3 Entropy and Lossless Coding

3.1 Entropy and Information

Source alphabet: regard amplitude value of image pixel as random process X

⇨ Set of possible outcomes denoted as source alphabet A_X with finite number elements a_i

$$A_X = \{a_0, a_1, \ldots\}$$

Example: grey levels for an 8 bit representation $A_X = \{0, 1, \ldots, 255\}$

Probability mass function (PMF): each amplitude value occurs at a certain probability with statistical properties described by

$$p_X(x) = \mathrm{Prob}(X = x) \quad \text{for each} \quad x \in A_X$$

Binary alphabet: special case $A_X = \{0,1\}$ extensively used in lossless coding

Expectation of random variable X defined by $\mathrm{E}\{X\} = \sum_{x \in A_X} x \cdot p_X(x)$

⇨ Statistical average or mean

Entropy of a Random Variable

Given: random variable X with source alphabet A_X and PMF $p_X(x)$

Self-information associated with event $X = x$

$$h_X(x) = -\log_2 p_X(x)$$

Entropy of random variable X defined as expected value of self-information

$$H(X) = \mathrm{E}\{h_X(x)\} = -\sum_{x \in A_X} p_X(x) \log_2 p_X(x)$$

⇨ Average amount of information contained in random variable X

Basic properties of information:

- Unit of information is [bit] if binary logarithm $\log_2(.)$ is used
- Information is positive $h_X(x) \geq 0$
- Information $h_X(x)$ strictly increases with decreasing probability $p_X(x)$ (unlikely events carry more information when occurring)

Entropy of a Random Variable (cont.)

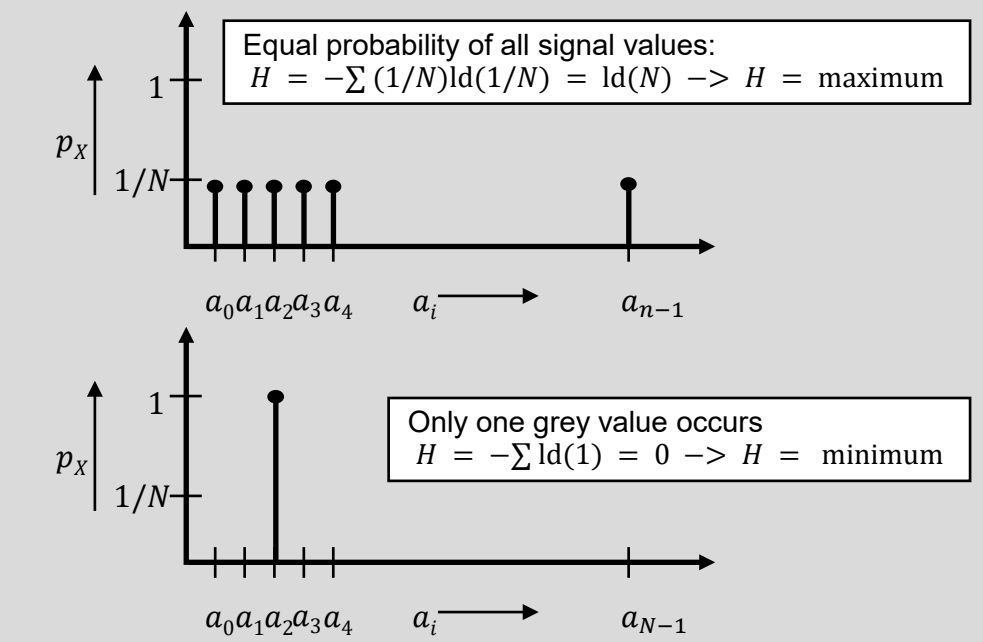
Entropy is bounded below and above by

$$0 \leq H(X) \leq \log_2 \|A_X\|$$

Equality if only one outcome can occur

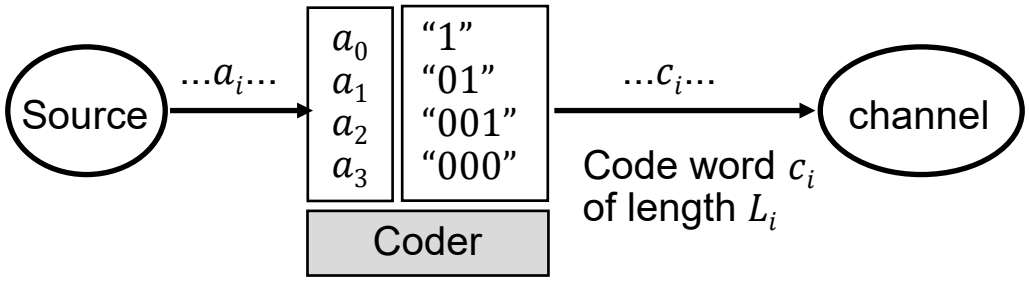
Maximum if all outcomes are equally probable

Example:



Entropy and Bit Rate

Average code word length R of a code $\{c_i\}$ assigned to source X with alphabet A_X and PMF $p_X(x)$:



$$R = \sum_{x \in A_X} L(c_x) \cdot p_X(x)$$

Shannon's noiseless source coding theorem:

- Entropy $H(X)$ is the lower bound for the average word length R, i.e. there can't be a code with an average word length smaller than $H(X)$

$$R \geq H(X)$$

- Average word length R can approach the entropy $H(X)$ arbitrarily close if sufficiently large number of symbols are coded jointly

Redundancy of a code is defined as:

$$r = R - H(X) \geq 0$$

3.2 Variable Length Codes

Given: Memoryless random process X with alphabet A_X and PMF p_X

Variable length code: design a set of code words $\{c_x\}$, such that

- Each element x is associated with a distinct code word c_x
- c_x is a string of length $L_x = L(c_x)$ bits.
- Sequence of outcomes from the random process is represented by concatenated code words
- Each element x can be determined from concatenated codewords

Notation: codes with this property are said to be “uniquely decodable”

Prefix codes:

- No code word is a prefix of any other code word
- Symbol by symbol uniquely decodable in transmission order

Example of Non-Decodable Code

Consider the quaternary alphabet $A_X = \{a_0, a_1, a_2, a_3\}$

Symbol	Code word
a_0	"0"
a_1	"01"
a_2	"10"
a_3	"11"

Coding examples

- Sequence a_0, a_2, a_3, a_0, a_1 gives "0 10 11 0 01"
- Sequence a_1, a_0, a_3, a_0, a_1 gives "01 0 11 0 01"

Observation: same bit stream for different sequences of source symbols

- Ambiguous, not uniquely decodable
- Not a prefix code

Sequential Decoding of Prefix Codes

Decoding variable length codes

- Any prefix code can be uniquely decoded from a bit-stream
- Use sequential “match” and “move ahead“ principle

Sequential decoding algorithm

For *each length* $L = 1,2, \ldots$
    For *each element* $x = a_i$ in alphabet A_X
        Compare the first L bits of the received bit-stream with codeword c_x
        If a match is found
            symbol must be $x = a_i$ since no larger value of L will give a match (due to prefix condition)
            remove initial L bits from bit-stream
            apply algorithm recursively
    end
end

McMillan and Kraft Condition

McMillan theorem: A necessary condition for unique decodability is that the code word lengths $L_x = L(c_x)$ satisfy

$$\sum_{x \in A_X} 2^{-L_x} \leq 1$$

Kraft theorem: Given any set of lengths L_x satisfying the McMillan condition, there exists a prefix code

Conclusions

- McMillan inequality is both necessary and sufficient for unique decodability
- Also known as Kraft-McMillan inequality
- No need to consider any other codes than prefix codes
- But: prefix code is not unique
  - Example: flip "0" and "1" in unary code

Redundancy Free Coding

Optimal code without redundancy, i.e. $R = H(X)$, can be achieved if all individual code word lengths satisfy

$$L_i = -\log_2 p_X(a_i)$$

Since L_i must be an integer value, all probabilities have to be binary fractions

$$p_X(a_i) = 2^{-L_i}$$

Example for code without redundancy:

a_i	$p_X(a_i)$ $[L_i]$	Redundant code	Optimum code
a_0	0.500 [1]	"00"	"0"
a_1	0.250 [2]	"01"	"10"
a_2	0.125 [3]	"10"	"110"
a_3	0.125 [3]	"11"	"111"

With $\log_2 2^K = K$ we get

$$H(X) = 1.75 \text{ bit}$$
$$R = 1.75 \text{ bit}$$
$$r = 0$$

Redundancy of Prefix Codes

Theorem: for any PMF p_X a prefix code can be found whose rate R satisfies

$$H(X) \leq R \leq H(X) + 1$$

Proof

- Left hand inequality follows from Shannon's noiseless coding theorem
- Right hand inequality
  - choose $L_x = \lceil -\log_2 p_X(x) \rceil$
  - these lengths satisfy McMillan theorem
  - resulting prefix code has rate

$$\begin{aligned} R &= \sum_{x \in A_X} p_X(x) \lceil -\log_2 p_X(x) \rceil \\ &< \sum_{x \in A_X} p_X(x)(1 - \log_2 p_X(x)) \\ &= H(X) + 1 \end{aligned}$$

3.3 Huffman Coding

Algorithm for variable length coding proposed by *Huffman* in 1952 which always finds a prefix code with minimum redundancy:

Huffman code construction

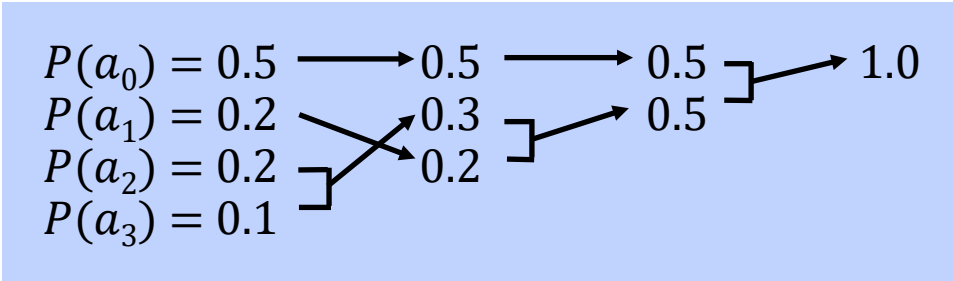
Start with list of symbol probabilities sorted in descending order

Repeat /* construct binary tree */
    Pick the two symbols with lowest probability
    - merge them into new auxiliary symbol
    - calculate the probability of the auxiliary symbol
Until only one symbol remains in auxiliary alphabet

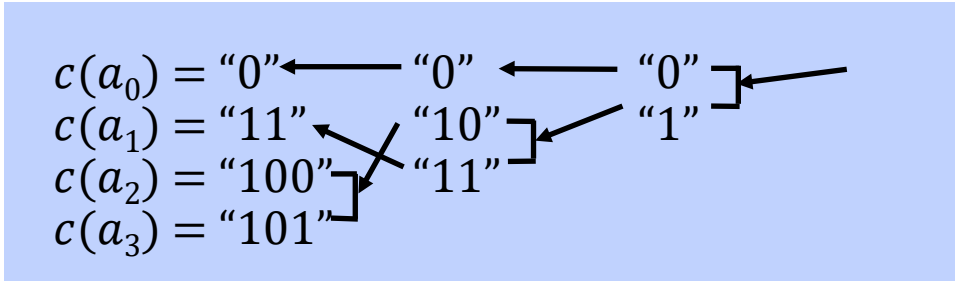
For all branches in tree /* convert tree into a prefix tree */
    Add “0” and “1” to upper and lower branch in binary tree
end
Construct code words for each symbol by traversing tree from root to leaves

Example for Huffman Code

Construction of binary tree



Convert tree into prefix code



Binary fixed length code	$R = 2$ bit
Huffman code	$R = 1.8$ bit
Entropy	$H(X) \cong 1.761$ bit
Redundancy of Huffman code	$r \cong 0.04$ bit, i.e. 2.2%

Huffman Code – Second Example

Symbol	Binary code	Probability							Huffman code
a_o	"000"	[00] 0.25	[00] 0.25	[00] 0.25	[00] 0.25	[00] 0.25	[00] 0.25	[0] 0.54	"00"
a_1	"001"	[10] 0.21	[10] 0.21	[10] 0.21	[10] 0.21	[10] 0.21	[1] 0.46	[1] 0.46	"10"
a_2	"010"	[010] 0.15	[010] 0.15	[010] 0.15	[010] 0.15	[01] 0.29	[01] 0.29		"010"
a_3	"011"	[011] 0.14	[011] 0.14	[011] 0.14	[011] 0.14				"011"
a_4	"100"	[1100] 0.0625	[1100] 0.0625	[110] 0.125	[11] 0.25	[11] 0.25			"1100"
a_5	"101"	[1101] 0.0625	[1101] 00.625						"1101"
a_6	"110"	[1110] 0.0625	[111] 0.125	[111] 0.125					"1110"
a_7	"111"	[1111] 0.0625							"1111"

Binary fixed length code	$R = 3$ bit
Huffman code	$R \cong 2.79$ bit
Entropy	$H(X) \cong 2.781$ bit
Redundancy of Huffman code	$r \cong 0.01$ bit, i.e. 0.3 %

Vector Huffman Coding

Redundancy of Huffman coding as for general prefix code given by

$$H(X) \leq R \leq H(X) + 1$$

⇨ Very inefficient if entropy of the source much smaller than 1 bit/symbol

Solution: vector Huffman coding

- Combine m source symbols into m-dimensional hyper-symbols
- Code hyper-symbols using Huffman algorithm

Redundancy of vector Huffman coding

$$H(X) \leq R \leq H(X) + \frac{1}{m}$$

Disadvantage: complexity

- Required alphabet has size of

$$\|A_X\|^m$$

symbols and grows exponentially with m

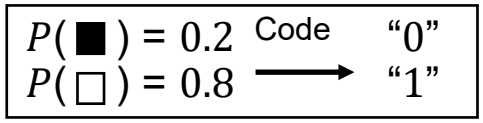
Example for Vector Huffman Coding

Encoding each sample with single code word

- Alphabet: { “b”; “w”}
- Probabilities $P(a_i)$ and Huffman code c_i and word length L:

$P(\text{“b”}) = 0.2$	$c_0 = \text{“0”}$	$L_0 = 1$
$P(\text{“w”}) = 0.8$	$c_1 = \text{“1”}$	$L_1 = 1$

- Entropy $H = -(0.2\,\mathrm{ld}0.2 + 0.8\,\mathrm{ld}0.8)$ bit / pixel $= 0.72$ bit / pixel
- Average code word length $R = 1$ bit / pixel

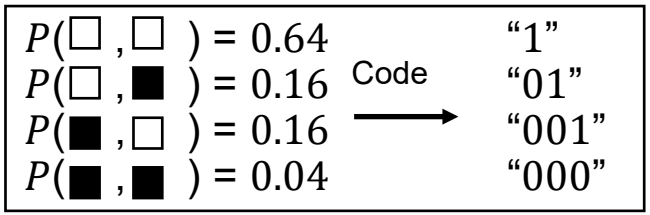


Encoding two successive samples with one code word

- Alphabet: { “w,w”; “w,b”; “b,w”; “b,b” }
- Probabilities $P(a_i)$ and Huffman code c_i and word length L:

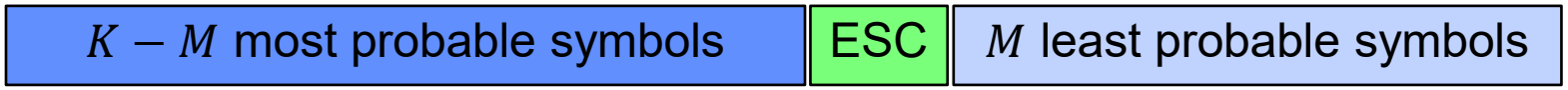
$P(\text{“w,w”}) = 0.64$	$c_0 = \text{“1”}$	$L_0 = 1$
$P(\text{“w,b”}) = 0.16$	$c_1 = \text{“01”}$	$L_1 = 2$
$P(\text{“b,w”}) = 0.16$	$c_2 = \text{“001”}$	$L_2 = 3$
$P(\text{“b,b”}) = 0.04$	$c_3 = \text{“000”}$	$L_3 = 3$

- Entropy:
  $H = -(0.64\,\mathrm{ld}0.64 + 2 \cdot 0.16\,\mathrm{ld}0.16 + 0.04\,\mathrm{ld}0.04)$ bit /two pixel
  $= 1.44$ bit / two pixel
  $= 0.72$ bit / pixel
- Average code word length:
  $R = (1 \cdot 0.64 + 2 \cdot 0.16 + 3 \cdot 0.16 + 3 \cdot 0.04)$ bit / two pixel
  $= 1.56$ bit / two pixel
  $= 0.78$ bit / pixel



Truncated Huffman Coding

Idea: limit size of Huffman code table and maximum code word length by Huffman coding only most probable symbols, speed up coding



Design of code words using truncated alphabet:

- Combine M least probable symbols of an alphabet of size K into auxiliary symbol “ESC” and calculate its symbol probability
- Construct Huffman code for alphabet consisting of remaining $K - M$ most probable symbols including the symbol “ESC”

Coding using truncated alphabet:

- Code $K - M$ most probable symbols as in conventional Huffman coding
- If symbol from M least probable set has to be coded, send “ESC” code and append $\log_2(M)$ bits to specify which symbol

Disadvantage: average code word length is increased, suitable M required

Adaptive Huffman Coding

Source statistics may be unknown in advance or vary over time / space

⇨ Symbol probabilities $p_X(x)$ have to be estimated

Forward adaptation

- Measure symbol probabilities by analyzing source data
- Transmit Huffman code table ahead of compressed bit-stream
- Implemented in JPEG standard (instead of available default tables)

Backward adaptation

- Measure symbol probabilities both at encoder and decoder, using the same previously decoded data
- Regularly generate identical Huffman tables at transmitter and receiver
- Saves overhead of forward adaptation but less efficient code tables since based on past observations
- Mostly avoided due to high computational burden at decoder

🗁 Demo 3 „Huffman Coding“

3.4 Unary Code

Given: memoryless source with an alphabet of all non-negative integers and a "geometric" distribution

Alphabet $A_X = \{0,1,2,\ldots\} = \mathrm{Z}_+$ PMF $p_X(x) = 2^{-(x+1)}$

Construct code words c_x as *x*-times "1" followed by a terminating "0"

Symbol	Code word
0	"0"
1	"10"
2	"110"
3	"1110"
...	...

- Length of each code word is $L(c_x) = x + 1$ bit
- Uniquely decodable, "comma" code
- Average code word length is minimum, i.e. unary code is an optimum code

Unary Code (cont.)

Proof that unary code reaches entropy

$$R = \sum_{x \in A_X} L(c_x) \cdot p_X(x)$$

$$= \sum_{x \in A_X} (x+1) \cdot p_X(x) = \sum_{x \in A_X} -\log_2 p_X(x) \cdot p_X(x) = H(X)$$

Generalization: consider geometric source with faster decay, i.e. distribution

$$p_X(x) = (1-\beta)\beta^x \text{ with } 0 < \beta < \frac{1}{2}; \quad x \geq 0$$

⇨ Unary code is still optimal prefix code

Proof: can be seen by applying Huffman code construction to source

However: unary code is not redundancy-free if $\beta < \frac{1}{2}$

3.5 Golomb and Rice Coding

Assumption: geometric source with slower decay, i.e. distribution

$$p_X(x) = (1-\beta)\beta^x \text{ with } \frac{1}{2} < \beta < 1\,; \quad x \geq 0$$

Idea: express each element x as quotient and remainder upon division of x by some integer number m

$$x = mx_\mathrm{q} + x_\mathrm{r} \text{ with } x_\mathrm{q} = \left\lfloor \frac{x}{m} \right\rfloor \text{ and } x_\mathrm{r} = x \bmod m$$

Distribution of new random variables

$$p_{X_\mathrm{q}}(x_\mathrm{q}) = \sum_{i=0}^{m-1} p_X\left(mx_\mathrm{q} + i\right) = \beta^{mx_\mathrm{q}}(1-\beta)\sum_{i=0}^{m-1} p_X(i) \text{ geometric source with } \beta^\mathrm{m}$$

$$p_{X_\mathrm{r}}(x_\mathrm{r}) = \frac{1-\beta}{1-\beta^m}\beta^{x_\mathrm{r}} \text{ for } 0 \leq x_\mathrm{r} < m$$

Random variables X_q and X_r are statistically independent

Optimum performance of Golomb code if $\beta^m \approx \frac{1}{2}$

Golomb Code

Algorithm for Golomb coding:

- Select integer divisor m such that $\beta^m \approx 1/2$
- Optimally encode quotient x_q using unary code
- Use truncated binary code to represent remainder x_r with maximum code word length of $k = \lceil \log_2 m \rceil$ bit:

  if $x_\mathrm{r} < 2^k - m$ code x_r as binary with $k-1$ bit
  if $x_\mathrm{r} \geq 2^k - m$ code $x_\mathrm{r} + (2^k - \mathrm{m})$ as binary with k bit

- Concatenate bits for x_q and x_r

Example for Golomb code with parameter $m = 5$ ⇨ $k = 3$ and $2^k - m = 3$

$$x = 21: \quad x_\mathrm{q} = \left\lfloor \frac{21}{5} \right\rfloor = 4, \qquad x_\mathrm{r} = 21 \bmod 5 = 1$$

⇨ Since $x_\mathrm{r} = 1 < 2^3 - 5 = 3$, the codeword is constructed with $k-1 = 2$ bits for x_r as

$$\text{“}\underbrace{11110}_{x_\mathrm{q}=4}\ \underbrace{01}_{x_\mathrm{r}=1}\text{“}$$

Golomb Codes for Different Parameters m

x	Binary	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$
0	0000	0	0 0	0 0	0 00	0 00
1	0001	10	0 1	0 10	0 01	0 01
2	0010	110	10 0	0 11	0 10	0 10
3	0011	1110	10 1	10 0	0 11	0 110
4	0100	11110	110 0	10 10	10 00	0 111
5	0101	111110	110 1	10 11	10 01	10 00
6	0110	1111110	1110 0	110 0	10 10	10 01
7	0111	11111110	1110 1	110 10	10 11	10 10
8	1000	111111110	11110 0	110 11	110 00	10 110
9	1001	1111111110	11110 1	1110 0	110 01	10 111
10	1010	11111111110	111110 0	1110 10	110 10	110 00
...	...	...	...	...	...	...

Rice Code

Special case of Golomb code simplifying implementation

- Restrict Golomb parameter m to exact power of two

$$m = 2^k$$

- x_q formed by discarding least significant k bits from x, write as unary code
- Discarded k bits form remainder x_r, code with constant codeword length of $k = \log_2 m$ bits

Example for Rice code with parameter $m = 4$ ⇨ $k = \log_2 4 = 2$

$$x = 21: \quad x_\mathrm{q} = \left\lfloor \frac{21}{4} \right\rfloor = 5, \qquad x_\mathrm{r} = 21 \bmod 4 = 1$$

Binary representation: $x =$ “101 01”: $x_\mathrm{q} =$ “101”, $x_\mathrm{r} =$ “ 01”

Resulting codeword: “$\underbrace{111110}_{x_\mathrm{q}=5}\ \underbrace{01}_{x_\mathrm{r}=1}$“

Rice Codes for Different Parameters k

x	Binary	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$
0	0000	0	0 0	0 00	0 000	0 0000
1	0001	10	0 1	0 01	0 001	0 0001
2	0010	110	10 0	0 10	0 010	0 0010
3	0011	1110	10 1	0 11	0 011	0 0011
4	0100	11110	110 0	10 00	0 100	0 0100
5	0101	111110	110 1	10 01	0 101	0 0101
6	0110	1111110	1110 0	10 10	0 110	0 0110
7	0111	11111110	1110 1	10 11	0 111	0 0111
8	1000	111111110	11110 0	110 00	10 000	0 1000
9	1001	1111111110	11110 1	110 01	10 001	0 1001
10	1010	11111111110	111110 0	110 10	10 010	0 1010
...	...	...	...	...	...	...

Golomb Parameter Estimation

Many sources follow a roughly geometric distribution, e.g. run lengths

- Golomb parameter $m = 2^k$ has to be estimated

Approach: use expected value for geometric distribution

$$\mathrm{E}\{X\} = \sum_{x=0}^{\infty} (1-\beta) x \beta^x = (1-\beta) \sum_{x=1}^{\infty} (x-1) \beta^{x-1}$$
$$= \left\{ (1-\beta) \frac{\mathrm{d}}{\mathrm{d}\beta} \sum_{x=0}^{\infty} \beta^x \right\} - 1 = \left\{ (1-\beta) \frac{\mathrm{d}}{\mathrm{d}\beta} \frac{1}{1-\beta} \right\} - 1 = \frac{\beta}{1-\beta}$$

Approximation for $(1-\beta) << 1$

$$\beta^m = (1-(1-\beta))^m \approx 1 - m(1-\beta)$$
$$\approx 1 - \frac{m}{\mathrm{E}\{X\}} \overset{!}{\approx} \frac{1}{2}$$

for optimum performance of Golomb code

Select m such that

$$m = 2^k \approx \frac{1}{2}\mathrm{E}\{X\} \quad \rightarrow \quad k = \max\left\{0, \left\lceil \log_2 \left(\frac{1}{2}\mathrm{E}\{X\} \right) \right\rceil \right\}$$

also suitable strategy if $(1-\beta) \approx 1$

Adaptive Golomb Coder

Non-stationary source: parameters of Golomb code may change over time

- Employ adaptation strategy
- Used by JPEG-LS lossless image compression standard

Adaptive Golomb coder

Initialize

$A =$ initial estimate of $\mathrm{E}\{X\}$ and $N = 1$ /* ratio A/N is estimate of mean */

For each $n = 0,1,2,\ldots$

Set $k = \max\{0, \lceil \log_2(A/(2N)) \rceil\}$ /* pick best Golomb code */

Code symbol $x(n)$ using the Golomb code with parameter k

If $N = N_{\max}$

Set $A \leftarrow \lfloor A/2 \rfloor$ and $N \leftarrow \lfloor N/2 \rfloor$ /* avoid overflow, forget past */

Update $A \leftarrow A + x(n)$ and $N \leftarrow N + 1$

end

Exponential Golomb Code

Universal code with parameter k, often abbreviated as Exp-Golomb code

- Formally not a Golomb code, instead related to Elias gamma code
- Used in H.264/AVC video coding standard to encode non-negative integers

Coding algorithm

- Set y as x in binary except last k bits and increase y by 1
- Write length $L(y) - 1$ as unary code
- Write y as binary without MSB

} Elias gamma coding of value y

- Append k bits

Example for $k = 0$

- $x = 12$: $y =$ "1100"+1 = "1101", $L(y) - 1 = 3$ ⇨ "1110", "101"

Example for $k = 1$

- $x = 12$: $y =$ "110"+1 = "111", $L(y) - 1 = 2$ ⇨ "110", "11", "0"

Exponential Golomb Codes for Different Parameters k

x	Binary	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$
0	0000	0	0 0	0 00	0 000	0 0000
1	0001	10 0	0 1	0 01	0 001	0 0001
2	0010	10 1	10 0 0	0 10	0 010	0 0010
3	0011	110 00	10 0 1	0 11	0 011	0 0011
4	0100	110 01	10 1 0	10 0 00	0 100	0 0100
5	0101	110 10	10 1 1	10 0 01	0 101	0 0101
6	0110	110 11	110 00 0	10 0 10	0 110	0 0110
7	0111	1110 000	110 00 1	10 0 11	0 111	0 0111
8	1000	1110 001	110 01 0	10 1 00	10 0 000	0 1000
9	1001	1110 010	110 01 1	10 1 01	10 0 001	0 1001
10	1010	1110 011	110 10 0	10 1 10	10 0 010	0 1010
...	...	...	...	...	...	...

3.6 Arithmetic Coding

Basic idea: assign one variable length code word to whole data string instead of one code word for each source symbol

- Reaches entropy bound closer than Huffman (redundancy of 1 bit/sample)
- Less complex than vector Huffman coding

Arithmetic coding: sequence of symbols $\boldsymbol{X}$ is transformed to a probability interval with known lower and upper boundary

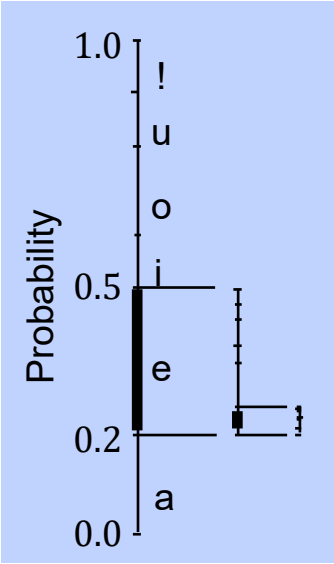
- Length of probability interval is proportional to likelihood of string
- Interval contains one binary number that can be represented by R bits with

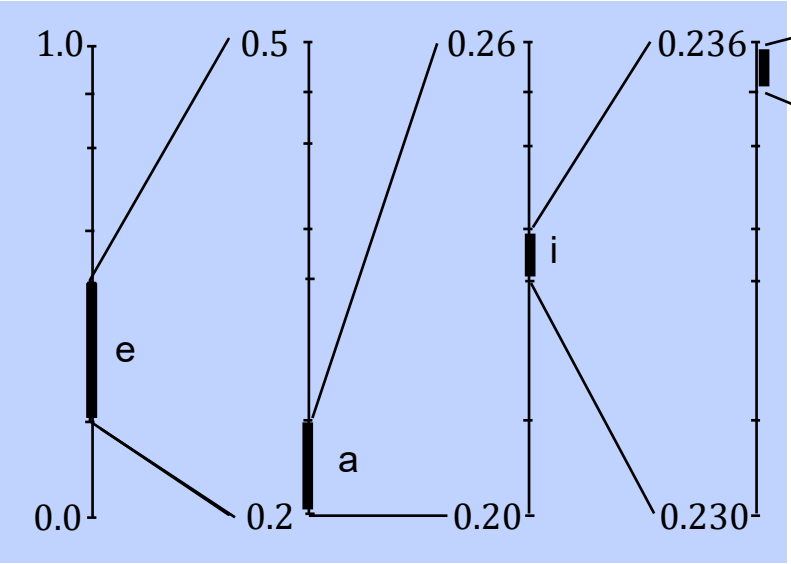
$$H(\boldsymbol{X}) \leq R \leq H(\boldsymbol{X}) + 2$$

Sequential processing in arithmetic coding

- Whenever lower and upper bound of probability interval have same MSBs, these can be shifted out
- EOF symbol required to define end of code word

Example for Arithmetic Coding





0.2360

0.2354

width ~ prob. of string

Model for alphabet {a,e,i,o,u,!}

Symbol	Probability	Range
a	.2	[0, 0.2)
e	.3	[0.2, 0.5)
i	.1	[0.5, 0.6)
o	.2	[0.6, 0.8)
u	.1	[0.8, 0.9)
!	.1	[0.9, 1.0)

Coding sequence "e,a,i,!":

1. Start from interval $[0, 1]$,
2. Symbol "e" with range $[0.2, 0.5]$
   narrows interval to $[0.2, 0.5]$ and length 0.3
3. Symbol "a" with range $[0, 0.2]$,
   narrows interval to $[0.20, 0.26]$ and length 0.06
4. Symbol "i" with range $[0.5, 0.6]$
   narrows interval to $[0.230, 0.236]$ and length 0.006
5. (e,a,i,!) is in interval $[0.2354, 0.2360]$

Binary code word is "00111100011"

Corresponds to $P = 0.2358398438$ from interval

Decoding sequence "e,a,i,!":

1. Code word is included in interval e
   for first letter
2. Simulating the encoding process
   reconstructs the whole sequence

Arithmetic Coding Algorithm

Arithmetic coding (Elias algorithm)

Calculate list of symbol probabilities $p_i, i = 0, \ldots, N-1$
Calculate cumulative probabilities $q_0 = 0\,; q_i = \sum_{k=1}^{i} p_{k-1}, i = 0, \ldots, N-1$

Set lower boundary $l_0 = 0$, upper boundary $u_0 = 1$ /* initialize boundaries */
Set interval $d_0 = 1$ /* initialization of interval */

For n-th symbol $x(n) = a_i$ to be send

$d_n = d_{n-1} \cdot p_i$ /* set length of interval */
$l_n = l_{n-1} + d_{n-1} \cdot q_i$ /* set lower boundary of interval */
$u_n = l_n + d_n$ /* set upper boundary of interval */
if (first new bits of binary representation of l_n and u_n are the same) then
shift new bits out
End

Send finalization bits /* such that binary code interval is within l_n and u_n */

Implementation of Arithmetic Coding

Advantages of arithmetic coding

- Encoding a sequence of symbols leads to smaller code rate R, redundancy when coding N symbols is bound by $2/N$ bits per symbol
- Probability model can be continuously adapted if this happens on both sides simultaneously

Disadvantage: Calculation of interval boundaries using the Elias algorithm principally requires infinitely high precision computation

Remedy: finite precision implementation using rescaling of interval length when bits have been shifted out to receiver

⇨ Known as *arithmetic coding algorithm*

Entropy and Lossless Coding - Summary

- Redundancy reduction exploits signal statistics
- Entropy is lower bound for average code word length
- Huffman code is an optimum prefix code
- Golomb-Rice code optimum solution for exponentially distributed sources
- Arithmetic coding as universal coding method for strings of symbols