

# Multimedia Communications

## Context-based Compression



---

# Context-based Coding

- If in a sequence to be encoded, certain symbols occur with much higher probability than others, more compression can be achieved
- Several ways to achieve this situation:
  - Transform sequence to another one with desired property
  - Use a different probability distribution for each symbol (symbol being encoded has high probability)
- If the transformation or probability distribution is based on the history of the sequence, since history is available to both encoder and decoder, there is no need for transmitting additional information.
- These schemes are called context based coding schemes.

---

# Predictive Coding

- Exp:
  - 1,2,5,7,2,-2,0,-5,-3,-1,1,-2,-7,-4,-2,1,3,4
  - Prediction rule:  $r=x(n)-(2+x(n-1))$
  - Residual sequence: 1,-1,1,0,-7,-4,0,-7,0,0,0,-5,1,0,1,0,-1
- A different kind of prediction is used when encoding nonnumerical data (e.g., text): preceding symbols (context) are used to predict the probability model for the symbol being coded
- Exp: encoding the word *probability*
- If we ignore the first four letters that are coded (prob) the probability of a is 0.06.
- If we use the information that the letter before a is b, this will boost the probability of an a occurring

---

## Prediction with partial match (ppm)

- We would like to use large contexts to determine the probability of the symbol being encoded
- Large context requires estimation and storage of an extremely large number of conditional probabilities
- Instead of estimating these probabilities ahead of time, we can reduce the burden by estimating them as the coding proceeds
- This way we only need to store contexts that have occurred which is a much smaller number than all possible contexts.

---

# Basic algorithm

- The size of the largest context is predetermined
- If the symbol to be encoded has not previously been encountered in this context, an **escape** symbol is encoded and the algorithm attempts to use the next smaller context
- If the symbol has not occurred in this context either, the size of the context is further reduced
- This process continues until either we obtain a context that has previously been encountered with this symbol or we conclude that the symbol has not been encountered previously in any context

---

# Basic algorithm

- The case when the symbol is considered by itself is called zero-order context
- If a symbol is encountered for the very first time, we would use a model in which all symbols have equal probability (called context order  $-1$ )
- Each time a symbol is encountered, the count corresponding to that symbol is updated

---

# Example

- Exp: this\_is\_the\_tithe
- Assumption: the first seven characters have been coded and the various counts are as the following tables
- The word length for arithmetic coding is four

-1 order

Letter	Count	Cum_count
t	1	1
h	1	2
i	1	3
s	1	4
e	1	5
_	1	6

0 order

Letter	Count	Cum_count
t	1	1
h	1	2
i	2	4
s	2	6
_	1	7
<esc>	1	8

---

# Example

first order

Context	Letter	Count	Cum_count
t	h	1	1
	<esc>	1	2
h	i	1	1
	<esc>	1	2
i	s	2	2
	<esc>	1	3
_	i	1	1
	<esc>	1	2
s	_	1	1
	<esc>	1	2

second order

Context	Letter	Count	Cum_count
th	i	1	1
	<esc>	1	2
hi	s	1	1
	<esc>	1	2
is	_	1	1
	<esc>	1	2
s_	i	1	1
	<esc>	1	2
_i	s	1	1
	<esc>	1	2

---

## Example

- The next letter to be coded is \_
- The second order context for \_ is is
- Cum\_count=1 and we have

$$l = 0 + \left\lfloor (15 - 0 + 1) \times \frac{0}{2} \right\rfloor = 0 = 0000$$

$$u = 0 + \left\lfloor (15 - 0 + 1) \times \frac{1}{2} \right\rfloor - 1 = 7 = 0111$$

- As the MSB of l and u are the same, we shift that bit out, shift a 0 into the LSB of l and a 1 into the LSB of u

Transmitted sequence: 0

l=0000

u=1111

---

## Example

- The next letter to be coded is t
- The second order context for t is s\_
- t has not appeared in this context before
- We code a escape symbol:

$$l = 0 + \left\lfloor (15 - 0 + 1) \times \frac{1}{2} \right\rfloor = 8 = 1000$$

$$u = 0 + \left\lfloor (15 - 0 + 1) \times \frac{2}{2} \right\rfloor - 1 = 15 = 1111$$

- As the MSB of l and u are the same, we shift that bit out, shift a 0 into the LSB of l and a 1 into the LSB of u

Transmitted sequence: 01

l=0000

u=1111

- We add an entry in s\_ context for t

---

## Example

- Next we check the first-order context of t which is \_
- t has not happened in this context before
- We code another escape symbol:

$$l = 0 + \left\lfloor (15 - 0 + 1) \times \frac{1}{2} \right\rfloor = 8 = 1000$$

$$u = 0 + \left\lfloor (15 - 0 + 1) \times \frac{2}{2} \right\rfloor - 1 = 15 = 1111$$

- As the MSB of l and u are the same, we shift that bit out, shift a 0 into the LSB of l and a 1 into the LSB of u

Transmitted sequence: 011

l=0000

u=1111

- Add an entry in \_ context for t

---

## Example

- Next we check the zero-order context of  $t$
- The zero order context for  $t$  is in the table

$$l = 0 + \left\lfloor (15 - 0 + 1) \times \frac{0}{8} \right\rfloor = 0 = 0000$$

$$u = 0 + \left\lfloor (15 - 0 + 1) \times \frac{1}{8} \right\rfloor - 1 = 1 = 0001$$

- As the 3 MSB of  $l$  and  $u$  are the same, we shift them out, shift a 0 into the LSBs of  $l$  and a 1 into the LSBs of  $u$

Transmitted sequence: 011000

$l=0000$

$u=1111$

---

# Example

0 order

Letter	Count	Cum_count
t	2	2
h	1	3
i	2	5
s	2	7
_	1	8
<esc>	1	9

first order

Context	Letter	Count	Cum_count
t	h	2	1
	<esc>	1	2
h	i	1	1
	<esc>	1	2
i	s	2	2
	<esc>	1	3
-	i	1	1
_	t	1	2
	<esc>	1	3
s	_	1	1
	<esc>	1	2

---

# Example

second order

Context	Letter	Count	Cum_count
th	i	1	1
	<esc>	1	2
hi	s	1	1
	<esc>	1	2
is	_	2	2
	<esc>	1	3
s_	i	1	1
	t	1	2
	<esc>	1	3
_i	s	1	1
	<esc>	1	2
_t	h	1	1
	<esc>	1	2

---

# Exclusion principle

- The basic idea behind arithmetic coding is the division of the unit interval into subintervals, each of which represent a particular letter
- The smaller the subinterval, the more bits are required to distinguish it from other subintervals
- If we can reduce the number of symbols, the size of subintervals increase, leading to a reduction in the number of bits required for encoding
- Exclusion principle provides this kind of reduction

---

## Exclusion principle

- Exp: Suppose we want to encode a in sequence *proba* and the contexts are as the following tables
- As a does not occur in two letter context, we issue an escape symbol and reduce the size of the context
- Checking the one-letter context, we see that a does occur with a count of 4 out of total of 21
- By sending escape symbol in context of ob, we have already signaled the decoder that the symbol being coded is not any of the letters previously encountered in the context of ob
- Therefore we can increase the size of the subinterval corresponding to a by temporarily removing l and o from the table

---

# Example

Context	Letter	Count	Cum_count
ob	l	10	10
	o	3	13
	<esc>	2	15
<b>b</b>	l	5	5
	o	3	8
	a	4	12
	r	2	14
	e	2	16
	<esc>	5	21

Context	Letter	Count	Cum_count
b	a	4	4
	r	2	6
	e	2	8
	<esc>	3	11

---

# Burrows-Wheeler Transform(BWT)

- BWT also uses context of the symbols being coded for lossless compression
- BWT requires the entire sequence to be coded be available to the encoder before coding takes place

## BWT

- Given a sequence of length  $N$ , we create  $N-1$  sequences each of them a cyclic shift of the original sequence
- These  $N$  sequences are arranged in lexicographic (dictionary) order
- The encoder then transmits the sequence of length  $N$  created by taking the last letter of each sorted cyclically shifted sequence

---

# Burrows-Wheeler Transform(BWT)

- The sequence of last letters (L) and the position of the original sequence in the sorted list are coded and sent to the decoder
- We start with a sequence of length N and end with a representation that contains N+1 elements!
- The sequence has a structure that makes it highly suitable for compression
- A coding method called move-to-front (mtf) is particularly effective in the type of structure exhibited by sequence L

# Exp

- Sequence to encode: this\_is\_the
- 11 characters in the sequence: 11 cyclic permutations of the sequence
- The sequences sorted in lexicographic order are as bellow

0	_is_thetis
1	_thetis_is
2	etis_is_th
3	hetis_is_t
4	his_is_thet
5	is_is_thet
6	is_thetis_
7	s_is_theti
8	s_thetis_i
9	thetis_is_
10	this_is_the

L:sshtth\_ii\_e

Encoding sequence:  
L and index 10

---

# BWT decoder

- The first step is to generate sequence F consisting the first elements of each row
- F is simply the sequence L in lexicographic order
- F: \_\_ehhiisstt
- We use L and F to generate a transformation T that will tell us in which order to read elements of L to generate the original sequence

# BWT decoder

- Using the index (10) we decode the last character of the original sequence to be **e**
- Since the rows in the table are the cyclic shift of each other, the character before e is the last element of row starting with e (which is **h**)
- If we have two identical letters in L, the order in which they occur is the same as the order in which the same letters occur in F

0	-									s
1	-									s
2	e									h
3	h									t
4	h									t
5	i									h
6	i									-
7	s									i
8	s									i
9	t									-
10	t									e

---

# Move-to-Front Coding

- A coding scheme that takes advantage of long runs of identical symbols is the move-to-front (mtf) coding
- In this coding scheme, we start with some initial listing of the source alphabet
- The symbol at the top of the list is assigned number 0, the next one is assigned number one and so on.
- The first time a particular symbol occurs, the number corresponding to its place in the list is transmitted and the symbol is moved to top of the list
- If we have a run of this symbol, we transmit a sequence of 0s

---

# Exp

- $L = \text{sshtth\_ii\_e}$
- $A = \{\_, e, h, i, s, t\}$

0	1	2	3	4	5
_	e	h	i	s	t

Encoding sequence: 4

0	1	2	3	4	5
s	_	e	h	i	t

Encoding sequence: 403

0	1	2	3	4	5
h	s	-	e	i	t

Encoding sequence: 40350135015