# Context-based Coding

National Chiao Tung University
Chun-Jen Tsai
10/23/2014

# Rationale

- We get more compression when the message has a more skewed set of probabilities
  - Certain symbols occur with higher probability than others
  - We can look for ways to represent the message that would result in greater skew
- One effective way is to look at the probability of occurrence of a letter in the context in which it occurs
  - Examine the history of the sequence before determining the likely probabilities of different values the symbol can take
  - For example, the Markov models

# Shannon's Experiments

- For English text, Shannon showed the role of context in two letter-guessing experiments
  - Experiment 1: Each letter is guessed once, the correct answer will be given upon a wrong guess

| | |
|---|---|
| Actual Text | THE ROOM WAS NOT VERY LIGHT A SMALL OBLONG |
| Subject Performance | ----ROO------NOT-V-----I------SM----OBL--- |

    The dashes represent the letters that were correctly guessed

  - Experiment 2: Keep guessing until you get the answer, the number of guesses for each letter represents the message:

    11117521111113221911111811112195111183111

# Remarks on the Experiments

- For the first experiment, "–" has highest probability; for the second experiment, "1" is most probable
- Key question: can you reliably decode the encoded message?
- Shannon used these experiments to estimate the entropy of the English alphabet†. If $N-1$ preceding letters are known, the entropy $F_N$ of the next letter is estimated to be:

$$0.6 \leq F_{100} \leq 1.3$$

† C. E. Shannon, "Prediction and entropy of printed English," *The Bell System Technical Journal*, 30:50–64, January 1951.

# Difficulties in Practice

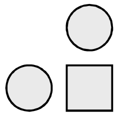
- Human subjects are much better at predicting the next letter in a sequence
  - Hard to formulate the behavior mathematically
- Grammar is hypothesized to be innate to humans
  - Development of a predictor as efficient as a human for language is hard
- However, the key idea is:

*Given the context, some symbols will occur more often than others. If the context is known to both encoder and decoder, we can use this skewed distribution to increase compression ratio.*

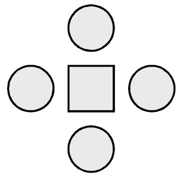
# Examples of Context

- For the word “*probability*”,
  - “*b*” is the first-order context for “*a*”
  - “*ob*” is the second-order context for “*a*”
- For an grayscale image, we can use a neighborhood structure to define context
  - To put condition on the square pixel, we can use the circular pixels as the context:

causal context
(scanline-order)



Non-causal context



- Higher order (or larger) context usually produces higher degree of skewed distribution

# Predictive Coding vs. Context Coding

- For predictive coding, we use a deterministic model to de-correlate the data
  - The final data that enters the entropy coder are treated as IID data
- For context-based coding, we use a conditional probability model to skew the distribution of the data
  - The entropy use the skewed distribution to encode the original (unprocessed) data

# Context-based Coding

- Use conditional probability to skew distribution
  - Unconditional probability: $P(\text{‘}h\text{’}) = 0.05$, $P(\text{‘}u\text{’}) = 0.02$.
  - Conditional probability: $P(\text{‘}h\text{’} \mid \text{‘}t\text{’}) = 0.3$, $P(\text{‘}u\text{’} \mid \text{‘}q\text{’}) = 0.99$.
- Practical issues:
  - Should dynamic or static statistics be used?
  - Using high-order context requires a (extremely) large probability table
- Solutions:
  - Adaptive scheme
  - Using contexts of variable sizes

# Prediction with Partial Match

- Prediction with Partial Match (PPM) was proposed by Cleary and Witten in 1984
- Key idea: instead of estimating these probabilities ahead of time, we estimate the probabilities as the coding proceeds
  - Only need to store those contexts that have occurred in the sequence being encoded
  - Need to code letters that have not occurred previously in this context $\rightarrow$ using escape symbol

# A Simple Example

- Input sequence: *probability*
- Current symbol: $a$
- Check if $P(a \mid prob)$ availble $\rightarrow$ fourth-order context
  - If yes, encode $a$, update $P(a \mid prob)$
  - If not, send escape code,
    then check $P(a \mid rob) \rightarrow$ third-order context
  - . . . continue checking low-order contexts . . .
  - If $P(a \mid b)$ is not available, check $P(a) \rightarrow$ zero-order context
- If '$a$' has never happened before, use $P(a) = 1/M$, where $M$ is the alphabet size, to encode $a$
  - The equi-probable model is called '$-1$' order context

# The PPM Algorithm

- The maximal context order, $N$, must be selected in advance
- For $K = N\ ..\ 1$
  - Read $K$ symbols before current symbol $s$
  - If the $K$-$th$ order context is not available, decrement $K$ and continue
  - Otherwise, if the $K$-$th$ order context does not contain $s$
    - Encode an escape symbol
    - Decrement $K$ and continue
  - Otherwise, encode $s$ using the $K$-$th$ order context and break loop
- If $s$ is not encoded, use $-1$ order statistics
  - All symbols have equal probabilities
- After encoding, update context statistics (not for escape code)

# Remarks on the PPM Algorithm

- Each time a symbol is encountered, the count corresponding to that symbol is updated
- The number of counts to be assigned to the escape symbol is not obvious
  - Cleary and Witten gave the escape symbol a count of one, thus inflating the total count by one
- For coding of a symbol, arithmetic code is often used
  - AC adapts to conditional probabilities easily

# Example: English Text, $N = 2$ (1/8)

- Input sequence: *this∆is∆the∆tithe*
- Encoded symbols: *this∆is*
- Word length for AC is 6: $l = 000000$ and $u = 111111$.

–1 order context:

| *Letter* | *Count* | *Cum_Count* |
|---|---|---|
| *t* | 1 | 1 |
| *h* | 1 | 2 |
| *i* | 1 | 3 |
| *s* | 1 | 4 |
| *e* | 1 | 5 |
| *∆* | 1 | 6 |
| *Total Count* | | 6 |

Zero-order context

| *Letter* | *Count* | *Cum_Count* |
|---|---|---|
| *t* | 1 | 1 |
| *h* | 1 | 2 |
| *i* | 2 | 4 |
| *s* | 2 | 6 |
| *∆* | 1 | 7 |
| *<ESC>* | 1 | 8 |
| *Total Count* | | 8 |

# Example: English Text, $N = 2$ (2/8)

- The first-order contexts after coding *this∆is* are:

| *Context* | *Letter* | *Count* | *Cum_Count* |
|---|---|---|---|
| *t* | *h* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |
| *h* | *i* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |
| *i* | *s* | 2 | 2 |
| | *<ESC>* | 1 | 3 |
| | *Total Count* | | 3 |
| *∆* | *i* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |
| *s* | *∆* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |

# Example: English Text, $N = 2$ (3/8)

- The next symbol is "$\Delta$", and the 2nd-order contexts of "$\Delta$" is "$is$":

| *Context* | *Letter* | *Count* | *Cum_Count* |
|---|---|---|---|
| *th* | *i* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |
| *hi* | *s* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |
| *is* | $\Delta$ | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |
| $s\Delta$ | *i* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |
| $\Delta i$ | *s* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |

# Example: English Text, $N = 2$ (4/8)

- To encode "$\Delta$", the update equations for the lower and upper limits of AC are

$$l = 0 + \left\lfloor (63 - 0 + 1) \times \frac{0}{2} \right\rfloor = 0 = 000000$$

$$u = 0 + \left\lfloor (63 - 0 + 1) \times \frac{1}{2} \right\rfloor - 1 = 31 = 011111$$

- As $l$ and $u$ are both less than $0.5$ $(= 100000)$, we perform an $E_1$ scaling:

Transmitted sequence: 0

$l$ = 000000

$u$ = 111111

| *Context* | *Letter* | *Count* | *Cum_Count* |
|---|---|---|---|
| *is* | $\Delta$ | 2 | 2 |
| | *<ESC>* | 1 | 3 |
| *Total Count* | | | 3 |

# Example: English Text, $N = 2$ (5/8)

- Next symbol is "$t$" and its 2nd-order context is $s\Delta$.
  $\rightarrow$ first appearance; encode an escape symbol!
  $\rightarrow$ the $l$ and $u$ of AC are updated as

$$l = 0 + \left\lfloor (63 - 0 + 1) \times \frac{1}{2} \right\rfloor = 32 = 100000$$

$$u = 0 + \left\lfloor (63 - 0 + 1) \times \frac{2}{2} \right\rfloor - 1 = 63 = 111111$$

| | | | |
|---|---|---|---|
| $s\Delta$ | $i$ | 1 | 1 |
| | <ESC> | 1 | 2 |
| | Total Count | | 2 |

- As $l$ and $u$ are both larger than $0.5$ (= 100000), we perform an $E_2$ scaling:

Transmitted sequence: 01
$l$ = 000000
$u$ = 111111

| Context | Letter | Count | Cum_Count |
|---|---|---|---|
| $s\Delta$ | $i$ | 1 | 1 |
| | $t$ | 1 | 2 |
| | <ESC> | 1 | 3 |
| | Total Count | | 3 |

# Example: English Text, $N = 2$ (6/8)

- Check the first-order context of $t$, which is $\Delta$
  $\rightarrow$ $t$ has not previously occurred in this context
  $\rightarrow$ encode another escape:

$$l = 0 + \left\lfloor (63 - 0 + 1) \times \frac{1}{2} \right\rfloor = 32 = 100000$$

$$u = 0 + \left\lfloor (63 - 0 + 1) \times \frac{2}{2} \right\rfloor - 1 = 63 = 111111$$

| | | | |
|---|---|---|---|
| $\Delta$ | *i* | 1 | 1 |
| | *<ESC>* | 1 | 2 |
| | *Total Count* | | 2 |

- As $l$ and $u$ are both larger than $0.5$ (= $100000$), we perform an $E_2$ scaling:

Transmitted sequence: 011
$l = 000000$
$u = 111111$

| *Context* | *Letter* | *Count* | *Cum_Count* |
|---|---|---|---|
| $\Delta$ | *i* | 1 | 1 |
| | *t* | 1 | 2 |
| | *<ESC>* | 1 | 3 |
| | *Total Count* | | 3 |

# Example: English Text, $N = 2$ (7/8)

- Now, check the updated zero-order contexts to see if $t$ has occurred before:

| Letter | Count | Cum_Count |
|---|---|---|
| $t$ | 1 | 1 |
| $h$ | 1 | 2 |
| $i$ | 2 | 4 |
| $s$ | 2 | 6 |
| $\Delta$ | 2 | 8 |
| <*ESC*> | 1 | 9 |
| Total Count | | 9 |

- Encode $t$ using zero-order contexts:

$$l = 0 + \left\lfloor (63 - 0 + 1) \times \frac{0}{9} \right\rfloor = 0 = 000000$$

$$u = 0 + \left\lfloor (63 - 0 + 1) \times \frac{1}{9} \right\rfloor - 1 = 6 = 000110$$

# Example: English Text, $N = 2$ (8/8)

- The three most significant bits of both $l$ and $u$ are the same, so we shift them out and update $l$ and $u$:

$$\begin{aligned}&\text{Transmitted sequence: } 011000\\ &l = 000000\\ &u = 110111\end{aligned}$$

- The next symbol is $h$. There is no 2nd-order context of $\Delta t$. Create the context '$\Delta t$' with the initial occurrence of $h$ and <$ESC$> in the context, and go directly to the 1st-order context of $t$:

$$l = 0 + \left\lfloor (55 - 0 + 1) \times \frac{0}{2} \right\rfloor = 0 = 000000$$

$$u = 0 + \left\lfloor (55 - 0 + 1) \times \frac{1}{2} \right\rfloor - 1 = 27 = 011011$$

$\rightarrow$ Transmitted sequence: 0110000

# Escape Symbol Design

- There are several ways to set escape symbol count
  - Method A – the count is one for the escape symbol (ppma)

| *Context* | *Symbol* | *Count* |
|---|---|---|
| *prob* | *a* | 10 |
| | *l* | 9 |
| | *o* | 3 |
| | *<ESC>* | 1 |
| *Total Count* | | 23 |

  - Method B & C – the count equals the number of symbols in the context (more symbols, higher probability to escape)

| *Context* | *Symbol* | *Count* |
|---|---|---|
| *prob* | *a* | 9 |
| | *l* | 8 |
| | *o* | 2 |
| | *<ESC>* | 3 |
| *Total Count* | | 22 |

Method B

| *Context* | *Symbol* | *Count* |
|---|---|---|
| *prob* | *a* | 10 |
| | *l* | 9 |
| | *o* | 3 |
| | *<ESC>* | 3 |
| *Total Count* | | 25 |

Method C

# Context Length Selection

- A long context will result in a high conditional probability of the following symbol
  - Close to deterministic context (i.e., a context that always followed by the same symbol)
  - Wasting bits coding escape symbols
- Cleary and Teahan proposed a context length selection algorithm called ppm* in 1997:
  - For input symbol $s$, look for the longest deterministic context
  - If there is no deterministic context for $s$, an escape symbol is encoded and the algorithm default back to the ppm algorithm
  - PPM* is roughly 6% more efficient than PPM method C

# The Exclusion Principle

- Arithmetic coding uses subintervals of [0, 1) to represent symbols
  - Smaller subinterval leads to more bit length
  - Exclude the unused subintervals in the higher-order context
- Example: encode "*proba*"

| *Context* | *Symbol* | *Count* |
|---|---|---|
| *ob* | *l* | 10 |
| | *o* | 3 |
| | *<ESC>* | 2 |
| *Total Count* | | 15 |
| *b* | *l* | 5 |
| | *o* | 3 |
| | *a* | 4 |
| | *r* | 2 |
| | *e* | 2 |
| | *<ESC>* | 5 |
| *Total Count* | | 21 |

Escape from the context of '*ob*' means the current symbol is not an '*l*' or '*o*', we can use a smaller probability table to encode '*a*'

| *Context* | *Symbol* | *Count* |
|---|---|---|
| *b* | *a* | 4 |
| | *r* | 2 |
| | *e* | 2 |
| | *<ESC>* | 3 |
| *Total Count* | | 11 |

# Burrows-Wheeler Transform

- Often, we can apply an invertible “coordinate-transform” to our data so that the transformed data is easier to compress than the original data
- Question:
  - Can we transform a sequence so that dictionary-based compression techniques can do a better job?
- The three-step Burrows-Wheeler Transform (BWT) published in 1994 is one of such transforms
  - Step I: cyclic permutations
  - Step II: lexicographical sorting
  - Step III: removal of redundant information and transmit the rest to the decoder

# Example of BWT (1/2)

- Input sequence: *this_is_the*
- Forward transform:

**Step I:**

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

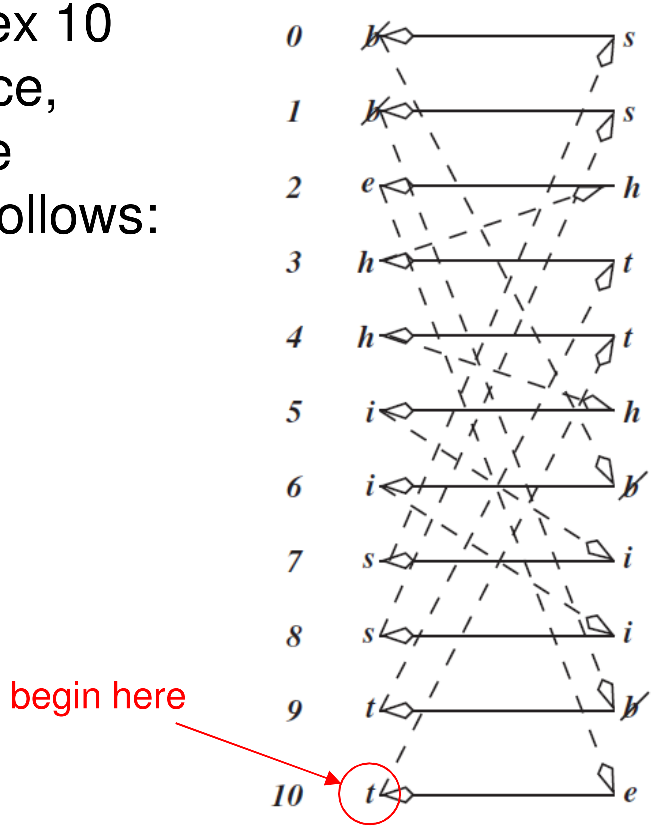
**Step II:**

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

**Step III:** transmit the index 10 and the sequence *L*: *sshtth_ii_e* to the decoder

# Example of BWT (2/2)

- Backward transform:
  - Given the sequence L, we can reconstruct the first column sequence F in step II by alphabetical sorting
  - With L, F, and the index 10 of the original sequence, we can reconstruct the original sequence as follows:



# Move-to-Front Coding (MTF)

- A coding scheme that takes advantage of long runs of identical symbols
  - Start with an indexed listing of the source alphabet with 0 as the first index
  - If a symbol occurs, its index is transmitted, then it is moved to the top of the list – If we have a run of this symbol, we will transmit a sequence of 0s
- Example: encode *sshtth_ii_e* using the initial list:

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

The encoded sequence is: 4 0 3 5 0 1 3 5 0 1 5.

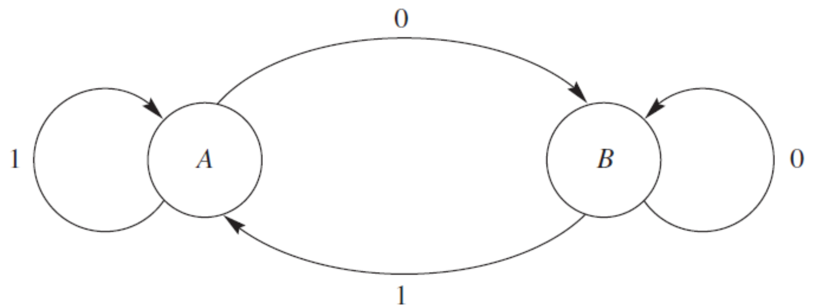
# Associative Coder of Buyanovsky

- Context information can also be used in dictionary-based technique. One such techniques is the ACB codec by Buyanovsky in 1994.
  - Dictionary: a sorted list of context-content pairs; sorting is based on context string (from right-to-left)
  - Look-ahead buffer: next letters to be coded
  - Algorithm:

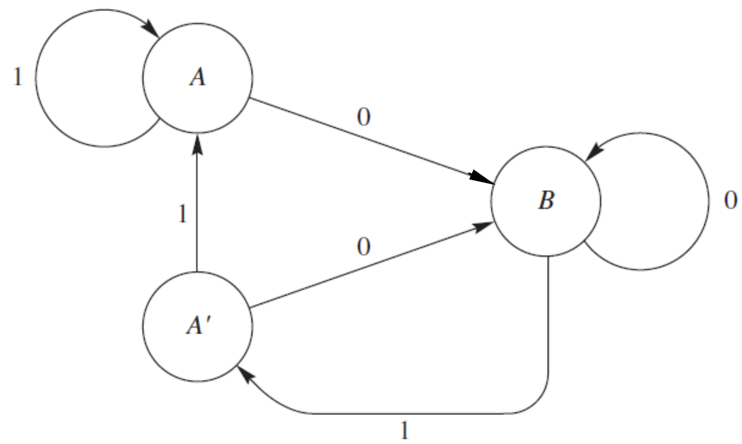
```
while (!EOF) do
begin
  i = position of the best matching context in dictionary
  j = position of the best matching content in dictionary
  count = count of matched prefix chars in found content
  ch = first mismatching character
  output(j-i, count, ch)
  update of dictionary
  shift look-ahead buffer by count+1 positions to the right
end
```

# Dynamic Markov Compression

- We have learned data compression with static 1st-order Markov model
  - Static high-order models are expensive to construct
- To adapt to longer context, we can grow the first-order model by dynamically "cloning" a state:



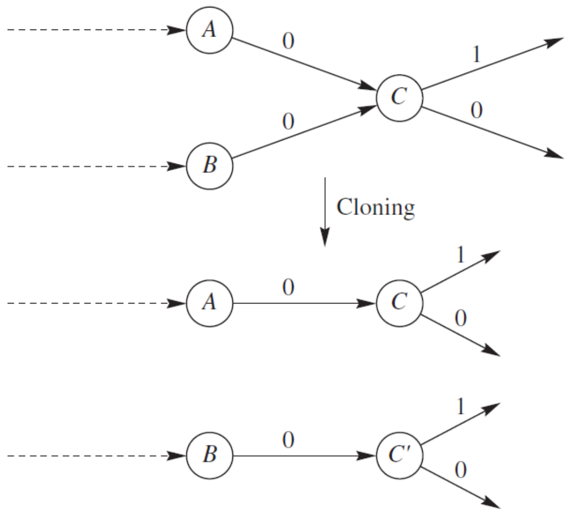

Original first-order model, state $A$ context could be "0" or "1"




A new state $A'$ that reflect the context "$0 \ldots 01$"

# The Cloning Process in DMC

- When two different states can leads into the same destination state, we can “clone” the destination:




- Only clone a state if rate reduction is expected
  - Happens when both source states enter the destination with high probability