CS3236 Lecture Notes #0: Introduction

Jonathan Scarlett

January 8, 2020

Useful references:

- Cover/Thomas Chapter 1
- MacKay Chapter 1
- Blog posts on probability[1], conditional probability,[2] and information theory[3]

1 Overview of the Course

Goals:

- Students are expected to learn the most fundamental topics in information theory, understand proofs of mathematical theorems on the limits of data compression and communication, and know a few practical compression and communication methods.
- There are extensive more advanced topics that we will not cover, but students should end up in a position where they can read about them in textbooks and perhaps journal papers.

Main topics:

- Information measures (e.g., entropy, mutual information)
- Symbol-wise source coding
- Block source coding
- Channel coding
- Continuous-valued sources and channels
- Practical channel codes

[1] https://jeremykun.com/2013/01/04/probability-theory-a-primer/
[2] https://jeremykun.com/2013/03/28/conditional-partitioned-probability-a-primer/
[3] https://jeremykun.com/2015/02/16/a-proofless-introduction-to-information-theory/

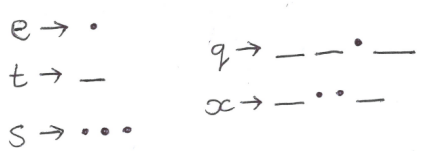
A large number of topics not covered:

- Sources and channels with memory (only briefly mentioned)
- Most practical channel coding methods (e.g., turbo codes, LDPC codes, polar codes, BCH codes, Reed-Solomon codes, Reed-Muller codes, etc.) and decoding methods (e.g., belief propagation)
- Multi-user communication
- Communication with feedback
- Finite-length analysis
- Channels with state
- Zero-error capacity
- Wireless communication
- Information-theoretic secrecy/privacy
- Information-theoretic cryptography
- Information theory and statistics
- Information-theoretic understanding of machine learning and data science

2 A Preview of Data Compression

Familiar examples of compression.

- When we compress a file to .zip or .rar it gets smaller, and yet we can still recover the contents. How/why is this possible? This is the problem of **lossless compression**.
- When we convert a file from .bmp to .jpeg, we lose some quality, but hopefully not too much. However, we cannot convert back to the higher-quality image. This is the problem of **lossy compression**.
- Concepts in compression go further back than computers – recall Morse code:



Example 1: Sparse binary string.

- Suppose that we want to efficiently store

  $$0000000\mathbf{1}00000000000000000000\mathbf{1}00000000000000000000000\mathbf{1}000000000,$$

  i.e., a string of length 64 with only three 1s and the rest 0s.
- Storing this "as is" requires 64 bits (a bit being a 1 or 0).

- Alternative scheme:
  - Index the string positions from 0 to 63, and consider their binary format (e.g., $0 \to 000000$, $7 \to 000111$, $63 \to 111111$)
  - Store the 3 positions where the long string has value 1, using 6 bits per position.
- This permits only 18 bits of storage instead of 64.

Example 2: Equal number of 1s and 0s.

- Suppose that we need a system that can compress sequences of the form

$$101101010110011101001100110010010110101110101010001001001010011,$$

  i.e., still length 64, but now half ones and half zeros.
- Again, storing "as is" requires 64 bits.
- The number of strings with half zeros and half ones is $\binom{64}{32}$. Let's aim to compress them down to some number $L < 64$ of bits. How small can L be?
- With L bits (each 0 or 1), we can make 2^L combinations. Since each of the $\binom{64}{32}$ strings have to be stored as a different combination, we clearly need $2^L \geq \binom{64}{32}$, or equivalently

$$L \geq \log_2 \binom{64}{32} \approx 60.7.$$

- So we can't hope to do much better than direct storage!

Example 3: English text.

- English text clearly has a fair bit of redundancy, since we can "throw away" several letters but still (usually) recover the original text:

C_N Y__ F_LL _N TH_ V_W_LS _N TH_S S_NT_NC_?

- If we (somewhat naively) store English text in some binary format in a letter-by-letter fashion, we can exploit the fact that some letters are more common than others, e.g., map 'e' to a short binary sequence, and 'x' to a long binary sequence.
  - Morse code is an early example of this idea (but not quite "binary"!)
  - Can we construct an "optimal" mapping?
- The savings are much greater if we exploit the fact that different *groups* of letters are more likely to appear together (e.g., if we have already seen "Fill in the blan", then there is clearly a much more likely letter than 'e' coming next!)
- Spoiler: While it requires 5 bits (or at least $\log_2 27 \approx 4.75$) to uniquely identify one of 27 characters ('a' to 'z' and also spaces), the actual "information content" of each letter in English text is only about 1.34 bits. This will mean that we can compress down by a factor of at least $3\times$ without losing anything.

Information-theoretic viewpoint.

- Information theory adopts probabilistic models, e.g., a string of 1000 English characters is modeled by some joint distribution $P_{X_1X_2\ldots X_{999}X_{1000}}$, where each X_i takes some value in $a \ldots z$ (or space).
- Two distinct approaches to compression:
  - (Variable-length) Map more probable sequences to shorter binary strings, at the expense of mapping less probable sequences to longer strings. **How low can the average length be?**
  - (Fixed-length) Map the most probable sequences to binary strings of a given length, at the expense of not having enough such strings for the low-probability sequences. **How low can the length be while having a very low probability of failure?**
- **Source coding theorem (informal).** In both of these settings, the fundamental compression limit is given by a source-dependent quantity known as the *(Shannon) entropy* H. The (average) storage length can be arbitrarily close to H, but can never be any lower than H.

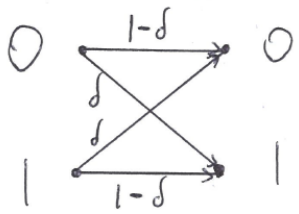
3 A Preview of Data Communication

Familiar examples of communication.

- When military pilots want to read a sequence of letters over an intercom, they use "alpha", "bravo", "Charlie", etc.
- If someone on the other end of the phone is having trouble hearing us, we might repeat the same thing 2–3 times to make sure they hear it.
- If we're talking to someone in their non-native language, we might talk slower.
- Our WiFi slows down as we move further away from the router.
- Common theme: Send information *more slowly* but also *more reliably.*
  - For a given reliability, how slow do we need to go?

Simple communication setting.

- Let's suppose that we are communicating in binary:
  - A "transmitter" sends a sequence of 0s and 1s
  - A "receiver" receives the sequence *with some corruptions*: Each bit is flipped (from 0 to 1, or from 1 to 0) independently with probability $\delta \in \left(0, \frac{1}{2}\right)$.
  - This is depicted in the following "channel transition diagram":



– e.g., The sequence 01101000 might be received as 0**0**101**1**00 (two corruptions)

Approach 1: Uncoded communication.

- Suppose that the transmitter wants to send one of 16 messages (e.g., it has done a weather reading and wants to send one of 16 possibilities among "sunny", "rainy", "partly cloudy", etc.)
- Naively, it can do this by mapping each outcome to a unique sequence of 4 bits (e.g., sunny $\to$ 0000, rainy $\to$ 1010, etc.)
- Since each bit is flipped with probability δ, the probability of all 4 bits coming out correct is $(1-\delta)^4$. For instance, if $\delta = 0.1$, we have $\mathbb{P}[\text{correct}] = 0.9^4 = 0.6561$.
- Things get worse as we send more messages, e.g., if we encode one of $2^8 = 256$ messages to a length-8 binary string and transmit it, we get $\mathbb{P}[\text{correct}] = (1-\delta)^8$, which is roughly 0.43 when $\delta = 0.1$.

Approach 2: Repetition code.

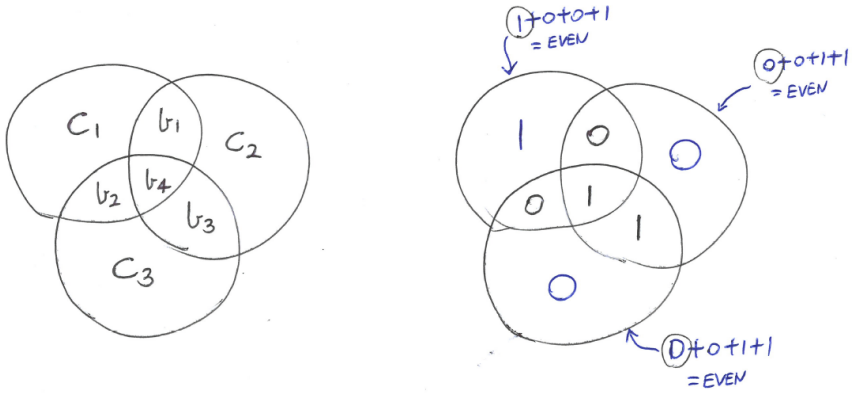
- As mentioned above, let's try transmitting slower but more reliably!
- Let's start with just sending one of two messages, which we will label as 0 and 1.
- Repetition code R_3 of length 3:
  - To send "0", transmit the sequence "000"
  - To send "1", transmit the sequence "111"
  - At the receiver, take the majority vote (e.g., 000 or 010 get decoded as "0", whereas 111 or 110 get decoded as "1")
- Clearly, we get correct decoding if there are no flips or one flip, so $\mathbb{P}[\text{correct}] = (1-\delta)^3 + 3\delta(1-\delta)^2$, which equals 0.972 when $\delta = 0.1$.
- We can then transmit, say, one of 16 messages by mapping (e.g.) 0101 to 000111000111. The probability of getting back the correct message is $0.972^4 \approx 0.893$
  - A fair bit more reliable than uncoded – but three times slower!
- We can do the same with more repetitions:
  - e.g., map 0101 to 0000000111111100000001111111 (repetition code R_7)
  - Any given bit (out of the 4 sent) is decoded correctly with probability

$$(1-\delta)^7 + 7\delta(1-\delta)^6 + \binom{7}{2}\delta^2(1-\delta)^5 + \binom{7}{3}\delta^3(1-\delta)^4 \approx 0.9973$$

    (This is the probability that a Binomial$(7, 0.1)$ random variable is at most 3)
  - The overall message is decoded correctly with probability $\approx 0.9973^4 \approx 0.989$.
  - Now the communication is very reliable, but we are 7 times slower than uncoded! Do we have to keep getting slower and slower?

Approach 3: Hamming code.

- Here we give a famous example of how to map a binary string of length 4 (so 16 messages) to a binary string of length 7 while still being able to correct one bit flip.
- The technique: In the following figure, fill in $b_1b_2b_3b_4$ ('b' for 'bit') with the original four bits, and assign $c_1c_2c_3$ ('c' for 'check') the values that make the three bits in their circle add up to an even number. (An example is shown on the right)



- Observe that any single bit flip (whether it be one of the b_i or one of the c_i) changes a unique combination of circles from "even" to "odd"! Therefore, if a single bit flip occurs, we can uniquely identify which bit caused it, and therefore correct it.
  - We can also distinguish the case that no bit flips occurred, and hence all 3 circles remain "even".
- Therefore

$$\begin{aligned}\mathbb{P}[\text{correct}] &\geq \mathbb{P}[\text{zero or one bit flip(s)}] \\ &= (1-\delta)^7 + 7\delta(1-\delta)^6 \\ &\approx 0.85,\end{aligned}$$

  where the last line holds when $\delta = 0.1$.
- Nearly as reliable as the repetition code, despite mapping to only 7 bits instead of 12! (i.e., we are transmitting a fair bit "less slowly")

Preview of information-theoretic results.

- **Definition.** If we map k bits to n bits in the encoding procedure, then the *rate* is $\frac{k}{n}$ (e.g., $\frac{4}{7}$ for the above Hamming code, $\frac{1}{3}$ for the repetition code, 1 for uncoded)
- Clearly, there is an inherent trade-off between rate and error probability.
  - Higher rate = Send faster
  - Lower error probability = Send more reliably
- **Channel coding theorem (informal).** There exists a channel-dependent quantity called the *(Shannon) capacity* C such that arbitrarily small error probability can be achieved for all rates less than C, but for no rates higher than C.

- In the above example with $\delta = 0.1$, we get $C \approx 0.531$. So for arbitrarily small error probability (e.g., $\mathbb{P}[\text{error}] \leq 10^{-10}$), not only is it unnecessary to multiply the number of bits by a higher and higher number, but we can get away with fewer than double the original number (!)
- Caveat: We may need to code over a much longer block length (e.g., map $k = 5000$ bits to $n = 10000$ bits, rather than mapping $k = 3$ bits to $n = 6$ bits)

Principles of information theory:

- First fundamental limits, then practical methods
- First asymptotic results, then finite-length refinements
- Mathematically tractable yet powerful probabilistic models