Data Compression Techniques

Part 1: Entropy Coding

Lecture 4: Asymmetric Numeral Systems

Juha Kärkkäinen

08.11.2017

Asymmetric Numeral Systems

Asymmetric numeral systems (ANS) is a recent entropy coding technique that combines the advantages of Huffman and arithmetic coding:

- Similarly to arithmetic coding, ANS gets very close to the entropy.
- Similarly to Huffman coding, ANS can be implemented to run much faster than arithmetic coding.

The basic ANS coding procedure is:

$$\text{source string} \mapsto \text{integer} \mapsto \text{code string}$$

The mapping from integers to code strings is simple:

- Represent the integer in binary using (only) as many bits as needed.
- Remove the leading bit, which is always 1.

The opposite direction is equally simple:

- Add 1 to to the front of the bit string and interpret as an integer.

The missing piece is the mapping from source strings to integers.

The source string to integer mapping is based on the following observations:

- An integer x corresponds to a code string of length $\lfloor \log x \rfloor$ and thus should correspond to a source string of probability about $2^{-\lfloor \log x \rfloor} \approx 1/x$ to get close to entropy.
- The code string with probability 0.5 for each bit has the right probability as a string. Thus the code string to integer mapping is the right kind of mapping.

We want to generalize the mapping to handle

- larger alphabets and
- arbitrary probabilities.

Let us define the code string to integer mapping using arithmetic operations without relying directly on the binary representation of integers.

Let x be an integer encoding some binary string B. The encoding function $C(x, s)$ adds the bit $s \in \{0, 1\}$ to the end of the string and returns the new integer:

$$C(x, s) = 2x + s$$

Starting with $x = 1$, which encodes the empty string, we can construct the encoding for any string by repeated application of the encoding function.

The decoding function $D(x)$ is the reverse mapping that extracts a bit from the end and returns the new integer and the extracted bit:

$$D(x) = (\lfloor x/2 \rfloor,\ x \bmod 2)$$

The bits are extracted until x becomes 1.

The encoding and decoding functions can be described using the following ANS coding table.

$C(x,s)$		0	1	2	3	4	5	6	7	8	9	...
x	$s=0$	0		1		2		3		4		
	$s=1$		0		1		2		3		4	

Using the table, the functions can be computed as follows:

- To compute $C(x,s)$, find x on the row corresponding to s and return the value on the first row in that column.
- To compute $D(x)$, find x on the first row and return the value below it as well as the bit on the row containing that value.

Example

$C(3,1) = 7$ and $D(7) = (3,1)$.

By modifying the table, we can represent inequal probabilities. For example, the probabilities $P(0) = 1/4$ and $P(1) = 3/4$ could be represented with the following table.

$C(x,s)$		0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
x	$s=0$	0				1				2				3		
	$s=1$		0	1	2		3	4	5		6	7	8		9	

Example

Using the above table, the bitstring 0111 is encoded by the integer 13:

$$1 \xrightarrow{0} 4 \xrightarrow{1} 6 \xrightarrow{1} 9 \xrightarrow{1} 13$$

and the integer 12 decodes into the bitstring 110 (in reverse order):

$$12 \xrightarrow{0} 3 \xrightarrow{1} 2 \xrightarrow{1} 1$$

The generalization to larger alphabets can be accomplished by adding more rows, one for each symbol.

Each symbol row in the coding table contains 0, 1, 2, ... Thus the code table can be described by specifying for each column, which symbol row contains a value in that column. Let $E : \mathbb{N} \to \Sigma$ be such mapping. In other words, $E(x)$ is

- the last symbol encoded by the integer x, or equivalently
- the first symbol extracted during decoding, i.e., $D(x) = (\cdot, E(x))$.

We can also consider E to be the (infinite) string $E(0)E(1)E(2)\ldots$, which we call the ANS coding sequence. For example, the coding table on the preceding slide corresponds to the string $\texttt{011101110111...} = (\texttt{0111})^\infty$.

Example

For $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ with the probabilities $P(\texttt{a}) = 0.2$, $P(\texttt{b}) = 0.5$ and $P(\texttt{c}) = 0.3$, we could choose $E = (\texttt{aabbbbbccc})^\infty$ as the ANS coding sequence and obtain the following coding table.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
a	0	1									2	3									
b			0	1	2	3	4						5	6	7	8	9				
c								0	1	2								3	4	5	

Define the following operations on the sequence E:

$$\text{p-rank}_E(x) = \text{the number of occurrences of } E(x) \text{ in } E[0..x-1]$$
$$\text{select}_E(x, s) = \text{the position of } (x+1)^{\text{th}} \text{ occurrence of } s \text{ in } E$$

Example

x	0	1	2	3	4	5	6	7	8	9	10	11	12	...
$E(x)$	a	a	b	b	b	b	b	c	c	c	a	a	b	...
$\text{p-rank}_E(x)$	0	1	0	1	2	3	4	0	1	2	2	3	5	...
$\text{select}_E(x, \texttt{b})$	2	3	4	5	6	12	13	14	15	16	22	23	24	...

Then the encoding and decoding functions can be defined for any coding sequence as follows:

$$C(x, s) = \text{select}_E(x, s)$$
$$D(x) = (\text{p-rank}_E(x), E(x))$$

A good coding sequence E for a probability distribution P over an alphabet Σ should satisfy, for all $x \in \mathbb{N}$ and $s \in \Sigma$,

$$C(x, s) = \text{select}_E(x, s) \approx x/P(s)$$

Then a string $S \in \Sigma^*$ is encoded with an integer of value about $1/P(S)$ and the code string has a length of about $\log(1/P(S))$.

There are many possible ANS coding schemes. We will look at the two standard examples uABS and rANS, and then some implementation techniques leading to a practical variant called tANS.

uABS (uniform Asymmetric Binary Systems)

Let $\Sigma = \{0, 1\}$ with $P(1) = p$ and $P(0) = 1 - p$. Define the code sequence by

$$E(x) = \lceil (x+1)p \rceil - \lceil xp \rceil$$

Then the encoding function is

$$C(x, s) = \begin{cases} \lceil \frac{x+1}{1-p} \rceil - 1 & \text{if } s = 0 \\ \lfloor \frac{x}{p} \rfloor & \text{if } s = 1 \end{cases}$$

and the decoding function for is

$$D(x) = \begin{cases} (x - \lceil xp \rceil, E(x)) & \text{if } E(x) = 0 \\ (\lceil xp \rceil, E(x)) & \text{if } E(x) = 1 \end{cases}$$

There is also a variant using $\lfloor \cdot \rfloor$ instead of $\lceil \cdot \rceil$.

Example ($p = 0.3$)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
0		0	1		2	3		4	5	6		7	8		9	10		11	12	
1	0			1			2				3			4			5			

rANS (range ANS)

Let $\Sigma = \{s_1, s_2, \ldots, r_\sigma\}$. Choose an integer L and integers f_i, $i \in [1..\sigma]$, so that $f_i/L \approx P(s_i)$ and $\sum_{i\in[1..\sigma]} f_i = L$. In other words, we will approximate the probabilities with rational numbers f_i/L. Then set

$$E = (s_1^{f_i} s_2^{f_2} \ldots s_\sigma^{f_\sigma})^\infty$$

See slide 7 for an example.

Setting $F_i = \sum_{j\in[1..i-1]} f_j$, the encoding function is

$$C(x, s_i) = \lfloor x/f_i \rfloor \cdot L + x \bmod f_i + F_i$$

The decoding function is

$$D(x) = (\lfloor x/L \rfloor \cdot f_i + x \bmod L - F_i,\ s_i) \text{ for } i \text{ s.t. } s_i = E(x)$$

In practice, L should be a power of two to speed up decoding.

Renormalization

To avoid dealing with very large integers, we need occasionally to renormalize similarly to arithmetic coding. This is done with the decoding function.

- During encoding, we use the standard code string decoding function $D(x) = (\lfloor x/2 \rfloor, x \bmod 2)$ and output the resulting bit.
- During decoding, we use the source string decoding function and output the resulting source symbols.

With arithmetic coding renormalization does not change the eventual output. However, since the decoding function extracts bits in the reverse order, the extrated bits/symbols are from the middle of the sequence, not from the beginning as in arithmetic coding. This is not a problem, though, as long as the encoder–decoder symmetry is maintained:

- Whenever the encoder reads source symbols and encodes them into the integer, the decoder extracts source symbols from the integer and writes them to output.
- Whenever the encoder renormalizes and outputs code bits, the decoder reads code bits and encodes them to the integer.

Let R be a suitable integer:

- For uABS, $p = P(1)$ should be a multiple of $1/R$.
- For rANS, R should be a multiple of L.

The decoding procedure under renormalization is:

- If $x \geq R$, extract and output a source symbol.
- If $x < R$, read a code bit and encode it to the integer.

Let C_s be the source encoding function and D_c the code decoding function. The encoding procedure under renormalization is as follows when s is the next source symbol to be encoded.

- If $D_c(C_s(x, s)) \geq R$, extract and output a code bit (renormalize).
- If $D_c(C_s(x, s)) < R$, encode s to the integer.

The symmetry between encoder and decoder is maintained because:

- The encoding rules ensure that x never grows too big. Thus, if the encoder renormalizes computing $y = D_c(x)$, then $y < R$ so that the decoder does the inverse given integer y.
- The conditions on R ensure that, if the encoder computes $y = C_s(x, s)$, then $y \geq R$ so that the decoder does the inverse given integer y.

Example

For rANS with $L = 8$ let $E = (\texttt{abbbbbcc})^\infty$, i.e, $P(\texttt{a}) \approx 1/8$, $P(\texttt{b}) \approx 5/8$ and $P(\texttt{c}) \approx 2/8$. The coding table is

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	0								1							
b		0	1	2	3	4				5	6	7	8	9		
c							0	1							2	3

We use renormalization with $R = 8$. The string $\texttt{cab}$ is encoded as follows. The encoding starts with $x = 2$ allowing immediate encoding of the first symbol and ends with extracting code bits until x is 1.

$$2 \xrightarrow{\texttt{c}} 14 \underset{0}{\rightarrow} 7 \underset{1}{\rightarrow} 3 \underset{1}{\rightarrow} 1 \xrightarrow{\texttt{a}} 8 \xrightarrow{\texttt{b}} 12 \underset{0}{\rightarrow} 6 \underset{0}{\rightarrow} 3 \underset{1}{\rightarrow} 1$$

Inputs are above and outputs below the arrows. The decoder starts with $x = 1$, works in reverse order, and ends after outputting $n = 3$ symbols.

$$1 \xrightarrow{1} 3 \xrightarrow{0} 6 \xrightarrow{0} 6 \underset{\texttt{b}}{\rightarrow} 8 \underset{\texttt{a}}{\rightarrow} 1 \xrightarrow{1} 3 \xrightarrow{1} 7 \xrightarrow{0} 14 \underset{\texttt{c}}{\rightarrow} 2$$

tANS (table ANS)

rANS with renormalization has the property that the integer x is always in the range $[R..2R-1]$

- right after encoding a source symbol during encoding and
- right before decoding a source symbol during decoding.

The values of x in the range can be thought of as states of computation. The state transitions are then:

- Encoding: zero or more renormalization steps followed by a source symbol encoding.
- Decoding: source symbol extraction followed by zero or more code bit encodings.

The idea of tANS is to store all possible transitions into a table, and then do encoding and decoding with table lookups.

Example

Let $E = (\texttt{abbbbbcc})^\infty$ and $L = R = 8$ (see slide 14).
The tANS encoding table is:

	8	9	10	11	12	13	14	15
a	$\underset{000}{\rightarrow} 8$	$\underset{100}{\rightarrow} 8$	$\underset{010}{\rightarrow} 8$	$\underset{110}{\rightarrow} 8$	$\underset{001}{\rightarrow} 8$	$\underset{101}{\rightarrow} 8$	$\underset{011}{\rightarrow} 8$	$\underset{111}{\rightarrow} 8$
b	$\rightarrow 12$	$\rightarrow 13$	$\underset{0}{\rightarrow} 9$	$\underset{1}{\rightarrow} 9$	$\underset{0}{\rightarrow} 10$	$\underset{1}{\rightarrow} 10$	$\underset{0}{\rightarrow} 11$	$\underset{1}{\rightarrow} 11$
c	$\underset{00}{\rightarrow} 14$	$\underset{10}{\rightarrow} 14$	$\underset{01}{\rightarrow} 14$	$\underset{11}{\rightarrow} 14$	$\underset{00}{\rightarrow} 15$	$\underset{10}{\rightarrow} 15$	$\underset{01}{\rightarrow} 15$	$\underset{11}{\rightarrow} 15$

The string `cab` is encoded as follows. The first symbol is read just to set the initial state without output.

$$\overset{\texttt{c}}{\rightarrow} 14 \underset{011}{\overset{\texttt{a}}{\rightarrow}} 8 \overset{\texttt{b}}{\rightarrow} 12 \underset{001}{\rightarrow}$$

The last bits of output identify the final state 12. The full output is 011001, the same as on slide 14.

Example (continued)

Decoding table

8	$\xrightarrow[\texttt{a}]{b_1 b_2 b_3} 8 + 4b_1 + 2b_2 + b_3$
9	$\xrightarrow[\texttt{b}]{b_1} 10 + b_1$
10	$\xrightarrow[\texttt{b}]{b_1} 12 + b_1$
11	$\xrightarrow[\texttt{b}]{b_1} 14 + b_1$
12	$\xrightarrow[\texttt{b}]{} 8$
13	$\xrightarrow[\texttt{b}]{} 9$
14	$\xrightarrow[\texttt{c}]{b_1 b_2} 8 + 2b_1 + b_2$
15	$\xrightarrow[\texttt{c}]{b_1 b_2} 12 + 2b_1 + b_2$

Decoding starts with reading $\log R = 3$ bits to establish the initial state as $8 + 4b_1 + 2b_2 + b_3$. Then decoding proceeds according to the table. The last transition reads no bits.

$$\overset{100}{\to} 12 \underset{\texttt{b}}{\to} 8 \overset{110}{\to} 14 \underset{\texttt{c}}{\to}$$

Large Code Alphabet

ANS can also be modified for larger code alphabet $\Gamma = [0..\gamma - 1]$.

The coding function is then

$$C(x, s) = \gamma x + s$$

and the decoding function is

$$D(x) = (\lfloor x/\gamma \rfloor, x \bmod \gamma)$$

With tANS the set of states becomes $[R..\gamma R - 1]$.

A larger code alphabet can speed up coding by avoiding dealing with individual bits. In particular, $\gamma = 2^8$ allows byte level processing.

Summary: Entropy Coding

- Huffman coding, arithmetic coding and ANS coding address the same problem: encoding data using as few bits as possible given a probability distribution over the data. Such methods are often called entropy coding. ANS offering fast coding and close to entropy compression is the state of the art in entropy coding.
- Some codes such as Elias γ code are not explicitly constructed from a probability distribution. However, since the codeword lengths can be interpreted as probabilities, there is an underlying implicit probability distribution (see also Exercise 2.3).
- Entropy coding assumes that the probability distribution of each symbol is independent of other symbols. When this is not the case, such as with natural language text, other techniques are needed. However, even then entropy coding is usually applied in the last stage of compression to produce the actual bits.