ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

School of Computer and Communication Sciences

Handout 14 Information Theory and Coding
Solutions to Homework 6 Oct. 28, 2019

Problem 1. Since L is linear, we know that

$$L(\lambda x) = \lambda L(x)$$

for any $\lambda \in \mathbb{R}$. Similarly, g is concave so it must satisfy the following by definition.

$$g(\lambda x_1 + (1-\lambda)x_2) \geq \lambda g(x_1) + (1-\lambda)g(x_2)$$

for any $\lambda \in [0,1]$. Combining these two statements, the following steps show that f is concave.

$$\begin{aligned} f(\lambda x_1 + (1-\lambda)x_2) &= g(L(\lambda x_1 + (1-\lambda)x_2)) \\ &= g(\lambda L(x_1) + (1-\lambda)L(x_2)) \qquad (1) \\ &\geq \lambda g(L(x_1)) + (1-\lambda)g(L(x_2)) \qquad (2) \\ &= \lambda f(x_1) + (1-\lambda)f(x_2) \end{aligned}$$

where (1) uses the linearity property of L and (2) uses the concavity property of g.

Problem 2.

(a) Let $s(m) = 0 + 1 + \cdots + (m-1) = m(m-1)/2$. Suppose we have a string of length $n = s(m)$. Then, we can certainly parse it into m words of lengths 0, 1, ..., $(m-1)$, and since these words have different lengths, we are guaranteed to have a distinct parsing. Since a parsing with the maximal number of distinct words will have at least as many words as this particular parsing, we conclude that whenever $n = m(m-1)/2$, $c \geq m$ (and for $n > m(m-1)/2$ we can parse the first $m(m-1)/2$ letters to m, as we just described, and append the remaining letters to the last word to have a parsing into m distinct words).

(b) An all zero string of length $s(m)$ can be parsed into at most m words: in this case distinct words must have distinct lengths and the bound is met with equality.

(c) Now, given n, we can find m such that $s(m-1) \leq n < s(m)$. A string with n letters can be parsed into $m-1$ distinct words by parsing its initial segment of $s(m-1)$ letters with the above procedure, and concatenating the leftover letters to the last word. Thus, if a string can be parsed into $m-1$ distinct words, then $n < s(m)$, and in particular, $n < s(c+1) = c(c+1)/2$. From above, it is clear that no sequence will meet the bound with equality.

Problem 3.

(a) We have $\rho(X_1^\infty) = 0$. We show this by showing that $\rho(X_1^\infty) \leq \delta$ for any $\delta > 0$. To see the last statement, build an invertible FSM which "recognizes" a string of type "ab...ab" for a particular even length, call it L, and outputs lets say "0" at the end of this string and returns to the starting state. Hence this machine will output an infinite string of "0" when the input is X_1^∞. From each state (including the starting state) of the chain which recognizes the special string make an edge back to the starting state in the case the next input is not the correct one. The output for each such edge is $1 + \lceil \log L \rceil$ bits long, the first bit is 1 to indicate that it is not the special path and on the next $\lceil \log L \rceil$ bits we give the index of the state (in binary representation) from which the return edge is drawn. This machine is clearly lossless and has a compressibility of $1/L$ for the desired sequence.

(b) A machine as described above will have $\rho_M(X_1^\infty) = 1/4$. In fact, one cannot do better than this. Consider a cycle, when from a given state we get back to the same state. During such a cycle we have to output at least one symbol, because the machine has to be information lossless. In an L state machine we eventually create such a cycle within at most L steps. This means that we output at least one symbol for every L input symbols, so $\rho_M(X_1^\infty) \geq 1/L$.

(c) We have $\rho_{\text{LZ}} = 0$ since compressibility is non-negative and we know that the compressibility of LZ is at least as good as that of any FSM, i.e., we know that $\rho_{\text{LZ}}(X_1^\infty) \leq \rho(X_1^\infty)$.

(d) The dictionary increases by 1 every time and has size 2 in the beginning. Hence, if we look at lets say c steps of the algorithm then we need in total

$$\sum_{i=1}^{c} \lceil \log(1+i) \rceil \leq c \log(2(c+1))$$

bits to describe the output.

What are the words which we are using. Note that the parsing is *a*, *b*, *ab*, *aba*, *ba*, *bab*,... Note that in average at most every second step the length of the used dictionary word increases by 1, i.e., we have a linear increase in the used dictionary words. Therefore, if we compute the total length which we have parsed after c steps, this length increases like the square of c.

It follows that the ratio of the total number of bits used divided by the total length described behaves like $1/c$, i.e., it tends to 0.

Problem 4.

(a) Since the channel is symmetric, the input distribution that maximizes the mutual information is the uniform one. Therefore, $C = 1 + \epsilon \log_2(\epsilon) + (1-\epsilon)\log_2(\epsilon)$ which is equal to 0 when $\epsilon = \frac{1}{2}$.

(b) We have

- $I(X^n; Y^n) = I(X_2^n; Y^{n-1}) + I(X_2^n; Y_n | Y^{n-1}) + I(X_1; Y^n | X_2^n)$.
- $X_2^n = Y^{n-1}$ and $Y_1, \ldots, Y_n$ are i.i.d. and uniform in $\{0,1\}$, so $I(X_2^n; Y^{n-1}) = H(Y^{n-1}) = n-1$.
- Y_n is independent of (X_2^n, Y^{n-1}), so $I(X_2^n; Y_n | Y^{n-1}) = 0$.
- X_1 is independent of (Y^n, X_2^n), so $I(X_1; Y^n | X_2^n) = 0$.

Therefore, $I(X^n; Y^n) = n-1$.

(c) W is independent of Y^n, so $I(W; Y^n) = 0 = nC$.

Problem 5.

(a) Even though the decoder can not find out the value of u_{i+l+1} from the description of the word w, it can determine u_{i+l+1} once it receives the index of the next word w': either

  (i) w' is not the newly added word, in which case u_{i+1+l} is the first letter of w', or

  (ii) w' is the newly added word, unknown to the decoder. But w is a prefix of w', so u_{i+l+1} is the first letter of w.

(b) As the algorithm always looks for the longest word w in the dictionary that matches the start of the as-yet-unprocessed segment of the input, wu_{i+l+1} is not in the dictionary before its addition to the dictionary in step 4. Thus the words added to the dictionary are distinct. For each occurrence of a word w in the parsing a word of the form wu with $u \in \mathcal{U}$ is added to the dictionary. Since these are distinct, w cannot appear more than $|\mathcal{U}|$ times in the parsing.

(c) There are $|\mathcal{U}|^i$ words of length i, and by (b), each can appear at most $\mathcal{U}$ times in the list $w_1, \dots, w_m$. As the algorithm never parses the null string, at most $F(k) = |\mathcal{U}| \sum_{i=1}^{k-1} |\mathcal{U}|^i$ words in the list are of length $k-1$ or less, and each of the remaining words in the list has length k or more. Thus $n \geq k[m - F(k)]$.

(d) By (c) $\frac{m(u^n)}{n} \leq \frac{1}{k} + \frac{F(k)}{n}$. Thus, $\limsup_{n\to\infty} m(u^n)/n \leq 1/k$. As k is arbitrary, the conclusion follows.

(e) The dictionary size s is initially $|\mathcal{U}|$, and increases by 1 at each parsing. Thus, after the m'th parse the algorithm has output m binary strings, each of length at most $\lceil \log_2(|\mathcal{U}| + m) \rceil$.

(f) Let $u^n = w_1 \dots w_m$ be the parsing of the input by the algorithm. Let z_i be the state the IL machine is in at just before it reads w_i, and z_{i+1} be the state just after it has read w_i. Let t_i be the binary string output by the IL machine while it reads w_i. No string t can occur in $t_1, \dots, t_m$ more than $k = s^2|\mathcal{U}|$ times. If it did, there will be a state-pair (z, z') that occurs among (z_i, z_{i+1}) more than $|\mathcal{U}|$ times with $t_i = t$. By (b) there will be $w_i \neq w_j$ with $t_i = t_j = t$ and $(z_i, z_{i+1}) = (z_j, z_{j+1})$. But this contradicts the IL property of the machine. Thus the output of the IL machine $t_1 \dots t_m$ is a concatenation of m binary strings with no string occurring more than $k = s^2|\mathcal{U}|$ times, so their total length is at least $L(m, k)$.

This version of Lempel-Ziv is known as the Lempel-Ziv-Welch (LZW) algorithm and is the one commonly implemented. In (f) we have shown that no IL machine with fewer than s states can output fewer than $L(m(u^n), s^2|\mathcal{U}|)$ bits when it processes u^n. In the notes on LZ we had shown that $L(m, k) \geq m \log(m/(8k))$. With $m = m(u^n)$ and $k = s^2|\mathcal{U}|$, by (e), the number of bits per letter LZW emits is at most

$$\frac{m}{n}\lceil \log(|\mathcal{U}| + m) \rceil - \frac{m}{n} \log \frac{m}{8k} \leq \frac{m}{n} \log \frac{16k(|\mathcal{U}| + m)}{m}$$

more than that of an IL machine. As n gets large, the argument of the log approaches $16k$, m/n approaches 0, and so the right hand side above approaches 0. Thus, LZW competes well against the class of finite state IL machines, and we see that it (just as LZ) is an asymptotically optimal method to compress an infinite sequence $u_1 u_2 \dots$.