CSC 310: Information Theory

University of Toronto, Fall 2011

Instructor: Radford M. Neal

Week 1

Two Problems of Information Theory

How to represent information compactly, in as few bits as possible.

Applications:

- Compressing text or program files (eg, gzip)
- Compressing images (eg, JPEG)
- Compressing video (eg, MPEG)

Text and programs need to be compressed *losslessly*.

For images and video, we might accept *lossy* compression, where the decompressed data isn't exactly the same as the original.

How to transmit/store information reliably, if bits are subject to error.

Applications:

- ECC memory
- Error correction on hard disks, CDs, DVDs
- Error correction for network packets
- 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.

  Results: 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.

  Results: Practical lossless data compression with "arithmetic coding".
  Practical error correction methods using "low-density parity-check codes" that are very nearly optimal.

  Research: Can we build practical *lossy* data compression methods that use the theory to produce near-optimal results?

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. Other problems might need a modified theory — eg, suppose we send a message to several people, and require only that at least one decode it correctly?
- The theory can provide insights useful for problems other than communicaton — eg, cryptography, machine learning.

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.

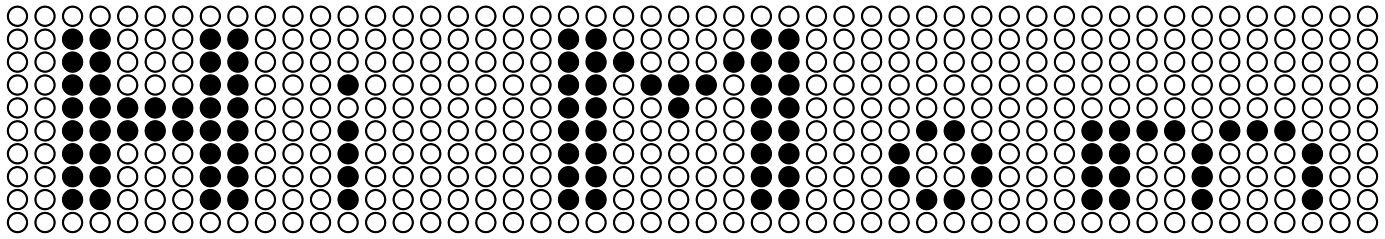
Linear algebra is the basis for practical error-correcting codes. Only fairly elementary stuff is needed here too, with one twist — we deal not with vectors of real numbers, but with vectors of bits, with modulo 2 arithmetic.

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, simple 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 of black pixels) by coding an entire run in just one byte (8 bits).

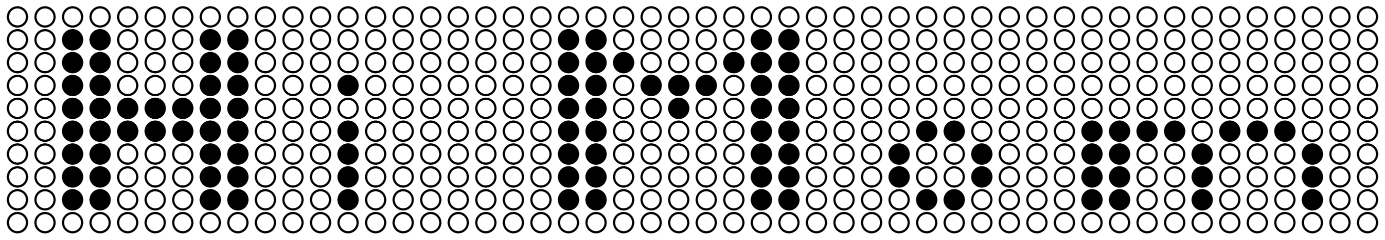
We'll use the first bit of a byte to specify the pixel colour (0=white, 1=black). The other seven bits specify the length of the run in binary (from 0 to 127).

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 nice 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 well, 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 now must encode the first run of 52 white pixels as seven runs of length 7, plus an eighth run of length 3, for a total of 24 bits. 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 ≤ 7:	3 bits each

Total: $1+2\times 24+4\times 12+3\times 6+100\times 3 = 459$ 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 might expect that run-length coding with 3-bit lengths will reduce the space needed to store an image by roughly 8%, on average.

We might hope for greater compression. Two of many 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. A new model may be needed when we deal with a new source of data.

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 way to capture the expected structure 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.

Example of Error Correction: A Repetition Code

Suppose we're trying to send a series of bits through a channel that sometimes randomly changes a bit from 0 to 1 or vice versa.

One way to improve reliability despite this "noise" is to send each bit *three times*.

Example: if we want to communicate the bit sequence 01101, we actually send 000111111000111.

The receiver looks at the bits in groups of three, and decodes each group to the bit that occurs most often in the group.

Here's an example in which the correct message is decoded despite four transmission errors:

$$\begin{array}{cccccc} & & \text{messsage transmitted} & & \\ 01101 & \rightarrow & 000111111000111 & & \\ & \rightarrow & 010111011100101 & \rightarrow & 01101 \\ & & \text{noisy message received} & & \end{array}$$

How Much Does a Repetition Code Improve Reliability?

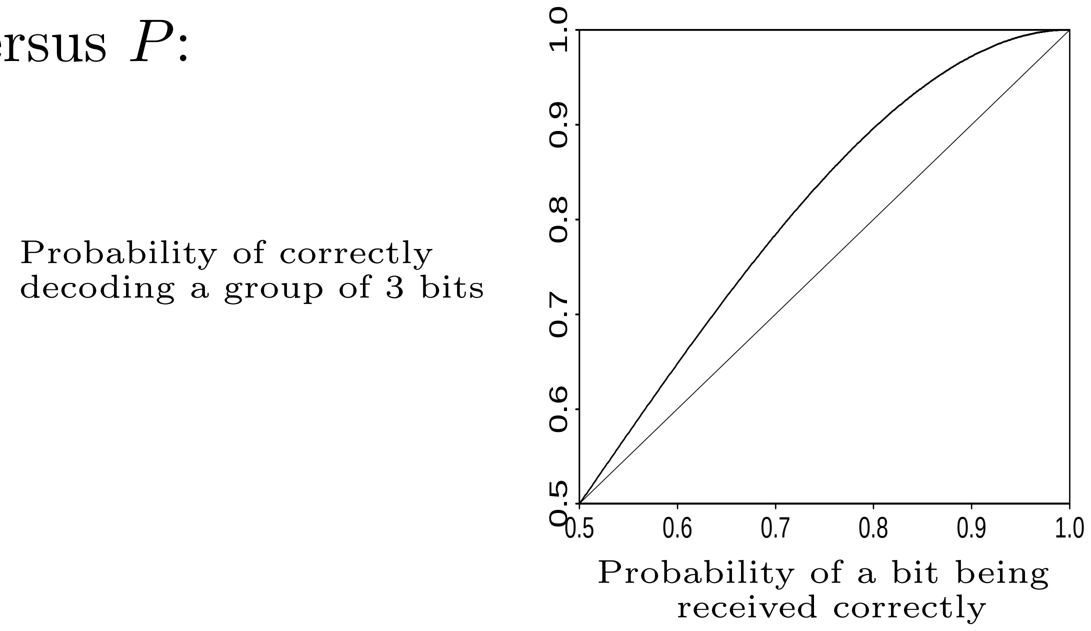
Repeating each bit three times allows us to correct *one error* in each group of three bits, but not more errors.

Suppose each bit has probability P of being received correctly, independently for each bit.

Then the probability that a group of three repeated bits will be decoded correctly is

$$\Pr(0 \text{ errors}) \;+\; \Pr(1 \text{ error}) \quad = \quad P^3 + 3P^2(1-P)$$

Here's a plot of this versus P:



Repeating More Times

Suppose we repeat an odd number, n, times. We'll be able to decode correctly as long as there are less than $n/2$ errors in these bits.

As n gets bigger, the decoding error probability goes down. Here's what happens with a 10% probability of transmission error (ie, $P = 0.9$):

Number of repetitions	Probability of incorrect decoding
3	0.028
5	0.0086
7	0.0027
9	0.00089
11	0.00030
13	0.000099
15	0.000034

It seems that we can push the error probability as close to zero as we wish by using more repetitions.

But What's the Cost?

Unfortunately, though we can push the error probability to zero, the *rate* at which we transmit useful information goes to zero at the same time!

Fortunately, there are cleverer schemes than repetition codes.

Unfortunately, the simple schemes (and many not-so-simple ones) still have this property — reducing the error probability to zero requires reducing the information rate to zero too.

Is this inevitable?

The Surprising Answer

Shannon proved in 1948 that we actually *can* transmit with arbitrarily small error at any rate up to the *capacity* of the channel.

For the example with $P = 0.9$, the capacity turns out to be 0.531. So we should be able to obtain nearly zero decoding errors while transmitting only twice as many bits as are in the message (rate 0.5).

One catch: To get lower and lower error rates, we need to transmit using bigger and bigger *blocks*.

Eg, for rate 0.5, we encode a block of $n/2$ bits into n bits. Obtaining a very low decoding error probability will require that n be quite big.

Can We Do This in Practice?

For decades after Shannon's theoretical work, it was widely believed that near-optimal coding could not be achieved in practice.

However, practical methods for transmitting at close to capacity have been developed in the last twenty years.

I tried the example with $P = 0.9$ (ie, 10% transmission errors) using a simple type of "low-density parity-check code".

I used blocks of 4000 message bits, encoded into 10000 transmitted bits (a rate of 0.4).

In a test transmitting 1000 such blocks, there were no errors in the decoded messages.

Error-Correction Topics We'll Cover

- Information channels, particularly the *Binary Symmetric Channel*.
- General aspects of error-correcting codes.
- Properties of *linear codes*, and some practical examples.
- **Shannon's noisy coding theorem:** Information can be transmitted with *arbitrarily small probability of error* at any rate below the *capacity* of the channel.
- Low-density parity-check codes, which come close to achieving this theoretical promise in practice.

Other Topics I May Cover

- Information theory with continuous variables.
- Shannon's *rate distortion theory* of lossy data compression.
- Practical aspects of lossy data compression.
- Other applications of information theory.