

Problem 2.1 *The idea behind Arithmetic Coding: infinite case*

Recall that in class, we mentioned that any prefix-free code can be thought of as a binary search tree on the unit interval. This is related to the way in which every real number between 0 and 1 can be expressed as a semi-infinite binary string, and the fact that the uniform random variable on $[0, 1]$ is equivalent to an infinite sequence of fair coin tosses. We explore this connection in this problem, in a simplified way of course.

Arithmetic coding is actually used in many real-world systems for a diverse range of applications. In real systems, computational limitations like finite word-lengths impose interesting challenges. To overcome these, a lot of important techniques have been developed. If you are interested, please consult the literature.

- Assume that X is a real-valued random variable that is uniformly distributed on $[0, 1]$. Let $B(x) = 0.B_1B_2B_3\dots$ be the binary expansion of x so that $x = \sum_{i=1}^{\infty} B_i(x)2^{-i}$. Show that the sequence $\{B_i(X)\}$ is an i.i.d. sequence of fair Bernoulli random variables. (You can ignore any problems that arise from the fact that $0.01111\dots$ and 0.1 encode the same real number. Just break ties however you want.)
- Let B_i be an infinite sequence of fair Bernoulli random variables. Argue why $X = \sum_{i=1}^{\infty} B_i2^{-i}$ is a uniform random variable on $[0, 1]$. You can refer to the previous part.
- Let C_i be an infinite sequence of unfair Bernoulli random variables with $p(1) = p \neq \frac{1}{2}$ and $p(0) = 1 - p$. Use them to construct a real valued random variable Y as the limit of the sequence of random variables Y_i defined by the recursive procedure below (involving the helper random variable R_i).

$$\begin{aligned} Y_0 &= 0 \\ R_0 &= 1 \\ Y_i &= Y_{i-1} + R_{i-1}(1-p)C_i \\ R_i &= R_{i-1}(pC_i + (1-p)(1-C_i)) \end{aligned}$$

Show that Y_{∞} is a uniform random variable on the unit interval $[0, 1]$.

- Argue why knowing the real-valued realization of $Y_{\infty} = y$ essentially tells you the realization of the entire sequence of the C_i . (i.e., argue why the mapping given above from binary sequences to a real number is essentially invertible.)
- Let Y be a uniform random variable on $[0, 1]$. Give an explicit procedure for recovering the sequence C_i from it.
- Use the previous parts to give a procedure for generating an infinite sequence of unfair coin tosses C_i from an infinite sequence of fair coin tosses B_i .

- Suppose that we decide to encode an infinite binary sequence of i.i.d. unfair coin tosses C_i into an infinite binary sequence of fair coin tosses B_i . Use the previous parts to give a way of doing it. (reverse the procedure from the previous part.) This is the infinite arithmetic encoding of the source C_i . The previous part gives the decoding algorithm.
- Extend the above to the case where you want to uniquely encode an i.i.d. sequence drawn from an arbitrary known Discrete Memoryless Source (i.e. move from a binary alphabet for C_i to an arbitrary finite alphabet) into a semi-infinite sequence of fair coin tosses and then be able to uniquely recover the original realization of the DMS from the realization of the fair coin tosses. (Once again, feel free to ignore any problems that come from ties.)
- Extend the above to the case where you want to uniquely encode the sample path coming from a known finite-alphabet Markov chain.

Problem 2.2 *Arithmetic Coding: Finite case and entropy.*

The previous questions probably resulted in answers that involved going from one infinite sequence to another by way of an infinite-precision real number. Practically speaking, this would involve “slurping” up the entire semi-infinite sequence from the source realization, and only then being able to emit outputs. (an infinite-delay system.)

Suppose that you wanted to avoid having to pass through infinity. This is important not only for practical reasons, but for conceptual ones as well. It is hard to understand in what sense one infinite binary string is a “compressed” version of another since they are both infinitely long. Compression is a concept that makes sense with finite numbers and in the limit.

- First consider the following finite-version of the problem: Suppose that you just have N i.i.d. source symbols coming from an unfair Bernoulli source. Apply the recursive procedure to generate the Y_i . At the end, you will have a pair of random variables Y_N and R_N . Give an interpretation to Y_N and R_N in terms of probability and distribution functions.
- Suppose that we decide to encode c_1^N by the shortest binary string $B_1, B_2, \dots, B_{L(c_1^N)}$ which when interpreted as a binary expansion $Q = \sum_{i=1}^N B_i 2^{-i}$ lies in the interval $[Y_N(c_1^N), Y_N(c_1^N) + R_N(c_1^N)]$. Give a bound on the expected length of the resulting binary string in terms of the entropy.
- Is the fixed→variable code given in the previous step uniquely decodeable? Why or why not?
- Suppose instead that we decide to encode c_1^N by the shortest binary string $B_1, B_2, \dots, B_{L(c_1^N)}$ which when interpreted as a binary expansion $Q = \sum_{i=1}^N B_i 2^{-i}$ has $[Q, Q + 2^{-L(c_1^N)}]$ contained entirely within $[Y_N(c_1^N), Y_N(c_1^N) + R_N(c_1^N)]$. Give a bound on the expected length of the resulting binary string in terms of the entropy.
- Is the fixed→variable code given in the previous step uniquely decodeable? Why or why not?

- *Come up with a variation on the arithmetic coding ideas above to give a variable→fixed length code. Argue why it is reasonably efficient (in terms of approaching entropy).*
- *Use the ideas from above to give a source encoder and decoder that each maintain memory and work in a mostly on-line fashion. By this I mean that the encoder spits out encoded digits (fair coins) as it consumes source symbols, though it might consume a variable number of source symbols before each outputted digit. Similarly, the decoder should spit out decoded symbols as it receives digits, though once again it might consume a variable number of digits for each symbol.*

Problem 2.3 *Run-length encoding*

Run-length encoding is a popular variable→variable rate coding strategy that is often used for its simplicity. Consider an unfair sequence of i.i.d. coin tosses. Parse the sequence into a sequence of run-lengths which count how often the digit repeats before it changes. Assume that you know the starting digit.

- *Show that the sequence of run-lengths uniquely specifies the original sequence given that we know the starting digit.*
- *Show that the sequence of run-lengths is i.i.d. if the original sequence was i.i.d.*
- *Use a uniquely decodeable code of your choice for the positive integers and use simulations to estimate the compression ratio achieved by run-length encoding for $p = 0.1$. Compare to the entropy bound.*
- *Suppose you cap the run-lengths at 3 and you use the 4-th symbol to encode the run-length of larger than 3. Now, encode the run-lengths using the optimal Huffman code on the 4 symbols. (Consider $p = 0.1$) Compare the expected compression ratio achieved against the compression ratio achieved by the direct Huffman code that considers two digits at a time.*
- *Comment on the usefulness of run-length encoding.*