About Lempel-Ziv Compression Algorithm

Xiao Liang

Computer Science Department
Stony Brook University

Nov. 6th, 2014

Motivation

- Data need to be compressed to save storage space.
- Hoffman Coding is good, but it requires prior knowledge about the distribution of the data. Otherwise:
  - 1st round to record the frequency
  - 2nd round to do the compression Time consuming
- We need an adaptive compression algorithm that does not assume any a priori knowledge of the symbol probabilities; Do the compression in one scan.
- The solution is Lempel-Ziv compression.

About Huffman Coding

Bounds for Huffman Coding

Let L be the expected length of a single letter in Huffman coding, and $H(P)$ is the entropy for the distribution of the letters, then

$$H(P) \leq L \leq H(P) + 1$$

Proof Outlines:
Lower Bound follows from Shannon's Theorem.
Upper Bound:

- Huffman coding is optimal in terms of expected length.
- Kraft inequality

Proofs of the Bounds for Huffman Coding (1/2)

Kcraft Inequality

Let $\ell_1, \ell_2, \ldots, \ell_k \in \mathbb{N}$. Then there exists a prefix binary tree with k leaves at distance (depth) ℓ_i for $i = 1, \ldots, k$ from the root if and only if

$$\sum_{i=1}^{k} 2^{-\ell_i} \leq 1$$

This can be easily proven by Mathematical Induction.

Let $\ell_a = \lceil -\log_2(p_a) \rceil$, since $\sum_{a \in A} 2^{-\ell_a} = \sum_{a \in A} 2^{-\lceil -\log_2(p_a) \rceil} \leq \sum_{a \in A} 2^{\log_2(p_a)} = 1$ according to Kcraft inequality, we know there exists a binary tree with $|A|$ leaves and the corresponding prefix tree has a string of length ℓ_a for $a \in A$.

Proofs of the Bounds for Huffman Coding (2/2)

Since we have argued that Huffman coding is optimal in terms of expected length, it is better than the prefix code tree we considered in last slides corresponding to $\ell_a = \lceil -\log_2(p_a) \rceil$.
Thus:

$$\begin{aligned} L &\le \sum_{a \in A} p_a \ell_a \\ &= \sum_{a \in A} p_a \lceil -\log_2(p_a) \rceil \\ &\le \sum_{a \in A} p_a (1 - \log_2 p_a) \\ &= H(P) + 1 \end{aligned}$$

LZ and Variants

LZ77 and LZ78 are the two lossless data compression algorithms published in papers by Abraham Lempel and Jacob Ziv in 1977 and 1978.

LZ77 Family	LZR	LZSS	LZB	LZH		
LZ78 Family	LZW	LZC	LZT	LZMW	LZJ	LZFG

Table: Variants Based LZ Algorithm

The "zip" and "unzip" use the LZH technique while UNIX's compress methods belong to the LZW and LZC classes.

Agenda

1. Show how the LZ78 algorithm works
2. Analysis of LZ78's performance
3. (If time allows) Most Popular Implementation:

   Lemple-Ziv-Welch

LZ78 Algorithm

- Encoding
  1. new string $\rightarrow$ dictionary (Call it phrase)
  2. encode new strings using phrase in the dictionary, then add it in dictionary as a Phrase, which can be used to express new strings in the future
- Decoding
  Just the reverse of Encoding.

Illustrated by an example taken from Prof. Peter Shor's Lecture notes:
http://www-math.mit.edu/~djk/18.310/Lecture-Notes/LZ-worst-case.pdf

Do it on board.

Performance of LZ78 - Worst Case

Suppose input string (of length n) can be partitioned into $c(n)$ phrases. Then we will have at most

$$c(n)(\log_2 c(n) + log_2\alpha)$$

bits in the encoded data. (Denote $\alpha = |A|$, the size of Alphabet)

We can show that (Do it on the board):

$$c(n) \leq \frac{n}{\log_2 c(n) - 3}$$

Thus

$$\text{worst-case-encoding} \leq n + 4c(n) = n + O(\frac{n}{\log_2 n})$$

This is asymptotically optimal, since there is no way to compress all strings of length n into fewer than n bits. (Why? $\log_2(2^n) = n$)

Performance of LZ78 - i.i.d Results

Assume in a message of legnth n, each letter (from alphabet A) come from i.i.d. multinomial distribution where x_i has probability p_i. Then LZ78 can achieve:

$$n \cdot H(p_1, p_2, ..., p_{|A|}) + O(n)$$

Shannon's Noiseless Coding Theorem

In the i.i.d. setting described above, the best we can achieve is

$$|A| \log_2 n + n \cdot H(P) + c \cdot n \cdot \epsilon$$

(c and ϵ appears for some technical reasons in the derivatoin, for details, refer Shor's)

The first term is neglibile for large n, and we can let ϵ go to zero as $n \to \infty$ to get compression to $n \cdot H(P) + O(n)$ bits.

Performance of LZ78 - i.i.d Derivation [1]

Assume we have a Source. It emits letter x_i (in alphabet A) with probability p_i. Running the Source n times give us a string of length n, whose every letter is independently & identically distributed.

This length n string x can be expressed as:

$$x = x_1 x_2 x_3 ... x_n$$

It is possible that $x_i = x_j$ for $i \neq j$.

Due to the i.i.d. assumption, the probability of seeing this sequence is the products of the probability of each letter:

$$Pr(x) = \prod_{i=1}^{n} p_{x_i}$$

[1] From Prof. Peter Shor's notes

Now assume x is partintioned into $c(x)$ phrases under LZ78:

$$x = x_1x_2 \ldots x_n = y_1y_2y_3 \ldots y_{c(x)}$$

Then:

$$Pr(x) = \prod_{i=1}^{n} p_{x_i} = \prod_{i=1}^{c(x)} Pr(y_i)$$

Now, let's let c_ℓ be the number of phrases y_i of length ℓ. These are (because of the way Lempel-Ziv works) all distinct. Now we prove the following inequality which we will use later.

Ziv's Inequality

$$-\log_2(Pr(x)) \geq \sum_{\ell} c_\ell log_2 c_\ell$$

Proof of Ziv's Inequality

$$Pr(x) = \prod_{\ell} \prod_{|y_i|=\ell} Pr(y_i)$$

For a specific value of ℓ, $\sum_{|y_i|=\ell} Pr(y_i) \leq 1$. Thus:[2]

$$\prod_{|y_i|=\ell} Pr(y_i) \leq (\frac{1}{c_\ell})^{c_\ell}$$

Therefore:

$$\begin{aligned} -\log_2(Pr(x)) &= -\log_2(\prod_{\ell} \prod_{|y_i|=\ell} Pr(y_i)) \\ &= -\sum_{\ell} \log_2(\prod_{|y_i|=\ell} Pr(y_i)) \\ &\geq -\sum_{\ell} \log_2(\frac{1}{c_\ell})^{c_\ell} = \sum_{\ell} c_\ell \log_2 c_\ell \end{aligned}$$

[2] max the products of variables with fixed sum, Lagrange Multiplier method

Since we know that $\sum_\ell c_\ell = c(x)$, we have

$$\sum_\ell c_\ell \log_2 c_\ell = \sum_\ell c_\ell \left(\log_2 c(x) + \log_2 \frac{c_\ell}{c(x)} \right)$$
$$= c(x) \log_2 c(x) + c(x) \sum_\ell \frac{c_\ell}{c(x)} \log_2 \frac{c_\ell}{c(x)}$$

If we regard $\frac{c_\ell}{c(x)}$ as a probability distribution on length ℓ, then

$$-\sum_\ell \frac{c_\ell}{c(x)} \log_2 \frac{c_\ell}{c(x)}$$

is the entropy for this distribution.
Validity:
Sum to 1: $\sum_\ell \frac{c_\ell}{c(x)} = 1$,
Limited Expecctation: $\sum_\ell \ell \frac{c_\ell}{c(x)} = \frac{n}{c_\ell}$

The maximum possible entropy for a probability distribution on positive integers whose expected value is $\frac{n}{c_\ell}$ is $O(\log_2 \frac{n}{c_\ell})$. (WHY?)

" It is not hard to see why this should be true intuitively. If the expected value is $\frac{n}{c(x)}$, then most of the weight must be in the first $O(\frac{n}{c(x)})$ integers, and if a distribution is spread out over a sample space of size $O(\frac{n}{c(x)})$, the entropy is at most $O(\log_2 \frac{n}{c(x)})$."

In summary, we then have

$$-\log_2 Pr(x) \geq \sum_\ell c_\ell \log_2 c_\ell \geq c(x) \log_2 c(x) - c(x) O(\log_2 \frac{n}{c(x)})$$

i.e.

$$c(x) \log_2 c(x) \leq -\log_2 Pr(x) + O(\log_2 \frac{n}{c(x)})$$

Derivation Ended

$$c(x)\log_2 c(x) \le -\log_2 Pr(x) + O(\log_2 \frac{n}{c(x)})$$

- $c(x)\log_2 c(x)$ is approximately the length of encoded string.
- $-\log_2 Pr(x)$ is an approximation of entropy of input string
- $O(\log_2 \frac{n}{c(x)}) = O(\log\log n)$ since $\frac{n}{c(x)} = O(\log n)$

Performance of LZ78 - In Practice

Huffman algorithm (Unix "compact" program)
Lempel-Ziv algorithm (Unix "compress" program)
* Size of compressed file as percentage of the original file

	Adaptive Huffman	Lempel-Ziv
LaTeX file	66%	44%
Speech file	65%	64%
Image file	94%	88%

Table: Huffman v.s. Lempel-Ziv

The large text file described in the Statistical Distributions of English Text (containing the seven classic books with a 27-letter English alphabet) has a compression ratio of 36.3%. This corresponds to a rate of 2.9 bits/character
(Shannon: 2.3 bits/character)

Lempel-Ziv-Welch Compression Algorithm

In Terry Welch's paper "A Technique for High-Performance Data Compression" (1984), he proposed an improved variant of LZ78.

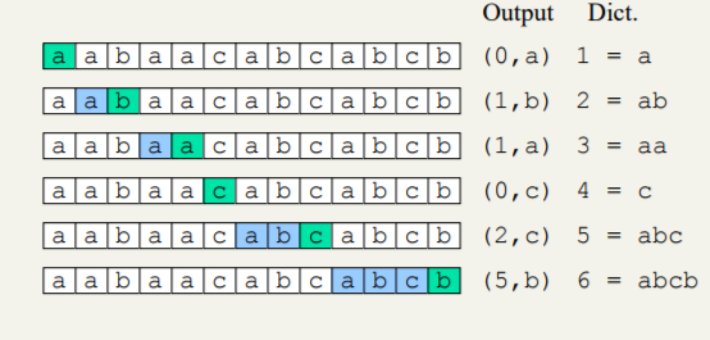

Figure: LZ78 Encoding Example

LZW do not output the letter (the second element in the output vector)

Lempel-Ziv-Welch Compression Algorithm: Encoding

Do it on the Board.

First of all, store the ASCII table in its Dictionary

Figure: LZW Encoding Example

Lempel-Ziv-Welch Compression Algorithm: Decoding

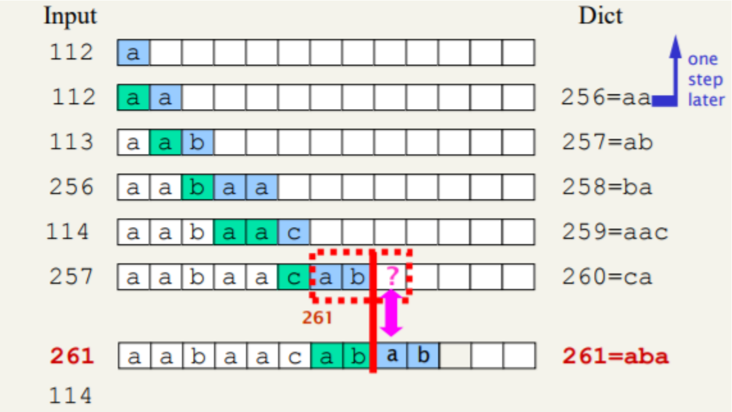

Figure: LZW Encoding Example [3]

[3]Figures are from Prof. Paolo Ferragina, Algoritmiper's notes

Useful Resources

1. Compression Algorithms: Huffman and Lempel-Ziv-Welch (LZW)
   http://web.mit.edu/6.02/www/s2012/handouts/3.pdf
2. Huffman Coding efficiency
   http://math.mit.edu/~shor/18.310/huffman.pdf
3. Lempel-Ziv worst case by Shor
   http://www-math.mit.edu/~djk/18.310/Lecture-Notes/LZ-worst-case.pdf
4. Lempel-Ziv average case by Shor
   http://math.mit.edu/~shor/18.310/lempel_ziv_notes.pdf
5. Shannon's Noisless Coding Theorem
   SBU: https://www.math.stonybrook.edu/~tony/archive/312s09/info6plus.pdf
   MIT: http://math.mit.edu/~shor/18.310/noiseless-coding
6. Lempel-Ziv-Welch Pseudocode
   http://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/
7. Illustration for LZ
   http://www.data-compression.com/lossless.html
8. LZ78 v.s. LZW
   http://pages.di.unipi.it/ferragina/Teach/InformationRetrieval/3-Lecture.pdf