Northwestern University
Department of Electrical and Computer Engineering

Lecture 7: Supplemental notes on LZ77.

The following are is a brief description of Lemple-Ziv compression adapted from notes by R. Gallager at MIT. These notes consider Markov Sources, but the argument goes through for any source for which the AEP holds.

1 Lempel-Ziv universal data compression

The Lempel-Ziv data compression algorithms are source coding algorithms which differ from those that we have previously studied (i.e. Huffman Codes and Shannon Codes) 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.

The Lempel-Ziv compression algorithms were developed in 1977-78. The first, LZ77, uses string-matching on a sliding window [1]; whereas the second, LZ78, uses an adaptive dictionary [2]. LZ78 was implemented many years ago in the UNIX `compress` algorithm, and in many other places.[1] Implementations of LZ77 are somewhat more recent (pkzip, gzip,Stacker, Microsoft Windows).[2] LZ77 compresses better, but is more computationally intensive.

In this lecture, we describe the LZ77 algorithm.[3] We will then give a high-level idea of why it works. Finally, we give an approximate analysis of its performance for Markov sources, showing that it is effectively optimal.[4] In other words, although this algorithm operates in ignorance of the source statistics, it compresses substantially as well as the best algorithm designed to work with those statistics.

[1] Unix `compress` uses a variation of the LZ78 algorithm, called the Lempel-Ziv-Welch (LZW) algorithm, designed by Jerry Welch in 1984. LZW is also used in several image formats such as Graphics Interchange Format (GIF) and Tag Image File Format (TIFF), as well as a part of the V.32 bis modem compression standard and PostScript Level 2. The patent for LZW compression is held by Unisys, which levies a fee on any application that uses the LZW compression algorithm.

[2] There are also numerous variations of the LZ77 protocol, with names such as LZR, LZSS, etc.

[3] A discussion of LZ78 can be found in Section 12.10 of Cover and Thomas.

[4] A proof of optimality for discrete ergodic sources can be found in [3].

1.1 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 random 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 of a string of n not-yet-encoded symbols $\mathbf{x}_{P+1}^{P+n}$ with a stored string $\mathbf{x}_{P+1-u}^{P+n-u}$ in the stored sequence 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:

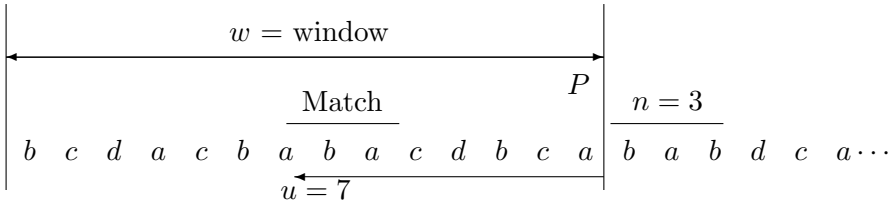
(a) Encode the first w symbols in a fixed-length code without compression, using $\lceil \log M \rceil$ bits per symbol. (Since $w\lceil \log M \rceil$ will be a vanishing fraction of the total number of encoded bits, we don't care how efficiently we encode this preamble.)

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

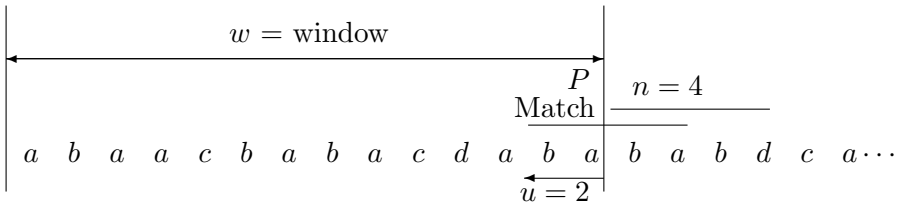
(c) Find the largest $n \geq 2$ such that $\mathbf{x}_{P+1}^{P+n} = \mathbf{x}_{P+1-u}^{P+n-u}$ for some u in the range $1 \leq u \leq w$. (Find the longest match of a string of $n \geq 2$ not-yet-encoded symbols starting with x_{P+1} with a string of n recently encoded symbols starting u symbols earlier,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

(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 using a fixed-length code of length $\lceil \log M \rceil$ bits. (In this case the decoder decodes a single symbol.)

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

2 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 $\overline{L} \approx \frac{2\lfloor \log(n^*) \rfloor + 1}{n^*} + (\log w)/n^* \approx (\log w)/n^*$ (for n^* large enough).

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 $H(X_2|X_1)$ of the source.[5] 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 $2^{-nH(X_2|X_1)}$. Second, given a string $\mathbf{x}_{P+1}^{P+n}$, let $N^w_{\mathbf{x}_{P+1}^{P+n}}$ be the number of occurances of the string in a given window of size w. Then for very large w, $N^w_{\mathbf{x}_{P+1}^{P+n}}/w \approx p_{\mathbf{X}^n}(\mathbf{x}_{P+1}^{P+n})$ (with high probability for large w). Thus,

[5] More generally the same argument applies for any source that has an entropy rate and satisfies the AEP (e.g. a stationary ergodic source)

we can approximate the number of occurances of a given string by $N^{w}_{\mathbf{x}_{P+1}^{P+n}} \approx w p_{\mathbf{X}^n}(\mathbf{x}_{P+1}^{P+n})$. This means that if we choose an n such that $2^{nH(X_2|X_1)} << w$, then the typical sequences of length n will occur in the window with high probability. Alternatively, if $2^{nH(X_2|X_1)} >> w$, then a typical sequences of length n will probably not occur. Consequently, we can conclude that a match usually occur for some n^* for which

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

or equivalently,

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

assuming that w is very large.

Each encoder operation therefore encodes a string of about n^* source symbols, and requires about $\log w \approx n^* H(X_2|X_1)$ 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 $\overline{L} \approx H(X_2|X_1)$ bits per source symbol, which, as we have seen, is the same value as when the source statistics are known and an optimal fixed-to variable uniquely decodable code is used.

The above argument is very imprecise; it is made precise in [3], but that will not be presented here. The imprecision above involves more than simply ignoring the approximation factors in the AEP. A more conceptual issue is that the strings of source symbols that must be encoded are somewhat special since they start at the end of the previous matches.

On the implementation side, one of the key issues is sorting out how to efficiently search for the longest match.

References

[1] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. on Information Theory*, pp. 337-343, May 1977.

[2] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. on Information Theory*, pp. 530-536, Sept. 1978.

[3] A. Wyner and J. Ziv, "The sliding window Lempel-Zi algorithm is asymptotically optimal," Proc. IEEE, pp. 872-877, June 1994.