Non-destructive encoder flushing for "arithmetic coding"

Andrea C. G. Mennucci [1]

June 30, 2019

1 Introduction

In this short essay we discuss *arithmetic coding*, as described in [1]. (The reference implementation is the code in `ArithCodeTut` [2] in `github.com`).

In particular we discuss the problem of *flushing*: a method implemented in the encoder that makes sure that the decoder has received all previously encoded symbols.

In [1], in the section *"Termination"*, the authors write [3]: *To finish the transmission, it is necessary to send [...] enough bits to ensure that the encoded string falls within the finale range. [...] The encoder need only transmit 01 if* `Low` $<$ `First_qtr`*, or 10 otherwise.*

While this approach works, it has a defect: after that sequence is emitted, it is unclear how the encoder could encode more symbols and send more bits to the decoder.

This may property may be of interest in some cases: indeed the Decoder can have an arbitrarily large delay in decoding the symbols that the encoder has already seen. (See next section). So a periodic flushing may be desirable — as long as encoding/decoding can proceed after flushing.

In the following a *Bernoulli symbol* is a symbol that has two choices, such as $0, 1$ or a, b, each choice with probability $1/2$ (unless otherwise stated).

The questions now is: can we justify the flushing output of the encoder as result of an input, let's say as iid Bernoulli symbols?

2 Example

We model the encoder as follows.

Let $[\alpha_j, \beta_j)$ the symbol interval stored in the Encoder after inserting the j-th symbol using `input_symbol()`: the encoder is in *dirty state*. Let $[\tilde\alpha_j, \tilde\beta_j]$ the symbol interval after the Encoder has emitted all bits (either by callback, or by polling), so the Encoder is in *clean state*: at this point either

$$0 \le \tilde\alpha_j < 1/4 \wedge 1/2 \le \tilde\beta_j \le 1$$

(condition (1a) in [1]) or

$$0 \le \tilde\alpha_j < 1/2 \wedge 3/4 \le \tilde\beta_j \le 1$$

(condition (1b) in [1])

Now suppose this happens.

The encoder receives many symbols, say N, but it does not emit any bit, neither store any virtual bits in `bits_to_follow`, because

$$0 \le \alpha_j < 1/4 \wedge 1/2 \le \beta_j \le 1$$

at all $j \le N$; so $\tilde\alpha_j = \alpha_j$, $\tilde\beta_j = \beta_j$. (A symmetric situation is possible)

At that time the decoder is still at the pristine state $[0, 1]$.

We suppose that the decoder knows that it will be flushed after receiving N symbols; either because this number was passed in a header of the encoded file, or because the N-th symbol is a special `WILL_FLUSH` symbol (similarly to the `EOF` symbol in [1]).

The encoder is then flushed, so that it emits enough bits so that the decoder can decode the N symbols. Since $\alpha_N < 1/4$, then the encoder sends a 0 and then a 1 (and would send `bits_to_follow` other ones, but in this example there are no virtual bits, for simplicity).

[1] Scuola Normale Superiore, Pisa, Italy

[2] `https://github.com/mennucc/ArithCodeTut.git`

[3] Rephrased for clarity

The decoder bit interval is now

$$[1/4,\, 1/2) = [0.01000\ldots0,\, 0.0111\ldots1]$$

(where on the right we express the binary representation of the interval in the algorithm); the decoder understands all symbols up to symbol N: indeed there is only one string of symbols such that $[\alpha_N, \beta_N] \supseteq [1/4,\, 1/2]$

3 Non-destructive flushing

3.1 Low case

We consider the case

$$0 \leq \alpha_N < 1/4 \wedge 1/2 \leq \beta_N \leq 1 \quad .$$

We wish explain the sequence "01" that the Encoder has sent for flushing, as the encoding of 3 i.i.d Bernoulli symbols. For simplicity though we express it as the sending of one symbol s uniformly distributed in $\{0, 1, 2, 3, 4, 5, 6, 7\}$: this symbol indeed is equivalent to "3 input bits" (up to numerical approximation).

So

$$\alpha_{N+1} = \alpha_N + \frac{7-s}{8}(\beta_N - \alpha_N) \quad , \quad \beta_{N+1} = \alpha_N + \frac{8-s}{8}(\beta_N - \alpha_N)$$

(we follow the convention of [1], that subintervals are in decreasing order as s increases — this is also the convention used in the reference code).

Since the width of $[\alpha_N, \beta_N]$ is at most 1, then the width of any subinterval $[\alpha_{N+1}, \beta_{N+1}]$ is at most $1/8$; so there must exist a choice of s such that $[\alpha_{N+1}, \beta_{N+1}]$ is contained in

$$[1/4,\, 1/2) = [0.01000\ldots0,\, 0.0111\ldots1]$$

(that has width $1/4$).

We will show that a possible choice is

$$s = \left\lfloor \frac{8\beta_N - 2}{(\beta_N - \alpha_N)} \right\rfloor - 1$$

Hence, if we input that symbol in the encoder after the N-th symbol, we will obtain two important results:

- the Encoder will emit 01, that will flush the system (it may emit a further bit; we skip the discussion);
- the Decoder will be able to decode the symbol, and Encoding/Decoding may proceed behind the flush.

Proof. To be consistent with what the decoder is receiving

$$1/4 \leq \alpha_{N+1} \quad , \quad \beta_{N+1} < 1/2 \tag{1}$$

that is

$$\begin{aligned}
1/4 \leq \alpha_{N+1} &\iff \\
1/4 \leq \alpha_N + \frac{7-s}{8}(\beta_N - \alpha_N) &\iff \\
\frac{2 - 8\alpha_N}{(\beta_N - \alpha_N)} \leq 7 - s &\iff \\
\frac{2 - \alpha_N - 7\beta_N}{(\beta_N - \alpha_N)} \leq -s &\iff \\
s \leq \frac{-2 + \alpha_N + 7\beta_N}{(\beta_N - \alpha_N)} = \frac{8\beta_N - 2}{(\beta_N - \alpha_N)} - 1 &
\end{aligned}$$

whereas

$$\begin{aligned}
\beta_{N+1} < 1/2 &\iff \\
\alpha_N + \frac{8-s}{8}(\beta_N - \alpha_N) < 1/2 &\iff \\
8 - s < \frac{4 - 8\alpha_N}{(\beta_N - \alpha_N)} &\iff \\
-s < \frac{4 - 8\beta_N}{(\beta_N - \alpha_N)} &\iff \\
s > \frac{8\beta_N - 4}{(\beta_N - \alpha_N)}
\end{aligned}$$

summarizing

$$\frac{8\beta_N - 4}{(\beta_N - \alpha_N)} < s \leq \frac{8\beta_N - 2}{(\beta_N - \alpha_N)} - 1 \tag{2}$$

We double-check that the interval contains an integer, indeed the difference of the extremes is

$$\frac{2}{(\beta_N - \alpha_N)} - 1 \geq 1$$

□

3.2 Examples

Some examples of possible values of

$$s = \left\lfloor \frac{8\beta_N - 2}{(\beta_N - \alpha_N)} \right\rfloor - 1$$

- If $\alpha_N = 1/4$ then

  $$\frac{8\beta_N - 2}{(\beta_N - 1/4)} = 8$$

  so $s = 7$, regardless of β_N.

- If $\alpha_N = 0$

  $$\frac{8\beta_N - 2}{\beta_N} = 8 - \frac{2}{\beta_N}$$

  so

  $$s = 7 - \left\lceil \frac{2}{\beta_N} \right\rceil \quad ,$$

  so $s \in \{5, 6, 7\}$;

- in particular if if $\beta_N = 1, \alpha_N = 0$ then $s = 5$.

3.3 High case

We consider the case

$$0 \leq \alpha_N < 1/2 \wedge 3/4 \leq \beta_N \leq 1 \quad .$$

In this case the encoder emits 10; a good choice is

$$s = \left\lfloor \frac{8\beta_N - 4}{(\beta_N - \alpha_N)} \right\rfloor - 1$$

Proof. To be consistent with what the decoder is receiving

$$1/2 \leq \alpha_{N+1} \quad , \quad \beta_{N+1} < 3/4 \tag{3}$$

that is

$$\begin{aligned} 1/2 \leq \alpha_{N+1} &\iff \\ 1/2 \leq \alpha_N + \frac{7-s}{8}(\beta_N - \alpha_N) &\iff \\ \frac{4 - 8\alpha_N}{(\beta_N - \alpha_N)} \leq 7 - s &\iff \\ \frac{4 - \alpha_N - 7\beta_N}{(\beta_N - \alpha_N)} \leq -s &\iff \\ s \leq \frac{-4 + \alpha_N + 7\beta_N}{(\beta_N - \alpha_N)} = \frac{8\beta_N - 4}{(\beta_N - \alpha_N)} - 1 & \end{aligned}$$

whereas

$$\begin{aligned} \beta_{N+1} < 3/4 &\iff \\ \alpha_N + \frac{8-s}{8}(\beta_N - \alpha_N) < 3/4 &\iff \\ 8 - s < \frac{6 - 8\alpha_N}{(\beta_N - \alpha_N)} &\iff \\ -s < \frac{6 - 8\beta_N}{(\beta_N - \alpha_N)} &\iff \\ s > \frac{8\beta_N - 6}{(\beta_N - \alpha_N)} & \end{aligned}$$

summarizing

$$\frac{8\beta_N - 6}{(\beta_N - \alpha_N)} < s \leq \frac{8\beta_N - 4}{(\beta_N - \alpha_N)} - 1 \tag{4}$$

We double-check that the interval contains an integer, indeed the difference of the extremes is

$$\frac{2}{(\beta_N - \alpha_N)} - 1 \geq 1$$

□

3.4 Remark on symmetry

Suppose

$$0 \leq \hat{\alpha}_N < 1/2 \wedge 3/4 \leq \hat{\beta}_N \leq 1 \quad .$$

This is symmetric of the "low" case, up to defining

$$\hat{\alpha}_j = 1 - \beta_j \quad , \quad \hat{\beta}_j = 1 - \alpha_j$$

by defining $\hat{s} = 8 - s$ the inequality (2) becomes the inequality (4); but the role of $\leq$ and $<$ is inverted.

References

[1] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.