CSC310 – Information Theory Sam Roweis

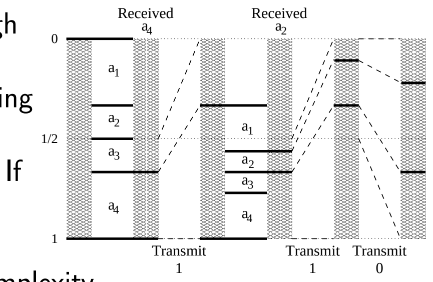
Lecture 8:

Arithmetic Coding – Practical Details

October 4, 2006

Reminder: Arithmetic Coding 1

- Represent a symbol sequence as a subinterval of $[0, 1)$ by recursively dividing the interval according to the probability of the next symbol.
- We encode this subinterval by transmitting a binary fraction that represents a number inside the interval, and for which any continuation would still lie in the interval.
- To avoid the need for extremely high numerical precision, we double the size of the subinterval during encoding whenever it is contained entirely within $[0, .5)$ or $[.5, 1)$ or $[.25, .75)$. If we do exactly the same doubling during decoding, we maintain correctness, without exponential complexity.



- Stream codes: no need to consider all possible blocks. Can implement using only fixed-point (scaled integer) arithmetic.

Arithmetic Coding Without Blocks (ver 1.1) 2

```
1) Initialize the interval $[u, v)$ to $u = 0$ and $v = 1$.
   Initialize the "opposite bit count" to $c = 0$.
2) For each source symbol, $a_i$, in turn:
   Compute $r = v - u$.
   Let $u = u + r \sum_{j=1}^{i-1} p_j$. Let $v = u + rp_i$.
   While $u \geq 1/2$ or $v \leq 1/2$ or $u \geq 1/4$ and $v \leq 3/4$:
     If $u \geq 1/2$:
       Transmit a 1 bit followed by $c$ 0 bits. Set $c$ to 0.
       Let $u = 2(u-1/2)$ and $v = 2(v-1/2)$.
     If $v \leq 1/2$:
       Transmit a 0 bit followed by $c$ 1 bits. Set $c$ to 0.
       Let $u = 2u$ and $v = 2v$.
     If $u \geq 1/4$ and $v \leq 3/4$:
       Set $c$ to $c + 1$.
       Let $u = 2(u-1/4)$ and $v = 2(v-1/4)$.
3) Transmit enough final bits to specify a number in $[u, v)$.
```

Precision of the Coding Interval 3

- Last class we recognized that since we are always expanding the coding interval to keep its size $\geq 1/4$ we only need a finite amount of precision to represent it.
- We proposed using fixed point arithmetic since it is faster and better defined on most machines.
- Thus, the ends of the coding interval will be represented by m-bit integers. The integer bounds u and v represent the interval
$$\big[\, u \times 2^{-m},\ (v{+}1) \times 2^{-m} \big)$$
(The addition of 1 to v allows the upper bound to be 1 without the need to use $m{+}1$ bits to represent v.)
- The received message will also be represented as an m-bit integer, t, (which at any point does not include further bits not yet read).
- With these representations, the arithmetic performed will never produce a result bigger than m bits + precision of probabilities.

Precision of the Symbol Probabilities

- Of course in any practical implementation, our symbol probabilities must also have some finite precision. This is fine, since small errors in these probabilities may reduce the coding efficiency very slightly but they won't affect the correctness of the encoder/decoder.
- The most common way to represent the symbol probabilities is as rational fractions with large denominators (to give reasonable accuracy).
- This is convenient, since we usually estimate the source probabilities for symbols $a_1, \ldots, a_i$ using counts $f_i > 0$:

$$p_i \;=\; f_i \,/ \sum_{j=1}^{I} f_j$$

- For example, these counts might come from examining the file we are about to compress or from a buffer of past messages.

Cumulative Symbol Probabilities

- For arithmetic coding, it's convenient to pre-compute the *cumulative probabilities*:

$$F_i \;=\; \sum_{j=1}^{i} p_j$$

  We define $F_0 = 0$. Obviously $F_I = 1$.
- We will assume that these cumulative probabilities F_i are represented using h bits of precision (either floating point or scaled fixed-point).

Encoding Using Integer Arithmetic

```
$u \leftarrow 0,\ v \leftarrow (2^m - 1),\ c \leftarrow 0$
For each source symbol, $a_i$, in turn:
  $r \leftarrow$ (double)$(v - u) + 1.0$
  $v \leftarrow u + \lfloor r * F_i \rfloor - 1$
  $u \leftarrow u + \lfloor r * F_{i-1} \rfloor$
  While $u \geq 2^m/2$ or $v < 2^m/2$ or ($u \geq 2^m/4$ and $v < 2^m * 3/4$):
    If $u \geq 2^m/2$:
      Transmit a 1 bit followed by $c$ 0 bits
      $c \leftarrow 0$
      $u \leftarrow 2 * (u - 2^m/2),\ v \leftarrow 2 * (v - 2^m/2) + 1$
    If $v < 2^m/2$:
      Transmit a 0 bit followed by $c$ 1 bits
      $c \leftarrow 0$
      $u \leftarrow 2 * u,\ v \leftarrow 2 * v + 1$
    If $u \geq 2^m/4$ and $v < 2^m * 3/4$:
      $c \leftarrow c + 1$
      $u \leftarrow 2 * (u - 2^m/4),\ v \leftarrow 2 * (v - 2^m/4) + 1$
Transmit a few final bits to specify a point in the interval:
If $u < 2^m/4$:    Transmit a 0 bit followed by $(c + 1)$ 1 bits.
Else:              Transmit a 1 bit followed by $(c + 1)$ 0 bits.
```

Decoding Using Integer Arithmetic

```
$u \leftarrow 0,\ v \leftarrow 2^m - 1,\ \ t \leftarrow$ first $m$ bits of the received message (pad with zeros if
                                                           total message bits $< m$)
Until last symbol decoded:
  $r \leftarrow$ (double)$(v - u) + 1.0;\quad w \leftarrow (($double$)(t - u) + 1.0) \,/\, r$
  Find $i$ such that $F_{i-1} \leq w < F_i$;   Output $a_i$ as the next decoded symbol
  $v \leftarrow u + \lfloor r * F_i \rfloor - 1$
  $u \leftarrow u + \lfloor r * F_{i-1} \rfloor$
  While $u \geq 2^m/2$ or $v < 2^m/2$ or $u \geq 2^m/4$ and $v < 2^m * 3/4$:
    If $u \geq 2^m/2$:
      $u \leftarrow 2 * (u - 2^m/2),\ v \leftarrow 2 * (v - 2^m/2) + 1$
      $t \leftarrow 2*(t - 2^m/2)$ + next message bit (set bit=0 if no more bits in message)
    If $v < 2^m/2$:
      $u \leftarrow 2 * u,\ v \leftarrow 2 * v + 1$
      $t \leftarrow 2 * t$ + next message bit (set bit=0 if no more bits in message)
    If $u \geq 2^m/4$ and $v < 2^m * 3/4$:
      $u \leftarrow 2 * (u - 2^m/4),\ v \leftarrow 2 * (v - 2^m/4) + 1$
      $t \leftarrow 2*(t - 2^m/4)$ + next message bit (set bit=0 if no more bits in message)
```

How Much Precision is Required?

- For this procedure to work properly, the loop that expands the interval must terminate. This requires that the interval never shrink to nothing — ie, we must always have $v \geq u$.
- This will be guaranteed as long as

$$\lfloor r * F_i \rfloor \;>\; \lfloor r * F_{i-1} \rfloor$$

  which holds if $f_i \geq 1$ (so that $F_i \geq F_{i-1} + 2^{-h}$) and $r \geq 2^h$.
- The expansion of the interval guarantees that $r \geq 2^m/4 + 1$.
- So the procedure will work as long as $2^h \leq 2^m/4 + 1$, or in other words as long as $m \geq h + 2$. We need to use at least as much precision (plus two bits more) to store our interval endpoints u, v as we use to store the (cumulative) symbol probabilities F_i.
- To obtain near-optimal coding, h may have to be relatively large, which will force us to use a large m as well.

Proving That the Decoder Finds the Right Symbol

- To show this, we need to show that if

$$F_{i-1} \;\leq\; ((\texttt{double})(t-u)+1.0)/r < \; F_i$$

  then

$$u + \lfloor r * F_{i-1} \rfloor \;\leq\; t \;\leq\; u + \lfloor r * F_i \rfloor - 1$$

- This can be proved as follows:

$F_i \;>\; (t-u+1.0)/r$
$\quad\Rightarrow\; r * F_i \;>\; t-u+1$
$\quad\Rightarrow\; u + r * F_i \;-1 > \; t$
$\quad\Rightarrow\; u + \lfloor r * F_i \rfloor \;-1 \geq \; t$ since t is integral

$F_{i-1} \;\leq\; (t-u+1.0)/r$
$\quad\Rightarrow\; r * F_{i-1} \leq \; t-u+1$
$\quad\Rightarrow\; u + r * F_{i-1} \leq \; t+1$
$\quad\Rightarrow\; u + r * F_{i-1} - 1 \leq \; t$
$\quad\Rightarrow\; u + \lfloor r * F_{i-1} \rfloor \leq \; t$ since $\lfloor r * F_i \rfloor \;>\; \lfloor r * F_{i-1} \rfloor$

Summary

- Arithmetic coding provides a practical way of encoding a source in a very nearly optimal way.
- Faster arithmetic coding methods that avoid multiplies and divides have been devised.
- **However:** It's not necessarily the best solution to *every* problem. Sometimes Huffman coding is faster and almost as good.
  Other codes may also be useful.
- Arithmetic coding is particularly useful for *adaptive* codes, in which probabilities constantly change. We just update the table of cumulative frequencies as we go.

History of Arithmetic Coding

- Elias — around 1960.
  Seen as a mathematical curiosity.
- Pasco, Rissanen – 1976.
  The beginnings of practicality.
- Rissanen, Langdon, Rubin, Jones – 1979.
  Fully practical methods.
- Langdon, Witten/Neal/Cleary — 1980's.
  Popularization.
- Many more... (eg, Moffat/Neal/Witten)
  Further refinements to the method.

Where Do the Probabilities Come From?

- So far, we've assumed that we "just know" the probabilities of the symbols, $p_1, \ldots, p_I$.
  Note: The transmitter and the receiver must both know the *same* probabilities.
- **This isn't realistic.**
  For instance, if we're compressing black-and-white images, there's no reason to think we know beforehand the fraction of pixels in the transmitted image that are black.
- **But could we make a good guess?**
  That might be better than just assuming equal probabilities. Most fax images are largely white, for instance. Guessing $P(\text{White}) = 0.9$ may usually be better than $P(\text{White}) = 0.5$.

The Penalty for Guessing Wrong.

- Suppose we use a code that would be optimal if the symbol probabilities were $q_1, \ldots, q_I$, but the real probabilities are $p_1, \ldots, p_I$. How much does this cost us?
- Assume we use large blocks or use arithmetic coding — so that the code gets down to the entropy, given the assumed probabilities.
- We can compute the difference in expected code length between an optimal code based on $q_1, \ldots, q_I$ and an optimal code based on the real probabilities, $p_1, \ldots, p_I$, as follows:

$$\sum_{i=1}^{I} p_i \log(1/q_i) - \sum_{i=1}^{I} p_i \log(1/p_i) = \sum_{i=1}^{I} p_i \log(p_i/q_i)$$

- This is the *relative entropy* of $\{p_i\}$ and $\{q_i\}$.
  It can never be negative. (See Section 2.6 of MacKay's book.)

Why Not Estimate the Probabilities And Then Send Them With the Data?

- One way to handle unknown probabilities is to have the transmitter *estimate* them, and then send these probabilities along with the compressed data, so that the receiver can uncompress the data correctly.
- **Example:** We might estimate the probabilitiy that a pixel in a black-and-white image is black by the *fraction* of pixels in the image we're sending that are black.
- **One problem:** We need some code for sending the estimated probabilities. How do we decide on that? We need to guess the probabilities for the different probabilities...

Why This Can't be Optimal

- This scheme may sometimes be a pragmatic solution, but it can't possibly be optimal, because the resulting code isn't *complete*.
- In a complete code, all sequences of code bits are possible (up to when the end of message is reached). A prefix code will not be complete if some nodes in its tree have only one child.
- Suppose we send a 3-by-5 black-and-white image by first sending the number of black pixels (0 to 15) and then the 15 pixels themselves, as one block, using probabilities estimated from the count sent.
- Some messages will not be possible, eg:

```
4   ○ ● ● ○ ○
    ● ● ● ● ○
    ○ ○ ● ● ●
```

  This can't happen, since the count of 4 is inconsistent with the image that follows.