Information Theory

Lecture 3

- Lossless source coding algorithms:
  - Huffman: CT5.6–8
  - Shannon-Fano-Elias: CT5.9
  - Arithmetic: CT13.3
  - Lempel-Ziv: CT13.4–5

Zero-Error Source Coding

- Huffman codes: algorithm & optimality
- Shannon-Fano-Elias codes
  - connection to Shannon(-Fano) codes, Fano codes, and *per symbol* arithmetic coding
  - within $2(1)$ symbol of the entropy
- Arithmetic codes
  - adaptable, probabilistic model
  - within 2 bits of the entropy *per sequence!*
- Lempel-Ziv codes
  - "basic" and "modified" LZ-algorithm
  - sketch of asymptotic optimality

Example: Encoding a Markov Source

- 2-state Markov chain $P_{01} = P_{10} = \frac{1}{3} \implies \mu_0 = \mu_1 = \frac{1}{2}$
- Sample sequence

$$s = 1000011010001111 = 1\,0^4\,1^2\,0\,1\,0^3\,1^4$$

- Probabilities of 2-bit symbols

	$p(00)$	$p(01)$	$p(10)$	$p(11)$	H	$L \geq$
sample	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{4}$	≈ 1.9056	16
model	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	≈ 1.9183	16

- Entropy rate
  $H(\mathcal{S}) = h(\frac{1}{3}) \approx 0.9183 \implies L \geq \lceil 14.6928 \rceil = 15$

Huffman Coding Algorithm

- Greedy bottom-up procedure
- Builds a complete D-ary codetree by combining the D symbols of lowest probabilities

$\Rightarrow$ need $|\mathcal{X}| = 1 \mod, D - 1$

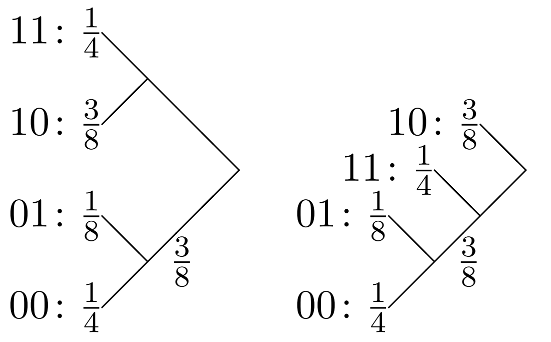
$\Rightarrow$ add dummy symbols of 0 probability if necessary

- Gives prefix code
- Probabilities of source symbols need to be available

$\Rightarrow$ coding long strings ("super symbols") becomes complex

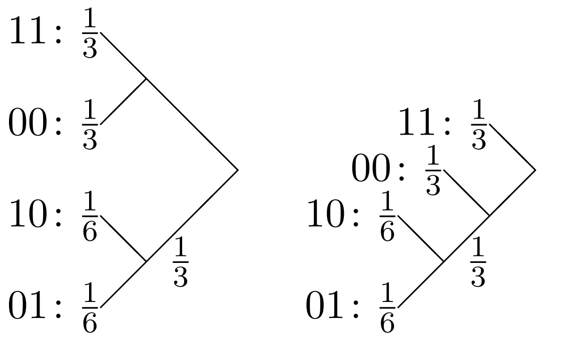
Huffman Code Examples

sample-based



$16,\ |1000001110000101| = 16$

model-based



$16,\ |001010000010010111| = 18$

Optimal Symbol Codes

- An optimal binary prefix code must satisfy

$$p(x) \leq p(y) \implies l(x) \geq l(y)$$

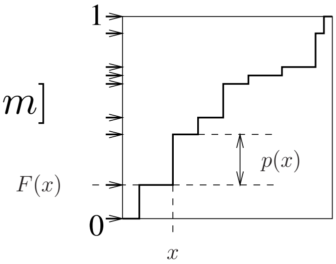
  - there are at least two codewords of maximal length
  - the longest codewords can be relabeled such that the two least probable symbols differ only in their last bit
- *Huffman codes are optimal prefix codes* (why?)
  - We know that

$$L = H(X) \iff l(x) = -\log p(x)$$

$\implies$ Huffman will give $L = H(X)$ when $-\log p(x)$ are integers (a *dyadic* distribution)

Cumulative Distributions and Rounding

- $X \in \mathcal{X} = \{1, 2, \ldots, m\}$; $p(x) = \Pr(X = x) > 0$
- Cumulative distribution function (cdf)



$$F(x) = \sum_{x' \leq x} p(x'), \quad x \in [0, m]$$

- Modified cdf

$$\bar{F}(x) = \sum_{x' < x} p(x') + \frac{1}{2} p(x), \quad x \in \mathcal{X}$$

  - only for $x \in \mathcal{X}$
  - $\bar{F}(x)$ known $\implies$ x known!

- We know that $l(x) \approx -\log p(x)$ gives a good code
- Use the binary expansion of $\bar{F}(x)$ as code for x; rounding needed
  - round to $\approx -\log p(x)$ bits
- Rounding: $[0, 1) \to \{0, 1\}^k$
  - Use base 2 fractions

$$f \in [0, 1) \implies f = \sum_{i=1}^{\infty} f_i 2^{-i}$$

  - Take the first k bits

$$\lfloor f \rfloor_k = f_1 f_2 \cdots f_k \in \{0, 1\}^k$$

  - For example, $\frac{2}{3} = 0.10101010 \cdots = 0.\overline{10} \implies \left\lfloor \frac{2}{3} \right\rfloor_5 = 10101$

Shannon-Fano-Elias Codes

- Shannon-Fano-Elias code (as it is described in CT)
  - $l(x) = \lceil \log \frac{1}{p(x)} \rceil + 1 \implies L < H(X) + 2$ [bits]
  - $c(x) = \lfloor \bar{F}(x) \rfloor_{l(x)} = \lfloor F(x) + \frac{1}{2} p(x) \rfloor_{l(x)}$
- *Prefix-free* if intervals $[0.c(x), 0.c(x) + 2^{-l(x)}]$ disjoint (why?)
  $\implies$ instantaneous code (check)
- Example:

	sample-based				model-based			
X	$p(x)$	$l(x)$	$\bar{F}(x)$	$c(x)$	$p(x)$	$l(x)$	$\bar{F}(x)$	$c(x)$
1(00)	1/4	3	1/8	001	1/3	3	1/6	001
2(01)	1/8	4	5/16	0101	1/6	4	5/12	0110
3(10)	3/8	3	9/16	100	1/6	4	7/12	1001
4(11)	1/4	3	7/8	111	1/3	3	5/6	110
	$L = 3.125 < H(X) + 2$				$L = 3.333 < H(X) + 2$			

- Shannon (or Shannon–Fano) code (see HW Prob. 1)
  - order the probabilities
  - $l(x) = \lceil \log \frac{1}{p(x)} \rceil \implies L < H(X) + 1$
  - $c(x) = \lfloor F(x) \rfloor_{l(x)}$
- Fano code (see CT p. 123)
  - $L < H(X) + 2$
  - order the probabilities
  - recursively split into subsets as nearly equiprobable as possible

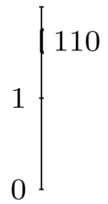
Intervals

- Dyadic intervals
  - A binary string can represent a subinterval of $[0,1)$

$$x_1 x_2 \cdots x_m \in \{0,1\}^m \implies x = \sum_{i=1}^{m} x_i 2^{m-i} \in \{0,1,\ldots,2^m-1\}$$

(the usual binary representation of x), then

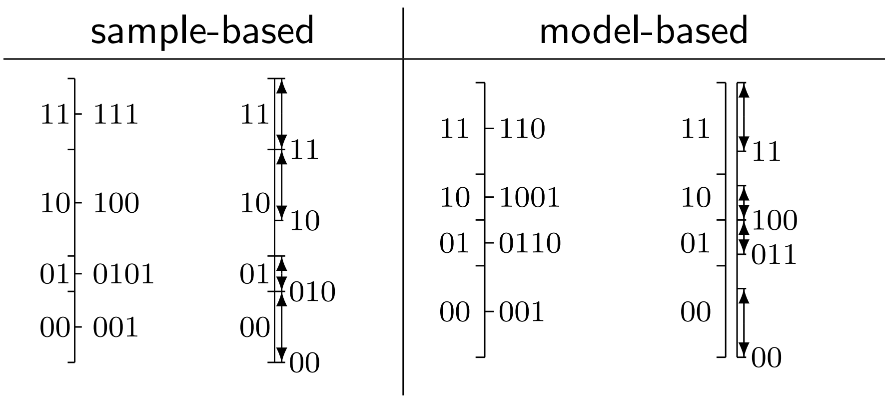
$$x_1 x_2 \cdots x_m \to \left[\frac{x}{2^m}, \frac{x+1}{2^m}\right) \subset [0,1)$$



  - For example, $110 \to \left[\frac{3}{4}, \frac{7}{8}\right)$

Arithmetic Coding – Symbol

- "Algorithm"
  - No preset codeword lengths for rounding off
  - Instead, *the largest dyadic interval inside the symbol interval* gives the codeword for the symbol
  - Example: Shannon-Fano-Elias vs. arithmetic symbol code



Arithmetic Coding – Stream

- Works for streams as well!
- Consider binary strings, order strings according to their corresponding integers (e.g., $0111 < 1000$), let

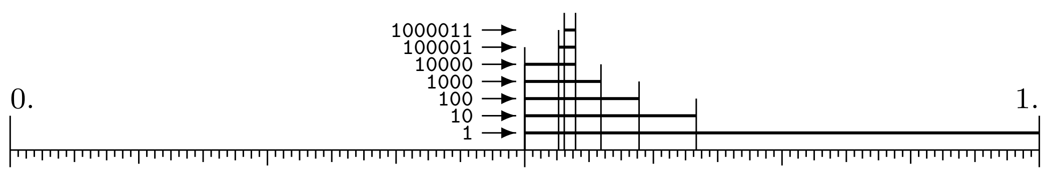
$$F(x_1^N) = \sum_{y_1^N \le x_1^N} \Pr(X_1^N = y_1^N) = \sum_{k: x_k = 1} p(x_1 x_2 \cdots x_{k-1} 0) + p(x_1^N)$$

Sum over all strings to the left of x_1^N in a binary tree (with $00 \cdots 0$ to the far left)

- Code x_1^N into largest interval inside

$$[F(x_1^N) - p(x_1^N), F(x_1^N))$$

- Markov source example (model-based)



Arithmetic Coding – Adaptive

- Only the distribution of the current symbol conditioned on the past symbols is needed at every step

⇒ Easily made adaptive: just estimate $p(x_{n+1}|x_1^n)$

- One such estimate is given by the Laplace model

$$\Pr(x_{n+1} = x|x_1^n) = \frac{n_x + 1}{n + |\mathcal{X}|}$$

Lempel-Ziv: A Universal Code

- Not a symbol code
- Quite another philosophy: parsings, phrases, dictionary
- A *parsing* divides x_1^n into phrases $y_1^{c(n)}$

$$x_1 x_2 \cdots x_n \to y_1, y_2, \ldots, y_{c(n)}$$

- In a *distinct parsing* phrases do not repeat
- The LZ algorithm performs a greedy distinct parsing, whereby each new phrase extends an old phrase by just 1 bit

⇒ The LZ code for the new phrase is simply the dictionary index of the old phrase followed by the extra bit

- There are several variants of LZ coding, we consider the “basic” and the “modified” LZ algorithms

The “Basic” Lempel-Ziv Algorithm

- Lempel-Ziv parsing and “basic” encoding of s

phrases	λ	1	0	00	01	10	100	011	11
indices	0	0	0	0	0	0	0	0	1
	0	0	0	0	1	1	1	1	0
	0	0	1	1	0	0	1	1	0
	0	1	0	1	0	1	0	1	0
encoding		,1	0,0	10,0	10,1	001,0	101,0	100,1	001,1

- Remarks
  - Parsing starts with empty string
  - First pointer sent is also empty
  - Only “important” index bits are used
  - Even so, “compressed” 16 bits to 25 bits

The “Modified” Lempel-Ziv Algorithm

- The second time a phrase occurs,
  - the extra bit is *known*
  - it cannot be extended a distinct third way
  - $\Rightarrow$ the second extension may overwrite the parent
- Lempel-Ziv parsing and “modified” encoding of s

phrases	λ	1	0	00	01	10	100	011	11
indices	0	0	0	0	0	0	1	1	0
	0	0	0	1	0	1	0	0	0
	0	1	0	0	0	1	0	1	1
encoding		,1	0,	0,0	00,	01,0	11,0	000,1	001,

$\Rightarrow$ saved 5 bits! (still 16:19 “compression”)

Asymptotic Optimality of LZ Coding

- Codeword lengths of Lempel-Ziv codes satisfy (index + extra bit)
$$l(x_1^n) \leq c(n)(\log c(n) + 1)$$
- Using a counting argument, the number of phrases $c(n)$ in a distinct parsing of a length n sequence is bounded as
$$c(n) \leq \frac{n}{\log n}(1 + o(1))$$
- Ziv's lemma relates distinct parsings and a k^{th}-order Markov approximation of the underlying distribution.

- Combining the above leads to the optimality result:
  - *For a stationary and ergodic source* $\{X_n\}$,
$$\limsup_{n\to\infty} \frac{1}{n} l(X_1^n) \leq H(\mathcal{S}) \qquad \text{a.s.}$$

Generating Discrete Distributions from Fair Coins

- A natural inverse to data compression
- Source encoders aim to produce i.i.d. fair bits (symbols)
- Source decoders *noiselessly* reproduce the original source sequence (with the proper distribution)

$\Rightarrow$ "Optimal" source decoders provide an *efficient* way to generate discrete random variables