ARITHMETIC CODING

6.441 Supplementary Notes 4, 2/24/94

Arithmetic coding is a somewhat different approach to data compression from the fixed to variable length and variable to fixed length encoders we have considered up to now. The idea in arithmetic coding is to map the source sequence u_1, u_2 ... into a point x in the unit interval on the real line and then to represent this point by its binary (i.e. radix 2) expansion,

$$x = \sum_{n=1}^{\infty} x_n 2^{-n}$$

where x_n is 0 or 1 for each n. If the mapping from u_1, u_2 ... to x is done in such a way that the random variable x is uniformly distributed on the real line, then each of the digits x_1, x_2 ... in the expansion of x is independent and equiprobably equal to 0 or 1.

To see how to accomplish this, consider a source with the alphabet {0,1,...K-1} and denote the random sequence produced by the source as $u_1, u_2, u_3, \ldots$. To start, assume that the source is memoryless and let $P_i = P[u_m=i]$, $0 \le i < K$, all $m \ge 1$. We shall see later that the memoryless assumption can easily be dispensed with. Let u^m be the initial string $u_1, u_2, \ldots u_m$ of the source output for each $m \ge 1$. The probability and self information of u^m are then

$$P[u^m] = \prod_{j=1}^{m} P[u_j] \; ; \; I(u^m) = \sum_{j=1}^{m} I(u_j)$$

In order to see how to map u_1, u_2 ... into x, let us recall some of the properties of the binary expansion, $x = \sum x_n 2^{-n}$. The first n bits, $x^n = (x_1, x_2, \ldots, x_n)$, of the expansion of x indicates that x lies in the interval

$$J(x^n) = \left[\sum_{i=1}^{n} x_i 2^{-i},\ \sum_{i=1}^{n} x_i 2^{-i} + 2^{-n}\right) \qquad (1)$$

Furthermore, $J(x^{n+1})$ is contained in $J(x^n)$ for each n. Also if $x_1, x_2, \ldots$ are independent and equiprobably 0 or 1, then the probability that x lies in a particular interval $J(x^n)$ is 2^{-n}.

Figures 1 and 2 illustrate how this can be generalized to a memoryless source. Define

$$f_1(u) = \sum_{i=0}^{u-1} P_i \qquad (2)$$

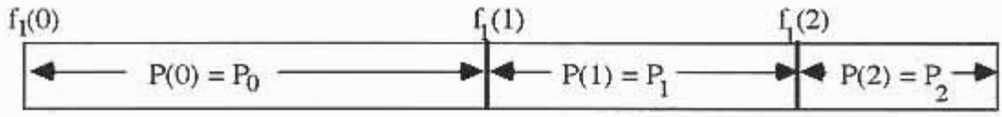


Figure 1

For $m \geq 1$ define

$$f(u^m) = f(u^{m-1}) + f_1(u_m)P(u^{m-1}); \quad m>1 \qquad (3)$$

where $f(u^0) = 0$ and $P(u^0) = 1$.

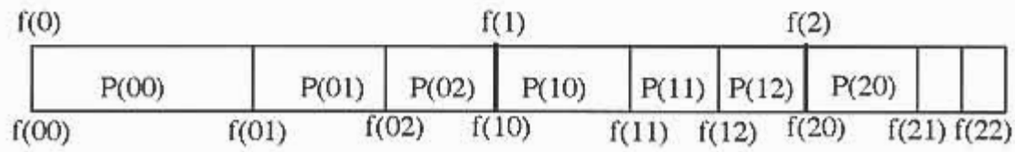


Figure 2

Letting $J(u^m)$ be the interval $[f(u^m), f(u^m) + P(u^m))$, we see that these intervals have the nesting property: $J(u^{m+1})$ is contained in $J(u^m)$ where u^m is the first m letters of u^{m+1}. For the given source probabilities, the width of an interval $J(u^m)$ is equal to the probability of the string u^m. Assuming that all the letter probabilities are strictly less than 1, the width of $J(u^m)$ then shrinks to 0 as $m \to \infty$ for all sequences $(u_1, u_2 \ldots u_m \ldots)$. Thus we can visualize x

as the point to which the sequence of nested intervals $J(u^m)$ shrinks as $m \to \infty$. There are some minor mathematical difficulties in requiring each sequence u^∞ to correspond to a different point x; for example if u^∞ comes from a binary source, then 10000... and 01111... each correspond to the same point. We ignore this for the time being since it is automatically taken care of when we come to grips with truncation errors.

We next visualize an encoder as operating on the letters $u_1\ u_2...$ as they are emitted from the source. On observing u^m, the encoder knows that the limit point x lies in the interval $J(u^m)$. Thus if $J(u^m)$ is contained in $J(x^n)$ for some binary string $x^n = x_1\ x_2...x_n$, then the encoder can emit x^n as the first n bits of the binary representation of x. As the source emits successive letters u_m, the interval $J(u^m)$ shrinks and successively more encoded bits can be emitted. For the largest n such that $J(u^m)$ is contained in $J(x^n)$, we have (see Fig. 3)

$$\sum_{i=1}^{n} x_i 2^{-i} \le f(u^m) < \sum_{i=1}^{n} x_i 2^{-i} + 2^{-n-1} < f(u^m) + P(u^m) < \sum_{i=1}^{n} x_i 2^{-i} + 2^{-n} \qquad (4)$$

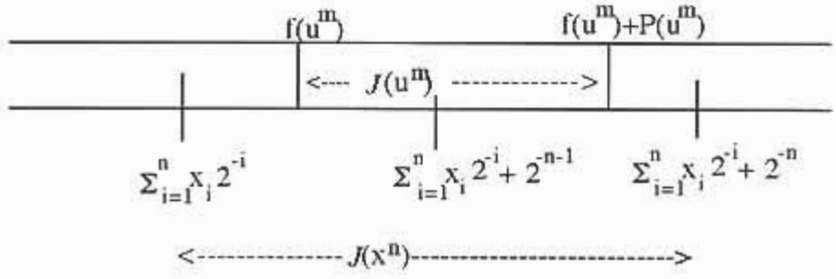


Figure 3

Key questions in determining how well this type of encoding performs are the questions of how rapidly binary digits come out of the encoder and how rapidly the source digits can be reconstructed from the encoded digits. In particular, we want to show that when the source has emitted u^m, the encoder will have emitted close to $I(u^m)$ binary digits, and that this will be sufficient for the decoder to decode all but the last few letters of u^m. We first analyze the

number of letters that the source must emit in order for the encoder to emit the first n binary digits. Let m(n) be the number of source letters that must be emitted by the source before the encoder can emit some given sequence x^n. Since $P(u^{m(n)})$ is the size of $J(u^{m(n)})$ and 2^{-n} is the size of $J(x^n)$, the fact that $J(u^{m(n)})$ is contained in $J(x^n)$ implies that $P[u^{m(n)}] \leq 2^{-n}$. Taking logs of this expression, $I(u^{m(n)}) \geq n$; intuitively this says that the source must produce at least n bits of information before the encoder can encode n bits. On the other hand, this inequality can be arbitrarily loose. In terms of figure 3 above, successive source letters could be selected in such a way that $J(u^m)$ continues to straddle the point $x_1 2^{-1} + \ldots + x_n 2^{-n} + 2^{-n-1}$ for arbitrarily large m. Thus we want to show that for each n, $E[I(u^{m(n)})]-n$ is bounded.

In order to accomplish this, let x^n be fixed and let x be the final encoded point. The point x, conditional on x^n, is a uniformly distributed random variable in the interval $J(x^n)$, but for the time being we consider it as a fixed value. Define D(x) as the distance between x and the nearest end point of $J(x^n)$ (see Fig. 4), i.e.,

$$D(x) = \min\left[x - \sum_{i=1}^{n} x_i 2^{-i} \ , \ \sum_{i=1}^{n} x_i 2^{-i} + 2^{-n} - x \right] \qquad (5)$$

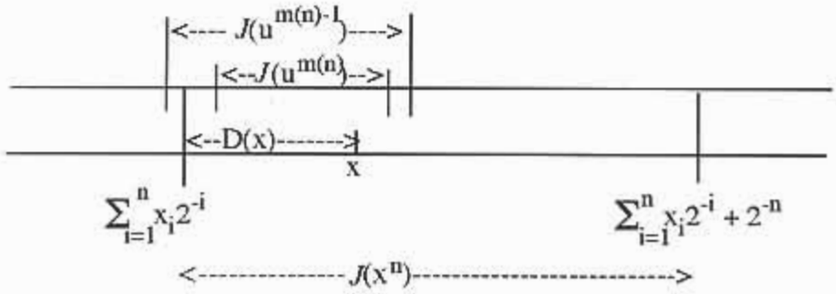


Figure 4

We note that the point x must be contained in $J(u^m)$ for all m. Also, since m(n), by definition, is the smallest m for which $J(u^m)$ is contained in $J(x^n)$, we see that $J(u^{m(n)-1})$

must contain one of the end points of $J(x^n)$ as well as x and thus must have a width of at least D(x). Thus, for the given x, $P(u^{m(n)-1}) \geq D(x)$, and

$$I(u^{m(n)-1}) \leq -\log D(x) \tag{6}$$

Now consider x as a random variable uniformly distributed over $J(x^n)$. D(x) is then uniformly distributed between 0 and 2^{-n-1} (i.e., D(x) is the distance to the nearest end point of $J(x^n)$, so is at most half the size of $J(x^n)$). Using (6), we see that, for the given x^n

$$E[I(u^{m(n)-1})] \leq -E[\log (D(x))]. \tag{7}$$

Note that as x varies within the given interval $J(x^n)$, $u^{m(n)-1}$ varies discretely, and the expectation is taken condtional on the given x^n. We can evaluate E[log (D(x))] since D(x) is uniformly distributed.

$$E[\log(D(x))] = \int_{D=0}^{2^{-n-1}} (2^{n+1})(\log D)\, dD = \left[2^{n+1} D \log(D/e)\right]_0^{2^{-n-1}} = -n-1-\log e$$

$$E[I(u^{m(n)-1}] \leq n+1+\log e = n + \log(2e) \tag{8}$$

Next define P_{min} as the probability of the least likely letter from the alphabet. Then $I(u^m) \leq I(u^{m-1}) + \log(1/P_{min})$ for all m and u^m, so that[1]

$$E[I(u^{m(n)})] \leq n + \log(2e/P_{min}) \tag{9}$$

---

[1]One might think that $E[I(u^{m(n)})] = E[I(u^{m(n)-1})] + H(U)$, but this is not usually true since m(n) is a random variable that depends on x. In particular, those source letters that produce outputs from the encoder tend to be less probable letters (on the average) than source letters that do not lead to outputs.

Since we have seen that $I(u^{m(n)}) \geq n$, we see that the encoder generates binary digits, on the average, with only a slight delay from the ideal of one binary digit for each bit of self information. Next we want to understand the delay between the generation of binary digits at the decoder and the generation of decoded source letters. We choose some arbitrary source sequence u^m and ask how many binary digits, n(m), must be received at the decoder before the sequence u^m can be decoded. When the decoder sees x^n, the decoder knows that x lies inside $J(x^n)$, and thus can decode u^m if $J(u^m)$ completely contains $J(x^n)$. Figure 5 shows the relationship of $J(u^m)$ to $J(x^{n(m)})$ and also shows $J(u^{m'})$, where m' is the

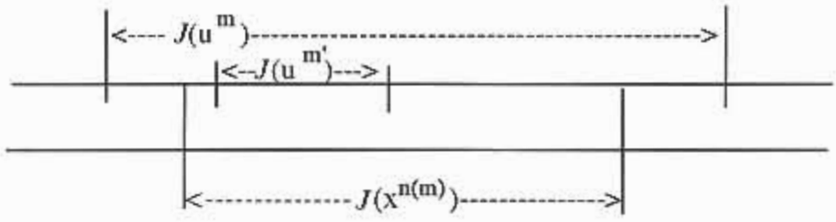


Figure 5

number of source letters that must be generated for the encoder to generate $J(x^{n(m)})$. Note that the number of binary digits n(m) depends on u^m and also on the subsequent source outputs. It is most convenient here, as in our previous analysis of the encoder, to first condition the argument on a given limit point x contained in the interval $J(u^m)$. As before, define D*(x) as the distance from x to the nearest edge of $J(u^m)$, i.e.,

$$D^*(x) = \min\left[x - f(u^m) \ , \ f(u^m)+P(u^m)-x\right] \tag{10}$$

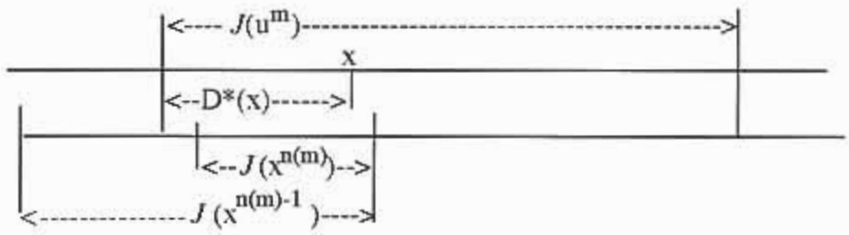


Figure 6

Since n(m) is by definition the number of binary digits required to decode u^m, we see that the interval $J(x^{n(m)-1})$ cannot lie inside $J(u^m)$ (see Fig. 6). Thus $J(x^{n(m)-1})$ contains one of the end points of $J(u^m)$ and of course also contains x. Thus the size of $J(x^{n(m)-1})$ (which is $2^{-n(m)+1}$) is at least D*(x). Thus, conditioned on x, we have $n(m)-1 \le -\log D^*(x)$. Next, for a given u^m, regard x as a random variable uniformly distributed over the interval $J(u^m)$. D*(x) is then uniformly distributed between 0 and $P(u^m)/2$ (i.e., half the size of $J(u^m)$) and E[D*(x)] can be evaluated as

$$E[\log(D^*(x))] = \int_{D^*=0}^{P(u^m)/2} (2/P(u^m)) \log(D^*)\, dD^* = \log\left(\frac{P(u^m)}{2e}\right)$$

Thus we can upper bound $E[n(m) \mid u^m]$ as

$$E[n(m) \mid u^m] \le 1 - \log(P(u^m)) + \log(2e) = I(u^m) + \log(4e) \qquad (11)$$

We now want to combine (9) and (11). Consider a given sequence u^m out of the decoder, and suppose that $x^{n(m)}$ is the required encoded sequence to decode u^m. Conditional on both u^m and $x^{n(m)}$, we see that x is uniformly distributed over $J(x^{n(m)})$, and thus the extended source sequence $u^{m'}$ required to produce $x^{n(m)}$ satisfies (from 9)

$$E[I(u^{m'})|x^{n(m)}] \le n(m) + \log(2e/P_{min})$$

Using (11) to take the expected value of this over n(m), we see that for any given u^m, the expected self information of the extended source sequence $u^{m'}$ required from the source to produce the n(m) binary digits needed to decode u^m satisfies

$$E[I(u^{m'}) \mid u^m] - I(u^m) \le \log(8e^2/P_{min}) \qquad (12)$$

The expectation here is over the source letters u_{m+1}, u_{m+2}, ... for the given sequence u^m. It is important to note that the bound does not depend on m or u^m. Note that on the average there is very little delay from encoder to decoder and that the average number of binary digits, over the long term is exactly H(U) binary digits per source letter. The bound in (12) is in terms of the additional self information needed in additional source letters u_{m+1},... until u^m can be decoded. To convert the bound into a bound on the number of letters, m'-m, let P_{max} be the maximum source letter probability. Then $\log(1/P_{max})$ is the minimum possible self information per source letter and

$$E[m'-m \mid u^m] \leq \frac{\log(8e^2/P_{min})}{\log(1/P_{max})} \qquad (13)$$

IMPLEMENTATION & ROUND OFF ERRORS:

In actual implementation, it is not possible to calculate the intervals used in encoding and decoding exactly. We view the arithmetic as being done using binary fixed point arithmetic with M binary digits of accuracy. There is some flexibility in how numbers are rounded to M bits, but essential that encoder and decoder use exactly the same rule and that the rounding is done at the appropriate time. It is also essential, since $P(u^m)$ is approaching 0 with m, that the intervals be renormalized as the intervals shrink.

The encoder operates as follows: source letters come in one at a time and the corresponding interval is calculated, with round off to M bits. Thus the encoder is a finite state encoder in the sense of the Lempel-Ziv notes. After each source letter enters the encoder, as many binary digits as possible are encoded using the rounded off intervals. After emitting a string of binary digits, the interval is renormalized in the sense that the most significant binary digits of $f(u^m)$ and $g(u^m)$ (i.e. the newly generated binary code digits) are shifted out of the arithmetic unit, with the remaining digits becoming more significant. (see figure 7).

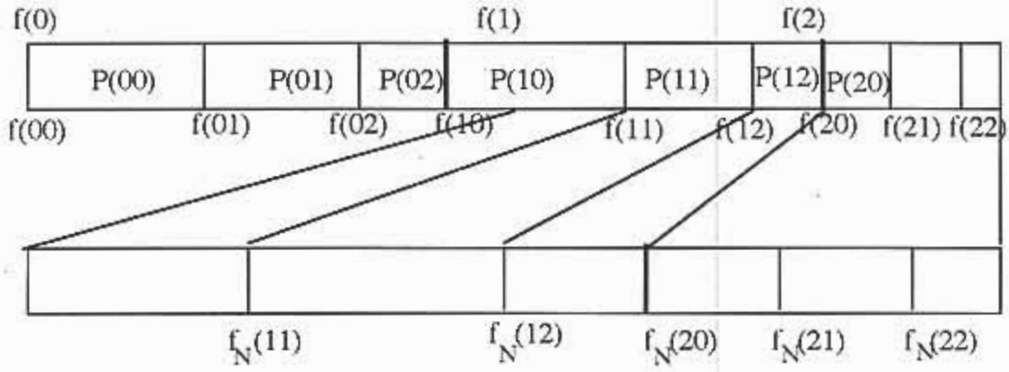


Figure 7

If $u_1=2$, then $x_1=1$ can be emitted by the encoder as soon as u_1 enters the encoder. The interval is then normalized by expanding the right half of the interval into the entire interval.

More precisely, the encoder keeps track of a normalized interval starting at $f_N(u^m)$ and ending at some point $g_N(u^m) = f_N(u^m)+P_N(u^m)$ where $P_N(u^m)$ is the width of the normalized interval. For reasons to be discussed later, it is desirable for the mapping from u^∞ to x to be one to one and onto. For this reason, we calculate the right and left end of each interval ($g_N(u^m)$ and $f_N(u^m)$) directly in such a way as to ensure that the right end of one interval is equal to the left end of the next contiguous interval. The width $P_N(u^m)$ of the interval is then $g_N(u^m) - f_N(u^m)$. We assume here that addition and subtraction are done without round off and that multiplication is done with some consistent round off rule (i.e., always round down, always round up, round to the nearest point, etc.). An algorithm is given below, but it has a subtle bug connected with Eqs. (15) and (16). After giving the algorithm, we explain the bug and then give a corrected version of (15) and (16).

ALGORITHM FOR ARITHMETIC ENCODING (WITH BUG)

0) Initially $f_N(u^0)=0$, $g_N(u^0)=1$, $m=0$.

1) Accept u_{m+1} into the encoder.

2) Calculate the new interval by the equations

$$P_N(u^m) = g_N(u^m) - f_N(u^m) \tag{14}$$

$$f_N(u^{m+1}) = f_N(u^m) + f_1(u_{m+1})P_N(u^m) \tag{15}$$

$$g_N(u^{m+1}) = f_N(u^m) + f_1(u_{m+1}+1)P_N(u^m) \tag{16}$$

where $f_1(K) = 1$

3) Find the longest string $y_1, y_2,...y_j$ such that

$$\sum_{i=1}^{j} y_i 2^{-i} \le f_N(u^{m+1}); \quad g_N(u^{m+1}) \le \sum_{i=1}^{j} y_i 2^{-i} + 2^{-j} \tag{17}$$

(Note that j might be 0).

Produce $y_1, y_2,...y_j$ as output and renormalize by the rule

$$f_N(u^{m+1}) := 2^j f_N(u^{m+1}) - \lfloor 2^j f_N(u^{m+1}) \rfloor \tag{18}$$

$$g_N(u^{m+1}) := 2^j g_N(u^{m+1}) - \lceil 2^j g_N(u^{m+1}) \rceil + 1 \tag{19}$$

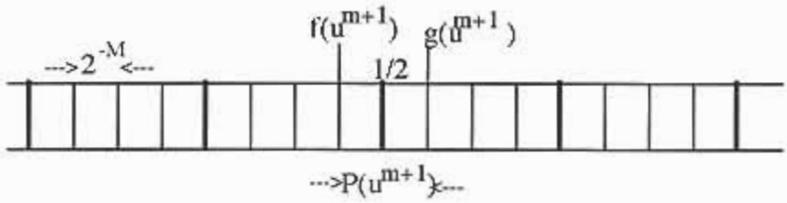
4) Increment m and goto step 1

Note that the renormalization in step 3 is done with no roundoff errors. One is simply eliminating the encoded binary digits (which are no longer needed for encoding), and adding less significant digits that increase the precision as the intervals shrink. One can visual8ize (18) as shifting the binary number f_N left j places and then dropping the integer part. Eq. (19) does the same thing, but also treats correctly the special case in which the resulting value of g_N is 1. In order to relate these normalized intervals to the actual intervals, as modified by roundoff errors, let $x^n = x_1, x_2,...x_n$ be the binary string already emitted by the encoder before u_{m+1} enters the encoder and let $\tilde{x}^n$ be the corresponding number $x_1 2^{-1} + x_2 2^{-2} + ... x_n 2^{-n}$. Then the normalized interval $[f_N(u^{m+1}), g_N(u^{m+1}))$ corresponds to the actual interval, with roundoff but no renormalization,

$$J(u^{m+1}) = [\tilde{x}^n + 2^{-n} f_N(u^{m+1}), \tilde{x}^n + 2^{-n} g_N(u^{m+1})) \tag{20}$$

In order to understand the bug in the above algorithm, consider the example of a ternary equiprobable source. First consider a long string of 1's as the input to the encoder. Without roundoff errors, $J(u^1) = [1/3, 2/3)$, $J(u^2) = [4/9, 5/9)$, and in general $J(u^m) =$

$[(1-3^{-m})/2, (1+3^{-m})/2)$. Thus, for this string, $J(u^m)$ continues to straddle the point 1/2 and no binary digits are emitted by the encoder. With only M binary digits of accuracy, however, the left and right ends of these intervals must each be multiples of 2^{-M}, and also must get close to 1/2, so that the way in which the roundoff is done is clearly important. Figure 8 illustrates the kind of problem that can arise. If the rounded off version of $Ju^m)$ becomes $[1/2-2^{-M}, 1/2+2^{-M})$, then no binary digit can be emitted, but when the next digit enters the encoder, it is not possible to split the interval into three distinct intervals. The simplest solution to this problem is round off in such a way as to avoid such small intervals around the point 1/2.



Note that the indicated interval around the point 1/2 does not allow a binary digit to be sent, but also, with the increments of size 2^{-M} as shown, forces an ambiguity in the next source letter since there are only two increments for an alphabet size of 3.

Figure 8

There are many ways of correcting this problem. The one we choose is slightly more complicated than necessary because we want to preserve the one to and and onto nature of the mapping. The approach used here is that whenever the left hand edge $f_N(u^{m+1})$ gets too close to the point 1/2, we move it to 1/2; to maintain the one to one onto mapping, we also move the right hand edge g_N of the adjoining interval for u' where $u'^m=u^m$ but $u'_{m+1} = u_{m+1}-1$. Figure 9 shows the correction that is made, and this is followed by the corrected algorithm. The parameter L is chosen small enough that $P_{min} > 2^{L-M}$. This guarantees that the interval at the beginning of an iteration is large enough to be split without rounding any interval size down to 0.

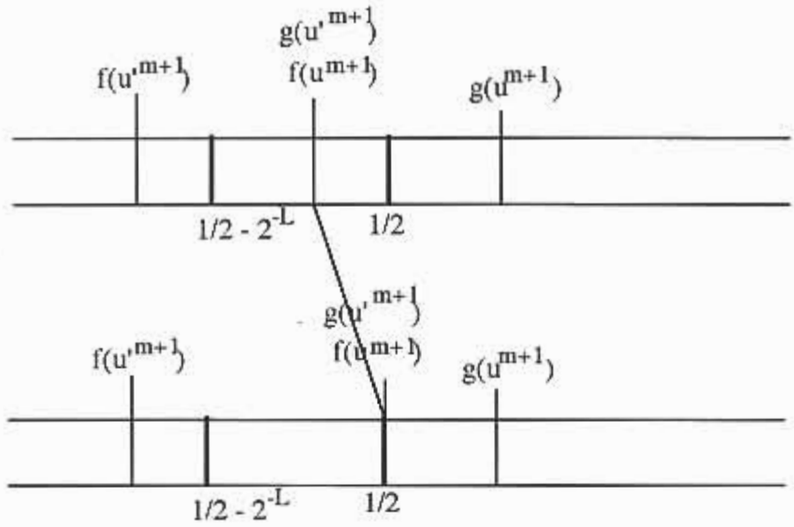


Figure 9
A correction to avoid intervals straddling the point 1/2.

ALGORITHM FOR ARITHMETIC ENCODING (WITHOUT BUG)

0) Initially $f_N(u^0)=0$, $g_N(u^0)=1$, $m=0$.

1) Accept u_{m+1} into the encoder.

2) Calculate the new interval by the equations

$$P_N(u^m) = g_N(u^m) - f_N(u^m) \tag{14}$$

$$f_N(u^{m+1}) = f_N(u^m) + f_1(u_{m+1})P_N(u^m) \tag{15}$$

$$g_N(u^{m+1}) = f_N(u^m) + f_1(u_{m+1}+1)P_N(u^m) \tag{16}$$

where $f_1(K) = 1$

If $1/2 - 2^{-L} < f_N(u^{m+1}) < 1/2 < g_N(u^{m+1})$ then

$$f_N(u^{m+1}) := 1/2 \tag{15a}$$

If $1/2 - 2^{-L} < g_N(u^{m+1}) < 1/2 < f_N(u^m) + f_1(u_{m+1}+2)P_N(u^m)$ then

$$g_N(u^{m+1}) := 1/2 \tag{16a}$$

(Note that $g_N(u^m) > 1/2$, so $g_N(u^{m+1})<1/2$ implies $u_{m+1}+2 \le K$)

3) Find the longest string $y_1, y_2, \ldots y_j$ such that

$$\sum_{i=1}^{j} y_i 2^{-i} \le f_N(u^{m+1}); \quad g_N(u^{m+1}) \le \sum_{i=1}^{j} y_i 2^{-i} + 2^{-j} \tag{17}$$

(Note that j might be 0).

Produce $y_1, y_2, \ldots y_j$ as output and renormalize by the rule

$$f_N(u^{m+1}) := 2^j f_N(u^{m+1}) - \lfloor 2^j f_N(u^{m+1}) \rfloor \tag{18}$$

$$g_N(u^{m+1}) := 2^j g_N(u^{m+1}) - \lceil 2^j g_N(u^{m+1}) \rceil + 1 \tag{19}$$

4) Increment m and goto step 1

Next consider the decoder. It is simplest to visualize the decoder as decoding one letter at a time and maintaining a queue of incoming binary digits and also a replica of the encoder. Initially, of course, m=1 and the queue is empty. The decoder, in attempting to decode u_{m+1} (with m initally 0), uses (14) to (16a) to calculate $f_N(u^{m+1})$ and $g_N(u^{m+1})$ for all K choices

of u_{m+1} (and, of course, the already decoded value of u^m). As new binary digits enter the queue, we can consider the queued digits as a normalized binary fraction of j significant bits, where j is the queue length. When this fraction, viewed as an interval of size 2^{-j}, lies within one of the K normalized intervals calculated above, the decoder decodes u_{m+1}, renormalizes f_N and g_N by the encoder rules and deletes the corresponding binary digits from the front of the queue. It then increments m and repeats the above process.

The encoder, in generating the string of binary digits used in decoding u_m, is constantly renormalizing while generating these binary digits, whereas the decoder is maintaining the normalization in effect when u_m was generated. Fortunately, because of the nesting property, each new letter to enter the encoder generates an interval strictly contained in the previous interval. Also, since no rounding off is incurred while renormalizing, (18) shows that preserving the nesting property at the encoder implies that the nesting property is also maintained at the decoder, even though the normalization is different.

Next note that when u_m enters the encoder, the interval end points are calculated to M binary digits of accuracy, and thus after M binary digits are emitted by the encoder, the resulting interval, according to the decoder normalization after decoding u_{m-1}, must have size 2^{-M} and thus u_m is decodable at this point if not before. This means that decoding always occurs with at most M binary digits in the queue. Thus the round off forced on us by using a real arithmetic unit also limits (to M) the difference between the number of binary digits required to decode u^m and the number of binary digits produced by encoding u^m. Thus using a small value of M can reduce the delay between encoding and decoding, although of course it also implies a loss in efficiency.

Note that occasionally the encoder will fall quite far behind (i.e., $I(u^{m(n)})$-n can become quite large (although not larger than M) in improbable cases even though $E[I(u^{m(n)})-n]$ is small). This causes an increase in roundoff error and causes the binary code digits to be not exactly

equiprobable. The loss in efficiency is quite negligible for large M, since the number of bits of accuracy is proportional to M (the number of bits of accuracy in the arithmetic unit) less $[I(u^{m(n)})-n]$.

For a source with memory, no new complications arise. The encoder simply uses $P(u_m|\, u_{m-1}...u_1)$ in place of $P(u_m)$. The replica of the encoder at the decoder also uses $P(u_m|\, u_{m-1}...u_1)$ in the same way. The encoder can also be adaptive. That is, the probability assignment for u_{m+1} can be based on the observed string u^m. Again the encoder and its replica at the decoder can operate in the same way. This, in fact, leads to a more general observation: any adaptive encoder can be viewed as estimating the probability assignment for u_{m+1} based on the observed string u^m. Any rule for doing this estimation is equivalent to a probabilistic source model with memory; i.e., $P(u_{m+1}|\, u^m)$ is matched to the adaptive probability of u_{m+1} based on the past history u^m. That is, adaptation is simply a convenient way of rationalizing some particular assignment of apriori probabilities to sequences. The adaptation can be viewed as dealing with the distant memory, and the probabilities found from adaptation can be viewed as short term memory. This means that the real question in data compression is not whether to use an adaptive scheme or a scheme with memory but rather to look at the tradeoff between complexity and different kinds of source memory.

References: Rissanen, J. & G.G.Langdon, "Universal Modeling and Coding", IEEE Trans. IT, Jan 1981, pp 12-23.

Rissanen, J. & G.G. Langdon, "Arithmetic Coding," IBM J. Res. and Dev., March 1979, pp. 149-162.