EQ2845 – Information Theory and Source Coding

Assignment #3

published on Feb 8th, due Feb 17th, 2024

1. (3 points)

   (a) We design an D-ary prefix-free code for an 6-ary random message U. Our specifications say that the codeword lengths should be $(l_1, l_2, \ldots, l_6) = (1, 1, 2, 3, 2, 3)$. Find a good lower bound on D. How this bound is improved if we would like to design a uniquely decodable code?

   (b) Consider a a random message U that takes on four values with probabilities $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{12}\right)$. Design all possible 2-ary Huffman codes for this random message. Conclude that there are optimal codes with codeword lengths for some symbols that exceed the Shannon code length $\lceil \log \frac{1}{p(x)} \rceil$.

   (c) Construct a binary Huffman code for the following distribution on five symbols: $\mathbf{p} = (0.3, 0.3, 0.2, 0.1, 0.1)$. What is the average length of this code? Construct a probability distribution $\mathbf{p}'$ on five symbols for which the code that you constructed has an average length (under $\mathbf{p}'$) equal to its entropy $H(\mathbf{p}')$.

   (d) Can $\mathbf{l}_1 = (1, 2, 2)$ and $\mathbf{l}_2 = (2, 2, 3, 3)$ be the word lengths of a binary Huffman code? Can $\mathbf{l}_3 = (1, 2, 2, 2, 2)$ and $\mathbf{l}_4 = (2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3)$ be the word lengths of a ternary Huffman code? Justify your answers.

2. (Run-length coding-4 points) Markov-1 sources can output long sub-sequences of 0s and/or 1s. Run-length coding schemes have been adapted to handle this situation. General run-length coding schemes map any input sequence of symbols from a finite alphabet to a sequence of symbol pairs, representing symbol values and run-lengths. In the case of binary sources, 0 and 1 runs will alternate and the transmission of symbol values is not necessary for each run. It is sufficient to indicate whether the input sequence starts with a 0 or a 1.

   (a) Write a Matlab function that replaces a string of 0s and 1s by run-length values. Also indicate whether your sequence starts with a 0 or a 1.

   (b) Assume that the run-length values are optimally encoded with a binary code. Write a Matlab function that takes the array of run-length values and calculate the length (in bits) of the optimum binary stream.

   (c) Write a Matlab function that produces a Markov-1 character string of 0s and 1s. Set $\alpha = \beta$ and allow for α the values 0.05:0.05:0.95. Generate for each α_i a source stream of $L = 19600$ bits. Compress each source stream with the run-length encoder. Plot the compression ratio (length in bits of the binary source stream / length in bits of the binary code stream) versus the values of α. What do you observe? Plot the pmf of the run-length values for the values $\alpha = 0.05$, 0.5, and 0.95. What do you observe? Explain your observation.

3. (Golomb coding-3 points) Golomb coding can achieve close to the optimal variable length coding performance for sources with roughly geometric distribution. We will encode the array of run-length values in Problem 2 and generate a sequence of binary Golomb codewords in order to compare its length to that of the optimum binary stream in Problem 2. We will use the following adaptive Golomb coder:

```
Initialize
N = 1
A = initial estimate of the average run-length value
Nmax = counter value for renormalization

FOR each run-length value r DO
estimate parameter k = max{0, ceil(log2(A/(2*N)))}

code r using Golomb code with parameter k

IF N = Nmax
A = floor(A/2)
N = floor(N/2)
END
A = A + current run-length value
N = N + 1
END
```

The Golomb code $\{0\cdots 01b_k\cdots b_2b_1\}$ with parameter k for the run-length value r is a combination of a unary code $\{0\cdots 01\}$ with a constant length code $\{b_k\cdots b_2b_1\}$, where $b_\nu \in \{0,1\}$. A unary codeword of length l has $l-1$ leading 0s and is terminated by the symbol 1. As the class of unary codes is efficient for sources with geometric distribution, short codewords are mapped to small source values. k is the length of the constant length code with 2^k different codewords. The value r_u that is coded with the unary code is

$$r_u = \text{floor}\left(\frac{r}{2^k}\right),$$

and the value r_c that is coded with the constant length code is given by

$$r_c = r \bmod 2^k.$$

This decomposition is unique and the run-length value r can be reconstructed with $r = 2^k r_u + r_c$.

(a) Write a Matlab function that takes an array of run-length values and determines the corresponding string of binary Golomb codewords.

(b) Concatenate the run-length encoder and the adaptive Golomb encoder. Consider to subtract the value 1 from all run-length values as our Golomb encoder expects input values from the set $\{0,1,2,\ldots\}$. Reuse the Markov-1 source from problem 2 to evaluate the performance of the concatenated encoder. Set $\alpha = \beta$ and allow for α the values 0.05:0.05:0.95. Generate for each α_i a source stream of $L = 19600$ bits. Compress each source stream with the run-length encoder. Plot the compression ratio dependent on the values of α for both the ideal encoder for the run-length values and the adaptive Golomb encoder. Explain your observations. What values do you suggest for `A` and `Nmax`?

4. (Arithmetic coding-5 points) We will realize a binary arithmetic encoder with finite precision. The probabilities are represented by P-bit integers, and the interval length by an N-bit integer. Let $\{x_n\}$ with $n = 0, 1, \ldots$ denote the stationary binary input sequence and let p_0 denote the P-bit integer which is used to represent the probability $f_X(x = 0)$:

$$p_0 = \lfloor 2^P f_X(x = 0) \rfloor$$

The output of the encoder can be written as a binary fraction of the evolving codeword. It consists of three key segments:

$$0.\underbrace{xxxxxxxx\ldots x}_{b-r-1\text{ bits}}\underbrace{011\ldots 1}_{r+1\text{ bits}}\underbrace{cc\ldots c}_{C}$$

The initial $b - r - 1$ bits of the codeword may be sent to the decoder so that we need not allocate storage for them. The content of the register C with $N + P$ bits is appended at the end of the codeword. A possible overflow of the register C may affect the $r + 1$ bits of the central segment which, at first hand, consists of r 1s with one leading 0. An overflow of the register C causes a carry that changes the central segment such that a leading 1 is followed by a sequence of 0s. In addition, a special state, identified by $r = -1$, is necessary to deal with the possibility that an overflow of C may occur when $r = 0$, causing the 0 bit to flip to a 1 with no subsequent 0s. More details on the binary arithmetic encoder in Fig. 1 can be found in the textbook JPEG 2000, pp 60, by Taubman and Marcellin.

The algorithm employs additions and shift operations to $N + P$ bit registers which may overflow. Matlab variables provide 52 bits in a floating point integer such that overflows can easily be detected. We recommend the values $N = 22$ and $P = 8$ for the Matlab implementation. After an overflow, the Matlab variable can be 'masked' to its original $N + P$ bits. We suggest to manipulate Matlab variables with functions like `bitshift` to operate on the bit-level, see Fig. 2.

(a) Write a Matlab function with the probability parameter $f_X(x = 0)$ which encodes a binary stream.

For the following questions, consider the Markov-1 source from problem 2. Select $\alpha = \beta$ and generate streams of length $L = 20000$. Modify the binary arithmetic encoder such that it compresses a binary Markov-1 source. For that, the symbol probability has to be replaced by transition probabilities.

(b) Plot the estimated entropy of the Markov-1 stream and the estimated entropy of the code stream for $\alpha = 0.1 : 0.05 : 0.9$.

(c) Plot the estimated entropy rate of the Markov-1 stream and the estimated entropy rate of the code stream for $\alpha = 0.1 : 0.05 : 0.9$. Hint: Reuse the Matlab function from problem 2 to estimate the entropy rate ($m = 10$).

(d) Plot also the compression ratio for $\alpha = 0.1 : 0.05 : 0.9$. Compare it to the compression ratio of the cascaded run-length/Golomb encoder. Explain your observation.

```
Initialize C=0, A=2^N, r=-1, b=0
FOR each n=0,1,... DO
T = A * p0
IF xn = 1
C = C + T
T = 2^P * A - T
END

IF C >= 2^(N+P)
C = MOD(C, 2^(N+P))
% propagate carry
emit-bit(1)
IF r>0
execute r-1 times, emit-bit(0)
SET r=0
ELSE
SET r=-1
END
END

WHILE T < 2^(N+P-1)
% renormalize once
b = b + 1
T = 2 * T
C = 2 * C
IF C >= 2^(N+P)
C = MOD(C, 2^(N+P))
% overflow of C
IF r<0
emit-bit(1)
ELSE
r = r + 1
END
ELSE
% no overflow of C
IF r>=0
emit-bit(0)
execute r times, emit-bit(1)
END
SET r = 0
END
END (while)

SET A = floor(T / 2^P)
END (for)

IF r>= 0
emit-bit(0)
execute r times, emit-bit(1)
END
emit N+P bits of register C
```

Figure 1: Binary arithmetic encoder.

```
T = 2^P * A - T        →   T = bitshift(A,P) - T;
C = MOD(C, 2^(N+P))    →   C = bitand(C, mask);
```

Figure 2: Pseudocode to m-code.