Lecture 5: Sources with Memory and the Lempel-Ziv Algorithm

1 Markov Sources

In previous lectures, we developed the basic coding results for discrete memoryless sources. Many of the results, in particular the Kraft inequality, the entropy bounds on expected length for uniquely decodable codes, and the Huffman algorithm, do not depend on the independence of successive source aymbols.

In this section, we extend these results to sources defined in terms of finite state Markov chains. We use the state of a Markov chain to represent the "memory" of the source, and use the transitions between states to represent the next symbol out of the source. Thus, for example, the state could be the previous symbol from the source, or could be the previous 300 symbols. We will be able to model as much memory as we choose, while still staying in the regime of finite state Markov chains.

Example 1: Consider a binary source with outputs $X_1, X_2, \ldots$. Assume that the output probabilities at time n depend on the outputs at times $n-1$ and $n-2$. These previous two outputs are modeled by a *state* S_{n-1}. In Figure 1, the states are represented as the nodes of the graph representing the Markov chain, and the source outputs as the transitions of the graph. Note, for example, that from the state S_{n-1}=01 (representing X_{n-2}=0, X_{n-1}=1), the output X_n=1 causes a transition to S_n=11 (representing X_{n-1}=1, X_n=1). For convenience, we view the chain as starting at time 0 in a state S_0 given by some arbitrary pmf.

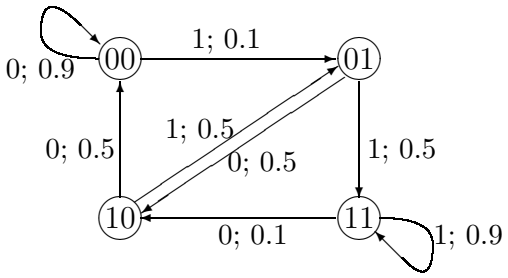


Figure 1: Markov source: Each transition $s' \to s$ is labelled by the corresponding source output and the transition probability $\Pr\{S_n = s | S_{n-1} = s'\}$.

Note that this particular source is characterized by long strings of zeros and long strings of ones interspersed with short transition regions. For example, starting in state 00, a representative output would be

$$000000001011111111111110111111110101000000000\cdots$$

Note that if $s_n = (x_{n-1}x_n)$ then the next state must be either $s_{n+1} = (x_n 0)$ or $s_{n+1} = (x_n 1)$; *i.e.*, each state has only two transitions coming out of it.

We now generalize the example above to an arbitrary discrete Markov source.

Definition: A *finite state Markov chain* is a sequence $S_0, S_1, \ldots$ of discrete chance variables from a finite alphabet, $\mathcal{S}$. There is a pmf $q_0(s), s \in \mathcal{S}$ on S_0, and for all $n \geq 1$ and all $s \in \mathcal{S}, s' \in \mathcal{S}$,

$$\Pr(S_n{=}s|S_{n-1}{=}s') = \Pr(S_n{=}s|S_{n-1}{=}s', \ldots, S_0{=}s'') = Q(s|s') \tag{1}$$

There is a *transition* from s' to s, denoted $s' \to s$, if $Q(s|s') > 0$.

Note that (1) says, first, that the probability of a state depends only on the previous state, and second, that these probabilities do not change with time.

Definition: A *Markov source* is a sequence of discrete chance variables $\mathcal{X}_1, \mathcal{X}_2, \ldots$ with a common alphabet $\mathcal{X}$ which is based on a finite state Markov chain $S_0, S_1, \ldots$. Each transition $(s' \to s)$ in the Markov chain is labelled with a symbol from $\mathcal{X}$; each symbol from $\mathcal{X}$ can appear on at most one outgoing transition from each state.

Note that the state alphabet $\mathcal{S}$ and the source alphabet $\mathcal{X}$ are in general different. Since each source symbol appears on at most one transition from each state, the initial state $S_0{=}s_0$, combined with the source output, $X_1{=}x_1, X_2{=}x_2, \ldots$, uniquely identifies the state sequence, and, of course, the state sequence uniquely specifies the source output sequence. If $x \in \mathcal{X}$ labels the transition $s' \to s$, then the conditional probability of that x is given by $P(x|s') = Q(s|s')$. Thus, for example, in the transition $00 \to 01$ in Figure 1, $Q(01|00) = P(1|00)$.

The reason that we distinguish the Markov chain alphabet from the source output alphabet is that we want the state to represent an arbitrary combination of past events rather than just the previous source output. It is this feature that permits Markov source models to reasonably model both simple and complex forms of memory.

A state s is *accessible* from state s' in a Markov chain if there is a path in the corresponding graph from $s' \to s$, *i.e.*, if $\Pr(S_n{=}s|S_0{=}s') > 0$ for some $n > 0$. The period of a state s is the greatest common divisor of the set of natural numbers n for which $\Pr(S_n{=}s|S_0{=}s) > 0$. A finite state Markov chain is *ergodic* if all states are accessible from all other states and if all states are aperiodic, *i.e.*, have period 1.

We will consider only Markov sources for which the Markov chain is ergodic. An important fact about ergodic Markov chains is that the chain has steady-state probabilities $q(s)$ for all $s \in \mathcal{S}$, given by the unique solution to the linear equations

$$\begin{aligned} q(s) &= \sum_{s' \in \mathcal{S}} q(s')Q(s|s'); \quad s \in \mathcal{S} \qquad (2) \\ \sum_{s \in \mathcal{S}} q(s) &= 1 \end{aligned}$$

These steady-state probabilities are approached asymptotically from any starting state, *i.e.*,

$$\lim_{n \to \infty} \Pr(S_n{=}s|S_0{=}s') = q(s) \qquad \text{for all } s, s' \in \mathcal{S} \tag{3}$$

1.1 Coding for Markov sources

The simplest approach to coding for Markov sources is that of using a separate prefix-free code for each state in the underlying Markov chain. That is, for each $s \in \mathcal{S}$, select a prefix-free code whose lengths $l(x,s)$ are appropriate for the conditional pmf $P(x|s) > 0$. The codeword lengths for the code used in state s must of course satisfy the Kraft inequality $\sum_x 2^{-l(x,s)} \leq 1$. The minimum expected length, $\overline{L}_{min}(s)$ for each such code can be generated by the Huffman algorithm and satisfies

$$H(X|s) \leq \overline{L}_{min}(s) < H(X|s) + 1 \tag{4}$$

where, for each $s \in \mathcal{S}$, $H(X|s) = \sum_{x\in\mathcal{X}} -P(x|s) \log P(x|s)$.

If the initial state S_0 is chosen according to the steady-state pmf $\{q(s); s \in \mathcal{S}\}$, then, from (2), the Markov chain remains in steady-state and the overall expected codeword length is given by

$$H(X|S) \leq \overline{L}_{\min} < H(X|S) + 1, \tag{5}$$

where

$$\overline{L}_{\min} = \sum_{s\in\mathcal{S}} q(s)\overline{L}_{\min}(s) \qquad \text{and} \tag{6}$$

$$H(X|S) = \sum_{s\in\mathcal{S}} q(s)H(X|s) \tag{7}$$

We assume that at time 0, the encoder first transmits the initial state s_0. If M' is the number of elements in the state space, then this can be done with $\lceil \log M' \rceil$ bits, but we ignore this since it is done only at the beginning of transmission and does not effect the long term expected number of bits per source letter. The encoder then successively encodes each source symbol x_n using the code for the state at time $n-1$. The decoder, after decoding the initial state s_0, can decode x_1 using the code based on state s_0. The decoder can then determine the state s_1, and from that can decode x_2 using the code based on s_1. The decoder can continue decoding each source symbol, and we see that the overall code is uniquely decodable. We next must understand the meaning of the conditional entropy in (7).

1.2 Conditional Entropy

It turns out that the conditional entropy $H(X|S)$ plays the same role in coding for Markov sources as the ordinary entropy $H(X)$ plays for the memoryless case. We can rewrite (6) as

$$H(X|S) = \sum_{s\in\mathcal{S}} \sum_{x\in\mathcal{X}} q(s)P(x|s) \log \frac{1}{P(x|s)}$$

We see that this is the expected value of the random variable $\log[1/P(X|S)]$.

An important entropy relation, for any discrete random variables, is

$$H(XS) = H(S) + H(X|S). \tag{8}$$

To see this,

$$\begin{aligned} H(XS) &= \sum_{s,x} q(s)P(x|s) \log \frac{1}{q(s)P(x|s)} \\ &= \sum_{s,x} q(s)P(x|s) \log \frac{1}{q(s)} + \sum_{s,x} q(s)P(x|s) \log \frac{1}{P(x|s)} \\ &= H(S) + H(X|S) \end{aligned}$$

Recall that in exercise 2.5(b), it was shown that

$$H(XS) \leq H(S) + H(X)$$

Comparing this and (8), it follows that

$$H(X|S) \leq H(X). \tag{9}$$

Eqn. (9) is an important inequality in information theory. If we view entropy $H(X)$ as a measure of mean uncertainty, then we should view conditional entropy $H(X|S)$ as a measure of mean uncertainty after the observation of the outcome of S. If X and S are not statistically independent, we would hypothesize that the observation of S reduces the mean uncertainty in X, and this equation indeed verifies this.

Example 1 continued: From Figure 1, it is clear from symmetry that, in steady state, $p_X(0) = p_X(1) = 1/2$. Thus $H(X) = 1$ bit. Conditional on S=00, X is binary with pmf $\{0.1, 0.9\}$, so $H(X|00) = -0.1 \log 0.1 - 0.9 \log 0.9 = 0.47$ bits. Similarly, $H(X|11) = 0.47$ bits, and $H(X|01) = H(X|10) = 1$ bit. Solving the steady-state equations in (2), we find that $q(00) = q(11) = 5/12$ and $q(01) = q(10) = 1/12$. Thus, the conditional entropy averaged over the states is $H(X|S) = 0.558$ bits.

For this example, it is particularly silly to use a different prefix-free code for the source output for each prior state. The problem is that the source is binary, and thus the prefix-free code will have length 1 for each symbol no matter what the state. As with the memoryless case, however, using fixed-to-variable length codes is a solution to these problems of small alphabet sizes and integer constraints on codeword lengths.

In general, then, we look at the use of fixed-to-variable length codes, using a different such code for each prior state. First, however, we must find how to evaluate $H(X_1 \cdots X_n|S_0)$, assuming that S_0 is chosen according to the steady state pmf $\{q(s)\}$. Carrying out this computation (which we omit), the result is that

$$H(X_1 \cdots X_n|S_0) = nH(X|S) \tag{10}$$

It follows, as in the discussion of fixed-to-variable length coding for memoryless sources, that the minimum expected codeword length per source symbol for n-to-variable-length source coding, with a separate code for each prior state, satisfies

$$H(X|S) \leq \overline{L}_{\min,n} < H(X|S) + 1/n \tag{11}$$

The asymptotic equipartition property also holds for Markov sources. Here, however, there are[1] approximately $2^{nH(X|S)}$ typical strings of length n, each with probability approximately equal to $2^{-nH(X|S)}$. It follows as in the memoryless case that $H(X|S)$ is the minimum possible rate at which source symbols can be encoded subject either to unique decodability or to fixed-to-fixed length encoding with small probability of failure. The arguments are essentially the same as in the memoryless case.

We have not carried out all the details of the Markov source analysis, but there are few new ideas that have not already been discussed in the memoryless source case. This is an interesting situation where many real sources can be reasonably modeled as Markov sources, and very few real sources can be reasonably modeled as memoryless sources. However, the understanding of source coding comes primarily from the study of memoryless sources - Markov sources are a simple extension given some familiarity with Markov chains.

2 Lempel-Ziv universal data compression

The Lempel-Ziv data compression algorithms are source coding algorithms which differ from those that we have previously studied in the following ways:

- They use variable-to-variable-length codes in which both the number of source symbols encoded and the number of encoded bits per codeword are variable. Moreover, the code is time-varying.
- They do not require prior knowledge of the source statistics, yet over time they adapt so that the average codeword length $\overline{L}$ per source letter is minimized. Such an algorithm is called *universal*.
- They have been widely used in practice; although newer schemes improve upon them, they provide a simple approach to understanding universal data compression algorithms.

Lempel and Ziv developed two universal data compression algorithms in 1977-78 that were widely used for many years. The first, LZ77, uses string-matching on a sliding window; the second, LZ78, uses an adaptive dictionary. LZ78 was implemented many years ago in the UNIX `compress` algorithm, and in many other places. Implementations of LZ77 are somewhat more recent (Stac Stacker, Microsoft Windows). LZ77 compresses better, but is more computationally intensive.

In this lecture, we describe the LZ77 algorithm. We will then give a high-level idea of why it works. Finally, we will give an upper bound on its data compression performance for Markov sources, showing that it is effectively optimal.

[1]There are additional details here about whether the typical sequences include the initial state or not, but these differences become unimportant as n becomes large.

2.1 The LZ77 algorithm

The LZ77 algorithm compresses a sequence $\mathbf{x} = x_1, x_2, \ldots$ from some given discrete alphabet $\mathcal{X}$ of size $M = |\mathcal{X}|$. At this point we do not assume any probabilistic model for the source, so $\mathbf{x}$ is simply a sequence of symbols, not a sequence of chance variables. We will denote a subsequence $(x_m, x_{m+1}, \ldots, x_n)$ of $\mathbf{x}$ by $\mathbf{x}_m^n$.

The algorithm keeps the w most recently encoded source symbols in memory. This is called a sliding window of size w. The number w is large, and can be thought of as being in the range of 2^{10} to 2^{17}, say. The parameter w is chosen to be a power of 2. Both complexity and performance increase with w.

Briefly, the algorithm operates as follows. Suppose that at some time the source symbols up to x_P have been encoded. The encoder looks for the longest match, say of length n, between the not-yet-encoded n-string $\mathbf{x}_{P+1}^{P+n}$ and a stored string $\mathbf{x}_{P+1-u}^{P+n-u}$ in the window of length w. The clever algorithmic idea in LZ77 is to encode this string of n symbols simply by encoding the integers n and u; *i.e.*, by pointing to the previous occurrence of this string in the sliding window. If the decoder maintains an identical window, then it can look up the string $\mathbf{x}_{P+1-u}^{P+n-u}$, decode it, and keep up with the encoder.

More precisely, the LZ77 algorithm operates as follows:

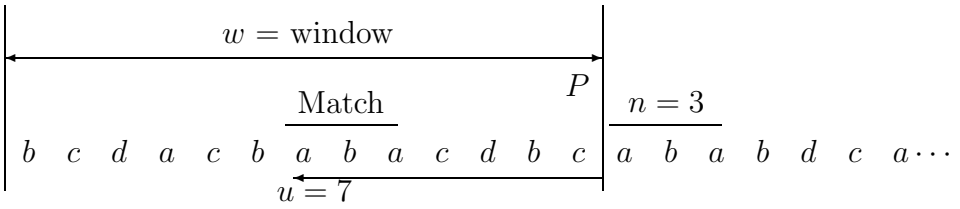
(a) Encode the first w symbols in a fixed-length code without compression, using $\lceil \log M \rceil$ bits per symbol. (Since $w\lceil \log M \rceil$ will be a vanishing fraction of the total number of encoded bits, we don't care how efficiently we encode this preamble.)

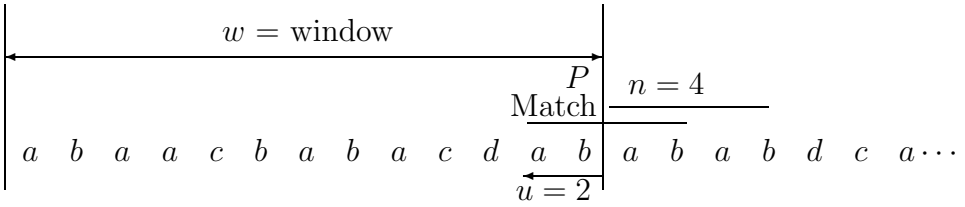
(b) Set the pointer $P = w$. (This indicates that all symbols up to and including x_P have been encoded.)

(c) Find the largest $n \geq 2$ such that $\mathbf{x}_{P+1}^{P+n} = \mathbf{x}_{P+1-u}^{P+n-u}$ for some u in the range $1 \leq u \leq w$. (Find the longest match of a string of $n \geq 2$ not-yet-encoded symbols starting with x_{P+1} with a string of n recently encoded symbols starting u symbols earlier, where $u \leq w$. The string $\mathbf{x}_{P+1}^{P+n}$ will be encoded by encoding the integers n and u.)

If no match exists for $n \geq 2$, then set $n = 1$ and encode a single source symbol x_{P+1} without compression.

Here are two examples. In the first, there is a match of size $n = 3$ with a string starting $u = 7$ symbols prior to the pointer. In the second, there is a match of size $n = 4$ with a string starting $u = 2$ symbols prior to the pointer. (This illustrates the possibility of overlap between the string and its matching string.)





(d) Encode the integer n into a codeword from the so-called unary-binary code. The positive integer n is encoded into the binary representation of n, preceded by a prefix of $\lfloor \log_2 n \rfloor$ zeroes; *i.e.*,

$$1 \to 1, 2 \to 010, 3 \to 011, 4 \to 00100,$$
$$5 \to 00101, 6 \to 00110, 7 \to 00111, 8 \to 0001000, \text{ etc.}$$

Thus the codewords starting with 0^n1 correspond to the 2^n integers in the range $2^n \leq m < 2^{n+1} - 1$. This code is prefix-free (picture the corresponding binary tree). It can be seen that the codeword for integer n has length $2\lfloor \log n \rfloor + 1$; we discuss the significance of this later.

(e) If $n > 1$, encode the positive integer $u \leq w$ using a fixed-length code of length $\log w$ bits. (At this point the decoder knows n, and can simply count back by u in the previously decoded string to find the appropriate n-tuple, even if there is overlap as above.)

If $n = 1$, encode the symbol $x_{P+1} \in \mathcal{X}$ without compression, *i.e.*, use a fixed-length code of length $\lceil \log M \rceil$. (In this case the decoder decodes a single symbol.)

(f) Set the pointer P to $P + n$ and go to step (c). (Iterate.)

3 Why LZ77 works

The motivation behind LZ77 is information-theoretic. The underlying idea is that if the AEP holds for the source, then a sliding window of length w will contain most of the typical strings that are likely to be emitted by the source up to some length n^* that depends on w and the (unknown) source statistics. Therefore, in steady state the encoder will usually be able to encode (at least) n^* source symbols at a time, using a number of bits which is not much larger than $\log w$ (as all parameters become large). The average number of bits per source symbol will therefore be $\overline{L} \approx (\log w)/n^*$.

Assume that the source is a Markov source (although, of course, the algorithm just stated is not based on any knowledge of those source statistics). We will then argue that $\overline{L}$ will be close to the conditional entropy $H(X|S)$ of the source. There are two essential parts to the argument. First, as noted earlier, a Markov source satisfies the AEP, and thus typical strings of length n have probability approximately equal to $2^{-nH(X|S)}$. Second, given a

string $\mathbf{x}_{P+1}^{P+n}$, the expected length back in the sequence to the previous appearance of that string can be shown to approximate $1/\Pr(\mathbf{x}_{P+1}^{P+n})$ with equality in the limit $P \to \infty$).

This means that if we choose an n such that $2^{nH(X|S)} << w$, the the typical sequences of length n will typically have many appearances within the window. Conversely, if $2^{nH(X|S)} >> w$, the the typical sequences of length n will only rarely have an appearance within the window. The conclusion is that the typical match will be for some n^* for which

$$2^{n^* H(X|S)} \approx w.$$

Thus

$$n^* \approx \frac{\log w}{H(X|S)}.$$

Each encoder operation therefore encodes a string of about n^* source symbols, and requires about $\log w \approx n^* H(X|S)$ bits to encode the match location u. The number of bits required by the unary-binary code to encode n is logarithmic in n and thus negligible compared to $\log w$, which is roughly linear in n. Note that we cannot optimize the code used to encode n, since the source statistics are not known. However, making it logarithmic in n guarantees that it will be negligible for large enough w. Thus, the algorithm requires roughly $\overline{L} \approx H(X|S)$ bits per source symbol, which, as we have seen, is optimal even if the source statistics are known.

This argument can be made precise, but we will not do so here.

4 Discussion

Let us recapitulate the basic ideas behind the LZ77 algorithm:

(a) Let N_x be the number of occurrences of symbol x in a window of size w. The WLLN asserts that the relative frequency N_x/w of appearances of x in the window will satisfy $N_x/w \approx p_X(x)$ with high probability. Similarly, let $N_{\mathbf{x}^n}$ be the number of occurrences of $\mathbf{x}^n$ which start in the window. The relative frequency $N_{\mathbf{x}^n}/w$ will then satisfy $N_{\mathbf{x}^n}/w \approx p_{\mathbf{X}^n}(\mathbf{x}^n)$ with high probability for very large w. This association of relative frequencies with probabilities is what makes LZ77 a universal algorithm which needs no prior knowledge of source statistics.[2]

(b) Next, as explained in the last section, the probability of a typical source string $\mathbf{x}^n$ for a Markov source is approximately $2^{-nH(X|S)}$. If $w >> 2^{nH(X|S)}$, then, according to the previous item, $N_{\mathbf{x}^n} \approx w p_{\mathbf{X}^n}(\mathbf{x}^n)$ should be large and $\mathbf{x}^n$ should occur in the window with high probability. Alternatively, if $w << 2^{nH(X|S)}$, then $\mathbf{x}^n$ will probably not occur. Consequently the match will usually occur for $n \approx (\log w)/H(X|S)$ as w becomes very large.

[2] As Yogi Berra said, "You can observe a whole lot just by watchin'."

(c) Finally, it takes about $\log w$ bits to point to the best match in the window. The unary-binary code used to encode the length n of the match usually requires a negligible number of bits relative to $\log w$.

Consequently, LZ77 requires about $\log w$ encoded bits for each group of about $(\log w)/H(X|S)$ source symbols, so it nearly achieves the optimal efficiency of $\overline{L} = H(X|S)$ bits/symbol, as w becomes very large.