**CSC 310: Information Theory** **Winter 2026**

# Tutorial 5: Arithmetic Coding, Integer Code, and LZ Compression

*Instructor: Swastik Kopparty* *TA: Ziyang Jin*

**Date: 04 Feb 2026**



## 5.1 Arithmetic Coding

Consider running the arithmetic coding on a distribution $X_1, \ldots, X_n$ on $\Sigma^n$, where $\Sigma = \{0, 1, 2, 3, 4, \ldots, 15\}$. Random variable $X_1$ is uniform on $\Sigma$, and for each $i \in \{1, \ldots, n-1\}$, we have

$$X_{i+1} = \begin{cases} X_i, & \text{with prob. } 1/2; \\ X_i + 1, & \text{with prob. } 1/2. \end{cases}$$

Note that $X_i + 1$ is computed mod 16.

**What is $H(X_1, \ldots, X_n)$?** First, we compute $H(X_1) = 16 \times \frac{1}{16} \times \log 16 = 4$. Observe that this is a Markov source ($X_{i+1}$ only depends on $X_i$). Given $X_i$, random variable $X_{i+1}$ is uniformly distributed among a set of two elements. Therefore, we have $H(X_{i+1}|X_i) = 2 \times \frac{1}{2} \times \log 2 = 1$. Applying the chain rule, we get the joint entropy

$$\begin{aligned} H(X_1, \ldots, H_n) &= H(X_1, \ldots, X_{n-1}) + H(X_n|X_1, \ldots, X_{n-1}) \\ &= H(X_1, \ldots, X_{n-2}) + H(X_{n-1}|X_1, \ldots, X_{n-2}) + H(X_n|X_1, \ldots, X_{n-1}) \\ &= H(X_1) + \sum_{i=1}^{n-1} H(X_{i+1}|X_1, \ldots, X_i) \\ &= H(X_1) + \sum_{i=1}^{n-1} H(X_{i+1}|X_i) && \text{(due to Markov source)} \\ &= 4 + (n-1) \cdot 1 \\ &= n + 3. \end{aligned}$$

**How good was the compression?** For arithmetic coding, the best achievable expected code length is the joint entropy $H(X_1, \ldots, X_n)$, up to a small constant. Therefore, running the arithmetic coding will give us expected compression length $n + 3$ bits. The per-symbol compression rate will be $(n+3)/n = 1 + \frac{3}{n} \to 1$ as $n \to \infty$.

**What happens if $(1/2, 1/2)$ got replaced by $(1/4, 3/4)$? What if it was $(0, 1)$?** In this case $(1/4, 3/4)$, we have the conditional entropy

$$H(X_{i+1}|X_i) = \frac{1}{4} \times \log 4 + \frac{3}{4} \times \log \frac{4}{3} = \frac{1}{2} + \frac{3}{4}(2 - \log 3) = 2 - \frac{3}{4} \log 3 \approx 0.81.$$

As a result, the joint entropy becomes

$$H(X_1, \ldots, H_n) = 4 + (n-1)(2 - \frac{3}{4}\log 3) \approx 4 + 0.81(n-1).$$

This is smaller than $n+3$ we had in the original $(1/2, 1/2)$ setup. The expected length will also be $4 + (n-1)(2 - \frac{3}{4}\log 3)$, which is shorter than the original setup.

If it was $(0, 1)$, then we have the conditional entropy $H(X_{i+1}|X_i) = 0$ as $X_i$ uniquely determines $X_{i+1}$. As a result, the joint entropy becomes $H(X_1, \ldots, H_n) = 4$. The expected compression length is 4 bits. As long as you know the first symbol $c \in \Sigma$, the next symbols will be $(c+1), (c+2), ...$ mod 16.

## 5.2 Quiz Time

We will take a 15-minute quiz.

## 5.3 Integer Code

An integer code is a mapping $C : \mathbb{N} \to \{0,1\}^*$ that maps an integer to a binary string. Integer code is prefix-free, and is decodable without external delimiters. Unlike Huffman codes, integer codes do not depend on a specific probability distribution. The goal of integer code is to represent an integer using a variable-length binary string without needing to know an upper bound in advance, while keeping the code uniquely decodable.

**Example.** Let's encode integer 93 using integer code. First, the binary representation of 93 is `1011101` as $1 + 4 + 8 + 16 + 64 = 93$. This is 7 bits long. So we write `111` as the binary representation of 7 and then insert 0's to `111` every other location. As a result, it becomes `010101`, which will be a representation of 7 in our integer code. Then we append a 1 to indicate the end of the "length indicator". After that, we append `1011101`, which is the binary representation of 93.

Therefore, the final code is `010101||1||1011101` = `01010111011101`. This is in total $6 + 1 + 7 = 14$ bits.

In general, for an integer $n$, encoding it using integer code following the above encoding procedure requires $2\lceil \log (\lceil \log n \rceil) \rceil + 1 + \lceil \log n \rceil$ bits.

## 5.4 Lempel–Ziv Compression

The method of Lempel-Ziv (LZ) compression is to replace a substring with a pointer to an earlier occurrence of the same substring. There are several versions of LZ compression algorithm, and we use the verion in the textbook *Information Theory, Inference, and Learning Algorithms* by David J. Mackay.

Consider running LZ compression to compress a string made of 0's and 1's, each sampled independently and identically with probability of 1 equal to 1/100. Observe the string is mostly 0's and occasionally has 1's. We can think of the string as runs of zeros separated by ones:

$$0^{L_1}10^{L_2}10^{L_3}1\ldots$$

where each $L_i$ has expected length 99. For a string of length $n$, the expected number of 1's is $n/100$. Therefore, since `1` appears not so frequently, we are okay paying 1 bit to encode `1`. If many zeros appear,

it is highly likely to appear in the dictionary built by LZ compression so far, so we can just point to the previous occurrence of zeros.

If, at the $n$th bit, we have enumerated $s(n)$ substrings, then we can give the value of the pointer in $\lceil \log s(n) \rceil$ bits. Observe that using 7 bits, we can index all $2^7 = 128$ entries of the LZ dictionary, which contains entries $\lambda, 0, 00, 000, \ldots, 0 \ldots 0$ where the last one has 127 zeros. The concentration bounds tells us that most $L_i$'s will be falling into interval $[99 - \epsilon \cdot 99, 99 + \epsilon \cdot 99]$ for some constant $\epsilon > 0$. Roughly speaking, we can use 7 bits to represent 99 bits. So it compresses quite a lot. When $n$ goes to infinity, this actually compresses to a per symbol cost of $H_2(1/100)$.