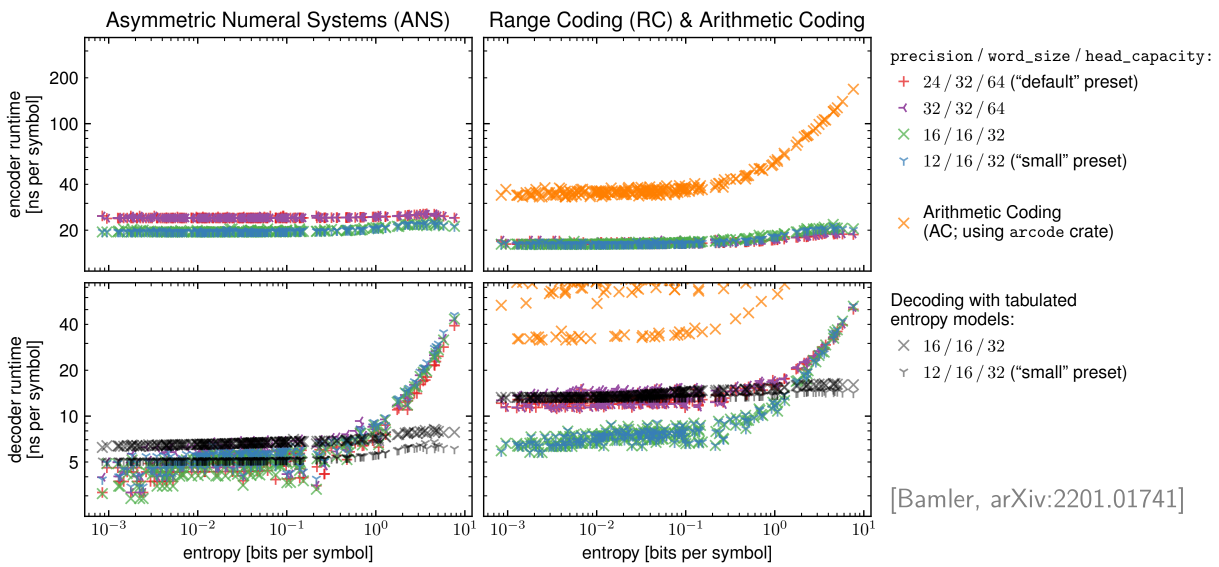
(instead of how many "0" or "1" bits have accumulated, as in our implementation of arithmetic coding on the last slide)

Empirical Compression Performances: bit rates



[Bamler, arXiv:2201.01741 (2022)]

Empirical Compression Performances: run times



[Bamler, arXiv:2201.01741]

Outlook

- **Next week (Lecture 6):** Asymmetric Numeral Systems
  - modern stream code that operates as a *stack* ("last-in-first-out")
  - conceptionally more difficult but easier to implement in real code → (which we will actually do for ANS)
  - uses "bits-back trick"
  - enables "bits-back trick" for latent variable models (→ Lecture 7)
- **Problem Set 7:** *use* range coding (from a library) for our autoregressive model of natural language from Problem Set 3.

  → removes overhead of symbol codes and achieves bit rates very close to the information content (less than 0.1 % overhead)