COMP2610/6261 - Information Theory

Lecture 17: Lempel-Ziv Coding and Summary

Mark Reid and Aditya Menon

Research School of Computer Science
The Australian National University

September 30th, 2014

Eliminating Repetition

What is a simple, short binary description of the following string?

$$\underbrace{\texttt{aaaaaaa}}_{7\texttt{a}s}\ \underbrace{\texttt{bbbbbbb}}_{7\texttt{b}s}\ \underbrace{\texttt{aaaaaaa}}_{7\texttt{a}s}\ \underbrace{\texttt{bbbbbbb}}_{7\texttt{b}s}$$

A simple symbol code for $\{\texttt{a}, \texttt{b}\}$, $C = \{0, 1\}$, uses 28 bits

Eliminating Repetition

What is a simple, short binary description of the following string?

$$\underbrace{\texttt{aaaaaaa}}_{7\texttt{a}s}\ \underbrace{\texttt{bbbbbbb}}_{7\texttt{b}s}\ \underbrace{\texttt{aaaaaaa}}_{7\texttt{a}s}\ \underbrace{\texttt{bbbbbbb}}_{7\texttt{b}s}$$

A simple symbol code for $\{\texttt{a},\texttt{b}\}$, $C = \{0,1\}$, uses 28 bits

Run-length coding using (count, symbol) saves 12 bits:

$$\underbrace{111}_{7}\underbrace{0}_{\texttt{a}}\ \underbrace{111}_{7}\underbrace{1}_{\texttt{b}}\ \underbrace{111}_{7}\underbrace{0}_{\texttt{a}}\ \underbrace{111}_{7}\underbrace{1}_{\texttt{b}}$$

Eliminating Repetition

What is a simple, short binary description of the following string?

$$\underbrace{\texttt{aaaaaaa}}_{7\texttt{a}s}\ \underbrace{\texttt{bbbbbbb}}_{7\texttt{b}s}\ \underbrace{\texttt{aaaaaaa}}_{7\texttt{a}s}\ \underbrace{\texttt{bbbbbbb}}_{7\texttt{b}s}$$

A simple symbol code for $\{\texttt{a}, \texttt{b}\}$, $C = \{0, 1\}$, uses 28 bits

Run-length coding using $(\text{count}, \text{symbol})$ saves 12 bits:

$$\underbrace{111}_{7}\underbrace{0}_{\texttt{a}}\ \underbrace{111}_{7}\underbrace{1}_{\texttt{b}}\ \underbrace{111}_{7}\underbrace{0}_{\texttt{a}}\ \underbrace{111}_{7}\underbrace{1}_{\texttt{b}}$$

- Makes no probabilistic assumptions about source.
- Doesn't always yield shorter strings:
  aa bb a b a $\rightarrow$ 10 0 10 1 01 0 01 1 01 0 (7 to 15 bits)
- Misses other structure: "2 repetitions of (7 as and 7 bs)"

Looking for Repetition

Consider a sequence that starts

$$\texttt{abbababbababbab}\ldots$$

We can describe each new part in terms of what we have seen so far:

Looking for Repetition

Consider a sequence that starts

$$\texttt{abbababbababbab}\ldots$$

We can describe each new part in terms of what we have seen so far:

1. A new a

Looking for Repetition

Consider a sequence that starts

$$\texttt{abbababbababbab}\ldots$$

We can describe each new part in terms of what we have seen so far:

1. A new `a`
2. A new `b`

Looking for Repetition

Consider a sequence that starts

`abbababbababbab...`

We can describe each new part in terms of what we have seen so far:

1. A new `a`
2. A new `b`
3. The same 1 symbol as 1 symbol ago

Looking for Repetition

Consider a sequence that starts

$$\underline{\mathtt{ab}}\mathtt{b}\color{red}{\mathtt{ab}}\color{black}{\mathtt{abbababbab}\ldots}$$

We can describe each new part in terms of what we have seen so far:

1. A new a
2. A new b
3. The same 1 symbol as 1 symbol ago
4. The same 2 symbols as 3 symbols ago

Looking for Repetition

Consider a sequence that starts

$$\underline{\texttt{abbab}}\texttt{abbababbab}\ldots$$

We can describe each new part in terms of what we have seen so far:

1. A new a
2. A new b
3. The same 1 symbol as 1 symbol ago
4. The same 2 symbols as 3 symbols ago
5. The same 10 symbols as 5 symbols ago

Looking for Repetition

Consider a sequence that starts

$$\texttt{abbababbababbab}\ldots$$

We can describe each new part in terms of what we have seen so far:

1. A new a $(0, \texttt{a})$
2. A new b $(0, \texttt{b})$
3. The same 1 symbol as 1 symbol ago $(1, 1, 1)$
4. The same 2 symbols as 3 symbols ago $(1, 3, 2)$
5. The same 10 symbols as 5 symbols ago $(1, 5, 10)$

00 01 10010001 10110010 11011001 ...

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
    1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
    1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
    2. Let $t \leftarrow$ last W symbols of $s\,x_n$

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. Let $t \leftarrow$ last W symbols of $s\,x_n$
   3. If t does not appear in $x_{n-W} \ldots x_{n-1}$

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. Let $t \leftarrow$ last W symbols of $s\,x_n$
   3. If t does not appear in $x_{n-W} \ldots x_{n-1}$
      1. If $s = \epsilon$ then output $(0, x_n)$ and continue; otherwise

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. Let $t \leftarrow$ last W symbols of $s\,x_n$
   3. If t does not appear in $x_{n-W} \ldots x_{n-1}$
      1. If $s = \epsilon$ then output $(0, x_n)$ and continue; otherwise
      2. Find smallest $0 \leq i < W$ such that $t = x_{n-i} \ldots x_{n-i+|t|-1} x_n$

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. Let $t \leftarrow$ last W symbols of $s\,x_n$
   3. If t does not appear in $x_{n-W} \ldots x_{n-1}$
      1. If $s = \epsilon$ then output $(0, x_n)$ and continue; otherwise
      2. Find smallest $0 \leq i < W$ such that $t = x_{n-i} \ldots x_{n-i+|t|-1} x_n$
      3. Output $(1, i, |s|)$

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. Let $t \leftarrow$ last W symbols of $s\,x_n$
   3. If t does not appear in $x_{n-W} \ldots x_{n-1}$
      1. If $s = \epsilon$ then output $(0, x_n)$ and continue; otherwise
      2. Find smallest $0 \leq i < W$ such that $t = x_{n-i} \ldots x_{n-i+|t|-1} x_n$
      3. Output $(1, i, |s|)$
   4. Else $s \leftarrow s\,x_n$

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
    1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
    2. Let $t \leftarrow$ last W symbols of $s\,x_n$
    3. If t does not appear in $x_{n-W} \ldots x_{n-1}$
        1. If $s = \epsilon$ then output $(0, x_n)$ and continue; otherwise
        2. Find smallest $0 \leq i < W$ such that $t = x_{n-i} \ldots x_{n-i+|t|-1} x_n$
        3. Output $(1, i, |s|)$
    4. Else $s \leftarrow s\,x_n$

Lempel-Ziv: Sliding Window (LZ77)

LZ77(Sequence $x_1 x_2 \ldots$, Window size $W > 0$)

1. Initialise $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. Let $t \leftarrow$ last W symbols of $s\, x_n$
   3. If t does not appear in $x_{n-W} \ldots x_{n-1}$
      1. If $s = \epsilon$ then output $(0, x_n)$ and continue; otherwise
      2. Find smallest $0 \leq i < W$ such that $t = x_{n-i} \ldots x_{n-i+|t|-1} x_n$
      3. Output $(1, i, |s|)$
   4. Else $s \leftarrow s\, x_n$

Notes:

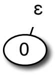
- The output is converted to binary. i is represented with $\lceil \log_2 W \rceil$ bits.
- The size output $|s|$ can be larger than W.
- Not very effective compression for short input sequences.
- Run-length encoding is essentially LZ77 with $W = 1$.

Extending Substrings

Consider the same sequence as before

abbababbababbab . . .

Scan sequence and record each previously unseen string:



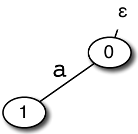
Extending Substrings

Consider the same sequence as before

$$\texttt{abbababbababbab}\ldots$$

Scan sequence and record each previously unseen string:

1. a $\quad\quad\quad\quad$ $(0, \texttt{a})$



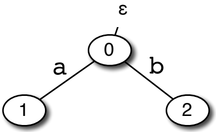
Extending Substrings

Consider the same sequence as before

$$\texttt{abbababbababbab}\ldots$$

Scan sequence and record each previously unseen string:

1. a $(0, \texttt{a})$
2. b $(0, \texttt{b})$



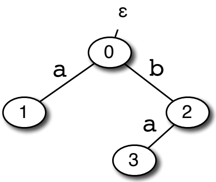
Extending Substrings

Consider the same sequence as before

$$\texttt{ab}\color{red}{\texttt{ba}}\texttt{babbababbab}\ldots$$

Scan sequence and record each previously unseen string:

1. a $(0, \texttt{a})$
2. b $(0, \texttt{b})$
3. ba $(2, \texttt{a})$



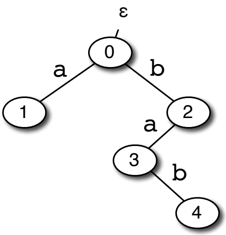
Extending Substrings

Consider the same sequence as before

$$\texttt{abbababbababbbab}\ldots$$

Scan sequence and record each previously unseen string:

1. a $(0, \texttt{a})$
2. b $(0, \texttt{b})$
3. ba $(2, \texttt{a})$
4. bab $(3, \texttt{b})$



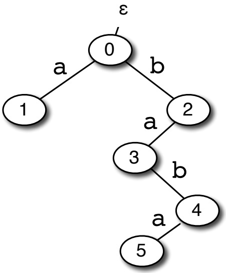
Extending Substrings

Consider the same sequence as before

$$\texttt{abbabab}\textcolor{red}{\texttt{baba}}\texttt{bbab}\ldots$$

Scan sequence and record each previously unseen string:

1. a $(0, \mathtt{a})$
2. b $(0, \mathtt{b})$
3. ba $(2, \mathtt{a})$
4. bab $(3, \mathtt{b})$
5. baba $(4, \mathtt{a})$



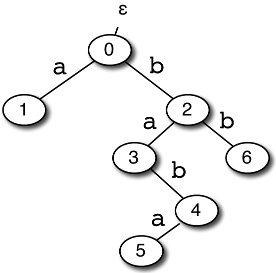
Extending Substrings

Consider the same sequence as before

$$\texttt{abbababbababbab}\ldots$$

Scan sequence and record each previously unseen string:

1. a $(0, \texttt{a})$
2. b $(0, \texttt{b})$
3. ba $(2, \texttt{a})$
4. bab $(3, \texttt{b})$
5. baba $(4, \texttt{a})$
6. bb $(2, \texttt{b})$



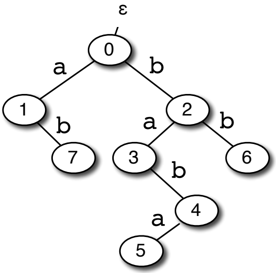
Extending Substrings

Consider the same sequence as before

$$\texttt{abbababbababbab}\ldots$$

Scan sequence and record each previously unseen string:

1. a — $(0, \texttt{a})$
2. b — $(0, \texttt{b})$
3. ba — $(2, \texttt{a})$
4. bab — $(3, \texttt{b})$
5. baba — $(4, \texttt{a})$
6. bb — $(2, \texttt{b})$
7. ab — $(1, \texttt{b})$



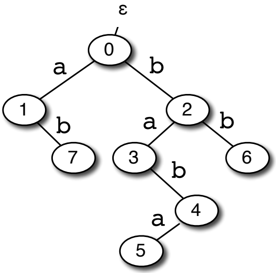
Extending Substrings

Consider the same sequence as before

$$\texttt{abbababbababbab}\ldots$$

Scan sequence and record each previously unseen string:

1. a $(0, \mathtt{a})$
2. b $(0, \mathtt{b})$
3. ba $(2, \mathtt{a})$
4. bab $(3, \mathtt{b})$
5. baba $(4, \mathtt{a})$
6. bb $(2, \mathtt{b})$
7. ab $(1, \mathtt{b})$



$$0\,0\;\;0\,1\;\;10\,0\;\;11\,1\;\;100\,0\;\;010\,1\;\;001\,1\ldots$$

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$
2. While input sequence has more symbols:

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. If $s\, x_n$ is not in dictionary then

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. If $s\,x_n$ is not in dictionary then
      1. Find index i such that $D[i] = s$ and output (i, x_n)

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. If $s\, x_n$ is not in dictionary then
      1. Find index i such that $D[i] = s$ and output (i, x_n)
      2. Update $D[|D|] \leftarrow s\, x_n$ and reset $s \leftarrow \epsilon$

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. If $s\,x_n$ is not in dictionary then
      1. Find index i such that $D[i] = s$ and output (i, x_n)
      2. Update $D[|D|] \leftarrow s\,x_n$ and reset $s \leftarrow \epsilon$
   3. Else $s \leftarrow s\,x_n$

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. If $s\,x_n$ is not in dictionary then
      1. Find index i such that $D[i] = s$ and output (i, x_n)
      2. Update $D[|D|] \leftarrow s\,x_n$ and reset $s \leftarrow \epsilon$
   3. Else $s \leftarrow s\,x_n$

Lempel-Ziv: Tree-Structured (LZ78)

See MacKay §6.4

LZ78(Sequence $x_1 x_2 \ldots$)

1. Initialise empty dictionary $D \leftarrow \{\}$ and $s \leftarrow \epsilon$
2. While input sequence has more symbols:
   1. $x_n \leftarrow$ next symbol from sequence (n is total symbols read)
   2. If $s\,x_n$ is not in dictionary then
      1. Find index i such that $D[i] = s$ and output (i, x_n)
      2. Update $D[|D|] \leftarrow s\,x_n$ and reset $s \leftarrow \epsilon$
   3. Else $s \leftarrow s\,x_n$

Notes:

- Only $\lceil \log_2 n \rceil$ bits of i need to be output at step n
- $|D|$ is the number of entries in the dictionary
- Basic algorithm has several inefficiencies (e.g., if aa and ab in dictionary, then a is not needed)
- Decoding via "identical twin"

Theory

- Both LZ77 and LZ78 are *optimal* in the sense that the expected bits per symbol from some source X converges to $H(X)$ as $N \to \infty$
- Proofs are involved and not covered in this course (See Cover & Thomas §13.5)

Lempel-Ziv: Theory and Practice

Theory

- Both LZ77 and LZ78 are *optimal* in the sense that the expected bits per symbol from some source X converges to $H(X)$ as $N \to \infty$
- Proofs are involved and not covered in this course (See Cover & Thomas §13.5)

Practice

- LZ77 forms the basis of `gzip`, WinZip, the PNG image format, as well as PDF and HTTP compression.
- Variants of LZ78 (a.k.a. LZW) are used for the GIF image format, UNIX `compress`, and early modem protocols.
- Run-Length Encoding (RLE), along with the *Burrows-Wheeler Transform* (BWT) and Huffman coding are used in the `bzip2` compressor

Summary and Reading

Summary:

- Run-length Encoding
- Lempel-Ziv Coding
  - Sliding Window (LZ77)
  - Tree-Structured (LZ78)

Reading

- MacKay §6.4
- Cover & Thomas §13.4

Compression: Review

Block Coding (Uniform Length)

- Lossy compression
- Typical Sets
- Source Coding Theorem

Compression: Review

Block Coding (Uniform Length)

- Lossy compression
- Typical Sets
- Source Coding Theorem

Source Coding Theorem for Block Codes (Lecture 11)

For all $\delta \in (0, 1)$ and $\epsilon > 0$ there is an N_0 such that for all $N > N_0$

$$\left| \frac{1}{N} H_\delta \left(X^N \right) - H(X) \right| < \epsilon$$

Compression: Review

Block Coding (Uniform Length)

- Lossy compression
- Typical Sets
- Source Coding Theorem

Symbol Coding (Variable Length)

- Kraft inequalities: Unique decodability limits compression
- Source Coding Theorem: Witnessed by Shannon Codes
- Huffman Coding: Optimal Symbol Coding
- Shannon-Fano-Elias Coding: Codes from Intervals

Compression: Review

Block Coding (Uniform Length)

- Lossy compression
- Typical Sets
- Source Coding Theorem

Symbol Coding (Variable Length)

- Kraft inequalities: Unique decodability limits compression
- Source Coding Theorem: Witnessed by Shannon Codes
- Huffman Coding: Optimal Symbol Coding
- Shannon-Fano-Elias Coding: Codes from Intervals

Kraft Inequality (Lecture 13)

The code lengths $\{\ell_1, \ldots, \ell_I\}$ for an alphabet of I symbols are for a uniquely decodable code if and only if $\sum_i 2^{-\ell_i} \leq 1$

Compression: Review

Block Coding (Uniform Length)

- Lossy compression
- Typical Sets
- Source Coding Theorem

Symbol Coding (Variable Length)

- Kraft inequalities: Unique decodability limits compression
- Source Coding Theorem: Witnessed by Shannon Codes
- Huffman Coding: Optimal Symbol Coding
- Shannon-Fano-Elias Coding: Codes from Intervals

Source Coding Theorem for Symbol Codes (Lecture 14)

For any ensemble X there exists a code C (the *Shannon Code*) such that

$$H(X) \leq L(C, X) \leq H(X) + 1$$

Compression: Review

Block Coding (Uniform Length)

- Lossy compression
- Typical Sets
- Source Coding Theorem

Symbol Coding (Variable Length)

- Kraft inequalities: Unique decodability limits compression
- Source Coding Theorem: Witnessed by Shannon Codes
- Huffman Coding: Optimal Symbol Coding
- Shannon-Fano-Elias Coding: Codes from Intervals

Stream Coding (Dynamic Codes)

- Arithmetic Coding: Probabilistic Models – e.g., Dirichlet
- Lempel-Ziv Coding: Model-free

The Limits of Compression

In each of the types of compression schemes the goal is to compress down to the entropy of the source.

Q: Why can't I repeatedly compress a message to make it smaller?

The Limits of Compression

In each of the types of compression schemes the goal is to compress down to the entropy of the source.

Q: Why can't I repeatedly compress a message to make it smaller?

In an *optimally* compressed message there is no redundancy — **every bit counts**! $\implies$ A single flipped bit can have a large effect on decompressed message.

The Limits of Compression

In each of the types of compression schemes the goal is to compress down to the entropy of the source.

Q: Why can't I repeatedly compress a message to make it smaller?

In an *optimally* compressed message there is no redundancy — **every bit counts**! $\Longrightarrow$ A single flipped bit can have a large effect on decompressed message.

Next Time:

Error Correction – Putting the redundancy back in!