Two Problems of Information Theory

- How can we represent information compactly, in as few bits as possible?

  Applications:
  - Compressing text or program files (gzip)
  - Compressing images (JPEG)
  - Compressing video (MPEG)

  Text and programs need to be compressed *losslessly*, but for images and video, we might accept *lossy* compression, in which the decompressed data isn't exactly the same as the original.

- How can we transmit or store information reliably, when our bits are subject to error?

  Applications:
  - ECC memory
  - Error correction on CDs
  - Communication with space probes

How Will We Tackle These Problems?

- We'll first look at some simple, practical ways in which we might try to solve them.

- We'll then develop a mathematical theory that shows how we can solve the problems systematically, and how well we can expect to do.

  Result: Two famous theorems proved by Claude Shannon in 1948.

- This theory is elegant, but it doesn't immediately produce practical solutions to the original problems that are as good as the theory says are possible. That took decades more work.

  Result: Practical methods based on "arithmetic coding" and "low-density parity-check codes" that are very nearly optimal.

But That's Not the End

- The theory and the optimal algorithms solve only part of the problem — we are left with the problem of finding an appropriate *model* for a source of data or for a channel.

- The optimal algorithms are fairly fast, but perhaps not fast enough for everyone — a non-optimal, less theoretical method might be better for some applications.

- The theory applies to problems of a certain sort. Problems with other characteristics need a different theory.

The Tools You'll Need

What background do you need to understand the theory and methods we'll look at?

Probability has a central role in both data compression and error-correcting codes. Only fairly elementary probability theory will be needed, however.

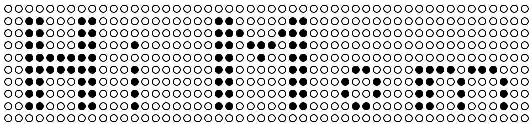
Linear algebra is the basis for practical error-correcting codes. Only fairly elementary stuff is needed here too, with one twist — we deal not with vectors of real numbers, but with vectors of bits, with modulo 2 arithmetic.

Plus, you'll need a modicum of skill at understanding and programming algorithms.

Example: Compressing a Black-and-White Image

Some images consist of an array of black and white pixels — eg, FAX images have this form.

An example of a 10 by 50 image:



If we encode black by 1 and white by 0, the number of bits needed for an image will equal the number of pixels — 500 bits for this image.

But can we do better? There are many more white pixels than black pixels, and both sometimes come in long runs. Can we exploit these properties in a compression scheme?

A Simple Compression Technique: Run-Length Coding

We can try to take advantage of the many long runs of white pixels (and some long runs of black pixels) by coding an entire run in just one byte (8 bits).

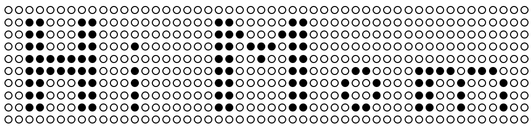
We'll use the first bit of a byte to specify the pixel colour (0=white, 1=black). The other seven bits specify the length of the run in binary (from 0 to 127).

We'll look at the pixels in "raster-scan" order — left to right, top to bottom. Assume that we know the dimensions of the image without having to encode that.

With this scheme we can compress a run of 127 white pixels to just 8 bits, for a compression ratio of $127/8 = 15.9$! Not all images have such long runs, however.

How Well Does it Work?

Let's see how well it works on "Hi Mom":



Here are the first few runs:

Run	**Encoding**
52 white	00110100
2 black	10000010
3 white	00000011
2 black	10000010
11 white	00001011
2 black	10000010

Altogether, there are 55 white runs and 54 black runs. Encoding each run takes 8 bits, for a total of $8 \times (55 + 54) = 872$ bits.

We've ended up expanding the data rather than compressing it!

An Improvement

Fortunately, the scheme can be improved.

Each run of white pixels must be followed by a run of black pixels, and vice versa. So we don't need to specify the colour for each run, just the colour for the first run.

A detail: If the 7 bits encoding the length of the run have their maximum value (127), then we assume the next run has the *same* colour. That way we can encode runs longer than 127 using several 7-bit counts.

This improvement reduces the number of bits needed to encode the "Hi Mom" image to $1 + 7 \times (55 + 54) = 764$ — but that's still worse than just using one bit for each of the 500 pixels!

A Scheme That's Better for the "Hi Mom" Image

One problem is that although our scheme compresses really long runs by a factor of 15.9, it isn't so good at the shorter runs that occur in the "Hi Mom" image.

A solution: Use fewer bits to specify the length of a run. Let's see what happens when we use three bits (for a maximum length of 7).

We'll now need to encode the first run of 52 white pixels as seven runs of length 7, plus an eighth run of length 3, requiring a total of 24 bits to encode. But we more than make up for this by using fewer bits for the short runs:

2 runs of length 52:	24 bits each
4 runs of length 23:	12 bits each
3 runs of length 11:	6 bits each
100 runs of length $\leq$7:	3 bits each

Total: $1+2\times24+4\times12+3\times6+100\times3 = 459$ bits.

Is This as Good as it Gets?

If we think the "Hi Mom" image is typical of the ones we want to encode, we could expect that by using run-length coding with 3-bit lengths, we could reduce the space needed to store an image by roughly 8%, on average.

We might hope for greater compression. Two possibilities:

- If the "Hi Mom" image is typical, runs of black pixels are usually shorter than runs of white pixels. Maybe we should encode their lengths with different numbers of bits.
- Consider these encoded bits: 011 000 010. They represent a run of 3 white pixels, a run of 0 black pixels, and a run of 2 white pixels — the same as a single run of 5 white pixels. Encoding will never produce this sequence, so we're wasting space.

Modeling and Coding

These two possible improvements apply to two conceptually distinct aspects of data compression:

Modeling is the development of knowledge of the types of images (or other data) that we're expecting will be stored (or transmitted) with the data compression program we're designing.

Run-length coding is based on a model saying that long runs of the same colour are likely.

Coding is the way we use the knowledge in our model to chose specific bit patterns to represent the data.

If our model tells us how long we expect the runs to be, we can then choose how many bits to use for a run length so as to minimize the average number of bits used.

Both aspects are essential. In particular, for data compression to be possible *we must have some knowledge of the type of data that is likely*.

Probabilistic Modeling and Coding

The modeling and coding for the run-length example was *ad hoc* — we just did whatever seemed to work well.

A more general approach is to build a model based on *probabilities* of symbols, and use these probabilities to choose a coding method.

Probabilistic modeling is an art, though there are general principles to help.

Probabilistic coding is a mathematical and algorithmic problem, which has been largely solved. The mathematical solution is Shannon's noiseless coding theorem of 1948. The algorithmic solutions took a bit longer.

A Probabilistic Model for Black-and-White Images

One simple way to capture the structure expected in images such as "Hi Mom" is to model the *conditional probabilities* for a pixel to be black *given* the colours of the pixels above it and to the left.

If our knowledge of the source of the images tells us what these conditional probabilities are, we can use a *static* model.

More likely, we don't know the probabilities, so we'll instead have to learn them as we go through the image — an *adaptive* model.

A data compression program that combines such a simple adaptive model with a near-optimal method of coding compresses the "Hi Mom" image to 256 bits, a compression ratio of $500/256 = 1.95$.

Data Compression Topics We'll Cover

- Lossless data compression must allow the data to be decompressed back to the original file. For what coding schemes is such decompression possible?
- Among such "uniquely decodable" coding schemes, which produce the shortest length codes, on average?
- **Shannon's noiseless coding theorem:** Average code length with such optimal encoding approaches the source's *entropy*.
- How we can encode optimally in practice using "arithmetic coding".
- Basics of *source modeling*, including "dictionary" techniques that combine modeling and coding.
- Later... A bit about lossy data compression.